TMS320C2x User's Guide

SPRU014

Second-Generation TMS320 User's Guide

Digital Signal Processing Products

TEXAS INSTRUMENTS

Second-Generation TMS320 User's Guide

Texas
Instruments

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to or to discontinue any semiconductor product or service identified in this publication without notice. TI advises its customers to obtain the latest version of the relevant information to verify, before placing orders, that the information being relied upon is current.

TI warrants performance of its semiconductor products to current specifications in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Unless mandated by government requirements, specific testing of all parameters of each device is not necessarily performed.

TI assumes no liability for TI applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Copyright © 1987, Texas Instruments Incorporated

Manual Update

Document Title: Second-Generation TMS320 Users Guide

Document Number: SPRU014 ECN Number: 526628

The following are changes to the Second-Generation TMS320 User's Guide. These changes will be incorporated in the next revision of the manual. Bars in the right margin indicate changes or additions to the manual.

Page	Change or Add
3-25	On the top line following the words "which do not affect the accumulator" add a footnote flag (†)and, on the bottom of the page, add the following footnote: †BIT instruction may affect the accumulator on the TMS32020 under certain circumstances. Refer to Section 4.3, Page 4-45.
4-45	Under the **Execution** heading, following the words "Affects TC", add the following warning: **Caution: See note on next page concerning execution by TMS32020.**

TEXAS INSTRUMENTS

Page | Change or Add

4-46 Under the example given for the Test Bit instruction add the following note:

> **Note:**
>
> This instruction may affect the contents of the accumulator on the TMS32020 if the following conditions occur:
>
> 1) Overflow mode set (OVM status register bit is set).
> 2) Two LSBs of BIT instruction opcode (bits 8 and 9 of the instruction word) are zero.
> 3) Addition of accumulator contents with contents of addressed memory would cause accumulator overflow.
>
> If all of the above conditions are met, the contents of the accumulator will be replaced by the positive or negative saturation value, depending on the polarity of the overflow.
>
> This situation can be avoided by any one of the following means:
>
> - Precede BIT instruction with ROVM, and follow BIT with SOVM instruction.
> - If direct addressing is being used, reorganize memory so that page relative locations 0, 4, 8, C, and 10 are not used.
> - If indirect addressing is used, select new ARP that is not AR0 or AR4, and restore code later (if necessary) with LARP AR0/4.
> - Use BITT instead of BIT. BITT does not affect accumulator under any circumstances.
> - Use TMS320C25 (pin and object code compatible) instead.
>
> This situation occurs only when the BIT instruction is executed by a TMS32020 that is in the saturation mode.

Page | Change or Add

5-32 Following the last paragraph of section 5.5.2 (under the words "past processing") add the following note:

> **Note:**
>
> Under certain circumstances, executing the BIT instruction may affect the contents of the accumulator on the TMS32020 device. For more information, refer to the instruction definition in Section 4.3 Page 4-45.

Contents

Section *Page*

1 Introduction **1-1**
1.1 General Description 1-3
1.2 Key Features 1-4
1.3 Typical Applications 1-5
1.4 How To Use This Manual 1-7
1.5 References 1-9

2 Pinouts and Signal Descriptions **2-1**
2.1 TMS320C2x Pinouts 2-2
2.2 TMS320C2x Signal Descriptions 2-3

3 Architecture **3-1**
3.1 Architectural Overview 3-2
3.2 Functional Block Diagram 3-5
3.3 Internal Hardware Summary 3-7
3.4 Memory Organization 3-11
3.4.1 Data Memory 3-11
3.4.2 Program Memory 3-12
3.4.3 Memory Maps 3-13
3.4.4 Memory-Mapped Registers 3-16
3.4.5 Auxiliary Registers 3-16
3.4.6 Memory Addressing Modes 3-19
3.4.7 Memory-to-Memory Moves 3-20
3.5 Central Arithmetic Logic Unit (CALU) 3-22
3.5.1 Scaling Shifter 3-23
3.5.2 ALU and Accumulator 3-24
3.5.3 Multiplier, T and P Registers 3-26
3.6 System Control 3-28
3.6.1 Program Counter and Stack 3-28
3.6.2 Pipeline Operation 3-29
3.6.3 Reset 3-41
3.6.4 Status Registers 3-42
3.6.5 Timer Operation 3-44
3.6.6 Repeat Counter 3-46
3.6.7 Powerdown Mode (TMS320C25) 3-46
3.7 External Memory and I/O Interface 3-47
3.7.1 Memory Combinations 3-47
3.7.2 Internal Clock Timing Relationships 3-48
3.7.3 General-Purpose I/O Pins (BIO and XF) 3-49
3.8 Interrupts 3-52
3.8.1 Interrupt Operation 3-52
3.8.2 External Interrupt Interface 3-53
3.9 Serial Port 3-56
3.9.1 Transmit and Receive Operations 3-58
3.9.2 Timing and Framing Control 3-60
3.9.3 Burst-Mode Operation 3-61
3.9.4 Continuous Operation Using Frame Sync Pulses (TMS320C25) 3-62
3.9.5 Continuous Operation Without Frame Sync Pulses (TMS320C25) . . . 3-64

3.9.6 Initialization of Continuous Operation Without Frame Sync Pulses (TMS320C25) 3-66
3.10 Multiprocessing and Direct Memory Access (DMA) 3-68
3.10.1 Synchronization 3-68
3.10.2 Global Memory 3-69
3.10.3 The Hold Function 3-71

4 Assembly Language Instructions **4-1**
4.1 Memory Addressing Modes 4-2
4.1.1 Direct Addressing Mode 4-2
4.1.2 Indirect Addressing Mode 4-4
4.1.3 Immediate Addressing Mode 4-9
4.2 Instruction Set 4-10
4.2.1 Symbols and Abbreviations 4-10
4.2.2 Instruction Set Summary 4-12
4.3 Individual Instruction Descriptions 4-17

5 Software Applications **5-1**
5.1 Processor Initialization 5-2
5.2 Program Control 5-7
5.2.1 Subroutines 5-7
5.2.2 Software Stack 5-10
5.2.3 Timer Operation 5-11
5.2.4 Single-Instruction Loops 5-13
5.2.5 Computed GOTOs 5-14
5.3 Interrupt Service Routine 5-16
5.3.1 Context Switching 5-16
5.3.2 Interrupt Priority 5-22
5.4 Memory Management 5-23
5.4.1 Block Moves 5-23
5.4.2 Configuring On-Chip RAM 5-25
5.4.3 Using On-Chip RAM for Program Execution 5-28
5.5 Fundamental Logical and Arithmetic Operations 5-31
5.5.1 Status Register Effect on Data Processing 5-31
5.5.2 Bit Manipulation 5-32
5.6 Advanced Arithmetic Operations 5-34
5.6.1 Overflow Management 5-34
5.6.2 Scaling 5-35
5.6.3 Moving Data 5-35
5.6.4 Multiplication 5-37
5.6.5 Division 5-42
5.6.6 Floating-Point Arithmetic 5-45
5.6.7 Indexed Addressing 5-49
5.6.8 Extended-Precision Arithmetic 5-49
5.7 Application-Oriented Operations 5-60
5.7.1 Companding 5-60
5.7.2 FIR/IIR Filtering 5-61
5.7.3 Adaptive Filtering 5-63
5.7.4 Fast Fourier Transforms (FFT) 5-68
5.7.5 PID Control 5-75

6 Hardware Applications 6-1
6.1 System Control Circuitry 6-3
6.1.1 Powerup Reset Circuit 6-3
6.1.2 Crystal Oscillator Circuit 6-5
6.1.3 User Target Design Considerations When Using the XDS 6-7
6.2 Interfacing Memories 6-10
6.2.1 Interfacing PROMs 6-11
6.2.2 Wait-State Generator 6-16
6.2.3 Interfacing EPROMs 6-19
6.2.4 Interfacing Static RAMs 6-24
6.2.5 Interface Timing Analysis 6-27
6.3 Direct Memory Access (DMA) 6-29
6.4 Global Memory 6-32
6.5 Interfacing Peripherals 6-34
6.5.1 Combo-Codec Interface 6-34
6.5.2 AIC Interface 6-37
6.5.3 Digital-to-Analog (D/A) Interface 6-39
6.5.4 Analog-to-Digital (A/D) Interface 6-40
6.5.5 I/O Ports 6-42
6.6 System Applications 6-45
6.6.1 Echo Cancellation 6-45
6.6.2 High-Speed Modem 6-45
6.6.3 Voice Coding 6-46
6.6.4 Graphics and Image Processing 6-47
6.6.5 High-Speed Control 6-47
6.6.6 Instrumentation and Numeric Processing 6-48

A TMS320 Second-Generation Digital Signal Processors Data Sheet A-1
B SMJ32020 Data Sheet B-1
C TMS320C2x System Migration C-1
D Instruction Cycle Timings D-1
E Development Support/Part Order Information E-1
F Memories, Analog Converters, Sockets, and Crystals F-1
G ROM Codes G-1
H Quality and Reliability H-1

Illustrations

Figure *Page*

1-1 TMS320 Device Evolution 1-1
2-1 TMS320C2x Pin Assignments 2-2
3-1 TMS320C2x Simplified Block Diagram 3-2
3-2 TMS320C2x Block Diagram 3-6
3-3 On-Chip Data Memory 3-12
3-4 Memory Maps 3-15
3-5 Indirect Auxiliary Register Addressing Example 3-17
3-6 Auxiliary Register File 3-18
3-7 Methods of Instruction Operand Addressing 3-20
3-8 Central Arithmetic Logic Unit (CALU) 3-23
3-9 Examples of TMS320C25 Carry Bit Operation 3-25
3-10 Program Counter, Stack, and Related Hardware 3-28
3-11 Three-Level Pipeline Operation (TMS320C25) 3-30
3-12 Two-Level Pipeline Operation 3-31
3-13 TMS320C25 Standard Pipeline Operation 3-32
3-14 Pipeline Operation of ADD Followed by SACL 3-34
3-15 Pipeline Operation with Wait States 3-35
3-16 Pipeline with External Data Bus Conflict 3-36
3-17 Pipeline Operation of Branch to On-Chip RAM 3-37
3-18 Pipeline Operation of RET from On-Chip RAM 3-38
3-19 Status Register Organization 3-42
3-20 Timer Block Diagram 3-45
3-21 Four-Phase Clock 3-49
3-22 BIO Timing Diagram 3-50
3-23 External Flag Timing Diagram 3-51
3-24 Interrupt Mask Register (IMR) 3-53
3-25 Internal Interrupt Logic Diagram 3-54
3-26 Interrupt Timing Diagram (TMS320C25) 3-55
3-27 The DRR and DXR Registers 3-57
3-28 Serial Port Block Diagram 3-58
3-29 Serial Port Transmit Timing Diagram 3-59
3-30 Serial Port Receive Timing Diagram 3-60
3-31 Burst-Mode Serial Port Transmit Operation 3-61
3-32 Burst-Mode Serial Port Receive Operation 3-61
3-33 Byte-Mode DRR Operation 3-62
3-34 Serial Port Transmit Continuous Operation (FSM = 1) 3-63
3-35 Serial Port Receive Continuous Operation (FSM = 1) 3-63
3-36 Serial Port Transmit Continuous Operation (FSM = 0) 3-65
3-37 Serial Port Receive Continuous Operation (FSM = 0) 3-65
3-38 Continuous Transmit Operation Initialization 3-67
3-39 Continuous Receive Operation Initialization 3-67
3-40 Synchronization Timing Diagram (TMS32020) 3-68
3-41 Synchronization Timing Diagram (TMS320C25) 3-69
3-42 Global Memory Access Timing 3-70
3-43 TMS320C25 Hold Timing Diagram 3-72
4-1 Direct Addressing Block Diagram 4-3
4-2 Indirect Addressing Block Diagram 4-4
5-1 On-Chip RAM Configurations 5-26
5-2 MACD Operation 5-36

5-3 Execution Time vs. Number of Multiply-Accumulates (TMS32020) 5-39
5-4 Execution Time vs. Number of Multiply-Accumulates (TMS320C25) 5-40
5-5 Program Memory vs. Number of Multiply-Accumulates 5-41
5-6 An In-Place DIT FFT with In-Order Outputs and Bit-Reversed Inputs 5-68
5-7 An In-Place DIT FFT with In-Order Inputs but Bit-Reversed Outputs 5-69
6-1 Powerup Reset Circuit 6-3
6-2 Voltage on TMS320C25 Reset Pin 6-4
6-3 Crystal Oscillator Circuit 6-5
6-4 Magnitude of Impedance of Oscillator LC Network 6-6
6-5 Direct Interface of TBP38L165-35 to TMS320C25 6-12
6-6 Interface Timing of TBP38L165-35 to TMS320C25 6-13
6-7 Interface of TBP38L165-35 to TMS320C25 6-14
6-8 Interface Timing of TBP38L165-35 to TMS320C25 (Address Decoding) . 6-15
6-9 One Wait-State Memory Access Timing 6-17
6-10 Wait-State Generator Design 6-18
6-11 Wait-State Generator Timing 6-19
6-12 Interface of WS57C65F-12 to TMS320C25 6-20
6-13 Interface Timing of WS57C65F-12 to TMS320C25 6-21
6-14 Interface of TMS27C64-20 to TMS320C25 6-22
6-15 Interface Timing of TMS27C64-20 to TMS320C25 6-23
6-16 Interface of CY7C169-25 to TMS320C25 6-25
6-17 Interface Timing of CY7C169-25 to TMS320C25 6-26
6-18 Direct Memory Access Using a Master-Slave Configuration 6-30
6-19 Direct Memory Access in a PC Environment 6-31
6-20 Global Memory Communication 6-33
6-21 Interface of TMS320C25 to TCM29C16 Codec 6-35
6-22 Interface of TLC32040 to TMS320C2x 6-38
6-23 Synchronous Timing of TLC32040 to TMS320C2x 6-38
6-24 Asynchronous Timing of TLC32040 to TMS320C2x 6-38
6-25 Interface of TLC7524 to TMS32020 6-39
6-26 Interface Timing of TLC7524 to TMS32020 6-40
6-27 Interface of TLC0820 to TMS32020 6-41
6-28 Interface Timing of TLC0820 to TMS32020 6-42
6-29 I/O Port Addressing 6-43
6-30 I/O Port Processor-to-Processor Communication 6-44
6-31 Echo Canceller 6-45
6-32 High-Speed Modem 6-46
6-33 Voice Coding System 6-46
6-34 Graphics System 6-47
6-35 Robot Axis Controller Subsystem 6-48
6-36 Instrumentation System 6-48
C-1 Serial Port System Migration C-8
E-1 TMS320C2x Development Tools E-1
E-2 TMS320C2x XDS/22 System Configuration E-6
E-3 TMS320 AIB System Configuration E-8
E-4 TMS320 Device Nomenclature E-13
E-5 TMS320 Development Tool Nomenclature E-14
F-1 Crystal Connection F-33
G-1 TMS320C2x ROM Code Flowchart G-2

Tables

Table *Page*

1-1 TMS320C2x Processors Overview 1-3
1-2 Typical Applications of the TMS320 Family 1-5
2-1 TMS320C2x Signal Descriptions 2-4
3-1 TMS320C2x Internal Hardware 3-8
3-2 Memory-Mapped Registers 3-16
3-3 PM Shift Modes 3-27
3-4 Instruction Pipeline Sequence 3-33
3-5 Status Register Field Definitions 3-43
3-6 Interrupt Locations and Priorities 3-52
3-7 Serial Port Bits, Pins, and Registers 3-56
3-8 Global Data Memory Configurations 3-70
4-1 Indirect Addressing Arithmetic Operations 4-6
4-2 Bit Fields for Indirect Addressing 4-7
4-3 Instruction Symbols 4-11
4-4 Instruction Set Summary 4-13
5-1 Program Space and Time Requirements for μ-/A-Law Companding 5-60
5-2 256-Tap Adaptive Filtering Memory Space and Time Requirements 5-67
5-3 Bit-Reversal Algorithm for an 8-Point Radix-2 DIT FFT 5-69
5-4 FFT Memory Space and Time Requirements 5-75
6-1 Timing Parameters of TBP38L165-35 Direct Interface to TMS320C25 6-13
6-2 Timing Parameters of TBP38L165-35 to TMS320C25 (Address Decoding) 6-15
6-3 Wait States Required for Memory/Peripheral Access 6-17
6-4 Timing Parameters of WS57C64F-12 Interface to TMS320C25 6-21
6-5 Timing Parameters of TMS27C64-20 Interface to TMS320C25 6-23
6-6 Timing Parameters of CY7C169-25 Interface to TMS320C25 6-24
D-1 TMS32020 Instructions by Cycle Class D-2
D-2 TMS32020 Instruction Cycle Timings D-3
D-3 TMS320C25 Instructions by Cycle Class D-4
D-4 Cycle Timings for Cycle Classes When Not in Repeat Mode D-5
D-5 Cycle Timings for Cycle Classes When in Repeat Mode D-7
E-1 TMS320C2x Digital Signal Processor Part Numbers E-11
E-2 TMS320C2x Support Tool Part Numbers E-11
E-3 Development Tool Connections to a Target System E-12
F-1 Commonly Used Crystal Frequencies F-33
H-1 Microprocessor and Microcontroller Tests H-5
H-2 TMS320C2x Transistors H-5

1. Introduction

The TMS320 family of 16/32-bit single-chip digital signal processors combines the flexibility of a high-speed controller with the numerical capability of an array processor, offering an inexpensive alternative to custom VLSI and multichip bit-slice processors for signal processing.

The TMS32010, the first digital signal processor in the TMS320 family, was introduced in 1983. Since that time, the TMS320 family has established itself as the industry standard for digital signal processing. The powerful instruction set, inherent flexibility, high-speed number-crunching capabilities, and innovative architecture have made the high-performance, cost-effective processors in the TMS320 family the ideal solution to many telecommunications, computer, commercial, industrial, and military applications.

The TMS320 family has now expanded into three generations of processors: TMS320C1x, TMS320C2x, and TMS320C3x (see Figure 1-1). Many features are common among these generations. Some specific features are added in each processor to provide different cost/performance tradeoffs. Software compatibility is maintained throughout the family to protect the user's investment in architecture. Each processor has software and hardware tools to facilitate rapid design.

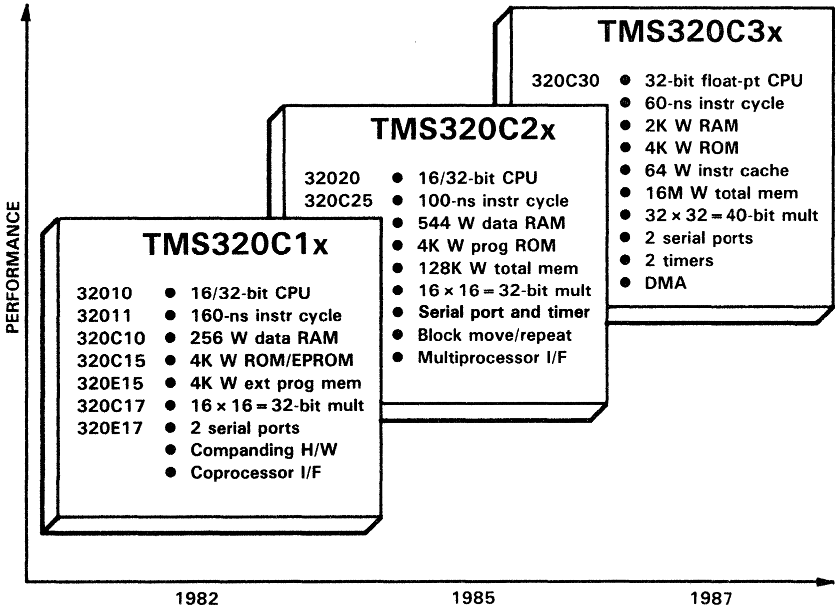


Figure 1-1. TMS320 Device Evolution

This document discusses the second-generation devices (TMS320C2x) within the TMS320 family. The specific members of the second-generation TMS320 include:

- TMS32020, an NMOS 20-MHz digital signal processor capable of twice the performance of the TMS320C1x devices, and
- TMS320C25, a CMOS 40-MHz version of the TMS32020 with twice the performance of the TMS32020.

Plans for expansion of the TMS320 family include more spinoffs of the existing generations as well as more powerful future generations of digital signal processors.

The TMS320 family combines the high performance and specialized features necessary in digital signal processing (DSP) applications with an extensive program of development support, including hardware and software development tools, product documentation, textbooks, newsletters, DSP design workshops, and a variety of application reports. See Appendix D for a discussion of the wide range of development tools available.

1.1 General Description

The combination of the TMS320's Harvard-type architecture (separate program and data buses) and its special digital signal processing (DSP) instruction set provide speed and flexibility to produce a microprocessor family capable of executing 10 MIPS (million instructions per second). The TMS320 family optimizes speed by implementing functions in hardware that other processors implement through software or microcode. This hardware-intensive approach provides the design engineer with power previously unavailable on a single chip.

The second generation of the TMS320 family includes two members, the TMS32020 and the TMS320C25. The architecture of these devices is based upon that of the TMS32010. Table 1-1 provides an overview of the TMS320C2x group of processors with comparisons of technology, memory, I/O, cycle timing, and package type.

Table 1-1. TMS320C2x Processors Overview

DEVICE	TECH	MEMORY ON-CHIP RAM	MEMORY ON-CHIP ROM	MEMORY OFF-CHIP PROG	MEMORY OFF-CHIP DATA	I/O† SER	I/O† PAR	I/O† DMA	CYCLE TIME (ns)	PACKAGE TYPE PGA	PACKAGE TYPE PLCC
TMS32020‡	NMOS	544	–	64K	64K	YES	16x16	YES	200	68	–
TMS320C25§	CMOS	544	4K	64K	64K	YES	16x16	CON	100	68	68

†SER = serial; PAR = parallel; DMA = direct memory access; CON = concurrent DMA.
‡Military version available; contact nearest TI sales office for details.
§Military version planned; contact nearest TI sales office for availability.

The **TMS32020**, processed in NMOS technology, is source-code upward compatible with the TMS32010 and in many applications is capable of two times the throughput of the TMS320C1x devices. It provides an enhanced instruction set (109 instructions), large on-chip data memory (544 words), large memory spaces, on-chip serial port, and a hardware timer.

The **TMS320C25**, the newest member of the TMS320 second generation, is processed in CMOS technology. The TMS320C25 is capable of executing many instructions in a 100-ns cycle time. It is pin-for-pin and object-code upward compatible with the TMS32020. The TMS320C25's enhanced feature set greatly increases the functionality of the device over the TMS32020. Enhancements include 24 additional instructions (133 total), eight auxiliary registers, an eight-level hardware stack, 4K words of on-chip program ROM, a bit-reversed indexed-addressing mode, and the low-power dissipation inherent to the CMOS process.

1.2 Key Features

Some of the key features of the TMS320C2x devices are listed below. Features specific to a particular device are noted by enclosing the device name in parentheses.

- Instruction cycle timing:
  - 100 ns (TMS320C25)
  - 200 ns (TMS32020)
- 544-word programmable on-chip data RAM
- 4K-word on-chip program ROM (TMS320C25)
- 128K-word total data/program memory space
- 32-bit ALU/accumulator
- 16 x 16-bit parallel multiplier with a 32-bit product
- Single-cycle multiply/accumulate instructions
- Repeat instructions for efficient use of program space and enhanced execution
- Block moves for data/program management
- On-chip timer for control operations
- Up to eight auxiliary registers with dedicated arithmetic unit
- Up to eight-level hardware stack
- Sixteen input and sixteen output channels
- 16-bit parallel shifter
- Wait states for communication to slower off-chip memories/peripherals
- Serial port for direct codec interface
- Synchronization input for synchronous multiprocessor configurations
- Global data memory interface
- TMS320C1x source-code upward compatibility
- Concurrent DMA using an extended hold operation (TMS320C25)
- Instructions for adaptive filtering, FFTs, and extended-precision arithmetic (TMS320C25)
- Bit-reversed indexed-addressing mode for radix-2 FFTs (TMS320C25)
- On-chip clock generator
- Single 5-V supply
- Device packaging:
  - 68-pin PGA
  - 68-lead PLCC (TMS320C25)
- Technology:
  - NMOS (TMS32020)
  - CMOS (TMS320C25)
- Commercial and military versions available.

1.3 Typical Applications

The TMS320 family's unique versatility and realtime performance offer flexible design approaches in a variety of applications. In addition, TMS320 devices can simultaneously provide the multiple functions often required in those complex applications. Table 1-2 lists typical TMS320 family applications.

Table 1-2. Typical Applications of the TMS320 Family

GENERAL-PURPOSE DSP	GRAPHICS/IMAGING	INSTRUMENTATION
Digital Filtering Convolution Correlation Hilbert Transforms Fast Fourier Transforms Adaptive Filtering Windowing Waveform Generation	3-D Rotation Robot Vision Image Transmission/ Compression Pattern Recognition Image Enhancement Homomorphic Processing Workstations Animation/Digital Map	Spectrum Analysis Function Generation Pattern Matching Seismic Processing Transient Analysis Digital Filtering Phase-Locked Loops
VOICE/SPEECH	**CONTROL**	**MILITARY**
Voice Mail Speech Vocoding Speech Recognition Speaker Verification Speech Enhancement Speech Synthesis Text-to-Speech	Disk Control Servo Control Robot Control Laser Printer Control Engine Control Motor Control	Secure Communications Radar Processing Sonar Processing Image Processing Navigation Missile Guidance Radio Frequency Modems
TELECOMMUNICATIONS		**AUTOMOTIVE**
Echo Cancellation ADPCM Transcoders Digital PBXs Line Repeaters Channel Multiplexing 1200 to 19200-bps Modems Adaptive Equalizers DTMF Encoding/Decoding Data Encryption	FAX Cellular Telephones Speaker Phones Digital Speech Interpolation (DSI) X.25 Packet Switching Video Conferencing Spread Spectrum Communications	Engine Control Vibration Analysis Antiskid Brakes Adaptive Ride Control Global Positioning Navigation Voice Commands Digital Radio Cellular Telephones
CONSUMER	**INDUSTRIAL**	**MEDICAL**
Radar Detectors Power Tools Digital Audio/TV Music Synthesizer Toys and Games Solid-State Answering Machines	Robotics Numeric Control Security Access Power Line Monitors	Hearing Aids Patient Monitoring Ultrasound Equipment Diagnostic Tools Prosthetics Fetal Monitors

Many of the TMS320C2x features, such as single-cycle multiply/accumulate instructions, 32-bit arithmetic unit, large auxiliary register file with a separate arithmetic unit, and large on-chip RAM and ROM, make the device particularly applicable in digital signal processing systems. At the same time, general-purpose applications are greatly enhanced by the large address spaces, on-chip timer, serial port, multiple interrupt structure, provision for external wait states, and capability for multiprocessor interface and direct memory access.

The TMS320C2x provides the flexibility to be configured to satisfy a wide range of system requirements. This allows the device to be applied in systems currently using costly bit-slice processors or custom ICs. Some of the system configurations are:

- A standalone system using on-chip memory,
- Parallel multiprocessing systems with shared global data memory, or
- Host/peripheral coprocessing using interface control signals.

1.4 How To Use This Manual

The purpose of this user's guide is to serve as a reference book for the TMS320C2x digital signal processors. Sections 2 through 6 provide specific information about the architecture and operation of the device. Electrical specifications and mechanical data can be found in the data sheet (Appendix A).

The following table lists each section and briefly describes the section contents.

Section 2.	Pinouts and Signal Descriptions. Drawings of the PGA and PLCC packages for TMS320C2x devices. Functional listings of the signals, their pin locations, and descriptions.
Section 3.	Architecture. TMS320C2x design description, hardware components, and device operation. Functional block diagram and internal hardware summary table.
Section 4.	Assembly Language Instructions. Addressing modes and format descriptions. Instruction set summary listed according to function. Alphabetized individual instruction descriptions with examples.
Section 5.	Software Applications. Software application examples for the use of various TMS320C2x instruction set features.
Section 6.	Hardware Applications. Hardware design techniques and application examples for interfacing to memories, peripherals, or other microcomputers/microprocessors. XDS design considerations. System applications.

Eight appendices are included to provide additional information.

Appendix A.	Second-Generation TMS320 Data Sheet. Electrical specifications, timing, and mechanical data for the TMS320C2x devices.
Appendix B.	SMJ32020 Data Sheet. Electrical specifications, timing, and mechanical data for this military devices.
Appendix C.	TMS320C2x System Migration. Information for upgrading a TMS320C1x to a TMS32020-based system and a TMS32020 to a TMS320C25-based system.
Appendix D.	TMS320C2x Instruction Cycle Timings. Listings of the number of cycles for an instruction to execute in a given memory configuration on the TMS32020 and the TMS320C25.
Appendix E.	Development Support/Part Order Information. Listings of the hardware and software available to support the TMS320C2x devices.

Appendix F.	Memories, Analog Converters, Sockets, and Crystals. Listings of the TI memories, analog conversion devices, and sockets available to support the TMS320C2x devices in DSP applications. Crystal specifications and vendors.
Appendix G.	ROM Codes. Discussion of ROM codes (mask options) and the procedure for implementation.
Appendix H.	Quality and Reliability. Discussion of Texas Instruments quality and reliability criteria for evaluating performance.

1.5 References

The following reference list contains useful information regarding functions, operations, and applications of digital signal processing. These books also provide other references to many useful technical papers. The reference list is organized into categories of general DSP, speech, image processing, and digital control theory, and alphabetized by author.

General Digital Signal Processing:

Antoniou, Andreas, *Digital Filters: Analysis and Design*. New York, NY: McGraw-Hill Company, Inc., 1979.

Brigham, E. Oran, *The Fast Fourier Transform*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1974.

Burrus, C.S. and Parks, T.W., *DFT/FFT and Convolution Algorithms*. New York, NY: John Wiley and Sons, Inc., 1984.

Digital Signal Processing Applications with the TMS320 Family, Texas Instruments, 1986; Prentice-Hall, Inc., 1987.

Gold, Bernard and Rabiner, Lawrence R., *Theory and Application of Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1975.

Gold, Bernard and Rader, C.M., *Digital Processing of Signals*. New York, NY: McGraw-Hill Company, Inc., 1969.

Hamming, R.W., *Digital Filters*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1977.

IEEE ASSP DSP Committee (Editor), *Programs for Digital Signal Processing*. New York, NY: IEEE Press, 1979.

Jackson, Leland B., *Digital Filters and Signal Processing*. Hingham, MA: Kluwer Academic Publishers, 1986.

Jones, D.L. and Parks, T.W., *A Digital Signal Processing Laboratory Using the TMS32010*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1987.

Morris, L. Robert, *Digital Signal Processing Software*. Ottawa, Canada: Carieton University, 1983.

Oppenheim, Alan V. (Editor), *Applications of Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1978.

Oppenheim, Alan V. and Schafer, R.W., *Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1975.

Oppenheim, Alan V. and Willsky, A.N. with Young, I.T., *Signals and Systems*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1983.

Parks, T.W. and Burrus, C.S., *Digital Filter Design*. New York, NY: John Wiley and Sons, Inc., 1987.

Speech:

Gray, A.H. and Markel, J.D., *Linear Prediction of Speech*. New York, NY: Springer-Verlag, 1976.

Jayant, N.S. and Noll, Peter, *Digital Coding of Waveforms*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1984.

Papamichalis, Panos, *Practical Approaches to Speech Coding*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1987.

Rabiner, Lawrence R. and Schafer, R.W., *Digital Processing of Speech Signals*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1978.

Image Processing:

Andrews, H.C. and Hunt, B.R., *Digital Image Restoration*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1977.

Gonzales, Rafael C. and Wintz, Paul, *Digital Image Processing*. Reading, MA: Addison-Wesley Publishing Company, Inc., 1977.

Pratt, William K., *Digital Image Processing*. New York, NY: John Wiley and Sons, 1978.

Digital Control Theory:

Jacquot, R., *Modern Digital Control Systems*. New York, NY: Marcel Dekker, Inc., 1981.

Katz, P., *Digital Control Using Microprocessors*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981.

Kuo, B.C., *Digital Control Systems*. New York, NY: Holt, Reinholt and Winston, Inc., 1980.

Moroney, P., *Issues in the Implementation of Digital Feedback Compensators*. Cambridge, MA: The MIT Press, 1983.

Phillips, C. and Nagle, H., *Digital Control System Analysis and Design*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1984.

2. Pinouts and Signal Descriptions

The TMS320C2x (second-generation TMS320) digital signal processors are available in a 68-pin grid array (PGA) package. The TMS320C25 is also packaged in a 68-pin plastic-leaded chip carrier (PLCC).

Adaptor sockets are commercially available to convert a TMS320C25 PLCC package to a TMS32020-like 68-pin grid array (PGA) footprint, thus maintaining plug-in compatibility.

When using the XDS emulator, refer to Section 6.1.3 for user target design considerations

This section provides the pinouts and signal definitions in the following subsections:

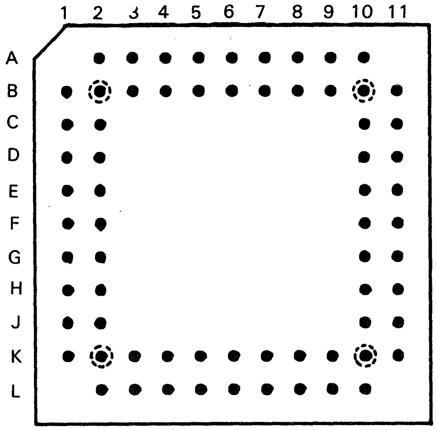
- TMS320C2x Pinouts (Section 2.1 on page 2-2)
- TMS320C2x Signal Descriptions (Section 2.2 on page 2-3)

Electrical specifications and mechanical data are given in Appendix A, the Second-Generation TMS320 Data Sheet.

2.1 TMS320C2x Pinouts

Figure 2-1 shows pinouts of the PGA packages for the TMS320C2x devices and the PLCC package for the TMS320C25. Refer to Section 6.1.3 for user target system design considerations when using the XDS.

68-PIN GB
PIN GRID ARRAY CERAMIC PACKAGE
(TOP VIEW)



68-PIN FN
PLASTIC LEADED CHIP CARRIER PACKAGE
(TOP VIEW)

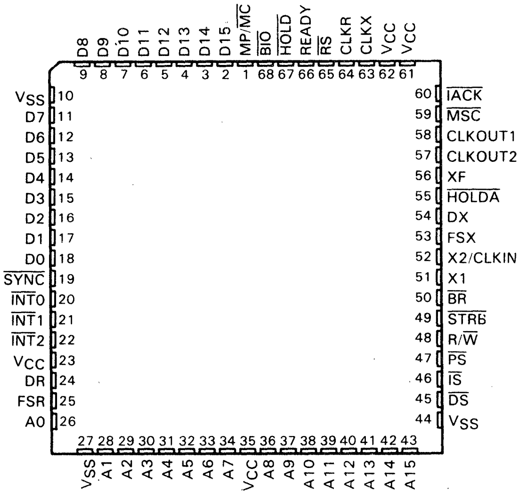


Figure 2-1. TMS320C2x Pin Assignments

2.2 TMS320C2x Signal Descriptions

The signal descriptions for the TMS320C2x devices are provided in this section. Table 2-1 lists each signal, its pin location (PGA/PLCC), function, and operating mode(s), i.e., input, output, or high-impedance state as indicated by I, O, or Z. The signals in Table 2-1 are grouped according to function and alphabetized within that grouping.

Table 2-1. TMS320C2x Signal Descriptions

SIGNAL	PIN (PGA/PLCC)	I/O/Z†	DESCRIPTION
ADDRESS/DATA BUSES			
A15 MSB	L10/43	O/Z	Parallel address bus A15 (MSB) through A0 (LSB). Multiplexed to address external data/program memory or I/O. Placed in high-impedance state in the hold mode.
A14	K9/42		
A13	L9/41		
A12	K8/40		
A11	L8/39		
A10	K7/38		
A9	L7/37		
A8	K6/36		
A7	K5/34		
A6	L5/33		
A5	K4/32		
A4	L4/31		
A3	K3/30		
A2	L3/29		
A1	K2/28		
A0 LSB	K1/26		
D15 MSB	B6/2	I/O/Z	Parallel data bus D15 (MSB) through D0 (LSB). Multiplexed to transfer data between the TMS320C2x and external data/program memory or I/O devices. Placed in high-impedance state when not outputting or when $\overline{RS}$ or $\overline{HOLD}$ is asserted.
D14	A5/3		
D13	B5/4		
D12	A4/5		
D11	B4/6		
D10	A3/7		
D9	B3/8		
D8	A2/9		
D7	B2/11		
D6	C1/12		
D5	C2/13		
D4	D1/14		
D3	D2/15		
D2	E1/16		
D1	E2/17		
D0 LSB	F1/18		
INTERFACE CONTROL SIGNALS			
$\overline{DS}$ $\overline{PS}$ $\overline{IS}$	K10/45 J10/47 J11/46	O/Z	Data, program, and I/O space select signals. Always high unless low level asserted for communicating to a particular external space. Placed in high-impedance state in the hold mode.
READY	B8/66	I	Data ready input. Indicates that an external device is prepared for the bus transaction to be completed. If the device is not ready (READY = 0), the TMS320C2x waits one cycle and checks READY again. READY also indicates a bus grant to an external device after a $\overline{BR}$ (bus request) signal.
R/$\overline{W}$	H11/48	O/Z	Read/write signal. Indicates transfer direction when communicating to an external device. Normally in read mode (high), unless low level asserted for performing a write operation. Placed in high-impedance state in the hold mode.
$\overline{STRB}$	H10/49	O/Z	Strobe signal. Always high unless asserted low to indicate an external bus cycle. Placed in high-impedance state in the hold mode.

† Input/Output/High-impedance state

Table 2-1 TMS320C2x Signal Descriptions (Continued)

SIGNAL	PIN (PGA/PLCC)	I/O/Z†	DESCRIPTION
MULTIPROCESSING SIGNALS			
$\overline{\mathrm{BR}}$	G11/50	O	Bus request signal. Asserted when the TMS320C2x requires access to an external global data memory space. READY is asserted to the device when the bus is available and the global data memory is available for the bus transaction.
$\overline{\mathrm{HOLD}}$	A7/67	I	Hold input. When asserted, the TMS320C2x places the data, address, and control lines in the high-impedance state.
$\overline{\mathrm{HOLDA}}$	E10/55	O	Hold acknowledge signal. Indicates that the TMS320C2x has gone into the hold mode and that an external processor may access the local external memory of the TMS320C2x.
$\overline{\mathrm{SYNC}}$	F2/19	I	Synchronization input. Allows clock synchronization of two or more TMS320C2x's. $\overline{\mathrm{SYNC}}$ is an active-low signal and must be asserted on the rising edge of CLKIN.
INTERRUPT AND MISCELLANEOUS SIGNALS			
$\overline{\mathrm{BIO}}$	B7/68	I	Branch control input. Polled by BIOZ instruction. If low, the TMS320C2x executes a branch. This signal must be active during the BIOZ instruction fetch.
$\overline{\mathrm{IACK}}$	B11/60	O	Interrupt acknowledge signal. Output is only valid while CLKOUT1 is low. Indicates receipt of an interrupt and that the program is branching to the interrupt-vector location indicated by A15-A0.
$\overline{\mathrm{INT}}2$ $\overline{\mathrm{INT}}1$ $\overline{\mathrm{INT}}0$	H1/22 G2/21 G1/20	I	External user interrupt inputs. Prioritized and maskable by the interrupt mask register and the interrupt mode bit.
MP/$\overline{\mathrm{MC}}$	A6/1	I	Microprocessor/microcomputer mode select pin for the TMS320C25 only. When asserted low (microcomputer mode), the pin causes the internal ROM to be mapped into the lower 4K words of the program memory map. In the microprocessor mode, the lower 4K words of program memory are external. On the TMS32020, MP/$\overline{\mathrm{MC}}$ must be connected to V_{CC}.
$\overline{\mathrm{MSC}}$	C10/59	O	Microstate complete signal. Asserted low and valid only during CLKOUT1 low when the TMS320C2x has just completed a memory operation, such as an instruction fetch or a data memory read/write. $\overline{\mathrm{MSC}}$ can be used to generate a one wait-state READY signal for slow memory.
$\overline{\mathrm{RS}}$	A8/65	I	Reset input. Causes the TMS320C2x to terminate execution and forces the program counter to zero. When brought to a high level, execution begins at location zero of program memory. $\overline{\mathrm{RS}}$ affects various registers and status bits.
XF	D11/56	O	External flag output (latched software-programmable signal). Used for signalling other processors in multiprocessor configurations or as a general-purpose output pin.

† Input/Output/High-impedance state

Table 2-1. TMS320C2x Signal Descriptions (Concluded)

SIGNAL	PIN (PGA/PLCC)	I/O/Z†	DESCRIPTION
			SUPPLY/OSCILLATOR SIGNALS
CLKOUT1	C11/58	O	Master clock output signal (CLKIN frequency/4). In this document (and on the TMS320C25), CLKOUT1 rises at the beginning of quarter-phase 3 (Q3) and falls at the beginning of quarter-phase 1 (Q1). See Appendix C for device phase definitions.
CLKOUT2	D10/57	O	A second clock output signal. In this document (and on the TMS320C25), CLKOUT2 rises at the beginning of quarter-phase 2 (Q2) and falls at beginning of quarter-phase 4 (Q4). See Appendix C for device phase definitions.
V_{CC}	A10/61 B10/62 H2/23 L6/35	I	Four 5-V supply pins, tied together externally. On the TMS32020, pin A6 is also a supply pin.
V_{SS}	B1/10 K11/44 L2/27	I	Three ground pins, tied together externally.
X1	G10/51	O	Output pin from the internal oscillator for the crystal. If a crystal is not used, this pin should be left unconnected.
X2/CLKIN	F11/52	I	Input pin to the internal oscillator from the crystal. If a crystal is not used, a clock may be input to the device on this pin.
			SERIAL PORT SIGNALS
CLKR	B9/64	I	Receive clock input. External clock signal for clocking data from the DR (data receive) pin into the RSR (serial port receive shift register). Must be present during serial port transfers.
CLKX	A9/63	I	Transmit clock input. External clock signal for clocking data from the XSR (serial port transmit shift register) to the DX (data transmit) pin. Must be present during serial port transfers.
DR	J1/24	I	Serial data receive input. Serial data is received in the RSR (serial port receive shift register) via the DR pin.
DX	E11/54	O/Z	Serial data transmit output. Serial data transmitted from the XSR (serial port transmit shift register) via the DX pin. Placed in high-impedance state when not transmitting.
FSR	J2/25	I	Frame synchronization pulse for receive input. The falling edge of the FSR pulse initiates the data-receive process, beginning the clocking of the RSR.
FSX	F10/53	I/O	Frame synchronization pulse for transmit input/output. The falling edge of the FSX pulse initiates the data-transmit process, beginning the clocking of the XSR. Following reset, the default operating condition of FSX is as an input. This pin may be selected by software to be an output when the TXM bit in the status register is set to 1.

† Input/Output/High-impedance state

3. Architecture

The architectural design of the TMS320C2x (second-generation TMS320) emphasizes overall system speed, communication, and flexibility in processor configuration. Control signals and instructions provide block memory transfers, communication to slower off-chip devices, and multiprocessing implementations. Increased throughput for many DSP applications is accomplished by single-cycle multiply/accumulate instructions, two large on-chip RAM blocks, eight auxiliary registers with a dedicated arithmetic unit, a serial port, hardware timer, faster I/O for data-intensive signal processing, and other features.

Major topics discussed in this section are listed below.

- Architectural Overview (Section 3.1 on page 3-2)
- Functional Block Diagram (Section 3.2 on page 3-5)
- Internal Hardware Summary (Section 3.3 on page 3-7)
- Memory Organization (Section 3.4 on page 3-11)
    - Data memory and program memory
    - Memory maps and memory-mapped registers
    - Auxiliary registers
    - Memory addressing modes
    - Memory-to-memory moves
- Central Arithmetic Logic Unit (CALU) (Section 3.5 on page 3-22)
    - Scaling shifter, ALU, and accumulator
    - Multiplier, T and P registers
- System Control (Section 3.6 on page 3-28)
    - Program counter and stack
    - Pipeline operation
    - Reset
    - Status registers
    - Timer operation
    - Repeat counter
    - Powerdown mode
- External Memory and I/O Interface (Section 3.7 on page 3-39)
    - Memory combinations
    - Internal clock timing relationships
    - External read and write cycles
    - General-purpose I/O pins (BIO and XF)
- Interrupts (Section 3.8 on page 3-47)
    - Interrupt operation
    - External interrupt interface
- Serial Port (Section 3.9 on page 3-51)
    - Transmit and receive operations
    - Timing and framing control
    - Burst mode and continuous mode operation
- Multiprocessing and Direct Memory Access (Section 3.10 on page 3-63)
    - Synchronization
    - Global memory
    - The hold function

3.1 Architectural Overview

The TMS320C2x high-performance digital signal processors, like the TMS320C1x devices, implement a Harvard-type architecture that maximizes processing power by maintaining two separate memory bus structures, program and data, for full-speed execution. Instructions are included to provide data transfers between the two spaces. Externally, the program and data memory can be multiplexed over the same bus so as to maximize the address range for both spaces while minimizing the pin count of the device.

Increased flexibility in system design is provided by two large on-chip data RAM blocks (a total of 544 16-bit words), one of which is configurable either as program or data memory (see Figure 3-1). An off-chip 64K-word directly addressable data memory address space is included to facilitate implementations of DSP algorithms.

The large on-chip 4K-word masked ROM on the TMS320C25 can be used to cost-reduce systems, thus providing for a true single-chip DSP solution (see Figure 3-1). Programs of up to 4K words can be masked into the internal program ROM. The remainder of the 64K-word program memory space is located externally. Large programs can execute at full speed from this memory space. Programs may also be downloaded from slow external memory to on-chip RAM for full-speed operation.

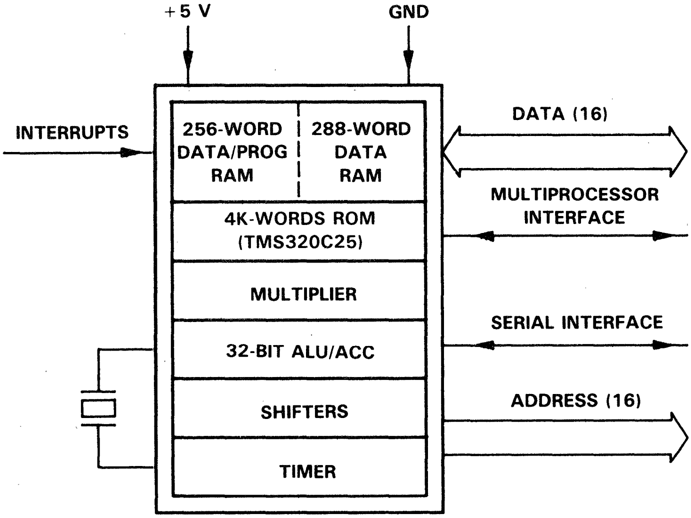


Figure 3-1. TMS320C2x Simplified Block Diagram

The TMS320C2x performs two's-complement arithmetic using the 32-bit ALU and accumulator. The ALU is a general-purpose arithmetic unit that operates using 16-bit words taken from data RAM or derived from immediate instructions or using the 32-bit result of the multiplier's product register. In addition to the usual arithmetic instructions, the ALU can perform Boolean operations, providing the bit manipulation ability required of a high-speed controller. The accumulator stores the output from the ALU and is the second input to the ALU. The accumulator is 32 bits in length and is divided into a high-order word (bits 31 through 16) and a low-order word (bits 15 through 0). Instructions are provided for storing the high- and low-order accumulator words in memory.

The multiplier performs a 16 x 16-bit two's-complement multiplication with a 32-bit result in a single instruction cycle. The multiplier consists of three elements: the T Register, P Register, and multiplier array. The 16-bit T Register temporarily stores the multiplicand; the P Register stores the 32-bit product. Multiplier values either come from data memory, from program memory when using the MAC/MACD instructions, or are derived immediately from the MPYK (multiply immediate) instruction word. The fast on-chip multiplier allows the device to efficiently perform fundamental DSP operations such as convolution, correlation, and filtering.

The TMS320C2x scaling shifter has a 16-bit input connected to the data bus and a 32-bit output connected to the ALU. The scaling shifter produces a left-shift of 0 to 16 bits on the input data, as programmed in the instruction. The LSBs of the output are filled with zeros, and the MSBs may be either filled with zeros or sign-extended, depending upon the state of the sign-extension mode bit of status register ST1. Additional shift capabilities enable the processor to perform numerical scaling, bit extraction, extended arithmetic, and overflow prevention.

The TMS320C2x local memory interface consists of a 16-bit parallel data bus (D15-D0), a 16-bit address bus (A15-A0), three pins for data/program memory or I/O space select ($\overline{\text{DS}}$, $\overline{\text{PS}}$, and $\overline{\text{IS}}$), and various system control signals. The R/$\overline{\text{W}}$ signal controls the direction of a data transfer, and the $\overline{\text{STRB}}$ signal provides a timing signal to control the transfer. When using on-chip program RAM, ROM, or high-speed external program memory, the TMS320C2x runs at full speed without wait states. The use of a READY signal allows wait-state generation for communicating with slower off-chip memories.

Up to eight levels of hardware stack are provided for saving the contents of the program counter during interrupts and subroutine calls. Instructions are available for saving the device's complete context. PUSH and POP instructions permit a level of nesting restricted only by the amount of available RAM. The interrupts used in these devices are maskable.

Control operations are supported on the TMS320C2x by an on-chip memory-mapped 16-bit timer, a repeat counter, three external maskable user interrupts, and internal interrupts generated by serial port operations or by the timer. A built-in mechanism protects from those instructions that are repeated or become multicycle due to the READY signal and from holds and interrupts.

An on-chip full-duplex serial port provides direct communication with serial devices such as codecs, serial A/D converters, and other serial systems. The interface signals are compatible with codecs and many other serial devices with a minimum of external hardware. The two serial port memory-mapped registers (the data transmit/receive registers) may be operated in either an 8-bit byte or 16-bit word mode. Each register has an external clock input, a framing synchronization input, and associated shift registers.

Serial communication can be used between processors in multiprocessing applications. The TMS320C2x has the capability of allocating global data memory space and communicating with that space via the $\overline{\text{BR}}$ (bus request) and READY control signals. The 8-bit memory-mapped global memory allocation register (GREG) specifies up to 32K words of the TMS320C2x data memory as global external memory. The contents of the register determine the size of the global memory space. If the current instruction addresses an operand within that space, $\overline{\text{BR}}$ is asserted to request control of the bus. The length of the memory cycle is controlled by the READY line.

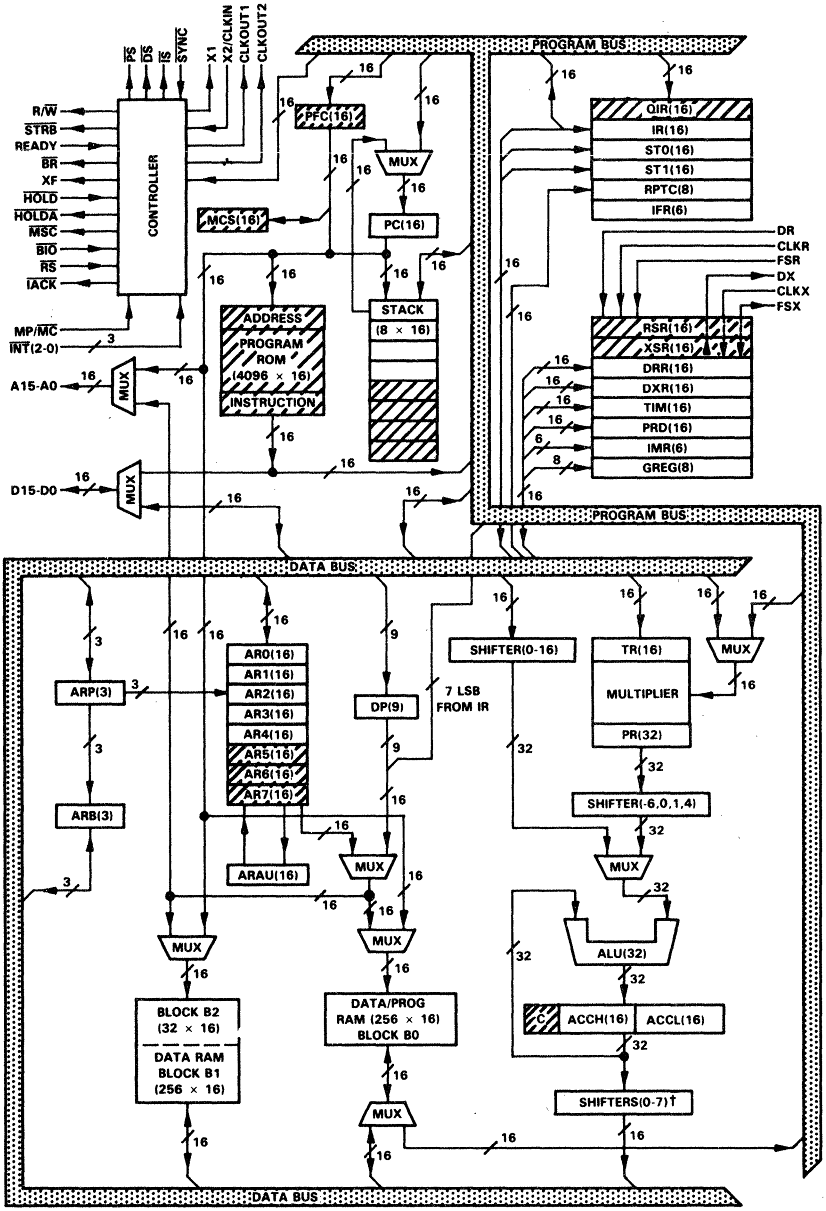
The TMS320C2x supports Direct Memory Access (DMA) to its external program/data memory using the $\overline{\text{HOLD}}$ and $\overline{\text{HOLDA}}$ signals. Another processor can take complete control of the TMS320C2x external memory by asserting $\overline{\text{HOLD}}$ low. This causes the TMS320C2x to place its address, data, and control lines in the high-impedance state. Signaling between the external processor and the TMS320C2x can be performed using interrupts. On the TMS320C25, two modes are available: a TMS32020-like mode in which execution is suspended during assertion of $\overline{\text{HOLD}}$, and a concurrent DMA mode in which the TMS320C25 continues to execute its program while operating from internal RAM or ROM, thus greatly increasing throughput in data-intensive applications.

3.2 Functional Block Diagram

The functional block diagram shown in Figure 3-2 outlines the principal blocks and data paths within the TMS320C2x processors. Further details of the functional blocks are provided in the succeeding sections. Refer to Section 3.3, the internal hardware summary, for definitions of the symbols used in Figure 3-2. The block diagram also shows all of the TMS320C2x interface pins. Note that the shaded areas on the block diagram indicate enhancements provided on the TMS320C25.

The TMS320C2x architecture is built around two major buses: the program bus and the data bus. The program bus carries the instruction code and immediate operands from program memory. The data bus interconnects various elements, such as the Central Arithmetic Logic Unit (CALU) and the auxiliary register file, to the data RAM. Together, the program and data buses can carry data from on-chip data RAM and internal or external program memory to the multiplier in a single cycle for multiply/accumulate operations.

The TMS320C2x has a high degree of parallelism; e.g., while the data is being operated upon by the CALU, arithmetic operations may also be implemented in the Auxiliary Register Arithmetic Unit (ARAU). Such parallelism results in a powerful set of arithmetic, logic, and bit-manipulation operations that may all be performed in a single machine cycle.



†Shifters on TMS32020 (0, 1, 4)
NOTE: Shaded areas are for TMS320C25 only.

Figure 3-2. TMS320C2x Block Diagram

3.3 Internal Hardware Summary

The TMS320C2x internal hardware implements functions that other processors typically perform in software or microcode. For example, the device contains hardware for single-cycle 16 x 16-bit multiplication, data shifting, and address manipulation. This hardware-intensive approach provides computing power previously unavailable on a single chip.

Table 3-1 presents a summary of the TMS320C2x internal hardware. This summary table, which includes the internal processing elements, registers, and buses, is alphabetized within each functional grouping. All of the symbols used in this table correspond to the symbols used in the block diagram of Section 3.2, the succeeding block diagrams in this section, and the text throughout this document.

Table 3-1. TMS320C2x Internal Hardware

UNIT	SYMBOL	FUNCTION
Accumulator	ACC (31-0) ACCH(31-16) ACCL(15-0)	A 32-bit accumulator split in two halves: ACCH (accumulator high) and ACCL (accumulator low). Used for storage of ALU output.
Arithmetic Logic Unit	ALU	A 32-bit two's-complement arithmetic logic unit having two 32-bit input ports and one 32-bit output port feeding the accumulator.
Auxiliary Register Arithmetic Unit	ARAU	A 16-bit unsigned arithmetic unit used to perform operations on auxiliary register data.
Auxiliary Register File	AR0-AR7 (15-0)	A register file containing five/eight 16-bit auxiliary registers (AR0-AR7), used for addressing data memory, temporary storage, or integer arithmetic processing through the ARAU.
Auxiliary Register File Bus	AFB(15-0)	A 16-bit bus that carries data from the AR pointed to by the ARP.
Auxiliary Register Pointer	ARP(2-0)	A 3-bit register used to select one of five/eight auxiliary registers.
Auxiliary Register Pointer Buffer	ARB(2-0)	A 3-bit register used to buffer the ARP. Each time the ARP is loaded, the old value is written to the ARB, except during an LST (load status register) instruction. When the ARB is loaded with an LST1, the same value is also copied into ARP.
Central Arithmetic Logic Unit	CALU	The grouping of the ALU, multiplier, accumulator, and scaling shifter.
Data Bus	D(15-0)	A 16-bit bus used to route data.
Data Memory Address Bus	DAB(15-0)	A 16-bit bus that carries the data memory address.
Data Memory Page Pointer	DP(8-0)	A 9-bit register pointing to the address of the current page. Data pages are 128 words each, resulting in 512 pages of addressable data memory space (some locations are reserved).
Direct Data Memory Address Bus	DRB(15-0)	A 16-bit bus that carries the 'direct' address for the data memory, which is the concatenation of the DP register with the seven LSBs of the instruction.
Global Memory Allocation Register	GREG(7-0)	An 8-bit memory-mapped register for allocating the size of the global memory space.
Instruction Register	IR(15-0)	A 16-bit register used to store the currently executing instruction.
Interrupt Flag Register	IFR(5-0)	A 6-bit flag register used to latch the active-low external user interrupts $\overline{\text{INT}}$(2-0) and the internal interrupts XINT/RINT (serial port transmit/receive) and TINT (timer) interrupts. The IFR is not accessible through software.
Interrupt Mask Register	IMR(5-0)	A 6-bit memory-mapped register used to mask interrupts.

†TMS320C25 only.

Table 3-1. TMS320C2x Internal Hardware (Continued)

UNIT	SYMBOL	FUNCTION
Microcall Stack†	MCS (15-0)	A single-word stack that temporarily stores the contents of the PFC while the PFC is being used to address data memory with the block move (BLKD/BLKP), multiply-accumulate (MAC/MACD), and table read/write (TBLR/TBLW) instructions.
Multiplier	MULT	A 16 x 16-bit parallel multiplier.
Period Register	PRD (15-0)	A 16-bit memory-mapped register used to reload the timer.
Prefetch Counter†	PFC (15-0)	A 16-bit counter used to prefetch program instructions. The PFC contains the address of the instruction currently being prefetched. It is updated when a new prefetch is initiated. The PFC is also used to address data memory when using the block move (BLKD/BLKP), multiply-accumulate (MAC/MACD), and table read/write (TBLR/TBLW) instructions.
Product Register	PR(31-0)	A 32-bit product register used to hold the multiplier product. The PR on the TMS320C25 can also be accessed as the most or least significant words using the SPH/SPL (store P register high/low) instructions.
Program Bus	P(15-0)	A 16-bit bus used to route instructions (and data for the MAC and MACD instructions).
Program Counter	PC (15-0)	A 16-bit program counter used to address program memory. The PC always contains the address of the next instruction to be executed. The PC contents are updated following each instruction decode operation. On the TMS32020, the operations of the TMS320C25 prefetch counter are performed by the program counter.
Program Memory Address Bus	PAB(15-0)	A 16-bit bus that carries the program memory address.
Queue Instruction Register†	QIR(15-0)	A 16-bit register used to store prefetched instructions.
Random Access Memory (data or program)	RAM (B0)	A RAM block with 256 x 16 locations configured either as data or program memory.
Random Access Memory (data only)	RAM (B1)	A data RAM block, organized as 256 x 16 locations.
Random Access Memory (data only)	RAM (B2)	A data RAM block, organized as 32 x 16 locations.
Repeat Counter	RPTC (7-0)	An 8-bit counter to control the repeated execution of a single instruction.
Serial Port Data Receive Register	DRR(15-0)	A 16-bit memory-mapped serial port data receive register. Only the eight LSBs are used in the byte mode.
Serial Port Data Transmit Register	DXR(15-0)	A 16-bit memory-mapped serial port data transmit register. Only the eight LSBs are used in the byte mode.
Serial Port Receive Shift Register†	RSR(15-0)	A 16-bit register used to shift in serial port data from the RX pin. RSR contents are sent to the DRR after a serial transfer is completed. RSR is not directly accessible through software.

†TMS320C25 only.

Table 3-1 TMS320C2x Internal Hardware (Concluded)

UNIT	SYMBOL	FUNCTION
Serial Port Transmit Shift Register†	XSR(15-0)	A 16-bit register used to shift out serial port data onto the DX pin. XSR contents are loaded from DXR at the beginning of a serial port transmit operation. XSR is not directly accessible through software.
Shifters	–	Shifters are located at the ALU input, the accumulator output, and the product register output. An in-place shifter is also located within the accumulator.
Stack	Stack(15-0)	A 4/8 x 16 hardware stack used to store the PC during interrupts or calls. The ACCL and data memory values may also be pushed onto and popped from the stack.
Status Registers	ST0,ST1 (15-0)	Two 16-bit status registers that contain status and control bits.
Temporary Register	TR(15-0)	A 16-bit register that holds either an operand for the multiplier or a shift code for the scaling shifter.
Timer	TIM (15-0)	A 16-bit memory-mapped timer (counter) for timing control.

†TMS320C25 only.

3.4 Memory Organization

The TMS320C2x provides a total of 544 16-bit words of on-chip data RAM, of which 288 words are always data memory and the remaining 256 words may be configured as either program or data memory. The TMS320C25 also provides 4K words of maskable program ROM. This section explains memory management using the on-chip data and program memory, memory maps, memory-mapped registers, auxiliary registers, memory addressing modes, and memory-to-memory moves.

3.4.1 Data Memory

The 544 words of on-chip data RAM are divided into three separate blocks (B0, B1, and B2), as shown in Figure 3-3. Of the 544 words, 256 words (block B0) are configurable as either data or program memory by instructions provided for that purpose; 288 words (blocks B1 and B2) are always data memory. A data memory size of 544 words allows the TMS320C2x to handle a data array of 512 words (256 words if on-chip RAM is used for program memory), while still leaving 32 locations for intermediate storage. See Section 3.4.3 for memory map configurations.

The TMS320C2x can address a total of 64K words of data memory. The on-chip data memory and internally reserved locations are mapped into the lower 1K words of the data memory space. Data memory is directly expandable up to 64K words while still maintaining full-speed operation. A READY line is provided for interface to slower, less-expensive memories, such as DRAMs.

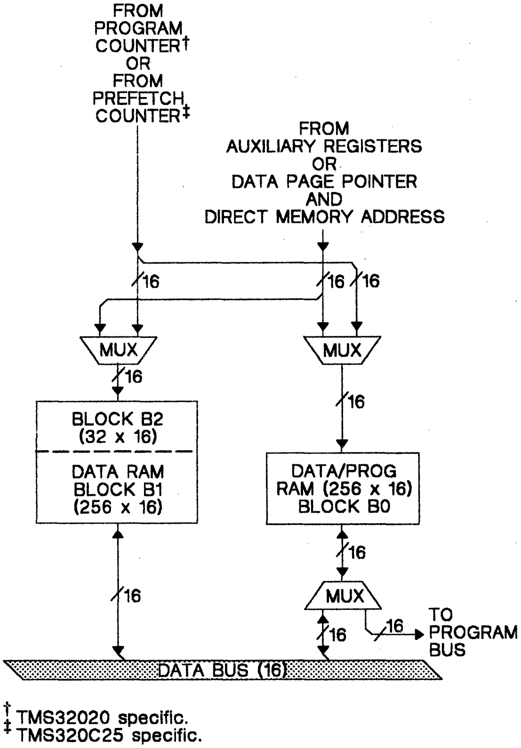


Figure 3-3. On-Chip Data Memory

3.4.2 Program Memory

On-chip program RAM, ROM, or high-speed external program memory can be used at full speed with no wait states. Alternatively, the READY line can interface the TMS320C2x to slower, less-expensive external memory. A total of 64K words of memory space is available. Internal RAM block B0 can be configured as program memory using instructions for that purpose. Execution from this block can be initiated after the memory space has been reconfigured. See Section 3.7.1 for a description of instruction execution using various memory configurations.

In addition, the TMS320C25 is equipped with 4K words of on-chip program ROM that can be mask-programmed at the factory with a customer's program. The on-chip ROM allows program execution at full speed without the need for high-speed external program memory. The use of this memory also allows the external data bus to be freed for access of external data memory.

Mapping of the first 4K-word block of off-chip/on-chip program memory is user-selectable by means of the MP/$\overline{\text{MC}}$ (microprocessor/microcomputer) pin on the TMS320C25. This permits the designer to accelerate time-to-market with a TMS320C25-based product by using external ROM, and cost-reducing it later with the 4K internal ROM without any PC-board redesign. Setting MP/$\overline{\text{MC}}$ high maps in the block of off-chip memory; holding the pin low maps in the block of on-chip ROM. The XF (external flag) pin can be used to toggle the MP/$\overline{\text{MC}}$ pin to dynamically enable or disable the on-chip ROM. Note that care must be taken and instruction pipeline operation (see Section 3.6.2) understood when using bank switching.

The MP/$\overline{\text{MC}}$ pin on the TMS320C25 is a V_{CC} pin on the TMS32020. This allows substitution of a TMS320C25 for a TMS32020 since the TMS320C25 automatically operates in the microprocessor mode and therefore is plug-in compatible in the system. See Section 2 for pinouts and signal descriptions.

3.4.3 Memory Maps

The TMS320C2x provides three separate address spaces for program memory, data memory, and I/O, as shown in Figure 3-4. These spaces are distinguished externally by means of the $\overline{\text{PS}}$, $\overline{\text{DS}}$, and $\overline{\text{IS}}$ (program, data, and I/O space select) signals. The $\overline{\text{PS}}$, $\overline{\text{DS}}$, $\overline{\text{IS}}$, and $\overline{\text{STRB}}$ signals are only active when external memory is being addressed. During an internal addressing cycle, these signals remain inactive high, thus preventing conflicts in memory addressing, e.g., when block B0 is configured as program memory.

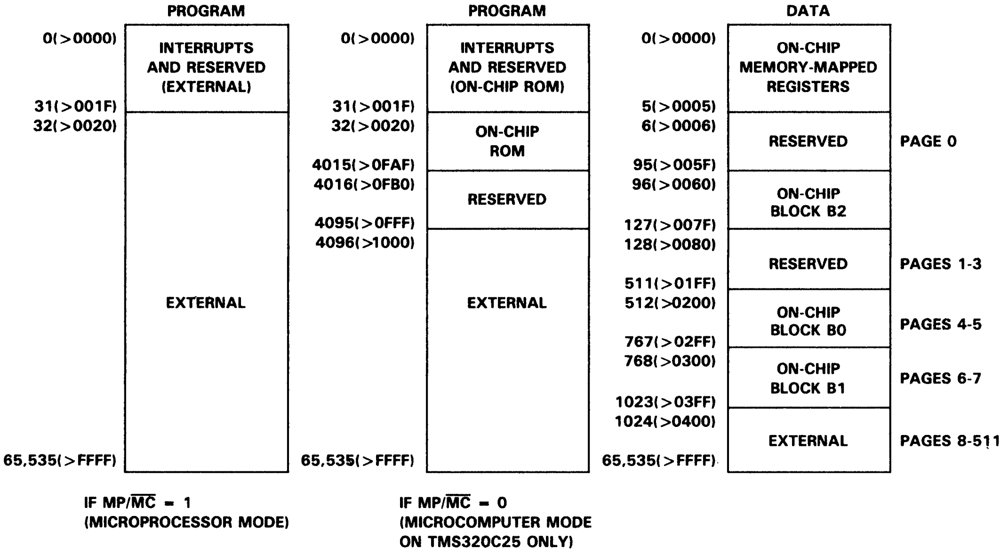
The on-chip memory blocks B0, B1, and B2 are comprised of a total of 544 words of RAM. Program/data RAM block B0 (256 words) resides in pages 4 and 5 of the data memory map when configured as data RAM and at addresses >FF00 to >FFFF when configured as program RAM. Block B1 (always data RAM) resides in pages 6 and 7, while block B2 resides in the upper 32 words of page 0. Note that the remainder of page 0 is composed of the memory-mapped registers and internally reserved locations, and pages 1-3 of the data memory map consist of internally reserved locations. The internally reserved locations may not be used for storage, and their contents are undefined when read. See Section 3.4.4 for further information on the memory-mapped registers.

The on-chip RAM is mapped into either the 64K-word data memory or program memory space, depending on the memory configuration (see Figure 3-4). The CNFD/CNFP instructions are used to configure block B0 as either data or program memory, respectively. The BLKP (block move from program memory to data memory) instruction may be used to download program information to block B0 when it is configured as data RAM. Then a CNFP (configure block as program memory) instruction may be used to convert it to program RAM (see the code example in Section 5.4.2). Regardless of the configuration, the user may still execute from external program memory. Note that when accessing internal program memory, external control lines remain inactive.

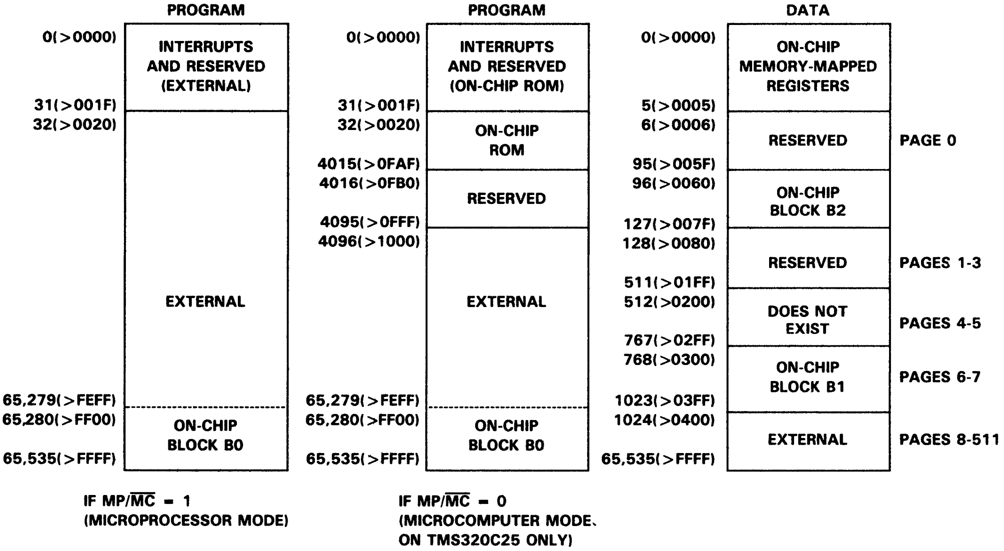
Reset configures block B0 as data RAM. Note that, due to internal pipelining, when the CNFD or CNFP instruction is used to remap RAM block B0, there is a delay before the new configuration becomes effective. This delay is one fetch cycle if execution is from internal program RAM. On the TMS32020, a delay of one fetch cycle occurs if execution is from external program memory. On the TMS320C25, there is a delay of two fetch cycles if execution is from

ROM or external program memory. This is particularly important if program execution is from the locations around >FF00. Accordingly, a CNFP instruction must be placed at location >FEFD in external memory if execution is to continue from the first location in block B0. If a CNFP is placed at location >FEFD, and the instruction at location >FEFF is a two-word instruction, the second word of the instruction will be fetched from the first location in block B0. If execution is from above location >FF00 and block B0 is reconfigured, care must be taken to assure that execution resumes at the appropriate point in a new configuration.

On-chip program ROM on the TMS320C25 is located in the lower 4K words of program memory when selected by setting MP/$\overline{\text{MC}}$ = 0. When MP/$\overline{\text{MC}}$ = 1, the lower 4K words of program memory are external.



(a) MEMORY MAPS AFTER A CNFD INSTRUCTION



(b) MEMORY MAPS AFTER A CNFP INSTRUCTION

Figure 3-4. Memory Maps

3.4.4 Memory-Mapped Registers

The six registers mapped into the data memory space are listed in Table 3-2 and are shown in the block diagram of Figure 3-2.

The memory-mapped registers may be accessed in the same manner as any other data memory location, with the exception that block moves using the BLKD (block move from data memory to data memory) instruction cannot be performed from the memory-mapped registers.

Table 3-2. Memory-Mapped Registers

REGISTER NAME	ADDRESS LOCATION	DEFINITION
DRR(15-0)	0	Serial port data receive register
DXR(15-0)	1	Serial port data transmit register
TIM(15-0)	2	Timer register
PRD(15-0)	3	Period register
IMR (5-0)	4	Interrupt mask register
GREG(7-0)	5	Global memory allocation register

3.4.5 Auxiliary Registers

The TMS320C2x provides a register file containing up to eight auxiliary registers (AR0-AR7). The TMS32020 has five auxiliary registers, and the TMS320C25 has eight. This section discusses each register's function and how an auxiliary register is selected and stored.

The auxiliary registers may be used for indirect addressing of data memory or for temporary data storage. Indirect auxiliary register addressing (see Figure 3-5) allows placement of the data memory address of an instruction operand into one of the auxiliary registers. These registers are pointed to by a three-bit auxiliary register pointer (ARP) that is loaded with a value from 0 through 7, designating AR0 through AR7, respectively. The auxiliary registers and the ARP may be loaded either from data memory or by an immediate operand defined in the instruction. The contents of these registers may also be stored in data memory. (Section 4 describes the programming of the indirect addressing mode.)

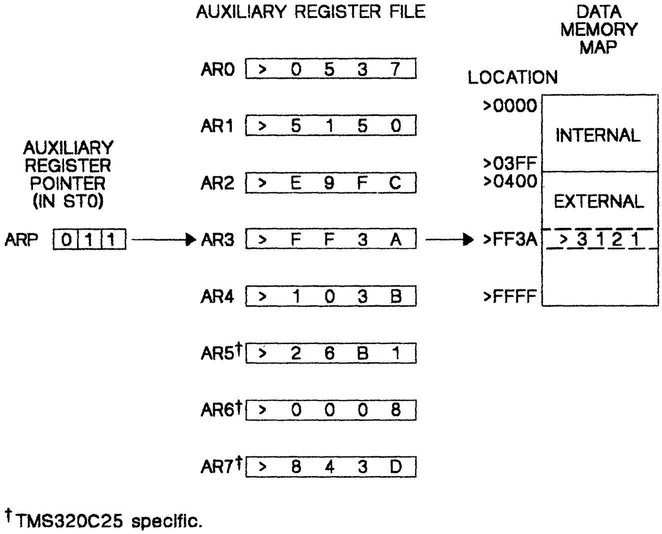
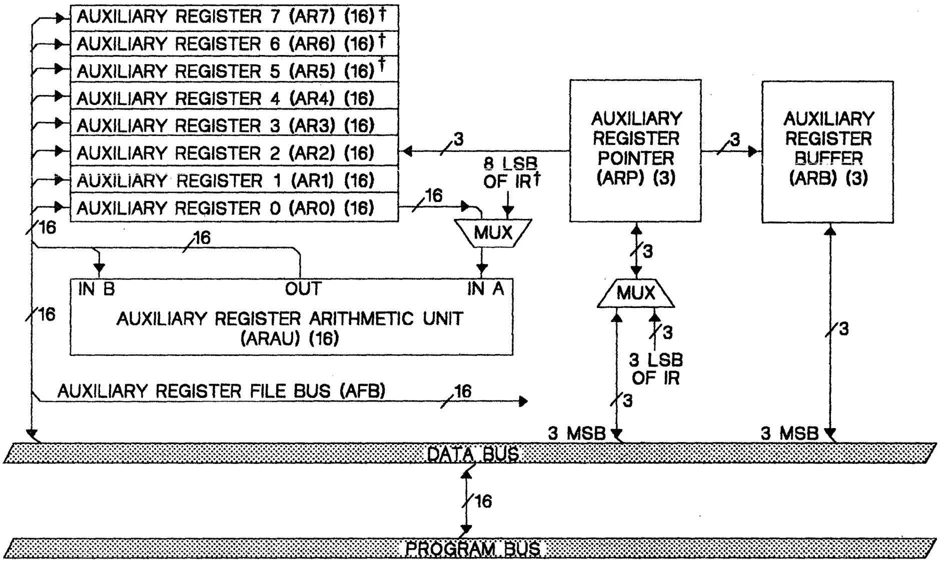


Figure 3-5. Indirect Auxiliary Register Addressing Example

The auxiliary register file (AR0-AR4 on the TMS32020 and AR0-AR7 on the TMS320C25) is connected to the Auxiliary Register Arithmetic Unit (ARAU), shown in Figure 3-6. The ARAU may autoindex the current auxiliary register while the data memory location is being addressed. Indexing by either ±1 or by the contents of AR0 may be performed. As a result, accessing tables of information does not require the Central Arithmetic Logic Unit (CALU) for address manipulation, thus freeing it for other operations.



† TMS320C25 specific.

Figure 3-6. Auxiliary Register File

As shown in Figure 3-6, auxiliary register 0 (AR0) or the eight LSBs of the instruction registers can be connected to one of the inputs of the ARAU. The other input is fed by the current AR (being pointed to by ARP). AR(ARP) refers to the contents of the current AR pointed to by ARP. The ARAU performs the following functions:

AR(ARP) + AR0 → AR(ARP)	Index the current AR by adding a 16-bit integer contained in AR0.
AR(ARP) - AR0 → AR(ARP)	Index the current AR by subtracting a 16-bit integer contained in AR0.
AR(ARP) + 1 → AR(ARP)	Increment the current AR by one.
AR(ARP) - 1 → AR(ARP)	Decrement the current AR by one.
AR(ARP) → AR(ARP)	AR(ARP) is unchanged.

In addition to the above functions, the ARAU on the TMS320C25 performs functions as follows:

AR(ARP) + IR(7-0) → AR(ARP)	Add 8-bit immediate value to the current AR.
AR(ARP) - IR(7-0) → AR(ARP)	Subtract 8-bit immediate value from the current AR.
AR(ARP) + rcAR0 → AR(ARP)	Bit-reversed indexing, add AR0 with reverse-carry (rc) propagation (see Section 4.1.2).
AR(ARP) - rcAR0 → AR(ARP)	Bit-reversed indexing, subtract AR0 with reverse-carry (rc) propagation (see Section 4.1.2).

Although the ARAU is useful for address manipulation in parallel with other operations, it may also serve as an additional general-purpose arithmetic unit since the auxiliary register file can directly communicate with data memory. The ARAU implements 16-bit unsigned arithmetic, whereas the CALU implements 32-bit two's-complement arithmetic. Instructions provide branches dependent on the comparison of the auxiliary register pointed to by ARP with AR0. The BANZ instruction permits the auxiliary registers to also be used as loop counters.

The three-bit auxiliary register pointer buffer (ARB), shown in Figure 3-6, provides storage for the ARP on subroutine calls and interrupts.

3.4.6 Memory Addressing Modes

The TMS320C2x can address a total of 64K words of program memory and 64K words of data memory. The on-chip data memory is mapped into the 64K-word data memory space. The on-chip ROM in the TMS320C25 is mapped into the program memory space when in the microcomputer mode. The memory maps, which change with the configuration of block B0, are described in detail in Section 3.4.4.

The 16-bit data address bus (DAB) addresses data memory in one of the following two ways:

1) By the direct address bus (DRB) using the direct addressing mode (e.g., ADD >10), or

2) By the auxiliary register file bus (AFB) using the indirect addressing mode (e.g., ADD *)

Operands are also addressed by the contents of the program counter in the immediate addressing mode.

Figure 3-7 illustrates operand addressing in the direct, indirect, and immediate addressing modes.

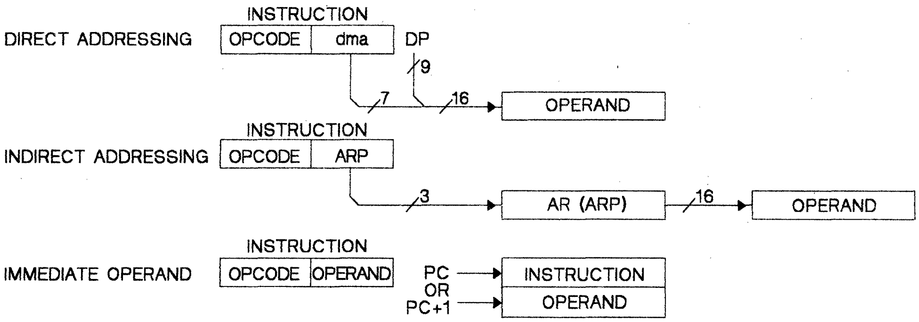


Figure 3-7. Methods of Instruction Operand Addressing

In the direct addressing mode, the 9-bit data memory page pointer (DP) points to one of 512 pages, each page consisting of 128 words. The data memory address (dma), specified by the seven LSBs of the instruction, points to the desired word within the page. The address on the direct address bus (DRB) is formed by concatenating the 9-bit DP with the 7-bit dma.

In the indirect addressing mode, the currently selected 16-bit auxiliary register AR(ARP) addresses the data memory through the auxiliary register file bus (AFB). While the selected auxiliary register provides the data memory address and the data is being manipulated by the CALU, the contents of the auxiliary register may be manipulated through the ARAU. See Figure 3-5 for an example of indirect auxiliary register addressing. The direct and indirect addressing modes are described in detail in Section 4.1.

When an immediate operand is used, it is either contained within the instruction word itself or, in the case of 16-bit immediate operands, the word following the instruction opcode.

3.4.7 Memory-to-Memory Moves

The TMS320C2x provides instructions for data and program block moves and for data move functions that efficiently utilize the configurable on-chip RAM.

The BLKD instruction moves a block within data memory, and the BLKP instruction moves a block from program memory to data memory. When used with the repeat instructions (RPT/RPTK), the BLKD/BLKP instructions efficiently perform block moves from on- or off-chip memory.

Implemented in on-chip RAM, the DMOV (data move) function on the TMS320C2x is equivalent to that of the TMS320C1x. DMOV allows a word to be copied from the currently addressed data memory location in on-chip RAM to the next higher location while the data from the addressed location is being operated upon in the same cycle (e.g., by the CALU). An ARAU operation may also be performed in the same cycle when using the indirect addressing mode. The DMOV function is useful for implementing algorithms that use the z^{-1} delay operation, such as convolutions and digital filtering where data is being passed through a time window. The data move function can be used anywhere within blocks B0, B1, or B2. It is continuous across the boundary of blocks B0 and B1 but cannot be used with off-chip data memory. The MACD (multiply and accumulate with data move) and the LTD (load T register, accumulate previous product, and move data) instructions use the data move function.

The TBLR/TBLW (table read/write) instructions allow words to be transferred between program and data spaces. TBLR is used to read words from on-chip ROM or off-chip program ROM/RAM into the data RAM. TBLW is used to write words from on-chip data RAM to off-chip program RAM.

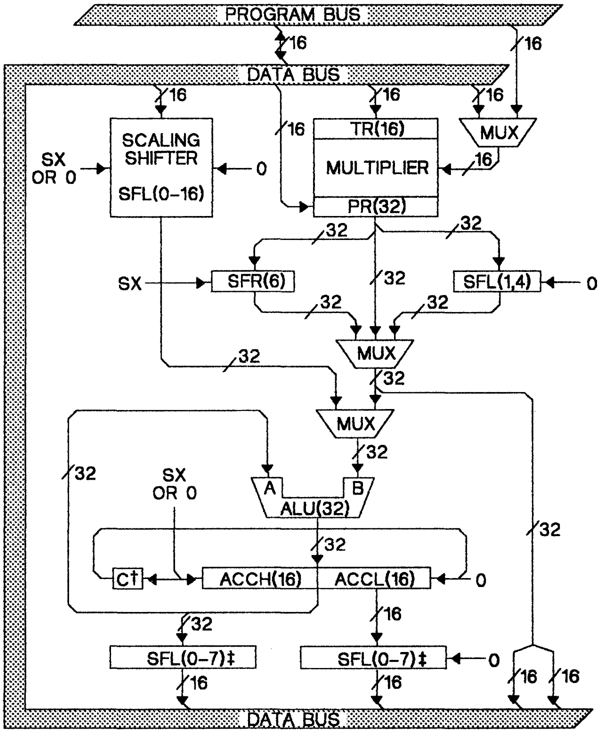
3.5 Central Arithmetic Logic Unit (CALU)

The TMS320C2x Central Arithmetic Logic Unit (CALU) contains a 16-bit scaling shifter, a 16 x 16-bit parallel multiplier, a 32-bit Arithmetic Logic Unit (ALU), a 32-bit accumulator (ACC), and additional shifters at the outputs of both the accumulator and the multiplier. This section describes the CALU components and their functions. Figure 3-8 is a block diagram showing the components of the CALU. In the figure, note that SFL and SFR indicate shifts to the left or right, respectively.

The following steps occur in the implementation of a typical ALU instruction:

1) Data is fetched from the RAM on the data bus,
2) Data is passed through the scaling shifter and the ALU where the arithmetic is performed, and
3) The result is moved into the accumulator.

One input to the ALU is always provided from the accumulator, and the other input may be transferred from the Product Register (PR) of the multiplier or from the scaling shifter that is loaded from data memory.



† TMS320C25 specific.
‡ Shifters on the TMS32020 of 0, 1, or 4.

Figure 3-8. Central Arithmetic Logic Unit (CALU)

3.5.1 Scaling Shifter

The TMS320C2x provides a scaling shifter that has a 16-bit input connected to the data bus and a 32-bit output connected to the ALU (see Figure 3-8). The scaling shifter produces a left shift of 0 to 16 bits on the input data, as programmed in the instruction. The LSBs of the output are filled with zeros, and the MSBs may be either filled with zeros or sign-extended, depending upon the status programmed into the SXM (sign-extension mode) bit of status register ST1.

The TMS320C2x also contains several other shifters, which allow it to perform numerical scaling, bit extraction, extended-precision arithmetic, and overflow prevention. These shifters are connected to the output of the multiplier and the accumulator.

3.5.2 ALU and Accumulator

The TMS320C2x 32-bit ALU and accumulator implement a wide range of arithmetic and logical functions, the majority of which execute in a single clock cycle. Once an operation is performed in the ALU, the result is transferred to the accumulator where additional operations such as shifting may occur. Data that is input to the ALU may be scaled by the scaling shifter.

The ALU is a general-purpose arithmetic unit that operates on 16-bit words taken from data RAM or derived from immediate instructions. In addition to the usual arithmetic instructions, the ALU can perform Boolean operations, providing the bit manipulation ability required of a high-speed controller. One input to the ALU is always provided from the accumulator, and the other input may be provided from the Product Register (PR) of the multiplier or the input scaling shifter that has fetched data from the RAM on the data bus. After the ALU has performed the arithmetic or logical operations, the result is stored in the accumulator.

The 32-bit accumulator (see Figure 3-8) is split into two 16-bit segments for storage in data memory: ACCH (accumulator high) and ACCL (accumulator low). Shifters at the output of the accumulator provide a left-shift of 0 to 7 places on the TMS320C25 and of 0, 1, or 4 places on the TMS32020. This shift is performed while the data is being transferred to the data bus for storage. The contents of the accumulator remain unchanged. When the ACCH data is shifted left, the LSBs are transferred from the ACCL, and the MSBs are lost. When ACCL is shifted left, the LSBs are zero-filled, and the MSBs are lost.

The TMS320C2x supports floating-point operations for applications requiring a large dynamic range. The NORM (normalization) instruction is used to normalize fixed-point numbers contained in the accumulator by performing left shifts. The LACT (load accumulator with shift specified by the T register) instruction denormalizes a floating-point number by arithmetically left-shifting the mantissa through the input scaling shifter. The shift count, in this case, is the value of the exponent specified by the four low-order bits of the T register (TR). ADDT and SUBT (add to/subtract from accumulator with shift specified by the T register) instructions have also been provided to allow additional arithmetic operations.

The accumulator overflow saturation mode may be programmed through the SOVM and ROVM (set/reset overflow mode) instructions. When the accumulator is in the overflow saturation mode and an overflow occurs, the overflow flag is set and the accumulator is loaded with either the most positive or the most negative number depending upon the direction of overflow. The value of the accumulator upon saturation is >7FFFFFFF (positive) or >80000000 (negative). If the OVM (overflow mode) status register bit is reset and an overflow occurs, the overflowed results are loaded into the accumulator without modification. (Note that logical operations cannot result in overflow.)

The TMS320C2x can execute a variety of branch instructions that depend on the status of the ALU and accumulator. These instructions include the BV (branch on overflow) and BZ (branch on accumulator equal to zero). In addition, the BACC (branch to address in accumulator) instruction provides the ability to branch to an address specified by the accumulator. Bit test instructions (BIT and BITT), which do not affect the accumulator, allow the testing of a specified bit of a word in data memory.

The accumulator on the TMS320C25 also has an associated carry bit that is set or reset depending on various operations within the device. The carry bit allows more efficient computation of extended-precision products and additions or subtractions. It is also useful in overflow management. The carry bit is affected by most arithmetic instructions as well as the shift and rotate instructions. It is not affected by loading the accumulator, logical operations, or other such nonarithmetic or control instructions. It is also not affected by the multiply (MPY, MPYK, and MPYU) instructions, but is affected by the accumulation process in the MAC and MACD instructions. Examples of carry bit operation are shown in Figure 3-9.

```
C  MSB              LSB                   C  MSB              LSB

X  F F F F  F F F F  ACC                  X  0 0 0 0  0 0 0 0  ACC
 +                 1                       -                 1
1  0 0 0 0  0 0 0 0                       0  F F F F  F F F F

X  7 F F F  F F F F  ACC                  X  8 0 0 0  0 0 0 0  ACC
 +                 1  (OVM=0)              -                 1  (OVM=0)
0  8 0 0 0  0 0 0 0                       1  7 F F F  F F F F

1  0 0 0 0  0 0 0 0  ACC                  0  F F F F  F F F F  ACC
 +                 0  (ADDC                -                 0  (SUBB
0  0 0 0 0  0 0 0 1  INSTRUCTION)         1  F F F F  F F F E  INSTRUCTION)
```

Figure 3-9. Examples of TMS320C25 Carry Bit Operation

The value added to or subtracted from the accumulator, shown in the examples of Figure 3-9, may come from either the input scaling shifter or the shifter at the output of the P register. The carry bit is set if the result of an addition or accumulation process generates a carry, or reset to zero if the result of a subtraction generates a borrow. Otherwise, it is reset after an addition or set after a subtraction.

The ADDC (add to accumulator with carry) and SUBB (subtract from accumulator with borrow) instructions provided on the TMS320C25 use the previous value of carry in their addition/subtraction operation (see these instructions in Section 4 for more detailed information).

The one exception to operation of the carry bit, as shown in Figure 3-9, is in the use of the ADDH (add to high accumulator) and SUBH (subtract from high accumulator) instructions. The ADDH instruction can only set the carry bit if a carry is generated, and the SUBH instruction can only reset the carry bit if a borrow is generated; otherwise, neither instruction can affect it.

Two branch instructions, BC and BNC, have been provided for branching on the status of the carry bit. The SC, RC, and LST1 instructions can also be used to load the carry bit. The carry bit is set to one on a hardware reset.

The SFL and SFR (in-place one-bit shift to the left/right) instructions on the TMS320C2x and the ROL and ROR (rotate to the left/right) instructions on the TMS320C25 implement shifting or rotating of the contents of the accumulator through the carry bit. The SXM bit affects the definition of the SFR (shift accumulator right) instruction. When SXM = 1, SFR performs an arithmetic right shift, maintaining the sign of the accumulator data. When SXM = 0, SFR performs a logical shift, shifting out the LSB and shifting in a zero for the MSB. The SFL (shift accumulator left) instruction is not affected by the SXM bit and behaves the same in both cases, shifting out the MSB and shifting in a zero. Repeat (RPT or RPTK) instructions may be used with the shift and rotate instructions for multiple shift counts.

3.5.3 Multiplier, T and P Registers

The TMS320C2x utilizes a 16 x 16-bit hardware multiplier, which is capable of computing a signed or unsigned 32-bit product in a single machine cycle. All multiply instructions, except the MPYU (multiply unsigned) instruction on the TMS320C25, perform a signed multiply operation in the multiplier. That is, the two numbers being multiplied are treated as two's-complement numbers, and the result is a 32-bit two's-complement number. As shown in Figure 3-8, the following two registers are associated with the multiplier:

- A 16-bit temporary register (TR) that holds one of the operands for the multiplier, and
- A 32-bit product register (PR) that holds the product.

The output of the product register can be left-shifted 1 or 4 bits. This is useful for implementing fractional arithmetic or justifying fractional products. The output of the PR can also be right-shifted 6 bits to enable the execution of up to 128 consecutive multiply/accumulates without the possibility of overflow.

An LT (load T register) instruction normally loads the TR to provide one operand (from the data bus), and the MPY (multiply) instruction provides the second operand (also from the data bus). A multiplication can also be performed with an immediate operand using the MPYK instruction. In either case, a product can be obtained every two cycles.

Two multiply/accumulate instructions (MAC and MACD) fully utilize the computational bandwidth of the multiplier, allowing both operands to be processed simultaneously. The data for these operations may reside anywhere in internal or external memory, or can be transferred to the multiplier each cycle via the program and data buses. This provides for single-cycle multiply/accumulates when used with repeat (RPT/RPTK) instructions. Note that the DMOV portion of the MACD instruction will not function with external data memory addresses. On the TMS32020, the multiplier and multiplicand must reside in separate on-chip RAM blocks. On the TMS320C25, the MAC and MACD instructions can be used with both operands in either internal or external memory or one each in on-chip RAM. The SQRA (square/add) and SQRS (square/subtract) instructions pass the same value to both inputs of the multiplier for squaring a data memory value.

The MPYU instruction on the TMS320C25 performs an unsigned multiplication, which greatly facilitates extended-precision arithmetic operations. The unsigned contents of the T register are multiplied by the unsigned contents of the addressed data memory location, with the result placed in the P register. This allows operands of greater than 16 bits to be broken down into 16-bit words and processed separately to generate products of greater than 32 bits.

After the multiplication of two 16-bit numbers, the 32-bit product is loaded into the 32-bit Product Register (PR) on the TMS320C2x. The product from the PR may be transferred to the ALU.

Four product shift modes (PM) are available at the Product Register (PR) output, which are useful when performing multiply/accumulate operations, fractional arithmetic, or justifying fractional products. The PM field of status register ST1 specifies the PM shift mode, as shown in Table 3-3.

Table 3-3. PM Shift Modes

IF PM IS:	RESULT
00	No shift
01	Left shift of 1 bit
10	Left shift of 4 bits
11	Right shift of 6 bits

Left shifts specified by the PM value are useful for implementing fractional arithmetic or justifying fractional products. For example, the product of either two normalized, 16-bit, two's-complement numbers or two Q15 numbers contains two sign bits, one of which is redundant. Q15 format, one of the various types of Q format, is a number representation commonly used when performing operations on non-integer numbers (see Section 5.6.6 for an explanation and examples of Q15 representation). The single-bit left-shift eliminates this extra sign bit from the product when it is transferred to the accumulator. This results in the accumulator contents being formatted in the same manner as the multiplicands. Similarly, the product of either a normalized, 16-bit, two's-complement or Q15 number and a 13-bit, two's-complement constant contains five sign bits, four of which are redundant. This is the case, for example, when using the MPYK instruction. Here the four-bit shift properly aligns the result as it is transferred to the accumulator.

Use of the right-shift PM value allows the execution of up to 128 consecutive multiply/accumulate operations without the threat of an arithmetic overflow, thereby avoiding the overhead of overflow management. The shifter can be disabled to cause no shift in the product when working with integer or 32-bit precision operations. This allows compatibility with TMS320C1x code to be maintained. Note that the PM right shift is always sign-extended regardless of the state of SXM.

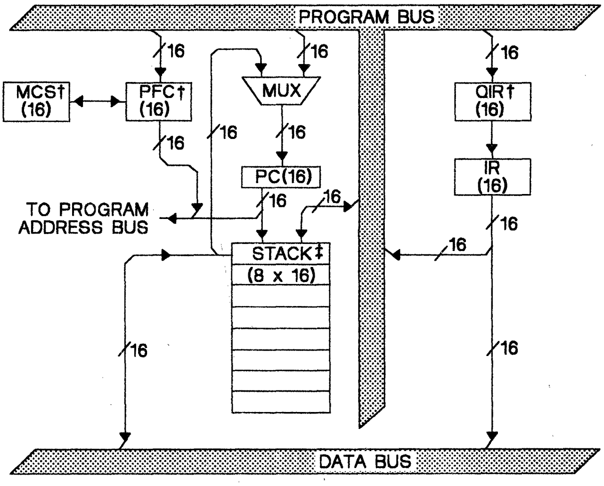
The four least significant bits of the T register (TR) also define a variable shift through the scaling shifter for the LACT/ADDT/SUBT (load/add-to/subtract-from accumulator with shift specified by the TR) instructions. These instructions are useful in floating-point arithmetic where a number needs to be denormalized, i.e., floating-point to fixed-point conversion. The BITT (bit test) instruction allows testing of a single bit of a word in data memory based on the value contained in the four LSBs of the TR.

3.6 System Control

System control on the TMS320C2x is provided by the program counter, hardware stack, PC-related hardware, the external reset signal, interrupts (see Section 3.8), the status registers, the on-chip timer, and the repeat counter. The following sections describe the function of each of these components in system control and pipeline operation.

3.6.1 Program Counter and Stack

The TMS320C2x contains a 16-bit Program Counter (PC) and a hardware stack of four (TMS32020) or eight (TMS320C25) locations for PC storage (see Figure 3-10). The program counter addresses internal and external program memory in fetching instructions. The stack is used during interrupts and subroutines.



† TMS320C25 specific.
‡ Four-level stack provided on the TMS32020.

Figure 3-10. Program Counter, Stack, and Related Hardware

The program counter addresses program memory, either on-chip or off-chip, via the Program Address Bus (PAB). Through the PAB, an instruction is fetched from program memory and loaded into the Instruction Register (IR). When the IR is loaded, the PC is ready to start the next instruction fetch cycle. The PC may address on-chip RAM block B0 when B0 is configured as program memory, or the on-chip ROM provided on the TMS320C25. The PC also addresses off-chip program memory through the external address bus A15-A0 and the external data bus D15-D0.

Data memory is addressed by the program counter during a BLKD instruction, which moves data blocks from one section of data memory to another. The contents of the accumulator may be loaded into the PC in order to implement "computed GOTO" operations. This can be accomplished using the BACC (branch to address in accumulator) or CALA (call subroutine indirect) instructions.

To start a new fetch cycle, the PC is loaded either with PC+1 or with a branch address (for instructions such as branches, calls, or interrupts). In the case of conditional branches where the branch is not taken, the PC is incremented once more beyond the location of the branch address.

The TMS320C2x also has a feature, which allows the execution of the next single instruction N+1 times. N is defined by loading an 8-bit counter RPTC (repeat counter). If this repeat feature is used, the instruction is executed, and the RPTC is decremented until the RPTC goes to zero. This feature is useful with many instructions, such as NORM (normalize contents of accumulator), MACD (multiply and accumulate with data move), and SUBC (conditional subtract). When used with some multicycle instructions, such as MACD, the repeat features can result in these instructions effectively executing in a single cycle.

The stack is 16 bits wide and four (TMS32020) or eight (TMS320C25) levels deep. The PC stack is accessible through the use of the PUSH and POP instructions. Whenever the contents of the PC are pushed onto the top of the stack, the previous contents of each level are pushed down, and the bottom (fourth/eighth) location of the stack is lost. Therefore, data will be lost if more than four/eight successive pushes occur before a pop. The reverse happens on pop operations. Any pop after three/seven sequential pops yields the value at the bottom stack level. All of the stack levels then contain the same value. Two additional instructions, PSHD and POPD, push a data memory value onto the stack or pop a value from the stack to data memory. These instructions allow a stack to be built in data memory for the nesting of subroutines/interrupts beyond four/eight levels.

Note that on the TMS32020, the TBLR/TBLW, MAC/MACD, and BLKD/BLKP instructions use one level of the stack. The TMS320C25 contains a separate stack for use with these instructions, and no level of the PC is used.

3.6.2 Pipeline Operation

Instruction pipelining consists of the sequence of external bus operations that occurs during instruction execution. The prefetch-decode-execute pipeline is essentially invisible to the user, except in some cases where the pipeline must be broken (such as for branch instructions). In the operation of the pipeline, the prefetch, decode, and execute operations are independent, which allows instruction executions to overlap. Thus, during any given cycle, two or three different instructions can be active, each at a different stage of completion, resulting in the respective two-level pipeline on the TMS32020 or the three-level pipeline on the TMS320C25.

The difference in pipeline levels does not necessarily affect instruction execution speed, but merely changes the fetch/decode sequence. Most instructions execute in the same number of cycles regardless of whether they are executed from internal RAM, ROM, or external program memory. The effects of pipelining are included in the instruction cycle timings for the TMS32020 and TMS320C25 listed in Appendix D.

Additional PC-related hardware (see Figure 3-10) is provided on the TMS320C25 to allow three-level pipelining for higher performance. Included in the related hardware are the Prefetch Counter (PFC), the 16-bit MicroCall Stack (MCS) register, the Instruction Register (IR), and the Queue Instruction Register (QIR).

In the three-level pipeline on the TMS320C25, the PFC contains the address of the next instruction to be prefetched. Once an instruction is prefetched, the instruction is loaded into the IR, unless the IR still contains an instruction currently executing, in which case the prefetched instruction is stored in the QIR. The PFC is then incremented, and after the current instruction has completed execution, the instruction in the QIR is loaded into the IR to be executed.

The PC contains the address of the next instruction to be executed, and is not used directly in instruction fetch operations, but merely serves as a reference pointer to the current position within the program. The PC is incremented as each instruction is executed. When interrupts or subroutine call instructions occur, the contents of the PC are pushed onto the stack to preserve return linkage to the previous program context.

The prefetch, decode, and execute operations of the pipeline are independent, thus allowing instruction executions to overlap. During any given cycle, three different instructions can be active, each at a different stage of completion. Figure 3-11 shows the operation of the three-level pipeline for single-word, single-cycle instructions executing from either internal program ROM or external memory with no wait states.

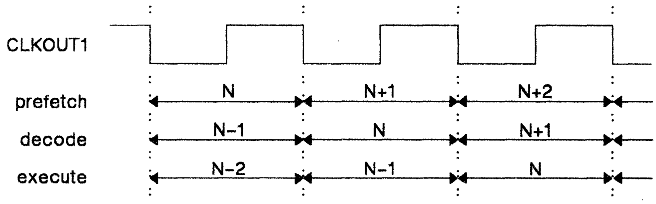


Figure 3-11. Three-Level Pipeline Operation (TMS320C25)

Pipelining is reduced to two levels when execution is from internal program RAM due to the fact that an instruction in internal RAM can be fetched and decoded in the same cycle. Thus, separate prefetch and decode operations are not required, as shown in Figure 3-12.

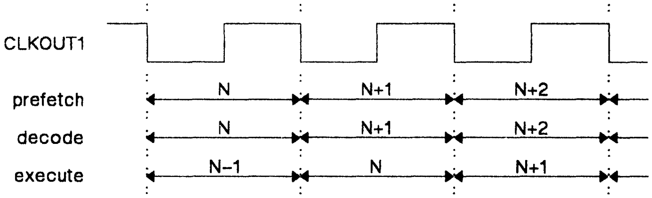


Figure 3-12. Two-Level Pipeline Operation

The following paragraphs describe, in detail, the operation of the TMS320C25 pipeline. This description, in conjunction with Appendix D, gives sufficient information for predicting the operation of the TMS320C25 for hardware interface optimization, accurate program cycle counting, and simulation modelling. Often it is not necessary to understand the intricate detail of the pipeline to design with the TMS320C25. Therefore, if the user is not specifically interested in these details, it is suggested that this description be skipped.

The TMS320C25 executes most of its instructions in a single cycle, because all the instructions are straight decodes and highly pipelined as opposed to microcode. The basic pipeline operation is 3.25 cycles deep where the device sequence on any given cycle is fetching the third instruction, decoding the second instruction, and executing the first. Figure 3-13 shows the internal operation of the TMS320C25 pipeline in reference to quarter phases 1 through 4 (Q1-Q4).

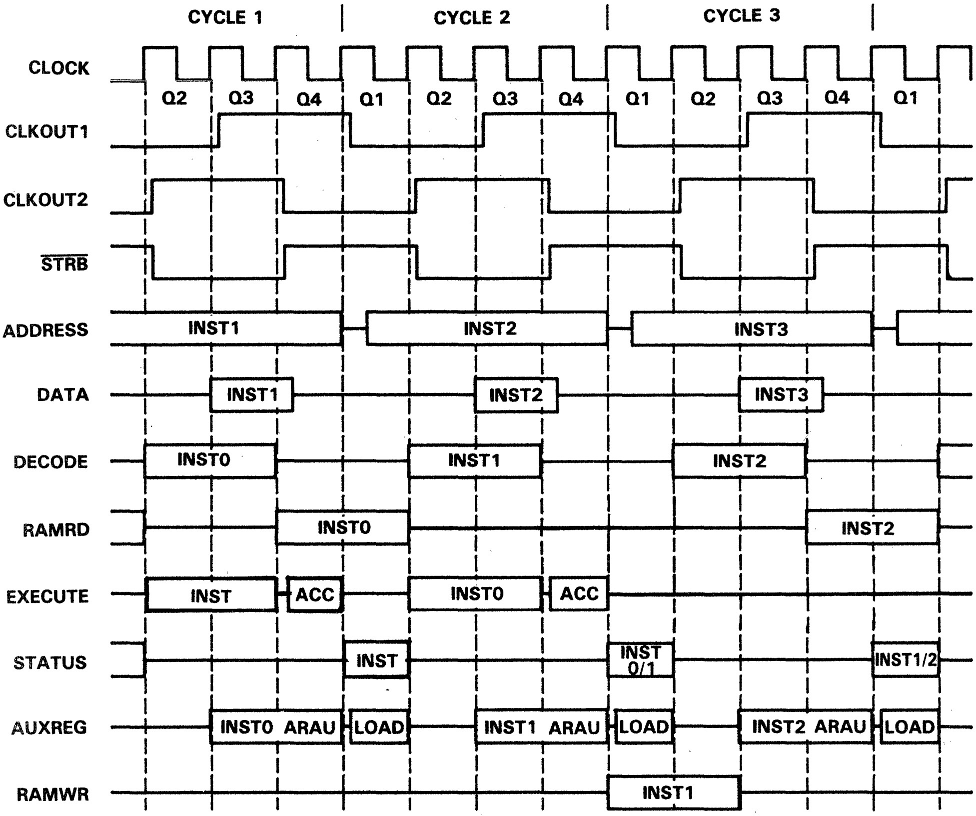


Figure 3-13. TMS320C25 Standard Pipeline Operation

The TMS320C25 machine cycle, externally referenced by the falling edges of the CLKOUT1 signal, consists of four internal cycles (or CLKIN cycles). This allows internal operations of the pipeline to execute as fast as 1/4 the machine cycle. The sequence of a general instruction execution in the pipeline is shown in Table 3-4.

Table 3-4. Instruction Pipeline Sequence

CYCLE	Q PHASE	OPERATION
1	1	New PC is output on address bus
	2	External read of instruction
	3	External read of instruction
	4	External read of instruction
2	1	
	2	Instruction decode
	3	Instruction decode/ARAU execution
	4	On-chip RAM access/ARAU execution
3	1	On-chip RAM access/load new AR value/update ARP
	2	ALU execution
	3	ALU execution
	4	Load accumulator
4	1	Load status register

When using an add instruction (e.g., ADD *+,12,AR4), the device fetches the instruction in cycle 1. During Q2 and Q3 of cycle 2, the instruction is decoded. This includes the ALU command decode as well as generation of the data operand fetch address. In this case, the address comes from an auxiliary register. During Q4 of cycle 2 and Q1 of cycle 3, the operand is fetched from the RAM location. The increment of the auxiliary register is performed during Q3 and Q4 of cycle 2, and the value is loaded into the auxiliary register in Q1 of cycle 3. The ARP is also updated in Q1 of cycle 3. During Q2 and Q3 of cycle 3, the data is passed through the barrel shifter to execute the 12-bit left-shift, and the data is added by the ALU to the contents in the accumulator. In Q4 of the third cycle, the ALU result is loaded into the accumulator. The status of the ALU operation is loaded into the status register in Q1 of the fourth cycle. The bits being loaded into the status register at this time consist of the current ALU status and the ARP associated with the next instruction.

In the case of a store instruction (e.g., SACL *0-,3,AR2), the device operates the first two cycles in the same manner as the ADD instruction. In Q1 and Q2 of the third cycle, the data in the accumulator is passed through a barrel shifter, left-shifted 3 bits, and zero-filled. The lower 16 bits of the shifted value are written to the address specified by the current auxiliary register. During Q3 and Q4 of the third cycle, the index register (AR0) is added to the contents of the current auxiliary register and loaded back into the current auxiliary register in Q1 of the fourth phase. In Q1 of the fourth cycle, the auxiliary register pointer is changed to AR2. There is no execution phase of this instruction. Figure 3-14 shows the ADD and SACL instructions operating back-to-back in a program sequence. It is assumed that both instructions reside in external, zero wait-state memory and that the data resides in on-chip RAM.

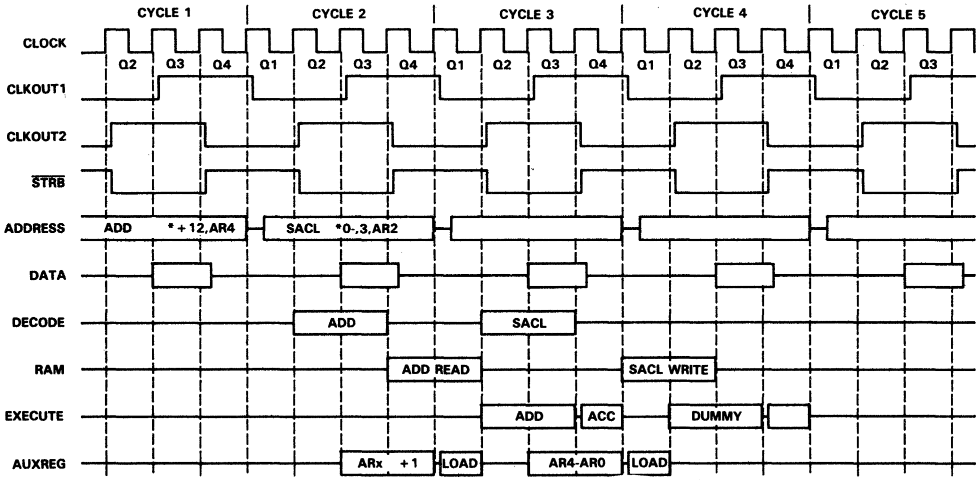


Figure 3-14. Pipeline Operation of ADD Followed by SACL

When the device is reading instructions out of on-chip ROM, the basic internal operation of the pipeline is the same. The only difference is that the control lines (i.e., $\overline{\mathrm{STRB}}$, $\overline{\mathrm{PS}}$, and R/$\overline{\mathrm{W}}$) are inactive. If the device is fetching the instructions from on-chip RAM, the pipeline is shortened to 2.5 cycles since the device can fetch the instruction in half a cycle as opposed to the full cycle required in an external or on-chip ROM fetch. The instruction is fetched during Q4 and Q1, then decoded in Q2 and Q3. The rest of the pipeline tracks as described above.

Some operations add additional machine cycles to the instruction execution without damaging the integrity of the program or hardware. External wait states, multiplexed data bus conflicts, two-word instructions, and program counter discontinuities are included in these operations, as described in the following paragraphs.

Wait States. The TMS320C25 is designed to be interfaced to slower external devices through the use of hardware-generated wait states. This applies to the program, data, and I/O memory spaces of the Harvard architecture. Wait states are a direct delay on the instruction pipeline. Each wait state inserted during the instruction fetch contributes an additional machine cycle in the pipeline execution of the instruction. In addition, any wait state incurred when accessing external data or I/O space also contributes an additional machine cycle to the pipeline execution of the instruction. This factor applies to all instructions. Figure 3-15 describes how the pipeline reacts to wait states in external program memory. Note that the wait state added in cycle 2 results in a no-execution operation in cycle 4.

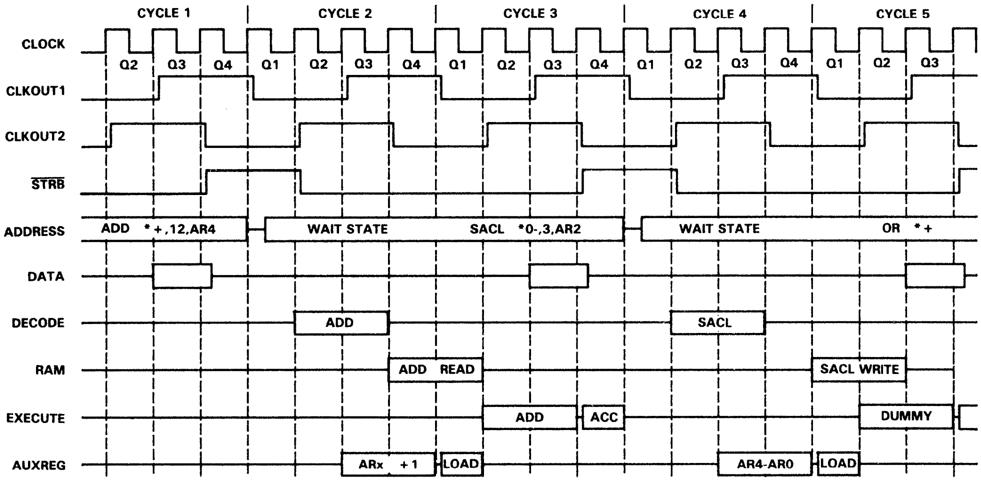


Figure 3-15. Pipeline Operation with Wait States

Multiplexed External Data Bus. The external data bus is multiplexed to support all three memory spaces of the TMS320C25. Therefore, external fetches to multiple spaces in the same instruction add additional machine cycles to the pipeline execution of the instruction. This is due to the fact that the external fetch takes a full cycle whereas the internal equivalent takes two quarter phases and can be included in the execution stage of the three-deep pipeline. Accessing the data memory space is controlled by setting of the data page pointer or the value contained in the auxiliary register used in any instruction. Also affecting the pipeline in this manner is the access of the I/O bus or the tables in program memory (i.e., IN, OUT, TBLR, and TBLW). Figure 3-16 shows how the pipeline processes an instruction with external program and data access.

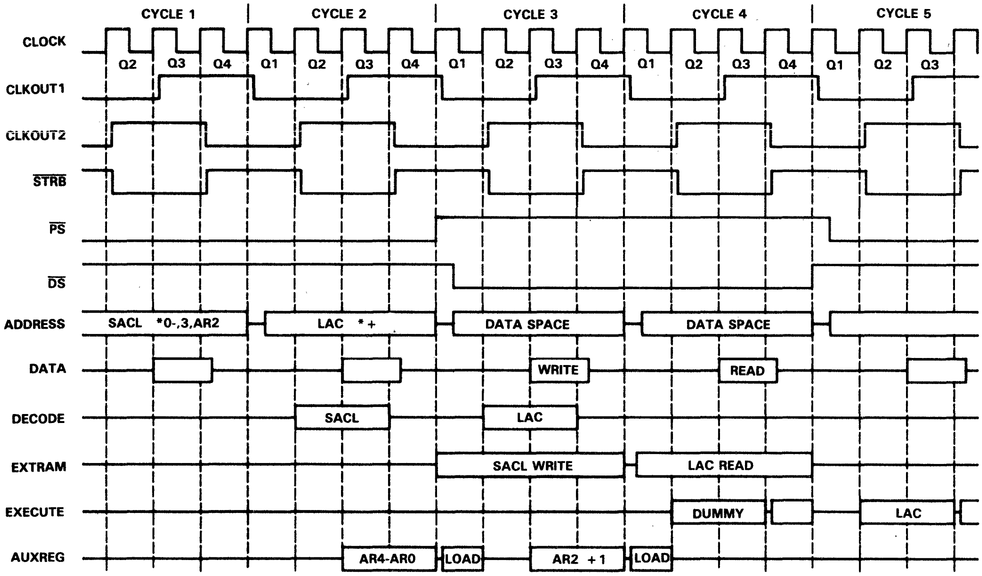


Figure 3-16. Pipeline with External Data Bus Conflict

Two-Word Instructions. All two-word instructions take an additional cycle to fetch the 16-bit immediate operand following the instruction mnemonic. The first set of instructions for which this applies is the long immediate instructions. The instruction mnemonic is followed by a 16-bit immediate operand to be executed upon in the ALU. The second set applies to those instructions that use the PFC register as a second data addressing unit on some optimized instructions, e.g., the multiply/accumulate and block move instructions (MAC, MACD, BLKP, and BLKD). In the second set, the extra cycle only appears once in a repeat loop. The third set involves conditional branches not taken.

Program Counter Discontinuities. Since the TMS320C25 is pipelined, a change (other than an increment) in the program counter requires that the pipeline be flushed. This applies to all branches, subroutine calls, software trap, interrupt traps, and return. The pipeline, being three deep, has the next instruction already loaded when the branch occurs. At this point, this instruction will not affect any data or registers, so it is cleared from the pipeline. Therefore, two dead execution cycles are inserted while waiting for the pipeline to reload. The device only takes one additional cycle if the destination of the branch is in on-chip RAM block 0. The pipeline is only two-deep in this case and only takes one cycle to reload. Figure 3-17 shows a branch from normal execution to an address in on-chip RAM, and Figure 3-18 shows an example of a return executed from on-chip RAM to a location in off-chip memory.

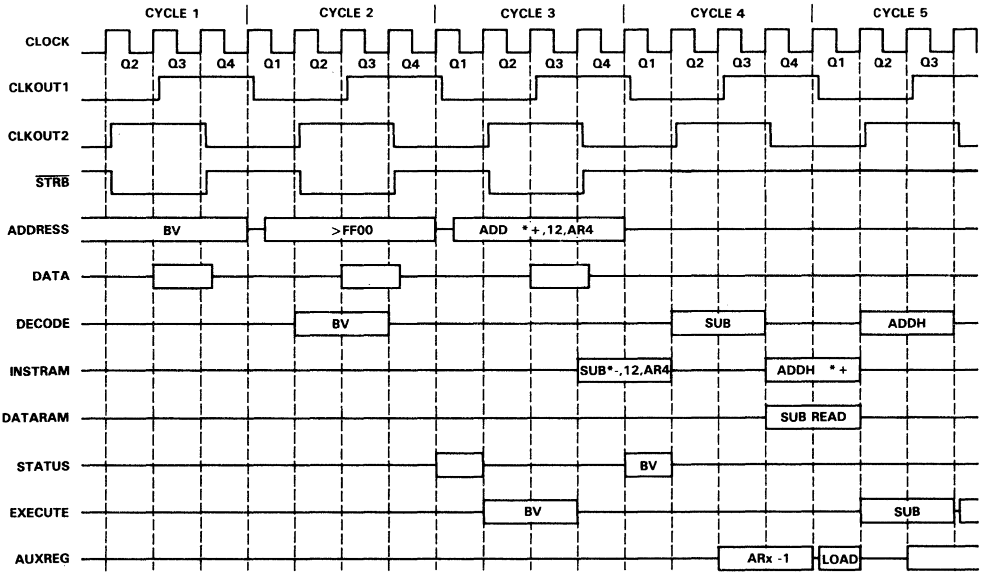


Figure 3 17. Pipeline Operation of Branch to On-Chip RAM

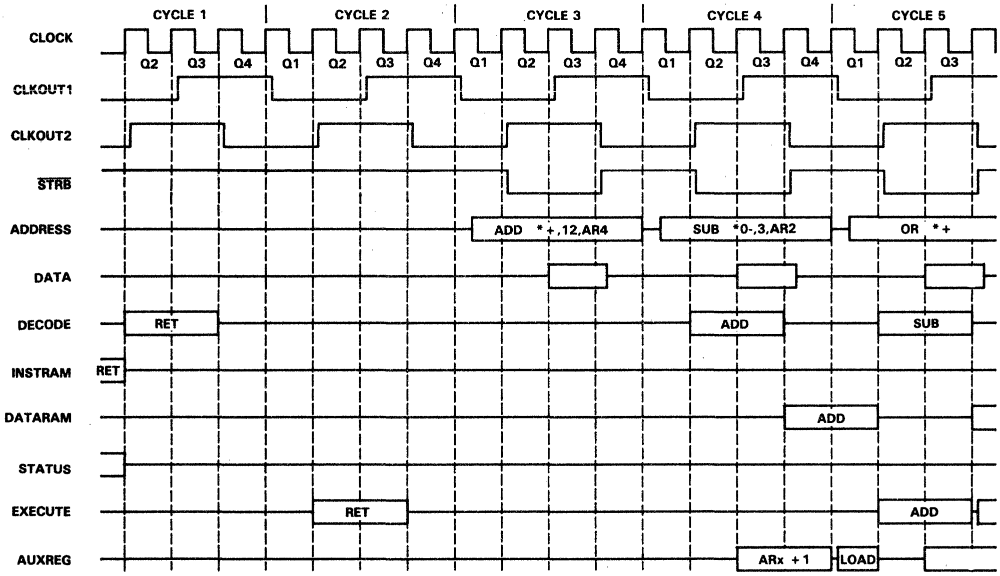


Figure 3-18. Pipeline Operation of RET from On-Chip RAM

Interrupts are hardware-generated discontinuities to the sequential accessing of the program counter. The interrupt is executed based upon instruction execution complete, rather than memory operation complete. The instruction that is currently executing at the time of an interrupt executes completely. The interrupt traps following the completion of that instruction before the start of the execution of the next instruction. In this case, the repeated instruction is considered one execution; therefore, the repeat loop finishes before the interrupt trap is taken. This gives priority to the algorithm over the interrupt service. The interrupt operation in reference to the pipeline execution is illustrated in the data sheet timing diagrams (see Appendix A). Note that when interrupt vectors reside in external memory running with one wait state, there are two interrupt acknowledge ($\overline{\text{IACK}}$) pulses. If this is a problem, the $\overline{\text{IACK}}$ line should be gated with READY.

Hardware Aspects of the Pipeline

Viewing these effects on the pipeline at the hardware level requires additional explanation due to the lack of visibility of on-chip operations or optimization of the pipeline execution. The following paragraphs describe the effects of $\overline{\mathrm{HOLD}}/\overline{\mathrm{HOLDA}}$, $\overline{\mathrm{RS}}$, interrupts, accumulator store, on-chip program access, external data access, and repeats as they are visible from the pins of the device. In the cases of $\overline{\mathrm{RS}}$, interrupts, and $\overline{\mathrm{HOLD}}/\overline{\mathrm{HOLDA}}$, the effects on the pipeline are shown in the data sheet timing diagrams (see Appendix A).

Reset. The reset interrupt is a totally non-maskable interrupt. When executed, it stops operation of the pipeline and flushes the unexecuted parts. The reset pulse must be at least three CLKOUT cycles wide. After the second CLKOUT cycle has completed (before the third rising edge of CLKOUT1), the device has brought all outputs into a high-impedance state. After the rising edge of $\overline{\mathrm{RS}}$, the device begins to fetch the reset vector. Since the pipeline is empty, it does not execute the reset vector branch until two cycles later. If the $\overline{\mathrm{HOLD}}$ line is brought low during the active reset, the device does not start the fetch of the reset vector until after the active $\overline{\mathrm{HOLD}}$ is removed, and the device deactivates the $\overline{\mathrm{HOLDA}}$ line. When $\overline{\mathrm{HOLD}}$ is activated with $\overline{\mathrm{RS}}$ to allow bootloading of the code, the $\overline{\mathrm{HOLDA}}$ line will go active low in three cycles, regardless of whether or not the $\overline{\mathrm{RS}}$ line has gone high. This is useful in that the $\overline{\mathrm{HOLDA}}$ line can be used to enable the release of the $\overline{\mathrm{RS}}$ line and guarantee the required three-cycle reset.

Interrupts. The effects of an interrupt become apparent on the hardware when a interrupt acknowledge ($\overline{\mathrm{IACK}}$) signal is valid on the rising edge of CLKOUT2. This signifies the fetch of the first word of the interrupt vector. If wait states are generated in the memory segment where the interrupt vector resides, an additional $\overline{\mathrm{IACK}}$ pulse occurs for each wait state added. If this causes a problem with the external interface, $\overline{\mathrm{IACK}}$ can be gated with READY to only accept the last interrupt acknowledge pulse. Note that the BIOZ instruction tests the level of the $\overline{\mathrm{BIO}}$ pin during the instruction fetch phase of the pipeline.

Hold/Hold Acknowledge. The hold operation, like that of interrupt, takes second priority to algorithm execution; therefore, the hold will not be acknowledged until after the currently running instruction is completed (a minimum of three cycles). This includes repeated instructions. The next instruction, after the final instruction executed before $\overline{\mathrm{HOLDA}}$, is latched into the pipeline and executed two cycles after the $\overline{\mathrm{HOLDA}}$ line goes inactive high. The second instruction after the last instruction executed is fetched two cycles again after the $\overline{\mathrm{HOLDA}}$ line goes inactive high. If the HM bit of status register ST1 is set high, the TMS320C25 stops execution and sits idle until the hold is removed. This lowers power consumption by removing the drive of the memory address and control lines and also stopping major parts of the internal CPU circuits from switching and drawing power. This can be used as a hardware powerdown mode. If the HM bit is low, the TMS320C25 continues executing any instruction that can be executed with on-chip resources only. This means both program and data reside in on-chip memory. The device will continue to operate normally unless an off-chip access is required by an instruction, at which time the processor adds wait states until the hold state is removed. When running from on-chip resources with HM = 0, the processor acknowledges $\overline{\mathrm{HOLD}}$ with $\overline{\mathrm{HOLDA}}$ during a multicycle instruction.

On-Chip Program Access. When executing from on-chip resources, the pipeline is visible only in the $\overline{\text{MSC}}$ line, which signals microstate complete when active low on the rising edge of CLKOUT2. Note that executing from on-chip program memory does not allow instruction accessing of external data memory to run in a single cycle. The normal operation of the instruction takes only two quarter phases of the execution cycle to fetch the on-chip data memory, whereas off-chip access requires all four quarter phases. The pipeline is, however, optimized to handle a repeated instruction that accesses external data memory with only one extra cycle for the first external fetch.

External Program/Data Access. Visibility of the pipeline when using external program and data memory requires a monitoring of the $\overline{\text{MSC}}$, $\overline{\text{STRB}}$, $\overline{\text{PS}}$, and $\overline{\text{DS}}$ lines. The $\overline{\text{MSC}}$ line indicates at the rising edge of CLKOUT2 whether or not the cycle is the beginning of a new instruction fetch; i.e., $\overline{\text{MSC}}$ active low indicates the completion of an instruction and the acquisition of another instruction. The $\overline{\text{PS}}$ (program select) line indicates that the data bus is currently being used to fetch an instruction. A step in the pipeline is not indicated since the $\overline{\text{PS}}$ line remains while the pipeline is fetching instructions externally. To track the fetches, the $\overline{\text{STRB}}$ line, which frames external accesses, must be monitored.

The $\overline{\text{PS}}$ line being active low does not necessarily mean that the device is fetching an instruction. In the cases of table read/write (TBLR/TBLW), multiply/accumulate (MAC/MACD), and block transfer (BLKP) instructions, the device uses the $\overline{\text{PS}}$ line active low to access tables.

To monitor external data memory fetches, the user should watch the data select ($\overline{\text{DS}}$) line in conjunction with the $\overline{\text{STRB}}$ line. An active low on the $\overline{\text{DS}}$ line indicates the data bus is currently being used to access data memory space. This line remains low for two memory fetches in the case of an accumulator store followed by an ALU instruction, both operating with off-chip memory. However, two $\overline{\text{STRB}}$ pulses will identify the individual access. Likewise, the line remains low for many cycles in the case of a repeated instruction. I/O space access operates similarily to data space operation with the OUT and IN instructions replacing the save and ALU instruction.

A clear understanding of this information in conjunction with the data in Appendix D of the TMS320C2x User's Guide should be sufficient to predict correctly the operation of the TMS320C25 pipeline.

3.6.3 Reset

Reset ($\overline{\text{RS}}$) is a non-maskable external interrupt that can be used at any time to put the TMS320C2x into a known state. Reset is typically applied after powerup when the machine is in a random state.

Driving the $\overline{\text{RS}}$ signal low causes the TMS320C2x to terminate execution and forces the program counter to zero. $\overline{\text{RS}}$ affects various registers and status bits. At powerup, the state of the processor is undefined. For correct system operation after powerup, a reset signal must be asserted low for at least three clock cycles to guarantee a reset of the device (see Section 5.1 for other important reset considerations). Processor execution begins at location 0, which normally contains a B (branch) statement to direct program execution to the system initialization routine (also see Section 5.1 for an initialization routine example). Section 6.1 provides system control circuitry design examples.

Upon receiving an $\overline{\text{RS}}$ signal, the following actions take place:

1) A logic 0 is loaded into the CNF (configuration control) bit in status register ST1, causing all RAM to be configured as data memory.

2) The Program Counter (PC) is set to 0, and the address bus A15-A0 is driven with all zeroes while $\overline{\text{RS}}$ is low.

3) The data bus D15-D0 is placed in the high-impedance state.

4) All memory and I/O space control signals ($\overline{\text{PS}}$, $\overline{\text{DS}}$, $\overline{\text{IS}}$, R/$\overline{\text{W}}$, $\overline{\text{STRB}}$, and $\overline{\text{BR}}$) are de-asserted by setting them to high levels while $\overline{\text{RS}}$ is low.

5) All interrupts are disabled by setting the INTM (interrupt mode) bit to 1. (Note that $\overline{\text{RS}}$ is non-maskable.) The interrupt flag register (IFR) is reset to all zeroes.

6) Status bits:
   0 → OV and 1 → XF (TMS32020); in addition, on the TMS320C25,
   1 → SXM, 0 → PM, 1 → HM, 0 → FO, 1 → C, and 1 → FSM.
   (The remaining status bits on the TMS320C2x are unchanged.)

7) The global memory allocation register (GREG) is cleared to make all memory local.

8) The RPTC (repeat counter) is cleared.

9) The DX (data transmit) pin is placed in the high-impedance state. Any transmit/receive operations on the serial port are terminated, and the TXM (transmit mode) bit is reset to a low level. This configures the FSX framing pulse to be an input. A transmit/receive operation may be started by framing pulses only after the removal of $\overline{\text{RS}}$.

10) The TIM register is set to the maximum value (>FFFF) on reset for both the TMS32020 and TMS320C25. The PRD register on the TMS320C25 is also initialized by reset to >FFFF. The TMS32020 requires a software initialization of the PRD register (see Example 5-1). The TIM register begins decrementing only after $\overline{\text{RS}}$ is de-asserted.

11) The $\overline{\text{IACK}}$ (interrupt acknowledge) signal is generated in the same manner as a maskable interrupt.

12) The state of the RAM is undefined following $\overline{RS}$.

13) The ARB, ARP, DP, IMR, OVM, and TC bits are not initialized by reset. Therefore, it is critical that these bits be initialized in software by the user following reset.

Execution starts from location 0 of program memory when the $\overline{RS}$ signal is taken high. Note that if $\overline{RS}$ is asserted while in the hold mode, normal reset operation occurs internally, but all buses and control lines remain in the high-impedance state. Upon release of $\overline{HOLD}$ and $\overline{RS}$, execution starts from location zero. The TMS320C2x can be held in the reset state indefinitely.

3.6.4 Status Registers

Two status registers, ST0 and ST1, contain the status of various conditions and modes. The status registers can be stored into data memory and loaded from data memory, thus allowing the status of the machine to be saved and restored for interrupts and subroutines. All status bits are written to and read from using LST/LST1 and SST/SST1 instructions, respectively (with the exception of INTM, which cannot be loaded via an LST instruction).

Figure 3-19 shows the organization of both status registers, indicating all status bits contained in each. Note that the DP, ARP, and ARB registers are shown as separate registers in the processor block diagram of Figure 3-2. Because these registers do not have separate instructions for storing them into RAM, they are included in the status registers. As shown in Figure 3-19, several bits in the status registers are reserved and read as logic one's by the LST and LST1 instructions.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST0	ARP			OV	OVM	1	INTM	DP								

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ST1	ARB			CNF	TC	SXM	C†	1	1	HM†	FSM†	XF	FO	TXM	PM	

†On the TMS32020, bits 5, 6, and 9 of ST1 are logic one's.

Figure 3-19. Status Register Organization

Some additional instructions or functions may affect the status bits, as indicated in Table 3-5.

Table 3-5. Status Register Field Definitions

FIELD	FUNCTION
ARB	Auxiliary Register Pointer Buffer. Whenever the ARP is loaded, the old ARP value is copied to the ARB except during an LST instruction. When the ARB is loaded via an LST1 instruction, the same value is also copied to the ARP.
ARP	Auxiliary Register Pointer. This three-bit field selects the AR to be used in indirect addressing. When ARP is loaded, the old ARP value is copied to the ARB register. ARP may be modified by memory-reference instructions when using indirect addressing, and by the LARP, MAR, and LST instructions. ARP is also loaded with the same value as ARB when an LST1 instruction is executed.
C[†]	Carry bit. This bit is set to 1 if the result of an addition generates a carry, or reset to 0 if the result of a subtraction generates a borrow. Otherwise, it is reset after an addition or set after a subtraction, except if the instruction is ADDH or SUBH. ADDH can only set and SUBH only reset the carry bit, but cannot affect it otherwise. The shift and rotate instructions also affect this bit, as well as the SC, RC, and LST1 instructions. Two branch instructions, BC and BNC, have been provided to branch on the status of C. C is set to 1 on a reset.
CNF	On-Chip RAM Configuration Control bit. If set to 0, block B0 is configured as data memory; otherwise, block B0 is configured as program memory. The CNF may be modified by the CNFD, CNFP, and LST1 instructions. $\overline{\text{RS}}$ resets the CNF to 0.
DP	Data Memory Page Pointer. The 9-bit DP register is concatenated with the 7 LSBs of an instruction word to form a direct memory address of 16 bits. DP may be modified by the LST, LDP, and LDPK instructions.
FO	Format bit. When set to 0, the serial port registers are configured as 16-bit registers. When set to 1, the port registers are configured to receive and transmit eight-bit bytes. FO may be modified by the FORT and LST1 instructions. FO is reset to 0.
FSM[†]	Frame Synchronization Mode bit. This bit indicates whether the serial port operates with or without frame sync pulses. When FSM = 1, the serial port operation is initiated following a frame sync pulse on the FSX/FSR inputs. When FSM = 0, the FSX/FSR inputs are ignored and the serial port operates continuously with no frame sync pulses required. The bit is set to 1 by a reset.
HM[†]	Hold Mode bit. When HM = 1, the processor halts internal execution when acknowledging an active $\overline{\text{HOLD}}$. When HM = 0, the processor may continue execution out of internal program memory but puts its external interface in a high-impedance state. This bit is set to 1 by a reset.
INTM	Interrupt Mode bit. When set to 0, all unmasked interrupts are enabled. When set to 1, all maskable interrupts are disabled. INTM is set and reset by the DINT and EINT instructions. $\overline{\text{RS}}$ and $\overline{\text{IACK}}$ also set INTM. INTM has no effect on the unmaskable $\overline{\text{RS}}$ interrupt. Note that INTM is unaffected by the LST instruction.
OV	Overflow Flag bit. As a latched overflow signal, OV is set to 1 when overflow occurs in the ALU. Once an overflow occurs, the OV remains set until a reset, BV, BNV, or LST instruction clears the OV.

[†] TMS320C25 only.

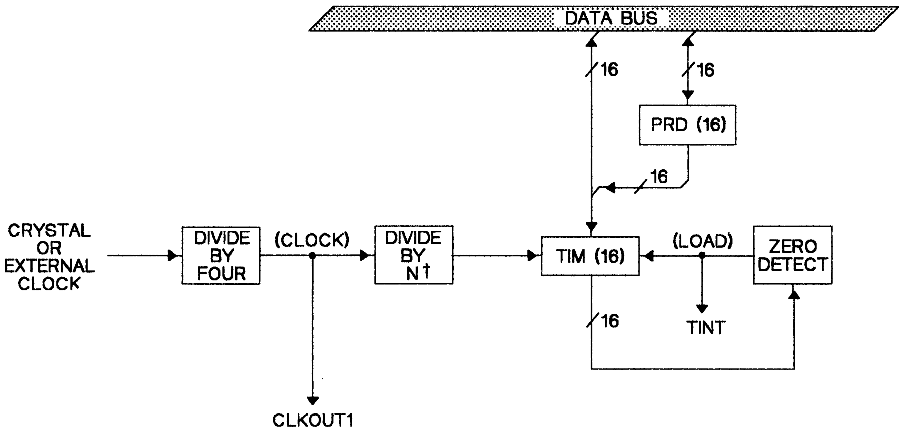
Table 3-5. Status Register Field Definitions (Concluded)

FIELD	FUNCTION
OVM	Overflow Mode bit. When set to 0, overflowed results overflow normally in the accumulator. When set to 1, the accumulator is set to either its most positive or negative value upon encountering an overflow. The SOVM and ROVM instructions set and reset this bit, respectively. LST may also be used to modify the OVM.
PM	Product Shift Mode. If these two bits are 00, the multiplier's 32-bit product is loaded into the ALU with no shift. If PM = 01, the PR output is left-shifted one place and loaded into the ALU, with the LSBs zero-filled. If PM = 10, the PR output is left-shifted by four bits and loaded into the ALU, with the LSBs zero-filled. PM = 11 produces a right shift of six bits, sign-extended. Note that the PR contents remain unchanged. The shift takes place when transferring the contents of the PR to the ALU. PM is loaded by the SPM and LST1 instructions. The PM bits are cleared by $\overline{\text{RS}}$.
SXM	Sign-Extension Mode bit. SXM = 1 produces sign extension on data as it is passed into the accumulator through the scaling shifter. SXM = 0 suppresses sign extension. SXM does not affect the definition of certain instructions; e.g., the ADDS instruction suppresses sign extension regardless of SXM. This bit is set and reset by the SSXM and RSXM instructions, and may also be loaded by LST1. SXM is set to 1 by $\overline{\text{RS}}$.
TC	Test/Control Flag bit. The TC bit is affected by the BIT, BITT, CMPR, LST1, and NORM instructions. The TC bit is set to a 1 if a bit tested by BIT or BITT is a 1, if a compare condition tested by CMPR exists between AR0 and another AR pointed to by ARP, or if the exclusive-OR function of the two MSBs of the accumulator is true when tested by a NORM instruction. Two branch instructions, BBZ and BBNZ, provide branching on the status of the TC.
TXM	Transmit Mode bit. TXM = 1 configures the serial port's FSX pin to be an output. In this mode, a pulse is produced on FSX when DXR is loaded. Transmission then starts on the DX pin. TXM = 0 configures the FSX pin to be an input. TXM is set and reset by the STXM and RTXM instructions and may also be loaded by LST1. $\overline{\text{RS}}$ resets TXM to 0.
XF	XF pin status bit. This status bit indicates the state of the XF pin, a general-purpose output pin. XF is set and reset by the SXF and RXF instructions or may be loaded by LST1. XF is set to 1 by $\overline{\text{RS}}$.

3.6.5 Timer Operation

The TMS320C2x provides a memory-mapped 16-bit timer (TIM) register and a 16-bit period (PRD) register, as shown in Figure 3-20. The on-chip timer is a down counter that is continuously clocked by CLKOUT1 on the TMS320C25. The timer on the TMS32020 is clocked by a signal whose frequency is CLKOUT1/4 or whose period is 4 × CLKOUT1 cycles.

The TIM register is set to the maximum value (>FFFF) on reset for both the TMS32020 and TMS320C25. The PRD register on the TMS320C25 is also initialized by reset to >FFFF. The TMS32020 requires a software initialization of the PRD register (see Example 5-1). The TIM register begins decrementing only after $\overline{\text{RS}}$ is de-asserted. Following this, the TIM and PRD registers may be reloaded under program control. See Section 3.6.3 for reset information.



† The divide ratio where N = 4 on the TMS32020 and N = 1 on the TMS320C25.

Figure 3-20. Timer Block Diagram

The TIM register, data memory location 2, holds the current count of the timer. At every N × CLKOUT1 cycle where N = 4 on the TMS32020 and N = 1 on the TMS320C25, the TIM register is decremented by one. The PRD register, data memory location 3, holds the starting count for the timer. A timer interrupt (TINT) is generated every time the timer decrements to zero. The timer is reloaded with the value contained in the period (PRD) register within the next cycle after it reaches zero so that interrupts can be programmed to occur at regular intervals of (PRD + 1) cycles of CLKOUT1 on the TMS320C25 or (4 × PRD) cycles of CLKOUT1 on the TMS32020. This feature is useful for control operations and for synchronously sampling or writing to peripherals. By programming the PRD register from 1 to 65,535 (>FFFF), a TINT can be generated every 2 to 65,536 cycles on the TMS320C25. Note that, on the TMS32020, a TINT can be generated every 4 to 262,140 cycles. A PRD register value of zero is not allowed.

The timer and period registers can be read from or written to on any cycle. The count can be monitored by reading the TIM register. A new counter period can be written to the period register without disturbing the current timer count. The timer will then start the new period after the current count is complete. If both the PRD and TIM registers are loaded with a new period, the timer begins decrementing the new period without generating an interrupt. Thus, the programmer has complete control of the current and next periods of the timer.

If the timer is not used, TINT should be masked or all maskable interrupts disabled by a DINT instruction. The PRD register can then be used as a general-purpose data memory location. If TINT is used, the PRD and TIM registers should be programmed before unmasking the TINT.

3.6.6 Repeat Counter

The repeat counter (RPTC) is an 8-bit counter, which when loaded with a number N, causes the next single instruction to be executed N + 1 times. The RPTC can be loaded with a number from 0 to 255 using either the RPT (repeat) or RPTK (repeat immediate) instructions. This results in a maximum of 256 executions of a given instruction. RPTC is cleared by reset.

The repeat feature can be used with instructions such as multiply/accumulates (MAC/MACD), block moves (BLKD/BLKP), I/O transfers (IN/OUT), and table read/writes (TBLR/TBLW). These instructions, which are normally multicycle, are pipelined when using the repeat feature, and effectively become single-cycle instructions. For example, the table read instruction may take three or more cycles to execute, but when repeated, a table location can be read every cycle. Note that not all instructions can be repeated (see Section 4.3 and Appendix D for more information).

3.6.7 Powerdown Mode (TMS320C25)

When operated in the powerdown mode, the TMS320C25 enters a dormant state and requires approximately one-half the power normally needed to supply the device (see the data sheet, Appendix A). Powerdown mode is invoked either by executing an IDLE instruction or by driving the $\overline{\text{HOLD}}$ input low while the HM status bit is set to one.

While in powerdown mode, all of the internal contents of the TMS320C25 are maintained to allow operation to continue unaltered when powerdown mode is terminated. During that time, the data and address lines and the control signals ($\overline{\text{PS}}$, $\overline{\text{DS}}$, $\overline{\text{IS}}$, $\overline{\text{STRB}}$, and R/$\overline{\text{W}}$) are all maintained in the high-impedance state. Powerdown mode is terminated upon receipt of an interrupt when an IDLE instruction is being executed or by removal of the $\overline{\text{HOLD}}$ input. (Refer to the IDLE instruction description in Section 4 and the hold function description in Section 3.10.3 for further information.)

3.7 External Memory and I/O Interface

The TMS320C2x supports a wide range of system interfacing requirements. Data, program, and I/O address spaces provide interfacing to memory and I/O, thus maximizing system throughput. The local memory interface consists of:

- A 16-bit parallel data bus (D15-D0),
- A 16-bit address bus (A15-A0),
- Data, program, and I/O space select ($\overline{DS}$, $\overline{PS}$, and $\overline{IS}$) signals, and
- Various system control signals.

The R/$\overline{W}$ (read/write) signal controls the direction of the transfer, and $\overline{STRB}$ (strobe) provides a timing signal to control the transfer.

The TMS320C2x I/O space consists of 16 input and 16 output ports. These ports provide the full 16-bit parallel I/O interface via the data bus on the device. A single input or output operation, using the IN or OUT instructions, typically takes two cycles; however, when used with the repeat counter, the operation becomes single-cycle.

I/O design is simplified by having I/O treated the same way as memory. I/O devices are mapped into the I/O address space using the processor's external address and data buses in the same manner as memory-mapped devices. When addressing internal memory, the data bus must be in the high-impedance state and the control signals go to an inactive state (logic high). Refer to Section 5 for the effect instructions have on I/O.

Interfacing to memory and I/O devices of varying speeds is accomplished by using the READY line. When communicating with slower devices, the TMS320C2x processor waits until the other device completes its function, signals the processor via the READY line, and continues execution (see Section 6).

3.7.1 Memory Combinations

The exact sequence of operations performed as instructions execute depends on the areas in memory where the instructions and operands are located. There are six possible combinations of program and data memory since information can be located in either internal RAM, external memory, or internal ROM (available only on the TMS320C25). The six possible combinations are:

- Program Internal RAM/Data Internal (PI/DI)
- Program Internal RAM/Data External (PI/DE)
- Program External/Data Internal (PE/DI)
- Program External/Data External (PE/DE)
- Program Internal ROM/Data Internal (PR/DI) on the TMS320C25
- Program Internal ROM/Data External (PR/DE) on the TMS320C25.

Appendix D provides cycle timings for instructions both when repeated and when not repeated. The following is a summary of program execution, organized according to memory configuration.

PI/DI or PR/DI	When both program and data memory are on-chip, the processor runs at full speed with no wait states. Note that IN and OUT instructions have different cycle timings when program memory is internal; IN requires two cycles to execute whereas OUT requires only one cycle.
PE/DI	This memory mode can run at full speed if external program memory is sufficiently fast since internal data operations can occur coincident with external program memory accesses. If external program memory is not fast enough, wait states may be generated using the READY input.
PI/DE, PE/DE, or PR/DE	Additional cycles are required to execute instructions that reference an external data memory space. At least two cycles are required to execute 'read from external data memory' instructions such as ADD, LAR, etc. Further additional cycles may be required due to wait states if external data memory is not fast enough to be accessed within a single cycle. Note, however, that the TMS320C25 has the capability of executing 'write to external data memory' instructions in a single cycle when program memory is internal (two cycles are required if program memory is also external). Additional cycles are also required in this case if external data memory is not sufficiently fast.

In all memory configurations where the same bus is used to communicate with external data, program, or I/O space, the number of cycles required to execute a particular instruction may further vary depending on whether the next instruction fetch is from internal or external program memory. Instruction execution and operation of the pipeline are discussed in Section 3.6.2 and in the succeeding subsections.

3.7.2 Internal Clock Timing Relationships

The crystal or external clock source frequency is divided to produce an internal four-phase clock. The four phases are defined by CLKOUT1 and CLKOUT2, as shown in Figure 3-21. In this document (and on the TMS320C25), the start of quarter-phase 3 (Q3) is defined as the rising edge of CLKOUT1. Refer to Appendix C for device phase definitions.

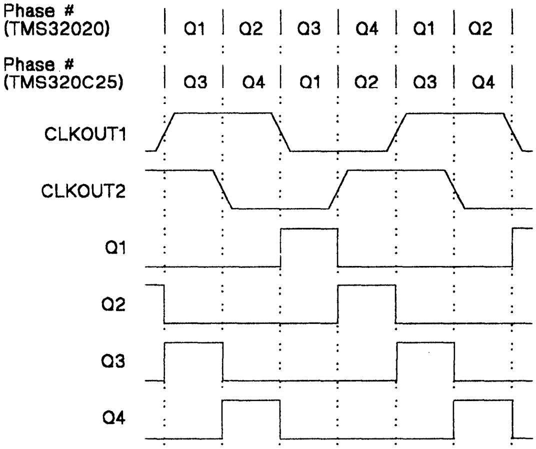


Figure 3-21. Four-Phase Clock

3.7.3 General-Purpose I/O Pins ($\overline{\text{BIO}}$ and XF)

The TMS320C2x has two general-purpose pins that are software-controlled. The $\overline{\text{BIO}}$ pin is a branch control input pin, and the XF pin is an external flag output pin.

The $\overline{\text{BIO}}$ pin is useful for monitoring peripheral device status. It is especially useful as an alternative to using an interrupt when it is necessary not to disturb time-critical loops. When the $\overline{\text{BIO}}$ input pin is active (low), execution of the BIOZ instruction causes a branch to occur.

In Figure 3-22, $\overline{\text{BIO}}$ is sampled at the end of Q4 (Q2 on the TMS32020). The timing diagram shown is for a sequence of single-cycle, single-word instructions without branches located in external memory. Because of variations in pipelining due to instructions prior to and following the BIOZ instruction, this timing may vary. Therefore, it is recommended that several cycles of setup be provided if $\overline{\text{BIO}}$ is to be recognized on a particular cycle.

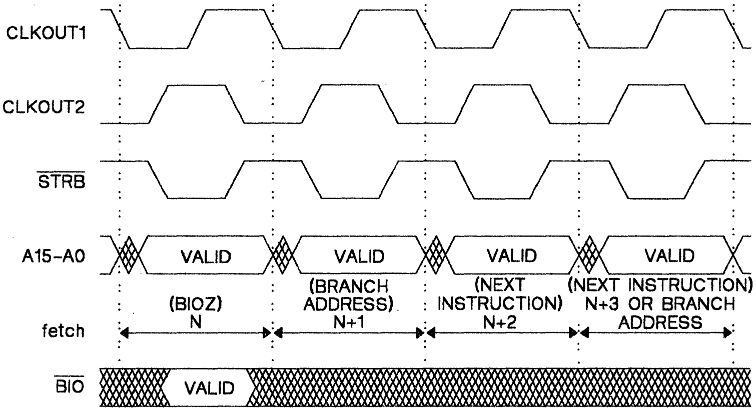
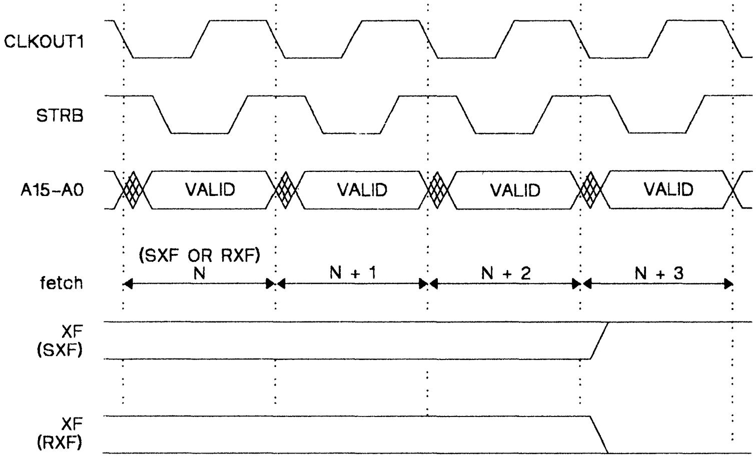


Figure 3-22. BIO Timing Diagram

The XF (external flag) output pin is set to a high level by the SXF (set external flag) instruction and reset to a low level by the RXF (reset external flag) instruction. XF is set high by RS.

The relationship between the time the SXF/RXF instruction is fetched before the XF pin is set or reset is shown in Figure 3-23. As with BIO, the timing shown for XF is for a sequence of single-cycle, single-word instructions located in external memory. Actual timing may vary with different instruction sequences.



NOTES: 1. N is the program memory location for the current instruction.
2. This example only shows the execution of single-cycle instructions fetched from external program memory.

Figure 3-23. External Flag Timing Diagram

3.8 Interrupts

The TMS320C2x has three external maskable user interrupts ($\overline{\text{INT}}$2-$\overline{\text{INT}}$0), available for external devices that interrupt the processor. Internal interrupts are generated by the serial port (RINT and XINT), by the timer (TINT), and by the software interrupt (TRAP) instruction. Interrupts are prioritized with reset ($\overline{\text{RS}}$) having the highest priority and the serial port transmit interrupt (XINT) having the lowest priority.

3.8.1 Interrupt Operation

This subsection details interrupt organization and management. Vector locations and priorities for all internal and external interrupts are shown in Table 3-6. The TRAP instruction, used for software interrupts, is not prioritized but is included here since it has its own vector location. Each interrupt address has been spaced apart by two locations so that branch instructions can be accommodated in those locations if desired.

Table 3-6. Interrupt Locations and Priorities

INTERRUPT NAME	MEMORY LOCATION	PRIORITY	FUNCTION
$\overline{\text{RS}}$	0	1 (highest)	External reset signal
$\overline{\text{INT}}$0	2	2	External user interrupt #0
$\overline{\text{INT}}$1	4	3	External user interrupt #1
$\overline{\text{INT}}$2	6	4	External user interrupt #2
	8-23		Reserved locations
TINT	24	5	Internal timer interrupt
RINT	26	6	Serial port receive interrupt
XINT	28	7 (lowest)	Serial port transmit interrupt
TRAP	30	N/A	TRAP instruction address

When an interrupt occurs, it is stored in the 6-bit Interrupt Flag Register (IFR). This register is set by the external user interrupts $\overline{\text{INT}}$(2-0) and the internal interrupts RINT, XINT, and TINT. Each interrupt is stored in the IFR until it is recognized, and then automatically cleared by the $\overline{\text{IACK}}$ (interrupt acknowledge) signal or the $\overline{\text{RS}}$ (reset) signal. The $\overline{\text{RS}}$ signal is not stored in the IFR. No instructions are provided for reading from or writing to the IFR.

The TMS320C2x has a memory-mapped Interrupt Mask Register (IMR) for masking external and internal interrupts. The layout of the register is shown in Figure 3-24. A 1 in bit positions 5 through 0 of the IMR enables the corresponding interrupt, provided that INTM = 0. The IMR is accessible with both read and write operations but cannot be read using BLKD. When the IMR is read, the unused bits (15 through 6) are read as one's. The lower six bits are used to write to or read from the IMR. Note that $\overline{\text{RS}}$ is not included in the IMR, and therefore the IMR has no effect on reset.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED										XINT	RINT	TINT	$\overline{\text{INT2}}$	$\overline{\text{INT1}}$	$\overline{\text{INT0}}$

Figure 3-24. Interrupt Mask Register (IMR)

The INTM (interrupt mode) bit, which is bit 9 of status register ST0, enables or disables all maskable interrupts. INTM = 0 enables all the unmasked interrupts, and INTM = 1 disables these interrupts. The INTM is set to 1 by the $\overline{\text{IACK}}$ (interrupt acknowledge) signal, the DINT instruction, or a reset. This bit is reset to 0 by the EINT instruction. Note that the INTM does not actually modify the IMR or IFR.

The TMS320C2x has a built-in mechanism for protecting multicycle instructions from interrupts. If an interrupt occurs during a multicycle instruction, the interrupt is not processed until the instruction is completed. This mechanism also applies to instructions that become multicycle due to the READY signal.

In addition, the device does not allow interrupts to be processed when an instruction is being repeated via the RPT or RPTK instructions. The interrupt is stored in the IFR until the repeat counter (RPTC) decrements to zero, and then the interrupt is processed. Even if the interrupt is de-asserted while the TMS320C2x is processing the RPT or RPTK, the interrupt will still be latched by IFR and pending until RPTC decrements to zero.

If both the $\overline{\text{HOLD}}$ line and an interrupt go active during a multicycle instruction or a repeat loop, the $\overline{\text{HOLD}}$ takes control of the processor at the end of the instruction or loop. When $\overline{\text{HOLD}}$ is released, the interrupt is acknowledged.

Interrupts cannot be processed between EINT and the next instruction in a program sequence. For example, if an interrupt occurs during an EINT instruction execution, the device always completes EINT as well as the following instruction before the pending interrupt is processed. This insures that a RET can be executed before the next interrupt is processed, assuming that a RET instruction follows the EINT. The state of the machine, upon receiving an interrupt, may be saved and restored (see Section 5.3.1).

3.8.2 External Interrupt Interface

Interrupts may be asynchronously edge- or level-triggered. In the functional logic organization for $\overline{\text{INT}}$(2-0), shown in Figure 3-25, the external interrupt $\overline{\text{INT0}}$ is connected to an edge-triggered flip-flop. The $\overline{\text{INT0}}$ signal is ORed with the interrupt edge flip-flop Q output and synchronized with internal quarter-phases 1 and 2 to produce an interrupt signal (see Appendix B for phase relationships on the TMS32020). In this way, the device can handle both edge-triggered and level-triggered interrupts.

Due to the level sensitivity of the external interrupts and the synchronization of the interrupts (first on Q2, then on Q1 of the following machine cycle), it is necessary to bring the $\overline{\text{INT}}$ line inactive high at least two cycles before enabling interrupts (EINT). If this criteria is not met, the TMS320C25 will immediately take the interrupt trap following the EINT plus the next instruction.

If the INTM bit and flag register have been properly enabled, the interrupt signal is accepted by the processor. An $\overline{\text{IACK}}$ (interrupt acknowledge) signal is then generated. The $\overline{\text{IACK}}$ clears the appropriate interrupt edge flip-flop and disables the INTM latch. The logic is the same for $\overline{\text{INT}}$1 and $\overline{\text{INT}}$2.

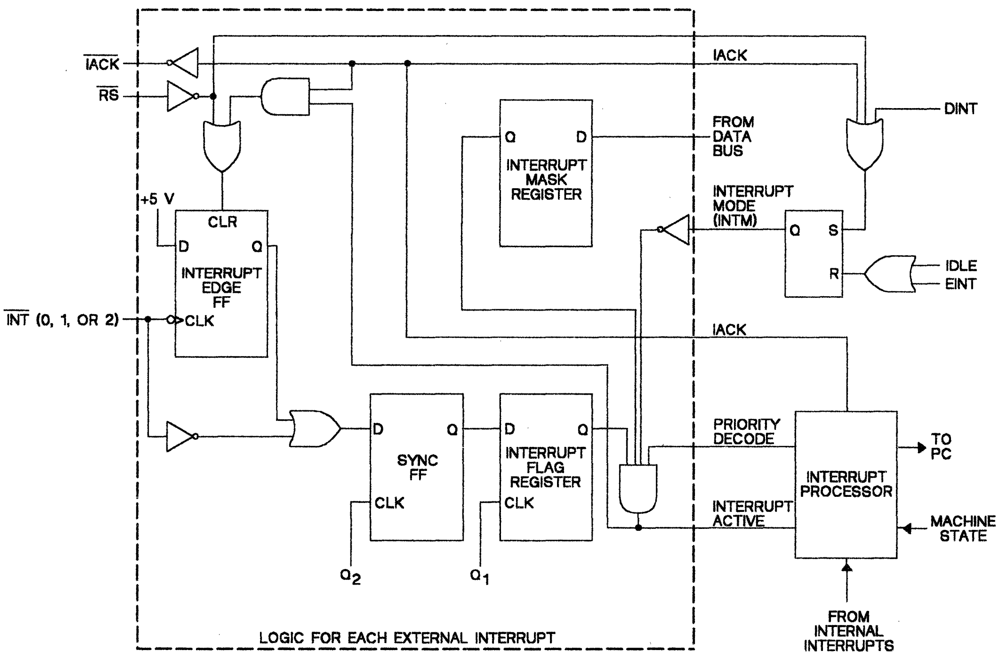
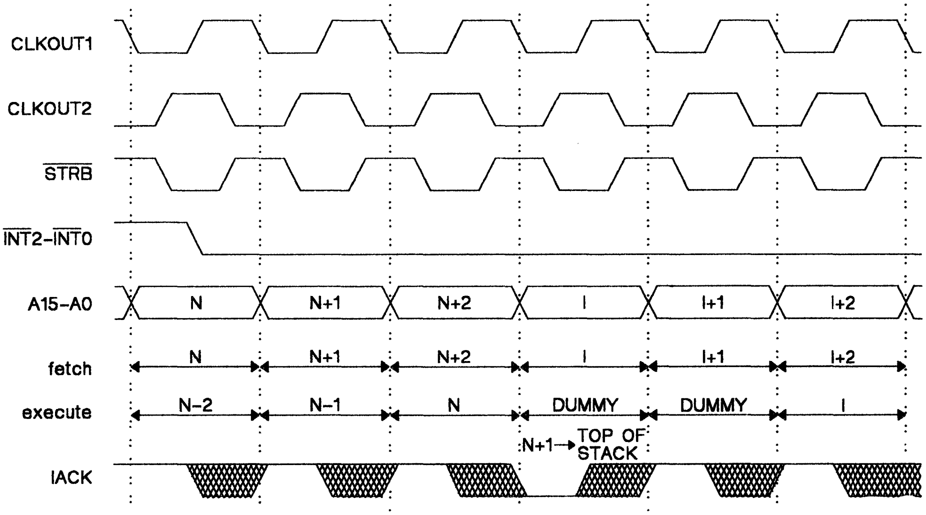


Figure 3-25. Internal Interrupt Logic Diagram

In a typical interrupt ($\overline{\text{INT}}$2-$\overline{\text{INT}}$0) operation, the interrupt is generated by a negative-going edge and the IFR bit is set. Since INTM is disabled when the interrupt is acknowledged, the level may continue to be present on the $\overline{\text{INT}}$ input without generating further interrupts. If the level is removed before an EINT instruction is executed, no further interrupts are generated. If a low level continues to be present after the EINT, another interrupt is generated after the EINT/next instruction sequence. In addition, if the $\overline{\text{INT}}$ pin is pulsed between the previous $\overline{\text{IACK}}$ and EINT, another interrupt is generated after EINT/RET, because the corresponding IFR bit is again set.

Figure 3-26 shows an interrupt, interrupt acknowledge, and various other signals for the special case of single-cycle instructions. An interrupt generated during the current (N) fetch cycle still allows the fetch and execution of that instruction. The N+1 and N+2 instructions are also fetched, then discarded, and the address N+1 is pushed onto the top of the stack. The instruction is fetched again upon a return command from the interrupt routine.

Two dummy execute cycles occur on an interrupt, as shown in the timing diagram for the TMS320C25 (Figure 3-26). The $\overline{IACK}$ signal is asserted low during CLKOUT1 low when the device initiates a fetch from interrupt location I. Therefore, an external device can determine the interrupt that occurred by latching the address bus value present on A4-A1 with the rising edge of CLKOUT2 when $\overline{IACK}$ is low.



NOTES: 1. N is the program memory location for the current instruction.
2. I is the interrupt vector location in program memory for the active interrupt.
3. For simplicity, this example only shows the execution of single-cycle instructions fetched from external program memory, rather than multicycle instructions.

Figure 3-26. Interrupt Timing Diagram (TMS320C25)

3.9 Serial Port

A full-duplex on-chip serial port provides direct communication with serial devices such as codecs, serial A/D converters, and other serial systems. The interface signals are compatible with codecs and many other serial devices with a minimum of external hardware. The serial port may also be used for intercommunication between processors in multiprocessing applications.

Both receive and transmit operations are double-buffered on the TMS320C25, thus allowing a continuous bit stream even if FSX is an output. The use of the frame sync mode (FSM) bit provides continuous operation that once initiated requires no further frame synchronization pulses. No minimum CLKR/CLKX frequency (f_{min} = 0 Hz) is required for serial port operation.

The bits, pins, and registers that control serial port operation are listed in Table 3-7. Availability of a function on a particular device is also indicated.

Table 3-7. Serial Port Bits, Pins, and Registers

SERIAL PORT BITS/PINS/REGISTERS		TMS32020	TMS320C25
FO	Format bit	Yes	Yes
TXM	Transmit mode bit	Yes	Yes
FSM	Frame synchronization mode bit	No	Yes
CLKX	Transmit clock signal	Yes	Yes
CLKR	Receive clock signal	Yes	Yes
DX	Transmitted serial data signal	Yes	Yes
DR	Received serial data signal	Yes	Yes
FSX	Transmit framing synchronization signal	Yes	Yes
FSR	Receive framing synchronization signal	Yes	Yes
DXR	Data transmit register	Yes	Yes
DRR	Data receive register	Yes	Yes
XSR	Transmit shift register	No	Yes
RSR	Receive shift register	No	Yes

The serial port uses two memory-mapped registers: the data transmit register (DXR) that holds the data to be transmitted by the serial port, and the data receive register (DRR) that holds the received data (see Figure 3-27). Both registers operate in either the 8-bit byte mode or 16-bit word mode, and may be accessed in the same manner as any other data memory location. Each register has an external clock, a framing synchronization pulse, and associated shift registers. Any instruction accessing data memory can be used to read from or write to these registers; however, the BLKD (block move from data memory to data memory) instruction cannot be used to read these registers. The DXR and DRR registers are mapped into locations 0 and 1 in the data address space. The XSR and RSR registers are not directly accessible through software.

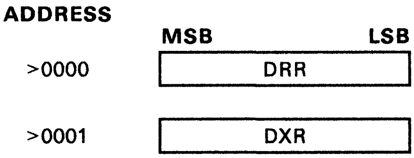


Figure 3-27. The DRR and DXR Registers

If the serial port is not being used, the DXR and DRR registers can be used as general-purpose registers. In this case, the CLKR or FSR should be connected to a logic low to prevent a possible receive operation from being initiated.

Three bits in status register ST1 are used to control the serial port operation: FO, TXM, and FSM. The FO (format) bit defines whether data to be transmitted and received is an 8-bit byte or a 16-bit word. If FO = 0, the data is formatted in 16-bit words. If FO = 1, the data is formatted in 8-bit bytes. In the 8-bit mode, only the eight least-significant bits are used for transmit/receive operations. The FO bit is loaded by the FORT (format serial port registers) instruction. On reset, FO is set to 0.

The TXM (transmit mode) bit is used to determine if the frame synchronization pulse for the transmit operation is generated externally or internally. If TXM = 1, the FSX pin becomes an output pin, and a framing pulse is produced on the FSX pin every time the DXR register is loaded. This framing pulse is synchronized with the rising edge of CLKX. If TXM = 0, the FSX pin becomes an input pin. The TMS320C2x then waits for an external synchronization pulse before beginning transmission. On a reset, TXM is set to zero, configuring FSX to be an input. The TXM bit can be loaded by the LST1, STXM, or RTXM instructions. If DXR on the TMS32020 is loaded before the previous word is completely sent, the serial port immediately begins transmitting the new word. The bits of the previous word that have not been sent are lost. If TXM = 1 on the TMS32020, a new FSX pulse is generated. If TXM = 0, the serial port immediately begins transmitting the new word without waiting for a new external FSX pulse.

The FSM (frame synchronization mode) status register bit is used to select whether frame sync pulses are required for each serial port transfer. If FSM = 1, frame sync pulses are required; if FSM = 0, they are not required. FSM is set by the SFSM (set frame synchronization mode) instruction and cleared by the RFSM (reset frame synchronization mode) instruction. When FSM = 1 and frame sync pulses are required, an FSX pulse will cause the XSR to be loaded with data from the DXR, and transmission will begin. If an FSX is presented prior to the last bit of the current transmission, the XSR will be reloaded from the DXR, thus aborting the current transmission and immediately beginning a new one.

The frame sync mode is useful in communicating to 'PCM highways.' For ATT T1 and CCITT G711/712 lines, the processor can communicate directly in these formats by counting the transmitted/received bytes in software and performing SFSM/RFSM instructions as needed to set/reset the FSM bit.

3.9.1 Transmit and Receive Operations

The transmit and receive sections of the serial port are implemented separately to allow independent transmit and receive operations. Externally, the serial port interface is implemented using the six serial port pins. Figure 3-28 shows the registers and pins used in transmit and receive operations. Note that on the TMS32020, the DXR and XSR are combined as a single register, as well as the DRR and RSR.

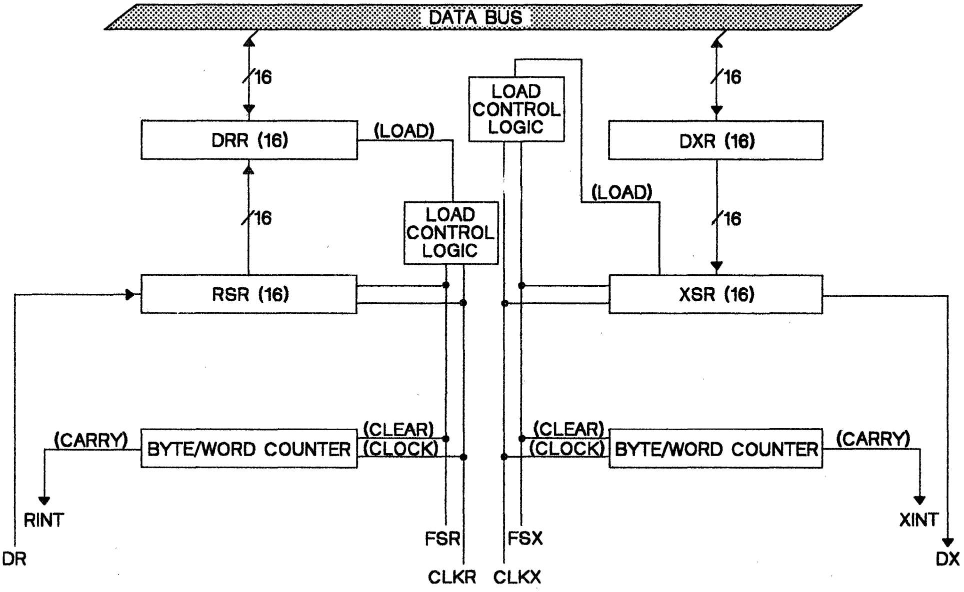


Figure 3-28. Serial Port Block Diagram

The data on the DX and DR pins is clocked out of or into the XSR or RSR on the TMS320C25 by the CLKX or CLKR signal, respectively. On the TMS32020, the data on the pins is clocked directly out of the DXR or into the DRR. CLKX and CLKR are only required to be present during actual serial port transfers, and may be stopped when no data is being transferred. Data bits can be transferred in either 8-bit bytes or 16-bit words. Data is clocked out to DXR on the rising edges of CLKX, while data is clocked in from DRR on the falling edges of CLKR. The MSB of the data is transferred first.

The XSR and RSR are connected to the DXR and DRR, respectively. For transmit operations, the contents of DXR are transferred to XSR when a new transmission begins. For a receive operation, the contents of RSR are transferred to DRR when all of the bits have been received. Thus, the serial port is double-buffered since data may be transferred to or from the DXR or DRR while another transmit or receive operation is being performed.

Serial port transfers on the TMS320C25 are generally initiated by a frame sync pulse. The exception to this is when the continuous mode of operation is used with FSM = 0, as described in a subsequent paragraph. Frame sync pulses are input on FSX for transmit operations and on FSR for receive operations.

The transmit timing diagram is shown in Figure 3-29. The transmit operation begins when data is written into the data transmit register (DXR). The TMS320C2x begins transmitting data when the frame synchronization pulse (FSX) goes low while CLKX is high or going high. The data, starting with the MSB, is then shifted out via the DX pin with the rising edge of CLKX. When all bits have been transmitted, an internal transmit interrupt (XINT) is generated on the falling edge of CLKX. When the serial port is not transmitting, DX is placed in the high-impedance state.

DX and FSX are unaffected by assertion of the $\overline{\text{HOLD}}$ input. Upon assertion of $\overline{\text{HOLD}}$, any serial port transmission in progress on the DX pin is completed before DX is placed in the high-impedance state. FSX remains configured as either an input or output, remaining low if it is an output.

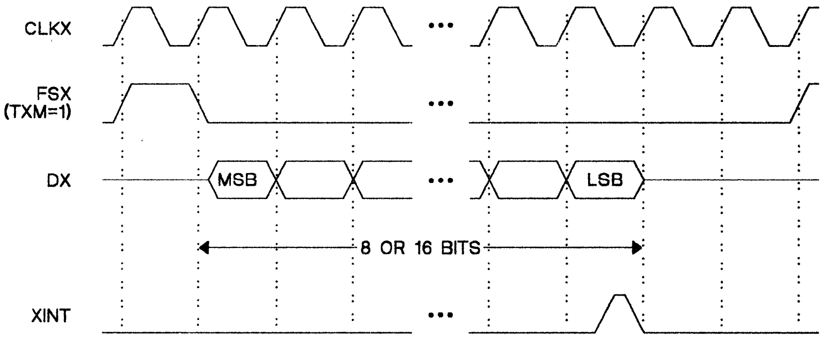


Figure 3-29. Serial Port Transmit Timing Diagram

The receive operation is similar to the transmit operation. The receive timing diagram is shown in Figure 3-30. Reception is initiated by a frame synchronization pulse on the FSR pin. After FSR goes low, data on the DR pin is clocked into the RSR register on the TMS320C25 (DRR register on the TMS32020) on every negative-going edge of CLKR. The first data bit is considered the MSB, and RSR is filled accordingly. After all the bits have been received, (as specified by FO), an internal receive interrupt (RINT) is generated on the falling edge of CLKR and the contents of RSR are transferred to DRR. If DRR is read before an RINT is received, the bits that have not been clocked in yet at the time of the read will contain the data from a previous transfer. To prevent a possible overrun of the DRR register, the DRR must be read before the next FSR.

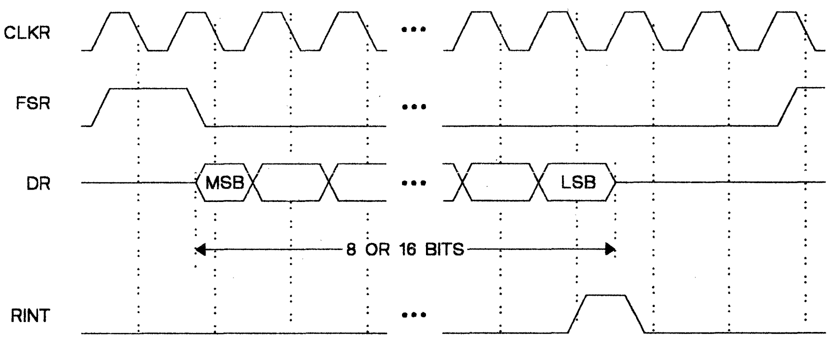


Figure 3-30. Serial Port Receive Timing Diagram

3.9.2 Timing and Framing Control

Upon completion of a serial port transfer, an internal interrupt is generated. The RINT interrupt is generated for a receive operation, and XINT is generated for a transmit operation. RINT and XINT are generated on the rising edge of CLKR and CLKX, respectively, after the last bit is transferred. Note that if DRR is read before a RINT is received, it will contain the data from the previous operation. Similarly, if DXR is loaded more than once after an XINT is generated (in the continuous transmission mode), only the last value written will be loaded into XSR for the next transmit operation.

When the TMS320C2x is reset, TXM is cleared to zero, and DX is placed in a high-impedance state. Any transmit or receive operation that is in progress when the reset occurs is terminated.

The transmit framing synchronization pulse can be generated internally or externally. The maximum speed of the serial port is 5 MHz. The timing of the serial port signals is compatible with the TI/Intel 29C1x series codecs. The timing is also compatible with the AMI S3506 series codecs if the frame synchronization signals are inverted.

Serial port transfers on the TMS320C25 are generally initiated by a frame sync pulse, except when the continuous mode of operation is used with FSM = 0. Frame sync pulses are input on FSX for transmit operations and on FSR for receive operations. If FSM = 1, frame sync pulses are required; if FSM = 0, they are not required. FSM is set by the SFSM (set frame synchronization mode) instruction and cleared by the RFSM (reset frame synchronization mode) instruction.

3.9.3 Burst-Mode Operation

In burst-mode serial port operation, transfers are separated in time by periods of no serial port activity (the serial port does not operate continuously). For burst-mode operation, FSM must be set to one. Timing of the serial port in this mode of operation is shown in Figure 3-31 and Figure 3-32.

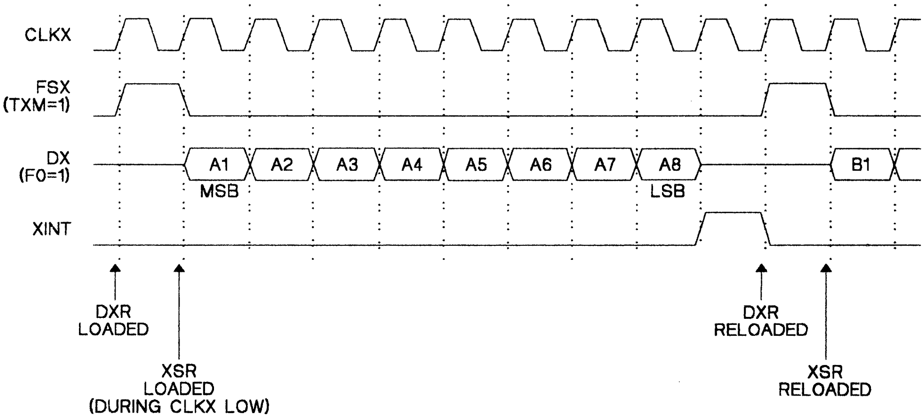


Figure 3-31. Burst-Mode Serial Port Transmit Operation

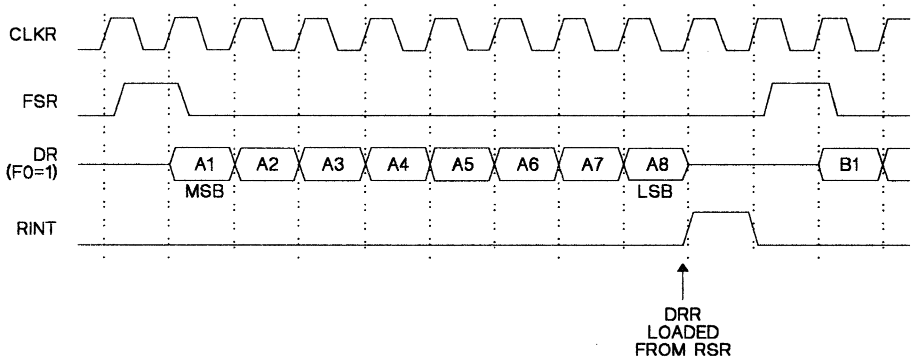


Figure 3-32. Burst-Mode Serial Port Receive Operation

When TXM = 1 (in the transmit operation) and the serial port register DXR is loaded, a framing pulse is generated on the next rising edge of CLKX. XSR is loaded with the current contents of DXR while FSX is high and CLKX is low. Transmission begins when FSX goes low while CLKX is high or is going high.

Figure 3-31 shows the timing for the byte mode (FO = 1). XINT is generated on the rising edge of CLKX after all 8 or 16 bits have been transmitted and DX is placed in the high-impedance state. If DXR is reloaded before the next rising edge of CLKX after XINT, FSX will again be generated as shown, and XSR will be reloaded.

The receive operation is similar to the transmit operation. The contents of RSR are loaded into DRR while CLKR is low, just after reception of the last bit sent by the transmitting device (see Figure 3-32). RINT is generated on the next rising edge of CLKR, and DRR may be read at any time before the reception of the final bit of the next transmission. When operating in the byte mode, the eight MSBs of the DRR are the contents of the eight LSBs of the DRR prior to reception of the current byte, as shown in Figure 3-33 for the TMS320C25. On the TMS32020, the most-significant byte is unaffected by successive 8-bit receive operations.

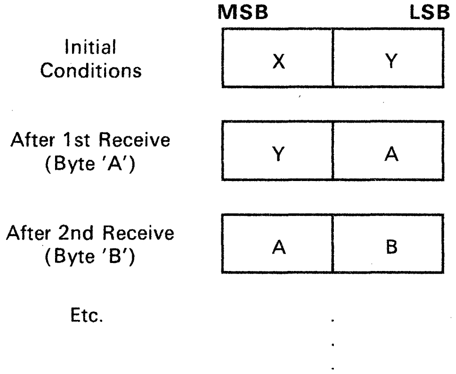


Figure 3-33. Byte-Mode DRR Operation (TMS320C25)

3.9.4 Continuous Operation Using Frame Sync Pulses (TMS320C25)

The TMS320C25 provides two modes of operation that allow the use of a continuous stream of serial data. When FSM = 1, frame sync pulses are required. Since DXR is double-buffered, continuous operation is achieved even if TXM = 1. Writing to DXR during a serial port transmission does not abort the transmission in progress, but instead DXR stores that data until XSR can be reloaded. As long as DXR is reloaded before the CLKX rising edge on the final bit being transmitted, the FSX pulse will go high on the rising edge of CLKX during the transmission of the final bit and fall on the next rising edge when transmission of the word just loaded begins. If DXR is not reloaded within this period and FSM = 1, the DX pin will be placed in a high-impedance state for at least one CLKX cycle until DXR is reloaded (as described in the previous section). Figure 3-34 and Figure 3-35 show the timing diagrams for the continuous operation with frame sync pulses.

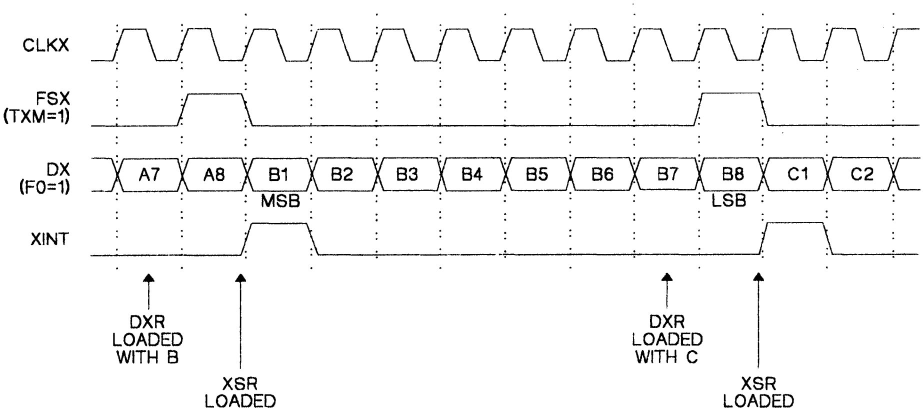


Figure 3-34. Serial Port Transmit Continuous Operation (FSM = 1)

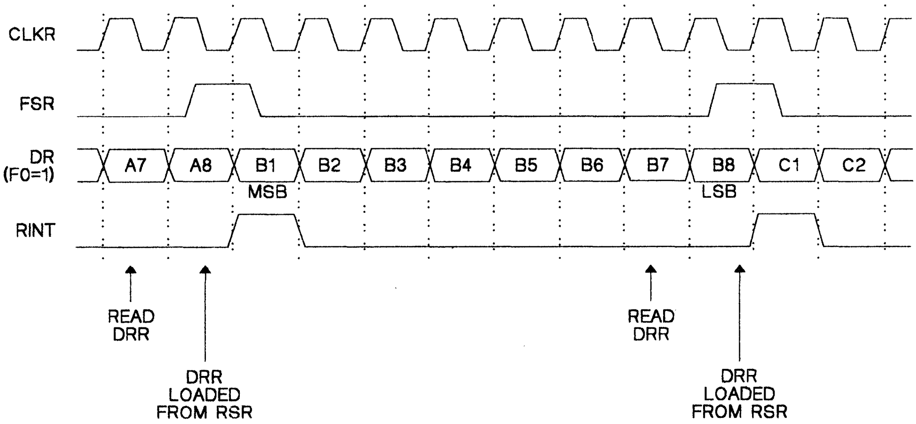


Figure 3-35. Serial Port Receive Continuous Operation (FSM = 1)

Continuous receive operation with FSM = 1 is identical to that of burst-mode operation with the exception that FSR is pulsed during reception of the final bit.

3.9.5 Continuous Operation Without Frame Sync Pulses (TMS320C25)

The continuous mode of operation on the TMS320C25 allows transmission and reception of a continuous bit stream without requiring frame sync pulses every 8 or 16 bits. This mode is selected by setting FSM = 0.

Figure 3-36 and Figure 3-37 show operation of the serial port for both states of FSM to illustrate differences in operation for each case. FSM is initially set to one, and frame sync pulses are required to initiate serial transfers. Before the completion of the transmission (i.e., before the next serial port interrupt), the FSM but must be reset to zero by means of an RFSM (reset FSM) instruction. RFSM can occur either before or after the write to DXR or read from DRR. From this point on, the FSX and FSR inputs are ignored, with transmission occurring every CLKX cycle and reception occurring every CLKR cycle as long as those clocks are present.

If FSX is configured as an output, it will remain low until FSM is set back to one and DXR is reloaded. If DXR is not reloaded with new data every XINT (every 8 or 16 CLKX cycles depending on FO), the last value loaded will be transmitted on DX continuously. Note that this is different from the case with FSM = 1 where DX is placed into a high-impedance state if DXR is not reloaded before transmission of the last bit of the current word in XSR. For example, if byte C is not loaded into DXR as indicated in Figure 3-36, bits of byte B (B1-B8) will be retransmitted instead of bits of byte C as shown.

For receive operations, DRR is loaded from RSR (and an RINT is generated) every 8 or 16 CLKR cycles (depending on FO), regardless of whether or not DRR has been read. An overrun of DRR is also possible with FSM = 1 if DRR is not read before the next RINT. The only way to stop continuous transmission or reception once started, when FSM = 0, is to either stop CLKX or CLKR or to perform an SFSM (set FSM) instruction.

Continuous transmission without frame sync pulses is very useful in communicating directly to telephone system PCM highways. For ATT T1 and CCITT G711/712 lines, FSX and FSR pulses are generated only every 24 or 32 bytes. By counting the transmitted and received bytes in software after an initial FSX or FSR and performing SFSM and RFSM instructions as required, the TMS320C25 can easily be made to communicate in these formats.

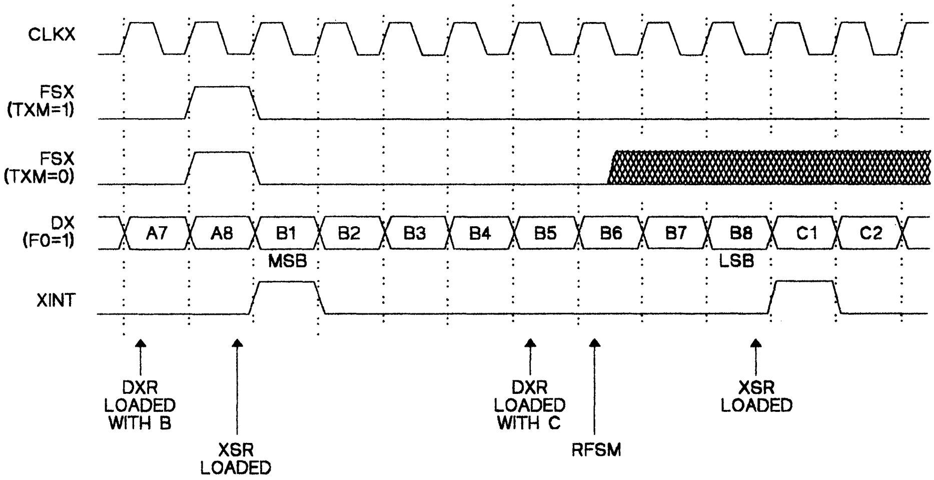


Figure 3-36. Serial Port Transmit Continuous Operation (FSM = 0)

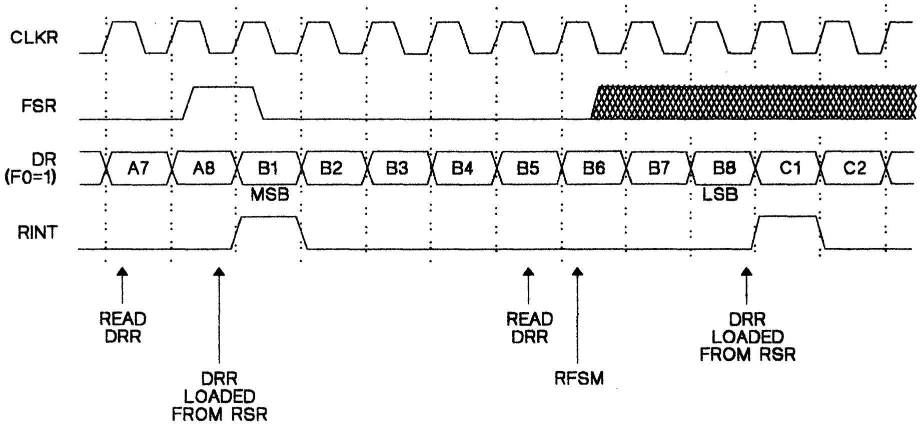


Figure 3-37. Serial Port Receive Continuous Operation (FSM = 0)

3.9.6 Initialization of Continuous Operation Without Frame Sync Pulses (TMS320C25)

FSM is normally initialized during an XINT or RINT service routine to enable or disable FSX and FSR, respectively, for the next serial port operation. It is necessary to start this mode with FSM = 1 so that the first data transferred out of the serial port is the data written to the DXR register. Otherwise, the serial port starts transmitting the contents of the shift register before loading it with the value stored in the DXR register. Upon each completion of a data packet transmission, it loads the data contained in the DXR register into the shift register and continues transmitting. After the first frame pulse has been generated by or sent to the TMS320C25, the FSM bit must be reset to 0 using the RFSM instruction. This must be done before the next serial port interrupt to assure continuous transmission. If continuous transmission is stopped via software, this initiation sequence must be repeated to restart the continuous mode operation.

As shown in Figure 3-38 and Figure 3-39, RFSM may occur before a write to DXR, regardless of the state of TXM. If TXM = 1, FSX is generated in a normal manner on the next rising edge of CLKX, but only once. If TXM = 0, the TMS320C25 waits to transmit until FSX is pulsed, but from then on, the FSX input is ignored. Note that just as in the case of continuous-mode operation without sync pulses described in Section 3.9.5, the first data written to DXR (byte A) is output twice unless DXR is reloaded before the second transmission is started. It is important to consider this dummy cycle when using continuous-mode serial operation.

The receive timings are the same as those for the transmit operations with TXM = 0. The TMS320C25 waits to receive data until FSR is pulsed, but thereafter the FSR input is ignored. No dummy cycle is associated with the receive operation due to the post-buffering nature of DRR as opposed to the prebuffering nature of DXR.

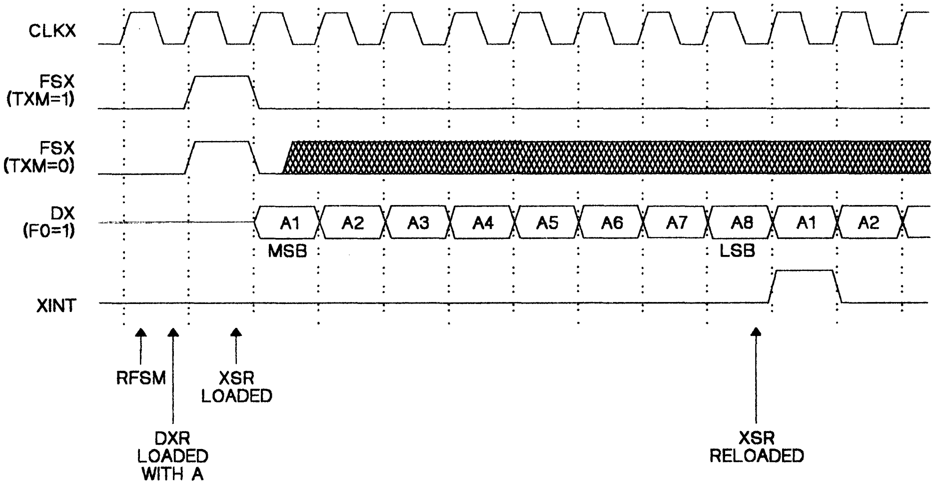


Figure 3-38. Continuous Transmit Operation Initialization

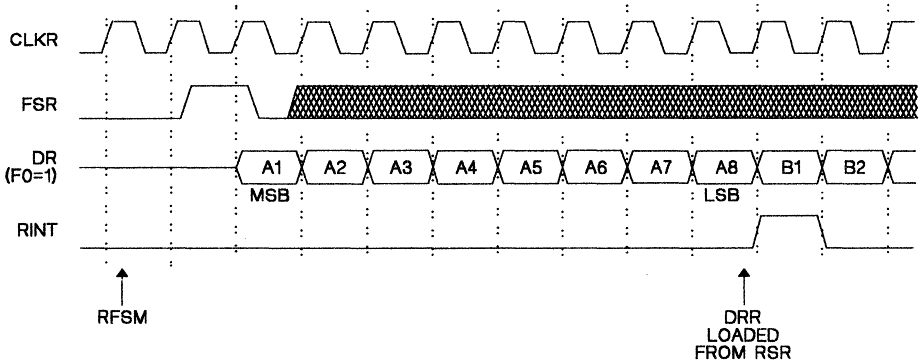


Figure 3-39. Continuous Receive Operation Initialization

3.10 Multiprocessing and Direct Memory Access (DMA)

The flexibility of the TMS320C2x allows configurations to satisfy a wide range of system requirements. Some of the system configurations using the TMS320C2x are as follows:

- A standalone system (single processor),
- A multiprocessor with devices in parallel,
- A host/slave multiprocessor with shared global data memory space, or
- A peripheral processor interfaced using processor-controlled signals to another device.

These system configurations are made possible by three specialized features of the TMS320C2x: the synchronization function utilizing the $\overline{\text{SYNC}}$ input, the global memory interface, and the hold function implemented with the $\overline{\text{HOLD}}$ and $\overline{\text{HOLDA}}$ pins. The following sections describe these functions in detail.

3.10.1 Synchronization

In a multiprocessor environment, the $\overline{\text{SYNC}}$ input can be used to greatly ease interface between processors. This input is used to cause each TMS320C2x in the system to synchronize their internal clocks, thereby allowing the processors to run in lock-step operation.

Multiple TMS320C2x devices are synchronized by using common $\overline{\text{SYNC}}$ and external clock inputs. A negative transition on $\overline{\text{SYNC}}$ sets each processor to internal quarter-phase one (Q1). This transition must occur synchronously with the rising edge of CLKIN. On the TMS320C25, there is a two CLKIN cycle delay following the cycle in which $\overline{\text{SYNC}}$ goes low, before the synchronized Q1 occurs. On the TMS32020, there is no delay.

The timing diagrams for the $\overline{\text{SYNC}}$ input are shown in Figure 3-40 and Figure 3-41 for the TMS32020 and TMS320C25, respectively. Note that the internal clock timing relationships are different in the TMS32020 and TMS320C25 (see Appendix C and Section 3.7.2).

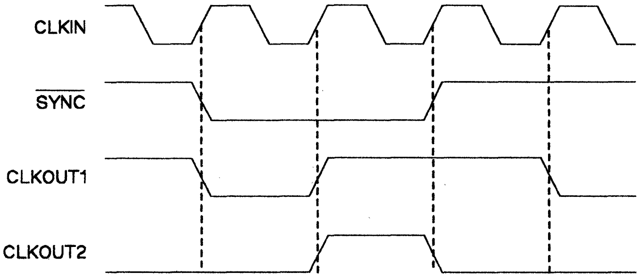


Figure 3-40. Synchronization Timing Diagram (TMS32020)

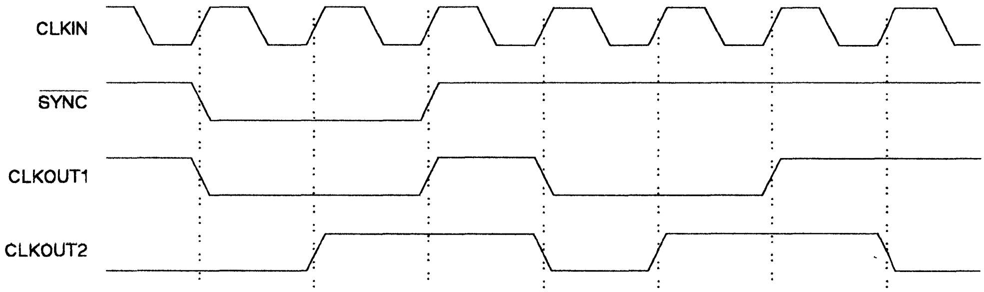


Figure 3-41. Synchronization Timing Diagram (TMS320C25)

Normally, $\overline{\text{SYNC}}$ is applied while $\overline{\text{RS}}$ is active. If $\overline{\text{SYNC}}$ is asserted after a reset, the following can occur:

1) The processor machine cycle is reset to Q1, provided that the timing requirements for $\overline{\text{SYNC}}$ are met. If $\overline{\text{SYNC}}$ is asserted at the beginning of Q1, Q3, or Q4, the current instruction is improperly executed. If $\overline{\text{SYNC}}$ is asserted at the beginning of Q2, the current instruction is executed properly.
2) If $\overline{\text{SYNC}}$ does not meet the timing requirements, unpredictable processor operation occurs. A reset should then be executed to place the processor back in a known state.

3.10.2 Global Memory

For multiprocessing applications, the TMS320C2x has the capability of allocating global data memory space and communicating with that space via the $\overline{\text{BR}}$ (bus request) and READY control signals.

Global memory is memory shared by more than one processor; therefore, access to it must be arbitrated. When using global memory, the processor's address space is divided into local and global sections. The local section is used by the processor to perform its individual function, and the global section is used to communicate with other processors.

A memory-mapped global memory allocation register (GREG) specifies part of the TMS320C2x's data memory as global external memory. GREG, which is memory-mapped at data memory address location 5, is an eight-bit register connected to the eight LSBs of the internal D bus. The upper eight bits of location 5 are nonexistent and read as one's.

The contents of GREG determine the size of the global memory space. The legal values of GREG and corresponding global memory spaces are shown in Table 3-8. Note that values other than those listed in the table lead to fragmented memory maps.

Table 3-8. Global Data Memory Configurations

GREG VALUE	LOCAL MEMORY RANGE	LOCAL MEMORY # WORDS	GLOBAL MEMORY RANGE	GLOBAL MEMORY # WORDS
000000XX	>0 - >FFFF	65,536	-------------	0
10000000	>0 - >7FFF	32,768	>8000 - >FFFF	32,768
11000000	>0 - >BFFF	49,152	>C000 - >FFFF	16,384
11100000	>0 - >DFFF	57,344	>E000 - >FFFF	8,192
11110000	>0 - >EFFF	61,440	>F000 - >FFFF	4,096
11111000	>0 - >F7FF	63,488	>F800 - >FFFF	2,048
11111100	>0 - >FBFF	64,512	>FC00 - >FFFF	1,024
11111110	>0 - >FDFF	65,024	>FE00 - >FFFF	512
11111111	>0 - >FEFF	65,280	>FF00 - >FFFF	256

When a data memory address, either direct or indirect, corresponds to a global data memory address (as defined by GREG), $\overline{BR}$ is asserted low with $\overline{DS}$ to indicate that the processor wishes to make a global memory access. External logic then arbitrates for control of the global memory, asserting READY when the TMS320C2x has control. The length of the memory cycle is controlled by the READY line. One wait-state timing is shown in Figure 3-42. Note that all signals not shown have the same timing as in the normal read or write case.

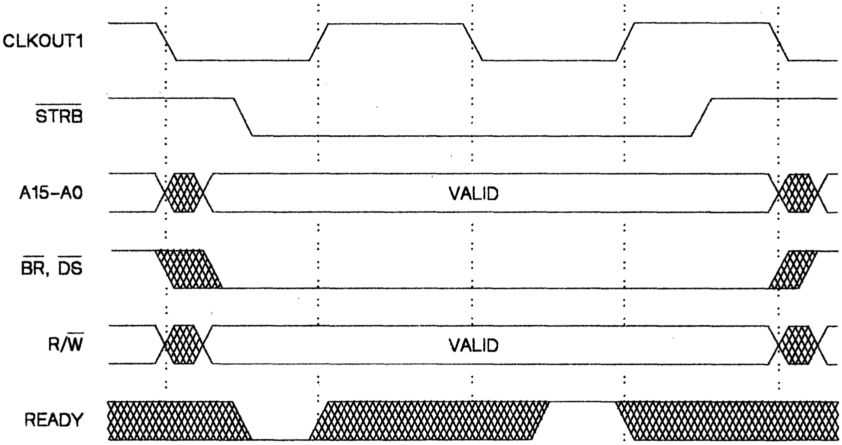


Figure 3-42. Global Memory Access Timing

3.10.3 The Hold Function

The TMS320C2x supports Direct Memory Access (DMA) to its local (off-chip) program, data, and I/O spaces. Two signals, $\overline{\mathrm{HOLD}}$ and $\overline{\mathrm{HOLDA}}$, are provided to allow another device to take control of the processor's buses. Upon receiving a $\overline{\mathrm{HOLD}}$ signal from an external device, the processor acknowledges by bringing $\overline{\mathrm{HOLDA}}$ low. The processor then places its address and data buses as well as all control signals ($\overline{\mathrm{PS}}$, $\overline{\mathrm{DS}}$, $\overline{\mathrm{IS}}$, R/$\overline{\mathrm{W}}$, and $\overline{\mathrm{STRB}}$) in the high-impedance state. The serial port output pins, DX and FSX, are not affected by $\overline{\mathrm{HOLD}}$. Signaling between the external processor and the TMS320C2x can be performed using interrupts.

The timing for the $\overline{\mathrm{HOLD}}$ and $\overline{\mathrm{HOLDA}}$ signals is shown in Figure 3-43. $\overline{\mathrm{HOLD}}$ has the same setup time as READY and is sampled at the beginning of quarter-phase 3 (see Appendix C for phase relationships on the TMS32020). If the setup time is met, it takes three machine cycles before the buses and control signals go to the high-impedance state. Note that unlike the external interrupts $\overline{\mathrm{INT}}$(2-0), $\overline{\mathrm{HOLD}}$ is not a latched input. The external device must keep $\overline{\mathrm{HOLD}}$ low until it receives a $\overline{\mathrm{HOLDA}}$ from the TMS320C2x.

If the TMS320C2x is in the middle of a multicycle instruction, it will finish the instruction before entering the hold state. After the instruction is completed, the buses are placed in the high-impedance state. This also applies to instructions that become multicycle due to insertion of wait states or to the use of RPT/RPTK instructions.

After $\overline{\mathrm{HOLD}}$ is de-asserted, program execution resumes from the same point at which it was halted. $\overline{\mathrm{HOLDA}}$ is removed synchronously with $\overline{\mathrm{HOLD}}$, as shown in Figure 3-43. If the setup time is met, two machine cycles are required before the buses and control signals become valid.

$\overline{\mathrm{HOLD}}$ is not treated as an interrupt. If the TMS320C2x was executing the IDLE instruction before entering the hold state, it resumes executing IDLE once it leaves the hold state.

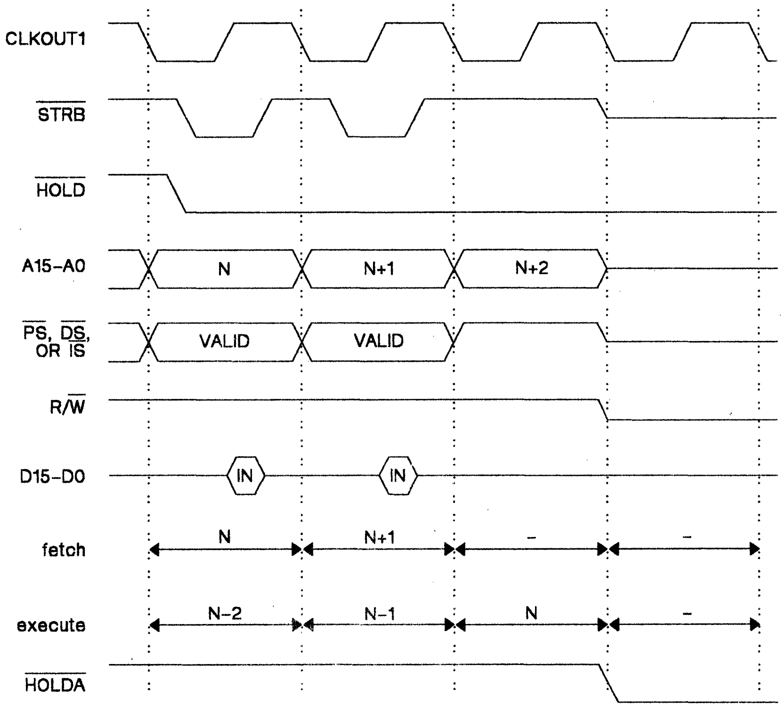
The hold function on the TMS320C25 has two distinct operating modes:

- A TMS32020-like mode, in which execution is suspended during assertion of $\overline{\mathrm{HOLD}}$, and
- A TMS320C25 concurrent DMA mode, in which the TMS320C25 continues to execute its program while operating from internal RAM or ROM, thus greatly increasing throughput in data-intensive applications.

The operating mode is selected by the HM (hold mode) status register bit on the TMS320C25. The $\overline{\mathrm{HOLD}}$ signal is pulled low, as shown in the first part of Figure 3-43. When HM = 1, the TMS320C25 halts program execution and enters the hold state directly. When HM = 0, the processor enters the hold state directly, as shown in Figure 3-43, if program execution is from external memory or if external data memory is being accessed. If program execution is from internal memory, however, and if no external data memory accesses are required, the processor enters the hold state externally, but program execution continues internally. This allows more efficient system operation since a program may continue executing while an external DMA operation is being performed.

Program execution ceases until $\overline{\text{HOLD}}$ is removed if the processor is in a hold state with HM = 0 and an internally executing program requires an external access, or if the program branches to an external address. Also, if a repeat instruction that requires the use of the external bus is executing with HM = 0 and a hold occurs, the hold state is entered after the current bus cycle. If this situation occurs with HM = 1, the hold state will not be entered until the repeat count is completed. HM is set and reset by the SHM (set hold mode) and RHM (reset hold mode) instructions, respectively.

All interrupts are disabled while $\overline{\text{HOLD}}$ is active with HM = 1. If an interrupt is received during this period, the interrupt is latched and remains pending. $\overline{\text{HOLD}}$ itself does not affect any interrupt flags or registers. If HM = 0, interrupts function normally.



NOTES: 1. N is the program memory location for the current instruction.
2. This example only shows the execution of single-cycle instructions fetched from external program memory.

Figure 3-43. TMS320C25 Hold Timing Diagram

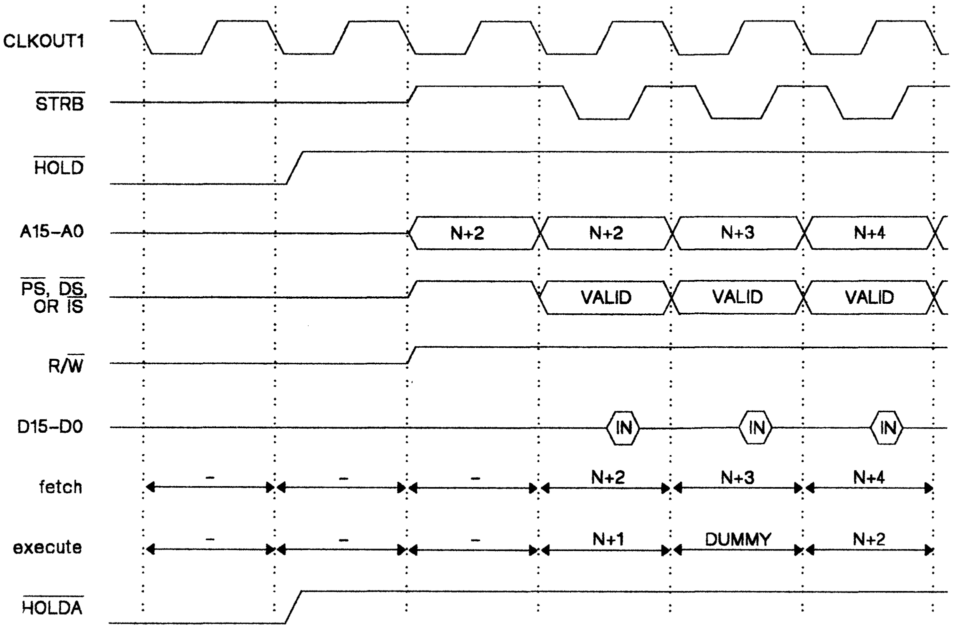


Figure 3-43. TMS320C25 Hold Timing Diagram (Concluded)

4. Assembly Language Instructions

The TMS320C2x instruction set supports numeric-intensive signal processing operations as well as general-purpose applications, such as multiprocessing and high-speed control. TMS320C1x source code is upward-compatible with TMS320C2x source code. TMS32020 object code is upward-compatible with TMS320C25 object code.

This section describes the assembly language instructions for the TMS320C2x microprocessor. Included in this section are the following major topics:

- Memory Addressing Modes (Section 4.1 on page 4-2)
  - Direct addressing
  - Indirect addressing (using eight auxiliary registers)
  - Immediate addressing

- Instruction Set (Section 4.2 on page 4-10)
  - Symbols and abbreviations used in the instructions
  - Instruction set summary (listed according to function)

- Individual Instruction Descriptions (Section 4.3 on page 4-17)
  - Presented in alphabetical order and providing the following:
    - Assembler syntax
    - Operands
    - Execution
    - Encoding
    - Description
    - Words
    - Cycles
    - Repeatability
    - Example(s)

4.1 Memory Addressing Modes

The TMS320C2x instruction set provides three memory addressing modes:

- Direct addressing mode
- Indirect addressing mode
- Immediate addressing mode

Both direct and indirect addressing can be used to access data memory. Direct addressing concatenates seven bits of the instruction word with the nine bits of the data memory page pointer to form the 16-bit data memory address. Indirect addressing accesses data memory through the auxiliary registers. In immediate addressing, the data is based on a portion of the instruction word(s). The following sections describe each addressing mode and give the opcode formats and some examples for each mode.

4.1.1 Direct Addressing Mode

In the direct memory addressing mode, the instruction word contains the lower seven bits of the data memory address (dma). This field is concatenated with the nine bits of the data memory page pointer (DP) register to form the full 16-bit data memory address. Thus, the DP register points to one of 512 possible 128-word data memory pages, and the 7-bit address in the instruction points to the specific location within that data memory page. The DP register is loaded through the LDP (load data memory page pointer), LDPK (load data memory page pointer immediate), or LST (load status register ST0) instructions.

Note:

The data page pointer is not initialized by reset and is therefore undefined after powerup. The TMS320C2x development tools, however, utilize default values for many parameters, including the data page pointer. Because of this, programs that do not explicitly initialize the data page pointer may execute improperly, depending on whether they are executed on a TMS320C2x device or using a development tool. Thus, it is critical that all programs initialize the data page pointer in software.

Figure 4-1 illustrates how the 16-bit data address is formed.

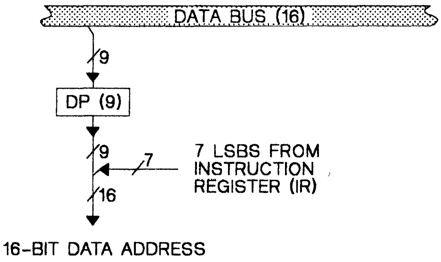


Figure 4-1. Direct Addressing Block Diagram

Direct addressing can be used with all instructions except CALL, the branch instructions, immediate operand instructions, and instructions with no operands. The direct addressing format is as follows:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode								0	dma						

Bits 15 through 8 contain the opcode. Bit 7 = 0 defines the addressing mode as direct, and bits 6 through 0 contain the data memory address (dma).

Example of Direct Addressing Format:

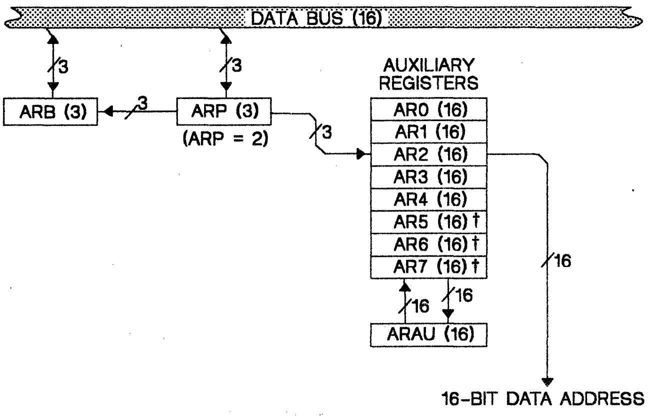
ADD 9,5 Add to accumulator the contents of data memory location 9 left-shifted 5 bits.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1

The opcode of the ADD 9,5 instruction is >05 and appears in bits 15 through 8. The notation >nn indicates nn is a hexadecimal number. The shift count of >5 appears in bits 11 through 8 of the opcode. The data memory address >09 appears in bits 6 through 0.

4.1.2 Indirect Addressing Mode

The auxiliary registers (AR) provide flexible and powerful indirect addressing. Five auxiliary registers (AR0-AR4) are provided on the TMS32020, and eight auxiliary registers (AR0-AR7) are available on the TMS320C25. To select a specific auxiliary register, the Auxiliary Register Pointer (ARP) is loaded with a value from 0 through 4 or 7, designating AR0 through AR4 or AR7, respectively (see Figure 4-2).



†TMS320C25 specific.

Figure 4-2. Indirect Addressing Block Diagram

The contents of the auxiliary registers may be operated upon by the Auxiliary Register Arithmetic Unit (ARAU), which implements 16-bit unsigned arithmetic. The ARAU performs auxiliary register arithmetic operations in the same cycle as the execution of the instruction. (Note that the increment or decrement of the indicated AR is always executed after the use of that AR in the instruction.)

In indirect addressing, any location in the 64K data memory space can be accessed via the 16-bit addresses contained in the auxiliary registers. These may be loaded by the instructions LAR (load auxiliary register), LARK (load auxiliary register immediate), and LRLK (load auxiliary register long immediate). The auxiliary registers on the TMS320C25 may be modified by ADRK (add to auxiliary register short immediate) or SBRK (subtract from auxiliary register short immediate). The TMS320C2x auxiliary registers may also be modified by the MAR (modify auxiliary register) instruction or, equivalently, by the indirect addressing field of any instruction supporting indirect addressing. AR(ARP) denotes the auxiliary register selected by ARP

The following symbols are used in indirect addressing, including bit-reversed (BR) addressing:

*	Contents of AR(ARP) are used as the data memory address.
*-	Contents of AR(ARP) are used as the data memory address, then decremented after the access.
*+	Contents of AR(ARP) are used as the data memory address, then incremented after the access.
*0-	Contents of AR(ARP) are used as the data memory address, and the contents of AR0 subtracted from it after the access.
*0+	Contents of AR(ARP) are used as the data memory address, and the contents of AR0 added to it after the access.
*BR0-	Contents of AR(ARP) are used as the data memory address, and the contents of AR0 subtracted from it, with reverse carry (rc) propagation, after the access (TMS320C25 specific).
*BR0+	Contents of AR(ARP) are used as the data memory address, and the contents of AR0 added to it, with reverse carry (rc) propagation, after the access (TMS320C25 specific).

There are two main types of indirect addressing with indexing:

- Regular indirect addressing with increment or decrement, and
- Indirect addressing with indexing based on the value of AR0:
  - Indexing by adding or subtracting the contents of AR0, or
  - Indexing by adding or subtracting the contents of AR0 with the carry propagation reversed (for FFTs on the TMS320C25).

In either case, the contents of the auxiliary register pointed to by the ARP register are used as the address of the data memory operand. Then, the ARAU performs the specified mathematical operation on the indicated auxiliary register. Additionally, the ARP may be loaded with a new value. All indexing operations are performed on the current auxiliary register in the same cycle as the original instruction.

Indirect auxiliary register addressing allows for post-access adjustments of the auxiliary register pointed to by the ARP. The adjustment may be an increment or decrement by one or based upon the contents of AR0.

Bit-reversed addressing modes on the TMS320C25 allow efficient I/O to be performed for the resequencing of data points in a radix-2 FFT program. The direction of carry propagation in the ARAU is reversed when this mode is selected and AR0 is added to/subtracted from the current auxiliary register. Typical use of this addressing mode requires that AR0 first be set to a value corresponding to one-half of the array size, and AR(ARP) be set to the base address of the data (the first data point). See Section 5.7.4 for an FFT example using bit-reversed addressing modes.

Indirect addressing can be used with all instructions except immediate operand instructions and instructions with no operands. The indirect addressing format is as follows:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode								1	IDV	INC	DEC	NAR	Y		

Bits 15 through 8 contain the opcode, and bit 7 = 1 defines the addressing mode as indirect. Bits 6 through 0 contain the indirect addressing control bits.

Bit 6 contains the increment/decrement value (IDV). The IDV determines whether AR0 will be used to increment or decrement the current auxiliary register. If bit 6 = 0, an increment or decrement (if any) by one occurs to the current auxiliary register. If bit 6 = 1, AR0 may be added to or subtracted from the current auxiliary register as defined by bits 5 and 4.

Bits 5 and 4 control the arithmetic operation to be performed with AR(ARP) and AR0. When set, bit 5 indicates that an increment is to be performed. If bit 4 is set, a decrement is to be performed. Table 4-1 shows the correspondence of bit pattern and arithmetic operation.

Table 4-1. Indirect Addressing Arithmetic Operations

BITS 6	 5	 4	ARITHMETIC OPERATION
0	0	0	No operation on AR(ARP)
0	0	1	AR(ARP) - 1 → AR(ARP)
0	1	0	AR(ARP) + 1 → AR(ARP)
0	1	1	Reserved
1	0	0	AR(ARP) - AR0 → AR(ARP) [reverse carry propagation]†
1	0	1	AR(ARP) - AR0 → AR(ARP)
1	1	0	AR(ARP) + AR0 → AR(ARP)
1	1	1	AR(ARP) + AR0 → AR(ARP) [reverse carry propagation]†

†TMS320C25 specific.

Bit 3 and bits 2 through 0 control the Auxiliary Register Pointer (ARP). Bit 3 (NAR) determines if a new value is loaded into the ARP. If bit 3 = 1, the contents of bits 2 through 0 (Y = next ARP) are loaded into the ARP. If bit 3 = 0, the contents of the ARP remain unchanged.

Table 4-2 shows the bit fields, notation, and operation used for indirect addressing.

Table 4-2. Bit Fields for Indirect Addressing

INSTRUCTION FIELD BITS							NOTATION	OPERATION
15 – 8	7	6	5	4	3	2 1 0		
← Opcode →	1	0	0	0	0	← Y →	*	No manipulation of ARs/ARP
← Opcode →	1	0	0	0	1	← Y →	*,Y	Y → ARP
← Opcode →	1	0	0	1	0	← Y →	*-	AR(ARP)-1 → AR(ARP)
← Opcode →	1	0	0	1	1	← Y →	*-,Y	AR(ARP)-1 → AR(ARP) Y → ARP
← Opcode →	1	0	1	0	0	← Y →	*+	AR(ARP)+1 → AR(ARP)
← Opcode →	1	0	1	0	1	← Y →	*+,Y	AR(ARP)+1 → AR(ARP) Y → ARP
← Opcode →	1	1	0	0	0	← Y →	*BR0-	AR(ARP)-rcAR0 → AR(ARP)†
← Opcode →	1	1	0	0	1	← Y →	*BR0-,Y	AR(ARP)-rcAR0 → AR(ARP) Y → ARP†
← Opcode →	1	1	0	1	0	← Y →	*0-	AR(ARP)-AR0 → AR(ARP)
← Opcode →	1	1	0	1	1	← Y →	*0-,Y	AR(ARP)-AR0 → AR(ARP) Y → ARP
← Opcode →	1	1	1	0	0	← Y →	*0+	AR(ARP)+AR0 → AR(ARP)
← Opcode →	1	1	1	0	1	← Y →	*0+,Y	AR(ARP)+AR0 → AR(ARP) Y → ARP
← Opcode →	1	1	1	1	0	← Y →	*BR0+	AR(ARP)+rcAR0 → AR(ARP)†
← Opcode →	1	1	1	1	1	← Y →	*BR0+,Y	AR(ARP)+rcAR0 → AR(ARP) Y → ARP†

†BR = bit-reversed addressing mode and rc = reverse carry propagation (TMS320C25).

For some instructions, the notation in Table 4-2 includes a shift code, e.g., *0+,8,3 where 8 is the shift code and Y = 3.

The CMPR (compare auxiliary register with AR0), and BBZ/BBNZ (branch if TC bit equal/not equal to zero) instructions facilitate conditional branches based on comparisons between the contents of AR0 and the contents of AR(ARP).

The auxiliary registers may also be used for temporary storage via the load and store auxiliary register instructions, LAR and SAR, respectively.

The following examples illustrate the indirect addressing format:

Example 1:

ADD *+,8 Add to the accumulator the contents of the data memory address defined by the contents of the current auxiliary register. This data is left-shifted 8 bits before being added. The current auxiliary register is autoincremented by one. The opcode is >08A0, as shown below.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0

Example 2:

ADD *,8	As in Example 1, but with no autoincrement; the opcode is >0880.

Example 3:

ADD *-,8	As in Example 1, except that the current auxiliary register is decremented by one; the opcode is >0890.

Example 4:

ADD *0+,8	As in Example 1, except that the contents of auxiliary register AR0 are added to the current auxiliary register; the opcode is >08E0.

Example 5:

ADD *0-,8	As in Example 1, except that the contents of auxiliary register AR0 are subtracted from the current auxiliary register; the opcode is >08D0.

Example 6:

ADD *+,8,3	As in Example 1, except that the auxiliary register pointer (ARP) is loaded with the value 3 for subsequent instructions; the opcode is >08AB.

Example 7:

ADD *BR0-,8	The opcode is >08C0. The contents of auxiliary register AR0 are subtracted from the current auxiliary register with reverse carry propagation (TMS320C25).

Example 8:

ADD *BR0+,8	The opcode is >08F0. The contents of auxiliary register AR0 are added to the current auxiliary register with reverse carry propagation (TMS320C25).

4.1.3 Immediate Addressing Mode

In immediate addressing, the instruction word(s) contains the value of the immediate operand. The TMS320C2x has both single-word (8-bit and 13-bit constant) short immediate instructions and two-word (16-bit constant) long immediate instructions. The immediate operand is contained within the instruction word itself in short immediate instructions. In long immediate instructions, the word following the instruction opcode is used as the immediate operand.

The following short immediate instructions contain the immediate operand in the instruction word and execute within a single instruction cycle. The length of the constant operand is instruction-dependent. Note that the ADDK, ADRK, SBRK, and SUBK instructions are available on the TMS320C25.

ADDK	Add to accumulator short immediate (8-bit absolute constant)
ADRK	Add to auxiliary register short immediate (8-bit absolute constant)
LACK	Load accumulator short immediate (8-bit absolute constant)
LARK	Load auxiliary register short immediate (8-bit absolute constant)
LARP	Load auxiliary register pointer (3-bit constant)
LDPK	Load data memory page pointer immediate (9-bit constant)
MPYK	Multiply immediate (13-bit two's-complement constant)
RPTK	Repeat instruction as specified by immediate value (8-bit constant)
SBRK	Subtract from auxiliary register short immediate (8-bit absolute constant)
SUBK	Subtract from accumulator short immediate (8-bit absolute constant).

Example of short immediate addressing format:

RPTK 99 Execute the instruction following this instruction 100 times.

With the RPTK instruction, the immediate operand is contained as a part of the instruction opcode. The instruction format for RPTK is as follows:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	0	1	1	8-Bit Constant							

For long immediate instructions, the constant is a 16-bit value in the word following the opcode. The 16-bit value can be optionally used as an absolute constant or as a two's-complement value.

ADLK	Add to accumulator long immediate with shift (absolute or two's complement)
ANDK	AND immediate with accumulator with shift
LALK	Load accumulator long immediate with shift (absolute or two's complement)
LRLK	Load auxiliary register long immediate
ORK	OR immediate with accumulator with shift
SBLK	Subtract from accumulator long immediate with shift (absolute or two's complement)
XORK	Exclusive-OR immediate with accumulator with shift.

Example of long immediate addressing format:

ADLK 16384,2	Add to the accumulator the value 16384 with a shift to the left of two, effectively adding 65536 to the contents of the accumulator.

The ADLK instruction uses the word following the instruction opcode as the immediate operand. The instruction format for ADLK is as follows:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	Shift				0	0	0	0	0	0	1	0
16-Bit Constant															

4.2 Instruction Set

The following sections list the symbols and abbreviations used in the instruction set summary and in the instruction descriptions. The complete instruction set summary is organized according to function. A detailed description of each instruction is listed in the instruction set summary.

4.2.1 Symbols and Abbreviations

Table 4-3 lists symbols and abbreviations used in the instruction set summary (Table 4-4) and the individual instruction descriptions.

Table 4-3. Instruction Symbols

SYMBOL	MEANING
A	Port address
ACC	Accumulator
ARB	Auxiliary register pointer buffer
ARn	Auxiliary register n (AR0, AR1 assembler symbols equal to 0 or 1)
ARP	Auxiliary register pointer
B	4-bit field specifying a bit code
BIO	Branch control input
C	Carry bit
CM	2-bit field specifying compare mode
CNF	On-chip RAM configuration control bit
D	Data memory address field
DATn	Label assigned to data memory location n
dma	Data memory address
DP	Data page pointer
FO	Format status bit
FSM	Frame synchronization mode bit
HM	Hold mode bit
I	Addressing mode bit
INTM	Interrupt mode flag bit
K	Immediate operand field
MCS	Microcall stack
>nn	nn = hexadecimal number (others are decimal values)
OV	Overflow mode flag bit
OVM	Overflow mode bit
P	Product register
PA	Port address (PA0-PA15 assembler symbols equal to 0 through 15)
PC	Program counter
PFC	Prefetch counter
PM	2-bit field specifying P register output shift code
pma	Program memory address
PRGn	Label assigned to program memory location n
R	3-bit operand field specifying auxiliary register
RPTC	Repeat counter
S	4-bit left-shift code
STn	Status register n (ST0 or ST1)
SXM	Sign-extension mode bit
T	Temporary register
TC	Test control bit
TOS	Top of stack
TXM	Transmit mode bit
X	3-bit accumulator left-shift field
XF	XF pin status bit
→	Is assigned to
| |	An absolute value
< >	User-defined items
[ ]	Optional items
( )	Contents of
{ }	Alternative items, one of which must be entered
	Blanks or spaces must be entered where shown.

4.2.2 Instruction Set Summary

Table 4-4 shows the instruction set summary for the TMS320C25 processor, which is a superset of the TMS320C1x and TMS320C2x instruction sets. Included in the instruction set are four special groups of instructions to improve overall processor throughput and ease of use.

- Extended-precision arithmetic (ADDC, SUBB, MPYU, BC, BNC, SC, and RC)
- Adaptive filtering (MPYA, MPYS, and ZALR)
- Control and I/O (RHM, SHM, RTC, STC, RFSM, and SFSM)
- Accumulator and register (SPH, SPL, ADDK, SUBK, ADRK, SBRK, ROL, and ROR).

The instruction set summary is arranged according to function and alphabetized within each functional grouping. Additional information is presented in the individual instruction descriptions in the following section. The symbol † indicates instructions that are specific to the TMS320C2x instruction set. The symbol ‡ indicates instructions that are specific to the TMS320C25 instruction set.

Table 4-4. Instruction Set Summary

ACCUMULATOR MEMORY REFERENCE INSTRUCTIONS			
Mnemonic and Description		Words	16-Bit Opcode MSB LSB
ABS	Absolute value of accumulator	1	1100 1110 0001 1011
ADD	Add to accumulator with shift	1	0000 SSSS I DDD DDDD
ADDC‡	Add to accumulator with carry	1	0100 0011 I DDD DDDD
ADDH	Add to high accumulator	1	0100 1000 I DDD DDDD
ADDK‡	Add to accumulator short immediate	1	1100 1100 KKKK KKKK
ADDS	Add to low accumulator with sign-extension suppressed	1	0100 1001 I DDD DDDD
ADDT†	Add to accumulator with shift specified by T register	1	0100 1010 I DDD DDDD
ADLK†	Add to accumulator long immediate with shift	2	1101 SSSS 0000 0010
AND	AND with accumulator	1	0100 1110 I DDD DDDD
ANDK†	AND immediate with accumulator with shift	2	1101 SSSS 0000 0100
CMPL†	Complement accumulator	1	1100 1110 0010 0111
LAC	Load accumulator with shift	1	0010 SSSS I DDD DDDD
LACK	Load accumulator short immediate	1	1100 1010 KKKK KKKK
LACT†	Load accumulator with shift specified by T register	1	0100 0010 I DDD DDDD
LALK†	Load accumulator long immediate with shift	2	1101 SSSS 0000 0001
NEG†	Negate accumulator	1	1100 1110 0010 0011
NORM†	Normalize contents of accumulator	1	1100 1110 1010 0010
OR	OR with accumulator	1	0100 1101 I DDD DDDD
ORK†	OR immediate with accumulator with shift	2	1101 SSSS 0000 0101
ROL‡	Rotate accumulator left	1	1100 1110 0011 0100
ROR‡	Rotate accumulator right	1	1100 1110 0011 0101
SACH	Store high accumulator with shift	1	0110 1XXX I DDD DDDD
SACL	Store low accumulator with shift	1	0110 0XXX I DDD DDDD
SBLK†	Subtract from accumulator long immediate with shift	2	1101 SSSS 0000 0011
SFL†	Shift accumulator left	1	1100 1110 0001 1000
SFR†	Shift accumulator right	1	1100 1110 0001 1001
SUB	Subtract from accumulator with shift	1	0001 SSSS I DDD DDDD
SUBB‡	Ssubtract from accumulator with borrow	1	0100 1111 I DDD DDDD
SUBC	Conditional subtract	1	0100 0111 I DDD DDDD
SUBH	Subtract from high accumulator	1	0100 0100 I DDD DDDD
SUBK‡	Ssubtract from accumulator short immediate	1	1100 1101 KKKK KKKK
SUBS	Subtract from low accumulator with sign extension suppressed	1	0100 0101 I DDD DDDD
SUBT†	Subtract from accumulator with shift specified by T register	1	0100 0110 I DDD DDDD
XOR	Exclusive-OR with accumulator	1	0100 1100 I DDD DDDD
XORK†	Exclusive-OR immediate with accumulator with shift	2	1101 SSSS 0000 0110
ZAC	Zero accumulator	1	1100 1010 0000 0000
ZALH	Zero low accumulator and load high accumulator	1	0100 0000 I DDD DDDD
ZALR‡	Zero low accumulator and load high accumulator with rounding	1	0111 1011 I DDD DDDD
ZALS	Zero accumulator and load low accumulator with sign extension suppressed	1	0100 0001 I DDD DDDD

†These instructions are specific to the TMS320C2x instruction set.
‡These instructions are specific to the TMS320C25 instruction set.

Table 4-4. Instruction Set Summary (Continued)

AUXILIARY REGISTERS AND DATA PAGE POINTER INSTRUCTIONS			
Mnemonic and Description		**Words**	**16-Bit Opcode** **MSB LSB**
ADRK‡	Add to auxiliary register short immediate	1	0111 1110 KKKK KKKK
CMPR†	Compare auxiliary register with auxiliary register AR0	1	1100 1110 0101 00KK
LAR	Load auxiliary register	1	0011 0RRR I DDD DDDD
LARK	Load auxiliary register short immediate	1	1100 0RRR KKKK KKKK
LARP	Load auxiliary register pointer	1	0101 0101 1000 1RRR
LDP	Load data memory page pointer	1	0101 0010 I DDD DDDD
LDPK	Load data memory page pointer immediate	1	1100 100K KKKK KKKK
LRLK†	Load auxiliary register long immediate	2	1101 0RRR 0000 0000
MAR	Modify auxiliary register	1	0101 0101 I DDD DDDD
SAR	Store auxiliary register	1	0111 0RRR I DDD DDDD
SBRK‡	Subtract from auxiliary register short immediate	1	0111 1111 KKKK KKKK

T REGISTER, P REGISTER, AND MULTIPLY INSTRUCTIONS			
Mnemonic and Description		**Words**	**16-Bit Opcode** **MSB LSB**
APAC	Add P register to accumulator	1	1100 1110 0001 0101
LPH†	Load high P register	1	0101 0011 I DDD DDDD
LT	Load T register	1	0011 1100 I DDD DDDD
LTA	Load T register and accumulate previous product	1	0011 1101 I DDD DDDD
LTD	Load T register, accumulate previous product, and move data	1	0011 1111 I DDD DDDD
LTP†	Load T register and store P register in accumulator	1	0011 1110 I DDD DDDD
LTS†	Load T register and subtract previous product	1	0101 1011 I DDD DDDD
MAC†	Multiply and accumulate	2	0101 1101 I DDD DDDD
MACD†	Multiply and accumulate with data move	2	0101 1100 I DDD DDDD
MPY	Multiply (with T register, store product in P register)	1	0011 1000 I DDD DDDD
MPYA‡	Multiply and accumulate previous product	1	0011 1010 I DDD DDDD
MPYK	Multiply immediate	1	101K KKKK KKKK KKKK
MPYS‡	Multiply and subtract previous product	1	0011 1011 I DDD DDDD
MPYU‡	Multiply unsigned	1	1100 1111 I DDD DDDD
PAC	Load accumulator with P register	1	1100 1110 0001 0100
SPAC	Subtract P register from accumulator	1	1100 1110 0001 0110
SPH‡	Store high P register	1	0111 1101 I DDD DDDD
SPL‡	Store low P register	1	0111 1100 I DDD DDDD
SPM†	Set P register output shift mode	1	1100 1110 0000 10KK
SQRA†	Square and accumulate	1	0011 1001 I DDD DDDD
SQRS†	Square and subtract previous product	1	0101 1010 I DDD DDDD

†These instructions are specific to the TMS320C2x instruction set.
‡These instructions are specific to the TMS320C25 instruction set.

Table 4-4. Instruction Set Summary (Continued)

BRANCH/CALL INSTRUCTIONS			
Mnemonic and Description		**Words**	**16-Bit Opcode** **MSB LSB**
B	Branch unconditionally	2	1111 1111 1DDD DDDD
BACC†	Branch to address specified by accumulator	1	1100 1110 0010 0101
BANZ	Branch on auxiliary register not zero	2	1111 1011 1DDD DDDD
BBNZ†	Branch if TC bit ≠ 0	2	1111 1001 1DDD DDDD
BBZ†	Branch if TC bit = 0	2	1111 1000 1DDD DDDD
BC‡	Branch on carry	2	0101 1110 1DDD DDDD
BGEZ	Branch if accumulator ≥ 0	2	1111 0100 1DDD DDDD
BGZ	Branch if accumulator > 0	2	1111 0001 1DDD DDDD
BIOZ	Branch on I/O status = 0	2	1111 1010 1DDD DDDD
BLEZ	Branch if accumulator ≤ 0	2	1111 0010 1DDD DDDD
BLZ	Branch if accumulator < 0	2	1111 0011 1DDD DDDD
BNC‡	Branch on no carry	2	0101 1111 1DDD DDDD
BNV†	Branch if no overflow	2	1111 0111 1DDD DDDD
BNZ	Branch if accumulator ≠ 0	2	1111 0101 1DDD DDDD
BV	Branch on overflow	2	1111 0000 1DDD DDDD
BZ	Branch if accumulator = 0	2	1111 0110 1DDD DDDD
CALA	Call subroutine indirect	1	1100 1110 0010 0100
CALL	Call subroutine	2	1111 1110 1DDD DDDD
RET	Return from subroutine	1	1100 1110 0010 0110
TRAP†	Software interrupt	1	1100 1110 0001 1110
I/O AND DATA MEMORY OPERATIONS			
Mnemonic and Description		**Words**	**16-Bit Opcode** **MSB LSB**
BLKD†	Block move from data memory to data memory	2	1111 1101 I DDD DDDD
BLKP†	Block move from program memory to data memory	2	1111 1100 I DDD DDDD
DMOV	Data move in data memory	1	0101 0110 I DDD DDDD
FORT†	Format serial port registers	1	1100 1110 0000 111K
IN	Input data from port	1	1000 AAAA I DDD DDDD
OUT	Output data to port	1	1110 AAAA I DDD DDDD
RFSM‡	Reset serial port frame synchronization mode	1	1100 1110 0011 0110
RTXM†	Reset serial port transmit mode	1	1100 1110 0010 0000
RXF†	Reset external flag	1	1100 1110 0000 1100
SFSM‡	Set serial port frame synchronization mode	1	1100 1110 0011 0111
STXM†	Set serial port transmit mode	1	1100 1110 0010 0001
SXF†	Set external flag	1	1100 1110 0000 1101
TBLR	Table read	1	0101 1000 I DDD DDDD
TBLW	Table write	1	0101 1001 I DDD DDDD

†These instructions are specific to the TMS320C2x instruction set.
‡These instructions are specific to the TMS320C25 instruction set.

Table 4-4. Instruction Set Summary (Concluded)

CONTROL INSTRUCTIONS			
Mnemonic and Description		Words	16-Bit Opcode MSB LSB
BIT†	Test bit	1	1001 BBBB I DDD DDDD
BITT†	Test bit specified by T register	1	0101 0111 I DDD DDDD
CNFD†	Configure block as data memory	1	1100 1110 0000 0100
CNFP†	Configure block as program memory	1	1100 1110 0000 0101
DINT	Disable interrupt	1	1100 1110 0000 0001
EINT	Enable interrupt	1	1100 1110 0000 0000
IDLE†	Idle until interrupt	1	1100 1110 0001 1111
LST	Load status register ST0	1	0101 0000 I DDD DDDD
LST1†	Load status register ST1	1	0101 0001 I DDD DDDD
NOP	No operation	1	0101 0101 0000 0000
POP	Pop top of stack to low accumulator	1	1100 1110 0001 1101
POPD†	Pop top of stack to data memory	1	0111 1010 I DDD DDDD
PSHD†	Push data memory value onto stack	1	0101 0100 I DDD DDDD
PUSH	Push low accumulator onto stack	1	1100 1110 0001 1100
RC‡	Reset carry bit	1	1100 1110 0011 0000
RHM‡	Reset hold mode	1	1100 1110 0011 1000
ROVM	Reset overflow mode	1	1100 1110 0000 0010
RPT†	Repeat instruction as specified by data memory value	1	0100 1011 I DDD DDDD
RPTK†	Repeat instruction as specified by immediate value	1	1100 1011 KKKK KKKK
RSXM†	Reset sign-extension mode	1	1100 1110 0000 0110
RTC‡	Reset test/control flag	1	1100 1110 0011 0010
SC‡	Set carry bit	1	1100 1110 0011 0001
SHM‡	Set hold mode	1	1100 1110 0011 1001
SOVM	Set overflow mode	1	1100 1110 0000 0011
SST	Store status register ST0	1	0111 1000 I DDD DDDD
SST1†	Store status register ST1	1	0111 1001 I DDD DDDD
SSXM†	Set sign-extension mode	1	1100 1110 0000 0111
STC‡	Set test/control flag	1	1100 1110 0011 0011

†These instructions are specific to the TMS320C2x instruction set.
‡These instructions are specific to the TMS320C25 instruction set.

4.3 Individual Instruction Descriptions

Each instruction in the instruction set summary is described in the following pages. Instructions are listed in alphabetical order. Information, such as assembler syntax, operands, operation, encoding, description, words, cycles, and examples, is provided for each instruction. An example instruction is provided to familiarize the user with the special format used and explain its content. Refer to Section 4.1 for further information on memory addressing. Code examples using many of the instructions are given in Section 5 on Software Applications.

Syntax

Direct: [<label>] EXAMPLE <dma>[,<shift>]
Indirect: [<label>] EXAMPLE {ind}[,<shift>[,<next ARP>]]
Immediate: [<label>] EXAMPLE [<constant>]

Each instruction begins with an assembler syntax expression. The optional comment field that concludes the syntax is not included in the syntax expression. Space(s) are required between each field (label, command, operand, and comment fields) as shown in the syntax. The syntax example illustrates both direct and indirect addressing, as well as immediate addressing in which the operand field includes <constant>.

The indirect addressing operand options, including bit-reversed (BR) addressing, are as follows:

TMS32020: {*|*+|*-|*0+|*0-}
TMS320C25: {*|*+|*-|*0+|*0-|*BR0+|*BR0-}

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$
$0 \leq \text{constant} \leq 255$

Operands may be constants or assembly-time expressions referring to memory, I/O and register addresses, pointers, shift counts, and a variety of constants. The operand values used in the example syntax are shown. Note that the next ARP on the TMS32020 is ≤ 4 for auxiliary registers AR0-AR4.

Execution

(PC) + 1 → PC
(ACC) + [(dma) x 2^{shift}] → ACC

If SXM = 1:
Then (dma) is sign-extended.
If SXM = 0:
Then (dma) is not sign-extended.

Affects OV; affected by OVM and SXM.
Affects C (TMS320C25).

An example of the instruction operation sequence is provided, describing the processing that takes place when the instruction is executed. Conditional effects of status register specified modes are also given. Those bits in the TMS320C2x status registers affected by the instruction are also listed.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	0	0	0	Shift				0	Data Memory Address						
Indirect:	0	0	0	0	Shift				1	See Section 4.1						
Immediate:	1	0	0	13-Bit Constant												

Opcode examples are shown of both direct and indirect addressing or of the use of an immediate operand.

Description Instruction execution and its effect on the rest of the processor or memory contents are described. Any constraints on the operands imposed by the processor or the assembler are discussed. The description parallels and supplements the information given by the execution block.

Words 1

The digit specifies the number of memory words required to store the instruction and its extension words.

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

The table shows the number of cycles required for a given TMS320C2x instruction to execute in a given memory configuration when executed as a single instruction or in the repeat mode. The column headings in the tables indicate the program source location (PI, PE, or PR) and data destination or source (DI or DE), defined as follows:

PI	The instruction executes from internal program memory (RAM).
PR	The instruction executes from internal program memory (ROM).
PE	The instruction executes from external program memory.
DI	The instruction executes using internal data memory.
DE	The instruction executes using external data memory.

The number of cycles required for each instruction is given in terms of the program/data memory and I/O access times as defined in the following listing:

p Program memory wait states. Represents the number of clock cycles the device waits for external program memory to respond to an access. T_{ac} is the access time, in nanoseconds, (maximum) required by the TMS320C2x for an external memory access to be made with no wait states. T_{mem} is the memory device access time, and T_p is the clock period (4/crystal frequency).

$p = 0;$ If $T_{mem} \leq T_{ac}$
$p = 1;$ If $T_{ac} < T_{mem} \leq (T_p + T_{ac})$
$p = 2;$ If $(T_p + T_{ac}) < T_{mem} \leq (T_p \times 2 + T_{ac})$
$p = k;$ If $[T_p \times (k-1) + T_{ac}] < T_{mem} \leq (T_p \times k + T_{ac})$

d Data memory wait states. Represents the number of cycles the device must wait for external data memory to respond to an access. This number is calculated in the same way as the p number.

i I/O memory wait states. Represents the number of cycles the device must wait for external I/O memory to respond to an access. This number is calculated in the same way as the p number.

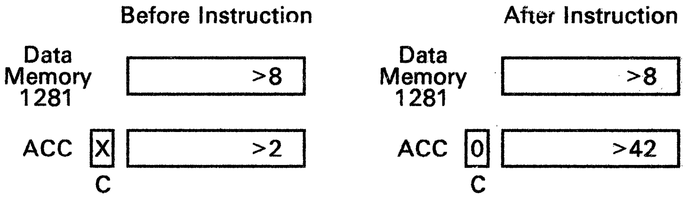
Other abbreviations used in the tables and their meanings are as follows:

br	Branch from ...
int	Internal program memory.
INT	Interrupt.
ext	External program memory.
n	The number of times an instruction is executed when using the RPT or RPTK instruction.

Refer to Appendix D for further information on instruction cycle classifications and timings

Example

```
ADD    DAT1,3   (DP = 10)
or
ADD    *,3      If current auxiliary register contains 1281.
```



The sample code presented in the above format shows the effect of the code on memory and/or registers. The use of the carry bit (C) provided on the TMS320C25 is shown in the small box.

ABS Absolute Value of Accumulator ABS

Syntax [<label>] ABS

Operands None

Execution (PC) + 1 → PC
|(ACC)| → ACC

Affects OV; affected by OVM.
Affects C (TMS320C25).
Not affected by SXM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	0	1	1	0	1	1

Description If the contents of the accumulator are greater than or equal to zero, the accumulator is unchanged by the execution of ABS. If the contents of the accumulator are less than zero, the accumulator is replaced by its two's-complement value.

Note that >80000000 is a special case. When the overflow mode is not set, the ABS of >80000000 is >80000000. When in the overflow mode, the ABS of >80000000 is >7FFFFFFF. In either case, the OV status bit is set. The carry bit (C) on the TMS320C25 is always reset to zero by the execution of this instruction.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example `ABS`

	C	Before Instruction		C	After Instruction
ACC	X	>1234	ACC	0	>1234
ACC	X	>FFFFFFFF	ACC	0	>1

ADD Add to Accumulator with Shift ADD

Syntax

Direct: [<label>] ADD <dma>[,<shift>]
Indirect: [<label>] ADD {ind}[,<shift>[,<next ARP>]]

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$
$0 \leq \text{shift} \leq 15$ (defaults to 0)

Execution

$(\text{PC}) + 1 \rightarrow \text{PC}$
$(\text{ACC}) + [(\text{dma}) \times 2^{\text{shift}}] \rightarrow \text{ACC}$

If SXM = 1:
 Then (dma) is sign-extended.
If SXM = 0:
 Then (dma) is not sign-extended.

Affects OV; affected by OVM and SXM.
Affects C (TMS320C25).

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	0	0	0	Shift				0	Data Memory Address						
Indirect:	0	0	0	0	Shift				1	See Section 4.1						

Description

The contents of the addressed data memory location are left-shifted and added to the accumulator. During shifting, low-order bits are zero-filled. High-order bits are sign-extended if SXM = 1 and zero-filled if SXM = 0. The result is stored in the accumulator.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
ADD    DAT1,3   (DP = 10)
```
or
```
ADD    *,3      If current auxiliary register contains 1281.
```

	Before Instruction			After Instruction
Data Memory 1281		>8	Data Memory 1281		>8
ACC	X	>2	ACC	0	>42
	C			C	

ADDC Add to Accumulator with Carry (TMS320C25) ADDC

Syntax

Direct: [<label>] ADDC <dma>
Indirect: [<label>] ADDC {ind}[,<next ARP>]

Operands

$0 \le \text{dma} \le 127$
$0 \le \text{next ARP} \le 7$

Execution

(PC) + 1 → PC
(ACC) + (dma) + (C) → ACC

Affects OV and C; affected by OVM.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	0	0	0	1	1	0	Data Memory Address						
Indirect:	0	1	0	0	0	0	1	1	1	See Section 4.1						

Description The contents of the addressed data memory location and the value of the carry bit are added to the accumulator. The carry bit is then affected in the normal manner.

The ADDC instruction can be used in performing multiple-precision arithmetic.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example 1

```
ADDC   DAT5     (DP = 8)
or
ADDC   *        If current auxiliary register contains 1029.
```

	Before Instruction	After Instruction
Data Memory 1029	>4	>4
ACC	>13	>18
C	1	0

Example 2

```
ADDC   DAT5      (DP = 8)
or
ADDC   *         If current auxiliary register contains 1029.
```

		Before Instruction			After Instruction
Data Memory 1029		>0	Data Memory 1029		>0
ACC	1	>FFFFFFFF	ACC	1	>0
	C			C	

Syntax

Direct: [<label>] ADDH <dma>
Indirect: [<label>] ADDH {ind}[,<next ARP>]

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution

$(PC) + 1 \rightarrow PC$
$(ACC) + [(dma) \times 2^{16}] \rightarrow ACC$

Affects OV; affected by OVM.
Affects C (TMS320C25).
Low-order bits of the ACC not affected.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	0	1	0	0	0	0	Data Memory Address						
Indirect:	0	1	0	0	1	0	0	0	1	See Section 4.1						

Description

The contents of the addressed data memory location are added to the upper half of the accumulator (bits 31 through 16). Low-order bits are unaffected by ADDH. The carry bit (C) on the TMS320C25 is set if the result of the addition generates a carry; otherwise, C is unaffected. The carry bit can only be set, not reset, by the ADDH instruction.

The ADDH instruction may be used in performing 32-bit arithmetic.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
ADDH   DAT5    (DP = 8)
```
or
```
ADDH   *       If current auxiliary register contains 1029.
```

	Before Instruction		After Instruction
Data Memory 1029	>4	Data Memory 1029	>4
ACC	1 >13 C	ACC	1 >40013 C

ADDK Add to Accumulator Short Immediate (TMS320C25) ADDK

Syntax [<label>] ADDK <constant>

Operands 0 ≤ constant ≤ 255

Execution (PC) + 1 → PC
(ACC) + 8-bit positive constant → ACC

Affects OVM and C; affected by OVM.
Not affected by SXM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	0	0	8-Bit Constant							

Description The 8-bit immediate value is added, right-justified, to the accumulator with the result replacing the accumulator contents. The immediate value is treated as an 8-bit positive number, regardless of the value of SXM.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'C25	not repeatable					

Example `ADDK >5`

	Before Instruction		After Instruction
ACC	X >79B2E1	ACC	0 >79B2E6
	C		C

ADDS Add to Accumulator with Sign-Extension Suppressed ADDS

Syntax

Direct: [<label>] ADDS <dma>
Indirect: [<label>] ADDS {ind}[,<next ARP>]

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution

(PC) + 1 → PC
(ACC) + (dma) → ACC
(dma) is a 16-bit unsigned number.

Affects OV; affected by OVM.
Affects C (TMS320C25).
Not affected by SXM.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	0	1	0	0	1	0	Data Memory Address						
Indirect:	0	1	0	0	1	0	0	1	1	See Section 4.1						

Description

The contents of the specified data memory location are added with sign-extension suppressed. The data is treated as a 16-bit unsigned number, regardless of SXM. The accumulator behaves as a signed number. Note that ADDS produces the same results as an ADD instruction with SXM = 0 and a shift count of 0.

Words

1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
ADDS    DAT11   (DP = 6)
or
ADDS    *       If current auxiliary register contains 779.
```

	Before Instruction		After Instruction
Data Memory 779	>F006	Data Memory 779	>F006
ACC	X >3 C	ACC	0 >F009 C

ADDT Add to Accumulator with Shift Specified by T Register ADDT

Syntax

Direct: [<label>] ADDT <dma>
Indirect: [<label>] ADDT {ind}[,<next ARP>]

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution

(PC) + 1 → PC
$(\text{ACC}) + [(\text{dma}) \times 2^{\text{T register}(3\text{-}0)}] \rightarrow (\text{ACC})$

If SXM = 1:
Then (dma) is sign-extended.
If SXM = 0:
Then (dma) is not sign-extended.

Affects OV; affected by SXM and OVM.
Affects C (TMS320C25).

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	0	1	0	1	0	0	Data Memory Address						
Indirect:	0	1	0	0	1	0	1	0	1	See Section 4.1						

Description

The data memory value is left-shifted and added to the accumulator, with the result replacing the accumulator contents. The left-shift is defined by the four LSBs of the T register, resulting in shift options from 0 to 15 bits. Sign extension on the data memory value is controlled by SXM.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
ADDT  DAT127    (DP = 4)
or
ADDT  *         If current auxiliary register contains 639.
```

	Before Instruction	After Instruction
Data Memory 639	>9	>9
T	>FF94	>FF94
ACC	>F715	>F7A5
C	X	0

ADLK Add to Accumulator Long Immediate with Shift ADLK

Syntax [<label>] ADLK <constant>[,<shift>]

Operands 16-bit constant
0 ≤ shift ≤ 15 (defaults to 0)

Execution (PC) + 2 → PC
$(\mathrm{ACC}) + [\mathrm{constant} \times 2^{\mathrm{shift}}] \rightarrow \mathrm{ACC}$

If SXM = 1:
 Then -32768 ≤ constant ≤ 32767.
If SXM = 0:
 Then 0 ≤ constant ≤ 65535.

Affects OV; affected by OVM and SXM.
Affects C (TMS320C25).

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	Shift				0	0	0	0	0	0	1	0
16-bit Constant															

Description The 16-bit immediate value, left-shifted as specified, is added to the accumulator. The result replaces the accumulator contents. SXM determines whether the constant is treated as a signed two's-complement number or as an unsigned number. The shift count is optional and defaults to zero.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	2	2	2+2p	2+2p	–	–
'C25	2	2	2+2p	2+2p	2	2
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example

```
ADLK   5,8
```

	Before Instruction			After Instruction
ACC	X	>10EF	ACC	0	>15EF
	C			C	

ADRK Add to Auxiliary Register Short Immediate (TMS320C25) ADRK

Syntax [<label>] ADRK <constant>

Operands 0 ≤ constant ≤ 255

Execution (PC) + 1 → PC
AR(ARP) + 8-bit positive constant → AR(ARP)

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	0	8-Bit Constant							

Description The 8-bit immediate value is added, right-justified, to the currently selected auxiliary register with the result replacing the auxiliary register contents. The addition takes place in the ARAU, with the immediate value treated as an 8-bit positive integer.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'C25	not repeatable					

Example

```
ADRK   >80     (ARP = 5)
```

	Before Instruction		After Instruction
AR5	>4321	AR5	>43A1

Syntax

Direct: [<label>] AND <dma>
Indirect: [<label>] AND {ind}[,<next ARP>]

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution

(PC) + 1 → PC
(ACC(15-0)).AND.(dma) → ACC(15-0)
0 → ACC(31-16)

Not affected by SXM.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	0	1	1	1	0	0	Data Memory Address						
Indirect:	0	1	0	0	1	1	1	0	1	See Section 4.1						

Description

The lower half of the accumulator is ANDed with the contents of the addressed data memory location. The upper half of the accumulator is ANDed with all zeroes. Therefore, the upper half of the accumulator is always zeroed by the AND instruction.

Words

1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
AND    DAT16    (DP = 4)
or
AND    *        If current auxiliary register contains 528.
```

		Before Instruction			After Instruction
Data Memory 528		>FF	Data Memory 528		>FF
ACC	X	>12345678	ACC	X	>00000078
	C			C	

ANDK AND Immediate with Accumulator with Shift ANDK

Syntax [<label>] ANDK <constant>[,<shift>]

Operands 16-bit constant
0 ≤ shift ≤ 15 (defaults to 0)

Execution (PC) + 2 → PC
(ACC(30-0)).AND.[(constant x 2^{shift})] → ACC(30-0)
0 → ACC(31) and all other bit positions unoccupied by shifted constant.

Not affected by SXM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	Shift				0	0	0	0	0	1	0	0
16-bit Constant															

Description The 16-bit immediate constant is left-shifted as specified and ANDed with the accumulator. The result is left in the accumulator. Low-order bits below and high-order bits above the shifted value are treated as zeroes, clearing the corresponding bits in the accumulator. Note that the accumulator's most-significant bit is always zeroed regardless of the shift-code value.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	2	2	2+2p	2+2p	–	–
'C25	2	2	2+2p	2+2p	2	2
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example `ANDK >FFFF,12`

	Before Instruction		After Instruction
ACC	X (C) >12345678	ACC	X (C) >02345000

APAC Add P Register to Accumulator APAC

Syntax [<label>] APAC

Operands None

Execution (PC) + 1 → PC
(ACC) + (shifted P register) → ACC

Affects OV; affected by PM and OVM.
Affects C (TMS320C25).
Not affected by SXM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	0	1	0	1	0	1

Description The contents of the P register are shifted as defined by the PM status bits and added to the contents of the accumulator. The result is left in the accumulator. APAC is not affected by the SXM bit of the status register; the P register is always sign-extended.

The APAC instruction is a subset of the LTA, LTD, MAC, MACD, MPYA, and SQRA instructions.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example `APAC (PM = 0)`

		Before Instruction			After Instruction
P		>40	P		>40
ACC	X	>20	ACC	0	>60
	C			C	

B Branch Unconditionally B

Syntax [<label>] B <pma>[,{ind}[,<next ARP>]]

Operands 0 ≤ pma ≤ 65535
0 ≤ next ARP ≤ 7

Execution pma → PC
Modify AR(ARP) and ARP as specified.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1	1	See Section 4.1						
Program Memory Address															

Description The current auxiliary register and ARP are modified as specified, and control passes to the designated program memory address (pma). Note that no AR or ARP modification occurs if nothing is specified in those fields. Pma can be either a symbolic or a numeric address.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	2 (br int-to-int)		2+p (int-to-ext)		–	–
	2+p (ext-to-int)		2+2p (ext-to-ext)		–	–
'C25	True Conditions:					
	Destination on-chip RAM:					
	2	2	2+2p	2+2p	2	2
	Destination on-chip ROM:					
	3	3	3+2p	3+2p	3	3
	Destination external memory:					
	3+p	3+p	3+3p	3+3p	3+p	3+p
	False Condition:					
	Destination anywhere:					
	2	2	2+2p	2+2p	2	2
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example

```
B    PRG191      191 is loaded into the program counter, and
                 the program continues running from that
                 location.
```

BACC Branch to Address Specified by Accumulator BACC

Syntax [<label>] BACC

Operands None

Execution (ACC(15-0)) → PC

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	1	0	0	1	0	1

Description The branch uses the lower half of the accumulator (bits 15-0) for the branch address.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	2	2	2+p	2+p	–	–
'C25	Destination on-chip RAM:					
	2	2	2+p	2+p	2	2
	Destination on-chip ROM:					
	3	3	3+p	3+p	3	3
	Destination external memory:					
	3+p	3+p	3+2p	3+2p	3+p	3+p
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example `BACC`

	C	Before Instruction	C	After Instruction
PC		>16E4		>9545
ACC	X	>F7FF9545	X	>F7FF9545

BANZ Branch on Auxiliary Register Not Zero BANZ

Syntax [<label>] BANZ <pma>[,{ind}[,<next ARP>]]

Operands $0 \leq \text{pma} \leq 65535$
$0 \leq \text{next ARP} \leq 7$

Execution If AR(ARP) ≠ 0:
Then pma → PC;
Else (PC) + 2 → PC.
Modify AR(ARP) as specified.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	1	1	1	See Section 4.1						
Program Memory Address															

Description Control is passed to the designated program memory address (pma) if the current auxiliary register is not equal to zero. Otherwise, control passes to the next instruction. The current auxiliary register and ARP are also modified as specified.

The current auxiliary register is either incremented or decremented from zero when the branch is not taken. Note that the AR modification defaults to *- (decrement current AR by one) when nothing is specified, making it compatible with the TMS320C1x. Pma can be either a symbolic or a numeric address.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	2 (br int-to-int) 2+p (ext-to-int)		2+p (int-to-ext) 2+2p (ext-to-ext)		– –	– –
'C25	True Conditions:					
	Destination on-chip RAM:					
	2	2	2+2p	2+2p	2	2
	Destination on-chip ROM:					
	3	3	3+2p	3+2p	3	3
	Destination external memory:					
	3+p	3+p	3+3p	3+3p	3+p	3+p
	False Condition:					
	Destination anywhere:					
	2	2	2+2p	2+2p	2	2
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example 1

```
BANZ    PRG35,*-
```

	Before Instruction		After Instruction
AR	>1	AR	>0
PC	>46	PC	>35
or			
AR	>0	AR	>FFFF
PC	>46	PC	>48

Example 2

```
BANZ    PRG64,*+
```

	Before Instruction		After Instruction
AR	>FFFF	AR	>0
PC	>117	PC	>64
or			
AR	>0	AR	>1
PC	>117	PC	>119

Note:

BANZ is designed for loop control using the auxiliary registers as loop counters. Using *0+ or *0- allows modification of the loop counter by a variable step size. Care must be exercised when doing this, however, because the auxiliary registers behave as modulo 65536 counters, and zero may be passed without being detected if AR0 > 1.

BBNZ Branch on Bit Not Equal to Zero BBNZ

Syntax [<label>] BBNZ <pma>[,{ind}[,<next ARP>]]

Operands
0 ≤ pma ≤ 65535
0 ≤ next ARP ≤ 7

Execution
If test/control (TC) status bit = 1:
 Then pma → PC;
 Else (PC) + 2 → PC.
Modify AR (ARP) and ARP as specified.

Affected by TC.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	1	1	See Section 4.1						
Program Memory Address															

Description The current auxiliary register and ARP are modified as specified. Control then passes to the designated program memory address if TC = 1. Otherwise, control passes to the next instruction. Note that no AR or ARP modification occurs if nothing is specified in those fields. Pma can be either a symbolic or a numeric address. Note that the TC bit may be affected by the BIT, BITT, CMPR, LST1, NORM, RTC, and STC instructions.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	2 (br int-to-int) 2+p (ext-to-int)		2+p (int-to-ext) 2+2p (ext-to-ext)		– –	– –
'C25	True Conditions:					
	Destination on-chip RAM:					
	2	2	2+2p	2+2p	2	2
	Destination on-chip ROM:					
	3	3	3+2p	3+2p	3	3
	Destination external memory:					
	3+p	3+p	3+3p	3+3p	3+p	3+p
	False Condition:					
	Destination anywhere:					
	2	2	2+2p	2+2p	2	2
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example

```
BBNZ    PRG650     If TC = 1, 650 is loaded into the program
                   counter; otherwise, the program counter
                   is incremented by 2.
```

BBZ Branch on Bit Equal to Zero BBZ

Syntax [<label>] BBZ <pma>[,{ind}[,<next ARP>]]

Operands 0 ≤ pma ≤ 65535
0 ≤ next ARP ≤ 7

Execution If test/control (TC) status bit = 0:
Then pma → PC;
Else (PC) + 2 → PC.
Modify AR(ARP) and ARP as specified.

Affected by TC bit.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	1	See Section 4.1						
Program Memory Address															

Description The current auxiliary register and ARP are modified as specified. Control then passes to the designated program memory address if TC = 0. Otherwise, control passes to the next instruction. No AR or ARP modification occurs if nothing is specified in those fields. Pma can be either a symbolic or a numeric address. Note that the TC bit is affected by the BIT, BITT, CMPR, LST1, NORM, RTC, and STC instructions.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	2 (br int-to-int) 2+p (ext-to-int)		2+p (int-to-ext) 2+2p (ext-to-ext)		– –	– –
'C25	True Conditions:					
	Destination on-chip RAM:					
	2	2	2+2p	2+2p	2	2
	Destination on-chip ROM:					
	3	3	3+2p	3+2p	3	3
	Destination external memory:					
	3+p	3+p	3+3p	3+3p	3+p	3+p
	False Condition:					
	Destination anywhere:					
	2	2	2+2p	2+2p	2	2
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example

```
BBZ     PRG325     If TC = 0, 325 is loaded into the program
                   counter; otherwise, the program counter
                   is incremented by 2.
```

BC Branch on Carry (TMS320C25) BC

Syntax [<label>] BC <pma>[,{ind}[,<next ARP>]]

Operands $0 \le \text{pma} \le 65535$
$0 \le \text{next ARP} \le 7$

Execution If carry bit C = 1:
Then pma → PC;
Else (PC) + 2 → PC.
Modify AR(ARP) and ARP as specified.

Affected by C.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	1	1	0	1	See Section 4.1						
Program Memory Address															

Description The current auxiliary register and ARP are modified as specified. Control then passes to the designated program memory address if the carry bit C 1. Otherwise, control passes to the next instruction. Note that no AR or ARP modification occurs if nothing is specified in those fields. Pma can be either a symbolic or a numeric address.

Note that the carry bit C is affected by all add, subtract, and accumulate instructions as well as the ABS, LST1, NEG, RC, SC, rotate, and shift instructions. The carry bit is not affected by execution of BC, BNC, or nonarithmetic instructions.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'C25	True Conditions:					
	Destination on-chip RAM:					
	2	2	2+2p	2+2p	2	2
	Destination on-chip ROM:					
	3	3	3+2p	3+2p	3	3
	Destination external memory:					
	3+p	3+p	3+3p	3+3p	3+p	3+p
	False Condition:					
	Destination anywhere:					
	2	2	2+2p	2+2p	2	2
	Cycle Timings for a Repeat Execution					
'C25	not repeatable					

Example

```
BC    PRG512      If the carry bit C = 1, 512 is loaded into
                  the program counter; otherwise, the PC is
                  incremented by 2.
```

BGEZ Branch if Accumulator Greater Than or Equal to Zero BGEZ

Syntax [<label>] BGEZ <pma>[,{ind}[,<next ARP>]]

Operands $0 \le \text{pma} \le 65535$
$0 \le \text{next ARP} \le 7$

Execution If (ACC) $\ge$ 0:
Then pma → PC;
Else (PC) + 2 → PC.
Modify AR (ARP) and ARP as specified.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	0	1	See Section 4.1						
Program Memory Address															

Description The current auxiliary register and ARP are modified as specified. Control then passes to the designated program memory address (pma) if the contents of the accumulator are greater than or equal to zero. Otherwise, control passes to the next instruction. Note that no AR or ARP modification occurs if nothing is specified in those fields. Pma can be either a symbolic or a numeric address.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	2 (br int-to-int) 2+p (ext-to-int)		2+p (int-to-ext) 2+2p (ext-to-ext)		– –	– –
'C25	True Conditions:					
	Destination on-chip RAM:					
	2	2	2+2p	2+2p	2	2
	Destination on-chip ROM:					
	3	3	3+2p	3+2p	3	3
	Destination external memory:					
	3+p	3+p	3+3p	3+3p	3+p	3+p
	False Condition:					
	Destination anywhere:					
	2	2	2+2p	2+2p	2	2
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example

```
BGEZ    PRG217    217 is loaded into the program counter if
                  the accumulator is greater than or equal
                  to zero.
```

BGZ Branch if Accumulator Greater Than Zero BGZ

Syntax [<label>] BGZ <pma>[,{ind}[,<next ARP>]]

Operands $0 \leq \text{pma} \leq 65535$
$0 \leq \text{next ARP} \leq 7$

Execution If (ACC) > 0:
 Then pma → PC;
 Else (PC) + 2 → PC.
Modify AR(ARP) and ARP as specified.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	1	1	See Section 4.1						
Program Memory Address															

Description The current auxiliary register and ARP are modified as specified. Control then passes to the designated program memory address (pma) if the contents of the accumulator are greater than zero. Otherwise, control passes to the next instruction. Note that no AR or ARP modification occurs if nothing is specified in those fields. Pma can be either a symbolic or a numeric address.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	2 (br int-to-int)		2+p (int-to-ext)		–	–
	2+p (ext-to-int)		2+2p (ext-to-ext)		–	–
'C25	True Conditions:					
	Destination on-chip RAM:					
	2	2	2+2p	2+2p	2	2
	Destination on-chip ROM:					
	3	3	3+2p	3+2p	3	3
	Destination external memory:					
	3+p	3+p	3+3p	3+3p	3+p	3+p
	False Condition:					
	Destination anywhere:					
	2	2	2+2p	2+2p	2	2
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example

```
BGZ    PRG342     342 is loaded into the program counter if
                  the accumulator is greater than zero.
```

BIOZ Branch on I/O Status Equal to Zero BIOZ

Syntax [<label>] BIOZ <pma>[,{ind}[,<next ARP>]]

Operands $0 \leq \text{pma} \leq 65535$
$0 \leq \text{next ARP} \leq 7$

Execution

If $\overline{\mathrm{BIO}} = 0$:
 Then pma → PC;
 Else (PC) + 2 → PC.
Modify AR(ARP) and ARP as specified.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	1	0	1	See Section 4.1						
Program Memory Address															

Description The current auxiliary register and ARP are modified as specified. Control then passes to the designated program memory address (pma) if the $\overline{\mathrm{BIO}}$ pin is low. Otherwise, control passes to the next instruction. Note that no AR or ARP modification occurs if nothing is specified in those fields. Pma can be either a symbolic or a numeric address.

BIOZ in conjunction with the $\overline{\mathrm{BIO}}$ pin can be used to test if a peripheral is ready to send or receive data. Polling the $\overline{\mathrm{BIO}}$ pin using BIOZ may be preferable to an interrupt when executing time-critical loops.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	2 (br int-to-int)		2+p (int-to-ext)		–	–
	2+p (ext-to-int)		2+2p (ext-to-ext)		–	–
'C25	True Conditions:					
	Destination on-chip RAM:					
	2	2	2+2p	2+2p	2	2
	Destination on-chip ROM:					
	3	3	3+2p	3+2p	3	3
	Destination external memory:					
	3+p	3+p	3+3p	3+3p	3+p	3+p
	False Condition:					
	Destination anywhere:					
	2	2	2+2p	2+2p	2	2
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example

```
BIOZ    PRG64     If the BIO- pin is active (low), then
                  a branch to location 64 occurs.
```

Syntax

Direct: [<label>] BIT <dma>,<bit code>
Indirect: [<label>] BIT {ind},<bit code>[,<next ARP>]

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$
$0 \leq \text{bit code} \leq 15$

Execution

(PC) + 1 → PC
(dma bit at bit address (15-bit code)) → TC.

Affects TC.

Encoding

	15	14	13	12	11–8	7	6–0
Direct:	1	0	0	1	Bit Code	0	Data Memory Address
Indirect:	1	0	0	1	Bit Code	1	See Section 4.1

Description

The BIT instruction copies the specified bit of the data memory value to the TC bit of status register ST1. Note that the BITT, CMPR, LST1, and NORM instructions also affect the TC bit in status register ST1. A bit code value is specified that corresponds to a certain bit address in the instruction, as given by the following table:

Bit Address	Bit Code 11	10	9	8
(LSB) 0	1	1	1	1
1	1	1	1	0
2	1	1	0	1
3	1	1	0	0
4	1	0	1	1
5	1	0	1	0
6	1	0	0	1
7	1	0	0	0
8	0	1	1	1
9	0	1	1	0
10	0	1	0	1
11	0	1	0	0
12	0	0	1	1
13	0	0	1	0
14	0	0	0	1
(MSB) 15	0	0	0	0

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
BIT  >0,>8    (DP = 488)
or
BIT  *,8      If current auxiliary register contains >F400.
```

	Before Instruction		After Instruction
Data Memory >F400	>7E98	Data Memory >F400	>7E98
TC	>0	TC	>1

BITT Test Bit Specified by T Register BITT

Syntax

Direct: [<label>] BITT <dma>
Indirect: [<label>] BITT {ind}[,<next ARP>]

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution

(PC) + 1 → PC
(dma bit at bit address (15-T register(3-0))) → TC

Affects TC.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	1	0	1	1	1	0	Data Memory Address						
Indirect:	0	1	0	1	0	1	1	1	1	See Section 4.1						

Description

The BITT instruction copies the specified bit of the data memory value to the TC bit of status register ST1. Note that the BIT, CMPR, LST1, and NORM instructions also affect the TC bit in status register ST1. The bit address is specified by a bit code value contained in the LSBs of the T register, as given in the following table:

	Bit Code			
Bit Address	3	2	1	0
(LSB) 0	1	1	1	1
1	1	1	1	0
2	1	1	0	1
3	1	1	0	0
4	1	0	1	1
5	1	0	1	0
6	1	0	0	1
7	1	0	0	0
8	0	1	1	1
9	0	1	1	0
10	0	1	0	1
11	0	1	0	0
12	0	0	1	1
13	0	0	1	0
14	0	0	0	1
(MSB) 15	0	0	0	0

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
BITT   >0     Value in T register points to bit 14 of
              data word (DP = 240).
 or
BITT   *      If current auxiliary register contains >7800.
```

	Before Instruction		After Instruction
Data Memory >7800	>4DC8	Data Memory >7800	>4DC8
TR	>1	TR	>1
TC	>0	TC	>1

BLEZ Branch if Accumulator Less Than or Equal to Zero BLEZ

Syntax [<label>] BLEZ <pma>[,{ind}[,<next ARP>]]

Operands 0 ≤ pma ≤ 65535
0 ≤ next ARP ≤ 7

Execution If (ACC) ≤ 0:
Then pma → PC;
Else (PC) + 2 → PC.
Modify AR(ARP) and ARP as specified.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	1	See Section 4.1						
Program Memory Address															

Description The current auxiliary register and ARP are modified as specified. Control then passes to the designated program memory address (pma) if the contents of the accumulator are less than or equal to zero. Otherwise, control passes to the next instruction. Note that no AR or ARP modification occurs if nothing is specified in those fields. Pma can be either a symbolic or a numeric address.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	2 (br int-to-int)		2+p (int-to-ext)		–	–
	2+p (ext-to-int)		2+2p (ext-to-ext)		–	–
'C25	True Conditions:					
	Destination on-chip RAM:					
	2	2	2+2p	2+2p	2	2
	Destination on-chip ROM:					
	3	3	3+2p	3+2p	3	3
	Destination external memory:					
	3+p	3+p	3+3p	3+3p	3+p	3+p
	False Condition:					
	Destination anywhere:					
	2	2	2+2p	2+2p	2	2
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example

```
BLEZ    PRG63     63 is loaded into the program counter if
                  the accumulator is less than or equal to
                  zero.
```

BLKD Block Move from Data Memory to Data Memory BLKD

Syntax

Direct: [<label>] BLKD <dma1>,<dma2>
Indirect: [<label>] BLKD <dma1>,{ind}[,<next ARP>]

Operands

$0 \leq \text{dma1} \leq 65535$
$0 \leq \text{dma2} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution

TMS32020:

```
(PC) + 2 → TOS
dma1 → PC

If (repeat counter) ≠ 0:
  Then (dma1, addressed by PC) → dma2,
    Modify AR(ARP) and ARP as specified,
    (PC) + 1 → PC,
    (repeat counter) - 1 → repeat counter.

  Else (dma1, addressed by PC) → dma2
    Modify AR(ARP) and ARP as specified.
(TOS) → PC
```

TMS320C25:

```
(PC) + 2 → PC
(PFC) → MCS
dma1 → PFC

If (repeat counter) ≠ 0:
  Then (dma1, addressed by PFC) → dma2,
    Modify AR(ARP) and ARP as specified,
    (PFC) + 1 → PFC,
    (repeat counter) - 1 → repeat counter.

  Else (dma1, addressed by PFC) → dma2
    Modify AR(ARP) and ARP as specified.
(MCS) → PFC
```

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	1	1	1	1	1	1	0	1	0	Data Memory Address 2						
	Data Memory Address 1															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Indirect:	1	1	1	1	1	1	0	1	1	See Section 4.1						
	Data Memory Address 1															

Description

Consecutive memory words are moved from a source data memory block to a destination data memory block. The starting address (lowest) of the source block is defined by the second word of the instruction. The starting address of the destination block is defined by either the dma contained in the opcode (for direct addressing) or the current AR (for indirect addressing). In the indirect addressing mode, both the current AR and ARP may be modified in the usual manner. In the direct addressing mode, dma2 is used as the destination address for the block move but is not modified upon repeated executions of the instruction. Thus, the contents of memory at the dma2 address will be the same as the contents of memory at the last dma1 address in a repeat sequence. RPT or RPTK must be used with the BLKD instruction, in the indirect addressing mode, if more than one word is to be moved. The number of words to be moved is one greater than the number contained in the repeat counter RPTC at the beginning of the instruction. At the end of this instruction, the RPTC contains zero and, if using indirect addressing, AR(ARP) will be modified to contain the address after the end of the destination block. Note that the source and destination blocks do NOT have to be entirely on-chip or off-chip. However, BLKD cannot be used to transfer data from a memory-mapped register to any other location in data memory.

The PC points to the instruction following BLKD after execution. Interrupts are inhibited during a BLKD operation used with RPT or RPTK.

The BLKD instruction on the TMS32020 uses one level of stack. Therefore, the value on the bottom of the stack is lost since the stack is pushed and popped during the instruction operation.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	Data source internal:†					
	3	3+d	3+2p	3+d+2p	–	–
	Data source external:†					
	3+d	4+2d	3+d+2p	4+2d+2p	–	–
'C25	Source data in on-chip RAM:					
	3	3+d	3+2p	3+d+2p	3	3+d
	Source data in external memory:					
	4+d	4+2d	4+d+2p	4+2d+2p	4+d	4+2d
	Cycle Timings for a Repeat Execution					
'20	Data source internal:†					
	2+n	2+n+nd	2+n+2p	2+n+nd+2p	–	–
	Data source external:†					
	2+n+nd	2+2n+2nd	2+n+nd +2p	2+2n+2nd +2p	–	–
'C25	Source data in on-chip RAM:					
	2+n	2+n+nd	2+n+2p	2+n+nd+2p	2+n	2+n+nd
	Source data in external memory:					
	3+n+nd	2+2n+2nd	3+n+nd +2p	2+2n+2nd +2p	3+n+nd	2+2n+2nd

†Column headings 'DI/DE' refer to data destination.

Example

```
RPTK 2
BLKD >F400,*+ If current auxiliary register contains 1030.
```

dma1

	Before Instruction		After Instruction
Data Memory 62464	>7F98	Data Memory 62464	>7F98
Data Memory 62465	>FFE6	Data Memory 62465	>FFE6
Data Memory 62466	>9522	Data Memory 62466	>9522

dma2

	Before Instruction		After Instruction
Data Memory 1030	>8DEE	Data Memory 1030	>7F98
Data Memory 1031	>9315	Data Memory 1031	>FFE6
Data Memory 1032	>2531	Data Memory 1032	>9522

BLKP Block Move from Program Memory to Data Memory BLKP

Syntax

Direct: [<label>] BLKP <pma>,<dma>
Indirect: [<label>] BLKP <pma>,{ind}[,<next ARP>]

Operands

$0 \leq \text{pma} \leq 65535$
$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution

TMS32020:

```
(PC) + 2 → TOS
pma → PC

If (repeat counter) ≠ 0:
  Then (pma, addressed by PC) → dma,
    Modify AR(ARP) and ARP as specified,
    (PC) + 1 → PC,
    (repeat counter) - 1 → repeat counter.

  Else (pma, addressed by PC) → dma
    Modify AR(ARP) and ARP as specified.
(TOS) → PC
```

TMS320C25:

```
(PC) + 2 → PC
(PFC) → MCS
pma → PFC

If (repeat counter) ≠ 0:
  Then (pma, addressed by PFC) → dma,
    Modify AR(ARP) and ARP as specified,
    (PFC) + 1 → PFC,
    (repeat counter) - 1 → repeat counter.

  Else (pma, addressed by PFC) → dma
    Modify AR(ARP) and ARP as specified.
(MCS) → PFC
```

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	1	1	1	1	1	1	0	0	0	Data Memory Address						
	Program Memory Address															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Indirect:	1	1	1	1	1	1	0	0	1	See Section 4.1						
	Program Memory Address															

Description

Consecutive memory words are moved from a source program memory block to a destination data memory block. The starting address (lowest) of the source block is defined by the second word of the instruction. The starting address of the destination block is defined by either the dma contained in the opcode (for direct addressing) or the current AR (for indirect addressing). In the indirect addressing mode, both the ARP and the current AR may be modified in the usual manner. In the direct addressing mode, dma is used as the destination address for the block move but is not modified by repeated executions of the instruction. Thus, the contents of memory at the dma address will be the same as the contents of memory at the last pma address in a repeat sequence. RPT or RPTK must be used with the BLKP instruction if more than one word is to be moved. The number of words to be moved is one greater than the number contained in the repeat counter RPTC at the beginning of the instruction. At the end of this instruction, the RPTC contains zero and, if using indirect addressing, AR(ARP) will be modified to contain the address after the end of the destination block. Note that source and destination blocks do NOT have to be entirely on-chip or off-chip.

The PC points to the instruction following BLKP after execution. Interrupts are inhibited during a BLKP operation.

The BLKD instruction on the TMS32020 uses one level of stack. Therefore, the value on the bottom of the stack is lost since the stack is pushed and popped during the instruction operation.

If the MP/$\overline{\text{MC}}$ pin on the TMS320C25 is low at the time of execution of this instruction and the program memory address used is less than 4096, an on-chip ROM location will be read.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	Program source internal:†					
	3	3+d	3+2p	3+d+2p	–	–
	Program source external:†					
	3+p	4+d+p	3+3p	4+d+3p	–	–
'C25	Table in on-chip RAM:					
	3	3+d	4+2p	4+d+2p	4	4+d
	Table in on-chip ROM:					
	4	4+d	4+2p	4+d+2p	4	4+d
	Table in external memory:					
	4+p	4+d+p	4+3p	4+d+3p	4+p	4+d+p
	Cycle Timings for a Repeat Execution					
'20	Program source internal:†					
	2+n	2+n+nd	2+n+2p	2+n+nd+2p	–	–
	Program source external:†					
	2+n+np	2+2n+nd +np	2+n+np +2p	2+2n+nd+np +2p	–	–
'C25	Table in on-chip RAM:					
	2+n	2+n+nd	3+n+2p	3+n+nd+2p	3+n	3+n+nd
	Table in on-chip ROM:					
	3+n	3+n+nd	3+n+2p	3+n+nd+2p	3+n	3+n+nd
	Table in external memory:					
	3+n+np	2+2n+nd +np	3+n+np +2p	2+2n+nd+np +2p	3+n+np	2+2n+nd +np

†Column headings 'DI/DE' refer to data destination.

BLKP Block Move from Program Memory to Data Memory BLKP

Example

```
RPTK 2
BLKP 65120,*+ If current auxiliary register contains 2048.
```

pma

	Before Instruction		After Instruction
Program Memory 65120	>A089	Program Memory 65120	>A089
Program Memory 65121	>2DCE	Program Memory 65121	>2DCE
Program Memory 65122	>3A9F	Program Memory 65122	>3A9F

dma

	Before Instruction		After Instruction
Data Memory 2048	>1234	Data Memory 2048	>A089
Data Memory 2049	>2005	Data Memory 2049	>2DCE
Data Memory 2050	>E98C	Data Memory 2050	>3A9F

BLZ Branch if Accumulator Less Than Zero BLZ

Syntax [<label>] BLZ <pma>[,{ind}[,<next ARP>]]

Operands $0 \leq \text{pma} \leq 65535$
$0 \leq \text{next ARP} \leq 7$

Execution

```
If (ACC) < 0:
  Then pma → PC;
  Else (PC) + 2 → PC.
Modify AR(ARP) and ARP as specified.
```

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	1	1	See Section 4.1						
Program Memory Address															

Description The current auxiliary register and ARP are modified as specified. Control then passes to the designated program memory address (pma) if the contents of the accumulator are less than zero. Otherwise, control passes to the next instruction. Note that no AR or ARP modification occurs when nothing is specified in those fields. Pma can be either a symbolic or a numeric address.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	2 (br int-to-int) 2+p (ext-to-int)		2+p (int-to-ext) 2+2p (ext-to-ext)		– –	– –
'C25	True Conditions:					
	Destination on-chip RAM:					
	2	2	2+2p	2+2p	2	2
	Destination on-chip ROM:					
	3	3	3+2p	3+2p	3	3
	Destination external memory:					
	3+p	3+p	3+3p	3+3p	3+p	3+p
	False Condition:					
	Destination anywhere:					
	2	2	2+2p	2+2p	2	2
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example

```
BLZ    PRG481    481 is loaded into the program counter if
                 the accumulator is less than zero.
```

Syntax [<label>] BNC <pma>[,{ind}[,<next ARP>]]

Operands
0 ≤ pma ≤ 65535
0 ≤ next ARP ≤ 7

Execution
If carry bit C = 0:
 Then pma → PC;
 Else (PC) + 2 → PC.
Modify AR(ARP) and ARP as specified.

Affected by C.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	1	1	1	1	See Section 4.1						
Program Memory Address															

Description The current auxiliary register and ARP are modified as specified. Control then passes to the designated program memory address if the carry bit C = 0. Otherwise, control passes to the next instruction. Note that no AR or ARP modification occurs when nothing is specified in those fields. Pma can be either a symbolic or a numeric address.

Note that the carry bit C is affected by all add, subtract, and accumulate instructions as well as the ABS, LST1, NEG, RC, SC, rotate, and shift instructions. The carry bit is not affected by execution of the BC, BNC, or nonarithmetic instructions.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'C25	True Conditions:					
	Destination on-chip RAM:					
	2	2	2+2p	2+2p	2	2
	Destination on-chip ROM:					
	3	3	3+2p	3+2p	3	3
	Destination external memory:					
	3+p	3+p	3+3p	3+3p	3+p	3+p
	False Condition:					
	Destination anywhere:					
	2	2	2+2p	2+2p	2	2
	Cycle Timings for a Repeat Execution					
'C25	not repeatable					

Example

```
BNC   PRG325      If the carry bit C = 0, 325 is loaded into
                  the program counter. Otherwise, the PC is
                  incremented by 2.
```

Syntax [<label>] BNV <pma>[,{ind}[,<next ARP>]]

Operands
0 ≤ pma ≤ 65535
0 ≤ next ARP ≤ 7

Execution
If overflow OV status bit = 0:
  Then pma → PC;
  Else (PC) + 2 → PC and 0 → OV.
Modify AR(ARP) and ARP as specified.

Affects OV; affected by OV.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	1	1	1	See Section 4.1						
Program Memory Address															

Description The current auxiliary register and ARP are modified as specified. Control then passes to the designated program memory address (pma) if the OV (overflow flag) is clear. Otherwise, the OV is cleared, and control passes to the next instruction. Note that no AR or ARP modification occurs if nothing is specified in those fields. Pma can be either a symbolic or a numeric address.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	2 (br int-to-int)		2+p (int-to-ext)		–	–
	2+p (ext-to-int)		2+2p (ext-to-ext)		–	–
'C25	True Conditions:					
	Destination on-chip RAM:					
	2	2	2+2p	2+2p	2	2
	Destination on-chip ROM:					
	3	3	3+2p	3+2p	3	3
	Destination external memory:					
	3+p	3+p	3+3p	3+3p	3+p	3+p
	False Condition:					
	Destination anywhere:					
	2	2	2+2p	2+2p	2	2
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example

```
BNV    PRG315     315 is loaded into the program counter
                  if the overflow flag is clear.  OV is
                  cleared.
```

BNZ Branch if Accumulator Not Equal to Zero BNZ

Syntax [<label>] BNZ <pma>[,{ind}[,<next ARP>]]

Operands $0 \leq \text{pma} \leq 65535$
$0 \leq \text{next ARP} \leq 7$

Execution
```
If (ACC) ≠ 0:
 Then pma → PC;
 Else (PC) + 2 → PC.
Modify AR(ARP) and ARP as specified.
```

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	1	1	See Section 4.1						
Program Memory Address															

Description The current auxiliary register and ARP are modified as specified. Control then passes to the designated program memory address (pma) if the contents of the accumulator are not equal to zero. Otherwise, control passes to the next instruction. Note that no AR or ARP modification occurs if nothing is specified in those fields. Pma can be either a symbolic or a numeric address.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	2 (br int-to-int) 2+p (ext-to-int)		2+p (int-to-ext) 2+2p (ext-to-ext)		– –	– –
'C25	True Conditions:					
	Destination on-chip RAM:					
	2	2	2+2p	2+2p	2	2
	Destination on-chip ROM:					
	3	3	3+2p	3+2p	3	3
	Destination external memory:					
	3+p	3+p	3+3p	3+3p	3+p	3+p
	False Condition:					
	Destination anywhere:					
	2	2	2+2p	2+2p	2	2
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example
```
BNZ   PRG320     320 is loaded into the program counter
                 if the accumulator does not equal zero.
```

Syntax [<label>] BV <pma>[,{ind}[,<next ARP>]]

Operands

0 ≤ pma ≤ 65535
0 ≤ next ARP ≤ 7

Execution

If overflow (OV) status bit = 1:
 Then pma → PC and 0 → OV;
 Else (PC) + 2 → PC.
Modify AR(ARP) and ARP as specified.

Affects OV; affected by OV.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	1	See Section 4.1						
Program Memory Address															

Description The current auxiliary register and ARP are modified as specified, and the overflow flag is cleared. Control passes to the designated program memory address (pma) if the OV (overflow flag) is set. Otherwise, control passes to the next instruction. Note that no AR or ARP modification occurs if nothing is specified in those fields. Pma can be either a symbolic or a numeric address.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	2 (br int-to-int)		2+p (int-to-ext)		–	–
	2+p (ext-to-int)		2+2p (ext-to-ext)		–	–
'C25	True Conditions:					
	Destination on-chip RAM:					
	2	2	2+2p	2+2p	2	2
	Destination on-chip ROM:					
	3	3	3+2p	3+2p	3	3
	Destination external memory:					
	3+p	3+p	3+3p	3+3p	3+p	3+p
	False Condition:					
	Destination anywhere:					
	2	2	2+2p	2+2p	2	2
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example

```
BV    PRG610    If an overflow has occurred since the over-
                flow flag was last cleared, then 610 is
                loaded in the program counter.  OV is
                cleared.
```

BZ Branch if Accumulator Equals Zero BZ

Syntax [<label>] BZ <pma>[,{ind}[,<next ARP>]]

Operands $0 \le \text{pma} \le 65535$
$0 \le \text{next ARP} \le 7$

Execution If (ACC) = 0:
  Then pma → PC;
  Else (PC) + 2 → PC.
Modify AR(ARP) and ARP as specified.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	1	0	1	See Section 4.1						
Program Memory Address															

Description The current auxiliary register and ARP are modified as specified. Control then passes to the designated program memory address (pma) if the contents of the accumulator are equal to zero. Otherwise, control passes to the next instruction. Note that no AR or ARP modification occurs if nothing is specified in those fields. Pma can be either a symbolic or a numeric address.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	2 (br int-to-int) 2+p (ext-to-int)		2+p (int-to-ext) 2+2p (ext-to-ext)		– –	– –
'C25	True Conditions:					
	Destination on-chip RAM:					
	2	2	2+2p	2+2p	2	2
	Destination on-chip ROM:					
	3	3	3+2p	3+2p	3	3
	Destination external memory:					
	3+p	3+p	3+3p	3+3p	3+p	3+p
	False Condition:					
	Destination anywhere:					
	2	2	2+2p	2+2p	2	2
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example

```
BZ    PRG102      102 is loaded into the program counter if
                  the accumulator is equal to zero.
```

Syntax [<label>] CALA

Operands None

Execution (PC) + 1 → TOS
(ACC(15-0)) → PC

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	1	0	0	1	0	0

Description The current program counter is incremented and pushed onto the top of the stack. Then, the contents of the lower half of the accumulator are loaded into the PC. The carry bit on the TMS320C25 is unaffected by this instruction.

The CALA instruction is used to perform computed subroutine calls.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	2	2	2+p	2+p	–	–
'C25	Destination on-chip RAM:					
	2	2	2+p	2+p	2	2
	Destination on-chip ROM:					
	3	3	3+p	3+p	3	3
	Destination external memory:					
	3+p	3+p	3+2p	3+2p	3+p	3+p
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example `CALA`

	Before Instruction		After Instruction
PC	>25	PC	>83
ACC	>83	ACC	>83
Stack (20)	>32 >75 >84 >49	Stack (20)	>26 >32 >75 >84
Stack (C25)	>32 >75 >84 >49 >0 >0 >0 >0	Stack (C25)	>26 >32 >75 >84 >49 >0 >0 >0

CALL Call Subroutine CALL

Syntax [<label>] CALL <pma>[,{ind}[,<next ARP>]]

Operands
0 ≤ pma ≤ 65535
0 ≤ next ARP ≤ 7

Execution
(PC) + 2 → TOS
pma → PC

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	0	1	See Section 4.1						
Program Memory Address															

Description The current auxiliary register and ARP are modified as specified, and the PC (program counter) is incremented by two and pushed onto the top of the stack. The specified program memory address (pma) is then loaded into the PC. Note that no AR or ARP modification occurs if nothing is specified in those fields. Pma can be either a symbolic or a numeric address.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	2 (br int-to-int)		2+p (int-to-ext)		–	–
	2+p (ext-to-int)		2+2p (ext-to-ext)		–	–
'C25	True Conditions:					
	Destination on-chip RAM:					
	2	2	2+2p	2+2p	2	2
	Destination on-chip ROM:					
	3	3	3+2p	3+2p	3	3
	Destination external memory:					
	3+p	3+p	3+3p	3+3p	3+p	3+p
	False Condition:					
	Destination anywhere:					
	2	2	2+2p	2+2p	2	2
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example

```
CALL    PRG109
```

	Before Instruction		After Instruction
PC	>33	PC	>6D
Stack (20)	>71 >48 >16 >80	Stack (20)	>35 >71 >48 >16
Stack (C25)	>71 >48 >16 >80 >0 >0 >0 >0	Stack (C25)	>35 >71 >48 >16 >80 >0 >0 >0

CMPL Complement Accumulator CMPL

Syntax [<label>] CMPL

Operands None

Execution (PC) + 1 → PC
$(\overline{\text{ACC}}) \rightarrow \text{ACC}$

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	1	0	0	1	1	1

Description The contents of the accumulator are replaced with its logical inversion (one's complement).

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example `CMPL`

	Before Instruction		After Instruction
ACC	X (C) >F7982513	ACC	X (C) >0867DAEC

CMPR Compare Auxiliary Register with Auxiliary Register AR0 CMPR

Syntax [<label>] CMPR <constant>

Operands 0 ≤ CM ≤ 3

Execution (PC) + 1 → PC
Compare AR(ARP) to AR0, placing result in TC bit of status register ST1.

Affects TC.
Not affected by SXM; does not affect SXM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	1	0	1	0	0	CM	

Description The CMPR instruction performs the following comparisons dependent on the value of CM:

If CM = 00, test if AR(ARP) = AR0
If CM = 01, test if AR(ARP) < AR0
If CM = 10, test if AR(ARP) > AR0
If CM = 11, test if AR(ARP) ≠ AR0

If the result of a test is true, a one is loaded into the TC status bit. Otherwise, TC is loaded with a zero. The auxiliary registers are treated as unsigned integers in the comparison.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example

```
CMPR    2       (ARP = 4)
```

	Before Instruction		After Instruction
AR0	>FFFF	AR0	>FFFF
AR4	>7FFF	AR4	>7FFF
TC	>1	TC	>0

CNFD Configure Block as Data Memory CNFD

Syntax [<label>] CNFD

Operands None

Execution (PC) + 1 → PC
0 → RAM configuration control (CNF) status bit

Affects CNF.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	0	0	0	1	0	0

Description On-chip RAM block 0 is configured as data memory. The block is mapped to locations 512 through 767 in data memory. This instruction is the complement of the CNFP instruction and sets the CNF bit in status register ST1 to a zero. CNF is also loaded by the CNFP and LST1 instructions.

On the TMS32020, the instruction fetch immediately following a CNFD or CNFP instruction uses the old CNF value. The second fetch uses the new CNF value, even if it is the fetch of the second word of a two-word instruction.

On the TMS320C25, the next two instruction fetches immediately following a CNFD or CNFP instruction use the old value of CNF.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example

```
CNFD          A zero is loaded into the CNF status bit, thus
              configuring block B0 as data memory (see
              memory maps in Section 3.4).
```

CNFP Configure Block as Program Memory CNFP

Syntax [<label>] CNFP

Operands None

Execution (PC) + 1 → PC
1 → RAM configuration control (CNF) status bit

Affects CNF.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	0	0	0	1	0	1

Description On-chip RAM block 0 is configured as program memory. The block is mapped to locations 65280 through 65535 in program memory space. This instruction is the complement of the CNFD instruction and sets the CNF bit in status register ST1 to a one. CNF is also loaded by the CNFD and LST1 instruction.

Configuring this block as program memory allows the use of the program counter as an address generator to access data from on-chip RAM. Used in conjunction with the repeat instructions, this allows two data memory locations to be addressed simultaneously, one from the auxiliary registers and one from the program counter. Instructions that take advantage of this feature are the MAC, MACD, BLKD, and BLKP instructions.

On the TMS32020, the instruction fetch immediately following a CNFD or CNFP instruction uses the old CNF value. The second fetch uses the new CNF value, even if it is the fetch of the second word of a two-word instruction.

On the TMS320C25, the next two instruction fetches immediately following a CNFD or CNFP instruction use the old value of CNF.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example

```
CNFP          The CNF bit is set to a logic 1, thus config-
              uring block B0 as program memory (see memory
              maps in Section 3.4)
```

Syntax [<label>] DINT

Operands None

Execution (PC) + 1 → PC
1 → interrupt mode (INTM) status bit

Affects INTM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	0	0	0	0	0	1

Description The interrupt mode (INTM) status bit is set to logic 1. Maskable interrupts are disabled immediately after the DINT instruction executes. Note that the LST instruction does not affect INTM.

The unmaskable interrupt, $\overline{\text{RS}}$, is not disabled by this instruction, and the interrupt mask register (IMR) is unaffected. Interrupts are also disabled by a reset.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example

```
DINT          Maskable interrupts are disabled, and INTM is
              set to one.
```

DMOV Data Move in Data Memory DMOV

Syntax

Direct: [<label>] DMOV <dma>
Indirect: [<label>] DMOV {ind}[,<next ARP>]

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution

$(\text{PC}) + 1 \rightarrow \text{PC}$
$(\text{dma}) \rightarrow \text{dma} + 1$

Affected by CNF.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	1	0	1	1	0	0	Data Memory Address						
Indirect:	0	1	0	1	0	1	1	0	1	See Section 4.1						

Description

The contents of the specified data memory address are copied into the contents of the next higher address. DMOV works only within the on-chip data RAM blocks B0, B1, and B2. It works within block B0 if it is configured as data memory, and the data move function is continuous across the boundaries of blocks B0 and B1; ie., it works for locations 512 to 1023. The data move function cannot be used on external data memory. If used on external data memory or memory-mapped registers, DMOV will read the specified memory location but will perform no other operations.

When data is copied from the addressed location to the next higher location, the contents of the addressed location remain unaltered.

The data move function is useful in implementing the z^{-1} delay encountered in digital signal processing. The DMOV function is included in the LTD and MACD instructions (see the LTD and MACD instructions for more information).

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
DMOV    DAT8    (DP=4)
```

or

```
DMOV    *       If current auxiliary register contains 520.
```

	Before Instruction		After Instruction
Data Memory 520	>43	Data Memory 520	>43
Data Memory 521	>2	Data Memory 521	>43

Syntax [<label>] EINT

Operands None

Execution (PC) + 1 → PC
0 → interrupt-mode (INTM) status bit

Affects INTM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0

Description The interrupt-mode flag (INTM) in the status register is cleared to logic 0. Maskable interrupts are enabled after the instruction following EINT executes. This allows an interrupt service routine to re-enable interrupts and execute a RET instruction before any other pending interrupts are processed. Note that the LST instruction does not affect INTM. (See the DINT instruction for further information.)

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example

```
EINT          Unmasked interrupts are enabled, and INTM is
              set to zero.
```

FORT Format Serial Port Registers FORT

Syntax [<label>] FORT <constant>

Operands Constant = 0 or 1

Execution (PC) + 1 → PC
Constant → format (FO) status bit

Affects FO.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	0	0	1	1	1	FO

Description The format (FO) status bit is loaded by the instruction with the LSB specified in the instruction. The FO bit is used to control the formatting of the transmit and receive shift registers of the serial port. If FO = 0, the registers are configured to receive/transmit 16-bit words. If FO = 1, the registers are configured to receive/transmit 8-bit bytes. FO is set to zero on a reset.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example

```
FORT    1       The FO status bit is loaded with 1, making
                the bit length of the serial port 8 bits.
```

Syntax [<label>] IDLE

Operands None

Execution **TMS32020:**

(PC) + 1 → PC

TMS320C25:

(PC) + 1 → PC
0 → interrupt mode (INTM) status bit

Affects INTM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	0	1	1	1	1	1

Description The IDLE instruction forces the program being executed to wait until an interrupt or reset occurs. The PC is incremented only once, and the device remains in an idle state until interrupted. On the TMS32020, the INTM bit must be set to zero in order for the maskable interrupts to be recognized. On the TMS320C25, INTM is automatically set to zero. Execution of the IDLE instruction causes the TMS320C25 to enter the powerdown mode (see Section 3.6.7).

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1 (min waits for INT)		1+p (min waits for INT)		–	–
'C25	(Interrupt) destination on-chip ROM: 3 (min waits for INT) (Interrupt) destination external memory: 3+2p (min waits for INT)					
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example

```
IDLE          The processor idles until a reset or unmasked
              interrupt occurs.
```

Syntax

Direct: [<label>] IN <dma>,<PA>
Indirect: [<label>] IN {ind},<PA>[,<next ARP>]

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$
$0 \leq \text{port address PA} \leq 15$

Execution

(PC) + 1 → PC
Port address → address bus A3-A0
0 → address bus A15-A4
Data bus D15-D0 → dma

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	1	0	0	0	Port Address				0	Data Memory Address						
Indirect:	1	0	0	0	Port Address				1	See Section 4.1						

Description

The IN instruction reads a 16-bit value from one of the external I/O ports into the specified data memory location. The $\overline{\text{IS}}$ line goes low to indicate an I/O access, and the $\overline{\text{STRB}}$, R/$\overline{\text{W}}$, and READY timings are the same as for an external data memory read.

Words

1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	1+i	2+d+i	2+p+i	3+d+p+i	–	–
'C25	2+i	2+d+i	2+p+i	3+d+p+i	2+i	2+d+i
	Cycle Timings for a Repeat Execution					
'20	n+ni	2n+nd+ni	2n+p+ni	3n+nd+p+ni	–	–
'C25	1+n+ni	2n+nd+ni	1+n+p+ni	1+2n+nd+p +ni	1+n+ni	2n+nd+ni

Example

```
IN      STAT,PA5        Read in word from peripheral on port
                        address 5.  Store in data memory
                        location STAT.
```

or

```
LRLK    1,520           Load AR1 with decimal 520.
LARP    1               Load ARP with decimal 520.
IN      *-,PA1,0        Read in word from peripheral on port
                        address 1.  Store in data memory
                        location 520.  Decrement AR1 to 519.
                        Load the ARP with 0.
```

LAC Load Accumulator with Shift LAC

Syntax

Direct: [<label>] LAC <dma>[,<shift>]
Indirect: [<label>] LAC {ind}[,<shift>[,<next ARP>]]

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$
$0 \leq \text{shift} \leq 15$ (defaults to 0)

Execution

$(\text{PC}) + 1 \rightarrow \text{PC}$
$(\text{dma}) \times 2^{\text{shift}} \rightarrow \text{ACC}$

If SXM = 1:
Then (dma) is sign-extended.
If SXM = 0:
Then (dma) is not sign-extended.

Affected by SXM.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	0	1	0	Shift				0	Data Memory Address						
Indirect:	0	0	1	0	Shift				1	See Section 4.1						

Description

The contents of the specified data memory address are left-shifted and loaded into the accumulator. During shifting, low-order bits are zero-filled. High-order bits are sign-extended if SXM = 1 and zeroed if SXM = 0.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
LAC    DAT6,4   (DP = 8)
or
LAC    *,4      If current auxiliary register contains 1030.
```

		Before Instruction			After Instruction
Data Memory 1030		>1	Data Memory 1030		>1
ACC	X	>12345678	ACC	X	>10
	C			C	

LACK Load Accumulator Immediate Short LACK

Syntax [<label>] LACK <constant>

Operands 0 ≤ constant ≤ 255

Execution (PC) + 1 → PC
8-bit positive constant → ACC

Not affected by SXM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	0	1	0	8-Bit Constant							

Description The 8-bit constant is loaded into the accumulator right-justified. The upper 24 bits of the accumulator are zeroed (i.e., sign extension is suppressed).

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example `LACK >15`

	Before Instruction		After Instruction
ACC	X | >31	ACC	X | >15
	C		C

LACT — Load Accumulator with Shift Specified by T Register — LACT

Syntax

Direct: [<label>] LACT <dma>
Indirect: [<label>] LACT {ind}[,<next ARP>]

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution

$(\text{PC}) + 1 \rightarrow \text{PC}$
$(\text{dma}) \times 2^{\text{T register(3-0)}} \rightarrow \text{ACC}$

If SXM = 1:
Then (dma) is sign-extended.
If SXM = 0:
Then (dma) is not sign-extended.

Affected by SXM.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	0	0	0	1	0	0	Data Memory Address						
Indirect:	0	1	0	0	0	0	1	0	1	See Section 4.1						

Description

The LACT instruction loads the accumulator with a data memory value that has been left-shifted. The left-shift is specified by the four LSBs of the T register, resulting is shift options from 0 to 15 bits. Using the T register's contents as a shift code provides a variable shift mechanism.

LACT may be used to denormalize a floating-point number if the actual exponent is placed in the four LSBs of the T register and the mantissa is referenced by the data memory address. Note that this method of denormalization can only be used when the magnitude of the exponent is four bits or less.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
LACT   DAT1    (DP = 6)
```
or
```
LACT   *        If current auxiliary register contains 769.
```

	C	Before Instruction		C	After Instruction
Data Memory 769		>1376	Data Memory 769		>1376
ACC	X	>98F7EC83	ACC	X	>13760
T		>3014	T		>3014

LALK Load Accumulator Long Immediate with Shift LALK

Syntax [<label>] LALK <constant>[,<shift>]

Operands 16-bit constant
$0 \leq \text{shift} \leq 15$ (defaults to 0)

Execution (PC) + 2 → PC
Constant x 2^{shift} → ACC

If SXM = 1:
Then $-32768 \leq \text{constant} \leq 32767$.
If SXM = 0:
Then $0 \leq \text{constant} \leq 65535$.

Affected by SXM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	Shift				0	0	0	0	0	0	0	1
16-Bit Constant															

Description The left-shifted 16-bit immediate value is loaded into the accumulator. The shifted 16-bit constant is sign-extended if SXM = 1; otherwise, the high-order bits of the accumulator (past the shift) are set to zero. Note that the MSB of the accumulator can only be set if SXM = 1 and a negative number is loaded. The shift count is optional and defaults to zero.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	2	2	2+2p	2+2p	–	–
'C25	2	2	2+2p	2+2p	2	2
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example 1 `LALK >F794,8 (SXM=1)`

		Before Instruction			After Instruction
ACC	X	>12345678	ACC	X	>FFF79400
	C			C	

Example 2 `LALK >F794,8 (SXM=0)`

		Before Instruction			After Instruction
ACC	X	>12345678	ACC	X	>F79400
	C			C	

LAR Load Auxiliary Register LAR

Syntax

Direct: [<label>] LAR <AR>,<dma>
Indirect: [<label>] LAR <AR>,{ind}[,<next ARP>]

Operands

$0 \le \text{dma} \le 127$
$0 \le$ auxiliary register AR ≤ 7
$0 \le$ next ARP ≤ 7

Execution

(PC) + 1 → PC
(dma) → auxiliary register AR

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	0	1	1	0	AR			0	Data Memory Address						
Indirect:	0	0	1	1	0	AR			1	See Section 4.1						

Description

The contents of the specified data memory address are loaded into the designated auxiliary register (AR).

The LAR and SAR (store auxiliary register) instructions can be used to load and store the auxiliary registers during subroutine calls and interrupts. If an auxiliary register is not being used for indirect addressing, LAR and SAR enable the register to be used as an additional storage register, especially for swapping values between data memory locations without affecting the contents of the accumulator.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	2n+nd	n+p	2n+nd+p	n	2n+nd

Example 1

```
LAR    AR0,DAT10     (DP = 4)
```

	Before Instruction		After Instruction
Data Memory 522	>18	Data Memory 522	>18
AR0	>6	AR0	>18

Example 2

```
LARP  AR4
LAR   AR4,*-
```

	Before Instruction		After Instruction
Data Memory 617	>32	Data Memory 617	>32
AR4	>617	AR4	>32

Note:

LAR, in the indirect addressing mode, ignores any AR modifications if the AR specified by the instruction is the same as that pointed to by the ARP. Therefore, in Example 2, AR4 is not decremented after the LAR instruction.

LARK Load Auxiliary Register Immediate Short LARK

Syntax [<label>] LARK <AR>,<constant>

Operands
0 ≤ constant ≤ 255
0 ≤ auxiliary register AR ≤ 7

Execution
(PC) + 1 → PC
8-bit constant → auxiliary register AR

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	AR			8-Bit Constant							

Description The 8-bit positive constant is loaded into the designated auxiliary register (AR) right-justified and zero-filled (i.e., sign-extension suppressed).

LARK is useful for loading an initial loop counter value into an auxiliary register for use with the BANZ instruction.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example `LARK AR0,>15`

	Before Instruction		After Instruction
AR0	>0	AR0	>15

LARP Load Auxiliary Register Pointer LARP

Syntax [<label>] LARP <constant>

Operands 0 ≤ constant ≤ 7

Execution (PC) + 1 → PC
(ARP) → ARB
Constant → ARP

Affects ARP and ARB.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	0	1	1	0	0	0	1	ARP		

Description The auxiliary register pointer is loaded with the contents of the three LSBs of the instruction (a 3-bit constant identifying the desired auxiliary register). The old ARP is copied to the ARB field of status register ST1. ARP can also be modified by the LST, LST1, and MAR instructions, as well as any instruction that is used in the indirect addressing mode.

The LARP instruction is a subset of MAR; i.e., the opcode is the same as MAR in the indirect addressing mode. The following instruction has the same effect as LARP:

```
MAR *,<constant>
```

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example

```
LARP   1        Any succeeding instructions will use
                auxiliary register AR1 for indirect
                addressing.
```

LDP Load Data Memory Page Pointer

Syntax

Direct: [<label>] LDP <dma>
Indirect: [<label>] LDP {ind}[,<next ARP>]

Operands

0 ≤ dma ≤ 127
0 ≤ next ARP ≤ 7

Execution

(PC) + 1 → PC
Nine LSBs of (dma) → data page pointer register (DP) status bits

Affects DP.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	1	0	0	1	0	0	Data Memory Address						
Indirect:	0	1	0	1	0	0	1	0	1	See Section 4.1						

Description

The nine LSBs of the contents of the addressed data memory location are loaded into the DP (data memory page pointer) register. The DP and 7-bit data memory address are concatenated to form 16-bit data memory addresses. The DP may also be loaded by the LST and LDPK instructions.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	2n+nd	n+p	2n+nd+p	n	2n+nd

Example

```
LDP   DAT127   (DP = 511)
or
LDP   *        If current auxiliary register contains 65535.
```

	Before Instruction		After Instruction
Data Memory 65535	>FEDC	Data Memory 65535	>FEDC
DP	>1FF	DP	>DC

LDPK Load Data Memory Page Pointer Immediate LDPK

Syntax [<label>] LDPK <constant>

Operands 0 ≤ constant ≤ 511

Execution (PC) + 1 → PC
Constant → data memory page pointer (DP) status bits

Affects DP.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	0	0	DP								

Description The DP (data memory page pointer) register is loaded with a 9-bit constant. The DP and 7-bit data memory address are concatenated to form 16-bit direct data memory addresses. DP ≥ 8 specifies external data memory. DP = 4 through 7 specifies on-chip RAM blocks B0 or B1. Block B2 is located in the upper 32 words of page 0. DP may also be loaded by the LST and LDP instructions.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example

```
LDPK    64    The data page pointer is set to 64.
```

Syntax

Direct: [<label>] LPH <dma>
Indirect: [<label>] LPH {ind}[,<next ARP>]

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution

(PC) + 1 → PC
(dma) → P register(31-16)

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	1	0	0	1	1	0	Data Memory Address						
Indirect:	0	1	0	1	0	0	1	1	1	See Section 4.1						

Description

The P register high-order bits are loaded with the contents of data memory. The low-order P register bits are unaffected.

The LPH instruction is particularly useful for restoring the high-order bits of the P register after subroutine calls or interrupts.

Words 1

Cycles

	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
Cycle Timings for a Single Instruction						
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
Cycle Timings for a Repeat Execution						
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
LPH    DAT0    (DP = 4)
or
LPH    *        If current auxiliary register contains 512.
```

	Before Instruction		After Instruction
Data Memory 512	>F79C	Data Memory 512	>F79C
P	>30079844	P	>F79C9844

LRLK Load Auxiliary Register Long Immediate LRLK

Syntax [<label>] LRLK <AR>,<constant>

Operands 0 ≤ auxiliary register ≤ 7
0 ≤ constant ≤ 65535

Execution (PC) + 2 → PC
Constant → AR

Not affected by SXM; does not affect SXM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	0	AR			0	0	0	0	0	0	0	0
16-Bit Constant															

Description The 16-bit immediate value is loaded into the auxiliary register specified by the AR field. The specified constant must be an unsigned integer, and its value is not affected by SXM.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	2	2	2+2p	2+2p	–	–
'C25	2	2	2+2p	2+2p	2	2
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example `LRLK AR3,>3080`

	Before Instruction		After Instruction
AR3	>7F80	AR3	>3080

Syntax

Direct: [<label>] LST <dma>
Indirect: [<label>] LST {ind}[,<next ARP>]

Operands $0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution (PC) + 1 → PC
(dma) → status register ST0

Affects ARP, OV, OVM, and DP.
Does not affect INTM or ARB.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	1	0	0	0	0	0	Data Memory Address						
Indirect:	0	1	0	1	0	0	0	0	1	See Section 4.1						

Description Status register ST0 is loaded with the addressed data memory value. Note that the INTM (interrupt mode) bit is unaffected by LST. ARB is also unaffected even though a new ARP is loaded. If a next ARP value is specified via the indirect addressing mode, the specified value is ignored. Instead, ARP is loaded with the value contained within the addressed data memory word.

The LST instruction is used to load status register ST0 after interrupts and subroutine calls. The ST0 contains the status bits: OV (overflow flag) bit, OVM (overflow mode) bit, INTM (interrupt mode) bit, ARP (auxiliary register pointer), and DP (data memory page pointer). These bits were stored (by the SST instruction) in the data memory word as follows:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARP			OV	OVM	1	INTM	DP								

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	2n+nd	n+p	2n+nd+p	n	2n+nd

Example 1

```
LARP    0
LST     *,1     The data memory word addressed by the
                contents of auxiliary register AR0 is
                loaded into status register ST0, except
                for the INTM bit.  Note that even though
                a next ARP value is specified, that value
                is ignored, and even though a new ARP is
                loaded, the old ARP is not loaded into ARB.
```

Example 2

```
LST    >60      (DP = 0)
```

	Before Instruction	After Instruction
Data Memory 96	>2404	>2404
ST0	>6E00	>2604
ST1	>0580	>0580

Example 3

```
LARP   AR4      (AR4 = >3FF)
LST    *-
```

	Before Instruction	After Instruction
AR4	>3FF	>3FE
Data Memory 1023	>CE06	>CE06
ST0	>FC04	>CC06
ST1	>E780	>E780

Example 4

```
LARP   AR4      (AR4 = >3FF)
LST    *-,1
```

	Before Instruction	After Instruction
AR4	>3FF	>3FE
Data Memory 1023	>EE04	>EE04
ST0	>EE00	>EE04
ST1	>F780	>F780

LST1 Load Status Register ST1 LST1

Syntax

Direct: [<label>] LST1 <dma>
Indirect: [<label>] LST1 {ind}[,<next ARP>]

Operands

$0 \le \text{dma} \le 127$
$0 \le \text{next ARP} \le 7$

Execution

(PC) + 1 → PC
(dma) → status register ST1
(ARB) → ARP

Affects ARP, ARB, CNF, TC, SXM, XF, FO, TXM, and PM.
Affects C, HM, and FSM (TMS320C25)

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	1	0	0	0	1	0	Data Memory Address						
Indirect:	0	1	0	1	0	0	0	1	1	See Section 4.1						

Description

Status register ST1 is loaded with the data memory value. The bits of the data memory value, which are loaded into ARB, are also loaded into ARP to facilitate context switching. Note that if a next ARP value is specified via the indirect addressing mode, the specified value is ignored.

LST1 is used to load status bits after interrupts and subroutine calls. ST1 contains the status bits: ARB (auxiliary register pointer buffer), CNF (RAM configuration control) bit, TC (test/control) bit, SXM (sign-extension mode) bit, XF (external flag) bit, FO (serial port format) bit, TXM (transmit mode) bit, and the PM (product register shift mode) bit. ST1 on the TMS320C25 also contains the status bits: C (carry) bit, HM (hold mode) bit, and FSM (frame synchronization mode) bit. On the TMS32020, bits 5, 6, and 9 are one's. The bits loaded into status register ST1 from the data memory word are as follows:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARB			CNF	TC	SXM	C†	1	1	HM†	FSM†	XF	FO	TXM	PM	

†On the TMS32020, bits 5, 6, and 9 are one's.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	2n+nd	n+p	2n+nd+p	n	2n+nd

Example 1

```
LARP    3
LST1    *-      The data memory word addressed by the
                contents of auxiliary register AR3
                replaces the status bits of status
                register ST1, and AR3 is decremented.
```

Example 2

```
LST1    >61     (DP = 0)
```

	Before Instruction		After Instruction
Data Memory 97	>0580	Data Memory 97	>0580
ST0	>AC00	ST0	>0C00
ST1	>0581	ST1	>0580

Example 3

```
LARP    AR4     (AR4 = >3FE)
LST1    *-
```

	Before Instruction		After Instruction
AR4	>3FE	AR4	>3FD
Data Memory 1022	>4F90	Data Memory 1022	>4F90
ST0	>FC04	ST0	>5C04
ST1	>E780	ST1	>4F90

Example 4

```
LARP    AR4     (AR4 = >3FE)
LST1    *-,1
```

	Before Instruction		After Instruction
AR4	>3FE	AR4	>3FD
Data Memory 1022	>6190	Data Memory 1022	>6190
ST0	>FE04	ST0	>7E04
ST1	>0593	ST1	>6190

LT Load T Register LT

Syntax

Direct: [<label>] LT <dma>
Indirect: [<label>] LT {ind}[,<next ARP>]

Operands

0 ≤ dma ≤ 127
0 ≤ next ARP ≤ 7

Execution

(PC) + 1 → PC
(dma) → T register

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	0	1	1	1	1	0	0	0	Data Memory Address						
Indirect:	0	0	1	1	1	1	0	0	1	See Section 4.1						

Description

The T register is loaded with the contents of the specified data memory address (dma). The LT instruction may be used to load the T register in preparation for multiplication. See the LTA, LTD, LTP, LTS, MPY, MPYK, MPYA, MPYS, and MPYU instructions.

Words

1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
LT      DAT24      (DP = 8)
or
LT      *          If current auxiliary register contains 1048.
```

	Before Instruction		After Instruction
Data Memory 1048	>62	Data Memory 1048	>62
T	>3	T	>62

LTA Load T Register and Accumulate Previous Product LTA

Syntax

Direct: [<label>] LTA <dma>
Indirect: [<label>] LTA {ind}[,<next ARP>]

Operands

0 ≤ dma ≤ 127
0 ≤ next ARP ≤ 7

Execution

(PC) + 1 → PC
(dma) → T register
(ACC) + (shifted P register) → ACC

Affects OV; affected by OVM and PM.
Affects C (TMS320C25).

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	0	1	1	1	1	0	1	0	Data Memory Address						
Indirect:	0	0	1	1	1	1	0	1	1	See Section 4.1						

Description

The T register is loaded with the contents of the specified data memory address (dma). The contents of the product register, shifted as defined by the PM status bits, are added to the accumulator, with the result left in the accumulator.

The function of the LTA instruction is included in the LTD instruction.

Words

1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
LTA   DAT36      (DP = 6, PM = 0)
or
LTA   *          If current auxiliary register contains 804.
```

LTA Load T Register and Accumulate Previous Product LTA

	C	Before Instruction	C	After Instruction
Data Memory 804		>62		>62
T		>3		>62
P		>F		>F
ACC	X	>5	0	>14

LTD Load T Register, Accumulate Previous Product, and Move Data LTD

Syntax

Direct: [<label>] LTD <dma>
Indirect: [<label>] LTD {ind}[,<next ARP>]

Operands

0 ≤ dma ≤ 127
0 ≤ next ARP ≤ 7

Execution

(PC) + 1 → PC
(dma) → T register
(dma) → dma + 1
(ACC) + (shifted P register) → ACC

Affects OV; affected by OVM and PM.
Affects C (TMS320C25).

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	0	1	1	1	1	1	1	0	Data Memory Address						
Indirect:	0	0	1	1	1	1	1	1	1	See Section 4.1						

Description

The T register is loaded with the contents of the specified data memory address (dma). The contents of the P register, shifted as defined by the PM status bits, are added to the accumulator, and the result is placed in the accumulator. The contents of the specified data memory address are also copied to the next higher data memory address.

This instruction is valid for blocks B1 and B2, and is also valid for block B0 if block B0 is configured as data memory. The data move function is continuous across the boundary of blocks B0 and B1, but cannot be used with external data memory or memory-mapped registers. This function is described under the instruction DMOV. Note that if used with external data memory, the function of LTD is identical to that of LTA.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
LTD    DAT126   (DP = 7, PM = 0)
or
LTD    *        If current auxiliary register contains 1022.
```

	Before Instruction	After Instruction
Data Memory 1022	>62	>62
Data Memory 1023	>0	>62
T	>3	>62
P	>F	>F
ACC	>5	>14
C	X	0

LTP Load T Register and Store P Register in Accumulator LTP

Syntax

Direct: [<label>] LTP <dma>
Indirect: [<label>] LTP {ind}[,<next ARP>]

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution

(PC) + 1 → PC
(dma) → T register
(shifted P register) → ACC

Affected by PM.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	0	1	1	1	1	1	0	0	Data Memory Address						
Indirect:	0	0	1	1	1	1	1	0	1	See Section 4.1						

Description

The T register is loaded with the contents of the addressed data memory location, and the product register is stored in the accumulator. The shift at the output of the product register is controlled by the PM status bits.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
LTP    DAT36     (DP = 6, PM = 0)
or
LTP    *         If current auxiliary register contains 804.
```

	Before Instruction			After Instruction
Data Memory 804	>62		Data Memory 804	>62
T	>3		T	>62
P	>F		P	>F
ACC (C: X)	>5		ACC (C: X)	>F

LTS Load T Register, Subtract Previous Product LTS

Syntax

Direct: [<label>] LTS <dma>
Indirect: [<label>] LTS {ind}[,<next ARP>]

Operands $0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution (PC) + 1 → PC
(dma) → T register
(ACC) - (shifted P register) → ACC

Affects OV; affected by PM and OVM.
Affects C (TMS320C25).

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	1	1	0	1	1	0	Data Memory Address						
Indirect:	0	1	0	1	1	0	1	1	1	See Section 4.1						

Description The T register is loaded with the contents of the addressed data memory location. The contents of the product register, shifted as defined by the contents of the PM status bits, are subtracted from the accumulator. The result is left in the accumulator.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
LTS    DAT36     (DP = 6, PM = 0)
or
LTS    *         If current auxiliary register contains 804.
```

	Before Instruction		After Instruction
Data Memory 804	>62	Data Memory 804	>62
T	>3	T	>62
P	>F	P	>F
ACC	X C >5	ACC	0 C >FFFFFFF6

MAC Multiply and Accumulate MAC

Syntax

Direct: [<label>] MAC <dma>,<pma>
Indirect: [<label>] MAC <pma>,{ind}[,<next ARP>]

Operands

$0 \leq \text{pma} \leq 65535$
$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution

TMS32020:

```
(PC) + 2 → TOS
(pma) → PC

If (repeat counter) ≠ 0:
  Then (ACC) + (shifted P register) → ACC,
    (dma) → T register,
    (dma) x (pma, addressed by PC) → P register,
    Modify AR(ARP) and ARP as specified,
    (PC) + 1 → PC,
    (repeat counter) - 1 → repeat counter.
  Else (ACC) + (shifted P register) → ACC
    (dma) → T register
    (dma) x (pma, addressed by PC) → P register
    Modify AR(ARP) and ARP as specified.
(TOS) → PC
```

Affects OV; affected by OVM and PM.

TMS320C25:

```
(PC) + 2 → PC
(PFC) → MCS
(pma) → PFC

If (repeat counter) ≠ 0:
  Then (ACC) + (shifted P register) → ACC,
    (dma) → T register,
    (dma) x (pma, addressed by PFC) → P register,
    Modify AR(ARP) and ARP as specified,
    (PFC) + 1 → PFC,
    (repeat counter) - 1 → repeat counter.
  Else (ACC) + (shifted P register) → ACC
    (dma) → T register
    (dma) x (pma, addressed by PFC) → P register
    Modify AR(ARP) and ARP as specified.
(MCS) → PFC
```

Affects C and OV; affected by OVM and PM.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	1	1	1	0	1	0	Data Memory Address						
	Program Memory Address															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Indirect:	0	1	0	1	1	1	0	1	1	See Section 4.1						
	Program Memory Address															

Description The MAC instruction multiplies a data memory value (specified by dma) by a program memory value (specified by pma). It also adds the previous product, shifted as defined by the PM status bits, to the accumulator. The data and program memory locations on the TMS320C25 may be any non-reserved, on-chip or off-chip memory locations. If the program memory is block B0 of on-chip RAM, then the CNF bit must be set to one. On the TMS32020, data and program memory locations must reside on-chip. Note that on both devices, the upper eight bits of the program memory address should be set to >FF in order to address B0 program RAM, and the upper six bits of dma should be set to 0 to address a location below 1024. When used in the direct addressing mode, the dma cannot be modified during repetition of the instruction.

When the MAC instruction is repeated, the program memory address contained in the PC/PFC is incremented by one during its operation. This enables accessing a series of operands in memory. MAC is useful for long sum-of-products operations, since MAC becomes a single-cycle instruction once the RPT pipeline is started.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	3	N/A	3+2p	N/A	–	–
'C25	Table in on-chip RAM:					
	3	4+d	4+2p	5+d+2p	4	5+d
	Table in on-chip ROM:					
	4	5+d	4+2p	5+d+2p	4	5+d
	Table in external memory:					
	4+p	5+d+p	4+3p	5+d+3p	4+p	5+d+p
	Cycle Timings for a Repeat Execution					
'20	2+n	N/A	2+n+2p	N/A	–	–
'C25	Table in on-chip RAM:					
	2+n	2+2n+nd	3+n+2p	3+2n+nd+2p	3+n	3+2n+nd
	Table in on-chip ROM:					
	3+n	3+2n+nd	3+n+2p	3+2n+nd+2p	3+n	3+2n+nd
	Table in external memory:					
	3+n+np	3+2n+nd +np	3+n+np +2p	3+2n+nd+np +2p	3+n+np	3+2n+nd +np

Example

```
SPM   3         Select a shift-right-by-6 mode on PR output.
CNFP            Config block B0 as program memory (>FFXX).
LARP  1         Use AR1 to address block B1.
LRLK  1,768     Point to lowest location in RAM block B1.
RPTK  255       Compute 256 sum-of-product operations.
MAC   >FF00,*+  Multiply/accumulate and increment AR1.
```

The following example shows register and memory contents before and after the third step repeat loop:

		Before Instruction			After Instruction
AR1		>302	AR1		>303
RPT		>FD	RPT		>FC
PC/PFC		>FF02	PC/PFC		>FF03
Data Memory 770		>23	Data Memory 770		>23
Program Memory 65282		>FAAA	Program Memory 65282		>FAAA
P		>458972	P		>FFFF453E
ACC	X	>723EC41	ACC	0	>7250266
	C			C	

MACD Multiply and Accumulate with Data Move MACD

Syntax

Direct: [<label>] MACD <dma>,<pma>
Indirect: [<label>] MACD <pma>,{ind}[,<next ARP>]

Operands

0 ≤ pma ≤ 65535
0 ≤ dma ≤ 127
0 ≤ next ARP ≤ 7

Execution

TMS32020:

```
(PC) + 2 → TOS
(pma) → PC

If (repeat counter) ≠ 0:
  Then (ACC) + (shifted P register) → ACC,
   (dma) → T register,
   (dma) x (pma, addressed by PC) → P register,
   Modify AR(ARP) and ARP as specified,
   (PC) + 1 → PC,
   (repeat counter) - 1 → repeat counter.

  Else (ACC) + (shifted P register) → ACC
   (dma) → T register
   (dma) x (pma, addressed by PC) → P register
   Modify AR(ARP) and ARP as specified.
(TOS) → PC
```

Affects OV; affected by OVM and PM.

TMS320C25:

```
(PC) + 2 → PC
(PFC) → MCS
(pma) → PFC

If (repeat counter) ≠ 0:
  Then (ACC) + (shifted P register) → ACC,
   (dma) → T register,
   (dma) x (pma, addressed by PFC) → P register,
   Modify AR(ARP) and ARP as specified,
   (PFC) + 1 → PFC,
   (repeat counter) - 1 → repeat counter.

  Else (ACC) + (shifted P register) → ACC
   (dma) → T register
   (dma) x (pma, addressed by PFC) → P register
   Modify AR(ARP) and ARP as specified.
(MCS) → PFC
```

Affects C and OV; affected by OVM and PM.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	1	1	1	0	0	0	Data Memory Address						
	Program Memory Address															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Indirect:	0	1	0	1	1	1	0	0	1	See Section 4.1						
	Program Memory Address															

MACD Multiply and Accumulate with Data Move MACD

Description The MACD instruction multiplies a data memory value (specified by dma) by a program memory value (specified by pma). It also adds the previous product, shifted as defined by the PM status bits, to the accumulator. The data and program memory locations on the TMS320C25 may be any non-reserved, on-chip or off-chip memory locations. If the program memory is block B0 of on-chip RAM, then the CNF bit must be set to one. On the TMS32020, data and program memory locations must reside on-chip. Note that on both devices, the upper eight bits of the program memory address should be set to >FF in order to address B0 program RAM, and the upper six bits of dma should be set to 0 to address a location below 1024. When used in the direct addressing mode, the dma cannot be modified during repetition of the instruction. If MACD addresses one of the memory-mapped registers or external memory as a data memory location, the effect of the instruction will be that of a MAC instruction (see the DMOV instruction description).

MACD functions in the same manner as MAC, with the addition of data move for block B0, B1, or B2. Otherwise, the effects are the same as for MAC. This feature makes MACD useful for applications such as convolution and transversal filtering.

When the MACD instruction is repeated, the program memory address contained in the PC/PFC is incremented by one during its operation. This enables accessing a series of operands in memory. When used with RPT or RPTK, MACD becomes a single-cycle instruction once the RPT pipeline is started.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	3	N/A	3+2p	N/A	–	–
'C25	Table in on-chip RAM:					
	3	4+d	4+2p	5+d+2p	4	5+d
	Table in on-chip ROM:					
	4	5+d	4+2p	5+d+2p	4	5+d
	Table in external memory:					
	4+p	5+d+p	4+3p	5+d+3p	4+p	5+d+p
	Cycle Timings for a Repeat Execution					
'20	2+n	N/A	2+n+2p	N/A	–	–
'C25	Table in on-chip RAM:					
	2+n	2+2n+nd	3+n+2p	3+2n+nd+2p	3+n	3+2n+nd
	Table in on-chip ROM:					
	3+n	3+2n+nd	3+n+2p	3+2n+nd+2p	3+n	3+2n+nd
	Table in external memory:					
	3+n+np	3+2n+nd +np	3+n+np +2p	3+2n+nd+np +2p	3+n+np	3+2n+nd +np

Example

```
SPM    0          Select no shift mode on PR output.
SOVM              Set overflow mode.
CNFP            Config block B0 as program memory (>FFXX).
LARP   3          Use AR3 to address block B1.
LRLK   3,1023     Point to highest location in RAM block B1.
RPTK   255        Compute 1 sample of a length-256
                  convolution.
MACD   >FF00,*-   Multiply/accumulate, shift data word in
                  block B1, and decrement AR3.
```

The following example shows register and memory contents before and after the third step repeat loop:

	Before Instruction		After Instruction
AR1	>3FD	AR1	>3FC
RPT	>FD	RPT	>FC
PC/PFC	>FF02	PC/PFC	>FF03
Data Memory 1021	>23	Data Memory 1021	>23
Data Memory 1022	>7FC	Data Memory 1022	>23
Program Memory 65282	>FAAA	Program Memory 65282	>FAAA
P	>458972	P	>FFFF453E
ACC	>723EC41	ACC	>76975B3
C	X	C	0

Note:

The data move function for MACD can only occur within on-chip data RAM blocks B0, B1, and B2.

MAR Modify Auxiliary Register MAR

Syntax

Direct: [<label>] MAR <dma>
Indirect: [<label>] MAR {ind}[,<next ARP>]

Operands $0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution (PC) + 1 → PC
Modifies ARP, AR(ARP) as specified by the indirect addressing field (acts as a NOP in direct addressing).

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	1	0	1	0	1	0	Data Memory Address						
Indirect:	0	1	0	1	0	1	0	1	1	See Section 4.1						

Description The MAR instruction acts as a no-operation instruction in the direct addressing mode. In the indirect addressing mode, the auxiliary registers and the ARP are modified; however, no use is made of the memory being referenced. MAR is used only to modify the auxiliary registers or the ARP. The old ARP is copied to the ARB field of status register ST1. Note that any operation that MAR performs can also be performed with any instruction that supports indirect addressing. ARP may also be loaded by an LST instruction.

In the direct addressing mode, MAR is a NOP. Also, the instruction LARP is a subset of MAR (i.e., MAR *,4 performs the same function as LARP 4).

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example 1 `MAR *,1 Load the ARP with 1`

	Before Instruction		After Instruction
ARP	0	ARP	1

Example 2 `MAR *- Decrement current auxiliary register (in this case, AR1)`

	Before Instruction		After Instruction
AR1	>35	AR1	>34

Example 3

```
MAR   *+,5  Increment current auxiliary register (AR1) and
            load ARP with 5.
```

	Before Instruction		After Instruction
AR1	>34	AR1	>35
ARP	1	ARP	5

Syntax

Direct: [<label>] MPY <dma>
Indirect: [<label>] MPY {ind}[,<next ARP>]

Operands

$0 \le \text{dma} \le 127$
$0 \le \text{next ARP} \le 7$

Execution

(PC) + 1 → PC
(T register) x (dma) → P register

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	0	1	1	1	0	0	0	0	Data Memory Address						
Indirect:	0	0	1	1	1	0	0	0	1	See Section 4.1						

Description

The contents of the T register are multiplied by the contents of the addressed data memory location. The result is placed in the P register.

Words

1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
MPY    DAT13    (DP = 8)
or
MPY    *        If current auxiliary register contains 1037.
```

	Before Instruction		After Instruction
Data Memory 1037	>7	Data Memory 1037	>7
T	>6	T	>6
P	>36	P	>2A

MPYA Multiply and Accumulate Previous Product (TMS320C25) MPYA

Syntax

Direct: [<label>] MPYA <dma>
Indirect: [<label>] MPYA {ind}[,<next ARP>]

Operands 0 ≤ dma ≤ 127
0 ≤ next ARP ≤ 7

Execution (PC) + 1 → PC
(ACC) + (shifted P register) → ACC
(T register) x (dma) → P register

Affects C and OV; affected by OVM and PM.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	0	1	1	1	0	1	0	0	Data Memory Address						
Indirect:	0	0	1	1	1	0	1	0	1	See Section 4.1						

Description The contents of the T register are multiplied by the contents of the addressed data memory location. The result is placed in the P register. The previous product, shifted as defined by the PM status bits, is also added to the accumulator.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
MPYA   DAT13   (DP = 6, PM = 0)
```
or
```
MPYA   *       If current auxiliary register contains 781.
```

	Before Instruction	After Instruction
Data Memory 781	>7	>7
T	>6	>6
P	>36	>2A
ACC	>54	>8A
C	X	0

Syntax [<label>] MPYK <constant>

Operands $-4096 \le \text{constant} \le 4095$
$-2^{12} \le \text{constant} \le 2^{12} - 1$

Execution (PC) + 1 → PC
(T register) x constant → P register

Not affected by SXM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	13-Bit Constant												

Description The contents of the T register are multiplied by the signed, 13-bit constant. The result is loaded into the P register. The immediate field is right-justified and sign-extended before multiplication, regardless of SXM.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example

```
MPYK    -9
```

	Before Instruction		After Instruction
T	>7	T	>7
P	>2A	P	>FFFFFFC1

MPYS Multiply and Subtract Previous Product (TMS320C25) MPYS

Syntax

Direct: [<label>] MPYS <dma>
Indirect: [<label>] MPYS {ind}[,<next ARP>]

Operands 0 ≤ dma ≤ 127
0 ≤ next ARP ≤ 7

Execution (PC) + 1 → PC
(ACC) - (shifted P register) → ACC
(T register) x (dma) → P register

Affects C and OV; affected by OVM and PM.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	0	1	1	1	0	1	1	0	Data Memory Address						
Indirect:	0	0	1	1	1	0	1	1	1	See Section 4.1						

Description The contents of the T register are multiplied by the contents of the addressed data memory location. The result is placed in the P register. The previous product, shifted as defined by the PM status bits, is also subtracted from the accumulator.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
MPYS   DAT13   (DP = 6, PM = 0)
or
MPYS   *       If current auxiliary register contains 781.
```

		Before Instruction			After Instruction
Data Memory 781		>7	Data Memory 781		>7
T		>6	T		>6
P		>36	P		>2A
ACC	X	>54	ACC	1	>1E
	C			C	

MPYU Multiply Unsigned (TMS320C25) MPYU

Syntax

Direct: [<label>] MPYU <dma>
Indirect: [<label>] MPYU {ind}[,<next ARP>]

Operands

0 ≤ dma ≤ 127
0 ≤ next ARP ≤ 7

Execution

(PC) + 1 → PC
Unsigned (T register) x unsigned (dma) → P register

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	1	1	0	0	1	1	1	1	0	Data Memory Address						
Indirect:	1	1	0	0	1	1	1	1	1	See Section 4.1						

Description

The unsigned contents of the T register are multiplied by the unsigned contents of the addressed data memory location. The result is placed in the P register. Note that the multiplier acts as a 17 x 17-bit signed multiplier for this instruction, with the MSB of both operands forced to zero.

The shifter at the output of the P register will always invoke sign-extension on the P register when PM = 3 (right-shift by 6 mode). Therefore, this shift mode should not be used if unsigned products are desired.

The MPYU instruction is particularly useful for computing multiple-precision products, such as when multiplying two 32-bit numbers to yield a 64-bit product.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
MPYU  DAT16   (DP = 4)
```
or
```
MPYU  *       If current auxiliary register contains 528.
```

	Before Instruction		After Instruction
Data Memory 528	>FFFF	Data Memory 528	>FFFF
T	>FFFF	T	>FFFF
P	>1	P	>FFFE0001

NEG Negate Accumulator NEG

Syntax [<label>] NEG

Operands None

Execution (PC) + 1 → PC
(ACC) x -1 → ACC

Affects OV; affected by OVM.
Affects C (TMS320C25).

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	1	0	0	0	1	1

Description The contents of the accumulator are replaced with its arithmetic complement (two's complement). The OV bit is set when taking the NEG of >80000000. If OVM = 1, the accumulator contents are replaced with >7FFFFFFF. If OVM = 0, the result is >80000000. The carry bit C on the TMS320C25 is reset to zero by this instruction for all nonzero values of the accumulator, and set to one if the accumulator equals zero.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example `NEG`

		Before Instruction			After Instruction
ACC	X	>FFFFF228	ACC	0	>DD8
	C			C	

Syntax [<label>] NOP

Operands None

Execution (PC) + 1 → PC

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0

Description No operation is performed. The NOP instruction affects only the PC. NOP functions in the same manner as the MAR instruction in the direct addressing mode; NOP has the same opcode as MAR in the direct addressing mode with dma = 0.

The NOP instruction is useful as a pad or temporary instruction during program development.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example

```
NOP
```

NORM Normalize Contents of Accumulator NORM

Syntax

```
[<label>] NORM        (TMS32020)
[<label>] NORM {ind}  (TMS320C25)
```

Operands None

Execution

TMS32020:

```
(PC) + 1 → PC
If (ACC(31)).XOR.(ACC(30)) = 0:
  Then TC → 0,
    (ACC) x 2 → ACC,
    Modify AR(ARP) as specified;
  Else TC → 1.
```

Affects TC; affected by TC.

TMS320C25:

```
(PC) + 1 → PC
If (ACC) = 0:
  Then TC → 1;
  Else, if (ACC(31)).XOR.(ACC(30)) = 0:
    Then TC → 0,
      (ACC) x 2 → ACC,
      Modify AR(ARP) as specified;
    Else TC → 1.
```

Affects TC; affected by TC.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	1	Modify AR			0	0	1	0

Description The NORM instruction is provided for normalizing a signed number that is contained in the accumulator. Normalizing a fixed-point number separates it into a mantissa and an exponent. To do this, the magnitude of a sign-extended number must be found. ACC bit 31 is exclusive-ORed with ACC bit 30 to determine if bit 30 is part of the magnitude or part of the sign extension. If they are the same, they are both sign bits, and the accumulator is left-shifted to eliminate the extra sign bit. The AR(ARP) is modified as specified to generate the magnitude of the exponent. It is assumed that AR(ARP) is initialized before the normalization begins. The default modification of the AR(ARP) is an increment.

Multiple executions of the NORM instruction may be required to completely normalize a 32-bit number in the accumulator. Although using NORM with RPT or RPTK does not cause execution of NORM to "fall out" of the repeat loop automatically when the normalization is complete, no operation is performed for the remainder of the repeat loop. Note that NORM functions on both positive and negative two's-complement numbers.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example 1 31-Bit Normalization:

```
      LARP   1      Use AR1 for exponent storage.
      LARK   1,0    Clear out exponent counter.
LOOP  NORM   *+     One bit is normalized.
      BBZ    LOOP   If TC = 0, magnitude not found yet.
```

Example 2 15-Bit Normalization:

```
    LARP   1      Use AR1 to store the exponent.
    LARK   1,15   Initialize exponent counter.
    RPTK   14     15-bit normalization is specified
                  (yielding a 4-bit exponent and 16-bit
                  mantissa).
    NORM   *-     NORM automatically stops shifting
                  when the first significant magnitude
                  bit is found, performing NOPs for the
                  remainder of the repeat loop.
```

The first method is used to normalize a 32-bit number and yields a 5-bit exponent magnitude. The second method is used to normalize a 16-bit number and yields a 4-bit exponent magnitude. If the number requires only a small amount of normalization, the first method may be preferable to the second. This results because Example 1 runs only until normalization is complete. Example 2 always executes all 15 cycles of the repeat loop. Specifically, Example 1 is more efficient if the number requires five or less shifts. If the number requires six or more shifts, Example 2 is more efficient.

Note:

The TMS32020 accepts only the NORM instruction (no operand). Source code compatibility of the TMS320C25 with the TMS32020 allows the NORM instruction to also be used without a specified operand. In that case, any comments on the same line as the instruction will be interpreted as the operand. If the first character is an asterisk (*), then the instruction will be assembled as NORM * with no auxiliary register modification taking place upon execution. The user is therefore advised to replace the NORM instructions with NORM *+ when the default modification of increment is desired.

The resulting value in the auxiliary register will not be the real exponent of the number for all modification options. However, it can always be used to obtain the exponent.

Syntax

Direct: [<label>] OR <dma>
Indirect: [<label>] OR {ind}[,<next ARP>]

Operands

0 ≤ dma ≤ 127
0 ≤ next ARP ≤ 7

Execution

(PC) + 1 → PC
(ACC(15-0)) .OR.dma → ACC(15-0)
(ACC(31-16)) → ACC(31-16)

Not affected by SXM.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	0	1	1	0	1	0	Data Memory Address						
Indirect:	0	1	0	0	1	1	0	1	1	See Section 4.1						

Description

The low-order bits of the accumulator are ORed with the contents of the addressed data memory location. The high-order bits of the accumulator are ORed with all zeroes. Therefore, the upper half of the accumulator is unaffected by this instruction.

Words

1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
OR   DAT8    (DP = 8)
```
or
```
OR   *       Where current auxiliary register contains 1032.
```

	C	Before Instruction	C	After Instruction
Data Memory 1032		>F000		>F000
ACC	X	>100002	X	>10F002

ORK OR Immediate with Accumulator with Shift ORK

Syntax [<label>] ORK <constant>[,<shift>]

Operands 16-bit constant
$0 \leq \text{shift} \leq 15$ (defaults to 0)

Execution (PC) + 2 → PC
(ACC(30-0)).OR.[constant x 2^{shift}] → ACC(30-0)
(ACC(31)) → ACC(31)

Not affected by SXM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	Shift				0	0	0	0	0	1	0	1
16-Bit Constant															

Description The left-shifted 16-bit immediate constant is ORed with the accumulator. The result is left in the accumulator. Low-order bits below and high-order bits above the shifted value are treated as zeroes. The corresponding bits of the accumulator are unaffected. Note that the most-significant bit of the accumulator is not affected, regardless of the shift code value.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	2	2	2+2p	2+2p	–	–
'C25	2	2	2+2p	2+2p	2	2
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example

```
ORK    >FFFF,8
```

		Before Instruction			After Instruction
ACC	X	>12345678	ACC	X	>12FFFF78
	C			C	

Syntax

Direct: [<label>] OUT <dma>,<PA>
Indirect: [<label>] OUT {ind},<PA>[,<next ARP>]

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$
$0 \leq \text{port address PA} \leq 15$

Execution

(PC) + 1 → PC
Port address PA → address bus A3-A0
0 → address bus A15-A4
(dma) → data bus D15-D0

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	1	1	1	0	Port Address				0	Data Memory Address						
Indirect:	1	1	1	0	Port Address				1	See Section 4.1						

Description

The OUT instruction writes a 16-bit value from a data memory location to the specified I/O port. The $\overline{\text{IS}}$ line goes low to indicate an I/O access, and the $\overline{\text{STRB}}$, R/$\overline{\text{W}}$, and READY timings are the same as for an external data memory write. OUT is a single-cycle instruction when in the PI/DI memory configuration (see Appendix E).

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1+i	2+d+i	2+p+i	3+d+p+i	–	–
'C25	1+i	2+d+i	2+p+i	3+d+p+i	1+i	2+d+i
	Cycle Timings for a Repeat Execution					
'20	n+ni	2n+nd+ni	2n+p+ni	3n+nd+p+ni	–	–
'C25	n+ni	2n+nd+ni	1+n+p+ni	1+2n+nd+p+ni	n+ni	2n+nd+ni

Example

```
OUT    >78,7     (DP = 4) Output data word stored in data
                 memory location >78 to peripheral on
                 port address 7.
OUT    *,>F      Output data word referenced by current
                 auxiliary register to peripheral on port
                 address >F.
```

PAC Load Accumulator with P Register PAC

Syntax [<label>] PAC

Operands None

Execution (PC) + 1 → PC
(shifted P register) → ACC

Affected by PM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	0	1	0	1	0	0

Description The contents of the P register are loaded into the accumulator, shifted as specified by the PM status bits.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example `PAC (PM = 0)`

	Before Instruction	After Instruction
P	>144	>144
ACC	>23	>144
C	X	X

POP Pop Top of Stack to Low Accumulator POP

Syntax [<label>] POP

Operands None

Execution
(PC) + 1 → PC
(TOS) → ACC(15-0)
0 → ACC(31-16)
Pop stack one level.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	0	1	1	1	0	1

Description The contents of the top of the stack (TOS) are copied to the low accumulator, and the stack popped after the contents are copied. The upper half of the accumulator is set to all zeros.

The hardware stack is a last-in, first-out stack with four (TMS32020) or eight (TMS320C25) locations. Any time a pop occurs, every stack value is copied to the next higher stack location, and the top value is removed from the stack. After a pop, the bottom two stack words will have the same value. Because each stack value is copied, if more than three/seven pops (due to POP, POPD, or RET instructions) occur before any pushes occur, all levels of the stack contain the same value. No provision exists to check stack underflow.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	2	2	2+p	2+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	2n	2n	2n+p	2n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example `POP`

	Before Instruction	After Instruction
ACC	>82	>45
C	X	X
Stack (20)	>45 >16 >7 >33	>16 >7 >33 >33
Stack (C25)	>45 >16 >7 >33 >42 >56 >37 >61	>16 >7 >33 >42 >56 >37 >61 >61

POPD Pop Top of Stack to Data Memory POPD

Syntax

Direct: [<label>] POPD <dma>
Indirect: [<label>] POPD {ind}[,<next ARP>]

Operands

0 ≤ dma ≤ 127
0 ≤ next ARP ≤ 7

Execution

(PC) + 1 → PC
(TOS) → dma
POP stack one level.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	1	1	1	0	1	0	0	Data Memory Address						
Indirect:	0	1	1	1	1	0	1	0	1	See Section 4.1						

Description

The value from the top of the stack is transferred into the data memory location specified by the instruction. The values are also popped in the lower three (TMS32020) or seven locations (TMS320C25) of the stack. The hardware stack is described in the previous instruction POP. The lowest stack location remains unaffected. No provision exists to check stack underflow.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	2	2+d	2+p	2+d+p	–	–
'C25	1	1+d	1+p	2+d+p	1	1+d
	Cycle Timings for a Repeat Execution					
'20	2n	2n+nd	2n+p	2n+nd+p	–	–
'C25	n	n+nd	n+p	1+n+nd+p	n	n+nd

Example

```
POPD   DAT100   (DP = 8)
```
or
```
POPD   *        If current auxiliary register contains 1124.
```

	Before Instruction		After Instruction
Data Memory 1124	>55	Data Memory 1124	>92
Stack (20)	>92 >72 >8 >44	Stack (20)	>72 >8 >44 >44
Stack (C25)	>92 >72 >8 >44 >81 >75 >32 >AA	Stack (C25)	>72 >8 >44 >81 >75 >32 >AA >AA

PSHD Push Data Memory Value onto Stack PSHD

Syntax

Direct: [<label>] PSHD <dma>
Indirect: [<label>] PSHD {ind}[,<next ARP>]

Operands $0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution (dma) → TOS
(PC) + 1 → PC
Push all stack locations down one level.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	1	0	1	0	0	0	Data Memory Address						
Indirect:	0	1	0	1	0	1	0	0	1	See Section 4.1						

Description The value from the data memory location specified by the instruction is transferred to the top of the stack. The values are also pushed down in the lower three (TMS32020) or seven locations (TMS320C25) of the stack, as described in the instruction PUSH. The lowest stack location is lost.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	2	2+d	2+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	2n	2n+nd	2n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
PSHD    DAT127   (DP = 3)
or
PSHD    *        If current auxiliary register contains 511.
```

	Before Instruction		After Instruction
Data Memory 511	>65	Data Memory 511	>65
Stack (20)	>2 >33 >78 >99	Stack (20)	>65 >2 >33 >78
Stack (C25)	>2 >33 >78 >99 >42 >50 >0 >0	Stack (C25)	>65 >2 >33 >78 >99 >42 >50 >0

PUSH Push Low Accumulator onto Stack PUSH

Syntax [<label>] PUSH

Operands None

Execution (PC) + 1 → PC
Push all stack locations down one level.
(ACC(15-0)) → TOS

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	0	1	1	1	0	0

Description The contents of the lower half of the accumulator are copied onto the top of the hardware stack. The stack is pushed down before the accumulator value is copied.

The hardware stack is a last-in, first-out stack with four (TMS32020) or eight locations (TMS320C25). If more than four/eight pushes (due to CALA, CALL, PSHD, PUSH, or TRAP instructions) occur before a pop, the first data values written will be lost with each succeeding push.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	2	2	2+p	2+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	2n	2n	2n+p	2n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example `PUSH`

	Before Instruction		After Instruction
ACC	C: X >7	ACC	C: X >7
Stack (20)	>2 >5 >3 >0	Stack (20)	>7 >2 >5 >3
Stack (C25)	>2 >5 >3 >0 >12 >86 >54 >3F	Stack (C25)	>7 >2 >5 >3 >0 >12 >86 >54

Syntax [<label>] RC

Operands None

Execution (PC) + 1 → PC
0 → carry bit C in status register ST1

Affects C.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	1	1	0	0	0	0

Description The carry bit C in status register ST1 is reset to logic zero. The carry bit may also be loaded directly by the LST1 and SC instructions.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'C25	n	n	n+p	n+p	n	n

Example

```
RC              The carry bit C is reset to logic zero.
```

RET Return from Subroutine RET

Syntax [<label>] RET

Operands None

Execution (TOS) → PC
Pop stack one level.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	1	0	0	1	1	0

Description The contents of the top stack register are copied into the program counter. The stack is then popped one level. RET is used with CALA and CALL for subroutines.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	2	2	2+p	2+p	–	–
'C25	Destination on-chip RAM:					
	2	2	2+p	2+p	2	2
	Destination on-chip ROM:					
	3	3	3+p	3+p	3	3
	Destination external memory:					
	3+p	3+p	3+2p	3+2p	3+p	3+p
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example `RET`

	Before Instruction		After Instruction
PC	>96	PC	>37
Stack (20)	>37 >45 >75 >21	Stack (20)	>45 >75 >21 >21
Stack (C25)	>37 >45 >75 >21 >3F >45 >6E >6E	Stack (C25)	>45 >75 >21 >3F >45 >6E >6E >6E

RFSM Reset Serial Port Frame Synchronization Mode (TMS320C25) RFSM

Syntax [<label>] RFSM

Operands None

Execution (PC) + 1 → PC
0 → FSM status bit in status register ST1

Affects FSM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	1	1	0	1	1	0

Description The RFSM status bit resets the FSM status bit to logic zero. In this mode, external FSR pulses are not required to initiate the receive operation for each word received, but rather only one FSR pulse is required to initiate a "continuous mode" of operation. The same holds true for FSX when TXM = 0. After the first FSR/FSX pulse, these inputs are then in a "don't care" state. If TXM = 1, FSX is pulsed the first time DXR is loaded, but remains low thereafter. See Section 3.9 for further details on the operation of the serial port. FSM may also be loaded by the LST1 and SFSM instructions.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'C25	n	n	n+p	n+p	n	n

Example

```
RFSM                FSM is reset, putting the serial port
                    in a mode of operation where frame
                    synchronization pulses are not required.
                    This allows a continuous bit stream to
                    be transmitted/received without FSX/FSR
                    pulses every 8/16 bits.
```

Syntax [<label>] RHM

Operands None

Execution (PC) + 1 → PC
0 → HM status bit in status register ST1

Affects HM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	1	1	1	0	0	0

Description The RHM instruction resets the HM status bit to logic zero. In this mode, the TMS320C2x is not halted during the assertion of $\overline{\text{HOLD}}$ when executing from on-chip program memory (either RAM or ROM), but instead places its external buses in the high-impedance state and continues execution until an external access must be made. External access can mean (in addition to the normal connotation) the following conditions:

MP/$\overline{\text{MC}}$	**CNF**	**PC**
0	0	PC 4096
0	1	$4096 \leq \text{PC} \leq 65279$
1	0	Any PC value (normal TMS32020-type hold mode)
1	1	$\text{PC} \leq 65279$

HM can also be loaded by the LST1 and SHM instructions.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'C25	n	n	n+p	n+p	n	n

Example

```
RHM          HM is reset, implementing the TMS320C25
             hold mode for on-chip program execution.
```

ROL Rotate Accumulator Left (TMS320C25) ROL

Syntax [<label>] ROL

Operands None

Execution

(PC) + 1 → PC
(ACC(31)) → C
(ACC(30-0)) → ACC(31-1)
(C, before ROL) → ACC(0)

Affects C.
Not affected by SXM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	1	1	0	1	0	0

Description The ROL instruction rotates the accumulator left one bit. The MSB is shifted into the carry bit, and the value of the carry bit from before the execution of the instruction is shifted into the LSB.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'C25	n	n	n+p	n+p	n	n

Example `ROL`

	Before Instruction		After Instruction
ACC	1	>B0001234	ACC	1	>60002469
	C			C	

ROR Rotate Accumulator Right (TMS320C25) ROR

Syntax [<label>] ROR

Operands None

Execution
(PC) + 1 → PC
(ACC(0)) → C
(ACC(31-1)) → ACC(30-0)
(C, before ROR) → ACC(31)

Affects C.
Not affected by SXM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	1	1	0	1	0	1

Description The ROR instruction rotates the accumulator right one bit. The LSB is shifted into the carry bit, and the value of the carry bit from before the execution of the instruction is shifted into the MSB.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'C25	n	n	n+p	n+p	n	n

Example `ROR`

	Before Instruction		After Instruction
ACC	0 >B0001234	ACC	0 >5800091A
	C		C

ROVM Reset Overflow Mode ROVM

Syntax [<label>] ROVM

Operands None

Execution (PC) + 1 → PC
0 → OVM status bit in status register ST0

Affects OVM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	0	0	0	0	1	0

Description The OVM status bit is reset to logic zero, which disables the overflow mode. If an overflow occurs with OVM reset, the OV (overflow flag) is set, and the overflowed result is placed in the accumulator.

OVM may also be loaded by the LST and SOVM instructions.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example `ROVM` The overflow mode bit OVM is reset, disabling the overflow mode on any subsequent arithmetic operations.

RPT Repeat Instruction as Specified by Data Memory Value RPT

Syntax

Direct: [<label>] RPT <dma>
Indirect: [<label>] RPT {ind}[,<next ARP>]

Operands $0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution $(\text{PC}) + 1 \rightarrow \text{PC}$
$(\text{dma}(7\text{-}0)) \rightarrow \text{RPTC}$

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	0	1	0	1	1	0	Data Memory Address						
Indirect:	0	1	0	0	1	0	1	1	1	See Section 4.1						

Description The eight LSBs of the addressed data memory value are loaded into the repeat counter (RPTC). This causes the following instruction to be executed one time more than the number loaded into the RPTC (provided that it is a repeatable instruction). Interrupts are masked out until the next instruction has been executed the specified number of times. (Interrupts cannot be allowed during the RPT/next instruction sequence, because the RPTC cannot be saved during a context switch.) The RPTC counter is cleared on a $\overline{\text{RS}}$.

RPT and RPTK are especially useful for repeating instructions, such as BLKP, BLKD, IN, MAC, MACD, NORM, OUT, TBLR, TBLW, and others.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example

```
RPT   DAT127   (DP = 31)
SFR
or
RPT   *        If current auxiliary register contains 4095.
SFR
```

	C	Before Instruction		C	After Instruction
Data Memory 4095		>C	Data Memory 4095		>C
ACC	X	>12345678	ACC	0	>12345

RPTK Repeat Instruction as Specified by Immediate Value RPTK

Syntax [<label>] RPTK <constant>

Operands 0 ≤ constant ≤ 255

Execution (PC) + 1 → PC
Constant → RPTC

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	0	1	1	8-Bit Constant							

Description The 8-bit immediate value is loaded into the RPTC (repeat counter). This causes the following instruction to be executed one time more than the number loaded into the RPTC (provided that it is a repeatable instruction). Interrupts are masked out until the next instruction has been executed the specified number of times. (Interrupts cannot be allowed during the RPT/next instruction sequence because the RPTC cannot be saved during a context switch.) The RPTC is cleared on a RS.

RPT and RPTK are especially useful for repeating instructions, such as BLKP, BLKD, IN, MAC, MACD, NORM, OUT, TBLR, TBLW, and others.

Words 1

Cycles

<table>
<tr><td></td><th colspan="6">Cycle Timings for a Single Instruction</th></tr>
<tr><td></td><th>PI/DI</th><th>PI/DE</th><th>PE/DI</th><th>PE/DE</th><th>PR/DI</th><th>PR/DE</th></tr>
<tr><td>'20</td><td>1</td><td>1</td><td>1+p</td><td>1+p</td><td>–</td><td>–</td></tr>
<tr><td>'C25</td><td>1</td><td>1</td><td>1+p</td><td>1+p</td><td>1</td><td>1</td></tr>
<tr><td></td><th colspan="6">Cycle Timings for a Repeat Execution</th></tr>
<tr><td>'20</td><td colspan="4">not repeatable</td><td>–</td><td>–</td></tr>
<tr><td>'C25</td><td colspan="6">not repeatable</td></tr>
</table>

Example

```
LRLK   AR2,>200    Load AR2 with the address of X.
LARP   2
ZAC                Clear the accumulator.
MPYK   0           Clear the P register.
RPTK   2           Repeat next instruction 3 times.
SQRA   *+          Compute X**2 + Y**2 + Z**2.
APAC
```

RSXM Reset Sign-Extension Mode RSXM

Syntax [<label>] RSXM

Operands None

Execution (PC) + 1 → PC
0 → SXM sign-extension mode status bit

Affects SXM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	0	0	0	1	1	0

Description The RSXM instruction resets the SXM status bit to logic zero, which suppresses sign-extension on shifted data memory values for the following arithmetic instructions: ADD, ADDT, ADLK, LAC, LACT, LALK, SBLK, SUB, and SUBT.

The RSXM instruction affects the definition of the SFR instruction. SXM may also be loaded by the LST1 and SSXM instructions.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example

```
RSXM             SXM is reset, disabling sign-extension on
                 subsequent instructions.
```

Syntax [<label>] RTC

Operands None

Execution (PC) + 1 → PC
0 → TC test/control flag in status register ST1

Affects TC.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	1	1	0	0	1	0

Description The TC (test/control) flag in status register ST1 is reset to logic zero. TC may also be loaded by the LST1 and STC instructions.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'C25	n	n	n+p	n+p	n	n

Example

```
RTC         TC (test/control) flag is reset to logic zero.
```

RTXM Reset Serial Port Transmit Mode RTXM

Syntax [<label>] RTXM

Operands None

Execution (PC) + 1 → PC
0 → TXM transmit mode status bit

Affects TXM mode bit.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	1	0	0	0	0	0

Description The RTXM instruction resets the TXM status bit, which configures the serial port transmit section in a mode where it is controlled by an FSX (external framing pulse). The transmit operation is started when an external FSX pulse is applied. TXM may also be loaded by the LST1 and STXM instructions.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example

```
RTXM          TXM is reset, configuring FSX as an input.
```

Syntax [<label>] RXF

Operands None

Execution (PC) + 1 → PC
0 → XF external flag pin and status bit

Affects XF.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	0	0	1	1	0	0

Description The XF pin and XF status bit in status register ST1 are reset to logic zero. XF may also be loaded by the LST1 and SXF instructions.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example

```
RXF          XF pin and status bit are reset to logic zero.
```

SACH Store High Accumulator with Shift SACH

Syntax

Direct: [<label>] SACH <dma>[,<shift>]
Indirect: [<label>] SACH {ind}[,<shift>[,<next ARP>]]

Operands

0 ≤ dma ≤ 127
0 ≤ next ARP ≤ 7
0 ≤ shift ≤ 0, 1, or 4 (defaults to 0) on the TMS32020
0 ≤ shift ≤ 7 (defaults to 0) on the TMS320C25

Execution

(PC) + 1 → PC
16 MSBs of (ACC) x 2^{shift} → dma

Not affected by SXM.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	1	0	1	Shift			0	Data Memory Address						
Indirect:	0	1	1	0	1	Shift			1	See Section 4.1						

Description

The SACH instruction copies the entire accumulator into a shifter, where it shifts the entire 32-bit number 0, 1, or 4 bits on the TMS32020, or anywhere from 0 to 7 bits on the TMS320C25.It then copies the upper 16 bits of the shifted value into data memory. The accumulator itself remains unaffected.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	2+d	1+p	3+d+p	–	–
'C25	1	1+d	1+p	2+d+p	1	1+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	3n+nd+p	–	–
'C25	n	n+nd	n+p	1+n+nd+p	n	n+nd

Example

```
SACH   DAT10,4   (DP = 4)
```
or
```
SACH   *,4       If current auxiliary register contains 522.
```

	C	Before Instruction	C	After Instruction
ACC	X	>4208001	X	>4208001
Data Memory 522		>0		>4208

Syntax

Direct: [<label>] SACL <dma>[,<shift>]
Indirect: [<label>] SACL {ind}[,<shift>[,<next ARP>]]

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$
$0 \leq \text{shift} \leq 0, 1, \text{or } 4$ (defaults to 0) on the TMS32020
$0 \leq \text{shift} \leq 7$ (defaults to 0) on the TMS320C25

Execution

(PC) + 1 → PC
16 LSBs of (ACC) x 2^{shift} → dma

Not affected by SXM.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	1	0	0	Shift			0	Data Memory Address						
Indirect:	0	1	1	0	0	Shift			1	See Section 4.1						

Description

The low-order bits of the accumulator are shifted left 0, 1, or 4 bits on the TMS32020 or anywhere from 0 to 7 bits on the TMS320C25, as specified by the shift code, and stored in data memory. The low-order bits are filled with zeros, and the high-order bits are lost. The accumulator itself is unaffected.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	2+d	1+p	3+d+p	–	–
'C25	1	1+d	1+p	2+d+p	1	1+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	3n+nd+p	–	–
'C25	n	n+nd	n+p	1+n+nd+p	n	n+nd

Example

```
SACL  DAT11,1  (DP = 4)
```
or
```
SACL  *,1      If current auxiliary register contains 523.
```

	Before Instruction		After Instruction	
	C	ACC	C	ACC
ACC	X	>7C638421	X	>7C638421
Data Memory 523		>5		>842

Syntax

Direct: [<label>] SAR <AR>,<dma>
Indirect: [<label>] SAR <AR>,{ind}[,<next ARP>]

Operands

0 ≤ dma ≤ 127
0 ≤ auxiliary register AR ≤ 7
0 ≤ next ARP ≤ 7

Execution

(PC) + 1 → PC
(AR) → dma

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	1	1	0	AR			0	Data Memory Address						
Indirect:	0	1	1	1	0	AR			1	See Section 4.1						

Description

The contents of the designated auxiliary register (AR) are stored in the addressed data memory location.

When modifying the contents of the current auxiliary register in the indirect addressing mode, SAR ARn (when n = ARP) stores the value of the auxiliary register contents before it is incremented, decremented, or indexed by AR0.

Words

1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	1	2+d	1+p	3+d+p	–	–
'C25	1	1+d	1+p	2+d+p	1	1+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	3n+nd+p	–	–
'C25	n	n+nd	n+p	1+n+nd+p	n	n+nd

Example 1

```
SAR AR0,DAT30   (DP = 6)
```
or
```
SAR AR0,*       If current auxiliary register contains 798.
```

	Before Instruction		After Instruction
AR0	>37	AR0	>37
Data Memory 798	>18	Data Memory 798	>37

Example 2

```
LARP   AR0
SAR    AR0,*0+
```

AR0	>401	AR0	>802
Data Memory 1025	>0	Data Memory 1025	>401

SBLK Subtract from Accumulator Long Immediate with Shift SBLK

Syntax [<label>] SBLK <constant>[,<shift>]

Operands 16-bit constant
$0 \leq \text{shift} \leq 15$ (defaults to 0)

Execution (PC) + 2 → PC
(ACC) - [constant x 2^{shift}] → ACC

If SXM = 1:
Then $-32768 \leq$ constant ≤ 32767.
If SXM = 0:
Then $0 \leq$ constant ≤ 65535.

Affects OV; affected by OVM and SXM.
Affects C (TMS320C25).

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	Shift				0	0	0	0	0	0	1	1
16-Bit Constant															

Description The immediate field of the instruction is subtracted from the accumulator. The result replaces the accumulator contents. SXM determines whether the constant is treated as a signed two's-complement number or as an unsigned number. The shift count is optional and defaults to zero.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	2	2	2+2p	2+2p	–	–
'C25	2	2	2+2p	2+2p	2	2
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example `SBLK 5,12`

	C	ACC
Before Instruction	X	>3FC0EF
After Instruction	1	>3F70EF

SBRK Subtract from Auxiliary Register Short Immediate (TMS320C25) SBRK

Syntax [<label>] SBRK <constant>

Operands 0 ≤ constant ≤ 255

Execution (PC) + 1 → PC
AR(ARP) - 8-bit positive constant → AR(ARP)

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1	8-Bit Constant							

Description The 8-bit immediate value is subtracted, right-justified, from the currently selected auxiliary register with the result replacing the auxiliary register contents. The subtraction takes place in the ARAU, with the immediate value treated as an 8-bit positive integer.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'C25	not repeatable					

Example `SBRK >FF (ARP = 7)`

	Before Instruction		After Instruction
AR7	>0	AR7	>FF01

SC Set Carry Bit (TMS320C25) SC

Syntax [<label>] SC

Operands None

Execution (PC) + 1 → PC
1 → carry bit C in status register ST1

Affects C.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	1	1	0	0	0	1

Description The carry bit C in status register ST1 is set to logic one. The carry bit may also be loaded directly by the LST1 and RC instructions.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'C25	n	n	n+p	n+p	n	n

Example

```
SC                  Carry bit C is set to logic one.
```

SFL Shift Accumulator Left SFL

Syntax [<label>] SFL

Operands None

Execution **TMS32020:**

(PC) + 1 → PC
(ACC(30-0)) → ACC(31-1)
0 → ACC(0)

Not affected by SXM bit.

TMS320C25:

(PC) + 1 → PC
(ACC(31)) → C
(ACC(30-0)) → ACC(31-1)
0 → ACC(0)

Affects C.
Not affected by SXM bit.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	0	1	1	0	0	0

Description The SFL instruction shifts the entire accumulator left one bit. The least-significant bit is filled with a zero. On the TMS32020, the most-significant bit is lost. On the TMS320C25, the most-significant bit is shifted into the carry bit (C). Note that SFL, unlike SFR, is unaffected by SXM.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example `SFL`

Before Instruction

ACC X >B0001234
C

After Instruction

ACC 1 >60002468
C

SFR Shift Accumulator Right SFR

Syntax [<label>] SFR

Operands None

Execution **TMS32020:**
(PC) + 1 → PC
If SXM = 0:
 Then (ACC(31-1)) → ACC (30-0) and 0 → ACC(31).
If SXM = 1:
 Then (ACC(31-1)) → ACC(30-0) and (ACC(31)) → ACC(31).

Affected by SXM bit.

TMS320C25:
(PC) + 1 → PC
If SXM = 0:
 Then (ACC(0)) → C
  (ACC(31-1)) → ACC (30-0) and 0 → ACC(31).
If SXM = 1:
 Then (ACC(0)) → C
  (ACC(31-1)) → ACC(30-0) and (ACC(31)) → ACC(31).

Affects C.
Affected by SXM bit.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	0	1	1	0	0	1

Description The SFR instruction shifts the accumulator right one bit. If SXM = 1, the instruction produces an arithmetic right shift. The sign bit (MSB) is unchanged and is also copied into bit 30. Bit 0 is shifted into the carry bit (C).

If SXM = 0, the instruction produces a logical right shift. All of the accumulator bits are shifted by one bit to the right. The least-significant bit is shifted into the carry bit, and the most-significant bit is filled with a zero.

On the TMS32020, note that bit 0 is lost.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example 1 `SFR` `(SXM = 0)`

	C	Before Instruction		C	After Instruction
ACC	X	>B0001234	ACC	0	>5800091A

Example 2 `SFR` `(SXM = 1)`

	C			C	
ACC	X	>B0001234	ACC	0	>D800091A

SFSM Set Serial Port Frame Synchronization Mode (TMS320C25) SFSM

Syntax [<label>] SFSM

Operands None

Execution (PC) + 1 → PC
1 → FSM status bit in status register ST1

Affects FSM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	1	1	0	1	1	1

Description The SFSM instruction sets the FSM status bit to logic one. In this mode, an external FSR pulse is required for a receive operation, and an external FSX pulse is required if TXM = 0. If TXM = 1, FSX pulses are generated in the normal manner every time the transmit shift register XSR is loaded. See Section 3.7 for details on the operation of the serial port. FSM may also be loaded by the LST1 and RFSM instructions.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'C25	n	n	n+p	n+p	n	n

Example

```
SFSM          FSM is set, putting the serial port in a
              mode of operation where frame synchronization
              pulses are required for each word to be
              transmitted or received.
```

Syntax .[<label>] SHM

Operands None

Execution (PC) + 1 → PC
1 → HM status bit in status register ST1

Affects HM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	1	1	1	0	0	1

Description The SHM instruction sets the HM status bit to logic one. In this mode, the TMS320C25 is halted in the normal manner whenever $\overline{\text{HOLD}}$ is asserted, regardless of the PC value or the state of the MP/$\overline{\text{MC}}$ pin. HM may also be loaded by the LST1 and RHM instructions.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'C25	n	n	n+p	n+p	n	n

Example

```
SHM                 HM is set, implementing the normal
                    (TMS32020-type) hold mode of operation.
```

SOVM Set Overflow Mode SOVM

Syntax [<label>] SOVM

Operands None

Execution (PC) + 1 → PC
1 → overflow mode (OVM) status bit

Affects OVM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	0	0	0	0	1	1

Description The OVM status bit is set to logic one, which enables the overflow (saturation) mode. If an overflow occurs with OVM set, the overflow flag OV is set, and the accumulator is set to the largest representable 32-bit positive (>7FFFFFFF) or negative (>80000000) number according to the direction of overflow.

OVM may also be loaded by the LST and ROVM instructions.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example

```
SOVM            The overflow mode bit OVM is set, enabling
                the overflow mode on any subsequent
                arithmetic operations.
```

SPAC Subtract P Register from Accumulator SPAC

Syntax [<label>] SPAC

Operands None

Execution (PC) + 1 → PC
(ACC) - (shifted P register) → ACC

Affects OV; affected by PM and OVM.
Affects C (TMS320C25).
Not affected by SXM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	0	1	0	1	1	0

Description The contents of the P register, shifted as defined by the PM status bits, are subtracted from the contents of the accumulator. The result is stored in the accumulator. Note that SPAC is unaffected by SXM; the P register is always sign-extended.

The SPAC instruction is a subset of LTS, MPYS, and SQRS.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example

```
SPAC          (PM = 0)
```

	C	Before Instruction	C	After Instruction
P		>24		>24
ACC	X	>3C	1	>18

SPH Store High P Register (TMS320C25) SPH

Syntax

Direct: [<label>] SPH <dma>
Indirect: [<label>] SPH {ind}[,<next ARP>]

Operands $0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution (PC) + 1 → PC
(PR shifter output (31-16)) → dma

Affected by PM.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	1	1	1	1	0	1	0	Data Memory Address						
Indirect:	0	1	1	1	1	1	0	1	1	See Section 4.1						

Description The high-order bits of the P register, shifted as specified by the PM bits, are stored in data memory. Neither the P register nor the accumulator are affected by this instruction. High-order bits are sign-extended when the right-shift by 6 mode is selected. Low-order bits are taken from the low P register when left-shifts are selected.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'C25	1	1+d	1+p	2+d+p	1	1+d
	Cycle Timings for a Repeat Execution					
'C25	n	n+nd	n+p	1+n+nd+p	n	n+nd

Example

```
SPH   DAT3    (DP = 4, PM = 2)
```
or
```
SPH   *       If current auxiliary register contains 515.
```

	Before Instruction		After Instruction
P	>FE079844	P	>FE079844
Data Memory 515	>4567	Data Memory 515	>E079

SPL Store Low P Register (TMS320C25) SPL

Syntax

Direct: [<label>] SPL <dma>
Indirect: [<label>] SPL {ind}[,<next ARP>]

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution

(PC) + 1 → PC
(PR shifter output (15-0)) → dma

Affected by PM.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	1	1	1	1	0	0	0	Data Memory Address						
Indirect:	0	1	1	1	1	1	0	0	1	See Section 4.1						

Description

The low-order bits of the P register, shifted as specified by the PM bits, are stored in data memory. Neither the P register nor the accumulator are affected by this instruction. High-order bits are taken from the high P register when the right-shift by 6 mode is selected. Low-order bits are zero-filled when left-shifts are selected.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'C25	1	1+d	1+p	2+d+p	1	1+d
	Cycle Timings for a Repeat Execution					
'C25	n	n+nd	n+p	1+n+nd+p	n	n+nd

Example

```
SPL    DAT3     (DP = 4, PM = 2)
```
or
```
SPL    *        If current auxiliary register contains 515.
```

	Before Instruction		After Instruction
P	>FE079844	P	>FE079844
Data Memory 515	>4567	Data Memory 515	>8440

SPM Set P Register Output Shift Mode SPM

Syntax [<label>] SPM <constant>

Operands 0 ≤ constant ≤ 3

Execution (PC) + 1 → PC
Constant → product register shift mode (PM) status bits

Affects PM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	0	0	1	0	PM	

Description The two low-order bits of the instruction word are copied into the PM field of status register ST1. The PM status bits control the P register output shifter. This shifter has the ability to shift the P register output either one or four bits to the left or six bits to the right, or to perform no shift. The bit combinations and their meanings are shown below.

PM	ACTION
00	No shift of multiplier output
01	Output left-shifted 1 place and zero-filled
10	Output left-shifted 4 places and zero-filled
11	Output right-shifted 6 places, sign-extended; LSB bits lost.

The left-shifts allow the product to be justified for fractional arithmetic. The right-shift by six bits has been incorporated to implement up to 128 multiply-accumulate processes without the possibility of overflow occurring. PM may also be loaded by an LST1 instruction.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example

```
SPM    3         Product register shift mode 3 is selected,
                 causing all subsequent transfers from the
                 product register to the ALU to be shifted
                 to the right six places.
```

SQRA Square and Accumulate Previous Product SQRA

Syntax

Direct: [<label>] SQRA <dma>
Indirect: [<label>] SQRA {ind}[,<next ARP>]

Operands

0 ≤ dma ≤ 127
0 ≤ next ARP ≤ 7

Execution

(PC) + 1 → PC
(ACC) + (shifted P register) → ACC
(dma) → T register
(dma) x (dma) → P register

Affects OV; affected by PM and OVM.
Affects C (TMS320C25).

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	0	1	1	1	0	0	1	0	Data Memory Address						
Indirect:	0	0	1	1	1	0	0	1	1	See Section 4.1						

Description The contents of the P register, shifted as defined by the PM status bits, are added to the accumulator. The addressed data memory value is then loaded into the T register, squared, and stored in the P register.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
SQRA  DAT30   (DP = 6, PM = 0)
```
or
```
SQRA  *       If current auxiliary register contains 798.
```

	Before Instruction		After Instruction
Data Memory 798	>F	Data Memory 798	>F
T	>3	T	>F
P	>12C	P	>E1
ACC	X >1F4 C	ACC	0 >320 C

SQRS Square and Subtract Previous Product SQRS

Syntax

Direct: [<label>] SQRS <dma>
Indirect: [<label>] SQRS {ind}[,<next ARP>]

Operands

0 ≤ dma ≤ 127
0 ≤ next ARP ≤ 7

Execution

(PC) + 1 → PC
(ACC) - (shifted P register) → ACC
(dma) → T register
(dma) x (dma) → P register

Affects OV; affected by PM and OVM.
Affects C (TMS320C25).

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	1	1	0	1	0	0	Data Memory Address						
Indirect:	0	1	0	1	1	0	1	0	1	See Section 4.1						

Description

The contents of the P register, shifted as defined by the PM status bits, are subtracted from the accumulator. The addressed data memory value is then loaded into the T register, squared, and stored into the P register.

Words

1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
SQRS   DAT9     (DP = 6, PM = 0)
or
SQRS   *        If current auxiliary register contains 777.
```

		Before Instruction			After Instruction
Data Memory 777		>8	Data Memory 777		>8
T		>1124	T		>8
P		>190	P		>40
ACC	X	>1450	ACC	1	>12C0
	C			C	

SST Store Status Register ST0 SST

Syntax

Direct: [<label>] SST <dma>
Indirect: [<label>] SST {ind}[,<next ARP>]

Operands

$0 \le \text{dma} \le 127$
$0 \le \text{next ARP} \le 7$

Execution

(PC) + 1 → PC
(status register ST0) → dma

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	1	1	1	0	0	0	0	Data Memory Address						
Indirect:	0	1	1	1	1	0	0	0	1	See Section 4.1						

Description

Status register ST0 is stored in data memory.

In the direct addressing mode, status register ST0 is always stored in page 0 regardless of the value of the DP register. The processor automatically forces the page to be 0, and the specific location within that page is defined in the instruction. Note that the DP register is not physically modified. This allows storage of the DP register in the data memory on interrupts, etc., in the direct addressing mode without having to change the DP. In the indirect addressing mode, the data memory address is obtained from the auxiliary register selected. (See the LST instruction for more information.)

The SST instruction can be used to store status register ST0 after interrupts and subroutine calls. The ST0 contains the status bits: OV (overflow flag) bit, OVM (overflow mode) bit, INTM (interrupt mode) bit, ARP (auxiliary register pointer) bit, and DP (data memory page pointer) bit. The status bits are stored in the data memory word as follows:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARP			OV	OVM	1	INTM	DP								

Note that SST * may be used to store status register ST0 anywhere in data memory, while SST in the direct addressing mode is forced to page 0.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	2+d	1+p	3+d+p	–	–
'C25	1	1+d	1+p	2+d+p	1	1+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	3n+nd+p	–	–
'C25	n	n+nd	n+p	1+n+nd+p	n	n+nd

Example

```
SST   DAT96     (DP = don't care)
or
SST   *         If current auxiliary register contains 96.
```

	Before Instruction		After Instruction
Status Register ST0	>A408	Status Register ST0	>A408
Data Memory 96	>A	Data Memory 96	>A408

SST1 Store Status Register ST1 SST1

Syntax

Direct: [<label>] SST1 <dma>
Indirect: [<label>] SST1 {ind}[,<next ARP>]

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution

(PC) + 1 → PC
(status register ST1) → dma

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	1	1	1	0	0	1	0	Data Memory Address						
Indirect:	0	1	1	1	1	0	0	1	1	See Section 4.1						

Description

Status register ST1 is stored in data memory. In the direct addressing mode, status register ST1 is always stored in page 0 regardless of the value of the DP register. The processor automatically forces the page to be 0, and the specific location within that page is defined in the instruction. Note that the DP register is not physically modified. This allows the storage of the DP in the data memory on interrupts, etc., in the direct addressing mode without having to change the DP. In the indirect addressing mode, the data memory address is obtained from the auxiliary register selected. (See the LST1 instruction for more information.)

SST1 is used to store status bits after interrupts and subroutine calls. ST1 contains the status bits: ARB (auxiliary register pointer buffer), CNF (RAM configuration control) bit, TC (test/control) bit, SXM (sign-extension mode) bit, XF (external flag) bit, FO (serial port format) bit, TXM (transmit mode) bit, and the PM (product register shift mode) bit. ST1 on the TMS320C25 also contains the status bits: C (carry) bit, HM (hold mode) bit, and FSM (frame synchronization mode) bit. The bits loaded into status register ST1 from the data memory word are as follows:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARB			CNF	TC	SXM	C†	1	1	HM†	FSM†	XF	FO	TXM	PM	

†On the TMS32020, bits 5, 6, and 9 are one's.

Note that SST1 * may be used to store status register ST1 anywhere in data memory, while SST1 in the direct addressing mode is forced to page 0.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	2+d	1+p	3+d+p	–	–
'C25	1	1+d	1+p	2+d+p	1	1+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	3n+nd+p	–	–
'C25	n	n+nd	n+p	1+n+nd+p	n	n+nd

Example

```
SST1    DAT97    (DP = don't care)
SST1    *        If current auxiliary register contains 97.
```

	Before Instruction		After Instruction
Status Register ST1	>A7E0	Status Register ST1	>A7E0
Data Memory 97	>B	Data Memory 97	>A7E0

Syntax [<label>] SSXM

Operands None

Execution (PC) + 1 → PC
1 → SXM status bit in status register ST1

Affects SXM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	0	0	0	1	1	1

Description The SSXM instruction sets the SXM status bit to logic 1, which enables sign-extension on shifted data memory values for the following arithmetic instructions: ADD, ADDT, ADLK, LAC, LACT, LALK, SBLK, SUB, and SUBT.

SSXM also affects the definition of the SFR instruction. SXM may also be loaded by the LST1 and RSXM instructions.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example

```
SSXM            SXM is set, enabling sign extension on
                subsequent instructions.
```

Syntax [<label>] SSXM

Operands None

Execution (PC) + 1 → PC
1 → TC test/control flag in status register ST1

Affects TC.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	1	1	0	0	1	1

Description The TC (test/control) flag in status register ST1 is set to logic one. TC may also be loaded by the LST1 and RTC instructions.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'C25	n	n	n+p	n+p	n	n

Example

```
STC             TC (test/control) flag is set to logic one.
```

STXM Set Serial Port Transmit Mode STXM

Syntax [<label>] STXM

Operands None

Execution (PC) + 1 → PC
1 → TXM status bit in status register ST1

Affects TXM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	1	0	0	0	0	1

Description The STXM instruction sets the TXM status bit to logic 1, which configures the serial port transmit section to a mode where the FSX pin behaves as an output. A pulse is produced on the FSX pin each time the DXR register is loaded internally. The transmission is initiated by the negative edge of this pulse. TXM may also be loaded by the LST1 and RTXM instructions. If the FSM status bit is a logic zero and serial port operation has already started, the FSX pin will be driven low if TXM = 1.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example

```
STXM            TXM is set, configuring FSX as an output.
```

SUB Subtract from Accumulator with Shift SUB

Syntax

Direct: [<label>] SUB <dma>[,<shift>]
Indirect: [<label>] SUB {ind}[,<shift>[,<next ARP>]]

Operands

$0 \le \text{dma} \le 127$
$0 \le \text{next ARP} \le 7$
$0 \le \text{shift} \le 15$ (defaults to 0)

Execution

$(\text{PC}) + 1 \rightarrow \text{PC}$
$(\text{ACC}) - [(\text{dma}) \times 2^{\text{shift}}] \rightarrow \text{ACC}$

If SXM = 1:
Then (dma) is sign-extended.
If SXM = 0:
Then (dma) is not sign-extended.

Affects OV; affected by OVM and SXM.
Affects C (TMS320C25).

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	0	0	1	Shift				0	Data Memory Address						
Indirect:	0	0	0	1	Shift				1	See Section 4.1						

Description

The contents of the addressed data memory location are left-shifted and subtracted from the accumulator. During shifting, low-order bits are zero-filled. High-order bits are sign-extended if SXM = 1 and zero-filled if SXM = 0. The result is stored in the accumulator.

Words 1

Cycles

	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
	Cycle Timings for a Single Instruction					
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
SUB   DAT80    (DP = 8)
or
SUB   *        If current auxiliary register contains 1104.
```

	C	Before Instruction		C	After Instruction
Data Memory 1104		>11	Data Memory 1104		>11
ACC	X	>24	ACC	1	>13

SUBB Subtract from Accumulator with Borrow (TMS320C25) SUBB

Syntax

Direct: [<label>] SUBB <dma>
Indirect: [<label>] SUBB {ind}[,<next ARP>]

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution

(PC) + 1 → PC
(ACC) - (dma) - ($\overline{C}$) → ACC

Affects C and OV; affected by OVM.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	0	1	1	1	1	0	Data Memory Address						
Indirect:	0	1	0	0	1	1	1	1	1	See Section 4.1						

Description The contents of the addressed data memory location and the value of the carry bit are subtracted from the accumulator. The carry bit is then affected in the normal manner (see Section 3.5.2).

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
SUBB   DAT5    (DP = 8)
or
SUBB   *       If current auxiliary register contains 1029.
```

	Before Instruction	After Instruction
Data Memory 1029	>6	>6
ACC	>6	>FFFFFFFF
C	0	0

In the above example, C is originally zeroed, presumably from the result of a previous subtract instruction that performed a borrow. The effective operation performed was 6 - 6 - ($\overline{0}$) = -1, generating another borrow (and resetting carry again) in the process.

The SUBB instruction can be used in performing multiple-precision arithmetic.

SUBC Conditional Subtract SUBC

Syntax

Direct: [<label>] SUBC <dma>
Indirect: [<label>] SUBC {ind}[,<next ARP>]

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution

(PC) + 1 → PC
(ACC) - [(dma) x 2^{15}] → ALU output

If ALU output ≥ 0:
Then (ALU output) x 2 + 1 → ACC;
Else (ACC) x 2 → ACC.

Affects OV.
Affects C (TMS320C25).
Not affected by OVM (no saturation) or SXM.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	0	0	1	1	1	0	Data Memory Address						
Indirect:	0	1	0	0	0	1	1	1	1	See Section 4.1						

Description

The SUBC instruction performs conditional subtraction, which may be used for division. The 16-bit dividend is placed in the low accumulator, and the high accumulator is zeroed. The divisor is in data memory. SUBC is executed 16 times for 16-bit division. After completion of the last SUBC, the quotient of the division is in the lower-order 16-bit field of the accumulator, and the remainder is in the high-order 16 bits of the accumulator. SUBC assumes the divisor and the dividend are both positive.

If the 16-bit dividend contains less than 16 significant bits, the dividend may be placed in the accumulator left-shifted by the number of leading non-significant zeroes. The number of executions of SUBC is reduced from 16 by that number. One leading zero is always significant.

Note that SUBC affects OV but is not affected by OVM, and therefore the accumulator does not saturate upon positive or negative overflows when executing this instruction.

Words

1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
RPTK    15
SUBC    DAT2      (DP = 4)
or
RPTK    15
SUBC    *          If current auxiliary register contains 514.
```

	Before Instruction		After Instruction	
	C		C	
Data Memory 514		>7		>7
ACC	X	>41	1	>20009

SUBH Subtract from High Accumulator SUBH

Syntax

Direct: [<label>] SUBH <dma>
Indirect: [<label>] SUBH {ind}[,<next ARP>]

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution

(PC) + 1 → PC
$(\text{ACC}) - [(\text{dma}) \times 2^{16}] \rightarrow \text{ACC}$

Affects OV; affected by OVM.
Affects C (TMS320C25).

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	0	0	1	0	0	0	Data Memory Address						
Indirect:	0	1	0	0	0	1	0	0	1	See Section 4.1						

Description

The contents of the addressed data memory location are subtracted from the upper 16 bits of the accumulator. The 16 low-order bits of the accumulator are unaffected. The result is stored in the accumulator. The carry bit C on the TMS320C25 is reset if the result of the subtraction generates a borrow; otherwise, C is unaffected.

The SUBH instruction can be used for performing 32-bit arithmetic.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
SUBH    DAT33    (DP = 6)
or
SUBH    *        If current auxiliary register contains 801.
```

	Before Instruction		After Instruction
Data Memory 801	>4	Data Memory 801	>4
ACC	C: X >A0013	ACC	C: 1 >60013

SUBK Subtract from Accumulator Short Immediate (TMS320C25) SUBK

Syntax [<label>] SUBK <constant>

Operands $0 \leq \text{constant} \leq 255$

Execution (PC) + 1 → PC
(ACC) - 8-bit positive constant → ACC

Affects C and OV: affected by OVM.
Not affected by SXM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	0	1	8-Bit Constant							

Description The 8-bit immediate value is subtracted, right-justified, from the accumulator with the result replacing the accumulator contents. The immediate value is treated as an 8-bit positive number, regardless of the value of SXM.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'C25	not repeatable					

Example `SUBK >12`

	Before Instruction		After Instruction	
	C		C	
ACC	X	>37	1	>25

SUBS Subtract from Low Accumulator with Sign-Extension Suppressed SUBS

Syntax

Direct: [<label>] SUBS <dma>
Indirect: [<label>] SUBS {ind}[,<next ARP>]

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution

$(\text{PC}) + 1 \rightarrow \text{PC}$
$(\text{ACC}) - (\text{dma}) \rightarrow \text{ACC}$

Affects OV; affected by OVM.
Affects C (TMS320C25).
Not affected by SXM.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	0	0	1	0	1	0	Data Memory Address						
Indirect:	0	1	0	0	0	1	0	1	1	See Section 4.1						

Description

The contents of the addressed data memory location are subtracted from the accumulator with sign-extension suppressed. The data is treated as a 16-bit unsigned number, regardless of SXM. The accumulator behaves as a signed number. SUBS produces the same result as a SUB instruction with SXM = 0 and a shift count of 0.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
SUBS    DAT2    (DP = 16)
or
SUBS     *       If current auxiliary register contains 2050.
```

	Before Instruction		After Instruction
Data Memory 2050	>F003	Data Memory 2050	>F003

	Before Instruction			After Instruction
ACC	C: X	>F105	ACC C: 1	>102

SUBT Subtract from Accumulator with Shift Specified by T Register SUBT

Syntax

Direct: [<label>] SUBT <dma>
Indirect: [<label>] SUBT {ind}[,<next ARP>]

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution

$(\text{PC}) + 1 \rightarrow \text{PC}$
$(\text{ACC}) - [(\text{dma}) \times 2^{\text{T register(3-0)}}] \rightarrow (\text{ACC})$

If SXM = 1:
Then (dma) is sign-extended.
If SXM = 0:
Then (dma) is not sign-extended.

Affects OV; affected by SXM and OVM.
Affects C (TMS320C25).

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	0	0	1	1	0	0	Data Memory Address						
Indirect:	0	1	0	0	0	1	1	0	1	See Section 4.1						

Description

The data memory value is left-shifted and subtracted from the accumulator. The left-shift is defined by the four LSBs of the T register, resulting in shift options from 0 to 15 bits. The result replaces the accumulator contents. Sign-extension on the data memory value is controlled by the SXM status bit.

Words

1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
SUBT   DAT127   (DP = 4)
or
SUBT   *        If current auxiliary register contains 639.
```

	Before Instruction	After Instruction
Data Memory 639	>6	>6
T	>FF98	>FF98
ACC	>FDA5	>F7A5
C	X	1

Syntax [<label>] SXF

Operands None

Execution (PC) + 1 → PC
1 → external flag (XF) pin and status bit

Affects XF.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	0	0	1	1	0	1

Description The XF pin and the XF status bit in status register ST1 are set to logic 1. XF may also be loaded by the LST1 and RXF instructions.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	n	n	n+p	n+p	–	–
'C25	n	n	n+p	n+p	n	n

Example

```
SXF          The XF pin and status bit are set to logic 1.
```

Syntax

Direct: [<label>] TBLR <dma>
Indirect: [<label>] TBLR {ind}[,<next ARP>]

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution

TMS32020:

```
(PC) + 1 → TOS
(ACC(15-0)) → PC

If (repeat counter) ≠ 0:
  Then (pma) → dma,
    Modify AR(ARP) and ARP as specified,
    (PC) + 1 → PC,
    (repeat counter) - 1 → repeat counter.

  Else (pma) → dma
    Modify AR(ARP) and ARP as specified.
(TOS) → PC
```

TMS320C25:

```
(PC) + 1 → PC
(PFC) → MCS
(ACC(15-0)) → PFC

If (repeat counter) ≠ 0:
  Then (pma, addressed by PFC) → dma,
    Modify AR(ARP) and ARP as specified,
    (PFC) + 1 → PFC,
    (repeat counter) - 1 → repeat counter.

  Else (pma, addressed by PFC) → dma
    Modify AR(ARP) and ARP as specified.
(MCS) → PFC
```

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	1	1	0	0	0	0	Data Memory Address						
Indirect:	0	1	0	1	1	0	0	0	1	See Section 4.1						

Description

The TBLR instruction transfers a word from a location in program memory to a data memory location specified by the instruction. The program memory address is defined by the low-order 16 bits of the accumulator. For this operation, a read from program memory is performed, followed by a write to data memory. When in the repeat mode, TBLR effectively becomes a single-cycle instruction, and the program counter that contains the ACCL is incremented once each cycle. On the TMS32020, the contents of the lowest stack location are lost when using the TBLR instruction.

If the MP/$\overline{\text{MC}}$ pin on the TMS320C25 is low at the time of execution of this instruction and the program memory address used is less than 4096, an on-chip ROM location will be read.

Words

1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	Table in internal program memory:					
	3	3+d	3+p	3+d+p	–	–
	Table in external program memory:					
	3+p	4+d+p	3+2p	4+d+2p	–	–
'C25	Table in on-chip RAM:					
	2	2+d	3+p	3+d+p	3	3+d
	Table in on-chip ROM:					
	3	3+d	4+p	4+d+p	4	4+d
	Table in external memory:					
	3+p	3+d+p	4+2p	4+d+2p	4+p	4+d+p
	Cycle Timings for a Repeat Execution					
'20	Table in internal program memory:					
	2+n	2+n+nd	2+n+p	2+n+nd+p	–	–
	Table in external program memory:					
	2+n+np	2+2n+nd +np	2+n+np +p	2+2n+nd+np +p	–	–
'C25	Table in on-chip RAM:					
	1+n	1+n+nd	2+n+p	2+n+nd+p	2+n	2+n+nd
	Table in on-chip ROM:					
	2+n	2+n+nd	3+n+p	3+n+nd+p	3+n	3+n+nd
	Table in external memory:					
	2+n+np	1+2n+nd+np	3+n+np +p	2+2n+nd+np +p	3+n+np	2+2n+nd +np

Example

```
TBLR    DAT6    (DP = 4)
TBLR    *       If current auxiliary register contains 518.
```

	Before Instruction		After Instruction
ACC	>23	ACC	>23
Program Memory 23	>306	Program Memory 23	>306
Data Memory 518	>75	Data Memory 518	>306

Syntax

Direct: [<label>] TBLW <dma>
Indirect: [<label>] TBLW {ind}[,<next ARP>]

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution

TMS32020:

```
(PC) + 1 → TOS
(ACC(15-0)) → PC

If (repeat counter) ≠ 0:
  Then (dma) → pma,
   Modify AR(ARP) and ARP as specified,
   (PC) + 1 → PC,
   (repeat counter) - 1 → repeat counter.

  Else (dma) → pma
   Modify AR(ARP) and ARP as specified.
(TOS) → PC
```

TMS320C25:

```
(PC) + 1 → PC
(PFC) → MCS
(ACC(15-0)) → PFC

If (repeat counter) ≠ 0:
  Then (dma, addressed by PFC) → pma,
   Modify AR(ARP) and ARP as specified,
   (PFC) + 1 → PFC,
   (repeat counter) - 1 → repeat counter.

  Else (dma, addressed by PFC) → pma
   Modify AR(ARP) and ARP as specified.
(MCS) → PFC
```

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	1	1	0	0	1	0	Data Memory Address						
Indirect:	0	1	0	1	1	0	0	1	1	See Section 4.1						

Description

The TBLW instruction transfers a word in data memory to program memory. The data memory address is specified by the instruction, and the program memory address is specified by the lower 12 bits of the accumulator. A read from data memory is followed by a write to program memory to complete the instruction. When in the repeat mode, TBLW effectively becomes a single-cycle instruction, and the program counter that contains the ACCL is incremented once each cycle. On the TMS32020, the contents of the lowest stack location are lost when using the TBLW instruction.

If the $\mathrm{MP}/\overline{\mathrm{MC}}$ pin on the TMS320C25 is low at the time of execution of this instruction and the program memory address used is less than 4096, an on-chip ROM location will be addressed but not written to.

Words

1

Cycles

		PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
	Cycle Timings for a Single Instruction						
'20	Table in internal program memory:	3	3+d	3+p	3+d+p	–	–
	Table in external program memory:	3+p	4+d+p	3+2p	4+d+2p	–	–
'C25	Table in on-chip RAM:	2	3+d	3+p	4+d+p	3	4+d
	Table in on-chip ROM:	not applicable					
	Table in external memory:	2+p	3+d+p	3+2p	4+d+2p	3+p	4+d+p
	Cycle Timings for a Repeat Execution						
'20	Table in internal program memory:	2+n	2+n+nd	2+n+p	2+n+nd+p	–	–
	Table in external program memory:	2+n+np	2+2n+nd +np	2+n+np +p	2+2n+nd+np +p	–	–
'C25	Table in on-chip RAM:	1+n	2+n+nd	2+n+p	3+n+nd+p	2+n	3+n+nd
	Table in on-chip ROM:	not applicable					
	Table in external memory:	1+n+np	1+2n+nd+np	2+n+np +p	2+2n+nd+np +p	2+n+np	2+2n+nd +np

Example

```
TBLW  DAT5    (DP = 32)
TBLW  *       If current auxiliary register contains 4101.
```

	Before Instruction		After Instruction
ACC	>257	ACC	>257
Data Memory 4101	>4339	Data Memory 4101	>4339
Program Memory 257	>306	Program Memory 257	>4339

Syntax [<label>] TRAP

Operands None

Execution (PC) + 1 → stack
30 → PC

Not affected by INTM; does not affect INTM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	1	1	0	0	0	0	1	1	1	1	0

Description The TRAP instruction is a software interrupt that transfers program control to program memory location 30 and pushes the program counter plus one onto the hardware stack. The instruction at location 30 may contain a branch instruction to transfer control to the TRAP routine. Putting the PC + 1 onto the stack enables an RET instruction to pop the return PC (points to instruction after the TRAP) from the stack.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	2	2	2+p	2+p	–	–
'C25	Destination on-chip RAM:					
	2	2	2+p	2+p	2	2
	Destination on-chip ROM:					
	3	3	3+p	3+p	3	3
	Destination external memory:					
	3+p	3+p	3+2p	3+2p	3+p	3+p
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example

```
TRAP          Control is passed to program memory location
              30.  PC + 1 is pushed onto the stack.
```

XOR Exclusive-OR with Accumulator XOR

Syntax

Direct: [<label>] XOR <dma>
Indirect: [<label>] XOR {ind}[,<next ARP>]

Operands

0 ≤ dma ≤ 127
0 ≤ next ARP ≤ 7

Execution

(PC) + 1 → PC
(ACC(15-0)).XOR.dma → ACC(15-0)
(ACC(31-16)) → ACC(31-16)

Not affected by SXM.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	0	1	1	0	0	0	Data Memory Address						
Indirect:	0	1	0	0	1	1	0	0	1	See Section 4.1						

Description

The low half of the accumulator is exclusive-ORed with the contents of the addressed data memory location. The upper half of the accumulator is not affected by this instruction.

Words

1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
XOR   DAT127    (DP = 511)
or
XOR   *         If current auxiliary register contains 65535.
```

	Before Instruction		After Instruction
Data Memory 65535	>F0F0	Data Memory 65535	>F0F0
ACC	X (C) >12345678	ACC	X (C) >1234A688

XORK XOR Immediate with Accumulator with Shift XORK

Syntax [<label>] XORK <constant>[,<shift>]

Operands 16-bit constant
0 ≤ shift ≤ 15 (defaults to 0)

Execution (PC) + 2 → PC
(ACC(30-0)).XOR.[constant x 2^{shift}] → ACC(30-0)
(ACC(31)) → ACC(31)

Not affected by SXM.

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	Shift				0	0	0	0	0	1	1	0
16-Bit Constant															

Description The left-shifted 16-bit immediate constant is exclusive-ORed with the accumulator, leaving the result in the accumulator. Low-order bits below and high-order bits above the shifted value are treated as zeroes, thus not affecting the corresponding bits of the accumulator. Note that the most-significant bit of the accumulator is not affected, regardless of the shift code value.

Words 2

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	2	2	2+2p	2+2p	–	–
'C25	2	2	2+2p	2+2p	2	2
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example `XORK >FFFF,8`

	Before Instruction		After Instruction
ACC	X >12345678	ACC	X >12CBA978
	C		C

ZAC Zero Accumulator ZAC

Syntax [<label>] ZAC

Operands None

Execution (PC) + 1 → PC
0 → ACC

Encoding

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0

Description The contents of the accumulator are replaced with zero. The ZAC instruction has been implemented as a special case of LACK. (ZAC assembles as LACK 0.)

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	1	1+p	1+p	–	–
'C25	1	1	1+p	1+p	1	1
	Cycle Timings for a Repeat Execution					
'20	not repeatable				–	–
'C25	not repeatable					

Example `ZAC`

		Before Instruction			After Instruction
ACC	X	>A5A5A5A5	ACC	X	>0
	C			C	

ZALH Zero Low Accumulator and Load High Accumulator ZALH

Syntax

Direct: [<label>] ZALH <dma>
Indirect: [<label>] ZALH {ind}[,<next ARP>]

Operands

0 ≤ dma ≤ 127
0 ≤ next ARP ≤ 7

Execution

(PC) + 1 → PC
0 → ACC(15-0)
(dma) → ACC(31-16)

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	0	0	0	0	0	0	Data Memory Address						
Indirect:	0	1	0	0	0	0	0	0	1	See Section 4.1						

Description

ZALH loads a data memory value into the high-order half of the accumulator. The low-order bits of the accumulator are zeroed.

ZALH is useful for 32-bit arithmetic operations.

Words

1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PP/DI	PR/DE
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
ZALH   DAT3   (DP = 32)
```
or
```
ZALH   *      If current auxiliary register contains 4099.
```

	Before Instruction		After Instruction
Data Memory 4099	>3F01	Data Memory 4099	>3F01
ACC	X (C) >77FFFF	ACC	X (C) >3F010000

ZALR Zero Low Accumulator, Load High Accumulator with Rounding (TMS320C25) ZALR

Syntax

Direct: [<label>] ZALR <dma>
Indirect: [<label>] ZALR {ind}[,<next ARP>]

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution

(PC) + 1 → PC
>8000 → ACC(15-0)
(dma) → ACC(31-16)

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	1	1	1	0	1	1	0	Data Memory Address						
Indirect:	0	1	1	1	1	0	1	1	1	See Section 4.1						

Description

The ZALR instruction loads a data memory value into the high-order half of the accumulator with rounding the value by adding 1/2 LSB; i.e., the 15 low bits (bits 0-14) of the accumulator are set to zero and bit 15 of the accumulator is set to one.

ZALR is a derivative instruction from ZALH.

Words 1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	**PI/DE**	**PE/DI**	**PE/DE**	**PR/DI**	**PR/DE**
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
ZALR   DAT3    (DP = 32)
or
ZALR   *       If current auxiliary register contains 4099.
```

	Before Instruction		After Instruction
Data Memory 4099	>3F01	Data Memory 4099	>3F01
ACC X (C)	>77FFFF	ACC X (C)	>3F018000

ZALS Zero Accumulator, Load Low Accumulator with Sign-Extension Suppressed ZALS

Syntax

Direct: [<label>] ZALS <dma>
Indirect: [<label>] ZALS {ind}[,<next ARP>]

Operands

$0 \leq \text{dma} \leq 127$
$0 \leq \text{next ARP} \leq 7$

Execution

$(\text{PC}) + 1 \rightarrow \text{PC}$
$0 \rightarrow \text{ACC}(31\text{-}16)$
$(\text{dma}) \rightarrow \text{ACC}(15\text{-}0)$

Not affected by SXM.

Encoding

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct:	0	1	0	0	0	0	0	1	0	Data Memory Address						
Indirect:	0	1	0	0	0	0	0	1	1	See Section 4.1						

Description

The contents of the addressed data memory location are loaded into the 16 low-order bits of the accumulator. The upper half of the accumulator is zeroed. The data is treated as a 16-bit unsigned number rather than a two's-complement number. Therefore, there is no sign-extension with this instruction, regardless of the state of SXM. (ZALS behaves the same as a LAC instruction with no shift and SXM = 0.)

ZALS is useful for 32-bit arithmetic operations.

Words

1

Cycles

	Cycle Timings for a Single Instruction					
	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
'20	1	2+d	1+p	2+d+p	–	–
'C25	1	2+d	1+p	2+d+p	1	2+d
	Cycle Timings for a Repeat Execution					
'20	n	2n+nd	n+p	2n+nd+p	–	–
'C25	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd

Example

```
ZALS   DAT1    (DP = 6)
or
ZALS   *       If current auxiliary register contains 769.
```

	Before Instruction		After Instruction
Data Memory 769	>F7FF	Data Memory 769	>F7FF
ACC X C	>7FF00033	ACC X C	>F7FF

5. Software Applications

The TMS320C2x microprocessor/microcomputer design emphasizes overall speed, communication, and flexibility. Many instructions are tailored to digital signal processing tasks, providing single-cycle multiply/accumulates, adaptive filtering support, and many other features. General-purpose instructions support floating-point, extended-precision, logical processing, and control applications.

This section provides explanations of how to use the various TMS320C2x processor and instruction set features along with assembly language coding examples. More information about specific applications can be found in the book, *Digital Signal Processing Applications with the TMS320 Family*.

Major topics discussed in this section are listed below.

- Processor Initialization (Section 5.1 on page 5-2)
- Program Control (Section 5.2 on page 5-7)
    - Subroutines
    - Software stack
    - Timer operation
    - Single-instruction loops
    - Computed GOTOs
- Interrupt Service Routines (Section 5.3 on page 5-16)
    - Context switching
    - Interrupt priority
- Memory Management (Section 5.4 on page 5-23)
    - Block moves
    - Configuring on-chip RAM
    - Using on-chip RAM for program execution
- Fundamental Logical and Arithmetic Operations (Section 5.5 on page 5-31)
    - Status register effects
    - Bit manipulation
- Advanced Arithmetic Operations (Section 5.6 on page 5-34)
    - Overflow management
    - Scaling
    - Moving data
    - Multiplication
    - Division
    - Floating-point arithmetic
    - Indexed addressing
    - Extended-precision arithmetic
- Application-Oriented Operations (Section 5.7 on page 5-60)
    - Companding
    - Filtering
    - Fast Fourier Transforms (FFT)
    - PID control.

5.1 Processor Initialization

Prior to the execution of a digital signal processing algorithm, it is necessary to initialize the processor. Generally, initialization takes place anytime the processor is reset.

When reset is activated by applying a low level to the $\overline{\text{RS}}$ (reset) input for at least three cycles, the TMS320C2x terminates execution and forces the program counter (PC) to zero. Program memory location 0 normally contains a B (branch) instruction in order to direct program execution to the system initialization routine. The hardware reset also initializes various registers and status bits.

After reset, the processor should be initialized to meet the requirements of the system. Instructions should be executed that set up operational modes, memory pointers, interrupts, and the remaining functions necessary to meet system requirements.

To configure the processor after reset, the following internal functions should be initialized:

- Memory-mapped registers
- Interrupt structure
- Mode control (OVM, SXM, FO, TXM, PM; plus HM and FSM on TMS320C25)
- Memory control (CNF)
- Auxiliary registers and the auxiliary register pointer (ARP)
- Data memory page pointer (DP).

The OVM (overflow mode), TC (test/control flag), and IMR (interrupt mask register) bits are not initialized by reset. The auxiliary register pointer (ARP), auxiliary register pointer buffer (ARB), and data memory page pointer (DP) are also not initialized by reset.

Example 5-1 and Example 5-2 show coding for initializing the TMS32020 and TMS320C25, respectively, to the following machine state, in addition to the initialization performed during the hardware reset:

- All interrupts enabled
- Overflow mode (OVM) disabled
- Data memory page pointer (DP) set to zero
- Auxiliary register pointer (ARP) set to four (TMS32020) or seven (TMS320C25)
- Internal memory filled with zero.

Example 5-1. Processor Initialization (TMS32020)

```
       TITL 'PROCESSOR INITIALIZATION'
       IDT  'EXAMPLE'
       DEF  RESET,INT0,INT1,INT2
       DEF  TINT,RINT,XINT,USER
       REF  ISR0,ISR1,ISR2
       REF  TIME,RCV,XMT,PROC
*
* PROCESSOR INITIALIZATION FOR THE TMS32020.
* RESET AND INTERRUPT VECTOR SPECIFICATION.
* BRANCHES FOR EXTERNAL AND INTERNAL INTERRUPTS.
*
       AORG >0000
RESET  B    INIT      ; RS- BEGINS PROCESSING HERE.
*
INT0   B    ISR0      ; INT0- BEGINS PROCESSING HERE.
INT1   B    ISR1      ; INT1- BEGINS PROCESSING HERE.
INT2   B    ISR2      ; INT2- BEGINS PROCESSING HERE.
*
       AORG >0018
TINT   B    TIME      ; TIMER INTERRUPT PROCESSING.
RINT   B    RCV       ; SERIAL PORT RECEIVE PROCESSING.
XINT   B    XMT       ; SERIAL PORT TRANSMIT PROCESSING.
*
USER   B    PROC      ; TRAP VECTOR PROCESSING BEGINS.
*
* THE BRANCH INSTRUCTION AT PROGRAM MEMORY LOCATION 0 DIRECTS
* EXECUTION TO BEGIN HERE FOR RESET PROCESSING THAT INITIAL-
* IZES THE PROCESSOR. WHEN RESET IS APPLIED, THE FOLLOWING
* CONDITIONS ARE ESTABLISHED FOR THE STATUS AND OTHER
* INTERNAL REGISTERS:
*
*        ARP   OV   OVM  1  INTM     DP
* ST0:   XXX   0    X    1   1     XXXXXXXXX
*
*        ARB   CNF  TC  SXM  11111  XF  FO  TXM  PM
* ST1:   XXX   0    X   X    11111  1   0   0    XX
*
* REGISTER   ADDRESS        DATA
*   DRR       >0000    XXXX XXXX XXXX XXXX
*   DXR       >0001    XXXX XXXX XXXX XXXX
*   TIM       >0002    1111 1111 1111 1111
*   PRD       >0003    XXXX XXXX XXXX XXXX
*   IMR       >0004    1111 1111 11XX XXXX
*   GREG      >0005    1111 1111 0000 0000
*
*         RESERVED   XINT   RINT   TINT   INT2   INT1   INT0
* IMR:   1111111111   X      X      X      X      X      X
*
INIT   ROVM           ; DISABLE OVERFLOW MODE.
       LDPK 0         ; POINT DP REGISTER TO DATA PAGE 0.
       LARP 4         ; POINT TO AUXILIARY REGISTER 4.
       LACK >3F       ; LOAD ACCUMULATOR WITH >3F.
       SACL 4         ; ENABLE ALL INTERRUPTS VIA IMR.
       LALK >FFFF     ; LOAD ACCUMULATOR WITH >FFFF.
       SACL 3         ; INITIALIZE PERIOD REGISTER.
       SSXM           ; SET SIGN-EXTENSION MODE TO 1.
       SPM  0         ; SET PM BITS TO 0.
```

```
*
* INTERNAL DATA MEMORY INITIALIZATION.
*
       ZAC             ; ZERO THE ACCUMULATOR.
       LARK AR4,>60    ; POINT TO BLOCK B2.
       RPTK 31
       SACL *+         ; STORE ZERO IN ALL 32 LOCATIONS.
*
       LRLK AR4,>200   ; POINT TO BLOCK B0.
       RPTK 255
       SACL *+         ; ZERO ALL OF PAGES 4 AND 5.
*
       LRLK AR4,>300   ; POINT TO BLOCK B1.
       RPTK 255
       SACL *+         ; ZERO ALL OF PAGES 6 AND 7.
*
* THE PROCESSOR IS INITIALIZED. THE REMAINING APPLICATION-
* DEPENDENT PART OF THE SYSTEM (BOTH ON- AND OFF-CHIP) SHOULD
* NOW BE INITIALIZED.
*
       EINT            ; ENABLE ALL INTERRUPTS.
```

Example 5-2. Processor Initialization (TMS320C25)

```
       TITL 'PROCESSOR INITIALIZATION'
       IDT  'EXAMPLE'
       DEF  RESET,INT0,INT1,INT2
       DEF  TINT,RINT,XINT,USER
       REF  ISR0,ISR1,ISR2
       REF  TIME,RCV,XMT,PROC

* PROCESSOR INITIALIZATION FOR THE TMS320C25.
* RESET AND INTERRUPT VECTOR SPECIFICATION.
* BRANCHES FOR EXTERNAL AND INTERNAL INTERRUPTS.
*
       AORG >0000
RESET  B    INIT   ; RS- BEGINS PROCESSING HERE.
*
INT0   B    ISR0   ; INT0- BEGINS PROCESSING HERE.
INT1   B    ISR1   ; INT1- BEGINS PROCESSING HERE.
INT2   B    ISR2   ; INT2- BEGINS PROCESSING HERE.
*
       AORG >0018
TINT   B    TIME   ; TIMER INTERRUPT PROCESSING.
RINT   B    RCV    ; SERIAL PORT RECEIVE PROCESSING.
XINT   B    XMT    ; SERIAL PORT TRANSMIT PROCESSING.
*
USER   B    PROC   ; TRAP VECTOR PROCESSING BEGINS.
*
* THE BRANCH INSTRUCTION AT PROGRAM MEMORY LOCATION 0 DIRECTS
* EXECUTION TO BEGIN HERE FOR RESET PROCESSING THAT INITIAL-
* IZES THE PROCESSOR. WHEN RESET IS APPLIED, THE FOLLOWING
* CONDITIONS ARE ESTABLISHED FOR THE STATUS AND OTHER
* INTERNAL REGISTERS:
*
*        ARP   OV   OVM  1  INTM      DP
* ST0:   XXX    0    X   1    1    XXXXXXXXX
*
*        ARB   CNF  TC  SXM  C   11  HM  FSM XF  FO  TXM  PM
* ST1:   XXX    0    X   1   1   11   1   1   1   0   0   00
*
* REGISTER   ADDRESS          DATA
*   DRR       >0000    XXXX XXXX XXXX XXXX
*   DXR       >0001    XXXX XXXX XXXX XXXX
*   TIM       >0002    1111 1111 1111 1111
*   PRD       >0003    1111 1111 1111 1111
*   IMR       >0004    1111 1111 11XX XXXX
*   GREG      >0005    1111 1111 0000 0000
*
*         RESERVED   XINT   RINT   TINT   INT2   INT1   INT0
* IMR:   1111111111    X      X      X      X      X      X
*
INIT   ROVM        ; DISABLE OVERFLOW MODE
       LDPK 0      ; POINT DP REGISTER TO DATA PAGE 0.
       LARP 7      ; POINT TO AUXILIARY REGISTER 7.
       LACK >3F    ; LOAD ACCUMULATOR WITH >3F.
       SACL 4      ; ENABLE ALL INTERRUPTS VIA IMR.
```

```
*
* INTERNAL DATA MEMORY INITIALIZATION.
*
       ZAC               ; ZERO THE ACCUMULATOR.
       LARK  AR7,>60     ; POINT TO BLOCK B2.
       RPTK  31
       SACL  *+          ; STORE ZERO IN ALL 32 LOCATIONS.
*
       LRLK  AR7,>200    ; POINT TO BLOCK B0.
       RPTK  255
       SACL  *+          ; ZERO ALL OF PAGES 4 AND 5.
*
       LRLK  AR7,>300    ; POINT TO BLOCK B1.
       RPTK  255
       SACL  *+          ; ZERO ALL OF PAGES 6 AMD 7.
*
* THE PROCESSOR IS INITIALIZED. THE REMAINING APPLICATION-
* DEPENDENT PART OF THE SYSTEM (BOTH ON- AND OFF-CHIP) SHOULD
* NOW BE INITIALIZED.
*
       EINT              ; ENABLE ALL INTERRUPTS.
```

5.2 Program Control

To facilitate the TMS320C2x's use in general-purpose high-speed processing, a variety of instructions are provided for software stack expansion, subroutine calls, timer operation, single-instruction loops, and external branch control. Descriptions and examples of how to use these features of the TMS320C2x are given in this section.

5.2.1 Subroutines

The TMS320C2x has a 16-bit Program Counter (PC) and a four-level (TMS32020) or eight-level (TMS320C25) hardware stack for PC storage. The CALL and CALA subroutine calls store the current contents of the program counter on the top of the stack. The RET (return from subroutine) instruction then pops the top of the stack to the program counter.

Example 5-3 illustrates the use of a subroutine to determine the square root of a 16-bit number. Processing proceeds in the main routine to the point where the square root of a number should be taken. At this point a CALL is made to the subroutine, transferring control to that section of the program memory for execution and then returning to the calling routine via the RET instruction when execution has completed.

Example 5-3. Subroutines

```
* AUTOCORRELATION
*
* THIS ROUTINE PERFORMS A CORRELATION OF TWO VECTORS AND THEN
* CALLS A SQUARE ROOT SUBROUTINE THAT WILL DETERMINE THE RMS
* AMPLITUDE OF THE WAVEFORM.
*
AUTOC
          .
          .
          .
       LAC   ENERGY
       CALL  SQRT
       SACL  ENERGY
          .
          .
          .
*
* SQUARE ROOT
*
* THIS SUBROUTINE DETERMINES THE SQUARE ROOT OF A NUMBER X
* THAT IS LOCATED IN THE LOW HALF OF THE ACCUMULATOR WHEN
* THE ROUTINE IS CALLED. THE FRACTIONAL SQUARE ROOT OF X IS
* TAKEN, WHERE 0 < X < 1 AND WHERE 1 IS REPRESENTED BY
* >7FFF. THE RESULT IS RETURNED TO THE CALLING ROUTINE IN
* THE ACCUMULATOR
*
ST0    EQU   >60        ; SAVED STATUS REGISTER ST0 ADDRESS
ST1    EQU   >61        ; SAVED STATUS REGISTER ST1 ADDRESS
NUMBER EQU   >62        ; NUMBER X WHOSE SQUARE ROOT IS TAKEN
TEMPR  EQU   >63        ; INTERMEDIATE ROOTS
GUESS  EQU   >64        ; SQUARE ROOT OF X
*
SQRT   SST   ST0        ; SAVE STATUS REGISTER ST0.
       SST1  ST1        ; SAVE STATUS REGISTER ST1.
       LDPK  0          ; LOAD DATA PAGE POINTER = 0.
       SSXM             ; SET SIGN-EXTENSION MODE.
       SPM   1          ; LEFT-SHIFT PR OUTPUT TO ACCUMULATOR.
       SACL  NUMBER     ; SAVE X.
       LARP  AR1        ; INITIALIZE VARIABLES FOR SQUARE ROOT.
       LARK  AR1,11     ; 12 ITERATIONS
       LALK  >800       ; ASSUME X IS LESS THAN >200.
       SACL  GUESS      ; SET INITIAL GUESS TO >800.
       SACL  TEMPR      ; SET FIRST INTERMEDIATE ROOT TO >800.
       SACH  ROOT       ; SET SQUARE ROOT VALUE TO 0.
       LAC   NUMBER     ; LOAD X INTO THE ACCUMULATOR.
       SBLK  >200       ; TEST IF X IS LESS THAN >200.
       BLZ   SQRTLP     ; IF YES, TAKE THE ROOT;
       LAC   GUESS,3    ; IF NO, THEN  REINITIALIZE.
       SACL  GUESS      ; SET INITIAL GUESS TO >4000.
       SACL  TEMPR      ; SET FIRST INTERMEDIATE ROOT TO >4000.
       LARK  AR1,14     ; 15 ITERATIONS
*
* SQUARE ROOT LOOP
*
SQRTLP SQRA  TEMPR      ; SQUARE TEMPORARY (INTERMEDIATE) ROOT.
       ZALH  NUMBER     ; CHECK IF RESULT IS LESS THAN X.
       SPAC
       BLZ   NEXTLP     ; IF IT'S NOT, SKIP ROOT UPDATE.
       ZALH  TEMPR      ; IF IT IS, SET ROOT EQUAL TEMPR.
       SACH  ROOT
```

```
NEXTLP LAC   GUESS,15  ; SCALE DOWN GUESS BY 2 TO CONVERGE.
       SACH  GUESS
       ADDH  ROOT      ; ADD CURRENT ROOT ESTIMATE.
       SACH  TEMPR     ; UPDATE TEMPORARY ROOT VALUE.
       BANZ  SQRTLP    ; REPEAT SPECIFIED NO. OF ITERATIONS.
       LAC   ROOT      ; LOAD THE ROOT OF X.
       LST1  ST1       ; RESTORE STATUS REGISTER ST1.
       LST   ST0       ; RESTORE STATUS REGISTER ST0.
       RET
```

Hardware stack allocation involves its use in interrupts, subroutine calls, pipelined instructions, and the emulator (XDS). The TMS320C2x disables all interrupts when taking an interrupt trap. If interrupts are enabled more than one instruction before the return of the interrupt service routine, the routine can also be interrupted, thus using another level of the hardware stack. This condition should be considered when managing the use of the stack. When nesting subroutine calls, each call uses a level of the stack. The number of levels used by the interrupt must be remembered as well as the depth of the nesting of subroutines. One level of the stack is reserved for the emulator (XDS) to be used for breakpoint/single-step operations. If the XDS is not used, this extra level is available for internal use. Given these constraints, the following listings describe possible allocations of the hardware stack levels:

TMS32020:

- 1 level reserved for emulator (XDS) stack
- 1 level reserved for TRAP (software interrupt) instruction
- 1 level reserved for interrupt service routines (ISR)
- 1 level available for subroutine calls.

TMS320C25:

- 1 level reserved for emulator (XDS) stack
- 1 level reserved for TRAP (software interrupt) instruction
- 1 level reserved for interrupt service routines (ISR)
- 5 levels available for subroutine calls.

or:

- 1 level reserved for emulator (XDS) stack
- 1 level reserved for TRAP (software interrupt) instruction
- 2 levels reserved for interrupt service routines (ISR)
- 4 levels available for subroutine calls.

When two levels are allocated for ISRs on the TMS320C25, the individual ISRs can utilize one level of subroutine calls or one level of interrupt nesting.

5.2.2 Software Stack

Provisions have been made on the TMS320C2x for extending the hardware stack into data memory. This is useful for deep subroutine nesting or stack overflow protection.

The hardware stack is accessible via the accumulator using the PUSH and POP instructions. Two additional instructions, PSHD and POPD, are included in the instruction set so that the stack may be directly stored to and recovered from data memory.

A software stack can be implemented by using the POPD instruction at the beginning of each subroutine in order to save the PC in data memory. Then before returning, a PSHD is used to put the proper value back onto the top of the stack.

When the stack has three (TMS32020) or seven (TMS320C25) values stored on it and two or more values are to be put on the stack before any other values are popped off, a subroutine that expands the stack is needed, such as shown in Example 5-4. In this example, the main program stores the stack starting location in memory in AR2 and indicates to the subroutine whether to push data from memory onto the stack or pop data from the stack to memory. If a zero is loaded into the accumulator before calling the subroutine, the subroutine pushes data from memory to the stack. If a one is loaded into the accumulator, the subroutine pops data from the stack to memory.

Since the CALL instruction uses the stack to save the program counter, the subroutine pops this value into the accumulator and utilizes the BACC (branch to address specified by accumulator) instruction to return to the main program. This prevents the program counter from being stored into a memory location. The subroutine in Example 5-4 uses the BANZ (branch on auxiliary register not zero) instruction to control all of its loops.

Example 5-4. Software Stack Expansion

```
* THIS ROUTINE EXPANDS THE STACK WHILE LETTING THE MAIN
* PROGRAM DETERMINE WHERE TO STORE THE STACK CONTENTS OR FROM
* WHERE TO RECOVER THEM.
*
STACK  LARP 2        ; USE AR2.
       BNZ  PO       ; IF POPD IS NEEDED, GOTO PO.
       POP           ; ELSE, SAVE PROGRAM COUNTER.
       RPTK 6        ; LOAD REPEAT COUNTER.
       PSHD *+       ; PUT MEMORY IN STACK.
       BACC          ; RETURN TO MAIN PROGRAM.
PO     POP           ; SAVE PROGRAM COUNTER.
       MAR  *-       ; ALIGN STACK POINTER.
       RPTK 6        ; LOAD REPEAT COUNTER.
       POPD *-       ; PUT STACK IN MEMORY.
       MAR  *+       ; REALIGN STACK POINTER.
       BACC          ; RETURN TO MAIN PROGRAM.
```

5.2.3 Timer Operation

The TMS320C2x provides a 16-bit on-chip timer and its associated interrupt to perform various functions at regular time intervals. The timer is a down counter that is continuously clocked by CLKOUT1 on the TMS320C25, and counts (PRD + 1) cycles of CLKOUT1. The timer is clocked by CLKOUT1/4 on the TMS32020, and counts (4 × PRD) cycles of CLKOUT1. By programming the period (PRD) register from 1 to 65,535 (>FFFF), a timer interrupt (TINT) can be generated every 2 to 65,536 cycles on the TMS320C25. Note that a TINT can be generated every 4 to 262,140 cycles on the TMS32020. (A period register value of zero is not allowed.)

Two memory-mapped registers are used to operate the timer. The timer (TIM) register, data memory location 2, holds the current count of the timer. At every CLKOUT1 cycle, the TIM register is decremented by one. The PRD register, data memory location 3, holds the starting count for the timer. When the TIM register decrements to zero, a timer interrupt (TINT) is generated. In the following cycle, the contents of the PRD register are loaded into the TIM register. In this way, a TINT is generated every (PRD + 1) cycles of CLKOUT1 on the TMS320C25 or (4 × PRD) cycles of CLKOUT1 on the TMS32020.

The timer and period registers can be read from or written to on any cycle. The count can be monitored by reading the TIM register. A new counter period can be written to the PRD register without disturbing the current timer count. The timer will then start the new period after the current count is complete. If both the PRD and TIM registers are loaded with a new period, the timer begins decrementing the new period without generating an interrupt. Thus, the programmer has complete control of the current and next periods of the timer.

The TIM register is set to the maximum value on reset (>FFFF) for both the TMS32020 and TMS320C25. The PRD register is also initialized by reset on the TMS320C25 to >FFFF. The TMS32020 requires a software initialization of the PRD register (see Example 5-1). The TIM register begins decrementing only after $\overline{\text{RS}}$ is de-asserted. If the timer is not used, TINT should be masked. The PRD register can then be used as a general-purpose data memory location. If TINT is used, the PRD and TIM registers should be programmed before unmasking the TINT.

Example 5-5 and Example 5-6 show the assembly code that implements the use of the timer to divide down the CLKOUT1 signal. To generate a 9600-Hz clock signal, the PRD register should be loaded with 520. In the timer interrupt service routine, the XF line is toggled. The XF output is also used as an input for $\overline{\text{BIO}}$ in this example. The output of XF will provide a 50-percent duty cycle clock signal as long as the main routine or other interrupt routines do not disable interrupts. Interrupts may be disabled by direct or implied use of DINT, or by executing instructions in the repeat mode. The value for the PRD register is calculated as follows:

TMS32020:

CLKOUT1/(4 × PRD) = 2 × frequency of signal
5 MHz/(4 × 65) = 2 × 9600 Hz (= 9615 Hz for divided signal)

TMS320C25:

CLKOUT1/(PRD + 1) = 2 × frequency of signal
10 MHz/(520 + 1) = 2 × 9600 Hz (= 9597 Hz for divided signal)

Example 5-5. Clock Divider Using Timer (TMS32020)

```
* SETUP FOR INTERRUPT SERVICE ROUTINE.
*
        LACK 65
        SACL DMA3     ; LOAD THE PERIOD REGISTER.
        LACK 8
        OR   DMA4
        SACL DMA4     ; ENABLE THE TIMER INTERRUPT.
        EINT          ; ENABLE INTERRUPTS.
          .
          .
          .
* I/O SERVICE ROUTINE.
*
TIME    BIOZ SET1     ; CHECK THE CURRENT XF STATE.
        RXF           ; XF WAS HIGH; SET IT LOW.
        EINT          ; ENABLE INTERRUPTS.
        RET           ; RETURN TO INTERRUPTED CODE.
SET1    SXF           ; XF WAS LOW; SET IT HIGH.
        EINT          ; ENABLE INTERRUPTS.
        RET           ; RETURN TO INTERRUPTED CODE.
```

Example 5-6. Clock Divider Using Timer (TMS320C25)

```
* SETUP FOR INTERRUPT SERVICE ROUTINE.
*
        LALK 520
        SACL DMA3    ; LOAD THE PERIOD REGISTER.
        LACK 8
        OR   DMA4
        SACL DMA4    ; ENABLE THE TIMER INTERRUPT.
        EINT         ; ENABLE INTERRUPTS.
         .
         .
         .
* I/O SERVICE ROUTINE.
*
TIME    BIOZ SET1    ; CHECK THE CURRENT XF STATE.
        RXF          ; XF WAS HIGH; SET IT LOW.
        EINT         ; ENABLE INTERRUPTS.
        RET          ; RETURN TO INTERRUPTED CODE.
SET1    SXF          ; XF WAS LOW; SET IT HIGH.
        EINT         ; ENABLE INTERRUPTS.
        RET          ; RETURN TO INTERRUPTED CODE.
```

5.2.4 Single-Instruction Loops

When programming time-critical high-computational tasks, it is often necessary to repeat the same operation many times. For these cases, repeat instructions that allow the execution of the next single instruction N+1 times are provided. N is defined by an eight-bit repeat counter (RPTC), which is loaded by the RPT or RPTK instructions. The instruction immediately following is then executed, and the RPTC is decremented until it reaches zero.

When using the repeat feature, the instruction being repeated is fetched only once. As a result, many multicycle instructions become single-cycle when repeated. This is especially useful for I/O instructions, such as TBLR/TBLW, IN/OUT, or BLKD/BLKP.

Since the instruction is fetched and internally latched, the program bus can be used to fetch or write a second operand in parallel to operations using the data bus. With the instruction latched for repeated execution, the program counter can be loaded with a data address and incremented on succeeding executions to fetch data in successive memory locations. As an example, the MAC instruction fetches the multiplicand from program memory via the program bus. Simultaneous with the program bus fetch, the second multiplicand is fetched from data memory via the data bus. In addition to these data fetches, preparation is made for accesses in the following cycles by incrementing the program counter and by indexing the auxiliary register. TBLR is another example of an instruction that benefits from simultaneous transfers of data on both the program and data buses. In this case, data values from a table in program memory may be read and transferred to data memory. When repeated, the program overhead of reading the instruction from program memory must be executed only once, thus allowing the rest of the executions to operate in a single cycle.

Programs, such as those implementing digital filters, require loops that execute in a minimum amount of time. Example 5-7 shows the use of the RPT or RPTK instructions.

Example 5-7. Instruction Repeating

```
* THIS ROUTINE USES THE RPT INSTRUCTION TO SET UP THE LOOP
* COUNTER IN ONE CYCLE. THE FOLLOWING EQUATION IS IMPLEMENTED
* IN THIS ROUTINE:
*
*             10
*            -----
*            \       X(I) x Y(I)
*            /
*            -----
*            I = 1
*
* THIS ROUTINE ASSUMES THAT THE X VALUES ARE LOCATED IN
* ON-CHIP RAM BLOCK B0, AND THE Y VALUES IN BLOCK B1. WHEN
* REPLACING RPT NUM WITH RPTK 9, THE PROGRAM WILL EXECUTE
* THE SAME WAY.
*
SERIES LARP AR4
       CNFP           ; CONFIG BLOCK B0 AS PROGRAM MEMORY.
       LACK 9         ; SET COUNTER TO 9.
       SACL NUM       ; (NUM) = 9.
       LRLK AR4,>300  ; POINT AT BEGINNING OF DATA.
       MPYK >0        ; CLEAR P REGISTER.
       ZAC            ; CLEAR ACCUMULATOR.
       RPT  NUM       ; EXECUTE NEXT INSTRUCTION 10 TIMES.
       MAC  >FF00,*+  ; MULTIPLY-ACCUMULATE; INCREMENT AR4.
       APAC
       RET            ; RETURN TO MAIN PROGRAM.
```

5.2.5 Computed GOTOs

Processing may be executed in a time- and process-dependent or selected way. Following a specific time or data processing path may then result in selecting one of several processing options.

A simple computed GOTO can be programmed in the TMS320C2x by using the CALA instruction. This instruction uses the contents of the accumulator as the direct address of the call. Thus, the call address can be computed in the ALU, as shown in Example 5-8.

Example 5-8. Computed GOTO

```
* TASK CONTROLLER
*
* THIS MAIN TASK ROUTINE CONTROLS THE ORDER OF EXECUTION
* AND SCHEDULING OF TASKS. WHEN AN INTERRUPT OCCURS, THE
* INTERRUPT SERVICE ROUTINE IS EXECUTED TO PROCESS THE INPUT
* AND OUTPUT DATA SAMPLES. AFTER THE INTERRUPT SERVICE
* ROUTINE HAS COMPLETED, THE PROCESSOR BEGINS EXECUTION WITH
* THE INSTRUCTION FOLLOWING THE IDLE INSTRUCTION. THIS
* ROUTINE SELECTS THE TASK APPROPRIATE FOR THE CURRENT
* SAMPLE CYCLE, CALLS THE TASK AS A SUBROUTINE, AND BRANCHES
* BACK TO THE IDLE TO WAIT FOR THE NEXT SAMPLE INTERRUPT
* WHEN THE SCHEDULED TASK HAS COMPLETED EXECUTION.
*
WAIT   IDLE             ; WAIT FOR SAMPLE INTERRUPT.
       LAC  SAMPLE      ; FETCH SAMPLE COUNT VALUE.
       SUB  ONE         ; DECREMENT THE SAMPLE COUNT.
       BGEZ OVRSAM      ; TEST FOR END OF BAUD INTERVAL.
       LACK 15          ; INIT COUNT FOR NEW BAUD INTERVAL.
OVRSAM SACL SAMPLE      ; SAVE NEW COUNT VALUE.
       ADLK TSKSEQ      ; ADD TASK TABLE BASE ADDRESS.
       TBLR TEMP        ; READ SUBROUTINE TASK ADDRESS.
       LAC  TEMP        ; LOAD ACCUMULATOR FOR TASK CALL.
       CALA             ; EXECUTE APPROPRIATE TASK.
       B    WAIT
*
TSKSEQ EQU  $
       DATA DUMMY       ; 15 - UNUSED CYCLE
       DATA DUMMY       ; 14 - UNUSED CYCLE
       DATA DUMMY       ; 13 - UNUSED CYCLE
       DATA DUMMY       ; 12 - UNUSED CYCLE
       DATA BDCLK2      ; 11 - COMPUTE ENERGY E(11)
       DATA DUMMY       ; 10 - UNUSED CYCLE
       DATA OUT         ; 9 - COMMUNICATE WITH U-CONTROLLER
       DATA DECODE      ; 8 - DECODE/GET SCRAMBLED DIBIT
       DATA DEMODB      ; 7 - DEMODULATE IN MIDDLE OF BAUD
       DATA DUMMY       ; 6 - UNUSED CYCLE
       DATA AGCUPT      ; 5 - UPDATE AGC EVERY 3RD BAUD
       DATA DUMMY       ; 4 - UNUSED CYCLE
       DATA BDCLK1      ; 3 - COMPUTE ENERGY E(3)
       DATA DUMMY       ; 2 - UNUSED CYCLE
       DATA DUMMY       ; 1 - UNUSED CYCLE
       DATA DUMMY       ; 0 - UNUSED CYCLE
```

5.3 Interrupt Service Routine

Interrupts on the TMS320C2x are prioritized and vectored. When an interrupt occurs, the corresponding flag is set in the Interrupt Flag Register (IFR). If the corresponding bit in the Interrupt Mask Register (IMR) is set and interrupts are enabled (INTM=0), then interrupt processing begins.

When the interrupt vector is loaded into the program counter, interrupts are disabled (INTM=1) and a branch is made to the appropriate routine via the branch instruction stored at the associated vector location. Since all interrupts are disabled, interrupt processing may proceed without further interruption unless the interrupt service routine (ISR) re-enables interrupts.

Unless the interrupt service routines are simple I/O handlers, the processing in each ISR generally must assure that the processor context is preserved during execution. The context must be saved before executing the routine itself and restored when the routine is finished. A common routine or routines individualized for each interrupt may be used to secure the context of the processor during interrupt processing. Context switching is also useful for subroutine calls, especially when extensive use is made of the stack or auxiliary registers. Code examples of context switching and an interrupt service routine are provided in this section.

5.3.1 Context Switching

Context switching, commonly required when processing a subroutine call or interrupt, may be quite extensive or simple, depending on the system requirements. On the TMS320C2x, the program counter is stored automatically on the hardware stack. If there is any important information in the other TMS320C2x registers, such as the status or auxiliary registers, these must be saved by software command. A stack in data memory, identified by an auxiliary register, is useful for storing the machine state when processing interrupts.

Examples of saving and restoring the state of the TMS32020 are given in Example 5-9 and Example 5-10. Auxiliary register 4 (AR4) is used in both examples as the stack pointer. As the stack grows, it expands into lower memory addresses. The registers saved are the status registers (ST0 and ST1), accumulator (ACCH and ACCL), product register (PR), temporary register (TR), all four levels of the hardware stack, and the auxiliary registers (AR0 through AR4).

Example 5-9. Context Save (TMS32020)

```
      TITL 'CONTEXT SAVE'
      DEF  SAVE
*
* CONTEXT SAVE ON SUBROUTINE CALL OR INTERRUPT.
*
* ASSUME AR4 IS THE STACK POINTER AND AR4 = 128.
*
SAVE  LARP 4        ; (ARP) -> ARB, 4 -> ARP,AR4 = 128
      MAR  *-       ;                        AR4 = 127
*
* SAVE THE STATUS REGISTERS.
      SST1 *-       ; ST1 -> (127),          AR4 = 126
      SST  *-       ; ST0 -> (126),          AR4 = 125
*
* SAVE THE ACCUMULATOR.
      SACH *-       ; ACCH -> (125),         AR4 = 124
      SACL *-       ; ACCL -> (124),         AR4 = 123
*
* SAVE THE P REGISTER.
      SPM  0        ; NO SHIFT ON PR OUTPUT
      PAC
      SACH *-       ; PRH -> (123),          AR4 = 122
      SACL *-       ; PRL -> (122),          AR4 = 121
*
* SAVE THE T REGISTER.
      MPYK >1
      PAC
      SACL *-       ; TR  -> (121),          AR4 = 120
*
* SAVE ALL FOUR LEVELS OF THE HARDWARE STACK.
      RPTK 3
      POPD *-       ; TOS   (4) -> (120),    AR4 = 119
                    ; STACK(3) -> (119),     AR4 = 118
                    ; STACK(2) -> (118),     AR4 = 117
                    ; BOS   (1) -> (117),    AR4 = 116
*
* SAVE AUXILIARY REGISTERS AR0 THROUGH AR3.
      SAR  AR0,*-   ; AR0 -> (116),          AR4 = 115
      SAR  AR1,*-   ; AR1 -> (115),          AR4 = 114
      SAR  AR2,*-   ; AR2 -> (114),          AR4 = 113
      SAR  AR3,*-   ; AR3 -> (113),          AR4 = 112
*
* SAVE IS COMPLETE.
```

Example 5-10. Context Restore (TMS32020)

```
      TITL 'CONTEXT RESTORE'
      DEF  RESTOR
*
* CONTEXT RESTORE AT THE END OF A SUBROUTINE OR INTERRUPT.
*
* ASSUME AR4 IS THE STACK POINTER AND AR4 = 112.
*
RESTORLARP 4       ; (ARP) -> ARB, 4 -> ARP,AR4 = 112
      MAR  *+      ;                        AR4 = 113
*
* RESTORE AUXILIARY REGISTERS AR0 THROUGH AR3.
      LAR  AR3,*+  ; (113) -> AR3,          AR4 = 114
      LAR  AR2,*+  ; (114) -> AR2,          AR4 = 115
      LAR  AR1,*+  ; (115) -> AR1,          AR4 = 116
      LAR  AR0,*+  ; (116) -> AR0,          AR4 = 117
*
* RESTORE ALL FOUR LEVELS OF THE HARDWARE STACK.
      RPTK 3
      PSHD *+      ; (117) -> BOS  (1),     AR4 = 118
                   ; (118) -> STACK(2),     AR4 = 119
                   ; (119) -> STACK(3),     AR4 = 120
                   ; (120) -> TOS  (4),     AR4 = 121
*
* THE RETURN PC IS NOW ON THE HARDWARE STACK FOR THE
* RET INSTRUCTION. NOTE THAT THE LOWER 16 BITS OF THE
* P REGISTER MUST BE LOADED VIA THE T REGISTER AND THAT
* THE STACK POINTER IS POINTING AT THE VALUE TO BE LOADED
* IN THE T REGISTER.
*
* RESTORE THE LOW P REGISTER.
      MAR  *+      ;                        AR4 = 122
      LT   *-      ; (122) -> TR,           AR4 = 121
      MPYK >1      ; (TR)  -> PRL,          AR4 = 121
*
* RESTORE THE T REGISTER.
      LT   *+      ; (121) -> TR,           AR4 = 122
      MAR  *+      ;                        AR4 = 123
*
* RESTORE THE HIGH P REGISTER.
      LPH  *+      ; (123) -> PRH,          AR4 = 124
*
* RESTORE THE ACCUMULATOR.
      ZALS *+      ; (124) -> ACCL,         AR4 = 125
      ADDH *+      ; (125) -> ACCH,         AR4 = 126
*
* RESTORE THE STATUS REGISTERS.
      LST  *+      ; (126) -> ST0,          AR4 = 127
      LST1 *       ; (127) -> ST1,          AR4 = 128
*
* RESTORE IS COMPLETE.
      EINT         ; ENABLE INTERRUPTS.
      RET          ; RETURN TO INTERRUPTS OR CALLING ROUTINE.
```

Examples of saving and restoring the state of the TMS320C25 are given in Example 5-11 and Example 5-12. Auxiliary register 7 (AR7) is used in both examples as the stack pointer. As the stack grows, it expands into lower memory addresses. The registers saved are the status registers (ST0 and ST1), accumulator (ACCH and ACCL), product register (PR), temporary register (TR), all eight levels of the hardware stack, and the auxiliary registers (AR0 through AR6).

The routines in Example 5-11 and Example 5-12 are protected against interrupts, allowing context switches to be nested. This is accomplished by the use of the MAR *- and MAR *+ instructions at the beginning of the context save and context restore routines, respectively. Note that the last instruction of the context save decrements AR7 while the context restore is completed with an additional increment of AR7. This prevents the loss of data if a context save or restore routine is interrupted.

Example 5-11. Context Save (TMS320C25)

```
        TITL 'CONTEXT SAVE'
        DEF  SAVE
*
* CONTEXT SAVE ON SUBROUTINE CALL OR INTERRUPT.
*
* ASSUME AR7 IS THE STACK POINTER AND AR7 = 128.
*
SAVE    LARP AR7       ; (ARP) -> ARB, 7 -> ARP,AR7 = 128
        MAR  *-        ;                        AR7 = 127
*
* SAVE THE STATUS REGISTERS.
        SST1 *-        ; ST1 -> (127),          AR7 = 126
        SST  *-        ; ST0 -> (126),          AR7 = 125
*
* SAVE THE ACCUMULATOR.
        SACH *-        ; ACCH -> (125),         AR7 = 124
        SACL *-        ; ACCL -> (124),         AR7 = 123
*
* SAVE THE P REGISTER.
        SPM  0         ; NO SHIFT ON PR OUTPUT
        SPH  *-        ; PRH -> (123),          AR7 = 122
        SPL  *-        ; PRL -> (122),          AR7 = 121
*
* SAVE THE T REGISTER.
        MPYK 1         ; PR = TR
        SPL  *-        ; TR  -> (121),          AR7 = 120
*
* SAVE ALL EIGHT LEVELS OF THE HARDWARE STACK.
        RPTK 7
        POPD *-        ; TOS  (8) -> (120),     AR7 = 119
*                      ; STACK(7) -> (119),     AR7 = 118
*                      ; STACK(6) -> (118),     AR7 = 117
*                      ; STACK(5) -> (117),     AR7 = 116
*                      ; STACK(4) -> (116),     AR7 = 115
*                      ; STACK(3) -> (115),     AR7 = 114
*                      ; STACK(2) -> (114),     AR7 = 113
*                      ; BOS  (1) -> (113),     AR7 = 112
*
* SAVE AUXILIARY REGISTERS AR0 THROUGH AR6.
        SAR  AR0,*-  ; AR0 -> (112),            AR7 = 111
        SAR  AR1,*-  ; AR1 -> (111),            AR7 = 110
        SAR  AR2,*-  ; AR2 -> (110),            AR7 = 109
        SAR  AR3,*-  ; AR3 -> (109),            AR7 = 108
        SAR  AR4,*-  ; AR4 -> (108),            AR7 = 107
        SAR  AR5,*-  ; AR5 -> (107),            AR7 = 106
        SAR  AR6,*-  ; AR6 -> (106),            AR7 = 105
*
* SAVE IS COMPLETE.
```

Example 5-12. Context Restore (TMS320C25)

```
       TITL 'CONTEXT RESTORE'
       DEF  RESTOR
*
* CONTEXT RESTORE AT THE END OF A SUBROUTINE OR INTERRUPT.
*
* ASSUME AR7 IS THE STACK POINTER AND AR7 = 105.
*
RESTOR LARP AR7    ; (ARP) -> ARB, 7 -> ARP,AR7 = 105
       MAR  *+     ;                        AR7 = 106
*
* RESTORE AUXILIARY REGISTERS AR0 THROUGH AR6.
       LAR  AR6,*+ ; (106) -> AR6,          AR7 = 107
       LAR  AR5,*+ ; (107) -> AR5,          AR7 = 108
       LAR  AR4,*+ ; (108) -> AR4,          AR7 = 109
       LAR  AR3,*+ ; (109) -> AR3,          AR7 = 110
       LAR  AR2,*+ ; (110) -> AR2,          AR7 = 111
       LAR  AR1,*+ ; (111) -> AR1,          AR7 = 112
       LAR  AR0,*+ ; (112) -> AR0,          AR7 = 113
*
* RESTORE ALL EIGHT LEVELS OF THE HARDWARE STACK.
       RPTK 7
       PSHD *+     ; (113) -> BOS  (1),     AR7 = 114
*                  ; (114) -> STACK(2),     AR7 = 115
*                  ; (115) -> STACK(3),     AR7 = 116
*                  ; (116) -> STACK(4),     AR7 = 117
*                  ; (117) -> STACK(5),     AR7 = 118
*                  ; (118) -> STACK(6),     AR7 = 119
*                  ; (119) -> STACK(7),     AR7 = 120
*                  ; (120) -> TOS  (8),     AR7 = 121
*
* THE RETURN PC IS NOW ON TOP OF THE STACK FOR THE RET
* INSTRUCTION. THE LOWER 16 BITS OF THE P REGISTER MUST
* BE LOADED VIA THE T REGISTER AND THE STACK POINTER BE
* POINTING AT THE VALUE TO BE LOADED IN THE T REGISTER.
*
* RESTORE THE LOW P REGISTER.
       MAR  *+     ; SKIP T REGISTER,       AR7 = 122
       LT   *-     ; (122) -> TR,           AR7 = 121
       MPYK 1      ; (TR)  -> PRL
*
* RESTORE THE T REGISTER.
       LT   *+     ; (121) -> TR,           AR7 = 122
       MAR  *+     ; SKIP P REGISTER LOW,   AR7 = 123
*
* RESTORE THE HIGH P REGISTER.
       LPH  *+     ; (123) -> PRH,          AR7 = 124
*
* RESTORE THE ACCUMULATOR.
       ZALS *+     ; (124) -> ACCL,         AR7 = 125
       ADDH *+     ; (125) -> ACCH,         AR7 = 126
*
* RESTORE THE STATUS REGISTERS.
       LST  *+     ; (126) -> ST0,          AR7 = 127
       LST1 *+     ; (127) -> ST1,          AR7 = 128
*
* RESTORE IS COMPLETE.
       EINT        ; ENABLE INTERRUPTS.
       RET         ; RETURN TO INTERRUPTS OR CALLING ROUTINE.
```

5.3.2 Interrupt Priority

Interrupts on the TMS320C2x are prioritized in hardware. This allows interrupts that occur simultaneously to be serviced in a prioritized order. Sometimes priority may be determined by frequency or rate of occurrence. An infrequent, but lengthy, interrupt service routine (ISR) might need to be interrupted by a more frequently occurring interrupt. In the routine of Example 5-13, the ISR for $\overline{\text{INT}}$1 temporarily modifies the interrupt mask register (IMR) to permit interrupt processing when an interrupt on $\overline{\text{INT}}$0 (but no other interrupt) occurs. When the routine has finished processing, the IMR is restored to its original state. Example 5-13 is written for the TMS320C25; however, AR4 can be substituted for AR7 when using the TMS32020.

Example 5-13. Interrupt Service Routine

```
       TITL 'INTERRUPT SERVICE ROUTINE'
       DEF  ISR1
       REF  IMR
*
* INTERRUPT PROCESSING FOR EXTERNAL INTERRUPT INT1-.
*
* THIS ROUTINE MAY BE INTERRUPTED BY AN INTERRUPT FROM THE
* EXTERNAL INTERRUPT INT0-, BUT NO OTHER.
*
ISR1   LARP AR7     ; 7 --> ARP
       MAR  *-      ;               AR7 = AR7 - 1
       SST1 *-      ; ST1  --> *AR7, AR7 = AR7 - 1
       SST  *-      ; ST0  --> *AR7, AR7 = AR7 - 1
       SACH *-      ; ACCH --> *AR7, AR7 = AR7 - 1
       SACL *-      ; ACCL --> *AR7, AR7 = AR7 - 1
       LDPK 0       ; DP = 0
       PSHD IMR     ; IMR --> TOS
       LACK >0001   ; MASK FOR INT0-
       AND  IMR     ; MASK CURRENT IMR CONTENTS.
       SACL IMR     ; ACC --> IMR
       EINT         ; ENABLE INTERRUPTS.
*
* MAIN PROCESSING SECTION FOR ISR1.
        .
        .
        .
*
       DINT         ; DISABLE INTERRUPTS.
       LDPK 0       ; DP = 0
       POPD IMR     ; TOS --> IMR
       LARP AR7     ; 7 --> ARP
       MAR  *+      ;               AR7 = AR7 + 1
       ZALS *+      ; *AR7 --> ACCL, AR7 = AR7 + 1
       ADDH *+      ; *AR7 --> ACCH, AR7 = AR7 + 1
       LST  *+      ; *AR7 --> ST0,  AR7 = AR7 + 1
       LST1 *+      ; *AR7 --> ST1,  AR7 = AR7 + 1
       EINT         ; ENABLE INTERRUPTS.
       RET
```

5.4 Memory Management

The structure of the TMS320C2x memory map is programmable and can vary for each application. Instructions are provided for moving blocks of data or program memory, configuring a block of on-chip data RAM as program memory, and defining part of external data memory as global. Explanations and examples of moving, configuring, and manipulating memory are provided in this section.

5.4.1 Block Moves

Since the TMS320C2x directly addresses a large amount of memory, blocks of data or program code can be stored off-chip in slow memories and then loaded on-chip for faster execution. Data can also be moved from on-chip to off-chip for storage or for multiprocessor data transfers.

The BLKD and BLKP instructions facilitate memory-to-memory block moves on the TMS320C2x. The BLKD instruction moves a block within data memory as shown in Example 5-14. Data may also be transferred between data memory and program memory by means of the TBLR and TBLW instructions. The instructions IN and OUT are used to transfer data between the data memory and the I/O space.

Example 5-14. Moving External Data to Internal Data Memory with BLKD

```
* THIS ROUTINE USES THE BLKD INSTRUCTION TO MOVE A BLOCK OF
* EXTERNAL DATA MEMORY (DATA PAGES 8 AND 9) TO INTERNAL BLOCK
* B1 (DATA PAGES 6 AND 7).
*
MOVED   LARP AR2
        LRLK AR2,>300 ; DESTINATION IS BLOCK B1 IN RAM.
        RPTK 255      ; REPEAT NEXT INSTRUCTION 256 TIMES.
        BLKD >400,*+  ; MOVE EXTERNAL BLOCK TO BLOCK B1.
        RET           ; RETURN TO MAIN PROGRAM.
```

For systems that have external program memory but no external data memory, BLKP can be used to move program memory blocks into data memory. Example 5-15 demonstrates how to use the BLKP instruction.

Example 5-15. Moving Program Memory to Data Memory with BLKP

```
* THIS ROUTINE USES THE BLKP INSTRUCTION TO MOVE DATA VALUES
* FROM PROGRAM MEMORY INTO DATA MEMORY. SPECIFICALLY, THE
* VALUES IN LOCATIONS 2, 3, 4, AND 5 IN PROGRAM MEMORY ARE
* MOVED TO LOCATIONS 512, 513, 514, AND 515 IN DATA MEMORY.
*
MOVEP   LARP AR2      ; SET REFERENCE FOR INDIRECT ADDRESSING.
        LRLK AR2,512 ; LOAD BEGINNING OF BLOCK B0 IN AR2.
        RPTK 3       ; SET UP LOOP.
        BLKP >2,*+   ; PUT DATA INTO DATA RAM.
        RET          ; RETURN TO MAIN PROGRAM.
```

Another method for transferring data from program memory into data memory makes use of the TBLR instruction. By using the TBLR instruction, a calculated, rather than predetermined, location of a block of data in program memory may be specified for transfer. A routine using this approach is shown in Example 5-16.

Example 5-16. Moving Program Memory to Data Memory with TBLR

```
* THIS ROUTINE USES THE TBLR INSTRUCTION TO MOVE DATA VALUES
* FROM PROGRAM MEMORY INTO DATA MEMORY. BY USING THIS ROUTINE,
* THE PROGRAM MEMORY LOCATION IN THE ACCUMULATOR FROM WHICH
* DATA IS TO BE MOVED TO A SPECIFIC DATA MEMORY LOCATION CAN
* BE SPECIFIED. ASSUME THAT THE ACCUMULATOR CONTAINS THE
* ADDRESS IN PROGRAM MEMORY FROM WHICH TO TRANSFER THE DATA.
*
TABLER LARP AR3
       LRLK AR3,380 ; DESTINATION ADDRESS = PAGE 7.
       RPTK 127     ; TRANSFER 128 VALUES.
       TBLR *+      ; MOVE DATA INTO DATA RAM.
       RET          ; RETURN TO CALLING PROGRAM.
```

In cases where systems require that temporary storage be allocated in the program memory, TBLW can be used to transfer data from internal data memory to external program memory. The code in Example 5-17 demonstrates how this may be accomplished.

Example 5-17. Moving Internal Data Memory to Program Memory with TBLW

```
* THIS ROUTINE USES THE TBLW INSTRUCTION TO MOVE DATA VALUES
* FROM INTERNAL DATA MEMORY TO EXTERNAL PROGRAM MEMORY. THE
* CALLING ROUTINE MUST SPECIFY THE DESTINATION PROGRAM MEMORY
* ADDRESS IN THE ACCUMULATOR. ASSUME THAT THE ACCUMULATOR
* CONTAINS THE ADDRESS IN PROGRAM MEMORY INTO WHICH THE DATA
* IS TRANSFERRED.
*
TABLEW LARP AR4
       LRLK AR4,380 ; SOURCE ADDRESS = PAGE 7.
       RPTK 127     ; TRANSFER 128 VALUES.
       TBLW *+      ; MOVE DATA TO EXTERNAL PROGRAM RAM.
       RET          ; RETURN TO CALLING PROGRAM.
```

The IN and OUT instructions are used to transfer data between the data memory and the I/O space, as shown in Example 5-18 and Example 5-19.

Example 5-18. Moving Data from I/O Space into Data Memory with IN

```
* THIS ROUTINE USES THE IN INSTRUCTION TO MOVE DATA VALUES
* FROM THE I/O SPACE INTO DATA MEMORY. DATA ACCESSED FROM
* I/O PORT 15 IS TRANSFERRED TO SUCCESSIVE MEMORY LOCATIONS
* ON DATA PAGE 5.
*
INPUT  LARP AR2
       LRLK AR2,>2C0 ; DESTINATION ADDRESS = PAGE 5.
       RPTK 63       ; TRANSFER 64 VALUES.
       IN   PA15,*+  ; MOVE DATA INTO DATA RAM.
       RET           ; RETURN TO CALLING PROGRAM.
```

Example 5-19. Moving Data from Data Memory to I/O Space with OUT

```
* THIS ROUTINE USES THE OUT INSTRUCTION TO MOVE DATA VALUES
* FROM THE DATA MEMORY TO THE I/O SPACE. DATA IS TRANSFERRED
* TO I/O PORT 8 FROM SUCCESSIVE MEMORY LOCATIONS ON DATA
* PAGE 4.
*
OUTPUT LARP  AR4
       LRLK  AR4,>200 ; SOURCE ADDRESS = PAGE 4.
       RPTK  63       ; TRANSFER 64 VALUES.
       OUT   PA8,*+   ; MOVE DATA FROM DATA RAM.
       RET            ; RETURN TO CALLING PROGRAM.
```

5.4.2 Configuring On-Chip RAM

The large amount of external memory and the configurability of on-chip RAM simplify the downloading of data or program memory into the TMS320C2x. Also, since data in the RAM is preserved when redefining on-chip RAM, block B0 can be configured dynamically as either data or program memory. Figure 5-1 illustrates the changes in on-chip RAM when switching configurations.

On-chip memory is configured by a reset or by the CNFD and CNFP instructions. Block B0 is configured as data memory by executing CNFD or reset. A CNFP instruction configures block B0 as program memory.

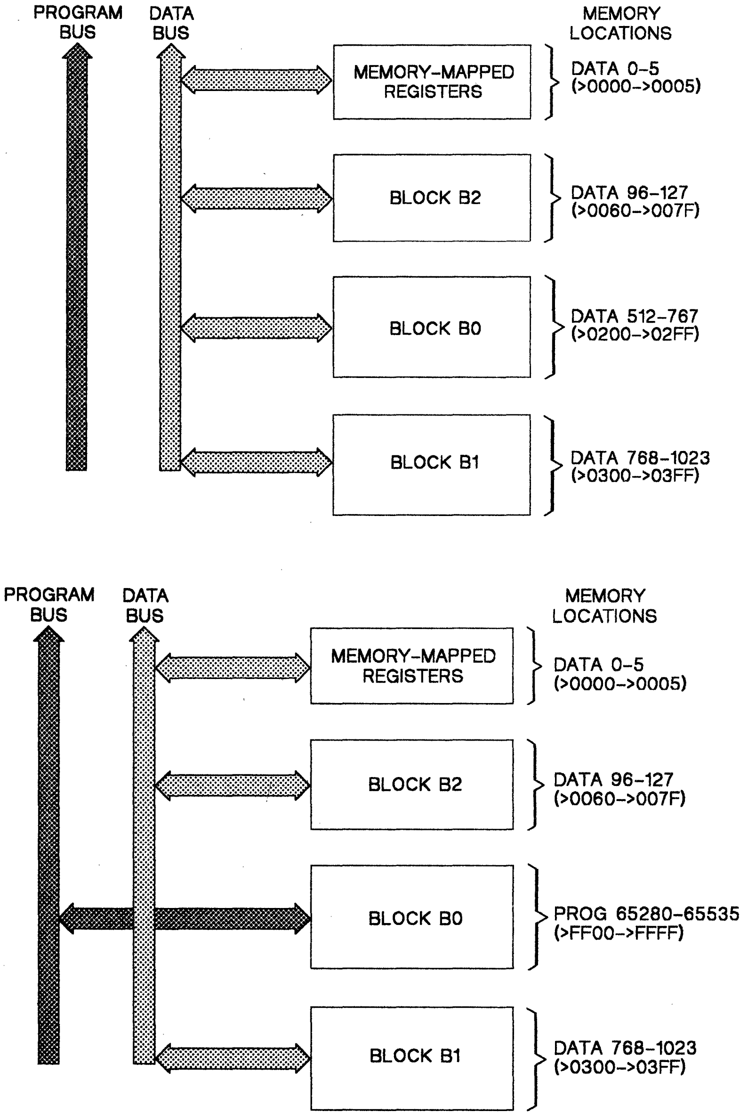


Figure 5-1. On-Chip RAM Configurations

Configuring block B0 as program memory is useful for implementing adaptive filters or similar applications at full speed with only on-chip memories. Example 5-20 illustrates the use of the configuration modes to utilize block B0 as data and program memory while executing from on-chip program ROM.

Example 5-20. Configuring and Using On-Chip RAM

```
       TITL 'ADAPTIVE FILTER'
       DEF  ADPFIR
       DEF  X,Y
*
* THIS 128-TAP ADAPTIVE FIR FILTER USES ON-CHIP MEMORY BLOCK
* B0 FOR COEFFICIENTS AND BLOCK B1 FOR DATA SAMPLES. THE
* NEWEST INPUT SHOULD BE IN MEMORY LOCATION X WHEN CALLED.
* THE OUTPUT WILL BE IN MEMORY LOCATION Y WHEN RETURNED.
*
COEFFP EQU  >FF00            ; B0 PROGRAM MEMORY ADDRESS
COEFFD EQU  >0200            ; B0 DATA MEMORY ADDRESS
ONE    EQU  >7A              ; CONSTANT ONE        (DP=6)
BETA   EQU  >7B              ; ADAPTATION CONSTANT (DP=6)
ERR    EQU  >7C              ; SIGNAL ERROR        (DP=6)
ERRF   EQU  >7D              ; ERROR FUNCTION      (DP=6)
Y      EQU  >7E              ; FILTER OUTPUT       (DP=6)
X      EQU  >7F              ; NEWEST DATA SAMPLE  (DP=6)
FRSTAP EQU  >0380            ; NEXT NEWEST DATA SAMPLE
LASTAP EQU  >03FF            ; OLDEST DATA SAMPLE
*
* FINITE IMPULSE  RESPONSE (FIR) FILTER.
*
ADPFIR CNFP                  ; CONFIGURE B0 AS PROGRAM:
       MPYK 0                ; Clear the P register.
       LAC  ONE,14           ; Load output rounding bit.
       LARP AR3
       LRLK AR3,LASTAP       ; Point to the oldest sample.
FIR    RPTK 127
       MACD COEFFP,*-        ; 128-tap FIR filter.
       CNFD                  ; CONFIGURE B0 AS DATA:
       APAC
       SACH Y,1              ; Store the filter output.
       NEG
       ADD  X,15             ; Add the newest input.
       SACH ERR,1            ; err(n) = x(n) - y(n)
*
* LMS ADAPTATION OF FILTER COEFFICIENTS.
*
       LT   ERR
       MPY  BETA             ; 128-TAP FIR FILTER.
       PAC                   ; errf(n) = beta * err(n)
       ADD  ONE,14           ; ROUND THE RESULT.
       SACH ERRF,1
       LARP AR3
       LARK AR1,127          ; 128 COEFFICIENTS TO UPDATE.
       LRLK AR2,COEFFD       ; POINT TO THE COEFFICIENTS.
       LRLK AR3,LASTAP       ; POINT TO THE DATA SAMPLES.
       DMOV X                ; INCLUDE NEWEST SAMPLE.
       LT   ERRF
       MPY  *-,AR2           ; P = 2*beta*err(n)*x(n-k)
*
ADAPT  ZALH *,AR3            ; LOAD ACCH WITH ak(n).
       ADD  ONE,15           ; LOAD ROUNDING BIT.
       APAC                  ; ak(n+1) = ak(n) + P
*      MPY  *-,AR2           ; P = 2*beta*err(n)*x(n-k)
       SACH *+,0,AR1         ; STORE ak(n+1).
       BANZ ADAPT,*-,AR2     ; END OF LOOP TEST.
       RET                   ; RETURN TO CALLING ROUTINE.
```

Note that a more definitive example of the use of the TMS320C25 for adaptive filtering is provided in Section 5.7.3.

5.4.3 Using On-Chip RAM for Program Execution

In using on-chip memory (block B0) for program execution, this memory must first be loaded with executable code from external memories while configured as data memory. On-chip execution is initiated by using the CNFP instruction to reconfigure block B0 as program memory and performing a branch or call to an on-chip RAM address. By configuring block B0 as program memory and executing from this internal memory, full-speed execution can be achieved in systems using slower external memory. Example 5-21 illustrates how a program may be written to be loaded into and executed from on-chip memory.

One group of instructions, the branch/call instructions, are impacted by the location of execution. Normally, by using labels, the assembler properly determines the location to which a branch is taken. Since the code is relocated prior to execution from on-chip memory, it is necessary to alter the address determined by the assembler for branch instructions. This alteration is necessary so that the branch address that is determined can be consistent with the address space used during execution. In Example 5-21, this is accomplished by adding an offset value (OFFSET) to the branch label representing the destination address in the operand field for each branch instruction. The offset address is determined by use of an EQU (equate) directive that subtracts the assembler location of the code to be relocated (equivalent to base-0 addressing) from the base address of the relocation address (internal block B0 address in this case).

Example 5-21. Program Execution from On mory

```
       AORG 0
RESET  B    INIT
*
* BRANCHES FOR EXTERNAL OR INTERNAL INTERRUPTS FOLLOW HERE AT
* THE DESIGNATED LOCATIONS AS REQUIRED.
*
       AORG >20
*
* A BRANCH INSTRUCTION AT PROGRAM MEMORY LOCATION 0 DIRECTS
* PROCESSOR EXECUTION HERE.
*
* INITIALIZE THE PROCESSOR.
*
INIT   ROVM                  ; DISABLE OVERFLOW MODE.
       SSXM                  ; SET SIGN EXTENSION.
       LDPK 0                ; POINT DP REGISTER TO DATA MEMORY PAGE 0.
       SPM  0                ; NO SHIFT ON PRODUCT REGISTER OUTPUT.
       LARP AR4              ; USE AUXILIARY REGISTER 4 (SET ARP = 4).
       LARK AR4,PRD          ; POINT AR4 TO PERIOD REGISTER.
       LALK >FFFF            ; SET ACCUMULATOR TO >FFFF.
       SACL *+               ; LOAD PERIOD REGISTER WITH MAXIMUM VALUE.
       SACL *+               ; ENABLE ALL INTERRUPTS VIA IMR.
       ZAC                   ; CLEAR ACCUMULATOR.
       SACH *                ; CLEAR GREG TO MAKE ALL MEMORY LOCAL.
*
* LOAD TIME-CRITICAL CODE FROM EXTERNAL SLOW MEMORY TO INTERNAL RAM.
*
       LARP AR1              ; USE AUXILIARY REGISTER 1 (SET ARP = 1).
       LRLK AR1,BLK0         ; POINT AR1 TO RECONFIGURABLE BLOCK B0.
       RPTK PROGL-1          ; LOAD REPEAT COUNTER WITH BLOCK LENGTH.
       BLKP PROG,*+          ; MOVE CODE FROM PROG MEMORY TO ON-CHIP RAM.
*
* INITIALIZE PARAMETERS FOR EXECUTION.
*
       LDPK 6                ; POINT DP REGISTER TO DATA MEMORY PAGE 6.
       LACK 1                ; SET ACCUMULATOR TO >0001.
       SACL ONE              ; STORE VALUE OF 1.
       LRLK AR1,BLK0+PROGL   ; POINT AR1 TO INTERNAL MEMORY ADDRESS.
       RPTK COEFL-1          ; LOAD REPEAT COUNTER WITH BLOCK LENGTH.
       BLKP COEF,*+          ; MOVE DATA FROM PROG MEMORY TO ON-CHIP RAM.
       CNFP                  ; CONFIGURE BLOCK B0 AS PROGRAM MEMORY.
       LALK >FF00            ; LOAD ACC WITH PROG ADDR IN INTERNAL RAM.
       BACC                  ; BRANCH TO ON-CHIP EXECUTION ADDRESS.
*
* SIGNAL PROCESSING CODE TO BE EXECUTED FROM ON-CHIP RAM.
*
PROG   EQU  $
LPTS   BIOZ GET-PROG+ONCHIP   ; WAIT FOR INPUT SIGNAL.
       B    LPTS-PROG+ONCHIP  ; BRANCH IF NO SIGNAL.
GET    OUT  FILOUT,PA2        ; OUTPUT LAST FILTER OUTPUT.
       IN   FILIN,PA2         ; INPUT NEW SIGNAL SAMPLE.
       LRLK AR1,BLK1+SIGNAL   ; POINT AR1 TO SIGNAL DATA TO PROCESS.
       ZAC                    ; CLEAR THE ACCUMULATOR.
       MPYK 0                 ; CLEAR THE P REGISTER.
```

```
        RPTK 15                   REPEAT MACD INSTRUCTION FOR 16 TAPS.
        MACD ONCHIP+COEFF,        MULTIPLY, ACCUMULATE, SAMPLE DELAY.
        APAC                    ; ACCUMULATE THE LAST PRODUCT.
        SACH FILOUT,1           ; SAVE THE RESULT.
        B    LPTS-PROG+ONCHIP   ; LOOP TO WAIT FOR NEXT SAMPLE.
PROGE   EQU  $                  ; SAVE THE RESULT.
PROGL   EQU  PROGE-PROG         ; PROGRAM CODE LENGTH.
ONCHIP  EQU  >FF00              ; BASE ADDRESS EXECUTION.
*
* COEFFICIENT DATA TO BE LOADED INTO ON-CHIP RAM.
*
COEF    DATA 385,-1196,1839,-2009
        DATA 1390,407,-4403,19958
        DATA 19958,-4403,407,1390
        DATA -2009,1839,-1196,385
COEFE   EQU  $
COEFL   EQU  COEFE-COEF         ; COEFFICIENT DATA LENGTH.
*
* INTERNAL MEMORY CONSTANTS
*
BLK0    EQU  >200
BLK1    EQU  >300
*
* DATA PAGE 0 (BLOCK B2) - DATA MEMORY LABELS.
*
        DORG 0
DRR     BSS  1                ; SERIAL PORT DATA RECEIVE REGISTER.
DXR     BSS  1                ; SERIAL PORT DATA TRANSMIT REGISTER.
TIM     BSS  1                ; TIMER REGISTER.
PRD     BSS  1                ; PERIOD REGISTER.
IMR     BSS  1                ; INTERRUPT MASK REGISTER.
GREG    BSS  1                ; GLOBAL MEMORY ALLOCATION REGISTER.
*
* DATA PAGE 4 (BLOCK B0) - DATA MEMORY LABELS.
*
        DORG 0
B0      BSS  PROGL            ; LOCATIONS FOR INTERNAL PROGRAM CODE.
COEFF   BSS  COEFL            ; LOCATIONS FOR COEFFICIENT MEMORY.
*
* DATA PAGE 6 (BLOCK B1) - DATA MEMORY LABELS.
*
        DORG 0
ONE     BSS  1                ; RESERVED FOR DATA VALUE OF 1.
FILOUT  BSS  1                ; FILTER OUTPUT SIGNAL VALUE.
FILIN   BSS  1                ; FILTER INPUT SIGNAL VALUE.
SIGNAL  BES  14               ; LAST SIGNAL DELAY VALUE.
        END
```

5.5 Fundamental Logical and Arithmetic Operations

Although the TMS320C2x instruction set is oriented toward digital signal processing, the same fundamental operations of a general-purpose processor are included. This section explains basic operations of the TMS320C2x Central Arithmetic Logic Unit (CALU), particularly accumulator operations, the status register effect on data processing, and bit manipulation.

The TMS320C2x provides a complete set of logical operations, including AND, OR, XOR, and CMPL (complement) instructions. This enables the device to perform any logical function. These instructions may be used to perform sign magnitude to two's complement or the reverse conversions.

The contents of the accumulator may be stored in data memory using the SACH and SACL instructions or stored in the stack by using the PUSH instruction. The accumulator may be loaded from data memory using the ZALH and ZALS instructions, which zero the accumulator before loading the data value. The ZAC instruction zeroes the accumulator. POP can be used to restore the accumulator contents from the stack.

The accumulator is also affected by the ABS and NEG instructions. ABS replaces the contents of the accumulator with the absolute value of its contents. NEG generates the arithmetic complement of the accumulator in two's-complement form.

5.5.1 Status Register Effect on Data Processing

Three data processing options allow the ALU to automatically suppress sign extension, manage overflow, or scale product accumulations. These options are enabled or disabled through bits in the status registers. These options function in parallel with normal execution of the instructions and cause no additional machine cycles, therefore no performance overhead.

The sign-extension mode option is used to determine whether or not the shifted data values fetched for ALU operations should be sign-extended. The SXM status bit controls this operation. This bit is set to '1' for enabling sign extension using the SSXM instruction, and set to '0' for suppressing sign extension using the RSXM instruction. This operation affects all the instructions that include a shift of the incoming data value (i.e., ADD, ADDT, ADLK, LAC, LACT, LALK, SBLK, SFR, SUB, and SUBT).

The overflow mode option is used to minimize the effects of an arithmetic overflow by forcing the accumulator to saturate at the largest positive value (or in the case of underflow, the largest negative value). The OVM status bit controls this operation. The overflow mode is enabled by setting the OVM bit to a '1' using the SOVM instruction, and reset using the ROVM instruction. This feature affects all arithmetic operations in the ALU.

The product register shift mode option forces all products to be shifted before they are accumulated. The products can be left-shifted one bit to delete the extra sign bit in the multiply of two 16-bit signed numbers. The products can be left-shifted four bits to delete the extra sign bits in multiplying a 16-bit data value by a 13-bit constant. The product shifter can also be used to shift all products six bits to the right to allow up to 128 product accumulations without the threat of an arithmetic overflow, thereby avoiding the overhead of overflow management. The shifter can be disabled to cause no shift in the product when working with integer or 32-bit precision operations. This also maintains compatibility with TMS320C1x code. These operations are controlled by the value contained in the PM bits of status register ST1. The PM bits are set using the SPM instruction. This feature affects all the instructions that use the product of the multiplier (i.e., APAC, LTA, LTD, LTP, LTS, MAC, MACD, MPYA, MPYS, PAC, SPAC, SPH, SPL, SQRA, and SQRS).

5.5.2 Bit Manipulation

The BIT instruction tests any of the 16 bits of the addressed data word. The specified bit is copied into the TC of the status register. The bit tested is specified by a bit code in the opcode of the instruction. Either the BBZ (branch on TC bit = 0) or BBNZ (branch on TC bit = 1) instructions check the bit and allow branching to a service routine.

Bit testing is useful in control applications where a number of states or conditions may be latched externally and read into the TMS320C2x via an IN instruction. At this point, individual bits can be tested and branches taken for appropriate processing.

Since the BIT instruction requires the bit code to be specified with the instruction, it cannot be placed in a loop to test several different bits of a data word or bits determined by prior processing for efficient use. The TMS320C2x also has a BITT instruction in which the bit code is specified in the T register. Since the T register can easily be modified, BITT may be used to test all bits of a data word if placed within a loop or to test a bit location determined by past processing.

Example 5-22. Using BIT and BBZ

```
* THIS ROUTINE USES THE BIT INSTRUCTION TO TEST THE CONDITION
* OF AN EXTERNAL MUX. BIT 4 DETERMINES THE UTILITY OF THE
* REMAINING DATA. IF ZERO, A COUNTER IS INCREMENTED. IF ONE,
* ADDITIONAL PROCESSING OCCURS AND THE COUNTER IS CLEARED.
* THE ROUTINE IS INVOKED WHENEVER A TIMER INTERRUPT OCCURS.
*
TIME    SST  ST0         ; SAVE STATUS REGISTER ST0.
        LDPK 0
        LARP AR3
        IN   DAT,PA8     ; READ IN VALUE.
        BIT  DAT,>B      ; TEST BIT 4.
        BBZ  INCR        ; BRANCH AND INCREMENT IF POSITIVE.
         .
         .
         .
        LARK AR3,0       ; CLEAR THE COUNTER.
        LST  ST0         ; RELOAD THE STATUS REGISTER.
        EINT             ; ENABLE INTERRUPTS.
        RET              ; RETURN TO INTERRUPTED ROUTINE.
*
INCR    MAR  *+          ; INCREMENT THE COUNTER.
        LST  ST0         ; RELOAD THE STATUS REGISTER.
        EINT             ; ENABLE INTERRUPTS.
        RET              ; RETURN TO INTERRUPTED ROUTINE.
```

Example 5-23. Using BITT and BBNZ

```
* THIS ROUTINE USES THE BITT INSTRUCTION TO TEST THE CONDITION
* OF AN EXTERNAL MUX. A BIT IN THE MUX IS SIGNIFICANT ONLY
* WHEN PRIOR PROCESSING HAS DESIGNATED THE BIT TO BE ACTIVE.
* INDIVIDUAL PROCESSING WILL TAKE PLACE BASED UPON THE STATE
* OF THE TESTED BIT. THE BITS ARE TESTED EACH TIME A TIMER
* INTERRUPT OCCURS.
*
TIME   SST  STO          ; SAVE STATUS REGISTER STO.
       LDPK 0
       LARP AR3
       LAR  AR3,BCNT     ; LOAD COUNT OF ACTIVE BITS.
       LRLK AR3,BTBL     ; LOAD THE BIT TABLE ADDRESS.
       IN   DAT,PA8      ; READ IN VALUE.
       B    LTEST,*-,4
TMLOOP LT   *+,3         ; LOAD BIT CODE.
       BITT DAT          ; TEST SPECIFIED BIT.
       BBNZ LTEST        ; BRANCH IF BIT IS ONE.
        .
        .
        .
LTEST  BANZ TMLOOP,*-,4
       LST  STO          ; RELOAD THE STATUS REGISTER.
       EINT              ; ENABLE INTERRUPTS.
       RET               ; RETURN TO INTERRUPTED ROUTINE.
```

5.6 Advanced Arithmetic Operations

The TMS320C2x provides special instructions that facilitate efficient execution of arithmetic-intensive DSP algorithms, such as MACD, SQRA, SUBC, and NORM. Explanations and examples of how to use these instructions with overflow management, and for data moves, multiplications, division, floating-point arithmetic, indexed addressing, and extended-precision arithmetic are included in this section.

5.6.1 Overflow Management

The TMS320C25 has four features that can be used to handle overflow management. These include the branch on overflow conditions, accumulator saturation (overflow mode), product register right shift, and accumulator right shift. These features provide several options for overflow protection within an algorithm.

A program can branch to an error handler routine on an overflow of the accumulator by using the BV (branch on overflow) instruction or bypass an error handler by using the BNV (branch if no overflow) instruction. These instructions can be performed after any ALU operation that may cause an accumulator overflow.

The overflow mode is a feature useful for DSP applications. This mode simulates the saturation effect characteristic of analog systems. When enabled, any overflow in the accumulator results in the accumulator contents being replaced with the largest positive value (>7FFFFFFF) if the overflowed number is positive, or the largest negative value (>80000000) if negative. The overflow mode is controlled by the OVM bit of status register ST0 and can be changed by the SOVM (set overflow mode), ROVM (reset overflow mode), or LST (load status register) instructions. Overflows can be detected in software by testing the OV (overflow) bit in status register ST0. When a branch is used to test the overflow bit, OV is automatically reset. Note that the OV bit does not function as a carry bit. It is set only when the absolute value of a number is too large to be represented in the accumulator, and it is not reset except by specific instructions.

Another method of overflow management, which applies to multiply-accumulate operations, is the use of the right shifter of the product register. The right shifter, which operates with no cycle overhead, allows up to 128 accumulations without the possibility of an overflow. The least-significant six bits of the product are lost, and the MSBs are filled with sign bits. This feature is initiated by setting the PM bits of status register ST1 to '11' using the SPM or LST1 instructions.

The TMS320C2x also has a right shift of the accumulator (using the SFR instruction) to scale down the accumulator when it nears overflow.

5.6.2 Scaling

Scaling the data coming into the accumulator or already in the accumulator is useful in signal processing algorithms. This is frequently necessary in adaptation or other algorithms that must compute and apply correction factors or normalize intermediate results. Scaling and normalizing are implemented on the TMS320C2x via right and left shifts in the accumulator and shifts of data on the incoming path to the accumulator.

Right and left shifts of the accumulator can be performed using the SFL and SFR instructions. SFL performs a logical left shift. SFR performs logical or arithmetic right shifts depending on the state of the SXM bit in the status register. A one in the SXM bit, corresponding to sign-extension enabled, causes an arithmetic shift to be performed.

In addition to the shift instructions, data can be left-shifted 0 to 15 bits when the accumulator is loaded using a LAC instruction, and left-shifted 0, 1, or 4 bits on the TMS32020 or 0 to 7 bits on the TMS320C25 when storing from the accumulator using SACH or SACL instructions. These shifts can be used for loading numbers into the high 16 bits of the accumulator and renormalizing the result of a multiply. The incoming left shift of 0 to 15 bits can be supplied in the instruction itself or can be taken from the lowest four bits of the T register. Left shifts of data fetched from data memory are available for loading the accumulator (LAC/LACT), adding to the accumulator (ADD/ADDT), and subtracting from the accumulator (SUB/SUBT). The contents of the P register may also be shifted prior to accumulation.

5.6.3 Moving Data

Many DSP applications must perform convolution operations or other operations similar in form. These operations require data to be shifted or delayed. The DMOV, LTD, and MACD instructions can perform the needed data moves for convolution.

The data move function allows a word to be copied from the currently addressed data memory location in on-chip RAM to the next higher location while the data from the addressed location is being operated upon (e.g., by the CALU). The data move and the CALU operation are performed in the same cycle. In addition, an ARAU operation may also be performed in the same cycle when using the indirect addressing mode. The data move function is useful in implementing algorithms, such as convolutions and digital filtering, where data is being passed through a time window. It models the z^{-1} delay operation encountered in those applications. The data move function is continuous across the boundary of the on-chip data memory blocks B0, B1, and B2. However, the data move function cannot be used if off-chip memory is referenced.

In Example 5-24, the following equation is implemented:

$$Y(n) = \sum_{k=0}^{2} H(k)\, X(n-k)$$

where the H values stay the same, and the X values are shifted each time the microprocessor performs one of the following series of multiplications (similar to operations performed in FIR filters):

First Series: $Y(2) = (H0)(X2)+(H1)(X1)+(H2)(X0)$
Second Series: $Y(3) = (H0)(X3)+(H1)(X2)+(H2)(X1)$
Third Series: $Y(4) = (H0)(X4)+(H1)(X3)+(H2)(X2)$

The MACD instruction, which combines accumulate and multiply operations with a data move, is tailored to the type of calculation shown in the summation equation above. In order to use MACD, the H values have been stored in block B0, configured as program RAM, and the X values have been read into block B1 of data RAM as shown in Figure 5-2.

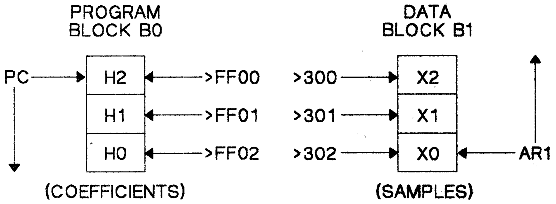


Figure 5-2. MACD Operation

Also in Example 5-24, the summation in the above equation is performed in the reverse order, i.e., from K = 2 to 0, due to the operation of the data move function. This results in the oldest X value being used and discarded first.

If the MACD instruction is replaced with the following two instructions, then the MAC instruction can be utilized with the same results.

```
MAC    *
DMOV   *-
```

In cases where many more than three MACD instructions are required, the RPT or RPTK instructions may be used with MACD, yielding the same computational results but using less assembly code.

Example 5-24. Using MACD for Moving Data

```
* THIS ROUTINE IMPLEMENTS A SINGLE PASS OF A THIRD-ORDER FIR
* FILTER. IT IS ASSUMED THAT THE H AND X VALUES HAVE ALREADY
* BEEN LOADED INTO THEIR RESPECTIVE MEMORY LOCATIONS, THAT
* THE ACCUMULATOR AND P REGISTER ARE BOTH RESET TO ZERO, AND
* THAT AR1 IS POINTING AT X0. NOTE THAT THE MACD INSTRUCTION
* MAY BE USED IN THE REPEAT MODE, BUT IT IS NOT IMPLEMENTED
* HERE.
*
FIR    CNFP             ; CONFIGURE BLOCK B0 AS PROGRAM MEMORY.
       LARP 1           ; AR1 SHOULD POINT AT THE X VALUES.
       MAC  >FF00,*-    ; P = (X0)(H2)
       MACD >FF01,*-    ; ACC = (X0)(H2)
       MACD >FF02,*     ; ACC = (X0)(H2) + (X1)(H1)
       APAC             ; ACC = (X0)(H2) + (X1)(H1) + (X2)(H0)
       CNFD             ; CONFIGURE BLOCK B0 AS DATA MEMORY.
       RET              ; RETURN TO MAIN PROGRAM.
```

5.6.4 Multiplication

The TMS320C2x hardware multiplier normally performs two's-complement 16-bit by 16-bit multiplies and produces a 32-bit result in one processor cycle. A single TMS320C25 instruction, MPYU, can be used to multiply two 16-bit unsigned numbers. To multiply two operands, one operand must be loaded into the T register (TR). The second operand is moved by the multiply instruction to the multiplier, which then produces the product in the P register (PR). Before another multiply can be performed, the contents of the PR must be moved to the accumulator. A single-multiply program is shown in Example 5-25. By pipelining multiplies and PR moves, most multiply operations can be performed with a single instruction.

A common operation in DSP algorithms is the summation of products. The MAC instruction, normally performed in multiple cycles, adds the contents of the PR to the accumulator and then simultaneously reads two values and multiplies them. When using the MAC instruction, a data memory value is multiplied by a program memory value. One of the operands can come from block B1 or B2 in on-chip data memory while the other operand may come from block B0. Block B0 must be configured as program memory when it supplies the second operand. Pipelining of the MAC instruction with a repeat instruction results in an execution time for each succeeding multiply-and-accumulate operation of only one cycle.

Example 5-25. Multiply

```
* THIS ROUTINE MULTIPLIES TWO VALUES IN DATA MEMORY LOCATIONS
* >200 AND >201 WITH THE RESULT STORED IN >202 AND >203.
*
MUL     LRLK AR1,>200    ; POINT AT BLOCK B0.
        LARP 1
        LT   *+          ; GET FIRST VALUE AT >200.
        MPY  *+          ; MULTIPLY BY VALUE AT >201.
        PAC              ; PUT RESULT IN ACCUMULATOR.
        SACL *+          ; STORE LOW WORD AT >202.
        SACH *           ; STORE HIGH WORD AT >203.
        RET              ; RETURN TO MAIN PROGRAM.
```

The pipelining of the MAC and MACD instructions incurs a certain amount of overhead in execution. In those cases where speed is more critical than program memory, it may be beneficial to use LTA or LTD and MPY instructions rather than MAC or MACD. Example 5-26 and Example 5-27 show an implementation of multiply-accumulates using the MAC instruction. Example 5-28 shows an implementation of multiply-accumulates using the LTA-MPY instruction pair. Figure 5-3, Figure 5-4, and Figure 5-5 provide graphically the information necessary to determine the efficiency of use for each of the techniques.

Example 5-26. Multiply-Accumulate Using the MAC Instruction (TMS32020)

```
*                        CLOCK  TOTAL CLOCK PROGRAM  TOTAL PROGRAM
*                       CYCLES     CYCLES    MEMORY       MEMORY
*
     LARP AR1              1                   1
     LRLK AR1,>300         2                   2
     CNFP                  1                   1
     ZAC                   1                   1
     MPYK 0                1                   1
     RPTK N-1              1                   1
     MAC  >FF00,*+     2 + N                   2
     APAC                  1      10 + N       1            10
```

Example 5-27. Multiply-Accumulate Using the MAC Instruction (TMS320C25)

```
*                        CLOCK  TOTAL CLOCK PROGRAM  TOTAL PROGRAM
*                       CYCLES     CYCLES    MEMORY       MEMORY
*
     LARP AR1              1                   1
     LRLK AR1,>300         2                   2
     CNFP                  1                   1
     ZAC                   1                   1
     MPYK 0                1                   1
     RPTK N-1              1                   1
     MAC  >FF00,*+     3 + N                   2
     APAC                  1      11 + N       1            10
```

Example 5-28. Multiply-Accumulate Using the LTA-MPY Instruction Pair

```
*                 CLOCK  TOTAL CLOCK PROGRAM  TOTAL PROGRAM
*                 CYCLES    CYCLES    MEMORY      MEMORY
*
    ZAC             1                   1
    LT    D1        1\                  1\
    MPY   C1        1 |                 1 |
    LTA   D2        1 |                 1 |
    MPY   C2        1 |                 1 |
     .              |                   |
     .               > 2N                > 2N
     .              |                   |
    LTA   DN        1 |                 1 |
    MPY   CN        1/                  1/
    APAC            1     2 + 2N        1         2 + 2N
```

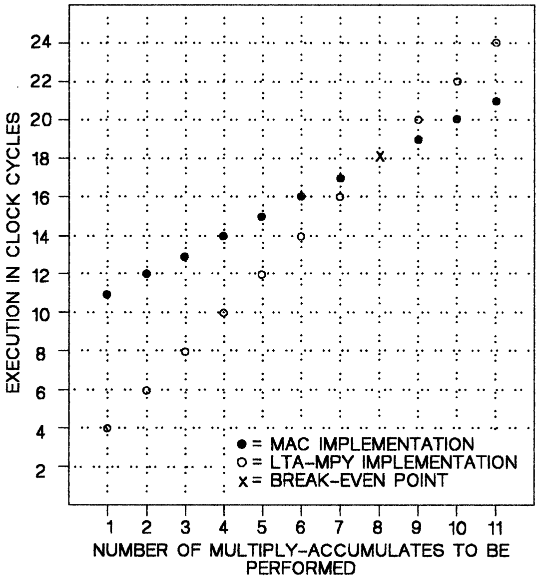


Figure 5-3. Execution Time vs. Number of Multiply-Accumulates (TMS32020)

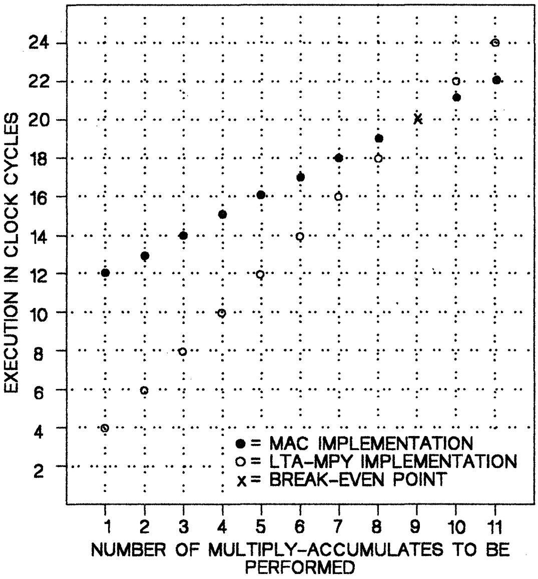


Figure 5-4. Execution Time vs. Number of Multiply-Accumulates (TMS320C25)

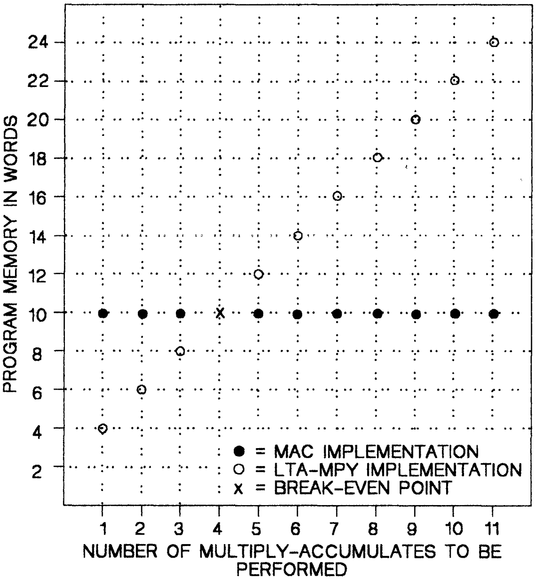


Figure 5-5. Program Memory vs. Number of Multiply-Accumulates

In numerical analysis, it is often necessary to square numbers along with adding or subtracting. The TMS320C2x has two instructions, SQRA and SQRS, that accomplish this in a single machine cycle. The result of the previous operation in the PR is first added to the accumulator if SQRA is used, or subtracted from the accumulator if SQRS is used. Then the data value addressed is squared, and the result is stored in the PR. Example 5-29 uses the SQRA instruction to perform the computation.

Example 5-29. Using SQRA

```
* THIS ROUTINE USES THE SQRA INSTRUCTION TO COMPUTE THE
* SQUARE OF THE DISTANCE BETWEEN TWO POINTS WHERE D**2
* IS DEFINED AS FOLLOWS:
*
*   D**2 = (XA - XB)**2 + (YA - YB)**2
*
DIST   LAC  XA
       SUB  XB
       SACL XT       ; XT = XA - XB
*
       LAC  YA
       SUB  YB
       SACL YT       ; YT = YA - YB
*
       SQRA XT       ; (P) = XT**2
       ZAC           ; (ACC) = 0
       SQRA YT       ; (P) = YT**2, (ACC) = XT**2
       APAC          ; (ACC) = XT**2 + YT**2 = D**2
*
       RET           ; RETURN TO MAIN PROGRAM.
```

When performing multiply-and-accumulate operations, it may be desirable to shift the product before adding it to the accumulator. This can be accomplished simultaneously with the MAC instruction by using the product shift mode on the TMS320C25. This mode, controlled by two bits in the PM field of status register ST1, shifts the value from the PR while it is transferred to the accumulator. The contents of the PR are not shifted.

5.6.5 Division

Division is implemented on the TMS320C2x by repeated subtractions using SUBC, a special conditional subtract instruction. Given a 16-bit positive dividend and divisor, the repetition of the SUBC command 16 times produces a 16-bit quotient in the low accumulator and a 16-bit remainder in the high accumulator.

SUBC implements binary division in the same manner as is commonly done in long division. The dividend is shifted until subtracting the divisor no longer produces a negative result. For each subtract that does not produce a negative answer, a one is put in the LSB of the quotient and then shifted. The shifting of the remainder and quotient after each subtract produces the separation of the quotient and remainder in the low and high halves of the accumulator.

There are similarities between long division and the SUBC method of division. Both methods are used to divide 33 by 5.

LONG DIVISION:

```
                  0000000000000110     Quotient
0000000000000101 )0000000000100001
                              -101
                               110
                              -101
                                11     Remainder
```

SUBC METHOD:

```
32  HIGH ACC         LOW ACC      0              COMMENT
|----------------| |----------------|
0000000000000000   0000000000100001  (1)  Dividend is loaded into ACC. The
             -10   1000000000000000       divisor is left-shifted 15 and sub-
             -10   0111111111011111       tracted from ACC. The subtraction
                                          is negative, so discard the result
|                | |                |     and shift left the ACC one bit.
0000000000000000   0000000001000010  (2)  2nd subtract produces negative
             -10   1000000000000000       answer, so discard result and shift
             -10   0111111110111110       ACC (dividend) left.

                  •                            •
                  •                            •
                  •                            •
|                | |                |
0000000000000100   0010000000000000  (14) 14th SUBC command. The result
             -10   1000000000000000       is positive. Shift result left and
0000000000000001   1010000000000000       replace LSB with '1'.

|                | |                |
0000000000000011   0100000000000001  (15) Result is again positive. Shift
             -10   1000000000000000       result left and replace LSB with '1'.
0000000000000000   1100000000000001

|                | |                |
0000000000000001   1000000000000011  (16) Last subtract. Negative answer, so
             -10   1000000000000000       discard result and shift ACC left.
                 - 1111111111111101

0000000000000011   0000000000000110       Answer reached after 16 SUBC
                                          instructions.
|   REMAINDER    | |    QUOTIENT    |
```

The condition of the divisor, less than the shifted dividend, is determined by the sign of the result, both the dividend and divisor must be positive when using the SUBC command. Thus, the sign of the quotient must be determined and the quotient computed using the absolute value of the dividend and divisor.

Integer and fractional division can be implemented with the SUBC instruction as shown in Example 5-30 and Example 5-31, respectively. When implementing a divide algorithm, it is important to know if the quotient can be represented as a fraction and the degree of accuracy to which the quotient is to be computed. For integer division, the absolute value of the numerator must be greater than the absolute value of the denominator. For fractional division, the absolute value of the numerator must be less than the absolute value of the denominator.

Example 5-30. Using SUBC for Integer Division

```
* THIS ROUTINE IMPLEMENTS INTEGER DIVISION.
*
DN1    LT   NUMERA   ; GET SIGN OF QUOTIENT.
       MPY  DENOM
       PAC
       SACH TEMSGN   ; SAVE SIGN OF QUOTIENT.
       LAC  DENOM
       ABS
       SACL DENOM    ; MAKE DENOMINATOR POSITIVE.
       LAC  NUMERA   ; ALIGN NUMERATOR.
       ABS
*
* IF DIVISOR AND DIVIDEND ARE ALIGNED, DIVISION CAN START
* HERE.
*
       RPTK 15
       SUBC DENOM    ; 16-CYCLE DIVIDE LOOP.
       SACL QUOT
       LAC  TEMSGN
       BGEZ DONE     ; DONE IF SIGN IS POSITIVE.
       ZAC
       SUB  QUOT
       SACL QUOT     ; NEGATE QUOTIENT IF NEGATIVE.
DONE   LAC  QUOT
       RET           ; RETURN TO MAIN PROGRAM.
```

Example 5-31. Using SUBC for Fractional Division

```
* THIS ROUTINE IMPLEMENTS FRACTIONAL DIVISION.
*
DN1    LT   NUMERA   ; GET SIGN OF QUOTIENT.
       MPY  DENOM
       PAC
       SACH TEMSGN   ; SAVE SIGN OF QUOTIENT.
       LAC  DENOM
       ABS
       SACL DENOM    ; MAKE DENOMINATOR POSITIVE.
       ZALH NUMERA   ; ALIGN NUMERATOR.
       ABS
*
* IF DIVISOR AND DIVIDEND ARE ALIGNED, DIVISION CAN START
* HERE.
*
       RPTK 14
       SUBC DENOM    ; 15-CYCLE DIVIDE LOOP.
       SACL QUOT
       LAC  TEMSGN
       BGEZ DONE     ; DONE IF SIGN IS POSITIVE.
       ZAC
       SUB  QUOT
       SACL QUOT     ; NEGATE QUOTIENT IF NEGATIVE.
DONE   LAC  QUOT
       RET           ; RETURN TO MAIN PROGRAM.
```

5.6.6 Floating-Point Arithmetic

Floating-point numbers are often represented on microprocessors in a two-word format of mantissa and exponent. The mantissa is stored in one word. The exponent, the second word, indicates how many bit positions from the left the decimal point is located. If the mantissa is 16 bits, a 4-bit exponent is sufficient to express the location of the decimal point. Because of its 16-bit word size, the 16/4-bit floating-point format functions most efficiently on the TMS320C2x. The theory and implementation of floating-point arithmetic has been presented in an application report in the book, *Digital Signal Processing Applications with the TMS320 Family*.

Operations in the TMS320C2x central ALU are performed in two's-complement fixed-point notation. To implement floating-point arithmetic, operands must be converted to fixed point for arithmetic operations, and then converted back to floating point.

Conversion to floating-point notation is performed by normalizing the input data (i.e., shifting the MSB of the data word into the MSB of the internal memory word). The exponent word then indicates how many shifts are required. To multiply two floating-point numbers, the mantissas are multiplied and the exponents added. The resulting mantissa must be renormalized. (Since the input operands are normalized, no more than one left shift is required to normalize the result.)

Floating-point addition or subtraction requires shifting the mantissa so that the exponents of the two operands match. The difference between the exponents is used to left-shift the lower power operand before adding. Then, the output of the add must be renormalized.

TMS320C2x instructions useful in floating-point operations are the NORM, LACT, ADDT, and SUBT instructions. NORM may be used to convert fixed-point numbers to floating-point. LACT may be used to convert back to fixed-point numbers. Addition and subtraction can be computed in floating point using ADDT and SUBT.

Example 5-32 and Example 5-33 perform a floating-point multiply on the TMS32020 and TMS320C25, respectively. The mantissas are assumed to be in Q15 format. Q15, one of the various types of Q format, is a number representation commonly used when performing operations on non-integer numbers. In Q format, the Q number (15 in Q15) denotes how many digits are located to the right of the binary point. A 16-bit number in Q15 format, therefore, has an assumed binary point immediately to the right of the most significant bit. Since the most significant bit constitutes the sign of the number, then numbers represented in Q15 may take on values from +1 (represented by +0.99997...) to -1.

Example 5-32. Using NORM for Floating-Point Multiply (TMS32020)

```
* THIS SUBROUTINE PERFORMS A FLOATING-POINT MULTIPLY USING
* THE NORM INSTRUCTION. THE INPUTS AND OUTPUTS ARE OF THE
* FORM:
*
*       C = MC * 2**EC
*
* SINCE THE MANTISSAS, MA AND MB, ARE NORMALIZED, MC CAN BE
* NORMALIZED WITH A LEFT SHIFT OF EITHER 0 OR 1 IN THE
* ACCUMULATOR. THE EXPONENT OF THE RESULT IS ADJUSTED
* APPROPRIATELY. FOR EXAMPLE, MULTIPLICATION OF THE TWO
* NUMBERS A AND B, WHERE A = 0.1 * 2**2 AND B = 0.1 * 2**4,
* PROCEEDS AS FOLLOWS:
*
*      1) A * B = 0.01 * 2**6
*      2) A * B = 0.1 * 2**5     (NORMALIZED RESULT)
*
MULT   LAC  EA
       ADD  EB         ; EC = EXPONENT OF RESULT BEFORE
       SACL EC         ; NORMALIZATION.
       LT   MA
       MPY  MB
       PAC             ; (ACC) = MA * MB
*
       SFL             ; TAKES CARE OF REDUNDANT SIGN BIT.
       LARP AR0
       LAR  AR0,0      ; AR0 IS INITIALIZED TO 0.
*
       NORM            ; FINDS MSB AND MODIFIES AR0.
*
       SACH MC         ; MC = MA * MB (NORMALIZED)
       SAR  AR0,TMP
       LAC  EC
       SUB  TMP
       SACL EC
       RET             ; RETURN TO MAIN PROGRAM.
```

Example 5-33. Using NORM for Floating-Point Multiply (TMS320C25)

```
* THIS SUBROUTINE PERFORMS A FLOATING-POINT MULTIPLY USING
* THE NORM INSTRUCTION. THE INPUTS AND OUTPUTS ARE OF THE
* FORM:
*
*          C = MC * 2**EC
*
* SINCE THE MANTISSAS, MA AND MB, ARE NORMALIZED, MC CAN BE
* NORMALIZED WITH A LEFT SHIFT OF EITHER 0 OR 1 IN THE
* ACCUMULATOR. THE EXPONENT OF THE RESULT IS ADJUSTED
* APPROPRIATELY. FOR EXAMPLE, MULTIPLICATION OF THE TWO
* NUMBERS A AND B, WHERE A = 0.1 * 2**2 AND B = 0.1 * 2**4,
* PROCEEDS AS FOLLOWS:
*
*        1) A * B = 0.01 * 2**6
*        2) A * B = 0.1 * 2**5    (NORMALIZED RESULT)
*
MULT   LAC  EA
       ADD  EB         ; EC = EXPONENT OF RESULT BEFORE
       SACL EC         ; NORMALIZATION.
       LT   MA
       MPY  MB
       PAC             ; (ACC) = MA * MB
*
       SFL             ; TAKES CARE OF REDUNDANT SIGN BIT.
       LARP AR5
       LAR  AR5,EC     ; AR5 IS.INITIALIZED WITH EC.
*
       NORM *-         ; FINDS MSB AND MODIFIES AR5.
*
       SACH MC         ; MC = MA * MB (NORMALIZED)
       SAR  AR5,EC
       RET             ; RETURN TO MAIN PROGRAM.
```

Floating-point implementation programs often require denormalization as well as normalization to return results in a 16-bit format. Example 5-34 and Example 5-35 are tailored for denormalizing numbers that were normalized using the NORM instruction. This program assumes that the mantissa is in the accumulator and the exponent is in an auxiliary register, which is the format of the NORM instruction after execution.

Example 5-34. Using LACT for Denormalization (TMS32020)

```
* THIS ROUTINE DENORMALIZES NUMBERS NORMALIZED BY THE NORM
* INSTRUCTION. THE DENORMALIZED NUMBER WILL BE IN THE
* ACCUMULATOR.
*
DENORM LARP 1          ; USE AR1 TO POINT AT BLOCK B0.
       LRLK AR1,>200
       SAR  AR0,*+     ; STORE EXPONENT AT >200.
       SACH *-         ; STORE MANTISSA AT >201.
*
* SUBTRACT EXPONENT FROM 16 TO DETERMINE THE NUMBER OF SHIFTS
* REQUIRED TO DENORMALIZE.
*
       LAC  *          ; LOAD ACCUMULATOR WITH EXPONENT.
       BZ   OUT        ; CHECK FOR ZERO EXPONENT.
       LACK >10
       SUB  *
       SACL *
       LT   *+
       LACT *          ; DENORMALIZE NUMBER.
       RET             ; RETURN TO MAIN PROGRAM.
OUT    MAR  *+         ; POINT TO MANTISSA.
       ZALH *          ; LOAD ACCUMULATOR WITH RESULT.
       RET             ; RETURN TO MAIN PROGRAM.
```

Example 5-35. Using LACT for Denormalization (TMS320C25)

```
* THIS ROUTINE DENORMALIZES NUMBERS NORMALIZED BY THE NORM
* INSTRUCTION (NORM *-). THE DENORMALIZED NUMBER WILL BE IN
* THE ACCUMULATOR.
*
DENORM LARP 1          ; USE AR1 TO POINT AT BLOCK B0.
       LRLK AR1,>200
       SAR  AR4,*+     ; STORE EXPONENT AT >200.
       SACH *-         ; STORE MANTISSA AT >201.
*
       LAC  *          ; LOAD ACCUMULATOR WITH EXPONENT.
       BZ   OUT        ; CHECK FOR ZERO EXPONENT.
       LT   *+
       LACT *          ; DENORMALIZE NUMBER.
       RET             ; RETURN TO MAIN PROGRAM.
OUT    MAR  *+         ; POINT TO MANTISSA.
       ZALH *          ; LOAD ACCUMULATOR WITH RESULT.
       RET             ; RETURN TO MAIN PROGRAM.
```

5.6.7 Indexed Addressing

The Auxiliary Register Arithmetic Unit (ARAU) allows the next indirect address to be calculated using increment/decrement calculations or indexed addressing in parallel to the current arithmetic operation. For example, in the multiplication of two matrices, the operation requires addressing across the rows (incrementing the address by one) or down the columns (incrementing by n). Example 5-36 gives the code for multiplying a row times a column of two 10 x 10 matrices. The first matrix resides in data RAM block B1, and the second matrix resides in block B0.

Example 5-36. Row Times Column

```
      LARK 0,>A       ; SET INDEX TO 10.
      LARP 1          ; USE AR1 FOR ADDRESSING THE COLUMN.
      LRLK 1,>300     ; POINT AR1 TO THE START OF BLOCK B1.
      CNFP            ; SET B0 TO PROG ADDRESS FOR PIPELINE.
      ZAC             ; INITIALIZE THE ACCUMULATOR.
      MPYK 0          ; CLEAR THE PRODUCT REGISTER.
      RPTK 9          ; REPEAT 10 TIMES AS MATRIX DIMENSION.
      MAC  >FF00,*0+  ; MULTIPLY ROW TIMES COLUMN.
      APAC            ; EXECUTE FINAL ACCUMULATION.
*                     ; ACCUMULATOR CONTAINS PRODUCT.
```

The algorithm in Example 5-36 executes in 22 machine cycles. The key to this performance is the parallel addressing of both multiplicands simultaneously. The operation is made possible by the use of the data bus to fetch one multiplicand and the program bus to fetch the other. The auxiliary register indexes down the column of one matrix while the PC generates incremental addressing of each row of the other matrix. Each cycle of the repeat loop performs the following operations:

1) Accumulates the previous product,
2) Multiplies the row element times the column element,
3) Increments the row address, and
4) Indexes the column address.

5.6.8 Extended-Precision Arithmetic

Numerical analysis, floating-point computations, or other operations may require arithmetic to be executed with more than 32 bits of precision. Since the TMS320C2x are 16/32-bit fixed-point processors, software is required for the extended-precision of arithmetic operations. Subroutines that perform the extended-arithmetic functions for both the TMS32020 and TMS320C25 are provided in the examples of this section. The technique consists of performing the arithmetic by parts, similar to the way in which longhand arithmetic is done.

The TMS320C25 has two features that help to make extended-precision calculations more efficient. One of the features is the carry status bit. This bit is affected by all arithmetic operations of the accumulator (ABS, ADD, ADDC, ADDH, ADDK, ADDS, ADDT, ADLK, APAC, LTA, LTD, LTS, MAC, MACD, MPYA, MPYS, NEG, SBLK, SPAC, SQRA, SQRS, SUB, SUBB, SUBC, SUBH, SUBK, SUBS, and SUBT). The carry bit is also affected by the rotate and shift accumulator instructions (ROL, ROR, SFL, and SFR) or may be explicitly modified by the load status register ST1 (LST1), reset carry (RC), and set carry (SC) instructions. For proper operation, the overflow mode bit should be reset (OVM = 0) so that the accumulator results will not be loaded with the saturation value. Note that this means that some additional code may be required if overflow of the most significant portion of the result is expected.

The carry bit is set whenever the addition of a value from the input scaling shifter or the P register to the accumulator contents generates a carry out of bit 31. Otherwise, the carry bit is reset since the carry out of bit 31 is a zero. One exception to this case is the ADDH instruction which can only set the carry bit. This allows the accumulation to generate the proper single carry when either the addition to the lower or upper half of the accumulator actually causes the carry. The following examples help to demonstrate the significance of the carry bit on the TMS320C25 for additions:

```
C  MSB        LSB                C  MSB        LSB
X  F F F F  F F F F  ACC         X  F F F F  F F F F  ACC
 +                1               + F F F F  F F F F
1  0 0 0 0  0 0 0 0              1  F F F F  F F F E

X  7 F F F  F F F F  ACC         X  7 F F F  F F F F  ACC
 +                1               + F F F F  F F F F
0  8 0 0 0  0 0 0 0              1  7 F F F  F F F E

X  8 0 0 0  0 0 0 0  ACC         X  8 0 0 0  0 0 0 0  ACC
 +                1               + F F F F  F F F F
0  8 0 0 0  0 0 0 1              1  7 F F F  F F F F

1  0 0 0 0  0 0 0 0  ACC         1  F F F F  F F F F  ACC
 +                0  (ADDC)       +                0  (ADDC)
0  0 0 0 0  0 0 0 1              1  0 0 0 0  0 0 0 0

1  8 0 0 0  F F F F  ACC         1  8 0 0 0  F F F F  ACC
 + 0 0 0 0  0 0 0 0  (ADDH)       + 7 F F F  0 0 0 0  (ADDH)
1  8 0 0 0  F F F F              1  F F F F  F F F F
```

Example 5-37 shows an implementation of two 64-bit numbers added to each other to obtain a 64-bit result. This example, which adds 16-bit parts and generates a carry (C) bit in the accumulator, will run on the TMS32020.

Example 5-37. 64-Bit Addition (TMS32020)

```
* TWO 64-BIT NUMBERS ARE ADDED TO EACH OTHER PRODUCING A
* 64-BIT RESULT. THE NUMBERS X (X3,X2,X1,X0) AND Y
* (Y3,Y2,Y1,Y0) ARE ADDED RESULTING IN W (W3,W2,W1,W0).
*
*            X3 X2 X1 X0
*          + Y3 Y2 Y1 Y0
*            -----------
*            W3 W2 W1 W0
*
ADD64   ZALS X0      ; ACC = 00 X0
        ADDS Y0      ; ACC = 00 X0 + 00 Y0 = C W0
        SACL W0
        SACH CARRY
        LAC  CARRY   ; ACC = 00 C
        ADDS X1      ; ACC = 00 C + 00 X1
        ADDS Y1      ; ACC = 00 C + 00 X1 + 00 Y1 = C W1
        SACL W1
        SACH CARRY
        LAC  CARRY   ; ACC = 00 C
        ADDS X2      ; ACC = 00 C + 00 X2
        ADDS Y2      ; ACC = 00 C + 00 Y2 + 00 Y2 = C W2
        SACH CARRY
        LAC  CARRY   ; ACC = 00 C
        ADDS X3      ; ACC = 00 C + 00 X3
        ADDS Y3      ; ACC = 00 C + 00 X3 + 00 Y3 = C W3
        SACL W3
        RET
```

Example 5-38 performs the same addition as Example 5-37 but is specific to the TMS320C25. This implementation makes use of the carry (C) status bit, adding 32-bit parts.

Example 5-38. 64-Bit Addition (TMS320C25)

```
* TWO 64-BIT NUMBERS ARE ADDED TO EACH OTHER PRODUCING A
* 64-BIT RESULT. THE NUMBERS X (X3,X2,X1,X0) AND Y
* (Y3,Y2,Y1,Y0) ARE ADDED RESULTING IN W (W3,W2,W1,W0).
*
*            X3 X2 X1 X0
*          + Y3 Y2 Y1 Y0
*            -----------
*            W3 W2 W1 W0
*
ADD64   ZALH X1      ; ACC = X1 00
        ADDS X0      ; ACC = X1 X0
        ADDS Y0      ; ACC = X1 X0 + 00 Y0
        ADDH Y1      ; ACC = X1 X0 + Y1 Y0 = W1 W0
        SACL W0
        SACH W1
        ZALH X3      ; ACC = X3 00
        ADDC X2      ; ACC = X3 X2 + C
        ADDS Y2      ; ACC = X3 X2 + 00 Y2 + C
        ADDH Y3      ; ACC = X3 X2 + Y3 Y2 + C = W3 W2
        SACL W2
        SACH W3
        RET
```

In a similar way to addition, the carry bit on the TMS320C25 is reset whenever the input scaling shifter or the P-register value subtracted from the accumulator contents generates a borrow into bit 31. Otherwise, the carry bit is set since no borrow into bit 31 is required. One exception to this case is the SUBH instruction which can only reset the carry bit. This allows the generation of the proper single carry when either the subtraction from the lower or upper half of the accumulator actually causes the borrow. The following examples help to demonstrate the significance of the carry bit for subtractions:

```
C  MSB            LSB               C  MSB            LSB
X  0 0 0 0  0 0 0 0  ACC            X  0 0 0 0  0 0 0 0  ACC
 -                1                  - F F F F  F F F F
0  F F F F  F F F F                 0  0 0 0 0  0 0 0 1

X  7 F F F  F F F F  ACC            X  7 F F F  F F F F  ACC
 -                1                  - F F F F  F F F F
1  7 F F F  F F F E                 0  8 0 0 0  0 0 0 0

X  8 0 0 0  0 0 0 0  ACC            X  8 0 0 0  0 0 0 0  ACC
 -                1                  - F F F F  F F F F
1  7 F F F  F F F F                 0  8 0 0 0  0 0 0 1

0  0 0 0 0  0 0 0 0  ACC            0  F F F F  F F F F  ACC
 -                0  (SUBB)          -                0  (SUBB)
0  F F F F  F F F F                 1  F F F F  F F F E

0  8 0 0 0  F F F F  ACC            0  8 0 0 0  F F F F  ACC
 - 0 0 0 1  0 0 0 0  (SUBH)          - F F F F  0 0 0 0  (SUBH)
0  7 F F F  F F F F                 0  8 0 0 1  F F F F
```

Example 5-39 implements the subtraction of two 64-bit numbers on the TMS32020. A borrow (B) is generated within the accumulator for each of the 16-bit parts of the subtraction operation.

Example 5-39. 64-Bit Subtraction (TMS32020)

```
* TWO 64-BIT NUMBERS ARE SUBTRACTED, PRODUCING A 64-BIT
* RESULT. THE NUMBER Y (Y3,Y2,Y1,Y0) IS SUBTRACTED FROM
* X (X3,X2,X1,X0) RESULTING IN W (W3,W2,W1,W0).
*
*            X3 X2 X1 X0
*          - Y3 Y2 Y1 Y0
*            -----------
*            W3 W2 W1 W0
*
SUB64   ZALS X0     ; ACC = 00 X0
        SUBS Y0     ; ACC = 00 X0 - 00 Y0 = B W0
        SACL W0
        SACH BORROW
        LAC  BORROW ; ACC = B
        ADDS X1     ; ACC = B + 00 X1
        SUBS Y1     ; ACC = B + 00 X1 - 00 Y1 = B W1
        SACL W1
        SACH BORROW
        LAC  BORROW ; ACC = B
        ADDS X2     ; ACC = B + 00 X2
        SUBS Y2     ; ACC = B + 00 X2 - 00 Y2 = B W2
        SACL W2
        SACH BORROW
        LAC  BORROW ; ACC = B
        ADDS X3     ; ACC = B + 00 X3
        SUBS Y3     ; ACC = B + 00 X3 - 00 Y3 = B W3
        SACL W3
        RET
```

The advantage of using the carry (C) status bit on the TMS320C25 in implementing the same subtraction as Example 5-39 is shown in the coding of Example 5-40.

Example 5-40. 64-Bit Subtraction (TMS320C25)

```
* TWO 64-BIT NUMBERS ARE SUBTRACTED, PRODUCING A 64-BIT
* RESULT. THE NUMBER Y (Y3,Y2,Y1,Y0) IS SUBTRACTED FROM
* X (X3,X2,X1,X0) RESULTING IN W (W3,W2,W1,W0).
*
*            X3 X2 X1 X0
*          - Y3 Y2 Y1 Y0
*            -----------
*            W3 W2 W1 W0
*
SUB64   ZALH X1     ; ACC = X1 00
        ADDS X0     ; ACC = X1 X0
        SUBS Y0     ; ACC = X1 X0 - 00 Y0
        SUBH Y1     ; ACC = X1 X0 - Y1 Y0 = W1 W0
        SACL W0
        SACH W1
        ZALS X2     ; ACC = 00 X2
        SUBB Y2     ; ACC = 00 X2 - 00 Y2 - C
        ADDH X3     ; ACC = X3 X2 - 00 Y2 - C
        SUBH Y3     ; ACC = X3 X2 - Y3 Y2 - C = W3 W2
        SACL W2
        SACH W3
        RET
```

The second feature of the TMS320C25 aiding in extended-precision calculations is the MPYU (unsigned multiply) instruction. The MPYU instruction allows two unsigned 16-bit numbers to be multiplied and the 32-bit result placed in the product register in a single cycle. Efficiency is gained by the ability to generate partial products from the 16-bit portions of a 32-bit or larger value instead of having to split the value into 15-bit or smaller parts.

Example 5-41 and Example 5-42 show implementations of multiplying two 32-bit numbers to obtain a 64-bit result. The coding of Example 5-41 will perform the 32-bit multiply on a TMS32020. The advantage in using the MPYU instruction can be observed in Example 5-42, which will execute on the TMS320C25.

Example 5-41. 32 x 32-Bit Multiplication (TMS32020)

```
* TWO 32-BIT NUMBERS ARE MULTIPLIED, PRODUCING A 64-BIT RESULT.
* THE NUMBERS X (X1,X0) AND Y (Y1,Y0) ARE MULTIPLIED RESULTING
* IN W (W3,W2,W1,W0).
*
*                    X1 X0
*          x         Y1 Y0
*             -----------
*                    X0*Y0
*                 X1*Y0
*                 X0*Y1
*              X1*Y1
*             -----------
*              W3 W2 W1 W0
*
* THE PROCEDURE FOR MULTIPLICATION IS TO SEPARATE THE 32-BIT
* MAGNITUDE VALUES OF X AND Y INTO THREE PARTS OF 2, 15, AND 15
* BITS EACH. THE MULTIPLICATION BY PARTS THEN PRODUCES A 5-PART
* RESULT OF 3, 15, 15, 15, AND 15 BITS, WHICH ARE RECOMBINED
* INTO FOUR DATA WORDS OF 16 BITS EACH.
*
*                 X2 X1 X0
*       x         Y2 Y1 Y0
*       -----------------
*                    X0*Y0
*                 X1*Y0
*                 X0*Y1
*              X2*Y0
*              X1*Y1
*              X0*Y2
*           X2*Y1
*           X1*Y2
*       X2*Y2
*       -----------------
*           W4 W3 W2 W1 W0
*
* DETERMINE THE SIGN OF THE PRODUCT.
*
MPY32   ZALS X1       ; ACCL = SXXX XXXX XXXX XXXX
        XOR  Y1       ; ACCL = S--- ---- ---- ----
        SACH SIGN,1   ; SAVE THE PRODUCT SIGN 0=+, 1=-.
*
* TAKE THE ABSOLUTE VALUE OF BOTH X AND Y AND REPARTITION.
*
ABSX    ZALH X1       ; ACC = X1 00
        ADDS X0       ; ACC = X1 X0
        ABS
        SACH X1,1     ; SAVE |X2X1|.
        AND  M7FFF
        SACL X0       ; SAVE |X0|.
        ZALS X1
        SACH X2,1     ; SAVE |X2|.
        AND  M7FFF
        SACL X1       ; SAVE |X1|.
ABSY    ZALH Y1       ; ACC = Y1 00
        ADDS Y0       ; ACC = Y1 Y0
        ABS
        SACH Y1,1     ; SAVE |Y2Y1|.
```

```
        AND   M7FFF
        SACL  Y0        ; SAVE |Y0|.
        ZALS  Y1
        SACH  Y2,1      ; SAVE |Y2|.
        AND   M7FFF
        SACL  Y1        ; SAVE |Y1|.
*
* MULTIPLY |X| AND |Y| TO PRODUCE |W|.
*
MULT    LAC   X2        ; 'AND' FUNCTION IS A 1-BIT BY
        AND   Y2        ; 1-BIT MULTIPLICATION.
        SACL  W4        ; SAVE PARTIAL |W4|.
        LT    X0        ; T = X0
        MPY   Y0        ; T = X0, P = X0*Y0
        PAC   W0        ; T = X0, P = X0*Y0, ACC = X0*Y0
        SACH  W1,1      ; SAVE PARTIAL |W1|.
        AND   M7FFF
        SACL  W0        ; SAVE |W0|.
        ZALS  W1        ; T = X0, P = X0*Y0
*                       ; ACC = X0*Y0*2**-16
        MPY   Y1        ; T = X0, P = X0*Y1
*                       ; ACC = X0*Y0*2**-16
        LTA   X1        ; T = X1, P = X0*Y1
*                       ; ACC = X0*Y1 + X0*Y0*2**-16
        MPY   Y0        ; T = X1, P = X1*Y0
*                       ; ACC = X0*Y1 + X0*Y0*2**-16
        LTA   X0        ; T = X0, P = X1*Y0
*                       ; ACC = X1*Y0 + X0*Y1 + X0*Y0*2**-16
        SACH  W2,1      ; SAVE PARTIAL |W2|.
        AND   M7FFF
        SACL  W1        ; SAVE |W1|.
        ZALS  W2        ; T = X0, P = X1*Y0
*                       ; ACC = (X1*Y0+X0*Y1)*2**-16
        MPY   Y2        ; T = X0, P = X0*Y2
*                       ; ACC = (X1*Y0+X0*Y1)*2**-16
        LTA   X1        ; T = X1, P = X0*Y2
*                       ; ACC = X0*Y2 + (X1*Y0+X0*Y1)*2**-16
        MPY   Y1        ; T = X1, P = X1*Y1
*                       ; ACC = X0*Y2 + (X1*Y0+X0*Y1)*2**-16
        LTA   X2        ; T = X2, P = X1*Y1
*                       ; ACC = X1*Y1 + X0*Y2 + (X1*Y0+X0*Y1)*2**-16
        MPY   Y0        ; T = X2, P = X2*Y0
*                       ; ACC = X1*Y1 + X0*Y2 + (X1*Y0+X0*Y1)*2**-16
        LTA   X1        ; T = X1, P = X2*Y0
*                       ; ACC = X2*Y0 + X1*Y1 +X0*Y2
*                       ;     + (X1*Y0+X0*Y1)*2**-16
        SACH  W3,1      ; SAVE PARTIAL |W3|.
        AND   M7FFF
        SACL  W2        ; SAVE |W2|.
        ZALS  W3        ; T = X1, P = X2*Y0
*                       ; ACC = (X2*Y0+X1*Y1+X0*Y2)*2**-16
        MPY   Y2        ; T = X1, P = X1*Y2
*                       ; ACC = (X2*Y0+X1*Y1+X0*Y2)*2**-16
        LTA   X2        ; T = X2, P = X1*Y2
*                       ; ACC = X1*Y2 + (X2*Y0+X1*Y1+X0*Y2)*2**-16
        MPY   Y1        ; T = X2, P = X2*Y1
*                       ; ACC = X1*Y2 + (X2*Y0+X1*Y1+X0*Y2)*2**-16
        APAC            ; ACC = X2*Y1 + X1*Y2
*                       ;     + (X2*Y0+X1*Y1+X0*Y2)*2**-16
        ADD   W4,15     ; ACC = X2*Y2*2**15 + X2*Y1 + X1*Y2
*                       ;       + (X2*Y0+X1*Y1+X0*Y2)*2**-16
```

```
        SACH W4,1     ; SAVE |W4|.
        AND  M7FFF
        SACL W3       ; SAVE |W3|.
*
* RECOMBINE W AND GENERATE TWO'S-COMPLEMENT RESULT.
*
        ZAC
        SUB  SIGN
        SACL SIGN     ; SIGN 0=+, -1=-.
*
        LAC  W1,15    ; ACC = |W1 00|
        ADD  W0       ; ACC = |W1 W0|
        ADD  SIGN
        XOR  SIGN     ; COMPLEMENT W0 WHEN SIGN = -1.
        SACL W0       ; SAVE W0.
        SACH W1       ; SAVE PARTIAL |W1|.
        LAC  W2,14    ; ACC = |W2 00|
        ADD  W1       ; ACC = |W2 W1|
        XOR  SIGN     ; COMPLEMENT W1 WHEN SIGN = -1.
        SACL W1       ; SAVE W1.
        SACH W2       ; SAVE PARTIAL |W2|.
        LAC  W3,13    ; ACC = |W3 00|
        ADD  W2       ; ACC = |W3 W2|
        XOR  SIGN     ; COMPLEMENT W2 WHEN SIGN = -1.
        SACL W2       ; SAVE W2.
        SACH W3       ; SAVE PARTIAL |W3|.
        LAC  W4,12    ; ACC = |W4 00|
        ADD  W3       ; ACC = |W4 W3|
        XOR  SIGN     ; COMPLEMENT W3 WHEN SIGN = -1.
        SACL W3       ; SAVE W3.
        RET
```

Example 5-42. 32 x 32-Bit Multiplication (TMS320C25)

```
* TWO 32-BIT NUMBERS ARE MULTIPLIED, PRODUCING A 64-BIT
* RESULT. THE NUMBERS X (X1,X0) AND Y (Y1,Y0) ARE
* MULTIPLIED RESULTING IN W (W3,W2,W1,W0).
*
*                  X1 X0
*          x       Y1 Y0
*            -----------
*                  X0*Y0
*               X1*Y0
*               X0*Y1
*            X1*Y1
*            -----------
*            W3 W2 W1 W0
*
* DETERMINE THE SIGN OF THE PRODUCT.
*
MPY32  ZALS X1      ; ACCL = SXXX XXXX XXXX XXXX
       XOR  Y1      ; ACCL = S--- ---- ---- ----
       SACH SIGN,1 ; SAVE THE PRODUCT SIGN 0=+, 1=-.
*
* TAKE THE ABSOLUTE VALUE OF BOTH X AND Y.
*
ABSX   ZALH X1      ; ACC = X1 00
       ADDS X0      ; ACC = X1 X0
       ABS
       SACH X1      ; SAVE |X1|.
       SACL X0      ; SAVE |X0|.
ABSY   ZALH Y1      ; ACC = Y1 00
       ADDS Y0      ; ACC = Y1 X0
       ABS
       SACH Y1      ; SAVE |Y1|.
       SACL Y0      ; SAVE |Y0|.
*
* MULTIPLY |X| AND |Y| TO PRODUCE |W|.
*
MULT   LT   X0      ; T = X0
       MPYU Y0      ; T = X0, P = X0*Y0
       SPL  W1      ; SAVE |W0|.
       SPH  W0      ; SAVE PARTIAL |W1|.
       MPYU Y1      ; T = X0, P = X0*Y1
       LTP  X1      ; T = X1, P = X0*Y1, ACC = X0*Y1
       MPYU Y0      ; T = X1, P = X1*Y0, ACC = X0*Y1
       ADDS W1      ; T = X1, P = X1*Y0,
*                   ; ACC = X0*Y1 + X0*Y0*2**-16
       MPYA Y1      ; T = X1, P = X1*Y1,
*                   ; ACC = X1*Y0 + X0*Y1 + X0*Y0*2**-16
       SACL W1      ; SAVE |W1|.
       SACH W2      ; SAVE PARTIAL |W2|.
       ZALS W2      ; P = X1*Y1,
*                   ; ACC = (X1*Y0 + X0*Y1)*2**-16
       BNC  SUM     ; TEST FOR CARRY FROM W2.
       ADDH ONE
SUM    APAC         ; ACC = X1*Y1 + (X1*Y0 + X0*Y1)*2**-16
       SACL W2      ; SAVE |W2|.
       SACH W3      ; SAVE |W3|.
```

```
*
* TEST THE SIGN OF THE PRODUCT; NEGATE IF NEGATIVE.
*
        LAC   SIGN
        BZ    DONE     ; RETURN IF POSITIVE.
*
        ZALH  W1       ; ACC = |W1 00|
        ADDS  W0       ; ACC = |W1 W0|
        CMPL
        ADD   ONE      ; ACC = W1 W0 AND CARRY GENERATION
        SACL  W0       ; SAVE W0.
        SACH  W1       ; SAVE W1.
        ZALS  W2       ; ACC = |00 W2|
        ADDH  W3       ; ACC = |W3 W2|
        CMPL
        ADDC  ONE      ; ACC = W3 W2
        SACL  W2       ; SAVE W2.
        SACH  W3       ; SAVE W3.
DONE    RET
```

5.7 Application-Oriented Operations

The TMS320C2x has been designed to provide efficient implementations of many common digital signal processing algorithms. The architecture supporting these design features was discussed in Section 3. In general, the features provide efficient solutions to numerically intensive problems usually characterized by multiply/accumulates. Some device-specific features that aid in the implementation of specific algorithms include companding, filtering, Fast Fourier Transforms (FFT), and PID control. These applications require I/O performed either in parallel or serial. Hardware requirements for I/O are discussed in Sections 3 and 6.

5.7.1 Companding

In the area of telecommunications, one of the primary concerns is the I/O bandwidth in the communications channel. One way to minimize this bandwidth is by companding (COMpress/exPAND). Companding is defined by two international standards, A-law and μ-law, both based on the compression of the equivalent of 13 bits of dynamic range into an 8-bit code. The standard employed in the United States and Japan is μ-law companding. The European standard is referred to as A-law companding. Detailed descriptions and code examples of μ-law and A-law companding are presented in an application report on companding routines included in the book, *Digital Signal Processing Applications with the TMS320 Family.*

The technique of companding allows the digital sample information corresponding to a 13-bit dynamic range to be transmitted as 8-bit data. For processing in the TMS320C2x, it is necessary to convert the 8-bit (logarithmic) sign-magnitude data to a 16-bit two's-complement (linear) format. Prior to output, the linear result must be converted to the compressed or companded format. Table lookup or conversion subroutines may be used to implement these functions.

Software routines for μ-law and A-law companding, flowcharts, companding algorithms, and detailed descriptions are provided in the application report on companding routines mentioned above. The algorithm space and time requirements for μ-law and A-law companding on the TMS32020/C25 are given in Table 5-1.

Table 5-1. Program Space and Time Requirements for μ-/A-Law Companding

FUNCTION	MEMORY WORDS		PROGRAM CYCLES		TIME (μs) REQD†	
	Program	Data	Initialization	Loop‡	'20	'C25
μ-Law:						
Compression	74	8	19	45	9	45
Expansion	276	2	14	5	1	0.5
A-Law:						
Compression	100	8	19	50	10	5
Expansion	276	2	14	5	1	0.5

†Assuming initialization
‡Worst case

In expanding from the 8-bit data to the 13-bit linear representation, table lookup is very effective since the table length is only 256 words. This is especially true for a microcomputer design since the TMS320C25 has 4K words of mask-programmable ROM. The table lookup technique requires three instructions (four words of program memory), one data memory location, 256 words of table memory, and seven instruction cycles (program in on-chip ROM) to execute.

```
LAC   SAMPLE   ; LOAD 8-BIT DATA.
ADLK  MUTABL   ; ADD THE CONVERSION TABLE BASE ADDRESS.
TBLR  SAMPLE   ; READ THE CORRESPONDING LINEAR VALUE.
```

The above conversion could be programmed as a subroutine. This would eliminate the need for a table, but would increase execution time and require additional data memory locations.

When the output data has been determined in a system transmitting companded data, a compression of the data must be performed. The compression reduces the data back to the 8-bit format. Unless memory for a table of length 16384 is acceptable, the table lookup approach must be abandoned for conversion routines. Details of these implementations may be found in the application report on companding.

Access to new companding code as it becomes available is provided via the TMS320 DSP Bulletin Board Service. The bulletin board contains TMS320 source code from application reports included in *Digital Signal Processing Applications with the TMS320 Family*. See the *TMS320 Family Development Support Reference Guide* for information on how to access the bulletin board.

5.7.2 FIR/IIR Filtering

Digital filters are a common requirement for digital signal processing systems. The filters fall into two basic categories: Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) filters. For either category of filter, the coefficients of the filter (weighting factors) may be fixed or adapted during the course of the signal processing. The theory and implementation of digital filters has been presented and discussed in an application report (see the book, *Digital Signal Processing Applications with the TMS320 Family*). The TMS320C25 reduces the execution time of all filters by virtue of its 100-ns instruction cycle time.

IIR filters benefit from the 100-ns instruction cycle time of the TMS320C25. IIR filters typically require fewer multiply/accumulates. Correspondingly, the amount of data memory for samples and coefficients is not usually the limiting factor. Because of sensitivity to quantization of the coefficients themselves, IIR filters are usually implemented in cascaded second-order sections. This translates to instruction code consisting of LTD-MPY instruction pairs rather than MACDs. Example 5-43 provides an implementation of a second-order IIR filter.

Example 5-43. Implementing an IIR Filter

```
*
* THE FOLLOWING EQUATIONS ARE USED TO IMPLEMENT AN IIR FILTER:
*
*    d(n) = x(n)   + d(n-1)a1 + d(n-2)a2
*    y(n) = d(n)b0 + d(n-1)b1 + d(n-2)b2
*
START  IN    XN,PA0  ; INPUT NEW VALUE XN
       LAC   XN,15   ; LOAD ACCUMULATOR WITH XN
*
       LT    DNM1
       MPY   A1
*
       LTD   DNM2
       MPY   A2
*
       APAC
       SACH  DN,1    ; d(n) = x(n) + d(n-1)a1 + d(n-2)a2
       ZAC
       MPY   B2
*
       LTD   DNM1
       MPY   B1
*
       LTD   DN
       MPY   B0
*
       APAC
       SACH  YN,1    ; y(n) = d(n)b0 + d(n-1)b1 + d(n-2)b2
       OUT   YN,PA1  ; YN IS THE OUTPUT OF THE FILTER
```

FIR filters also benefit from the faster instruction cycle time. In addition, an FIR filter requires many more multiply/accumulates than does the IIR filter with equivalent sharpness at the cutoff frequencies and distortion and attenuation in the passbands and stopbands. The TMS320C2x can help solve this problem by making longer filters feasible to implement. This is accomplished by allowing the coefficients to be fetched from program memory at the same time as a sample is being fetched from data memory. The simple implementation of this process uses the MACD instruction with the RPT/RPTK instruction.

```
       RPTK 255
       MACD COEFFP,*-
```

The coefficients on the TMS32020 may be stored anywhere in on-chip RAM. Filters of up to 256 taps can be implemented at an execution speed of 200 ns per tap.

The coefficients on the TMS320C25 may be stored anywhere in program memory (reconfigurable on-chip RAM, on-chip ROM, or external memories). When the coefficients are stored in on-chip ROM or externally, the entire on-chip data RAM may be used to store the sample sequence. Ultimately, this allows filters of up to 512 taps to be implemented on the TMS320C25. The filter executes at full speed or 100 ns per tap as long as the memory supports full-speed execution.

5.7.3 Adaptive Filtering

With FIR/IIR filtering, the filter coefficients may be fixed or adapted. If the coefficients are adapted or updated with time, then another factor impacts the computational capacity. This factor is the requirement to adapt each of the coefficients, usually with each sample. The MPYA or MPYS and ZALR instructions on the TMS320C25 aid with this adaptation to reduce the execution time.

A means of adapting the coefficients on the TMS320C2x is the Least-Mean-Square (LMS) algorithm given by the following equation:

$$b_k(i+1) = b_k(i) + 2B\ e(i)\ x(i-k)$$

$$\text{where } e(i) = x(i) - y(i)$$

$$\text{and } y(i) = \sum_{k=0}^{N-1} b_k\ x(i-k)$$

Quantization errors in the updated coefficients can be minimized if the result is obtained by rounding rather than truncating. For each coefficient in the filter at a given point in time, the factor 2*B*e(i) is a constant. This factor can then be computed once and stored in the T register for each of the updates. Thus, the computational requirement has become one multiply/accumulate plus rounding. Without the new instructions, the adaptation of each coefficient is five instructions corresponding to five clock cycles. This is shown in the following instruction sequence:

```
LRLK AR2,COEFFD  ; LOAD ADDRESS OF COEFFICIENTS.
LRLK AR3,LASTAP  ; LOAD ADDRESS OF DATA SAMPLES.
LARP AR2
LT   ERRF        ; errf = 2*B*e(i)
 .
 .
 .
ZALH *,AR3       ; ACC = bk(i)*2**16
ADD  ONE,15      ; ACC = bk(i)*2**16 + 2**15
MPY  *-,AR2
APAC             ; ACC = bk(i)*2**16 + errf*x(i-k) + 2**15
SACH *+          ; SAVE bk(i+1).
 .
 .
 .
```

When the MPYA and ZALR instructions on the TMS320C25 are used, the adaptation reduces to three instructions corresponding to three clock cycles, as shown in the following instruction sequence. Note that the processing order has been slightly changed to incorporate the use of the MPYA instruction. This is due to the fact that the accumulation performed by the MPYA is the accumulation of the previous product.

```
     LRLK AR2,COEFFD  ; LOAD ADDRESS OF COEFFICIENTS.
     LRLK AR3,LASTAP  ; LOAD ADDRESS OF DATA SAMPLES.
     LARP AR2
     LT   ERRF        ; errf = 2*B*e(i)
       .
       .
       .
     ZALR *,AR3       ; ACC = bk(i)*2**16 + 2**15
     MPYA *-,AR2      ; ACC = bk(i)*2**16 + errf*x(i-k) + 2**15
*                     ; PREG = errf*x(i-k+1)
     SACH *+          ; SAVE bk(i+1).
       .
       .
       .
```

Example 5-44 shows a routine to filter a signal and update the coefficients. Example 5-45 and Example 5-46 provide the conclusion to the adaptive FIR filter routine for the TMS32020 and TMS320C25, respectively.

Adaptive filter length is restricted both by execution time and memory. There is obviously more processing to be completed per sample due to the adaptation, and the adaptation itself dictates that the coefficients be stored in the reconfigurable block of on-chip RAM. Thus, the practical limit of an adaptive filter with no external data memory is 256 taps.

Example 5-44. 256-Tap Adaptive FIR Filter

```
        TITL 'ADAPTIVE FILTER'
        DEF  ADPFIR
        DEF  X,Y
*
* THIS 256-TAP ADAPTIVE FIR FILTER USES ON-CHIP MEMORY BLOCK
* B0 FOR COEFFICIENTS AND BLOCK B1 FOR DATA SAMPLES. THE
* NEWEST INPUT SHOULD BE IN MEMORY LOCATION X WHEN CALLED.
* THE OUTPUT WILL BE IN MEMORY LOCATION Y WHEN RETURNED.
* ASSUME THAT THE DATA PAGE IS 0 WHEN THE ROUTINE IS CALLED.
*
COEFFP  EQU  >FF00          ; B0 PROGRAM MEMORY ADDRESS
COEFFD  EQU  >0200          ; B0 DATA MEMORY ADDRESS
*
ONE     EQU  >7A            ; CONSTANT ONE        (DP=0)
BETA    EQU  >7B            ; ADAPTATION CONSTANT (DP=0)
ERR     EQU  >7C            ; SIGNAL ERROR        (DP=0)
ERRF    EQU  >7D            ; ERROR FUNCTION      (DP=0)
Y       EQU  >7E            ; FILTER OUTPUT       (DP=0)
X       EQU  >7F            ; NEWEST DATA SAMPLE  (DP=0)
FRSTAP  EQU  >0300          ; NEXT NEWEST DATA SAMPLE
LASTAP  EQU  >03FF          ; OLDEST DATA SAMPLE
*
* FINITE IMPULSE  RESPONSE (FIR) FILTER.
*
ADPFIR  CNFP                ; CONFIGURE B0 AS PROGRAM:
        MPYK 0              ; Clear the P register.
        LAC  ONE,14         ; Load output rounding bit.
        LARP AR3
        LRLK AR3,LASTAP     ; Point to the oldest sample.
FIR     RPTK 255
        MACD COEFFP,*-      ; 256-tap FIR filter.
        CNFD                ; CONFIGURE B0 AS DATA:
        APAC
        SACH Y,1            ; Store the filter output.
        NEG
        ADD  X,15           ; Add the newest input.
        SACH ERR,1          ; err(i) = x(i) - y(i)
*
* LMS ADAPTATION OF FILTER COEFFICIENTS.
*
        LT   ERR
        MPY  BETA
        PAC                 ; errf(i) = beta * err(i)
        ADD  ONE,14         ; ROUND THE RESULT.
        SACH ERRF,1
*
        MAR  *+
        LAC  X              ; INCLUDE NEWEST SAMPLE.
        SACL *
*
        LRLK AR2,COEFFD     ; POINT TO THE COEFFICIENTS.
        LRLK AR3,LASTAP     ; POINT TO THE DATA SAMPLES.
        LT   ERRF
        MPY  *-,AR2         ; P = 2*beta*err(i)*x(i-255)
```

Example 5-45. Adaptive Filter Routine Concluded (TMS32020)

```
*
ADAPT   ZALH  *,AR3        ; LOAD ACCH WITH b255(i).
        ADD   ONE,15       ; ADD ROUNDING BIT.
        APAC               ; b255(i+1) = b255(i) + P
        MPY   *-,AR2       ; P = 2*beta*err(i)*x(i-254)
        SACH  *+           ; STORE b255(i+1).
*
        ZALH  *,AR3        ; LOAD ACCH WITH b254(i).
        ADD   ONE,15       ; ADD ROUNDING BIT.
        APAC               ; b254(i+1) = b254(i) + P
        MPY   *-,AR2       ; P = 2*beta*err(i)*x(i-253)
        SACH  *+           ; STORE b254(i+1).
*
        ZALH  *,AR3        ; LOAD ACCH WITH b253(i).
        ADD   ONE,15       ; ADD ROUNDING BIT.
        APAC               ; b253(i+1) = b253(i) + P
        MPY   *-,AR2       ; P = 2*beta*err(i)*x(i-252)
        SACH  *+           ; STORE b253(i+1).
          .
          .
          .
        ZALH  *,AR3        ; LOAD ACCH WITH b1(i).
        ADD   ONE,15       ; ADD ROUNDING BIT.
        APAC               ; b1(i+1) = b1(i) + P
        MPY   *-,AR2       ; P = 2*beta*err(i)*x(i-0)
        SACH  *+           ; STORE b1(i+1).
*
        ZALH  *            ; LOAD ACCH WITH b0(i).
        ADD   ONE,15       ; ADD ROUNDING BIT.
        APAC               ; b0(i+1) = b0(i) + P
        SACH  *+           ; STORE b0(i+1).
*
        RET                ; RETURN TO CALLING ROUTINE.
```

Example 5-46. Adaptive Filter Routine Concluded (TMS320C25)

```
*
ADAPT   ZALR *,AR3          ; LOAD ACCH WITH b255(i) & ROUND.
        MPYA *-,AR2         ; b255(i+1) = b255(i) + P
*                           ; P = 2*beta*err(i)*x(i-254)
        SACH *+             ; STORE b255(i+1).
*
        ZALR *,AR3          ; LOAD ACCH WITH b254(i) & ROUND.
        MPYA *-,AR2         ; b254(i+1) = b254(i) + P
*                           ; P = 2*beta*err(i)*x(i-253)
        SACH *+             ; STORE b254(i+1).
*
        ZALR *,AR3          ; LOAD ACCH WITH b253(i) & ROUND.
        MPYA *-,AR2         ; b253(i+1) = b253(i) + P
*                           ; P = 2*beta*err(i)*x(i-252)
        SACH *+             ; STORE b253(i+1).
          .
          .
          .
        ZALR *,AR3          ; LOAD ACCH WITH b1(i) & ROUND.
        MPYA *-,AR2         ; b1(i+1) = b1(i) + P
*                           ; P = 2*beta*err(i)*x(i-0)
        SACH *+             ; STORE b1(i+1).
*
        ZALR *              ; LOAD ACCH WITH b0(i) & ROUND.
        APAC                ; b0(i+1) = b0(i) + P
        SACH *+             ; STORE b0(i+1).
*
        RET                 ; RETURN TO CALLING ROUTINE.
```

Table 5-2 provides a comparison of data memory, program memory, and CPU cycles for a 256-tap adaptive FIR filter implementation using the TMS32020 and TMS320C25. Note that n = 256 in the table.

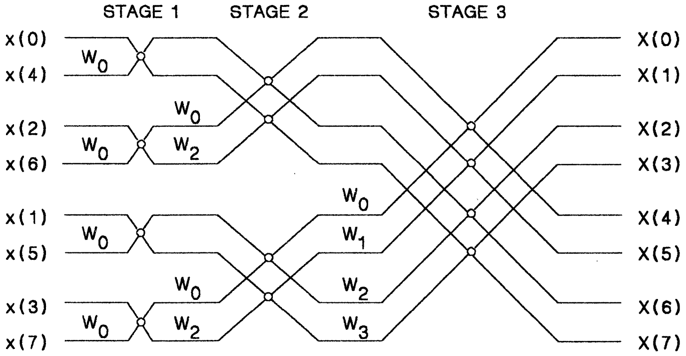
Table 5-2. 256-Tap Adaptive Filtering Memory Space and Time Requirements

DEVICE	WORDS IN MEMORY Data	 Program	CPU CYCLES
TMS32020	5 + 2n	29 + 5n	30 + 6n
TMS320C25	5 + 2n	30 + 3n	33 + 4n

5.7.4 Fast Fourier Transforms (FFT)

Fourier transforms are an important tool often used in digital signal processing systems. The purpose of the transform is to convert information from the time domain to the frequency domain. The inverse Fourier transform converts information back to the time domain from the frequency domain. Implementations of Fourier transforms that are computationally efficient are known as Fast Fourier Transforms (FFTs). The theory and implementation of FFTs on the TMS32020 has been discussed in an application report in the book, *Digital Signal Processing Applications with the TMS320 Family*. The TMS320C25 reduces the execution time of all FFTs by virtue of its 100-ns instruction cycle time.

In addition to the shorter cycle time, an addressing feature has been added to the TMS320C25 which provides execution speed and program memory enhancements for radix-2 FFTs. As demonstrated in Figure 5-6 and Figure 5-7 the inputs or outputs of an FFT are not in sequential order, i.e., they are scrambled. The scrambling of the data addressing is a direct result of the radix-2 FFT derivation. Observation of the figures and the relationship of the input and output addressing in each case reveal that the address indexing is a bit-reversed order, as shown in Table 5-3. As a result, either the data input sequence or the data output sequence must be scrambled in association with the execution of the FFT.



LEGEND FOR TWIDDLE FACTOR: $W_0 = W_8^0$ $W_1 = W_8^1$ $W_2 = W_8^2$ $W_3 = W_8^3$

Figure 5-6. An In-Place DIT FFT with In-Order Outputs and Bit-Reversed Inputs

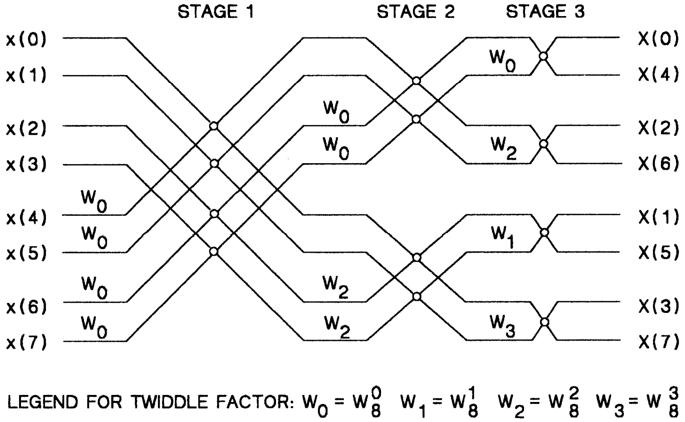


Figure 5-7. An In-Place DIT FFT with In-Order Inputs but Bit-Reversed Outputs

Table 5-3. Bit-Reversal Algorithm for an 8-Point Radix-2 DIT FFT

INDEX	BIT PATTERN	BIT-REVERSED PATTERN	BIT-REVERSED INDEX
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

On the TMS32020, the bit reversal is handled by loading the accumulator with pairs of points that needed to be swapped and then storing them back in the swapped locations. An addressing feature that uses reverse carry-bit propagation allows the TMS320C25 to scramble the inputs or outputs while it is performing the I/O. The addressing mode is part of the indirect addressing implemented with the auxiliary registers and the associated arithmetic unit. In this mode (a derivative of indexed addressing), a value (index) contained in AR0 is either added or subtracted from the auxiliary register being pointed to by the ARP. However, instead of propagating the carry bit in the forward direction, it is propagated in the reverse direction. The result is a scrambling in the address access.

The procedure for generating the bit-reversal address sequence is to load AR0 with a value corresponding to one-half the length of the FFT and to load another auxiliary register, e.g., AR1, with the base address of the data array. Implementations of FFTs involve complex arithmetic; as a result, there are two data memory locations (one real and one imaginary) associated with every data sample. Generally, the samples are stored in memory in pairs with the real part in the even address locations and the imaginary part in the odd address location. This means that the offset from the base address for any given sample is twice the sample index. Real input data is easily transferred into the data memory and stored in the scrambled order, with every other location in the data memory representing the imaginary part of the data.

The following list shows the contents of auxiliary register AR1 when AR0 is initialized with a value of 8 (8-point FFT) and when data is being transferred by the code that follows.

```
       MSB                                         LSB
AR0:    0 0 0 0    0 0 0 0    0 0 0 0    1 0 0 0   8-Point FFT

AR1:    0 0 0 0    0 0 1 0    0 0 0 0    0 0 0 0   Base Address

       RPTK   7
       IN     *BR0+,PA0

AR1:    0 0 0 0    0 0 1 0    0 0 0 0    0 0 0 0   XR(0)

AR1:    0 0 0 0    0 0 1 0    0 0 0 0    1 0 0 0   XR(4)

AR1:    0 0 0 0    0 0 1 0    0 0 0 0    0 1 0 0   XR(2)

AR1:    0 0 0 0    0 0 1 0    0 0 0 0    1 1 0 0   XR(6)

AR1:    0 0 0 0    0 0 1 0    0 0 0 0    0 0 1 0   XR(1)

AR1:    0 0 0 0    0 0 1 0    0 0 0 0    1 0 1 0   XR(5)

AR1:    0 0 0 0    0 0 1 0    0 0 0 0    0 1 1 0   XR(3)

AR1:    0 0 0 0    0 0 1 0    0 0 0 0    1 1 1 0   XR(7)
```

Example 5-47 consists of lists of macros used in the implementation of FFTs. The first macro implements the bit reversal in the way necessary for the TMS32020 and is not necessary for implementations on the TMS320C25.

Example 5-47. FFT Macros

```
BITREV  $MACRO  PR,PI,QR,QI
*
* BIT REVERSAL CODE - SWAP PR AND QR, SWAP PI AND QI.
*
        ZALH :PR:
        ADDS :QR:
        SACL :PR:
        SACH :QR:
        ZALH :PI:
        ADDS :QI:
        SACL :PI:
        SACH :QI:
        $END
*
COMBO   $MACRO  R1,I1,R2,I2,R3,I3,R4,I4
*
* CALCULATE PARTIAL TERMS FOR R3, R4, I3, AND I4.
        LAC  :R3:,14    ACC   := (1/4)(R3)
        ADD  :R4:,14    ACC   := (1/4)(R3+R4)
        SACH :R3:,1     R3    := (1/2)(R3+R4)
        SUB  :R4:,15    ACC   := (1/4)(R3+R4)-(1/2)(R4)
        SACH :R4:,1     R4    := (1/2)(R3-R4)
        LAC  :I3:,14    ACC   := (1/4)(I3)
        ADD  :I4:,14    ACC   := (1/4)(I3+I4)
        SACH :I3:,1     I3    := (1/2)(I3+I4)
        SUB  :I4:,15    ACC   := (1/4)(I3+I4)-(1/2)(I4)
        SACH :I4:,1     I4    := (1/2)(I3-I4)
*
* CALCULATE PARTIAL TERMS FOR R2, R4, I2, AND I4.
        LAC  :R1:,14    ACC   := (1/4)(R1)
        ADD  :R2:,14    ACC   := (1/4)(R1+R2)
        SACH :R1:,1     R1    := (1/2)(R1+R2)
        SUB  :R2:,15    ACC   := (1/4)(R1+R2)-(1/2)(R2)
        ADD  :I4:,15    ACC   := (1/4)[(R1-R2)+(I3-I4)]
        SACH :R2:       R2    := (1/4)[(R1-R2)+(I3-I4)]
        SUBH :I4:       ACC   := (1/4)[(R1-R2)-(I3-I4)]
        DMOV :R4:       I4    := R4 = (1/2)(R3-R4)
        SACH :R4:       R4    := (1/4)[(R1-R2)-(I3-I4)]
        LAC  :I1:,14    ACC   := (1/4)(I1)
        ADD  :I2:,14    ACC   := (1/4)(I1+I2)
        SACH :I1:,1     I1    := (1/2)(I1+I2)
        SUB  :I2:,15    ACC   := (1/4)(I1+I2)-(1/2)(I2)
        SUB  :I4:,15    ACC   := (1/4)[(I1-I2)-(I3-I4)]
        SACH :I2:       I2    := (1/4)[(I1-I2)-(I3-I4)]
        ADDH :I4:       ACC   := (1/4)[(I1-I2)+(I3-I4)]
        SACH :I4:       I4    := (1/4)[(I1-I2)+(I3-I4)]
*
* CALCULATE PARTIAL TERMS FOR R1, R3, I1, AND I3.
        LAC  :R1:,15    ACC   := (1/4)(R1+R2)
        ADD  :R3:,15    ACC   := (1/4)[(R1+R2)+(R3+R4)]
        SACH :R1:       R1    := (1/4)[(R1+R2)+(R3+R4)]
        SUBH :R3:       ACC   := (1/4)[(R1+R2)-(R3+R4)]
        SACH :R3:       R3    := (1/4)[(R1+R2)-(R3+R4)]
        LAC  :I1:,15    ACC   := (1/4)(I1+I2)
        ADD  :I3:,15    ACC   := (1/4)[(I1+I2)+(I3+I4)]
        SACH :I1:       I1    := (1/4)[(I1+I2)+(I3+I4)]
        SUBH :I3:       ACC   := (1/4)[(I1+I2)-(I3+I4)]
        SACH :I3:       I3    := (1/4)[(I1+I2)-(I3+I4)]
        $END
```

```
*
ZERO    $MACRO  PR,PI,QR,QI
*
* CALCULATE Re[P+Q] AND Re[P-Q]
        LAC  :PR:,15    ACC   := (1/2)(PR)
        ADD  :QR:,15    ACC   := (1/2)(PR+QR)
        SACH :PR:       PR    := (1/2)(PR+QR)
        SUBH :QR:       ACC   := (1/2)(PR+QR)-(QR)
        SACH :QR:       QR    := (1/2)(PR-QR)
*
* CALCULATE Im[P+Q] AND Im[P-Q]
        LAC  :PI:,15    ACC   := (1/2)(PI)
        ADD  :QI:,15    ACC   := (1/2)(PI+QI)
        SACH :PI:       PI    := (1/2)(PI+QI)
        SUBH :QI:       ACC   := (1/2)(PI+QI)-(QI)
        SACH :QI:       QI    := (1/2)(PI-QI)
        $END
*
PIBY4   $MACRO  PR,PI,QR,QI,W
*
        LT   :W:        T REGISTER := W=COS(PI/4)=SIN(PI/4)
        LAC  :QI:,14    ACC   := (1/4)(QI)
        SUB  :QR:,14    ACC   := (1/4)(QI-QR)
        SACH :QI:,1     QI    := (1/2)(QI-QR)
        ADD  :QR:,15    ACC   := (1/4)(QI+QR)
        SACH :QR:,1     QR    := (1/2)(QI+QR)
        LAC  :PR:,14    ACC   := (1/4)(PR)
        MPY  :QR:       P REGISTER := (1/4)(QI+QR)*W
        APAC            ACC   := (1/4)[PR+(QI+QR)*W]
        SACH :PR:,1     PR    := (1/2)[PR+(QI+QR)*W]
        SPAC            ACC   := (1/4)(PR)
        SPAC            ACC   := (1/4)[PR-(QI+QR)*W]
        SACH :QR:,1     QR    := (1/2)[PR-(QI+QR)*W]
        LAC  :PI:,14    ACC   := (1/4)(PI)
        MPY  :QI:       P REGISTER := (1/4)(QI-QR)*W
        APAC            ACC   := (1/4)[PI+(QI-QR)*W]
        SACH :PI:,1     PI    := (1/2)[PI+(QI-QR)*W]
        SPAC            ACC   := (1/4)(PI)
        SPAC            ACC   := (1/4)[PI-(QI-QR)*W]
        SACH :QI:,1     QI    := (1/2)[PI-(QI-QR)*W]
        $END
*
PIBY2   $MACRO  PR,PI,QR,QI
*
* CALCULATE Re[P+jQ] AND Re[P-jQ]
        LAC  :PI:,15    ACC   := (1/2)(PI)
        SUB  :QR:,15    ACC   := (1/2)(PI-QR)
        SACH :PI:       PI    := (1/2)(PI-QR)
        ADDH :QR:       ACC   := (1/2)(PI-QR)+(QR)
        SACH :QR:       QR    := (1/2)(PI+QR)
*
* CALCULATE Im[P+jQ] AND Im[P-jQ]
        LAC  :PR:,15    ACC   := (1/2)(PR)
        ADD  :QI:,15    ACC   := (1/2)(PR+QI)
        SACH :PR:       PR    := (1/2)(PR+QI)
        SUBH :QI:       ACC   := (1/2)(PR+QI)-(QI)
        DMOV :QR:       QR    -> QI
        SACH :QR:       QR    := (1/2)(PR-QI)
        $END
```

```
*
PI3BY4  $MACRO   PR,PI,QR,QI,W
*
        LT    :W:        T REGISTER := W=COS(PI/4)=SIN(PI/4)
        LAC   :QI:,14    ACC   := (1/4)(QI)
        SUB   :QR:,14    ACC   := (1/4)(QI-QR)
        SACH  :QI:,1     QI    := (1/2)(QI-QR)
        ADD   :QR:,15    ACC   := (1/4)(QI+QR)
        SACH  :QR:,1     QR    := (1/2)(QI+QR)
        LAC   :PR:,14    ACC   := (1/4)(PR)
        MPY   :QI:       P REGISTER := (1/4)(QI-QR)*W
        APAC             ACC   := (1/4)[PR+(QI-QR)*W]
        SACH  :PR:,1     PR    := (1/2)[PR+(QI-QR)*W]
        SPAC             ACC   := (1/4)(PR)
        SPAC             ACC   := (1/4)[PR-(QI-QR)*W]
        MPY   :QR:       P REGISTER := (1/4)(QI+QR)*W
        SACH  :QR:,1     QR    := (1/2)[PR-(QI-QR)*W]
        LAC   :PI:,14    ACC   := (1/4)(PI)
        SPAC             ACC   := (1/4)[PI-(QI+QR)*W]
        SACH  :PI:,1     PI    := (1/2)[PI-(QI+QR)*W]
        APAC             ACC   := (1/4)(PI)
        APAC             ACC   := (1/4)[PI+(QI+QR)*W]
        SACH  :QI:,1     QI    := (1/2)[PI+(QI+QR)*W]
        $END
```

Example 5-48 shows the bit-reversal addressing capability of the TMS320C25 for implementing an 8-point DIT FFT. On the TMS320C25 the following instructions input the data and store it in memory in the bit-reversed sequence:

```
RPTK   7
IN     *BR0+,PA0
```

This code combines the functions of input and bit-reversal addressing which were previously implemented separately on the TMS32020. The following implementation uses a separate bit-reverse macro:

```
RPTK   7
IN     *0+,PA0

BITREV    X1R,X1I,X4R,X4I
BITREV    X3R,X3I,X6R,X6I
```

Example 5-48. An 8-Point DIT FFT

```
X0R    EQU  00
X0I    EQU  01
X1R    EQU  02
X1I    EQU  03
X2R    EQU  04
X2I    EQU  05
X3R    EQU  06
X3I    EQU  07
X4R    EQU  08
X4I    EQU  09
X5R    EQU  10
X5I    EQU  11
X6R    EQU  12
X6I    EQU  13
X7R    EQU  14
X7I    EQU  15
W      EQU  16
WVALUE EQU  >5A82     ; VALUE FOR SIN(45) OR COS(45)
*
* INITIALIZE FFT PROCESSING.
*
FFT    SPM  0         ; NO SHIFT OF PR OUTPUT
       SSXM           ; SET SIGN-EXTENSION MODE.
       ROVM           ; RESET OVERFLOW MODE.
       LDPK 4         ; SET DATA PAGE POINTER TO 4.
       LALK WVALUE    ; GET TWIDDLE FACTOR VALUE.
       SACL W         ; STORE SIN(45) OR COS(45).
*
* INPUT SAMPLES, STORING IN BIT-REVERSED ORDER.
*
       LARK AR0,8     ; LOAD LENGTH OF FFT IN AR0.
       LRLK AR1,>200  ; LOAD AR1 WITH DATA PAGE 4 ADDRESS.
       LARP AR1
       RPTK 7
       IN   *BR0+,PA0 ; ONLY REAL-VALUED INPUT
*
* 1ST & 2ND STAGES COMBINED WITH DIVIDE-BY-4 INTERSTAGE
* SCALING.
*
       COMBO  X0R,X0I,X1R,X1I,X2R,X2I,X3R,X3I,
       COMBO  X4R,X4I,X5R,X5I,X6R,X6I,X7R,X7I.
*
* 3RD STAGE WITH DIVIDE-BY-2 INTERSTAGE SCALING.
*
       ZERO    X0R,X0I,X4R,X4I
       PIBY4   X1R,X1I,X5R,X5I,W
       PIBY2   X2R,X2I,X6R,X6I
       PI3BY4  X3R,X3I,X7R,X7I,W
*
* OUTPUT SAMPLES, SUPPLYING IN SEQUENTIAL ORDER.
*
       LRLK AR1,>200  ; LOAD AR1 WITH DATA PAGE 4 ADDRESS.
       RPTK 15
       OUT  *+,PA0    ; COMPLEX-VALUED OUTPUT
       RET
```

Table 5-4 provides a comparison of execution speed, program memory, and data memory for an 8-point DIT FFT implementation using the TMS32020 and TMS320C25.

Table 5-4. FFT Memory Space and Time Requirements

DEVICE	WORDS IN MEMORY Data	Program	CPU CYCLES	TIME (µs)
TMS32020	17	169	216	43.2
TMS320C25	17	153	178	17.8

5.7.5 PID Control

Control systems are concerned with regulating a process and achieving a desired behaviour or output from the process. A control system consists of three main components: sensors, actuators, and a controller. Sensors measure the behavior of the system. Actuators supply the driving force to ensure the desired behaviour. The controller generates actuator commands corresponding to the error conditions observed by the sensors and the control algorithms programmed in the controller. The controller typically consists of an analog or digital processor.

Analog control systems are usually based on fixed components and are not programmable. They are also limited to using single-purpose characteristics of the error signal, such as P (proportional), I (integral), and D (derivative), or their combination. These limitations, along with other disadvantages of analog systems such as component aging and temperature drift, are causing digital control systems to increasingly replace analog systems in most control applications.

Digital control systems that use a microprocessor/microcontroller are able to implement more sophisticated algorithms of modern control theory, such as state models, deadbeat control, state estimation, optimal control, and adaptive control. Digital control algorithms deal with the processing of digital signals and are similar to DSP algorithms. The TMS320C2x instruction set can therefore be used very effectively in digital control systems.

The most commonly used algorithm in both analog and digital control systems is the PID (Proportional, Integral, and Derivative) algorithm. The classical PID algorithm is given by

$$u(t) = K_p\, e(t) + K_i \int e dt + K_d\, de/dt$$

The PID algorithm must be converted into a digital form for implementation on a microprocessor. Using a rectangular approximation for the integral, the PID algorithm can be approximated as

$$u(n) = u(n\text{-}1) + K_1\, e(n) + K_2\, e(n\text{-}1) + K_3\, e(n\text{-}2)$$

This algorithm is implemented in Example 5-49.

Example 5-49. PID Control

```
        TITL 'PID CONTROL'
        DEF  PID
*
* THIS ROUTINE IMPLEMENTS A PID ALGORITHM.
*
UN      EQU  0         ; OUTPUT OF CONTROLLER
E0      EQU  1         ; LATEST ERROR SAMPLE
E1      EQU  2         ; PREVIOUS ERROR SAMPLE
E2      EQU  3         ; OLDEST ERROR SAMPLE
K1      EQU  4         ; GAIN CONSTANT
K2      EQU  5         ; GAIN CONSTANT
K3      EQU  6         ; GAIN CONSTANT
*
* ASSUME DATA PAGE 0 IS SELECTED.
*
PID     IN   E0,PA0    ; READ NEW ERROR SAMPLE
        LAC  UN        ; ACC = u(n-1)
        LT   E2        ; LOAD T REG WITH OLDEST SAMPLE
        MPY  K2        ; P = K2*e(n-2)
        LTD  E1        ; ACC = u(n-1)+K2*e(n-2)
        MPY  K1        ; P = K1*e(n-1)
        LTD  E0        ; ACC = u(n-1)+K1*e(n-1)+K2*e(n-2)
        MPY  K0        ; P = K0*e(n)
        APAC           ; ACC = u(n-1)+K0*e(n)+K1*e(n-1)
*                      ;       +K2*e(n-2)
        SACH UN,1      ; STORE OUTPUT
        OUT  UN,PA1    ; SEND IT
```

The PID loop takes 13 cycles to execute (2.6 μs at a 20-MHz clock rate or 1.3 μs at a 40-MHz clock rate). The TMS320 can also be used to implement more sophisticated algorithms such as state modeling, adaptive control, state estimation, Kalman filtering, and optimal control. Other functions that can be implemented are noise filtering, stability analysis, and additional control loops.

6. Hardware Applications

The TMS320C2x has the power and flexibility to satisfy a wide range of system requirements. The 128K address space for program and data memory can be used to interface external memories or to implement single-chip solutions. Peripheral devices can be interfaced to the TMS320C2x to perform analog signal acquisition at different levels of signal quality.

Information and examples on how to interface the TMS320C2x to external devices are presented in this section. The examples given are general enough to be easily adapted for a particular system requirement. For more detailed information, refer to the application reports, "Hardware Interfacing to the TMS32020" and "TMS32020 and MC68000 Interface," included in the book, *Digital Signal Processing Applications with the TMS320 Family, Volume I.* Refer also to the application report, *Hardware Interfacing to the TMS320C25,* published separately. Appendix F provides listings and brief information regarding TI memories, peripherals, and analog conversion devices that are used in many of the applications in this section.

The following buses, port, and control signals provide system interface to the TMS320C2x processor:

- 16-bit address bus (A15-A0)
- 16-bit data bus (D15-D0)
- Serial port
- $\overline{\text{PS}}$, $\overline{\text{DS}}$, $\overline{\text{IS}}$ (program, data, I/O space select)
- R/$\overline{\text{W}}$ (read/write) and $\overline{\text{STRB}}$ (strobe)
- $\overline{\text{READY}}$ and $\overline{\text{MSC}}$ (microstate complete)
- $\overline{\text{HOLD}}$ and $\overline{\text{HOLDA}}$ (hold acknowledge)
- $\overline{\text{INT}}$(2-0) and $\overline{\text{IACK}}$ (interrupt acknowledge)
- $\overline{\text{BIO}}$ (branch control) and XF (external flag)
- $\overline{\text{SYNC}}$ (synchronization) and $\overline{\text{BR}}$ (bus request)

Major hardware applications discussed in this section are listed below and on the next page.

- System Control Circuitry (Section 6.1 on page 6-3)
  - Powerup reset circuit
  - Crystal oscillator circuit
  - User target design considerations when using the XDS
- Interfacing Memories (Section 6.2 on page 6-10)
  - Interfacing PROMs
  - Wait-state generator
  - Interfacing EPROMs
  - Interfacing static RAMs
  - Interface timing analysis
- Direct Memory Access (Section 6.3 on page 6-29)

- Global Memory (Section 6.4 on page 6-32)
- Interfacing Peripherals (Section 6.5 on page 6-34)
  - Combo-codec interface
  - AIC interface
  - D/A interface
  - A/D interface
  - I/O ports
- System Applications (Section 6.6 on page 6-45)
  - Echo cancellation
  - High-speed modem
  - Voice coding
  - Graphics and image processing
  - High-speed control
  - Instrumentation and numeric processing

6.1 System Control Circuitry

The system control circuitry performs functions that are critical for proper system initialization and operation. A powerup reset circuit design and a crystal oscillator circuit design are presented in this section. The powerup reset circuit assures that a reset of the part occurs only after the oscillator is running and stabilized. The oscillator circuit described allows the use of third-overtone crystals that are more readily available at frequencies above 20 MHz. For a more detailed discussion of system control circuitry, refer to the application report, *Hardware Interfacing to the TMS320C25.*

6.1.1 Powerup Reset Circuit

The reset circuit shown in Figure 6-1 performs a powerup reset; i.e., the TMS320C2x is reset when power is applied. Note that the switch circuit must include debounce circuitry. Driving the $\overline{\text{RS}}$ signal low initializes the processor. Reset affects several registers and status bits (see Section 3.6.2 for a detailed description of the effect of reset on processor status).

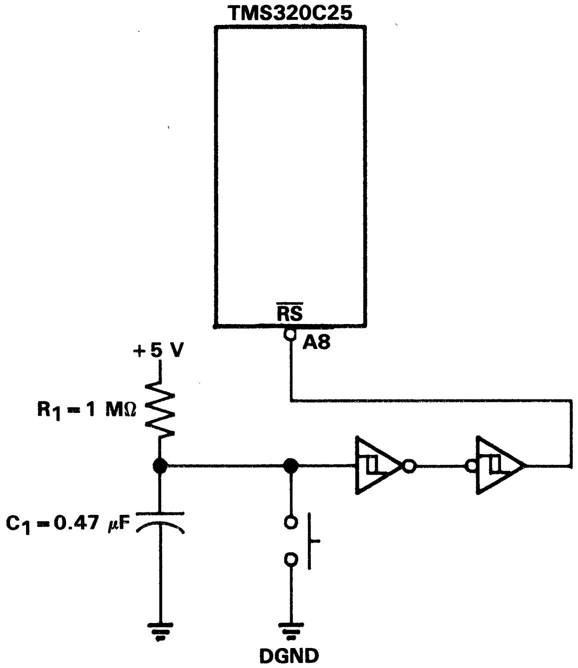


Figure 6-1. Powerup Reset Circuit

For proper system initialization, the reset signal must be applied for at least three CLKOUT cycles, i.e., 300 ns for a TMS320C25 operating at 40 MHz. Upon powerup, it can take several to hundreds of milliseconds before the system oscillator reaches a stable operating state. Therefore, the powerup reset circuit should generate a low pulse on the reset line until the oscillator is stable (i.e., 100 to 200 ms).

The voltage on the reset pin ($\overline{RS}$) is controlled by the R_1C_1 network (see Figure 6-1). After a reset, this voltage rises exponentially according to the time constant R_1C_1, as shown in Figure 6-2.

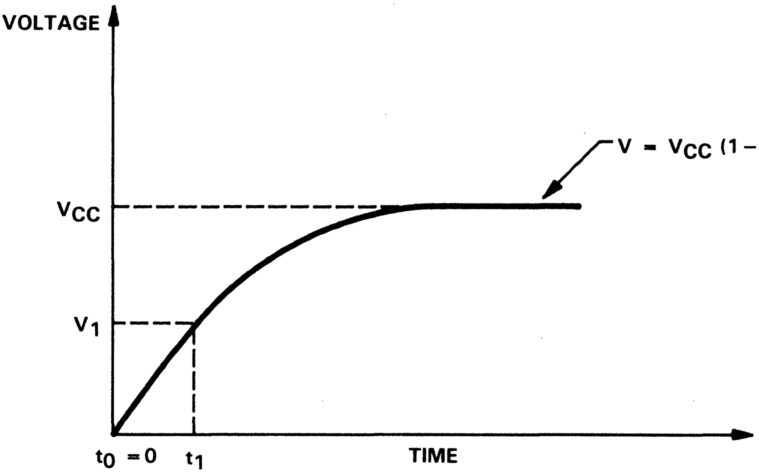


Figure 6-2. Voltage on TMS320C25 Reset Pin

The duration of the low pulse on the reset pin is approximately t_1, which is the time it takes for the capacitor C_1 to be charged to 1.5 V. This is approximately the voltage at which the reset input switches from a logic level 0 to a logic level 1. The capacitor voltage is given by

$$V = V_{CC} \left[1 - e^{-\frac{t}{\tau}} \right] \tag{1}$$

where $\tau = R_1C_1$ is the reset circuit time constant. Solving (1) for t gives

$$t = -R_1C_1 \ln \left[1 - \frac{V}{V_{CC}} \right] \tag{2}$$

For example, setting the following:

$R_1 = 1\ M\Omega$	$V_{CC} = 5\ V$
$C_1 = 0.47\ \mu F$	$V = V_1 = 1.5\ V$

gives $t = t_1 = 167$ ms. In this case, the reset circuit of Figure 6-1 can generate a low pulse of long enough duration (167 ms) to ensure the stabilization of the oscillator upon powerup in most systems.

6.1.2 Crystal Oscillator Circuit

The crystal oscillator circuit shown in Figure 6-3 is designed to operate at 40.96 MHz. Since crystals with fundamental oscillation frequencies of 30 MHz and above are not readily available, a parallel-resonant third-overtone oscillator is used. If a packed clock oscillator is used, oscillator design is of no concern. A third-overtone 40.96-MHz crystal is shown in Figure 6-3.

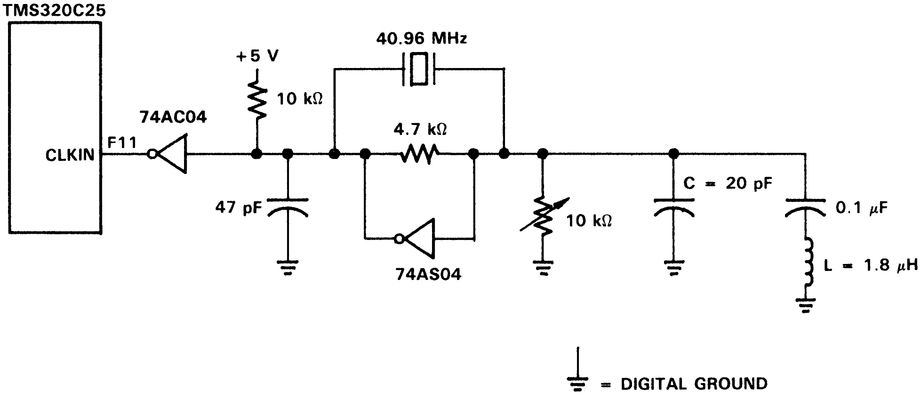


Figure 6-3. Crystal Oscillator Circuit

The 74AS04 inverter in Figure 6-3 provides the 180-degree phase shift that a parallel oscillator requires. The 4.7-kΩ resistor provides the negative feedback that keeps the oscillator in a stable state; i.e., the poles of the system are constrained in a narrow region about the jω axis of the s-plane (analog domain). The 10-kΩ potentiometer is used to bias the 74AS04 in the linear region.

In a third-overtone oscillator, the crystal fundamental frequency must be attenuated so that oscillation is at the third harmonic. This is achieved with an LC circuit that filters out the fundamental, thus allowing oscillation at the third harmonic.

The impedance of the LC network must be inductive at the crystal fundamental frequency and capacitive at the third harmonic. The impedance of the LC circuit is given by

$$z(\omega) = \frac{\frac{L}{C}}{j\left[\omega L - \frac{1}{\omega C}\right]} \tag{3}$$

Therefore, the LC circuit has a pole at

$$\omega_p = \frac{1}{\sqrt{LC}} \tag{4}$$

At frequencies significantly lower than ω_p, the $1/(\omega C)$ term in (3) becomes the dominating term, while ωL can be neglected. This gives

$$z(\omega) = j\omega L \qquad \text{for } \omega << \omega_p \tag{5}$$

In (5), the LC circuit appears inductive at frequencies lower than ω_p. On the other hand, at frequencies much higher than ω_p, the ωL term is the dominant term in (3), and $1/(\omega C)$ can be neglected. This gives

$$z(\omega) = \frac{1}{j\omega C} \qquad \text{for } \omega >> \omega_p \tag{6}$$

The LC circuit in (6) appears increasingly capacitive as frequency increases above ωp. This is shown in Figure 6-4, which is a plot of the magnitude of the impedance of the LC circuit of Figure 6-3 versus frequency.

Based on the discussion above, the design of the LC circuit proceeds as follows: Choose the pole frequency ωp approximately halfway between the crystal fundamental and the third harmonic. The circuit now appears inductive at the fundamental frequency and capacitive at the third harmonic.

In the oscillator of Figure 6-3, ωp = 26.5 MHz, which is approximately halfway between the fundamental and the third harmonic; and C = 20 pF. Then, using (4), L = 1.8 μH.

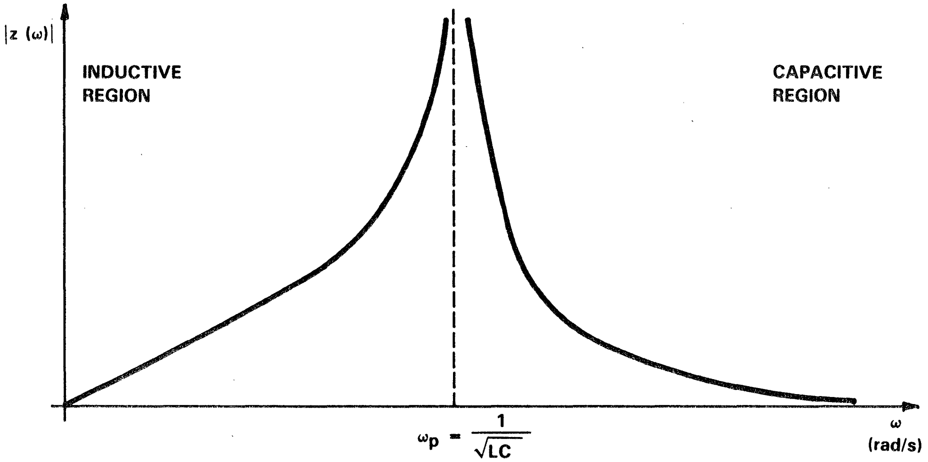


Figure 6-4. Magnitude of Impedance of Oscillator LC Network

6.1.3 User Target Design Considerations When Using the XDS

The architecture for the TMS320C2x emulator (XDS) maximizes speed and performance. No external serial logic levels have been added to any of the address, data, or control signals other than those added to the setup times of READY, $\overline{\text{RS}}$, $\overline{\text{BIO}}$, and $\overline{\text{HOLD}}$, and the propagation delay of $\overline{\text{HOLDA}}$ (hold acknowledge). The additional loading on outputs induced by the XDS is comprehended in the XDS and TMS320C2x device design, thus allowing the user the full drive as specified in the TMS320C2x device data sheet. The DC loading characteristics of inputs is defined in Chapter 9 of the *TMS320C2x XDS User's Guide*.

The emulator architecture works closely with the user's system design to allow the user's memory to have maximum access times. Areas of close interaction between the emulator and target system are:

- Bus control
- READY timing and memory substitution
- Reset and hold
- Miscellaneous considerations.

Bus Control

When the emulator is halted from the keyboard or any of the breakpoint functions, the current state of the device being emulated is extracted by the control processor. This processor communicates with the emulated device over the emulated device's data bus. Additional communication is generated by commands entered from the keyboard.

Before communication between the control processor and the device being emulated begins, the control processor generates an interlock sequence on the emulated device's $\overline{\text{HOLD}}$ input in order to define data bus ownership. Once the target $\overline{\text{HOLD}}$ is deactivated, this interlock prevents the target system from receiving an active $\overline{\text{HOLDA}}$ until the emulator has completed accessing the processor resources. The emulator will not attempt to use the data bus until the interlock is successful, thus guaranteeing that it will not try to use the data bus when $\overline{\text{HOLDA}}$ is asserted to the target system.

When communication between the control processor and the device being emulated is complete, the hold interlock is released, and the target system can again receive hold acknowledge when $\overline{\text{HOLD}}$ is asserted. At this point, the emulator is waiting for another command from the keyboard. Communication between the device being emulated and the control process occurs when $\overline{\text{DS}}$, $\overline{\text{PS}}$, $\overline{\text{IS}}$, and $\overline{\text{HOLDA}}$ are all high.

The target system should drive the data bus only when the following conditions are met:

- $\overline{\text{HOLDA}}$ is active, or
- $\overline{\text{DS}}$, $\overline{\text{PS}}$, or $\overline{\text{IS}}$ is active and R/$\overline{\text{W}}$ is high.

The XDS hardware uses the data bus only while the above signals are inactive. When these rules are not followed, the XDS gives a 'PROCESSOR SYNC LOST' 1160 error. This error may also be caused by signal-to-signal shorts in the target system, misalignment of the target connector, poor grounding of the target connector, or wiring errors on the target system.

READY and Memory Substitution

Since the XDS adds one internal level of 7 ns in series with the READY input, the user's system is left with only 10 ns to generate READY. This can be accomplished by generating READY with a 10-ns TIBPAL16R4 device. READY should be generated off $\overline{\text{DS}}$, $\overline{\text{PS}}$, or $\overline{\text{IS}}$ and the decode of the address lines.

The target system must present a valid READY high on each external access, even when using the XDS substitution memory. Suggested implementation of READY logic on the target system should hold READY high until target memory requiring wait states is addressed.

The XDS provides two types of memory substitution: fast static RAM at a fixed address and slower dynamic RAM at mappable addresses. The user is responsible for deselecting target memory residing in the same address of the emulator's fast static memory if this emulator memory is mapped in. (Note that the target should not drive the data bus on a read.) This fast static emulator substitution memory consists of 8K words of fast static RAM, which can be individually mapped in as 4K words of program memory starting at address 0000 and 4K words of data memory starting at location 0000. In this case, the target system cannot drive the data bus even though $\overline{\text{DS}}$ or $\overline{\text{PS}}$ is active. Although this emulator static RAM can operate with zero wait states, target wait states can be modeled by the user using the target READY signal. However, this sensitivity requires that the target system eventually respond with a valid READY high or the emulator waits until it does.

The slower dynamic RAM controls bus access through the $\overline{\text{DS}}$ or $\overline{\text{PS}}$ control signals. The target system can drive the data bus when $\overline{\text{PS}}$ or $\overline{\text{IS}}$ is asserted. Emulator logic assures that $\overline{\text{DS}}$, $\overline{\text{PS}}$, and $\overline{\text{IS}}$ are returned to their inactive state when the dynamic RAM substitution memory uses the data bus on reads.

The dynamic RAM substitution memory always uses more than one clock to return data. An access to address space mapped to the dynamic substitution memory is accompanied by the assertion of $\overline{\text{DS}}$ or $\overline{\text{PS}}$, and $\overline{\text{STRB}}$. When the target logic generates a READY high condition, the device appears to complete the memory cycle by driving $\overline{\text{DS}}$, $\overline{\text{PS}}$, $\overline{\text{IS}}$, or $\overline{\text{STRB}}$ to their inactive states at their normal switching times. The device under emulation is held not ready for at least one extra clock cycle or until the memory substitution data is available. The memory substitution data is then driven onto the data bus on reads while all bus control signals at the target connector are high.

Note that additional wait states can be added with the use of the target READY line. In this case, the memory control lines model the target access timing. However, the program cycle count is affected by the additional cycles internal to the emulator's access of the dynamic RAM. Since the system responds to the READY line, the target must eventually return a valid READY high on each access.

Miscellaneous Considerations

When the XDS is powered-up, the device under emulation is placed in the run mode with all memory substitution turned off. The control processor does not attempt to communicate with the device under emulation until the user communicates with the emulator. If the target system is asserting $\overline{\text{RS}}$, $\overline{\text{HOLD}}$, or not

READY continuously to the device under emulation, the control processor cannot gain control of the device under emulation and reports a 'PROCESSOR SYNC LOST' 1160 error. This condition can be caused by a powered-up emulator plugged into a powered-down target system. Although the $\overline{\text{RS}}$, $\overline{\text{HOLD}}$, and READY are pulled up with resistors on the emulator, the impedance of the powered-down target system can assert a control signal or load the data bus so that the XDS cannot function properly.

The conductive foam on the XDS target cable must be removed along with the foam on the logic show pod prior to XDS powerup. Failure to do so can also cause the 'PROCESSOR SYNC LOST' 1160 error.

TMS320C25 Designs Using $\overline{\text{HOLD}}$ and $\overline{\text{HOLDA}}$. When the target system asserts $\overline{\text{HOLD}}$ active low while the emulator is processing user-envoked commands requiring access of the device-under-emulation resources, the target will not receive $\overline{\text{HOLDA}}$ until the command is complete.

When interfacing to dynamic RAM in the target system, the user should use READY rather than $\overline{\text{HOLD}}$ to insert refresh cycles. A user-invoked command could hold off $\overline{\text{HOLDA}}$ long enough to lose charge in the dynamic cells. Likewise, the refresh cycle in a 'RAS ONLY REFRESH' system could conflict with the emulator system controlling addressing during command processing if the address lines to the DRAMs are not buffered.

Stack Usage. An interrupt is used to halt the device being emulated, thereby using one of the emulated device stack locations. When an XDS is to be used, the applications programmer should reserve one level of the stack for code development.

Transmission Line Phenomena. Since the XDS target cable is approximately 20 inches, use of advanced CMOS or fast/advanced Schottky TTL may cause line reflections (ringing above input thresholds) on input lines to the XDS. Series termination resistors (22 to 68 ohms) can help to eliminate this problem. In some cases where significant additional signal length is added to XDS outputs, the series resistors on the XDS may not be sufficient to control reflections. In this case, additional corrective actions may be necessary.

Clock Source. The XDS does not support the use of a crystal in the target system. The emulator's clock source can be selected from three sources:

1) A clock (with TTL levels) driven up the target cable on pin F11 (PGA) or pin 35 (PLCC),
2) A socketed changeable crystal on the emulator board (Y1), or
3) A socketed changeable canned TTL oscillator on the EMU (U9).

TMS32020/TMS320C25. The XDS supports both the TMS32020 and TMS320C25. The operating frequencies are 20 MHz and 40 MHz, respectively. The unit is shipped configured as a TMS320C25 emulator, but it can easily be converted to a TMS32020 emulator by replacing the TMS320C25 device on the emulator with the TMS32020 device found in the spare parts kit. The crystal, TTL oscillator, and/or input clock frequency must be adjusted to correspond to TMS32020 specifications. See Section 9 in the *TMS320C2x XDS User's Guide* for additional timing and loading information.

6.2 Interfacing Memories

The TMS320C2x can be interfaced with PROMs, EPROMs, and static RAMs. The speed, cost, and power limitations imposed by a particular application determine the selection of a specific memory device. If speed and maximum throughput are desired, the TMS320C2x can run with no wait states. In this case, memory accesses are performed in a single machine cycle. Alternatively, slower memories can be accessed by introducing an appropriate number of wait states or slowing down the system clock. The latter approach is more appropriate when interfacing to memories with access times slightly longer than those required by the TMS320C2x at full speed.

When wait states are required, the number of wait states depends on the memory access time (see Section 6.2.3). With no wait states, the READY input to the TMS320C2x can be pulled high. If one or more wait states are required, the READY input must be driven low during the cycles in which the TMS320C2x enters a wait state.

The TMS320C2x implements two separate and distinct memory spaces: program space (64K words) and data space (64K words). Distinction between the two spaces is made through the use of the $\overline{PS}$ (program space) and $\overline{DS}$ (data space) pins. A third space, the I/O space, is also available for interfacing with peripherals. This space is selected by the $\overline{IS}$ (I/O space) pin, and is discussed in Section 6.5.

The following brief discussion describes the TMS320C2x read and write cycles. For the memory read and write timing diagrams, refer to the TMS320C2x Data Sheet of Appendix A. For further information about read and write operation, see Sections 3.7.3 and 3.7.4. Throughout this section, Q is used to indicate the duration of a quarter phase of the output clock (CLKOUT1 or CLKOUT2). Memory interfaces discussed in this section assume that the TMS320C2x is running at 40 MHz; i.e., Q = 25 ns.

In a read cycle, the following sequence occurs:

1) Near the beginning of the machine cycle (CLKOUT1 goes low), the address bus and one of the memory select signals ($\overline{PS}$, $\overline{DS}$, or $\overline{IS}$) becomes valid. R/$\overline{W}$ goes high to indicate a read cycle.

2) $\overline{STRB}$ goes low no less than $t_{su(A)}$ = Q-12 ns after the address bus is valid.

3) Early in the second half of the cycle, the READY input is sampled. READY must be stable (low or high) at the TMS320C25 no later than $t_{d(SL-R)}$ = Q-20 ns after $\overline{STRB}$ goes low.

4) With no wait states (READY is high), data must be available no later than $t_{a(SL)}$ = 2Q-23 ns after $\overline{STRB}$ goes low.

The sequence of events that occurs during an external write cycle is the same as the above, with the following differences:

1) R/$\overline{W}$ goes low to indicate a write cycle.

2) The data bus begins to be driven approximately concurrently with $\overline{STRB}$ going low.

3) The data bus enters a high-impedance state no later than $t_{dis(D)} = Q+15$ ns after $\overline{STRB}$ goes high.

6.2.1 Interfacing PROMs

Program memory in a TMS320C2x system can be implemented through the use of PROMs. Two different approaches for interfacing PROMs to the TMS320C2x can be taken depending on whether or not any of the memories in the system require wait states. When no wait states are required for any of the memories, READY can be tied high, and the interface to the PROMs becomes a direct connection. In this first approach, address decoding is not required since the system contains only a small amount of one type of memory. When some of the system memories require wait states, address decoding must be performed to distinguish between two or more memory types with different access times. In the second approach, a valid READY signal that meets the TMS320C2x timing requirements must be provided. An efficient method of accomplishing this is to use one section of circuitry to generate the address decode, and a second, independent section to generate the READY signal. These two approaches are discussed in this section. For more detailed information, see *Hardware Interfacing to the TMS320C25*.

An example of a no-wait-state memory system is the direct PROM interface design shown in Figure 6-5. In this design, the TMS320C25 is interfaced with the Texas Instruments TBP38L165-35, a low-power 2K x 8-bit PROM. The interface timing for the design of Figure 6-5 is shown in Figure 6-6. The same techniques can be used with both of the TMS320C2x devices. The TMS320C25 has been chosen for the following examples because it has the most stringent timing requirements.

The TMS320C25 expects data to be valid no later than 2Q-23 ns after $\overline{STRB}$ goes low. (This is 27 ns for a TMS320C25 operating at 40 MHz.) The access times of the TBP38L165-35 are 35 ns maximum from address $t_{a(A)}$, and 20 ns maximum from chip enable $t_{a(S)}$. On the TMS320C25, address becomes valid a minimum of $t_{su} = Q-12$ ns $= 13$ ns before $\overline{STRB}$ goes low. Therefore, the data appears on the data bus within 27 ns after $\overline{STRB}$ goes low, as required by the TMS320C25.

When a read cycle is followed by a write cycle, care must be taken to avoid bus conflict. Bus conflict also may occur when a TMS320C25 write cycle is followed by a memory read cycle. In this case, the TMS320C25 data lines must enter a high-impedance state before the memory starts driving the data bus.

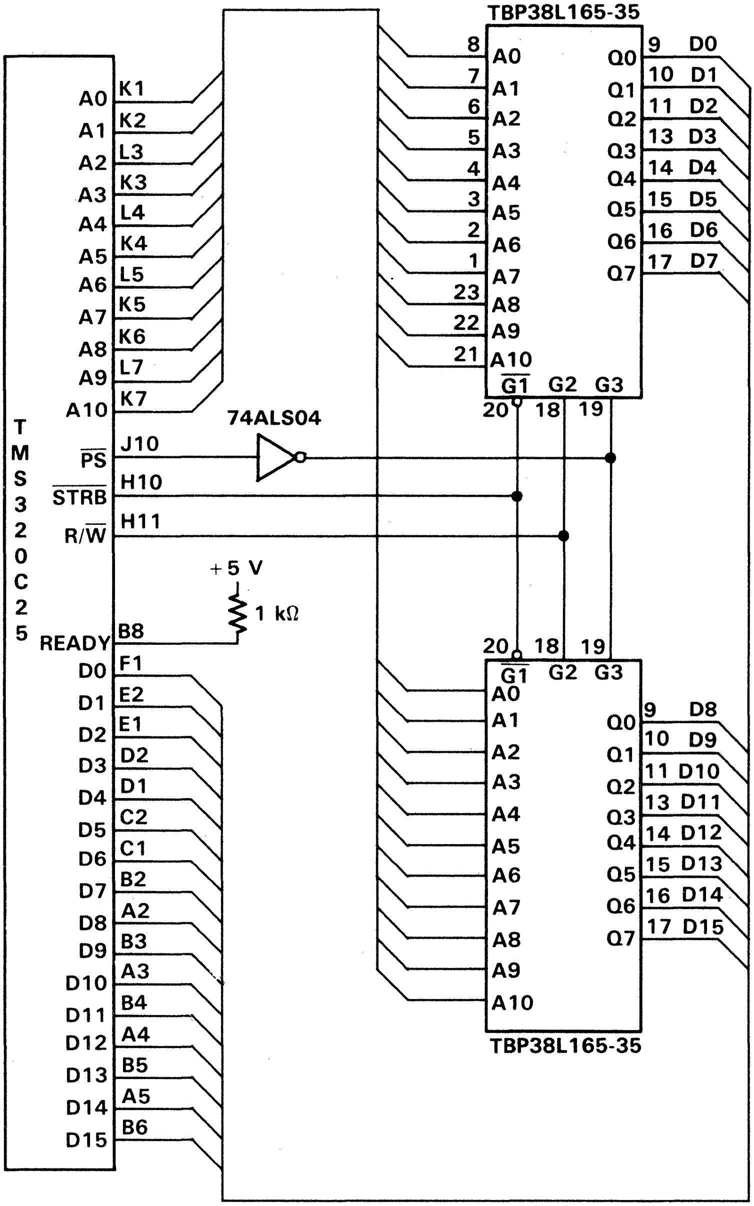


Figure 6-5. Direct Interface of TBP38L165-35 to TMS320C25

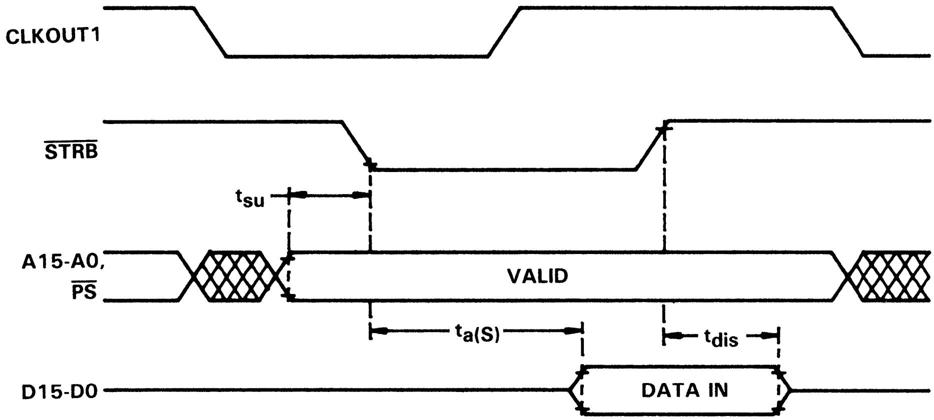


Figure 6-6. Interface Timing of TBP38L165-35 to TMS320C25

The most critical timing parameters of the TBP38L165-35 direct interface to the TMS320C25 are summarized in Table 6-1.

Table 6-1. Timing Parameters of TBP38L165-35 Direct Interface to TMS320C25

DESCRIPTION	SYMBOL USED IN FIGURE 6-6	VALUE
Address setup time	t_{su}	13 ns (min)
TMP38L165-35 access time from chip enable	$t_{a(S)}$	20 ns (max)
TMP38L165-35 disable time	t_{dis}	15 ns (max)

The second design example considers the interface of PROMs to the TMS320C25 using address decoding. An approach that can be used to meet the READY timing requirements is shown in Figure 6-7. This design utilizes one address decoding scheme to generate READY, and a second address decoding scheme to enable the different memory banks. In this design, the memories with no wait states are mapped at the upper half (upper 32K) of the program space. The lower half is used for memories with one or more wait states. This decoding is implemented with the 74AS20 four-input NAND gate.

Address decoding is implemented by the 74AS138. This decoding separates the program space into eight segments of 8K words each. The first four of these segments (lower 32K of address space) are enabled by the $\overline{Y0}$, $\overline{Y1}$, $\overline{Y2}$, and $\overline{Y3}$ outputs of the 74AS138. These segments are used for memories with one or more wait states. The other four segments select memories with no wait states (the TBP38L165s are mapped in segment 5, starting at address >8000). Note that in Figure 6-7, R/$\overline{W}$ is used to enable the 74AS138. This prevents a bus conflict from occurring if an attempt is made to write to the PROMs. Figure 6-8 shows the timing for the circuit shown in Figure 6-7. READY goes high 10 ns (worst case) after the address has become valid.

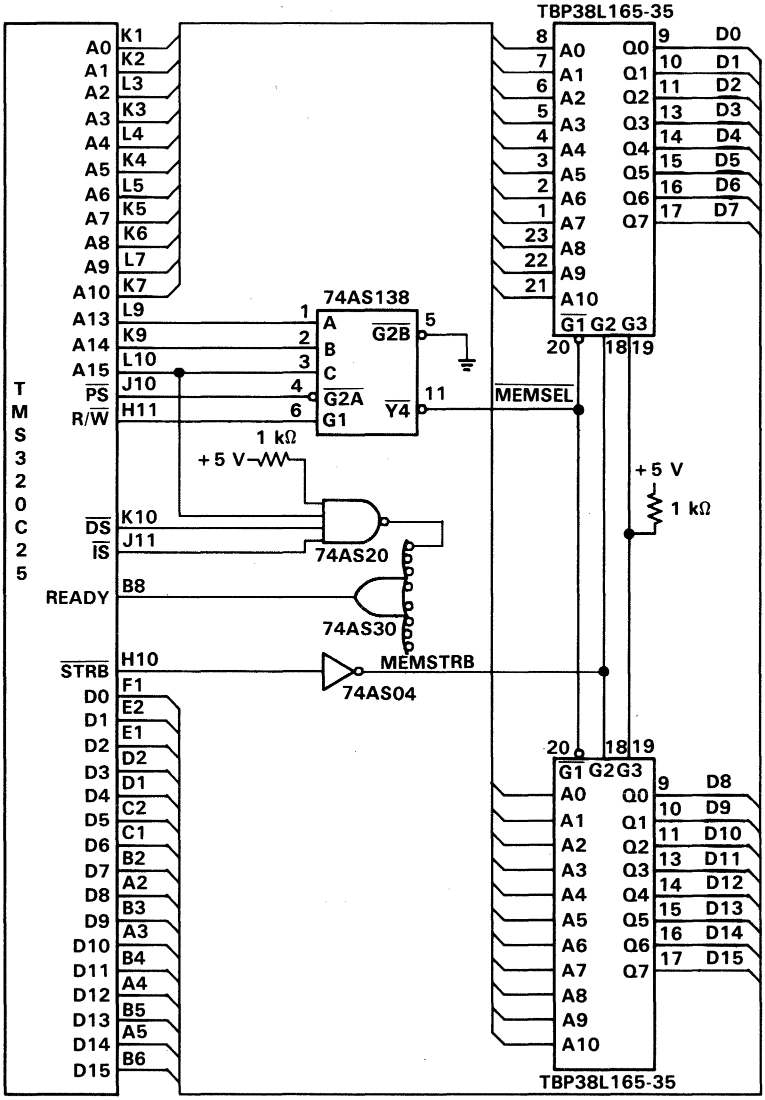


Figure 6-7. Interface of TBP38L165-35 to TMS320C25

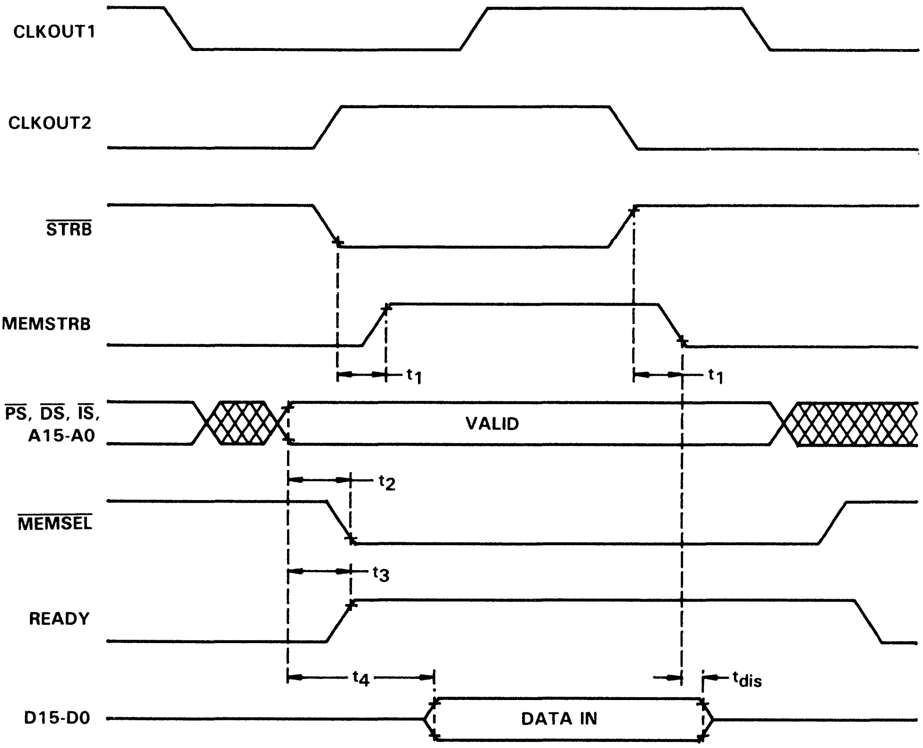


Figure 6-8. Interface Timing of TBP38L165-35 to TMS320C25 (Address Decoding)

The most critical timing parameters of the TBP38L165-35 interface with address decoding to the TMS320C25 are summarized in Table 6-2.

Table 6-2. Timing Parameters of TBP38L165-35 to TMS320C25 (Address Decoding)

DESCRIPTION	SYMBOL USED IN FIGURE 6-8	VALUE
Propagation delay through the 74AS04	t_1	5 ns (max)
Propagation delay through the 74AS138	t_2	10 ns (max)
Address valid to READY	t_3	10 ns (max)
TBP38L165-35 disable time	t_{dis}	15 ns (max)

6.2.2 Wait-State Generator

The READY input of the TMS320C2x allows the capability to interface with memory and peripherals that cannot be accessed in a single cycle. The number of cycles in a memory or I/O access is determined by the state of the READY input. If READY is high when the TMS320C2x samples the READY input, the memory access ends at the next falling edge of CLKOUT1. If READY is low, the memory cycle is extended by one machine cycle, and all other signals remain valid. Figure 6-9 shows a one-wait state memory access. Note that for on-chip program and data memory accesses, the READY input is ignored. Refer to *Hardware Interfacing to the TMS320C25* for detailed information regarding wait-state generation.

The automatic generation of one wait state can be accomplished by the use of the MicroState Complete ($\overline{\text{MSC}}$) signal. The $\overline{\text{MSC}}$ output is asserted low during CLKOUT1 low to indicate the beginning of an internal or external memory or I/O operation (see Figure 6-9). By gating $\overline{\text{MSC}}$ with the address and $\overline{\text{PS}}$, $\overline{\text{DS}}$, and/or $\overline{\text{IS}}$, a one-wait READY signal can be generated.

An alternative approach for the generation of wait states when interfacing with memories and peripherals consists of a wait-state generator. In this design, READY must be valid (low or high) no later than Q-20 ns = 5 ns after $\overline{\text{STRB}}$ goes low. If READY is high, then the memory/peripheral access is completed with the present machine cycle. If READY is low, the access is extended to the next machine cycle; i.e., a wait state is introduced. The number of wait states required depends on the access time t_a of the particular memory device or peripheral. If $t_a < 40$ ns, no wait states are required. If 40 ns $< t_a <$ 140 ns, one wait state must be inserted. In general, N wait states are required for a particular access if

TMS32020: $[200(N-1) + 85]\text{ns} < t_a \leq [200N + 85]\text{ns}$

TMS320C25: $[100(N-1) + 40]\text{ns} < t_a \leq [100N + 40]\text{ns}$

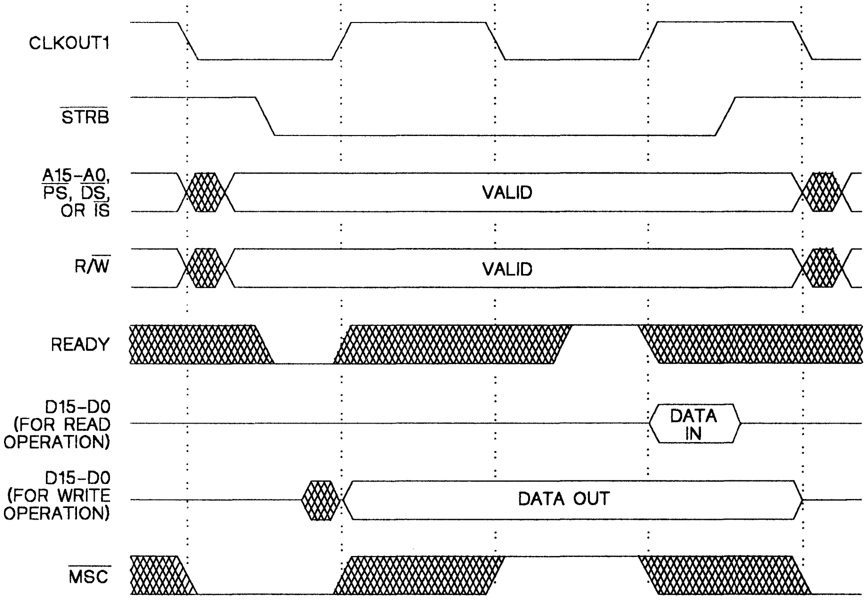


Figure 6-9. One Wait-State Memory Access Timing

The information on the number of wait states required for a memory or peripheral access is summarized in Table 6-3.

Table 6-3. Wait States Required for Memory/Peripheral Access

NUMBER OF WAIT STATES REQUIRED	TMS32020 ACCESS TIME	TMS320C25 ACCESS TIME
0	t_a < 85 ns	t_a < 40 ns
1	85 ns < t_a < 285 ns	40 ns < t_a < 140 ns
2	285 ns < t_a < 485 ns	140 ns < t_a < 240 ns
3	485 ns < t_a < 685 ns	240 ns < t_a < 340 ns
4	685 ns < t_a < 885 ns	340 ns < t_a < 440 ns

A wait-state generator design and timing are shown in Figure 6-10 and Figure 6-11, respectively. In the case of one wait state, time t_1 in Figure 6-11 is the time from address valid to memory select of the particular device that requires the wait state. This corresponds to the propagation delay through the address decode logic. For a 74AS138 decoder, t_1 = 10 ns (max).

Time t_2 is the time from memory select going low to CLKOUT2 going low.

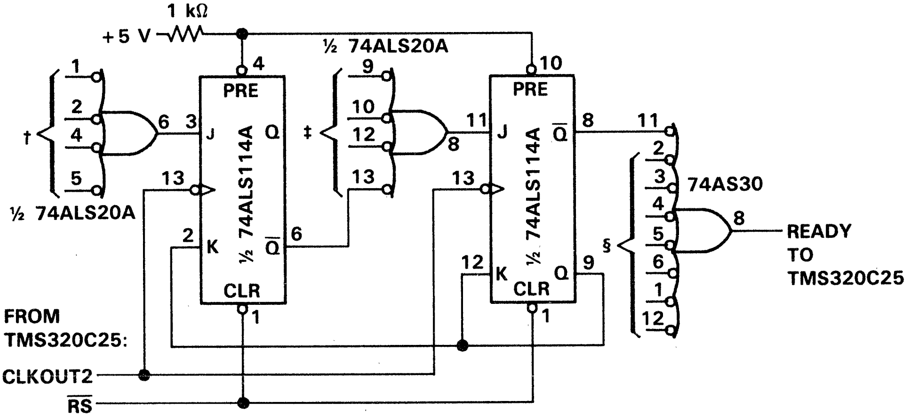
$$t_2 = t_p + t_{su} = 11 \text{ ns} + 20 \text{ ns} = 31 \text{ ns}$$

Time t_3 is the time from CLKOUT2 going low to READY going high.

$$t_3 = 19\text{ ns} + 5\text{ ns} = 24\text{ ns}$$

READY must remain high until it is sampled again, shortly after CLKOUT1 goes high. In Figure 6-10, READY remains high well after CLKOUT1 goes high. At the falling edge of CLKOUT2, the inputs to the J-K flip-flop are J = 1 and K = $\overline{Q}$ = 1, and the flip-flop is in the toggle mode. When CLKOUT2 goes low, $\overline{Q}$ goes back to logic 1. READY goes low and stays low until one of the inputs of the 74AS30 is pulled low.

To implement two wait states, a second J-K flip-flop is utilized as shown in Figure 6-10. This delays READY going high by an additional machine cycle (see Figure 6-11). If more wait states are required, additional J-K flip-flops must be included in the wait-state generator design.



† Connections to other devices in the system that require two wait states. (Inputs not used by other devices should be pulled up.)

‡ Connections to other devices in the system that require one wait state. (Inputs not used by other devices should be pulled up.)

§ Connections to other devices in the system that require zero wait states. (Inputs not used by other devices should be pulled up.)

Figure 6-10. Wait-State Generator Design

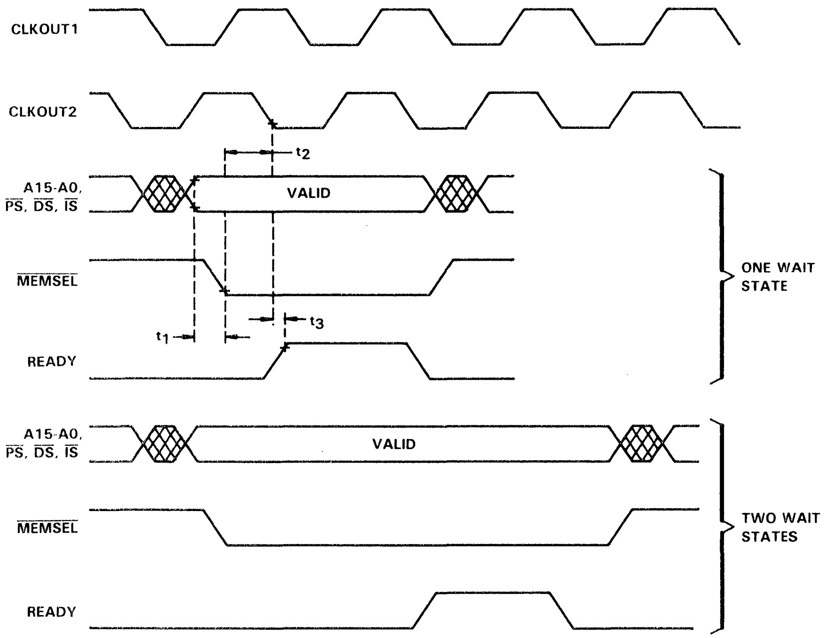


Figure 6-11. Wait-State Generator Timing

6.2.3 Interfacing EPROMs

EPROMs can be a valuable tool for debugging TMS320C2x algorithms during the prototyping stages of a design, and may even be desirable for production. Two different EPROM interfaces to the TMS320C2x are discussed: a direct interface of an EPROM that requires no wait states, and EPROM interfaces that require one and two wait states.

A direct interface similar to that used for PROMs may be implemented when EPROM access time meets the TMS320C2x timing specifications. A Texas Instruments TMS27C292-35 2K x 8-bit EPROM can interface directly to the TMS320C25 with no wait states. The TMS27C292-35 is a CMOS EPROM with access times of 35 ns from valid address and 25 ns from chip select.

When slower, less costly EPROMs are used, a simple flip-flop circuit (see Section 6.2.2 for wait-state generator design) can be used to generate one or more wait states. Figure 6-12 shows an EPROM interface with one wait state, where Wafer Scale WS57C64F-12 8K x 8-bit EPROMs are interfaced to the TMS320C25. The WS57C64F-12 is the slowest member of the WS57C64F EPROM series, but still meets the specifications for one wait state. With slower EPROMs, however, data output turnoff can be slow, and must be taken into consideration in the design. The WS57C64F-12s are mapped at address >2000. Figure 6-13 provides the interface timing diagram.

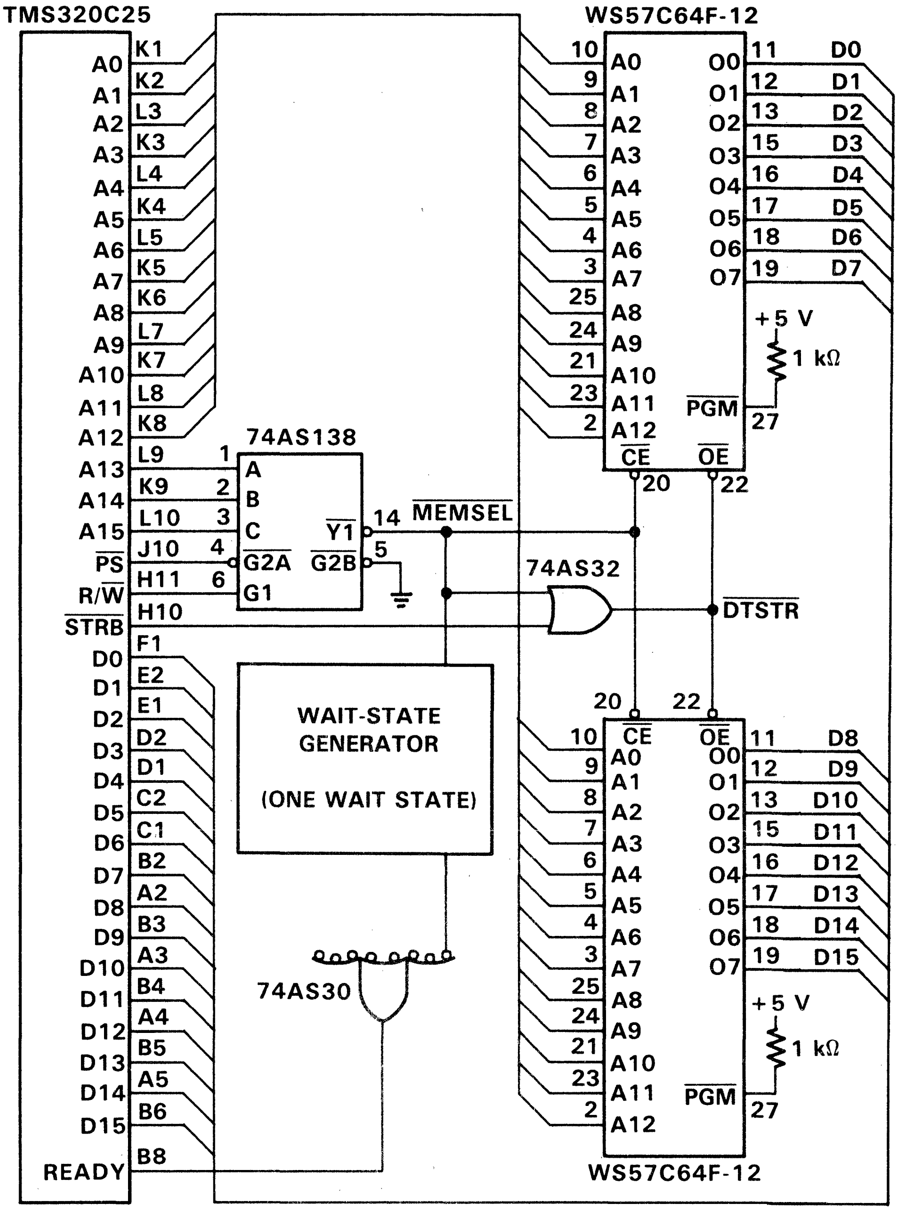


Figure 6-12. Interface of WS57C65F-12 to TMS320C25

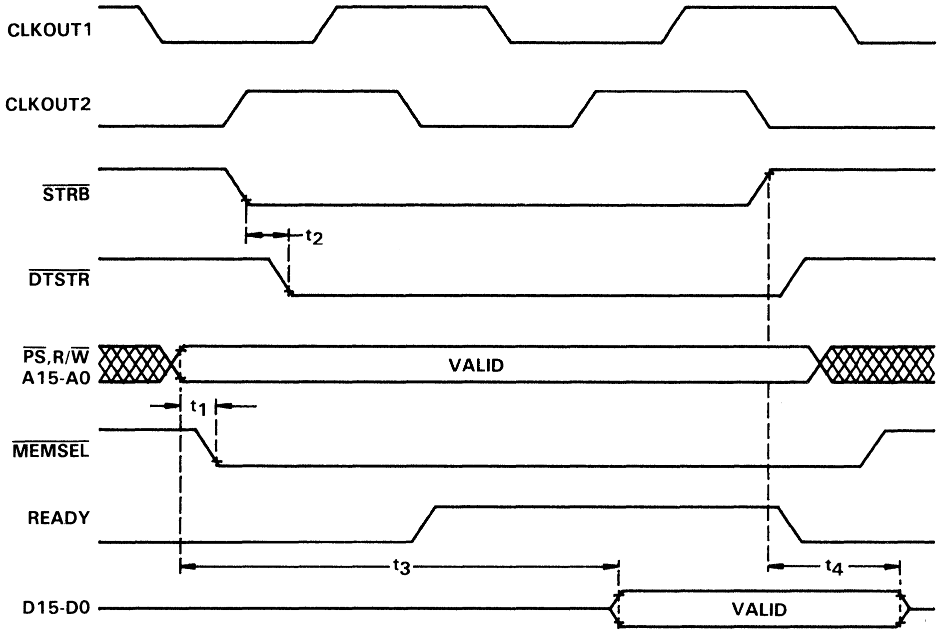


Figure 6-13. Interface Timing of WS57C65F-12 to TMS320C25

Table 6-4 summarizes the most critical timing parameters of the WS57C64F-12 interface to the TMS320C25.

Table 6-4. Timing Parameters of WS57C64F-12 Interface to TMS320C25

DESCRIPTION	SYMBOL USED IN FIGURE 6-13	VALUE
Address valid to $\overline{\text{MEMSEL}}$ low	t_1	10 ns (max)
$\overline{\text{STRB}}$ low to $\overline{\text{DTSTR}}$ low	t_2	5.8 ns (max)
TMS320C25 address valid to WS57C64F-12 data valid	t_3	130 ns (max)
$\overline{\text{STRB}}$ high to WS57C64F-12 output disable	t_4	40.8 ns (max)

An EPROM interface with two wait states is shown in Figure 6-14, in which the TMS27C64-20 is interfaced to the TMS320C25. The TMS27C64-20 is a CMOS 8K x 8-bit EPROM with an access time of 200 ns. The timing diagram is shown in Figure 6-15.

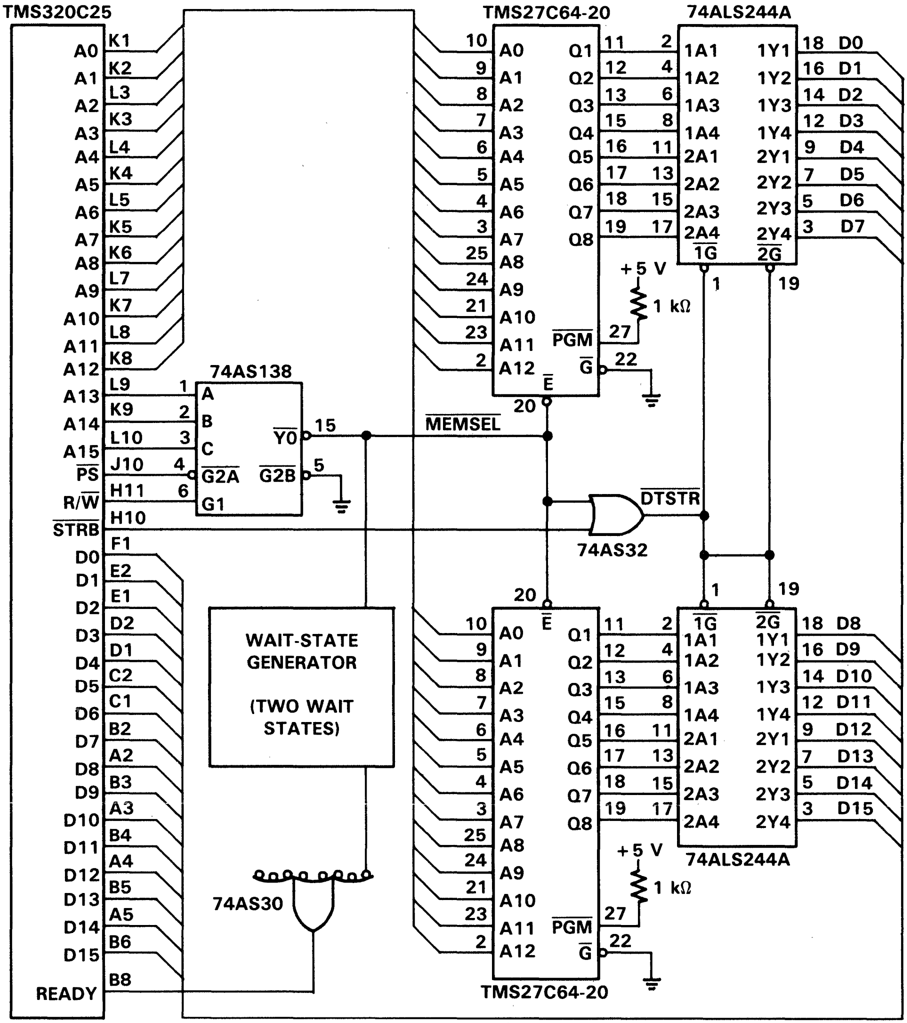


Figure 6-14. Interface of TMS27C64-20 to TMS320C25

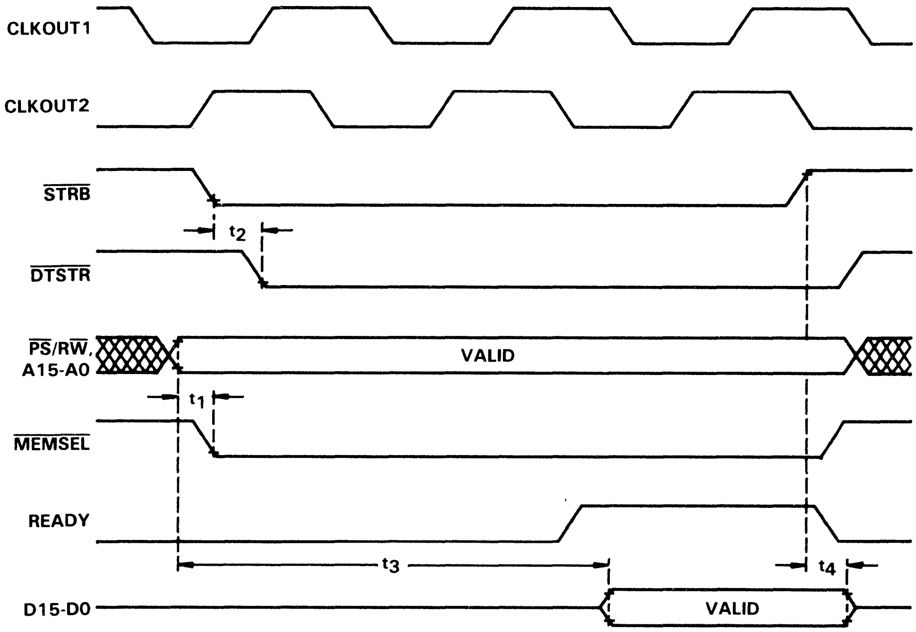


Figure 6-15. Interface Timing of TMS27C64-20 to TMS320C25

Table 6-5 summarizes the most critical timing parameters of the TMS27C64-20 interface to the TMS320C25.

Table 6-5. Timing Parameters of TMS27C64-20 Interface to TMS320C25

DESCRIPTION	SYMBOL USED IN FIGURE 6-15	VALUE
Address valid to $\overline{\text{MEMSEL}}$ low	t_1	10 ns (max)
$\overline{\text{STRB}}$ low to $\overline{\text{DTSTR}}$ low	t_2	5.8 ns (max)
TMS320C25 address valid to TMS27C64-20 data valid	t_3	220 ns (max)
$\overline{\text{STRB}}$ high to TMS27C64-20 output disable	t_4	18.8 ns (max)

For detailed information regarding EPROM interfacing, see the apaplication report, *Hardware Interfacing to the TMS320C25*.

6.2.4 Interfacing Static RAMs

Interfacing external RAM to the TMS320C2x can be useful for expanding internal data memory or implementing additional RAM program memory. Static RAM can be used as data memory to extend the TMS320C2x 544-word internal RAM. When used as program memory, object code can be downloaded into the RAM and executed. In the first case, the static RAM is mapped into the data space, while in the second case it is mapped into the program space.

In cases where RAMs of different speeds are used, separate schemes for address decoding and READY generation can be used to meet READY timing requirements in a similar manner to that used for the PROM interface described in Section 6.2.1. RAMs with similar access times may then be grouped together in one segment of memory.

The static RAM for this interface is the Cypress Semiconductor CY7C169-25 4K x 4-bit static RAM. This RAM has a 25-ns access time from address $t_{a(A)}$ and a 15-ns access time from chip enable $t_{a(CE)}$. Note that these access times are fast enough so that a wait-state generator is not required for this interface. If, however, RAMs that require wait states are used in the system, the wait-state generator described in Section 6.2.2 can be used.

The design shown in Figure 6-16 utilizes a similar approach to the one described in Sections 6.2.1 and 6.2.3; i.e., one address decoding scheme is used to generate READY, and a second address decoding scheme enables the static RAM. In this design, RAMs with no wait states are mapped at the lower half (lower 32K words) of the TMS320C25 data space. The upper half is used for memories with one or more wait states. Figure 6-17 shows the timing for memory read and write cycles.

Table 6-6 summarizes the most critical timing parameters of the CY7C169-25 interface to the TMS320C25.

Table 6-6. Timing Parameters of CY7C169-25 Interface to TMS320C25

DESCRIPTION	SYMBOL USED IN FIGURE 6-17	VALUE
Address valid to READY valid	t_1	10.8 ns (max)
$\overline{\text{STRB}}$ low to $\overline{\text{MEMSEL}}$ low	t_2	8.5 ns (max)
$\overline{\text{STRB}}$ high to $\overline{\text{MEMSEL}}$ high	t_3	7.5 ns (max)
CLKOUT1 low to TMS320C25 data bus entering the high-impedance state	t_4	15 ns (max)
$\overline{\text{MEMSEL}}$ low to CY7C169-25 driving the data bus	t_5	5 ns (min)
$\overline{\text{MEMSEL}}$ low to CY7C169-25 data valid	t_6	15 ns (max)
$\overline{\text{MEMSEL}}$ high to CY7C169-25 entering the high-impedance state	t_7	15 ns (max)
Data setup time for a write	t_8	32 ns (min)
Data hold time	t_9	7.5 ns (min)

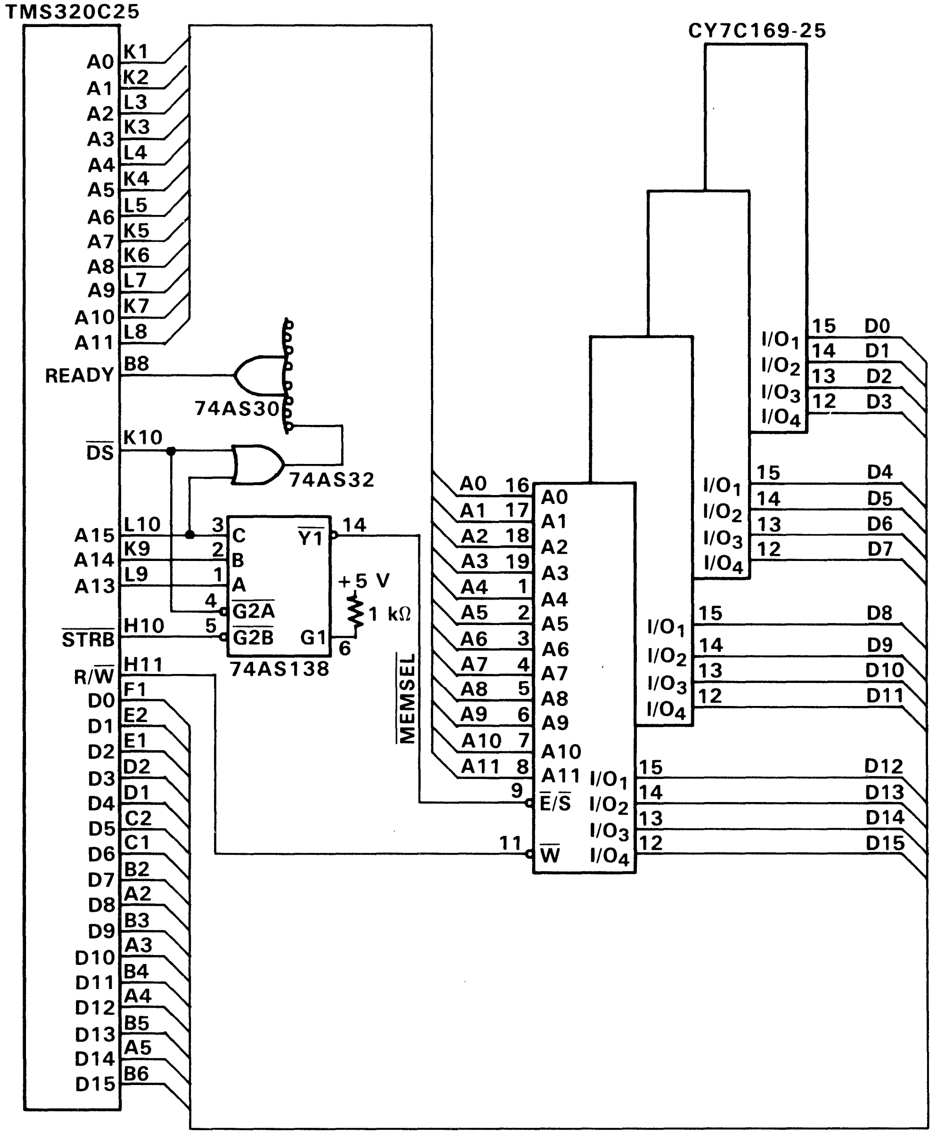


Figure 6-16. Interface of CY7C169-25 to TMS320C25

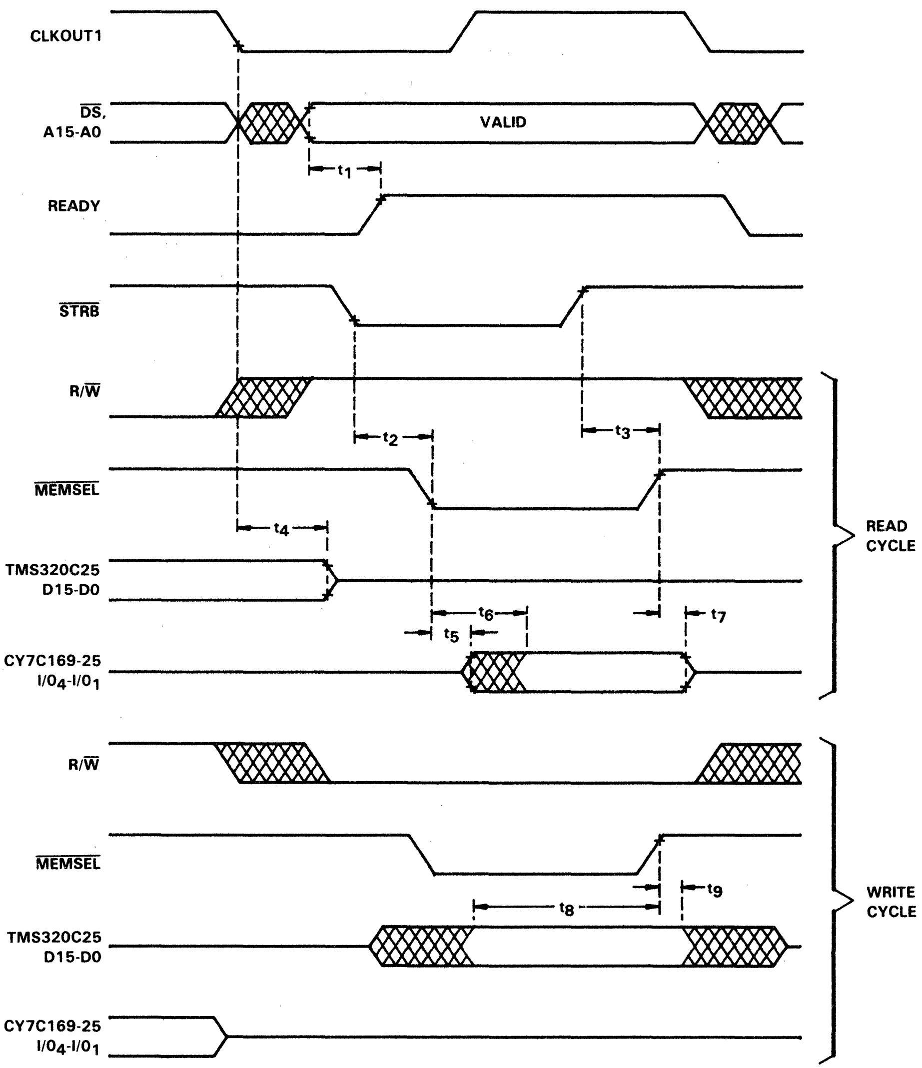


Figure 6-17. Interface Timing of CY7C169-25 to TMS320C25

6.2.5 Interface Timing Analysis

When interpreting TMS320C25 timing specifications, particularly in the area of memory interface timing, it is necessary to understand clock input and clock timing relationships shown in timing diagrams as compared with the actual data sheet specifications. If interpreted incorrectly, the specifications may suggest that interfacing to the device is more constrained than necessary. Without exception, the TMS320C25 meets every specification given in the data sheet (Appendix A). Some timings are specified more conservatively than others, due to yield distributions, etc., but at the minimum stated, each TMS320C25 is guaranteed by Texas Instruments to conform explicitly to the data sheet timings.

Clock input and internal clock timing relationships must be considered in the interpretation of output timing characteristics and requirements. At the clock input to the device, only the rising edges of the clock are used to initiate transitions on internal clocks and output signals. Thus, with an input clock of a stable frequency (regardless of duty cycle variation within specifications), extremely symmetric timing is exhibited throughout the device. A significant consequence of this is that CLKOUT1, CLKOUT2, and $\overline{\text{STRB}}$ timing skews (with respect to each other) and high and low pulse widths are integer multiples of Q (the input clock period or one-fourth of the output clock period) to within a few nanoseconds. This occurs because transitions on the output signals are initiated directly from the internal clocks (Q1-Q4), and driven through identical output buffer circuits. Since the internal clocks are very symmetric, close tracking of these outputs results. The large skews in these timings, as shown in the data sheet, are a factor of temperature and process. Since there is no variation in process and negligible variation in temperature across a single device, the skew of the outputs relative to the inputs is consistent for all outputs. Regardless of the magnitude of such skews, interfaces to the TMS320C25 can be designed independent of these skews in most cases.

Interface timings to be considered include READY, memory read, and $\overline{\text{MSC}}$ timings. With regard to READY, there are two pairs of related timings in which one timing can be met without the other one being met, with the device still guaranteed to function properly. These pairs of timings are t_d(SL-R) and t_d(C2H-R), and t_h(SL-R) and t_h(C2H-R). These front-end and back-end READY timings are specified with respect to $\overline{\text{STRB}}$ and CLKOUT2. For zero wait-state accesses, READY is referenced to $\overline{\text{STRB}}$, but for wait-state accesses, $\overline{\text{STRB}}$ remains low and another timing reference is required. Note that the actual timings for each of these parameter pairs are identical and the timings with respect to CLKOUT2 and $\overline{\text{STRB}}$ are equivalent. Therefore, if READY timing meets the requirements with respect to one of these references (but not necessarily the other), the timing requirements of the device are satisfied regardless of the actual skews between the two signals. For the purpose of interface timing, t_d(C2-S) can be assumed to be 0 ns with respect to other signals on the TMS320C25. The same is also true of t_d(C1-S) and t_w(SL); these timings can be assumed to be Q and 2Q, respectively. These relationships are accounted for in specifications and device testing.

In memory read operations, the two key timings, ta(A) and t_{su}(D)R, are related by ta(A) = t_{su}(A) + t_w(SL) - t_{su}(D)R. However, when the worst case t_w(SL) specifications are used in this equation to generate an expression for t_a(A), the result differs from the specification for t_a(A) in the data sheet. Both the specification for $t_a(A)$ and $t_{su}(D)R$ are tested explicitly on the device and guaranteed. This again justifies the assumption that $t_w(SL)$ can be assumed to be 2Q with respect to other signals on the device. This is confirmed by the fact that if $t_w(SL) = 2Q$ is used to calculate $t_a(A)$, consistency results in all of these related timings. If an interface is designed where $t_{su}(D)R$ is met but $t_a(A)$ is not met due to actual signal skews, the interface is still guaranteed to function with the TMS320C25. The same is true (but not as likely in reality) if an interface is designed where $t_a(A)$ is met but $t_{su}(D)R$ is not. Thus, even if $t_w(SL)$ is actually less than 2Q, meeting either $t_a(A)$ or $t_{su}(D)R$ is still sufficent to guarantee a valid memory cycle since both parameters are guaranteed independently.

Note that when considered in the absolute sense, timings such as $t_w(SL)$ will have some finite tolerance, although considerably less than that specified. For example, if $\overline{\text{STRB}}$ is used to generate a $\overline{\text{WE}}$ pulse for a device that specifies a minimum $\overline{\text{WE}}$ low pulse width, the data sheet specification for $\overline{\text{STRB}}$ low pulse width must be used for a worst-case design.

With regard to $\overline{\text{MSC}}$ timing, the $t_h(C2H\text{-}R)$ timing is a constraint that must be satisfied, and the $t_d(MSC)$ is a parameter more conservatively specified than many other timings. When considering these timing parameters and CLKOUT1/CLKOUT2 skews, the $\overline{\text{MSC}}$ does not meet worst-case timings for generating READY, the purpose for which the $\overline{\text{MSC}}$ signal was intended. The READY timing will be met by $\overline{\text{MSC}}$, however, regardless of when $\overline{\text{MSC}}$ goes high. This timing is explicitly guaranteed by $t_h(M\text{-}R) = 0$, even though $\overline{\text{MSC}}$ exhibits some finite skew from CLKOUT1.

6.3 Direct Memory Access (DMA)

Some advanced hardware design concepts supported by the TMS320C2x include Direct Memory Access (DMA) and global memory (see Section 6.4). Direct memory access can be used for multiprocessing by temporarily halting the execution of one or more processors to allow another processor to read from or write to the halted processor's local off-chip memory. Direct memory access to external program/data memory is performed using the $\overline{\mathrm{HOLD}}$ and $\overline{\mathrm{HOLDA}}$ signals.

The multiprocessing is typically a master-slave configuration. The master may initialize a slave by downloading a program into its program memory space and/or provide the slave with the necessary data to complete a task. In a typical TMS320C2x direct memory access scheme, the master may be a general-purpose CPU, another TMS320C2x, or perhaps even an analog-to-digital converter. A simple TMS320C2x master-slave configuration is shown in Figure 6-18. The master TMS320C2x takes complete control of the slave's external memory by asserting $\overline{\mathrm{HOLD}}$ low via its external flag (XF). This causes the slave to place its address, data, and control lines in a high-impedance state. By asserting $\overline{\mathrm{RS}}$ in conjunction with $\overline{\mathrm{HOLD}}$, the master processor can load the slave's local program memory with the necessary initialization code on reset or powerup. The two processors can be synchronized using the $\overline{\mathrm{SYNC}}$ pin to make the transfer over the memory bus faster and more efficient.

After control of the slave's buses is given up to the master processor, the slave alerts the master of this fact by asserting $\overline{\mathrm{HOLDA}}$. This signal may be tied to the master TMS320C2x's $\overline{\mathrm{BIO}}$ pin. The slave's XF pin may be used to indicate to the master when it has finished performing its task and needs to be reprogrammed or requires additional data to continue processing. In a multiple slave configuration, the priority of each slave's task may be determined by tying the slave's XF signals to the appropriate $\overline{\mathrm{INT}}$(2-0) pin on the master TMS320C2x.

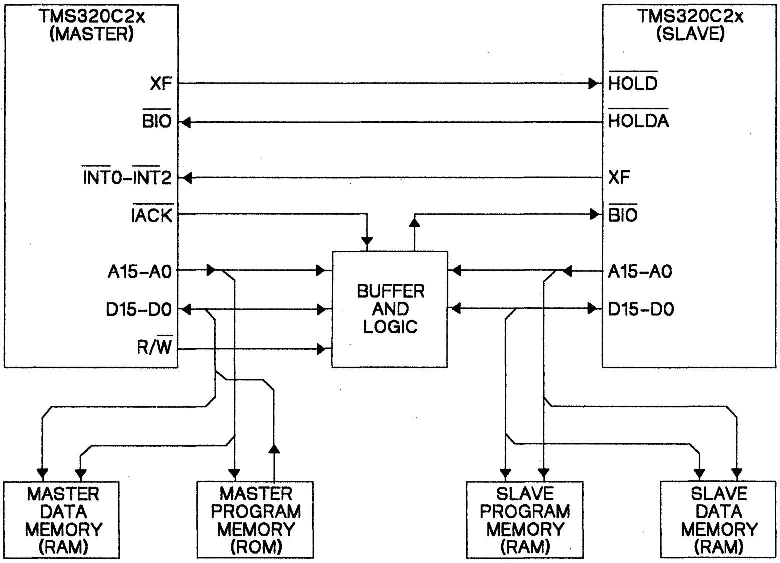


Figure 6-18. Direct Memory Access Using a Master-Slave Configuration

A PC environment presents another example of a potential direct memory access scheme where a system bus (the PC-bus) is used for data transfer. In this configuration, either the master CPU or a disk controller may place data onto the system bus, which can be downloaded into the local memory of the TMS320C2x. Here the TMS320C2x acts more like a peripheral processor with multifunction capability. In a speech application, for example, the master can load the TMS320C2x's program memory with algorithms to perform such tasks as speech analysis, synthesis, or recognition, and fill the TMS320C2x's data memory with the required speech templates. In another application example, the TMS320C2x can serve as a dedicated graphics engine. Programs can be stored in TMS320C2x program ROM or downloaded via the system bus into program RAM. Data can come from PC disk storage or provided directly by the master CPU.

Figure 6-19 depicts a direct memory access using a PC environment. In this configuration, decode and arbitration logic is used to control the direct memory access. When the address on the system bus resides in the local memory of the peripheral TMS320C2x, this logic asserts the $\overline{\text{HOLD}}$ signal of the TMS320C2x while sending the master a not-ready indication to allow wait states. After the TMS320C2x acknowledges the direct memory access by asserting $\overline{\text{HOLDA}}$, READY is asserted and the information transferred.

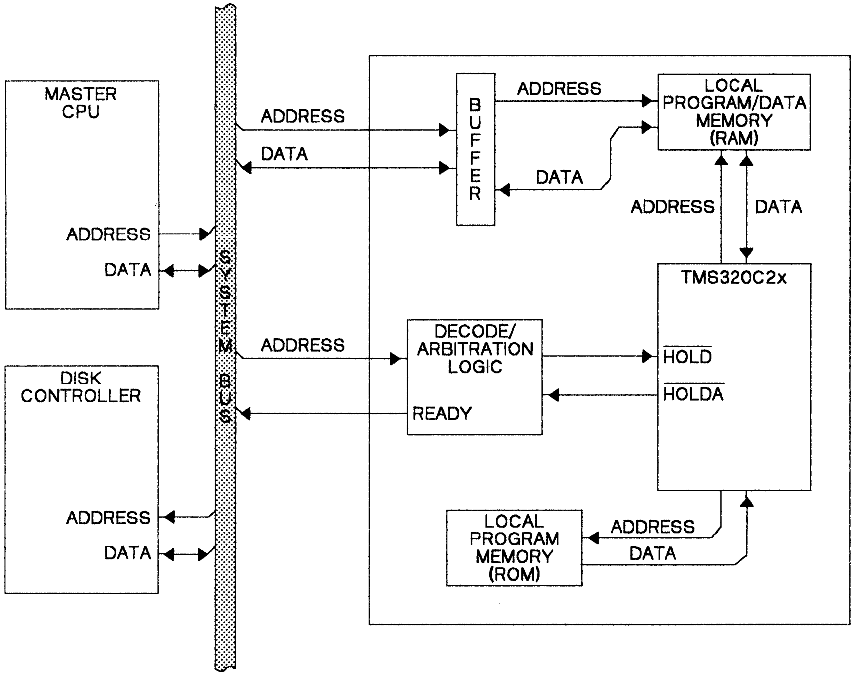


Figure 6-19. Direct Memory Access in a PC Environment

6.4 Global Memory

For multiprocessing applications, the external memory of the TMS320C2x can be divided into local and global sections. Special registers and pins included on the TMS320C2x allow multiple processors to share up to 32K words of global data memory space. This implementation facilitates efficient "shared data" multiprocessing where data is transferred between two or more processors. Unlike a direct memory access (DMA) scheme, reading or writing global memory does not require one of the processors to be halted.

Global memory can be used in various digital signal processing tasks such as filters or modems, where the algorithm being implemented may be divided into sections with a distinct processor dedicated to each section. In this multiprocessor scheme, the first and second processors may share global data memory, as well as the second and third, the third and fourth, etc. Arbitration logic is required to determine which section of the algorithm is executing and which processor has access to the global memory. With multiple processors dedicated to distinct sections of the algorithm, throughput may be increased via pipelined execution.

The size of the global memory is programmable between 256 and 32K locations in data memory by loading the global register (GREG). After global memory is defined in the GREG, the TMS320C2x asserts the $\overline{\mathrm{BR}}$ (bus request) signal before each global memory access. The processor then inserts wait states until a bus grant is given by asserting the READY line. Figure 6-20 illustrates such a global memory interface. Since the processors can be synchronized by using the $\overline{\mathrm{SYNC}}$ pin, the arbitration logic may be simplified and the address and data bus transfers more efficient (see Section 3.10.1 for information on synchronization).

The $\overline{\mathrm{SYNC}}$ pin on the TMS320C2x may also be used to synchronize several processors to allow for execution of redundant fail-safe systems. $\overline{\mathrm{SYNC}}$ permits instruction broadcasting between several processors and lock-step execution after initial synchronization.

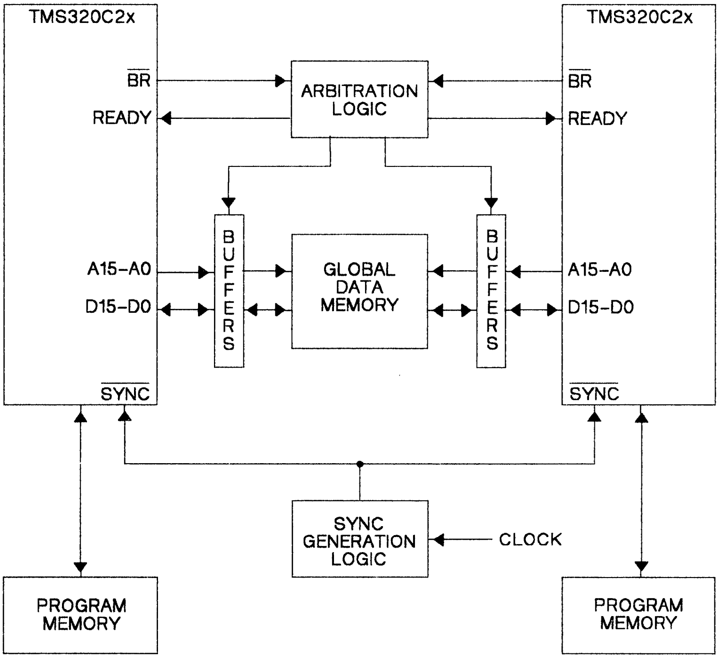


Figure 6-20. Global Memory Communication

6.5 Interfacing Peripherals

Most DSP systems implement some amount of I/O using peripherals in addition to any memory included in the system. This usually includes analog input and output, which can be performed through the parallel and serial I/O ports on the TMS320C2x.

When accessing the external parallel I/O ports, the access to the data bus is multiplexed over the same pins as for a program/data memory access. The I/O space is selected by the $\overline{\text{IS}}$ signal going active low, and the address of the port is placed on address bits A3-A0. Address bits A15-A4 are held low.

This section describes hardware interfaces to a TCM29C16 combo-codec, a TLC32040 analog interface circuit (AIC), a digital-to-analog (D/A) converter, and an analog-to-digital (A/D).

6.5.1 Combo-Codec Interface

Some areas of speech, telecommunications, and many other applications require low-cost analog-to-digital (A/D) and digital-to-analog (D/A) converters. Combo-codecs are most effective in serving DSP system data-conversion requirements. Combo-codecs are single-chip pulse-code-modulated encoders and decoders (PCM codecs), designed to perform the encoding (A/D conversion) and decoding (D/A conversion), as well as the antialiasing and smoothing filtering functions. Since combo-codecs perform these functions in a single 300-mil DIP package at low cost, they are extremely economical for providing system data-conversion functions.

Combo-codecs interface directly to the TMS320C2x by means of the serial port and provide a companded, PCM-coded digital representation of analog input samples. This PCM code is easily translated into linear form by the TMS320C2x for use in processing. The design discussed here and shown in Figure 6-21 uses a Texas Instruments TCM29C16 codec, interfaced using the serial port of the TMS320C25.

The TMS320C2x serial port provides direct synchronous communication with serial devices. The interface signals are compatible with codecs and other serial components so that minimal external hardware is required. Externally, the serial port interface is implemented using the following pins on the TMS320C25:

- DX (transmitted serial data)
- CLKX (transmit clock)
- FSX (transmit framing synchronization signal)
- DR (received serial data)
- CLKR (receive clock)
- FSR (receive framing synchronization signal)

Data on DX and DR are clocked by CLKX and CLKR, respectively. These clocks are only required during serial transfers on the TMS320C25. On the TMS32020, the clocks must be present at all times if the serial port is being used. Note that the TMS320C25 is double-buffered.

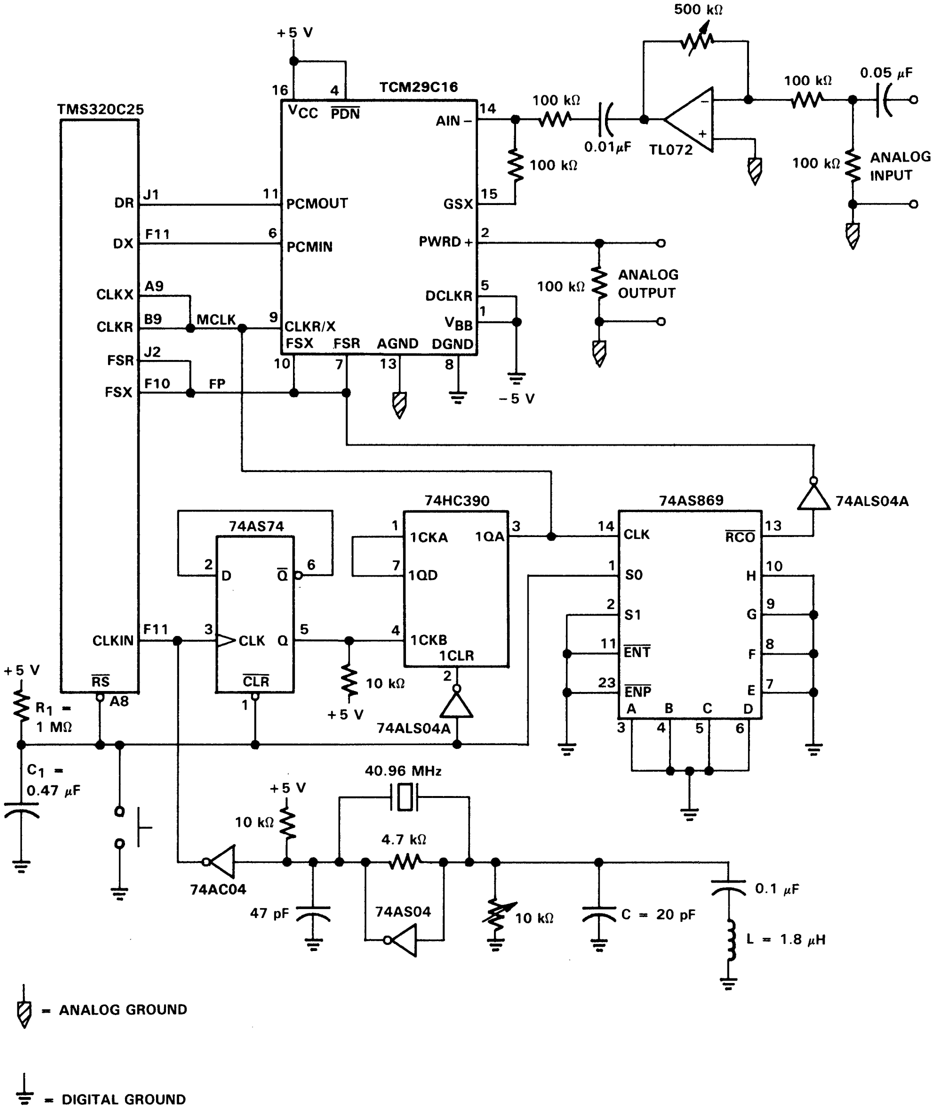


Figure 6-21. Interface of TMS320C25 to TCM29C16 Codec

Serial port transfers are initiated by framing pulses on the FSX and FSR pins for transmit and receive operations, respectively. For transmit operations, the FSX pin can be configured as an input or an output. This option is selected by the transmit mode (TXM) bit of status register ST1. In this design, FSX is assumed to be configured as an input; therefore, transmit operations are initiated by a framing pulse on the FSX pin. Upon completion of receive and transmit operations, an RINT (serial port receive interrupt) and an XINT (serial port transmit interrupt) are generated, respectively. Interface timing of the TMS320C25 to the TCM29C16 corresponds to the burst-mode serial port transmit and receive operations shown in Figure 3-28 and Figure 3-29, respectively. Continuous-mode operation using framing pulses or without framing pulses is also possible.

The format (FO) bit of status register ST1 is used to select the format (8-bit byte or 16-bit word) of the data to be received or transmitted. For interfacing the TMS320C25 to a codec, the format bit should be set to 1, formatting the data in 8-bit bytes.

The TMS320C25 interfaces directly to the codec, as shown in Figure 6-21, with no additional logic required. The PCM μ-law data generated by the codec at the PCMOUT pin is read by the TMS320C25 from the data receive (DR) pin, which is internally connected to the receive serial register (RSR). The data transmitted from the data transmit (DX) pin of the TMS320C25 is received by the PCMIN input of the codec. During the digital-to-analog conversion, this μ-law companded data must be converted back to a linear representation for use in the TMS320C25. The resulting analog waveform is lowpass-filtered by the codec's internal smoothing filter. Therefore, no additional filtering is required at the codec output (PWRO+). Software companding routines appropriate for use on the TMS320C25 are provided in the book, *Digital Signal Processing Applications with the TMS320 Family*.

The software required to initialize the TMS320C25-codec interface is provided in the Combo-Codec Interface section of the application report, *Hardware Interfacing to the TMS320C25.* This report also presents detailed information regarding codec interfacing.

A combo-codec configured in the fixed-data-rate mode requires the following external clock signals:

- A 2.048-MHz clock to be used as the master clock, and
- 8-kHz framing pulses to initialize the data transfers.

Both of these signals can be derived from the 40.96-MHz system clock with appropriate divider circuitry. This is the primary justification for selecting 40.96-MHz as the system clock frequency. The clock divider circuit consists of a 74AS74 D-type flip-flop, 74HC390 decade counter, and 74AS869 8-bit up/down counter. The hardware connections between these devices are shown in Figure 6-21.

To generate the 2.048-MHz master clock for the combo-codec, a division by 20 of the 40.96-MHz system clock is required. The 74HC390 contains on-chip two divide-by-2 and two divide-by-5 counters. Since the 74HC390 cannot be clocked with frequencies above approximately 27 MHz, a 74AS74 configured as a divide-by-2 of the 40.96-MHz clock is used.

The 74AS869 is configured to generate the 8-kHz clock pulse (the ripple carry output is 2.048 MHz/256 = 8 kHz). This pulse is used by the TMS320C25 and codec as a framing pulse to initiate data transfers.

The level of the analog input signal is controlled using the TL072 opamp connected in the inverting configuration (see Figure 6-21). Using the

500-kΩ potentiometer, the gain of this circuit can be varied from 0 to 5. The output of the 0.01-μF coupling capacitor drives the TCM29C16's internal opamp. This opamp is connected in the inverting configuration with unity gain (feedback and input impedances having the same value of 100 kΩ).

6.5.2 AIC Interface

For applications such as modems, speech, control, instrumentation, and analog interface for DSPs, a complete analog-to-digital (A/D) and digital-to-analog (D/A) input/output system on a single chip may be desired. The TLC32040 analog interface circuit (AIC) integrates on a single monolithic CMOS chip a bandpass, switched-capacitor, antialiasing-input filter, 14-bit resolution A/D and D/A converters, and a lowpass, switched-capacitor, output-reconstruction filter. The TLC32040 offers numerous combinations of master clock input frequencies and conversion/sampling rates, which can be changed via digital processor control.

Four serial port modes on the TLC32040 allow direct interface to TMS320C2x processors. When the transmit and receive sections of the AIC are operating synchronously, it can interface to two SN54299 or SN74299 serial-to-parallel shift registers. These shift registers can then interface in parallel to the TMS320C2x, other TMS320 digital signal processors, or to external FIFO circuitry. Output data pulses are emitted to inform the processor that data transmission is complete or to allow the DSP to differentiate between two transmitted bytes. A flexible control scheme is provided so that the functions of the AIC can be selected and adjusted coincidentally with signal processing via software control. Refer to the TLC32040 data sheet for detailed information on timing and device functions.

The AIC is easily interfaced to the TMS320C2x serial ports, as shown in Figure 6-22. The TMS320C2x can communicate with the AIC either synchronously or asynchronously depending on the information in the control register. The operating sequence for synchronous communication with the TMS320C2x, shown in Figure 6-23, is as follows:

1) The $\overline{\text{FSX}}$ or $\overline{\text{FSR}}$ pin is brought low.
2) One 16-bit word is transmitted or one 16-bit byte is received.
3) The $\overline{\text{FSX}}$ or $\overline{\text{FSR}}$ pin is brought high.
4) The $\overline{\text{EODX}}$ or $\overline{\text{EODR}}$ pin emits a low-going pulse.

For asynchronous communication, the operating sequence is similar, but $\overline{\text{FSX}}$ and $\overline{\text{FSR}}$ do not occur at the same time (see Figure 6-24). For proper operation, the TXM bit in the TMS320C2x control register should be set to 0 so that the $\overline{\text{FSX}}$ pin of the TMS320C2x is configured as an input, the format (FO) status bit is set to 0, and the AIC WORD/$\overline{\text{BYTE}}$ pin is at logic high. After each receive and transmit operation, the TMS320C2x asserts an internal receive (RINT) and transmit (XINT) interrupt, which may be used to control program execution.

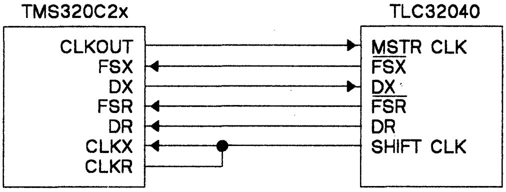


Figure 6-22. Interface of TLC32040 to TMS320C2x

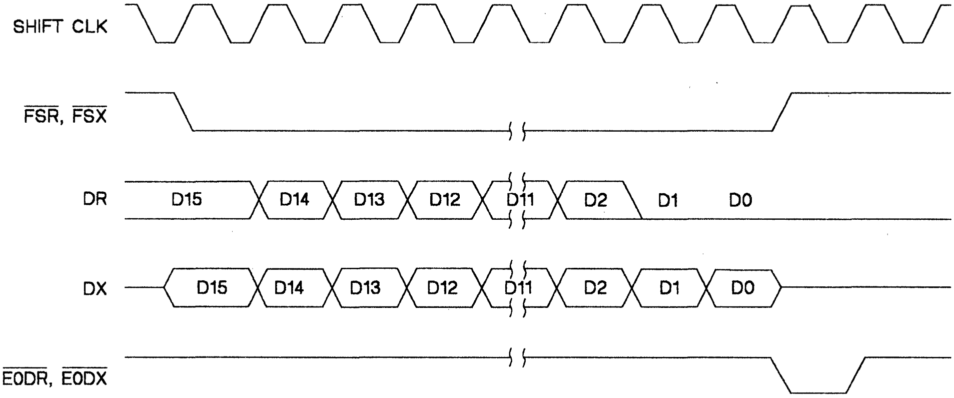


Figure 6-23. Synchronous Timing of TLC32040 to TMS320C2x

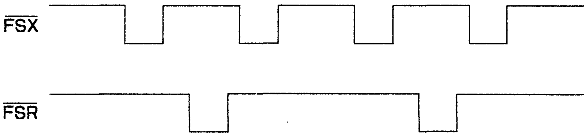
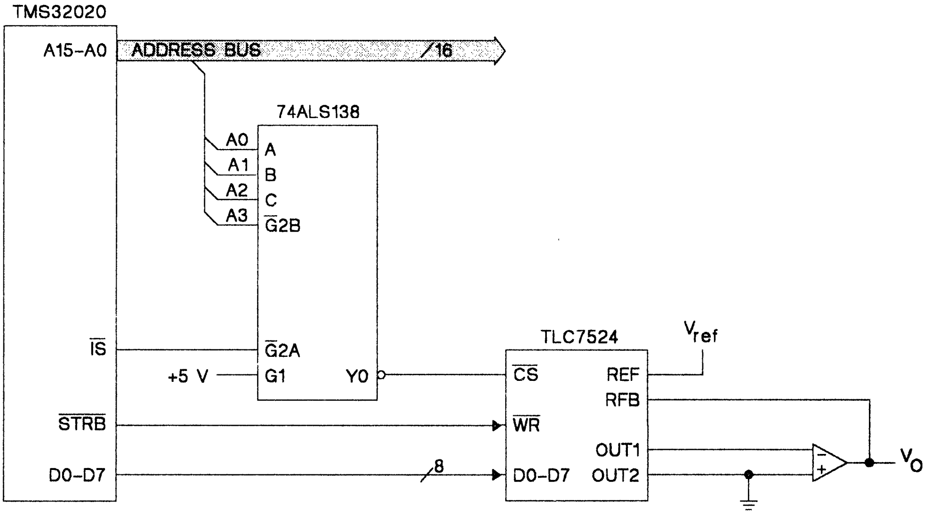


Figure 6-24. Asynchronous Timing of TLC32040 to TMS320C2x

For further information regarding the AIC interface, see page 11-196 of *Linear and Interface Circuits Applications, Volume 3: Peripheral Drivers, Data Acquisition Systems, Hall-Effect Devices*, published by Texas Instruments.

6.5.3 Digital-to-Analog (D/A) Interface

The high-speed operation of the internal logic circuitry of the TLC7524 8-bit digital-to-analog (D/A) converter allows an interface to the TMS32020 with a minimum of external circuitry required. Figure 6-25 shows the interface circuitry, which consists of one SN74ALS138 3-to-8-line decoder used to decode the address of the peripheral.



$V_O = -V_{ref} \frac{D}{256}$ where D = digital input

Figure 6-25. Interface of TLC7524 to TMS32020

When the TMS32020 executes an OUT instruction (see Figure 6-28), the peripheral address is placed on the address bus and the $\overline{IS}$ line goes low, indicating that the address on the bus corresponds to an I/O port and not external data or program memory. A low level at $\overline{IS}$ enables the 74ALS138 decoder, and the Y-output, corresponding to the address on the bus, is brought low. When the Y-output is brought low, the TLC7524 is enabled and the data appearing on the data bus is latched into the D/A converter by $\overline{STRB}$. The controlling software for the D/A interface is given on page 11-204 of *Linear and Interface Circuits Applications, Volume 3: Peripheral Drivers, Data Acquisition Systems, Hall-Effect Devices*, published by Texas Instruments.

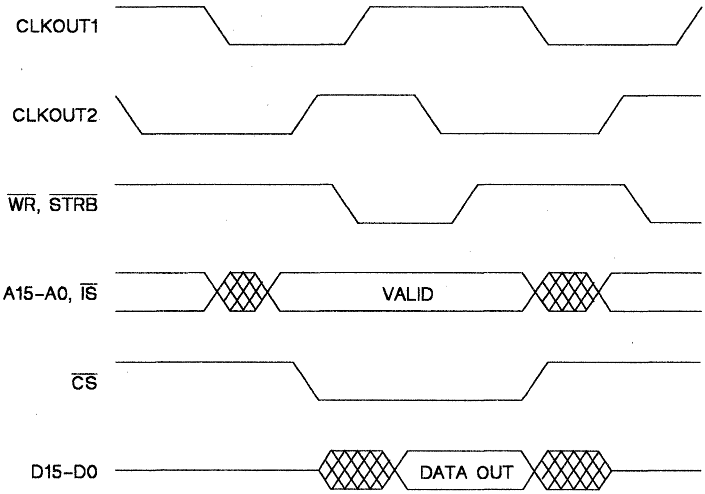


Figure 6-26. Interface Timing of TLC7524 to TMS32020

6.5.4 Analog-to-Digital (A/D) Interface

The TMS320C2x can be interfaced to 8-bit A/D converters, such as the TLC0820. However, because the control circuitry of the TLC0820 operates much more slowly than the TMS320C2x, it cannot be directly interfaced. In the TLC0820 to TMS32020 interface design shown in Figure 6-27, the following logic devices are used in the interface circuit:

- A 3-line to 8-line decoder (SN74ALS138)
- A quad 2-input NAND gate (SN74LS00)
- A hex inverter (SN74LS04)
- A quad 2-input OR gate (SN74LS32)
- A quad D-type flip-flop (SN74LS175)

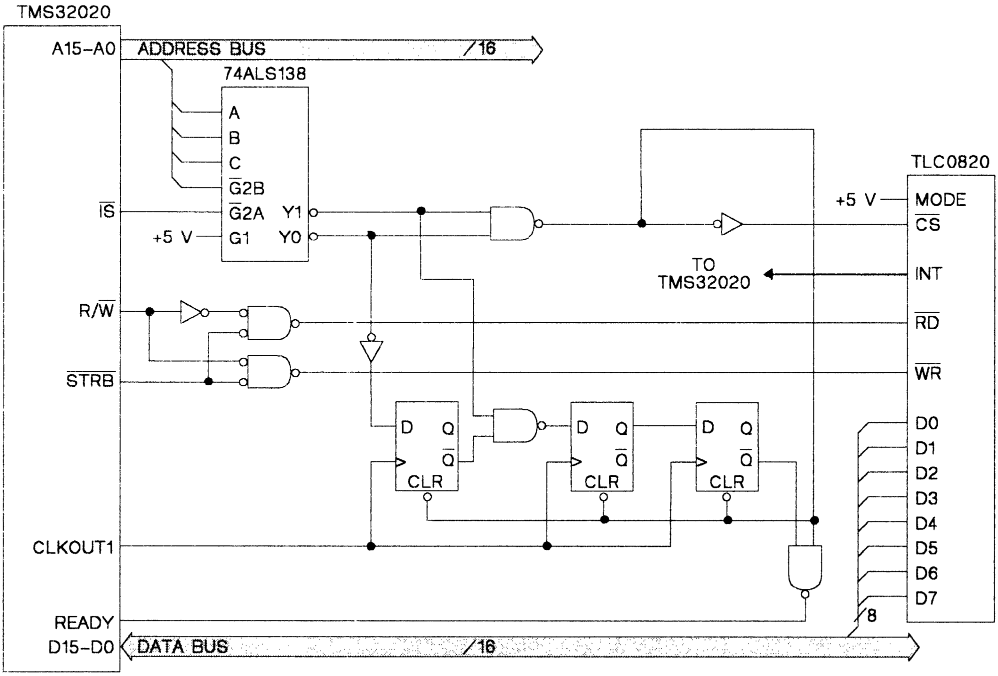


Figure 6-27. Interface of TLC0820 to TMS32020

The 74LS138 decodes the addresses assigned to the TLC0820. One of the addresses is used when performing a write operation; the other is used for a read operation. The two different addresses are necessary to ensure that the correct number of wait states is provided for the write and read operations. The controlling software for the A/D interface is given on page 11-206 of *Lin-ear and Interface Circuits Applications, Volume 3: Peripheral Drivers, Data Acquisition Systems, Hall-Effect Devices*, published by Texas Instruments.

With the TMS32020 running at 20 MHz and the TLC0820 configured as slow memory, three wait states are necessary to provide a write pulse of sufficient length. After conversion has begun (with the rising edge of the $\overline{WR}$ signal), the TMS32020 must wait at least 600 ns before the conversion result can be read. Sufficient delay should be provided in software. To read the conversion result, sufficient wait states must be provided to allow for the data access time (320 ns minimum) of the TLC0820. As shown in the IN instruction timing diagram of Figure 6-28, two wait states are provided when accessing port 1.

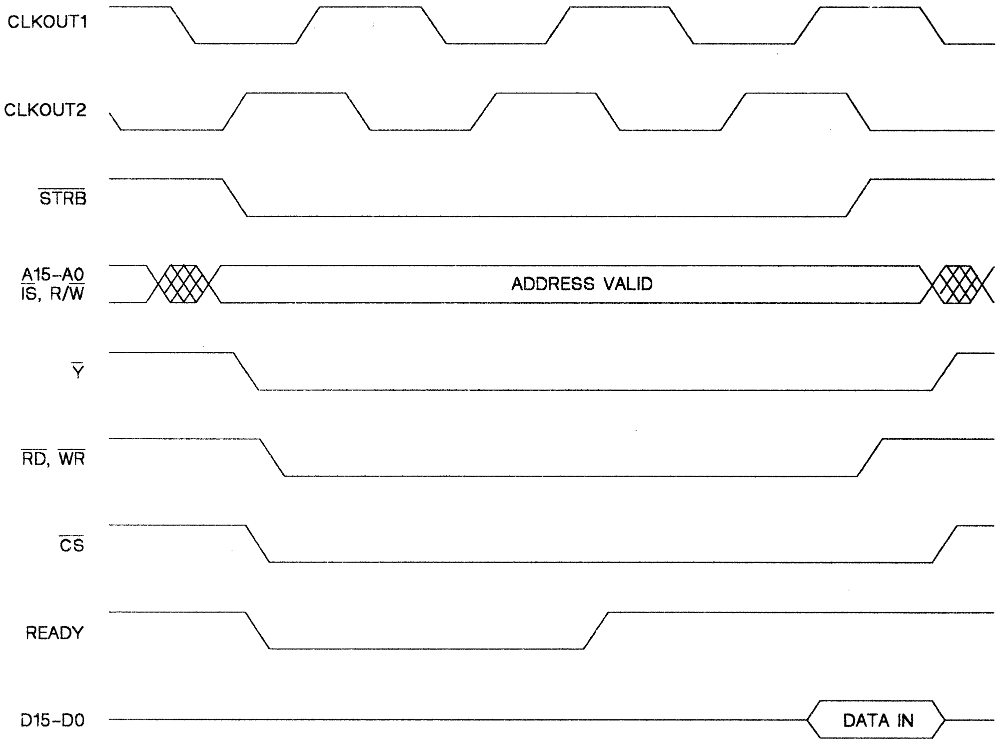


Figure 6-28. Interface Timing of TLC0820 to TMS32020

6.5.5 I/O Ports

I/O design on the TMS320C2x is treated the same way as memory. The I/O address space is distinguished from the local program/data memory space by the $\overline{IS}$ signal. $\overline{IS}$ goes low at the beginning of the memory cycle. All other control signals and timing parameters are the same as those for the program/data external memory interface.

The TMS320C2x software instructions can access 16 input and 16 output ports. The four least significant bits of the address bus specify the particular port being accessed. A pair of 74AS138s can be used to fully decode these address bits (see Figure 6-29).

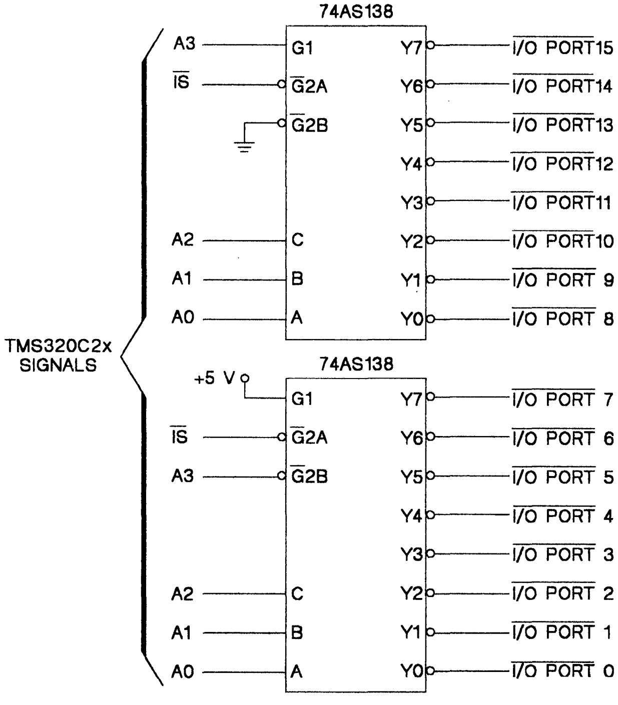


Figure 6-29. I/O Port Addressing

A simple interface between two processors can be implemented using up to 16 bidirectional I/O ports connected to the TMS320C2x. An interprocessor communication path can be formed by memory-mapping peripherals to the I/O ports of the TMS320C2x. In this manner, the TMS320C2x can connect to parallel A/Ds, registers, FIFOs, two-port memories, or other peripheral devices. In a multiprocessing scheme, intelligent peripherals can be memory-mapped into the I/O ports. Here the TMS320C2x can communicate with UARTs, general-purpose microprocessors, disk controllers, video controllers, or other peripheral processors.

Using an 8-bit general-purpose microprocessor, such as TI's TMS70C42, for a keyboard interface is an example of a TMS320C2x I/O port multiprocessing scheme, as shown in Figure 6-30. The TMS70C42 may be mapped into the TMS320C2x I/O space using latches to store the transferred data. In a single or multiple I/O port multiprocessing configuration, the four LSBs of the address bus are decoded to determine which of the 16 I/O ports on the TMS320C2x is being accessed. The TMS320C2x selects the I/O space ($\overline{IS}$) for its external bus and reads/writes data using the IN/OUT instructions.

Processor-controlled signals between the TMS320C2x and the peripheral device indicate when data is available to be read. This interprocessor communication is facilitated by using the input and output pins of the TMS70C42 (or other peripheral processor). In an I/O multiprocessing configuration, the I/O port address space is limited, and data transfers are relatively slow compared to a direct memory access or global memory configuration.

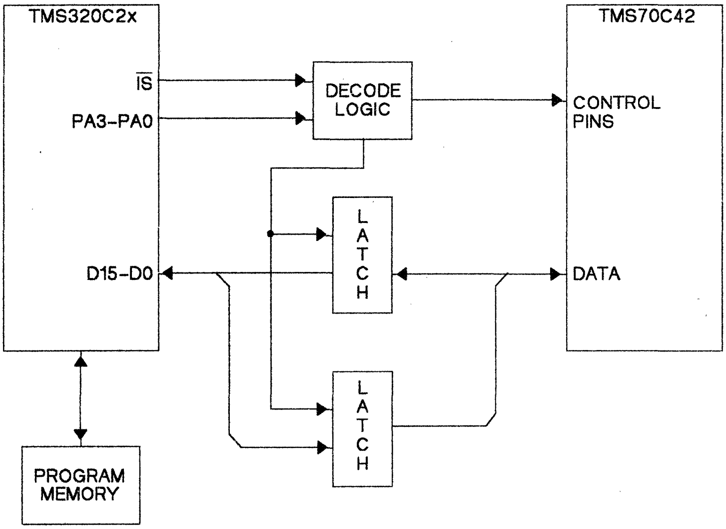


Figure 6-30. I/O Port Processor-to-Processor Communication

6.6 System Applications

The TMS320C2x is used in a wide variety of systems. Several applications in the areas of telecommunications, graphics and image processing, high-speed control, instrumentation, and numeric processing are described in the following paragraphs to illustrate basic approaches to system design using the TMS320C2x.

6.6.1 Echo Cancellation

Digital signal processing is extensively used in telecommunications applications. In echo cancellation, an adaptive FIR filter performs the modelling routine and signal modifications required to adaptively cancel the echo caused by impedance mismatches in telephone transmission lines. The TMS320C25's large on-chip RAM of 544 words and on-chip ROM of 4K words allow it to execute a 256-tap adaptive filter (32-ms echo cancellation) without external data or program memory. Figure 6-31 shows a common configuration for an echo canceller that uses a TCM29C16 codec interface.

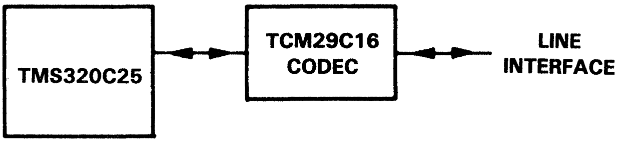


Figure 6-31. Echo Canceller

6.6.2 High-Speed Modem

In high-speed modems, a signal processor is used to perform functions such as modulation/demodulation, adaptive equalization, and echo cancellation. The TMS320C2x large memory space allows it to support multiple standards such as Bell 103, Bell 212A, V.22 bis, V.29, V.32, and V.33, as well as proprietary algorithms. The modem shown in Figure 6-32 consists of the host interface, controller, DSP, and analog front-end.

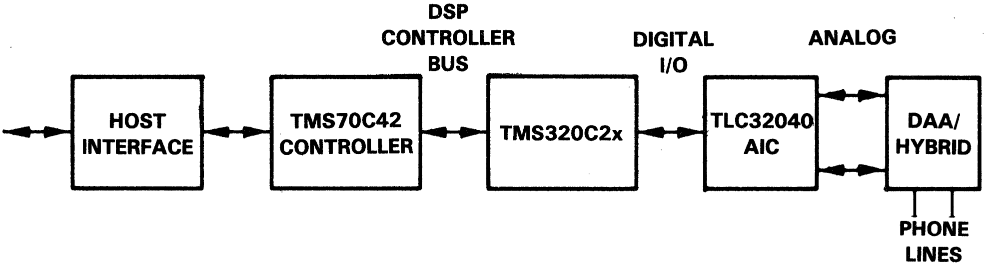


Figure 6-32. High-Speed Modem

6.6.3 Voice Coding

Voice coding techniques, such as full-duplex 32-kbps adaptive differential pulse-code modulation (CCITT G.721), 16-kbps subband coding, and linear predictive coding, are frequently used in voice transmission and storage. The speed of the TMS320C2x in performing arithmetic computations, normalization, and bit manipulation enables it to implement these functions usually internally (i.e., with no external devices). Figure 6-33 shows a voice coding system consisting of a TMS320C2x DSP, TCM29C16 codec or TLC32040 AIC, and optional external memory.

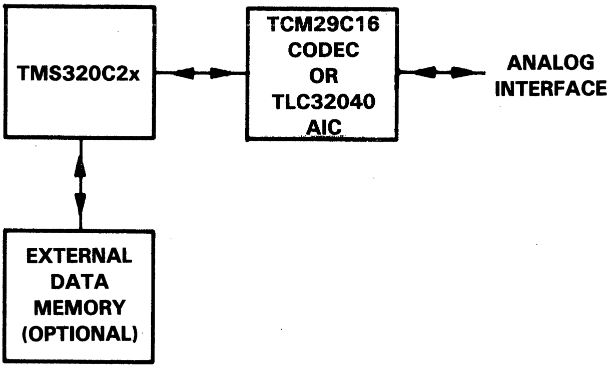


Figure 6-33. Voice Coding System

6.6.4 Graphics and Image Processing

In graphics and image processing applications, a signal processor's ability to interface with a host processor is important. The TMS320C2x multiprocessor interface enables it to be used in a variety of host/coprocessor configurations (see Figure 6-34 for an example of a graphics system configuration). Graphics and image processing applications can use the large directly addressable external data memory space and global memory capability to allow graphical images in memory to be shared with a host processor, thus minimizing data transfers. Indexed indirect addressing modes on the TMS320C2x allow matrices to be processed row-by-row when matrix multiplication is performed for 3-D image rotation, translation, and scaling.

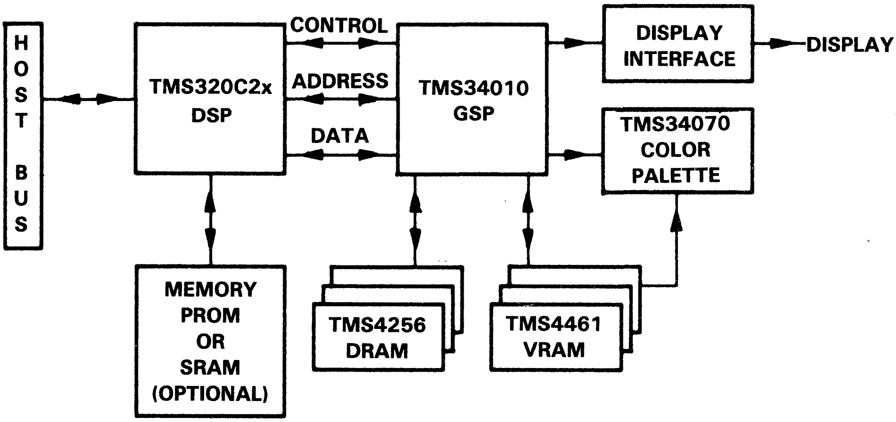


Figure 6-34. Graphics System

6.6.5 High-Speed Control

High-speed control applications, such as robotics, use the TMS320C2x general-purpose features for bit manipulation, logical operations, timing synchronization, and fast data transfers (10 million 16-bit words per second). In addition to the numeric-intensive control functions typical of robotic applications, the TMS320C2x provides a host interface whereby a robot can communicate to a central host processor (see Figure 6-35). The TMS320C2x is also used in the closed-loop systems of disk drives for signal conditioning, filtering, high-speed computing, and multichannel multiplexing.

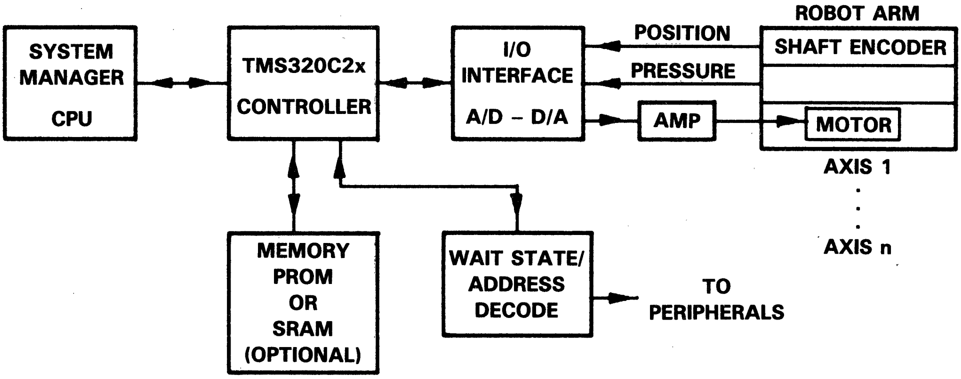


Figure 6-35. Robot Axis Controller Subsystem

6.6.6 Instrumentation and Numeric Processing

Instrumentation, such as spectrum analyzers, require a large data memory space and a processor such as the TMS320C2x capable of performing long-length FFTs and generating high-precision functions with minimal external hardware. Figure 6-36 shows an example of an instrumentation system. Numeric processing applications benefit from the high throughput, multiprocessing, and data memory expansion capabilities of the TMS320C2x.

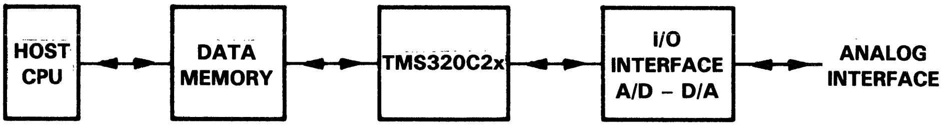


Figure 6-36. Instrumentation System

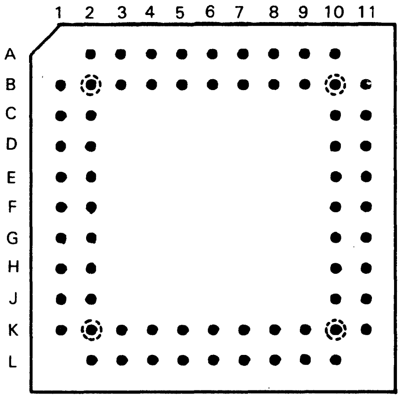
TMS320 SECOND-GENERATION DIGITAL SIGNAL PROCESSORS

MAY 1987 – REVISED SEPTEMBER 1987

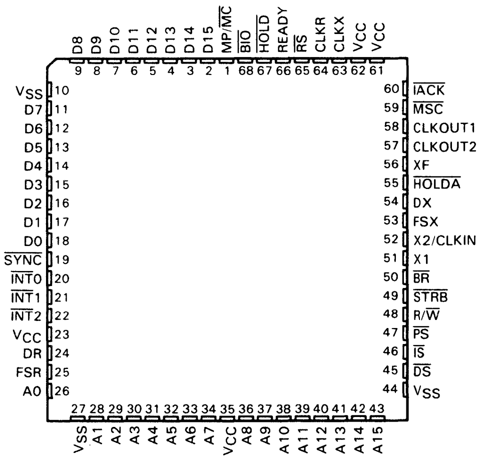
- 100-ns Instruction Cycle Time
- 544 Words of Programmable On-Chip Data RAM
- 4K Words of On-Chip Program ROM
- 128K Words of Data/Program Space
- 32-Bit ALU/Accumulator
- 16 × 16-Bit Multiplier with a 32-Bit Product
- Block Moves for Data/Program Management
- Repeat Instructions for Efficient Use of Program Space
- Serial Port for Direct Codec Interface
- Synchronization Input for Synchronous Multiprocessor Configurations
- Wait States for Communication to Slow Off-Chip Memories/Peripherals
- On-Chip Timer for Control Operations
- Single 5-V Supply
- Packaging: 68-Pin PGA and 68-Pin PLCC
- Commercial and Military Versions Available
- NMOS Technology:
  - TMS32020 . . . . . 200-ns cycle time
- CMOS Technology:
  - TMS320C25 . . . . 100-ns cycle time

This data sheet provides complete design documentation for the second-generation devices of the TMS320 family. This facilitates the selection of the devices best suited for user applications by providing all specifications and special features for each TMS320 member. This data sheet is divided into four major sections: architecture, electrical specifications (NMOS and CMOS), timing diagrams, and mechanical data. In each of these sections, generic information is presented first, followed by specific device information. An index is provided for quick reference to specific information about a device.

68-PIN GB
PIN GRID ARRAY CERAMIC PACKAGE†
(TOP VIEW)



68-PIN FN
PLASTIC LEADED CHIP CARRIER PACKAGE†
(TOP VIEW)



†See Pin Assignments Table and Pin Nomenclature Table (Page 2) for location and description of all pins.

description

The TMS320 family of 16/32-bit single-chip digital signal processors combines the flexibility of a high-speed controller with the numerical capability of an array processor, thereby offering an inexpensive alternative to multichip bit-slice processors. The highly paralleled architecture and efficient instruction set

TEXAS INSTRUMENTS
POST OFFICE BOX 1443 • HOUSTON, TEXAS 77001

PGA/PLCC PIN ASSIGNMENTS

FUNCTION	PIN	FUNCTION	PIN	FUNCTION	PIN	FUNCTION	PIN	FUNCTION	PIN	FUNCTION	PIN
A0	K1/26	A12	K8/40	D2	E1/16	D14	A5/3	INT2	H1/22	V_{CC}	H2/23
A1	K2/28	A13	L9/41	D3	D2/15	D15	B6/2	$\overline{IS}$	J11/46	V_{CC}	L6/35
A2	L3/29	A14	K9/42	D4	D1/14	DR	J1/24	MP/$\overline{MC}$†	A6/1	V_{SS}	B1/10
A3	K3/30	A15	L10/43	D5	C2/13	$\overline{DS}$	K10/45	$\overline{MSC}$	C10/59	V_{SS}	K11/44
A4	L4/31	$\overline{BIO}$	B7/68	D6	C1/12	DX	E11/54	$\overline{PS}$	J10/47	V_{SS}	L2/27
A5	K4/32	$\overline{BR}$	G11/50	D7	B2/11	FSR	J2/25	READY	B8/66	XF	D11/56
A6	L5/33	CLKOUT1	C11/58	D8	A2/9	FSX	F10/53	$\overline{RS}$	A8/65	X1	G10/51
A7	K5/34	CLKOUT2	D10/57	D9	B3/8	$\overline{HOLD}$	A7/67	R/$\overline{W}$	H11/48	X2/CLKIN	F11/52
A8	K6/36	CLKR	B9/64	D10	A3/7	$\overline{HOLDA}$	E10/55	$\overline{STRB}$	H10/49		
A9	L7/37	CLKX	A9/63	D11	B4/6	$\overline{IACK}$	B11/60	$\overline{SYNC}$	F2/19		
A10	K7/38	D0	F1/18	D12	A4/5	$\overline{INT0}$	G1/20	V_{CC}	A10/61		
A11	L8/39	D1	E2/17	D13	B5/4	$\overline{INT1}$	G2/21	V_{CC}	B10/62		

†On the TMS32020, MP/$\overline{MC}$ must be connected to V_{CC}.

PIN NOMENCLATURE

SIGNALS	I/O/Z†	DEFINITION
V_{CC}	I	5-V supply pins
V_{SS}	I	Ground pins
X1	O	Output from internal oscillator for crystal
X2/CLKIN	I	Input to internal oscillator from crystal or external clock
CLKOUT1	O	Master clock output (crystal or CLKIN frequency/4)
CLKOUT2	O	A second clock output signal
D15-D0	I/O/Z	16-bit data bus D15 (MSB) through D0 (LSB). Multiplexed between program, data, and I/O spaces.
A15-A0	O/Z	16-bit address bus A15 (MSB) through A0 (LSB)
$\overline{PS}$, $\overline{DS}$, $\overline{IS}$	O/Z	Program, data, and I/O space select signals
R/$\overline{W}$	O/Z	Read/write signal
$\overline{STRB}$	O/Z	Strobe signal
$\overline{RS}$	I	Reset input
$\overline{INT2}$-$\overline{INT0}$	I	External user interrupt inputs
MP/$\overline{MC}$	I	Microprocessor/microcomputer mode select pin
$\overline{MSC}$	O	Microstate complete signal
$\overline{IACK}$	O	Interrupt acknowledge signal
READY	I	Data ready input. Asserted by external logic when using slower devices to indicate that the current bus transaction is complete.
$\overline{BR}$	O	Bus request signal. Asserted when the TMS320C25 requires access to an external global data memory space.
XF	O	External flag output (latched software-programmable signal)
$\overline{HOLD}$	I	Hold input. When asserted, TMS320C25 goes into an idle mode and places the data, address, and control lines in the high impedance state.
$\overline{HOLDA}$	O	Hold acknowledge signal
$\overline{SYNC}$	I	Synchronization input
$\overline{BIO}$	I	Branch control input. Polled by BIOZ instruction.
DR	I	Serial data receive input
CLKR	I	Clock for receive input for serial port
FSR	I	Frame synchronization pulse for receive input
DX	O/Z	Serial data transmit output
CLKX	I	Clock for transmit output for serial port
FSX	I/O/Z	Frame synchronization pulse for transmit. Configurable as either an input or an output.

†I/O/Z denotes input/output/high-impedance state.

provide speed and flexibility to produce a MOS microprocessor family capable of executing 10 MIPS (million instructions per second). The TMS320 family optimizes speed by implementing functions in hardware that other processors implement through microcode or software. This hardware-intensive approach provides the design engineer with processing power previously unavailable on a single chip.

The TMS320 family consists of three generations of digital signal processors. The first generation contains the TMS32010 and its spinoffs. The second generation includes the TMS32020 and TMS320C25, which are described in this data sheet. The TMS320C30 is the third-generation processor, designed for higher performance. Many features are common among the TMS320 processors. Specific features are added in each processor to provide different cost/performance tradeoffs. Software compatibility is maintained throughout the family to protect the user's investment in architecture. Each processor has software and hardware tools to facilitate rapid design.

introduction

The TMS32010, the first NMOS digital signal processor in the TMS320 family, was introduced in 1983. Its powerful instruction set, inherent flexibility, high-speed number-crunching capabilities, and innovative architecture have made this high-performance, cost-effective processor the ideal solution to many telecommunications, computer, commercial, industrial, and military applications. Since that time, the TMS320C10, a low-power CMOS version of the industry-standard TMS32010, and other spinoff devices have been added to the first generation of the TMS320 family.

The second generation of the TMS320 family (referred to as TMS320C2x) includes two members, the TMS32020 and the TMS320C25. The architecture of these devices is based upon that of the TMS32010.

The **TMS32020**, processed in NMOS technology, is source-code compatible with the TMS32010 and in many applications is capable of two times the throughput of the first-generation devices. Its enhanced instruction set (109 instructions), large on-chip data memory (544 words), large memory spaces, on-chip serial port, and hardware timer make the TMS32020 a powerful addition to the TMS320 family.

The **TMS320C25** is the newest member of the TMS320 second generation. It is processed in CMOS technology, is capable of an instruction cycle time of 100 ns, and is pin-for-pin and object-code compatible with the TMS32020. The TMS320C25's enhanced feature set greatly increases the functionality of the device over the TMS32020. Enhancements include 24 additional instructions (133 total), eight auxiliary registers, an eight-level hardware stack, 4K words of on-chip program ROM, a bit-reversed indexed-addressing mode, and the low-power dissipation inherent to the CMOS process.

Table 1 provides an overview of the second-generation TMS320 processors with comparisons of memory, I/O, cycle timing, power, package type, technology, and military support. For specific availability, contact the nearest TI sales office.

TABLE 1. TMS320 SECOND-GENERATION DEVICE OVERVIEW

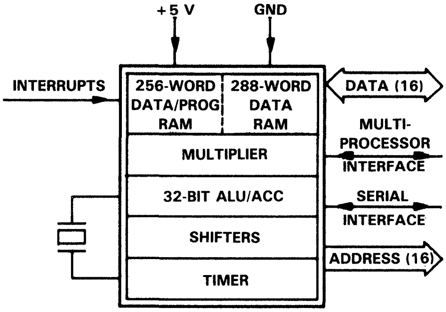
DEVICE	MEMORY				I/O†			TIMER	CYCLE TIME (ns)	TYP POWER (mW)	PACKAGE TYPE	
	ON-CHIP		OFF-CHIP									
	RAM	ROM	PROG	DATA	SER	PAR	DMA				PGA	PLCC
TMS32020‡ (NMOS)	544	–	64K	64K	YES	16 × 16	YES	YES	200	1250	68	–
TMS320C25‡ (CMOS)	544	4K	64K	64K	YES	16 × 16	CON	YES	100	500	68	68§

†SER = serial; PAR = parallel; DMA = direct memory access; CON = concurrent DMA.
‡Military version planned; contact nearest TI sales office for availability.
§PLCC version planned; contact nearest TI sales office for availability.

TMS320 SECOND-GENERATION DEVICES

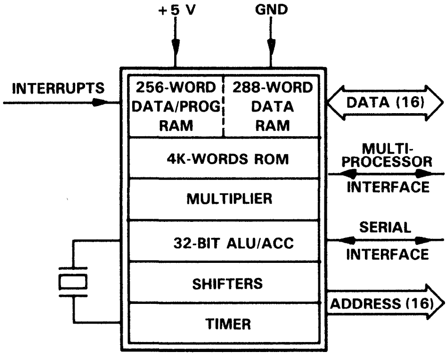
Key Features: TMS32020

- 200-ns Instruction Cycle Time
- 544 Words of On-Chip Data RAM
- 128K Words of Total Data/Program Memory Space
- Wait States for Communication to Slower Off-Chip Memories
- Object Code Compatible with the TMS32010
- Single-Cycle Multiply/Accumulate Instructions
- Repeat Instructions
- Global Data Memory Interface
- Block Moves for Data/Program Management
- Five Auxiliary Registers with Dedicated Arithmetic Unit
- Serial Port for Multiprocessing or Interfacing to Codecs, Serial Analog-to-Digital Converters, etc.
- On-Chip Clock Generator
- Single 5-V Supply
- NMOS Technology
- 68-Pin Grid Array (PGA) Package



Key Features: TMS320C25

- 100-ns Instruction Cycle Time
- 4K Words of On-Chip Program ROM
- 544 Words of On-Chip RAM
- 128K Words of Total Program/Data Memory Space
- Wait States for Communications to Slower Off-Chip Memories
- Object-Code Compatible with the TMS32020
- 24 Additional Instructions to Support Adaptive Filtering, FFTs, and Extended-Precision Arithmetic
- Block Moves for Data/Program Management
- Single-Cycle Multiply/Accumulate Instructions
- Eight Auxiliary Registers with Dedicated Arithmetic Unit
- Bit-Reversed Indexed-Addressing Mode for Radix-2 FFTs
- Double-Buffered Serial Port
- On-Chip Clock Generator
- Single 5-V Supply
- CMOS Technology
- 68-Pin Grid Array (PGA) Package and 68-Lead Chip Carrier (PLCC) Package



architecture

The TMS320 family utilizes a modified Harvard architecture for speed and flexibility. In a strict Harvard architecture, program and data memory lie in two separate spaces, permitting a full overlap of instruction fetch and execution. The TMS320 family's modification of the Harvard architecture allows transfers between program and data spaces, thereby increasing the flexibility of the device. This modification permits coefficients stored in program memory to be read into the RAM, eliminating the need for a separate coefficient ROM. It also makes available immediate instructions and subroutines based on computed values.

Increased throughput on the TMS320C2x devices for many DSP applications is accomplished by means of single-cycle multiply/accumulate instructions with a data move option, up to eight auxiliary registers with a dedicated arithmetic unit, and faster I/O necessary for data-intensive signal processing.

The architectural design of the TMS320C2x emphasizes overall speed, communication, and flexibility in processor configuration. Control signals and instructions provide floating-point support, block-memory transfers, communication to slower off-chip devices, and multiprocessing implementations.

32-bit ALU/accumulator

The 32-bit Arithmetic Logic Unit (ALU) and accumulator perform a wide range of arithmetic and logical instructions, the majority of which execute in a single clock cycle. The ALU executes a variety of branch instructions dependent on the status of the ALU or a single bit in a word. These instructions provide the following capabilities:

- Branch to an address specified by the accumulator
- Normalize fixed-point numbers contained in the accumulator
- Test a specified bit of a word in data memory.

One input to the ALU is always provided from the accumulator, and the other input may be provided from the Product Register (PR) of the multiplier or the input scaling shifter which has fetched data from the RAM on the data bus. After the ALU has performed the arithmetic or logical operations, the result is stored in the accumulator.

The 32-bit accumulator is split into two 16-bit segments for storage in data memory. Additional shifters at the output of the accumulator perform shifts while the data is being transferred to the data bus for storage. The contents of the accumulator remain unchanged.

scaling shifter

The TMS320C2x scaling shifter has a 16-bit input connected to the data bus and a 32-bit output connected to the ALU. The scaling shifter produces a left shift of 0 to 16 bits on the input data, as programmed in the instruction. The LSBs of the output are filled with zeroes, and the MSBs may be either filled with zeroes or sign-extended, depending upon the status programmed into the SXM (sign-extension mode) bit of status register ST1.

16 × 16-bit parallel multiplier

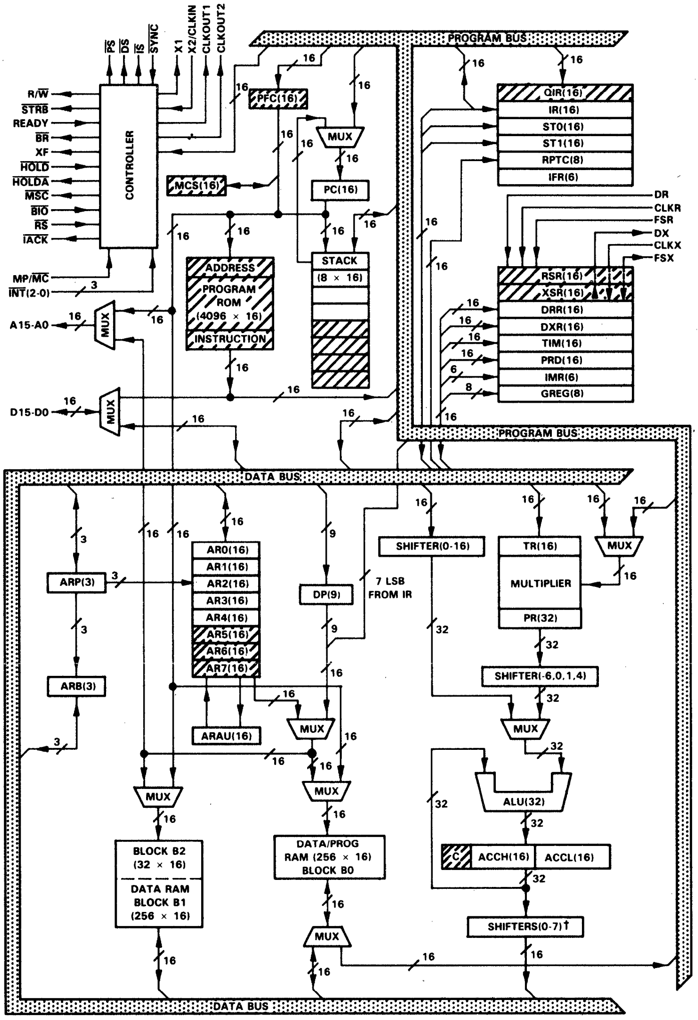
The 16 × 16-bit hardware multiplier is capable of computing a signed or unsigned 32-bit product in a single machine cycle. The multiplier has the following two associated registers:

- A 16-bit Temporary Register (TR) that holds one of the operands for the multiplier, and
- A 32-bit Product Register (PR) that holds the product.

Incorporated into the instruction set are single-cycle multiply/accumulate instructions that allow both operands to be processed simultaneously. The data for these operations may reside anywhere in internal or external memory, and can be transferred to the multiplier each cycle via the program and data buses.

Four product shift modes are available at the Product Register (PR) output that are useful when performing multiply/accumulate operations, fractional arithmetic, or justifying fractional products.

functional block diagram (TMS320C2x)



†Shifters on TMS32020 (0, 1, 4)

NOTE: Shaded areas are for TMS320C25 only.

LEGEND:

ACCH = Accumulator high
ACCL = Accumulator low
ALU = Arithmetic logic unit
ARAU = Auxiliary register arithmetic unit
ARB = Auxiliary register pointer buffer
ARP = Auxiliary register pointer
DP = Data memory page pointer
DRR = Serial port data receive register
DXR = Serial port data transmit register

IFR = Interrupt flag register
IMR = Interrupt mask register
IR = Instruction register
MCS = Microcall stack
QIR = Queue instruction register
PR = Product register
PRD = Period register for timer
TIM = Timer
TR = Temporary register

PC = Program counter
PFC = Prefetch counter
RPTC = Repeat instruction counter
GREG = Global memory allocation register
RSR = Serial port receive shift register
XSR = Serial port transmit shift register
AR0-AR7 = Auxiliary registers
ST0,ST1 = Status registers
C = Carry bit

timer

The TMS320C2x provides a memory-mapped 16-bit timer for control operations. The on-chip timer (TIM) register is a down counter that is continuously clocked by CLKOUT1 on the TMS320C25. The timer is clocked by CLKOUT1/4 on the TMS32020. A timer interrupt (TINT) is generated every time the timer decrements to zero. The timer is reloaded with the value contained in the period (PRD) register within the next cycle after it reaches zero so that interrupts may be programmed to occur at regular intervals of PRD + 1 cycles of CLKOUT1 on the TMS320C25 or 4 × PRD × CLKOUT1 cycles on the TMS32020.

memory control

The TMS320C2x provides a total of 544 16-bit words of on-chip data RAM, divided into three separate blocks (B0, B1, and B2). Of the 544 words, 288 words (blocks B1 and B2) are always data memory, and 256 words (block B0) are programmable as either data or program memory. A data memory size of 544 words allows the TMS320C2x to handle a data array of 512 words (256 words if on-chip RAM is used for program memory), while still leaving 32 locations for intermediate storage. When using block B0 as program memory, instructions can be downloaded from external program memory into on-chip RAM and then executed.

When using on-chip program RAM, ROM, or high-speed external program memory, the TMS320C2x runs at full speed without wait states. However, the READY line can be used to interface the TMS320C2x to slower, less-expensive external memory. Downloading programs from slow off-chip memory to on-chip program RAM speeds processing while cutting system costs.

The TMS320C2x provides three separate address spaces for program memory, data memory, and I/O. The on-chip memory is mapped into either the 64K-word data memory or program memory space, depending upon the memory configuration (see Figure 1). The CNFD (configure block B0 as data memory) and CNFP (configure block B0 as program memory) instructions allow dynamic configuration of the memory maps through software. Regardless of the configuration, the user may still execute from external program memory.

The TMS320C2x has six registers that are mapped into the data memory space: a serial port data receive register, serial port data transmit register, timer register, period register, interrupt mask register, and global memory allocation register.

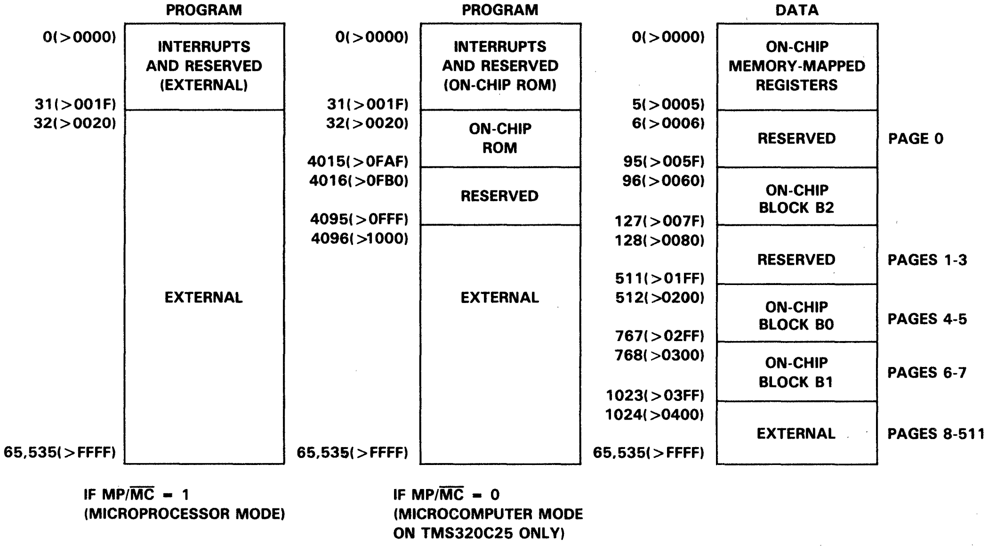
interrupts and subroutines

The TMS320C2x has three external maskable user interrupts $\overline{\text{INT2}}$-$\overline{\text{INT0}}$, available for external devices that interrupt the processor. Internal interrupts are generated by the serial port (RINT and XINT), by the timer (TINT), and by the software interrupt (TRAP) instruction. Interrupts are prioritized with reset ($\overline{\text{RS}}$) having the highest priority and the serial port transmit interrupt (XINT) having the lowest priority. All interrupt locations are on two-word boundaries so that branch instructions can be accommodated in those locations if desired.

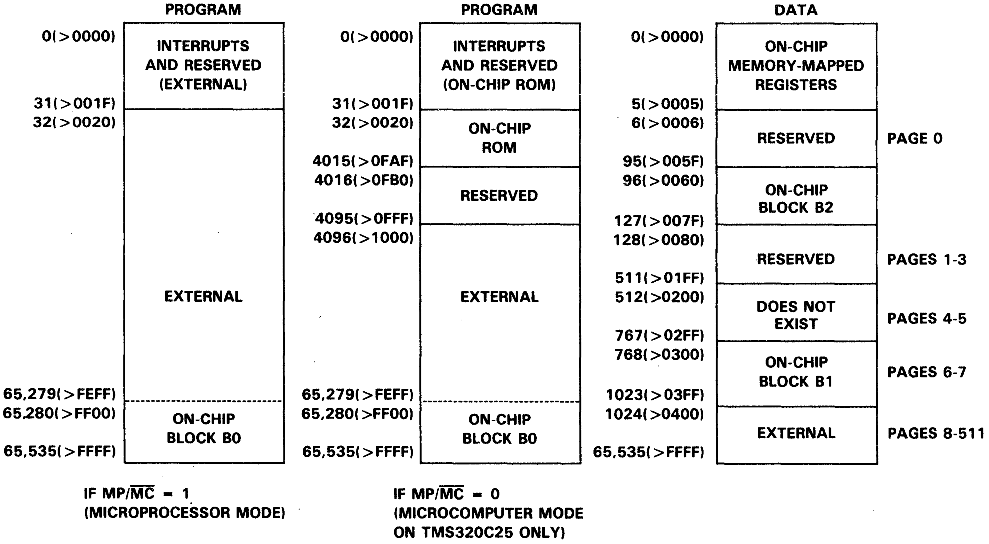
A built-in mechanism protects multicycle instructions from interrupts. If an interrupt occurs during a multicycle instruction, the interrupt is not processed until the instruction is completed. This mechanism applies to instructions that are repeated and to instructions that become multicycle due to the READY signal.

external interface

The TMS320C2x supports a wide range of system interfacing requirements. Program, data, and I/O address spaces provide interface to memory and I/O, thus maximizing system throughput. I/O design is simplified by having I/O treated the same way as memory. I/O devices are mapped into the I/O address space using the processor's external address and data buses in the same manner as memory-mapped devices. Interface to memory and I/O devices of varying speeds is accomplished by using the READY line. When transactions are made with slower devices, the TMS320C2x processor waits until the other device completes its function and signals the processor via the READY line. Then, the TMS320C2x continues execution.



(a) MEMORY MAPS AFTER A CNFD INSTRUCTION



(b) MEMORY MAPS AFTER A CNFP INSTRUCTION

FIGURE 1. MEMORY MAPS

A full-duplex serial port provides communication with serial devices, such as codecs, serial A/D converters, and other serial systems. The interface signals are compatible with codecs and many other serial devices with a minimum of external hardware. The serial port may also be used for intercommunication between processors in multiprocessing applications.

The serial port has two memory-mapped registers: the data transmit register (DXR) and the data receive register (DRR). Both registers operate in either the byte mode or 16-bit word mode, and may be accessed in the same manner as any other data memory location. Each register has an external clock, a framing synchronization pulse, and associated shift registers. One method of multiprocessing may be implemented by programming one device to transmit while the others are in the receive mode. The serial port on the TMS320C25 is double-buffered and fully static.

multiprocessing

The flexibility of the TMS320C2x allows configurations to satisfy a wide range of system requirements and can be used as follows:

- A standalone processor
- A multiprocessor with devices in parallel
- A slave/host multiprocessor with global memory space
- A peripheral processor interfaced via processor-controlled signals to another device.

For multiprocessing applications, the TMS320C2x has the capability of allocating global data memory space and communicating with that space via the $\overline{\text{BR}}$ (bus request) and READY control signals. Global memory is data memory shared by more than one processor. Global data memory access must be arbitrated. The 8-bit memory-mapped GREG (global memory allocation register) specifies part of the TMS320C2x's data memory as global external memory. The contents of the register determine the size of the global memory space. If the current instruction addresses an operand within that space, $\overline{\text{BR}}$ is asserted to request control of the bus. The length of the memory cycle is controlled by the READY line.

The TMS320C2x supports DMA (direct memory access) to its external program/data memory using the $\overline{\text{HOLD}}$ and $\overline{\text{HOLDA}}$ signals. Another processor can take complete control of the TMS320C2x's external memory by asserting $\overline{\text{HOLD}}$ low. This causes the TMS320C2x to place its address, data, and control lines in a high-impedance state, and assert $\overline{\text{HOLDA}}$. On the TMS320C25, program execution from on-chip ROM may proceed concurrently when the device is in the hold mode.

instruction set

The TMS320C2x microprocessor implements a comprehensive instruction set that supports both numeric-intensive signal processing operations as well as general-purpose applications, such as multiprocessing and high-speed control. The TMS32020 source code is upward-compatible with TMS320C25 source code. TMS32020 object code runs directly on the TMS320C25.

For maximum throughput, the next instruction is prefetched while the current one is being executed. Since the same data lines are used to communicate to external data/program or I/O space, the number of cycles may vary depending upon whether the next data operand fetch is from internal or external memory. Highest throughput is achieved by maintaining data memory on-chip and using either internal or fast external program memory.

addressing modes

The TMS320C2x instruction set provides three memory addressing modes: direct, indirect, and immediate addressing.

Both direct and indirect addressing can be used to access data memory. In direct addressing, seven bits of the instruction word are concatenated with the nine bits of the data memory page pointer to form the 16-bit data memory address. Indirect addressing accesses data memory through the auxiliary registers. In immediate addressing, the data is based on a portion of the instruction word(s).

In direct memory addressing, the instruction word contains the lower seven bits of the data memory address. This field is concatenated with the nine bits of the data memory page pointer to form the full 16-bit address. Thus, memory is paged in the direct addressing mode with a total of 512 pages, each page containing 128 words.

Up to eight auxiliary registers (AR0-AR7) provide flexible and powerful indirect addressing (five on the TMS32020, eight on the TMS320C25). To select a specific auxiliary register, the Auxiliary Register Pointer (ARP) is loaded with a value from 0 to 7 for AR0 through AR7, respectively.

There are seven types of indirect addressing: auto-increment or auto-decrement, post-indexing by either adding or subtracting the contents of AR0, single indirect addressing with no increment or decrement, and bit-reversal addressing (used in FFTs on the TMS320C25 only) with increment or decrement. All operations are performed on the current auxiliary register in the same cycle as the original instruction, following which the current auxiliary register and ARP may be modified.

repeat feature

A repeat feature, used with instructions such as multiply/accumulates, block moves, I/O transfers, and table read/writes, allows a single instruction to be performed up to 256 times. The repeat counter (RPTC) is loaded with either a data memory value (RPT instruction) or an immediate value (RPTK instruction). The value of this operand is one less than the number of times that the next instruction is executed. Those instructions that are normally multicycle are pipelined when using the repeat feature, and effectively become single-cycle instructions.

instruction set summary

Table 2 lists the symbols and abbreviations used in Table 3, the TMS320C25 instruction set summary. Table 3 consists primarily of single-cycle, single-word instructions. Infrequently used branch, I/O, and CALL instructions are multicycle. The instruction set summary is arranged according to function and alphabetized within each functional grouping. The symbol (†) indicates those instructions that are not included in the TMS32010 instruction set. The symbol (‡) indicates instructions that are not included in the TMS32020 instruction set.

TABLE 2. INSTRUCTION SYMBOLS

SYMBOL	MEANING
B	4-bit field specifying a bit code
CM	2-bit field specifying compare mode
D	Data memory address field
FO	Format status bit
I	Addressing mode bit
K	Immediate operand field
PA	Port address (PA0 through PA15 are predefined assembler symbols equal to 0 through 15, respectively.)
PM	2-bit field specifying P register output shift code
R	3-bit operand field specifying auxiliary register
S	4-bit left-shift code
X	3-bit accumulator left-shift field

TABLE 3. TMS320C25 INSTRUCTION SET SUMMARY

ACCUMULATOR MEMORY REFERENCE INSTRUCTIONS

MNEMONIC	DESCRIPTION	NO. WORDS	INSTRUCTION BIT CODE															
			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ABS	Absolute value of accumulator	1	1	1	0	0	1	1	1	0	0	0	0	1	1	0	1	1
ADD	Add to accumulator with shift	1	0	0	0	0	←S→				I	←D→						
ADDC‡	Add to accumulator with carry	1	0	1	0	0	0	0	1	1	I	←D→						
ADDH	Add to high accumulator	1	0	1	0	0	1	0	0	0	I	←D→						
ADDK‡	Add to accumulator short immediate	1	1	1	0	0	1	1	0	0	←K→							
ADDS	Add to low accumulator with sign extension suppressed	1	0	1	0	0	1	0	0	1	I	←D→						
ADDT†	Add to accumulator with shift specified by T register	1	0	1	0	0	1	0	1	0	I	←D→						
ADLK†	Add to accumulator long immediate with shift	2	1	1	0	1	←S→				0	0	0	0	0	0	1	0
AND	AND with accumulator	1	0	1	0	0	1	1	1	0	I	←D→						
ANDK†	AND immediate with accumulator with shift	2	1	1	0	1	←S→				0	0	0	0	0	1	0	0
CMPL†	Complement accumulator	1	1	1	0	0	1	1	1	0	0	0	1	0	0	1	1	1
LAC	Load accumulator with shift	1	0	0	1	0	←S→				I	←D→						
LACK	Load accumulator immediate short	1	1	1	0	0	1	0	1	0	←K→							
LACT†	Load accumulator with shift specified by T register	1	0	1	0	0	0	0	1	0	I	←D→						
LALK†	Load accumulator long immediate with shift	2	1	1	0	1	←S→				0	0	0	0	0	0	0	1
NEG†	Negate accumulator	1	1	1	0	0	1	1	1	0	0	0	1	0	0	0	1	1
NORM†	Normalize contents of accumulator	1	1	1	0	0	1	1	1	0	1	←D→						
OR	OR with accumulator	1	0	1	0	0	1	1	0	1	I	←D→						
ORK†	OR immediate with accumulator with shift	2	1	1	0	1	←S→				0	0	0	0	0	1	0	1
ROL‡	Rotate accumulator left	1	1	1	0	0	1	1	1	0	0	0	1	1	0	1	0	0
ROR‡	Rotate accumulator right	1	1	1	0	0	1	1	1	0	0	0	1	1	0	1	0	1
SACH	Store high accumulator with shift	1	0	1	1	0	1	←X→			I	←D→						
SACL	Store low accumulator with shift	1	0	1	1	0	0	←X→			I	←D→						
SBLK†	Subtract from accumulator long immediate with shift	2	1	1	0	1	←S→				0	0	0	0	0	0	1	1
SFL†	Shift accumulator left	1	1	1	0	0	1	1	1	0	0	0	0	1	1	0	0	0
SFR†	Shift accumulator right	1	1	1	0	0	1	1	1	0	0	0	0	1	1	0	0	1
SUB	Subtract from accumulator with shift	1	0	0	0	1	←S→				I	←D→						
SUBB‡	Subtract from accumulator with borrow	1	0	1	0	0	1	1	1	1	I	←D→						
SUBC	Conditional subtract	1	0	1	0	0	0	1	1	1	I	←D→						
SUBH	Subtract from high accumulator	1	0	1	0	0	0	1	0	0	I	←D→						
SUBK‡	Subtract from accumulator short immediate	1	1	1	0	0	1	1	0	1	←K→							
SUBS	Subtract from low accumulator with sign extension suppressed	1	0	1	0	0	0	1	0	1	I	←D→						
SUBT†	Subtract from accumulator with shift specified by T register	1	0	1	0	0	0	1	1	0	I	←D→						
XOR	Exclusive-OR with accumulator	1	0	1	0	0	1	1	0	0	I	←D→						
XORK†	Exclusive-OR immediate with accumulator with shift	2	1	1	0	1	←S→				0	0	0	0	0	1	1	0
ZAC	Zero accumulator	1	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0
ZALH	Zero low accumulator and load high accumulator	1	0	1	0	0	0	0	0	0	I	←D→						
ZALR‡	Zero low accumulator and load high accumulator with rounding	1	0	1	1	1	1	0	1	1	I	←D→						
ZALS	Zero accumulator and load low accumulator with sign extension suppressed	1	0	1	0	0	0	0	1	1	I	←D→						

†These instructions are not included in the TMS32010 instruction set.
‡These instructions are not included in the TMS32020 instruction set.

TABLE 3. TMS320C25 INSTRUCTION SET SUMMARY (CONTINUED)

AUXILIARY REGISTERS AND DATA PAGE POINTER INSTRUCTIONS

MNEMONIC	DESCRIPTION	NO. WORDS	INSTRUCTION BIT CODE															
			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADRK‡	Add to auxiliary register short immediate	1	0	1	1	1	1	1	1	0	◄—K—►							
CMPR†	Compare auxiliary register with auxiliary register AR0	1	1	1	0	0	1	1	1	0	0	1	0	1	0	0	◄CM►	
LAR	Load auxiliary register	1	0	0	1	1	0	◄—R—►			I	◄—D—►						
LARK	Load auxiliary register short immediate	1	1	1	0	0	0	◄—R—►			◄—K—►							
LARP	Load auxiliary register pointer	1	0	1	0	1	0	1	0	1	1	0	0	0	1	◄—R—►		
LDP	Load data memory page pointer	1	0	1	0	1	0	0	1	0	I	◄—D—►						
LDPK	Load data memory page pointer immediate	1	1	1	0	0	1	0	0	◄—DP—►								
LRLK†	Load auxiliary register long immediate	2	1	1	0	1	0	◄—R—►			0	0	0	0	0	0	0	0
MAR	Modify auxiliary register	1	0	1	0	1	0	1	0	1	I	◄—D—►						
SAR	Store auxiliary register	1	0	1	1	1	0	◄—R—►			I	◄—D—►						
SBRK‡	Subtract from auxiliary register short immediate	1	0	1	1	1	1	1	1	1	◄—K—►							

T REGISTER, P REGISTER, AND MULTIPLY INSTRUCTIONS

MNEMONIC	DESCRIPTION	NO. WORDS	INSTRUCTION BIT CODE															
			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
APAC	Add P register to accumulator	1	1	1	0	0	1	1	1	0	0	0	0	1	0	1	0	1
LPH†	Load high P register	1	0	1	0	1	0	0	1	1	I	◄—D—►						
LT	Load T register	1	0	0	1	1	1	1	0	0	I	◄—D—►						
LTA	Load T register and accumulate previous product	1	0	0	1	1	1	1	0	1	I	◄—D—►						
LTD	Load T register, accumulate previous product, and move data	1	0	0	1	1	1	1	1	1	I	◄—D—►						
LTP†	Load T register and store P register in accumulator	1	0	0	1	1	1	1	1	0	I	◄—D—►						
LTS†	Load T register and subtract previous product	1	0	1	0	1	1	0	1	1	I	◄—D—►						
MAC†	Multiply and accumulate	2	0	1	0	1	1	1	0	1	I	◄—D—►						
MACD†	Multiply and accumulate with data move	2	0	1	0	1	1	1	0	0	I	◄—D—►						
MPY	Multiply (with T register, store product in P register)	1	0	0	1	1	1	0	0	0	I	◄—D—►						
MPYA‡	Multiply and accumulate previous product	1	0	0	1	1	1	0	1	0	I	◄—D—►						
MPYK	Multiply immediate	1	1	0	1	◄—K—►												
MPYS‡	Multiply and subtract previous product	1	0	0	1	1	1	0	1	1	I	◄—D—►						
MPYU‡	Multiply unsigned	1	1	1	0	0	1	1	1	1	I	◄—D—►						
PAC	Load accumulator with P register	1	1	1	0	0	1	1	1	0	0	0	0	1	0	1	0	0
SPAC	Subtract P register from accumulator	1	1	1	0	0	1	1	1	0	0	0	0	1	0	1	1	0
SPH‡	Store high P register	1	0	1	1	1	1	1	0	1	I	◄—D—►						
SPL‡	Store low P register	1	0	1	1	1	1	1	0	0	I	◄—D—►						
SPM†	Set P register output shift mode	1	1	1	0	0	1	1	1	0	0	0	0	0	1	0	◄PM►	
SQRA†	Square and accumulate	1	0	0	1	1	1	0	0	1	I	◄—D—►						
SQRS†	Square and subtract previous product	1	0	1	0	1	1	0	1	0	I	◄—D—►						

†These instructions are not included in the TMS32010 instruction set.
‡These instructions are not included in the TMS32020 instruction set.

TABLE 3. TMS320C25 INSTRUCTION SET SUMMARY (CONTINUED)

BRANCH/CALL INSTRUCTIONS

MNEMONIC	DESCRIPTION	NO. WORDS	INSTRUCTION BIT CODE															
			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B	Branch unconditionally	2	1	1	1	1	1	1	1	1	1	◄——D——►						
BACC†	Branch to address specified by accumulator	1	1	1	0	0	1	1	1	0	0	0	1	0	0	1	0	1
BANZ	Branch on auxiliary register not zero	2	1	1	1	1	1	0	1	1	1	◄——D——►						
BBNZ†	Branch if TC bit ≠ 0	2	1	1	1	1	1	0	0	1	1	◄——D——►						
BBZ†	Branch if TC bit = 0	2	1	1	1	1	1	0	0	0	1	◄——D——►						
BC‡	Branch on carry	2	0	1	0	1	1	1	1	0	1	◄——D——►						
BGEZ	Branch if accumulator ≥ 0	2	1	1	1	1	0	1	0	0	1	◄——D——►						
BGZ	Branch if accumulator > 0	2	1	1	1	1	0	0	0	1	1	◄——D——►						
BIOZ	Branch on I/O status = 0	2	1	1	1	1	1	0	1	0	1	◄——D——►						
BLEZ	Branch if accumulator ≤ 0	2	1	1	1	1	0	0	1	0	1	◄——D——►						
BLZ	Branch if accumulator < 0	2	1	1	1	1	0	0	1	1	1	◄——D——►						
BNC‡	Branch on no carry	2	0	1	0	1	1	1	1	1	1	◄——D——►						
BNV†	Branch if no overflow	2	1	1	1	1	0	1	1	1	1	◄——D——►						
BNZ	Branch if accumulator ≠ 0	2	1	1	1	1	0	1	0	1	1	◄——D——►						
BV	Branch on overflow	2	1	1	1	1	0	0	0	0	1	◄——D——►						
BZ	Branch if accumulator = 0	2	1	1	1	1	0	1	1	0	1	◄——D——►						
CALA	Call subroutine indirect	1	1	1	0	0	1	1	1	0	0	0	1	0	0	1	0	0
CALL	Call subroutine	2	1	1	1	1	1	1	1	0	1	◄——D——►						
RET	Return from subroutine	1	1	1	0	0	1	1	1	0	0	0	1	0	0	1	1	0

I/O AND DATA MEMORY OPERATIONS

MNEMONIC	DESCRIPTION	NO. WORDS	INSTRUCTION BIT CODE															
			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BLKD†	Block move from data memory to data memory	2	1	1	1	1	1	1	0	1	I	◄——D——►						
BLKP†	Block move from program memory to data memory	2	1	1	1	1	1	1	0	0	I	◄——D——►						
DMOV	Data move in data memory	1	0	1	0	1	0	1	1	0	I	◄——D——►						
FORT†	Format serial port registers	1	1	1	0	0	1	1	1	0	0	0	0	0	1	1	1	FO
IN	Input data from port	1	1	0	0	0	◄—PA—►				I	◄——D——►						
OUT	Output data to port	1	1	1	1	0	◄—PA—►				I	◄——D——►						
RFSM‡	Reset serial port frame synchronization mode	1	1	1	0	0	1	1	1	0	0	0	1	1	0	1	1	0
RTXM†	Reset serial port transmit mode	1	1	1	0	0	1	1	1	0	0	0	1	0	0	0	0	0
RXF†	Reset external flag	1	1	1	0	0	1	1	1	0	0	0	0	0	1	1	0	0
SFSM‡	Set serial port frame synchronization mode	1	1	1	0	0	1	1	1	0	0	0	1	1	0	1	1	1
STXM†	Set serial port transmit mode	1	1	1	0	0	1	1	1	0	0	0	1	0	0	0	0	1
SXF†	Set external flag	1	1	1	0	0	1	1	1	0	0	0	0	0	1	1	0	1
TBLR	Table read	1	0	1	0	1	1	0	0	0	I	◄——D——►						
TBLW	Table write	1	0	1	0	1	1	0	0	1	I	◄——D——►						

† These instructions are not included in the TMS32010 instruction set.

‡ These instructions are not included in the TMS32020 instruction set.

TABLE 3. TMS320C25 INSTRUCTION SET SUMMARY (CONCLUDED)

CONTROL INSTRUCTIONS																		
MNEMONIC	DESCRIPTION	NO. WORDS	INSTRUCTION BIT CODE															
			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BIT†	Test bit	1	1	0	0	1	◄—B—►				I	◄—D—►						
BITT†	Test bit specified by T register	1	0	1	0	1	0	1	1	1	I	◄—D—►						
CNFD†	Configure block as data memory	1	1	1	0	0	1	1	1	0	0	0	0	0	0	1	0	0
CNFP†	Configure block as program memory	1	1	1	0	0	1	1	1	0	0	0	0	0	0	1	0	1
DINT	Disable interrupt	1	1	1	0	0	1	1	1	0	0	0	0	0	0	0	0	1
EINT	Enable interrupt	1	1	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0
IDLE†	Idle until interrupt	1	1	1	0	0	1	1	1	0	0	0	0	1	1	1	1	1
LST	Load status register ST0	1	0	1	0	1	0	0	0	0	I	◄—D—►						
LST1†	Load status register ST1	1	0	1	0	1	0	0	0	1	I	◄—D—►						
NOP	No operation	1	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0
POP	Pop top of stack to low accumulator	1	1	1	0	0	1	1	1	0	0	0	0	1	1	1	0	1
POPD†	Pop top of stack to data memory	1	0	1	1	1	1	0	1	0	I	◄—D—►						
PSHD†	Push data memory value onto stack	1	0	1	0	1	0	1	0	0	I	◄—D—►						
PUSH	Push low accumulator onto stack	1	1	1	0	0	1	1	1	0	0	0	0	1	1	1	0	0
RC‡	Reset carry bit	1	1	1	0	0	1	1	1	0	0	0	1	1	0	0	0	0
RHM‡	Reset hold mode	1	1	1	0	0	1	1	1	0	0	0	1	1	1	0	0	0
ROVM	Reset overflow mode	1	1	1	0	0	1	1	1	0	0	0	0	0	0	0	1	0
RPT†	Repeat instruction as specified by data memory value	1	0	1	0	0	1	0	1	1	I	◄—D—►						
RPTK†	Repeat instruction as specified by immediate value	1	1	1	0	0	1	0	1	1	◄—K—►							
RSXM†	Reset sign-extension mode	1	1	1	0	0	1	1	1	0	0	0	0	0	0	1	1	0
RTC‡	Reset test/control flag	1	1	1	0	0	1	1	1	0	0	0	1	1	0	0	1	0
SC‡	Set carry bit	1	1	1	0	0	1	1	1	0	0	0	1	1	0	0	0	1
SHM‡	Set hold mode	1	1	1	0	0	1	1	1	0	0	0	1	1	1	0	0	1
SOVM	Set overflow mode	1	1	1	0	0	1	1	1	0	0	0	0	0	0	0	1	1
SST	Store status register ST0	1	0	1	1	1	1	0	0	0	I	◄—D—►						
SST1†	Store status register ST1	1	0	1	1	1	1	0	0	1	I	◄—D—►						
SSXM†	Set sign-extension mode	1	1	1	0	0	1	1	1	0	0	0	0	0	0	1	1	1
STC‡	Set test/control flag	1	1	1	0	0	1	1	1	0	0	0	1	1	0	0	1	1
TRAP†	Software interrupt	1	1	1	0	0	1	1	1	0	0	0	0	1	1	1	1	0

† These instructions are not included in the TMS32010 instruction set.

‡ These instructions are not included in the TMS32020 instruction set.

development support

Texas Instruments offers an extensive line of development support products to assist the user in all aspects of TMS320 second-generation-based design and development. These products range from development and application software to complete hardware development and evaluation systems such as the XDS/22. Table 4 lists the development support products for the second-generation TMS320 devices.

System development begins with the use of the SoftWare Development System (SWDS) or Emulator (XDS). These tools allow the designer to evaluate the processor's performance, benchmark time-critical code, and determine the feasibility of using a TMS320 device to implement a specific algorithm.

Software and hardware can be developed in parallel by using the macro assembler/linker, simulator, and SoftWare Development System for software development and the XDS for hardware development. The assembler/linker translates the system's assembly source program into an object module that can be executed by the simulator, XDS, or SWDS. The XDS provides realtime in-circuit emulation and is a powerful tool for debugging and integrating software and hardware modules.

Additional support for the TMS320 products consists of extensive documentation and three-day DSP design workshops offered by the TI Regional Technology Centers (RTCs). The workshops provide hands-on experience with the TMS320 development tools. Refer to the *TMS320 Family Development Support Reference Guide* for further information about TMS320 development support products and DSP workshops. When technical questions arise regarding the TMS320, contact the Texas Instruments TMS320 Hotline, (713) 274-2320.

TABLE 4. TMS320 SECOND-GENERATION SOFTWARE AND HARDWARE SUPPORT

SOFTWARE TOOLS	PART NUMBER
Macro Assembler/Linker	
VAX VMS	TMDS3242210-08
TI/IBM MS/PC-DOS	TMDS3242810-02
Simulator	
VAX VMS	TMDS3242211-08
TI/IBM MS/PC-DOS	TMDS3242811-02
SoftWare Development System (SWDS)	TMDS3268821
Digital Filter Design Package (DFDP)	
IBM PC-DOS	DFDP-IBM002
DSP Software Library	
VAX VMS	TMDC3240212-18
TI/IBM MS/PC-DOS	TMDC3240812-12
HARDWARE TOOLS	**PART NUMBER**
Analog Interface Board (AIB)	RTC/EVM320C-06
XDS/22 Emulator	TMDS3262221
XDS/22 Upgrade	
Customer Upgrade	TMDS3282226
TMS320 Design Kit	TMS320DDK

documentation support

Extensive documentation supports the second-generation TMS320 devices from product announcement through applications development. The types of documentation include data sheets with design specifications, complete user's guides, and 750 pages of application reports published in the book, *Digital Signal Processing Applications with the TMS320 Family.* An application report, *Hardware Interfacing to the TMS320C25,* is available for that device.

A series of DSP textbooks is being published by Prentice-Hall and John Wiley & Sons to support digital signal processing research and education. The TMS320 newsletter, *Details on Signal Processing,* is published quarterly and distributed to update TMS320 customers on product information. The TMS320 DSP bulletin board service provides access to large amounts of information pertaining to the TMS320 family.

Refer to the *TMS320 Family Development Support Reference Guide* for further information about TMS320 documentation. To receive copies of second-generation TMS320 literature, call the Customer Response Center at 1-800-232-3200.

absolute maximum ratings over specified temperature range (unless otherwise noted)[†]

Supply voltage range, V_{CC}[‡]	−0.3 V to 7 V
Input voltage range	−0.3 V to 7 V
Output voltage range	−0.3 V to 7 V
Continuous power dissipation	2.0 W
Operating free-air temperature range	0 °C to 70 °C
Storage temperature range	−55 °C to 150 °C

[†] Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

[‡] All voltage values are with respect to V_{SS}.

recommended operating conditions

		MIN	NOM	MAX	UNIT
V_{CC} Supply voltage		4.75	5	5.25	V
V_{SS} Supply voltage			0		V
V_{IH} High-level input voltage	All inputs except CLKIN	2		V_{CC}+0.3	V
	CLKIN	2.4		V_{CC}+0.3	V
V_{IL} Low-level input voltage	All inputs except CLKIN	−0.3		0.8	V
	CLKIN	−0.3		0.8	V
I_{OH} High-level output current				300	µA
I_{OL} Low-level output current				2	mA
T_A Operating free-air temperature (see Notes 1 and 2)		0		70	°C

NOTES:
1. Case temperature (T_C) must be maintained below 90 °C.
2. $R_{\theta JA}$ = 36 °C/Watt, $R_{\theta JC}$ = 6 °C/Watt.

electrical characteristics over specified free-air temperature range (unless otherwise noted)

PARAMETER	TEST CONDITIONS	MIN	TYP[†]	MAX	UNIT
V_{OH} High-level output voltage	V_{CC} = MIN, I_{OH} = MAX	2.4	3		V
V_{OL} Low-level output voltage	V_{CC} = MIN, I_{OL} = MAX		0.3	0.6	V
I_Z Three-state current	V_{CC} = MAX	−20		20	µA
I_I Input current	V_I = V_{SS} to V_{CC}	−10		10	µA
I_{CC} Supply current	T_A = 0°C, V_{CC} = MAX, f_x = MAX			360	mA
	T_A = 25°C, V_{CC} = MAX, f_x = MAX		250		mA
	T_C = 90°C, V_{CC} = MAX, f_x = MAX			285	mA
C_I Input capacitance				15	pF
C_O Output capacitance				15	pF

[†] All typical values are at V_{CC} = 5 V, T_A = 25 °C.

Caution. This device contains circuits to protect its inputs and outputs against damage due to high static voltages or electrostatic fields. These circuits have been qualified to protect this device against electrostatic discharges (ESD) of up to 2 kV according to MIL-STD-883C, Method 3015; however, it is advised that precautions be taken to avoid application of any voltage higher than maximum rated voltages to these high-impedance circuits. During storage or handling, the device leads should be shorted together or the device should be placed in conductive foam. In a circuit, unused inputs should always be connected to an appropriate logic voltage level, preferably either V_{CC} or ground. Specific guidelines for handling devices of this type are contained in the publication "Guidelines for Handling Electrostatic-Discharge Sensitive (ESDS) Devices and Assemblies" available from Texas Instruments.

CLOCK CHARACTERISTICS AND TIMING

The TMS32020 can use either its internal oscillator or an external frequency source for a clock.

internal clock option

The internal oscillator is enabled by connecting a crystal across X1 and X2/CLKIN (see Figure 2). The frequency of CLKOUT1 is one-fourth the crystal fundamental frequency. The crystal should be fundamental mode, and parallel resonant, with an effective series resistance of 30 ohms, a power dissipation of 1 mW, and be specified at a load capacitance of 20 pF.

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
f_x	Input clock frequency	T_A = 0°C to 70°C	6.7		20.5	MHz
f_{sx}	Serial port frequency	T_A = 0°C to 70°C	50		2563	kHz
C1, C2		T_A = 0°C to 70°C		10		pF

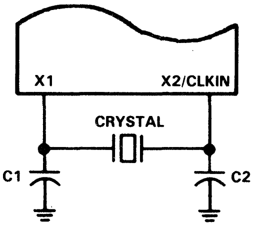


FIGURE 2. INTERNAL CLOCK OPTION

external clock option

An external frequency source can be used by injecting the frequency directly into X2/CLKIN with X1 left unconnected. The external frequency injected must conform to the specifications listed in the following table.

switching characteristics over recommended operating conditions (see Note 3)

PARAMETER		MIN	TYP	MAX	UNIT
$t_{c(C)}$	CLKOUT1/CLKOUT2 cycle time	195		597	ns
$t_{d(CIH-C)}$	CLKIN high to CLKOUT1/CLKOUT2/$\overline{\text{STRB}}$ high/low	25		60	ns
$t_{f(C)}$	CLKOUT1/CLKOUT2/$\overline{\text{STRB}}$ fall time			10	ns
$t_{r(C)}$	CLKOUT1/CLKOUT2/$\overline{\text{STRB}}$ rise time			10	ns
$t_{w(CL)}$	CLKOUT1/CLKOUT2 low pulse duration	2Q−15	2Q	2Q+15	ns
$t_{w(CH)}$	CLKOUT1/CLKOUT2 high pulse duration	2Q−15	2Q	2Q+15	ns
$t_{d(C1-C2)}$	CLKOUT1 high to CLKOUT2 low, CLKOUT2 high to CLKOUT1 high, etc.	Q−10	Q	Q+10	ns

NOTE 3: $Q = 1/4t_{c(C)}$.

timing requirements over recommended operating conditions (see Note 3)

		MIN	NOM	MAX	UNIT
$t_{c(CI)}$	CLKIN cycle time	48.8		150	ns
$t_{f(CI)}$	CLKIN fall time			10	ns
$t_{r(CI)}$	CLKIN rise time			10	ns
$t_{w(CIL)}$	CLKIN low pulse duration, $t_{c(CI)}$ = 50 ns (see Note 4)	10		40	ns
$t_{w(CIH)}$	CLKIN high pulse duration, $t_{c(CI)}$ = 50 ns (see Note 4)	10		40	ns
$t_{su(S)}$	$\overline{\text{SYNC}}$ setup time before CLKIN low	10		Q-10	ns
$t_{h(S)}$	$\overline{\text{SYNC}}$ hold time from CLKIN low	15			ns

NOTES: 3. $Q = 1/4t_{c(C)}$.
4. CLKIN duty cycle $[t_{r(CI)} + t_{w(CIH)}]/t_{c(CI)}$ must be within 40-60%.

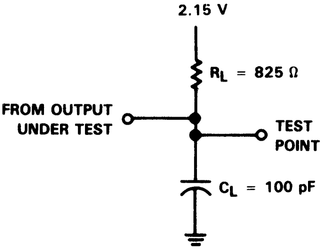
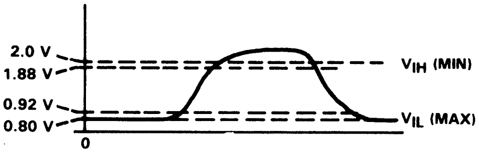
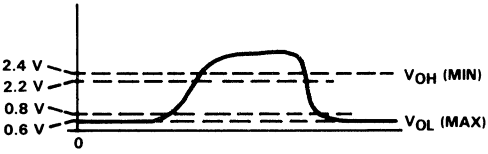


FIGURE 3. TEST LOAD CIRCUIT



(a) INPUT



(b) OUTPUTS

FIGURE 4. VOLTAGE REFERENCE LEVELS

MEMORY AND PERIPHERAL INTERFACE TIMING

switching characteristics over recommended operating conditions (see Note 3)

	PARAMETER	MIN	TYP	MAX	UNIT
$t_{d(C1\text{-}S)}$	$\overline{\text{STRB}}$ from CLKOUT1 (if $\overline{\text{STRB}}$ is present)	Q−15	Q	Q+15	ns
$t_{d(C2\text{-}S)}$	CLKOUT2 to $\overline{\text{STRB}}$ (if $\overline{\text{STRB}}$ is present)	−15	0	15	ns
$t_{su(A)}$	Address setup time before $\overline{\text{STRB}}$ low (see Note 5)	Q−30			ns
$t_{h(A)}$	Address hold time after $\overline{\text{STRB}}$ high (see Note 5)	Q−15			ns
$t_{w(SL)}$	$\overline{\text{STRB}}$ low pulse duration (no wait states, see Note 6)		2Q		ns
$t_{w(SH)}$	$\overline{\text{STRB}}$ high pulse duration (between consecutive cycles, see Note 6)		2Q		ns
$t_{su(D)W}$	Data write setup time before $\overline{\text{STRB}}$ high (no wait states)	2Q−45			ns
$t_{h(D)W}$	Data write hold time from $\overline{\text{STRB}}$ high	Q−15	Q		ns
$t_{en(D)}$	Data bus starts being driven after $\overline{\text{STRB}}$ low (write cycle)	0			ns
$t_{dis(D)}$	Data bus three-state after $\overline{\text{STRB}}$ high (write cycle)		Q	Q+30	ns
$t_{d(MSC)}$	$\overline{\text{MSC}}$ valid from CLKOUT1	−25	0	25	ns

NOTES: 3. Q = $1/4t_{c(C)}$.
5. A15-A0, $\overline{\text{PS}}$, $\overline{\text{DS}}$, $\overline{\text{IS}}$, R/$\overline{\text{W}}$, and $\overline{\text{BR}}$ timings are all included in timings referenced as "address."
6. Delays between CLKOUT1/CLKOUT2 edges and $\overline{\text{STRB}}$ edges track each other, resulting in $t_{w(SL)}$ and $t_{w(SH)}$ being 2Q with no wait states.

timing requirements over recommended operating conditions (see Note 3)

		MIN	NOM	MAX	UNIT
$t_{a(A)}$	Read data access time from address time (read cycle, see Notes 5 and 7)			3Q−70	ns
$t_{su(D)R}$	Data read setup time before $\overline{\text{STRB}}$ high	40			ns
$t_{h(D)R}$	Data read hold time from $\overline{\text{STRB}}$ high	0			ns
$t_{d(SL\text{-}R)}$	READY valid after $\overline{\text{STRB}}$ low (no wait states)			Q−40	ns
$t_{d(C2H\text{-}R)}$	READY valid after CLKOUT2 high			Q−40	ns
$t_{h(SL\text{-}R)}$	READY hold time after $\overline{\text{STRB}}$ low (no wait states)	Q−5			ns
$t_{h(C2H\text{-}R)}$	READY hold after CLKOUT2 high	Q−5			ns
$t_{d(M\text{-}R)}$	READY valid after $\overline{\text{MSC}}$ valid			2Q−50	ns
$t_{h(M\text{-}R)}$	READY hold time after $\overline{\text{MSC}}$ valid	0			ns

NOTES: 3. Q = $1/4t_{c(C)}$.
5. A15-A0, $\overline{\text{PS}}$, $\overline{\text{DS}}$, $\overline{\text{IS}}$, R/$\overline{\text{W}}$, and $\overline{\text{BR}}$ timings are all included in timings referenced as "address."
7. Read data access time is defined as $t_{a(A)} = t_{su(A)} + t_{w(SL)} - t_{su(D)R}$.

$\overline{\text{RS}}$, $\overline{\text{INT}}$, $\overline{\text{BIO}}$, AND XF TIMING

switching characteristics over recommended operating conditions (see Note 3)

PARAMETER		MIN	TYP	MAX	UNIT
$t_{d(RS)}$	CLKOUT1 low to reset state entered			45	ns
$t_{d(IACK)}$	CLKOUT1 to $\overline{\text{IACK}}$ valid	−25	0	25	ns
$t_{d(XF)}$	XF valid before falling edge of $\overline{\text{STRB}}$	Q−30			ns

NOTES: 3. Q = 1/4$t_{c(C)}$.

8. $\overline{\text{RS}}$, $\overline{\text{INT}}$, and $\overline{\text{BIO}}$ are asynchronous inputs and can occur at any time during a clock cycle. However, if the specified setup time is met, the exact sequence shown in the timing diagrams will occur.

timing requirements over recommended operating conditions (see Note 3)

		MIN	NOM	MAX	UNIT
$t_{su(IN)}$	$\overline{\text{INT}}$/$\overline{\text{BIO}}$/$\overline{\text{RS}}$ setup before CLKOUT1 high	50			ns
$t_{h(IN)}$	$\overline{\text{INT}}$/$\overline{\text{BIO}}$/$\overline{\text{RS}}$ hold after CLKOUT1 high	0			ns
$t_{f(IN)}$	$\overline{\text{INT}}$/$\overline{\text{BIO}}$ fall time			15	ns
$t_{w(IN)}$	$\overline{\text{INT}}$/$\overline{\text{BIO}}$ low pulse duration	$t_{c(C)}$			ns
$t_{w(RS)}$	$\overline{\text{RS}}$ low pulse duration	$3t_{c(C)}$			ns

NOTES: 3. Q = 1/4$t_{c(C)}$.

8. $\overline{\text{RS}}$, $\overline{\text{INT}}$, and $\overline{\text{BIO}}$ are asynchronous inputs and can occur at any time during a clock cycle. However, if the specified setup time is met, the exact sequence shown in the timing diagrams will occur.

$\overline{\text{HOLD}}$ TIMING

switching characteristics over recommended operating conditions (see Note 3)

PARAMETER		MIN	TYP	MAX	UNIT
$t_{d(C1L\text{-}AL)}$	$\overline{\text{HOLDA}}$ low after CLKOUT1 low	−25		25	ns
$t_{dis(AL\text{-}A)}$	$\overline{\text{HOLDA}}$ low to address three-state		15		ns
$t_{dis(C1L\text{-}A)}$	Address three-state after CLKOUT1 low ($\overline{\text{HOLD}}$ mode, see Note 9)			30	ns
$t_{d(HH\text{-}AH)}$	$\overline{\text{HOLD}}$ high to $\overline{\text{HOLDA}}$ high			50	ns
$t_{en(A\text{-}C1L)}$	Address driven before CLKOUT1 low ($\overline{\text{HOLD}}$ mode, see Note 9)			10	ns

NOTES: 3. Q = 1/4$t_{c(C)}$.

9. A15-A0, $\overline{\text{PS}}$, $\overline{\text{DS}}$, $\overline{\text{IS}}$, $\overline{\text{STRB}}$, and R/$\overline{\text{W}}$ timings are all included in timings referenced as "address."

timing requirements over recommended operating conditions (see Note 3)

		MIN	NOM	MAX	UNIT
$t_{d(C2H\text{-}H)}$	$\overline{\text{HOLD}}$ valid after CLKOUT2 high			Q−45	ns

NOTE 3: Q = 1/4$t_{c(C)}$.

SERIAL PORT TIMING

switching characteristics over recommended operating conditions (see Note 3)

PARAMETER		MIN	TYP	MAX	UNIT
$t_{d(CH-DX)}$	DX valid after CLKX rising edge (see Note 10)			100	ns
$t_{d(FL-DX)}$	DX valid after FSX falling edge (TXM = 0, see Note 10)			50	ns
$t_{d(CH-FS)}$	FSX valid after CLKX rising edge (TXM = 1)			60	ns

NOTES: 3. $Q = 1/4t_{c(C)}$.
10. The last occurrence of FSX falling and CLKX rising.

timing requirements over recommended operating conditions (see Note 3)

		MIN	NOM	MAX	UNIT
$t_{c(SCK)}$	Serial port clock (CLKX/CLKR) cycle time	390		20,000	ns
$t_{f(SCK)}$	Serial port clock (CLKX/CLKR) fall time			50	ns
$t_{r(SCK)}$	Serial port clock (CLKX/CLKR) rise time			50	ns
$t_{w(SCK)}$	Serial port clock (CLKX/CLKR) low pulse duration (see Note 11)	150		12,000	ns
$t_{w(SCK)}$	Serial port clock (CLKX/CLKR) high pulse duration (see Note 11)	150		12,000	ns
$t_{su(FS)}$	FSX/FSR setup time before CLKX/CLKR falling edge (TXM = 0)	20			ns
$t_{h(FS)}$	FSX/FSR hold time after CLKX/CLKR falling edge (TXM = 0)	20			ns
$t_{su(DR)}$	DR setup time before CLKR falling edge	20			ns
$t_{h(DR)}$	DR hold time after CLKR falling edge	20			ns

NOTES: 3. $Q = 1/4t_{c(C)}$.
11. The duty cycle of the serial port clock must be within 40-60%.

---

absolute maximum ratings over specified temperature range (unless otherwise noted)[†]

Supply voltage range, V_{CC}[‡] . . . . −0.3 V to 7 V
Input voltage range . . . . −0.3 V to 7 V
Output voltage range . . . . −0.3 V to 7 V
Continuous power dissipation . . . . 1.5 W
Operating free-air temperature range . . . . 0 °C to 70 °C
Storage temperature range . . . . −55 °C to 150 °C

[†] Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

[‡] All voltage values are with respect to V_{SS}.

recommended operating conditions

			MIN	NOM	MAX	UNIT
V_{CC}	Supply voltage		4.75	5	5.25	V
V_{SS}	Supply voltage			0		V
V_{IH}	High-level input voltage	All inputs except CLKIN/CLKX/CLKR/$\overline{\text{INT}}$ (0-2)	2.35		V_{CC}+0.3	V
		$\overline{\text{INT}}$ (0-2)	2.5		V_{CC}+0.3	V
		CLKIN/CLKX/CLKR	3.5		V_{CC}+0.3	V
V_{IL}	Low-level input voltage	All inputs except CLKIN	−0.3		0.8	V
		CLKIN	−0.3		0.8	V
I_{OH}	High-level output current				300	μA
I_{OL}	Low-level output current				2	mA
T_A	Operating free-air temperature		0		70	°C

electrical characteristics over specified free-air temperature range (unless otherwise noted)

PARAMETER			TEST CONDITIONS	MIN	TYP[†]	MAX	UNIT
V_{OH}	High-level output voltage		V_{CC} = MIN, I_{OH} = MAX	2.4	3		V
V_{OL}	Low-level output voltage		V_{CC} = MIN, I_{OL} = MAX		0.3	0.6	V
I_Z	Three-state current		V_{CC} = MAX	−20		20	μA
I_I	Input current		V_I = V_{SS} to V_{CC}	−10		10	μA
I_{CC}	Supply current	Normal	T_A = 0 °C, V_{CC} = MAX, f_x = MAX			185	mA
		Idle/$\overline{\text{HOLD}}$				100	
C_I	Input capacitance				15		pF
C_O	Output capacitance				15		pF

[†] All typical values are at V_{CC} = 5 V, T_A = 25 °C.

Caution. This device contains circuits to protect its inputs and outputs against damage due to high static voltages or electrostatic fields. These circuits have been qualified to protect this device against electrostatic discharges (ESD) of up to 2 kV according to MIL-STD-883C, Method 3015; however, it is advised that precautions be taken to avoid application of any voltage higher than maximum rated voltages to these high-impedance circuits. During storage or handling, the device leads should be shorted together or the device should be placed in conductive foam. In a circuit, unused inputs should always be connected to an appropriate logic voltage level, preferably either V_{CC} or ground. Specific guidelines for handling devices of this type are contained in the publication "Guidelines for Handling Electrostatic-Discharge Sensitive (ESDS) Devices and Assemblies" available from Texas Instruments.

CLOCK CHARACTERISTICS AND TIMING

The TMS320C25 can use either its internal oscillator or an external frequency source for a clock.

internal clock option

The internal oscillator is enabled by connecting a crystal across X1 and X2/CLKIN (see Figure 2). The frequency of CLKOUT1 is one-fourth the crystal fundamental frequency. The crystal should be either fundamental or overtone mode, and parallel resonant, with an effective series resistance of 30 ohms, a power dissipation of 1 mW, and be specified at a load capacitance of 20 pF. Note that overtone crystals require an additional tuned LC circuit (see the application report, *Hardware Interfacing to the TMS320C25*).

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
f_x	Input clock frequency	T_A = 0°C to 70°C	6.7		40.96	MHz
f_{sx}	Serial port frequency	T_A = 0°C to 70°C	0		5,120	kHz
C1, C2		T_A = 0°C to 70°C		10		pF

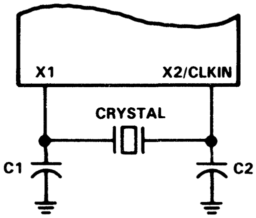


FIGURE 2. INTERNAL CLOCK OPTION

external clock option

An external frequency source can be used by injecting the frequency directly into X2/CLKIN with X1 left unconnected. The external frequency injected must conform to the specifications listed in the following table.

switching characteristics over recommended operating conditions (see Note 3)

PARAMETER		MIN	TYP	MAX	UNIT
$t_{c(C)}$	CLKOUT1/CLKOUT2 cycle time	97.7		597	ns
$t_{d(CIH-C)}$	CLKIN high to CLKOUT1/CLKOUT2/$\overline{\text{STRB}}$ high/low	5		30	ns
$t_{f(C)}$	CLKOUT1/CLKOUT2/$\overline{\text{STRB}}$ fall time			5	ns
$t_{r(C)}$	CLKOUT1/CLKOUT2/$\overline{\text{STRB}}$ rise time			5	ns
$t_{w(CL)}$	CLKOUT1/CLKOUT2 low pulse duration	2Q−8	2Q	2Q+8	ns
$t_{w(CH)}$	CLKOUT1/CLKOUT2 high pulse duration	2Q−8	2Q	2Q+8	ns
$t_{d(C1-C2)}$	CLKOUT1 high to CLKOUT2 low, CLKOUT2 high to CLKOUT1 high, etc.	Q−5	Q	Q+5	ns

NOTE 3: $Q = 1/4t_{c(C)}$.

timing requirements over recommended operating conditions (see Note 3)

		MIN	NOM	MAX	UNIT
$t_{c(CI)}$	CLKIN cycle time	24.4		150	ns
$t_{f(CI)}$	CLKIN fall time			5	ns
$t_{r(CI)}$	CLKIN rise time			5	ns
$t_{w(CIL)}$	CLKIN low pulse duration, $t_{c(CI)}$ = 50 ns (see Note 4)	5		20	ns
$t_{w(CIH)}$	CLKIN high pulse duration, $t_{c(CI)}$ = 50 ns (see Note 4)	5		20	ns
$t_{su(S)}$	$\overline{\text{SYNC}}$ setup time before CKLIN low	5		Q−5	ns
$t_{h(S)}$	$\overline{\text{SYNC}}$ hold time from CLKIN low	8			ns

NOTES: 3. $Q = 1/4t_{c(C)}$.
4. CLKIN duty cycle $[t_{r(CI)} + t_{w(CIH)}]/t_{c(CI)}$ must be within 40-60%.

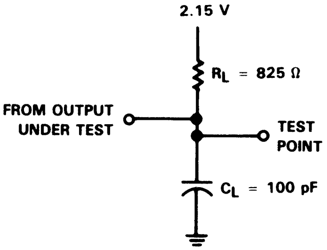
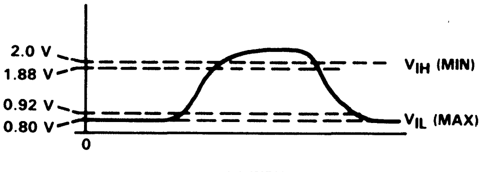
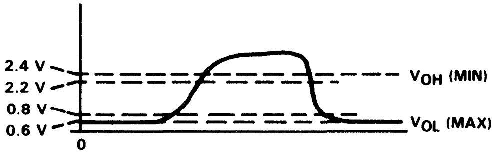


FIGURE 3. TEST LOAD CIRCUIT



(a) INPUT



(b) OUTPUTS

FIGURE 4. VOLTAGE REFERENCE LEVELS

MEMORY AND PERIPHERAL INTERFACE TIMING

switching characteristics over recommended operating conditions (see Note 3)

PARAMETER		MIN	TYP	MAX	UNIT
$t_{d(C1-S)}$	$\overline{\text{STRB}}$ from CLKOUT1 (if $\overline{\text{STRB}}$ is present)	Q−6	Q	Q+6	ns
$t_{d(C2-S)}$	CLKOUT2 to $\overline{\text{STRB}}$ (if $\overline{\text{STRB}}$ is present)	−6	0	6	ns
$t_{su(A)}$	Address setup time before $\overline{\text{STRB}}$ low (see Note 5)	Q−12			ns
$t_{h(A)}$	Address hold time after $\overline{\text{STRB}}$ high (see Note 5)	Q−8			ns
$t_{w(SL)}$	$\overline{\text{STRB}}$ low pulse duration (no wait states, see Note 6)		2Q		ns
$t_{w(SH)}$	$\overline{\text{STRB}}$ high pulse duration (between consecutive cycles, see Note 6)		2Q		ns
$t_{su(D)W}$	Data write setup time before $\overline{\text{STRB}}$ high (no wait states)	2Q−20			ns
$t_{h(D)W}$	Data write hold time from $\overline{\text{STRB}}$ high	Q−10	Q		ns
$t_{en(D)}$	Data bus starts being driven after $\overline{\text{STRB}}$ low (write cycle)	0			ns
$t_{dis(D)}$	Data bus three-state after $\overline{\text{STRB}}$ high (write cycle)		Q	Q+15	ns
$t_{d(MSC)}$	$\overline{\text{MSC}}$ valid from CLKOUT1	−12	0	12	ns

NOTES: 3. Q = 1/4$t_{c(C)}$.

5. A15-A0, $\overline{\text{PS}}$, $\overline{\text{DS}}$, $\overline{\text{IS}}$, R/$\overline{\text{W}}$, and $\overline{\text{BR}}$ timings are all included in timings referenced as "address."
6. Delays between CLKOUT1/CLKOUT2 edges and $\overline{\text{STRB}}$ edges track each other, resulting in $t_{w(SL)}$ and $t_{w(SH)}$ being 2Q with no wait states.

timing requirements over recommended operating conditions (see Note 3)

		MIN	NOM	MAX	UNIT
$t_{a(A)}$	Read data access time from address time (read cycle, see Notes 5 and 7)			3Q−35	ns
$t_{su(D)R}$	Data read setup time before $\overline{\text{STRB}}$ high	23			ns
$t_{h(D)R}$	Data read hold time from $\overline{\text{STRB}}$ high	0			ns
$t_{d(SL-R)}$	READY valid after $\overline{\text{STRB}}$ low (no wait states)			Q−20	ns
$t_{d(C2H-R)}$	READY valid after CLKOUT2 high			Q−20	ns
$t_{h(SL-R)}$	READY hold time after $\overline{\text{STRB}}$ low (no wait states)	Q+3			ns
$t_{h(C2H-R)}$	READY hold after CLKOUT2 high	Q+3			ns
$t_{d(M-R)}$	READY valid after $\overline{\text{MSC}}$ valid			2Q−25	ns
$t_{h(M-R)}$	READY hold time after $\overline{\text{MSC}}$ valid	0			ns

NOTES: 3. Q = 1/4$t_{c(C)}$.

5. A15-A0, $\overline{\text{PS}}$, $\overline{\text{DS}}$, $\overline{\text{IS}}$, R/$\overline{\text{W}}$, and $\overline{\text{BR}}$ timings are all included in timings referenced as "address."
7. Read data access time is defined as $t_{a(A)} = t_{su(A)} + t_{w(SL)} - t_{su(D)R}$.

$\overline{\text{RS}}$, $\overline{\text{INT}}$, $\overline{\text{BIO}}$, and XF TIMING

switching characteristics over recommended operating conditions (see Note 3)

PARAMETER		MIN	TYP	MAX	UNIT
$t_{d(RS)}$	CLKOUT1 low to reset state entered			22	ns
$t_{d(IACK)}$	CLKOUT1 to $\overline{\text{IACK}}$ valid	−6	0	12	ns
$t_{d(XF)}$	XF valid before falling edge of $\overline{\text{STRB}}$	Q−15			ns

NOTES: 3. Q = $1/4t_{c(C)}$.

8. $\overline{\text{RS}}$, $\overline{\text{INT}}$, and $\overline{\text{BIO}}$ are asynchronous inputs and can occur at any time during a clock cycle. However, if the specified setup time is met, the exact sequence shown in the timing diagrams will occur.

timing requirements over recommended operating conditions (see Note 3)

		MIN	NOM	MAX	UNIT
$t_{su(IN)}$	$\overline{\text{INT}}/\overline{\text{BIO}}/\overline{\text{RS}}$ setup before CLKOUT1 high	32			ns
$t_{h(IN)}$	$\overline{\text{INT}}/\overline{\text{BIO}}/\overline{\text{RS}}$ hold after CLKOUT1 high	0			ns
$t_{f(IN)}$	$\overline{\text{INT}}/\overline{\text{BIO}}$ fall time			8	ns
$t_{w(IN)}$	$\overline{\text{INT}}/\overline{\text{BIO}}$ low pulse duration	$t_{c(C)}$			ns
$t_{w(RS)}$	$\overline{\text{RS}}$ low pulse duration	$3t_{c(C)}$			ns

NOTES: 3. Q = $1/4t_{c(C)}$.

8. $\overline{\text{RS}}$, $\overline{\text{INT}}$, and $\overline{\text{BIO}}$ are asynchronous inputs and can occur at any time during a clock cycle. However, if the specified setup time is met, the exact sequence shown in the timing diagrams will occur.

$\overline{\text{HOLD}}$ TIMING

switching characteristics over recommended operating conditions (see Note 3)

PARAMETER		MIN	TYP	MAX	UNIT
$t_{d(C1L\text{-}AL)}$	$\overline{\text{HOLDA}}$ low after CLKOUT1 low	0		10	ns
$t_{dis(AL\text{-}A)}$	$\overline{\text{HOLDA}}$ low to address three-state		0		ns
$t_{dis(C1L\text{-}A)}$	Address three-state after CLKOUT1 low ($\overline{\text{HOLD}}$ mode, see Note 9)			20	ns
$t_{d(HH\text{-}AH)}$	$\overline{\text{HOLD}}$ high to $\overline{\text{HOLDA}}$ high			25	ns
$t_{en(A\text{-}C1L)}$	Address driven before CLKOUT1 low ($\overline{\text{HOLD}}$ mode, see Note 9)			8	ns

NOTES: 3. Q = $1/4t_{c(C)}$.

9. A15-A0, $\overline{\text{PS}}$, $\overline{\text{DS}}$, $\overline{\text{IS}}$, $\overline{\text{STRB}}$, and R/$\overline{\text{W}}$ timings are all included in timings referenced as "address."

timing requirements over recommended operating conditions (see Note 3)

		MIN	NOM	MAX	UNIT
$t_{d(C2H\text{-}H)}$	$\overline{\text{HOLD}}$ valid after CLKOUT2 high			Q−24	ns

NOTE 3: Q = $1/4t_{c(C)}$.

SERIAL PORT TIMING

switching characteristics over recommended operating conditions (see Note 3)

PARAMETER		MIN	TYP	MAX	UNIT
$t_{d(CH\text{-}DX)}$	DX valid after CLKX rising edge (see Note 10)			75	ns
$t_{d(FL\text{-}DX)}$	DX valid after FSX falling edge (TXM = 0, see Note 10)			40	ns
$t_{d(CH\text{-}FS)}$	FSX valid after CLKX rising edge (TXM = 1)			40	ns

NOTES: 3. Q = $1/4t_{c(C)}$.
10. The last occurrence of FSX falling and CLKX rising.

timing requirements over recommended operating conditions (see Note 3)

		MIN	NOM	MAX	UNIT
$t_{c(SCK)}$	Serial port clock (CLKX/CLKR) cycle time	200			ns
$t_{f(SCK)}$	Serial port clock (CLKX/CLKR) fall time			25	ns
$t_{r(SCK)}$	Serial port clock (CLKX/CLKR) rise time			25	ns
$t_{w(SCK)}$	Serial port clock (CLKX/CLKR) low pulse duration (see Note 11)	80			ns
$t_{w(SCK)}$	Serial port clock (CLKX/CLKR) high pulse duration (see Note 11)	80			ns
$t_{su(FS)}$	FSX/FSR setup time before CLKX/CLKR falling edge (TXM = 0)	18			ns
$t_{h(FS)}$	FSX/FSR hold time after CLKX/CLKR falling edge (TXM = 0)	20			ns
$t_{su(DR)}$	DR setup time before CLKR falling edge	10			ns
$t_{h(DR)}$	DR hold time after CLKR falling edge	20			ns

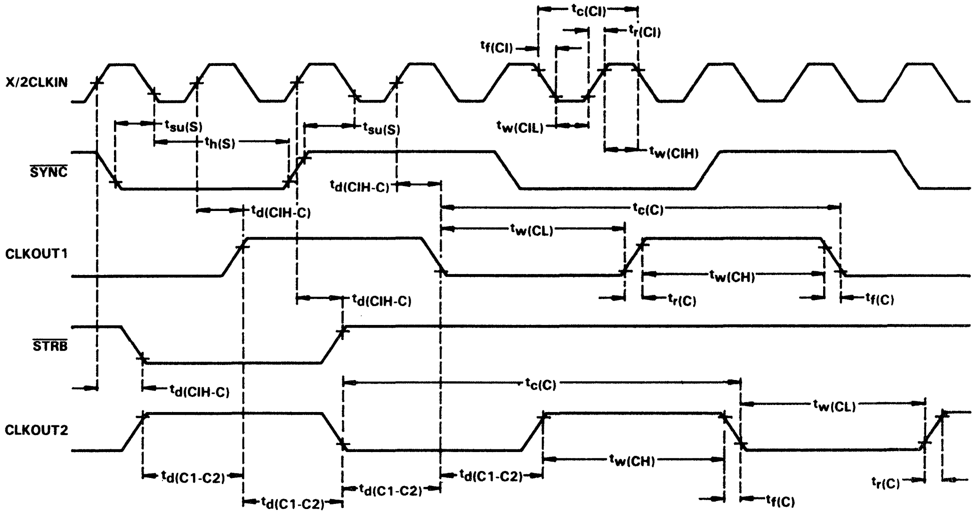
NOTES: 3. Q = $1/4t_{c(C)}$.
11. The duty cycle of the serial port clock must be within 40-60%.

TIMING DIAGRAMS

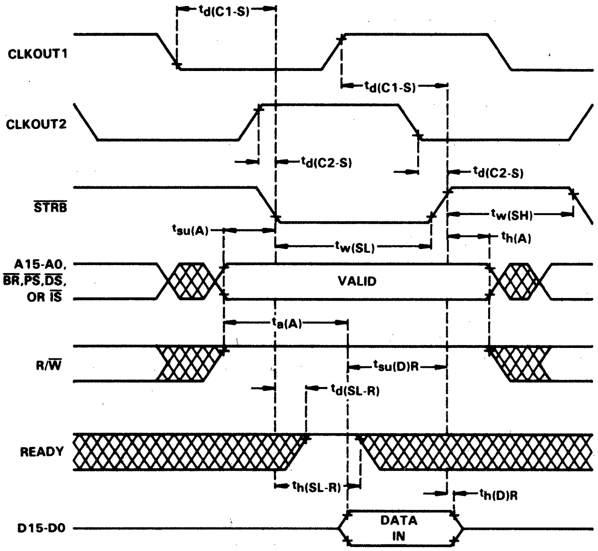
This section contains all the timing diagrams for the TMS320 second-generation devices. Refer to the top corner for the specific device.

Timing measurements are referenced to and from a low voltage of 0.8 volts and a high voltage of 2.0 volts, unless otherwise noted.

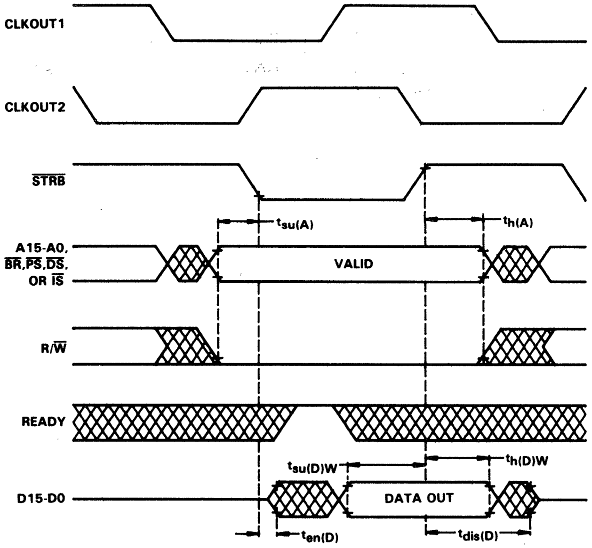
clock timing



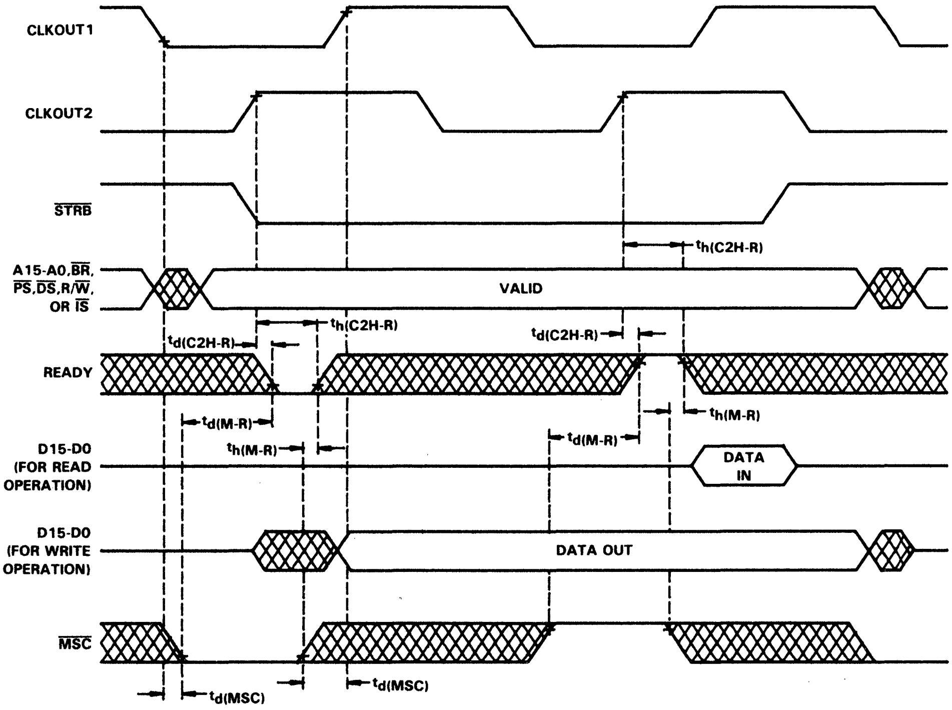
memory read timing



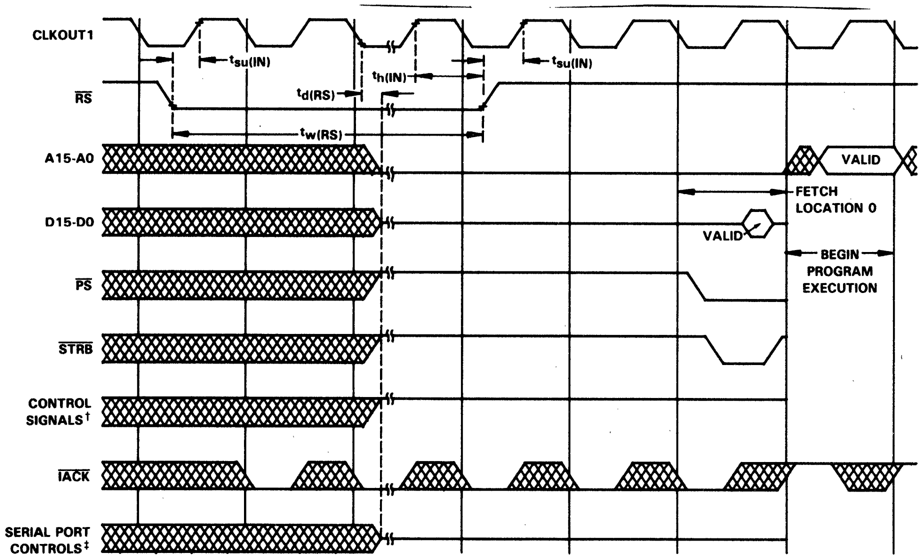
memory write timing



one wait-state memory access timing

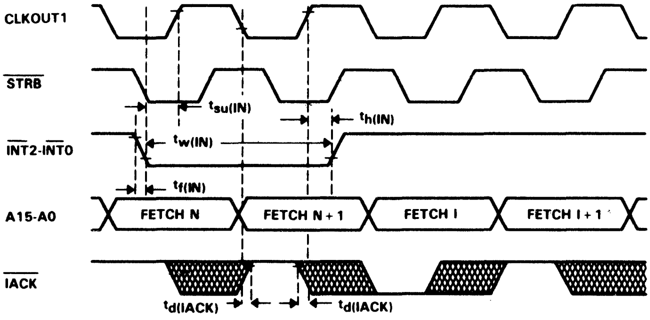


reset timing

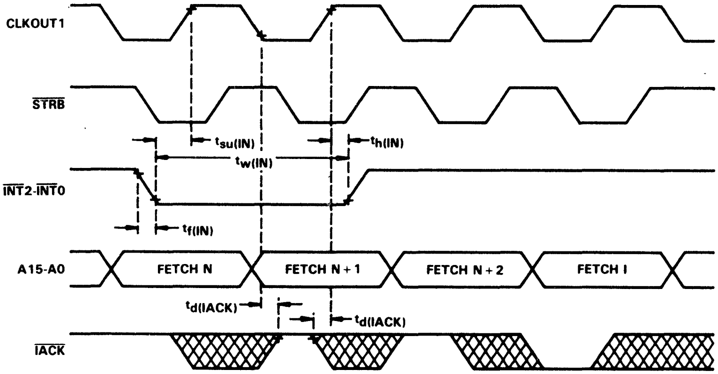


†Control signals are $\overline{DS}$, $\overline{IS}$, R/$\overline{W}$, and XF.
‡Serial port controls are DX and FSX.

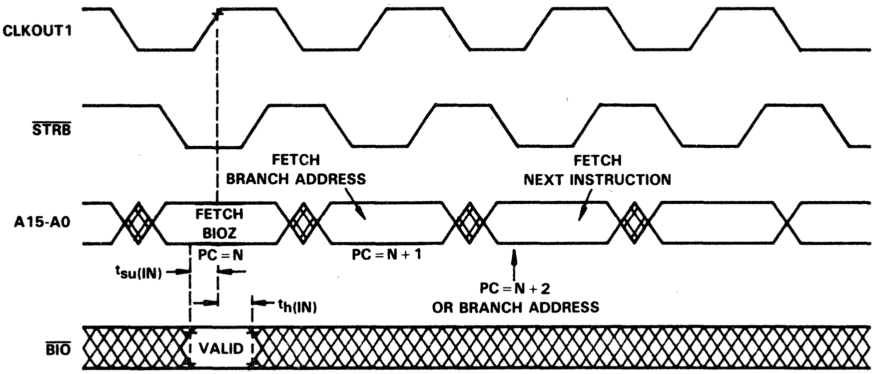
interrupt timing (TMS32020)



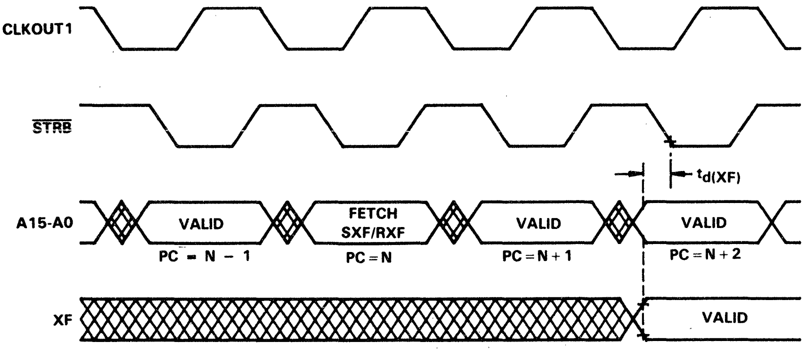
interrupt timing (TMS320C25)



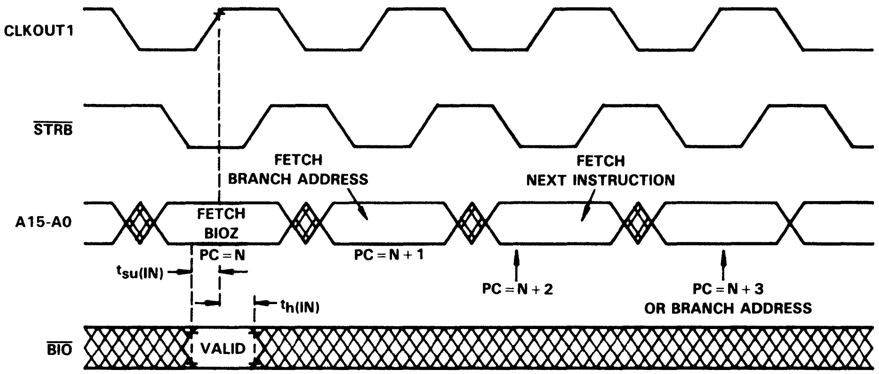
$\overline{\text{BIO}}$ timing



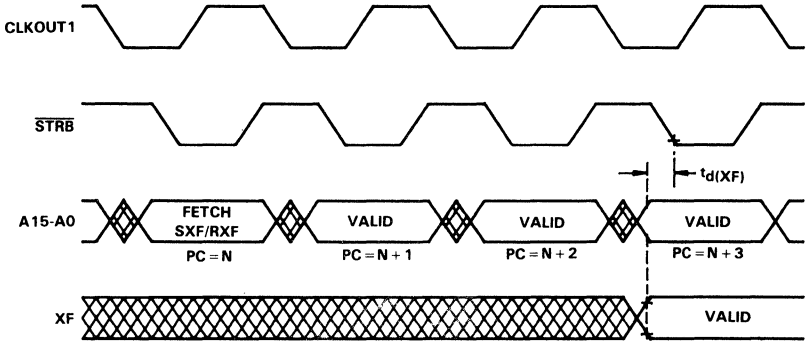
external flag timing



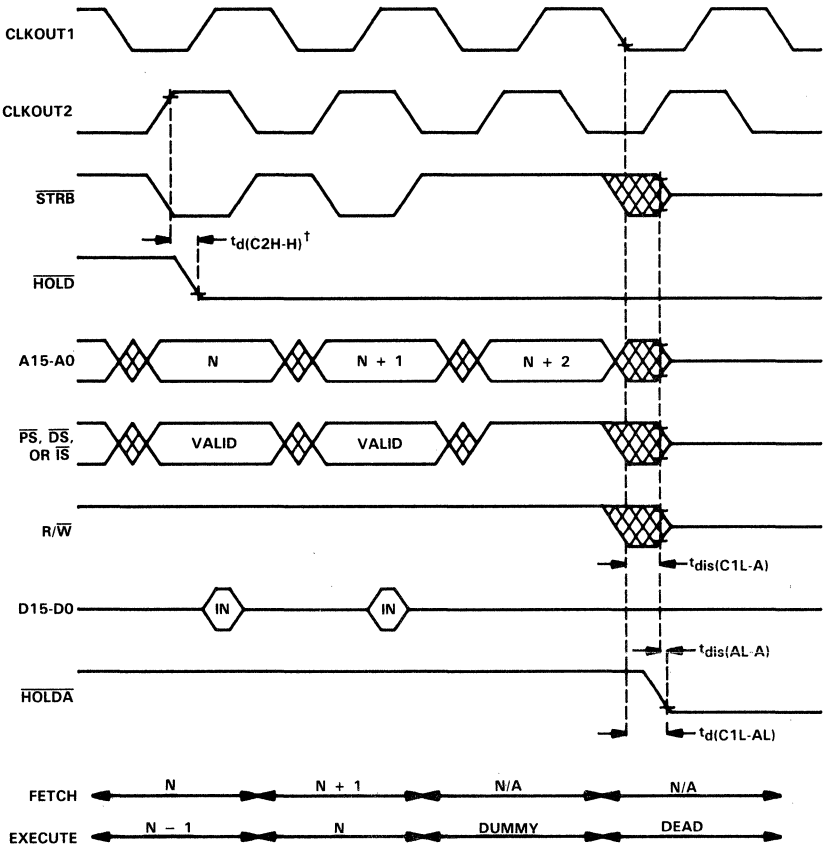
$\overline{\text{BIO}}$ timing



external flag timing

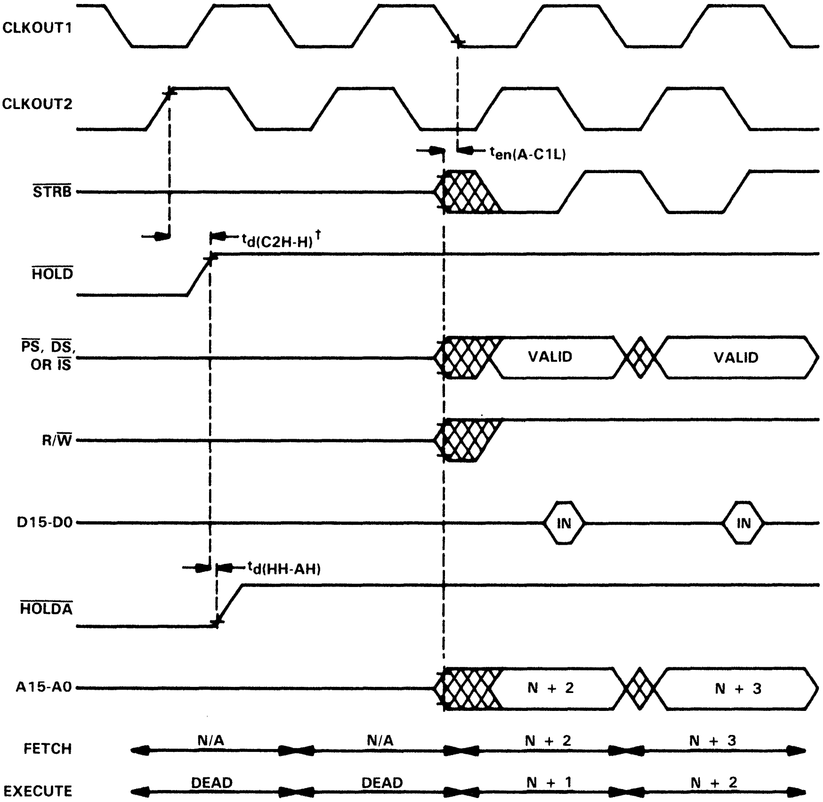


HOLD timing (part A)



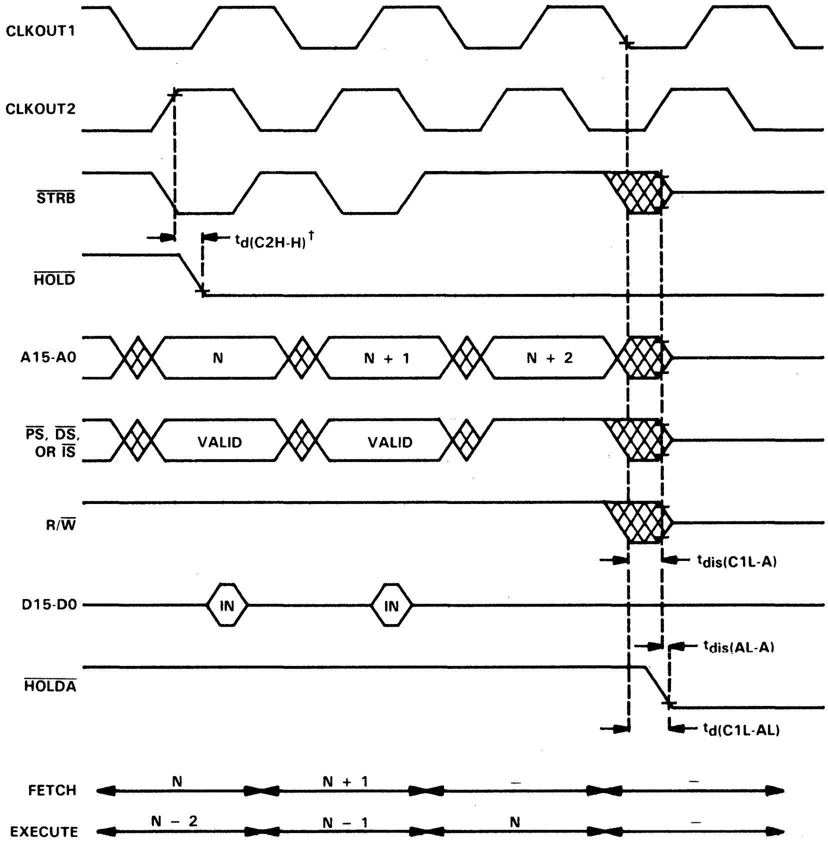
† HOLD is an asynchronous input and can occur at any time during a clock cycle. If the specified timing is met, the exact sequence shown will occur; otherwise, a delay of one CLKOUT2 cycle will occur.

$\overline{\text{HOLD}}$ timing (part B)



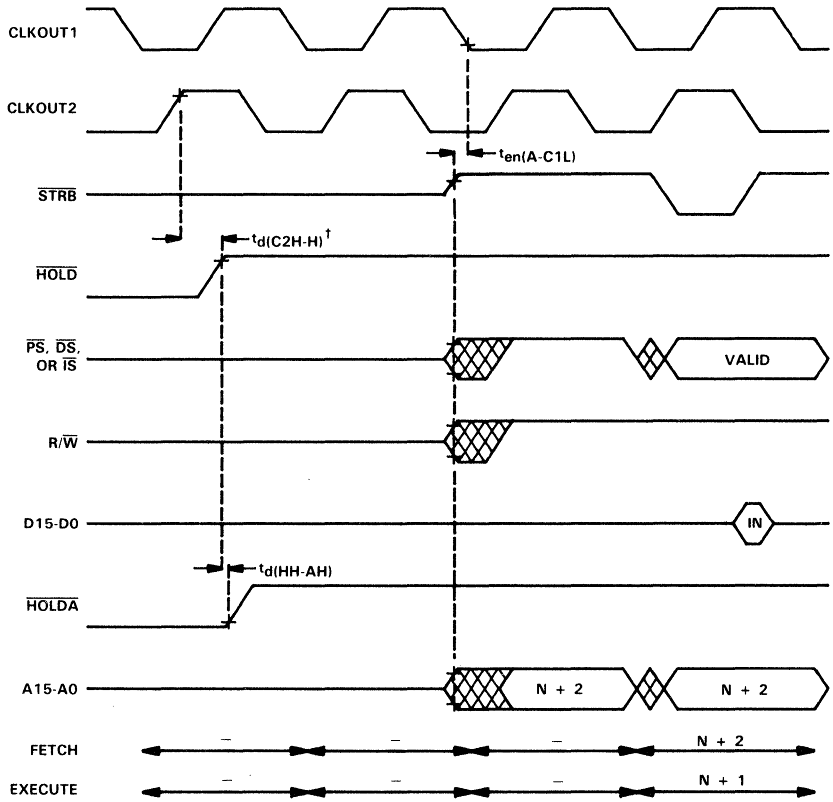
†HOLD is an asynchronous input and can occur at any time during a clock cycle. If the specified timing is met, the exact sequence shown will occur; otherwise, a delay of one CLKOUT2 cycle will occur.

HOLD timing (part A)



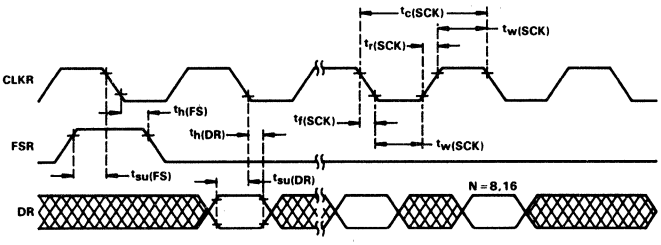
†HOLD is an asynchronous input and can occur at any time during a clock cycle. If the specified timing is met, the exact sequence shown will occur; otherwise, a delay of one CLKOUT2 cycle will occur.

$\overline{\text{HOLD}}$ timing (part B)

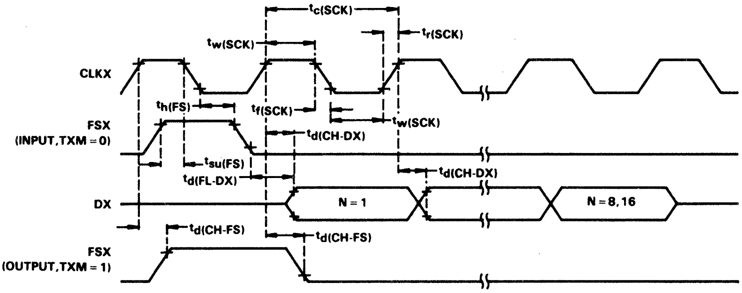


†$\overline{\text{HOLD}}$ is an asynchronous input and can occur at any time during a clock cycle. If the specified timing is met, the exact sequence shown will occur; otherwise, a delay of one CLKOUT2 cycle will occur.

serial port receive timing



serial port transmit timing

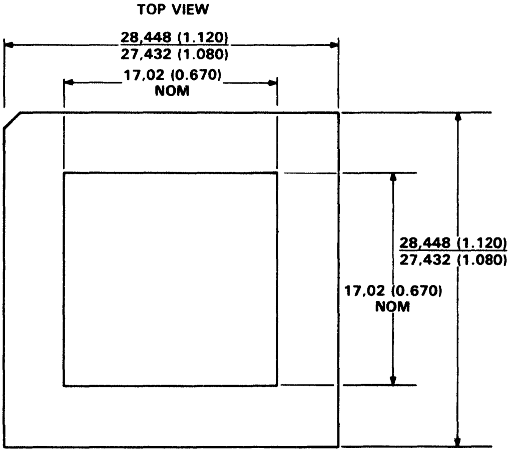


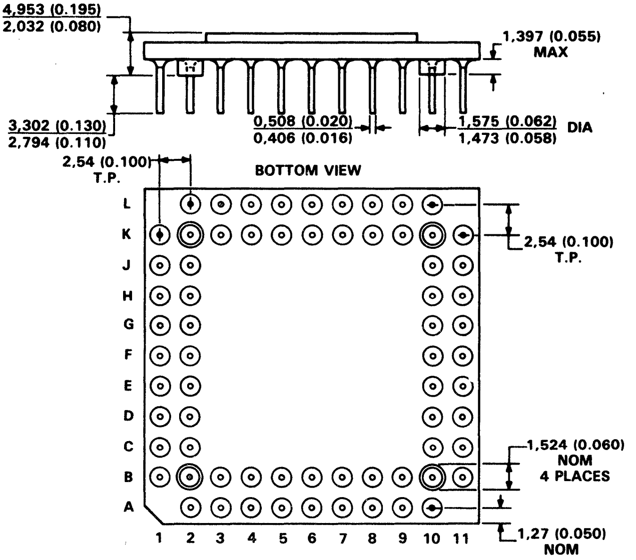
MECHANICAL DATA

68-pin GB grid array ceramic package

THERMAL RESISTANCE CHARACTERISTICS

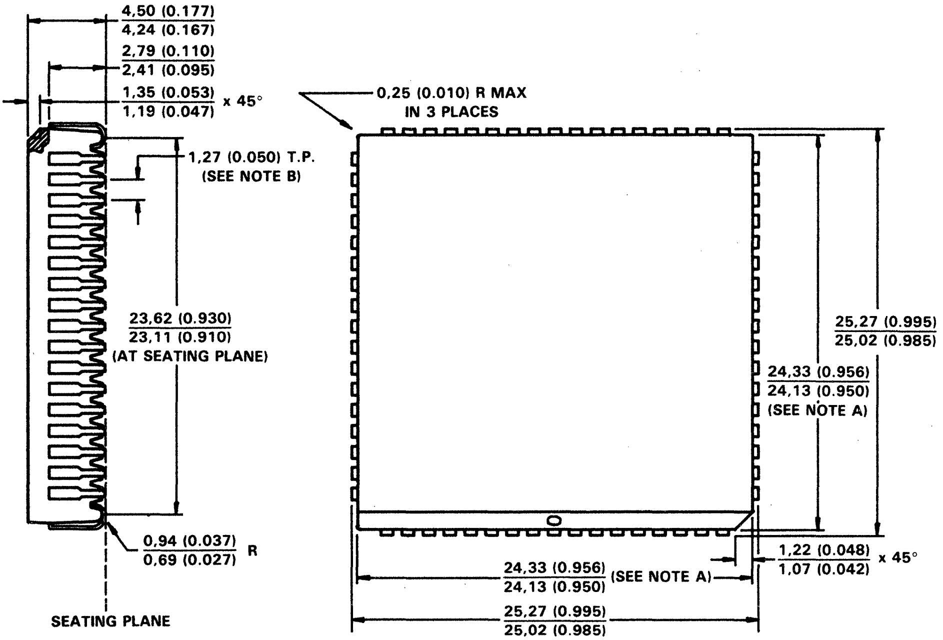
PARAMETER		MAX	UNIT
$R_{\theta JA}$	Junction-to-free-air thermal resistance	36	°C/W
$R_{\theta JC}$	Junction-to-case thermal resistance	6	°C/W

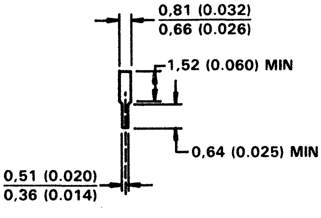




ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

68-pin plastic leaded chip carrier package (TMS320C25 only)





LEAD DETAIL

THERMAL RESISTANCE CHARACTERISTICS

PARAMETER		MAX	UNIT
$R_{\theta JA}$	Junction-to-free-air thermal resistance	46	°C/W
$R_{\theta JC}$	Junction-to-case thermal resistance	11	°C/W

NOTES: A. Centerline of center pin each side is within 0,10 (0.004) of package centerline as determined by this dimension.
B. Location of each pin is within 0,127 (0.005) of true position with respect to center pin on each side.

ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES.

INDEX

accumulator 5
ALU 5
architecture 5-9

block diagram 4, 6

description 1, 3
development support 15
DMA 9
documentation support 16

electrical specifications
  TMS32020 17-22
  TMS320C25 23-28
external interface 7

instruction set 9-14
interrupts 7

key features 4

mechanical data 41, 42
memory 7, 8
microcomputer/microprocessor mode 8
multiplier 5
multiprocessing 9

package types 41, 42
pin nomenclature 2
pinouts 1

repeat feature 10

serial port 9
shifter 5
subroutines 7

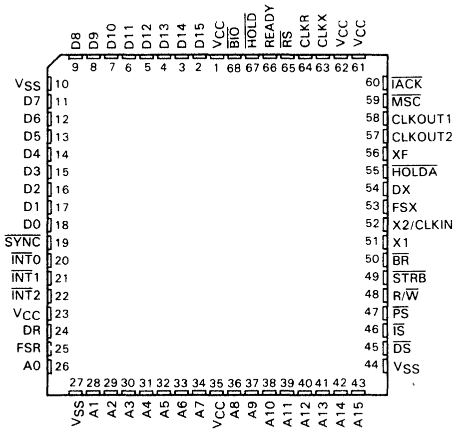
thermal data 41-42
timer 7
timing diagrams 29-40

SMJ32020
DIGITAL SIGNAL PROCESSOR

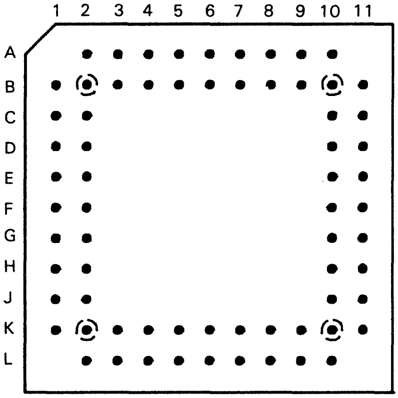
OCTOBER 1987

- S Temperature . . . −55°C to 100°C
- 200-ns Instruction Cycle Time
- 544 Words of Programmable On-Chip Data RAM
- 128K Words of Data/Program Space
- Sixteen Input and Sixteen Output Channels
- 16-Bit Parallel Interface
- Directly Accessible External Data Memory Space
- Global Data Memory Interface
- 16-Bit Instruction and Data Words
- 32-Bit ALU and Accumulator
- Single-Cycle Multiply/Accumulate Instructions
- 0 to 16-Bit Scaling Shifter
- Bit Manipulation and Logical Instructions
- Instruction Set Support for Floating-Point Operations
- Block Moves for Data/Program Management
- Repeat Instructions for Efficient Use of Program Space
- Five Auxiliary Registers and Dedicated Arithmetic Unit for Indirect Addressing
- Serial Port for Direct Codec Interface
- Synchronization Input for Synchronous Multiprocessor Configurations
- Wait States for Communication to Slow Off-Chip Memories/Peripherals
- On-Chip Timer for Control Operations
- Packaging:
  - 68 Pin Ceramic Leaded Chip Carrier
  - 68 Pin Grid Array
- Three External Maskable User Interrupts
- Input Pin Polled by Software Branch Instruction
- Programmable Output Pin for Signaling External Devices
- 2.4-Micron NMOS Technology
- Single 5-V Supply
- On-Chip Clock Generator
- Standard and Class B Processing
  SM Prefix—Standard
  SMJ Prefix—Class B

68-PIN FJ PACKAGE
CERAMIC LEADED CHIP CARRIER
(TOP VIEW)



68-PIN GB
PIN GRID ARRAY CERAMIC PACKAGE[†]
(TOP VIEW)



[†] See Pin Assignments Table (Page 2) and Pin Nomenclature Table (Page 3) for location and description of all pins.

TEXAS INSTRUMENTS
POST OFFICE BOX 1443 • HOUSTON, TEXAS 77001

description

The SMJ32020 Digital Signal Processor is a second-generation member of the SMJ320 group of military VLSI digital signal processors and peripherals. The SMJ32020 supports a wide range of digital signal processing applications, such as tactical communications, guidance, military modems, sonar, signal processing/AI, image processing, speech processing, spectrum analysis, audio processing, digital filtering, high-speed control, graphics, and other computation-intensive applications.

With a 200-ns instruction cycle time and an innovative memory configuration, the SMJ32020 performs operations necessary for many realtime digital signal processing algorithms. Since most instructions require only one cycle, the SMJ32020 is capable of executing five million instructions per second. On-chip data RAM of 544 16-bit words, direct addressing of up to 64K words of external data memory space and 64K words of external program memory space, and multiprocessor interface features for sharing memory minimize unnecessary data transfers to take full advantage of the capabilities of the processor.

PGA/CLCC PIN ASSIGNMENTS

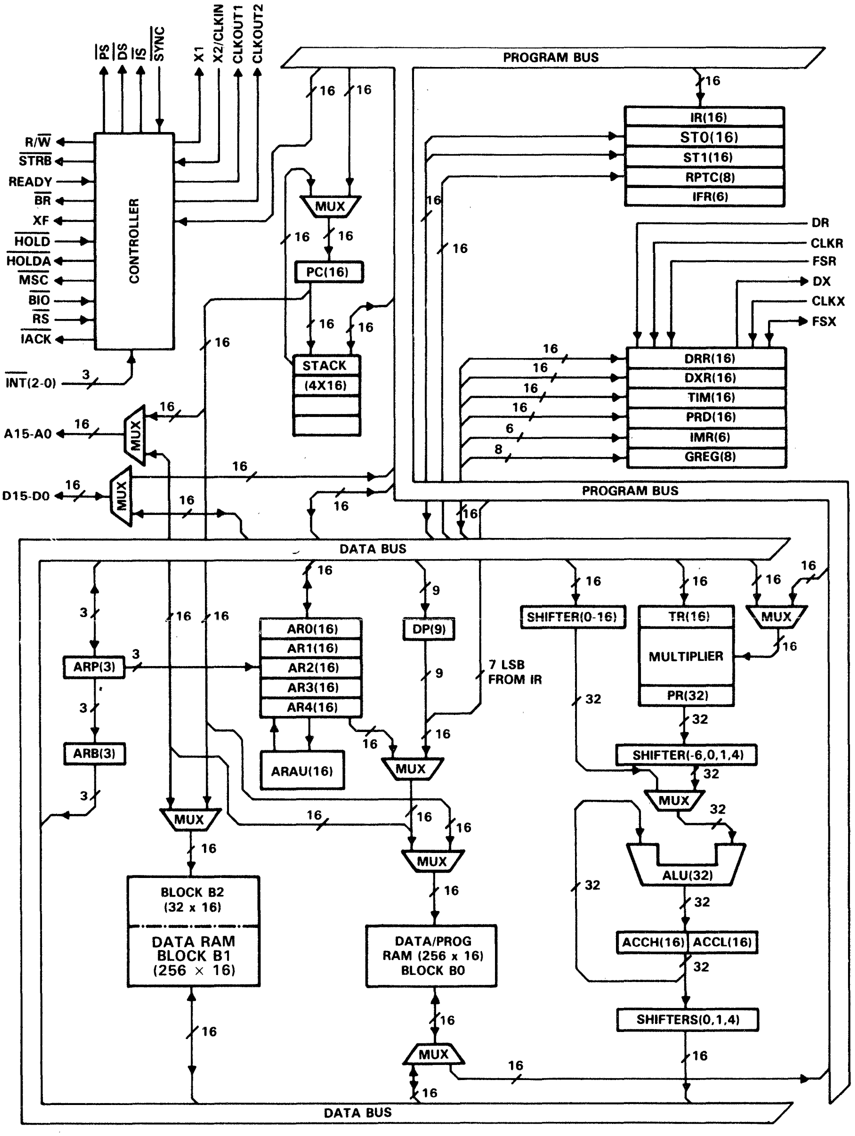
FUNCTION	PIN	FUNCTION	PIN	FUNCTION	PIN	FUNCTION	PIN	FUNCTION	PIN	FUNCTION	PIN
A0	K1/26	A12	K8/40	D2	E1/16	D14	A5/3	INT2	H1/22	V_{CC}	H2/23
A1	K2/28	A13	L9/41	D3	D2/15	D15	B6/2	$\overline{\text{IS}}$	J11/46	V_{CC}	L6/35
A2	L3/29	A14	K9/42	D4	D1/14	DR	J1/24	V_{CC}	A6/1	V_{SS}	B1/10
A3	K3/30	A15	L10/43	D5	C2/13	$\overline{\text{DS}}$	K10/45	$\overline{\text{MSC}}$	C10/59	V_{SS}	K11/44
A4	L4/31	$\overline{\text{BIO}}$	B7/68	D6	C1/12	DX	E11/54	$\overline{\text{PS}}$	J10/47	V_{SS}	L2/27
A5	K4/32	$\overline{\text{BR}}$	G11/50	D7	B2/11	FSR	J2/25	READY	B8/66	XF	D11/56
A6	L5/33	CLKOUT1	C11/58	D8	A2/9	FSX	F10/53	$\overline{\text{RS}}$	A8/65	X1	G10/51
A7	K5/34	CLKOUT2	D10/57	D9	B3/8	$\overline{\text{HOLD}}$	A7/67	R/$\overline{\text{W}}$	H11/48	X2/CLKIN	F11/52
A8	K6/36	CLKR	B9/64	D10	A3/7	$\overline{\text{HOLDA}}$	E10/55	$\overline{\text{STRB}}$	H10/49		
A9	L7/37	CLKX	A9/63	D11	B4/6	$\overline{\text{IACK}}$	B11/60	$\overline{\text{SYNC}}$	F2/19		
A10	K7/38	D0	F1/18	D12	A4/5	$\overline{\text{INT0}}$	G1/20	V_{CC}	A10/61		
A11	L8/39	D1	E2/17	D13	B5/4	$\overline{\text{INT1}}$	G2/21	V_{CC}	B10/62		

PIN NOMENCLATURE

SIGNALS	I/O/Z[†]	DEFINITION
V_{CC}	I	5-V supply pins
V_{SS}	I	Ground pins
X1	O	Output from internal oscillator for crystal
X2/CLKIN	I	Input to internal oscillator from crystal or external clock
CLKOUT1	O	Master clock output (crystal or CLKIN frequency/4)
CLKOUT2	O	A second clock output signal
D15-D0	I/O/Z	16-bit data bus D15 (MSB) through D0 (LSB). Multiplexed between program, data, and I/O spaces.
A15-A0	O/Z	16-bit address bus A15 (MSB) through A0 (LSB)
$\overline{PS}$, $\overline{DS}$, $\overline{IS}$	O/Z	Program, data, and I/O space select signals
R/$\overline{W}$	O/Z	Read/write signal
$\overline{STRB}$	O/Z	Strobe signal
$\overline{RS}$	I	Reset input
$\overline{INT2}$-$\overline{INT0}$	I	External user interrupt inputs
$\overline{MSC}$	O	Microstate complete signal
$\overline{IACK}$	O	Interrupt acknowledge signal
READY	I	Data ready input. Asserted by external logic when using slower devices to indicate that the current bus transaction is complete.
$\overline{BR}$	O	Bus request signal. Asserted when the SMJ32020 requires access to an external global data memory space.
XF	O	External flag output (latched software-programmable signal)
$\overline{HOLD}$	I	Hold input. When asserted, SMJ32020 goes into an idle mode and places the data, address, and control lines in the high impedance state.
$\overline{HOLDA}$	O	Hold acknowledge signal
$\overline{SYNC}$	I	Synchronization input
$\overline{BIO}$	I	Branch control input. Polled by BIOZ instruction.
DR	I	Serial data receive input
CLKR	I	Clock for receive input for serial port
FSR	I	Frame synchronization pulse for receive input
DX	O/Z	Serial data transmit output
CLKX	I	Clock for transmit output for serial port
FSX	I/O/Z	Frame synchronization pulse for transmit. Configurable as either an input or an output.

[†] I/O/Z denotes input/output/high-impedance state.

functional block diagram



architecture

The SMJ32020 architecture is based upon that of the TMS32010, a first-generation member of the TMS320 family. The SMJ32020 increases performance of DSP algorithms through innovative additions to the TMS architecture. SMJ32010 source code is upward-compatible with SMJ32020 source code and can be assembled using the TMS32020 Macro Assembler.

Increased throughput on the SMJ32020 for many DSP applications is accomplished by means of single-cycle multiply/accumulate instructions with a data move option, five auxiliary registers with a dedicated arithmetic unit, and faster I/O necessary for data-intensive signal processing.

The architectural design of the SMJ32020 emphasizes overall speed, communication, and flexibility in processor configuration. Control signals and instructions provide floating-point support, block-memory transfers, communication to slower off-chip devices, and multiprocessing implementations.

Two large on-chip RAM blocks, configurable either as separate program and data spaces or as two contiguous data blocks, provide increased flexibility in system design. Maintaining program memory off-chip allows large address spaces from which large programs of up to 64K words can operate at full speed. Programs can also be downloaded from slow external memory to high-speed on-chip RAM. A 64K-word data memory address space is included to facilitate implementation of DSP algorithms. The VLSI implementation of the SMJ32020 incorporates all of these features as well as many others, such as a hardware timer, serial port, and block data transfer capabilities.

32-bit ALU/accumulator

The 32-bit Arithmetic Logic Unit (ALU) and accumulator perform a wide range of arithmetic and logical instructions, the majority of which execute in a single clock cycle. The ALU executes a variety of branch instructions dependent on the status of the ALU or a single bit in a word. These instructions provide the following capabilities:

- Branch to an address specified by the accumulator
- Normalize fixed-point numbers contained in the accumulator
- Test a specified bit of a word in data memory.

One input to the ALU is always provided from the accumulator, and the other input may be provided from the Product Register (PR) of the multiplier or the input scaling shifter which has fetched data from the RAM on the data bus. After the ALU has performed the arithmetic or logical operations, the result is stored in the accumulator.

The 32-bit accumulator is split into two 16-bit segments for storage in data memory. Additional shifters at the output of the accumulator perform shifts while the data is being transferred to the data bus for storage. The contents of the accumulator remain unchanged.

scaled shifter

The SMJ32020 scaling shifter has a 16-bit input connected to the data bus and a 32-bit output connected to the ALU. The scaling shifter produces a left shift of 0 to 16 bits on the input data, as programmed in the instruction. The LSBs of the output are filled with zeroes, and the MSBs may be either filled with zeroes or sign-extended, depending upon the status programmed into the SXM (sign-extension mode) bit of status register ST0.

16 × 16-bit parallel multiplier

The SMJ32020 has a two's-complement 16 × 16-bit hardware multiplier, which is capable of computing a 32-bit product in a single machine cycle. The multiplier has the following two associated registers:

- A 16-bit Temporary Register (TR) that holds one of the operands for the multiplier, and
- A 32-bit Product Register (PR) that holds the product.

Incorporated in the instruction set are single-cycle multiply/accumulate instructions that allow both operands to be processed simultaneously. The data for these operations resides in the on-chip RAM blocks and can be transferred to the multiplier each cycle via the program and data buses.

Four product shift modes are available at the Product Register (PR) output that are useful when performing multiply/accumulate operations, fractional arithmetic, or justifying fractional products.

timer

The SMJ32020 provides a memory-mapped 16-bit timer for control operations. The on-chip timer (TIM) register is a down counter that is continuously clocked by an internal clock. This clock is derived by dividing the CLKOUT1 frequency by 4. A timer interrupt (TINT) is generated every time the timer decrements to zero. The timer is reloaded with the value contained in the period (PRD) register within the same cycle that it reaches zero so that interrupts may be programmed to occur at regular intervals of 4 × (PRD) cycles of CLKOUT1.

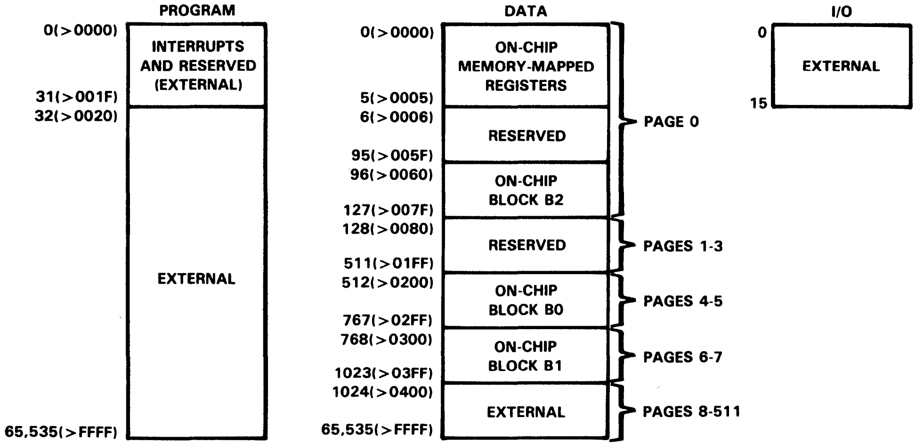
memory control

The SMJ32020 provides a total of 544 16-bit words of on-chip data RAM, divided into three separate blocks (B0, B1, and B2). Of the 544 words, 288 words (blocks B1 and B2) are always data memory, and 256 words (block B0) are programmable as either data or program memory. A data memory size of 544 words allows the SMJ32020 to handle a data array of 512 words (256 words if on-chip RAM is used for program memory), while still leaving 32 locations for intermediate storage. When using block B0 as program memory, instructions can be downloaded from external program memory into on-chip RAM and then executed.

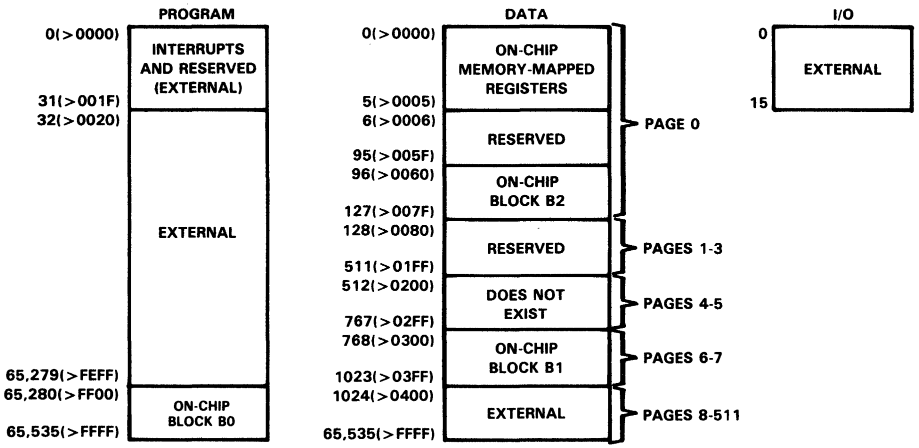
When using on-chip program RAM or high-speed external program memory, the SMJ32020 runs at full speed without wait states. However, the READY line can be used to interface the device to slower, less-expensive external memory. Downloading programs from slow off-chip memory to on-chip program RAM speeds processing while cutting system costs.

The SMJ32020 provides three separate address spaces for program memory, data memory, and I/O. The on-chip memory is mapped into either the 64K-word data memory or program memory space, depending upon the memory configuration. The CNFD (configure block B0 as data memory) and CNFP (configure block B0 as program memory) instructions allow dynamic configuration of the memory maps through software. Regardless of the configuration, the user may still execute from external program memory.

Six registers are mapped into the data memory space: a serial port data receive register, serial port data transmit register, timer register, period register, interrupt mask register, and global memory allocation register.



(a) ADDRESS MAPS AFTER A CNFD INSTRUCTION



(b) ADDRESS MAPS AFTER A CNFP INSTRUCTION

FIGURE 1. MEMORY MAPS

interrupts and subroutines

The SMJ32020 has three external maskable user interrupts INT2-INT0, available for external devices that interrupt the processor. Internal interrupts are generated by the serial port (RINT and XINT), by the timer (TINT), and by the software interrupt (TRAP) instruction. Interrupts are prioritized with reset (RS) having the highest priority and the serial port transmit interrupt (XINT) having the lowest priority. All interrupt locations are on two-word boundaries so that branch instructions can be accommodated in those locations if desired.

A built-in mechanism protects multicycle instructions from interrupts. If an interrupt occurs during a multicycle instruction, the interrupt is not processed until the instruction is completed. This mechanism applies both to instructions that are repeated or become multicycle due to the READY signal.

external interface

The SMJ32020 supports a wide range of system interfacing requirements. Program, data, and I/O address spaces provide interface to memory and I/O, thus maximizing system throughput. I/O design is simplified by having I/O treated the same way as memory. I/O devices are mapped into the I/O address space using the processor's external address and data buses in the same manner as memory-mapped devices. Interface to memory and I/O devices of varying speeds is accomplished by using the READY line. When transitions are made with slower devices, the processor waits until the other device completes its function and signals the processor via the READY line. Then, the SMJ32020 continues execution.

A serial port provides communication with serial devices, such as codecs, serial A/D converters, and other serial systems. The interface signals are compatible with codecs and many other serial devices with a minimum of external hardware. The serial port may also be used for intercommunication between processors in multiprocessing applications.

The serial port has two memory-mapped registers: the data transmit register (DXR) and the data receive register (DRR). Both registers operate in either the byte mode or 16-bit word mode, any may be accessed in the same manner as any other data memory location. Each register has an external clock, a framing synchronization pulse, and associated shift registers. One method of multiprocessing may be implemented by programming one device to transmit while the others are in the receive mode.

multiprocessing

The flexibility of the SMJ32020 allows configurations to satisfy a wide range of system requirements and can be used as follows:

- A standalone processor
- A multiprocessor with devices in parallel
- A slave/host multiprocessor with global memory space
- A peripheral processor interfaced via processor-controlled signals to another device.

For multiprocessing applications, the SMJ32020 has the capability of allocating global data memory space and communicating with that space via the BR (bus request) and READY control signals. Global memory is data memory shared by more than one processor. Global data memory access must be arbitrated. The 8-bit memory-mapped GREG (global memory allocation register) specifies part of the SMJ32020's data memory as global external memory. The contents of the register determine the size of the global memory space. If the current instruction addresses an operand within that space, BR is asserted to request control of the bus. The length of the memory cycle is controlled by the READY line.

The SMJ32020 supports DMA (direct memory access) to its external program/data memory using the HOLD and HOLDA signals. Another processor can take complete control of the SMJ32020's external memory by asserting HOLD low. This causes the SMJ32020 to place its address, data, and control lines in a high-impedance state, and assert HOLDA.

instruction set

The SMJ32020 microprocessor implements a comprehensive instruction set that supports both numeric-intensive signal processing operations as well as general-purpose applications, such as multiprocessing and high-speed control. The SMJ32010 source code is upward-compatible with SMJ32020 source code.

For maximum throughput, the next instruction is prefetched while the current one is being executed. Since the same data lines are used to communicate to external data/program or I/O space, the number of cycles may vary depending upon whether the next data operand fetch is from internal or external program memory. Highest throughput is achieved by maintaining data memory on-chip and using either internal or fast external program memory.

addressing modes

The SMJ32020 instruction set provides three memory addressing modes: direct, indirect, and immediate addressing.

Both direct and indirect addressing can be used to access data memory. In direct addressing, seven bits of the instruction word are concatenated with the nine bits of the data memory page pointer to form the 16-bit data memory address. Indirect addressing accesses data memory through the five auxiliary registers. In immediate addressing, the data is based on a portion of the instruction word(s).

In direct memory addressing, the instruction word contains the lower seven bits of the data memory address. This field is concatenated with the nine bits of the data memory page pointer to form the full 16-bit address. Thus, memory is paged in the direct addressing mode with a total of 512 pages, each page containing 128 words.

Five auxiliary registers (AR0-AR4) provide flexible and powerful indirect addressing. To select a specific auxiliary register, the Auxiliary Register Pointer (ARP) is loaded with either 0, 1, 2, 3, or 4 for AR0 through AR4, respectively.

There are five types of indirect addressing: auto-increment or auto-decrement, post-indexing by either adding or subtracting the contents of AR0, or single indirect addressing with no increment or decrement. All operations are performed on the current auxiliary register in the same cycle as the original instruction, followed by a new ARP value being loaded.

repeat feature

A repeat feature, used with instructions such as multiply/accumulates, block moves, I/O transfers, and table read/writes, allows a single instruction to be performed up to 256 times. The repeat counter (RPTC) is loaded with either a data memory value (RPT instruction) or an immediate value (RPTK instruction). The value of this operand is one less than the number of times that the next instruction is executed. Those instructions that are normally multicycle are pipelined when using the repeat feature, and effectively become single-cycle instructions.

instruction set summary

Table 1 lists the symbols and abbreviations used in Table 2, the instruction set summary. Table 2 consists primarily of single-cycle, single-word instructions. Infrequently used branch, I/O, and CALL instructions are multicycle. The instruction set summary is arranged according to function and alphabetized within each functional grouping. The symbol (†) indicates those instructions that are not included in the SMJ32010 instruction set.

TABLE 1. INSTRUCTION SYMBOLS

SYMBOL	MEANING
B	4-bit field specifying a bit code
CM	2-bit field specifying compare mode
D	Data memory address field
FO	Format status bit
I	Addressing mode bit
K	Immediate operand field
PA	Port address (PA0 through PA15 are predefined assembler symbols equal to 0 through 15, respectively).
PM	2-bit field specifying P register output shift code
R	3-bit operand field specifying auxiliary register
S	4-bit left-shift code
X	3-bit accumulator left-shift field

TABLE 2. INSTRUCTION SET SUMMARY

ACCUMULATOR MEMORY REFERENCE INSTRUCTIONS																		
MNEMONIC	**DESCRIPTION**	**NO. WORDS**	**INSTRUCTION BIT CODE**															
			15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
ABS	Absolute value of accumulator	1	1	1	0	0	1	1	1	0	0	0	0	1	1	0	1	1
ADD	Add to accumulator with shift	1	0	0	0	0	◄—S—►				I	◄—D—►						
ADDH	Add to high accumulator	1	0	1	0	0	1	0	0	0	I	◄—D—►						
ADDS	Add to low accumulator with sign extension suppressed	1	0	1	0	0	1	0	0	1	I	◄—D—►						
ADDT†	Add to accumulator with shift specified by T register	1	0	1	0	0	1	0	1	0	I	◄—D—►						
ADLK†	Add to accumulator long immediate with shift	2	1	1	0	1	◄—S—►				0	0	0	0	0	0	1	0
AND	AND with accumulator	1	0	1	0	0	1	1	1	0	I	◄—D—►						
ANDK†	AND immediate with accumulator with shift	2	1	1	0	1	◄—S—►				0	0	0	0	0	1	0	0
CMPL†	Complement accumulator	1	1	1	0	0	1	1	1	0	0	0	1	0	0	1	1	1
LAC	Load accumulator with shift	1	0	0	1	0	◄—S—►				I	◄—D—►						
LACK	Load accumulator immediate short	1	1	1	0	0	1	0	1	0	◄—K—►							
LACT†	Load accumulator with shift specified by T register	1	0	1	0	0	0	0	1	0	I	◄—D—►						
LALK†	Load accumulator long immediate with shift	2	1	1	0	1	◄—S—►				0	0	0	0	0	0	0	1
NEG†	Negate accumulator	1	1	1	0	0	1	1	1	0	0	0	1	0	0	0	1	1
NORM†	Normalize contents of accumulator	1	1	1	0	0	1	1	1	0	1	0	1	0	0	0	1	0
OR	OR with accumulator	1	0	1	0	0	1	1	0	1	I	◄—D—►						
ORK†	OR immediate with accumulator with shift	2	1	1	0	1	◄—S—►				0	0	0	0	0	1	0	1
SACH	Store high accumulator with shift	1	0	1	1	0	1	◄—X—►			I	◄—D—►						
SACL	Store low accumulator with shift	1	0	1	1	0	0	◄—X—►			I	◄—D—►						
SBLK†	Subtract from accumulator long immediate with shift	2	1	1	0	1	◄—S—►				0	0	0	0	0	0	1	1
SFL†	Shift accumulator left	1	1	1	0	0	1	1	1	0	0	0	0	1	1	0	0	0
SFR†	Shift accumulator right	1	1	1	0	0	1	1	1	0	0	0	0	1	1	0	0	1
SUB	Subtract from accumulator with shift	1	0	0	0	1	◄—S—►				I	◄—D—►						
SUBC	Conditional subtract	1	0	1	0	0	0	1	1	1	I	◄—D—►						
SUBH	Subtract from high accumulator	1	0	1	0	0	0	1	0	0	I	◄—D—►						
SUBS	Subtract from low accumulator with sign extension suppressed	1	0	1	0	0	0	1	0	1	I	◄—D—►						
SUBT†	Subtract from accumulator with shift specified by T register	1	0	1	0	0	0	1	1	0	I	◄—D—►						
XOR	Exclusive-OR with accumulator	1	0	1	0	0	1	1	0	0	I	◄—D—►						
XORK†	Exclusive-OR immediate with accumulator with shift	2	1	1	0	1	◄—S—►				0	0	0	0	0	1	1	0
ZAC	Zero accumulator	1	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0
ZALH	Zero low accumulator and load high accumulator	1	0	1	0	0	0	0	0	0	I	◄—D—►						
ZALS	Zero accumulator and load low accumulator with sign extension suppressed	1	0	1	0	0	0	0	0	1	I	◄—D—►						

†These instructions not included in the SMJ32010 instruction set.

TABLE 2. INSTRUCTION SET SUMMARY (CONTINUED)

AUXILIARY REGISTERS AND DATA PAGE POINTER INSTRUCTIONS

MNEMONIC	DESCRIPTION	NO. WORDS	INSTRUCTION BIT CODE															
			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CMPR†	Compare auxiliary register with auxiliary register AR0	1	1	1	0	0	1	1	1	0	0	1	0	1	0	0	◄CM►	
LAR	Load auxiliary register	1	0	0	1	1	0	◄R►			I	◄D►						
LARK	Load auxiliary register immediate short	1	1	1	0	0	0	◄R►			◄K►							
LARP	Load auxiliary register pointer	1	0	1	0	1	0	1	0	1	1	0	0	0	1	◄R►		
LDP	Load data memory page pointer	1	0	1	0	1	0	0	1	0	I	◄D►						
LDPK	Load data memory page pointer immediate	1	1	1	0	0	1	0	0	◄K►								
LRLK†	Load auxiliary register long immediate	2	1	1	0	1	0	◄R►			0	0	0	0	0	0	0	0
MAR	Modify auxiliary register	1	0	1	0	1	0	1	0	1	I	◄D►						
SAR	Store auxiliary register	1	0	1	1	1	0	◄R►			I	◄D►						

T REGISTER, P REGISTER, AND MULTIPLY INSTRUCTIONS

MNEMONIC	DESCRIPTION	NO. WORDS	INSTRUCTION BIT CODE															
			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
APAC	Add P register to accumulator	1	1	1	0	0	1	1	1	0	0	0	0	1	0	1	0	1
LPH†	Load high P register	1	0	1	0	1	0	0	1	1	I	◄D►						
LT	Load T register	1	0	0	1	1	1	1	0	0	I	◄D►						
LTA	Load T register and accumulate previous product	1	0	0	1	1	1	1	0	1	I	◄D►						
LTD	Load T register, accumulate previous product, and move data	1	0	0	1	1	1	1	1	1	I	◄D►						
LTP†	Load T register and store P register in accumulator	1	0	0	1	1	1	1	1	0	I	◄D►						
LTS†	Load T register and subtract previous product	1	0	1	0	1	1	0	1	1	I	◄D►						
MAC†	Multiply and accumulate	2	0	1	0	1	1	1	0	1	I	◄D►						
MACD†	Multiply and accumulate with data move	2	0	1	0	1	1	1	0	0	I	◄D►						
MPY	Multiply (with T register, store product in P register)	1	0	0	1	1	1	0	0	0	I	◄D►						
MPYK	Multiply immediate	1	1	0	1	◄K►												
PAC	Load accumulator with P register	1	1	1	0	0	1	1	1	0	0	0	0	1	0	1	0	0
SPAC	Subtract P register from accumulator	1	1	1	0	0	1	1	1	0	0	0	0	1	0	1	1	0
SPM†	Set P register output shift mode	1	1	1	0	0	1	1	1	0	0	0	0	0	1	0	◄PM►	
SQRA†	Square and accumulate	1	0	0	1	1	1	0	0	1	I	◄D►						
SQRS†	Square and subtract previous product	1	0	1	0	1	1	0	1	0	I	◄D►						

†These instructions not included in the SMJ32010 instruction set.

TABLE 2. INSTRUCTION SET SUMMARY (CONTINUED)

BRANCH CALL INSTRUCTIONS

MNEMONIC	DESCRIPTION	NO. WORDS	INSTRUCTION BIT CODE															
			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B	Branch unconditionally	2	1	1	1	1	1	1	1	1	1	←D→						
BACC[†]	Branch to address specified by accumulator	1	1	1	0	0	1	1	1	0	0	0	1	0	0	1	0	1
BANZ	Branch on auxiliary register not zero	2	1	1	1	1	1	0	1	1	1	←D→						
BBNZ[†]	Branch if TC bit ≠ 0	2	1	1	1	1	1	0	0	1	1	←D→						
BBZ[†]	Branch if TC bit = 0	2	1	1	1	1	1	0	0	0	1	←D→						
BGEZ	Branch if accumulator ≥ 0	2	1	1	1	1	0	1	0	0	1	←D→						
BGZ	Branch if accumulator > 0	2	1	1	1	1	0	0	0	1	1	←D→						
BIOZ	Branch on I/O status = 0	2	1	1	1	1	1	0	1	0	1	←D→						
BLEZ	Branch if accumulator ≤ 0	2	1	1	1	1	0	0	1	0	1	←D→						
BLZ	Branch if accumulator < 0	2	1	1	1	1	0	0	1	1	1	←D→						
BNV[†]	Branch if no overflow	2	1	1	1	1	0	1	1	1	1	←D→						
BNZ	Branch if accumulator ≠ 0	2	1	1	1	1	0	1	0	1	1	←D→						
BV	Branch on overflow	2	1	1	1	1	0	0	0	0	1	←D→						
BZ	Branch if accumulator = 0	2	1	1	1	1	0	1	1	0	1	←D→						
CALA	Call subroutine indirect	1	1	1	0	0	1	1	1	0	0	0	1	0	0	1	0	0
CALL	Call subroutine	2	1	1	1	1	1	1	1	0	1	←D→						
RET	Return from subroutine	1	1	1	0	0	1	1	1	0	0	0	1	0	0	1	1	0

CONTROL INSTRUCTIONS

MNEMONIC	DESCRIPTION	NO. WORDS	INSTRUCTION BIT CODE															
			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BIT[†]	Test bit	1	1	0	0	1	←B→				I	←D→						
BITT[†]	Test bit specified by T register	1	0	1	0	1	0	1	1	1	I	←D→						
CNFD[†]	Configure block as data memory	1	1	1	0	0	1	1	1	0	0	0	0	0	0	1	0	0
CNFP[†]	Configure block as program memory	1	1	1	0	0	1	1	1	0	0	0	0	0	0	1	0	1
DINT	Disable interrupt	1	1	1	0	0	1	1	1	0	0	0	0	0	0	0	0	1
EINT	Enable interrupt	1	1	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0
IDLE[†]	Idle until interrupt	1	1	1	0	0	1	1	1	0	0	0	0	1	1	1	1	1
LST	Load status register ST0	1	0	1	0	1	0	0	0	0	I	←D→						
LST1[†]	Load status register ST1	1	0	1	0	1	0	0	0	1	I	←D→						
NOP	No operation	1	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0
POP	Pop top of stack to low accumulator	1	1	1	0	0	1	1	1	0	0	0	0	1	1	1	0	1
POPD[†]	Pop top of stack to data memory	1	0	1	1	1	1	0	1	0	I	←D→						
PSHD[†]	Push data memory value onto stack	1	0	1	0	1	0	1	0	0	I	←D→						
PUSH	Push low accumulator onto stack	1	1	1	0	0	1	1	1	0	0	0	0	1	1	1	0	0
ROVM	Reset overflow mode	1	1	1	0	0	1	1	1	0	0	0	0	0	0	0	1	0
RPT[†]	Repeat instruction as specified by data memory value	1	0	1	0	0	1	0	1	1	I	←D→						
RPTK[†]	Repeat instruction as specified by immediate value	1	1	1	0	0	1	0	1	1	←K→							
RSXM[†]	Reset sign-extension mode	1	1	1	0	0	1	1	1	0	0	0	0	0	0	1	1	0
SOVM	Set overflow mode	1	1	1	0	0	1	1	1	0	0	0	0	0	0	0	1	1
SST	Store status register ST0	1	0	1	1	1	1	0	0	0	I	←D→						
SST1[†]	Store status register ST1	1	0	1	1	1	1	0	0	1	I	←D→						
SSXM[†]	Set sign-extension mode	1	1	1	0	0	1	1	1	0	0	0	0	0	0	1	1	1
TRAP[†]	Software interrupt	1	1	1	0	0	1	1	1	0	0	0	0	1	1	1	1	0

[†] These instructions not included in the SMJ32010 instruction set.

TABLE 2. INSTRUCTION SET SUMMARY (CONCLUDED)

I/O AND DATA MEMORY OPERATIONS

MNEMONIC	DESCRIPTION	NO. WORDS	INSTRUCTION BIT CODE															
			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BLKD†	Block move from data memory to data memory	2	1	1	1	1	1	1	0	1	I	←D→						
BLKP†	Block move from program memory to data memory	2	1	1	1	1	1	1	0	0	I	←D→						
DMOV	Data move in data memory	1	0	1	0	1	0	1	1	0	I	←D→						
FORT†	Format serial port registers	1	1	1	0	0	1	1	1	0	0	0	0	0	1	1	1	FO
IN	Input data from port	1	1	0	0	0	←PA→				I	←D→						
OUT	Output data to port	1	1	1	1	0	←PA→				I	←D→						
RTXM†	Reset serial port transmit mode	1	1	1	0	0	1	1	1	0	0	0	1	0	0	0	0	0
RXF†	Reset external flag	1	1	1	0	0	1	1	1	0	0	0	0	0	1	1	0	0
STXM†	Set serial port transmit mode	1	1	1	0	0	1	1	1	0	0	0	1	0	0	0	0	1
SXF†	Set external flag	1	1	1	0	0	1	1	1	0	0	0	0	0	1	1	0	1
TBLR	Table read	1	0	1	0	1	1	0	0	0	I	←D→						
TBLW	Table write	1	0	1	0	1	1	0	0	1	I	←D→						

†These instructions not included in the SMJ32010 instruction set.

development support

Texas Instruments offers an extensive line of development support products to assist the user in all aspects of SMJ320 second-generation-based design and development. These products range from development and application software to complete hardware development and evaluation systems such as the XDS/22. Table 4 lists the development support products for the second-generation SMJ320 devices.

System development begins with the use of the SoftWare Development System (SWDS) or Emulator (XDS). These tools allow the designer to evaluate the processor's performance, benchmark time-critical code, and determine the feasibility of using a SMJ320 device to implement a specific algorithm.

Software and hardware can be developed in parallel by using the macro assembler/linker, simulator, and SoftWare Development System for software development and the XDS for hardware development. The assembler/linker translates the system's assembly source program into an object module that can be executed by the simulator, XDS, or SWDS. The XDS provides realtime in-circuit emulation and is a powerful tool for debugging and integrating software and hardware modules.

Additional support for the SMJ320 products consists of extensive documentation and three-day DSP design workshops offered by the TI Regional Technology Centers (RTCs). The workshops provide hands-on experience with the TMS320 development tools. Refer to the *TMS320 Family Development Support Reference Guide* for further information about TMS320 development support products and DSP workshops. When technical questions arise regarding the TMS320, contact the Texas Instruments TMS320 Hotline, (713) 274-2320.

TABLE 4. TMS320 SECOND-GENERATION SOFTWARE AND HARDWARE SUPPORT

SOFTWARE TOOLS	PART NUMBER
Macro Assembler/Linker	
VAX VMS	TMDS3242210-08
TI/IBM MS/PC-DOS	TMDS3242810-02
Simulator	
VAX VMS	TMDS3242211-08
TI/IBM MS/PC-DOS	TMDS3242811-02
SoftWare Development System (SWDS)	TMDS3268821
Digital Filter Design Package (DFDP)	
IBM PC-DOS	DFDP-IBM002
DSP Software Library	
VAX VMS	TMDC3240212-18
TI/IBM MS/PC-DOS	TMDC3240812-12
HARDWARE TOOLS	**PART NUMBER**
Analog Interface Board (AIB)	RTC/EVM320C-06
XDS/22 Emulator	TMDS3262221
XDS/22 Upgrade	
Customer Upgrade	TMDS3282226
TMS320 Design Kit	TMS320DDK

documentation support

Extensive documentation supports the second-generation TMS320 devices from product announcement through applications development. The types of documentation include data sheets with design specifications, complete user's guides, and 750 pages of application reports published in the book, *Digital Signal Processing Applications with the TMS320 Family.* An application report, *Hardware Interfacing to the TMS320C25,* is available for that device.

A series of DSP textbooks is being published by Prentice-Hall and John Wiley & Sons to support digital signal processing research and education. The TMS320 newsletter, *Details on Signal Processing,* is published quarterly and distributed to update SMJ320 customers on product information. The TMS320 DSP bulletin board service provides access to large amounts of information pertaining to the SMJ320 family.

Refer to the *TMS320 Family Development Support Reference Guide* for further information about SMJ320 documentation. To receive copies of second-generation TMS320 literature, call the Customer Response Center at 1-800-232-3200.

absolute maximum ratings over specified temperature range (unless otherwise noted)[†]

Supply voltage range, V_{CC}[‡]	−0.3 V to 7 V
Input voltage range	−0.3 V to 7 V
Output voltage range	−0.3 V to 7 V
Continuous power dissipation	2.0 W
Minimum operating free-air temperature	−55 °C
Maximum operating case temperature	100 °C
Storage temperature range	−55 °C to 150 °C

[†] Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

[‡] All voltage values are with respect to V_{SS}.

recommended operating conditions

			MIN	NOM	MAX	UNIT
V_{CC}	Supply voltage		4.5	5	5.5	V
V_{SS}	Supply voltage			0		V
V_{IH}	High-level input voltage	All inputs except CLKIN	2.2			V
		CLKIN	2.6			
V_{IL}	Low-level input voltage	All inputs except CLKIN			0.8	V
		CLKIN			0.8	
I_{OH}	High-level output current				300	µA
I_{OL}	Low-level output current				2	mA
T_C	Operating case temperature				100	°C
T_A	Operating free-air temperature		−55			°C

electrical characteristics over specified free-air temperature range (unless otherwise noted)

PARAMETER		TEST CONDITIONS	MIN	TYP[§]	MAX	UNIT
V_{OH}	High-level output voltage	V_{CC} = MIN, I_{OH} = MAX	2.4	3		V
V_{OL}	Low-level output voltage	V_{CC} = MIN, I_{OL} = MAX		0.3	0.6	V
I_Z	Three-state current	V_{CC} = MAX	−20		20	µA
I_I	Input current	V_I = V_{SS} to V_{CC}	−10		10	µA
I_{CC}	Supply current	T_A = −55 °C, V_{CC} = MAX, f_x = MAX			400	mA
		T_A = 25 °C, V_{CC} = MAX, f_x = MAX		250		
		T_C = 100 °C, V_{CC} = MAX, f_x = MAX			285	
C_I	Input capacitance			15		pF
C_O	Output capacitance			15		pF

[§] All typical values are at V_{CC} = 5 V, T_A = 25 °C.

Caution. This device contains circuits to protect its inputs and outputs against damage due to high static voltages or electrostatic fields. These circuits have been qualified to protect this device against electrostatic discharges (ESD) of up to 2 kV according to MIL-STD-883C, Method 3015; however, it is advised that precautions be taken to avoid application of any voltage higher than maximum rated voltages to these high-impedance circuits. During storage or handling, the device leads should be shorted together or the device should be placed in conductive foam. In a circuit, unused inputs should always be connected to an appropriate logic voltage level, preferably either V_{CC} or ground. Specific guidelines for handling devices of this type are contained in the publication "Guidelines for Handling Electrostatic-Discharge Sensitive (ESDS) Devices and Assemblies" available from Texas Instruments.

CLOCK CHARACTERISTICS AND TIMING

The SMJ32020 can use either its internal oscillator or an external frequency source for a clock.

internal clock option

The internal oscillator is enabled by connecting a crystal across X1 and X2/CLKIN (see Figure 2). The frequency of CLKOUT1 is one-fourth the crystal fundamental frequency. The crystal should be fundamental mode, and parallel resonant, with an effective series resistance of 30 ohms, a power dissipation of 1 mW, and be specified at a load capacitance of 20 pF.

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
f_x	Input clock frequency	T_A = −55°C MIN T_C = 100°C MAX	6.7		20	MHz
f_{sx}	Serial port frequency		417.5		2500	kHz
C1, C2				10		pF

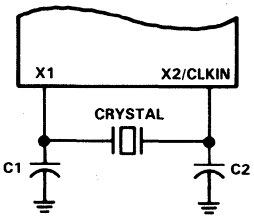


FIGURE 2. INTERNAL CLOCK OPTION

external clock option

An external frequency source can be used by injecting the frequency directly into X2/CLKIN with X1 left unconnected. The external frequency injected must conform to the specifications listed in the following table.

switching characteristics over recommended operating conditions (see Note 1)

PARAMETER		MIN	TYP	MAX	UNIT
$t_{c(C)}$	CLKOUT1/CLKOUT2 cycle time	200		600	ns
$t_{d(CIH-C)}$	CLKIN high to CLKOUT1/CLKOUT2/$\overline{\text{STRB}}$ high/low	15		50	ns
$t_{f(C)}$	CLKOUT1/CLKOUT2/$\overline{\text{STRB}}$ fall time			10	ns
$t_{r(C)}$	CLKOUT1/CLKOUT2/$\overline{\text{STRB}}$ rise time			10	ns
$t_{w(CL)}$	CLKOUT1/CLKOUT2 low pulse duration	2Q−15	2Q	2Q+15	ns
$t_{w(CH)}$	CLKOUT1/CLKOUT2 high pulse duration	2Q−15	2Q	2Q+15	ns
$t_{d(C1-C2)}$	CLKOUT1 high to CLKOUT2 low, CLKOUT2 high to CLKOUT1 high, etc.	Q−10	Q	Q+10	ns

NOTE 1: $Q = 1/4t_{c(C)}$.

timing requirements over recommended operating conditions (see Note 1)

		MIN	NOM	MAX	UNIT
$t_{c(CI)}$	CLKIN cycle time	50.0		150	ns
$t_{w(CIL)}$	CLKIN low pulse duration, $t_{c(CI)}$ = 50 ns (see Note 2)		20		ns
$t_{w(CIH)}$	CLKIN high pulse duration, $t_{c(CI)}$ = 50 ns (see Note 2)		20		ns
$t_{su(S)}$	$\overline{\text{SYNC}}$ setup time before CLKIN low	10			ns
$t_{h(S)}$	$\overline{\text{SYNC}}$ hold time from CLKIN low	2Q-10			ns

NOTES: 1. $Q = 1/4t_{c(C)}$.
2. CLKIN duty cycle $[t_{r(CI)} + t_{w(CIH)}]/t_{c(CI)}$ must be within 40-60%. Rise and fall times must be less than 10 ns.

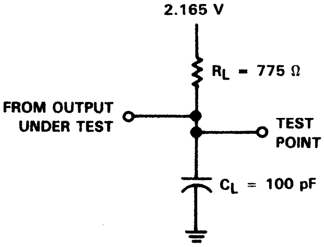
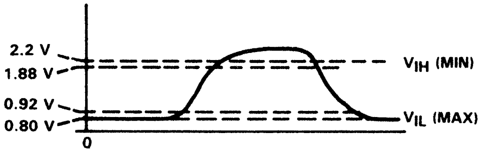
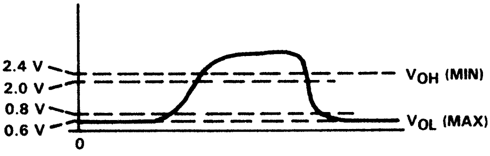


FIGURE 3. TEST LOAD CIRCUIT



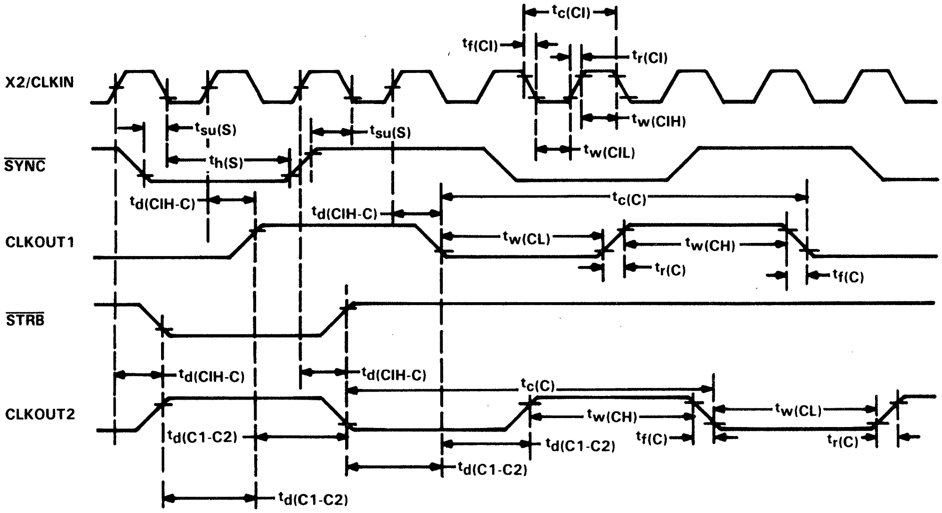
(a) INPUT



(b) OUTPUTS

FIGURE 4. VOLTAGE REFERENCE LEVELS

clock timing



MEMORY AND PERIPHERAL INTERFACE TIMING

switching characteristics over recommended operating conditions (see Note 1)

PARAMETER		MIN	TYP	MAX	UNIT
$t_{d(C1\text{-}S)}$	$\overline{\text{STRB}}$ from CLKOUT1 (if $\overline{\text{STRB}}$ is present)	Q − 25	Q	Q + 25	ns
$t_{d(C2\text{-}S)}$	CLKOUT2 to $\overline{\text{STRB}}$ (if $\overline{\text{STRB}}$ is present)	− 15	0	15	ns
$t_{su(A)}$	Address setup time before $\overline{\text{STRB}}$ low (see Note 3)	Q − 35			ns
$t_{h(A)}$	Address hold time after $\overline{\text{STRB}}$ high (see Note 3)	Q − 15			ns
$t_{w(SL)}$	$\overline{\text{STRB}}$ low pulse duration (no wait states, see Note 4)		2Q		ns
$t_{w(SH)}$	$\overline{\text{STRB}}$ high pulse duration (between consecutive cycles, see Note 4)		2Q		ns
$t_{su(D)W}$	Data write setup time before $\overline{\text{STRB}}$ high (no wait states)	2Q − 45			ns
$t_{h(D)W}$	Data write hold time from $\overline{\text{STRB}}$ high	Q − 15	Q		ns
$t_{en(D)}$	Data bus starts being driven after $\overline{\text{STRB}}$ low (write cycle)	0†			ns
$t_{dis(D)}$	Data bus three-state after $\overline{\text{STRB}}$ high (write cycle)		Q	Q + 30†	ns
$t_{d(MSC)}$	$\overline{\text{MSC}}$ valid from CLKOUT1	− 25	0	25	ns

†These values were derived from characterization data and are not tested.

NOTES: 1. Q = 1/4$t_{c(C)}$.

3. A15-A0, $\overline{\text{PS}}$, $\overline{\text{DS}}$, $\overline{\text{IS}}$, R/$\overline{\text{W}}$, and $\overline{\text{BR}}$ timings are all included in timings referenced as "address."
4. Delays between CLKOUT1/CLKOUT2 edges and $\overline{\text{STRB}}$ edges track each other, resulting in $t_{w(SL)}$ and $t_{w(SH)}$ being 2Q with no wait states.

timing requirements over recommended operating conditions (see Note 1)

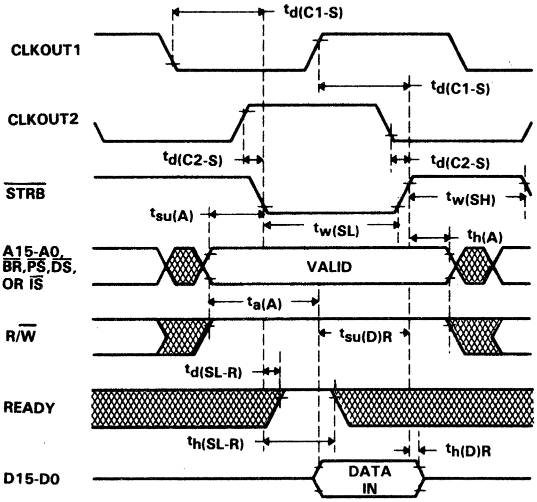
		MIN	NOM	MAX	UNIT
$t_{a(A)}$	Read data access time from address time (read cycle, see Notes 3 and 5)			3Q − 75	ns
$t_{su(D)R}$	Data read setup time before $\overline{\text{STRB}}$ high	40			ns
$t_{h(D)R}$	Data read hold time from $\overline{\text{STRB}}$ high	0			ns
$t_{d(SL\text{-}R)}$	READY valid after $\overline{\text{STRB}}$ low (no wait states)			Q − 40	ns
$t_{d(C2H\text{-}R)}$	READY valid after CLKOUT2 high			Q − 40†	ns
$t_{h(SL\text{-}R)}$	READY hold time after $\overline{\text{STRB}}$ low (no wait states)	Q − 5			ns
$t_{h(C2H\text{-}R)}$	READY hold after CLKOUT2 high	Q − 5†			ns
$t_{d(M\text{-}R)}$	READY valid after $\overline{\text{MSC}}$ valid			2Q − 50†	ns
$t_{h(M\text{-}R)}$	READY hold time after $\overline{\text{MSC}}$ valid	0†			ns

†These values were derived from characterization data and are not tested.

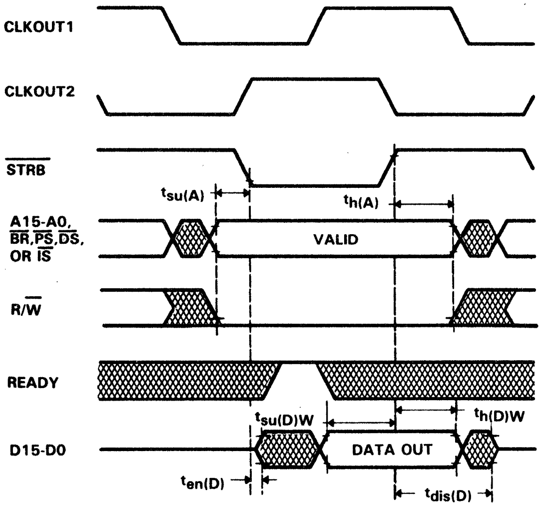
NOTES: 1. Q = 1/4$t_{c(C)}$.

3. A15-A0, $\overline{\text{PS}}$, $\overline{\text{DS}}$, $\overline{\text{IS}}$, R/$\overline{\text{W}}$, and $\overline{\text{BR}}$ timings are all included in timings referenced as "address."
5. Read data access time is defined as $t_{a(A)} = t_{su(A)} + t_{w(SL)} - t_{su(D)R}$.

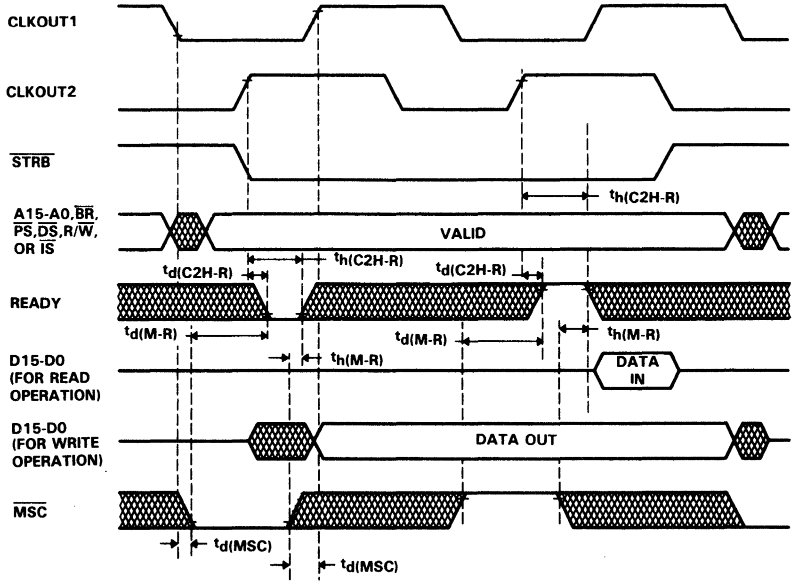
memory read timing



memory write timing



one wait-state memory access timing



$\overline{\text{RS}}$, $\overline{\text{INT}}$, $\overline{\text{BIO}}$, AND XF TIMING

switching characteristics over recommended operating conditions (see Notes 1 and 6)

PARAMETER		MIN	TYP	MAX	UNIT
$t_{d(RS)}$	CLKOUT1 low to reset state entered			45†	ns
$t_{d(IACK)}$	CLKOUT1 to $\overline{\text{IACK}}$ valid	−25	0	25	ns
$t_{d(XF)}$	XF valid before falling edge of $\overline{\text{STRB}}$	Q−35			ns

†These values were derived from characterization data and are not tested.

NOTES: 1. $Q = 1/4t_{c(C)}$.

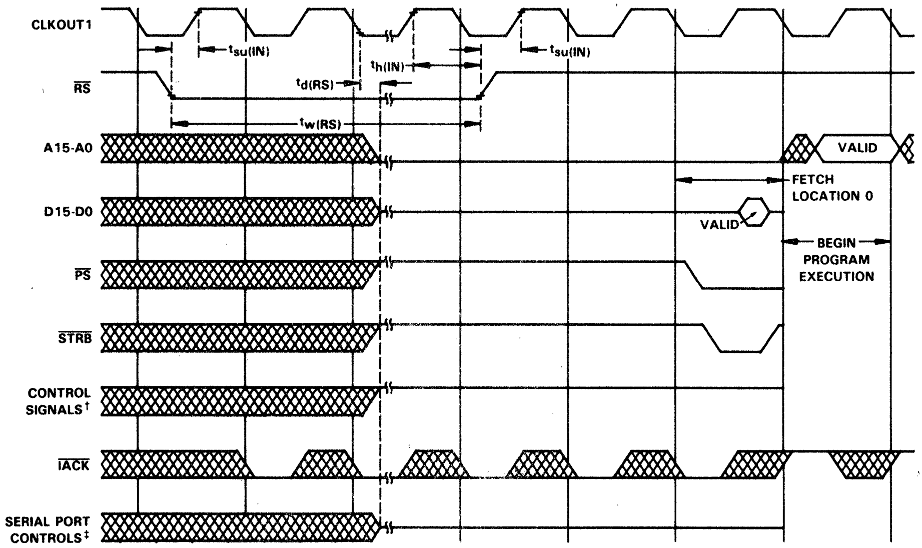
6 $\overline{\text{RS}}$, $\overline{\text{INT}}$, and $\overline{\text{BIO}}$ are asynchronous inputs and can occur at any time during a clock cycle. However, if the specified setup time is met, the exact sequence shown in the timing diagrams will occur. $\overline{\text{INT}}/\overline{\text{BIO}}$ fall time must be less than 10 ns.

timing requirements over recommended operating conditions (see Notes 1 and 6)

		MIN	NOM	MAX	UNIT
$t_{su(IN)}$	$\overline{\text{INT}}/\overline{\text{BIO}}/\overline{\text{RS}}$ setup before CLKOUT1 high	50			ns
$t_{h(IN)}$	$\overline{\text{INT}}/\overline{\text{BIO}}/\overline{\text{RS}}$ hold after CLKOUT1 high	0			ns
$t_{w(IN)}$	$\overline{\text{INT}}/\overline{\text{BIO}}$ low pulse duration	$t_{c(C)}$			ns
$t_{w(RS)}$	$\overline{\text{RS}}$ low pulse duration	$3t_{c(C)}$			ns

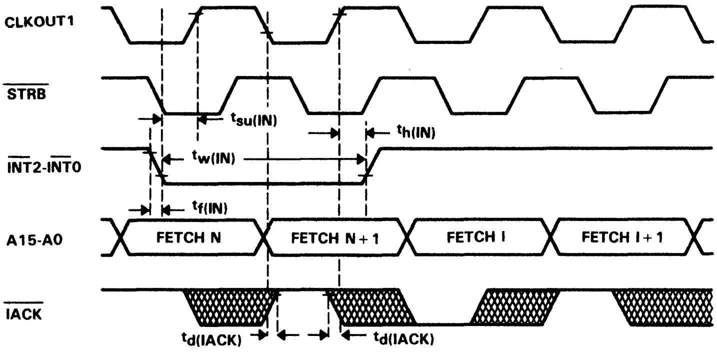
NOTES: 1. $Q = 1/4t_{c(C)}$.

6. $\overline{\text{RS}}$, $\overline{\text{INT}}$, and $\overline{\text{BIO}}$ are asynchronous inputs and can occur at any time during a clock cycle. However, if the specified setup time is met, the exact sequence shown in the timing diagrams will occur. $\overline{\text{INT}}/\overline{\text{BIO}}$ fall time must be less than 10 ns.

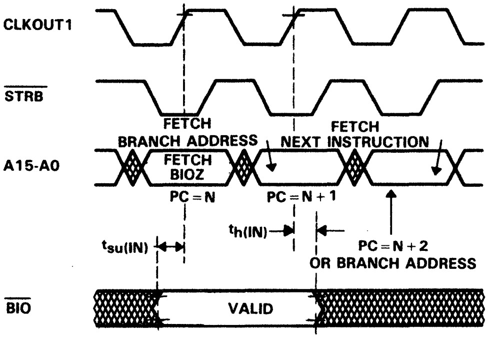


†Control signals are $\overline{\text{DS}}$, $\overline{\text{IS}}$, R/$\overline{\text{W}}$ and XF.
‡Serial port controls are $\overline{\text{DX}}$ and FSX.

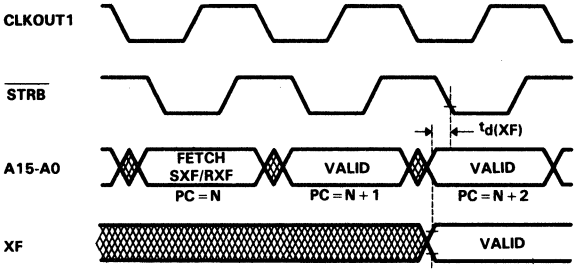
interrupt timing



$\overline{\text{BIO}}$ timing



external flag timing



$\overline{\text{HOLD}}$ TIMING

switching characteristics over recommended operating conditions (see Note 1)

PARAMETER		MIN	TYP	MAX	UNIT
$t_{d(C1L-AL)}$	$\overline{\text{HOLDA}}$ low after CLKOUT1 low	−25		25	ns
$t_{dis(AL-A)}$	$\overline{\text{HOLDA}}$ low to address three-state		15		ns
$t_{dis(C1L-A)}$	Address three-state after CLKOUT1 low ($\overline{\text{HOLD}}$ mode, see Note 7)			30†	ns
$t_{d(HH-AH)}$	$\overline{\text{HOLD}}$ high to $\overline{\text{HOLDA}}$ high			50	ns
$t_{en(A-C1L)}$	Address driven before CLKOUT1 low ($\overline{\text{HOLD}}$ mode, see Note 7)			10†	ns

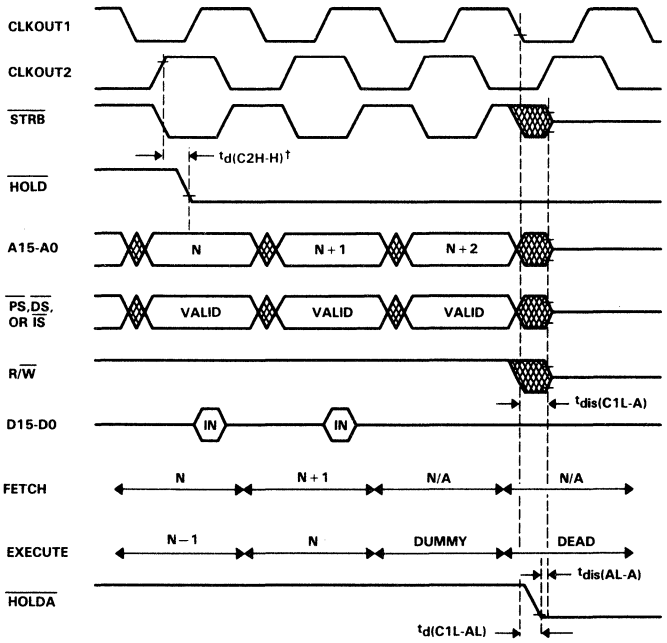
NOTES: 1. $Q = 1/4t_{c(C)}$.
7. A15-A0, $\overline{\text{PS}}$, $\overline{\text{DS}}$, $\overline{\text{IS}}$, $\overline{\text{STRB}}$, and R/$\overline{\text{W}}$ timings are all included in timings referenced as "address."

timing requirements over recommended operating conditions (see Note 1)

		MIN	NOM	MAX	UNIT
$t_{d(C2H-H)}$	$\overline{\text{HOLD}}$ valid after CLKOUT2 high			Q-40†	ns

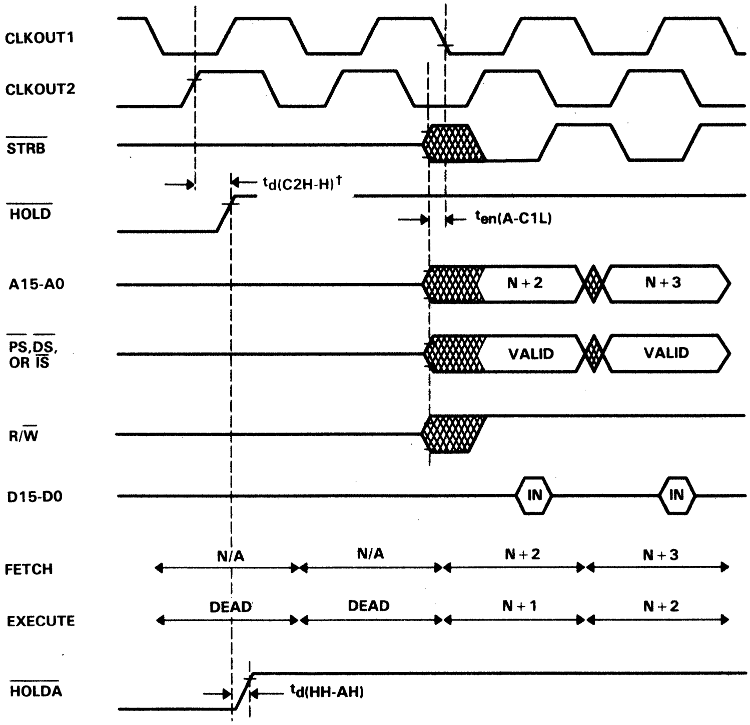
†These values were derived from characterization data and are not tested.
NOTE 1: $Q = 1/4t_{c(C)}$.

$\overline{\text{HOLD}}$ timing (part A)



†$\overline{\text{HOLD}}$ is an asynchronous input and can occur at any time during a clock cycle. If the specified timing is met, the exact sequence shown will occur; otherwise, a delay of on CLKOUT2 cycle will occur.

HOLD timing (part B)



---

† HOLD is an asynchronous input and can occur at any time during a clock cycle. If the specified timing is met, the exact sequence shown will occur; otherwise, a delay of on CLKOUT2 cycle will occur.

SERIAL PORT TIMING

switching characteristics over recommended operating conditions (see Note 1)

PARAMETER		MIN	TYP	MAX	UNIT
$t_{d(CH\text{-}DX)}$	DX valid after CLKX rising edge (see Note 8)			100	ns
$t_{d(FL\text{-}DX)}$	DX valid after FSX falling edge (TXM = 0, see Note 8)		50		ns
$t_{d(CH\text{-}FS)}$	FSX valid after CLKX rising edge (TXM = 1)			60	ns

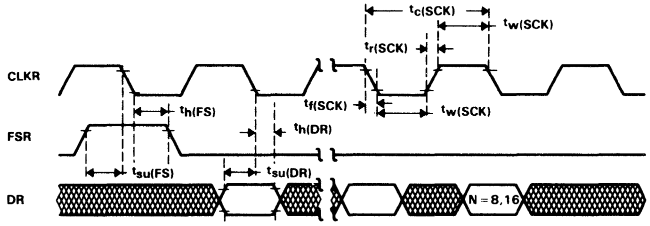
NOTES: 1. Q = $1/4t_{c(C)}$.
8. The last occurrence of FSX falling and CLKX rising.

timing requirements over recommended operating conditions (see Note 1)

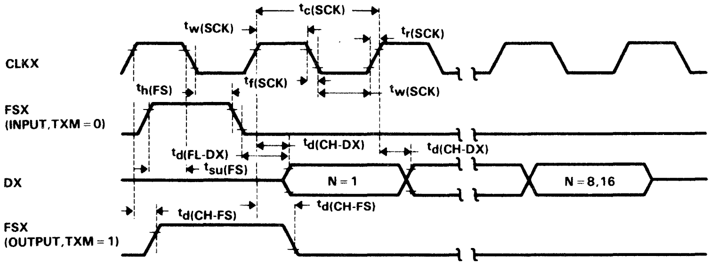
		MIN	NOM	MAX	UNIT
$t_{c(SCK)}$	Serial port clock (CLKX/CLKR) cycle time	400		24,000	ns
$t_{w(SCK)}$	Serial port clock (CLKX/CLKR) low pulse duration (see Note 9)		$0.5t_{c(SCK)}$		ns
$t_{w(SCK)}$	Serial port clock (CLKX/CLKR) high pulse duration (see Note 9)		$0.5t_{c(SCK)}$		ns
$t_{su(FS)}$	FSX/FSR setup time before CLKX/CLKR falling edge (TXM = 0)	20			ns
$t_{h(FS)}$	FSX/FSR hold time after CLKX/CLKR falling edge (TXM = 0)	20			ns
$t_{su(DR)}$	DR setup time before CLKR falling edge	20			ns
$t_{h(DR)}$	DR hold time after CLKR falling edge	20			ns

NOTES: 1. Q = $1/4t_{c(C)}$.
9. The duty cycle of the serial port clock must be within 40-60%. Serial port clock (CLKX/CLKR) rise and fall times must be less than 50 ns.

serial port receive timing



serial port transmit timing

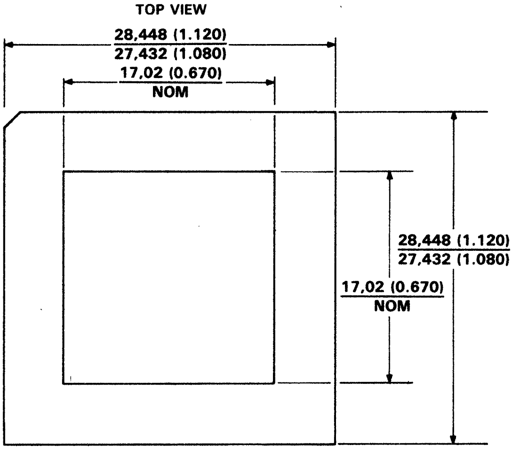


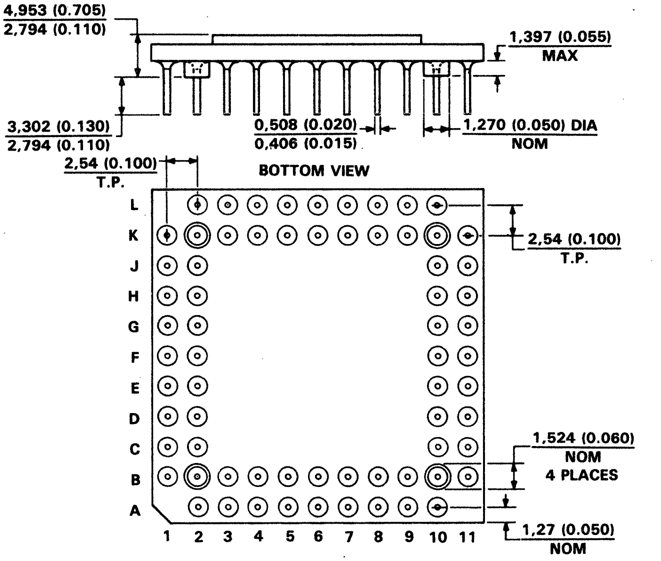
MECHANICAL DATA

68-pin GB pin grid array ceramic package

THERMAL RESISTANCE CHARACTERISTICS

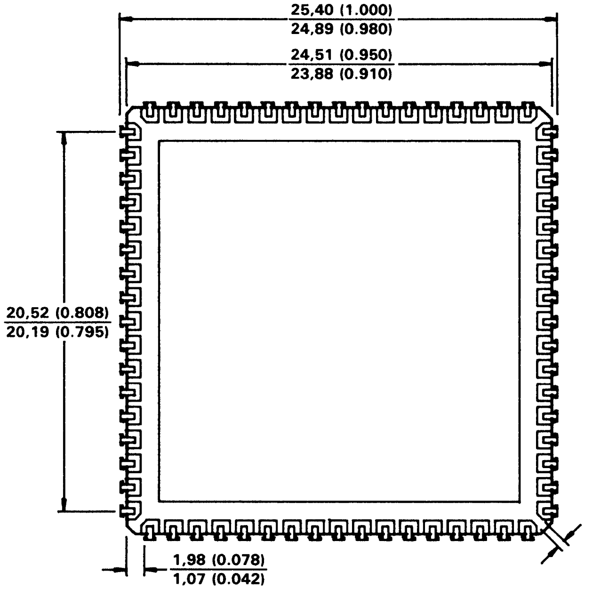
PARAMETER		MAX	UNIT
$R_{\theta JA}$	Junction-to-free-air thermal resistance	36	°C/W
$R_{\theta JC}$	Junction-to-case thermal resistance	6	°C/W





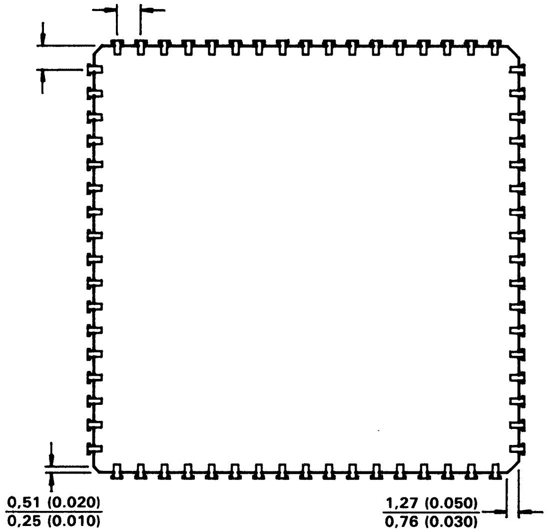
ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

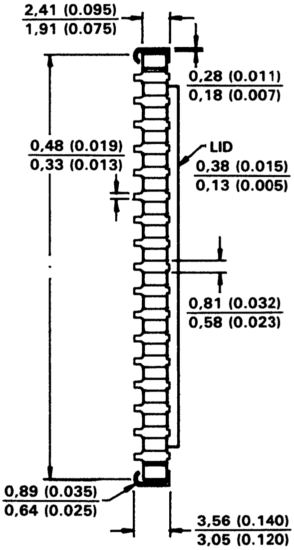
FJ ceramic leaded chip carrier packages



THERMAL RESISTANCE CHARACTERISTICS

PARAMETER		MAX	UNIT
$R_{\theta JA}$	Junction-to-free-air thermal resistance	36	°C/W
$R_{\theta JC}$	Junction-to-case thermal resistance	6	°C/W





ALL DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

C. TMS320C2x System Migration

This appendix contains information necessary to upgrade a first-generation TMS320 (TMS320C1x) program to a TMS32020-based system or a TMS32020 program to a TMS320C25-based system. The information consists of a detailed list of the programming differences and hardware and timing differences between the respective processors.

The two major sections are as follows:

- TMS320C1x to TMS32020 System Migration (Section C.1 on page C-2)
- TMS32020 to TMS320C25 System Migration (Section C.2 on page C-4)

C.1 TMS320C1x to TMS32020 System Migration

This section lists the programming differences that should be considered in migrating from a TMS320C1x to a TMS32020 processor.

1) Instructions are compatible only at the mnemonic level. TMS320C1x source programs should be reassembled using a TMS32020 assembler before execution.

2) The memory map on the TMS32020 is different from the memory map on the TMS320C1x. Page 0 of the TMS32020's data memory map contains only block B2 (32 words) and the memory-mapped registers. The primary on-chip RAM blocks B0 and B1 reside on pages 4-7 when all RAM is configured as data memory. It should be noted that there may be cases in TMS320C1x programs where the BANZ instruction has been used to implement both a loop counter and a memory address pointer for tables based at location 0 in memory. Since blocks B0, B1, and B2 in the TMS32020 are located at addresses other than 0, programs being migrated from the TMS320C1x to the TMS32020 should implement this type of BANZ loop using two separate auxiliary registers, one for loop count and one for memory address.

3) The SXM bit must be set to 1 and the PM bits must be set to 0 to ensure that TMS32020 CALU operations behave in the same manner as the TMS320C1x. The SXM and PM bits are unaffected by a reset and are in a random state after powerup.

4) The organization of status register ST0 is different on the two processors as shown below.

TMS320C1x Status Register ST0:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OV	OVM	INTM	1	1	1	1	ARP	1	1	1	1	1	1	1	DP

TMS32020 Status Register ST0:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARP			OV	OVM	1	INTM	DP								

5) In the direct addressing mode, the SST (store status register ST0) instruction of the TMS32020 sets DP = 0, rather than 1 as on the TMS320C1x. The SST1 instruction also sets DP = 0 in the direct addressing mode. Note that in the direct addressing mode, data memory addressing values should be between 96 and 127 to store the status registers in block B2.

6) When modifying the contents of the current auxiliary register in the indirect addressing mode on the TMS32020, the SAR (store auxiliary register) instruction for ARn when n = ARP stores the value of the auxiliary register contents BEFORE it is incremented, decremented, or indexed by AR0. The TMS320C1x stores the incremented/decremented value.

7) All of the TMS32020 branch and call instructions, except for BACC (branch to address specified by accumulator) and CALA (call subroutine indirect), allow both auxiliary register and auxiliary register pointer (ARP) modification in the seven LSBs of the opcode.

8) The SACL (store low accumulator with shift) instruction on the TMS32020 allows shift codes of 0, 1, and 4.

9) A multiplication of >8000 x >8000 on the TMS32020 yields the correct result of >40000000, not >C0000000 as on the TMS320C1x.

10) The multiply instructions, MPY and MPYK, are not interrupt-protected on the TMS32020 since the capability now exists to restore the P register directly.

11) The IN and OUT opcodes now have a 4-bit port address to allow for a total of 16 I/O ports on the TMS32020.

12) A TBLW (table write) instruction on the TMS32020 to program memory locations 0-7 can be distinguished externally from an OUT instruction to port addresses 0-7 via the $\overline{\text{PS}}$ and $\overline{\text{IS}}$ (program and I/O space select) strobes.

13) The SUBC (conditional subtract) instruction is a true single-cycle instruction on the TMS32020 and can be used with the repeat instructions, RPT or RPTK. On the TMS320C1x, SUBC cannot be followed immediately by another instruction that uses the accumulator.

14) When modifying the auxiliary registers in the indirect addressing mode on the TMS32020, the auxiliary registers act as 16-bit, rather than 8-bit, counters (i.e., 'wraparound' occurs modulo 2^{16}, instead of modulo 2^8 as on the TMS320C1x). When used with the BANZ (branch on auxiliary register not zero) instruction, the auxiliary registers on the TMS32020 act as 16-bit counters, rather than 9-bit counters as on the TMS320C1x.

C.2 TMS32020 to TMS320C25 System Migration

This section lists the programming, hardware, and timing differences that should be considered in migrating from the TMS32020 to the TMS320C25.

1) Instructions are fully compatible at the object code level. TMS32020 object (memory image) code can be used directly on the TMS320C25 processor.

2) Instructions are compatible at the source code level. The NORM instruction that previously had no operands now has an optional operand to define the auxiliary register modification. Any comments on the same line in the source code file will be interpreted as the operand if no other operand is specified. NORM instructions should be modified to specify the default operand, *+.

3) When zero is loaded into the accumulator and the NORM instruction is executed, the auxiliary register (ARx) on the TMS320C25 is modified and the TC is set on the first execution. On the TMS32020, the auxiliary register (ARx) is incremented each execution cycle and the TC is not set.

4) Execution cycle timings of instructions have been modified. Most TMS320C25 instructions execute in a single machine cycle. The number of cycles for some multicycle instructions have been changed. Refer to Appendix D for detailed information on instruction cycle timings. By following the entries in this appendix, the key timing differences can be noted.

5) The IDLE instruction automatically sets the INTM bit in status register ST0 to a zero. This assures that an external interrupt will 'wake up' the processor. The instruction also requires three memory cycles to execute on the TMS320C25 rather than one as on the TMS32020.

6) In general, all branch, call, and return instructions that reload the program counter (PC) should be counted as three-cycle instructions when evaluating code execution timings on the TMS320C25.

7) When an interrupt occurs, one additional instruction cycle will be present on the TMS320C25 prior to interrupt acknowledge. When the device is released from the hold mode, there will be one additional cycle preceding the first valid memory fetch.

8) The store instructions (SACH, SACL, etc.) execute in one less cycle on the TMS320C25 than on the TMS32020 when data is stored to external data memory.

9) The MAC and MACD instructions require one extra cycle, going from three to four cycles. The extra cycle is in the instruction read and setup overhead, and repeated execution will be one cycle per execution as on the TMS32020.

10) The delay for a new memory configuration to become effective when using the CNFD or CNFP instructions on the TMS320C25 is two instruction fetches (for single-cycle instructions) when executing from external memory or internal ROM, as compared to one instruction fetch for the TMS32020. Thus, on the TMS320C25, a CNFP instruction must be placed at location 65277 if execution is to continue from the first location in block B0. When execution is from internal RAM on the TMS320C25, however, this delay is one instruction fetch as on the TMS32020.

11) The timer on the TMS320C25 is clocked by CLKOUT1 and counts PRD + 1 CLKOUT1 cycles, whereas the timer on the TMS32020 is clocked by CLKOUT1/4 and counts 4 × PRD cycles. Therefore, to count an equivalent amount of time on the TMS320C25 using the same input clock frequency, PRD values from the TMS32020 must first be multiplied by four and then decremented by one. If different input clock frequencies are used, this must also be accounted for by multiplying the PRD value for the TMS320C25 obtained above by the ratio of the TMS320C25 input clock frequency to the TMS32020 input clock frequency.

12) To simplify device timing descriptions, the internal clock phase reference numbers have been redefined in the TMS320C25. The new clock phase definitions have quarter-phase 1 (Q1) beginning a bus cycle, as opposed to a cycle beginning with Q3 as in the TMS32020. Note that no changes have been made to any of the device logic; the clock phases have merely been renamed.

13) The effect of the $\overline{\text{SYNC}}$ input, although functionally the same on the TMS32020 and TMS320C25, is delayed by two cycles on the TMS320C25 from that of the TMS32020. Accordingly, the exact timings produced with the application of $\overline{\text{SYNC}}$ on the two devices may differ depending on the clock phase in which $\overline{\text{SYNC}}$ is applied. Due to the two-cycle offset between the clock phase definitions on the two devices (see the previous paragraph) and the two-cycle delay in the effect of $\overline{\text{SYNC}}$ on the TMS320C25, the clock timings produced when the two devices are running in synchronization are identical. That is, a TMS32020 and a TMS320C25 can be operated together in synchronization in a system using the same $\overline{\text{SYNC}}$ input.

14) On the TMS320C25, both the timer (TIM) and period (PRD) registers are initialized to >FFFF on reset, while on the TMS32020, only the TIM register is initialized.

15) Several bits (C, HM, and FSM) have been added to status register ST1 on the TMS320C25, as shown below.

TMS32020 Status Register ST1:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARB			CNF	TC	SXM	1	1	1	1	1	XF	FO	TXM	PM	

TMS320C25 Status Register ST1:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARB			CNF	TC	SXM	C	1	1	HM	FSM	XF	FO	TXM	PM	

The FSM, HM, and C status register bits are initialized by reset and are all set to one when reset occurs. Note that the new bits are assigned polarities in such a way that the values of the corresponding bits on the TMS32020 invoke a TMS32020-like operation on the TMS320C25.

The SXM and PM status register bits that were previously uninitialized on the TMS32020 are now initialized by reset on the TMS320C25. When the TMS320C25 is reset, SXM is set to one, and the PM bits are set to zero.

16) Four differences between the serial ports on the TMS32020 and TMS320C25 that impact system migration are:

   a) The double-buffering on the TMS320C25 serial port greatly increases the amount of time available for processing serial port interrupts and affects how the FSR and FSX pulse are used. As a result of the double-buffering, both edges of the FSR and FSX pulses are used on the TMS320C25 instead of only the falling edge, as is the case on the TMS32020.

   On the TMS32020, the falling edge of the FSX pulse is used to start transmission of the data present in the DXR (transmit register). Likewise, the falling edge of the FSR pulse is used to start reception of data into the DRR (receive register). The limitations on the FSR and FSX pulses are a minimum setup time (20 ns) and a minimum hold time (20 ns). Once serial port operation begins, the receipt or transmission of the register's contents, either 8 or 16 bits, is completed even if the FSR or FSX signals change to a logic high level. A new transfer of data into the DRR or out of the DXR only begins when the next falling edge of the FSR or FSX pulses occurs.

   On the TMS320C25, the double-buffering affects the use of the FSR and FSX pins and consequently the serial port operation itself. For the transmit operation, the TMS320C25 provides a separate XSR (transmit shift register), necessitating the use of the rising edge of the FSX pulse. Data is transferred from the DXR to the XSR on the first falling CLKX (serial transmit clock) following a rising FSX. At this point, the data is in the XSR and waiting to be shifted out or transmitted. Transmission begins on the first falling CLKX following the falling FSX, and continues with the subsequent bits in the XSR as long as the FSX signal remains low. If the FSX signal goes high before the last transmission has completed, the contents of the DXR are transferred to the XSR and the previous transmission is aborted. Transmission of this new information begins after the FSX signal goes low again.

   Similarly for the receive operation, the TMS320C25 has a separate RSR (receive shift register). In this case, the data is transferred from the RSR to the DRR when the last bit has been received. Therefore, if a new transfer is initiated by toggling the FSR pin, the previous reception is aborted and the contents of RSR are not transferred to DRR.

   Consequently, there is one additional limitation on the FSR and FSX pulses on the TMS320C25. FSR and FSX must have a minimum low pulse duration to allow the complete transfer of all 8 or 16 bits of data into and out of RSR and XSR, respectively.

Unlike the TMS32020, loading the DXR does not interfere with transmission. There is no restriction on when the DXR can be loaded when using external FSX. Correspondingly, DRR may be read at any time during the reception of the current data, extending the time allowed to respond to the receive interrupt and to read the previous word of data.

b) The fully static operation of the TMS320C25 effectively places no lower limit on serial port clock frequency.

c) Serial port interrupts are generated half of a CLKR or CLKX cycle later on the TMS320C25 than they are on the TMS32020. Specifically, on the TMS32020, RINT and XINT are generated on the falling edge of CLKR and CLKX, respectively, during transfer of the last bit. On the TMS320C25, RINT and XINT are generated on the rising edge of CLKR or CLKX after the last bit has been transferred. This should not be critical for TMS32020 programs running on the TMS320C25 since double-buffering of the serial port on the TMS320C25 allows more time for processing of serial port interrupts. Some modification of TMS32020 programs may, however, be required to take advantage of the double-buffering, depending on how serial port interrupt servicing is implemented.

d) The DRR behaves differently when operating the TMS320C25 serial port in byte mode than it does on the TMS32020. On the TMS32020, the contents of the most-significant byte of DRR remain unchanged once byte mode is initiated by executing a FORT instruction. On the TMS320C25, however, each time a new byte is received, the previous contents of the least-significant byte of DRR are transferred to the most significant byte of DRR.

Figure C-1 illustrates the behavior of DRR on both the TMS32020 and the TMS320C25 processors.

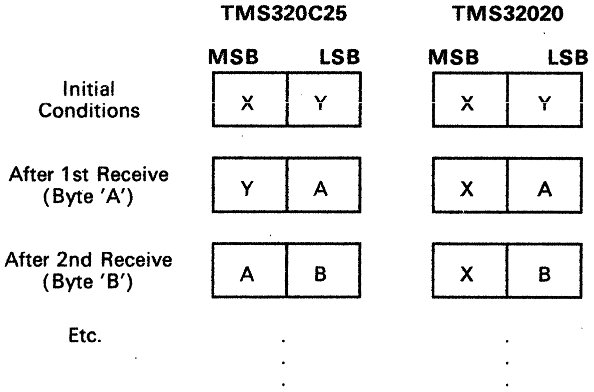


Figure C-1. Serial Port System Migration

D. Instruction Cycle Timings

This appendix details the instruction cycle timings for the TMS32020 and TMS320C25 processors. Instructions for each device are first listed in a table according to cycle classification. Then each class of instructions is listed in another table(s), showing the number of cycles required for a given TMS320C2x instruction to execute in a given memory configuration when executed as a single instruction or in the repeat mode. The column headings in the tables indicate the program source location (PI, PE, or PR) and data destination or source (DI or DE), defined as follows:

PI	The instruction executes from internal program memory (RAM).
PR	The instruction executes from internal program memory (ROM).
PE	The instruction executes from external program memory.
DI	The instruction executes using internal data memory.
DE	The instruction executes using external data memory.

The number of cycles required for each instruction is given in terms of the program/data memory and I/O access times as defined in the following listing:

p Program memory wait states. Represents the number of clock cycles the device waits for external program memory to respond to an access. T_{ac} is the access time, in nanoseconds, (maximum) required by the TMS320C2x for an external memory access to be made with no wait states. T_{mem} is the memory device access time, and T_p is the clock period (4/crystal frequency).

$p = 0;$ If $T_{mem} \leq T_{ac}$
$p = 1;$ If $T_{ac} < T_{mem} \leq (T_p + T_{ac})$
$p = 2;$ If $(T_p + T_{ac}) < T_{mem} \leq (T_p \times 2 + T_{ac})$
$p = k;$ If $[T_p \times (k\text{-}1) + T_{ac}] < T_{mem} \leq (T_p \times k + T_{ac})$

d Data memory wait states. Represents the number of cycles the device must wait for external data memory to respond to an access. This number is calculated in the same way as the p number.

i I/O memory wait states. Represents the number of cycles the device must wait for external I/O memory to respond to an access. This number is calculated in the same way as the p number.

Other abbreviations used in the tables and their meanings are as follows:

br	Branch from ...
int	Internal program memory.
INT	Interrupt.
ext	External program memory.
n	The number of times an instruction is executed when using the RPT or RPTK instruction.

D.1 TMS32020 Instruction Cycle Timings

Table D-1 lists the TMS32020 instructions according to cycle classification. Table D-2 shows the number of cycles required for a given TMS32020 instruction to execute in a given memory configuration when executed as a single instruction or in the repeat mode, respectively.

Table D-1. TMS32020 Instructions by Cycle Class

CLASS	INSTRUCTION
I	ADD ADDH ADDS ADDT AND BIT BITT DMOV LAC LACT LAR LDP LPH LST LST1 LT LTA LTD LTP LTS MPY OR RPT SQRA SQRS SUB SUBC SUBH SUBS SUBT XOR ZALH ZALS (RPT not repeatable)
II	SACH SACL SAR SST SST1
III	ABS APAC CMPL CMPR CNFD CNFP DINT EINT FORT LACK LARK LARP LDPK MAR MPYK NEG NOP NORM PAC ROVM RPTK RSXM RTXM RXF SFL SFR SOVM SPAC SPM SSXM STXM SXF ZAC (LACK, LARK, LDPK, MPYK, RPTK, SPM, ZAC not repeatable)
IV	ADLK ANDK LALK LRLK ORK SBLK XORK (all not repeatable)
V	MAC MACD
VI	B BANZ BBNZ BBZ BGEZ BGZ BIOZ BLEZ BLZ BNV BNZ BV BZ CALL (all not repeatable)
VII	BACC CALA POP PUSH RET TRAP (BACC, CALA, RET, TRAP not repeatable)
VIII	IN OUT
IX	TBLR TBLW
X	BLKD
XI	BLKP
XII	POPD PSHD
XIII	IDLE (not repeatable)

Table D-2. TMS32020 Instruction Cycle Timings

CLASS	WHEN NOT IN REPEAT MODE				WHEN IN REPEAT MODE			
	PI/DI	PI/DE	PE/DI	PE/DE	PI/DI	PI/DE	PE/DI	PE/DE
I	1	2+d	1+p	2+d+p	n	2n+nd	n+p	2n+nd+p
II	1	2+d	1+p	3+d+p	n	2n+nd	n+p	3n+nd+p
III	1	1	1+p	1+p	n	n	n+p	n+p
IV	2	2	2+2p	2+2p	not repeatable			
V	3	N/A	3+2p	N/A	2+n	N/A	2+n+2p	N/A
VI	2 (br int-to-int) 2+p (ext-to-int)		2+p (int-to-ext) 2+2p (ext-to-ext)		not repeatable not repeatable			
VII	2	2	2+p	2+p	2n	2n	2n+p	2n+p
VIII	1+i	2+d+i	2+p+i	3+d+p+i	n+ni	2n+nd+ni	2n+p+ni	3n+nd+p +ni
IX	Table in internal program memory:				Table in internal program memory:			
	3	3+d	3+p	3+d+p	2+n	2+n+nd	2+n+p	2+n+nd+p
	Table in external program memory:				Table in external program memory:			
	3+p	4+d+p	3+2p	4+d+2p	2+n+np	2+2n+nd +np	2+n+np+p	2+2n+nd +np+p
X	Data source internal:†				Data source internal:†			
	3	3+d	3+2p	3+d+2p	2+n	2+n+nd	2+n+2p	2+n+nd +2p
	Data source external:†				Data source external:†			
	3+d	4+2d	3+d+2p	4+2d+2p	2+n+nd	2+2n+2nd	2+n+nd +2p	2+2n+2nd +2p
XI	Program source internal:†				Program source internal:†			
	3	3+d	3+2p	3+d+2p	2+n	2+n+nd	2+n+2p	2+n+nd +2p
	Program source external:†				Program source external:†			
	3+p	4+d+p	3+3p	4+d+3p	2+n+np	2+2n+nd +np	2+n+np +2p	2+2n+nd +np+2p
XII	2	2+d	2+p	2+d+p	2n	2n+nd	2n+p	2n+nd+p
XIII	1 (minimum waits for INT)		1+p (minimum waits for INT)		not repeatable			

†Column headings 'DI/DE' refer to data destination.

D.2 TMS320C25 Instruction Cycle Timings

Table D-3 lists the TMS320C25 instructions according to cycle classification. Table D-4 and Table D-5 show the number of cycles required for a given TMS320C25 instruction to execute in a given memory configuration when executed as a single instruction or in the repeat mode, respectively.

Table D-3. TMS320C25 Instructions by Cycle Class

CLASS	INSTRUCTION
I	ADD ADDC ADDH ADDS ADDT AND BIT BITT DMOV LAC LACT LPH LT LTA LTD LTP LTS MPY MPYA MPYS MPYU PSHD OR RPT SQRA SQRS SUB SUBB SUBC SUBH SUBS SUBT XOR ZALH ZALR ZALS (RPT not repeatable)
II	LAR LDP LST LST1
III	POPD SACH SACL SAR SPH SPL SST SST1
IV	ABS ADDK ADRK APAC CMPL CMPR CNFD CNFP DINT EINT FORT LACK LARK LARP LDPK MAR MPYK NEG NOP NORM PAC POP PUSH RC RFSM RHM ROL ROR ROVM RPTK RSXM RTC RTXM RXF SBRK SC SFL SFR SFSM SHM SOVM SPAC SPM SSXM STC STXM SUBK SXF ZAC (ADDK, ADRK, LACK, LARK, LDPK, MPYK, RPTK, SBRK, SPM, SUBK, and ZAC not repeatable)
V	ADLK ANDK LALK LRLK ORK SBLK XORK (All not repeatable)
VI	MAC MACD
VII	B BANZ BBNZ BBZ BC BGEZ BGZ BIOZ BLEZ BLZ BNC BNV BNZ BV BZ CALL (All not repeatable)
VIII	BACC CALA RET TRAP (All not repeatable)
IX	IN
X	OUT
XI	TBLR
XII	TBLW (Table in ROM not applicable)
XIII	BLKD
XIV	BLKP
XV	IDLE (not repeatable)

Table D-4. Cycle Timings for Cycle Classes When Not in Repeat Mode

CLASS	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
I	1	2+d	1+p	2+d+p	1	2+d
II	1	2+d	1+p	2+d+p	1	2+d
III	1	1+d	1+p	2+d+p	1	1+d
IV	1	1	1+p	1+p	1	1
V	2	2	2+2p	2+2p	2	2
VI	Table in on-chip RAM:					
	3	4+d	4+2p	5+d+2p	4	5+d
	Table in on-chip ROM:					
	4	5+d	4+2p	5+d+2p	4	5+d
	Table in external memory:					
	4+p	5+d+p	4+3p	5+d+3p	4+p	5+d+p
VII	True Conditions:					
	Destination on-chip RAM:					
	2	2	2+2p	2+2p	2	2
	Destination on-chip ROM:					
	3	3	3+2p	3+2p	3	3
	Destination external memory:					
	3+p	3+p	3+3p	3+3p	3+p	3+p
	False Condition:					
	Destination anywhere:					
	2	2	2+2p	2+2p	2	2
VIII	Destination on-chip RAM:					
	2	2	2+p	2+p	2	2
	Destination on-chip ROM:					
	3	3	3+p	3+p	3	3
	Destination external memory:					
	3+p	3+p	3+2p	3+2p	3+p	3+p
IX	2+i	2+d+i	2+p+i	3+d+p+i	2+i	2+d+i
X	1+i	2+d+i	2+p+i	3+d+p+i	1+i	2+d+i
XI	Table in on-chip RAM:					
	2	2+d	3+p	3+d+p	3	3+d
	Table in on-chip ROM:					
	3	3+d	4+p	4+d+p	4	4+d
	Table in external memory:					
	3+p	3+d+p	4+2p	4+d+2p	4+p	4+d+p
XII	Table in on-chip RAM:					
	2	3+d	3+p	4+d+p	3	4+d
	Table in on-chip ROM:					
	not applicable					
	Table in external memory:					
	2+p	3+d+p	3+2p	4+d+2p	3+p	4+d+p

Table D-4. Cycle Timings for Cycle Classes When Not in Repeat Mode (Concluded)

CLASS	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
XIII	Source data in on-chip RAM:					
	3	3+d	3+2p	3+d+2p	3	3+d
	Source data in external memory:					
	4+d	4+2d	4+d+2p	4+2d+2p	4+d	4+2d
XIV	Table in on-chip RAM:					
	3	3+d	4+2p	4+d+2p	4	4+d
	Table in on-chip ROM:					
	4	4+d	4+2p	4+d+2p	4	4+d
	Table in external memory:					
	4+p	4+d+p	4+3p	4+d+3p	4+p	4+d+p
XV	(Interrupt) destination on-chip ROM 3 (minimum waits for INT) (Interrupt) destination external memory 3+2p (minimum waits for INT)					

Table D-5. Cycle Timings for Cycle Classes When in Repeat Mode

CLASS	PI/DI	PI/DE	PE/DI	PE/DE	PR/DI	PR/DE
I	n	1+n+nd	n+p	1+n+nd+p	n	1+n+nd
II	n	2n+nd	n+p	2n+nd+p	n	2n+nd
III	n	n+nd	n+p	1+n+nd+p	n	n+nd
IV	n	n	n+p	n+p	n	n
V	not repeatable					
VI	Table in on-chip RAM:					
	2+n	2+2n+nd	3+n+2p	3+2n+nd+2p	3+n	3+2n+nd
	Table in on-chip ROM:					
	3+n	3+2n+nd	3+n+2p	3+2n+nd+2p	3+n	3+2n+nd
	Table in external memory:					
	3+n+np	3+2n+nd+np	3+n+np+2p	3+2n+nd+np +2p	3+n+np	3+2n+nd+np
VII	not repeatable					
VIII	not repeatable					
IX	1+n+ni	2n+nd+ni	1+n+p+ni	1+2n+nd+p +ni	1+n+ni	2n+nd+ni
X	n+ni	2n+nd+ni	1+n+p+ni	1+2n+nd+p +ni	n+ni	2n+nd+ni
XI	Table in on-chip RAM:					
	1+n	1+n+nd	2+n+p	2+n+nd+p	2+n	2+n+nd
	Table in on-chip ROM:					
	2+n	2+n+nd	3+n+p	3+n+nd+p	3+n	3+n+nd
	Table in external memory:					
	2+n+np	1+2n+nd+np	3+n+np+p	2+2n+nd+np +p	3+n+np	2+2n+nd+np
XII	Table in on-chip RAM:					
	1+n	2+n+nd	2+n+p	3+n+nd+p	2+n	3+n+nd
	Table in on-chip ROM:					
	not applicable					
	Table in external memory:					
	1+n+np	1+2n+nd+np	2+n+np+p	2+2n+nd+np+p	2+n+np	2+2n+nd+np
XIII	Source data in on-chip RAM:					
	2+n	2+n+nd	2+n+2p	2+n+nd+2p	2+n	2+n+nd
	Source data in external memory:					
	3+n+nd	2+2n+2nd	3+n+nd+2p	2+2n+2nd+2p	3+n+nd	2+2n+2nd
XIV	Table in on-chip RAM:					
	2+n	2+n+nd	3+n+2p	3+n+nd+2p	3+n	3+n+nd
	Table in on-chip ROM:					
	3+n	3+n+nd	3+n+2p	3+n+nd+2p	3+n	3+n+nd
	Table in external memory:					
	3+n+np	2+2n+nd+np	3+n+np+2p	2+2n+nd+np +2p	3+n+np	2+2n+nd+np
XV	not repeatable					

E. Development Support/Part Order Information

This section provides development support information, device part numbers, and support tool ordering information for all TMS320C2x (second-generation TMS320) products. Figure E-1 shows the software and hardware development tools available for the TMS320C2x, including the development environment when using the C compiler (see Section E.1.4).

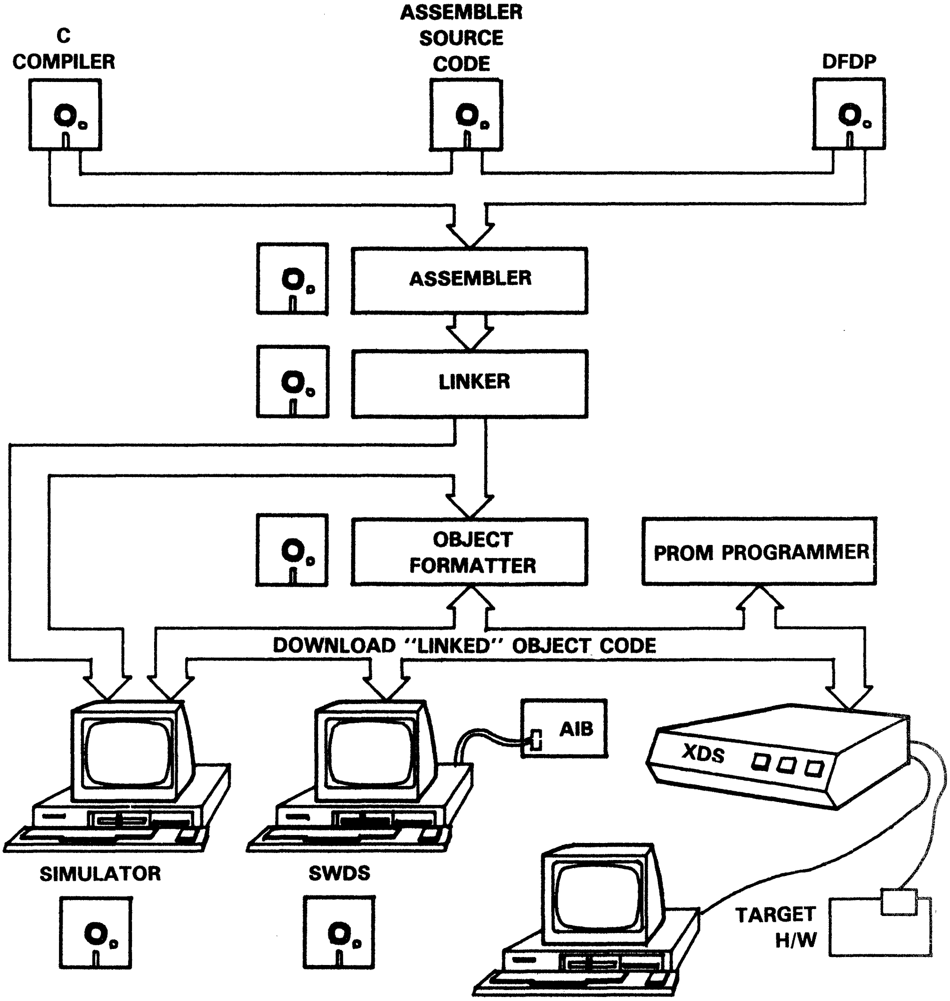


Figure E-1. TMS320C2x Development Tools

Extensive documentation, including application reports, user's guides, and textbooks, is available to support DSP design, research, and education. To order TMS320 literature, contact the TI Customer Response Center (CRC) hotline number, 1-800-232-3200. For more information about support products and documentation, refer to the *TMS320 Family Development Support Reference Guide*.

The nearest TI field sales office can be contacted for support tool availability or further details (see list of sales offices and distributors at end of book). For technical support, contact the TMS320 DSP hotline, (713) 274-2320.

The major topics discussed in this section are listed below.

- Development Support (Section E.1 on page E-3)
  - TMS320C2x Macro Assembler/Linker
  - TMS320C2x Simulator
  - TMS320C2x SoftWare Development System (SWDS)
  - TMS320C25 C Compiler
  - TMS320C2x Emulator (XDS/22)
  - TMS320C2x XDS/22 Upgrade
  - TMS320 Analog Interface Board
  - TMS320 Design Kit
  - Digital Filter Design Package (DFDP)
  - DSP Software Library
  - TMS320 DSP Hotline/Bulletin Board Service

- Part Order Information (Section E.2 on page E-11)
  - Device part numbers
  - Software and hardware support tools part numbers
  - Device and support tool prefix designators
  - Device and support tool nomenclature

E.1 Second-Generation TMS320 Development Support

Texas Instruments offers extensive development support and complete documentation with the second-generation TMS320 digital signal processors. Tools are provided to evaluate the performance of the processors, develop algorithm implementations, and fully integrate the design's software and hardware modules. Development operations are performed with the TMS320C2x Macro Assembler/Linker, Simulator, SoftWare Development System (SWDS), C Compiler, Emulator (XDS), and other support products.

A description and key features for each TMS320C2x development support tool is provided in the following subsections. For more information about support products, refer to the *TMS320 Family Development Support Reference Guide*. For ordering information, see Section E.2.

E.1.1 TMS320C2x Macro Assembler/Linker

The TMS320C2x Macro Assembler translates TMS320C2x assembly language source code into executable object code. The assembler allows the programmer to work with mnemonics rather than hexadecimal machine instructions and to reference memory locations with symbolic addresses. The macro assembler supports macro calls and definitions along with conditional assembly.

The TMS320C2x Linker permits a program to be designed and implemented in separate modules that will later be linked together to form the complete program. The linker resolves external definitions and references for relocatable code, creating an object file that can be executed by the TMS320C2x Simulator, Emulator, or DSP device. The output of the linker can be downloaded into the simulator, XDS, SWDS, or PROM programmer.

The following key features distinguish the TMS320C2x Macro Assembler/Linker:

- Macro capabilities and library functions
- Conditional assembly
- Relocatable modules
- Complete error diagnostics
- Symbol table and cross reference.

The macro assembler/linker is currently available for the VAX/VMS and MS/PC-DOS operating systems.

E.1.2 TMS320C2x Simulator

The TMS320C2x Simulator is a software program that simulates operation of the TMS320C2x to allow program verification. In the debug mode, the state of the simulated TMS320C2x can be monitored while the program is executing. The simulator uses the object code produced by the TMS320C2x Macro Assembler/Linker. During program execution, the internal registers and memory of the simulated device are modified as each instruction is interpreted by the host computer. Once program execution is suspended, the internal registers and both program and data memories can be inspected and/or modified. In addition, files can be associated with the I/O ports.

The following features highlight simulator capability for effective TMS320C2x software development:

- Program debug/verification
- Single-step option
- Trace/breakpoint capabilities
- Full access to simulated registers and memories
- I/O device simulation.

The simulator is currently available for the VAX/VMS and MS/PC-DOS operating systems.

E.1.3 TMS320C2x SoftWare Development System (SWDS)

The SoftWare Development System (SWDS) is a PC-resident tool that allows software simulation in realtime for the TMS320C2x. The SWDS provides the system interface necessary to write, assemble/link, load, and debug the TMS320C2x code on a PC workstation. The SWDS is capable of single-stepping through the code or setting software breakpoints for monitoring register or memory contents during execution. It can also associate files with I/O ports so that specific I/O values may be used during test and debug.

The SWDS consists of three parts:

1) A circuit board, resident in the PC, that contains the TMS320C2x and program and data memory.
2) Two small cable adaptor boards situated outside the PC and connected to the SWDS via two 40-conductor ribbon cables. The cable adaptor boards included with the system are:
    a) The PGA Adaptor Connector that connects the SWDS to a TMS320C2x target system via a 68-pin grid array footprint.
    b) The Analog Interface Board (AIB) Adaptor Connector that connects the SWDS directly to the TMS320 AIB.
3) Software that includes TMS320C2x assembler/linker software, the DSP Software Library (see Section E.1.10), and SWDS monitor software.

The SWDS is designed to function in the IBM-PC/AT and compatible environment, as well as in any TI PC environment (including the TI Business Pro). MS-DOS version 2.0 or later is required.

The development system occupies 64K bytes of PC memory. It is equipped with 24K words (48 kbytes) of static RAM, and allows the TMS320C2x to execute at full speed. Note that the SWDS does not address target memory.

The SWDS is configured to emulate the TMS320C25 upon shipment; i.e., a TMS320C25 and a 40-MHz oscillator are on-board. A TMS32020 and a 20-MHz crystal are included with the system to accommodate TMS32020 emulation. The target system may supply a TTL clock source, in which case the upper limit on the clock speed is dictated by the speed of the processor on the PC board. If the user's target system has no provision for a clock source, the external clock is specified in the debug monitor initialization command and an external crystal is connected to the SWDS.

E.1.4 TMS320C25 C Compiler

A full Kernigan and Ritchie C compiler is provided for the TMS320C25. The compiler accepts a digital signal processing program written in the widely used C language and outputs TMS320C2x assembly language source code. The TMS320C2x mnemonics are then converted to object code by a TMS320C2x assembler.

This high-level language compiler allows time-critical routines written in assembly language to be called from within the C program. The converse is also available; assembly routines may call C functions. The output of the compiler can be edited prior to assembly/link to further optimize the performance of the code. The compiler is also capable of accepting other programs written in C. Refer back to Figure E-1 for a diagram of the development environment when using the C compiler.

Included with the shipment of the TMS320C25 C compiler is an enhanced assembler/linker. The output of this assembler/linker can be downloaded and used with any of the existing tools (simulator, XDS, SWDS, or PROM programmer).

The compiler is currently available for the VAX/VMS and MS/PC-DOS operating systems.

E.1.5 TMS320C2x Emulator (XDS)

The TMS320C2x Emulator (XDS/22) is a user-friendly system that has all the features necessary for realtime in-circuit emulation. This allows integration of hardware and software modules in the debug mode. By setting breakpoints based on internal conditions or external events, execution of the program can be suspended and control given to the debug mode. In the debug mode, all registers and memory locations can be inspected and modified. Single-step execution is available. Full-trace capabilities at full speed and a reverse assembler that translates machine code back into assembly instructions also increase debugging productivity. Using a standard RS-232-C port, the object file produced by the TMS320C2x Macro Assembler/Linker can be downloaded into the emulator, which then can be controlled through a terminal.

Two 4K x 16-word banks of high-speed static RAM can be mapped into a fixed address space starting at 0 for both program and data memory. Also available are 64K words of dynamic RAM, which can be mapped into the user's program and data address spaces. The XDS is capable of executing out of target memory to utilize the full TMS320C2x program/data address range.

For multiprocessing configurations, up to nine emulators can be daisy-chained together.

The XDS/22 emulator is a completely self-contained system with power supply. With three RS-232-C ports, the XDS/22 Emulator can be interfaced to a terminal, host computer for source or object downloading/uploading capabilities, and printer or PROM programmer.

The key features of the TMS320C2x XDS/22 Emulator are as follows:

- 40-MHz full-speed in-circuit emulation
- Supports all second-generation TMS320 family members
- PLCC target connector with pin grid array (PGA) adaptor
- 4K words each of program and data memory (zero wait states)
- 64K-word PROM memory expansion board (wait states)
- Breakpoint, trace, and timing (BTT) capabilities
- Single-step execution
- Line-by-line assembler/reverse assembler
- Enhanced decimal parameter entry and display
- Use of target system CLKIN signal or internal crystal
- Host-independent upload/download capabilities to/from program or data memory
- Ability to inspect and modify registers and program/data memory
- Supports multiprocessor configurations
- Logic tracing with extended data/address logic analyzer interface.

Figure E-2 shows a block diagram of a typical system configuration using the TMS320C2x XDS/22 Emulator.

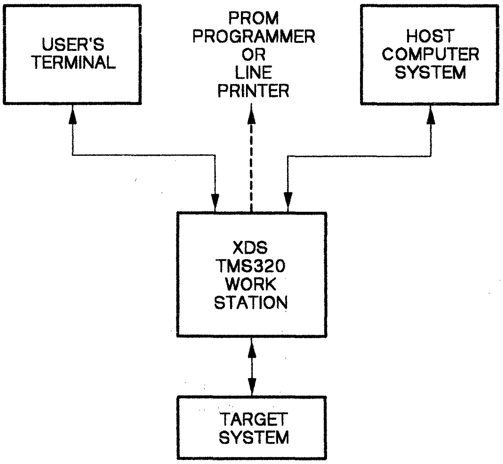


Figure E-2. TMS320C2x XDS/22 System Configuration

E.1.6 TMS320C2x XDS/22 Upgrade

Texas Instruments offers a TMS320C2x XDS upgrade kit, which can extend the functionality of existing development systems at a minimum of cost through an enhancement of current customer equipment. For example, the upgrade kit can enable a TMS32020 XDS/22 to emulate operation of both second-generation devices. Upgrade kits allow upgrade only within a generation, not from a first- to a second-generation XDS.

The TMS320C2x XDS upgrade kit consists of the following contents:

- Firmware (2 PALs, 2 EPROMs)
- TMS320C25 and crystal
- 4K x 16 high-speed static RAM (2 sets)
- 40-MHz breakpoint, trace, and timing board
- PGA/PLCC target connector.

E.1.7 TMS320 Analog Interface Board

The TMS320 Analog Interface Board (AIB) is an analog-to-digital, digital-to-analog conversion board used as a preliminary target system with either the TMS320C2x SWDS, XDS, or another emulator (see Figure E-3). The AIB is an educational tool that provides a simple, inexpensive way to become familiar with digital signal processing (DSP) techniques.

The AIB allows testing of application programs with analog I/O by providing an interface to the TMS320C2x. The AIB provides 12-bit A/D and D/A converters with expansion ports for additional A/D and D/A converters. Key features of the AIB are as follows:

- 12-bit analog-to-digital converter with sample and hold
- 12-bit digital-to-analog converter
- One 16-bit output port for additional D/A or user-defined application
- One 16-bit input port for additional A/D or user-defined application
- Two lowpass filters; an audio amplifier
- Prototyping area for user applications.

The AIB runs at full speed up to 20 MHz for TMS320 family members. The AIB sample rate clock, derived from the TMS320 CLKOUT signal, may be programmed to provide periodic analog input, output, or both. There are two analog lowpass filters on the AIB. One filter on the A/D input band-limits the input to minimize aliasing effects. The other filter smooths the output of the D/A. Filter frequency response is controlled by varying the external components in the filter stages. Filter cutoff is set to 4.7 kHz, but may be (plug) programmed. An audio amplifier that drives an 8-ohm speaker is provided for applications with audio output.

An AIB adaptor board is required to convert the 40-pin dual-inline socket for the TMS320C1x to accommodate the 68-pin grid array package of the TMS32020. An additional adaptor socket is necessary for TMS320C25 operation. Contact the nearest TI Sales Office for a list of commercially available adaptor socket vendors.

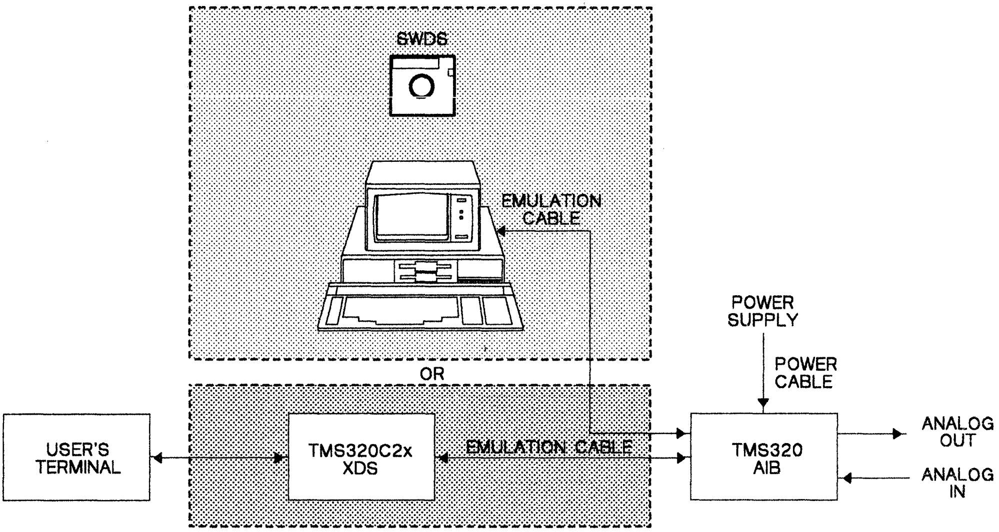


Figure E-3. TMS320 AIB System Configuration

E.1.8 TMS320 Design Kit

The TMS320 DSP Design Kit has been created by Texas Instruments to aid the user in becoming familiar with the TMS320 family of digital signal processors, thus accelerating the evaluation of these devices. The kit contains the following:

- Samples: one TMS32020GBL, one TMS32010NL, one Codec (TCM2916), and four preprogrammed PROMs (TBP38L165-45).
- ADPCM Design Example using the TMS32010.
- FFT Design Example using the TMS32020.
- *Digital Signal Processing Applications with the TMS320 Family*, a comprehensive 750-page book filled with TMS320 applications.
- Digital Signal Processing Software Library, containing source code for most of the DSP applications discussed in the Applications Book as well as other valuable routines.
- TMS320C1x and TMS320C2x User's Guides.
- Latest copy of the TMS320 quarterly newsletter, *Details on Signal Processing*.

The Design Kit is available through local TI authorized distributors or directly from Texas Instruments. Contact the nearest TI Sales Office for more information.

E.1.9 Digital Filter Design Package (DFDP)

The Digital Filter Design Package (DFDP) from Atlanta Signal Processors, Inc. (ASPI) is a user-friendly, menu-driven software package intended to speed the design of digital filters with floating-point accuracy or fixed-point economy in a variety of filter structures. The package consists of four interactive filter design modules capable of performing the following functions:

1) Designing FIR filters (Kaiser window)
2) Designing FIR filters (Parks-McClellan)
3) Designing IIR filters (Butterworth, Chebychev I and II, and elliptic)
4) Generating TMS320C1x or TMS320C2x assembly code by converting the ASCII file containing the filter coefficients into fully commented assembly language code for those devices.

Cascade and parallel structures as well as higher-performance lattice, normalized lattice, and orthogonal forms are included in the modules.

The DFDP can design filters to meet any piecewise linear response specification, evaluate filter characteristics before and after coefficient quantization, and design special-purpose FIR filters, such as multiband filters, differentiators, Hilbert transformers, and raised-cosine filters. The DFDP can also generate coefficients for filter implementations on any general-purpose processor or signal processing chip, as well as fully commented assembly language code for a variety of DSP chips. Magnitude, log magnitude, and impulse responses can be plotted for printer or screen display; in addition, the phase, group delay, and pole-zero map can be plotted for IIR filters. After the filter is designed, the user can generate code associated with the filter using the CGEN design module.

The DFDP runs on the IBM PS/2, IBM PC/XT/AT, and compatible systems. Operating systems must have 192 kbytes of memory available. For more information, contact Atlanta Signal Processors, Inc. (404-892-7265) or the nearest TI field sales office.

E.1.10 DSP Software Library

The Digital Signal Processing Software Library contains the major DSP routines (FFT, FIR/IIR filtering, and floating-point operations) and application algorithms (echo cancellation, ADPCM, and DTMF coding/decoding) presented in the book, *Digital Signal Processing Applications with the TMS320 Family.* These routines and algorithms are written in either TMS320C1x and/or TMS320C2x source code. In addition, macros for the TMS320C1x are included in the library.

The software package consists of four diskettes for use with the TI/IBM MS/PC-DOS (version 1.1 or later) or a 1600 BPI magnetic tape for the VAX/VMS version. All the directories on the MS/PC-DOS version are contained on the magnetic tape for the VMS version. Each directory contains a README.LIS file briefly describing the contents of the files in the directory and the reference to the code. The book, *Digital Signal Processing Applications with the TMS320 Family,* is the major reference for the theory and algorithms, and also provides printed code in the appendices of each application report.

The software library and applications book are included in the purchase of a TMS320 Design Kit (see Section E.1.7). The library can also be ordered separately through TI (see Table E-2 for ordering information).

All the software in the library is copyrighted by Texas Instruments. The library is continually being updated; therefore, check the TMS320 DSP Bulletin Board (713-274-2323) for update information.

E.1.11 TMS320 DSP Hotline/Bulletin Board Service

The TMS320 group at Texas Instruments provides a DSP Hotline to answer TMS320 technical questions such as device problems, development tools, documentation, upgrades, and new TMS320 products. The hotline is open five days a week from 8:00 AM to 4:30 PM Central Time. The phone number is (713) 274-2320. For pricing and availability of TMS320 devices and development tools, contact the nearest TI sales office. To order literature, call the Customer Response Center (CRC) at (800) 232-3200.

The TMS320 DSP Bulletin Board Service is a telephone-line computer bulletin board that provides access to information pertaining to TMS320 devices. Specification updates for current or new TMS320 devices and development tools are communicated via the bulletin board as the information becomes available. The Bulletin Board Service can be accessed by dialing (713) 274-2323 with a 1200-bps modem.

The bulletin board contains TMS320 source code from the application reports included in the book, *Digital Signal Processing Applications with the TMS320 Family*. The bulletin board also provides new DSP application software as it becomes available. See the *TMS320 Family Development Support Reference Guide* for information on how to access the bulletin board.

E.2 Part Order Information

This section provides the device and support tool part numbers. Table E-1 lists the part numbers for all the second-generation members of the TMS320 family. Table E-2 gives ordering information for TMS320C2x hardware and software support tools. Table E-3 provides a list and description of the development tool connections to a target system. A discussion of the TMS320 family device and development support tool prefix and suffix designators is included to assist in understanding the TMS320 product numbering system.

Table E-1. TMS320C2x Digital Signal Processor Part Numbers

DEVICE	TECHNOLOGY	OPERATING FREQUENCY	PACKAGE TYPE	TYPICAL DISSIPATION
TMS32020GBL	2.4-µm NMOS	20 MHz†	Ceramic 68-pin PGA	1200 mW
TMS320C25GBL	1.8-µm CMOS	40 MHz‡	Ceramic 68-pin PGA	450 mW
TMS320C25FNL	1.8-µm CMOS	40 MHz‡	Plastic 68-lead PLCC	450 mW

†Military version available.
‡Military version planned; contact nearest sales office for availability.

Table E-2. TMS320C2x Support Tool Part Numbers

TOOL DESCRIPTION	OPERATING SYSTEM	PART NUMBER
SOFTWARE		
Macro Assembler/Linker	VAX VMS MS/PC-DOS	TMDS3242210-08 TMDS3242810-02
Simulator	VAX VMS MS/PC-DOS	TMDS3242211-08 TMDS3242811-02
SoftWare Development System	MS/PC-DOS	TMDS3268821
C Compiler (TMS320C25)	VAX VMS MS/PC-DOS	TMDS3242255-08 TMDS3242855-02
Digital Filter Design Package	IBM PC-DOS	DFDP/IBM002
DSP Software Library	VAX VMS MS/PC-DOS	TMDC3240212-18 TMDC3240812-12
HARDWARE		
XDS/22 Emulator†		TMDS3262221
XDS/22 Upgrade: Customer Upgrade†		TMDS3282226
Analog Interface Board		RTC/EVM320C-06
Analog Interface Board Adaptor		RTC/ADP320A-06
TMS320 Design Kit		TMS320DDK

†See Table E-3 for a list of connections to a target system.

Table E-3. Development Tool Connections to a Target System

TOOL	TARGET CONN.	INCL.	OPT.	PART NUMBER
TMS320C25 XDS/22	68-pin PGA/PLCC	X		
	68-pin PGA/PLCC		X	TMDX3288825
	68-pin PGA		X	TMDX3288820
TMS320C25 XDS/22 Upgrade	68-pin PGA/PLCC	X		
	68-pin PGA/PLCC		X	TMDX3288825
	68-pin PGA		X	TMDX3288820

E.2.1 Device and Development Support Tool Prefix Designators

To assist the user in understanding the stages in the product development cycle, Texas Instruments assigns prefix designators in the part number nomenclature. A device prefix designator has three options: TMX, TMP, and TMS, and a development support tool prefix designator has two options: TMDX and TMDS. These prefixes are representative of the evolutionary stages of product development from engineering prototypes (TMX/TMDX) through fully qualified production devices (TMS/TMDS). This development flow is defined below.

Device Development Evolutionary Flow:

TMX	Experimental device that is not necessarily representative of the final device's electrical specifications.
TMP	Final silicon die that conforms to the device's electrical specifications but has not completed quality and reliability verification.
TMS	Fully qualified production device.

Support Tool Development Evolutionary Flow:

TMDX	Development support product that has not yet completed Texas Instruments internal qualification testing.
TMDS	Fully qualified development support product.

TMX and TMP devices and TMDX development support tools are shipped against the following disclaimer:

> "Developmental product is intended for internal evaluation purposes."

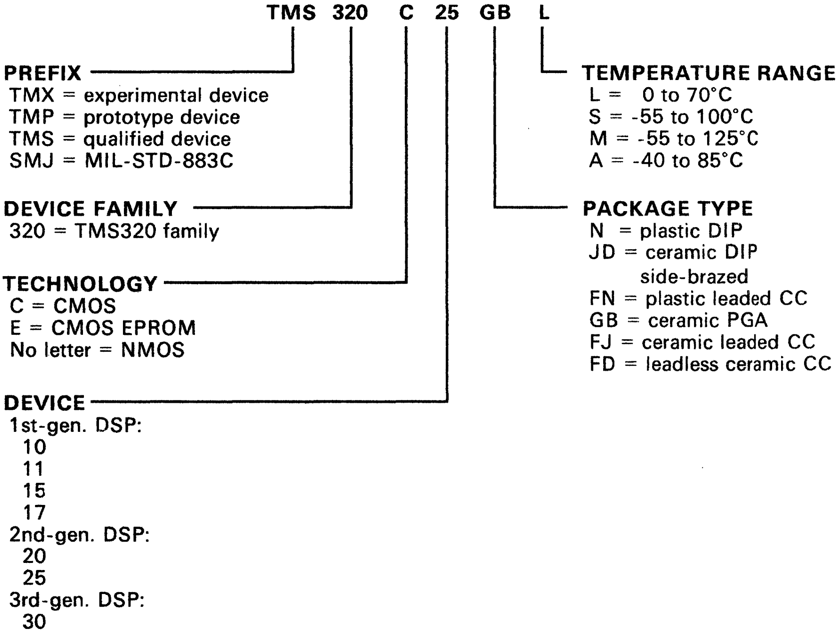
Note:

Texas Instruments recommends that prototype devices (TMX or TMP) not be used in production systems since their expected end-use failure rate is undefined but predicted to be greater than standard qualified production devices.

TMS devices and TMDS development support tools have been fully characterized and the quality and reliability of the device has been fully demonstrated. Texas Instruments standard warranty applies.

E.2.2 Device and Development Support Tool Nomenclature

In addition to the prefix, the device nomenclature includes a suffix that follows the device family name. This suffix indicates the package type (e.g., N, FN, or GB) and temperature range (e.g., L). Figure E-4 provides a legend for reading the complete device name for any TMS320 family member.



TMS 320 C 25 GB L

PREFIX
TMX = experimental device
TMP = prototype device
TMS = qualified device
SMJ = MIL-STD-883C

DEVICE FAMILY
320 = TMS320 family

TECHNOLOGY
C = CMOS
E = CMOS EPROM
No letter = NMOS

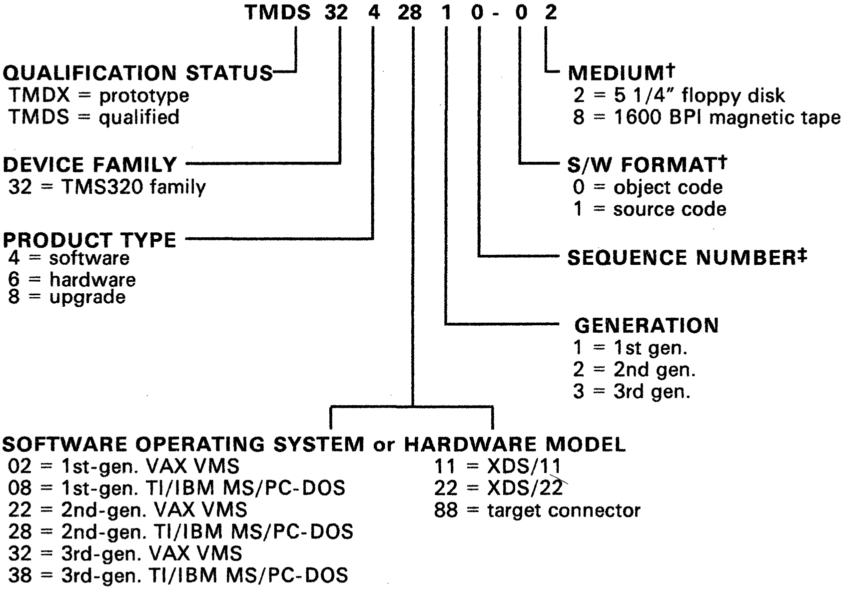
DEVICE
1st-gen. DSP:
10
11
15
17
2nd-gen. DSP:
20
25
3rd-gen. DSP:
30

TEMPERATURE RANGE
L = 0 to 70°C
S = -55 to 100°C
M = -55 to 125°C
A = -40 to 85°C

PACKAGE TYPE
N = plastic DIP
JD = ceramic DIP side-brazed
FN = plastic leaded CC
GB = ceramic PGA
FJ = ceramic leaded CC
FD = leadless ceramic CC

Figure E-4. TMS320 Device Nomenclature

Figure E-5 provides a legend for reading the part number for any TMS320 hardware or software development tool.



TMDS 32 4 28 1 0 - 0 2

QUALIFICATION STATUS
TMDX = prototype
TMDS = qualified

DEVICE FAMILY
32 = TMS320 family

PRODUCT TYPE
4 = software
6 = hardware
8 = upgrade

MEDIUM†
2 = 5 1/4″ floppy disk
8 = 1600 BPI magnetic tape

S/W FORMAT†
0 = object code
1 = source code

SEQUENCE NUMBER‡

GENERATION
1 = 1st gen.
2 = 2nd gen.
3 = 3rd gen.

SOFTWARE OPERATING SYSTEM or HARDWARE MODEL
02 = 1st-gen. VAX VMS
08 = 1st-gen. TI/IBM MS/PC-DOS
22 = 2nd-gen. VAX VMS
28 = 2nd-gen. TI/IBM MS/PC-DOS
32 = 3rd-gen. VAX VMS
38 = 3rd-gen. TI/IBM MS/PC-DOS

11 = XDS/11
22 = XDS/22
88 = target connector

† Software only.
‡ Hardware only.

Figure E-5. TMS320 Development Tool Nomenclature

F. Memories, Analog Converters, Sockets, and Crystals

This appendix provides product information regarding memories, analog converters, and sockets, which are manufactured by Texas Instruments and compatible with the TMS320C2x. Information is also given regarding crystal frequencies, specifications, and vendors.

The contents of the major areas in this appendix are listed below.

- TI Memories and Analog Converters (Section F.1 on page F-2)
  - EPROM memories
  - Codecs and filters
  - Analog interface circuits
  - A/D and D/A converters.
- TI Sockets for PGA and PLCC Packages (Section F.2 on page F-28)
  - Production sockets
  - Burn-in/test sockets.
- Crystals (Section F.3 on page F-33)
  - Commonly used crystal frequencies
  - Crystal specification requirements
  - Vendors of suitable crystals.

F.1 TI Memories and Analog Converters

This section provides pages of product information taken from data sheets for EPROM memories, codecs, analog interface circuits, and D/A and D/A converters.

All of these devices can be interfaced with TMS320C2x processors (see Section 6 for hardware interface designs). Refer to *Digital Signal Processing Applications with the TMS320 Family* for additional information on interfaces using memories and analog conversion devices.

The following paragraphs give the name of each device and where the data sheet for that device is located in order to obtain further specification information if desired.

Data sheets for EPROM memories are located in the *MOS Memory Data Book* (SMYD006). The name of the device and the page number in the book on which the device is introduced are listed.

TMS27C64	(page 6-21)
TMS27C128	(page 6-29)
TMS27C256	(page 6-37)
TMX27C512	(page 6-45)

Another EPROM memory, TMS27C291/292, is described in a data sheet (SMLS291A).

The TCM29C13/14/16/17 codecs and filters are described in the data sheet beginning on page 2-111 of the *Telecommunications Circuits Data Book* (SCT001). An analog interface for the DSP using a codec and filter is provided by the TCM29C18/19 (data sheet number SCT021).

The data sheet for the TLC32040 analog interface circuit is provided in the *Interface Circuits Data Book*, beginning on page 2-271.

In the same book are data sheets for A/D and D/A converters. The name of the device and the page on which it is introduced are as follows:

TLC0820	(page 2-113)
TLC1205/1225	(page 2-181)
TLC7524	(page 2-243)

ADVANCE INFORMATION

TMS27C64
65,536-BIT ERASABLE PROGRAMMABLE READ-ONLY MEMORY

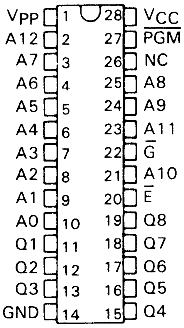
SEPTEMBER 1985

- **Organization . . . 8K × 8**
- **Single 5-V Power Supply**
- **Pin Compatible with Existing 64K EPROMs and TMS2732A**
- **All Inputs/Outputs Fully TTL Compatible**
- **Max Access/Min Cycle Time**

'27C64-1,	**'27C64-15**	**150 ns**
'27C64-2,	**'27C64-20**	**200 ns**
'27C64,	**'27C64-25**	**250 ns**
'27C64-3,	**'27C64-30**	**300 ns**
'27C64-4,	**'27C64-45**	**450 ns**

- **HVCMOS Technology**
- **3-State Output Buffers**
- **400 mV Guaranteed DC Noise Immunity with Standard TTL Loads**
- **Low Power Dissipation (V_{CC} = 5.25 V)**
  —Active . . . 158 mW Worst Case
  —Standby . . . 1.4 mW Worst Case
  (CMOS-Input Levels)

J PACKAGE
(TOP VIEW)



PIN NOMENCLATURE	
A0-A12	Address Inputs
$\overline{E}$	Chip Enable/Power Down
$\overline{G}$	Output Enable
GND	Ground
NC	No Connection
$\overline{PGM}$	Program
Q1-Q8	Outputs
V_{CC}	5-V Power Supply
V_{PP}	12.5-V Power Supply

description

The TMS27C64 series are 65,536-bit, ultraviolet-light erasable, electrically programmable read-only memories. These devices are fabricated using HVCMOS technology for high speed and simple interface with MOS and bipolar circuits. All inputs (including program data inputs) can be driven by Series 74 TTL circuits without the use of external pull-up resistors, and each output can drive one Series 74 TTL circuit without external resistors. The data outputs are three-state for connecting multiple devices to a common bus. The TMS27C64 is pin compatible with existing 28-pin ROMs and EPROMs. It is offered in a dual-in-line ceramic package (J suffix) rated for operation from 0°C to 70°C.

Since these EPROMs operate from a single 5-V supply (in the read mode), they are ideal for use in microprocessor-based systems. One other (12.5 V) supply is needed for programming, but all programming signals are TTL level. For programming outside the system, existing EPROM programmers can be used. Locations may be programmed singly, in blocks, or at random.

operation

There are seven modes of operation for the TMS27C64 listed on the following page. Read mode requires a single 5-V supply. All inputs are TTL level except for Vpp during programming (12.5 V) and 12 V on A9 for signature mode.

TEXAS INSTRUMENTS
POST OFFICE BOX 1443 • HOUSTON, TEXAS 77001

ADVANCE INFORMATION

TMS27C128
131,072-BIT ERASABLE PROGRAMMABLE READ-ONLY MEMORY

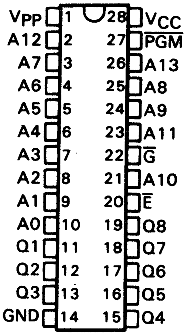
OCTOBER 1984—REVISED SEPTEMBER 1985

- **Organization . . . 16K × 8**
- **Single 5-V Power Supply**
- **Pin Compatible with Existing 64K and 128K EPROMs**
- **All Inputs/Outputs Fully TTL Compatible**
- **Max Access/Min Cycle Time**

'27C128-1,	**'27C128-15**	**150 ns**
'27C128-2,	**'27C128-20**	**200 ns**
'27C128,	**'27C128-25**	**250 ns**
'27C128-3,	**'27C128-30**	**300 ns**
'27C128-4,	**'27C128-45**	**450 ns**

- **HVCMOS Technology**
- **3-State Output Buffers**
- **400 mV Guaranteed DC Noise Immunity with Standard TTL Loads**
- **Low Power Dissipation (V_{CC} = 5.25 V)**
  —Active . . . 158 mW Worst Case
  —Standby . . . 1.4 mW Worst Case
  (CMOS-Input Levels)

J PACKAGE
(TOP VIEW)



PIN NOMENCLATURE	
A0-A13	Address Inputs
$\overline{E}$	Chip Enable/Power Down
$\overline{G}$	Output Enable
GND	Ground
$\overline{PGM}$	Program
Q1-Q8	Outputs
V_{CC}	5-V Power Supply
V_{PP}	12.5-V Power Supply

description

The TMS27C128 series are 131,072-bit, ultraviolet-light erasable, electrically programmable read-only memories. These devices are fabricated using HVCMOS technology for high speed and simple interface with MOS and bipolar circuits. All inputs (including program data inputs) can be driven by Series 74 TTL circuits without the use of external pull-up resistors, and each output can drive one Series 74 TTL circuit without external resistors. The data outputs are three state for connecting multiple devices to a common bus. The TMS27C128 is pin compatible with existing 28-pin ROMs and EPROMs. It is offered in a dual-in-line ceramic package (J suffix) rated for operation from 0°C to 70°C.

Since these EPROMs operate from a single 5-V supply (in the read mode), they are ideal for use in microprocessor-based systems. One other (12.5 V) supply is needed for programming, but all programming signals are TTL level. For programming outside the system, existing EPROM programmers can be used. Locations may be programmed singly, in blocks, or at random.

operation

There are seven modes of operation for the TMS27C128 listed on the following page. Read mode requires a single 5-V supply. All inputs are TTL level except for V_{PP} during programming (12.5 V) and 12 V on A9 for signature mode.

TEXAS INSTRUMENTS
POST OFFICE BOX 1443 • HOUSTON, TEXAS 77001

TMS27C256
262,144-BIT ERASABLE PROGRAMMABLE READ-ONLY MEMORY

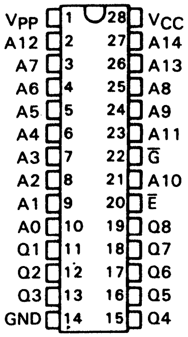
SEPTEMBER 1984 – REVISED NOVEMBER 1985

- **Organization . . . 32K × 8**
- **Single 5-V Power Supply**
- **Pin Compatible with Existing 128K and 256K EPROMs**
- **All Inputs/Outputs Fully TTL Compatible**
- **Max Access/Min Cycle Time**

'27C256-1,	**'27C256-17**	**170 ns**
'27C256-2,	**'27C256-20**	**200 ns**
'27C256,	**'27C256-25**	**250 ns**
'27C256-3,	**'27C256-30**	**300 ns**
'27C256-4,	**'27C256-45**	**450 ns**

- **HVCMOS Technology**
- **3-State Output Buffers**
- **400 mV Guaranteed DC Noise Immunity with Standard TTL Loads**
- **Low Power Dissipation (V_{CC} = 5.25 V)**
  - **Active . . . 210 mW Worst Case**
  - **Standby . . . 1.4 mW Worst Case (CMOS-Input Levels)**

J PACKAGE
(TOP VIEW)



PIN NOMENCLATURE	
A0-A14	Address Inputs
$\overline{E}$	Chip Enable/Power Down
$\overline{G}$	Output Enable
GND	Ground
Q1-Q8	Outputs
V_{CC}	5-V Power Supply
V_{PP}	12.5-V Power Supply

description

The TMS27C256 series are 262,144-bit, ultraviolet-light erasable, electrically programmable read-only memories. These devices are fabricated using HVCMOS technology for high speed and simple interface with MOS and bipolar circuits. All inputs (including program data inputs) can be driven by Series 74 TTL circuits without the use of external pull-up resistors, and each output can drive one Series 74 TTL circuit without external resistors. The data outputs are three state for connecting multiple devices to a common bus. The TMS27C256 is pin compatible with existing 28-pin ROMs and EPROMs. It is offered in a dual-in-line ceramic package (J suffix) rated for operation from 0°C to 70°C.

Since these EPROMs operate from a single 5-V supply (in the read mode), they are ideal for use in microprocessor-based systems. One other (12.5 V) supply is needed for programming, but all programming signals are TTL level. For programming outside the system, existing EPROM programmers can be used. Locations may be programmed singly, in blocks, or at random.

operation

There are seven modes of operation for the TMS27C256 listed on the following page. Read mode requires a single 5-V supply. All inputs are TTL level except for Vpp during programming (12.5 V) and 12 V on A9 for signature mode.

TEXAS INSTRUMENTS
POST OFFICE BOX 1443 • HOUSTON, TEXAS 77001

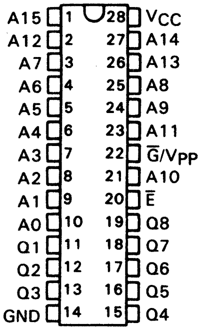
ADVANCE INFORMATION

TMS27C512
524,288-BIT ERASABLE PROGRAMMABLE READ-ONLY MEMORY

NOVEMBER 1985 – REVISED AUGUST 1986

- **Organization . . . 64K × 8**
- **Single 5-V Power Supply**
- **Pin Compatible with Existing NMOS 512K EPROMs**
- **All Inputs/Outputs Fully TTL Compatible**
- **Max Access/Min Cycle Time**
  - **'27C512-2, '27C512-20 200 ns**
  - **'27C512, '27C512-25 250 ns**
  - **'27C512-3, '27C512-30 300 ns**
  - **'27C512-4, '27C512-45 450 ns**
- **HVCMOS Technology**
- **3-State Output Buffers**
- **400 mV Guaranteed DC Noise Immunity with Standard TTL Loads**
- **Low Power Dissipation (V_{CC} = 5.25 V)**
  - **—Active . . . 263 mW Worst Case**
  - **—Standby . . . 1.4 mW Worst Case (CMOS-Input Levels)**

J PACKAGE
(TOP VIEW)



PIN NOMENCLATURE	
A0-A15	Address Inputs
E̅	Chip Enable/Power Down
GND	Ground
Q1-Q8	Outputs
V_{CC}	5-V Power Supply
G̅/V_{PP}	12.5-V Power Supply/ Output Enable

description

The TMS27C512 series are 524,288-bit, ultraviolet-light erasable, electrically programmable read-only memories. These devices are fabricated using HVCMOS technology for high speed and simple interface with MOS and bipolar circuits. All inputs (including program data inputs) can be driven by Series 74 TTL circuits without the use of external pull-up resistors, and each output can drive one Series 74 TTL circuit without external resistors. The data outputs are three state for connecting multiple devices to a common bus. The TMS27C512 is pin compatible with existing 28-pin ROMs and EPROMs. It is offered in a dual-in-line ceramic package (J suffix) rated for operation from 0°C to 70°C.

Since these EPROMs operate from a single 5-V supply (in the read mode), they are ideal for use in microprocessor-based systems. One other (12.5 V) supply is needed for programming, but all programming signals are TTL level. For programming outside the system, existing EPROM programmers can be used. Locations may be programmed singly, in blocks, or at random.

operation

There are seven modes of operation for the TMS27C512 listed on the following page. Read mode requires a single 5-V supply. All inputs are TTL level except for Vpp during programming (12.5 V) and 12 V on A9 for signature mode.

TEXAS INSTRUMENTS
POST OFFICE BOX 1443 ● HOUSTON, TEXAS 77001

ADVANCE INFORMATION

TMS27C291, TMS27C292
16,384-BIT UV ERASABLE PROGRAMMABLE READ-ONLY MEMORIES

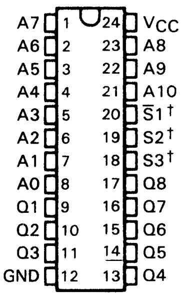
SEPTEMBER 1986–REVISED NOVEMBER 1986

- **Organization . . . 2K × 8**
- **Single 5-V Power Supply**
- **Pin Compatible with Existing 2K × 8 Bipolar/CMOS PROMs**
- **All Inputs/Outputs TTL Compatible**
- **High Speed**
- **Max Access/Min Cycle Time**

V_{CC} ± 5%		
'27C291-3	**'27C292-3**	**35 ns**
'27C291	**'27C292**	**45 ns**
'27C291-5	**'27C292-5**	**50 ns**
V_{CC} ± 10%		
'27C291-45	**'27C292-45**	**45 ns**
'27C291-50	**'27C292-50**	**50 ns**

- **Low-Power CMOS Technology**
- **3-State Output Buffers**
- **400 mV Guaranteed DC Noise Immunity with Standard TTL Loads**
- **Low Power Dissipation . . . 394 mW Max**
- **Eraseable**
- **100% Pretestable**

J PACKAGE
(TOP VIEW)



PIN NOMENCLATURE	
A0-A10	Address Inputs
GND	Ground
Q1-Q8	Outputs
S̄1, S2, S3	Chip Selects
V_{CC}	5-V Power Supply

†Pins 18-20 have different pin assignments and functions in the program mode (see page 3).

description

The TMS27C291 and TMS27C292 series are 16,384-bit, ultraviolet-light erasable, electrically programmable read-only memories. These devices are fabricated using CMOS technology for high speed and simple interface with MOS and bipolar circuits. All inputs (including program data inputs) can be driven by Series 74 TTL circuits without the use of external pull-up resistors, and each output can drive eight Series 74 TTL circuits without external resistors. The data outputs are three state for connecting multiple devices to a common bus. These devices are pin compatible with existing 24-pin PROMs and EPROMs. They are offered in dual-in-line ceramic packages (J suffix). The package for the TMS27C291 is designed for insertion in mounting-hole rows on 7,62-mm (300-mil) centers, and the package for the TMS27C292 is designed for insertion on 15,24-mm (600-mil) centers. The TMS27C291 and TMS27C292 are guaranteed for operation from 0°C to 70°C.

operation

There are eight modes of operation for the TMS27C291 and TMS27C292 listed on the following page. Read mode requires a single 5-V supply. All inputs are TTL or CMOS levels except for V_{PP} (pin 20) during programming (13.5 V).

FUNCTION (PINS)	MODE										
	Read	Output Disable	Output Disable	Output Disable	Program Verify	Program Inhibit	Fast Program	Blank Check Ones	Blank Check Zeros	Signature	
$\overline{S}1/V_{PP}$[†] (20)	V_{IL}	V_{IH}	X	X	V_{PP}	V_{PP}	V_{PP}	V_{IL}	V_{IL}	V_{IL}	
$S2/\overline{VFY}$[†] (19)	V_{IH}	X	V_{IL}	X	V_{IL}[§]	V_{IH}[§]	V_{IH}[§]	V_{IL}	V_{IH}	V_{IH}	
$S3/\overline{PGM}$[†] (18)	V_{IH}	X	X	V_{IL}	V_{IH}[§]	V_{IH}[§]	V_{IL}[§]	V_{PP}	V_{PP}	V_{IH}	
V_{CC} (24)	V_{CC}	V_{CC}	V_{CC}	V_{CC}	V_{CC}	V_{CC}	V_{CC}	V_{CC}	V_{CC}	V_{CC}	
A9 (22)	X	X	X	X	X	X	X	X	X	V_{PP}	V_{PP}
A0 (8)	X	X	X	X	X	X	X	X	X	V_{IL}	V_{IH}
Q1-Q8 (9-17)	D_{OUT}	HI-Z	HI-Z	HI-Z	D_{OUT}	HI-Z	D_{IN}	Ones	Zeros	CODE MFG 97	 DEV 02

[†] Pin assignment for program mode.
[‡] X can be V_{IL} or V_{IH}.
[§] Programming levels for V_{IL} and V_{IH}.

read/output disable

When the outputs of two or more TMS27C291's or TMS27C292's are connected in parallel on the same bus, the output of any particular device in the circuit can be read with no interference from the competing outputs of the other devices. To read the output of the TMS27C291 or TMS27C292, a low-level signal is applied to $\overline{S}1$ and a high-level signal is applied to S2 and S3. Any other combination of logic states on these three inputs will disable the outputs. Output data is accessed at pins Q1 through Q8.

erasure

Before programming, the TMS27C291 or TMS27C292 is erased by exposing the chip through the transparent lid to high intensity ultraviolet light (wavelength 2537 angstroms). The recommended minimum exposure dose (UV intensity × exposure time) is fifteen watt-seconds per square centimeter. A typical 12 milliwatt per square centimeter, filterless UV lamp will erase the device in 21 minutes. The lamp should be located about 2.5 centimeters above the chip during erasure. It should be noted that normal ambient light contains the correct wavelength for erasure. Therefore, when using the TMS27C291 or TMS27C292, the window should be covered with an opaque label.

ADVANCE INFORMATION

TCM29C13, TCM29C14, TCM29C16, TCM29C17
COMBINED SINGLE-CHIP PCM CODEC AND FILTER

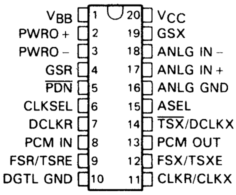
D2765, APRIL 1986

- **Replaces Use of TCM2910A and TCM2911A in Tandem with TCM2912B/C**
- **Reliable Silicon-Gate CMOS Technology**
- **Low Power Consumption:**
  - **Operating Mode . . . 80 mW Typical**
  - **Power-Down Mode . . . 5 mW Typical**
- **Excellent Power Supply Rejection Ratio Over Frequency Range of 0 to 50 kHz**
- **No External Components Needed for Sample, Hold, and Auto-Zero Functions**
- **Precision Internal Voltage References**
- **Direct Replacement for Intel 2913, 2914, 2916, and 2917**
- **Formerly TCM4913, TCM4914, TCM4916, TCM4917, Respectively**

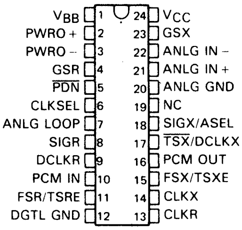
FEATURE TABLE

FEATURE	29C13	29C14	29C16	29C17
Number of Pins:				
24		X		
20	X			
16			X	X
μ-law/A-law Coding:				
μ-law	X	X	X	
A-law	X	X		X
Data Timing Rates:				
Variable Mode				
64 kHz to 2.048 MHz	X	X	X	X
Fixed Mode				
1.536 MHz	X	X		
1.544 MHz	X	X		
2.048 MHz	X	X	X	X
Loopback Test Capability		X		
8th-Bit Signaling		X		

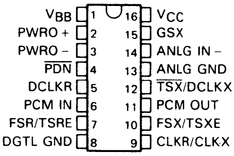
TCM29C13
J DUAL-IN-LINE PACKAGE
(TOP VIEW)



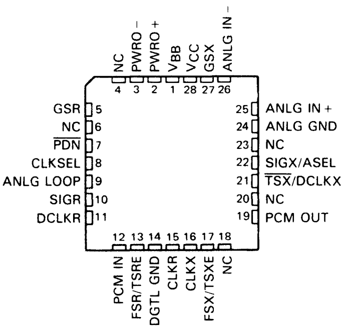
TCM29C14
J DUAL-IN-LINE PACKAGE
(TOP VIEW)



TCM29C16, TCM29C17
J DUAL-IN-LINE PACKAGE
(TOP VIEW)



TCM29C14 . . . FN PACKAGE
(TOP VIEW)



NC—No internal connection

Caution. These devices have limited built-in gate protection. The leads should be shorted together or the device placed in conductive foam during storage or handling to prevent electrostatic damage to the MOS gates.

TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

description

The TCM29C13, TCM29C14, TCM29C16, and TCM29C17 are single-chip pulse-code-modulated encoders and decoders (PCM codecs) and PCM line filters. These devices provide all the functions required to interface a full-duplex (4-wire) voice telephone circuit with a time-division-multiplexed (TDM) system. These devices are intended to replace the TCM2910A or TCM2911A in tandem with the TCM2912B/C. Primary applications of the devices include:

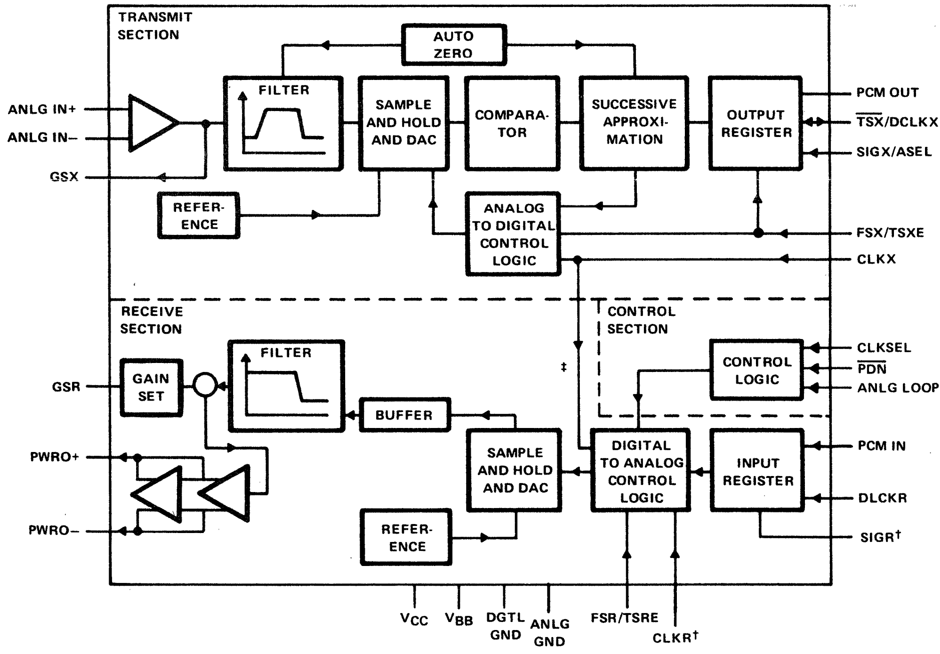
- Line Interface for Digital Transmission and Switching of T1 Carrier, PABX, and Central Office Telephone Systems
- Subscriber Line Concentrators
- Digital Encryption Systems
- Digital Voice Band Data Storage Systems
- Digital Signal Processing

These devices are designed to perform the transmit encoding (A/D conversion) and receive decoding (D/A conversion) as well as the transmit and receive filtering functions in a pulse-code-modulated system. They are intended to be used at the analog termination of a PCM line or trunk.

The TCM29C13, TCM29C14, TCM29C16, and TCM29C17 provide the bandpass filtering of the analog signals prior to encoding and after decoding. These combination devices perform the encoding and decoding of voice and call progress tones as well as the signaling and supervision information.

The TCM29C13, TCM29C14, TCM29C16, and TCM29C17 are characterized for operation from 0 °C to 70 °C.

functional block diagram



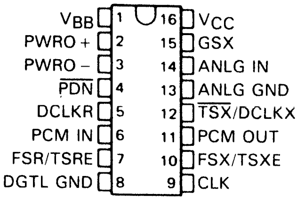
† TCM29C14 ONLY
‡ TCM29C13, TCM29C16, AND TCM29C17 ONLY

TCM29C18, TCM29C19
ANALOG INTERFACE FOR DSP

D3036, AUGUST 1987

- **Reliable Silicon-Gate CMOS Technology**
- **Low Power Consumption**
  Operating Mode . . . 80 mW
  Power-Down Mode . . . 5 mW
  μ-Law Coding
- **Excellent Power Supply Rejection Ratio over Frequency Range of 0 to 50 kHz**
- **No External Components Needed for Sample, Hold, and Auto-Zero Functions**
- **Precision Internal Voltage References**
- **Single Chip Contains A/D, D/A, and Associated Filters**

N DUAL-IN-LINE PACKAGE
(TOP VIEW)



FEATURE TABLE

16 Pins
μ-Law Coding
Variable Mode:
64 kHz to 2.048 MHz
Fixed Mode:
2.048 MHz (TCM29C18),
1.536 MHz (TCM29C19)
8-Bit Resolution
12-Bit Dynamic Range

description

The TCM29C18 and the TCM29C19 are low-cost single-chip pulse-code-modulated encoders and decoders (PCM codecs) and PCM line filters. These devices incorporate both the A/D and D/A functions, an anti-aliasing filter (A/D), and a smoothing filter (D/A). The TCM29C18 and the TCM29C19 are ideal for use with the TMS320 family members, particularly those featuring a serial port such as the TMS32020, TMS32011, and TMS320C25.

Primary applications of these devices include:

Digital Encryption Systems
Digital Voice-Band Data Storage Systems
Digital Signal Processing

These devices are designed to perform encoding of analog input signals (A/D conversion) and decoding of digital PCM signals (D/A conversion). They are useful for implementation in the analog interface of a digital-signal processing system. Both devices also provide band-pass filtering of the analog signals prior to encoding and smoothing after decoding.

The analog input is encoded into an 8-bit digital representation by use of the μ-law encoding scheme (CCITT G.711) which equates to 12 bits of resolution for low amplitude signals. Similarly, the decoding section converts 8-bit PCM data into an analog signal with 12 bits of dynamic range. The filter characteristics (bandpass) for the encoder and decoder are determined by a single clock input (CLK). The filter roll-off (−3 dB) is derived by:

$$f_{CO} = k \cdot f_{CLK}/256 \text{ for the TCM29C18 or } f_{CO} = k \cdot f_{CLK}/192 \text{ for the TCM29C19}$$

where k has a value of 0.44 for the high-frequency roll-off point, and a value of 0.019 for the low-frequency roll-off point.

ADVANCE INFORMATION

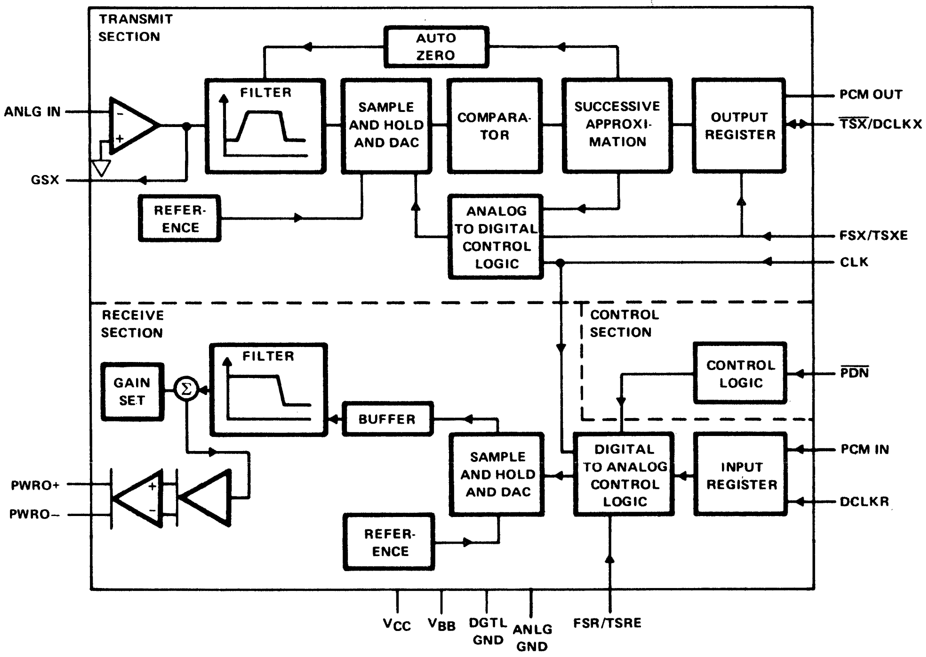
ADVANCE INFORMATION documents contain information on new product in the sampling or preproduction phase of development. Characteristic data and other specifications are subject to change without notice.

description (continued)

The sampling rate of the ADC is determined by the Frame Sync Clock, FSX; the sampling rate of the DAC is determined by the Frame Sync Clock, FSR. Once a conversion is initiated by FSX or FSR, data is clocked in or out on the next consecutive eight clock pulses in the fixed data rate mode. Likewise, data may also be transferred on the next eight consecutive clock pulses of the data clocks, DCLKX and DCLKR, in the variable data rate mode. In the variable data rate mode, DCLKX and DCLKR are independent, but must be in the range from $f_{CLK}/32$ to f_{CLK}.

The TCM29C18 and TCM29C19 are characterized for operation over the temperature range of 0 °C to 70 °C.

functional block diagram

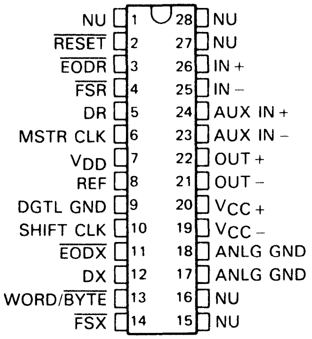


TLC32040M, TLC32040I
ANALOG INTERFACE CIRCUIT

D2964, FEBRUARY 1987

- **ADVANCED LinCMOS™ Silicon Gate Process Technology**
- **14-Bit Dynamic Range ADC and DAC**
- **10-Bit ADC and DAC Linearity Over Any 10-Bit Range**
- **Variable ADC and DAC Sampling Rate Up to 19,200 Samples per Second**
- **Switched-Capacitor Antialiasing Input Filter and Output-Reconstruction Filter**
- **Serial Port for Direct Interface to TMS32011, TMS32020, and TMS32025 Digital Processors**
- **Synchronous or Asynchronous ADC and DAC Conversion Rates with Programmable Incremental ADC and DAC Conversion Timing Adjustments**
- **Serial Port Interface to SN54299 or SN74299 Serial-to-Parallel Shift Registers for Parallel Interface to TMS32010 or Other Digital Processors**

J OR N PACKAGE
(TOP VIEW)



NU Nonusable; no external connection should be made to these pins

PRODUCT PREVIEW

description

The TLC32040 is a complete analog-to-digital and digital-to-analog input/output system on a single monolithic CMOS chip. This device integrates a bandpass switched-capacitor antialiasing input filter, a 14-bit resolution A/D converter, four microprocessor-compatible serial port modes, a 14-bit resolution D/A converter, and a low-pass switched-capacitor output-reconstruction filter. The device offers numerous combinations of Master Clock input frequencies and conversion/sampling rates, which can be changed via digital processor control.

Typical applications for this IC include modems (7.2-, 8-, 9.6-, 14.4-, and 19.2-kHz sampling rate), analog interface for digital signal processors, speech recognition/storage systems, industrial process control, biomedical instrumentation, acoustical signal processing, spectral analysis, data acquisition, and instrumentation recorders. Four serial modes, which allow direct interface to the TMS32011, TMS32020, and TMS32025 digital signal processors, are provided. Also, when the transmit and receive sections of the Analog Interface Circuit (AIC) are operating synchronously, it will interface to two SN54299 or SN74299 serial-to-parallel shift registers. These serial-to-parallel shift registers can then interface in parallel to the TMS32010, other digital signal processors, or external FIFO circuitry. Output data pulses are emitted to inform the processor that data transmission is complete, or to allow the DSP to differentiate between two transmitted bytes. A flexible control scheme is provided so that the functions of the IC can be selected and adjusted coincidentally with signal processing via software control.

The antialiasing input filter comprises seventh-order and fourth-order CC-type (Chebyshev/elliptic transitional) low-pass and high-pass filters, respectively, and a fourth-order equalizer. The input filter is implemented in switched-capacitor technology and is preceded by a continuous time filter to eliminate any possibility of aliasing caused by sampled data filtering. When no filtering is desired, the entire composite filter can be switched out of the signal path. A selectable, auxiliary, differential analog input is provided for applications where more than one analog input is required.

PRODUCT PREVIEW

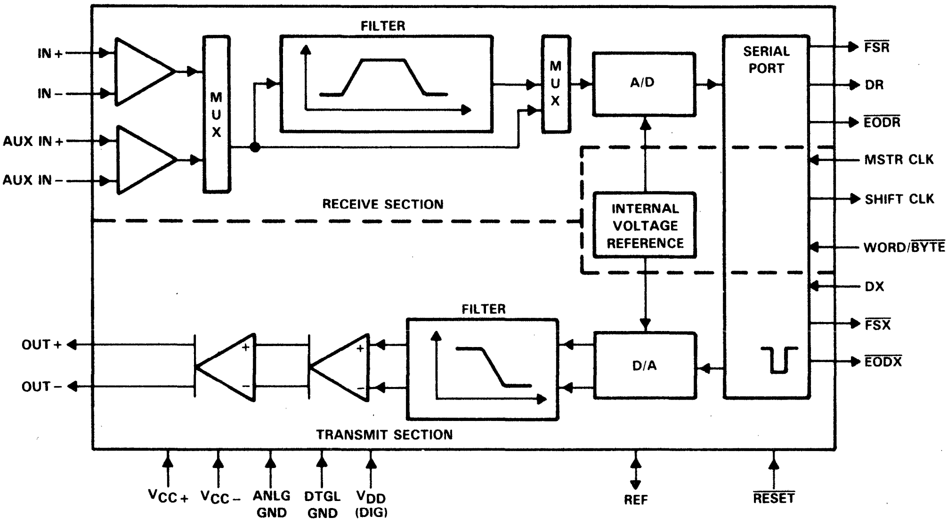
description (continued)

The A/D and D/A converters each have 14 bits of resolution with 10 bits of integral linearity guaranteed over any 10-bit range. The A/D and D/A architectures guarantee no missing codes and monotonic operation. An internal voltage reference is provided to ease the design task and to provide complete control over the performance of the IC. The internal voltage is brought out to a pin and is available to the designer. Separate analog and digital voltage supplies and grounds are provided to minimize noise and ensure a wide dynamic range. Also, the analog circuit path contains only differential circuitry to keep noise to an absolute minimum. The only exception is the DAC sample-and-hold, which utilizes pseudo-differential circuitry.

The output-reconstruction filter is a seventh-order CC-type (Chebyshev/elliptic transitional low-pass filter with a fourth-order equalizer) and is implemented in switched-capacitor technology. This filter is followed by a continuous-time filter to eliminate images of the digitally encoded signal.

The TLC32040M is characterized for operation over the full military temperature range of −55°C to 125°C, and the TLC32040I is characterized for operation from −40°C to 85°C.

functional block diagram



PRINCIPLES OF OPERATION

analog input

Two sets of analog inputs, IN+, IN−, and AUX IN+, AUX IN−, are provided. Each input set can be operated in either differential or single-ended modes, since sufficient common-mode range and rejection are provided. Normally, the IN+ and IN− inputs are used; however, the auxiliary inputs, AUX IN+ and AUX IN−, can be used if a second input is required. The gain for the IN+, IN−, and auxiliary AUX IN+ and AUX IN− inputs can be programmed to either 1, 2, or 4 (see the Gain Control Table). Either input circuit can be selected via software control. It is important to note that a wide dynamic range is assured by the differential internal analog architecture and by the separate analog and digital voltage supplies and grounds.

A/D bandpass filter, A/D bandpass filter clocking, and A/D conversion rate timing

The A/D bandpass filter can be selected or bypassed via software control. The frequency response of this filter is presented in the following pages. This response results when the switched-capacitor filter clock frequency is 288 kHz. Several possible options can be used to attain a 288-kHz switched-capacitor filter clock. When the filter clock frequency is not 288 kHz, the filter transfer function is frequency-scaled by the ratio of the actual clock frequency to 288 kHz. The low-frequency roll-off of the high-pass section is 300 kHz. However, the high-pass section low-frequency roll-off can be changed to 200 kHz with a metal mask option.

The Internal Timing Configuration and AIC DX Data Word Format sections of this data sheet indicate the many options for attaining a 288-kHz bandpass switched-capacitor filter clock. These sections indicate that the RX Counter A can be programmed to give a 288-kHz bandpass-switched capacitor filter clock for several Master Clock input frequencies.

The A/D conversion rate is then attained by frequency-dividing the 288-kHz bandpass switched-capacitor filter clock with the RX Counter B. Thus, unwanted aliasing is prevented because the A/D conversion rate is an integral submultiple of the bandpass switched-capacitor filter sampling rate, and the two rates are synchronously locked.

A/D converter performance specifications

Fundamental performance specifications for the A/D converter circuitry are presented in the A/D converter operating characteristics section of this data sheet. The realization of the A/D converter circuitry with switched-capacitor techniques provides an inherent sample-and-hold.

PRODUCT PREVIEW

analog output

The analog output circuitry is an analog output power amplifier. Both noninverting and inverting amplifier outputs are brought out of the IC. This amplifier can drive transformer hybrids or low-impedance loads directly in either a differential or single-ended configuration.

D/A low-pass filter, D/A low-pass filter clocking, and D/A conversion rate timing

The frequency response of this filter is presented in the following pages. This response results when the low-pass switched-capacitor filter clock frequency is 288 kHz. Like the A/D filter, the transfer function of this filter is frequency-scaled when the clock frequency is not 288 kHz. A continuous-time filter is provided on the output of the D/A low-pass filter to greatly attenuate any switched-capacitor clock feedthrough.

The D/A conversion rate is then attained by frequency-dividing the 288-kHz switched-capacitor filter clock with TX Counter B. Thus, unwanted aliasing is prevented because the D/A conversion rate is an integral submultiple of the switched-capacitor low-pass filter sampling rate, and the two rates are synchronously locked.

asynchronous versus synchronous operation

If the transmit section of the AIC (low-pass filter and DAC) and receive section (bandpass filter and ADC) are operated asynchronously, the low-pass and band-pass filter clocks are independently generated from the Master Clock signal. Also, the D/A and A/D conversion rates are independently determined. If the transmit and receive sections are operated synchronously, the low-pass filter clock drives both low-pass and band-pass filters. In synchronous operation, the A/D conversion timing is derived from, and is equal to, the D/A conversion rate timing. (See description of the WORD/$\overline{\text{BYTE}}$ pin in the Pin Functional Description Section.)

D/A converter performance specifications

Fundamental performance specifications for the D/A converter circuitry are presented in the D/A converter operating characteristics section of the data sheet. The D/A converter has a sample-and-hold that is realized with a switched-capacitor ladder.

system frequency response correction

Sin x/x correction circuitry is performed in digital signal processor software. The system frequency response can be corrected via DSP software to ±0.1 dB accuracy to a band-edge of 3000 Hz for all sampling rates. This correction is accomplished with a first-order digital correction filter, which requires only seven TMS320 instruction cycles. With a 200-ns instruction cycle, seven instructions represent an overhead factor of only 1.1% and 1.3% for sampling rates of 8 and 9.6 kHz, respectively (see the sin x/x Correction Section for more details).

serial port

The serial port has four possible modes that are described in detail in the pin description section. These modes are briefly described below.

1. The transmit and receive sections of the AIC are operated asynchronously, and the AIC serial port interfaces directly with the TMS32011.
2. The transmit and receive sections of the AIC are operated asynchronously, and the AIC serial port interfaces directly with the TMS32020 and the TMS32025.
3. The transmit and receive sections of the AIC are operated synchronously, and the AIC serial port interfaces directly with the TMS32011.
4. The transmit and receive sections of the AIC are operated synchronously, and the AIC serial port interfaces directly with the TMS32020, TMS32025, or two SN54299 or SN74299 serial-to-parallel shift registers, which can then interface in parallel to the TMS32010, to any other digital signal processor, or to external FIFO circuitry.

testing

An addendum accompanying this data sheet fully describes the test capabilities of the IC, provided by the design.

internal voltage reference

The internal reference eliminates the need for an external voltage reference, and thus provides overall circuit cost reduction. Additionally, the internal reference makes the performance of the IC less susceptible to noise. Thus, the internal reference eases the design task and provides complete control over the performance of the IC. The internal reference is brought out to a pin and is available to the designer. To keep the amount of noise on the reference signal to a minimum, an external capacitor may be connected between REF and ANLG GND.

PRINCIPLES OF OPERATION (continued)

reset

A reset function is provided to initiate serial communications between the AIC and DSP and to allow fast, cost-effective testing during manufacturing. The reset function will initialize all AIC registers, including the control register. The reset pin has an internal pull-up resistor. After a negative-going pulse on the $\overline{\text{RESET}}$ pin, the AIC will be initialized. This initialization allows normal serial port communications activity to occur between AIC and DSP (see AIC DX Data Word Format section).

loopback

This feature allows the user to test the circuit remotely. In loopback, the OUT + and OUT − pins are internally connected to the IN + and IN − pins. Thus, the DAC bits (d15 to d2), which are transmitted to the DX pin, can be compared with the ADC bits (d15 to d2), which are received from the DR pin. An ideal comparison would be that the bits on the DR pin equal the bits on the DX pin. However, in practice there will be some difference in these bits due to the ADC and DAC output offsets.

The loopback feature is implemented with digital signal processor control by transmitting the appropriate serial port bit to the control register (see AIC Data Word Format section).

PIN NAME	 NO.	I/O	DESCRIPTION
ANLG GND	17,18		Analog ground return for all internal analog circuits. Not internally connected to DGTL GND.
AUX IN +	24	I	Noninverting auxiliary analog input stage. This input can be switched into the bandpass filter and A/D converter path via software control. If the appropriate bit in the Control register is a 1, the auxiliary inputs will replace the IN + and IN − inputs. If the bit is a 0, the IN + and IN − inputs will be used (see the AIC DX Data Word Format section).
AUX IN −	23	I	Inverting auxiliary analog input (see the above AUX IN + pin description).
DGTL GND	9		Digital ground for all internal logic circuits. Not internally connected to ANLG GND.
DR	5	O	This pin is used to transmit the ADC output bits from the AIC to the TMS320 serial port. This transmission of bits from the AIC to the TMS320 serial port is synchronized with the SHIFT CLK signal.
DX	12	I	This pin is used to receive the DAC input bits and timing and control information from the TMS320. This serial transmission from the TMS320 serial port to the AIC is synchronized with the SHIFT CLK signal.
$\overline{\text{EODR}}$	2	O	(See the WORD/$\overline{\text{BYTE}}$ pin description and the Serial Port Timing Diagram.) During the word-mode timing, this signal is a low-going pulse that occurs immediately after the 16 bits of A/D information have been transmitted from the AIC to the TMS320 serial port. This signal can be used to interrupt a microprocessor upon completion of serial communications. Also, this signal can be used to strobe and enable external serial-to-parallel shift registers, latches, or external FIFO RAM, and to facilitate parallel data bus communications between the AIC and the serial-to-parallel shift registers. During the byte-mode timing, this signal goes low after the first byte has been transmitted from the AIC to the TMS320 serial port and is kept low until the second byte has been transmitted. The TMS32011 can use this low-going signal to differentiate between the two bytes as to which is first and which is second.

PRODUCT PREVIEW

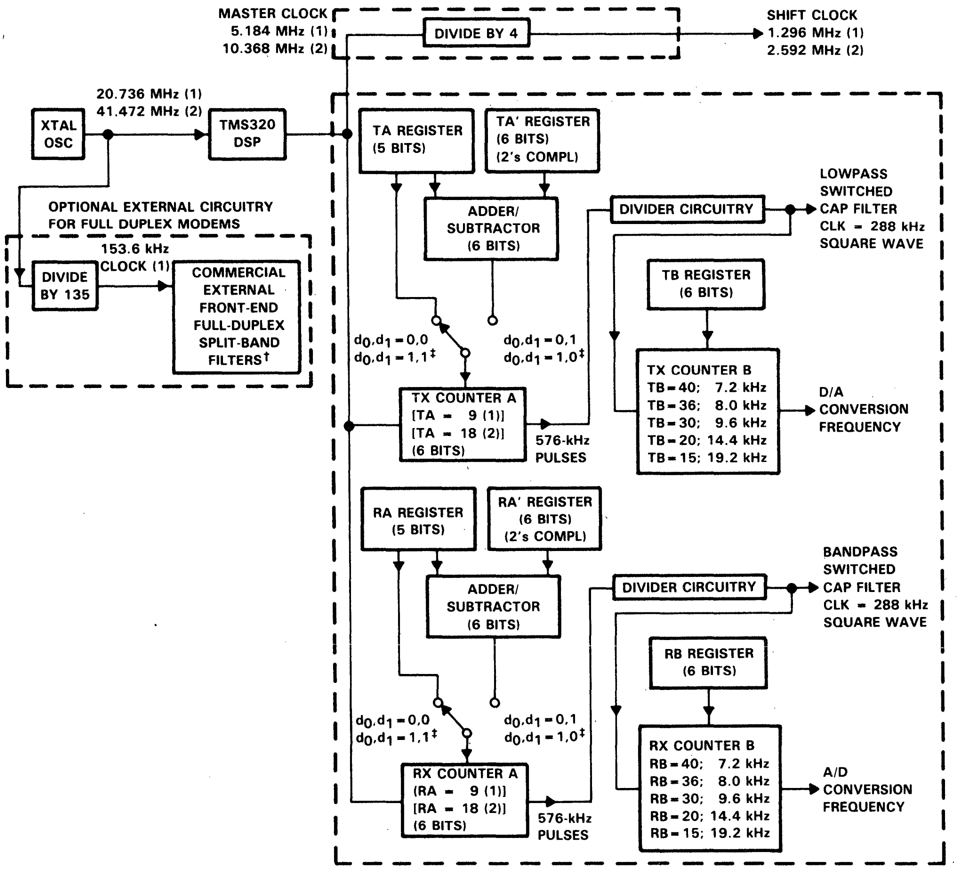
PRODUCT PREVIEW

PIN NAME	PIN NO.	I/O	DESCRIPTION
$\overline{\text{EODX}}$	11	O	(See the WORD/$\overline{\text{BYTE}}$ pin description and the Serial Port Timing Diagram.) During the word-mode timing, this signal is a low-going pulse that occurs immediately after the 16 bits of D/A converter and control or register information have been transmitted from the TMS320 serial port to the AIC. This signal can be used to interrupt a microprocessor upon the completion of serial communications. Also, this signal can be used to strobe and enable external serial-to-parallel shift registers, latches, or an external FIFO RAM, and to facilitate parallel, data-bus communications between the AIC and the serial-to-parallel shift registers. During the byte-mode timing, this signal goes low after the first byte has been transmitted from the TMS320 serial port to the AIC and is kept low until the second byte has been transmitted. The TMS32011 can use this low-going signal to differentiate between the two bytes as to which is first and which is second.
$\overline{\text{FSR}}$	4	O	In the serial transmission modes, which are described in the WORD/$\overline{\text{BYTE}}$ pin description, the $\overline{\text{FSR}}$ pin is held low during bit transmission. When the $\overline{\text{FSR}}$ pin goes low, the TMS320 serial port will begin receiving bits from the AIC via the DR pin of the AIC. The most significant DR bit will be present on the DR pin before $\overline{\text{FSR}}$ goes low. (See Serial Port Timing and Internal Timing Configuration Diagrams.)
$\overline{\text{FSX}}$	14	O	When this pin goes low, the TMS320 serial port will begin transmitting bits to the AIC via the DX pin AIC. In all serial transmission modes, which are described in the WORD/$\overline{\text{BYTE}}$ pin description, the $\overline{\text{FSX}}$ pin is held low during bit transmission (see Serial Port Timing and Internal Timing Configuration Diagrams).
IN+	26	I	Noninverting input to analog input amplifier stage
IN−	25	I	Inverting input to analog input amplifier stage
MSTR CLK	6	I	The Master Clock signal is used to derive all the key logic signals of the AIC, such as the Shift Clock, the switched-capacitor filter clocks, and the A/D and D/A timing signals. The Internal Timing Configuration diagram shows how these key signals are derived. The frequencies of these key signals are synchronous submultiples of the Master Clock frequency to eliminate unwanted aliasing when the sampled analog signals are transferred between the switched-capacitor filters and the A/D and D/A converters (see the Internal Timing Configuration).
OUT+	22	O	Noninverting output of analog output power amplifier. Can drive transformer hybrids or high-impedance loads directly in either a differential or a single-ended configuration.
OUT−	21	O	Inverting output of analog output power amplifier; functionally identical with and complementary to OUT+.
REF	8		The internal voltage reference is brought out to this pin.
$\overline{\text{RESET}}$	2	I	A reset function is provided to initialize the TA, TA′, TB, RA, RA′, RB, and control registers. This reset function initiates serial communications between the AIC and DSP. The reset function will initialize all AIC registers including the control register. After a negative-going pulse on the $\overline{\text{RESET}}$ pin, the AIC registers will be initialized to provide an 8-kHz data conversion rate for a 5.184-MHz master clock input signal. The conversion rate adjust registers, TA′ and RA′, will be reset to 1. The CONTROL register bits will be reset as follows (see AIC DX Data Word Format section). d7 = 1, d6 = 1, d5 = 1, d4 = 0, d3 = 0, d2 = 1 This initialization allows normal serial-port communication to occur between AIC and DSP. This pin has an internal pull-up resistor and is set to a high logic level unless it is pulled to ground.
SHIFT CLK	10	O	The Shift Clock signal is obtained by dividing the Master Clock signal frequency by four. This signal is used to clock the serial data transfers of the AIC, described in the WORD/$\overline{\text{BYTE}}$ pin description below (see the Serial Port Timing and Internal Timing Configuration diagram).
V_{DD}	7		Digital supply voltage, 5 V ±5%
V_{CC+}	20		Positive analog supply voltage, 5 V ±5%
V_{CC-}	19		Negative analog supply voltage −5 V ±5%

PIN			DESCRIPTION
NAME	NO.	I/O	
WORD/$\overline{\text{BYTE}}$	13	I	This pin, in conjunction with a bit in the CONTROL register, is used to establish one of four serial modes. These four serial modes are described below. This pin has an internal pull-up resistor and is set to a logic high unless it is pulled to ground., *AIC transmit and receive sections are operated asynchronously.* The following description applies when the AIC is configured to have asynchronous transmit and receive sections. If the appropriate data bit in the Control register is a 0 (see the AIC DX Data Word Format), the transmit and receive sections will be asynchronous. L Serial port will directly interface with the serial port of the TMS32011 and communicates in two 8-bit bytes. The operation sequence is as follows (see Serial Port Timing diagrams). 1. The $\overline{\text{FSX}}$ or $\overline{\text{FSR}}$ pin is brought low. 2. One 8-bit byte is transmitted or one 8-bit byte is received. 3. The $\overline{\text{EODX}}$ or $\overline{\text{EODR}}$ pin is brought low. 4. The $\overline{\text{FSX}}$ or $\overline{\text{FSR}}$ pin emits a positive frame-sync pulse that is four Shift Clock cycles wide. 5. One 8-bit byte is transmitted or one 8-bit byte is received. 6. The $\overline{\text{EODX}}$ or $\overline{\text{EODR}}$ pin is brought high. 7. The $\overline{\text{FSX}}$ or $\overline{\text{FSR}}$ pin is brought high. H Serial port will directly interface with the serial port of the TMS32020 and communicates in one 16-bit word. The operation sequence is as follows (see Serial Port Timing diagrams): 1. The $\overline{\text{FSX}}$ or $\overline{\text{FSR}}$ pin is brought low. 2. One 16-bit word is transmitted or one 16-bit word is received. 3. The $\overline{\text{FSX}}$ or $\overline{\text{FSR}}$ pin is brought high. 4. The $\overline{\text{EODX}}$ or $\overline{\text{EODR}}$ pin emits a low-going pulse. *AIC transmit and receive sections are operated synchronously.* If the appropriate data bit in the Control register is a 1, the transmit and receive sections will be configured to be synchronous. In this case, the bandpass switched-capacitor filter and the A/D conversion timing will be derived from the TX Counter A, TX Counter B, and TA, TA', and TB registers, rather than the RX Counter A, RX Counter B, and RA, RA', and RB registers. In this case, the AIC $\overline{\text{FSX}}$ and $\overline{\text{FSR}}$ timing will be identical, as will the $\overline{\text{EODX}}$ and $\overline{\text{EODR}}$ timing. The synchronous operation sequences are as follows (see Serial Port Timing diagrams). L Serial port will directly interface with the serial port of the TMS32011 and communicates in two 8-bit bytes. The operation sequence is as follows (see Serial Port Timing diagrams): 1. The $\overline{\text{FSX}}$ and $\overline{\text{FSR}}$ pins are brought low. 2. One 8-bit byte is transmitted and one 8-bit byte is received. 3. The $\overline{\text{EODX}}$ and $\overline{\text{EODR}}$ pins are brought low. 4. The $\overline{\text{FSX}}$ and $\overline{\text{FSR}}$ pins emit positive frame-sync pulses that are four Shift Clock cycles wide. 5. One 8-bit byte is transmitted and one 8-bit byte is received. 6. The $\overline{\text{EODX}}$ and $\overline{\text{EODR}}$ pins are brought high. 7. The $\overline{\text{FSX}}$ and $\overline{\text{FSR}}$ pins are brought high. H Serial port will directly interface with the serial port of the TMS32020 and communicates in one 16-bit word. The operation sequence is as follows (see Serial Port Timing diagrams): 1. The $\overline{\text{FSX}}$ and $\overline{\text{FSR}}$ pins are brought low. 2. One 16-bit word is transmitted and one 16-bit word is received. 3. The $\overline{\text{FSX}}$ and $\overline{\text{FSR}}$ pins are brought high. 4. The $\overline{\text{EODX}}$ or $\overline{\text{EODR}}$ pins emit low-going pulses. Since the transmit and receive sections of the AIC are now synchronous, the AIC serial port, with additional NOR and AND gates, will interface to two SN54299 or SN74299 serial-to-parallel shift registers. Interfacing the AIC to the SN54299 or SN74299 shift register allows the AIC to interface to an external FIFO RAM and facilitates parallel, data bus communications between the AIC and the digital signal processor. The operation sequence is the same as the above sequence (see Serial Port Timing diagrams).

PRODUCT PREVIEW

INTERNAL TIMING CONFIGURATION



NOTE: Frequency 1, 20.736 MHz, is used to show how 153.6 kHz (for a commercially available modem split-band filter clock), popular speech and modem sampling signal frequencies, and an internal 288-kHz switched-capacitor filter clock can be derived synchronously and as submultiples of the crystal oscillator frequency. Since these derived frequencies are synchronous submultiples of the crystal frequency, aliasing does not occur as the sampled analog signal passes between the analog converter and switched-capacitor filter stages. Frequency 2, 41.472 MHz, is used to show that the AIC can work with high-frequency signals, which are used by high-speed digital signal processors.

†Split-band filtering can alternatively be performed after the analog input function via software in the TMS320.

‡These control bits are described in the AIC DX Data Word Format section.

explanation of internal timing configuration

All of the internal timing of the AIC is derived from the high-frequency clock signal that drives the Master Clock input pin. The Shift Clock signal, which strobes the serial port data between the AIC and DSP, is derived by dividing the Master Clock input signal frequency by four.

TX Counter A and TX Counter B, which are driven by the Master Clock signal, determine the D/A conversion period timing. Similarly, RX Counter A and RX Counter B determine the A/D conversion period timing. In order for the switched-capacitor low-pass and band-pass filters to meet their transfer function specifications, the frequency of the clock inputs of the switched-capacitor filter must be 288 kHz. If the frequencies of the clock inputs are not 288 kHz, the filter transfer function frequencies are scaled by the ratios of the clock frequencies to 288 kHz. Thus, to obtain the specified filter responses, the combination of Master Clock frequency and TX Counter A and RX Counter A values must yield 288-kHz switched-capacitor clock signals. These 288-kHz clock signals can then be divided by the TX Counter B and RX Counter B to establish the D/A and A/D conversion period timings.

TX Counter A and TX Counter B are reloaded every D/A conversion period, while RX Counter A and RX Counter B are reloaded every A/D conversion period. The TX Counter B and RX Counter B are loaded with the values in the TB and RB Registers respectively. Via software control, the TX Counter A can be loaded with either the TA Register, the TA Register less the TA′ Register, or the TA Register plus the TA′ Register. By selecting the TA Register less the TA′ Register option, the upcoming conversion period timing will occur earlier by an amount of time that equals TA′ times the signal period of the Master Clock. By selecting the TA Register plus the TA′ Register option, the upcoming conversion period timing will occur later by an amount of time that equals TA′ times the signal period of the Master Clock. Thus, the D/A conversion timing can be advanced or retarded. An identical ability to alter the A/D conversion timing is provided. In this case, however, the RX Counter A can be programmed via software control with the RA Register, the RA Register less the RA′ Register, or the RA Register plus the RA′ Register.

The above feature is particularly useful for modem applications. This feature allows controlled changes in the A/D and D/A conversion timing. This feature can be used to enhance signal-to-noise performance, to perform frequency-tracking functions, and to generate nonstandard modem frequencies.

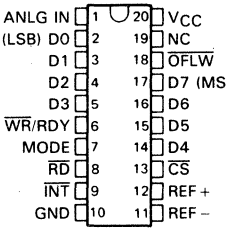
If the transmit and receive sections are configured to be synchronous (see WORD/$\overline{\text{BYTE}}$ pin description), then both the low-pass and bandpass switched-capacitor filter clocks are derived from TX Counter A. Also, both the D/A and A/D conversion timing are derived from the TX Counter A and TX Counter B. When the transmit and receive sections are configured to be synchronous, the RX Counter A, RX Counter B, RA Register, RA′ Register, and RB Registers are not used.

TLC0820A, TLC0820B
ADVANCED LinCMOS™ HIGH-SPEED 8-BIT ANALOG-TO-DIGITAL CONVERTERS USING MODIFIED "FLASH" TECHNIQUES

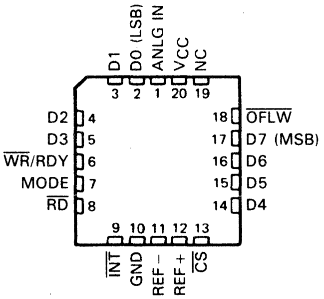
D2873, SEPTEMBER 1986—REVISED DECEMBER 1987

- **Advanced LinCMOS™ Silicon-Gate Technology**
- **8-Bit Resolution**
- **Differential Reference Inputs**
- **Parallel Microprocessor Interface**
- **Conversion Access Time Over Temperature Range**
  Write-Read Mode . . . 1.18 μs and 1.92 μs
  Read Mode . . . 2.6 μs Max
- **No External Clock or Oscillator Components Required**
- **On-Chip Track-and-Hold**
- **Low Power Consumption . . . 50 mW Typ**
- **Single 5-V Supply**
- **TLC0820B is Direct Replacement for National Semiconductor ADC0820B/BC and Analog Devices AD7820L/C/U;**
  TLC0820A is Direct Replacement for National Semiconductor ADC0820C/CC and Analog Devices AD7820K/B/T

TLC0820AM, TLC0820BM . . . DW, J OR N PACKAGE
TLC0820AI, TLC0820BI . . . DW OR N PACKAGE
TLC0820AC, TLC0820BC . . . DW OR N PACKAGE
(TOP VIEW)



TLC0820AM, TLC0820BM . . . FK PACKAGE
TLC0820AI, TLC0820BI . . . FN PACKAGE
TLC0820AC, TLC0820BC . . . FN PACKAGE
(TOP VIEW)



NC—No internal connection

description

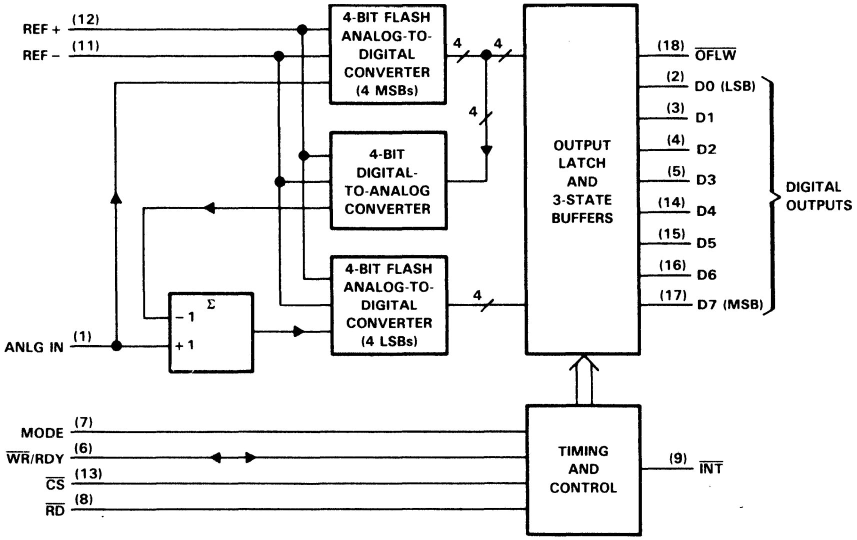
The TLC0820A and TLC0820B are Advanced LinCMOS™ 8-bit analog-to-digital converters each consisting of two 4-bit "flash" converters, a 4-bit digital-to-analog converter, a summing (error) amplifier, control logic, and a result latch circuit. The modified "flash" technique allows low-power integrated circuitry to complete an 8-bit conversion in 1.18 microseconds over temperature. The on-chip track-and-hold circuit has a 100-nanosecond sample window and allows the TLC0820A and TLC0820B to convert continuous analog signals having slew rates of up to 100 millivolts per microsecond without external sampling components. TTL-compatible three-state output drivers and two modes of operation allow interfacing to a variety of microprocessors. Detailed information on interfacing to most popular microprocessors is readily available from the factory.

The TLC0820AM and TLC0820BM are available in the DW or N plastic and the J ceramic packages and are characterized for operation over the full military temperature range of −55°C to 125°C. The TLC0820AI and TLC0820BI are characterized for operation from −40°C to 85°C. The TLC0820AC and TLC0820BC are characterized for operation from 0°C to 70°C.

ADVANCE INFORMATION

Advanced LinCMOS is a trademark of Texas Instruments.

functional block diagram



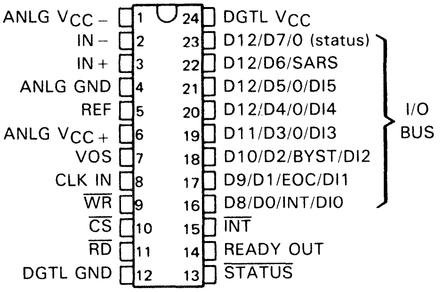
PRODUCT PREVIEW

TLC1205A, TLC1205B, TLC1225A, TLC1225B SELF-CALIBRATING 12-BIT-PLUS-SIGN UNIPOLAR OR BIPOLAR ANALOG-TO-DIGITAL CONVERTERS

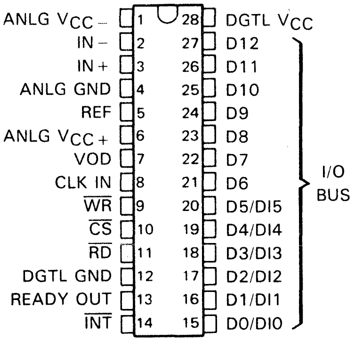
D2982, FEBRUARY 1987

- ADVANCED LinCMOS™ Technology
- Self-Calibration Eliminates Expensive Trimming at Factory and Offset Adjustment in the Field
- 12-Bit Plus Sign Unipolar or Bit Bipolar
- ± ½ and ± 1 LSB Linearity Error in Unipolar Configuration
- 10 µs Conversion Time (Mode 2) (clock = 2.6 MHz)
  20 µs Conversion Time (Mode 1) (clock = 2.6 MHz)
- Compatible with All Microprocessors
- True Differential Analog Voltage Inputs
- 0 to 5 V Analog Voltage Range with Single 5-V Supply (Unipolar Configuration)
- − 5 V to 5 V Analog Voltage Range with ± 5-V Supplies (Bipolar Configuration)
- Low Power . . . 25 mW Maximum
- Replaces National Semiconductor ADC1205 and ADC1225 in Mode 1 Operation

TLC1205
J OR N DUAL-IN-LINE PACKAGE
(TOP VIEW)



TLC1225
J OR N DUAL-IN-LINE PACKAGE
(TOP VIEW)

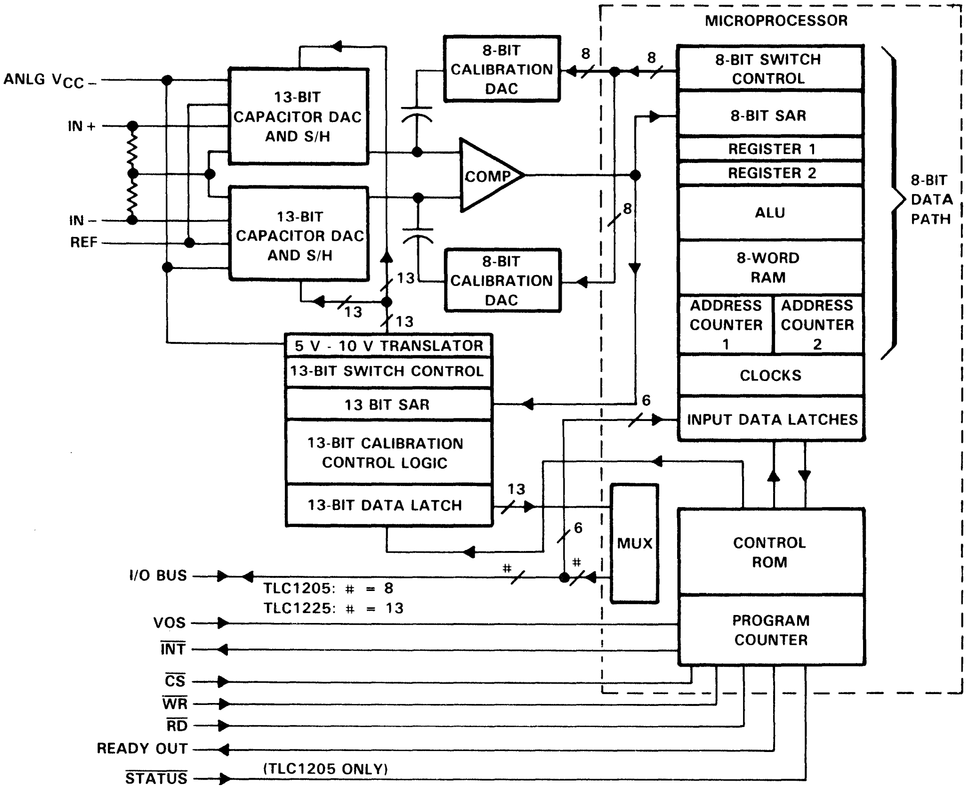


description

The TLC1205 and TLC1225 converters are manufactured with Texas Instruments highly efficient ADVANCED LinCMOS™ technology. Either of the TLC1205 or TLC1225 CMOS analog-to digital converters can be operated as a unipolar or bipolar converter. A unipolar input (0 to 5 V) can be accommodated with a single 5-volt supply, while a bipolar input (− 5 V to 5 V) requires the addition of a 5-volt negative supply. Conversion is performed via the successive-approximation method. The 24-pin TLC1205 outputs the converted data in two 8-bit bytes, while the TLC1225 outputs the converted data in a parallel word and interfaces directly to a 16-bit data bus. Negative numbers are given in the 2's complement data format. All digital signals are fully TTL and CMOS compatible.

These converters utilize a self-calibration technique by which seven of the internal capacitors in the capacitive ladder of the A/D conversion circuitry can be automatically or manually calibrated. If the converters are operated in Mode 1, one of the seven internal capacitors is calibrated during the first part of the conversion sequence. For example, one capacitor is calibrated during the first conversion. The next capacitor is calibrated during the second conversion. If the converters are operated in Mode 2, the internal capacitors are calibrated during a nonconversion, capacitor-calibrate cycle in which all seven of the internal capacitors are calibrated at the same time. A Mode 2 conversion requires only 10 µs (2.6 MHz clock) after the nonconversion, capacitor-calibrating cycle has been completed. The calibration or conversion cycle may be initiated at any time by issuing the proper address to the data bus. The self-calibrating techniques eliminate the need for expensive trimming of thin-film resistors at the factory and provide excellent performance at low cost.

ADVANCED LinCMOS™ is a trademark of Texas Instruments Incorporated functional block diagram



In Mode 1, these converters are replacements for National Semiconductor ADC1205 and ADC1225 integrated circuits. The Mode 1 conversion time for guaranteed accuracy is 51 clock cycles. In the Mode 2 operation, these devices are no longer true replacements. However, the Mode 2 conversion time for guaranteed accuracy is only 26 clock cycles.

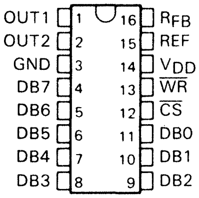
The TLC1205AM, TLC1205BM, TLC1225AM, and TLC1225BM are characterized for operation over the full military temperature range of −55°C to 125°C. The TLC1205AI, TLC1205BI, TLC1225AI, and TLC1225BI are characterized for operation from −40°C to 85°C.

TLC7524
Advanced LinCMOS™ 8-BIT MULTIPLYING DIGITAL-TO-ANALOG CONVERTER

D3008, SEPTEMBER 1986

- **Advanced LinCMOS™ Silicon-Gate Technology**
- **Easily Interfaced to Microprocessors**
- **On-Chip Data Latches**
- **Guaranteed Monotonicity**
- **Segmented High-Order Bits Ensure Low-Glitch Output**
- **Designed to be Interchangeable with Analog Devices AD7524, PMI PM-7524, and Micro Power Systems MP7524**
- **Fast Control Signaling for Digital Signal Processor Applications Including Interface with TMS320**

D OR N PACKAGE
(TOP VIEW)



KEY PERFORMANCE SPECIFICATIONS	
Resolution	8 Bits
Linearity error	½ LSB Max
Power dissipation at $V_{DD} = 5$ V	5 mW Max
Settling time	100 ns Max
Propagation delay	80 ns Max

description

The TLC7524 is an Advanced LinCMOS™ 8-bit digital-to-analog converter (DAC) designed for easy interface to most popular microprocessors.

The TLC7524 is an 8-bit multiplying DAC with input latches and with a load cycle similar to the "write" cycle of a random access memory. Segmenting the high-order bits minimizes glitches during changes in the most-significant bits, which produce the highest glitch impulse. The TLC7524 provides accuracy to ½ LSB without the need for thin-film resistors or laser trimming, while dissipating less than 5 milliwatts typically.

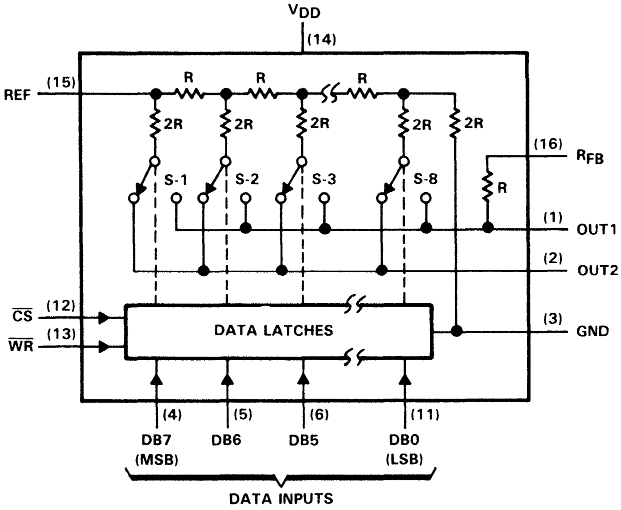
Featuring operation from a 5-V to 15-V single supply, the TLC7524 interfaces easily to most microprocessor buses or output ports. Excellent multiplying (2 or 4 quadrant) makes the TLC7524 an ideal choice for many microprocessor-controlled gain-setting and signal-control applications.

The TLC7524I is characterized for operation from −25°C to 85°C, and the TLC7524C is characterized for operation from 0°C to 70°C.

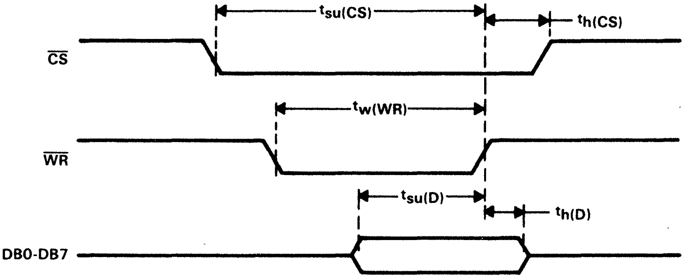
Advanced LinCMOS is a trademark of Texas Instruments Incorporated

TEXAS INSTRUMENTS
POST OFFICE BOX 655012 • DALLAS, TEXAS 75265

functional block diagram



operating sequence



IC SOCKETS
PLASTIC LEADED CHIP CARRIER

PERFORMANCE SPECIFICATIONS

Mechanical

Recommended PCB thickness range: 0.062 in to 0.092 in
Recommended PCB hole size range: 0.032 in to 0.042 in
Vibration: 15 G
Shock: 100 G
Solderability: Per MIL-STD 202, Method 208
Insertion force: 0.59 lbs per position
Withdrawal force: 0.25 lbs per position
Normal force: 200 g min, 450 g typ
Wipe: 0.075 in min
Durability: 5 cycles min
Contact retention: 1.5 lbs min

Electrical

Current carrying capacity: 1 A
Insulation resistance: 5000 MΩ min
Dielectric withstanding voltage: 1000 V ac rms min
Capacitance: 1.0 pF max

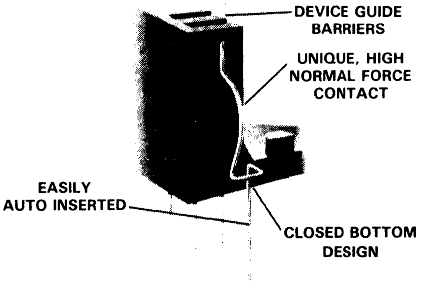
Environmental

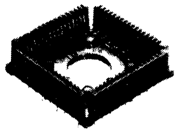
Operating temperature:
Operating: − 40°C to 85°C
Storage: −40°C to 95°C
Temperature cycling with humidity: will conform to final EIA specifications
Shelf life: 1 year min

MATERIALS

Body — Ryton R-4 (40% glass) U/L 94-VO rating
Contacts — CDA 510 spring temper
Contact finish — 90/10 tin (200 μin −400 μin) over 40 μin copper

Contact factory for detailed information



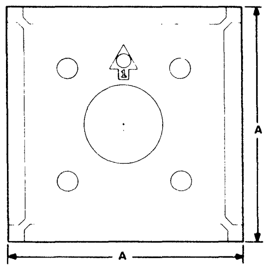


PART NUMBER SYSTEM

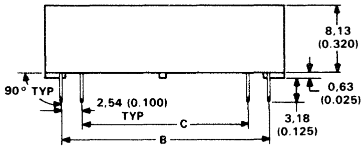
CPR PH XXX − X − X − 0

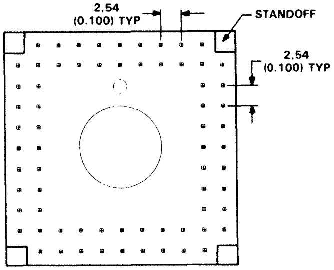
- Contact surface 1 — tin lead plating
- Contact spacing 1 — 0.050 in
- Number of pos (044, 052, 068, 084)
- Plated thru hole, solder tail
- TI socket Series
  Plastic leaded chip carrier

PLASTIC LEADER CHIP CARRIER CPR SERIES



Device guide barriers not shown

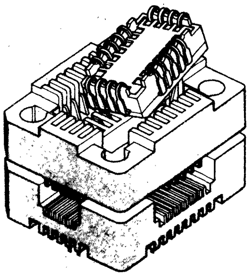




Pos	A	B	C
44	21,43 (0.844	17,78 (0.700)	12,70 (0.500)
52	23,98 (0.944)	20,32 (0.800)	15,24 (0.600)
68	29,06 (1.144)	25,40 (1.000)	20,32 (0.800)
84	34,14 (1.344)	30,48 (1.200)	25,40 (1.000)

Extraction tool available, consult factory.

IC SOCKETS
PLCC BURN-IN/TEST



PRODUCT FEATURES

Can be loaded by top actuated insertion or press-in insertion, either manually or automatically
High reliability due to high pressure contact point
Open body and high stand-off design provide high efficiency in heat dissipation
High durability up to 10,000 cycles
Compact design

PERFORMANCE SPECIFICATIONS

Mechanical

Durability: 10,000 cycles
Operating Temperature: 180°C max

Electrical

Contact rating: 1.0 A per contact
Contact resistance: 30 mΩ max
Insulation resistance: 1000 MΩ min
Dielectric withstanding voltage: 500 V ac rms min

MATERIALS

Body — ultem glass filled (U/L 94 VO)
Contact — copper alloy
Plating — overall gold plate

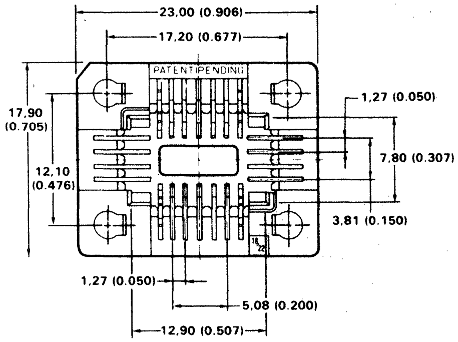
PART NUMBER SYSTEM

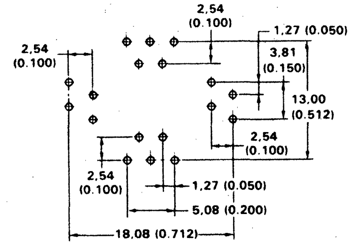
CPJ AA33A – XXX B

Number of positions

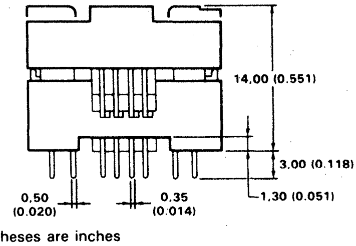
TI series socket

PLCC BURN-IN/TEST SOCKETS CPJ SERIES





SIZES: 18 PIN
22 PIN



Dimensions in parentheses are inches
Contact factory for detailed information

IC SOCKETS
HIGH DENSITY PIN GRID ARRAY

PERFORMANCE SPECIFICATIONS

Mechanical

Accommodates IC leads 0.015 in to 0.021 in diameter
Recommended PCB thickness range: 0.062 in to 0.092 in
Recommended PCB hole size range: 0.032 in to 0.042 in
Recommended hole grid pattern: 0.100 in ± 0.002 in each direction
Vibration: 15 G, 10-2000 Hz per MIL-STD 1344A, Method 2005.1 Test Condition III
Shock: 100 G, sawtooth waveform, 2 shocks each direction per MIL-STD 202, Method 213, Test Condition I
Durability: 5 cycles, 10 mΩ max contact resistance change per MIL-STD 1344, Method 2016
Solderability: per MIL-STD 202, Method 208
Insertion force: 3.6 oz (102 g) per pin typ using 0.018 in diameter test pin
Withdrawal force: 0.5 oz (14 g) per pin min using 0.018 in diameter test pin

Electrical

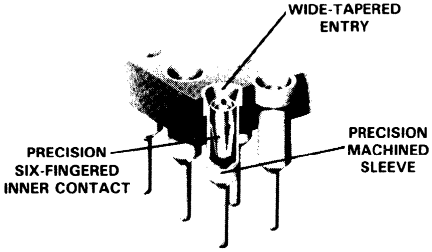
Contact rating: 1.0 A per contact
Contact resistance: 20 mΩ max initial
Insulation resistance: 1000 MΩ at 500 V dc per MIL-STD 1344, Method 3003.1
Dielectric withstanding voltage: 1000 V ac rms per MIL-STD 1344, Method 3001.1
Capacitance: 1.0 pF max per MIL-STD 202, Method 305

Environmental

Operating temperature: −65 °C to 125 °C, gold; −40 °C to 100 °C, tin
Corrosive atmosphere: 10 mΩ max contact resistance change when exposed to 22% ammonium sulfide for 4 hours
Gas tight: 10 mΩ max contact resistance change when exposed to nitric acid vapor for 1 hour
Temperature soak: 10 mΩ max contact resistance change when exposed to 105 °C temperature for 48 hours
Shell life: 12 months min

MATERIALS

Body — PBT polyester U/L94-VO rating
On request, G10/FR4 or Mylar film
Outer sleeve — Machined Brass (QQ-B-626)
Inner contact — Beryllium copper (QQ-C-530) heat treated
Plating: (specified by part number)



Inner contact — 30 µin gold over 50 µin nickel or 100 µin tin/lead over 50 µin nickel
Outer sleeve — 10 µin gold over 50 µin nickel or 50 µin tin/lead over 50 µin nickel

PART NUMBER SYSTEM

C X G XX – XXX X X – X X

Pin Length

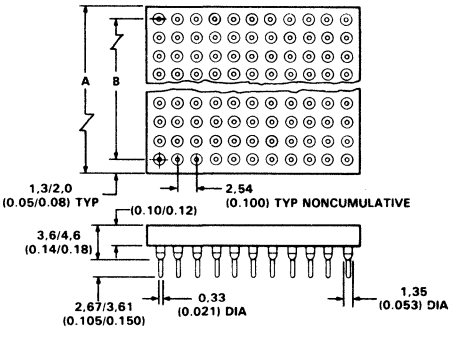
W/W	S/T
3-0.510 long	9-0.105/0.150

Plating

P/N	Sleeve	Clip
0	Gold	Gold
5	Tin	Gold

Body Style and Orientation
Contact Loading Pattern
Number of Pins 024 to 324
Overall Grid Size 5 x 5 = 05 to 18 x 18 = 18
Pin Grid Array
BODY MATERIAL
G – Glass Filled Epoxy
P – PBT Polyester
TI Socket

PIN GRID ARRAY



Insulator Size	A ±0.010	B ±0.005†
9 x 9	(0.950) 24,13	(0.800) 20,32
10 x 10	(1.050) 26,67	(0.900) 22,86
11 x 11	(1.150) 29,21	(1.000) 25,40
12 x 12	(1.250) 31,75	(1.100) 27,94
13 x 13	(1.350) 34,29	(1.200) 30,48
14 x 14	(1.450) 36,83	(1.300) 33,02
15 x 15	(1.550) 39,37	(1.400) 35,56
16 x 16	(1.650) 41,91	(1.500) 38,10
17 x 17	(1.750) 44,45	(1.600) 40,64
18 x 18	(1.850) 46,99	(1.700) 43,18

[†] Noncumulative
Dimensions in parentheses are inches
Consult factory for detailed information

IC SOCKETS
BURN-IN/TEST PIN GRID ARRAY

PERFORMANCE SPECIFICATIONS

Mechanical

Accommodates IC leads per specific IC device
Recommended PCB thickness range: 0.062 in to 0.092 in
Recommended PCB hole size range: 0.032 in to 0.042 in
Durability: 5000 cycles, 10 mΩ max contact resistance change per MIL-STD 1344, Method 2016
Solderability: per MIL-STD 202, Method 208

Electrical

Contact rating: 1.0 A per contact
Contact resistance: 20 mΩ max initial
Insulation resistance: 1.0 MΩ at 500 V dc per MIL-STD 1344, Method 3003.1
Dielectric withstanding voltage: 700 V ac rms per MIL-STD 1344, Method 3001.1
Capacitance: 1.0 pF max per MIL-STD 202, Method 305

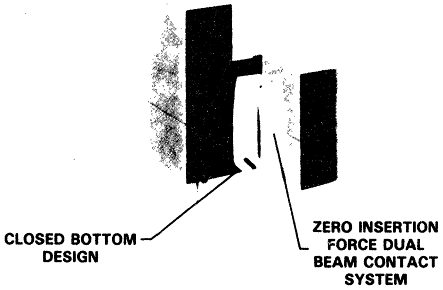
Environmental

Operating temperature: −65°C to 170°C
Humidity: 10 mΩ max contact resistance change when tested per MIL-STD 202, Method 103B
Temperature soak; 10 mΩ max contact resistance change when exposed to 105°C temperature for 48 hours
Shelf life: 12 months max

MATERIALS

Body — CZF Series: PPS (polyphenylen sufide) glass filled U/L 94 VO rating, −65°C to 170°C
Contact — Beryllium copper
Plating:† Overall gold plate min 4 μin over min 70 μin nickel plating

†For additional plating option consult the factory.

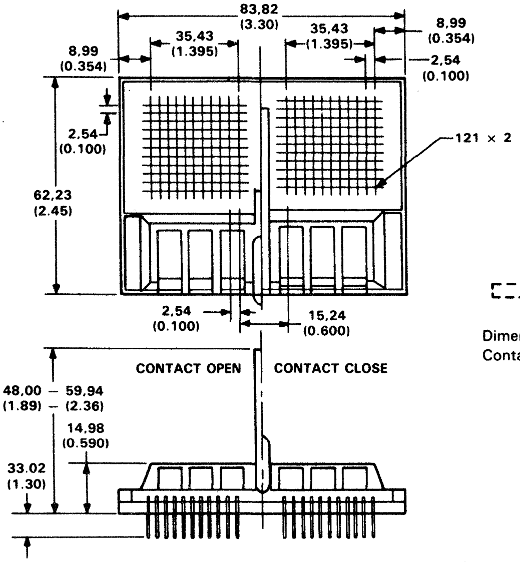


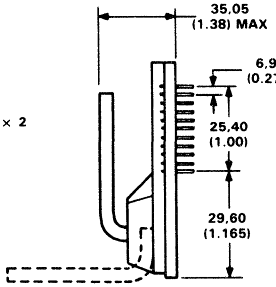
PART NUMBER SYSTEM

C XX X – XXX – XX – 01

- C — TI Series socket
- XX — Style ZF – Zero force
- X — Configuration W – 11 x 11 x 2
- XXX — Number of positions
- XX — Plating 37 – overall gold plate

BURN-IN TEST PIN GRID ARRAY





Dimensions in parentheses are inches
Contact factory for detailed information

Appendix F - Crystals

Vendors of crystals suitable for use with TMS320 devices are listed below.

RXD, Inc.
Norfolk, NB
(800) 228-8108

CTS Knight, Inc.
Contact the local distributor.

N.E.L. Frequency Controls, Inc.
Burlington, WI
(414) 763-3591

G. ROM Codes

Board space is often a critical concern in many DSP applications. In order to reduce chip count and provide the customer with a single-chip solution, Texas Instruments offers microcomputer versions for TMS320C2x (second-generation TMS320) devices. The on-chip ROM of these processors can be masked with the customer's own code. This allows the user to take advantage of the general-purpose features of TI's digital signal processors while at the same time customizing the processor to suit a specific application.

To facilitate design, all prototype work is performed using a standard TMS320C2x microprocessor. TMS320C2x development tools permit a designer to test and refine algorithms for immediate results. When the algorithm has been finalized, the customer can submit the code to Texas Instruments to be masked into the on-chip ROM of the device.

The MP/MC (microprocessor/microcomputer) mode, offered on maskable TMS320C2x devices such as the TMS320C25, can shorten design and field upgrade cycle times, thereby reducing expense. This mode permits the customer to use the TMS320C2x as a standard device operating out of external program memory. When TMS320C2x code is altered during design, the delays associated with new silicon processing are avoided. Field upgrade cycle times and the associated expense of inventory obsolescence when the code is altered are also avoided. Note that the TMS32020 has no on-chip ROM and operates in the microprocessor mode only.

An entire algorithm or an often-used routine may be masked into the on-chip ROM space of a TMS320C2x device. TMS320C2x programs can also be expanded using external memory. With a reduced chip count and this program memory flexibilty, multiple functions can be more easily implemented in a single hardware device, thus enhancing a product's capabilities. The TMS320C25 with 4K words of on-chip ROM can be ordered as a masked device. The customer's code must fit within the specified ROM size of the processor.

Figure G-1 illustrates the procedure flow for implementing TMS320C2x masked parts. With any masked device order, there is a one-time charge of $6000 for mask tooling which includes 10 prototypes. A non-cancellable minimum production order per year of 5000 units is required for the TMS320C25.

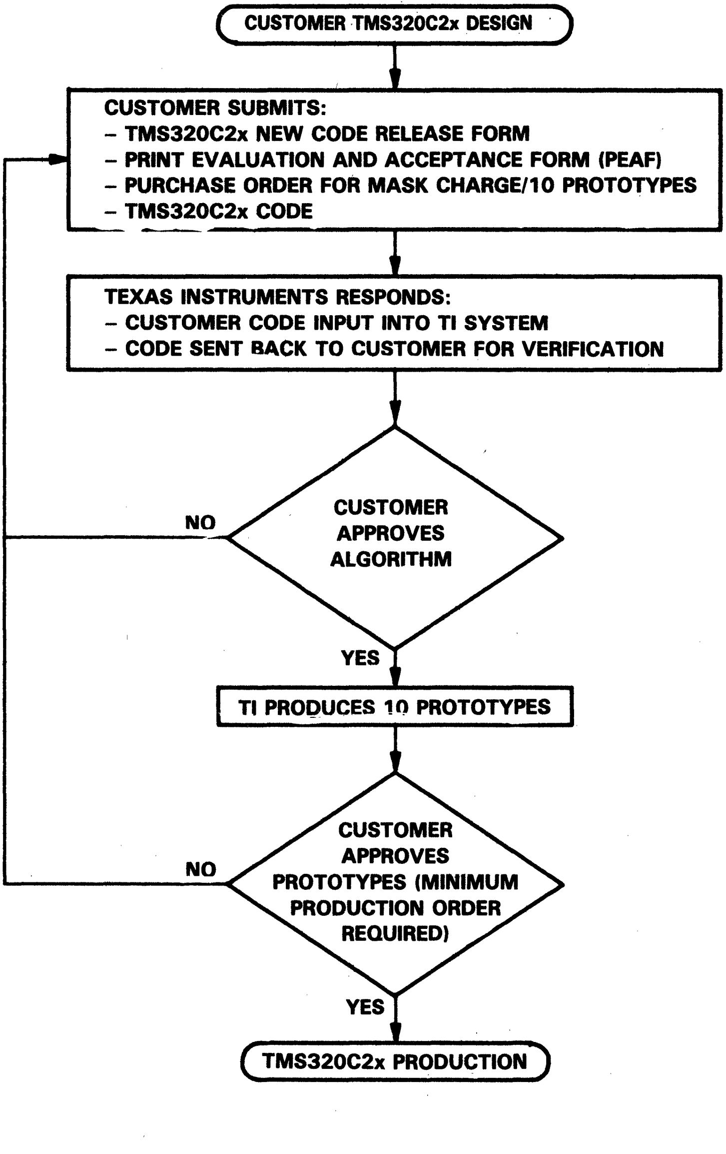


Figure G-1. TMS320C2x ROM Code Flowchart

Leadtimes for the first 10 prototype units begin when the customer has formally verified that TI has recorded his code correctly. Leadtimes for the first production order begin once the customer formally approves the masked prototypes. The typical leadtime for masked TMS320C2x prototypes is 8 to 10 weeks and for masked TMS320C2x production 12 to 16 weeks. Texas Instruments constantly strives to improve these leadtimes and reserves the right to make changes at any time. Please contact the nearest TI Sales Office for current leadtimes, further information on these procedures, and confirmation of the mask/production requirements.

A TMS320C2x ROM code may be submitted in one of the following formats (the preferred media is 5 1/4" floppies):

FLOPPY:	TI Cross-Assembler Format
EPROM:	TMS27C64, TMS2508, TMS2516, TMS2532, TMS2564
PROM:	TBP28S166, TBP28S86

When a code is submitted to Texas Instruments for a masked device, the code is reformatted to accommodate the TI mask generation system. System level verification by the customer is therefore necessary. Although the code has been reformatted, it is important that the changes remain transparent to the user and not affect the execution of the algorithm. The formatting changes made involve deletion of all address tags (unnecessary in a ROM code device) and addition of data in the reserved locations of the ROM for device ROM test. Note that because these changes have been made, a checksum comparison is not a valid means of verification.

ROM code algorithms may also be submitted by secure electronic transfer via a modem. Contact the nearest TI sales office for further information.

With each masked device order, the customer must sign a disclaimer stating:

> "The units to be shipped against this order were assembled, for expediency purposes, on a prototype (i.e., non-production qualified) manufacturing line, the reliability of which is not fully characterized. Therefore, the anticipated inherent reliability of these prototype units cannot be expressly defined."

and a release stating:

> "Any masked ROM device may be resymbolized as TI standard product and resold as though it were an unprogrammed version of the device at the convenience of Texas Instruments."

ROM codes will be deleted from the TI system after one year from the last delivery.

H. Quality and Reliability

The quality and reliability performance of Texas Instruments Microprocessor and Microcontroller Products, which includes the three generations of TMS320 digital signal processors, relies on feedback from:

- Our customers
- Our total manufacturing operation from front-end wafer fabrication to final shipping inspection
- Product quality and reliability monitoring.

Our customer's perception of quality must be the governing criterion for judging performance. This concept is the basis for Texas Instruments Corporate Quality Policy, which is as follows:

> "For every product or service we offer, we shall define the requirements that solve the customer's problems, and we shall conform to those requirements without exception."

Texas Instruments offers a leadership reliability qualification system, based on years of experience with leading-edge memory technology as well as years of research into customer requirements. Quality and reliability programs at TI are therefore based on customer input and internal information to achieve constant improvement in quality and reliability.

H.1 Reliability Stress Tests

Accelerated stress tests are performed on new semiconductor products and process changes to ensure product reliability excellence. The typical test environments used to qualify new products or major changes in processing are:

- High-temperature operating life
- Storage life
- Temperature cycling
- Biased humidity
- Autoclave
- Electrostatic discharge
- Package integrity
- Electromigration
- Channel-hot electrons (performed on geometries less than 2.0 μm).

Typical events or changes that require internal requalification of product include:

- New die design, shrink, or layout
- Wafer process (baseline/control systems, flow, mask, chemicals, gases, dopants, passivation, or metal systems)
- Packaging assembly (baseline control systems or critical assembly equipment)
- Piece parts (such as lead frame, mold compound, mount material, bond wire, or lead finish)
- Manufacturing site.

TI reliability control systems extend beyond qualification. Total reliability controls and management include product ramp monitor as well as final product release controls. MOS memories, utilizing high-density active elements, serve as the leading indicator in wafer-process integrity at TI MOS fabrication sites, enhancing all MOS logic device yields and reliability. TI places more than 200,000 MOS devices per month on reliability test to ensure and sustain built-in product excellence.

Table H-1 lists the microprocessor and microcontroller reliability tests, the duration of the test, and sample size. The following defines and describes those tests in the table.

AOQ (Average Outgoing Quality)	Amount of defective product in a population, usually expressed in terms of parts per million (PPM).
FIT (Failure In Time)	Estimated field failure rate in number of failures per billion power-on device hours; 1000 FITS equals 0.1 percent fail per 1000 device hours.
Operating lifetest	Device dynamically exercised at a high ambient temperature (usually 125°C) to simulate field usuage that would

expose the device to a much lower ambient temperature (such as 55°C). Using a derived high temperature, a 55°C ambient failure rate can be calculated.

High-temperature storage	Device exposed to 150°C unbiased condition. Bond integrity is stressed in this environment.
Biased humidity	Moisture and bias used to accelerate corrosion-type failures in plastic packages. Conditions include 85°C ambient temperature with 85-percent relative humidity (RH). Typical bias voltage is +5 V and ground on alternating pins.
Autoclave (pressure cooker)	Plastic-packaged devices exposed to moisture at 121°C using a pressure of one atmosphere above normal pressure. The pressure forces moisture permeation of the package and accelerates corrosion mechanisms (if present) on the device. External package contaminates can also be activated and caused to generate inter-pin current leakage paths.
Temperature cycle	Device exposed to severe temperature extremes in an alternating fashion (-65°C for 15 minutes and 150°C for 15 minutes per cycle) for at least 1000 cycles. Package strength, bond quality, and consistency of assembly process are stressed in this environment.
Thermal shock	Test similar to the temperature cycle test, but involving a liquid-to-liquid transfer, per MIL-STD-883C, Method 1011.
PIND	Particle Impact Noise Detection test. A non-destructive test to detect loose particles inside a device cavity.
Mechanical Sequence:	
Fine and gross leak	Per MIL-STD-883C, Method 1014.5
Mechanical shock	Per MIL-STD-883C, Method 2002.3, 1500 g, 0.5 ms, Condition B
PIND (optional)	Per MIL-STD-883C, Method 2020.4
Vibration, variable frequency	Per MIL-STD-883C, Method 2007.1, 20 g, Condition A
Constant acceleration	Per MIL-STD-883C, Method 2001.2, 20 kg, Condition D, Y1 Plane min
Fine and gross leak	Per MIL-STD-883C, Method 1014.5

Electrical test	To data sheet limits

Thermal Sequence:

Fine and gross leak	Per MIL-STD-883C, Method 1014.5
Solder heat (optional)	Per MIL-STD-750C, Method 1014.5
Temperature cycle (10 cycles minimum)	Per MIL-STD-883C, Method 1010.5, -65 to +150°C, Condition C
Thermal shock (10 cycles minimum)	Per MIL-STD-883C, Method 1011.4, -55 to +125°C, Condition B
Moisture resistance	Per MIL-STD-883C, Method 1004.4
Fine and gross leak	Per MIL-STD-883C, Method 1014.5
Electrical test	To data sheet limits

Thermal/Mechanical Sequence:

Fine and gross leak	Per MIL-STD-883C, Method 1014.5
Temperature cycle (10 cycles minimum)	Per MIL-STD-883C, Method 1010.5, -65 to +150°C, Condition C
Constant acceleration	Per MIL-STD-883C, Method 2001.2, 30 kg, Y1 Plane
Fine and gross leak	Per MIL-STD-883C, Method 1014.5
Electrical test	To data sheet limits

Electrostatic discharge	Per MIL-STD-883C, Method 3015
Solderability	Per MIL-STD-883C, Method 2003.3
Solder heat	Per MIL-STD-750C, Method 2031, 10 sec
Salt atmosphere	Per MIL-STD-883C, Method 1009.4, Condition A, 24 hrs min
Lead pull	Per MIL-STD-883C, Method 2004.4, Condition A
Lead integrity	Per MIL-STD-883C, Method 2004.4, Condition B1
Electromigration	Accelerated stress testing of conductor patterns to ensure acceptable lifetime of power-on operation
Resistance to solvents	Per MIL-STD-883C, Method 2015.4

Table H-1. Microprocessor and Microcontroller Tests

TEST	DURATION	SAMPLE SIZE PLASTIC	 CERAMIC
Operating life, 125°C, 5.0 V	1000 hrs	195	195
Operating life, 150°C, 5.0 V	1000 hrs	77*	77
Storage life, 150°C	1000 hrs	129	129
Biased 85°C/85 percent RH, 5.0 V	1000 hrs	129	-
Autoclave, 121°C, 1 ATM	240 hrs	105	-
Temperature cycle, -65 to 150°C	1000 cyc	129	129
Thermal shock, -65 to 150°C	500 cyc	129	129
Electrostatic discharge, ±2 kV		12	12
Latch-up (CMOS devices only)		5	5
Mechanical sequence		-	38
Thermal sequence		-	38
Thermal/mechanical sequence		-	38
PIND		-	15
Internal water vapor		-	5
Solderability		22	22
Solder heat		22	22
Resistance to solvents		12	12
Lead integrity		15	15
Lead pull		15	-
Lead finish adhesion		15	15
Salt atmosphere		15	15
Flammability (UL94-V0)		3	-
Thermal impedance		5	5

*If junction temperature does not exceed plasticity of package.

Table H-2 provides a list of the TMS320C2x devices, the approximate number of transistors, and the equivalent gates. The numbers have been determined from design verification runs.

Table H-2. TMS320C2x Transistors

DEVICE	# TRANSISTORS	# GATES
NMOS: TMS32020	80K	27K
CMOS: TMS320C25	160K	40K

TI Qualification test updates are available upon request at no charge. TI will consider performing any additional reliability test(s), if requested. For more information on TI quality and reliability programs, contact the nearest TI field sales office.

Note:

Texas Instruments reserves the right to make changes in MOS Semiconductor test limits, procedures, or processing without notice. Unless prior arrangements for notification have been made, TI advises all customers to reverify current test and manufacturing conditions prior to relying on published data.

Index

A

- A/D converters (TI) F-2
- A/D interface 6-40
- ABS
  - Absolute Value of Accumulator 4-21
- Absolute Value of Accumulator
  - ABS 4-21
- accumulator 3-8, 3-24
- adaptive filtering 5-63
- ADD 3-33
  - Add to Accumulator with Shift 4-22
- Add P Register to Accumulator
  - APAC 4-34
- Add to Accumulator with Shift
  - ADD 4-22
- Add to High Accumulator
  - ADDH 4-25
- ADDC
  - Add to Accumulator with Carry (TMS320C25) 4-23
- ADDH
  - Add to High Accumulator 4-25
- addition 3-25, 5-35, 5-50
- ADDK
  - Add to Accumulator Short Immediate (TMS320C25) 4-26
- address bus (A15-A0) 2-4
- addressing modes 4-2
- ADDS
  - Add to Accumulator with Sign-Extension Suppressed 4-27
- ADDT 3-24
  - Add to Accumulator with Shift Specified by T Register 4-28
- ADLK
  - Add to Accumulator Long Immediate with Shift 4-30
- ADRK
  - Add to Auxiliary Register Short Immediate (TMS320C25) 4-31
- AIC interface 6-37
- A-law/μ-law companding 5-60
- analog converters (TI) F-2
- analog interface board (AIB) E-7
- analog interface circuits (TI) F-2
- AND
  - AND with Accumulator 4-32
- AND with Accumulator
  - AND 4-32
- ANDK
  - AND Immediate with Accumulator with Shift 4-33
- APAC
  - Add P Register to Accumulator 4-34
- applications 1-5
- architectural overview 3-2
- architecture 3-1
- arithmetic logic unit (ALU) 3-8, 3-24
- arithmetic operations 5-31, 5-34
- assembler C-2, E-3
- assembly language instructions 4-1
- auxiliary register arithmetic unit (ARAU) 3-8, 3-17, 3-20, 5-49
- auxiliary register file bus (AFB) 3-8, 3-19
- auxiliary register pointer (ARP) 3-8, 3-16, 3-20, 3-43
- auxiliary register pointer buffer (ARB) 3-8, 3-19, 3-43
- auxiliary registers 3-8, 3-16, 4-4, C-2

B

- B
  - Branch Unconditionally 4-35
- BACC
  - Branch to Address Specified by Accumulator 4-36
- BANZ
  - Branch on Auxiliary Register Not Zero 4-37
- BBNZ 5-32
  - Branch on Bit Not Equal to Zero 4-39
- BBZ 5-32

  Branch on Bit Equal to Zero 4-40
BC
  Branch on Carry (TMS320C25) 4-41
BGEZ
  Branch if Accumulator Greater Than or Equal to Zero 4-42
BGZ
  Branch if Accumulator Greater Than Zero 4-43
BIO- 2-5, 3-49
BIOZ
  Branch on I
    O Status Equal to Zero 4-44
BIT 3-25, 5-32
  Test Bit 4-45
bit manipulation 5-32
bit-reversed (BR) addressing 3-18, 4-5, **4-7, 5-68**
BITT 3-25, 5-32
  Test Bit Specified by T Register 4-47
BLEZ
  Branch if Accumulator Less Than or Equal to Zero 4-49
BLKD 3-20, 5-23
  Block Move from Data Memory to Data Memory 4-50
BLKP 3-20, 5-23
  Block Move from Program Memory to Data Memory 4-53
block diagram 3-5
block moves 3-20, 5-23
blocks B0-B2 3-9, 3-11, 3-13, C-2
BLZ
  Branch if Accumulator Less Than Zero 4-56
BNC
  Branch on No Carry (TMS320C25) 4-57
BNV 5-34
  Branch if No Overflow 4-58
BNZ
  Branch if Accumulator Not Equal to Zero 4-59
BR- 2-5, 3-70
Branch if Accumulator Equals Zero
  BZ 4-61
Branch if Accumulator Greater Than Zero
  BGZ 4-43
Branch if Accumulator Less Than Zero
  BLZ 4-56
Branch if Accumulator Not Equal to Zero
  BNZ 4-59
Branch if No Overflow
  BNV 4-58
Branch on Auxiliary Register Not Zero
  BANZ 4-37
Branch on Bit Equal to Zero
  BBZ 4-40
Branch on Bit Not Equal to Zero
  BBNZ 4-39
Branch on Carry (TMS320C25)
  BC 4-41
Branch on I
  O Status Equal to Zero
    BIOZ 4-44
Branch on No Carry (TMS320C25)
  BNC 4-57
Branch on Overflow
  BV 4-60
Branch to Address Specified by Accumulator
  BACC 4-36
Branch Unconditionally
  B 4-35
branches 3-25, 3-37, 5-28, C-4
bulletin board E-10
burst-mode operation 3-61
BV 5-34
  Branch on Overflow 4-60
BZ
  Branch if Accumulator Equals Zero 4-61

C

C compiler E-5
CALA 5-7
  Call Subroutine Indirect 4-62
CALL 5-7
  Call Subroutine 4-64
Call Subroutine
  CALL 4-64
Call Subroutine Indirect
  CALA 4-62
calls 5-28, C-4
carry bit (C) 3-25, 3-43, 5-50, C-6
central arithmetic logic unit (CALU) 3-8, 3-22
CLKOUT1 2-6, 3-44, 3-48
CLKOUT2 2-6, 3-48
CLKR 2-6, 3-56, C-7
CLKX 2-6, 3-56, C-7
clock phases 3-48, C-5
clock timing 3-44, 3-48
CMPL
  Complement Accumulator 4-66
CMPR

- Compare Auxiliary Register with Auxiliary Register AR0 4-67
- CNF 3-43
- CNFD 3-13, 5-25, C-5
  - Configure Block as Data Memory 4-68
- CNFP 3-13, 5-25, C-5
  - Configure Block as Program Memory 4-69
- **codec interface 6-34**
- codecs (TI) F-2
- **companding 5-60**
  - **A-law/μ-law 5-60**
  - **sign-magnitude data 5-60**
  - **two's-complement data 5-60**
- Complement Accumulator
  - CMPL 4-66
- computed GOTO 5-14
- Conditional Subtract
  - SUBC 4-171
- Configure Block as Data Memory
  - CNFD 4-68
- Configure Block as Program Memory
  - CNFP 4-69
- context switching 5-16
- continuous-mode operation 3-62, 3-64, 3-66
- **control system 6-47**
- **convolution 5-36**
- crystal oscillator circuit 6-5
- **crystals F-33**
- cycle timings (instructions) C-4, D-1

D

- D/A converters (TI) F-2
- **D/A interface 6-39**
- data address bus (DAB) 3-8, 3-19
- data bus (D15-D0) 2-4, 3-8
- data memory 3-11
- data memory addressing 3-19
- data memory expansion 3-11
- data memory page pointer (DP) 3-8, 3-20, 3-43
- Data Move in Data Memory
  - DMOV 4-71
- **data moves 3-20, 5-35**
- data receive register (DRR) 3-9, 3-16, 3-56, C-7
- data transmit register (DXR) 3-9, 3-16, 3-56
- decode (pipeline) 3-29
- **denormalization 5-47**
- development support E-1
  - **analog interface board (AIB) E-7**
  - C compiler E-5
  - DFDP (digital filter design **package) E-9**
  - **DSP Software Library E-9**
  - emulator (XDS) E-5
  - macro assembler/linker E-3
  - simulator E-4
  - SoftWare Development System (SWDS) E-4
  - **TMS320 Design Kit E-8**
  - TMS320 DSP bulletin board service E-10
  - **TMS320 DSP hotline E-10**
  - **XDS/22 upgrade E-7**
- digital filter design package **(DFDP) E-9**
- **digital filters 5-61**
- DINT
  - Disable Interrupt 4-70
- direct address bus (DRB) 3-8, 3-19
- direct addressing mode 3-19, 4-2, C-2
- **direct memory access (DMA) 3-71, 6-29**
- Disable Interrupt
  - DINT 4-70
- **disk drives 6-47**
- **division 5-42**
- **DMOV 3-20, 5-35**
  - Data Move in Data Memory 4-71
- DR 2-6, 3-56
- DS- 2-4, 3-13
- **DSP Software Library E-9**
- DX 2-6, 3-56

E

- **echo cancellation 6-45**
- EINT 3-53, 3-54
  - Enable Interrupt 4-73
- electrical specifications A-1
- emulator (XDS) E-5
- Enable Interrupt
  - EINT 4-73
- **EPROM interfacing 6-19**
- EPROM memories (TI) F-2
- EXAMPLE
  - Example Instruction 4-18
- Example Instruction
  - EXAMPLE 4-18
- Exclusive-OR with Accumulator
  - XOR 4-184
- execute (pipeline) 3-29

extended-precision arithmetic 5-49
external clock (CLKX) 3-56
external flag (XF) 3-44, 3-50
external memory interface 3-47
external program/data access 3-40

F

Fast Fourier Transforms (FFT) 5-68
fetch (pipeline) 3-29
filtering 5-61
FIR filters 5-61
fixed-point conversion 4-116
floating-point arithmetic 3-24, 5-45
floating-point conversion 3-27, 4-79, 5-45
format bit (FO) 3-43, 3-56
Format Serial Port Registers
  FORT 4-74
FORT C-7
  Format Serial Port Registers 4-74
frame sync pulses 3-56, 3-60, 3-62
frame synchronization mode bit (FSM) 3-43, 3-56, C-6
framing control 3-60
FSR 2-6, 3-56
FSX 2-6, 3-56

G

gates H-5
global memory 3-69, 6-32
global memory allocation register (GREG) 3-8, 3-16, 3-69, 3-70, 6-32
graphics 6-47

H

hardware applications 6-1
  direct memory access (DMA) 6-29
  disk drives 6-47
  echo cancellation 6-45
  global memory 6-32
  graphics 6-47
  high-speed control 6-47
  high-speed modem 6-45
  image processing 6-47
  instrumentation 6-48
  interfacing memories 6-10
  interfacing peripherals 6-34
  numeric processing 6-48
  robotics 6-47
  system applications 6-45
  system control circuitry 6-3
  user target design using XDS 6-7
  voice coding 6-46
hardware stack 3-10, 3-28, 5-9, 5-10, 5-16
Harvard architecture 1-3
HOLD- 2-5, 3-39, 3-53, 3-71
hold mode (HM) 3-43, C-6
HOLDA- 2-5, 3-39, 3-71
hotline E-10

I

I/O interface 3-47, 6-42
I/O port addressing 3-47, 6-42
IACK- 2-5, 3-52, 3-54
IDLE 3-71, C-4
  Idle Until Interrupt 4-75
Idle Until Interrupt
  IDLE 4-75
IIR filters 5-61
image processing 6-47
immediate addressing mode 3-19, 4-9
IN 3-47, 5-23
  Input Data from Port 4-76
indexed addressing 5-49
indirect addressing mode 3-19, 4-4, 5-35
initialization 3-66, 5-2
Input Data from Port
  IN 4-76
instruction cycle timings D-1
instruction pipeline 3-29
instruction register (IR) 3-8, 3-30
instruction set summary 4-12
instructions (assembly language) 4-1
instrumentation 6-48
interface timing analysis 6-27
interfacing memories 6-10
interfacing peripherals 6-34
internal hardware summary 3-7
interrupt acknowledge (IACK) 3-52, 3-54
interrupt flag register (IFR) 3-8, 3-52, 5-16
interrupt mask register (IMR) 3-8, 3-16, 3-52, 5-16
interrupt mode (INTM) 3-43, 3-52, 3-53, 3-54

interrupts 2-5, 3-39, 3-52, 3-54, 3-71, 5-16, C-4
- external interrupt interface 3-53
- operation 3-52
- priorities 3-52, 5-22
- service routine 5-16
- vector locations 3-52, 5-16

IS- 2-4, 3-13

K

key features 1-4

L

LAC
- Load Accumulator with Shift 4-77

LACK
- Load Accumulator Immediate Short 4-78

LACT 3-24, 5-45
- Load Accumulator with Shift Specified by T Register 4-79

LALK
- Load Accumulator Long Immediate with Shift 4-81

LAR
- Load Auxiliary Register 4-82

LARK
- Load Auxiliary Register Immediate Short 4-84

LARP
- Load Auxiliary Register Pointer 4-85

LDP
- Load Data Memory Page Pointer 4-86

LDPK
- Load Data Memory Page Pointer Immediate 4-87

left shifts 3-27, 5-31

linker E-3

Load Accumulator Immediate Short
- LACK 4-78

Load Accumulator Long Immediate with Shift
- LALK 4-81

Load Accumulator with P Register
- PAC 4-121

Load Accumulator with Shift
- LAC 4-77

Load Auxiliary Register
- LAR 4-82

Load Auxiliary Register Immediate Short
- LARK 4-84

Load Auxiliary Register Long Immediate
- LRLK 4-89

Load Auxiliary Register Pointer
- LARP 4-85

Load Data Memory Page Pointer
- LDP 4-86

Load Data Memory Page Pointer Immediate
- LDPK 4-87

Load High P Register
- LPH 4-88

Load Status Register ST0
- LST 4-90

Load Status Register ST1
- LST1 4-92

Load T Register
- LT 4-94

Load T Register and Accumulate Previous Product
- LTA 4-95

Load T Register, Subtract Previous Product
- LTS 4-100

logical operations 5-31

loop control 5-13

LPH
- Load High P Register 4-88

LRLK
- Load Auxiliary Register Long Immediate 4-89

LST
- Load Status Register ST0 4-90

LST1
- Load Status Register ST1 4-92

LT
- Load T Register 4-94

LTA 5-39
- Load T Register and Accumulate Previous Product 4-95

LTD 5-39
- Load T Register, Accumulate Previous Product, and Move Data 4-97

LTP
- Load T Register and Store P Register in Accumulator 4-99

LTS
- Load T Register, Subtract Previous Product 4-100

M

MAC 5-37, 5-38, C-4
  Multiply and Accumulate 4-101
MACD 5-37, 5-38, C-4
  Multiply and Accumulate with Data Move 4-104
macro assembler E-3
MAR
  Modify Auxiliary Register 4-107
mask options 3-12, G-1
memory 3-11, 5-23, C-2
  addressing modes 3-19, 4-2
  auxiliary registers 3-16
  block moves 5-23
  data memory 3-11
  global memory 3-69, 6-32
  memory expansion 3-11, 3-12
  memory maps 3-13
  memory-mapped registers 3-16
  program memory 3-12
memory addressing modes 3-19, 4-2
  direct addressing 3-19, 4-2, C-2
  immediate addressing 3-19, 4-9
  indirect addressing 3-19, 4-4
memory combinations 3-47
memory interface 6-10
memory management 5-23
memory maps 3-13
memory products (TI) F-2
memory-mapped registers 3-13, 3-16
microcall stack (MCS) register 3-9, 3-30
microcomputer mode 2-5, 3-12, 3-15
microprocessor mode 2-5, 3-12, 3-15
modem 6-45
Modify Auxiliary Register
  MAR 4-107
MP/MC- 2-5, 3-15
MPY 5-38, C-3
  Multiply 4-109
MPYA
  Multiply and Accumulate Previous Product (TMS320C25) 4-110
MPYK C-3
  Multiply Immediate 4-111
MPYS
  Multiply and Subtract Previous Product (TMS320C25) 4-112
MPYU 3-26
  Multiply Unsigned (TMS320C25) 4-113
MS/PC-DOS E-9
MSC- 2-5, 6-16
multiplexed external data bus 3-35
multiplication 5-37, 5-54, C-3
multiplier 3-9, 3-26, 5-37
Multiply
  MPY 4-109
Multiply and Accumulate
  MAC 4-101
Multiply and Accumulate with Data Move
  MACD 4-104
Multiply Immediate
  MPYK 4-111
Multiply Unsigned (TMS320C25)
  MPYU 4-113
multiprocessing 3-68

N

NEG
  Negate Accumulator 4-114
Negate Accumulator
  NEG 4-114
No Operation
  NOP 4-115
nomenclature E-14
NOP
  No Operation 4-115
NORM 3-24, 5-45, C-4
  Normalize Contents of Accumulator 4-116
normalization 5-45, 5-47
Normalize Contents of Accumulator
  NORM 4-116
numeric processing 6-48

O

on-chip program RAM execution 5-28
on-chip program ROM 3-14
on-chip RAM 3-9, 3-13, 5-25
on-chip RAM configuration control bit (CNF) 3-43
OR
  OR with Accumulator 4-118
OR Immediate with Accumulator with Shift
  ORK 4-119
OR with Accumulator
  OR 4-118
ordering information E-11
ORK
  OR Immediate with Accumulator with Shift 4-119

oscillator circuit 6-5
OUT 3-47, 5-23
    Output Data to Port 4-120
Output Data to Port
    OUT 4-120
overflow flag (OV) 3-43, 5-34
overflow managment 5-34
overflow mode (OVM) 3-44, 5-31, 5-34
overflow saturation mode 3-24

P

P register (PR) 3-25. 3-26, 5-37
PAC
    Load Accumulator with P Register 4-121
part numbers E-11
PC stack 5-7
PC/MS-DOS E-9
period register (PRD) 3-9, 3-16, 3-44, 5-11, C-5
peripheral interface 6-34
PID control 5-75
pinouts 2-2
pipeline operation 3-29
    decode 3-29
    execute 3-29
    fetch 3-29
    prefetch 3-29
    three-level pipeline 3-29
    two-level pipeline 3-29
    wait states 3-29
PM bits 3-44. 5-42. C-2
POP 3-29
    Pop Top of Stack to Low Accumulator 4-122
Pop Top of Stack to Data Memory
    POPD 4-124
Pop Top of Stack to Low Accumulator
    POP 4-122
POPD 3-29, 5-10
    Pop Top of Stack to Data Memory 4-124
powerdown mode 3-46
powerup reset circuit 6-3
prefetch (pipeline) 3-29
prefetch counter (PFC) 3-9, 3-30
product quality/reliability H-1
product register (PR) 3-9, 3-26, 5-37
product shift mode (PM) bits 3-27, 3-44, 5-32, 5-42, C-6
program access (on-chip) 3-40
program address bus (PAB) 3-9
program bus 3-9
program counter (PC) 3-9, 3-28, 3-37
program memory 3-12
program memory expansion 3-12
PROM interfacing 6-11
prototype devices G-1
PS- 2-4, 3-13
PSHD 3-29, 5-10
    Push Data Memory Value onto Stack 4-126
PUSH 3-29
    Push Low Accumulator onto Stack 4-128
Push Data Memory Value onto Stack
    PSHD 4-126
Push Low Accumulator onto Stack
    PUSH 4-128

Q

quality/reliability H-1
queue instruction register (QIR) 3-9, 3-30
Q15 format 3-27, 5-45

R

R/W- 2-4
RAM interfacing 6-24
RC
    Reset Carry Bit (TMS320C25) 4-129
READY 2-4, 3-70
receive framing synchronization signal (FSR) 3-56
receive shift register (RSR) 3-9, 3-56
received serial data (DR) 3-56
reliability tests H-2
repeat counter (RPTC) 3-9, 3-29, 3-46, 3-53, 5-13
reset (RS-) 2-5, 3-39, 3-41, 3-52, C-5
Reset Carry Bit (TMS320C25)
    RC 4-129
reset circuit 6-3
Reset External Flag
    RXF 4-142
Reset Hold Mode (TMS320C25)
    RHM 4-132
Reset Overflow Mode
    ROVM 4-135

- Reset Serial Port Transmit Mode
  - RTXM 4-141
- Reset Sign-Extension Mode
  - RSXM 4-139
- Reset Test
  - Control Flag (TMS320C25)
    - RTC 4-140
- RET 3-37, 3-53, 5-7
  - Return from Subroutine 4-130
- Return from Subroutine
  - RET 4-130
- reverse-carry (rc) propagation 3-18, 4-5, 4-7, 5-68
- RFSM
  - Reset Serial Port Frame Synchronization Mode (TMS320C25) 4-131
- RHM
  - Reset Hold Mode (TMS320C25) 4-132
- right shift 3-27, 5-31
- RINT 3-52, C-7
- robotics 6-47
- ROL
  - Rotate Accumulator Left (TMS320C25) 4-133
- ROM codes G-1
- ROR
  - Rotate Accumulator Right (TMS320C25) 4-134
- Rotate Accumulator Left (TMS320C25)
  - ROL 4-133
- Rotate Accumulator Right (TMS320C25)
  - ROR 4-134
- ROVM 3-24, 5-31, 5-34
  - Reset Overflow Mode 4-135
- RPT 3-46, 3-53, 5-13
  - Repeat Instruction as Specified by Data Memory Value 4-136
- RPTC 3-29
- RPTK 3-46, 3-53, 5-13
  - Repeat Instruction as Specified by Immediate Value 4-138
- RSXM 5-31
  - Reset Sign-Extension Mode 4-139
- RTC
  - Reset Test
    - Control Flag (TMS320C25) 4-140
- RTXM
  - Reset Serial Port Transmit Mode 4-141
- RXF 3-50
  - Reset External Flag 4-142

S

- SACH C-4
  - Store High Accumulator with Shift 4-143
- SACL 3-33, C-4
  - Store Low Accumulator with Shift 4-144
- SAR
  - Store Auxiliary Register 4-145
- SBLK
  - Subtract from Accumulator Long Immediate with Shift 4-147
- SBRK
  - Subtract from Auxiliary Register Short Immediate (TMS320C25) 4-148
- SC
  - Set Carry Bit (TMS320C25) 4-149
- scaling 5-35
- scaling shifter 3-23
- serial port 3-56, C-6
  - burst-mode operation 3-61
  - continuous-mode operation 3-62, 3-64, 3-66
  - timing and framing control 3-60
  - transmit/receive operations 3-58
- serial-port clock (CLKR) 3-56
- Set Carry Bit (TMS320C25)
  - SC 4-149
- Set External Flag
  - SXF 4-178
- Set Hold Mode (TMS320C25)
  - SHM 4-154
- Set Overflow Mode
  - SOVM 4-155
- Set P Register Output Shift Mode
  - SPM 4-159
- Set Serial Port Transmit Mode
  - STXM 4-168
- Set Sign-Extension Mode
  - SSXM 4-166
- Set Test
  - Control Flag (TMS320C25)
    - STC 4-167
- SFL 3-26, 5-35
  - Shift Accumulator Left 4-150
- SFR 3-26, 5-34
  - Shift Accumulator Right 4-151
- SFSM
  - Set Serial Port Frame Synchronization Mode (TMS320C25) 4-153
- Shift Accumulator Left
  - SFL 4-150
- Shift Accumulator Right

SFR 4-151
shift modes 3-27, 3-44, 5-31, 5-42
snifters 3-10, 5-34
  accumulator 3-23
  accumulator output 3-23, 5-34
  product register output 3-23, 5-34
  scaling shifter 3-23
SHM
  Set Hold Mode (TMS320C25) 4-154
signal descriptions 2-1
sign-extension mode 5-31
sign-extension mode bit (SXM) 3-26, 3-44, 5-31, C-6
sign-magnitude data 5-31, 5-60
simulator E-4
single-instruction loops 5-13
sockets (TI) F-28
software applications 5-1
SoftWare Development System (SWDS) E-4
Software Interrupt
  TRAP 4-183
software library E-9
software stack 5-10
software stack expansion 5-10
SOVM 3-24, 5-31, 5-34
  Set Overflow Mode 4-155
SPAC
  Subtract P Register from Accumulator 4-156
specifications A-1
SPH
  Store High P Register (TMS320C25) 4-157
SPL
  Store Low P Register (TMS320C25) 4-158
SPM 5-32, 5-34
  Set P Register Output Shift Mode 4-159
SQRA 3-26, 5-41
  Square and Accumulate Previous Product 4-160
SQRS 3-26, 5-41
  Square and Subtract Previous Product 4-161
Square and Accumulate Previous Product
  SQRA 4-160
Square and Subtract Previous Product
  SQRS 4-161
square-root routine 5-7
SST
  Store Status Register ST0 4-162
SST1
  Store Status Register ST1 4-164
SSXM 5-31
  Set Sign-Extension Mode 4-166
stack 3-28, 5-7, 5-9
static RAM interfacing 6-24
status registers 3-10, 3-42, 5-31, C-2, C-5
STC
  Set Test
    Control Flag (TMS320C25) 4-167
Store Auxiliary Register
  SAR 4-145
Store High Accumulator with Shift
  SACH 4-143
Store High P Register (TMS320C25)
  SPH 4-157
Store Low Accumulator with Shift
  SACL 4-144
Store Low P Register (TMS320C25)
  SPL 4-158
Store Status Register ST0
  SST 4-162
Store Status Register ST1
  SST1 4-164
STRB- 2-4
STXM
  Set Serial Port Transmit Mode 4-168
SUB
  Subtract from Accumulator with Shift 4-169
SUBB
  Subtract from Accumulator with Borrow (TMS320C25) 4-170
SUBC 5-42, C-3
  Conditional Subtract 4-171
SUBH
  Subtract from High Accumulator 4-173
SUBK
  Subtract from Accumulator Short Immediate (TMS320C25) 4-174
subroutines 5-7
SUBS
  Subtract from Low Accumulator with Sign-Extension Suppressed 4-175
SUBT 3-24
  Subtract from Accumulator with Shift Specified by T Register 4-176
Subtract from Accumulator with Shift
  SUB 4-169
Subtract from High Accumulator
  SUBH 4-173
Subtract P Register from Accumulator
  SPAC 4-156

subtraction 3-25, 5-35, 5-52
SXF 3-50
  Set External Flag 4-178
SXM 5-31, C-2
SYNC- 2-5, 3-68
synchronization 3-68, C-5
system applications 6-45
system control 3-28
system control circuitry 6-3
system migration C-1

T

T register (TR) 3-26, 5-37
Table Read
  TBLR 4-179
Table Write
  TBLW 4-181
TBLR 3-21, 5-23
  Table Read 4-179
TBLW 3-21, 5-23
  Table Write 4-181
temporary register (TR) 3-10, 3-26, 5-37
Test Bit
  BIT 4-45
Test Bit Specified by T Register
  BITT 4-47
test control flag bit (TC) 3-44
timer 3-10, 3-44, 5-11, C-5
timer interrupt (TINT) 3-45, 3-52, 5-11
timer register (TIM) 3-16, 3-44, 5-11
timing analysis for interfacing 6-27
timing control 3-60, 5-11
TMS320 Design Kit E-8
TMS320 development tool nomenclature E-14
TMS320 device nomenclature E-13
TMS320 DSP bulletin board service E-10
TMS320 DSP hotline E-10
TMS320C1x to TMS32020 system migration C-2
TMS320C25 1-3
TMS32020 1-3
TMS32020 to TMS320C25 system migration C-4
transistors H-5
transmit framing synchronization signal (FSX) 3-56
transmit mode bit (TXM) 3-44, 3-56
transmit shift register (XSR) 3-10, 3-56
transmitted serial data (DX) 3-56
TRAP 3-52
  Software Interrupt 4-183
two's-complement data 5-32, 5-38, 5-45, 5-60
two-word instructions 3-36

U

user target design using XDS 6-7

V

VAX/VMX E-9
VCC 2-6
voice coding 6-46
VSS 2-6

W

wait states 3-34, 6-16
wait-state generator 6-16

X

XDS design considerations 6-7
XDS emulator E-5
XDS/22 upgrade E-7
XF 2-5, 3-44, 3-50
XINT 3-52, C-7
XOR
  Exclusive-OR with Accumulator 4-184
XOR Immediate with Accumulator with Shift
  XORK 4-185
XORK
  XOR Immediate with Accumulator with Shift 4-185
XSR 3-56
X1 2-6
X2/CLKIN 2-6

Z

ZAC
    Zero Accumulator 4-186
ZALH
    Zero Low Accumulator and Load High Accumulator 4-187
ZALR
    Zero Low Accumulator, Load High Accumulator with Rounding (TMS320C25) 4-188
ZALS
    Zero Accumulator, Load Low Accumulator with Sign-Extension Suppressed 4-189
Zero Accumulator
    ZAC 4-186

TEXAS
INSTRUMENTS

December 1987
Printed in U.S.A.

SPRU014