

MICROCOMPUTER

MN1030/MN103S

MN1030/MN103S Series Instruction Manual

Pub.No.13250-040E

PanaXSeries is a trademark of Matsushita Electric Industrial Co., Ltd.

All other corporation names, logotype and product names written in this book are trademarks or registered trademarks of their respective corporations.

Request for your special attention and precautions in using the technical information and semiconductors described in this book

- (1) An export permit needs to be obtained from the competent authorities of the Japanese Government if any of the products or technologies described in this book and controlled under the "Foreign Exchange and Foreign Trade Law" is to be exported or taken out of Japan.
- (2) The technical information described in this book is limited to showing representative characteristics and applied circuits examples of the products. It neither warrants non-infringement of intellectual property right or any other rights owned by our company or a third party, nor grants any license.
- (3) We are not liable for the infringement of rights owned by a third party arising out of the use of the product or technologies as described in this book.
- (4) The products described in this book are intended to be used for standard applications or general electronic equipment (such as office equipment, communications equipment, measuring instruments and household appliances).
 - Consult our sales staff in advance for information on the following applications:
 - Special applications (such as for airplanes, aerospace, automobiles, traffic control equipment, combustion equipment, life support systems and safety devices) in which exceptional quality and reliability are required, or if the failure or malfunction of the products may directly jeopardize life or harm the human body.
 - Any applications other than the standard applications intended.
- (5) The products and product specifications described in this book are subject to change without notice for modification and/or improvement. At the final stage of your design, purchasing, or use of the products, therefore, ask for the most up-to-date Product Standards in advance to make sure that the latest specifications satisfy your requirements.
- (6) When designing your equipment, comply with the guaranteed values, in particular those of maximum rating, the range of operating power supply voltage, and heat radiation characteristics. Otherwise, we will not be liable for any defect which may arise later in your equipment.
 Even when the products are used within the guaranteed values, take into the consideration of incidence of break down and failure mode, possible to occur to semiconductor products. Measures on the systems such as redundant design, arresting the spread of fire or preventing glitch are recommended in order to prevent physical injury, fire, social damages, for example, by using the products.
- (7) When using products for which damp-proof packing is required, observe the conditions (including shelf life and amount of time let standing of unsealed items) agreed upon when specification sheets are individually exchanged.
- (8) This book may be not reprinted or reproduced whether wholly or partially, without the prior written permission of Matsushita Electric Industrial Co., Ltd.

If you have any inquiries or questions about this book or our semiconductors, please contact one of our sales offices listed at the back of this book.

About This Manual

This document contains a detailed description of the instruction set for the MN1030/MN103S Series.

This document concentrates on the AM32 microcontroller core. When the specifications differ between cores, separate descriptions appear next to icons indicating the appropriate core or cores

Chapter 1 provides an overview of the instruction set--instruction functions, formats, and the like.

Chapter 2 contains detailed descriptions of the individual instructions--operation, effect on PSW flags, and the like. Chapter 3 contains usage notes--a description of the pipeline architecture, programming notes, usage recommendations, and the like.

The Appendix contains charts for the entire instruction set and instruction mappings.

■ Finding Information

This document incorporates the following aids for locating necessary information as quickly as possible.

- (1) Index tabs in the inside margins of left-hand pages indicate Chapters.
- (2) The table of contents near the beginning of this document lists section headings.
- (3) As you flip through the document, the page header gives the chapter; the footer, the section heading.
- (4) The index near the end of this document lists page references for all instructions and instruction variants. In Chapter 2, the instruction mnemonic appears in the page footer for right-hand pages.

■ Related Manuals

The following related manuals are available. Please contact our sales representative for more details.

< For MN1030 Series Users >

MN1030 Series Cross Assembler User's Manual

<Describes the assembler syntax and notation>

MN1030 Series C Compiler User's Manual: Usage Guide

<Describes the installation, the commands, and options of the C Compiler>

MN1030/MN103S/MN103E Series C Compiler User's Manual: Language Description

<Describes the syntax of the C Compiler>

MN1030/MN103S/MN103E Series C Compiler User's Manual: Library Reference

<Describes the the standard library of the C Compiler>

MN1030/MN103S Series C Source Code Debugger for Windows® User's Manual

<Describes the use of the C source code debugger for Windows®>

MN1030/MN103S Series Installation Manual

<Describes the installation of the C compiler, cross-assembler and C source code debugger and the procedure for bringing up the in-circuit emulator>

< For MN103S Series Users >

MN103S Series Cross Assembler User's Manual

<Describes the assembler syntax and notation>

MN103S Series C Compiler User's Manual: Usage Guide

<Describes the installation, the commands, and options of the C Compiler>

MN1030/MN103S/MN103E Series C Compiler User's Manual: Language Description

<Describes the syntax of the C Compiler>

MN1030/MN103S/MN103E Series C Compiler User's Manual: Library Reference

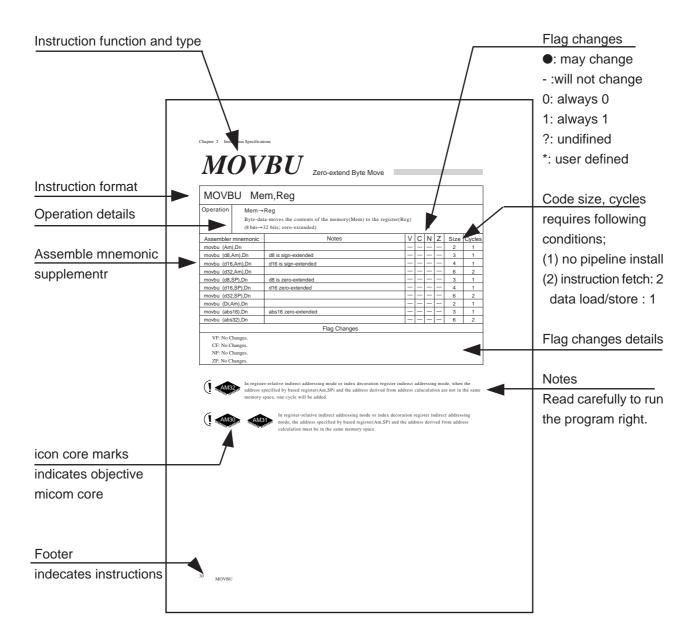
<Describes the the standard library of the C Compiler>

MN1030/MN103S Series C Source Code Debugger for Windows® User's Manual

<Describes the use of the C source code debugger for Windows®>

MN1030/MN103S Series Installation Manual

<Describes the installation of the C compiler, cross-assembler and C source code debugger and the procedure for bringing up the in-circuit emulator>



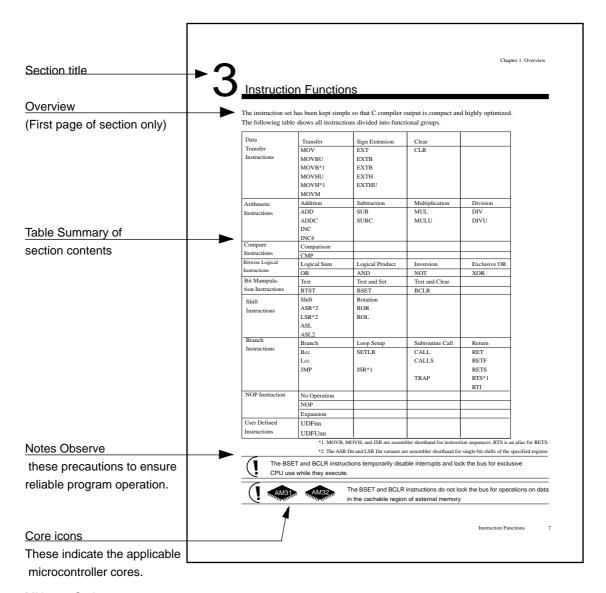
■ Page Layouts

The three layouts below are the standards for the three Chapters.

Chapter 1 pages give the section title, an overview, the main text, and notes.

Chapter 2 pages give the instruction syntax, operational description, and notes.

Chapter 3 pages feature such items as pipeline operation diagrams, code samples, and notes.



MN1030 Series

AM30: First generation AM31: Second generation

MN103S Series

AM32: Third generation

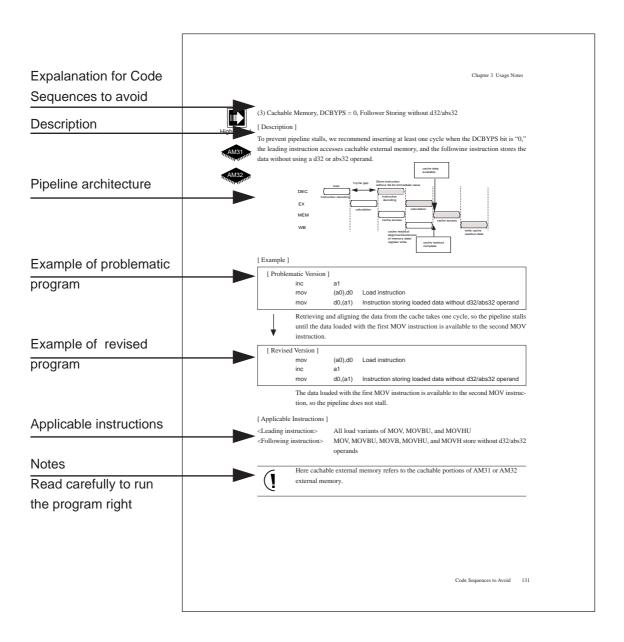


Table of Contents

Chapter 1 Overview

Chapter 2 Instruction Specifications

Chapter 3 Usage Notes

Appendix

Index

1

2

3

4

5

Table of Contents

Chapte	pter 1 Overview			
1	1 Instruction Set2			
2		ter Set		
	2.1	Data Register		
	2.2	Address Registers		
	2.3	Stack Pointer		
	2.4	Program Counter		
	2.5	Multiply/Divided Register		
	2.6	Processor Status Word		
	2.7	Loop Instruction Register		
	2.8	Loop Address Register		
3	Instru	ction Functions		
	3.1	Data Transfer Instructions		
	3.2	Arithmetic Instructions		
	3.3	Compare Instructions		
	3.4	Bitwise Logical Instructions		
	3.5	Bit Manipulation Instructions		
	3.6	Shift Instructions		
	3.7	Branch Instructions		
	3.8	NOP Instruction		
	3.9	User Defined Instructions		
4	Memo	ory Layout		
5	Addre	ssing Modes		
	5.1	Register Direct Addressing		
	5.2	Immediate Addressing		
	5.3	Register Indirect Addressing		
	5.4	Register Relative Indirect Addressing		
	5.5	Absolute Addressing		
	5.6	Register Indirect Addressing with Indexing		
6	Instru	ction Formats		
	6.1	Data Formats		
	6.2	Byte Order		
Chapte	r 2	Instruction Specifications		
	Symbo	ol Definitions24		
	Data 7	Transfer Instructions		
	M	OV Reg1,Reg2		
	M	OV Mem,Reg		
	M	OV Reg,Mem28		
	M	OV imm,Reg29		

Bit Manipular	tion Instructions
BTST	imm,Dn
BTST	imm,Mem
BSET	Dm,(An)
BSET	imm,Mem
BCLR	Dm,(An)
BCLR	imm,Mem
Shift Instructi	ions
ASR	Dm,Dn
ASR	imm8,Dn71
ASR	Dn
LSR	Dm,Dn
LSR	imm8,Dn
LSR	Dn
ASL	Dm,Dn
ASL	imm8,Dn
ASL2	Dn
ROR	Dn
ROL	Dn
Branch Instru	
Всс	label
Lcc	82
SETLB	83
JMP	(An)
JMP	label
CALL	label 85
CALLS	
	(An)
CALLS	
RET RETF	
RETS	
JSR	(An) 92
JSR	label
RTS	
RTI	
TRAP	
NOP Instructi	ion
NOP	97
User Defined	Instructions
UDFnn	Dm,Dn(nn=00 to 15, 20 to 35)
UDFnn	imm,Dn(nn=00 to 15, 20 to 35)
UDFUnn	imm.Dn(nn=00 to 15, 20 to 35)

	MOVRII	Mem,Reg	Λ
		Reg,Mem	
	MOVBO	Mem,Reg	
	MOVB	Reg,Mem 33	
		Mem,Reg	
		Reg,Mem	
		Mem,Reg 30	
		eg,Mem 3'	
	MOVM	(SP),regs 38	
	MOVM	egs,(SP)	
	EXT	Dn	
	EXTB	Dn	
	EXTBU	Dn	
	EXTH	Dn	
	EXTHU	Dn	
	CLR	Dn	5
Ar	thmetic In	structions	
	ADD	Reg1,Reg2	6
	ADD	imm,Reg47	7
	ADDC	Dm,Dn	8
	SUB	Reg1,Reg2	9
	SUB	imm,Reg50	0
	SUBC	Dm,Dn	1
	MUL	Dm,Dn	2
	MULU	Dm,Dn	3
	DIV	Dm,Dn	4
	DIVU	Dm,Dn	5
	INC	Reg	6
	INC4	An	7
Co	mpare Inst	ructions	
C 0	-		0
	CMP	Reg1,Reg2	
	CMP	imm,Reg	8
Bit	wise Logic	cal Instructions	
	AND	Dm,Dn	9
	AND	imm,Dn	9
	AND	imm,PSW	0
	OR	Dm,Dn	1
	OR	imm,Dn	1
	OR	imm,PSW	2
	XOR	Dm,Dn	3
	XOR	imm,Dn	3
	NOT	Dn 64	4

Chapter 3 Usage Notes_____

	Note	es to Programmers
1		line Architencture
-	1.1	Pipeline Operation
	1.2	Register to Register (RR) Operations
	1.3	Data Load Operations
	1.4	Data Store Operations
	1.5	Branching Operations
	1.6	Complex Instructions
	1.7	Special Instructions
	1.8	Pipeline Stall
2	Dans	gerous Code Sequences
	2.1	Load/Store Instructions
	2.2	Instructions Writing to IE and IM Bits
	2.3	Sequences Updating and Referencing PSW Flags
	2.4	Sequences Writing to and Referencing PSW Flags
	2.5	MUL/MULU after Write to A0
	2.6	Displacements with CALLS and JSR(AM31 Only)
	2.7	User Defined Instructions after MOVM[regs],(SP) (AM31 Only)
	2.8	BSET and BCLR with Cachable External Memory
3	Code	e Sequences to Avoid
	3.1	Time-Critical Code
	3.2	Load/Store Instructions
	3.3	Instructions Following Branch and Other Instructions
	3.4	Instructions Following Load Instructions
	3.5	Instructions Following DIV/DIVU with Zero Dividend
	3.6	Instructions Preceding Loc
	3.7	Instructions Preceding SETLB
	3.8	Instructions Preceding RETF
	3.9	Instructions at CALL/CALLS Targets
4	Boile	er Plate Code Sequences
•	4.1	Reset Routine
	4.2	Interrupt Handlers
	4.3	Function Called with CALL Only
	4.4	Function Called with Both CALL and CALLS
		endix160
		nstruction map
		•
	IND	
	Inde	x

Overview 1

1

Instruction Set

The MN1030/MN103S Series of 32-bit microcontrollers has a simple instruction set designed to make C compiler output compact and highly optimized. It minimizes code size by adopting a variable length instruction format with a basic instruction length of only one byte. It is thus able to minimize increases in assembler program code size even though the only data transfers supported by the simple instruction set are load and store.

CPU Cores

AM30, AM31, and AM32 are 32-bit embedded application microcontroller cores from the Matsushita AM Series of C-oriented 8-, 16-, and 32-bit microcontrollers. Their specifications differ for certain instructions. The following are brief overviews of these three cores.

MN1030 Series

AM30: First-generation microcontroller core



supporting connection to ROM, RAM, and Flash memory for instructions and RAM for data

AM31: Second-generation microcontroller core



supporting connection to cache memory for both instructions and data. General-purpose microcomputer based on this core:

MN103002A

MN103S Series

AM32: Third-generation microcontroller core



supporting connection to ROM, RAM, Flash memory, and cache memory for instructions and RAM and cache memory for data.

This document concentrates on the AM32 microcontroller core. When the specifications differ between cores, separate descriptions appear next to icons indicating the appropriate core or cores.

Register Set

The register set includes data registers for arithmetic and general use, address registers for use as pointers, and the stack pointer. This set greatly contributes to increasing the internal architecture's performance by reducing code size and boosting parallel use of pipeline stages.

This register set incorporates features enabling the use of C and other high-level languages.

	31	0
Data registers	D0	
_	D1	
	D2	
	D3	
	31	0
Address registers	A0	
· ·	A1	
	A2	
	A3	
	31	0
Stack pointer	SP	
	31	0
Program counter	PC	\neg ĭ
3		
	3 <u>1</u>	0
Multiply/divide register	MDR	
	15	0
Processor status word	PSW	
	24	0
Loop instruction register	31	0
Loop instruction register	LIR	
	31	0
Loop address register	LAR	$\overline{}$
-		

The Loop Instruction Register (LIR) and Loop Address Register (LAR) are for speeding up the branch to and execution of the first instruction in a loop. The SETLB (Set Loop Beginning) instruction loads them with the next four instruction bytes and the address of the fifth, respectively. The Lcc (Loop) instruction then uses these stored values to jump-start execution of the first instruction in the loop while fetching additional instruction bytes.

2.1 Data Registers

D0 to D3: Data Registers (32 bits x 4)

These four 32-bit registers are for arithmetic and general use. Data values are automatically zero-extended to 32 bits when they are loaded from memory. The EXTB and EXTH instructions are also available for sign-extending them once loaded.

For 8-bit data, a load operation copies the data from memory into the lowest eight bits of the register and zeros the other bits. A store copies the lowest eight bits of the register to memory. Following the load operation with an EXTB instruction sign-extends it from 8 bits to 32.

For 16-bit data, a load operation copies the data from memory into the lowest 16 bits of the register and zeros the other bits. A store copies the lowest 16 bits of the register to memory. Following the load operation with an EXTH instruction sign-extends it from 16 bits to 32.

2.2 Address Registers

A0 to A3: Address Registers (32 bits x 4)

These four 32-bit registers are for use as address pointers, so support only the operations relevant to address calculations: addition, subtraction, and comparison.

Because the contents are pointers, transfers to and from memory are always 32 bits wide.

2.3 Stack Pointer

SP: Stack Pointer (32 bits x 1)

This 32-bit pointer indicates the address at the top of the stack.

Because addressing is by the word, the lowest two bits of any value loaded into this register must be '00'--that is, a multiple of four.

2.4 Program Counter

PC: Program Counter (32 bit x1)

This 32-bit register holds the address of the instruction currently executing.

2.5 Multiply/Divide Register

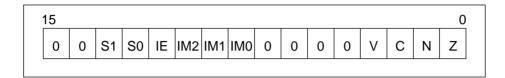
MDR: Multiply/ Divide Register (32 bits x 1)

This 32-bit register is for use by multiply and divide instructions. After a multiply operation, it holds the top 32 bits of the 64-bit result. After a divide operation, it holds the 32-bit remainder; before, the top 32 bits of the 64-bit dividend.

2.6 Processor Status Word

PSW: Processor Status Word (16 bits x 1)

This 16-bit register displays CPU status and controls certain operations. Examples of the former function include the flag bits indicating calculation results; of the latter, the interrupt mask level bits.



Z: Zero flag

This bit goes to "1" if the calculation leaves "0" in all bits of the result and to "0" otherwise. After a reset, it is "0."

N: Negative flag

This bit goes to "1" if the calculation leaves "1" in the most significant bit (MSB) of the result and to "0" otherwise. After a reset, it is "0."

C: Carry flag

This bit goes to "1" if the calculation produces a carry out of or borrow into the most significant bit (MSB) of the result and to "0" otherwise. After a reset, it is "0."

V: Overflow flag

This bit goes to "1" if the result exceeds the bounds for signed integers and to "0" otherwise. After a reset, it is "0."

IM2 to IM0: Interrupt mask level

These three bits offer a choice of eight interrupt mask levels from 0 (000B) to 7 (111B). The hardware accepts only interrupt requests with levels higher than the specified value, and, when it accepts one, sets these bits to the interrupt request level to block subsequent interrupt requests at that and lower levels until interrupt processing is complete. After a reset, all bits are "0" for an interrupt mask level of 0.

IE: Interrupt Enable

This control bit is normally "1" to enable interrupts. When the hardware accepts an interrupt request, however, this bit goes to "0" to disable further interrupts. To support nested interrupts, the user application program must, therefore, reset this bit to "1." After a reset, it is "0."

S1 to S0: Software Bits

These two bits are for operating system use in controlling software. They are not for use by user application programs. After a reset, they are both "0."

2.7 Loop Instruction Register

LIR: Loop Instruction Register (32 bits x 1)

This 32-bit register, used only by the SETLB (Set Loop Beginning) and Lcc (Loop) instructions, holds the first four instruction bytes of the loop for use in speeding up iterations. The SETLB instruction loads it prior to the loop, and the Lcc instruction at the end of the loop then executes the copy while the pipeline fetches more instruction bytes starting from the fifth.

For further details, see the SETLB description in Chapter 2.

2.8 Loop Address Register

LAR: Loop Address Register (32 bit x 1)

This 32-bit register, used only by the SETLB (Set Loop Beginning) and Lcc (Loop) instructions, holds the address of the fifth instruction byte of the loop.

Instruction Functions

The instruction set has been kept simple so that C compiler output is compact and highly optimized. The following table shows all instructions divided into functional groups.

Data	Transfer	Sign Extension	Clear	
Transfer	MOV	EXT	CLR	
Instructions	MOVBU	EXTB		
	MOVB*1	EXTB		
	MOVHU	EXTH		
	MOVH*1	EXTHU		
	MOVM			
Arithmetic	Addition	Subtraction	Multiplication	Division
Instructions	ADD	SUB	MUL	DIV
	ADDC	SUBC	MULU	DIVU
	INC			
	INC4			
Compare	Comparison			
Instructions	CMP			
Bitwise Logical	Logical Sum	Logical Product	Inversion	Exclusive OR
Instructions	OR	AND	NOT	XOR
Bit Manipula-	Test	Test and Set	Test and Clear	
tion Instructions	BTST	BSET	BCLR	
Shift	Shift	Rotation		
Instructions	ASR*2	ROR		
	LSR*2	ROL		
	ASL			
	ASL2			
Branch	Branch	Loop Setup	Subroutine Call	Return
Instructions	Bcc	SETLB	CALL	RET
	Lcc		CALLS	RETF
	JMP	JSR*1		RETS
			TRAP	RTS*1
				RTI
NOP Instruction	No Operation			
	NOP			
	Expansion			
User Defined	UDFnn			
Instructions	UDFUnn			

st1. MOVB, MOVH, and JSR are assembler shorthand for instruction sequences. RTS is an alias for RETS.



The BSET and BCLR instructions temporarily disable interrupts and lock the bus for exclusive CPU use while they execute.





The BSET and BCLR instructions do not lock the bus for operations on data in the cachable region of external memory.

^{*2.} The ASR Dn and LSR Dn variants are assembler shorthand for single-bit shifts of the specified register.

3.1 Data Transfer Instructions

Data transfer instructions copy data between registers or between a register and memory. They fall into three groups: MOV, EXT, and CLR.

The MOV group offers a variety of modes for addressing data and provides sign- and zero-extension as necessary for displacements, immediate values, etc.

The EXT group provides sign- and zero-extension within the specified register or to the Multiply/Divide Register (MDR).

The CLR instruction sets the specified register to zero.

Instruction	Description
MOV	Word (32-bit) transfer between registers, word transfer between a register
	and memory, or loading of an immediate value into a register
MOVBU	Byte transfer between registers with zero-extension for loads
MOVB*1	Byte transfer between registers with sign-extension for loads
MOVHU	Half-word (16-bit) transfer between registers with zero-extension for loads
MOVH*1	Half-word (16-bit) transfer between registers with sign-extension for loads
MOVM	Multiregister transfer to and from stack in memory
EXT	Sign-extension of 32-bit word register into Multiply/Divide Register (MDR)
EXTB	Sign-extension of byte to 32 bits
EXTBU	Zero-extension of byte to 32 bits
EXTH	Sign-extension of half-word to 32 bits
EXTHU	Zero-extension of half-word to 32 bits
CLR	Register clear

^{*1.} MOVB and MOVH are assembler shorthand for instruction sequences.

3.2 Arithmetic Instructions

Arithmetic instructions perform an arithmetic operation on the two source operands (or one), store the result in a register, and--except for INC and INC4 with address registers, ADD with the Stack Pointer (SP), etc.--update the PSW flags according to the result. Because of their frequent use in address calculations, there are separate instructions for incrementing by 1 and 4.

Instruction	Description
Addition	Addition with carry
Subtraction	Subtraction with carry
Multiplication (signed)	Multiplication (unsigned)
Division (signed)	Division (unsigned)

3.3 Compare Instructions

The compare instructions subtract an immediate value or the contents of a register from the contents of another register, setting PSW flags for use in conditional branch instructions.

Instruction	Description
CMP	Comparison

3.4 Bitwise Logical Instructions

Bitwise logical instructions perform a logical operation on the two source operands (or one), store the result in a register, and update the PSW flags according to the result.

Instruction	Description
AND	Logical Product
OR	Logical Sum
XOR	Exclusive OR
NOT	Inversion (ones complement)

3.5 Bit Manipulation Instructions

Bit manipulation instructions perform logical operations on the two source operands--an immediate value and a register, an immediate value and a memory location, a register and a memory location--and update the PSW flags according to the result.

Instruction	Description
BTST	Bit test
BSET	Bit test and set (byte)
BCLR	Bit test and clear (byte)

3.6 Shift Instructions

Shift instructions shift or rotate the specified register by the specified (or implied) amount and update the PSW flags according to the result.

Instruction	Description
ASR*2	Arithmetic shift right
LSR*2	Logical shift right
ASL	Arithmetic shift left
ASL2	Arithmetic 2-bit shift left
ROR	Single-bit rotation right
ROL	Single-bit rotation left

^{*2.} The ASR Dn and LSR Dn variants are assembler shorthand for single-bit shifts of the specified register.

3.7 Branch Instructions

Branch instructions change the flow of execution. In addition to the usual conditional branch (Bcc) instruction, there is a separate variant (Lcc) for use in loops. The latter relies on special registers to reduce the penalty normally associated with taking the branch and thus speed up loop execution. The subroutine call and return instructions feature high-performance specifications that automatically take care of manipulating the Program Counter (PC), saving the appropriate registers to and restoring them from the stack, and securing and releasing the necessary stack space.

Instruction	Description
Bcc	Conditional branch (relative to PC)
Lcc	Loop conditional branch (relative to PC)
SETLB	Loop setup
JMP	Unconditional branch (relative to PC or register indirect)
CALL	Subroutine call (high-performance variant)
CALLS	Subroutine call
RET	Return from subroutine (high-performance variant)
RETF	Return from subroutine (high-performance, high-speed variant)
RETS	Return from subroutine
JSR*3	Subroutine call
RTS*3	Return from subroutine
RTI	Return from interrupt handler
TRAP	Subroutine call to predetermined address

^{*3.} JSR is assembler shorthand for an instruction sequence.

3.8 NOP Instruction

The NOP instruction does nothing but consume one cycle. It does not affect any resources.

Instruction	Description
NOP	No Operation

3.9 User Defined Instructions

User defined instructions access add-on expansion units. They have a fixed format and reserved positions in the instruction mapping. For further details, refer to the documentation for the particular device.

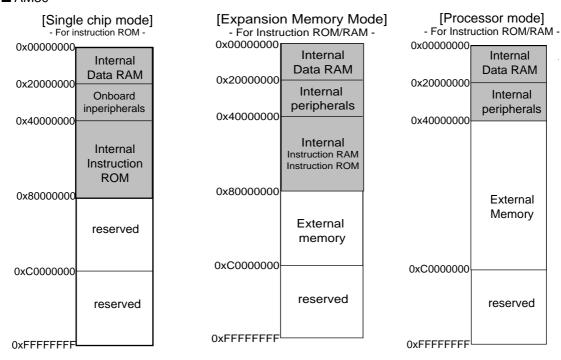
Instruction	Description
UDFnn	User defined instruction (with sign extension)
UDFUnn	User defined instruction (with zero extension)

^{*4.} RTS is an alias for RETS.

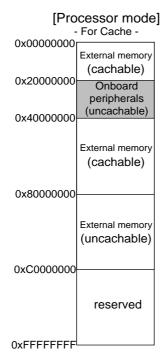


The MN1030/MN103S Series of 32-bit microcontrollers has a 4-gigabyte linear address space. Memory assignments within this address space follow the patterns below. Note how the memory map varies with such factors as internal memory configuration and memory mode. One assignment that is common throughout, however, is the location of the reset vector. It is always at 0x40000000.

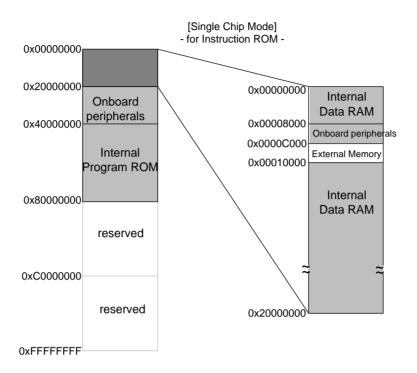
■ AM30



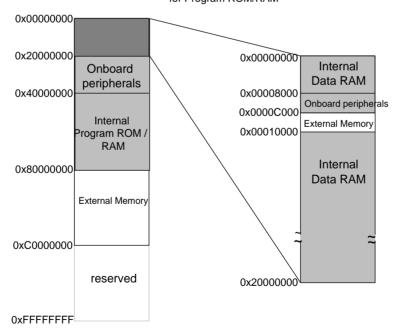
■ AM31



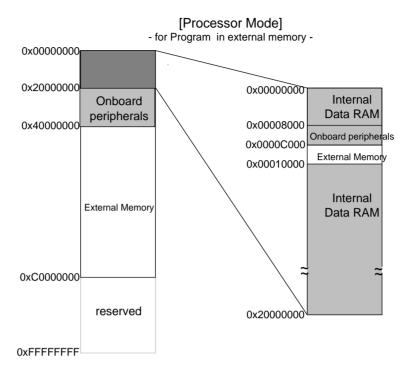
■ AM32

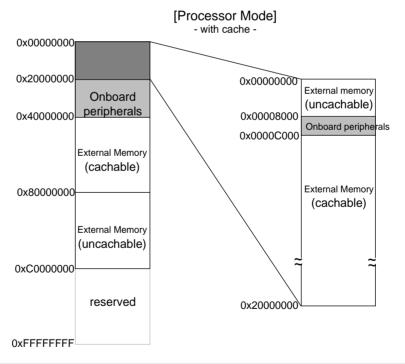


[Extended Memory Mode] - for Program ROM/RAM -



■ AM32













Memory layout varies with such factors as model and pin specifications. For further details, refer to the documentation for the particular device.

5

Addressing Modes

The addressing modes available consist of the following six most heavily used by C compilers.

- 1. Register direct
- 2. Immediate value
- 3. Register indirect
- 4. Register relative indirect
- 5. Absolute
- 6. Register indirect with indexing

Data transfer instructions offer all six addressing modes: register direct, immediate, register indirect, register relative indirect, absolute, and register indirect with indexing.

Register arithmetic instructions offer only two addressing modes: register direct and absolute.

Register indirect addressing with indexing is for more efficient access to arrays and the like.

■ Addressing Modes

Addressing Mode		Address Calculation	Final Address
Register direct	Dm/Dn		
	Am/An		
	SP/PSW/MDR		
Immediate	imm8/regs		
	imm16		
	imm24		
	imm32		
	imm40		
	imm48		
Register indirect	(am)/(An)	31 0	31 0
		Am/An	(32-bit address)
Register relative indirect	(d8,Am)/(d8,An)	31 0	31 0
	:d8 sigm-extended	Am/An	(32-bit address)
	(d16,Am)/(d16,An)	+	
	:d16 sign-extended	31 15 7 0	
	(d32,Am)/(d32,An)	d32/d6/d8	
	(Branch instructions	31 0	31 0
	only)	PC	(32-bit address)
	(d8,PC)	+	A
	:d8 sign-extended		
	(d16,PC)	31 15 7 0	
	:d16 sign-extended	d32/d16/d8	
	(d32,PC)		
	(d8,PC)	31 0	31 0
	:d8 zero-extended	SP	(32-bit address)
	(d16, SP)	+	
	:d16 zero-extended	31 15 7 0	
	(d32, SP)	d32/d16/d8	
Absolute	(abs16)	31 0	31 0
	:abs16 zero-extended	abs16/abs32	(32-bit address)
	(abs32)		
Register indirect with indexing		31 0	31 0
	(di,Am)/(Di,An)	Am/An	(32-bit address)
		+	<u> </u>
		31 0	
		Di	



The suffixes m, n, and i indicate the source, destination, and index registers, respectively. All three have the range 0 to 3.

5.1 Register Direct Addressing

Register direct addressing specifies an operand as the name of a register from the following list.

Dn/Dm (32-bit) Data register An/Am (32-bit) Address register

SP (32-bit) Stack Pointer

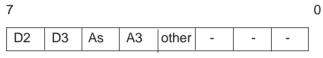
PSW (16-bit) Processor Status Word MDR (32-bit) Multiply/Divide Register

5.2 Immediate Addressing

Immediate addressing specifies an operand as a value incorporated as is into the instruction code. Examples include numbers for loading into registers, masks, and multiregister specifications (regs) for transfers to and from the stack.

These operands are abbreviated to imm8, imm16, imm24, imm32, imm40, and imm48, where the numeric suffix indicates the size in bits.

The abbreviation regs denotes an 8-bit immediate value containing five bits specifying the registers D2, D3, A2, and A3 individually and seven other registers as a group.



other : D0, D1, A0, A1, MDR, LIR, and LAR

- : Reserved (always set to 0)

5.3 Register Indirect Addressing

Register indirect addressing, (An) or (Am), specifies an address operand as the contents of a 32-bit address register.

Operand format: (An) or (Am)



32-bit value used as memory address

5.4 Register Relative Indirect Addressing

Register relative indirect addressing specifies an address operand as the sum of a displacement and a base address in an address register (An or Am), the Program Counter (PC), or Stack Pointer (SP). Displacements can be 8, 16, or 32 bits wide.

Short (8- or 16-bit) displacements are zero-extended for the base register Stack Pointer (SP) and sign-extended for the others (An, Am, and PC).

Operand formats:	(d8, An) or (d8, Am)	:d8 sign-extended
	(d16, An) or (d16, Am)	:d16 sign-extended
	(d32, An) or (d32, Am)	:
	(d8, PC)	:d8 sign-extended
	(d16, PC)	:d16 sign-extended
	(d32, PC)	:
	(d8, SP)	:d8 zero-extended
	(d16, SP)	:d16 zero-extended
	(d32, SP)	:
	An/Am	
	+	
	d8/d16/d32	
	PC	
	+	
	d8/d16/d32	
	SP	
	+	
	d8/d16/d32	
31		0

32-bit value used as memory address





The result of adding the displacement to the base register An, Am, or PC must be in the same memory address space as the address in the base register.



Any overflow arising during addition to the Program Counter (PC) is ignored. The effective address is the lowest 32 bits of the result.

5.5 Absolute Addressing

Absolute addressing specifies an address operand as a 16- or 32-bit value incorporated as is into the instruction code.

A 16-bit operand is zero-extended to 32 bits.

Operand formats: (abs16) :16-bit absolute address

(abs32) :32-bit absolute address

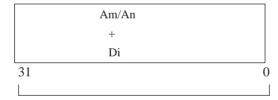
	zero extension		abs16	
31	16	15		0
	ab	s32		
31				0
	22.1%	,	1.1	

32-bit value used as memory address

5.6 Register Indirect Addressing with Indexing

Register indirect addressing with indexing specifies an address operand as the sum of a base address in an address register (An or Am) and an index in a data register (Di).

Operand format: (Di, An) or (Di, Am)



32-bit value used as memory address



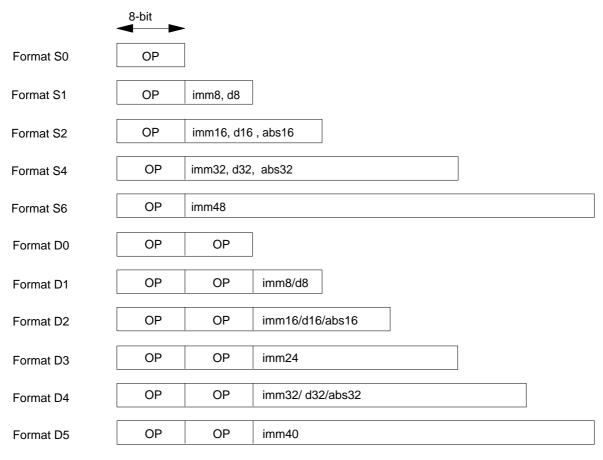


The result of adding the index to the base register (An or Am) must be in the same memory address space as the address in the base register.

6

Instruction Formats

There are 11 instruction formats.



The normal pattern consists of one or two opcode bytes following by an immediate value, displacement, or absolute value that is 8, 16, or 32 bits long. Formats S2, S4, S6, D2, D3, and D5, however, can have two or more such operands. For simplicity, the above diagram combines them under the immediate value labels imm16, imm24, imm32, imm40, and imm48, where the numeric suffix indicates the size in bits.

The following are the instructions affected.

imm16:	RET	regs, imm8	imm32:	CALL	(D16, PC), regs, imm8
	RETF	regs, imm8	imm40:	BTST	imm8, (abs32)
	BTST	imm8, (d8, An)		BSET	imm8, (abs32)
	BSET	imm8, (d8, An)		BCLR	imm8, (abs32)
	BCLR	imm8, (d8, An)	imm48:	CALL	(d32, PC), regs, imm8
imm24	BTST	imm8, (abs16)			
	BSET	imm8, (abs16)			
	BCLR	imm8, (abs16)			



The assembler does not normally specify the two operands regs and imm8 for the RET, RETF, and CALL instructions directly. It uses an indirect approach, specifying them, at the subroutine entry point, in a directive subsequently resolved by the linker. For further details, refer to the Cross Assembler User's Manual.

6.1 Data Formats

Processing uses four data types: bit, byte, half-word, and word. The last three can be either signed or unsigned. The sign bit is the most significant one (MSB) for the data size.

Alignment restrictions apply. The storage address for a word data item must have '00' in its lowest two bits-that is, must be a multiple of four. Similarly, that for a half-word data item must have '0' in its lowest bit-that is, must be a multiple of two.

(1) Bit data	
(2) Byte data	
unsigned 8-bit data	
signed 8-bit data (sign in bit 7)	(signed:MSB)
(3) Half-word data	
unsigned 16-bit data	
signed 16-bit data (sign in bit 15)	(signed: MSB)
(4) Word data	
unsigned 32-bit data	
signed 32-bit data (sign in bit 31)	(signed: MSB)

	MSB			LS
Bit number	31 24	23 16	15 8	7 (
address in memory	(4n+ 3)	(4n+ 2)	(4n+ 1)	4n
	Upper ha	alf word	Lower ha	alf word
word data	highest byte			lowest byte
half word data			highest byte	lowest byte
byte data				

6.2 Byte Order

Byte order is little endian: the bytes making up a 16- or 32-bit immediate value, displacement, or absolute value (imm16, d16, abs16, imm32, d32, or abs32) are stored from least significant byte to most as addresses increase.

[Example]

Little endian order stores the four bytes in the 32-bit immediate value 0x01234567 in the following order.

Address n	0x67
Address n+1	0x45
Address n+2	0x23
Address n+3	0x01

The formats with two or more operands (S2, S4, S6, D2, D3, and D5) maintain little endian order for their 16- or 32-bit immediate values and displacements (d16, abs16, d32, and abs32), but the order of the operands making up the fields abbreviated to imm16, imm24, imm32, imm40, and imm48 varies with the instruction.

RET/RETF regs, imm8

Address n	RET/RETF	RET or RETF Opecode
Address n+1	regs	
Address n+2	imm8	

BTST/BSET/BCLR imm8, (d8, An)

, (5.5)	,	
Address n	BTST/BSET/BCLR	BTST, BSET or BCLR Opecode
Address n+1		
Address n+2	d8	
Address n+3	imm8	

BTST/BSET/BCLR imm8, (abs16)

, (-/	
Address n	BTST/BSET/BCLR	BTST, BSET or BCLR Opecode
Address n+1		
Address n+2	abs16	
Address n+3		
Address n+4	imm8	

CALL (d16, PC), regs, imm8

Address n	CALL	C
Address n+1	d16	
Address n+2		
Address n+3	regs	
Address n+4	imm8	

.....CALL Opecode

BTST/BSET/BCLR imm8, (abs32)

Address n	BTST/BSET/BCLR
Address n+1	
Address n+2	abs32
Address n+3	
Address n+4	
Address n+5	
Address n+6	imm8

.....BTST, BSET or BCLR Opecode

CALL (d32, PC), regs, imm8

Address n	CALL
Address n+1	d32
Address n+2	
Address n+3	
Address n+4	
Address n+5	regs
Address n+6	imm8

.....CALL Opecode

Symbol Definitions

■ Following is the list of symbols used in the instruction specifications.

Reg :register (used for general meaning)
Am, An :address register (m, n=3 to 0)
Dm, Dn,Di :data register (m, n, i=3 to 0)
MDR :multiply/divide register
PSW :processor status word
PC :program counter
SP :stack pointer

LIR :loop instruction register
LAR :loop address register

{MDR,Dn} :64-bit data defined whose upper 32-bit in MDR and lower 32-bit in register Dn within a "{ }".

Mem :memory (used for general meaning)

imm :immediate value (used for general meaning)

imm8 :8-bit immediate value imm16 :16-bit immediate value imm32 :32-bit immediate value d8 :8-bit displacement d16 :16-bit displacement d32 :32-bit displacement abs16 :16-bit absolute abs32 :32-bit absolute :indirect addressing ()

Refer to "Chapter 1 section 5, Addressing Mode" for details.

regs :multiple registers specification

0x . . . : hexadecimal notation(the numbers following 0x are expressed in hexadecimal notation.)

.bpn :bit location ("n" means location of bit; 0 to 31)

.lsb :bit location (bit 0)
.msb :bit location (bit 31)

& :logical AND
| :logical OR

^ :exclusive OR

~ :bit inverted

<<n :n-bit shift left

>>n :n-bit shift right

:move

: :reflection of operation result

(sign_ext):sign-extend(zero_ext):zero-extendlabel:address

VF :overflow flag
CF :carry flag
NF :nagative flag
ZF :zero flag

temp :temporary register

 $\begin{array}{ll} mem8(xxx) & :8\mbox{-bit data in memory specified with } xxx \\ mem16(xxx) & :16\mbox{-bit data in memory specified with } xxx \\ mem32(xxx) & :32\mbox{-bit data in memory specified with } xxx \end{array}$

CodeSize :code size of assembler mnemonic

■ Following is the list of symbols used in flag changes.

("flag" is a general term of lower 4-bit(V, C, N, Z) of PSW.

:flag changes
:no flag change
:flag is always "0"
:flag is always "0"
:flag change undefined
:change by user defined

■ "Cycles" will be changed by status of pipeline or memory space to access.

"Cycles" written in this chapter are calculated on the following conditions;

- (1) No pipleline installation
- (2) Instruction fetch: 2 cycles, data load/store: 1 cycle

(ROM/RAM/flash build-in products:

Instructions: accessing internal instruction ROM space or internal instruction RAM space Data: accessomg internal data RAM space

Cache build-in products:

Instructions/data: when accessing cachable area, cache is always hit.

Refer to "Chapter 3, Using Instructions" for influence by pipleline installation, LSI Manual of each product for cycle changes in memory space.

■ Symbols for Notation

Each microcomputer core has different notations. Therefore, each notation is written with each microcomputer core mark in this manual. The microcomputer core marks are as followings;



Notice for AM30 core



Notice for AM31 core



Notice for AM32 core

Move

MOV Reg1,Reg2

Operation

Reg1→Reg2

Moves the contents of the register(Reg1) to the register(Reg2). Not

moves to the same register.

Assembler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycles
mov Dm,Dn	Dm=Dn cannot be specified	_	_	_	_	1	1
mov Dm,An		_	_	_	_	2	1
mov Am,Dn		-	_	_	_	2	1
mov Am,An	Am=An cannot be specified	_	_	_	_	1	1
mov SP,An		-	_	_	_	1	1
mov Am,SP		_	_	_	_	2	1
mov PSW,Dn	Zero-extends the upper 16 bits	_	_	_	_	2	1
mov Dm,PSW	Omits the upper 16 bits	•	•	•	•	2	1
mov MDR,Dn		_	_	_	_	2	1
mov Dm,MDR		_	_	_	_	2	1

Flag Changes

Other than mov Dm,PSW

VF: No Changes.

CF: No Changes.

NF: No Changes.

ZF: No Changes.

mov Dm,PSW

VF: Reflects the 3rd bit of Dm.

CF: Reflects the 2nd bit of Dm.

NF: Reflects the 1st bit of Dm.

ZF: Reflects the zero bit of Dm.



PSW-update by mov Dm,PSW instruction can be delayed for two instructions at most.

Especially at interrupting affected by IE bit or IM field, note that the instruction during updating will be executed in the status before/after updating.

MOV Mem,Reg

Operation

Mem→Reg

Word-data-moves the contents of the memory(Mem) to the register(Reg).

Assembler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycles
mov (Am),Dn		_	_	_	_	1	1
mov (d8,Am),Dn	d8 is sign-extended	_	_	_	_	3	1
mov (d16,Am),Dn	d16 is sign-extended	_	_	_		4	1
mov (d32,Am),Dn		_	_		_	6	2
mov (d8,SP),Dn	d8 is zero-extended	_	_		_	2	1
mov (d16,SP),Dn	d16 is zero-extended	_	_	_	_	4	1
mov (d32,SP),Dn		_	_		_	6	2
mov (Di,Am),Dn		_	_			2	1
mov (abs16),Dn	abs16 is zero-extended	_	_	_	_	3	1
mov (abs32),Dn		_	_		_	6	2
mov (Am),An		_	_	_		2	1
mov (d8,Am),An	d8 is sign-exended	_	_	_	_	3	1
mov (d16,Am),An	d16 is sign-extended	_	_	_	_	4	1
mov (d32,Am),An		_	_	_		6	2
mov (d8,SP),An	d8 is zero-extended	_	_	_	_	2	1
mov (d16,SP),An	d16 is zero-extended	-	_	_		4	1
mov (d32,SP),An		-	_	_	_	6	2
mov (Di,Am),An						2	1
mov (abs16),An	abs16 is zero-extended	_	_	_	_	4	1
mov (abs32),An		_	_	_	_	6	2
mov (d8,Am),SP	d8 is sign-extended	_	_	_	_	3	1
	Flor Changes						

Flag Changes

VF: No Changes. CF: No Changes. NF: No Changes. ZF: No Changes.



The operation of the memory(Mem) address other than multiple of four is not guaranteed.



In register-relative indirect addressing mode or index decoration register indirect addressing mode, when the address specified by based register(Am,SP) and the address derived from address caluculation are not in the same memory space, one cycle will be added.





In register-relative indirect addressing mode or index decoration register indirect addressing mode, the address specified by based register(Am,SP) and the address derived from address calculation must be in the same memory space.

MOV Reg,Mem

Operation

Reg1→Mem

Word-data-moves the contents of the memory(Mem) to the register(Reg).

Assembler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycles
mov Dm,(An)		_			_	1	1
mov Dm,(d8,An)	d8 is sign-extended	_	1			3	1
mov Dm,(d16,An)	d16 is sign-extended	_	1		_	4	1
mov Dm,(d32,An)		_			_	6	2
mov Dm,(d8,SP)	d8 is zero-extended	_		_	_	2	1
mov Dm,(d16,SP)	d16 is zero-extended	_	_	_	_	4	1
mov Dm,(d32,SP)		_			_	6	2
mov Dm,(Di,An)		_			_	2	2
mov Dm,(abs16)	abs16 is zero-extended	_		_		3	1
mov Dm,(abs32)		_		_		6	2
mov Am,(An)		_				2	1
mov Am,(d8,An)	d8 is sign-extended	_		_	_	3	1
mov Am,(d16,An)	d16 is sign-extended	_		_		4	1
mov Am,(d32,An)			l			6	2
mov Am,(d8,SP)	d8 is zero-extended	_		-	_	2	1
mov Am,(d16,SP)	d16 is zero-extended	_		_		4	1
mov Am,(d32,SP)		_		_		6	2
mov Am,(Di,An)		_		_	_	2	2
mov Am,(abs16)	abs16 is zero-extended					4	1
mov Am,(abs32)		_	_	_	_	6	2
mov SP,(d8,An)	d8 is sign-extended	_	_	_	_	3	1

Flag Changes

VF: No Changes. CF: No Changes. NF: No Changes. ZF: No Changes.



The operation of the memory(Mem) address other than multiple of four is not guaranteed.



In register-relative indirect addressing mode or index decoration register indirect addressing mode, when the address specified by based register(An,SP) and the address derived from address caluculation are not in the same memory space, one cycle will be added.





In register-relative indirect addressing mode or index decoration register indirect addressing mode, the address specified by based register(An,SP) and the address derived from address calculation must be in the same memory space.

MOV	imm,R	leg								
Operation imm→Reg Moves the contents of the immediate value(imm) to the register(Reg).										
Assembler	mnemonic	Notes	V	C	Ν	Ζ	Size	Cycles		
mov imm8,[On	imm8 issign-extended	_	_	_	_	2	1		
mov imm16	,Dn	imm16 is sign-extended	_	_	_	_	3	1		
mov imm32	,Dn		_	_	_	_	6	2		
mov imm8,	An	imm8 is zero-extended	_	_	_	_	2	1		
mov imm16	,An	imm16 is zero-extended	_	_	_	_	3	1		
mov imm32	,An		_		_	_	6	2		
		Flag Changes								
VF: No C	hanges.									
CF: No C	hanges.									
NF: No C	hanges.									
ZF: No C	hanges.									

MOVBU

Zero-extend Byte Move

MOVBU Mem,Reg

Operation

Mem→Reg

Byte-data-moves the contents of the memory(Mem) to the register(Reg)

(8 bits→32 bits; zero-extended)

Assembler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycles
movbu (Am),Dn		_	_	_	_	2	1
movbu (d8,Am),Dn	d8 is sign-extended	_	_	_	_	3	1
movbu (d16,Am),Dn	d16 is sign-extended	_	_	_	_	4	1
movbu (d32,Am),Dn		_	_	_	_	6	2
movbu (d8,SP),Dn	d8 is zero-extended	_	_	_	_	3	1
movbu (d16,SP),Dn	d16 zero-extended	_	_	_	_	4	1
movbu (d32,SP),Dn		_	_	_	_	6	2
movbu (Di,Am),Dn		_	_	_	_	2	1
movbu (abs16),Dn	abs16 zero-extended	_	_	_	_	3	1
movbu (abs32),Dn		_	_	_	_	6	2

Flag Changes

VF: No Changes. CF: No Changes. NF: No Changes.

ZF: No Changes.



In register-relative indirect addressing mode or index decoration register indirect addressing mode, when the address specified by based register(Am,SP) and the address derived from address caluculation are not in the same memory space, one cycle will be added.





In register-relative indirect addressing mode or index decoration register indirect addressing mode, the address specified by based register(Am,SP) and the address derived from address calculation must be in the same memory space.

MOVBU Reg,Mem

Operation

 $Reg \rightarrow Mem$

Byte-moves the contents of the register(Reg) to the memory(Mem).

(32 bits→8bits: Omit the upper)

Assembler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycles
movbu Dm,(An)		_	_	_	_	2	1
movbu Dm,(d8,An)	d8 is sign-extended	_	_	_	_	3	1
movbu Dm,(d16,An)	d16 is sign-extended	_	_	_	_	4	1
movbu Dm,(d32,An)		_	_	_	_	6	2
movbu Dm,(d8,SP)	d8 is zero-extended	_	_	_	_	3	1
movbu Dm,(d16,SP)	d16 zero-extended	_	_	_	_	4	1
movbu Dm,(d32,SP)		_	_	_	_	6	2
movbu Dm,(Di,An)		_	_	_	_	2	2
movbu Dm,(abs16)	abs16 zero-extended	_	_	_		3	1
movbu Dm,(abs32)		_	_	_	_	6	2

Flag Changes

VF: No Changes. CF: No Changes. NF: No Changes. ZF: No Changes.





In register-relative indirect addressing mode or index decoration register indirect addressing mode, when the address specified by based register(Am,SP) and the address derived from address caluculation are not in the same memory space, one cycle will be added.







In register-relative indirect addressing mode or index decoration register indirect addressing mode, the address specified by based register(An,SP) and the address derived from address calculation must be in the same memory space.



Sign-extend Byte Move

[combination of multiple instructions]

MOVB Mem,Reg

Operation

Mem→Reg

Byte-data-moves the contents of the memory(Mem) to the register(Reg).

(8 bits→32bits: sign-extended)

Assembler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycles
movb (Am),Dn		_	_	_	_	3	2
movb (d8,Am),Dn	d8 is sign-extended	_	_	_	_	4	2
movb (d16,Am),Dn	d16 is sign-extended	_	_	_	_	5	2
movb (d32,Am),Dn		_	_	_	_	7	3
movb (d8,SP),Dn	d8 is zero-extended	_	_	_	_	4	2
movb (d16,SP),Dn	d16 is zero-extended	_	_	_	_	5	2
movb (d32,SP),Dn		_	_	_	_	7	3
movb (Di,Am),Dn		_	_	_	_	3	2
movb (abs16),Dn	abs16 is zero-extended	_	_	_	_	4	2
movb (abs32),Dn		_	_	_	_	7	3

Flag Changes

VF: No Changes. CF: No Changes. NF: No Changes. ZF: No Changes.



This instruction is executed with the combination of multiple instructions and the assembler generates the following instructions.

MOVBU Mem, Reg **EXTB** Reg



The numbers of Size and Cycles contain those of the multiple instructions mentioned above. For the optimaization of assembler, the location within the multiple instructions may change the number of Cycles. Refer to "Chapter 3" Using the Instructions" for details.



In register-relative indirect addressing mode or index decoration register indirect addressing mode, when the address specified by based register(Am,SP) and the address derived from address caluculation are not in the same memory space, one cycle will be added.





In register-relative indirect addressing mode or index decoration register indirect addressing mode, the address specified by based register(Am,SP) and the address derived from address calculation must be in the same memory space.

MOVB Reg, Mem

Operation

Reg→Mem

Byte-data-moves the contents of the register(Reg) to thememory(Mem) .

(32 bits→8bits: Omit the upper)

Assembler mnemonic	Notes	V	С	Ν	Ζ	Sign	Cycles
movb Dm,(An)		_	_	_	_	2	1
movb Dm,(d8,An)	d8 is sign-extended	_	_	_	_	3	1
movb Dm,(d16,An)	d16 is sign-extended	_	_	_	_	4	1
movb Dm,(d32,An)		_	_	_	_	6	2
movb Dm,(d8,SP)	d8 is zero-extended	_	_	_	_	3	1
movb Dm,(d16,SP)	d16 is zero-extended	_	_	_	_	4	1
movb Dm,(d32,SP)		_	_	_	_	6	2
movb Dm,(Di,An)		_	_	_	_	2	2
movb Dm,(abs16)	abs16 is zero-extended	_	_	_	_	3	1
movb Dm,(abs32)		_	_	_	_	6	2

Flag Changes

VF: No Changes. CF: No Changes. NF: No Changes. ZF: No Changes.



This instruction is executed by overwriting the instructions and the assembler generates the following instructions.

MOVBU Reg, Mem



The numbers of Size and Cycles are those of the instructions mentioned above.



In register-relative indirect addressing mode or index decoration register indirect addressing mode, when the address specified by based register(An,SP) and the address derived from address caluculation are not in the same memory space, one cycle will be added.





In register-relative indirect addressing mode or index decoration register indirect addressing mode, the address specified by based register(An,SP) and the address derived from address calculation must be in the same memory space.



MOVHU Mem,Reg

Operation

Mem→Reg

Half-word-moves the contents of the memory(Mem) to the register(Reg).

(16 bits→35bits: zero-extended)

Assembler mnemonic	Notes	V	С	N	Ζ	Size	Cycles
movhu (Am),Dn		_	_	_	_	2	1
movhu (d8,Am),Dn	d8 is sign-extended	_	_	_	_	3	1
movhu (d16,Am),Dn	d16 is sign-extended	_	_	_	_	4	1
movhu (d32,Am),Dn		_	_	_	_	6	2
movhu (d8,SP),Dn	d8 is zero-extended	_	_	_	_	3	1
movhu (d16,SP),Dn	d16 is zero-extended	_	_	_	_	4	1
movhu (d32,SP),Dn		-	_	-	_	6	2
movhu (Di,Am),Dn		_	_	_	_	2	1
movhu (abs16),Dn	abs16 is zero-extended	_	_	_	_	3	1
movhu (abs32),Dn		_	_	_	_	6	2

Flag Changes

VF: No Changes. CF: No Changes. NF: No Changes.

ZF: No Changes.



The operation of the memory(Mem) address other than multiple of two is not guaranteed.



In register-relative indirect addressing mode or index decoration register indirect addressing mode, when the address specified by based register(Am,SP) and the address derived from address caluculation are not in the same memory space, one cycle will be added.





In register-relative indirect addressing mode or index decoration register indirect addressing mode, the address specified by based register(Am,SP) and the address derived from address calculation must be in the same memory space.

MOVHU Reg,Mem

Operation

 $Reg \rightarrow Mem$

 $Half\text{-}word\text{-}moves \ the \ contents \ of \ the \ register(Reg) \ to \ the \ memory(Mem).$

(32 bits→16bits: Omit the upper)

Assembler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycles
movhu Dm,(An)		_	_	_	_	2	1
movhu Dm,(d8,An)	d8 is sign-extended	_	_	_	_	3	1
movhu Dm,(d16,An)	d16 is sign-extended	_	_	_	_	4	1
movhu Dm,(d32,An)		_	_	_	_	6	2
movhu Dm,(d8,SP)	d8 is zero-extended	_	_	_	_	3	1
movhu Dm,(d16,SP)	d16 is zero-extended	_	_	_	_	4	1
movhu Dm,(d32,SP)		_	_	_	_	6	2
movhu Dm,(Di,An)		_	_	_	_	2	2
movhu Dm,(abs16)	abs16 is zero-extended	_	_	_	_	3	1
movhu Dm,(abs32)		_	_	_	_	6	2

Flag Changes

VF: No Changes. CF: No Changes. NF: No Changes. ZF: No Changes.



The operation of the memory(Mem) address other than multiple of two is not guaranteed.



In register-relative indirect addressing mode or index decoration register indirect addressing mode, when the address specified by based register(An,SP) and the address derived from address caluculation are not in the same memory space, one cycle will be added.





In register-relative indirect addressing mode or index decoration register indirect addressing mode, the address specified by based register(An,SP) and the address derived from address calculation must be in the same memory space.



Sign-extend Half Word Move [Combination of Multiple Instructions]

Mem,Reg MOVH

Operation

Mem→Reg

Half-word-moves the contents of the memory(Mem) to the register(Reg).

(16 bits→32bits: sign-extended)

Assembler mnemonic	Notes	V	С	N	Ζ	Size	Cycles
movh (Am),Dn		_	_	_		3	2
movh (d8,Am),Dn	d8 is sign-extended	_	_	_	_	4	2
movh (d16,Am),Dn	d16 is sign-extended	_	_	_	_	5	2
movh (d32,Am),Dn		_	_	_	_	7	3
movh (d8,SP),Dn	d8 is zero-extended	_	_	_	_	4	2
movh (d16,SP),Dn	d16 is zero-extended	_	_	_	_	5	2
movh (d32,SP),Dn		_	_	_	_	7	3
movh (Di,Am),Dn		_	_	_	_	3	2
movh (abs16),Dn	abs16 is zero-extended	_	_	_	_	4	2
movh (abs32),Dn		_	_	_		7	3

Flag Changes

VF: No Changes. CF: No Changes. NF: No Changes. ZF: No Changes.



The operation of the memory(Mem) address other than multiple of two is not guaranteed.



This instruction is executed with the combination of multiple instructions and the assembler generates the following instructions.

MOVHU Mem, Reg **EXTH** Reg



The numbers of Size and Cycles contain those of the multiple instructions mentioned above. For the optimaization of assembler, the location within the multiple instructions may change the number of Cycles. Refer to "Chapter 3" Using the



In register-relative indirect addressing mode or index decoration register indirect addressing mode, when the address specified by based register(Am,SP) and the address derived from address caluculation are not in the same memory space, one cycle will be added.







In register-relative indirect addressing mode or index decoration register indirect addressing mode, the address specified by based register(Am,SP) and the address derived from address calculation must be in the same memory space.

MOVH Reg, Mem

Operation

Reg→Mem

 $Half\text{-}word\text{-}moves \ the \ contents \ of \ the \ register(Reg) \ to \ the \ memory(Mem).$

(32 bits→16 bits: Omit the upper)

Assembler mnemonic	Notes	٧	С	Ν	Ζ	Size	Cycles
movh Dm,(An)		_	_	_	_	2	1
movh Dm,(d8,An)	d8 is sign-extended	_	_	_	_	3	1
movh Dm,(d16,An)	d16 is sign-extended	_	_	_	_	4	1
movh Dm,(d32,An)		_	_	_	_	6	2
movh Dm,(d8,SP)	d8 is zero-extended	_	_	_	_	3	1
movh Dm,(d16,SP)	d16 is zero-extended	_	_		_	4	1
movh Dm,(d32,SP)		_	_	_	_	6	2
movh Dm,(Di,An)		_	_	_	_	2	2
movh Dm,(abs16)	abs16 is zero-extended	_	_			3	1
movh Dm,(abs32)		_	_	_	_	6	2

Flag Changes

VF: No Changes.

CF: No Changes.

NF: No Changes.

ZF: No Changes.



The operation of the memory(Mem) address other than multiple of two is not guaranteed.



This instruction is executed by overwriting the instructions and the assembler generates the following instructions.

MOVHU Reg, mem



The numbers of Size and Cycles are those of the instructions mentioned above.



In register-relative indirect addressing mode or index decoration register indirect addressing mode, when the address specified by based register(An,SP) and the address derived from address caluculation are not in the same memory space, one cycle will be added.





In register-relative indirect addressing mode or index decoration register indirect addressing mode, the address specified by based register(An,SP) and the address derived from address calculation must be in the same memory space.

Move Between Multiple Memory and Register

MOVM (SP),regs Operation "Other" in the specified register; No "other" in the specified register; If regs = [Reg1, Reg2], If regs = [Reg1,Reg2,Reg3,Reg4,Reg5] (regs-specified registers=2) (regs-specified registers=11) mem32(SP+4)→reg1, mem32(SP+44)→D2,mem32(SP+40)→D3, mem32(SP)→reg2,SP+8→SP mem32(SP+36)→A2,mem32(SP+32)→A3, mem32(SP+28)→D0,mem32(SP+24)→D1, Off-set from SP address lower returned order before execution $mem32(SP+20)\rightarrow A0, mem32(SP+16)\rightarrow A1,$ before execution-(2)+0reg2 mem32(SP+12)→MDR,mem32(SP+8)→LIR, +8 SP after executionmem32(SP+4)→LAR,SP+48→SP returned order address upper address lower before execution SP before execution→ 0 dummy area LAR (11)+4LIR (10)+8Block-moves from the memory MDR(9)+12specified with SP to the multiple AI(8)+16A0(7)+20registers. (6)+24The "regs" specifies the multiple D0+28(4) *A3* +32registers to move data and it can specify (3) +36 A2 $\overline{D3}$ (2) +40each D2,D3, A2, A3 and the other D2+44 registers(D0, D1, A0, A1, MDR, LIR, SP after execution→ +48address upper LAR).

Assembler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycles
movm (SP),[Reg1,,Regn]	Block-moves from memory to multiple registers	_		_	_	2	1
	(regs-specified registers= 0)						
	(regs-specified registers = 1)	_		_	_	2	2
	(regs-specified registers= 2)	_		_	_	2	3
	(regs-specified registers = 3)	_		_	_	2	4
	(regs-specified registers = 4)	_		_	_	2	5
	(regs-specified registers = 7)	_		_	_	2	8
	(regs-specified registers = 8)	_		_	_	2	9
	(regs-specified registers = 9)	_	_	_	_	2	10
	(regs-specified registers= 10)	_		_	_	2	11
	(regs-specified registers= 11)	_			_	2	12

Flag Changes

VF: No Changes. CF: No Changes. NF: No Changes. ZF: No Changes.

The register(any of D2,D3,A2, A3 or other) specified by the assembler is separatedwith comma(,) for each and parenthsize with ([]). However, you can not specify the same register twice or more.



No order to speify the registers by the assembler, however, the order of the registers to be returned is fixed as D2, D3, A2, A3, other(D0, D1, A0, A1, MDR, LIR, LAR). (Non-specified registers will be skipped.) If specifying "other", 4 byte of dummy area will be stored at last to simplify of move area calculation(4 byte x 8). (No moving operation.) If not specifying "other", no dummy area will be stored.



Refer to "Appendix : Instruction set" for operating expressions of each register specified with "regs".

MOVM egs,(SP)

Operation

No "other" in the specified register; If regs = [Reg1,Reg2],

(regs-specified registers=2) reg1→mem32(SP-4),

reg2→mem32(SP-8),SP-8→SP

address lower saved order SP after execution reg2 (2)(1) regi SP before execution address upper

to the memory specified with SP.

The "regs" specifies the multiple

each D2,D3, A2, A3 and the other

LAR).

"Other" in the specified register; If regs = [Reg1,Reg2,Reg3,Reg4,Reg5] (regs-specified registers=11)

 $D2\rightarrow mem32(SP-4), D3\rightarrow mem32(SP-8),$ A2→mem32(SP-12),A3→mem32(SP-16),

 $Off-set from SP D0 \rightarrow mem32(SP-20), D1 \rightarrow mem32(SP-24),$

A0→mem32(SP-28),A1→mem32(SP-32),

MDR→mem32(SP-36),LIR→mem32(SP-40), LAR→mem32(SP-44),SP-48→SP

Off-set from SP address lower saved order before execution SP after execution→ -48 dummy area (11)-44 LAR(10)-40 I.IRBlock-moves from the multiple registers (9) -36 MDR (8) -32 *A1* -28 -24 -20 (7)A0(6) D1(5) D0registers to move data and it can specify (4) -16 A3(3) -12 (2) (1) -8 -4 registers(D0, D1, A0, A1, MDR, LIR, D2SP before executionaddress upper

Assemble	r mnemonic	Notes	V	С	N	Z	Size	Cycles
movm [Reg	1,,Regn],(SP)	Block-move from multiple registers to memory	-	_	_	_	2	1
		(regs-specified register = 0)						
		(regs-specified register = 1)	_	_	_	_	2	1
		(regs-specified registers = 2)	_	_	_	_	2	2
		(regs-specified registers = 3)	_	_	_	_	2	3
		(regs-specified registers= 4)	_	_	_	_	2	4
		(regs-specified registers= 7)	_	_	_	_	2	8
		(regs-specified registers= 8)	_	_	_	_	2	9
		(regs-specified registers = 9)	_	_	_	_	2	10
		(regs-specified registers = 10)	_	_	_	_	2	11
		(regs-specified registers = 11)	_	_	_	_	2	12

before execution

-8

-4

0

Flag Changes

VF: No Changes. CF: No Changes. NF: No Changes. ZF: No Changes.



The register(any of D2,D3,A2, A3 or other) specified by the assembler is separated with comma(,) for each and parenthsize with ([]). However, you can not specifiy the same register twice or more.



No order to speify the registers by the assembler, however, the order of the registers to be returned is fixed as D2, D3, A2, A3, other(D0, D1, A0, A1, MDR, LIR, LAR). (Non-specified registers will be skipped.) If specifying "other", 4 byte of dummy area will be stored at last to simplify of move area calculation (4 byte x 8). (No moving operation.) If not specifying "other", no dummy area will be stored.



Refer to "Appendix : Instruction set" for operating expressions of each specified register.



Sign-extend Word Data to 64 Bits

EXT	Dn									
Operation	If Dn.bp31= 0, 0x00000000→MDR If Dn.bp31= 1, 0xFFFFFFF→MDR Sign-extends the value of register Dn to 64 bits and moves the extended upper 32 bits to MDR. No changes for the contents of Dn register.									
Assembler	mnemonic	Notes V C N	I Z	Size	Cycles					
ext Dn		-	- -	_ 2	1					
		Flag Changes								
VF: No	changes									
CF: No	changes									
NF: No	NF: No changes									
ZF: No	changes									



Sign-extend Byte Data to 32 Bits

EXTB	Dn											
Operation	If Dn.bp	If Dn.bp7 = 0,Dn & 0x000000FF→Dn If Dn.bp7 = 1,Dn 0xFFFFFF00→Dn Sign-extends the lower 8 bits of register Dn to 32 bits and stores in register Dn.										
Assembler	mnemonic	Notes V C	N	Ζ	Size	Cycles						
extb Dn		- -	_	-	1	1						
		Flag Changes										
VF: No	changes											
CF: No	CF: No changes											
NF: No	NF: No changes											
ZF: No	changes											



Zero-extend Byte Data to 32 Bits

EXTBU Dn										
Operation	Dn & 0x	000000FF→Dn								
	Zero-extends the lower 8 bits of regiser Dn to 32 bits and stores in register Dn.									
Assembler	mnemonic	Notes V C N	1 Z	Size	Cycles					
extbu Dn		- - -	- -	- 1	1					
		Flag Changes		•						
VF: Noo	changes									
CF: No	changes									
NF: No	NF: No changes									
ZF: No	changes									



EXTH	l Dn										
Operation	If Dn.bp15 = 0, Dn & 0x0000FFFF→Dn If Dn.bp15 = 1, Dn 0xFFFF0000→Dn Sign-extends the lower 16 bits of register Dn to 32 bits and stores in register Dn.										
Assembler exth Dn	mnemonic	Notes V C I	N Z	Size 1	Cycles 1						
		Flag Changes									
VF: No	ochanges	5 5									
CF: No	changes										
NF: No	changes										
ZF: No	changes										



Zero-extend Half Word Data to 32 Bits

EXTHU Dn											
Operation	Dn&0x0	000FFFF→Dn									
	Zero-extends the lower 16 bits of register Dn to 32 bits and stores in register Dn.										
Assembler	mnemonic	Notes V C	N	Ζ	Size	Cycles					
exthu Dn			_	_	1	1					
		Flag Changes									
VF: No	changes										
CF: No	CF: No changes										
NF: No	NF: No changes										
ZF: No	changes										



Data Clear

CLR	Dn						
Operation	0→Dn						
	Clears th	e contents of register Dn.					
Assembler	mnemonic	Notes V C	N	Ζ	Size	Cycles	
clr Dn		• •	•	1	1	1	
		Flag Changes					
VF: Al	ways 0						
CF: Al	ways 0						
NF: Al	NF: Always 0						
ZF: Alv	ways 1						

One instruction is delayed when updating PSW by flag-change. However, Bcc and Lcc instructions can evaluate the flag before affecting the flag.



Addition

ADD Reg1,Reg2

Operation

Reg1+Reg2→Reg2

Adds the contents of the register(Reg1) and the register(Reg2) and stores the results in the register(Reg2).

Assembler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycles
add Dm,Dn		•	•	•	•	1	1
add Dm,An		•	•	•	•	2	1
add Am,Dn		•	•	•	•	2	1
add Am,An		•	•	•	•	2	1

Flag Changes

VF: 1 if an overflow is generated as 32 bits signed-numeric value; 0 otherwise.

CF: 1 if a carry is generated from bit 31; 0 otherwise.

NF: 1 if the bit 31 of the result is '1'; 0 otherwise.

ZF: 1 if the result is '0'; 0 othewise.



Updating of PSW due to flag changes is delayed for one insturuction.

ADD imm,Reg

Operation

imm+Reg→Reg

Adds the immediate value(imm) and the contents of the register(Reg) and stores the result in the register(Reg).

Assembler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycles
add imm8,Dn	imm8 is sign-extended.	•	•	•	•	2	1
add imm16,Dn	imm16 is sign-extended.	•	•	•	•	4	1
add imm32,Dn		•	•	•	•	6	2
add imm8,An	imm8 is sign-extended.	•	•	•	•	2	1
add imm16,An	imm16 is sign-extended.	•	•	•	•	4	1
add imm32,An		•	•	•	•	6	2
add imm8,SP	imm8 is sign-extended.	_	_	_	_	3	1
add imm16,SP	imm16 is sign-extended.	_	_	_	_	4	1
add imm32,SP			_	_	_	6	2

Flag Change

Other than add imm,SP

VF: 1 if an overflow is generated as 32 bits signed numeric value; 0 otherwise.

CF: 1 if a carry is generated from bit 31; 0 otherwise.

NF: 1 if the bit 31 of the result is '1'; 0 otherwise.

ZF: 1 if the result is '0'; 0 othewise.

add imm,SP

VF: No Changes.

CF: No Changes.

NF: No Changes.

ZF: No Changes.





ADDC Dm,Dn											
Operation	Dm+Dn-	-CF→Dn									
	Adds the contents of register Dm including C flag and register Dn and stores the result in the register Dn.										
Assembler	mnemonic	Notes V	С	N	Ζ	Size	Cycles				
addc Dm,Di	n	•	•	•	•	2	1				
		Flag Changes									
VF: 1 if	an overflow is g	enerated as 32 bits signed-numeric value; 0 otherwise.									
CF: 1 if	a carry is genera	ted from bit 31; 0 otherwise.									
NF: 1 if	the bit 31 of the	result is '1'; 0 otherwise.									
ZF: 1 if	the result is '0';	othewise.									





Subtraction

SUB Reg1,Reg2												
Operation	Reg2-Reg	eg1→Reg2										
	Subtract register(s the contents of the register(Reg1) from the register(Reg2).	eg2)	and	l sto	res	the resu	ılt in the				
Assembler i	mnemonic	Notes	V	С	N	Ζ	Size	Cycles				
sub Dm,Dn			•	•	•	•	2	1				
sub Dm,An			•	•	•	•	2	1				
sub Am,Dn			•	•	•	•	2	1				
sub Am,An			•	•	•	•	2	1				

Flag Changes

VF: 1 if an overflow is generated as 32 bits signed-numeric value; 0 otherwise.

CF: 1 if a carry is generated from bit 31; 0 otherwise.

NF: 1 if the bit 31 of the result is '1'; 0 otherwise.

ZF: 1 if the result is '0'; 0 othewise.



SUB imm,Reg

Operation

Reg- imm→Reg

Subtracts the immediate value from the register(Reg) and stores the result in the register(Reg).

Assembler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycles
sub imm32,Dn		•	•	•	•	6	2
sub imm32,An		•	•	•	•	6	2

Flag Changes

VF: 1 if an overflow is generated as 32 bits signed-numeric value; 0 otherwise.

CF: 1 if a carry is generated from bit 31; 0 otherwise.

NF: 1 if the bit 31 of the result is '1'; 0 otherwise.

ZF: 1 if the result is '0'; 0 othewise.



Updating of PSW due to flag changes is delayed for one insturuction.

However, Bcc and Lcc instructions can evaluate flag before reflecting to PSW.



Using add imm, Reg instruction may shrink the instruction code size.



Subtraction With Borrow

SUBC Dm,Dn											
Operation	Dn-Dm-	Dn-Dm-CF→Dn									
	Subtracts the contents of register Dm including C flag from register Dn and stores the result in register										
	Dn.										
Assembler	mnemonic	Notes V (CN	Z	Size	Cycles					
subc Dm,Di	n										
Flag Changes											
		Flag Changes									
VF: 1 if	an overflow is g	Flag Changes enerated as 32 bits signed-numeric value; 0 otherwise.									
1					I						
CF: 1 if a	a carry is genera	enerated as 32 bits signed-numeric value; 0 otherwise.									



Updating of PSW due to flag changes is delayed for one insturuction.



MUL Dm,Dn

Operation

 $(Dn*Dm)\rightarrow \{MDR, Dn\}$

Multiplicants the contents of register Dm(signed 32 bits integer: non-multiplier) and register Dn(signed 32 bits integer:multiplier) and stores the upper 32 bits of the result(64 bits) in MDR and the lower 32 bits in register Dn.

The siginificant number of bytes from the LSB of the multiplier loaded to Dn before the operation is judged, and the operation is only performed for the range(byte unit) containing these significant values. In other words, the smaller the contents loaded to Dn, the faster operation results can be obtained.

Assembler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycles
mul Dm,Dn	Dn = 0	?	?	•	•	2	3
	Value Dn can specify by 1-byte.	?	?	•	•	2	13
	Value Dn can specify by 2-byte.	?	?	•	•	2	21
	Value Dn can specify by 3-byte.	?	?	•	•	2	29
	Value Dn can specify by 4-byte.	?	?	•	•	2	34

Flag Changes

VF: Undefined.

CF: Undefined.

NF: 1 if the bit 31 of the result of the lower 32-bit is '1'; 0 otherwise.

ZF: 1 if the lower 32-bit of the result is '0'; 0 otherwise.



Updating of PSW due to flag changes is delayed for one insturuction.

However, Bcc and Lcc instructions can evaluate flag before reflecting to PSW.





Locating writing-instrucion to address register A0 at one or two instructions preceding this instrucion is prohibited. Refer to "Chapter 3, 2. 5 Notes for location of MUL/MULU instruction following A0 writinginstruction" for details. Some examples of prohibition are follows.

(Example1) A writing-instruction to register A0 is located in one instruction preceding this instruction

0x80040900, a0

mul d1, d0

(Example2) A writing-instruction to register A0 is located in two instructions preceding this instruction.

mov 0x80040900, a0

0x0c, d0 mov

d1, d0 mul

(Example2) An instruction to write in register A0 is located in one instruction preceding this instruction.

inc

d1, d0 mul



MULU Dm,Dn

Operation

 $(Dn*Dm)\rightarrow \{MDR, Dn\}$

Multiplicants the contents of register Dm(unsigned 32 bits integer: non-multiplier) and register Dn(unsigned 32 bits integer:multiplier) and stores the upper 32 bits of the result(64 bits) in MDR and the lower 32 bits in register Dn.

The significant number of bytes from the LSB of the multiplier loaded to Dn before the operation is judged, and the operation is only performed for the range(byte unit) containing these significant values. In other words, the smaller the contents loaded to Dn, the faster operation results can be obtained.

Assembler mnemonic	Notes	V	С	N	Ζ	Size	Cycles
mulu Dm,Dn	Dn = 0	?	?	•	•	2	3
	Value Dn can specify by 1-byte.	?	?	•	•	2	13
	Value Dn can specify by 2-byte.	?	?	•	•	2	21
	Value Dn can specify by 3-byte.	?	?	•	•	2	29
	Value Dn can specify by 4-byte.	?	?	•	•	2	34

Flag Changes

VF: Undefined. CF: Undefined.

NF: 1 if the bit 31 of the result of the lower 32-bit is '1'; 0 otherwise.

ZF: 1 if the lower 32-bit of the result is '0'; 0 otherwise.



Updating of PSW due to flag changes is delayed for one insturuction.

However, Bcc and Lcc instructions can evaluate flag before reflecting to PSW.





Locating writing-instruction to address register A0 at one or two instructions preceding this instruction is prohibited. Refer to "Chapter 3, 2. 5 Notes for location of MUL/MULU instruction following A0 writing-instruction" for details. Some examples of prohibition are follows.

(Example1) A writing-instruction to register A0 is located in one instruction preceding this instruction

mov 0x80040900, a0

mulu d1, d0

(Example2) A writing-instruction to register A0 is located in two instructions preceding this instruction.

mov 0x80040900, a0 mov 0x0c, d0

mulu d1, d0

(Example2) An instruction to write in register A0 is located in one instruction preceding this instruction.

inc a0 mulu d1, d0



Division with signed

DIV Dm, Dn

Operation

 $((MDR << 32)\&0xFFFFFFF00000000+Dn\)/Dm \rightarrow Dn \\ ((MDR << 32)\&0xFFFFFFFF00000000+Dn\)\%Dm \rightarrow MDR$

Divides signed 64-bit interger combined with MDR(undivided upper 32-bit) and Dn register(undevided lower 32-bit) by the contents of register Dm(division of signed 32-bit interger) stores the remainder(32-bit) in MDR and the quotient(32-bit) in register Dn. If the quotient can not be specified as signed 32-bit value, V flag will be '1' and MDR and register Dn will be undefined. When zero-division is performed(divisor=0), V flag will be '1'.

The significant number of bytes from the LSB of the 64-bit dividend obtained by linking MDR and Dn before the operation is judged (none that MDR is judged in word units), and the operation is only performed for the range containing these significant values. In other words, the smaller the dividend obtained by linking MDR and Dn, the faster operation results can be obtained.

Assembler mnemonic		Notes	V	С	N	Ζ	Size	Cycles
div Dm,Dn	Nomal performance	{MDR,Dn}=0	0	?	•	•	2	4
	(The operation	Value (MDR,Dn) can specify by 1-byte.	0	?	•	•	2	14
	performed normally)	Value (MDR,Dn) can specify by 2-byte.	0	?	•	•	2	22
		Value (MDR,Dn) can specify by 3-byte.	0	?	•	•	2	30
		Value {MDR,Dn} can specify by 4-byte or more.	0	?	•	•	2	38
	Divisor not specified	{MDR,Dn}=0	1	?	?	?	2	4
	as signed value	Value (MDR,Dn) can specify by 1-byte.	1	?	?	?	2	14
	or	Value (MDR,Dn) can specify by 2-byte.	1	?	?	?	2	22
	zero-division	Value (MDR,Dn) can specify by 3-byte.	1	?	?	?	2	30
		Value {MDR,Dn} can specify by 4-byte or more.	1	?	?	?	2	38

Flag Changes

Nomal performance(the operation performed normally)

VF: Always 0. CF: Undefined.

NF: 1 if MSB of the divisor(32-bit) is '1'. 0 otherwise.

ZF: 1 if the divisor(32-bit) is '0'. 0 otherwise.

Divisor not specified as signed value or zero-division performed.

VF: Always 1. CF: Undefined. NF: Undefined. ZF: Undefined.



After the operation, if V flag is '1', the other flog will be undefined. Also the divisor and the remainder will be undefined.





Division without signed

DIVU Dm,Dn

Operation

 $\label{eq:control_control_control} $$((MDR<<32)\&0xFFFFFFF00000000+Dn\)\Dm\to Dn $$((MDR<<32)\&0xFFFFFFF00000000+Dn\)\Dm\to MDR $$$

Divides signed 64-bit interger combined with MDR(undivided upper 32-bit) and Dn register(undevided lower 32-bit) by the contents of register Dm(division of signed 32-bit interger) stores the remainder(32-bit) in MDR and the quotient(32-bit) in register Dn. If the quotient can not be specified as signed 32-bit value, V flag will be '1' and MDR and register Dn will be undefined. When zero-division is performed(divisor=0), V flag will be '1'.

The significant number of bytes from the LSB of the 64-bit dividend obtained by linking MDR and Dn before the operation is judged (none that MDR is judged in word units), and the operation is only performed for the range containing these significant values. In other words, the smaller the dividend obtained by linking MDR and Dn, the faster operation results can be obtained.

Assembler mnemonic		Notes	V	С	Ν	Ζ	Size	Cycles
divu Dm,Dn	Nomal performance	{MDR,Dn}=0	0	?	•		2	4
	(The operation	Value (MDR,Dn) can specify by 1-byte.	0	?	•	•	2	14
	performed normally)	Value (MDR,Dn) can specify by 2-byte.	0	?	•	•	2	22
		Value (MDR,Dn) can specify by 3-byte.	0	?	•	•	2	30
		Value {MDR,Dn} can specify by 4-byte or more.	0	?	•	•	2	38
	Divisor not specified	{MDR,Dn}=0	1	?	?	?	2	4
	as signed value	Value (MDR,Dn) can specify by 1-byte.	1	?	?	?	2	14
	or	Value (MDR,Dn) can specify by 2-byte.	1	?	?	?	2	22
	zero-division	Value (MDR,Dn) can specify by 3-byte.	1	?	?	?	2	30
		Value (MDR,Dn) can specify by 4-byte or more.	1	?	?	?	2	38

Flag Changes

Nomal performance(the operation performed normally)

VF: Always 0.

CF: Undefined.

NF: 1 if MSB of the divisor(32-bit) is '1'. 0 otherwise.

ZF: 1 if the divisor(32-bit) is '0'. 0 otherwise.

Divisor not specified as signed value or zero-division performed.

VF: Always 1.

CF: Undefined.

NF: Undefined.

ZF: Undefined.



After the operation, if V flag is '1', the other flog will be undefined. Also the divisor and the remainder will be undefined.



Updating of PSW due to flag changes is delayed for one insturuction.



INC Reg

Operation

Reg+1→Reg

Adds '1' to the register(Reg) and stores the result in the register(Reg).

Assembler mnemonic	Notes	٧	С	Ν	Ζ	Size	Cycles
inc Dn		•	•	•	•	1	1
inc An		_	_	_	_	1	1

Flag Changes

inc Dn

VF: 1 if a divisor overflows as 32-bit signed numerical value. 0 otherwise.

CF: 1 if a carry is generated from bit 31. 0 otherwise.

NF: 1 if bit 31 of the result is '1'. 0 otherwise.

ZF: 1 if the operation result is '0'. 0 otherwise

Other than inc Dn

VF: No changes.

CF: No changes.

NF: No changes.

ZF: No changes.



Updating of PSW due to flag changes is delayed for one insturuction.



INC4	An									
Operation	An+4→An									
	Adds '4'	to register A4 and stores the result in register An.								
Assembler	mnemonic	Notes V C N	I Z	Size	Cycles					
inc4 An		-	- -	1	1					
		Flag Changes								
VF: No	changes.									
CF: No	changes.									
NF: No	changes.									
ZF: No	changes.									



Comparison

CMP Reg1,Reg2

Operation

Reg2-Reg:PSW

Subtracts the contents of the register(Reg1) from the register(Reg2) and reflects the result to the flag. Tha same register can not be specified.

Assembler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycles
cmp Dm,Dn	Dm=Dn cannot be specified	•	•	•	•	1	1
cmp Dm,An		•	•	•	•	2	1
cmp Am,Dn		•	•	•	•	2	1
cmp Am,An	Am=An cannot be specified	•	•	•	•	1	1

Flag Changes

VF: 1 if an overflows is generated as 32-bit signed numerical value. 0 otherwise.

CF: 1 if a borrow is generated from bit 31. 0 otherwise.

NF: 1 if bit 31 of the result is '1'. 0 otherwise.

ZF: 1 if the operation result is '0'. 0 otherwise



Updating of PSW due to flag changes is delayed for one insturuction.

However, Bcc and Lcc instructions can evaluate flag before reflecting to PSW.

Operation

Reg-imm:PSW

Subtracts the immediate value from the register(Reg) and reflets the resutls to the flag.

Assembler mnemonic	Notes	V	С	N	Z	Size	Cycles
cmp imm8,Dn	imm8 is signed-extended	•	•	•		2	1
cmp imm16,Dn	imm16 is signed-extended	•	•	•	•	4	1
cmp imm32,Dn		•	•	•	•	6	2
cmp imm8,An	imm8 is zero-extended	•	•	•	•	2	1
cmp imm16,An	imm16 is zero-extended	•	•	•	•	4	1
cmp imm32,An		•	•	•	•	6	2

Flag Changes

VF: 1 if an overflows is generated as 32-bit signed numerical value. 0 otherwise.

CF: 1 if a borrow is generated from bit 31. 0 otherwise.

NF: 1 if bit 31 of the result is '1'. 0 otherwise.

ZF: 1 if the operation result is '0'. 0 otherwise



Updating of PSW due to flag changes is delayed for one insturuction.



Logical AND

AND Dm,Dn

Operation

 $Dm&Dn\rightarrow Dn$

Performs a logical AND and stores the result in register Dn.

Assembler mnemonic	Notes	V	С	Ν	Z	Size	Cycles
and Dm,Dn		0	0		•	2	1

Flag Changes

VF: Always 0.

CF: Always 0.

NF: 1 if bit 31 of the result is '1'. 0 otherwise.

ZF: 1 if the operation result is '0'. 0 otherwise

Updating of PSW due to flag changes is delayed for one insturuction.

However, Bcc and Lcc instructions can evaluate flag before reflecting to PSW.

AND imm,Dn

Operation

 $imm&Dn \rightarrow Dn$

Performs a logical AND of the immediate value(imm) and register Dn and stores the result in register Dn.

Assembler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycles
and imm8,Dn	imm8 is zero-extended	0	0	•	•	3	1
and imm16,Dn	imm16 is zero-extended	0	0	•	•	4	1
and imm32,Dn		0	0	•	•	6	2

Flag Changes

VF: Always 0.

CF: Always 0.

NF: 1 if bit 31 of the result is '1'. 0 otherwise.

ZF: 1 if the operation result is '0'. 0 otherwise



Updating of PSW due to flag changes is delayed for one insturuction.

AND	AND imm,PSW												
Operation imm&PSW→PSW													
	Performs a logical AND of the immediate value(imm) and stores the result.												
Assembler	mnemonic	Notes	V	С	N	Z	Size	Cycles					
and imm16,	PSW	imm16 is zero-extended	•	•	•	•	4	1					
		Flag Changes											
VF: Will I	be set to bit 3 of	the result.											
CF: Will t	be set to bit 2 of	the result.											
NF: Will I	be set to bit 1 of	the result.											
ZF: Will b	ZF: Will be set to bit 0 of the result.												



Updating of PSW by and imm16, PSW is delayed for two instuructions at most.

Especially for interruption affected by IE bit or IM field, note that the instruction during updating will be executed in the status before/after updating.



Logical OR

OR Dm,Dn

Operation

 $Dm|Dn\rightarrow Dn$

Performs a logical OR of register Dm and register Dn and stores the result in register Dn.

Assembler mnemonic	Notes	V	С	Ν	Z	Size	Cycles
or Dm,Dn		0	0		•	2	1

Flag Changes

VF: Always 0. CF: Always 0.

NF: 1 if bit 31 of the result is '1'. 0 otherwise. ZF: 1 if the operation result is '0'. 0 otherwise

Updating of PSW due to flag changes is delayed for one insturuction.

However, Bcc and Lcc instructions can evaluate flag before reflecting to PSW.

OR imm, Dn

Operation

 $imm|Dn \rightarrow Dn$

Performs a logical OR of the immediate value(imm) and register Dn and stores the result in register Dn.

Assembler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycles
or imm8,Dn	imm8 is zero-extended	0	0	•	•	3	1
or imm16,Dn	imm16 is zero-extended	0	0	•	•	4	1
or imm32,Dn		0	0	•	•	6	2

Flag Changes

VF: Always 0. CF: Always 0.

NF: 1 if bit 31 of the result is '1'. 0 otherwise.

ZF: 1 if the operation result is '0'. 0 otherwise



Updating of PSW due to flag changes is delayed for one insturuction.

OR in	OR imm,PSW													
Operation	imm PSW→PSW													
	Performs a logical OR of the immediate value and PSW and stores the result in PSW.													
Assembler	mnemonic	Notes	V	С	Ν	Ζ	Size	Cycles						
or imm16,P	SW	imm16 is zero-extended	•	•	•	•	4	1						
		Flag Changes												
VF: Will	be set to bit 3 or	f the result.												
CF: Will be set to bit 2 of the result.														
NF: Will be set to bit 1 of the result.														
ZF: Will	ZF: Will be set to bit 0 of the result.													



Updating of PSW by and imm16, PSW is delayed for two instuructions at most.

Especially for interruption affected by IE bit or IM field, note that the instruction during updating will be executed in the status before/after updating.



Exclusive Logical OR

XOR	Dm,Dn										
Operation	Dm^Di	n→Dn									
	Performs an exclusive logical OR of register Dm and register Dn and stores the result in register Dn.										
Assembler	mnemonic	Notes V	С	Ν	Ζ	Size	Cycles				
xor Dm,Dn		0	0	•	•	2	1				
		Flag Changes									
VF: Alv	vays 0.										
CF: Alv	vays 0.										
NF: 1 if	bit 31 of the res	ult is '1'. 0 otherwise.									
ZF: 1 if	the operation re	sult is '0'. 0 otherwise									

Updating of PSW due to flag changes is delayed for one insturuction. However, Bcc and Lcc instructions can evaluate flag before reflecting to PSW.

XOR imm,Dn

Operation

imm^Dn→Dn

Performs an exclusive logical OR of the immediate value and register Dn and stores the result in register Dn.

Assembler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycles
xor imm16,Dn	imm16 is zero-extended	0	0	•	•	4	1
xor imm32,Dn		0	0	•	•	6	2

Flag Changes

VF: Always 0. CF: Always 0.

NF: 1 if bit 31 of the result is '1'. 0 otherwise. ZF: 1 if the operation result is '0'. 0 otherwise



Updating of PSW due to flag changes is delayed for one insturuction.



All Bits Inverted

NOT	Dn							
Operation	Dn^0xF	FFFFFF→Dn						
	Inverts a	l bits in register Dn and stores the result in register Dn.						
Assembler	mnemonic	Notes	V	С	Ν	Z	Size	Cycles
not Dn			0	0	•	•	2	1
		Flag Changes						
VF: Alwa	ays 0.							
CF: Alwa	ays 0.							
NF: 1 if t	oit 31 of the resu	it is '1'. 0 otherwise.						
ZF: 1 if the	he operation rest	ılt is '0'. 0 otherwise						



Updating of PSW due to flag changes is delayed for one insturuction.

However, Bcc and Lcc instructions can evaluate flag before reflecting to PSW.



Multiple Bits Test

BTST imm,Dn

Operation

imm&Dn→PSW

Performs a logical AND of the immediate value and the contents of register Dn and reflects the result to the flag.

Assembler mnemonic	Notes	V	С	N	Ζ	Size	Cycles
btst imm8,Dn	imm8 is zero-extended	0	0	•	•	3	1
btst imm16,Dn	imm16 is zero-extended	0	0	•	•	4	1
btst imm32,Dn		0	0	•	•	6	2

Flag Changes

VF: Always 0. CF: Always 0.

NF: 1 if bit 31 of the result is '1'. 0 otherwise. ZF: 1 if the operation result is '0'. 0 otherwise.



Updating of PSW due to flag changes is delayed for one insturuction.

However, Bcc and Lcc instructions can evaluate flag before reflecting to PSW.

BTST imm, Mem

Operation

imm & Mem:PSW

Performs a logical AND of the immediate value(imm) and the contents(byte data) of the memory(Mem) zero-extended to 32-bit and reflects the result to the flag.

Assembler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycles
btst imm8,(d8,An)	imm8 is zero-extended, d8 is sign-extended	0	0	0	•	4	4
btst imm8,(abs16)	imm8 is zero-extended, abs16 is zero-extended	0	0	0	•	5	4
btst imm8,(abs32)	imm8 is zero-extended	0	0	0	•	7	5

Flag Changes

VF: Always 0. CF: Always 0. NF: Always 0.

ZF: 1 if the operation result is '0'. 0 otherwise.



Updating of PSW due to flag changes is delayed for one insturuction.

However, Bcc and Lcc instructions can evaluate flag before reflecting to PSW.







btst imm8,(abs16) is only for Am32. Not usable for AM30/AM31.



Multiple Bits Test & Set

BSET Dm,(An)

Operation

 $mem8(An)(zero_ext) \rightarrow temp$

temp&Dm:PSW

temp|Dm→mem8(An)

- 1. Zero-extends the contents(byte data) of (An) to 32-bit and load to the internal temporary register(temp).
- 2. Performs a logical AND of the contents of the temporary register(temp) and the contents of register Dm and reflects the result to PSW.
- 3. Performs a logical OR of the contents of the temporary register(temp) and the contents of register Dm and stores the lower 8-bit of the result in (An).

Assembler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycles
bset Dm,(An)		0	0	0	•	2	5

Flag Changes

VF: Always 0. CF: Always 0. NF: Always 0.

ZF: 1 if the operation result is '0'. 0 otherwise.



Updating of PSW due to flag changes is delayed for one insturuction.

However, Bcc and Lcc instructions can evaluate flag before reflecting to PSW.



All the operation by this instruction will be done during bus-locked and interruption-disabled.





The operation corresponding to the data of cachable area in the external memory is not bus-locked.

BSET imm, Mem

Operation

Mem(zero_ext)→temp temp&imm:PSW temp|imm→Mem

- 1. Zero-extends the contents(byte data) of the memory(Mem) to 32-bit and loads to the internal temporary register(temp).
- 2. Performs a logical AND of the contents of the temporary register(temp) and the immediate value(imm) and reflects the result to PSW.
- 3. Performs a logical OR of the contents of the temporary register(temp) and immediate value(imm) and stores the lower 8-bit of the result in the memory(Mem).

Assembler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycles
bset imm8,(d8,An)	imm8 is zero-extended, d8 is sign-extended	0	0	0	•	4	5
bset imm8,(abs16)	imm8 is zero-extended, abs16 is zero-extended	0	0	0		5	5
bset imm8,(abs32)	imm8 is zero-extended	0	0	0	•	7	6

Flag Changes

VF: Always 0.

CF: Always 0. NF: Always 0.

ZF: 1 if the operation result is '0'. 0 otherwise.



Updating of PSW due to flag changes is delayed for one insturuction.

However, Bcc and Lcc instructions can evaluate flag before reflecting to PSW.



All the operation by this instruction will be done during bus-locked and interruption-disabled.





The operation corresponding to the data of cachable area in the external memory is not bus-locked.





btst imm8,(abs16) is only for Am32. Not usable for AM30/AM31.



Multiple Bits Test & Clear

BCLR Dm,(An)

Operation

mem8(An)(zero_ext)→temp

temp&Dm:PSW

temp&(Dm^0xFFFFFFF)→mem8(An)

- 1. Zero-extends the contents(byte data) of (An) to 32-bit and load to the internal temporary register(temp).
- 2. Performs a logical AND of the contents of the temporary register(temp) and the contents of register Dm and reflects the result to PSW.
- 3. Performs a logical AND of the contents of the temporary register(temp) and the logical-inverted data of the contents of register Dm and stores the lower 8-bit of the result in (An).

Assembler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycles
bclr Dm,(An)		0	0	0	•	2	5

Flag Changes

VF: Always 0. CF: Always 0. NF: Always 0.

ZF: 1 if the operation result is '0'. 0 otherwise.



Updating of PSW due to flag changes is delayed for one insturuction.

However, Bcc and Lcc instructions can evaluate flag before reflecting to PSW.



All the operation by this instruction will be done during bus-locked and interruption-disabled.





The operation corresponding to the data of cachable area in the external memory is not bus-locked.

BCLR imm, Mem

Operation

Mem(zero_ext)→temp

temp&imm:PSW

temp&(imm^0xFFFFFFF)→Mem

- 1. Zero-extends the contents(byte data) of the memory(Mem) to 32-bit and load to the internal temporary register(temp).
- 2. Performs a logical AND of the contents of the temporary register(temp) and the immediate value(imm) and reflects the result to PSW.
- 3. Performs a logical AND of the contents of the temporary register(temp) and the logical-inverted data of the immediate value(imm) and stores the lower 8-bit of the result in the memory(Mem).

Assembler mnemonic	Notes	V	С	Ν	Z	Size	Cycles
bclr imm8,(d8,An)	imm8 is zero-extended, d8 is sign-extended	0	0	0	•	4	5
bclr imm8,(abs16)	imm8 is zero-extended, abs16 is zero extended	0	0	0	•	5	5
bclr imm8,(abs32)	imm8 is zero-extended	0	0	0	•	7	6

Flag Changes

VF: Always 0. CF: Always 0.

NF: Always 0.

ZF: 1 if the operation result is '0'. 0 otherwise.



Updating of PSW due to flag changes is delayed for one insturuction.

However, Bcc and Lcc instructions can evaluate flag before reflecting to PSW.



All the operation by this instruction will be done during bus-locked and interruption-disabled.





The operation corresponding to the data of cachable area in the external memory is not buslocked.





bclr imm8,(abs16) is only for Am32. Not usable for AM30/AM31.



ASR Dm,Dn

Operation

If not (Dm&0x0000001F)=0

Dn.lsb→CF

 $(Dn \gg (Dm\&0x0000001F))(sign_ext)\rightarrow Dn$

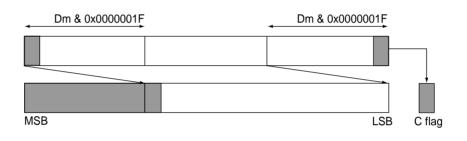
If (Dm & 0x0000001F) = 0

PC+2→PC

Performs an arithmetic shift right on the contents of register Dn for bits specified with the lower 5 bits of register Dm and stores the result in register Dn.

No shift-operation if the contents of lower 5 bits of register Dm is '0'.

The upper 27 bits of register Dm will be ignored.



Assembler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycle
asr Dm,Dn	Contents of lower 5 bits of Dm are other than '0'	?	•	•	•	2	1
	Contents of lower 5 bits of Dm are '0'	?	?	•	•		

Flag Changes

Contents of lower 5 bits of Dm are other than '0'

VF: Not specified.

CF: Reflects the bit value firstly shifted out.

NF: 1 if bit 31 of the result is '1'. 0 otherwise.

ZF: 1 if the result is '0'. 0 otherwise.

Contents of lower 5 bits of Dm are '0'

VF: Not specified.

CF: Not specified.

NF: 1 if bit 31 of Dn is '1'. 0 otherwise.

ZF: 1 if Dn is '0'. 0 otherwise.



Updating of PSW due to flag changes is delayed for one insturuction.

ASR imm8,Dn

Operation

If not (imm8 & 0x1F)=0

Dn.lsb→CF

 $(Dn>>(imm8\&0x1F))(sign_ext)\rightarrow Dn$

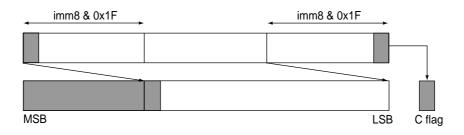
If (imm8&0x1F)=0

PC+3→PC

Performs an arithmetic shift right on the contents of register Dn for bits specified with the lower 5 bits of the immediate value(imm8) and stores the result in register Dn.

No shift-operation if the contents of lower 5 bits of the immediate value(imm8) is '0'.

The upper 3 bits of the immediate value(imm8) will be ignored.



Assembler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycle
asr imm8,Dn	Contents of lower 5 bits of imm8 are other than '0'	?	•	•	•	3	1
	Contents of lower 5 bits of imm8 are '0'	?	?	•	•		

Flag Changes

Contents of lower 5 bits of imm8 are other than '0'

VF: Not specified.

CF: Reflects the bit value firstly shifted out.

NF: 1 if bit 31 of the result is '1'. 0 otherwise.

ZF: 1 if the result is '0'. 0 otherwise.

Contents of lower 5 bits of imm8 are '0'

VF: Not specified.

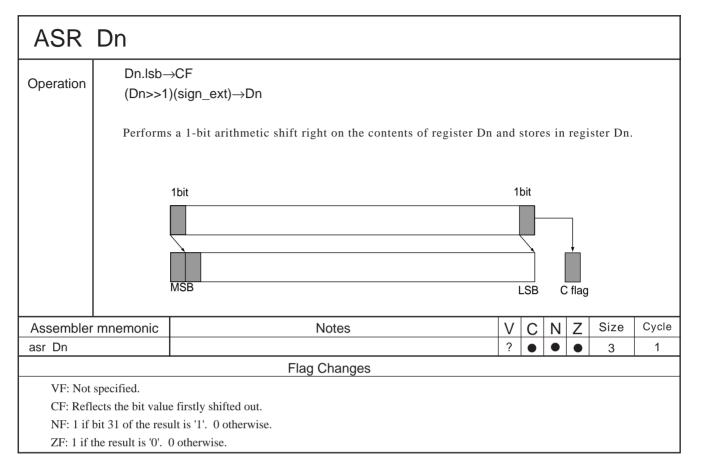
CF: Not specified.

NF: 1 if bit 31 of Dn is '1'. 0 otherwise.

ZF: 1 if Dn is '0'. 0 otherwise.

Updating of PSW due to flag changes is delayed for one insturuction.





Updating of PSW due to flag changes is delayed for one insturuction.

However, Bcc and Lcc instructions can evaluate flag before reflecting to PSW.

This instruction is executed by overwriting the instructions and the assembler generates the following instructions.

The numbers of Size and Cycles are those of the instruction mentioned above.



Logical Shift Right for Optional Bit

LSR Dm,Dn

Operation

If not (Dm&0x0000001F)=0

Dn.lsb→CF

 $(Dn>>(Dm\&0x0000001F))(zero_ext)\rightarrow Dn$

If (Dm&0x0000001F)=0

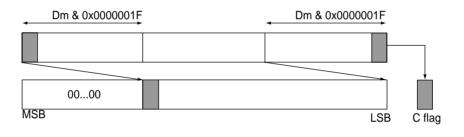
PC + 2→PC

Performs a logical shift right on the contents of register Dn for bits specified with the lower 5 bits of register Dm and stores the result in register Dn.

No shift-operation if the contents of lower 5 bits of register Dm is '0'.

The upper 27 bits of register Dm will be ignored.

'0' is input in MSB.



Assembler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycle
Isr Dm,Dn	Contents of lower 5 bits of Dm are other than '0'	?	•	•	•	2	1
	Contents of lower 5 bits of Dm are '0'	?	?	•	•		

Flag Changes

Contents of lower 5 bits of Dm are other than '0'

VF: Not specified.

CF: Reflects the bit value firstly shifted out.

NF: 1 if bit 31 of the result is '1'. 0 otherwise.

ZF: 1 if the result is '0'. 0 otherwise.

Contents of lower 5 bits of Dm are '0'

VF: Not specified.

CF: Not specified.

NF: 1 if bit 31 of Dn is '1'. 0 otherwise.

ZF: 1 if Dn is '0'. 0 otherwise.



Updating of PSW due to flag changes is delayed for one insturuction.

LSR imm8,Dn

Operation

If not (imm8&0x1F)=0

Dn.lsb→CF

(Dn>>(imm8&0x1F))(zero_ext)→Dn

If (imm8&0x1F)=0

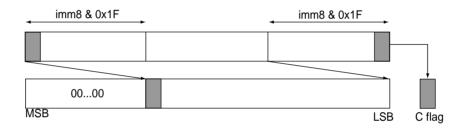
PC+3→PC

Performs a logical shift right on the contents of register Dn for bits specified with the lower 5 bits of the immediate value(imm8) and stores the result in register Dn.

No shift-operation if the contents of lower 5 bits of the immediate value(imm8) is '0'.

The upper 3 bits of the immediate value(imm8) will be ignored.

'0' is input in MSB.



Assembler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycle
Isr imm8,Dn	Contents of lower 5 bits of imm8 are other than '0'	?	•	•	•	3	1
	Contents of lower 5 bits of imm8 are '0'	2	?				

Flag Changes

Contents of lower 5 bits of imm8 are other than '0'

VF: Not specified.

CF: Reflects the bit value firstly shifted out.

NF: 1 if bit 31 of the result is '1'. 0 otherwise.

ZF: 1 if the result is '0'. 0 otherwise.

Contents of lower 5 bits of imm8 are '0'

VF: Not specified.

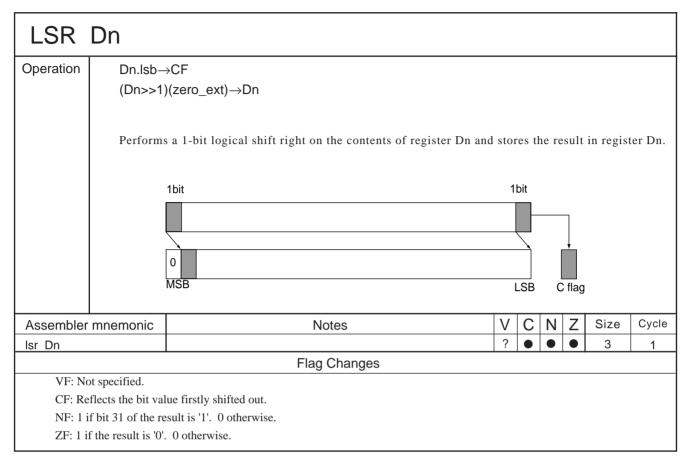
CF: Not specified.

NF: 1 if bit 31 of Dn is '1'. 0 otherwise.

ZF: 1 if Dn is '0'. 0 otherwise.

Updating of PSW due to flag changes is delayed for one insturuction.





Updating of PSW due to flag changes is delayed for one insturuction.

However, Bcc and Lcc instructions can evaluate flag before reflecting to PSW.



.This instruction is executed by overwriting the instructions and the assembler generates the following instructions.



The numbers of Size and Cycles are those of the instruction mentioned above.



ASL Dm,Dn

Operation

If not (Dm&0x0000001F)=0

 $Dn << (Dm\&0x0000001F) \rightarrow Dn$

If (Dm&0x0000001F)=0

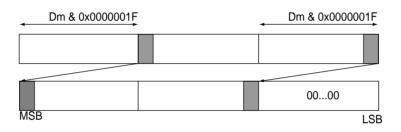
PC+2→PC

Performs an arithmetic shift left on the contents of register Dn for bits specified with the lower 5 bits of register Dm and stores the result in register Dn.

No shift-operation if the contents of lower 5 bits of register Dm is '0'.

The upper 27 bits of register Dm will be ignored.

'0' is input in LSB.



Assembler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycle
asl Dm,Dn	Contents of lower 5 bits of Dm are other than '0'	?	?	•	•	2	1
	Contents of lower 5 bits of Dm are '0'						

Flag Changes

Contents of lower 5 bits of Dm are other than '0'

VF: Not specified.

CF: Not specified.

NF: 1 if bit 31 of the result is '1'. 0 otherwise.

ZF: 1 if the result is '0'. 0 otherwise.

Contents of lower 5 bits of Dm are '0'

VF: Not specified.

CF: Not specified.

NF: 1 if bit 31 of Dn is '1'. 0 otherwise.

ZF: 1 if Dn is '0'. 0 otherwise.



Updating of PSW due to flag changes is delayed for one insturuction.

ASL imm8,Dn

Operation

If not (imm8&0x1F)=0

Dn<<(imm8&0x1F)→Dn

If (imm8&0x1F)=0

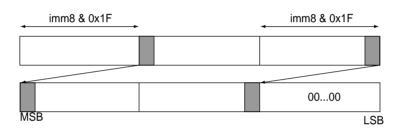
PC+3→PC

Performs an arithmetic shift left on the contents of register Dn for bits specified with the lower 5 bits of the immediate value(imm8) and stores the result in register Dn.

No shift-operation if the contents of lower 5 bits of the immediate value(imm8) is '0'.

The upper 3 bits of the immediate value(imm8) will be ignored.

'0' is input in LSB.



Assembler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycle
asl imm8,Dn	Contents of lower 5 bits of imm8 are other than '0'	?	?	•	•	3	1
	Contents of lower 5 bits of imm8 are '0'						

Flag Changes

Contents of lower 5 bits of imm8 are other than '0'

VF: Not specified.

CF: Not specified.

NF: 1 if bit 31 of the result is '1'. 0 otherwise.

ZF: 1 if the result is '0'. 0 otherwise.

Contents of lower 5 bits of imm8 are '0'

VF: Not specified.

CF: Not specified.

NF: 1 if bit 31 of Dn is '1'. 0 otherwise.

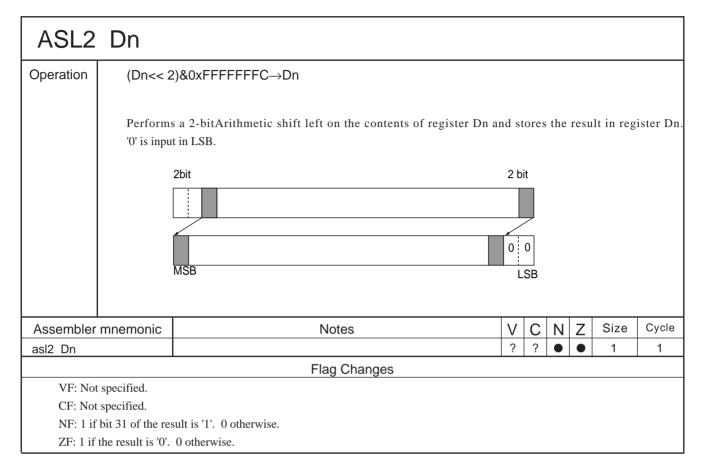
ZF: 1 if Dn is '0'. 0 otherwise.



Updating of PSW due to flag changes is delayed for one insturuction.



2-Bit Arithmetic Shift Left

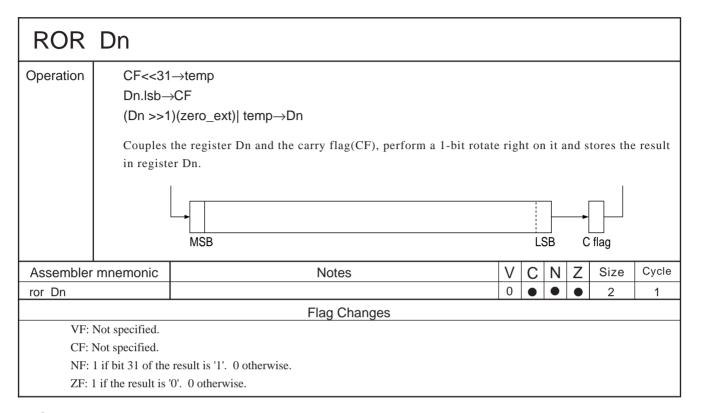




Updating of PSW due to flag changes is delayed for one insturuction.



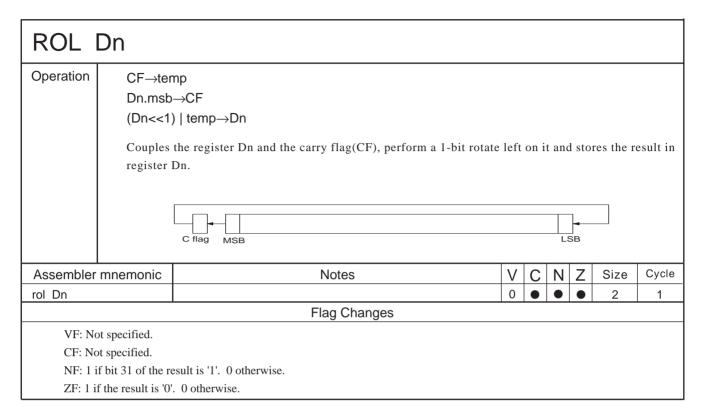
1-bit Rotate Right



Updating of PSW due to flag changes is delayed for one insturuction. However, Bcc and Lcc instructions can evaluate flag before reflecting to PSW.



1-bit Rotate Left



Updating of PSW due to flag changes is delayed for one insturuction.

However, Bcc and Lcc instructions can evaluate flag before reflecting to PSW.



Branch on Condition Codes

Bcc label

Operation

When branch is taken,

PC d8(sign_ext)→PC

Sign-extends 8-bit displacement(d8), adds program counter(PC) and stores the result in the program counter(PC).

Stores the result in the program counter(PC). Ignore it even if the result overflows.

When branch is not taken,

PC+CodeSize→PC

Executes following instructions.

Ass	ebler mnemonic		Notes	V	С	N	Ζ	Size	Cycles
beq	label	Z	Branches at Z flag set	_	_	_	—	2	3/1
bne	label	~Z	Branches at Z flag clear	_	_	_	_	2	3/1
bgt	label	~(Z (N^V))	Branches at < (with signed)	_	—	_	_	2	3/1
bge	label	~(N^V)	Branches at ≤ (with signed)	—	—	_	—	2	3/1
ble	label	Z (N^V)	Branches at ≥ (with signed)	—	_	_	—	2	3/1
blt	label	N^V	Branches at > (with signed)	_	_	_	_	2	3/1
bhi	label	~(C Z)	Branches at < (no signed)	_	_	_	_	2	3/1
bcc	label	~C	Branches at ≤, C flag clear (no signed)	_	_	_	_	2	3/1
bls	label	C Z	Branches at ≥ (no signed)	_	_	_	_	2	3/1
bcs	label	С	Branches at >, C flag clear (no signed)	_	_	_	_	2	3/1
bvc	label	~V	Branches at V flag clear	_	_	_	—	3	4/2
bvs	label	V	Branches at V flag set	_	_	_	_	3	4/2
bnc	label	~N	Branches at N flag clear	_	_	_	_	3	4/2
bns	label	N	Branches at N flag set					3	4/2
bra	label	None	Branches unconditionally	_	_		_	2	3

Flag Changes

VF; No changes

CF: No changes

NF: No changes

ZF: No changes



"Cycles" describes the cycles of "branch"/"not branch".



The cycles of "not branch" depend on status of an instruction queue.

Loop on Condition Codes

Lcc

Operation

When branch is taken,

LAR-4→PC

The instruction loaded to the loop instruction register (LIR) is executed and instruction fetch starts for the address loaded to the loop address register (LAR).

At the same time, 4 is subtracted from the loop address register (LAR) and the results are written into the PC. Stores the result in the program counter(PC). Ignore it even if the result overflows.

Lcc is used together with SETLB in order to increase the loop execution speed, and performs conditional branch to the top of the loop set by SETLB.

When branch is not taken,

LAR+1→PC

Executes the following instructions.

Assebler mnemonic		Notes	٧	С	Ν	Z	Size	Cycles
leq	Z	Branches at Z flag set	_	_	_	_	1	1/2
Ine	~Z	Branches at Z flag clear	_	_	_	_	1	1/2
lgt	~(Z (N^V))	Branches at <(with signed)	_	_	_	_	1	1/2
lge	~(N^V)	Branches at ≤(with signed)	_	_	_	_	1	1/2
lle	Z (N^V)	Branches at <u>≥</u> (with signed)	_	_	_	_	1	1/2
llt	N^V	Branches at >(with signed)	_	_	_	_	1	1/2
lhi	~(C Z)	Branches at <(no signed)	_	_	_	_	1	1/2
Icc	~C	Branches at \leq , C flag clear(no signed)	_	_	_	_	1	1/2
lls	C Z	Branches at <u>≥</u> (no signed)	_	_	_	_	1	1/2
lcs	С	Branches at <, C flag set(no signed)	_			_	1	1/2
Ira	None	Branches unconditionally	_	_	_	_	1	1

Flag Changes

VF: No changes CF: No changes NF: No changes ZF: No changes



The execution without corresponding to SETLB insturuction is not guaranteed.



"Cycles" describes the cycles of "branch"/"not branch".



The cycles of "not branch" depend on status of an instruction queue.



Set Loop Buffer

SETLB

Operation

mem32(PC+1)→LIR,

PC+5→LAR

The 4-byte instruction string and 5th byte address following to SETLB are loaded to the loop instruction register (LIR) and loop address register (LAR) respectively.

SETLB is used together with Lcc in order to increase the loop (the innermost loop) execution speed. The top of the loop is set by SETLB just before the loop entrance.

Assebler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycles
setlb		_	_	_	_	1	1

Flag Changes

=4n

VF; No changes CF: No changes NF: No changes

ZF: No changes









A method of storing in LIR depends on microcomputer core type(AM30/AM31/AM32).

When the instrucion strings following to SETLB are the following,

SETLB

Α

В

C

storing in LIR is as shown below.

AM30



AM31/AM32



Operation

An→PC

Stores the contents of register An in program counter(PC).

Assebler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycles	
jmp (An)		_	_	_	_	2	3	

Flag Changes

VF; No changes CF: No changes NF: No changes ZF: No changes

JMP label

Operation

If displacement from program counter(PC) ro label is performed in 16-bit,

PC+d16(sign_ext)→PC

Sign-extends d16, added to PC and stores the results in PC. Ignore it even if the results overflows and stores the result in PC.

If displacement from program counter(PC) ro label is performed in 32-bit,

PC+d32→PC

Adds the 32-bit displacement and PC and stores the results in PC. Ignore it even if the results overflows and stores the result in PC.

Assebler mnemonic	Notes	V	С	N	Z	Size	Cycles
jmp label	If displacement from program counter(PC) ro label is	_	_	_	_	3	2
	performed in 16-bit;						
	If displacement from program counter(PC) ro label is	_	_	_	_	5	3*
	performed in 32-bit						

Flag Changes

VF; No changes CF: No changes NF: No changes ZF: No changes



*: "Cycles" is four.

The assembler chooses the most suitable displacement; d16 or d32.



Subroutine Call

CALL label

Operation

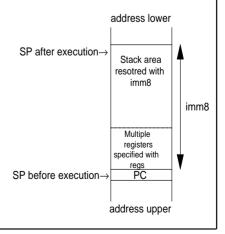
When displacement from program counter(PC) to label is performed within 16-bit,

```
If registers specified with "regs"= 2 PC+5\rightarrowmem32(SP-4) , reg2\rightarrowmem32(SP-8) , SP-imm8(zero_ext)\rightarrowSP, PC+5\rightarrowMDR , PC+d16(sign_ext)\rightarrowPC If registers specified with "regs"= 11 PC+5= 2mem32(SP) D2\rightarrowmem32(SP-4), D3\rightarrowmem32(SP-8), A2\rightarrowmem32(SP-12), A3\rightarrowmem32(SP-16), D0\rightarrowmem32(SP-20), D1\rightarrowmem32(SP-24) , A0\rightarrowmem32(SP-28), A1\rightarrowmem32(SP-32), MDR\rightarrowmem32(SP-36), LIR\rightarrowmem32(SP-40), LAR\rightarrowmem32(SP-44) , SP-imm8(zero_ext)\rightarrowSP, PC+5\rightarrowMDR, PC+d16(sign_ext)\rightarrowPC
```

When displacement from program counter(PC) to label is performed within 32-bit,

```
If registers specified with "regs"= 2 PC+7\rightarrow mem32(SP), reg1\rightarrow mem32(SP-4), reg2\rightarrow mem32(SP-8), SP\text{-}imm8(zero\_ext)\rightarrow SP, PC+7\rightarrow MDR, PC+d32\rightarrow PC If registers specified with "regs"= 11 PC+7\rightarrow mem32(SP) D2\rightarrow mem32(SP-4), D3\rightarrow mem32(SP-8), A2\rightarrow mem32(SP-12), A3\rightarrow mem32(SP-16), D0\rightarrow mem32(SP-20), D1\rightarrow mem32(SP-24), A0\rightarrow mem32(SP-28), A1\rightarrow mem32(SP-32), MDR\rightarrow mem32(SP-36), LIR\rightarrow mem32(SP-40), LAR\rightarrow mem32(SP-44), SP\text{-}imm8(zero\_ext)\rightarrow SP, PC+7\rightarrow MDR, PC+d32\rightarrow PC
```

This instruction branches to the specified address after saving the PC and multiple registers for the next instruction and restoring the stack area. The immediate value(regs) specifies the multiple registers to be saved and the immediate value(imm8:zero-extended) specifies the area to be restored (bytes). (Refer to MOVM instruction for details of "regs".) CALL is used together with RET or RETF to save/restore registers and allocate/deallocate the stack area quickly during returning from subroutine. The status of the stack frame after CALL is shown at the right.



Assebler mnemonic	Note	es	V	С	N	Ζ	Size	Cycles
call label	When displacement	registers specified with "regs"=0	_	_	_	_	5	2
	from program counter	registers specified with "regs"=1	_	_	_	_	5	3
	(PC) to label is	registers specified with regs = 2	_	_	_	_	5	4
	performed within 16-bit,	registers specified with "regs"=3	_	_	_	_	5	5
		registers specified with "regs"=4	_	_	_	_	5	6
		registers specified with "regs"=7	_	_	_	—	5	9
		registers specified with "regs"=8	_	_	_	_	5	10
		registers specified with "regs"=9	_	—	_	_	5	11
		registers specified with "regs"=10	_	_	_	_	5	12
		registers specified with regs =11	_	_	_	_	5	13
	When displacement	registers specified with regs =0	_	_	_	_	7	4*
	1	registers specified with regs =1	_	_	_	_	7	4*
	from program counter	registers specified with regs = 2	_	_	_	_	7	5*
	(PC) to label is	registers specified with regs=3	_	_	_	_	7	6*
	performed within 32-bit,	registers specified with regs =4	_	_	_	_	7	7*
		registers specified with "regs"=7	_	_	_	_	7	10*
		registers specified with regs = 8	_	_	_	_	7	11*
		registers specified with "regs"=9	_	_	_	—	7	12*
		registers specified with "regs"=10	_	_	_	_	7	13*
		registers specified with "regs"=11	_	_	_	_	7	14*
	Fla	ag Changes						

Flag Changes

VF; No changes CF: No changes NF: No changes ZF: No changes



Three operands of d16, regs, imm8 are used for the bit assignment.

The assember does not specify the multiple registers to be saved and the area to be restored(regs, imm8). Pseud instruction at subroutine CALL specifies them indirectly then finally the linker executes them. Refer to "Cross Assembler User's Manual" for details.



Assembler selects d16 or d32 for the best.



Refer to "Appendix Instruction Set" for operation expressions by each register specified with "regs".





*:"Cycles" is the figures mentioned above plus 1.



Subroutine Call

CALLS (An)

Operation

PC+2→mem32(SP).

PC+2→MDR,

An→PC

This instruction branches to the specified address after saving the PC for the next instruction to the stack. CALLS is used together with RETS in the case of registers to be saved and the stack area to be allocated are unclear, and to maintain compatibility(use with JSR).

Assebler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycles
calls (An)		_	_	_	_	2	3

Flag Changes

VF; No changes CF: No changes NF: No changes ZF: No changes

CALLS label

Operation

When displacement from program counter(PC) to label is performed within 16-bit,

PC+4→mem32(SP), PC+4→MDR, PC+d16(sign_ext)→PC

When displacement from program counter(PC) to label is performed within 32-bit,

PC+6 \rightarrow mem32(SP), PC+6 \rightarrow MDR , PC+d32 \rightarrow PC

This instruction branches to the specified address after saving the PC for the next instruction to the stack. Ignore them even if the result of addition is overflowed and store them into the PC. This instruction is used together with RETS in the case of registers to be saved and the stack area to be allocated are unclear, and to main compatibility(use with JSR).

Assebler mnemonic	Notes	٧	C	Ν	Ζ	Size	Cycles
calls label	When displacement from program counter(PC) to label is		_	_	_	4	3
	performed within 16-bit						
	When displacement from program counter(PC) to label is	_	_	_	_	6	3*
	performed within 32-bit						

Flag Changes

VF; No changes CF: No changes NF: No changes ZF: No changes



*: "Cycles" is four.



Return from subroutine

RET

Operation

If registers specified with "regs"= 2

SP+imm8(zero ext)→SP,

mem32(SP-4)→reg1, mem32(SP-8)→reg2,

mem32(SP)→PC

If registers specified with "regs"= 11

SP+imm8(zero_ext)→SP,

mem32(SP-4)→D2, mem32(SP-8)→D3, mem32(SP-12)→A2,

mem32(SP-16)→A3, mem32(SP-20)→D0, mem32(SP-24)→D1,

 $mem32(SP-28)\rightarrow A0$, $mem32(SP-32)\rightarrow A1$, $mem32(SP-36)\rightarrow MDR$,

 $mem32(SP-40) {\rightarrow} LIR, \, mem32(SP-44) {\rightarrow} LAR \; ,$

mem32(SP)→PC

This instruction branches to the return address after saving the PC and multiple registers for the next instruction and restoring the stack area. The immediate value(regs) specifies the multiple registers to be saved and the immediate value(imm8:zero-extended) specifies the area to be restored (bytes). (Refer to MOVM instruction for details of "regs".) CALL is used together with RET to save/restore registers and allocate/deallocate the stack area quickly during returning from subroutine. If the subroutine does not overwrite MDR, RETF deallocate quickly.

Assebler mnemonic	Notes	V	С	N	Ζ	Size	Ctcles
ret	Registers specified with "regs"=0	_	_	_	_	3	5*
	Registers specified with "regs"=1	_	_	_	—	3	5*
	Registers specified with "regs"=2	_	_	_	_	3	5*
	Registers specified with "regs"=3	_	_	_	_	3	5*
	Registers specified with "regs"=4	_	_	_	_	3	5
	Registers specified with "regs"=7	_	_	_	_	3	8
	Registers specified with "regs"=8	_	_	_	_	3	9
	Registers specified with "regs"=9	_	_	_	_	3	10
	Registers specified with "regs"=10	_	_	_	—	3	11
	Registers specified with "regs"=11	_	_	_		3	12

Flag Changes

VF; No changes

CF: No changes

NF: No changes

ZF: No changes



Two operands of regs, imm8 are used for the bit assignment.

The assember does not specify the multiple registers to be saved and the area to be restored(regs, imm8). Pseud instruction at subroutine CALL specifies them indirectly then finally the linker executes them.

Refer to "Cross Assembler User's Manual" for details.



Refer to "Appendix Instruction Set" for operation expressions by each register specified with "regs".



*: "Cycles" is four.



Return from Subroutine

RETF

Operation

If registers specified with "regs"= 2

SP+imm8(zero_ext)→SP, MDR→PC,

mem32(SP-4)→reg1, mem32(SP-8)→reg2

If registers specified with "regs"= 11

SP + imm8(zero_ext)→SP, MDR→PC,

mem32(SP-4) \rightarrow D2, mem32(SP-8) \rightarrow D3, mem32(SP-12) \rightarrow A2,

mem32(SP-16)→A3, mem32(SP-20)→D0, mem32(SP-24)→D1,

mem32(SP-28)→A0, mem32(SP-32)→A1, mem32(SP-36)→MDR,

mem32(SP-40)→LIR, mem32(SP-44)→LAR

This instruction branches to the return address in MDR after saving the multiple registers and restoring the stack area. The immediate value(regs) specifies the multiple registers to be returned and the immediate value(imm8:zero-extended) specifies the area to be restored (bytes). (Refer to MOVM for details of "regs".) CALL is used together with RETF to save/restore registers and allocate/deallocate the stack area quickly during returning from subroutine.

When overwriting MDR within suburoutine, the operation for returning is not guaranteed. (Use RET.)

Assebler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycles
retf	Registers specified with "regs"=0	_	—	_	_	3	2
	Registers specified with "regs"=1	_	—	_	_	3	2
	Registers specified with "regs"=2	_	_	_	_	3	3
	Registers specified with "regs"=3	_	_	_	_	3	4
	Registers specified with "regs"=4	_	—	—	_	3	5
	Registers specified with "regs"=7	_	—	_	_	3	8
	Registers specified with "regs"=8	_	—	_	_	3	9
	Registers specified with "regs"=9	_	_	_	_	3	10
	Registers specified with "regs"=10	_	_	_	_	3	11
	Registers specified with "regs"=11					3	12

Flag Changes

VF; No changes CF: No changes NF: No changes ZF: No changes



Two operands of regs, imm8 are used for the bit assignment.

The assember does not specify the multiple registers to be saved and the area to be restored(regs, imm8). Pseud instruction at subroutine CALL specifies them indirectly then finally the linker executes them.

Refer to "Cross Assembler User's Manual" for details.



When overwriting MDR within suburoutine, the operation for returning is not guaranteed.(Use RET.)



Refer to "Appendix Instruction Set" for operation expressions by each register specified with "regs".



Return from Subroutine

RETS								
Operation	mem32(SP)→PC			٦			
	RETS is	to the returning address stored in the stack. used together with CALLS. used to maintain compatibility(use with RTS).						
Assebler n	nnemonic	Notes V C N	Z Si	ze Cycle	;s			
rets			_ :	2 5*	П			
		Flag Changes						
VF; No chang	ges							
CF: No chang	CF: No changes							
NF: No chang	ges							
ZF: No chang	es							



*: "Cycles" is four.



Subroutine Call [Combination of Multiple Instructions]

JSR (An)

Operation

SP-4→SP,

PC+2→mem32(SP), PC+2→MDR

An→PC

(subroutine execution)

SP+4→SP

Branches to the specified address after saving the PC for the next instruction to the stack.

Assebler mnemonic	Notes	V	С	Ν	Ζ	Size	Cycles
jsr (An)		•	•	•	•	8	5

Flag Changes

VF: Depends on the subroutine processing

CF: Depends on the subroutine processing

NF: Depends on the subroutine processing

ZF: Depends on the subroutine processing



Updating of PSW due to flag changes is delayed for one insturuction.

However, Bcc and Lcc instructions can evaluate flag before reflecting to PSW.



This instruction is executed by overwriting the instructions and the assembler generates the following instructions.

ADD -4,SP CALLS (An) ADD 4,SP

The numbers of Size and Cycles are those of the instruction mentioned above. The optimization of assembler may change the cycles. Refer to "Chapter 3, Using Instructions".

JSR label

Operation

When displacement from program counter(PC) to label is performed within 16-bit,

SP-4→SP,

PC+4→mem32(SP), PC+4→MDR,

PC+d16(sign_ext)→PC,

(subroutine execution)

SP+4→SP

When displacement from program counter(PC) to label is performed within 32-bit,

SP-4→SP,

PC+6→mem32(SP), PC+6→MDR

PC+d32→PC,

(subroutine execution)

SP+4→SP

Branches to the specified address after saving the PC for the next instruction to the stack.

Ignores even if the results overflows and stores the result in PC.

Assebler mnemonic	Notes	V	С	N	Z	Size	Cycles
jsr label	When displacement from program counter(PC) to label is	•	•	•	•	10	5
	performed within 16-bit,						
	When displacement from program counter(PC) to label is	•	•	•	•	12	5*
	performed within 32-bit,						

Flag Changes

VF: Depends on the subroutine processing

CF: Depends on the subroutine processing

NF: Depends on the subroutine processing

ZF: Depends on the subroutine processing



Updating of PSW due to flag changes is delayed for one insturuction.

However, Bcc and Lcc instructions can evaluate flag before reflecting to PSW.



This instruction is executed by overwriting the instructions and the assembler generates the following instructions.

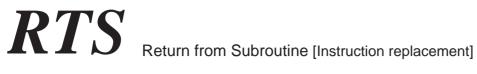
ADD -4,SP CALLS label ADD 4,SP



The numbers of Size and Cycles are those of the instruction mentioned above. The optimization of assembler may change the cycles. Refer to "Chapter 3, Using Instructions".



*: "Cycles" is six.



RTS						
Operation	mem32(SP)→PC					
	Branches to the returning address stored in the stack. RTS is used together with JSR to maintain compatibility.					
Assebler mnemonic		Notes V C	Ν	Ζ	Size	Cycles
rts			_	_	2	4
Flag Changes						
VF; No changes						
CF: No changes						
NF: No changes						
ZF: No changes						



This instruction is executed by overwriting the instructions and the assembler generates the following instructions.



The numbers of Size and Cycles are those of the instruction mentioned above.



Return from Program

RTI										
Operation	Operation mem16(SP)→PSW , mem32(SP+4)→PC , SP+8→SP Returns from the interrupt by branching to the return address stored in the stack after restoring the PSW contained in the stack.									
Assebler m	nemonic	Notes	٧	С	N	Z	Size	Cycles		
rti			•	•	•	•	2	4		
		Flag Changes				-				
VF: The	VF: The V flag of the saved PSW.									
CF: The V flag of the saved PSW.										
NF: The	NF: The V flag of the saved PSW.									
ZF: The	V flag of the s	ved PSW.								



Subroutine Call to a specified Address

TRAP							
Operation	PC+2→mem32(SP), 0x40000010→PC Branches to the specified address (0x40000010) after saving the PC of the next instruction to the stack. It is used for system call (calling the OS and library).						
Assebler r	mnemonic	Notes V C	N	Ζ	Size	Cycles	
trap		- -	_	-	2	4	
		Flag Changes					
VF; No	VF; No changes						
CF: No changes							
NF: No changes							
ZF: No	changes						



No Operation

NOP									
Operation	PC+1→	C C							
	No opera	ion.							
Assebler r	mnemonic	Notes V C N	Ζ	Size	Cycles				
nop				1	1				
		Flag Changes							
VF; N	VF; No changes								
CF: N	CF: No changes								
NF: N	o changes								
ZF: No	o changes								



UDFnn Dm,Dn (nn = 00 to 15, 20 to 35)

Operation

When nn=00 to 15,

Dm op Dn→Dn

Performs an operation on the contents of register Dm and register Dn and stores the result in register Dn. The operation and flag changes are user defined.

When nn=20 to 35,

Dm op Dn

Performs an operation on the contents of register Dm and register Dn but the result is not written into register Dn and the flags are not changed.

Assembler mnemonic	Notes	V	С	N	Ζ	Size	Cycles
udfnn Dm,Dn	When nn = 00 to 15		*	*	*	2	User defined
	When nn = 20 to 35					2	User defined

Flag Changes

When nn = 00 to 15

VF: User defined

CF: User defined

NF: User defined

ZF: User defined

When nn = 20 to 35

VF: No changes

CF: No changes

NF: No changes

ZF: No changes



Updating of PSW due to flag changes is delayed for one insturuction.

However, Bcc and Lcc instructions can evaluate flag before reflecting to PSW.





MOVM [regs],(SP) can not be located in one instruction before this instruction.

UDFnn imm, Dn (nn=00 to 15, 20 to 35)

Operation

When nn= 00 to 15,

imm op Reg→Reg

Performs an operation on the zero-extended immediate value(imm8 or imm16) or 32-bit immediate value(imm32) and register Dn, then stores the result in register Dn.

The contents of the operation and flag changes are user defined.

When nn = 20 to 35,

imm op Reg

Performs an operation on the zero-extended immediate value(imm8 or imm16) or 32-bit immediate value(imm32) and register Dn, then stores the result in register Dn.

The result is not written into register Dn and the flags are not changed.

Assembler mnemonic	Notes			Ν	Ζ	Size	Cycles
udfnn imm8,Dn	When nn=00 to 15, imm8 is sign-extended	*	*	*	*	3	User defined
udfnn imm16,Dn	When nn=00 to 15, imm16 is sign-extended		*	*	*	4	User defined
udfnn imm32,Dn	When nn=00 to 15,		*	*	*	6	User defined
udfnn imm8,Dn	When nn=20 to 35, imm8 is sign-extended	_	_	_	—	3	User defined
udfnn imm16,Dn	When nn=20 to 35, imm16 is sign-extended	_	_		_	4	User defined
udfnn imm32,Dn	When nn=20 to 35,	_	_			6	User defined

Flag Changes

When nn = 00 to 15

VF: User defined

CF: User defined

NF: User defined

ZF: User defined

When nn = 20 to 35

VF: No changes

CF: No changes

NF: No changes

ZF: No changes



Updating of PSW due to flag changes is delayed for one insturuction.

However, Bcc and Lcc instructions can evaluate flag before reflecting to PSW.



"nn=20 to 35" is only for AM31/AM32. It can not be used for AM30.



MOVM [regs],(SP) can not be located in one instruction before this instruction.



UDFUnn imm, Dn (nn=00 to 15, 20 to 35)

Operation

When nn=00 to 15,

imm op Reg→Reg

Performs an operation on the zero-extended immediate value(imm8 or imm16) or 32-bit immediate value(imm32) and register Dn, then stores the result in register Dn.

The contents of the operation and flag changes are user defined.

When nn=20 to 35, imm op Reg

Performs an operation on the zero-extended immediate value(imm8 or imm16) or 32-bit immediate value(imm32) and register Dn, then stores the result in register Dn.

The result is not written into register Dn and the flags are not changed.

Assembler mnemonic	Notes		С	Ν	Ζ	Size	Cycles
udfnn imm8,Dn	When nn=00 to 15, imm8 is zero-extended.	*	*	*	*	3	User defined
udfnn imm16,Dn	When nn=00 to 15, imm16 is zero-extended.	*	*	*	*	4	User defined
udfnn imm32,Dn	When nn=00 to 15,	*	*	*	*	6	User defined
udfnn imm8,Dn	When nn=20 to 35, imm8 is zero-extended.	_	_	_	_	3	User defined
udfnn imm16,Dn	When nn=20 to 35, imm16 is zero-extended.	_	_	_	_	4	User defined
udfnn imm32,Dn	When nn=20 to 35,	_	_	_	_	6	User defined

Flag Changes

When nn = 00 to 15

VF: User defined

CF: User defined

NF: User defined

ZF: User defined

When nn = 20 to 35

VF: No changes

CF: No changes

NF: No changes

ZF: No changes



Updating of PSW due to flag changes is delayed for one insturuction.

However, Bcc and Lcc instructions can evaluate flag before reflecting to PSW.



"nn=20 to 35" is only for AM31/AM32. It can not be used for AM30.



MOVM [regs],(SP) can not be located in one instruction before this instruction.

Notes to Programmers

The MN1030/MN103S Series of 32-bit microcontrollers incorporates the following enhancements for boosting throughput.

· Lower cycle counts

Additional hardware bypasses and augments the five-stage pipeline to increase the execution speeds of such instructions as Lcc, SETLB, RET, and RETF.

· Higher operating frequencies

Reorganizing the pipeline stages to eliminate the bottlenecks associated with such operations as aligning and expanding load data has permitted the use of shorter clock cycles.

To help your programs to take maximum advantage of these throughput enhancements, this Chapter describes the pipeline architecture, dangerous code sequences, code sequences to avoid, and boiler plate code sequences.

1. Pipeline Architecture

This Section covers the structure and operation of the five-stage pipeline incorporated into the MN1030/MN103S Series of 32-bit microcontrollers.

2. Dangerous Code Sequences

This Section describes instruction variants and instruction sequences that must be strictly avoided because they lead to faulty operation.

3. Code Sequences to Avoid

This Section describes instruction variants and instruction sequences that should be avoided not because they lead to faulty operation, but because they consume excess cycles.

4. Boiler Plate Code Sequences

This Section contains sample code for common programming tasks.

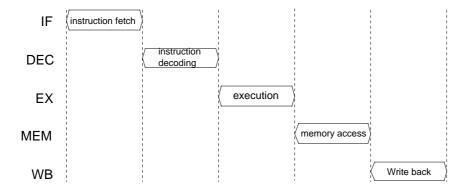
1

Pipeline Architecture

The 32-bit microcontrollers in the MN1030/MN103S Series boost throughput with a five-stage pipeline that, by overlapping instruction processing steps with stages operating in parallel, appears to execute a new instruction each machine cycle.

1.1 Pipeline Operation

The MN1030/MN103S pipeline has five stages.



Instruction fetch (IF): This stage reads in the instruction from memory

Instruction decoding (DEC): This stage decodes the instruction. For some branch instructions, it

also calculates the target address.

Execution (EX): This stage performs the calculation or calculates the target address for

the decoded instruction.

Memory access (MEM): This stage accesses memory and updates PSW flags, if required by the

instruction.

Write-back (WB): This stage stores the calculation result in a register. If the instruction

reads in data from memory, this stage aligns it, extends it, and stores the

result in a register.

The instruction fetch and instruction decoding stages access an instruction queue preloaded with instructions from memory. The instruction decoding stage does not start until this queue contains enough data to decode and execute the instruction. If the queue is empty (immediately after a branch, for example) or does not have all the bytes of an absolute address (abs) or immediate value (imm), the instruction decoding stage must wait at least one cycle.

Instruction queue operation can normally be safely ignored by the programmer because the hardware automatically controls it. Calculating code execution times, however, requires careful consideration of its operation.



For the AM31 and AM32 cores, accessing cachable memory takes both the memory access and write-back stages. The write-back stage then aligns the data, extends it, and stores the result in a register.

1.2 Register-to-Register (RR) Operations

For register-to-register operations, the pipeline stages perform the following operations.

DEC: This stage decodes the instruction.

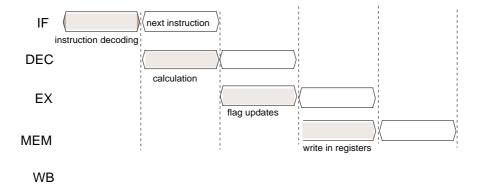
EX: This stage performs the calculation for the decoded instruction.

MEM: If the instruction updates PSW flags, this stage does so based on the result of the preceding

stage.

WB: This stage stores the result in a register.

The instructions in this group include addition, subtraction, logical operations, and shifts.



1.3 Data Load Operations

For data load operations, the pipeline stages perform the following operations.

DEC: This stage decodes the instruction.

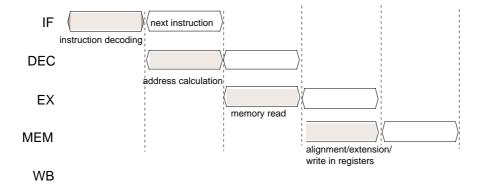
EX: This stage calculates the load address and determines the corresponding address space.

 $MEM: \quad \ This \ stage \ loads \ the \ data \ from \ memory \ or, \ if \ the \ memory \ is \ cachable, initiates \ cache \ access.$

WB: This stage aligns the data, extends it, and stores the result in a register. If the memory is

cachable, this stage reads the data from the cache before performing this operation.

The instructions in this group include data transfers from memory into a register.



1.4 Data Store Operations

For data store operations, the pipeline stages perform the following operations.

DEC: This stage decodes the instruction.

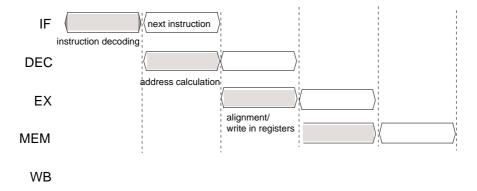
EX: This stage calculates the store address and determines the corresponding address space.

MEM: This stage aligns the data and writes it to memory or, if the memory is cachable, initiates

cache access.

WB: This stage does nothing or, if the memory is cachable, writes the data to the cache.

The instructions in this group include data transfers from a register to memory.



1.5 Branching Operations

For high-speed branching operations, the pipeline stages perform the following operations.

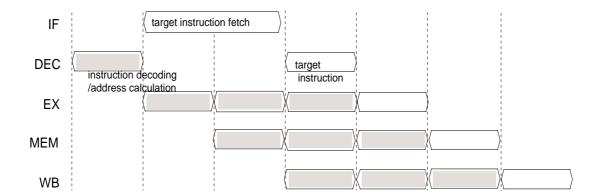
DEC: This stage decodes the instruction and calculates the jump target address for use in fetching the instruction there during the next machine cycle.

EX: This stage does nothing.

MEM: This stage does nothing.

WB: This stage does nothing.

The instructions in this group include conditional branches.



For normal branching operations, the pipeline stages perform the following operations.

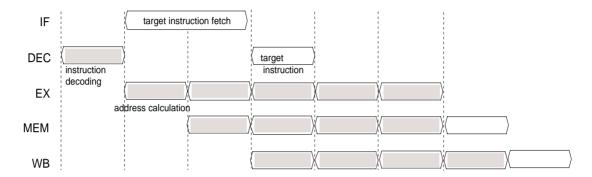
DEC: This stage decodes the instruction.

EX: This stage calculates the jump target address for use in fetching the instruction there during

the next machine cycle.

MEM: This stage does nothing.
WB: This stage does nothing.

The instructions in this group include unconditional branches using register indirect addressing.



1.6 Complex Instructions

Complex instructions always require multiple cycles for the execution, memory access, and write-back stages to complete. The pipeline stages perform the following operations.

DEC: This stage decodes the instruction.

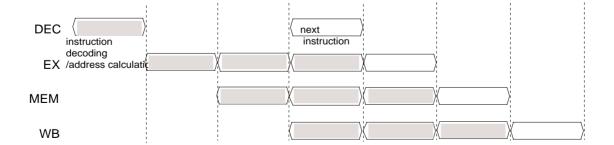
EX: This stage performs the calculation for the decoded instruction and calculates the memory access address or jump target for use in fetching the instruction there during the next

machine cycle.

WB: This stage stores the calculation result in a register or aligns the data read in from memory,

extends it, and stores the result in a register. For some cycles, it can be idle.

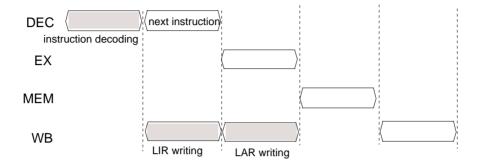
The instructions in this group include bit manipulations.



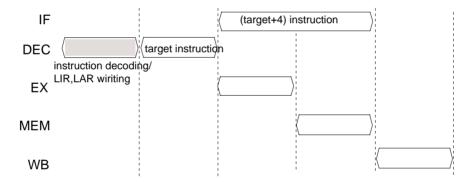
1.7 Special Instructions

Certain instructions—Lcc, SETLB, RET, and RETF, for example—reorder the later pipeline stages or bypass them with additional hardware to increase the execution speed.

Pipeline operation for SETLB



Pipeline operation for Lcc



1.8 Pipeline Stall

A pipeline stall is any situation interfering with the lockstep execution of the five pipeline stages as described above—an instruction whose execution is delayed for hardware reasons or one which cannot begin execution until a preceding instruction has completed execution.

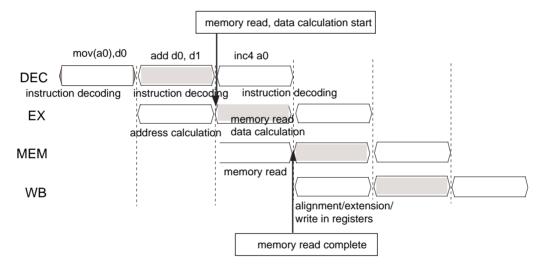
The following examples illustrate such situations arising with load instructions.

Data loaded required by next instruction Source code.

[Example]

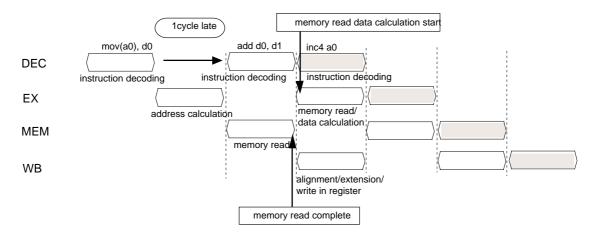
[Theoretical pipeline operation]							
mov	(a0), d0						
add	d0, d1						
inc4	a0						

[IMPOSSIBLE pipeline operation]



In this example, the ADD instruction immediately following the MOV instruction requires the data from data register D0. This arrangement does not execute as expected because the ADD instruction's execution stage is simultaneous with the MOV instruction's memory access stage, and the data from the latter is not available until one cycle later. As a result, the ADD instruction would use the old contents of D0—not what was intended.

[Actual pipeline operation]

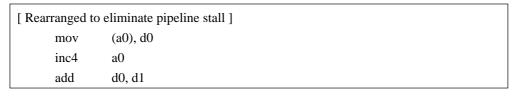


To ensure proper operation, therefore, the hardware inserts a 1-cycle delay so that the ADD instruction does not access D0 until the preceding MOV instruction has finished loading it from memory.

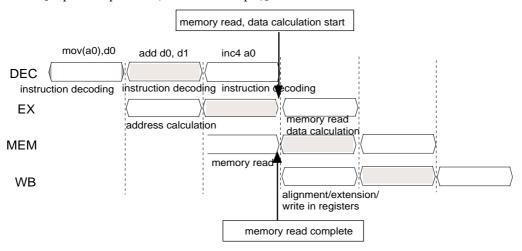
This type of gap in pipeline operation is called a pipeline stall.

Pipeline stalls reduce throughput and thus execution speed. One way to eliminate such pipeline stalls is to separate the two instructions with another—by moving in INC4, in our example. The resulting delay allows the MOV instruction to load D0 from memory by the time that the ADD instruction accesses that register, so the pipeline does not stall. It goes without saying, of course, that the inserted instruction (INC4 here) must not trigger a pipeline stall with the MOV instruction.

[Example]



[Pipleline operation (for above example)]



2

Dangerous Code Sequences

The following table lists instruction variants and instruction sequences that must be strictly avoided because they lead to faulty operation.

J 1					
Leading instruction	Following instruction	Relative position	Recommend	Section	Cores
Load/store instruction			The result of the	2-1	AM30
(d8,An), (d16,An),(d32,An),			address calculation		AM31
(d8,SP), (16,SP),(d32, SP),			must be in the		
MOV/MOVBU/MOVB/			same memory		
MOVHU/MOVH with			address space as		
addressing mode of (Di, An)			the address.		
E/IM writing instruction			Instructions		AM30
AND imm16,PSW,			writing to IE and	2-2	AM31
OR imm16,			IM bits (after 2		AM32
PSW, MOV Dm, PSW			cycles)		
Flag update instruction	Flag reference	following	required: more	2-3	AM30
CLR,ADD(except imm,SP)	ADDC Dm,Dn,	lollowing	than 1 cycle		AM31
ADDC,SUB,SUBC,MULU,	SUBC Dm,Dn		-		AM32
DIV,DIVU,INC Dn,CMP,	ROR Dm,				
AND(except imm16,PSW),	ROL Dn,				
OR(except imm16,PSW),	Users' optional				
XOR, NOT, BTST, BSET,	for flag refer-				
BCLR, ASR, LSR, ASL,	ence*1				
ASL2, ROR, ROL,	ence 1				
Users' defined updating flags					
Flag writing instruction	Flag reference		required: more		
AND imm16, PSW,		following	than 2 cycles	2-4	AM30
OR imm16, PSW,	ĺ ,	following			AM31
MOV Dm, PSW	SUBC Dm,Dn				AM32
MOV Dill, 13 W	ROR Dn				1111102
	ROL Dn Users' defned				
	for flag reference*				
A0 writing instruction		c 11 ·	required: more	2-5	AM30
MOV with A0 writing	MUL/MULU	following	than 2 cycles		111130
ADD writing result in A0					
SUB writing result in A0,					
INC, A0, INC4 A0					
CALLS label, JSR label			CALLS(d16,PC),	2-6	AM31
			JSR(d16,PC) is not		7111131
			available.		
MOVM [regs], (SP)		£-11:	required: more	2-7	AM31
	users' optional	following	than 1 cycle	/	7111131
SSET/BCLR			bus unlock:	2-8	AM31
			cachable region /	20	AM32
		1			

^{*} Availability varies with model.

The descriptions in this section have the following components.

[Description]

These portions describe the problems and include pipeline operation diagrams.

[Icons]



Warning: This icon indicates warnings that must always be observed.







These indicate the applicable microcontroller cores.

[Example]

These portions illustrate the problems with specific coding examples—both problematic versions that do not execute as expected and revised versions guaranteed to execute properly.

[Applicable Instructions]

These portions list the instruction variants and instruction sequences exhibiting the problem.



The coding examples labeled "problematic" below do not necessarily always fail to yield the expected results. If two instructions, A and B, must always be separated by at least one cycle, they will still execute as expected together if memory location, interrupts, and other factors conspire to delay the supply of instruction B to the pipeline. For "worst case" reliability, however, always use the revised version instead.



For ease of clarification, the coding examples below use NOPs to separate instructions, but these are not the only candidates. Any instruction in the vicinity that can be moved into the gap without changing program logic or presenting ordering problems of its own makes a better candidate.

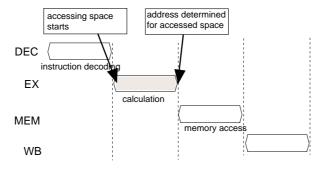
2.1 Load/Store Instructions



[Description]

To speed memory access, load/store instructions assume that the memory space accessed is that of the base address, so determine it from that in an operation that runs parallel to the address calculation. If the result is in a different memory space, however, faulty operation results. To avoid this problem, always make sure that the base address and the result are in the same memory space.





[Example]

Problematic Version

mov 0x20, a0

mov (0x80000000,a0), d0

1		base address		address calcul	ation result
		a0=0x00000020		a0+0x800000	0
				=0x800	0020
	ROM/RAM/Flash memory	internal data RAM space		external mem	ory space
╅	with Cache	cachable region for data		uncachable re	gion
			1:00		

This code does not execute as expected because the base address and the result are in different memory spaces.

Revised Version

mov 0x80000000, a0 mov (0x20, a0), d0

	base address a0=0x00000020	address calculation result a0+0x8000000
		=0x8000020
ROM/RAM/Flash memory	internal data RAM space	external memory space
with Cache	cachable region for data	uncachable region

same—

This code executes without problem because the base address and the result are in the same memory space.

[Applicable Instructions]

MOV, MOVBU, MOVB, MOVHU, and MOVH with the following addressing modes

Addressing Modes	Base Address
(d8,An)/(d16,An)/(d32,An)	An
(d8,SP)/(d16,SP)/(d32,SP)	SP
(Di,An)	An

2.2 Instructions Writing to IE and IM Bits

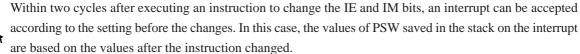


[Description]

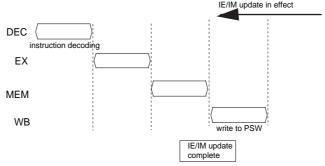
Notos

The use of a pipeline means that instructions writing to the interrupt enable (IE) and interrupt mask (IM) bits in the Processor Status Word (PSW) do not take effect until the write-back stage, two cycles after the execution stage. To put it another way, these instruction must precede by two cycles the code segment where such changes—enabling or disabling interrupts, for example—are to take effect.









[Example]

[
Problematic Version			
and	0xf7ff,psw		
mov	(a0),d0	or	
and	0xf7ff,psw		
nop			
mov	(a0),do		

Although the intention is to disable interrupts, because of the delay in updating the IE bit in the PSW, interrupts are still enabled when the MOV instruction starts executing.

Revised Version		
and	0xf7ff,psw	
nop		
nop		
mov	(a0),do	

Here the MOV instruction can never be pre-empted by an interrupt.

[Applicable Instructions]

AND imm16,PSW, OR imm16,PSW, MOV Dm,PSW



2.3 Sequences Updating and Referencing PSW Flags

Notes

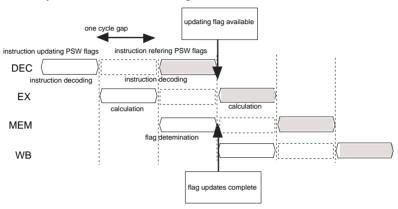
[Description]



The use of a pipeline means that instructions updating PSW flags do not take effect until the memory access stage, one cycle after the execution stage. There must, therefore, be a gap of at least one cycle before any instructions can correctly reference those PSW flags.







[Example]

[Problematic Version]
add d0,d1
addc d2,d3



Because the ADDC instruction starts executing before the ADD instruction has updated the carry flag, it uses the old contents of the carry flag.

[Revised Version]

add d0,d1

nop

addc d2,d3

The ADDC instruction uses the properly updated contents of the carry flag.

[Applicable Instructions]

<Instructions setting PSW flags> CLR; ADDC; SUB; SUBC; MUL; MULU; DIV; DIVU; CMP;

XOR; NOT; BTST; BSET; BCLR; ASR; LSR; ASL; ASL2; ROR; ROL; ADD (except ADD imm, SP) AND (except AND imm16, PSW) INC Dn OR (except OR imm16, PSW) User defined

instructions setting PSW flags*

<Instructions referencing PSW flags> User defined instructions referencing PSW flags*

(Availability varies with model.)

2.4 Sequences Writing to and Referencing PSW Flags



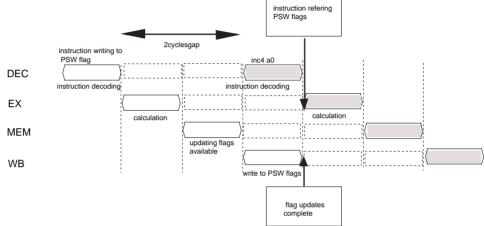
[Description]

The use of a pipeline means that instructions writing to PSW flags do not take effect until the write-back stage, two cycles after the execution stage. There must, therefore, be a gap of at least two cycles before any instructions can correctly reference those PSW flags.

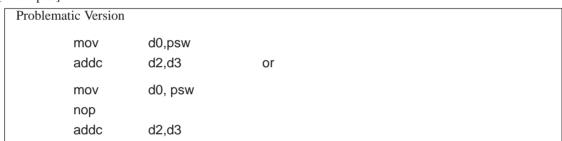








[Example]





Because the ADDC instruction starts executing before the MOV instruction has updated the carry flag, it uses the old contents of the carry flag.

[Revised Version]	
mov	d0,psw
nop	
nop	
addc	d2,d3

The ADDC instruction uses the properly updated contents of the carry flag.

[Applicable Instructions]

<Instructions writing to PSW flags> <Instructions referencing PSW flags>

AND imm16, PSW; OR imm16, PSW; MOV Dm, PSW User defined instructions referencing PSW flags*

(* Availability varies with model.)

2.5 MUL/MULU after Write to A0



[Description]

Notes

To boost execution speed, the AM30 versions of the MUL and MULU instructions modify pipeline operation in a way that interacts with the address register A0. As a result, there must be a gap of at least two cycles after any instruction that writes to A0 before these two instructions. Example



[Example]

[Problematic Version]			
mov	0x12345678,a0		
mul	d0,d1	or	
mov	0x12345678,a0		
nop			
mul	d0,d1		

Assembling the source file (program.asm) containing the above code with the -mno-nopmul command line option produces object code reflecting the source code exactly as written—

as103 -mno-popmul program.asm

that is, no additional NOPs. This object code, however, does not always perform as intended.

	3
[Revised Version]	
mov	0x12345678,a0
nop	
nop	
mul	d0,d1

Inserting the necessary NOPs or assembling the source file with the -mnopmul command line option produces object code with the necessary two-cycle gap and thus guaranteed to execute properly.

as103 -mnopmul program.asm

specifying these options is to insert two NOPs between Instruction writing to Ao and MUL/NULU and generate the object files.



The assembler defaults to the -mnopmul command line option.

[Applicable Instructions]

<Instruction writing to A0>

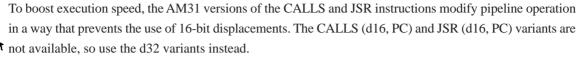
all MOV, ADD, and SUB instructions writing to A0; INC A0; INC4 A0

2.6 Displacements with CALLS and JSR (AM31 Only)



[Description]

Notes



The command line option for disabling the use of such 16-bit displacements is -mlongcalls.

[Example]

Assembling the source file (program.asm) containing the above code with the -mlongcalls command line option produces object code reflecting the source code exactly as written—

as103 -mlongcalls program.asm

specifying these option is to generate the object files as CALL(d32,PC) or JSR(d32,PC) without using CALLS(d16,PC) ir JSR(d16,PC)

[Applicable Instructions] CALLS (d16,PC), JSR (d16,PC)

(!

2.7 User Defined Instructions after MOVM [regs], (SP) (AM31 Only)

[Description]

Notes



To boost execution speed, the AM31 version of MOVM [regs], (SP) modifies pipeline operation in a way that introduces a structural dependency with immediately following user defined instructions (UDFnn and UDFUnn). As a result, there must be a gap of at least one cycle between the two.

[Example]

[Problematic Version]

movm [regs], (SP) udf00 d0, d1

Assembling the source file (program.asm) containing the above code with the -mno-nopmovm command line option produces object code reflecting the source code exactly as written—

as103 -mno-nopmovm program.asm

that is, no intervening NOP. This object code, however, does not always perform as intended.

[Revised Version]

movm [regs], (SP)

nop

udf00 d0, d1

Inserting the necessary NOP or assembling the source file with the -mnopmovm command line option produces object code with the necessary one-cycle gap and thus guaranteed to execute properly.

as103 -mnopmovm program.asm

Specifying these options is to insert one NOP between MOVM[regs],(SP) and UDFnn/UDFUnn and generates the object files.



The assembler defaults to the -mnopmovm command line option.

[Applicable Instructions]

<Leading instruction> MOVM[regs], (SP)
<User defined instructions> (UDFnn and UDFUnn)*

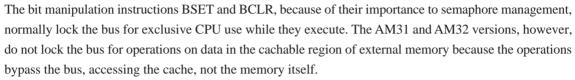
(* Availability varies with model.)

2.8 BSET and BCLR with Cachable External Memory



[Description]

Notes





[Applicable Instructions]

BSET, BCLR

Code Sequences to Avoid

The following table lists instruction variants and instruction sequences that should be strictly avoided because they lead to pipeline stalls and thus lower throughput.

Leading instruction	Following instruction	Relativeposition	Recommend	Section	Cores
Instructions needs	-	-	allocate in cachable region	3-1	AM30/
to execute with			of locked external memory		AM31
high-speed			space		AM32
Loard/Store	-	-	The result of the address	3-2	AM32
(d8,An),(d16,An),			calculation should be in the	52	
(d32,An), (d8,SP)			same memory address		
(d16,SP),(d32,SP),			space as the address in the		
MOV/MOVB/			base register.		
MOVB/MOVHU/			ouse registeri		
MOVH with					
addressing mode of					
(Di,An)					
Lcc	all instructions	target(branch)	Arrange short instructions	3-3(1)	AM30/
			with low per-cycle		AM31
			consumption rates at		AM32
			branch target (start of	3-3(2)	AM30/
target(branch			loop) at 8n-4+m for small		AM31AM32
except Lcc			values of m.		
Bcc and Lcc		following	Arrange short instructions	3-3(3)	AM30/
Dec and Lee			with low per-cycle con-		AM31
			sumption rates at branch tar-		AM32
			get at 8n+m for small val-		
			ues of m.		
all except target			Arrange short instructions	3-3(4)	AM30/
(branch)			with low per-cycle		AM31
(branch)			consumption rates		AM32
			immediately after branch		
			instruction.		
Loard	instructions using	following	insert 2 cycles		Am31/
(DCBYPS=0 in	loaded data, without			3-4(1)	AM32
cachable region of	imm32/d32/abs32				
the external memory	instructions using		insert 1 cycle	3-4(2)	Am31/
space)	loaded data, with				AM32
space)	d32				
	instructions storing		insert 1 cycle	3-4(3)	Am31/
	loaded data without				AM32
	d32/abs32				

Leading instruction	Following instruction	Relativeposition	Recommend	Section	Cores
Loard	instructions using word data	following		3-4(4)	AM30/AM31
(DCBYPS=1 in	loaded by LDUSE bit=1		insert 2 cycles		AM32
cachable region of	, without imm32/d32/abs32				
the extenal memory	instructions using word data		1	3-4(5)	
space, uncachable	loaded by LDUSE bit=1, with		insert 1 cycle		AM30/AM31
region of the extenal	d32				AM32
memory,	instructions using word data			3-4(6)	AM30/AM31
ROM space,	loaded,by LDUSE bit=0, without				AM32
RAM space,	imm32/d32/abs32		insert 1 cycle		
I/O space, or	instructions using byte/half-byte			3-4(7)	AM30/AM31
uncachable in the	data loaded without imm32/d32/				AM32
external memory	abs32		insert 2 cycles		
space)	instructions using byte/half-byte		1	3-4(8)	AM30/AM31
	data loaded with d32		insert 1 cycle		AM32
	instructions storing loaded data,				
	withd32/abs32		insert 1 cycle	3-4(9)	AM30/AM31
					AM32
DIV/DIVU	instructionsusing contents of Dn	following			
(dividend equals as 0)	for leading DIV/DIVU, without		insert 2 cycle	3-5(1)	AM30/AM31
	imm32 d32/abs32				AM32
	instructions using contents of Dn				
	for leading DIV/DIVU		insert 1 cycle	3-5(2)	AM30/AM31
	instructions storing contents of Dn				AM32
	for leading DIV/DIVU, without		insert 1 cycle	3-5(3)	AM30/AM31
	imm32/d32/abs32				AM32
SETLB	Lcc	following	insert 3 cycles		AM30/AM31
			-	3-6(1)	AM32
MOVM(SP),regs		following	insert 3 cycles	3-6(2)	AM30/AM31
RET/RETF		return			AM32
MOVM(SP), regs	SETLB	following	insert 1 cycle	3-7	AM30/AM31
RET/RETF		return			AM32
MOV Dm,MDR	RETF	following	insert 3 cycles	3-8(1)	AM30/AM31
EXT, MUL/MULU					AM32
DIV/DIVU		following	insert 2 cycles	3-8(2)	AM30/AM31
CALL/CALLS		target (branch)			AM32
MOVM(SP),regs		following	insert 3 cycles	3-8(3)	AM30/AM31
RET/RETF		return			AM32
CALL/CALLS	MOV MDR,Dn	target(branch)	insert 2 cycles	3-9	AM30/AM31
	DIV/DIVU				AM32

The descriptions in this section have the following components.

[Description]

These portions describe the recommendations—with pipeline operation diagrams as necessary.

[Icons]



High-speed: This icon indicates a recommendation for boosting throughput by avoiding pipeline stalls.

High-speed







These indicate the applicable microcontroller cores.

[Example]

These portions illustrate the recommendations with specific coding examples—both problematic versions that produce pipeline stalls and revised versions that run faster by avoiding pipeline stalls.

[Applicable Instructions]

These portions list the instruction variants and instruction sequences covered by the recommendation.

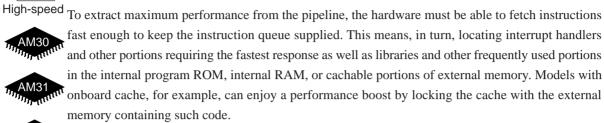


For ease of clarification, the coding examples below avoid pipeline stalls by moving INC instructions, but these are not the only candidates. Any instruction in the vicinity that can be moved into the gap without changing program logic or presenting ordering problems of its own makes a suitable candidate.

3.1 Time-Critical Code



[Description]





[Example]

_TEXT		CODE, PUBLIC,1
300_101	mov	0,d0
rts		
end		

The following command line relocates the machine code for the above source code (program.asm) to boost its execution speed.

> as103-o program.rf program.asm ld103-T@CODE=50000000 -o program.ex program.rf

The above command line generates an executable file (program.ex) with the corresponding machine code starting at address 0x50000000. For a model with onboard ROM, RAM, and Flash memory, the best choice of address is within the internal program ROM; for a model with onboard cache, in the cachable portions of external memory.

3.2 Load/Store Instructions

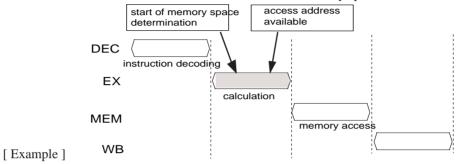


[Description]

High-speed



To speed memory access, load/store instructions assume that the memory space accessed is that of the base address, so determine it from that in an operation that runs parallel to the address calculation. If the result is in a different memory space, however, the hardware must repeat the memory space determination, a step that takes an additional cycle. For maximum throughput, always make sure that the base address and the result are in the same memory space.



[Problematic Version]	
mov	0x20,a0
mov	(0x80000000,a0),d0

	base address		address calculation result	
	a0=0x00000020		a0+0x8000020	
ROM/RAM/Flash memory	internal data RAM space		external memory space	
with Cache	cachable region for data		uncachable re	gion
		dif	ferent ——	

This code executes slower because the base address and the result are in different memory spaces.

[Revised Version]		
mov	0x80000000,a0	
mov	(0x20,a0).d0	

	base address		address calcula	ation result
	a0=0x80000000		0x20+a0=0x80000020	
ROM/RAM/Flash memory	internal data RAM space		external memory space	
with Cache	cachable region for data		uncachable region	

This code executes at maximum speed because the base address and the result are in the same

same

[Applicable Instructions]

memory space.

MOV, MOVBU, MOVB, MOVHU, and MOVH with the following addressing modes

Addressing mode	base address
(d8,An)/(d16,An)/(d32,An)	An
(d8,SP)/(d16,SP)/(d32,SP)	SP
(Di,An)	An

3.3 Instructions Following Branch and Other Instructions



(1) Lcc Branch Targets

High-speed





Because the instruction fetch hardware uses 8-byte alignment and the Loop Instruction Register (LIR) holds the first four bytes of the loop, arranging short instructions with low per-cycle consumption rates at the branch target (start of loop) at addresses 8n-4+m for small values of m produces higher throughput.



[Example]



align	8	
mov	0x00,a0	
clr	d0	
setlb		
mov	d0,(a0)	:instruction at address 8n-4
mov	d0,(0x10,a0)	
mov	d0,(0x20,a0)	:instruction at address 8n
mov	d0,(ox30,a0)	
inc4	a0	
cmp	0x10,a0	
llt		

Instruction at address 8n The above example shows a branch target at an address four bytes before a fetch boundary (8n-4). The SETLB instruction therefore loads the Loop Instruction Register (LIR) with the two instructions MOV D0, (A0) and MOV D0, (0x10,A0) and the Loop Address Register (LAR) with the boundary address (8n). When the Lcc instruction (LLT in this example) starts executing these instructions, it also simultaneously triggers a fetch of a full eight bytes, ensuring an ample supply of instructions and thus preventing a pipeline stall—for models using onboard ROM for instructions and onboard RAM for data and onboard cache models with no hits, anyway.

[Applicable Instructions]

<Leading instruction> Lcc

<Target(branch)> any instructions



(2) Branch Targets Other than Lcc

[Description]



Because the instruction fetch hardware uses 8-byte alignment, arranging short instructions with low per-cycle consumption rates at the branch target at addresses 8n+m for small values of m produces higher throughput.



[Example]



[Example]			
	align	8	
	nop		
	mov	0x00,a0	
	clr	d0	
LABEL	mov	d0,(a0)	:instruction at address 8n
	mov	d0,(0x10,a0)	
	mov	d0,(0x20,a0)	
	mov	d0,(0x30,a0)	
	inc4	a0	
	cmp	0x10,a0	
	bit	LABEL	

The above example shows a branch target at a fetch boundary (8n). When the branch instruction (BLT in this example) starts executing these instructions, it triggers a fetch of a full eight bytes, ensuring an ample supply of instructions and thus preventing a pipeline stall—for models using onboard ROM for instructions and onboard RAM for data and onboard cache models with no hits, anyway.

[Applicable Instructions]

<Leading instruction> Bcc, JMP, CALL, CALLS, RET, RETF, RETS, RTI, TRAP

<Target(branch)> Any instructions



(3) Instructions Following Bcc or Lcc

[Description]



Arranging short instructions with low per-cycle consumption rates after a conditional branch instruction maximizes supply, producing higher throughput.







blt	LABEL	
mul	d0,d1	Two-byte instruction requiring multiple cycles to execute



The above example shows a MUL instruction executed when the branch instruction (BLT in this example) fails. Because the instruction requires multiple cycles, the core has time to fetch more instruction bytes, ensuring a better supply and thus higher throughput.

[Applicable Instructions]

< Leading instruction> Bcc or Lcc <Target(branch)> Any instructions



(4) Instructions Following Non-Branch Instructions

[Description]



Grouping long instructions together risks having instruction queue consumption outstrip supply. For higher throughput, try to even out the consumption rate.



[Example]



Matter.	mo
	inc
AM32	ado
-7)-(C-	

m	IOV	0x12345678,d0	six-byte instruction
m	IOV	(a0),d1	one byte instruction
in	С	d0	one byte instruction
ac	dd	d0,d1	one byte instruction
m	IOV	0x9abcdef0,d2	six-byte instruction

The above example shows such a separation. This arrangement evens out the consumption rate, preventing pipeline stalls with the longer instructions and thus delivering higher throughput.

[Applicable Instructions]

<Leading instruction> Any instruction except Bcc, Lcc, JMP, CALL, CALLS, RET, RETF,

RETS, RTI, TRAP

<Target(branch)

Any instructions

3.4 Instructions Following Load Instructions

Load instructions normally require two cycles (memory access and write-back stages) to retrieve the data from memory, so closely following instructions requiring that data for operands can stall the pipeline.

Load instructions accessing cachable external memory initiate the cache access during the memory access stage and retrieve the data from the cache during the write-back stage. Those accessing other memory types (uncachable external memory, internal ROM, internal RAM, internal I/O region, or external memory for models without onboard cache) retrieve the data from memory during the memory access stage and align, extend, and write that data to a register during the write-back stage.

Setting the DCBYPS bit in the core's Memory Control Register (MEMCTRC) to "1" can, for models with operating frequencies low enough, shorten the cache initiate and retrieve process to a single cycle in certain circumstances. For further details, refer to the documentation for the particular device.

Setting the LDUSE bit in MEMCTRC to "0" makes a word (imm32, d32, or abs32) available for a calculation (address or otherwise) in following instructions after a single cycle for the following memory types (abbreviated to "any" in the rest of this section): cachable external memory with DCBYPS = 1, uncachable external memory, internal ROM, internal RAM, internal I/O region, and external memory for models without onboard cache—in other words, all types except cachable external memory with DCBYPS = 0. For further details, refer to the documentation for the particular device.



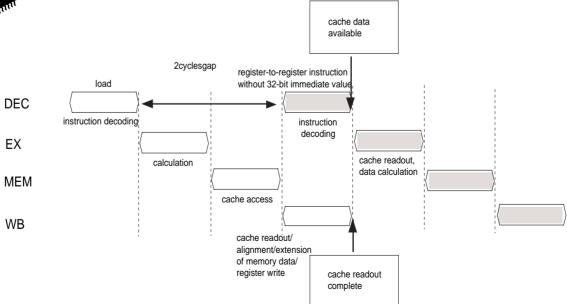
(1) Cachable Memory, DCBYPS = 0, Follower without imm32/d32/abs32

[Description]



To prevent pipeline stalls, we recommend inserting at least two cycles when the DCBYPS bit is "0," the leading instruction accesses cachable external memory, and the following instruction uses the loaded data other than as imm32, d32, or abs32 in its calculations.





[Example]

Problematic Ve	ersion	
inc	a2	
mov	(a0),d0	: Load instruction
inc	a3	
add	d1,d0	: Instruction using loaded data other than as imm32, d32 or abs32
		or
inc	a2	
inc	a3	
mov	(a0),d0	: Load instruction
add	d1,d0	I: nstruction using loaded data other than as imm32, d32 or abs32



Retrieving and aligning the data from the cache takes two cycles, so the pipeline stalls until the data loaded with the MOV instruction is available to the ADD instruction.

Revi	ised Version		
	mov	(a0),d0	: Load instruction
	inc	a2	
	inc	a3	
	add	d1,d0	: Instruction using loaded data other than as imm32, d32 or abs32

The data loaded with the MOV instruction is available to the ADD instruction, so the pipeline does not stall.

[Applicable Instructions]

<Leading instruction> All load variants of MOV, MOVBU, MOVB,* MOVHU, and MOVH*

(* The MOVB Mem, Reg and MOVH Mem, Reg variants require only a one-

cycle gap.)

<Following instruction> Any instruction using the loaded data other than as imm32, d32, or abs32 in its

calculations except the following; MOV PSW,Dn, MOV Dm,PSW, AND imm16,PSW, OR imm16,PSW, Bcc, Lcc, JMP, TRAP, NOP.



Here cachable external memory refers to the cachable portions of AM31 or AM32 external memory.



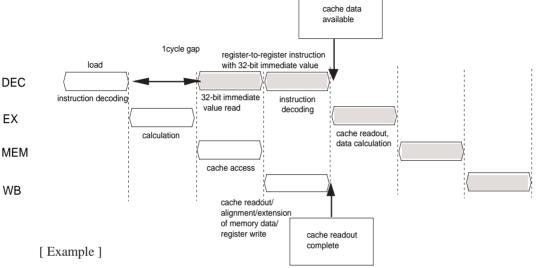
(2) Cachable Memory, DCBYPS = 0, Follower with d32

[Description]



To prevent pipeline stalls, we recommend inserting at least one cycle when the DCBYPS bit is "0," the leading instruction accesses cachable external memory, and the following instruction uses the data together with a 32-bit displacement [imm32?] operand.





[Problematic Version]	
inc	a1	
mov	(a0),d0	Load instruction
add	0x12345678,d0	Instruction using loaded data and 32-bit
		displacement d32 operand



Retrieving and aligning the data from the cache takes one cycle, so the pipeline stalls until the data loaded with the MOV instruction is available to the ADD instruction.

[Revised Version]		
mov	(a0),d0	Load instruction
inc	a1	
add	0x12345678,d0	Instruction using loaded data and 32-bit
		displacement d32 operand

The data loaded with the MOV instruction is available to the ADD instruction, so the pipeline does not stall.

[Applicable Instructions]

<Leading instruction> All load variants of MOV, MOVBU, and MOVHU

<Following instruction> MOV, MOVBU, and MOVHU variants with 32-bit operands (d32).



Here cachable external memory refers to the cachable portions of AM31 or AM32 external memory.



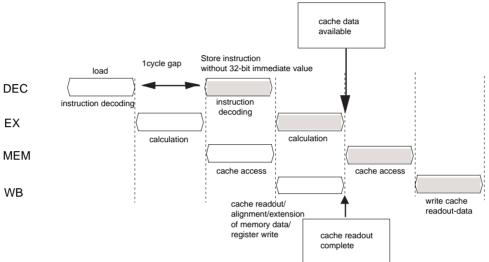
(3) Cachable Memory, DCBYPS = 0, Follower Storing without d32/abs32

[Description]



To prevent pipeline stalls, we recommend inserting at least one cycle when the DCBYPS bit is "0," the leading instruction accesses cachable external memory, and the following instruction stores the data without using a d32 or abs32 operand.





[Example]

[Problematic Version]		
inc	a1	
mov	(a0),d0	Load instruction
mov	d0,(a1)	Instruction storing loaded data without d32/abs32 operand



Retrieving and aligning the data from the cache takes one cycle, so the pipeline stalls until the data loaded with the first MOV instruction is available to the second MOV instruction.

[Revised Version]		
mov	(a0),d0	Load instruction
inc	a1	
mov	d0,(a1)	Instruction storing loaded data without d32/abs32 operand

The data loaded with the first MOV instruction is available to the second MOV instruction, so the pipeline does not stall.

[Applicable Instructions]

<Leading instruction> All load variants of MOV, MOVBU, and MOVHU

 $<\!\!Following\ instruction\!\!> MOV, MOVBU, MOVB, MOVHU, and MOVH\ store\ without\ d32/abs32$

operands



Here cachable external memory refers to the cachable portions of AM31 or AM32 external memory.



High-speed

(4) Other Memory, LDUSE = 1, Word Data, Follower without imm32/d32/abs32Here "other" means all memory types except cachable external memory with DCBYPS = 0.

[Description]

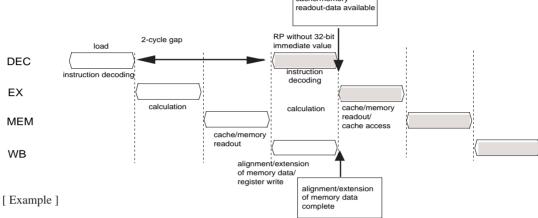


To prevent pipeline stalls, we recommend inserting at least two cycles when the LDUSE bit is "1," the leading instruction loads a word, and the following instruction uses the loaded data other than as imm32, d32, or abs32 in its calculations.

cache/memory







[Problematic Version]		
inc	a2	
mov	(a0),d0	Load instruction
inc	a3	
add	d1,d0	Instruction using loaded data other than as imm32, d32 or
		abs32
		or
inc	a2	
inc	a3	
mov	(a0),d0	Load instruction
add	d1,d0	Instruction using loaded data other than as imm32, d32 or
		abs32



Retrieving and aligning the data from the cache or memory takes two cycles, so the pipeline stalls until the data loaded with the MOV instruction is available to the ADD instruction.

[Revised Version]		
mov	(a0),d0	Load instruction
inc	a2	
inc	a3	
add	d1,d0	Instruction using loaded data other than as imm32, d32 or
		abs32

The data loaded with the MOV instruction is available to the ADD instruction, so the pipeline does not stall.

[Applicable Instructions]

<Leading instruction> All load variants of MOV

<Following instruction> Any instruction using the loaded data other than as imm32, d32, or abs32

in its calculations except the following; MOV PSW,Dn , MOV Dm,PSW,

AND imm16,PSW, Bcc, Lcc, JMP, TRAP, NOP.

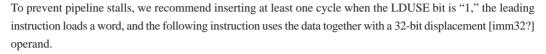


(5) Other Memory, LDUSE = 1, Word Data, Follower with d32

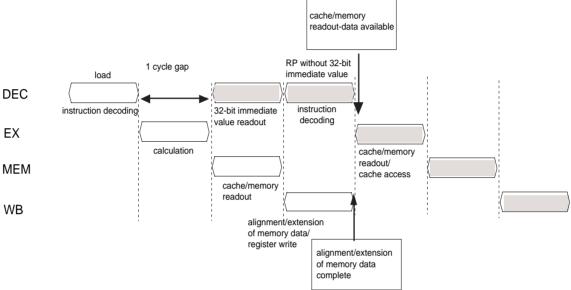
Here "other" means all memory types except cachable external memory with DCBYPS = 0.



[Description]







[Example]

[Problematic Version]		
inc	a1	
mov	(a0),d0	Load instruction
add	0x12345678,d0	Instruction using loaded data and 32-bit displacement
		operand



Retrieving and aligning the data from the cache or memory takes one cycle, so the pipeline stalls until the data loaded with the MOV instruction is available to the ADD instruction.

[Revised Version]		
mov	a(0),d0	Load instruction
inc	a1	
add	0x12345678,d0	Instructions using loaded data and 32-bit
		displacement operand

The data loaded with the MOV instruction is available to the ADD instruction, so the pipeline does not stall.

[Applicable Instructions]

<Leading instruction> All load variants of MOV

<Following instruction > MOV, MOVBU, and MOVHU variants with 32-bit displacement

operands, ADD imm32,Dn, OR imm32,Dn, XOR imm32,Dn, BTST imm32,Dn, UDFnn imm32,Dn, UDFUnn imm32,Dn.



(6) Other Memory, LDUSE = 0, Word Data, Follower without imm32/d32/abs32Here "other" means all memory types except cachable external memory with DCBYPS = 0.

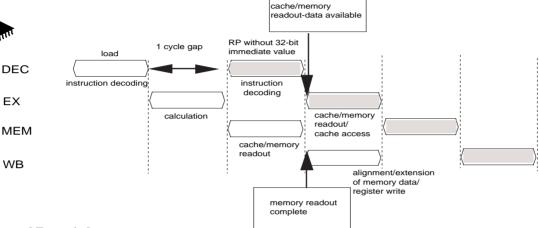
High-speed

[Description]



To prevent pipeline stalls, we recommend inserting at least one cycle when the LDUSE bit is "0," the leading instruction loads a word, and the following instruction uses the loaded data other than as imm32, d32, or abs32 in its calculations.





[Example]

[Problematic Version]			
inc	a1		
mov	(a0),d0	Load instruction	
add	d1,d0	Instruction using loaded data other than as imm32, d32	
		or abs32.	



Retrieving and aligning the data from the cache or memory takes two cycles, so the pipeline stalls until the data loaded with the MOV instruction is available to the ADD instruction.

[Revised Version]		
mov	(a0),d0	Load instruction
inc	a1	
add	d1,d0	Instruction using loaded data other than as imm32, d32
		or abs32.

The data loaded with the MOV instruction is available to the ADD instruction, so the pipeline does not stall.

[Applicable Instructions]

<Leading instruction> All load variants of MOV

<Following instruction> Any instruction using the loaded data other than as imm32, d32, or

abs32 in its calculations except the following; MOV PSW,Dn, MOV Dm,PSW, AND imm16,PSW, OR imm16,PSW, Bcc, Lcc, JMP, TRAP NOP.



High-speed

(7) Other Memory, Non-Word Data, Follower without imm32/d32/abs32 Here "other" means all memory types except cachable external memory with DCBYPS = 0.

[Description]



To prevent pipeline stalls, we recommend inserting at least two cycles when the leading instruction loads a byte or half-word, and the following instruction uses the loaded data oth



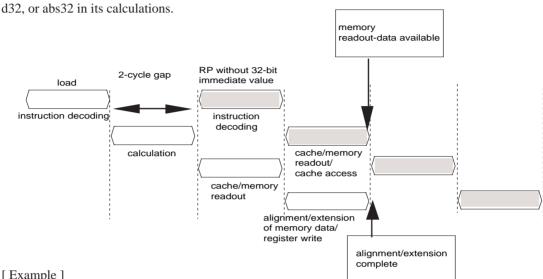


DEC

ΕX

MEM

WB



[Example]

[Problen	natic Version		
	inc	a1	
	movbu	(a0),d0	Load instruction
	inc	a2	
	add	d1,d0	Instruction using loaded data other than as imm32, d32
	or		
			abs32
			or
	inc	a1	
	inc	a2	
	movbu	(a0),d0	Load instruction
	add	d1,d0	Instruction using loaded data other than as imm32, d32
			or abs32



Retrieving and aligning the data from the cache or memory takes two cycles, so the pipeline stalls until the data loaded with the MOV instruction is available to the ADD instruction.

[Revised	Version	

movbu (a0),d0 Load instruction inc a1 inc a2 d1,d0 Instruction using loaded data other than as imm32, d32 or abs32. add

The data loaded with the MOV instruction is available to the ADD instruction, so the pipeline does not stall.

[Applicable Instructions]

<Leading instruction> All load variants of MOV, MOVBU, MOVB,* MOVHU, and

MOVH* (* The MOVB Mem, Reg and MOVH Mem, Reg variants

require only a one-cycle gap.)

<Following instruction> Any instruction using the loaded data other than as imm32, d32, or

abs32 in its calculations except the following; MOV PSW,Dn,

MOV Dm,PSW, AND imm16,PSW, OR imm16,PSW, Bcc, Lcc, JMP,

TRAP, NOP.



(8) Other Memory, Non-Word Data, Follower with d32

Here "other" means all memory types except cachable external memory with DCBYPS = 0.

Description]

ment [imm32?] operand.

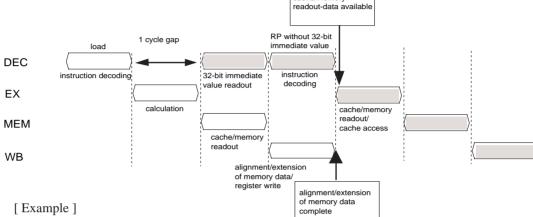


To prevent pipeline stalls, we recommend inserting at least one cycle when the leading instruction loads a byte or half-word, and the following instruction uses the data to -bit displace-

cache/memory







[Problematic Version]					
inc	a1				
movbu	(a0),d0	Load instruction			
add	0x12345678,d0	Instruction using loaded data and 32-bit			
		displacement operand			



Retrieving and aligning the data from the cache or memory takes one cycle, so the pipeline stalls until the data loaded with the MOVBU instruction is available to the ADD instruction.

[Revised Version]		
movbu	(a0),d0	Load instruction
inc	a1	
add	0x12345678,d0	Instruction using loaded data and 32-bit
		displacement operand

The data loaded with the MOVBU instruction is available to the ADD instruction, so the pipeline does not stall.

[Applicable Instructions]

<Leading instruction>

All load variants of MOVBU and MOVHU

<Following instruction>

MOVBU, MOVBU, and MOVHU variants with 32-bit displacement

operands; ADD imm32,Dn/An/SP, SUB imm32,Dn/An, CMP imm32,Dn/An, AND imm32,Dn, OR imm32,Dn, XOR imm32,Dn, BTST imm32,Dn, UDFnn imm32,Dn,

UDFUnn imm32,Dn



(9) Other Memory, Follower Storing without d32/abs32

Here "other" means all memory types except cachable external memory with DCBYPS = 0.

[Description]

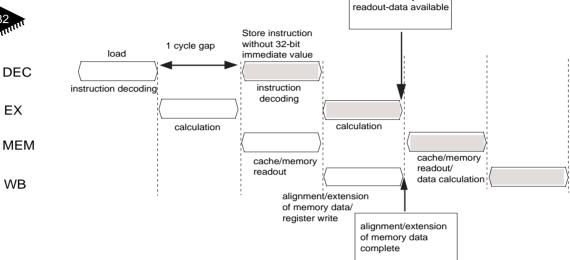


To prevent pipeline stalls, we recommend inserting at least one cycle when the following instruction stores the data without using a d32 or abs32 operand.

cache/memory







[Example]

[Problematic Version]	
inc a1	
movbu(a0),d0	Load instruction
mov d0,(a1)	Instruction storing loaded data without d32/abs32 operand



Retrieving and aligning the data from the cache or memory takes one cycle, so the pipeline stalls until the data loaded with the MOVBU instruction is available to the MOV instruction.

[Revised	Version]		
	movbu	(a0),d0	Load instruction
	inc	a1	
	mov	d0,(a1)	Instruction storing loaded data without d32/abs32 operand

The data loaded with the MOVBU instruction is available to the MOV instruction, so the pipeline does not stall.

[Applicable Instructions]

<Leading instruction> All load variants of MOVBU and MOVHU

<Following instruction> MOV, MOVBU, MOVB, MOVHU, and MOVH store without d32/

abs32 operands.

3.5 Instructions Following DIV/DIVU with Zero Dividend

The DIV and DIVU instructions minimize their execution times by minimizing the size of the dividend in bytes, but they still require a minimum of two cycles to store, during their write-back stage, the result in the destination data register (Dn)—even when the dividend, originally in Dn, is 0 and the result, stored in the same register, is the same.



(1) Follower without imm32/d32/abs32

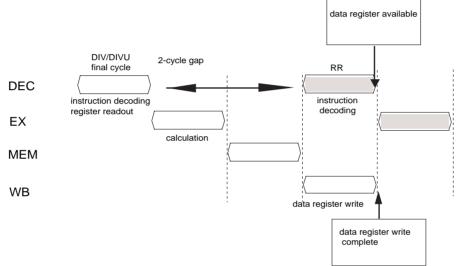
[Description]



To prevent pipeline stalls, we recommend inserting at least two cycles when the leading divide instruction has a zero dividend and the following instruction uses the result (Dn) other than as imm32, d32, or abs32 in its calculations.







[Example]

[Proble	ematic Version]	
	clr	d0	
	mov	d0,mdr	
	mov	0x05,d1	
	inc	a0	
	div	d1,d0	DIV/DIVU
	inc	a1	
	add	d1,d0	Instruction using result(dn) other than as imm32, d32 or abs32
			or
	clr	d0	
	mov	d0,mdr	
	mov	0x05,d1	
	inc	a0	
	inc	a0	
	inc	a1	
	div	d1,d0	DIV/DIVU
	add	d1,d0	Instruction using result(Dn) other than as imm32, d32 or
			abs32



Storing the result takes two cycles, so the pipeline stalls until the result of the divide instruction is available to the ADD instruction.

[Revised Version]		
clr	d0	
mov	d0,mdr	
mov	0x05,d1	
div	d1,do	DIV/DIVU
inc	a0	
inc	a1	
add	d1,d0	Instruction using result (Dn) other than as imm32, d32
		or abs32

The result of the divide instruction is available to the ADD instruction, so the pipeline does not stall.

[Applicable Instructions]

<Leading instruction> DIV/DIVU instruction with zero dividend

<Following instruction> Any instruction using the result (Dn) other than as imm32, d32, or abs32

in its calculations except the following; MOV PSW, Dn, MOV Dm,PSW, AND imm16,PSW, OR imm16,PSW, Bcc, Lcc, JMP, TRAP, NOP.

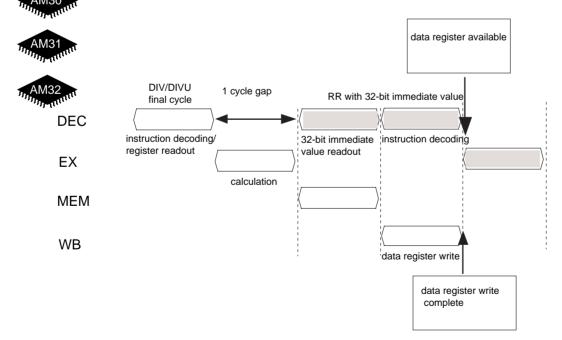


(2) Follower with d32

[Description]

High-speed

To prevent pipeline stalls, we recommend inserting at least one cycle when the leading divide instruction has a zero dividend and the following instruction uses the result (Dn) together with a 32-bit displacement [imm32?] operand.



[Example]

[Problematic Versi	ion]	
clr	d0	
mov	d0,mdr	
mov	0x05,d1	
inc	a0	
div	d1,d0	DIV/DIVU
add	0x12345678,d0	Instruction using result(Dn) and 32-bit
		displacement operand

V

Storing the result takes one cycle, so the pipeline stalls until the result of the divide instruction is available to the ADD instruction.

*			
[Revised	d Version]		
	clr	d0	
	mov	d0,mdr	
	mov	0x05,d1	
	div	d1,d0	DIV/DIVU
	inc	a0	
	add	0x12345678,d0	Instruction using result(dn) and 32-bit displacement
			operand

The result of the divide instruction is available to the ADD instruction, so the pipeline does not stall.

[Applicable Instructions]

<Leading instruction> DIV/DIVU instruction with zero dividend

<Following instruction> MOV, MOVBU, and MOVHU variants with 32-bit displacement

operands; ADD imm32,Dn/An/SP, SUB imm32,Dn/An, CMP imm32,Dn/An, AND imm32,Dn, OR imm32,Dn, XOR imm32,Dn, BTST imm32,Dn, UDFnn imm32,Dn,

UDFUnn imm32,Dn

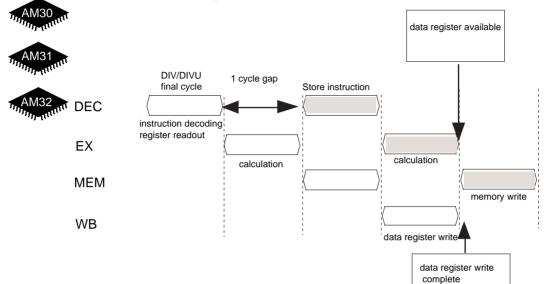
(3) Follower Storing without d32/abs32



High-speed

[Description]

To prevent pipeline stalls, we recommend inserting at least one cycle when the leading divide instruction has a zero dividend and the following instruction stores the result (Dn) without using an imm32, d32, or abs32 operand.



[Example]

[Problematic Versio	n]	
clr	d0	
mov	d0,mdr	
mov	0x05,d1	
inc	a0	
div	d1,d0	DIV/DIVU
mov	d0,(a1)	Instruction storing result(Dn) without d32 or abs32
		operand

Storing the result takes one cycle, so the pipeline stalls until the result (Dn) is available to the MOV instruction.

[Revised Version]		
clr	d0	
mov	d0,mdr	
mov	0x05,d1	
div	d1,d0	DIV/DIVU
inc	a0	
mov	d0,(a1)	Instruction storing result(Dn) without imm32, d32 or
		abs32 operand

[Applicable Instructions]

<Leading instruction>

DIV/DIVU instruction with zero dividend

<Following instruction>

Any instruction using the result (Dn) other than as imm32, d32, or abs32 in its calculations except the following; MOV PSW,Dn, MOV Dm,PSW, AND imm16,PSW, OR imm16,PSW, Bcc, Lcc, JMP, TRAP, NOP.

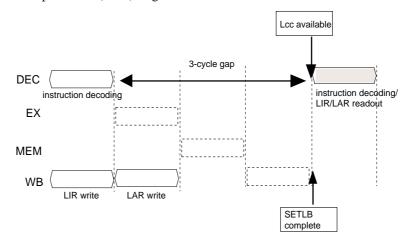
3.6 Instructions Preceding Lcc

The Lcc instruction bypasses the five-stage pipeline with additional hardware to increase the execution speed. To ensure proper interaction between the two, it must therefore wait for certain preceding instructions to complete.

(1) SETLB

[Description]

To ensure smooth pipeline operation, we recommend inserting at least three cycles between the Lcc instruction and the preceding SETLB instruction setting up the two registers that it must reference: Loop Instruction Register (LIR) and Loop Address (LAR) Register.



[Example]

Problematic Version	on]			
inc	d0			
setlb		SETLB		
inc	d1			
inc	d2			
lgt		Lcc	or	
inc	d0			
inc	d1			
setlb		SETLB		
inc	d2			
lgt		Lcc	or	
inc	d0			
inc	d1			
inc	d2			
setlb		SETLB		
lgt		Lcc		

Loading the LIR and LAR registers takes three cycles, so the pipeline stalls until their contents are available to the LGT instruction.

[Revised Version]			
setlb		SETLB	
ubc	d0		
inc	d1		
inc	d2		
lgt		Lcc	

The updated contents of the LIR and LAR registers are available to the LGT instruction, so the pipeline does not stall.

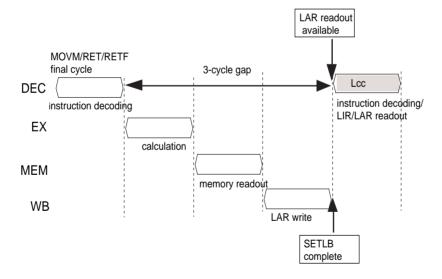
[Applicable Instructions]

<Leading instruction> SETLB <Following instruction> Lcc

(2) Other Instructions Modifying LIR and LAR

[Description]

To ensure smooth pipeline operation, we recommend inserting at least three cycles between the Lcc instruction and other preceding instructions (MOVM, RET, and RETF) modifying the two registers that it must reference: Loop Instruction Register (LIR) and Loop Address (LAR) Register.



[Example]

[Problematic Version]			
inc	d0		
movm	(sp),[other]	Instruction modifying LIR and LAF	?
inc	d1		
inc	d2		
lgt		Lcc	or
inc	d0		
inc	d1		
movm	(sp),[other]	Instruction modifying LIR and LAR	
inc	d2		
lgt	Lcc		or
inc	d0		
inc	d1		
inc	d2		
movm	(sp),pother]	Instruction modifying LIE and LAE	
lgt		Lcc	



Loading the LIR and LAR registers takes three cycles, so the pipeline stalls until their contents are available to the LGT instruction.

```
[ Revised Version ]

movm (sp),[other] Instruction modifying LIR and LAR

inc d0

inc d1

inc d2

lgt Lcc
```

The updated contents of the LIR and LAR registers are available to the LGT instruction, so the pipeline does not stall.

[Applicable Instructions]

<Leading instruction> MOVM (SP),regs, RET, RETF

<Following instruction> Lcc



High-speed

3.7 Instructions Preceding SETLB



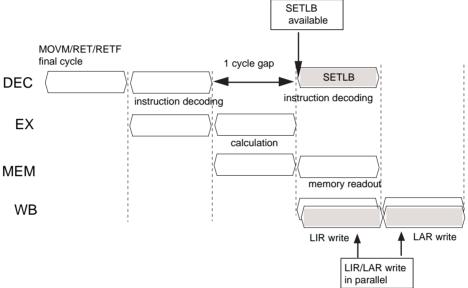
The SETLB instruction bypasses the five-stage pipeline with additional hardware to increase the execution speed. To ensure proper interaction between the two, it must wait for preceding instructions (MOVM, RET, and RETF) modifying the two registers that it loads: Loop Instruction Register (LIR) and Loop Address (LAR) Register.



[Description]

To ensure smooth pipeline operation, we therefore recommend inserting at least one cycle between such instructions and the SETLB instruction.





[Example]

[Problematic Version]		
inc	d0	
movm	(sp),[other]	Instruction modifying LIR and LAR
setlb		SETLB

Loading the LIR and LAR registers takes one cycle, so the pipeline stalls until their contents are available to the SETLB instruction.

[Revised Version]			
movm	(sp),[other]	Instruction modifying LIR and LAR	
inc	d0		
setlb		SETLB	

The updated contents of the LIR and LAR registers are available to the SETLB instruction, so the pipeline does not stall.

[Applicable Instructions]

<Leading instruction> MOVM (SP),regs; RET; RETF

<Following instruction> SETLB

3.8 Instructions Preceding RETF

The RETF instruction takes its return address from the Multiply/Divide Register (MDR), not the stack, so it must wait for preceding instructions modifying that register to complete. How long depends on the position of the MDR update relative to the end of the preceding instruction.



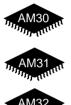
High-speed

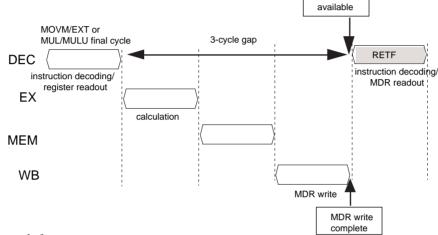
(1) MDR Update in Last Cycle

[Description]

To ensure smooth pipeline operation, we recommend inserting at least three cycles between such instructions and the RETF instruction.

RETF





[Problematic Version]				
_func				
_0func	FUNCINFO	_func,8,[]		
	inc	a0		
	mov	d0,mdr	Instruction modifying MDR in last	cycle
	inc	a1		
	inc	a2		
	retf		RETF	or
_func				
_0func		FUNCINFO	_func,8,[]	
_	inc	a0		
	inc	a1		
	mov	d0,mdr	Instruction modifying MDR in last	cycle
	inc	a2		
	retf		RETF	or
_func				
_0func	FUNCINFO	_func,8,[]		
	inc	a0		
	inc	a1		
	inc	a2		
	mov	d0,mdr	Instruction modifying MDR in last	cycle
	retf		RETF	



Loading the MDR register with the MOV instruction takes three cycles, so the pipeline stalls until that register's contents are available to the RETF instruction.

[Revised Version]			
_func			
_funco	FUNCINFO	_func,8,[]	
	mov	d0,mdr	Instruction modifying MDR in last cycle
	inc	a0	
	inc	a1	
	inc	a2	
	retf		retf

The updated contents of the MDR register are available to the RETF instruction, so the pipeline does not stall.

[Applicable Instructions]

<Leading instruction> MOVM Dm, MDR, EXT, MUL, MULU

<Following instruction> RETF



For a detailed description of FUNCINFO syntax, refer to the Cross Assembler User's Manual.



[Description]

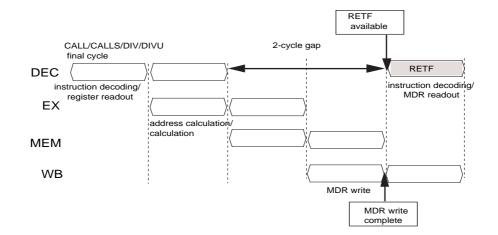
(2) MDR Update in Second Last Cycle

To ensure smooth pipeline operation, we recommend inserting at least two cycles between such instructions and the RETF instruction.









[Example]

[Problematic Version]				
_lab				
LABEL	FUNCINFO	_lab,8,[]		
	inc	a0		
	div	d1,d0	Instruction modifyint	MDR in second last cycle
	inc	a1		
	retf		RETF	or
_lab				
LABEL	FUNCINFO	_lab,8,[]		
	inc	a0		
inc	a1			
	div	d1,d0	Instruction modifying	g MDR in second last cycle
	retf		RETF	



Updating the MDR register with the DIV instruction takes two cycles, so the pipeline stalls until that register's contents are available to the RETF instruction.

Revised Version			
_lab			
LABEL	FUNCINFO	_lab,8,[]	
	div	d1,d0	Instruction modifying MDR in second last cycle
	inc	a0	
	inc	a1	
	retf		RETF

The updated contents of the MDR register are available to the RETF instruction, so the pipeline does not stall.

Applicable Instructions

<Leading instruction> DIV, DIVU, CALL, CALLS

<Following instruction> RETF



For a detailed description of FUNCINFO syntax, refer to the Cross Assembler User's Manual



High-speed

(3) MDR Update in Third Last Cycle

[Description]

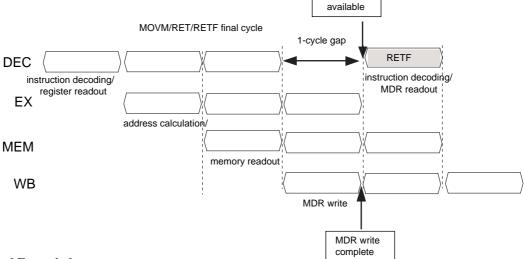
To ensure smooth pipeline operation, we recommend inserting at least one cycle between such instructions and the RETF instruction.

RETF









[Example]

[Problematic Version]		
_lab LABEL	FUNCINFO	_lab,8,[]	
LABEL	inc	_lab,o,[] d0	
	movm	(sp), [other]	Instruction modifying MDR in third last cycle
	retf	RETF	



RETF instruction Reloading the MDR register with the MOVM instruction takes one cycle, so the pipeline stalls until that register's contents are available to the RETF instruction.

[Revised Version] _lab			
LABEL	FUNCINFO movm	_lab,8,[] (sp),[other]	Instruction modifying MDR in third last cycle
	inc	d0	•
	retf		RETF

The updated contents of the MDR register are available to the RETF instruction, so the pipeline does not stall.

[Applicable Instructions]

<Leading instruction> MOVM (SP),regs, RET, RETF

<Following instruction> RETF



For a detailed description of FUNCINFO syntax, refer to the Cross Assembler User's Manual.

3.9 Instructions at CALL/CALLS Targets



High-speed

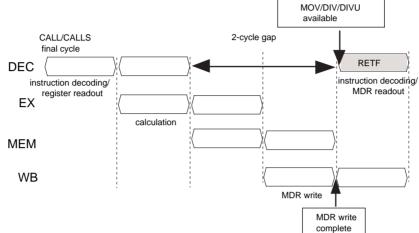
[Description]

To speed returns with the RETF instruction, the CALL and CALLS instructions store the return address in the Multiply/Divide Register (MDR), not on the stack. We therefore recommend inserting at least two cycles at the start of the subroutine to allow them time to complete the MDR update before using instructions (MOV, DIV, and DIVU) that reference that register.









[Example]

r				
[Problematic Version	1]			
	call	LABEL	CALL/CALLS	
	:			
	:			
LABEL	inc	a0		
	mov	mdr,d0	Instruction referring MDR	or
	call	LABEL	CALL/CALLS	
	:			
	:			
LABEL	mov	mdr,d0	Instruction referring MDR	
	inc	a0		
	inc	a1		



Loading the MDR register with the CALL/CALLS instruction takes two cycles, so the pipeline stalls until that register's contents are available to the MOV instruction.

[Revised Version]				
	call	LABEL	CALL/CALLS	
	:			
	:			
LABEL	inc	a0		
	inc	a1		
	mov	mdr,d0	Instruction referring MDR	

The updated contents of the MDR register are available to the MOV instruction, so the pipeline does not stall.

[Applicable Instructions]

<Leading instruction> CALL or CALLS

<Following instruction> MOV MDR, Dn, DIV, DIVU



The above coding examples omit the FUNCINFO directives required with each CALL instruction. For a detailed description of FUNCINFO syntax, refer to the Cross Assembler User's Manual.



Boiler Plate Code Sequences

4.1 Reset Routine



[Description]

A reset start leaves all registers with indeterminate contents, so always start by initiating registers. The Stack Pointer (SP) is particularly important.

If the user application program uses level interrupts, initialize the interrupt vector registers as well.

IVAR0	equ	0x20000000	
IVAR1	equ	0x20000004	
IVAR2	equ	0x20000008	
IVAR3	equ	0x2000000c	
IVAR4	equ	0x20000010	
IVAR5	equ	0x20000014	
IVAR6	equ	0x20000018	
;			
Reset routine			
;			
	org	0x40000000	; Entry point for reset
RESET:			
	jmp	STARTUP	
	org	0x40000008	; Entry point for nonmaskable interrupt
NMIROUTINE	:		
STARTUP:			
	mov	STCKTP,a0	; Initialize Stack Pointer(SP) to word boundary
			; (divisible by 4)
	mov	a0,sp	
	clr	d0	; Clear all registers
	mov	d0,d1	
	mov	d0,d2	
	mov	d0,d3	
	mov	d0,a0	
	mov	d0,a0	
	mov	a0,a1	
	mov	a0,a2	
	mov	a0,a3	
	mov	IRQ0ROUTINE,a0	; Set to lower 16 bits of entry point for level interrupt
handler			
	movhu	a0,(IVATo)	
	mov	IRQ1ROUTINE,a0	
	movhu	a0,(IVAR1)	
	mov	IRQ2ROUTINE,a0	
	movhu	a0,a0,(IVAR2)	
•			· · · · · · · · · · · · · · · · · · ·

mov	IRQ3ROUTINE,a0	
movhu	a0,(IVAR3)	
mov	IRQ4ROUTINE,a0	
movhu	a0,(IVAR4)	
mov	IRQ5ROUTINE,a0	
movhu	a0,(IVAR5)	
mov	IRQ6ROUTINE,a0	
movhu	a0,(IVAR6)	

4.2 Interrupt Handlers



[Description]

There are two types of interrupt handlers: for nonmaskable interrupts and for level interrupts.

For level interrupts, the Interrupt Group Register (IAGR) gives the group number. Assigning only one interrupt source to an interrupt level speeds response by eliminating the need for differentiation logic.

With store buffer models, always ensure clearing of the interrupt source by separating the write to the interrupt control register and the RTI instruction--with a vacuous read of that register, for example. For the serial debugger's special requirements for the nonmaskable interrupt handler, refer to the MN10300 Series C Source Code Debugger User's Manual.

```
Interrupt handler for NMI, level 0 interrupts and level 1 interrupts
IAGR
          equ
                0x34000200
IVAR0
          equ
                0x20000000
IVAR1
          equ
                0x20000004
IVAR2
                0x20000008
          equ
IVAR3
          equ
                0x2000000c
IVAR4
                0x20000010
          equ
IVAR5
          equ
                0x20000014
IVAR6
          equ
                0x20000018
G01CR
                0x34000100
          equ
G31CR
          equ
                0x3400010c
Nonmaskable interrupt handler
          0x40000008 ; Entry point for nonmaskable interrupts
    org
NMIROUTINE:
    add
          -8,sp
          a0,(4,sp)
                       ; Save registers used by handler
    mov
    mov
          d0,(sp)
          G01CR,a0
    mov
    mov
          (ao),d0
                       ; Get NMI source
```

```
; Processing according to source
                                                      ; Clear NMI source flag inG01CR
                   mov
                                        0x0f,d0
                   movbu
                                        d0,(a0)
                                        (sp),d0
                                                      ; Restore registers
                   mov
                                        (4,sp),a0
                   mov
                   add
                                        8,sp
                   rti
; Level 0 interrupt handler
; Here G3ICR lists only one source for level 0 interrupts
IRQ0ROUTINE:
                                                      ; Level 0 interrupt entry point from IVAR0
                   add
                                        -8,sp
                                                      ; Save registers used by handler
                   mov
                                        a0,(4,sp)
                                                      ; Process according to source
                    ....
                                        0x01,d0
                                                      ; Clear source flag in G3ICR
                   mov
                                        d0,(G3ICR)
                                                      ; Writing "1" to a bit clears corresonding source
                   mov
                   mov
                                        (sp),d0
                                                      ; Restore registers
                   mov
                                        (4,sp),a0
                   add
                                        8,sp
                   rti
; Level 1 interrupt handler
; Level 1 interrupts have multiple sources
IRQ1ROUTINE:
                                        ; Level 1 interrupt entry point from IVARI
                   add
                                        -16,sp
                   mov
                                        a0,(12,sp)
                                                      ; Save registers used by handler
                   mov
                                        a1,(8,sp)
                                        d0,(sp)
                   mov
                   movhu
                                        (IAGR),a0
                                                      ; Get interrupt source
                                        a0,a1
                   mov
                   add
                                        G01CR,a0
                                                      ; Get address for group's interrupt register
                   add
                                        IRQ_TBL,a1 ; Get entry point for group's handler
                                        (a0),d0
                                                      ; Get level 1 interrupt source
                   movhu
#IFDEFMULTIRQ
                                        0x0800,psw
                                                      ; Enable nested interrupts(IE=1)
                   or
#ENDIF
                                        (a1),al
                                                      ;Get handlers entry point
                   mov
                                                      ; Call handler, determine corresponding source
                   calls
                                        (a1)
                                                       ; f;or level 1 interrupt and process all corresponding
                                                       ; sources
                                        0x0f,d0
                                                      ; Clear source flog in corresponding interrupt
                   mov
                                                      ; register
                   movbu
                                        d0,(a0)
                                                      ; Writing "1" to a bit clears corresponding source
                   mov
                                        (sp),d0
                                                      ; Restore registers
                   mov
                                        (4,sp),a1
                                        (8,sp),a0
                   mov
                   add
                                        12,sp
                   rti
```

4.3 Function Called with CALL Only



[Description]

The CALL, RET, and RETF instruction require FUNCINFO directives to specify the registers to

If the subroutine does not modify the Multiply/Divide Register (MDR), the RETF instruction provides a faster return to the caller by using the return address in MDR.

For the procedures for calling C language functions from assembly language functions and assembly language functions from C language functions, refer to the C Compiler User's Manual for Operation.

```
; Calling function
_MAIN:
           add
                         -12,sp
                                          ; Secure space for subroutine parameters and return
value
                         d0,(8,sp)
                                          ; Save parameters on stack
           mov
                         d1,(4,sp)
           mov
                         _0FUNC
                                          ; Call subroutine
           call
           add
                         12,sp
; Subroutine
; Subroutine uses 28 bytes of local storage for saving A2 and other purposes
           FUNCINFO _FUNC,28,[a2]
_0FUNC
                                          ; Entry point for CALL instructions
                         ; Subroutine body
                         ; RETF may be used if subroutine does not modify MDR
           ret
```

4.4 Function Called with Both CALL and CALLS



[Description]

A function called with both the CALL and CALLS instructions must provide two entry points because CALLS does not provide CALL's register saving and local storage allocation features. The CALLS entry point must provide them.

If the subroutine does not modify the Multiply/Divide Register (MDR), the RETF instruction provides a faster return to the caller by using the return address in MDR.

For the procedures for calling C language functions from assembly language functions and assembly language functions from C language functions, refer to the C Compiler User's Manual for Operation.

```
; Calling function
_MAIN:
                                       ; Secure space for subroutine parameters and return
           add
                          -12,sp
                          d0,(8,sp)
                                       ; Save parameters on stack
           mov
           mov
                          d1,(4,sp)
                          _FUNC
                                       ; Call subroutine with CALL instruction
           call
           ....
           add
                          12,sp
           . . . .
           add
                          -12,sp
                                        ; Secure space for fubroutine parameters and return
           ....
           mov
                          d0,(8,sp)
                                       ; Save parameters on stack
                          d1,(4,sp)
           mov
           calls
                          _FUNC
                                       ; Call subroutine with CALL instruction
           add
                          12,sp
; Subroutine
; Subroutine uses 28 bytes of local storage for saving A2 and other purposes
; Subroutine may be called with CALL or CALLS instruction
_FUNC:
                                       ; CALLS instruction entry point must save registers and
                          [a2],(sp)
           movm
                                        ; allocate local storage
           add
                          -24,sp
_0FUNC
           FUNCINFO _FUNC,28[a2] ; Entry point for CALL instruction
                          ; Subroutine body
                          ; RETF may be used if subroutine does not modify MDR
           ret
```

4

Appendix

9		-	VF CF NF ZF Size	Z Z	<u>1</u>				2	:	2		
					-	H							
γ ΜΟΜ		Dm → Dn			•	1	S0 1000 DmDn						26
	MOV Dm,An	Dm → An	•		-1	2 1	D0 1111 0001 1110 DmAn						26
	MOV Am, Dn	Am → Dn	•			2 1	D0 1111 0001 1101 AmDn						26
	MOV Am,An	Am → An			,	-	S0 1001 AmAn						26
	MOV SP,An	SP → An			·	1	S0 0011 11An						26
	MOV Am,SP	An → SP			'	2 1	D0 1111 0010 1111 Am00						26
	MOV PSW,Dn	PSW(zero_ext) → Dn				2 1	Do 1111 0010 1110 01Dn						26
	MOV Dm,PSW	Dm → PSW	•	•	•	2 1	Do 1111 0010 1111 Dm11						26
	MOV MDR,Dn	MDR → Dn			'	2 1	Do 1111 0010 1110 00Dn						26
	MOV Dm,MDR	Dm → MDR			-	2 1	D0 1111 0010 1111 Dm11						26
	MOV (Am),Dn	mem32(Am) → Dn			Ė	-	S0 0111 DnAm						27
	MOV (d8,Am),Dn	mem32(d8(sign_ext)+Am) → Dn	•		'	3 1	D1 1111 1000 0000 DnAm <d8.< td=""><td><d8></d8></td><td></td><td></td><td></td><td></td><td>27</td></d8.<>	<d8></d8>					27
	MOV (d16,Am),Dn	mem32(d16(sign_ext)+Am) → Dn				4 1	D2 1111 1010 0000 DnAm <d16< td=""><td><d16< td=""><td>V</td><td></td><td></td><td></td><td>27</td></d16<></td></d16<>	<d16< td=""><td>V</td><td></td><td></td><td></td><td>27</td></d16<>	V				27
	MOV (d32,Am),Dn	mem32(d32+Am) → Dn	•	-	-	6 2	D4 1111 1100 0000 DnAm <d32< td=""><td><d32< td=""><td></td><td>:</td><td>^::</td><td></td><td>27</td></d32<></td></d32<>	<d32< td=""><td></td><td>:</td><td>^::</td><td></td><td>27</td></d32<>		:	^::		27
	MOV (d8,SP),Dn	mem32(d8(zero_ext)+SP) → Dn			.,	2 1	S1 0101 10Dn <d8></d8>						27
	MOV (d16,SP),Dn	mem32(d16(zero_ext)+SP) → Dn				4	D2 1111 1010 1011 01Dn	<d16< td=""><td>V</td><td></td><td></td><td></td><td>27</td></d16<>	V				27
	MOV (d32,SP),Dn	mem32(d32+SP) → Dn		-	<u> </u>	6 2	D4 1111 1100 1011 01Dn	<d32< td=""><td></td><td>:</td><td>^::</td><td></td><td>27</td></d32<>		:	^::		27
	MOV (Di,Am),Dn	mem32(Di+Am) → Dn	-		-	2 1	D0 1111 0011 00Dn DiAm						27
	MOV (abs16),Dn	mem32(abs16(zero_ext)) → Dn	•			3 1	S2 0011 00Dn <abs16< td=""><td>^</td><td></td><td></td><td></td><td></td><td>27</td></abs16<>	^					27
	MOV (abs32),Dn	mem32(abs32) → Dn	•		-	6 2	D4 1111 1100 1010 01Dn	<abs32< td=""><td></td><td></td><td>^</td><td></td><td>27</td></abs32<>			^		27
	MOV (Am),An	mem32(Am) → An	•		-	2 1	D0 1111 0000 0000 AnAm						27
	MOV (d8,Am),An	mem32(d8(sign_ext)+Am) → An	-	-	-	3 1	D1 1111 1000 0010 AnAm	<m><q8< td=""><td></td><td></td><td></td><td></td><td>27</td></q8<></m>					27
	MOV (d16,Am),An	mem32(d16(sign_ext)+Am) → An	•	-	-	4 1	D2 1111 1010 0010 AnAm <d16< td=""><td><d16< td=""><td>^</td><td></td><td></td><td></td><td>27</td></d16<></td></d16<>	<d16< td=""><td>^</td><td></td><td></td><td></td><td>27</td></d16<>	^				27
	MOV (d32,Am),An	mem32(d32+Am) → An	-		-	6 2	D4 1111 1100 0010 AnAm	<d32< td=""><td></td><td></td><td>^</td><td></td><td>27</td></d32<>			^		27
	MOV (d8,SP),An	mem32(d8(zero_ext)+SP) → An	•		- "	2 1	S1 0101 11An <d8></d8>						27
	MOV (d16,SP),An	mem32(d16(zero_ext)+SP) → An	•		-	4 1	D2 1111 1010 1011 00An	<d16< td=""><td>^.</td><td></td><td></td><td></td><td>27</td></d16<>	^.				27
	MOV (d32,SP),An	mem32(d32+SP) → An	-		-	6 2	D4 1111 1100 1011 00An	<d32< td=""><td></td><td></td><td><····</td><td></td><td>27</td></d32<>			< ····		27
	MOV (Di,Am),An	mem32(Di+Am → An	•		-	2 1	D0 1111 0011 10An DiAm						27
	MOV (abs16),An	mem32(abs16(zero_ext)) → An	•		-	4 1	D2 1111 1010 1010 00An	<abs></abs> abs16	^				27
	MOV (abs32),An	mem32(abs32) → An	•		-	6 2	D4 1111 1100 1010 00An	<abs32< td=""><td></td><td></td><td>^</td><td></td><td>27</td></abs32<>			^		27
	MOV (d8,Am),SP	mem32(d8(sign_ext)+Am) → SP			'	3 1	D1 1111 1000 1111 00Am <d8< td=""><td>< 8p></td><td></td><td></td><td></td><td></td><td>27</td></d8<>	< 8p>					27
	MOV Dm,(An)	Dm → mem32(An)	•		'	-	S0 0110 DmAn						28
	MOV Dm,(d8,An)	Dm → mem32(d8(sign_ext)+An)	-		'	3 1	D1 1111 1000 0111 DmAn <d8< td=""><td>< 8p></td><td></td><td></td><td></td><td></td><td>28</td></d8<>	< 8p>					28
	MOV Dm,(d16,An)	Dm → mem32(d16(sign_ext)+An)				1 1	D2 1111 1010 0001 DmAn	<d16< td=""><td>V</td><td></td><td></td><td></td><td>28</td></d16<>	V				28
	MOV Dm,(d32,An)	Dm → mem32(d32+An)	•			6 2	D4 1111 1100 0001 DmAn <d32< td=""><td><d32< td=""><td>::</td><td>:</td><td>^::</td><td></td><td>28</td></d32<></td></d32<>	<d32< td=""><td>::</td><td>:</td><td>^::</td><td></td><td>28</td></d32<>	::	:	^::		28
	MOV Dm,(d8,SP)	Dm → mem32(d8(zero_ext)+SP)	•		-	2 1	S1 0100 Dm10 <d8></d8>						28
	MOV Dm,(d16,SP)	Dm → mem32(d16(zero_ext)+SP)	•		-	4 1	D2 1111 1010 1001 Dm01	<d16< td=""><td>٨</td><td></td><td></td><td></td><td>28</td></d16<>	٨				28
	MOV Dm,(d32,SP)	Dm → mem32(d32+SP)	1			6 2	D4 1111 1100 1001 Dm01	<d32< td=""><td>:</td><td></td><td>^::</td><td></td><td>28</td></d32<>	:		^::		28
	MOV Dm,(Di,An)	Dm → mem32(Di+An)	•			2 2	D0 1111 0011 01Dm DiAn						28
	MOV Dm,(abs16)	Dm → mem32(abs16(zero_ext))	-		'	3 1	S2 0000 Dm01 <abs16< td=""><td>^</td><td></td><td></td><td></td><td></td><td>28</td></abs16<>	^					28
	(OC-1-) O / OW	(CCada/CCanada	_										

Page		28	28	28	28	28	28	28	28	28	28	28	59	29	59	59	59	59	30	30	30	30	30	30	30	30	30	30	31	31	31	31	31	31	31	31	31	31
Notes																																						
14																																						
13																																						
12					^:			1:			^:				1:			1:				1:			1:			^::				1:			1:			^:
=					:			;			:				:			;							:							;			:			:
10					:						:				:										:							:			:			i
0					:						:				:										:													:
				^:			^:			^:											^:			^:	.						1			<u>^</u>				
ode 8																																						
Machine Code			^		:					:	:	_		٨	1		_	:		_	:	:	_	:	1					٨	:	:					٨	
Macl			^:: ~::	9	32		9	32		<abs16< td=""><td><abs32< td=""><td>^::</td><td></td><td>^::</td><td><imm32< td=""><td></td><td> ^:</td><td><imm32< td=""><td></td><td>^::</td><td></td><td>2</td><td><u></u></td><td> 9</td><td></td><td></td><td><u></u></td><td><abs32< td=""><td></td><td>^:::</td><td></td><td></td><td></td><td></td><td>22</td><td></td><td>^:</td><td></td></abs32<></td></imm32<></td></imm32<></td></abs32<></td></abs16<>	<abs32< td=""><td>^::</td><td></td><td>^::</td><td><imm32< td=""><td></td><td> ^:</td><td><imm32< td=""><td></td><td>^::</td><td></td><td>2</td><td><u></u></td><td> 9</td><td></td><td></td><td><u></u></td><td><abs32< td=""><td></td><td>^:::</td><td></td><td></td><td></td><td></td><td>22</td><td></td><td>^:</td><td></td></abs32<></td></imm32<></td></imm32<></td></abs32<>	^::		^::	<imm32< td=""><td></td><td> ^:</td><td><imm32< td=""><td></td><td>^::</td><td></td><td>2</td><td><u></u></td><td> 9</td><td></td><td></td><td><u></u></td><td><abs32< td=""><td></td><td>^:::</td><td></td><td></td><td></td><td></td><td>22</td><td></td><td>^:</td><td></td></abs32<></td></imm32<></td></imm32<>		^:	<imm32< td=""><td></td><td>^::</td><td></td><td>2</td><td><u></u></td><td> 9</td><td></td><td></td><td><u></u></td><td><abs32< td=""><td></td><td>^:::</td><td></td><td></td><td></td><td></td><td>22</td><td></td><td>^:</td><td></td></abs32<></td></imm32<>		^::		2	<u></u>	9			<u></u>	<abs32< td=""><td></td><td>^:::</td><td></td><td></td><td></td><td></td><td>22</td><td></td><td>^:</td><td></td></abs32<>		^:::					22		^ :	
5		۲	AmAn <d8< td=""><td>An <d16< td=""><td>^n <d32< td=""><td></td><td>00 <d16< td=""><td>30 <d32< td=""><td>ے</td><td>l</td><td></td><td>n <d8< td=""><td></td><td>:</td><td></td><td></td><td> ;</td><td></td><td>٤</td><td>m. <d8< td=""><td>m <d16< td=""><td>.m <d32< td=""><td>n <d8< td=""><td>n <d16< td=""><td>n <d32< td=""><td>ے</td><td>:</td><td></td><td>٦</td><td>DmAn <d8< td=""><td>DmAn <d16< td=""><td>λn <d32< td=""><td>8b> 01</td><td>10 <d16< td=""><td>10 <d32< td=""><td>_</td><td>:</td><td>1000 Dm10 <abs32< td=""></abs32<></td></d32<></td></d16<></td></d32<></td></d16<></td></d8<></td></d32<></td></d16<></td></d8<></td></d32<></td></d16<></td></d8<></td></d8<></td></d32<></td></d16<></td></d32<></td></d16<></td></d8<>	An <d16< td=""><td>^n <d32< td=""><td></td><td>00 <d16< td=""><td>30 <d32< td=""><td>ے</td><td>l</td><td></td><td>n <d8< td=""><td></td><td>:</td><td></td><td></td><td> ;</td><td></td><td>٤</td><td>m. <d8< td=""><td>m <d16< td=""><td>.m <d32< td=""><td>n <d8< td=""><td>n <d16< td=""><td>n <d32< td=""><td>ے</td><td>:</td><td></td><td>٦</td><td>DmAn <d8< td=""><td>DmAn <d16< td=""><td>λn <d32< td=""><td>8b> 01</td><td>10 <d16< td=""><td>10 <d32< td=""><td>_</td><td>:</td><td>1000 Dm10 <abs32< td=""></abs32<></td></d32<></td></d16<></td></d32<></td></d16<></td></d8<></td></d32<></td></d16<></td></d8<></td></d32<></td></d16<></td></d8<></td></d8<></td></d32<></td></d16<></td></d32<></td></d16<>	^n <d32< td=""><td></td><td>00 <d16< td=""><td>30 <d32< td=""><td>ے</td><td>l</td><td></td><td>n <d8< td=""><td></td><td>:</td><td></td><td></td><td> ;</td><td></td><td>٤</td><td>m. <d8< td=""><td>m <d16< td=""><td>.m <d32< td=""><td>n <d8< td=""><td>n <d16< td=""><td>n <d32< td=""><td>ے</td><td>:</td><td></td><td>٦</td><td>DmAn <d8< td=""><td>DmAn <d16< td=""><td>λn <d32< td=""><td>8b> 01</td><td>10 <d16< td=""><td>10 <d32< td=""><td>_</td><td>:</td><td>1000 Dm10 <abs32< td=""></abs32<></td></d32<></td></d16<></td></d32<></td></d16<></td></d8<></td></d32<></td></d16<></td></d8<></td></d32<></td></d16<></td></d8<></td></d8<></td></d32<></td></d16<></td></d32<>		00 <d16< td=""><td>30 <d32< td=""><td>ے</td><td>l</td><td></td><td>n <d8< td=""><td></td><td>:</td><td></td><td></td><td> ;</td><td></td><td>٤</td><td>m. <d8< td=""><td>m <d16< td=""><td>.m <d32< td=""><td>n <d8< td=""><td>n <d16< td=""><td>n <d32< td=""><td>ے</td><td>:</td><td></td><td>٦</td><td>DmAn <d8< td=""><td>DmAn <d16< td=""><td>λn <d32< td=""><td>8b> 01</td><td>10 <d16< td=""><td>10 <d32< td=""><td>_</td><td>:</td><td>1000 Dm10 <abs32< td=""></abs32<></td></d32<></td></d16<></td></d32<></td></d16<></td></d8<></td></d32<></td></d16<></td></d8<></td></d32<></td></d16<></td></d8<></td></d8<></td></d32<></td></d16<>	30 <d32< td=""><td>ے</td><td>l</td><td></td><td>n <d8< td=""><td></td><td>:</td><td></td><td></td><td> ;</td><td></td><td>٤</td><td>m. <d8< td=""><td>m <d16< td=""><td>.m <d32< td=""><td>n <d8< td=""><td>n <d16< td=""><td>n <d32< td=""><td>ے</td><td>:</td><td></td><td>٦</td><td>DmAn <d8< td=""><td>DmAn <d16< td=""><td>λn <d32< td=""><td>8b> 01</td><td>10 <d16< td=""><td>10 <d32< td=""><td>_</td><td>:</td><td>1000 Dm10 <abs32< td=""></abs32<></td></d32<></td></d16<></td></d32<></td></d16<></td></d8<></td></d32<></td></d16<></td></d8<></td></d32<></td></d16<></td></d8<></td></d8<></td></d32<>	ے	l		n <d8< td=""><td></td><td>:</td><td></td><td></td><td> ;</td><td></td><td>٤</td><td>m. <d8< td=""><td>m <d16< td=""><td>.m <d32< td=""><td>n <d8< td=""><td>n <d16< td=""><td>n <d32< td=""><td>ے</td><td>:</td><td></td><td>٦</td><td>DmAn <d8< td=""><td>DmAn <d16< td=""><td>λn <d32< td=""><td>8b> 01</td><td>10 <d16< td=""><td>10 <d32< td=""><td>_</td><td>:</td><td>1000 Dm10 <abs32< td=""></abs32<></td></d32<></td></d16<></td></d32<></td></d16<></td></d8<></td></d32<></td></d16<></td></d8<></td></d32<></td></d16<></td></d8<></td></d8<>		:			;		٤	m. <d8< td=""><td>m <d16< td=""><td>.m <d32< td=""><td>n <d8< td=""><td>n <d16< td=""><td>n <d32< td=""><td>ے</td><td>:</td><td></td><td>٦</td><td>DmAn <d8< td=""><td>DmAn <d16< td=""><td>λn <d32< td=""><td>8b> 01</td><td>10 <d16< td=""><td>10 <d32< td=""><td>_</td><td>:</td><td>1000 Dm10 <abs32< td=""></abs32<></td></d32<></td></d16<></td></d32<></td></d16<></td></d8<></td></d32<></td></d16<></td></d8<></td></d32<></td></d16<></td></d8<>	m <d16< td=""><td>.m <d32< td=""><td>n <d8< td=""><td>n <d16< td=""><td>n <d32< td=""><td>ے</td><td>:</td><td></td><td>٦</td><td>DmAn <d8< td=""><td>DmAn <d16< td=""><td>λn <d32< td=""><td>8b> 01</td><td>10 <d16< td=""><td>10 <d32< td=""><td>_</td><td>:</td><td>1000 Dm10 <abs32< td=""></abs32<></td></d32<></td></d16<></td></d32<></td></d16<></td></d8<></td></d32<></td></d16<></td></d8<></td></d32<></td></d16<>	.m <d32< td=""><td>n <d8< td=""><td>n <d16< td=""><td>n <d32< td=""><td>ے</td><td>:</td><td></td><td>٦</td><td>DmAn <d8< td=""><td>DmAn <d16< td=""><td>λn <d32< td=""><td>8b> 01</td><td>10 <d16< td=""><td>10 <d32< td=""><td>_</td><td>:</td><td>1000 Dm10 <abs32< td=""></abs32<></td></d32<></td></d16<></td></d32<></td></d16<></td></d8<></td></d32<></td></d16<></td></d8<></td></d32<>	n <d8< td=""><td>n <d16< td=""><td>n <d32< td=""><td>ے</td><td>:</td><td></td><td>٦</td><td>DmAn <d8< td=""><td>DmAn <d16< td=""><td>λn <d32< td=""><td>8b> 01</td><td>10 <d16< td=""><td>10 <d32< td=""><td>_</td><td>:</td><td>1000 Dm10 <abs32< td=""></abs32<></td></d32<></td></d16<></td></d32<></td></d16<></td></d8<></td></d32<></td></d16<></td></d8<>	n <d16< td=""><td>n <d32< td=""><td>ے</td><td>:</td><td></td><td>٦</td><td>DmAn <d8< td=""><td>DmAn <d16< td=""><td>λn <d32< td=""><td>8b> 01</td><td>10 <d16< td=""><td>10 <d32< td=""><td>_</td><td>:</td><td>1000 Dm10 <abs32< td=""></abs32<></td></d32<></td></d16<></td></d32<></td></d16<></td></d8<></td></d32<></td></d16<>	n <d32< td=""><td>ے</td><td>:</td><td></td><td>٦</td><td>DmAn <d8< td=""><td>DmAn <d16< td=""><td>λn <d32< td=""><td>8b> 01</td><td>10 <d16< td=""><td>10 <d32< td=""><td>_</td><td>:</td><td>1000 Dm10 <abs32< td=""></abs32<></td></d32<></td></d16<></td></d32<></td></d16<></td></d8<></td></d32<>	ے	:		٦	DmAn <d8< td=""><td>DmAn <d16< td=""><td>λn <d32< td=""><td>8b> 01</td><td>10 <d16< td=""><td>10 <d32< td=""><td>_</td><td>:</td><td>1000 Dm10 <abs32< td=""></abs32<></td></d32<></td></d16<></td></d32<></td></d16<></td></d8<>	DmAn <d16< td=""><td>λn <d32< td=""><td>8b> 01</td><td>10 <d16< td=""><td>10 <d32< td=""><td>_</td><td>:</td><td>1000 Dm10 <abs32< td=""></abs32<></td></d32<></td></d16<></td></d32<></td></d16<>	λn <d32< td=""><td>8b> 01</td><td>10 <d16< td=""><td>10 <d32< td=""><td>_</td><td>:</td><td>1000 Dm10 <abs32< td=""></abs32<></td></d32<></td></d16<></td></d32<>	8b> 01	10 <d16< td=""><td>10 <d32< td=""><td>_</td><td>:</td><td>1000 Dm10 <abs32< td=""></abs32<></td></d32<></td></d16<>	10 <d32< td=""><td>_</td><td>:</td><td>1000 Dm10 <abs32< td=""></abs32<></td></d32<>	_	:	1000 Dm10 <abs32< td=""></abs32<>
4		AmAn		AmAn	AmAn	^:	Am00	Am00	11Am DiAn	Am00	Am00	1111 01An	√8r	<imm16< td=""><td>10Dn</td><td><:></td><td><imm16< td=""><td>11An</td><td>DnAm</td><td>DnAm</td><td>DnAm</td><td>DnAm</td><td>10Dn</td><td>10Dn</td><td>10Dn</td><td>DiAm</td><td>16</td><td>10Dn</td><td>DmAn</td><td></td><td></td><td>DmAn</td><td>Dm10</td><td>Dm10</td><td>Dm10</td><td>01Dm DiAn</td><td>16</td><td>Dm(</td></imm16<></td></imm16<>	10Dn	<:>	<imm16< td=""><td>11An</td><td>DnAm</td><td>DnAm</td><td>DnAm</td><td>DnAm</td><td>10Dn</td><td>10Dn</td><td>10Dn</td><td>DiAm</td><td>16</td><td>10Dn</td><td>DmAn</td><td></td><td></td><td>DmAn</td><td>Dm10</td><td>Dm10</td><td>Dm10</td><td>01Dm DiAn</td><td>16</td><td>Dm(</td></imm16<>	11An	DnAm	DnAm	DnAm	DnAm	10Dn	10Dn	10Dn	DiAm	16	10Dn	DmAn			DmAn	Dm10	Dm10	Dm10	01Dm DiAn	16	Dm(
8		0001	0011	0011	0011	×d8	1001	1001	11An	1000	1000		DnDn <imm8></imm8>		1100			1101	0100	0100	0100	0100	1011	1011	1011	00 00	<abs></abs> abs16	1010	0101	0101	0101	0101	1001	1001	1001	01Dn	<abs16< td=""><td>1000</td></abs16<>	1000
8		0000	1000	1010	1100	Am11	1010	1100	0011	1010	1100	1000	DnDn	11Dn	1100	AnAn	01An	1100	0000	1000	1010	1100	1000	1010	1100	0100	01Dn	1100	0000	1000	1010	1100	1000	1010	1100	0100	Dm10	1100
-		1111	1111	1111	1111	0100	1111	1111	1111	1111	1111	1111	1000	0010	1111	1001	0010	1111	1111	1111	1111	1111	1111	1111	1111	1111	0011	1111	1111	1111	1111	1111	1111	1111	1111	1111	0000	D4 1111
e For		00	5	D2	D4	S	D2	7	8	D2	D4	5	S	S2	D4	S1	S2	D4	8	7	D2	D4	D1	D2	D4	8	S2	D4	D0	10	D2	D4	_	D2	D4	D0	S2	D4
ode Cycl		2 1	3	4	6 2	2	4	6	2 2	4	6 2	3	2	3 1	6 2	2	3	6	2	3	4	6 2	3 1	1	6 2	2	3 1	6 2	2 1	3 1	4	6	3	4	6 2	2 2	3 1	6 2
Flag Code Cycle For VF CF NF 7F Size	i																						•					-										
Flag						٠															٠		-	٠				-		-	٠	٠	٠				•	•
VFICE		-			-								'	-					-				•		-		•	-		-					-	-		•
	1																			_	<u>ة</u>		Dn	۵			_											
																				t) → Dn	xt) ↓ Dn		1	1			∩											
ڍ			t)+An)	Am → mem32(d16(sign_ext)+An)		t)+SP)	Am → mem32(d16(zero_ext)+SP)			ext))		()+An)							ء	mem8(d8(sign_ext)+Am)(zero_ext)	mem8(d16(sign_ext)+Am)(zero_ext)	ے 1	mem8(d8(zero_ext)+SP)(zero_ext)	mem8(d16(zero_ext)+SP)(zero_ext)	mem8(d32+SP)(zero_ext) → Dn	u ↑	mem8(abs16(zero_ext))(zero_ext]	• Dn					(4S+((t)+SP)			ext))	
Operation			Am → mem32(d8(sign_ext)+An)	sign_e	+An)	Am → mem32(d8(zero_ext)+SP)	(zero_e	+SP)	۸n)	Am → mem32(abs16(zero_ext))	32)	SP → mem32(d8(sign_ext)+An)	۵	Dn		An	↑ An		mem8(Am)(zero_ext) → Dn	+Am)(z)+Am)(mem8(d32+Am)(zero_ext] →	+SP)(z	t)+SP)(o_ext)	mem8(Di+Am)(zero_ext) → Dn	ext))(ze	_ext) →		(c)	An)	An)	Dm → mem8(d8(zero_ext)+SP)	Dm → mem8(d16(zero_ext)+SP)	SP)	(u	6(zero_	5)
0		32(An)	32(d8(s	32(d16	Am → mem32(d32+An)	32(d8(z	32(d16	Am → mem32(d32+SP)	Am → mem32(Di+An)	32(abs	Am → mem32(abs32)	32(d8(s	imm8(sign_ext) → Dn	imm16(sign_ext) → Dn	ر	↑ Tx	ext) ↑	_	ero_ex	n_ext)-	gn_ext	/m)(zei	ro_ext)	ero_ex	3P)(zer	n)(zero	(zero_	mem8(abs32)(zero_ext)	8(An)	8(d8+A	8(d16+	8(d32+	8(d8(ze	8(d16(z	mem8(d32+SP)	8(Di+A	8(abs16	8(abs3;
		Am → mem32(An)	mem	mem	mem	mem	mem	mem	mem	mem	mem	mem3	sign_e	(sign_	imm32 → Dn	imm8(zero_ext] →	imm16(zero_ext)	imm32 → An	(Am)(z	(d8(sig	(d16(si	(d32+A	(d8(zei	(d16(ze	(d32+S	(Di+An	(abs16	(abs32	Dm → mem8(An)	mem	mem	mem	mem	mem	mem	mem	mem	mem{
		Am ↑	Am ↑	Am ↑	Am →	Am ↑	Am ↑	Am↑	Am ↑	Am ↑	Am ↑	SP ↑	jmm8(imm16	imm32	imm8(imm16	imm32	mem8	mem8	mem8	mem8	mem8	mem8			mem8	mem8	Dm	Dm → mem8(d8+An)	Dm → mem8(d16+An)	Dm → mem8(d32+An)	Dm	Dm	↑ Dm	Dm → mem8(Di+An)	Dm → mem8(abs16(zero_ext))	Dm → mem8(abs32)
				(c)	n)	(<u>a</u>	_	<u></u>	(;	_							٦		_	_					ď,		•					_				s32)
Mnemonic		(An)	(d8,An	(d16,A	(d32,A	(d8,SP	(d16,S	(d32,S	(Di,An)	(abs16	(abs32	(d8,An)	ng,bn	116,Dn	132,Dn	ı8,An	116,An	132,An	'Am),Dı	d8,Am)	d16,An	'd32,An	(48,SP)	'd16,SF	'd32,SF	Di,Am)	(abs16)	(abs32)	Jm,(An	Jm,(d8,	Jm,(d1t	Jm,(d3;	Jm,(d8,	Jm,(d1t	Jm,(d3;	Jm,(Di,	Jm,(ab	Jm,(ab
Mne		MOV Am,(An)	MOV Am,(d8,An)	MOV Am,(d16,An)	MOV Am,(d32,An)	MOV Am,(d8,SP)	MOV Am,(d16,SP)	MOV Am,(d32,SP)	MOV Am,(Di,An)	MOV Am,(abs16)	MOV Am,(abs32)	MOV SP,(d8,An)	MOV imm8,Dn	MOV imm16,Dn	MOV imm32,Dn	MOV imm8,An	MOV imm16,An	MOV imm32,An	MOVBU MOVBU (Am),Dn	MOVBU (d8,Am),Dn	MOVBU (d16,Am),Dn	MOVBU (d32,Am),Dn	MOVBU (d8,SP),Dn	MOVBU (d16,SP),Dn	MOVBU (d32,SP),Dn	MOVBU (Di,Am),Dn	MOVBU (abs16), Dn	MOVBU (abs32),Dn	MOVBU Dm,(An)	MOVBU Dm,(d8,An)	MOVBU Dm,(d16,An)	MOVBU Dm,(d32,An)	MOVBU Dm,(d8,SP)	MOVBU Dm,(d16,SP)	MOVBU Dm,(d32,SP)	MOVBU Dm,(Di,An)	MOVBU Dm,(abs16)	MOVBU Dm,(abs32)
<u>q</u>	-		Ø	Ø	MO	Ø	Ø	8	\ \ \ \ \ \ \ \ \ \ \ \ \ \	Ø	Ø	Ø	Ø	MO	Ø	ğ	8	8	BU MO	Ø	\ Ø	§ Ø	Ø	MO	Ø	ğ	Ø	MO	MO	MO	8	ğ	Θ	ğ	8	MO	MO	MO
Group	1	MOV																	MOVE																			

			-			ŀ	1	ļ																	
Group	Mnemonic	Operation	 	Ę F	VF CF NF ZF Size -mat	C ZF Si	ode Cyc	e For	~	2	3	4	2	Mach 6	Machine Code 6 7	ode 8	6		10	11	12	13	4	Notes	Page
MOVB	MOVB (Am), Dn	mem8(Am)(sign_ext) → Dn	•	-	-	•	3 2																		32
	MOVB (d8,Am),Dn	mem8(d8(sign_ext)+Am)(sign_ext) → Dn	•	-			4 2																		32
	MOVB (d16,Am),Dn	mem8(d16(sign_ext)+Am)(sign_ext) → Dn	•	٠		,	5 2																		32
	MOVB (d32,Am),Dn	mem8(d32+Am)(sign_ext) → Dn		٠			7 3																		32
	MOVB (d8,SP),Dn	mem8(d8(zero_ext)+SP)(sign_ext) → Dn				•	4																		32
	MOVB (d16,SP),Dn	mem8(d16(zero_ext)+SP)(sign_ext) → Dn			٠	,	5 2																		32
	MOVB (d32,SP),Dn	mem8(d32+SP)(sign_ext) → Dn					7 3																		32
	MOVB (Di,Am),Dn	mem8(Di+Am)(sign_ext) → Dn	٠	-		<u> </u>	3 2																		32
	MOVB (abs16),Dn	mem8(abs16(zero_ext))(sign_ext) → Dn		٠	٠	<u> </u>	4																		32
	MOVB (abs32),Dn	mem8(abs32)(sign_ext) → Dn		٠	٠	-	7 3																		32
	MOVB Dm,(An)	Dm → mem8(An)	•			,	2 1																		33
	MOVB Dm,(d8,An)	Dm → mem8(d8(sign_ext)+An)			٠		3																		33
	MOVB Dm,(d16,An)	Dm → mem8(d16(sign_ext)+An)		٠		•	1																		33
	MOVB Dm,(d32,An)	Dm → mem8(d32+An)			٠	<u> </u>	6 2																		33
	MOVB Dm,(d8,SP)	Dm → mem8(d8(zero_ext)+SP)				,	3																		33
	MOVB Dm,(d16,SP)	Dm → mem8(d16(zero_ext)+SP)			٠	•	4																		33
	MOVB Dm,(d32,SP)	Dm → mem8(d32+SP)		٠	٠		6 2																		33
	MOVB Dm,(Di,An)	Dm → mem8(Di+An)		٠	٠	,	2 2																		33
	MOVB Dm,(abs16)	Dm → mem8(abs16(zero_ext))	•				3																		33
	MOVB Dm,(abs32)	Dm → mem8(abs32)		٠		-	6 2																		33
МОУН	MOVHU (Am), Dn	mem16(Am)(zero_ext) → Dn	٠		٠		2 1	D0	1111	0000	0110	DnAm	μ												34
	MOVHU (d8,Am),Dn	mem16(d8(sign_ext)+Am)(zero_ext) → Dn	٠	٠	٠		3 1	D1	1111	1000	0110	DnAm	m <d8< td=""><td>^::</td><td>_</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>34</td></d8<>	^ ::	_										34
	MOVHU (d16,Am),Dn	mem16(d16(sign_ext)+Am)(zero_ext) → Dn	·		٠	·	4	D2	1111	1010	0110		DnAm <d16< td=""><td></td><td> ;</td><td></td><td><u>^</u></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>34</td></d16<>		;		<u>^</u>								34
	MOVHU (d32,Am),Dn	mem16(d32+Am)(zero_ext) → Dn			٠	•	6 2	D4	1111	1100	0110		DnAm <d32< td=""><td></td><td>1</td><td></td><td></td><td>1</td><td></td><td>1</td><td>^:</td><td></td><td></td><td></td><td>34</td></d32<>		1			1		1	^:				34
	MOVHU (d8,SP),Dn	mem16(d8(zero_ext)+SP)(zero_ext) → Dn	٠	٠	٠		3	70	1111	1000	1011	11Dn	ς 208	1	_										34
	MOVHU (d16,SP),Dn	mem16(d16(zero_ext)+SP)(zero_ext) → Dn				·	4	D2	1111	1010	1011	11 In	416 ر		;		^:								34
	MOVHU (d32,SP),Dn	mem16(d32+SP)(zero_ext) → Dn					6 2	D4	1111	1100	1011	11Dn	- cd32		1			:	i	i	^:				34
	MOVHU (Di,Am),Dn	mem16(Di+Am)(zero_ext) → Dn		٠	٠		2 1	00	1111	0100	10Dn	DiAm	_												34
	MOVHU (abs16),Dn	mem16(abs16(zero_ext))(zero_ext) → Dn	•			-	3	S2	0011	10Dn	<abs></abs> abs16	91		^:	_										34
	MOVHU (abs32),Dn	mem16(abs32)(zero_ext) → Dn	٠	٠	٠	-	6 2	D4	1111	1100	1010	11Dn	. <abs32< td=""><td>332</td><td></td><td></td><td></td><td>:</td><td>:</td><td>:</td><td>^::</td><td></td><td></td><td></td><td>34</td></abs32<>	332				:	:	:	^::				34
	MOVHU Dm,(An)	Dm → mem16(An)				•	2 1	D0	1111	0000	0111	DmAn	ç												35
	MOVHU Dm,(d8,An)	Dm → mem16(d8(sign_ext)+An)	٠	٠	٠	•	3	D1	1111	1000	0111	DmAn	n <d8< td=""><td>^:</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>35</td></d8<>	^ :											35
	MOVHU Dm,(d16,An)	Dm → mem16(d16(sign_ext)+An)	•	•		•	4	D2	1111	1010	0111	DmAn	n <d16< td=""><td>::</td><td>:</td><td></td><td>^:</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>35</td></d16<>	::	:		^:								35
	MOVHU Dm,(d32,An)	Dm → mem16(d32+An)					6 2	D4	1111	1100	0111		Dm.An <d32< td=""><td>2</td><td></td><td></td><td></td><td>:</td><td>:</td><td> </td><td>^::</td><td></td><td></td><td></td><td>32</td></d32<>	2				:	:		^::				32
	MOVHU Dm,(d8,SP)	Dm → mem16(d8(zero_ext)+SP)	٠	٠			3	Б	1111	1000	1001	Dm11	1 <d8< td=""><td>^:</td><td>_</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>35</td></d8<>	^ :	_										35
	MOVHU Dm,(d16,SP)	Dm → mem16(d16(zero_ext)+SP)	•	٠			4 1	D2	1111	1010	1001	Dm11	1 <d16< td=""><td> 6</td><td></td><td></td><td>^</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>35</td></d16<>	6			^								35
	MOVHU Dm,(d32,SP)	Dm → mem16(d32+SP)	-	٠		•	6 2	7	1111	1100	1001	Dm11	1 <d32< td=""><td></td><td>:</td><td></td><td></td><td>:</td><td>:</td><td>:</td><td>^:</td><td></td><td></td><td></td><td>35</td></d32<>		:			:	:	:	^:				35
	MOVHU Dm,(Di,An)	Dm → mem16(Di+An)	•	٠	٠	-	2 2	D0	1111	0100	11Dm	11Dm DiAn													35
	MOVHU Dm,(abs16)	Dm → mem16(abs16(zero_ext))	•	•	٠	-	3	S2	0000	Dm11	<abs16< td=""><td></td><td>i</td><td>1:</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>35</td></abs16<>		i	1:											35
	MOVHU Dm,(abs32)	Dm → mem16(abs32)	•	•		•	6 2	7	1111	1100	1000	Dm11	1 <abs32< td=""><td>332</td><td>:</td><td>:</td><td></td><td>:</td><td>:</td><td>;</td><td>^::</td><td></td><td></td><td></td><td>35</td></abs32<>	332	:	:		:	:	;	^::				35

Page	[36	36	36	36	36	36	30	36	36	36	36	37	37	37	37	37	37	37	37	37	37	38	3 8	38	38	1	38	38		38				38					38				\neg
Notes																							registers specified with regs = 0		registers specified with regs = 1	registers specified with regs = $2(*1)$		registers specified with regs = $3(*1)$	registers specified with regs = $=4(*1)$		redisters specified with reds = 7				registers specified with regs = 8					registers specified with regs = $9(*1)$				
4	f																																											
13																																												
12																																												
11																																												
10																																												
6																																												
ω g																																												
Machine Code 6 7																																												
Machi 6																																												
2																																												
4																							1																					
က																							<re>creds</re>	960 5																				
7																							1110	2																				
-																							1100	- 1																				
le For -mat																							S	5																				4
Flag Code Cycle	⊢	3 2	4 2	5 2	7 3	4	+	+	7	3	4 2	\vdash	2	2	+	6	+	4	9	2	+	9	+	+	2 2	2	+	2 4	2 5		ς α	_			2 9					2 10				\dashv
ZF S	Ì	-																						İ			1																	
Flag F NF	ŀ		•						•	٠						٠	ļ ·	ļ ·	١									•			'				ŀ									4
VF C	ŀ	-					1			•			'		·		·		·		·	·		l	•	•					·				' ,									\dashv
	Ī		Dn	uO ↑		٦	ځ	u ↑			'n													Ī		reg2,	ı	reg2,	33,		5	. A	F		, DO,	40,	MDR.	. Z	 :	yred2.	, -	<u>_</u>	Ä,	٦
Operation		mem16(Am)(sign_ext) → Dn	mem16(d8(sign_ext)+Am)(sign_ext) → [mem16(d16(sign_ext)+Am)(sign_ext) →	mem16(d32+Am)(sign_ext) → Dn	mem16(d8(zero ext)+SP)(sign ext) → Dn		₽l	mem16(d32+SP)(sign_ext) → Dn	mem16(Di+Am)(sign_ext) → Dn	mem16(abs16(zero_ext))(sign_ext) → Dn	mem16(abs32)(sign_ext) → Dn	m16(An)	Dm → mem16(d8(sign_ext)+An)	Dm → mem16(d16(sign_ext)+An)	Dm → mem16(d32+An)	Dm → mem16(d8(zero ext)+SP)	Dm → mem16(d16(zero ext)+SP)	Dm → mem16(d32+SP)	n16(Di+An)	Dm → mem16(abs16(zero ext))	m16(abs32)	C C		mem32(SP) → reg,SP+4 → SP	4) → reg1,mem32(SP) →		mem32(SP+8) → reg1,mem32(SP+4) → reg2, mem32(SP) → reg3,SP+12 → SP	mem32(SP+12) → D2,mem32(SP+8) → D3,	mem32(SP+4) \rightarrow A2,mem32(SP) \rightarrow A3, SP+16 \rightarrow SP	mem32/SP+28) → D0 mem32/SP+24) → D1	mem32(SP+20) → A0 mem32(SP+16) → A1	mem32(SP+12) → MDR.mem32(SP+8) → LIR	mem32(SP+4) → LAR,SP+32 → SP	mem32(SP+32) → reg,mem32(SP+28) → D0,	mem32(SP+24) →D1,mem32(SP+20) →A0,	mem32(SP+16) →A1,mem32(SP+12) →MDR,	mem32(SP+8) →LIR.mem32(SP+4) →LAR.	P P	mem32(SP+36)→rea1. mem32(SP+32)→rea2.	mem32(SP+28)→D0,mem32(SP+24)→D1,	mem32(SP+20)→A0,mem32(SP+16)→A1,	mem32(SP+12)→MDR,mem32(SP+8)→LIR,	mem32(SP+4)→LAR,SP+40 →SP
		mem16(Am	mem16(d8(.	mem16(d16	mem16(d32	mem16(d8(;	2 Poom 4 E (d 4 E	mem16(d1t	mem16(d32	mem16(Di+	mem16(abs	mem16(abs	Dm → mem16(An)	Dm → mer	Dm → men	Dm → mer	Dm → mer	Dm → men	Dm → men	Dm → mem16(Di+An)	Dm → men	Dm → mem16(abs32)	MOVM (SP). [rea1rean] PC+2 → PC		mem32(SP,	mem32(SP+	01-10-1	mem32(SP. mem32(SP)	mem32(SP.	mem32(SP+4)	mem32/SP-	mem32(SP-	mem32(SP,	mem32(SP-	mem32(SP-	mem32(SP-	mem32(SP-	mem32(SP-	SP+36 →SP	mem32(SP-	mem32(SP-	mem32(SP-	mem32(SP.	mem32(SP.
္	ŀ	·					١,																1regn]				_1_																	
Mnemonic		MOVH (Am),Dn	MOVH (d8,Am),Dn	MOVH (d16,Am),Dn	MOVH (d32,Am),Dn	MOVH (d8.SP).Dn	(046.00)	MOVH (a16,SP),Un	MOVH (d32,SP),Dn	MOVH (Di,Am),Dn	MOVH (abs16),Dn	MOVH (abs32).Dn	MOVH Dm.(An)	MOVH Dm.(d8.An)	MOVH Dm.(d16,An)	MOVH Dm.(d32,An)	MOVH Dm.(d8.SP)	MOVH Dm.(d16,SP)	MOVH Dm.(d32,SP)	MOVH Dm.(Di.An)	MOVH Dm.(abs16)	MOVH Dm.(abs32)	M (SP).freq	% 18/ 19/ III																				
۵		MOV	MOV	MOV	MOV	MO	Š	2	MOV	MOV	MOV	MO	MO	NO N	MOV	MOV	MOV	MO	MO	MO	MO	Q O																						\downarrow
Group		MOVH																					MVOM)																				

Group Mnemonic Cheration Flag VF CF NF ZI	Operation Flag Code Cycle	Flag VF CF NF ZI	Flag		TIL.	Code (ycle Fr	For -mat	2	3	4	5	Machine Code 6	Code 8	6	10	=	12	13	41	Notes	Page
		╽╽	1 1					╽┟														ı ŀ
MOVM MOVM (SP),[reg1,regn] mem32(SP+40)→reg1,mem32(SP+36)→reg2, - -					•	7	11 8	S1 11	1100 1110	0 <regs< td=""><td>st</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>registers specified with regs=10(*1)</td><td>88</td></regs<>	st										registers specified with regs=10(*1)	88
mem32(SP+32)→reg3,mem32(SP+28)→D0,	mem32(SP+32)→reg3,mem32(SP+28)→D0,																					
mem32(SP+24)→D1,mem32(SP+20)→A0, mem32(SP+16)→A1,mem32(SP+12)→MDR,	mem32(SP+24)→D1,mem32(SP+20)→A0, mem32(SP+16)→A1,mem32(SP+12)→MDR,																					
mem32(SP+8)→LIR,mem32(SP+4)→LAR,	mem32(SP+8)→LIR,mem32(SP+4)→LAR,																					
						\rightarrow	+	+														4
						7	12														rregisters specified with regs=11	88
mem3z(SP+30)→Az,mem3z(SF+32)→A3, mem32(SP+28)→D0,mem32(SP+24)→D1,	mem32(SP+28)→D0,mem32(SP+24)→D1.			_	_			—														
mem32(SP+20)→A0,mem32(SP+16)→A1,	mem32(SP+20)→A0,mem32(SP+16)→A1,																					
mem32(SP+12)→MDR,mem32(SP+8)→LIR,	mem32(SP+12)→MDR,mem32(SP+8)→LIR,				_																	
mem32(SP+4)→LAR,SP+48→SP	mem32(SP+4)→LAR,SP+48→SP																					
MOVM [reg1,regn],(SP) PC+2→PC 2		1			-2		-	S1 11	1100 1111	1 <regs< td=""><td> St</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>registers specified with regs =0</td><td>39</td></regs<>	St										registers specified with regs =0	39
reg→mem32(SP-4),SP-4→SP 2				- 2	7	_	-														registers specified with regs=1	39
reg1→mem32(SP-4),reg2→mem32(SP-8), 2		•	•		7		7														registers specified with regs= $2(*2)$	39
SP-8→SP	SP-8→SP				_	\dashv	\dashv															
reg1→mem32(SP-4),reg2→mem32(SP-8), 2					7		က														registers specified with regs= $3(*2)$	39
reg3→mem32(SP-12),SP-12→SP	reg3->mem32(SP-12),SP-12→SP					-																
D2→mem32(SP-4),D3→mem32(SP-8), 2		1	1	- 2	7		4														registers specified with regs= 4	39
A2→mem32(SP-12),A3→mem32(SP-16),	A2→mem32(SP-12),A3→mem32(SP-16),																					
SP-16→SP	SP-16→SP					+	1															
D0→mem32(SP-4),D1→mem32(SP-8), - - - - 2		•	•		7		ω														registers specified with regs= 7	33
A0→mem32(SP-12),A1→mem32(SP-16),	A0→mem32(SP-12),A1→mem32(SP-16),				_																	
MDR→mem32(SP-20), LIR→mem32(SP-24),	MDR→mem32(SP-20), LIR→mem32(SP-24),																					
LAR→mem32(SP-28),SP-32→SP	LAR →mem32(SP-28),SP-32→SP					+	+															
reg→mem32(SP-4),D0→mem32(SP-8), 7		1			7		6														registers specified with regs= 8(*2)	33
D1→mem32(SP-12),40→mem32(SP-16),	D1→mem32(SP-12),A0→mem32(SP-16),																					
A1→mem3z(3P-z4), 11D -mom32/SE 30) 1 AB -mom32/SE 30)	A1 ->mem3z(
SP-38→SP	SP-36→SP																					
reg1→mem32(SP-4),reg2→mem32(SP-8), 2		-	-		2		10														registers specified with regs= $9(*2)$	39
D0→mem32(SP-12),D1→mem32(SP-16),	D0→mem32(SP-12),D1→mem32(SP-16),																					
A0→mem32(SP-20),A1→mem32(SP-24),	A0→mem32(SP-20),A1→mem32(SP-24),																					
MDR→mem32(SP-28), LIR→mem32(SP-32),	MDR→mem32(SP-28), LIR→mem32(SP-32),																					
LAR→mem32(SP-36),SP-40→SP	LAR→mem32(SP-36),SP-40→SP																					
reg1→mem32(SP-4),reg2→mem32(SP-8), 2		1			-2		7														registers specified with regs= 10(*2)	39
reg3→mem32(SP-12),D0→mem32(SP-16),	reg3→mem32(SP-12),D0→mem32(SP-16),		_																			
D1→mem32(SP-20),A0→mem32(SP-24),	D1→mem32(SP-20),A0→mem32(SP-24),			_	_	_																
A1→mem32(SP-28),MDR→mem32(SP-32),	A1→mem32(SP-28),MDR→mem32(SP-32),				_	_																
LIR→mem32(SP-36),LAR→mem32(SP-40),,	LIR→mem32(SP-36),LAR→mem32(SP-40),,			_	_																	
SP-44→SP	SP-44→SP				╛	-	\dashv	\dashv														\Box

^{**}I: Registers specified with regn are returned in the order, D2, D3, A2 and A3 no matter when the assembler writes theses registers. Skip the registers which is not specified
*2: Registers specified with regn are saved in the order, D2, D3, A2 and A3 no matter when the assembler write these registers. Skip the registers which is not specified.

Group	Mnemonic Operation	Flag Code Code	Notes	Page
MOVM MOVM regs.(SP)	ps.(SP) D2 → mem32(SP-4), D3 → mem32(SP-8) A2 → mem32(SP-12), A3 → mem32(SP-16) D0 → mem32(SP-20), D1 → mem32(SP-24) A0 → mem32(SP-28), A1 → mem32(SP-32) MDR → mem32(SP-38), LIR → mem32(SP-40) LAR → mem32(SP-44), SP-48 → SP	2 12 S1 1100 1111 <regs></regs>	Registers specified with regs =11	30
EXT EXT Dn	IF (Dn.bp31=0), 0x00000000 → MDR IF (Dn.bp31=1), 0xFFFFFFFF → MDR	2 1 D0 1111 0010 1101 00Dn		40
EXTB EXTB Dn		1 80 0001		14
EXTBU EXTBU DN EXTH DN	Dn & 0x000000FF → Dn F (Dn.bp15=0), Dn & 0x0000FFFF → Dn F (Dn.bp15=1), Dn 0xFFFF0000 → Dn	1 1 S0 0001 10Dn		43
I⊋I		1 1 80 0001		44
CLR CLR Dn Arithmetic Operation Instructions	0 → Dn Instructions	0 0 1 1 1 50 0000 Dn00		45
ADD ADD Dm,Dn	On Dm+Dn →Dn	● ● ● 1 1 S0 1110 DmDn		46
ADD Dm,An	An Dm+An →An	• • • 2 1 D0 1111 0001 0101 AmDn		46
ADD Am,An		• • • 2 1 D0 1111 0001 0111		46
ADD imm8,Dn),Dn imm8(sign_ext) + Dn → Dn	• • • 2 1 S1 0010 10Dn <imm8></imm8>		47
ADD imm16,Dn	16,Dn imm16(sign_ext) + Dn → Dn	1010 1100		47
ADD imm32,Dn	imm32 + Dn → Dn	• • 6 2 D4 1111 1100 1100		47
ADD imm8,An	s,An Imm8(sign_ext) + An → An imm16(sign_ext) + An → An	• • • • 2 1 S1 W10 WAN camming>		4 47
ADD imm32,An	imm32 + An → An	b		47
ADD imm8,SP	3,SP imm8(sign_ext) + SP → SP	3 1 D1 1111 1000 1111 1110 <inmm8></inmm8>		47
ADD imm16,SP	imm16(sign_	1 D2 1111 1010 1111 1110		47
_		2 D4 1111 1100 1111		47
ADDC ADDC Dm,Dn SUB SUB Dm,Dn	,Dn Dm+Dn+CF ↓Dn Dn-Dm ↓Dn			84 84 64
SUB Dm,An	ın An - Dm → An	● ● ● 2 1 D0 1111 0001 0010 DmAn		49
SUB Am,Dn	on Dn - Am → Dn	● ● ● 2 1 D0 1111 0001 0001 AmDn		49
SUB Am,An	ın An-Am →An	• • • • 2 1 D0 1111 0001 0011 AmAn		49
SUB imm32,Dn	22,Dn Dn - imm32 → Dn	• • • 6 6 2 D4 1111 1100 1100 10Dn https://doi.org/10.100/100		20
-	_	1111 1100		20
		• • • 2 1 D0 1111 0001 1000		51
MUL MUL Dm,Dn	Dn (Dn* Dn) → { MDR , Dn }	? ? • • 2 3 D0 1111 0010 0100 DmDn	Dn =0	52
		13	Dn =value by 1 byte	25
		21	Dn =value by 2-byte	25
		2 29	Dn =value by 3-byte	52
		34	Dn =value by 4-byte	76

Milk	Group	Mnemonic	Operation	Flag Code Cycle	ZF S	ode Oyc	For -mat 1	2	8	4	2	Macr 6	Machine Code 6 7	8	0	10	7	12	13	4		Notes		Page
10 10 10 10 10 10 10 10	MULU	MULU Dm,Dn	(Dn* Dn) → { MDR , Dn }	٠-	-	\vdash	\vdash		1 1		۳ L										Dn = 0			53
Not Digner Other						13															Dn = value b	y 1 byte		53
DV Dn, Dn DN DEACH DN DEACH						21															Dn = value b	y 2-byte		53
No. Dip.						29															Dn = value b	y 3-byte		53
No PunDa Chan Cha						34															Dn = value b	y 4-byte		53
From +Don (MADR-cs. 22) & OutFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF	DIV	DIV Dm,Dn	((MDR<< 32) & 0xFFFFFFFF00000000 + Dn)	1/5	¿/• ¿	-					пОп										{MDR,Dn} =	0		54
Windless 30 & DorFefferFrondondonous - Don Am - Made Windless 30 & DorFefferFrondondonous - Don Am - Made		/Dm →Dn			14															{MDR,Dn}= v	ralue by 1 by	te	54	
Nov Dim Dim (Not Rec. 22) & defetifiere (Not Control Dim) Not Dim N			((MDR<< 32) & 0xFFFFFFFFF0000000 + Dn)			22															{MDR,Dn} =	value by 2-	byte	54
Note Decide Control		% Dm → MDR			30															{MDR,Dn} =	value by 3-l	3-byte	54	
PAND DIAN DIAN AND RANK SEAS A DAFFFFFFFROOROOOOU-DIAN DIAN DIAN DIAN DIAN DIAN DIAN DIAN						38															{MDR,Dn} =	value by 4-	byte or more	54
No. Dm - Dm No. Dm - Dm - Dm - Dm - Dm No. Dm - Dm - Dm - No. Dm - Dm No. Dm - Dm	DVIQ	DIVU Dm,Dn	((MDR<< 32) & 0xFFFFFFFFF00000000 + Dn)	3/2	¿/• ¿						ď										{MDR,Dn} =	0		22
Note Continued Note			/ Dm → Dn			14															{MDR,Dn} =	value by 1 b	yte	22
NC Dn			((MDR<< 32) & 0xFFFFFFFF00000000+ Dn)			22															{MDR,Dn} =	value by 2-1	2- byte	22
NC Dh			% Dm → MDR			30															{MDR,Dn} =	value by 3-b	yte	22
No. Dn Dn +1 - Dn No. +1 - An No. An +1 - An No						38															{MDR,Dn} =	value by 4-1	yte or more	22
Nick An	S	INC Dn	Dn +1 → Dn	•		-	000	- 1	00u															99
NPG4 An		INC An					08		101															56
CMP Dm.Dn Dn - Dn: PSW C		INC4 An	An + 4 → An		-				JAn															22
CMP Dm,Dn Dn - Dm : PSW O	Compa	rative Instructions																						
CMP Dm,An An - Dm : PSW • • • 2 1 DM 1111 0001 1001 AmD CMP Am,Dn Dn - Am: PSW • • • • • 1 1111 0001 1001 AmD CMP Am,An Dn - Immel(sign_ext): PSW • • • • 1 15 1011 AmA CMP Imm8,Dn Dn - Immel(sign_ext): PSW • • • • 1 15 1011 AmA CMP Imm8,Dn Dn - Immel(sign_ext): PSW • • • • 1 15 1011 AmA Immel(sign_ext): PSW • <td< td=""><td>CMP</td><td>CMP Dm,Dn</td><td>Dn - Dm : PSW</td><td>•</td><td></td><td>_</td><td>80</td><td></td><td>m Du</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>28</td></td<>	CMP	CMP Dm,Dn	Dn - Dm : PSW	•		_	80		m Du															28
CMP Am_Dn Dn - Am : PSW •		CMP Dm,An	An - Dm : PSW	•		_	8				nAn													58
CMP Ann, An An - Am: PSW • • • 1 1 SO 4 Intercept An - Am Anne An - Am : PSW		CMP Am,Dn	Dn - Am : PSW	\dashv	•	\rightarrow	8	- 1			П П													28
CMP imm8_Dn Dn - imm8 (signext): PSW •		CMP Am,An	An - Am : PSW	•	-	-			mAn															58
CMP imm32.Dn Dn - imm16 (sign_ext) : PSW •		CMP imm8,Dn	Dn - imm8(sign_ext) : PSW	•	•	-				~	٨													58
CMP imm32.Dn Dn - imm32 : PSW •		CMP imm16,Dn	Dn - imm16(sign_ext) : PSW	•	\dashv	\rightarrow				- 1		mm16		1					٨					28
CMP imm8,An An - imm8(zero_ext) : PSW Sero_ext) : PSW Sero_ext : PSW Sero		CMP imm32,Dn	Dn - imm32 : PSW	-	•	\dashv	72				- 1	m32		:	:	i	:							28
CMP imm16,An		CMP imm8,An	An - imm8(zero_ext) : PSW	•	•					mm8	^													58
CMP imm32,An		CMP imm16,An	An - imm16(zero_ext) : PSW		•	-				- 1		mm16		1:					٨					58
AND Dm,Dn AND imm8,Dn imm8(Zero_ext) & Dn → Dn AND imm16,Cero_ext) & Dn → Dn AND imm16,Dn imm8(Zero_ext) & SN → DN AND imm16,Dn imm8(Zero_ext) & SN → DN AND imm16,Dn imm8(Zero_ext) & PSW → PSW AND imm16,Dn imm8(Zero_ext) & PSW → PSW AND imm16,Dn imm8(Zero_ext) Dn → Dn AND imm16,Dn		CMP imm32,An	An - imm32 : PSW	•	•	-	4	- 1	- 1	- 1	- 1	nm32	:	:	:	i	:							58
AND Dim,Dh Imm8,Dh Imm8(Zero_ext) & Dh → Dh Oh Oh Oh Imm8,Dh Imm8(Zero_ext) & Dh → Dh Oh Oh Oh Oh Imm8,Dh Imm8(Zero_ext) & Dh → Dh Oh Oh Oh Oh Imm8,Dh Imm8(Zero_ext) & Dh → Dh Oh Oh Oh Oh Imm8,Dh Imm8(Zero_ext) & PSW → PSW OH Imm8,Dh Imm8(Zero_ext) & PSW → PSW OH Imm8,Dh Imm8(Zero_ext) Dh → Dh Oh Oh Oh Oh Oh Oh Oh Oh Imm8,Dh Imm8(Zero_ext) Dh → Dh Oh	Logical	Operation Instructions		-	ŀ	ŀ														Ī				ŀ
AND imm8,Dn imm8(zero_ext) & Dn → Dn 0 0 0 0 1 1 1111 1000 1110 00Dn circum8> AND imm16,Dn imm16(zero_ext) & Dn → Dn 0 0 0 0 0 1111 100 1110 00Dn circum8 <td>AND</td> <td>AND Dm,Dn</td> <td>Dm & Dn → Dn</td> <td>0</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>000</td> <td></td> <td>29</td>	AND	AND Dm,Dn	Dm & Dn → Dn	0						000														29
AND imm16.Dn imm16(zero_ext) & Dn → Dn 0 0 4 1 2 1111 100 1110 00D circum36 <th< td=""><td></td><td>AND imm8,Dn</td><td>imm8(zero_ext) & Dn → Dn</td><td>0</td><td>_</td><td>_</td><td></td><td></td><td></td><td></td><td></td><td>:</td><td>٨</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>29</td></th<>		AND imm8,Dn	imm8(zero_ext) & Dn → Dn	0	_	_						:	٨											29
AND imm32,Dn imm32 & Dn → Dn O o o o o o d<		AND imm16,Dn	imm16(zero_ext) & Dn → Dn	0	\dashv	\rightarrow	D2			- 1	- 1	mm16	:	1					٨					59
AND immt6,PSW immt6(zero_ext) & PSW → PSW •		AND imm32,Dn	imm32 & Dn → Dn		•							nm32	:	:	:	:	:							59
OR Dn,Dh OR Inm8(Dn Dn Dn → Dn Dn <t< td=""><td></td><td>AND imm16,PSW</td><td>imm16(zero_ext) & PSW → PSW</td><td>•</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>nm16</td><td>:</td><td>^:</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>90</td></t<>		AND imm16,PSW	imm16(zero_ext) & PSW → PSW	•								nm16	:	^:										90
imm8(zero_ext) Dn → Dn 0 0 4 1 1111 1000 1110 01Dn 4 inm82 10 inm82 Dn → Dn 4 inm	OR	OR Dm,Dn	Dm Dn → Dn		•		8				ď													61
imm16(zero_ext) Dh → Dh 0 0 4 1 D2 1111 1010 1110 OIDn cirrum16 > imm32 Dh → Dh 0		OR imm8,Dn		0	-	\rightarrow					- 1	8mm												61
imm32 Dn → Dn 0		OR imm16,Dn	imm16(zero_ext) Dn → Dn		•	\vdash						mm16	:	^::					٨					61
imm16(zero_ext) PSW → PSW ● ● ● 4 1 D2 1111 1010 1111 1101 <irram16< td=""><td></td><td>OR imm32,Dn</td><td>imm32 Dn → Dn</td><td>-</td><td>•</td><td>-</td><td>4</td><td></td><td></td><td>- 1</td><td></td><td>mm32</td><td>:</td><td>:</td><td>:</td><td>i</td><td>:</td><td></td><td></td><td></td><td></td><td></td><td></td><td>61</td></irram16<>		OR imm32,Dn	imm32 Dn → Dn	-	•	-	4			- 1		mm32	:	:	:	i	:							61
		OR imm16,PSW	imm16(zero_ext) PSW → PSW	•	•	4	D2					mm16		1										62

The file Date Date The file Date The fil					20		-	Ŀ	-					MA	1	o Pro						ŀ			Γ
Main Note Dec. On - 10 Main Note Dec. On	3roup			VF	H N	ZF	Code C Size	ycle To	3t 1	2	3	4	2	® 9 8	2 /	8 8	6	10	11	12		4	Notes	<u>R</u>	зде
Main continue Main continu				 																		ŀ		•	ſ
See FreeDiggs Part	OR	XOR Dm,Dn	Dm ^ Dn → Dn		•	•							ڄ											9	63
Main		XOR imm16,Dn	imm16(zero_ext) ^ Dn → Dn		•	•	_							m16	:	^::								9	33
NOT		XOR imm32,Dn	imm32 ^ Dn → Dn		•	•	\vdash							m32	:	:	1	:	1	1				9	က္က
BEST marking Interference and & Dit (Sept. Se	Ď	NOT Dn			0	•	_				l		ءِ ا											9	4
SET meritor meritor contact meritor cont	3it Instr	nctions																							1
SET mind (above) SET mind (a	STST	BTST imm8,Dn	imm8(zero_ext) & Dn:PSW			_						l	ı	8mi										9	55
STRT imm2QLPQ		BTST imm16,Dn	imm16(zero_ext) & Dn : PSW		_		-				l		l	16	:	1								9	32
STST imm6, lobe App Imm5 parts 2, and App Imm5 p		BTST imm32,Dn2	imm32 & Dn : PSW		•	•			1111					ım32	:	:	:	:	:	<u>^</u>				9	22
First immigration monoigleace and a monoigleace of a		BTST imm8,(d8,An)	imm8(zero_ext) &													m8								9	55
EST rime (joined) Immigration of the contribution of the contr			mem8(d8(sign_ext)+An)(zero_ext) : PSW																						
EST mms(class)		BTST imm8,(abs16)	imm8(zero_ext) &						1111					s16	:	^ ::		~ 8m				ž	ot used for AM30/AM31	9	52
SET mm8 (abs.d) mm8 (abs			mem8(abs16(zero_ext))(zero_ext): PSW																						
SEET Davidyon moned/designation and 3 - lenthy moned/designation and 3 - lenth		BTST imm8,(abs32)	imm8(zero_ext) & mem8(abs32)(zero_ext) : PSW											os32	i	i	i	i	i	^	<imms< td=""><td>^</td><td></td><td>9</td><td>55</td></imms<>	^		9	55
ESET hame, (Leb.A) merelécétégique, aot + Ani) merelécétégiq	SSET	BSET Dm,(An)											٩u											9	99
BSET Imm8, (lab.k.) Imm8 (lab.tics) Imm8			Carlo & Carlo Carl	+	+	+	+	+	_			- 1										+			Τ
BEET Imm8, (labsit) Imm08/caro		BSET imm8,(d8,An)	mem8(d8(sign_ext) + An)(zero_ext) → temp,						111		111					m8								9	7.0
SEET imm8, labs16 mem8(labs16/Earce_act) + Total Found			temp & imm8(zero_ext): PSW,																						
ESET imm8, (abs.52) mem8(abs.16(Eaco_axv))=cm_ox_0 Correct (abs.16) Correct (abs.			temp imm8(zero_ext) →																						
			mem8(d8(sign_ext) + An)		\dashv																				
BEET imm8 (deb52) PSW, Imm8 (deb52) PSW, Imm8 (deb52)		BSET imm8,(abs16)	mem8(abs16(zero_ext))(zero_ext) → temp,											s16	:	1		718				ž	ot used for AM30/AM31	9	22
ESET imm8(abs23) Enemp Famm8(abs23) Enemp Famm8(abs23) Enemp Enemp Famm8(abs23) Enemp Enemp Famm8(abs23) Enemp		temp & imm8(zero_ext) : PSW,																							
SET imm8 (debx32) Caro Card + Demonstrate (debx32) Card + Demonstrate (debx32) Caro Card + Demonstrate (debx32)			temp imm8(zero_ext) →																						
SeET imm8, labs23 mem8(abs32)/caro_axt) + lemp, mem8(abs			mem8(abs16(zero_ext))		-																				
BCLR Dm.(An) mem8(daylocaec_ext) - mem8(BSET imm8,(abs32)	mem8(abs32)(zero_ext) → temp ,											s32	i	i	i	i	i	^ :	<imm8< td=""><td>^:</td><td></td><td>9</td><td>22</td></imm8<>	^:		9	22
BCLR Dm,(An) mem8(dhx)(zero_ext) → mem8(dhx)(zero_ext) → temp & (Dm \ OxyFFFFFFF) → mem8(dhx)(zero_ext) → temp & (Dm \ OxyFFFFFFF) → mem8(dhx)(zero_ext) + An)/zero_ext) + An)/zero_ext) + An Zero_ext +			temp & imm8(zero_ext) : PSW,																						
BCLR Dm.(Ah) mem8(Ah)(zero_ext) → temp, temp & Dm.: PSW, temp & (Dm. AbFFFFFF) → mem8(AbSign_ext) → temp, temp & (Dm. AbFFFFFFF) → mem8(AbSign_ext) → temp, temp & (Imm8(AbSign_ext) → temp & (Imm8(AbSign_ext) → temp, temp & (Imm8(AbSign_ext) → temp & (Imm8(AbSign_ext) → temp, temp & (Imm8(AbSign_ext) → temp & (Imm8(A			temp imm8(zero_ext) → mem8(abs32)																						
temp & (Dm ^ 0xFFFFFFF) → mem8(Ab) mem(d8(sign_ext) + An)(Zero_ext) → temp, with mem(d8(sign_ext) + An)(Zero_ext) → temp, with mem8(d8(sign_ext) + An) mem8(d8(sign_ext) + An) mem8(d8(sign_ext) + An) mem8(d8(sign_ext) + An) mem8(d8(sign_ext) + Man) mem8(d8(sign_ext) + Man	BCLR	BCLR Dm,(An)	mem8(An)(zero_ext) → temp ,						1111				۸n											9	80
temp & (Dm ^ toxFFFFFFF) → mem8(An) mem8(des 63gn_ext) + An) temp & (imm8(zero_ext) → temp, a mem8(des 532) mem8(des 52co_ext) → temp & (imm8(zero_ext) → temp, a mem8(des 532) mem8			temp & Dm : PSW,																						
mem(d8(sign_ext) + An)(zero_ext) → temp,			temp & (Dm ^ 0xFFFFFFF) → mem8(An)			1		-			- 1														
temp & imm8(zero_ext) + PSW, temp & imm8(zero_ext) + An) mem8(ds(sign_ext) + Capt FFFFFF → mem8(ds(sign_ext) + An) mem8(ds(sign_ext) + Capt → temp & man (111 111 111 111 111 111 111 111 111		BCLR imm8,(d8,An)	mem(d8(sign_ext) + An)(zero_ext) → temp,													m8								9	60
temp & (imm8(zero_ext) + An) mem8(das(sign_ext) + An) mem8(das(sign_ext) + An) mem8(das(sign_ext) + An) mem8(das(sign_ext) + Emp , imm (imm8(zero_ext))			temp & imm8(zero_ext) : PSW,																						
mem8(ds\(sign_ext\) + An\) mem8(ds\(sign_ext\) + An\) 0			temp & (imm8(zero_ext) ^ 0xFFFFFFF) →																						
mem8(abs32)(zero_ext) → temp, & imm8(zero_ext) = PSW, temp & imm8(zero_ext) → temp & imm8(zero_ext) = PSW, temp & imm8(zero_ext)			mem8(d8(sign_ext) + An)																						
temp & imm8(zero_ext) : PSW, temp & imm8(zero_ext) > O v o free perference and imm8(zero_ext) > O v o free per		BCLR imm8,(abs16)	mem8(abs16(zero_ext))(zero_ext) → temp ,									000 0		s16	:	1		73				2	ot used for AM30/AM31	9	60
temp & (imm8(zero_ext) ^ \to NoFFFFFFF) → mem8(abs32)(zero_ext) + temp & imm8(zero_ext) > \to No + 7 & 6 & D5 & 1111 & 1110 & 0000 & 0001 & cabs32			temp & imm8(zero_ext) : PSW,																						
mem8(abs32)(zero_ext) 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0			temp & (imm8(zero_ext) ^ 0xFFFFFFF) →																						
mem8(abs32)(zero_ext) → temp & imm8(zero_ext) > PSW, 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0			mem8(abs16(zero_ext))	_	-			-																	
temp & imm8(zero_ext) : PSW, temp & (imm8(zero_ext) ^ 0xFFFFFFF > → mem8(abs32)		BCLR imm8,(abs32)	mem8(abs32)(zero_ext) → temp ,						1111	111	000	000		s32	:	:	i	:	:	^:	<imm8< td=""><td>^<u>·</u></td><td></td><td>9_</td><td>66</td></imm8<>	^ <u>·</u>		9_	66
temp & (imm8(zero_ext) ^ 0xFFFFFFFF → mem8(abs32) mem8(abs32)			temp & imm8(zero_ext): PSW,																						
mem8(abs32)			temp & (imm8(zero_ext) ^ 0xFFFFFFFF) →																						
			mem8(abs32)	\exists	\dashv	\Box	\exists	\dashv																	

						-	1																	ŀ	ſ
Group	Mnemonic	Operation	VF	Flag Code Cycle For VF VF CF NF ZF Size -m²	J F	Cod	e Cycle	For -mat	_	2	3	4	2	Mact 6	Machine Code 6 7	g 8	6	10	11	12	13	41	Notes	Page	<u>e</u>
Shift In:	Shift Instructions																								1
ASR	ASR Dm,Dn	IF ((Dm&0x0000001F) ≈ 0), Dn.lsb → CF,	خ	•	•	2	-	D0 1111	1111	0010		1011 DmDn	5											70	_
		(Dn >> (Dm & 0x0000001F))(sign_ext) → Dn																							
		IF ((Dm&0x0000001F)=0),PC + 2 → PC	٥.	<i>~</i>	•	_																			
	ASR imm8,Dn	IF ((imm8 & 0x1F) ≠0) , Dn.lsb → CF,	خ	•	•	3	-	1111	1111	1000	110	0 10E	n ⊲in	1100 10Dn <imm8></imm8>	^									71	
		(Dn >> (imm8 & 0x1F))(sign_ext) → Dn																							
		IF ((imm8 & 0x1F)=0), PC + 3 → PC	٠.	<i>-</i>	•	_																			
	ASR Dn	Dn.lsb → CF, (Dn >> 1)(sign_ext) → Dn	خ	•	•	3	-																	72	
LSR	LSR Dm,Dn	IF ((Dm&0x0000001F) ≠ 0),Dn.lsb → CF,	۰.	•	•	2	-	8	D0 1111 0010	0010		1010 DmDn	<u>ا</u>											73	
		(Dn >> (Dm & 0x0000001F))(zero_ext) → Dn																							
		IF ((Dm&0x0000001F)=0), PC + 2 → PC	۰.	~.	•	_																			
	LSR imm8,Dn	IF ((imm8 & 0x1F) ≠ 0),Dn.lsb → CF,	٠.	•	•	e	-	10	1111	1000	110	0 01E	n ⊲in	D1 1111 1000 1100 01Dn <imm8></imm8>										74	-
		(Dn >> (imm8 & 0x1F))(zero_ext) → Dn																							
		IF ((imm8 & 0x1F)=0), PC + 3 → PC	5	5	•	_																			
	LSR Dn	Dn.lsb → CF, (Dn >> 1)(zero_ext) → Dn	خ	•	•	3	-																	75	
ASL	ASL Dm,Dn	IF ((Dm & 0x0000001F) ≠ 0),	خ	خ	•	2	-	00	1111	D0 1111 0010		1001 DmDn	5											92	
		Dn << (Dm & 0x0000001F) → Dn																							
		IF ((Dm & 0x0000001F)=0), PC + 2 → PC																							
	ASL imm8,Dn	IF ((imm8 & 0x1F) ≠0),	٠.	~	•	e	-	D1 1111	1111	1000		0 000	n ⊲in	1100 00Dn <imm8></imm8>	^									77	
		Dn << (imm8 & 0x1F) → Dn																							
		IF ((imm8 & 0x1F)=0), PC + 3 → PC																							
ASL2	ASL2 Dn	(Dn << 2) & 0xFFFFFFC → Dn	خ	خ	•	1	1	08	0101	01Dn														78	
ROR	ROR Dn	CF << 31 → temp, Dn.lsb → CF,	0	•	•	2	1	D0	1111	0010	100	D0 1111 0010 1000 01Dn	ц											19	_
		(Dn >> 1)(zero_ext) temp → Dn																							
ROL	ROL Dn	CF → temp, Dn.msb → CF,	0	•	•	2	-	D0	1111	0010	100	D0 1111 0010 1000 00Dn	u											80	_
		(Dn << 1) temp → Dn																							

Group	p Mnemonic	Operation	Flag VF CF NF	Flag	ZF S	Code Cycle F	le For -mat	_	2	8	4	2	Mach 6	Machine Code 6 7	ge 8	6	10	0 11	12	13	41	Notes	ă.	Page
Branch	Branch Instructions																							1
Bcc	BEQ (d8,PC)	IF (ZF=1), PC + d8(sign_ext)→ PC IF (ZF=0), PC + 2 → PC	L'	-	,	2 3/1*	S1	1100	1000	8p>	1											Branch enable/disable	μ	81
	BNE (d8,PC)	IF $(ZF=0)$, PC + d8(sign_ext) \rightarrow PC		'		2 3/1*	S	1100	1001	8p>	1											Branch enable/disable	Ιω	81
	BGT (d8,PC)		1			2 3/1*	S	1100	0001	8b>	1											Branch enable/disable	1	18
	BGE (d8,PC)	IF ((NF ^ VF)=0),PC + d8(sign_ext) → PC IF ((NF ^ VF)=1),PC + 2 → PC				2 3/1*	S	1100	0010	862	1											Branch enable/disable	Ιω	81
	BLE (d8,PC)	sign_ext) → PC				2 3/1*	S1	1100	0011	8p>	1											Branch enable/disable	ω	18
	BLT (d8,PC)	IF ((NF ^ VF)=1),PC + d8(sign_ext) →PC IF ((NF ^ VF)=0),PC + 2 → PC		'		2 3/1*	٠ 2	1100	0000	8b>	1											Branch enable/disable	Ι ω	18
	BHI (d8,PC)	ext) → PC				2 3/1*	S1	1100	0101	8p>	1											Branch enable/disable	Ι	18
	BCC (d8,PC)	IF (CF = 0), PC + d8(sign_ext) \rightarrow PC IF (CF = 1), PC + 2 \rightarrow PC	1	'		2 3/1*	S1	1100	0110	8b>	1											Branch enable/disable	Ι ω	18
	BLS (d8,PC)	IF ((CF ZF)=1),PC + d8(sign_ext) → PC IF ((CF ZF)=0),PC + 2 → PC		'		2 3/1*	S	1100	0111	8b 8b	1											Branch enable/disable	ω	81
	BCS (d8,PC)	IF (CF = 1), PC + d8(sign_ext) \rightarrow PC IF (CF = 0), PC + 2 \rightarrow PC				2 3/1*	s S	1100	0100	8b>	1											Branch enable/disable	ω	18
	BVC (d8,PC)	IF (VF = 0), PC + d8(sign_ext) → PC IF (VF = 1), PC + 3 → PC				3 4/2*	10	1111	1000	1110	1000	80	1									Branch enable/disable	Ι ω	18
	BVS (d8,PC)	IF (VF = 1), PC + d8(sign_ext) → PC IF (VF = 0), PC + 3 → PC	1			3 4/2*	2	1111	1000	1110	1001	86	1									Branch enable/disable	Ι ω	18
	BNC (d8,PC)	IF (NF = 0), PC + d8(sign_ext) \rightarrow PC IF (NF = 1), PC + 3 \rightarrow PC				3 4/2*	7	1111	1000	1110	1010	86	1									Branch enable/disable	Ι ω	18
	BNS (d8,PC)	IF (NF = 1), PC + d8(sign_ext) \rightarrow PC IF (NF = 0), PC + 3 \rightarrow PC	1	1		3 4/2*	2	1111	1000	1110	1011	86	1									Branch enable/disable	Ι ω	18
	BRA (d8,PC)	PC + d8(sign_ext) →PC	•			2 3		S1 1100	1010	8b>	1												ω	81
CC	LEQ	IF (ZF=1), LAR - 4 →PC IF (ZF=0),PC + 1 →PC		'		1 1/2*	S0	1101	1000													Branch enable/disable	ω	82
	LNE	IF (ZF=0), LAR - 4 →PC IF (ZF=1), PC + 1 →PC				1 1/2*	So	1101	1001													Branch enable/disable	ω	82

*:Depends on the status of Instruction queue.

100 0001 100	Group Mnemonic Operation Flag code cycle	Operation	Flag VF CF NI	Flag	I III	ZF	Sode Cy	cle For	at	2	8	4	2	Machi 6	Machine Code 6 7	ω Θ	6	10	1	12	13	14		Notes	<u> </u>	Page
Branch enable/disable	LGT [F ((ZF (NF^VF))=0), LAR - 4 → PC				-	-	L 🔄		S0 1101		72												Branch enable/c	isable		22
Branch enable/disable	LGE IF ((NF v VP)=0), LAR - 4 →PC - 1 1/2* IF ((NF v VP)=1), PC + 1 →PC			•	-			OS	1101		0												Branch enable/c	isable	1	22
Description	LLE F ((\rightarrow \cdot \rightarrow \	1/2*	- 1 1/2*	- 1 1/2*	1 1/2*	1/2*		SO	1101		_												Branch enable/c	isable	ω	22
Branch enableidisable	LLT IF ((NF ^ VF)=1),LAR - 4 →PC 1 1/2* IF ((NF ^ VF)=0),PC + 1 →PC				-		*.	SO	1101		Q												Branch enable/c	isable	ω	22
Branch enable/disable	LHI IF ((CF ZF)=0), LAR - 4 → PC 1 1/2* IF ((CF ZF)=1), PC + 1 → PC		1	1		1 1/2*	<u>ت</u>	OS .	1101		5												Branch enable/c	isable	ω	ŭ
1001 Branch enable/disable Branch enable/disable 1010 1011 1010 1011 1000 1111 01An 1100 c416	LCC F (GF = 0), LAR · 4 → PC				-		0	s *	1101		0												Branch enable/c	isable		22
1010 1011 1014n 1010 1010 1011 1014n 1100 4016 > add: *4 cycyes for AM30 1100 4016. PC), registers specified with regs = 0 1101 4016. PC), registers specified with regs = 2 1101 1102 1103 	LLS IF ((CF ZF)=1), LAR - 4 → PC 1 1/2* IF ((CF ZF)=0), PC + 1 → PC	1			-		~	° So	1101		_												Branch enable/c	isable	Ι ω	22
1000 1111 01An 1100 cdf6	LCS IF (CF = 1), LAR · 4 → PC				-		N	80	1101		5												Branch enable/c	isable	ω	22
1000 1111 01An 1100 cd16	LRA - 4 → PC 1 1 1	1	- 1	- 1	1			SO	1101		0														ω.	Ñ
1100 cd16	SETLB SETLB	-	- 1	- 1	\vdash	\vdash	l –	တိ	S0 1101		_														<u> </u>	က္က
110 111 114 115 116 117	PC+5→LAR	LAR		,		+		1	;																1	Π.
1100	JMP (An) An → PC	3 8	3 8		3 8		<u>"</u>	SS DS	S2 1100					1											υ ω	4 4
1101 cd16 cregs cregs registers specified with regs = 0	5 3**	5 3**	5 3**	- 5 3**	5 3**	3**		S ₄	S4 1101				:	:	:	::	:	``:	^				**4 cycyes for A	M30	8	4
	5 2 9,PC+5→MDR,		- 2	- 2	2	2		4	S4 1100				i	^		st		8 mi	^				If label = (d16,P registers specifie	C), ed with regs = 0		35
2 8 4	PC + 5 → mem32(SP),reg1 → mem32(SP-4), 5 3 SP - imm8(zero_ext) → SP,PC + 5 → MDR, PC + d16(sim_ext) → PC	ن ا ا	ιο	ι ₀	2			1															If label = (d16,Pregisters specifie	C), ed with regs = 1	- ω	ž
with regs = 3 with regs = 4 with regs = 4	PC + 5 → mem32(SP), reg1 → mem32(SP-4), 5 4 reg2 → mem32(SP), SP - imm8(zero_ext) → SP, PC + 5 → MDR, PC + df6/sion ext) → PC	ιο	ιο	ιο	ro		+	1															If label = (d16,P	C), ed with regs = 2	Ι ω	22
with regs = 4	PC + 5 → mem32(SP), reg1 → mem32(SP-4), 5 5 freg2 → mem32(SP-8), spr - imm8(zero_ext) → SP, PC + 5 → MDR, PC + d16(sign_ext) → PC	ι	ις	ι	2		10	1															If label = (d16,P registers specifi	C), ad with regs = 3	W	22
	PC + 5 → mem32(SP), D2 → mem32(SP-4), 5 6 D3 → mem32(SP-8), A2 → mem32(SP-12), A3 → mem32(SP-16), SP - imm8(zero_ext) → SP, PC + 5 → MDR,		ιο	ιο	ro.		6																If label = (d16,P registers specifie	C), sd with regs = 4	w	22

*: Depends on the status of Instruction queue.

Mnemonic	Operation	Flag Code Cydle For	Flag	Cox ZF Sizt	de Cycle	For -mat	2	8	4	5	Macl 6	Machine Code 6	ω ω	o	10	7	12 1	13 14	Notes		Page
	PC + 5 → mem32(SP),DO → mem32(SP-4), D1 → mem32(SP-8),AO → mem32(SP-12), A1 → mem32(SP-16),MDR → mem32(SP-20), LIR → mem32(SP-24),LAR → mem32(SP-28), SP · imm8(zero_ext) → SP,PC + 5 → MDR, PC + d16(sign ext) → PC	•	1		0	\$ t	1100	1101	<d16< td=""><td> </td><td>1</td><td>v regs</td><td>\\ \\\ \stransformation \\ \stransformation</td><td>vimm8</td><td>^ 82</td><td></td><td></td><td></td><td>If label = (d16,PC), registers specified with regs = 7</td><td>h regs = 7</td><td>82</td></d16<>		1	v regs	\\ \\\ \stransformation \\ \stransformation	vimm8	^ 82				If label = (d16,PC), registers specified with regs = 7	h regs = 7	82
	PC+5 → mem32(SP), reg1 → mem32(SP-4), D0→ mem32(SP-8), D1 → mem32(SP-12), A0→ mem32(SP-16), A1→ mem32(SP-20), MDR→ mem32(SP-24), LIR→ mem32(SP-28), LAR→ mem32(SP-32), SP - imm8(zero_ext) → SP, PC + 5 → MDR, PC + d16(sign_ext) → PC			ιo	10														If label = (d16,PC), registers specified with regs	n regs = 8	82
	PC + 5 \rightarrow mem32 (SP)reg1 \rightarrow mem32(SP-4), reg2 \rightarrow mem32(SP-8).D0 \rightarrow mem32(SP-12), D1 \rightarrow mem32(SP-16),A0 \rightarrow mem32(SP-20), A1 \rightarrow mem32(SP-24),MDR \rightarrow mem32(SP-28), LIR \rightarrow mem32(SP-32),LAR \rightarrow mem32(SP-36), SP \rightarrow imm8(zero_ext) \rightarrow SP,PC + 5 \rightarrow MDR, PC + d16(sign_ext) \rightarrow PC		1	Ω	-														If label = (d16,PC), registers specified with regs = -	0 = 800 = 1	82
	PC + 5 \rightarrow mem32(SP),reg1 \rightarrow mem32(SP-4), reg2 \rightarrow mem32(SP-12), D0 \rightarrow mem32(SP-16),D1 \rightarrow mem32(SP-20), A0 \rightarrow mem32(SP-24),A1 \rightarrow mem32(SP-28), MDR \rightarrow mem32(SP-32),LR \rightarrow mem32(SP-38),LR \rightarrow mem32(SP-36), SP \rightarrow mem32(SP-40), SP \rightarrow mem32(SP-40), SP \rightarrow mem32(SP-40), PC \rightarrow m			ι ·	12														If label = (d16,PC), registers specified with regs = 10	h regs = 10	82
	PC + 5 → mem32(SP,D2→ mem32(SP-4), D3→ mem32(SP-8),A2→ mem32(SP-12), A3→ mem32(SP-16),D0→ mem32(SP-20), D1→ mem32(SP-24),A0→ mem32(SP-28), A1→ mem32(SP-32),MDR→ mem32(SP-36), LIR→ mem32(SP-40),LAR→ mem32(SP-44) SP - imm8(2ero_ext) → SP,PC + 5 → MDR, PC-416(sign_ext) → PC,		1	r.	13														If label = (d16, PC), registers specified with regs = 11	h regs = 11	85
	$PC+7 \rightarrow mem32(SP)$, SP - $imm8(zero_ext) \rightarrow SP,PC + 7 \rightarrow MDR$, $PC + d32 \rightarrow PC$	1	1	- 7	*4	Se 1	1101 1	1101 <c< td=""><td><d32< td=""><td>i</td><td>1</td><td>!</td><td>1</td><td>1</td><td>1</td><td><re>cregs</re></td><td>^::</td><td><imm8></imm8></td><td>If label = (d32,PC), registers specified with regs = 0 *: 5 cycyles for AM30</td><td>th regs = 0</td><td>85</td></d32<></td></c<>	<d32< td=""><td>i</td><td>1</td><td>!</td><td>1</td><td>1</td><td>1</td><td><re>cregs</re></td><td>^::</td><td><imm8></imm8></td><td>If label = (d32,PC), registers specified with regs = 0 *: 5 cycyles for AM30</td><td>th regs = 0</td><td>85</td></d32<>	i	1	!	1	1	1	<re>cregs</re>	^::	<imm8></imm8>	If label = (d32,PC), registers specified with regs = 0 *: 5 cycyles for AM30	th regs = 0	85
R R	$\label{eq:pc} \begin{split} & PC + 7 \rightarrow \text{mem32(SP),reg1} \rightarrow \text{mem32(SP-4),} \\ & SP \cdot \text{imm8(zero_ext)} \rightarrow SP, PC + 7 \rightarrow MDR, \\ & PC + d32 \rightarrow PC \end{split}$	1		- 7	*4														If label = (d32,PC), register specified with regs *: 5 cycles for AM30	regs = 1	85

Page	85			82				82					85						82							82							82							\Box
	F																		_																					\exists
Notes	If label = (d32,PC),	registers specified with regs = 2	*: 6 cycyles for AM30	If label = (d32,PC),	registers specified wtih regs = 3	*: 7 cycles for AM30		If label = (d32,PC),	registers specified with regs = 4	*: 8 cycles for AM30			If label = (d32,PC),	registers specified with regs = 7	*: 11 cycles for AM30				If label = (d32,PC),	registers specified with regs = 8	*: 12 cycles for AM30					If label = (d32,PC),	registers specified with regs = 9	*: 13 cycles for AM30					If label = (d32,PC),	registers specified with regs =10	* 14 cycles for AM30					
																																								\dashv
14	<>																																							
13																																								
12	^ s																																							
7	<re></re>																																							
10	1																																							
6	:																																							
ω																																								
Code 7	;																																							
Machine Code 6 7																																								
5																																								
4																																								
3	<d32< td=""><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></d32<>																																							
2	1101																																							
_	1101																																							
#	9S																																							
Cycle	2*			*9				*_					10*						*							12*							13*							
Code	7			7				7					7						7							7							7							
: ZF				•				•					•													٠							,							
Flag Code Cycle For VF CF NF ZF Size -ms	-			<u>'</u>				•					-																				'							
VF	-			,				,											,														,							
Operation	PC + 7 → mem32(SP),reg1→ mem32(SP-4),	reg2→ mem32(SP),SP - imm8(zero_ext) → SP,	PC + 7 → MDR,PC + d32 → PC	PC + 7 → mem32(SP),	reg1→ mem32(SP-4),reg2→ mem32(SP-8),	reg3→ mem32(SP),SP - imm8(zero_ext) → SP,	PC + 7 → MDR,PC + d32 → PC	PC + 7 → mem32(SP),D2→ mem32(SP-4),	D3→ mem32(SP-8),A2→ mem32(SP-12),	A3→ mem32(SP-16),	SP - imm8(zero_ext) → SP,PC + 7 → MDR,	PC + d32 → PC	PC + 7 → mem32(SP),D0→ mem32(SP-4),	D1→ mem32(SP-8),A0→ mem32(SP-12),	A1→ mem32(SP-16),MDR→ mem32(SP-20),	LIR→ mem32(SP-24),LAR→ mem32(SP-28),	SP - imm8(zero_ext) → SP,PC + 7 → MDR,	PC + d32 → PC	PC + 7 → mem32(SP),reg1 → mem32(SP-4),	D0→ mem32(SP-8),D1→ mem32(SP-12),	A0→ mem32(SP-16),A1→ mem32(SP-20),	MDR→ mem32(SP-24),LIR→ mem32(SP-28),	LAR→ mem32(SP-32),	SP - imm8(zero_ext) → SP,PC + 7 → MDR,	PC + d32 → PC	PC + 7 → mem32(SP),reg1 → mem32(SP-4),	reg2 → mem32(SP-8),D0→ mem32(SP-12),	D1→ mem32(SP-16),A0→ mem32(SP-20),	A1→ mem32(SP-24),MDR→ mem32(SP-28),	LIR→ mem32(SP-32),LAR→ mem32(SP-36),	SP - imm8(zero_ext) → SP,PC + 7 → MDR,	PC + d32 → PC	PC + 7 → mem32(SP),reg1→ mem32(SP-4),	reg2→ mem32(SP-8),reg3→ mem32(SP-12),	D0→ mem32(SP-16),D1→ mem32(SP-20),	A0→ mem32(SP-24),A1→ mem32(SP-28),	MDR→ mem32(SP-32),LIR→ mem32(SP-36),	LAR→ mem32(SP-40),	SP - imm8(zero_ext) → SP,PC + 7 → MDR,	PC + d32 → PC
Mnemonic	CALL label PC + 7	reg2→	PC + 7	PC + 7	_reg1→	reg3→	PC + 7	PC + 7	D3→ II	A3→ m	SP - irr	PC + d	PC + 7	D1 ↓ π	A1→ m	LIR→r	SP - irr	PC+d	PC + 7	<u>n</u> ↑00	A0→ m	MDR→	LAR→	SP - irr	PC + d	PC + 7	reg2 →		A1→ m	LIR→r	SP - irr	PC+d	PC + 7	reg2→		A0→ m	MDR→	LAR→	SP - irr	PC+d
Group	CALL																																							

ed with regs =11	with regs =11	with regs =11	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	with regs =11	with regs =11 with regs =0 with regs =1 with regs =2	with regs =11 0 with regs =0 with regs =1 with regs =2 with regs =3 with regs =3	with regs =11 with regs =0 with regs =1 with regs =2 with regs =3	with regs =11 0 0 with regs =0 with regs =1 with regs =2 with regs =3 with regs =3	with regs =11 with regs =0 with regs =1 with regs =2 with regs =3 with regs =4	with regs =11 with regs =0 with regs =2 with regs =3 with regs =3 with regs =4 with regs =7	with regs =11 0 with regs =0 with regs =1 with regs =2 with regs =3 with regs =4 with regs =7	with regs =11 with regs =0 with regs =1 with regs =2 with regs =3 with regs =4 with regs =7	with regs =11 with regs =0 with regs =2 with regs =3 with regs =4 with regs =8	with regs =11 o with regs =0 with regs =2 with regs =3 with regs =4 with regs =7 with regs =8	with regs =11 o with regs =0 with regs =2 with regs =3 with regs =4 with regs =7 with regs =8	with regs =11	with regs =11 with regs =0 with regs =2 with regs =3 with regs =4 with regs =7	with regs =11	with regs =11 0 0 with regs =0 with regs =2 with regs =3 with regs =4 with regs =8	with regs =11 0 0 with regs =0 with regs =2 with regs =3 with regs =4 with regs =8	with regs =11 0 0 with regs =0 with regs =2 with regs =3 with regs =7 with regs =8	with regs =11 0 0 with regs =0 with regs =2 with regs =3 with regs =4 with regs =7 with regs =8
registersspecified with regs =11 *: 5 cycles for AM30																						
			^ ::																			
		^	\\ \frac{1}{2}		\ \frac{1}{1} \\ \frac{1} \	\frac{\frac{1}{1}}{1}	\frac{1}{2}	\[\begin{array}{c c c c c c c c c c c c c c c c c c c	1 1													
	00An	00An 1111 <d16< td=""><td>cd16</td><td><d16 <d32="" <imm8=""></d16></td><td>cd16</td><td><d16 <d32="" <imm8=""></d16></td><td>cd16</td><td>cd16 cd32 climm8></td><td>cd16</td><td><pre><d16 <d32="" <imm8=""></d16></pre></td><td>cd16 cimm8></td><td>cd16 cimm8></td><td>cd16 cimm8></td><td><pre><d16 <d32="" <imm8=""></d16></pre></td><td>cd16 cd32 cimm8></td><td>cd16 cd32 cimm8></td><td>cd16 cd32</td><td>cd16 cd32 cimm8></td><td>4d16 cd16 cd32</td><td>4d16 cd16 cd32</td><td>cd16 cd32 cimm8></td><td>cd16 cd32 cd32</td></d16<>	cd16	<d16 <d32="" <imm8=""></d16>	cd16	<d16 <d32="" <imm8=""></d16>	cd16	cd16 cd32 climm8>	cd16	<pre><d16 <d32="" <imm8=""></d16></pre>	cd16 cimm8>	cd16 cimm8>	cd16 cimm8>	<pre><d16 <d32="" <imm8=""></d16></pre>	cd16 cd32 cimm8>	cd16 cd32 cimm8>	cd16 cd32	cd16 cd32 cimm8>	4d16 cd16 cd32	4d16 cd16 cd32	cd16 cd32 cimm8>	cd16 cd32 cd32
	1111 0000 1111 00An	1111 1010 1111	1111 1000 1111 1110 1111	0000 1111 1010 1111 1100 1111 1111 cregs	0000 1111 1010 1111 1100 1111 1111 cregs	0000 1111 1010 1111 1100 1111 <rees< td=""><td>0000 1111 1010 1111 1100 1111 <rees< td=""><td>1000 1111 1010 1111 1100 1111 <regs< td=""><td>0000 1111 1010 1111 1111 < regs</td><td>0000 1111 1100 1111 <pre>1111 </pre></td><td>1111 1111 Tregs</td><td>0000 1111 1010 1111 < regs</td><td>0000 1111 1100 1111 <pre>1111 </pre></td><td>1111 0000 1111 1100 1111 <rb></rb> 1111 </td><td>1111 0000 1111 1100 1111 <rb></rb> 1111 1111 1111 1111 1111 1111 1111</td><td>1111 0000 1111 1100 1111 < regs</td><td>0000 1111 1010 1111 1110 1111 <rees< td=""><td>1111 0000 1111 1100 1111 <rb></rb> 1111 1111 1111 1111 </td><td>1111 0000 1111 1100 1111 < regs</td><td>1111 0000 1111 1100 1111 <rb></rb> 1111 </td><td>1111 0000 1111 1100 1111 <rb></rb>1111 </td><td>1111 1111 1111 1111 1111 1111 1111 1111 1111</td></rees<></td></regs<></td></rees<></td></rees<>	0000 1111 1010 1111 1100 1111 <rees< td=""><td>1000 1111 1010 1111 1100 1111 <regs< td=""><td>0000 1111 1010 1111 1111 < regs</td><td>0000 1111 1100 1111 <pre>1111 </pre></td><td>1111 1111 Tregs</td><td>0000 1111 1010 1111 < regs</td><td>0000 1111 1100 1111 <pre>1111 </pre></td><td>1111 0000 1111 1100 1111 <rb></rb> 1111 </td><td>1111 0000 1111 1100 1111 <rb></rb> 1111 1111 1111 1111 1111 1111 1111</td><td>1111 0000 1111 1100 1111 < regs</td><td>0000 1111 1010 1111 1110 1111 <rees< td=""><td>1111 0000 1111 1100 1111 <rb></rb> 1111 1111 1111 1111 </td><td>1111 0000 1111 1100 1111 < regs</td><td>1111 0000 1111 1100 1111 <rb></rb> 1111 </td><td>1111 0000 1111 1100 1111 <rb></rb>1111 </td><td>1111 1111 1111 1111 1111 1111 1111 1111 1111</td></rees<></td></regs<></td></rees<>	1000 1111 1010 1111 1100 1111 <regs< td=""><td>0000 1111 1010 1111 1111 < regs</td><td>0000 1111 1100 1111 <pre>1111 </pre></td><td>1111 1111 Tregs</td><td>0000 1111 1010 1111 < regs</td><td>0000 1111 1100 1111 <pre>1111 </pre></td><td>1111 0000 1111 1100 1111 <rb></rb> 1111 </td><td>1111 0000 1111 1100 1111 <rb></rb> 1111 1111 1111 1111 1111 1111 1111</td><td>1111 0000 1111 1100 1111 < regs</td><td>0000 1111 1010 1111 1110 1111 <rees< td=""><td>1111 0000 1111 1100 1111 <rb></rb> 1111 1111 1111 1111 </td><td>1111 0000 1111 1100 1111 < regs</td><td>1111 0000 1111 1100 1111 <rb></rb> 1111 </td><td>1111 0000 1111 1100 1111 <rb></rb>1111 </td><td>1111 1111 1111 1111 1111 1111 1111 1111 1111</td></rees<></td></regs<>	0000 1111 1010 1111 1111 < regs	0000 1111 1100 1111 <pre>1111 </pre>	1111 1111 Tregs	0000 1111 1010 1111 < regs	0000 1111 1100 1111 <pre>1111 </pre>	1111 0000 1111 1100 1111 <rb></rb> 1111	1111 0000 1111 1100 1111 <rb></rb> 1111 1111 1111 1111 1111 1111 1111	1111 0000 1111 1100 1111 < regs	0000 1111 1010 1111 1110 1111 <rees< td=""><td>1111 0000 1111 1100 1111 <rb></rb> 1111 1111 1111 1111 </td><td>1111 0000 1111 1100 1111 < regs</td><td>1111 0000 1111 1100 1111 <rb></rb> 1111 </td><td>1111 0000 1111 1100 1111 <rb></rb>1111 </td><td>1111 1111 1111 1111 1111 1111 1111 1111 1111</td></rees<>	1111 0000 1111 1100 1111 <rb></rb> 1111 1111 1111 1111	1111 0000 1111 1100 1111 < regs	1111 0000 1111 1100 1111 <rb></rb> 1111	1111 0000 1111 1100 1111 <rb></rb> 1111	1111 1111 1111 1111 1111 1111 1111 1111 1111
	- 2 3 D0	- 2 3 D2 - 4 3 D2	. 2 3 3 00 00			2		2		9 2 4 9 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		2 4 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	8 2 4 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	8 2 4 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	8 2 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	8 2 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	8 2 4 5 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0					
	1																					
PC + d32 → PC	PC + 2 → mem32(SP),PC + 2 → MDR,	PC + 2 → mem32(SP),PC + 2 → MDR, An → PC IF (label = (d16,PC)), PC + 4 → mem32(SP), PC + 4 → MDR, PC +d16 (sign_ext) → PC	PC + 2 → mem32(SP),PC + 2 → MDR, An → PC IF (label = (d16,PC)), PC + 4 → mem32(SP), PC + 4 → MDR, PC + 416 (sign_ext) → PC IF ((label = (d32,PC)), PC + 6 → mem32(SP), PC + 6 → MDR, PC + d32 → PC	PC + 2 → mem32(SP), PC + 2 → MDR, An → PC IF (label = (d16,PC)), PC + 4 → mem32(SP), PC + 4 → MDR, PC +d16 (sign_ext) → PC IF ((label = (d32,PC)), PC + 6 → mem32(SP), PC + 6 → MDR, PC +d32 → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC	PC + 2 → mem32(SP),PC + 2 → MDR, An → PC IF (label = (d16.PC)), PC + 4 → mem32(SP), PC + 4 → MDR, PC + 416 (sign_ext) → PC IF ((label = (d32.PC)), PC + 6 → mem32(SP), PC + 6 → MDR, PC + 432 → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(Zero_ext) → SP, mem32(SP 4) → reg, mem32(SP) → PC	PC + 2 → mem32(SP), PC + 2 → MDR, An → PC IF (label = (d16,PC)), PC + 4 → mem32(SP), PC + 4 → MDR, PC +d16 (sign_ext) → PC IF ((label = (d32,PC)), PC + 6 → mem32(SP), PC + 6 → MDR, PC + d32 → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP-4) → reg, mem32(SP) → PC SP + imm8(zero_ext) → SP,	PC + 2 → mem32(SP),PC + 2 → MDR, An → PC IF (label = (d16.PC)), PC + 4 → mem32(SP), PC + 4 → MDR, PC + 416 (sign_ext) → PC IF ((label = (d32.PC)), PC + 6 → mem32(SP), PC + 6 → MDR, PC + 432 → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP 4) → reg, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP 4) → reg1, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP 4) → reg1, mem32(SP -8) → reg2,	PC + 2 → mem32(SP), PC + 2 → MDR, An → PC IF (label = (d16,PC)), PC + 4 → mem32(SP), PC + 4 → MDR, PC + 4d → mem32(SP), PC + 4 → MDR, PC + 6 → mem32(SP), PC + 6 → MDR, PC + 6 → mem32(SP), PC + 6 → MDR, PC + 6 → mem32(SP), PC + 6 → MDR, PC + 6 → mem32(SP), PC + 6 → MDR, PC + 6 → mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP - 4) → reg, mem32(SP - 3 → reg2, mem32(SP - 4) → reg1, mem32(SP - 3 → reg2, mem32(SP - 3 → PC)	PC + 2 → mem32(SP),PC + 2 → MDR, An → PC IF (label = (d16.PC)), PC + 4 → mem32(SP), PC + 4 → MDR, PC + 416 (sign_ext) → PC IF ((label = (d32.PC)), PC + 6 → mem32(SP), PC + 6 → MDR, PC + 432 → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP 4) → reg, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC	PC + 2 → mem32(SP),PC + 2 → MDR, An → PC IF (label = (d16,PC)), PC + 4 → mem32(SP), PC + 4 → MDR, PC + 416 (sign_ext) → PC IF ((label = (d32,PC)), PC + 6 → mem32(SP), PC + 6 → MDR, PC + 632 → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP - 4) → reg, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 9) → PC SP + imm8(zero_ext) → SP, mem32(SP - 12) → reg3, mem32(SP) → PC SP + imm8(zero_ext) → SP,	PC + 2 → mem32(SP),PC + 2 → MDR, An → PC IF (label = (d16.PC)), PC + 4 → mem32(SP), PC + 4 → MDR, PC + 410 ∈ (sign_ext) → PC IF (label = (d32.PC)), PC + 6 → mem32(SP), PC + 6 → MDR, PC + 432 → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP 4) → reg, mem32(SP -8) → reg2, mem32(SP 4) → reg1, mem32(SP-8) → reg2, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP-12) → reg3, mem32(SP-8) → reg2, mem32(SP-12) − reg3, mem32(SP-9) → reg2, mem32(SP-12) − reg3, mem32(SP-9) → PC SP + imm8(zero_ext) → SP, mem32(SP-12) − reg3, mem32(SP-9) → PC SP + imm8(zero_ext) → SP, mem32(SP-12) − reg3, mem32(SP-9) → PC SP + imm8(zero_ext) → SP, mem32(SP-12) − reg3, mem32(SP-9) → D3, mem32(SP-12) → D2, mem32(SP-8) → D3,	PC + 2 → mem32(SP),PC + 2 → MDR, An → PC IF (label = (d16.PC)), PC + 4 → mem32(SP), PC + 4 → MDR, PC + 416 (sign_ext) → PC IF ((label = (d32.PC)), PC + 6 → mem32(SP), PC + 6 → MDR, PC + 432 → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP - 4) → reg, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → PC SP + imm8(zero_ext) → SP, mem32(SP - 4) → D2, mem32(SP - 8) → D3, mem32(SP - 12) → A2, mem32(SP - 16) → A3, mem32(SP - 12) → PC	PC + 2 → mem32(SP),PC + 2 → MDR, An → PC IF (label = (d16,PC)), PC + 4 → mem32(SP), PC + 4 → MDR, PC + 416 (sign_ext) → PC IF ((label = (d32,PC)), PC + 6 → mem32(SP), PC + 6 → MDR, PC + 632 → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP - 4) → reg, mem32(SP - 8) → reg, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → PC SP + imm8(zero_ext) → SP, mem32(SP - 4) → D2, mem32(SP - 16) → A3, mem32(SP - 4) → D2, mem32(SP - 16) → A3, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP - 4) → D2, mem32(SP - 16) → A3, mem32(SP) → PC SP + imm8(zero_ext) → SP,	PC + 2 → mem32(SP), PC + 2 → MDR, An → PC IF (label = (d16,PC)), PC + 4 → mem32(SP), PC + 4 → MDR, PC + 410 ∈ (sign_ext) → PC IF (label = (d32,PC)), PC + 6 → mem32(SP), PC + 6 → MDR, PC + 632 → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP - 4) → reg, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → PC SP + imm8(zero_ext) → SP, mem32(SP - 4) → D2, mem32(SP - 8) → D3, mem32(SP - 4) → D2, mem32(SP - 16) → A3, mem32(SP - 4) → D0, mem32(SP - 16) → A3, mem32(SP - 4) → D0, mem32(SP - 8) → D1,	PC + 2 → mem32(SP), PC + 2 → MDR, An → PC IF (label = (d16,PC)), PC + 4 → mem32(SP), PC + 4 → MDR, PC + 410 ∈ (sign_ext) → PC IF ((label = (d32,PC)), PC + 6 → mem32(SP), PC + 6 → MDR, PC + 6 32 → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP - 4) → reg, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → PC SP + imm8(zero_ext) → SP, mem32(SP - 4) → D2, mem32(SP - 16) → A3, mem32(SP - 4) → D2, mem32(SP - 16) → A3, mem32(SP - 4) → D0, mem32(SP - 16) → A1, mem32(SP - 4) → D0, mem32(SP - 16) → A1,	PC + 2 → mem32(SP), PC + 2 → MDR, An → PC IF (label = (d16,PC)), PC + 4 → mem32(SP), PC + 4 → MDR, PC + 410 ∈ (sign_ext) → PC IF ((label = (d32,PC)), PC + 6 → mem32(SP), PC + 6 → MDR, PC + 6 32 → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP - 4) → reg, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → PC SP + imm8(zero_ext) → SP, mem32(SP - 4) → D2, mem32(SP - 8) → D3, mem32(SP - 4) → D2, mem32(SP - 16) → A3, mem32(SP - 4) → D0, mem32(SP - 16) → A1, mem32(SP - 4) → D0, mem32(SP - 16) → A1, mem32(SP - 12) → A0, mem32(SP - 16) → A1, mem32(SP - 20) → MDR, mem32(SP - 24) → LIR,	PC + 2 → mem32(SP),PC + 2 → MDR, An → PC IF (label = (d16,PC)), PC + 4 → mem32(SP), PC + 4 → MDR, PC + 416 (sign_ext) → PC IF ((label = (d32,PC)), PC + 6 → mem32(SP), PC + 6 → MDR, PC + 632 → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP - 4) → reg, mem32(SP - 8) → reg, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → D3, mem32(SP - 4) → D2, mem32(SP - 16) → A3, mem32(SP - 4) → D0, mem32(SP - 16) → A4, mem32(SP - 4) → D0, mem32(SP - 16) → A1, mem32(SP - 20) → MDR, mem32(SP - 24) → LIR, mem32(SP - 20) → MDR, mem32(SP - 24) → LIR, mem32(SP - 20) → MDR, mem32(SP - 24) → LIR, mem32(SP - 28) → LAR, mem32(SP - 24) → LIR,	PC + 2 → mem32(SP), PC + 2 → MDR, An → PC IF (label = (d16,PC)), PC + 4 → mem32(SP), PC + 4 → MDR, PC + 410 ∈ (sign_ext) → PC IF ((label = (d32,PC)), PC + 6 → mem32(SP), PC + 6 → MDR, PC + 6 32 → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP - 4) → reg, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → PC SP + imm8(zero_ext) → SP, mem32(SP - 4) → PC SP + imm8(zero_ext) → SP, mem32(SP - 4) → D2, mem32(SP - 16) → A3, mem32(SP - 4) → D0, mem32(SP - 16) → A4, mem32(SP - 20) → MDR, mem32(SP - 24) → LIR, mem32(SP - 20) → MDR, mem32(SP - 24) → LIR, mem32(SP - 28) → LAR, mem32(SP - 24) → LIR, mem32(SP - 28) → LAR, mem32(SP - 24) → LIR, mem32(SP - 28) → LAR, mem32(SP - 24) → LIR, mem32(SP - 28) → LAR, mem32(SP - 24) → LIR, mem32(SP - 28) → LAR, mem32(SP - 24) → LIR, mem32(SP - 28) → LAR, mem32(SP - 24) → LIR, mem32(SP - 28) → LAR, mem32(SP - 28) → PC	PC + 2 → mem32(SP), PC + 2 → MDR, An → PC IF (label = (d16,PC)), PC + 4 → mem32(SP), PC + 4 → MDR, PC + 41 → mem32(SP), PC + 4 → MDR, PC + 40 ← mem32(SP), PC + 6 → MDR, PC + 6 → mem32(SP), PC + 6 → MDR, PC + 632 → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → PC SP + imm8(zero_ext) → SP, mem32(SP - 4) → reg1, mem32(SP - 8) → PC SP + imm8(zero_ext) → SP, mem32(SP - 4) → D2, mem32(SP - 8) → D3, mem32(SP - 4) → D2, mem32(SP - 16) → A3, mem32(SP - 20) → MDR, mem32(SP - 24) → LIR, mem32(SP - 20) → MDR, mem32(SP - 24) → LIR, mem32(SP - 28) → LAR, mem32(SP - 24) → LIR, mem32(SP - 28) → LAR, mem32(SP - 24) → LIR, mem32(SP - 4) → reg1, mem32(SP - 24) → D0,	PC + 2 → mem32(SP), PC + 2 → MDR, An → PC IF (label = (d16,PC)), PC + 4 → mem32(SP), PC + 4 → MDR, PC + 41 → mem32(SP), PC + 4 → MDR, PC + 40 ← mem32(SP), PC + 6 → MDR, PC + 6 → mem32(SP), PC + 6 → MDR, PC + 632 → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP - 4) → reg, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → PC SP + imm8(zero_ext) → SP, mem32(SP - 4) → reg1, mem32(SP - 8) → D3, mem32(SP - 4) → D2, mem32(SP - 8) → D3, mem32(SP - 4) → D2, mem32(SP - 16) → A3, mem32(SP - 4) → D0, mem32(SP - 16) → A1, mem32(SP - 20) → MDR, mem32(SP - 24) → LIR, mem32(SP - 20) → MDR, mem32(SP - 24) → LIR, mem32(SP - 4) → reg1, mem32(SP - 9) → PC SP + imm8(zero_ext) → SP, mem32(SP - 4) → reg1, mem32(SP - 9) → D0, mem32(SP - 4) → reg1, mem32(SP - 9) → D0, mem32(SP - 4) → reg1, mem32(SP - 9) → D0, mem32(SP - 10) → MDR, mem32(SP - 10) → A0, mem32(SP - 10) → MDR, mem32(SP - 10) → A0, mem32(SP - 10) → D1, mem32(SP - 10) → A0,	PC + 2 → mem32(SP), PC + 2 → MDR, An → PC IF (label = (d16,PC)), PC + 4 → mem32(SP), PC + 4 → MDR, PC + 41 → mem32(SP), PC + 4 → MDR, PC + 46 → mem32(SP), PC + 6 → MDR, PC + 6 → mem32(SP), PC + 6 → MDR, PC + 632 → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP - 4) → reg, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → reg2, mem32(SP - 4) → reg1, mem32(SP - 8) → PC SP + imm8(zero_ext) → SP, mem32(SP - 4) → reg1, mem32(SP - 8) → D3, mem32(SP - 4) → D2, mem32(SP - 16) → A3, mem32(SP - 4) → D2, mem32(SP - 16) → A4, mem32(SP - 4) → D0, mem32(SP - 16) → A1, mem32(SP - 4) → reg1, mem32(SP - 8) → D1, mem32(SP - 4) → reg1, mem32(SP - 8) → D1, mem32(SP - 4) → reg1, mem32(SP - 8) → D0, mem32(SP - 4) → reg1, mem32(SP - 8) → D0, mem32(SP - 4) → reg1, mem32(SP - 9) → A0, mem32(SP - 4) → reg1, mem32(SP - 8) → D0, mem32(SP - 12) → A1, mem32(SP - 8) → D0, mem32(SP - 12) → A1, mem32(SP - 8) → M0R,	PC + 2 → mem32(SP), PC + 2 → MDR, An → PC IF (label = (d16,PC)), PC + 4 → mem32(SP), PC + 4 → MDR, PC + 4d → mem32(SP), PC + 4 → MDR, PC + 4d → mem32(SP), PC + 6 → MDR, PC + 6 → mem32(SP), PC + 6 → MDR, PC + 6d > mem32(SP), PC + 6 → MDR, PC + 6d > mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP) → PC SP + imm8(zero_ext) → SP, mem32(SP + 4) → reg1, mem32(SP - 8) → reg2, mem32(SP + 4) → reg1, mem32(SP - 8) → reg2, mem32(SP + 4) → reg1, mem32(SP - 8) → reg2, mem32(SP + 4) → reg1, mem32(SP - 8) → reg2, mem32(SP + 4) → reg1, mem32(SP - 8) → PC SP + imm8(zero_ext) → SP, mem32(SP + 12) → reg1, mem32(SP - 8) → D3, mem32(SP + 12) → A2, mem32(SP - 8) → D3, mem32(SP + 4) → D2, mem32(SP - 16) → A3, mem32(SP + 12) → A2, mem32(SP - 16) → A1, mem32(SP + 20) → MDR, mem32(SP - 24) → LIR, mem32(SP + 12) → reg1, mem32(SP - 24) → D0, mem32(SP + 12) → reg1, mem32(SP - 8) → D0, mem32(SP + 12) → reg1, mem32(SP - 8) → D0, mem32(SP + 20) → A1, mem32(SP - 24) → MDR, mem32(SP - 20) → A1, mem32(SP - 24) → MDR, mem32(SP - 20) → A1, mem32(SP - 24) → MDR, mem32(SP - 20) → A1, mem32(SP - 24) → MDR,
	CALLS (An)	CALLS (An)	CALLS (An)	CALLS (An) CALLS label RET	CALLS (An) CALLS label RET	CALLS (An) CALLS label RET	CALLS (An) CALLS label RET	CALLS (An) CALLS label RET	CALLS (An) CALLS label RET	CALLS (An) CALLS label RET	CALLS (an) RET	CALLS (an) RET	CALLS (an) RET	CALLS (an) RET	CALLS (An) RET	CALLS (An) CALLS label	CALLS (An) RET	CALLS (an) RET	CALLS (An) RET	CALLS (An) RET	CALLS (An) RET	

Page]	89						89							89							06	90		90		90			90			8					8				_
																													1			T										
S		regs = 9						registers specified with regs= 10							registers specified with regs= 11							register specified with regs = 0	egs= 1		registers specified with regs = 2		registers specified with regs= 3			registers specified with regs= 4			registers specified with regs = 7					registers specified with regs = 8				
Notes		registers specified with regs						ed with							ed with							d with r	register specified with regs= 1		ed with		ed with			ed with			ed with					ed with				
		specifi						specifi							specifi							specifie	specifie		specifi		specifi			specifi			specifi					specifi				
		egisters						egisters							egisters							egister :	egister (egisters		egisters			egisters			egisters					egisters				
4		2						2							2							2	2		2		2			2			2					2				
13																																										
12																																										
=																																										
10																																										
6																																										
_ ∞																																										
Machine Code																																										
Machir 6		< 8mmi>																				ر ه																				
5		vim																				<imm8< td=""><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></imm8<>																				
4		۲ S																				<u>د</u>																				
m		<re>sees</re>																				cregs																				
8		1111																				1110																				
-	1	1101																				1101																				
Flag Code Cycle For VF CF NF 7F Size		10 S2						11							12							2 \$2	2		8		4			2			80					6				
Code C)	270	3						3							3							т	ю		ю		m			m		\dashv	m					т				
TH.	į	-						,															,				,						•									
Flag		·						٠														٠	٠				٠											٠				
E 5	5	·						•							•							·	٠				•			•			•					٠				
	1	Ŀ													1							١.			•		•			•			•			_						
			eg2,	٦,	Α,	↓ LR	ي		eg2,	▶ D0,	A0,	MDR,	LAR,			_	A3,	٦,	٨٦,	↓ LIR	Q		_		_	eg2,	_	eg2,		_	,	A3,	_	_	٨,	↓ LIR			,00	A V	MDR,	LAR,
			3)→ re	^(9	4)	-(32)-	↑ 1		3)→ re	-16)-	4.	12)→ 1	40)→			↑ D3	, (9	(4:	(2)	-40)-	1	PC ↑	^ BC		^ BC	3)→ re	ړ ۲	3)→ r.		^ PG	<u>↑</u>	(9)	^ PC	7	(9)	-24)-		P P P	3) ↓ □	<u></u> (9	4 (4)	32)→
			?(SP-	(SP-1	(SP-2	32(SF	32(SP		?(SP-≀	32(SP	,(SP-2	(SP-3	2(SP-			SP-8)	(SP-1	(SP-2	(SP-3	32(SF	32(SP	MDR	MDR.		MDR	3(SP-≀	MDR.	?(SP-≀		MDR.	SP-8)	(SP-1	MDR.	SP-8)	(SP-1	32(SF		MDR.	3(SP-	(SP-1	(SP-2	2(SP-
ation		Š.	em32	em32	em32	mem	mem3	SP,	em32	memŝ	em32	em32.	em3		SP,	m32(\	em32,	em32	em32	mem.	mem3	, SP,	, SP,		, SP,	em32	SP,	em32		SP,	m32(em32	SP, I	m32(\	em32.	mem		, SP,	em32	em32	em32	nem32
Operation		T (xx	∍g1,m	DO,m	A0,m	MDR,	LAR,ı	³xt) ⊥	∌g1,m	reg3,1	л, П	A1,m	LIR,n	,.	ĭxt) ⊥	'2,me	A2,m	D0,m	A0,m	MDR,	LAR,) tx	,xt) →	ρ	,xt) ↓	∍g1,m	³xt) ⊥	∍g1,m.	reg3	ĭxt)	2,me	A2,m	₃xt) ⊥	0,me	A0,m	MDR,	LAR)xt)	₃g1,m	л, П	A1,m	LIR,n
		ero_6	4) → Γε	12)→	(0∑	28)→	36)→	:ero_e	‡)→ Γε	12)→	20)→	28)→,	. ←(98	↑ PC	:ero_e	†) → (t	12)→,	20)→	38)→	36)→	4	ero_e	:ero_e	1) → Γε	:ero_e	1)→ re	ero_e	#) ↑	12)→	:ero_e	4)→□	12)→	:ero_e	† (†	12)→,	(0:	28)→	ero_e	1) → Γε	12)→	20)→	38)→
		z)8mr	,(SP-4	(SP-1	(SP-	,(SP-	3-4S)	z)8mr	,(SP-4	1(SP-1	,(SP-	,(SP-	3-4S)	(SP)	z)8mr	,(SP-4	(SP-1	(SP-2	S-4S)	(SP-3	(SP-4	z)8mr	z)8mr	(SP-4	m8(z	SP-4	z)8mr	SP-4	(SP-1	z)8mr	(SP-4	SP-	z)8mr	,(SP-4	1(SP-1	(SP-2	(SP-2	m8(z	(SP-4	(SP-1	(SP-	(SP-2
		SP + imm8(zero_ext) → SP,	mem32(SP-4)→ reg1,mem32(SP-8)→ reg2,	mem32(SP-12)→ D0,mem32(SP-16)→ D1,	$\mathrm{mem32}(\mathrm{SP}\text{-}20) {\rightarrow} \ \mathrm{A0,mem32}(\mathrm{SP}\text{-}24) {\rightarrow} \ \mathrm{A1,}$	$mem32(SP\text{-}28) \!$	$mem32(SP\text{-}36) \!$	SP + imm8(zero_ext) → SP,	mem32(SP-4) \rightarrow reg1,mem32(SP-8) \rightarrow reg2,	$mem32(SP\text{-}12) \!\!\rightarrow\! reg3, mem32(SP\text{-}16) \!\!\rightarrow\! D0,$	$mem32(SP-20) \rightarrow D1, mem32(SP-24) \rightarrow A0,$	$mem32(SP\text{-}28) \!$	$mem32(SP\text{-}36) \!$	$mem32(SP) \to PC$	SP + imm8(zero_ext) → SP,	mem32(SP-4) \rightarrow D2,mem32(SP-8) \rightarrow D3,	mem32(SP-12)→ A2,mem32(SP-16)→ A3,	mem32(SP-20)→ D0,mem32(SP-24)→ D1,	mem32(SP-28)→ A0,mem32(SP-32)→ A1,	mem32(SP-36)→ MDR,mem32(SP-40)→ LIR,	mem32(SP-44)→ LAR,mem32(SP) → PC	SP + imm8(zero_ext) → SP,MDR → PC.	SP + imm8(zero_ext) → SP,MDR → PC,	mem32(SP-4)→ reg	SP + imm8(zero_ext) → SP,MDR → PC,	mem32(SP-4)→ reg1,mem32(SP-8)→ reg2,	$SP + imm8(zero_ext) \rightarrow SP, MDR \rightarrow PC,$	mem32(SP-4)→ reg1,mem32(SP-8)→ reg2,	mem32(SP-12)→ reg3	$SP + imm8(zero_ext) \rightarrow SP, MDR \rightarrow PC,$	mem32(SP-4)→ D2,mem32(SP-8)→ D3,	mem32(SP-12)→ A2,mem32(SP-16)→ A3,	$SP + imm8(zero_ext) \rightarrow SP, MDR \rightarrow PC,$	mem32(SP-4)→ D0,mem32(SP-8)→ D1,	$mem32(SP\text{-}12) \!$	mem32(SP-20)→ MDR,mem32(SP-24)→ LIR,	mem32(SP-28)→ LAR	SP + imm8(zero_ext) → SP,MDR → PC,	mem32(SP-4)→ reg1,mem32(SP-8)→ D0,	$\mathrm{mem32}(\mathrm{SP}\text{-}12) \!\!\rightarrow\! \mathrm{D1,mem32}(\mathrm{SP}\text{-}16) \!\!\rightarrow\! \to \mathrm{A0,}$	$\text{mem32(SP-20)} \rightarrow \text{A1,mem32(SP-24)} \rightarrow \text{MDR,}$	mem32(SP-28)→ LIR,mem32(SP-32)→ LAR,
		S		_			⊏	S						∟	S					_		S	S	_	<u></u> ω	۲	S	_	=	v	_	=	S					lω			_	_
Mnemonic																																										
Mne		_																				 <u> </u>																				
dņ	$\left\{ \right.$	RET																				F RETF																				
Group	J	RET																				RETF																				

PTT	.: 4 .: 0 2 2	· · · · · · · · · · · · · · · · · · ·	1	3 10	0 82	1101	1110 <regs< th=""><th>^</th><th><imm8></imm8></th><th></th><th></th><th>2</th><th>registers specified with regs = 9</th><th>06</th></regs<>	^	<imm8></imm8>			2	registers specified with regs = 9	06
								^ :	:imm8>			<u> </u>	egisters specified with regs = 9	06
	_													
	_				_									
	_													
		1												
		1		3 11								2	registers specified with regs= 10	90
		1												
	αž													
G B B														
a ae	rem32(SP-4)→ D2,mem32(SP-8)→ D3, rem32(SP-12)→ A2,mem32(SP-16)→ A3, rem32(SP-20)→ D0,mem32(SP-24)→ D1,			3 10								2	registers specified with regs= 11	06
me m	lem32(SP-12)→ A2, mem32(SP-16)→ A3, lem32(SP-20)→ D0, mem32(SP-24)→ D1,													
aw.	lem32(SP-20) → D0,mem32(SP-24) → D1,													
?:-	700 00/00													
me	mem32(SP-28)→ A0,mem32(SP-32)→ A1,													
me	mem32(SP-36)→ MDR,mem32(SP-40)→ LIR,													
me	mem32(SP-44)→ LAR													
RETS me	mem32(SP) → PC		·	2 5*	8	1111	0000 1111	1100				**	*: 4 cycles for AM30	91
JSR (An) SP	SP - 4 → SP, PC + 2 → mem32(SP)	•	•	8										92
PC	PC + 2 → MDR,An → PC,													
	(execute subroutine)													
SP	SP + 4 → SP													
JSR label IF (IF (label = (d16,PC)),	•	•	10 5										93
SP	SP - 4 → SP,PC + 4 → mem32(SP),													
PC	PC + 4 → MDR, PC + d16 (sign_ext) → PC													
	(execute subroutine)													
SP	SP+4 → SP													
<u> </u>	IF (label = (d32,PC)), ●	•	•	12 5*								*.	*: 6 cycles for AM30	93
S.	SP - 4 → SP, PC + 6 → (SP+3),													
PC	PC + 6 → MDR, PC + d32 → PC													
	(execute subroutine)													
SP	SP+4 → SP													
RTS me	mem32(SP) → PC		•	2 4	D0	1111	0000 1111	1101						94
RTI me	mem16(SP) → PSW,mem32(SP+4) → PC,	•	•	2 4										92
SP	SP + 8 → SP													
TRAP	PC + 2 → mem32(SP),0x40000010 → PC	-	-	2 4	8	1111	0000 1111	1110						96
NOP	PC+1→PC	•	1	_	So	1100	1011							97

Nigotion	Group	Mnemonic	Operation		Flag		Code	Ovcle Fo						Machir	1						Notes	Page
Dunop Du → Du → Du Dunop Du → Du Dunop Du → Du Dunop Du → Dunop Dun				Ŋ Ĭ	z L	F ZF	Size	ڄ د		7	m	4	2	9	- 1		12	13	44			
UPG2 DmDn Dm op Dn → Dn P P P P D 1111 0110 0100 000 DmD UPG2 DmDn Dm op Dn → Dn P P P P P D 1111 0110 0110 0010 DmD DmD UPG2 DmDn Dm op Dn → Dn P P P P D 1111 0110 0110 0110 DmD DmD <t< td=""><td>Extens</td><td>ion Instructions</td><td></td><td></td><td>ŀ</td><td></td><td>-</td><td>}</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>-</td><td></td><td></td></t<>	Extens	ion Instructions			ŀ		-	}												-		
Dmog Dn → Dn	UDF	UDF00 Dm,Dn	Dm op Dn → Dn	*	*	*	2	*	111				5									98
Dunog Du → Dun → Dunog Du → Dunog Dun g Dun → Dunog Dunog Dun → Dunog Dunog Dunog Dun → Dunog Dunog Dunog Dunog Dun → Dunog		UDF01 Dm,Dn	Dm op Dn → Dn	*			2	*					5									98
Dun op Du → Dun		UDF02 Dm,Dn		*	*		2	*					5									86
Dun op Dun → Dun → Dun Dun → Dun Dun op Dun → Dun Dun → Dun Dun op Dun → Dun → Dun Dun → Dun		UDF03 Dm,Dn	Dm op Dn → Dn	*			2	*	111				۵									98
Dun op Dn → Dn		UDF04 Dm,Dn	Dm op Dn → Dn	*	*		2	*	111	l			<u>ا</u>									86
Demograph → Dem		UDF05 Dm,Dn	Dm op Dn → Dn	*			_	*					۵									98
Dmop Dn → Dn → Dn Dmop Dn Dmo		UDF06 Dm,Dn	Dm op Dn → Dn	*			2	*					5									86
Dmop Dh → Dh		UDF07 Dm,Dn	Dm op Dn → Dn	*			2	*					5									86
Dmop Dn → Dn Dmop Dn Dmo		UDF08 Dm,Dn	Dm op Dn → Dn	*	*		2	*					<u>ا</u>									86
Dmop Dn → Dn Dmop Dn		UDF09 Dm,Dn	Dm op Dn → Dn	*			2	*					5									86
Dm op Dn → Dn C C C C Dn 1111 0110 1011 DmD Dn Dm op Dn → Dn Dm C C C D 1111 0110 1101 DmD Dn Dm op Dn → Dn Dm C C C D 1111 0110 110 DmD Dn Dm op Dn → Dn Dm Dn 1111 0110 0110 1111 0110 0110 DmD Dn Dm op Dn Dm op Dn Dm Dn 1111 0101 001 DmD Dn 001 1111 0101 0101 DmD Dn Dm op Dn Dm op Dn C C C C C D 1111 0101 0101 DmD Dn Dm op Dn Dm op Dn C C C C C C D 1111 0101 1010 DmD Dn Dm op Dn Dm Dm 1111 0101 0101 1011 0101 0101		UDF10 Dm,Dn	Dm op Dn → Dn	*			2	*		l	l	l	5									86
Dm op Dn → Dn N		UDF11 Dm,Dn	Dm op Dn → Dn	*			2	*	io 111				<u>۾</u>									86
Dm op Dn → Dn • • • • • • • 1111 0110 1101		UDF12 Dm,Dn	Dm op Dn → Dn	*	*		2	*	0 111				5									86
Dm op Dn → Dn T T T Dn 111 011		UDF13 Dm,Dn	Dm op Dn → Dn	*	*		2	*	io 111				5									86
Dm op Dn → Dn C		UDF14 Dm,Dn	Dm op Dn → Dn	*			2	*		l			5									86
Dm op Dn C C C C D 1111 0101 0000 DmDn Dm op Dn C C C C C D 1111 0101 0001 DmDn Dm op Dn C C C C C D 1111 0101 0001 DmDn Dm op Dn C C C C C C D 1111 0101 0101 DmDn Dm op Dn Dm op Dn C C C C C D 1111 0101 0101 DmDn Dm op Dn Dm op Dn C C C C C D 1111 0101 0101 DmDn Dm op Dn Dm op Dn Dm op Dn C C C C D 1111 0101 0101 DmDn Dm op Dn		UDF15 Dm,Dn		*			2	*					<u>۾</u>									86
Dm op Dn		UDF20 Dm,Dn	Dm op Dn			1	2	*	io 111				5									86
Dm op Dn C C C C D 1111 0101 070 Dm/D Dm op Dn C C C C C D 1111 0101 0101 Dm/D Dm op Dn C C C C C D 1111 0101 0101 Dm/D Dm op Dn Dm op Dn C C C C C D 1111 0101 0101 Dm/D Dm op Dn Dm op Dn C C C C C D 1111 0101 0101 Dm/D Dm op Dn Dm op Dn Dm C C C C C D 1111 0101 1010 Dm/D Dm op Dn Dm Dm Dm Dm Dm 1111 0101 1111 0101 1111 0101 1111 0101 1111 0101 1111 0101 1111 0101 0101 0101 </td <td></td> <td>UDF21 Dm,Dn</td> <td>Dm op Dn</td> <td></td> <td></td> <td></td> <td>2</td> <td>*</td> <td>111</td> <td></td> <td></td> <td></td> <td>5</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>98</td>		UDF21 Dm,Dn	Dm op Dn				2	*	111				5									98
Dm op Dn C C C C D 1111 0101 Dm Op Dn Dm op Dn C C C C C D 1111 0101 0100 DmDn Dm op Dn C C C C C D 1111 0101 0101 DmDn Dm op Dn C C C C C C D 1111 0101 0101 DmDn Dm op Dn Dm op Dn C C C C C D 1111 0101 1010 DmDn Dm op Dn Dm op Dn C C C C C D 1111 0101 1010 DmDn Dm op Dn Dm op Dn Dm C C C C C D 1111 0101 1101 DmDn Dm op Dn Dm Dn Dn 1111 0101 1111 0101 1111 0101		UDF22 Dm,Dn	Dm op Dn				2	*					<u>۾</u>									86
Dm op Dn C C C C D 1111 0101 0100 DmDn Dm op Dn C C C C C D 1111 0101 0101 DmDn Dm op Dn C C C C C D 1111 0101 0111 DmDn DmDn Dm op Dn C C C C C D 1111 0101 1011 DmDn Dm op Dn Dm op Dn C C C C D 1111 0101 1010 DmDn Dm op Dn Dm op Dn C C C C C D 1111 0101 1010 DmDn Dm op Dn Dm op Dn Dm C C C C C D 1111 0101 1111 DmDn Imm8(sign_ext) op Dn → Dn C C C C C C D D 1111		UDF23 Dm,Dn	Dm op Dn				2	*					۵									98
Dm op Dn -		UDF24 Dm,Dn	Dm op Dn	•			2	*	111				5									98
Dm op Dn -		UDF25 Dm,Dn	Dm op Dn	•	- 1	1	2	*	111				5									98
Dm op Dn - - - - 2 - Do 1111 0101 0111 Dm Op Dn Dm op Dn - - - - 2 - Do 1111 0101 1000 DmDn Dm op Dn - - - - - - 2 - D0 1111 0101 1001 DmDn Dm op Dn - - - - - - 2 - D0 1111 0101 1011 DmDn Dm op Dn -		UDF26 Dm,Dn	Dm op Dn				2	*					5									98
Dm op Dn - - - - 2 - Do 1111 0101 1000 DmDn Dm op Dn - <t< td=""><td></td><td>UDF27 Dm,Dn</td><td>Dm op Dn</td><td></td><td></td><td></td><td>2</td><td>*</td><td></td><td></td><td></td><td></td><td>۵</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>98</td></t<>		UDF27 Dm,Dn	Dm op Dn				2	*					۵									98
Dm op Dn - - - - 2 - Do 1111 0101 1001 Dm Op Dn Dm op Dn - - - - 2 - Do 1111 0101 1010 Dm Op Dn Dm op Dn -		UDF28 Dm,Dn	Dm op Dn			1	2	*	i0 111				5									98
Dm op Dn - - - - 2 - Do 1111 0101 1010 Dm Dn Dm op Dn - - - - - 2 - D0 1111 0101 1101 DmDn Dm op Dn - - - - - - 2 - D0 1111 0101 1101 DmDn Dm op Dn Dm - - - - - 2 - D0 1111 0101 1101 DmDn Inmak(sign_ext) op Dn → Dn -		UDF29 Dm,Dn	Dm op Dn				2	*	111				۵									98
Dm op Dn - - - - 2 - D0 1111 0101 1011 Dm Dn Dm op Dn - <td< td=""><td></td><td>UDF30 Dm,Dn</td><td>Dm op Dn</td><td></td><td></td><td>•</td><td>2</td><td>*</td><td></td><td></td><td></td><td></td><td>5</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>98</td></td<>		UDF30 Dm,Dn	Dm op Dn			•	2	*					5									98
Dm op Dn - - - - 2 + Do 1111 0101 1100 1100 DmDn Dm op Dn - <		UDF31 Dm,Dn	Dm op Dn	•		1	2	*					5									98
Dm op Dn - - - - 2 + Do 1111 0101 1101 Dm Dn Dm op Dn - <		UDF32 Dm,Dn	Dm op Dn	•	- 1	1	2	*	111				5									86
Dm op Dn - - - - 2 + Do 1111 0101 1111 DmDn Inme8(sign_ext) op Dn - - - - - - - 0 1111 1001 1111 DmDn Inme8(sign_ext) op Dn -Dn - - - - - 0 0 1111 1001 1111 DmDn Inme8(sign_ext) op Dn -Dn - - - - - 0 1111 1001 00Dn 0Dn Inme8(sign_ext) op Dn -Dn - - - - - - 0 1111 1001 0Dn Inme8(sign_ext) op Dn -Dn - - - - - - 0		UDF33 Dm,Dn	Dm op Dn		•	1	2	*	0 111	- 1			5									86
Dm op Dn - - - - 2 + D0 1111 0101 1111 DmDn imm8(sign_ext) op Dn → Dn - - - - - - - - 01 1111 1001 0000 00Dn imm8(sign_ext) op Dn → Dn - </td <td></td> <td>UDF34 Dm,Dn</td> <td>Dm op Dn</td> <td></td> <td></td> <td>1</td> <td>2</td> <td>*</td> <td></td> <td></td> <td></td> <td></td> <td><u>۾</u></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>98</td>		UDF34 Dm,Dn	Dm op Dn			1	2	*					<u>۾</u>									98
imm8(sign_ext) op Dn → Dn * * * * * * * * * * * * * * * * * * *		UDF35 Dm,Dn	Dm op Dn		•	-	2	*					5									86
imm8(sign_ext) op Dn → Dn r </td <td></td> <td>UDF00 imm8,Dn</td> <td>imm8(sign_ext) op Dn → Dn</td> <td>*</td> <td></td> <td></td> <td></td> <td>*</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td><></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>66</td>		UDF00 imm8,Dn	imm8(sign_ext) op Dn → Dn	*				*						<>								66
imm8(sign_ext) op Dn → Dn * * * * * * 01 1111 1001 0010 00Dn imm8(sign_ext) op Dn → Dn * * * * * * * * 01 1111 1001 001 00Dn imm8(sign_ext) op Dn → Dn * * * * * * * 01 1111 1001 010 00Dn imm8(sign_ext) op Dn → Dn * * * * * * 01 1111 1001 0100 00Dn		UDF01 imm8,Dn	imm8(sign_ext) op Dn → Dn	*	*	*	က	*	111				- 1	<>8mt								66
imm8(sign_ext) op Dn → Dn * * * * * * * 01 1111 1001 0011 00Dn imm8(sign_ext) op Dn → Dn * * * * * * * 01 1111 1001 0100 00Dn imm8(sign_ext) op Dn → Dn * * * * * * 01 1111 1001 0101 0101 00Dn		UDF02 imm8,Dn	imm8(sign_ext) op Dn → Dn	*	-		3	*	-			- 1	- 1	<>mr								66
imm8(sign_ext) op Dn → Dn * * * * * * D1 1111 1001 0100 00Dn inm8(sign_ext) op Dn → Dn * * * * * * * 01 1111 1001 0101 00Dn inm8(sign_ext) or Dn → Dn Dn * * * * * 1141 1001 0101 00Dn		UDF03 imm8,Dn		*	*	*	က	*	-					×								66
imm8(sign_ext) op Dn → Dn		UDF04 imm8,Dn	imm8(sign_ext) op Dn → Dn	*	-		8	*	-					<>mı								66
imm8(sign axt) on Dn → Dn → C + + + + + + 13 + 111 1001 0110 0100		UDF05 imm8,Dn	imm8(sign_ext) op Dn → Dn	*	*	*	е	*					- 1	<>mı								66
		UDF06 imm8,Dn	imm8(sign_ext) op Dn → Dn	*	*	*	3	*	111	- 1	- 1	- 1	- 1	<								66

Mnemonic	Operation	VF	VF CF NF ZF Size	FIZF	Size	-mat	ع ا -	7	3	4	c	2 9	- 1	8 9 1	10 11	12	13	14		
	-	-																	-	-
UDF07 imm8,Dn	imm8(sign_ext) op Dn → Dn	*	*	*	က	*	1111	1001	0111	l 00Dn	<	8								66
UDF08 imm8,Dn	imm8(sign_ext) op Dn → Dn	*	*	*	3	* D1	1111	1001	1000) 00Dn	<:>	8>								99
UDF09 imm8,Dn	imm8(sign_ext) op Dn → Dn	*	*	*	3	*	1111	1001	1001	I 00Dn	<imm8< td=""><td>8></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>66</td></imm8<>	8>								66
UDF10 imm8,Dn	imm8(sign_ext) op Dn → Dn	*	*	*	3	* D1	1111	1001	1010	00Dn	< 8mmi>	8>								66
UDF11 imm8,Dn	imm8(sign_ext) op Dn → Dn	*	*	*	က	*	1111	1001	101	n 000	<	8 								66
UDF12 imm8,Dn	imm8(sign_ext) op Dn → Dn	*	*	*	8	*	1111	1001	1100	00Dn	<8mmi>	8								66
UDF13 imm8,Dn	imm8(sign_ext) op Dn → Dn	*	*	*	3	*	1111	1001	1101	n 000	<8mmi>	8 								66
UDF14 imm8,Dn	imm8(sign_ext) op Dn → Dn	*	*	*	က	*	1111	1001	1110	00Dn	<	8 								66
UDF15 imm8,Dn	imm8(sign_ext) op Dn → Dn	*	*	*	က	*	1111	1001	1111	n 000		k8								66
UDF20 imm8,Dn	imm8(sign_ext) op Dn				က	*	1111	1001	0000	10Dn	<8mmi>	8							Not used for AM30	66
UDF21 imm8,Dn	imm8(sign_ext) op Dn				က	*	1111	1001	0001	10Dn	<>	8							Not used for AM30	66
UDF22 imm8,Dn	imm8(sign_ext) op Dn			•	8	*	1111	1001	0010	10Dn	<	8>							Not used for AM30	66
UDF23 imm8,Dn	imm8(sign_ext) op Dn				က	*	1111	1001	0011	10Dn	<8mmi>	8 							Not used for AM30	66
UDF24 imm8,Dn	imm8(sign_ext) op Dn			•	က	*	1111	1001	0100	10Dn	<mm></mm>	8 							Not used for AM30	66
UDF25 imm8,Dn	imm8(sign_ext) op Dn			-	က	*	1111	1001	0101	10Dn	<8mmi>	8							Not used for AM30	66
UDF26 imm8,Dn	imm8(sign_ext) op Dn				က	*	1111	1001	0110	10Dn	<	k8							Not used for AM30	66
UDF27 imm8,Dn	imm8(sign_ext) op Dn				က	*	1111	1001	0111	10Dn	<	8 ····							Not used for AM30	66
UDF28 imm8,Dn	imm8(sign_ext) op Dn			•	3	*	1111	1001	1000	10Dn	<8 will	8							Not used for AM30	66
UDF29 imm8,Dn	imm8(sign_ext) op Dn			-	ъ	*	1111	1001	1001	10Dn	<8mmi>	8							Not used for AM30	66
UDF30 imm8,Dn	imm8(sign_ext) op Dn	-	-	-	3	*	1111	1001	1010	10Dn	<8>	8>							Not used for AM30	66
UDF31 imm8,Dn	imm8(sign_ext) op Dn	•	-	-	3	*	1111	1001	1011	10Dn	<>	8>							Not used for AM30	66
UDF32 imm8,Dn	imm8(sign_ext) op Dn			•	3	* D1	1111	1001	1100	10Dn	< 8mmi>	8>							Not used for AM30	66
UDF33 imm8,Dn	imm8(sign_ext) op Dn	-	-	•	3	*	1111	1001	1101	10Dn	<>	8>							Not used for AM30	99
UDF34 imm8,Dn	imm8(sign_ext) op Dn	•	-		3	*	1111	1001	1110	10Dn	<8>	8							Not used for AM30	66
UDF35 imm8,Dn	imm8(sign_ext) op Dn	-	-	•	3	* D1	1111	1001	1111	10Dn	< 8mmi>	8>							Not used for AM30	66
UDF00 imm16,Dn	imm16(sign_ext) op Dn → Dn	*	*	*	4	* D2	1111	1011	0000	00Dn	<imm16.< td=""><td></td><td></td><td><</td><td></td><td></td><td></td><td></td><td></td><td>66</td></imm16.<>			<						66
UDF01 imm16,Dn	imm16(sign_ext) op Dn → Dn	*	*	*	4	* D2	1111	101	0001	00Dn	<imm16.< td=""><td></td><td></td><td>^</td><td></td><td></td><td></td><td></td><td></td><td>66</td></imm16.<>			^						66
UDF02 imm16,Dn	imm16(sign_ext) op Dn → Dn	*	*	*	4	* D2	1111	1011	0010) 00Dn	<imm16.< td=""><td>:</td><td></td><td><</td><td></td><td></td><td></td><td></td><td></td><td>99</td></imm16.<>	:		<						99
UDF03 imm16,Dn	imm16(sign_ext) op Dn → Dn	*	*	*	4	* D2	1111	1011	0011	I 00Dn	<imm16.< td=""><td></td><td></td><td><</td><td></td><td></td><td></td><td></td><td></td><td>99</td></imm16.<>			<						99
UDF04 imm16,Dn	imm16(sign_ext) op Dn → Dn	*	*	*	4	* D2	1111	1011	0100	00Dn	<imm16.< td=""><td>:</td><td>:</td><td><</td><td></td><td></td><td></td><td></td><td></td><td>99</td></imm16.<>	:	:	<						99
UDF05 imm16,Dn	imm16(sign_ext) op Dn → Dn	*	*		4	* D2	1111	1011	0101	00Dn	<imm16.< td=""><td></td><td></td><td>^:::</td><td></td><td></td><td></td><td></td><td></td><td>66</td></imm16.<>			^:::						66
UDF06 imm16,Dn	imm16(sign_ext) op Dn → Dn	*	*	*	4	* D2	1111	1011	0110) 00Dn	<imm16.< td=""><td>:</td><td></td><td><</td><td></td><td></td><td></td><td></td><td></td><td>99</td></imm16.<>	:		<						99
UDF07 imm16,Dn	imm16(sign_ext) op Dn → Dn	*	*	*	4	* D2	1111	1011	0111	I 00Dn	<imm16.< td=""><td></td><td></td><td>^</td><td></td><td></td><td></td><td></td><td></td><td>66</td></imm16.<>			^						66
UDF08 imm16,Dn	imm16(sign_ext) op Dn → Dn	*	*	*	4	* D2	1111	1011	1000	00Dn	<imm16.< td=""><td></td><td></td><td>^::</td><td></td><td></td><td></td><td></td><td></td><td>66</td></imm16.<>			^::						66
UDF09 imm16,Dn	imm16(sign_ext) op Dn → Dn	*	*	*	4	* D2	1111	1011	1001	n 000 n	<imm16.< td=""><td></td><td></td><td>^::</td><td></td><td></td><td></td><td></td><td></td><td>66</td></imm16.<>			^::						66
UDF10 imm16,Dn	imm16(sign_ext) op Dn → Dn	*	*	*	4	* D2	1111	1011	1010	00Dn	<imm16.< td=""><td>:</td><td>:</td><td>^</td><td></td><td></td><td></td><td></td><td></td><td>66</td></imm16.<>	:	:	^						66
UDF11 imm16,Dn	imm16(sign_ext) op Dn → Dn	*	*	*	4	* D2	1111	1011	1011	00Dn	<imm16.< td=""><td></td><td></td><td><</td><td></td><td></td><td></td><td></td><td></td><td>99</td></imm16.<>			<						99
UDF12 imm16,Dn	imm16(sign_ext) op Dn → Dn	*	*	*	4	* D2	1111	1011	1100) 00Dn	<imm16.< td=""><td></td><td></td><td><</td><td></td><td></td><td></td><td></td><td></td><td>99</td></imm16.<>			<						99
UDF13 imm16,Dn	imm16(sign_ext) op Dn → Dn	*	*	*	4	* D2	1111	1011	1101	n 000	<imm16.< td=""><td></td><td></td><td>^::</td><td></td><td></td><td></td><td></td><td></td><td>66</td></imm16.<>			^::						66
UDF14 imm16,Dn	imm16(sign_ext) op Dn → Dn	*	*	*	4	* D2	1111	1011	1110	00Dn	<imm16.< td=""><td></td><td></td><td>^::</td><td></td><td></td><td></td><td></td><td></td><td>66</td></imm16.<>			^::						66
10145 ::	imm16/cian oxt) on Dr. Dr.	•																	_	

Page	Ī	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66	66
Notes																																									
ž		0	0	0	0	0		0	0		0	0	0			0	0																	0	0	0	0	0	0		0
		r AM3	r AM3	ır AM3	r AM3	r AM3	r AM3	ır AM3	r AM3	r AM3	r AM3	r AM3	r AM3	r AM3	r AM3	r AM3	r AM3																	r AM3	r AM3	r AM3	r AM3	r AM3	r AM3	r AM3	r AM3
		Not used for AM30	Not used for AM30	Not used for AM30	Not used for AM30	Not used for AM30	Not used for AM30	Not used for AM30	Not used for AM30	Not used for AM30	Not used for AM30	Not used for AM30	Not used for AM30	Not used for AM30	Not used for AM30	Not used for AM30	Not used for AM30																	Not used for AM30	Not used for AM30	Not used for AM30	Not used for AM30	Not used for AM30	Not used for AM30	Not used for AM30	Not used for AM30
		Not	Not	Not u	Not	Not	Not	Not u	Not	Not	Not	Not	Not	Not	Not	Not	Not																	Not u	Not	Not	Not	Not	Not u	Not	Not
4																																									
13																																									
12																		^:	^ :	^ :	^:	^:	1:	^:	^ :	^ :	^:	^::	^::	^: :	^:	^:	^::	<u>^</u>	^ :	^:	^: :	^ :	^:	^:	^ <u>:</u>
7																	:		;	:	:	:				:	:	:		:		:	:	:	:	:	i	:			
10																																					:				
6																																			:		:	:			
		^:	^:	^::	^:	^:	1	^::	^:	<u>^</u>	^:	^::	^::	^:	<u>^</u>	^::	^:										:								:						
Sode 8		'															:				:																:	:			
Machine Code																	:													:	:	:		:	:	:					
		<imm16.< td=""><td><imm16.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<>	<imm16.< td=""><td><imm16.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<>	<imm16.< td=""><td><imm16.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<>	<imm16.< td=""><td><imm16.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<>	<imm16.< td=""><td><imm16.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<>	<imm16.< td=""><td><imm16.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<>	<imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<>	<imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<>	<imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<>	<imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<>	<imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<>	<imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<>	<imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm16.<></td></imm16.<></td></imm16.<></td></imm16.<>	<imm16.< td=""><td><imm16.< td=""><td><imm16.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm16.<></td></imm16.<></td></imm16.<>	<imm16.< td=""><td><imm16.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm16.<></td></imm16.<>	<imm16.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm16.<>	<imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<>	<imm32< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<>	<imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<>	<imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<>	<imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<>	<imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<>	<imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<>	<imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<>	<imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<>	<imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<>	<imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<>	<imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<>	<imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<>	<imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<></td></imm32.<>	<imm32< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32<>	<imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<>	<imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<>	<imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<></td></imm32.<>	<imm32.< td=""><td><imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<></td></imm32.<>	<imm32.< td=""><td><imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<></td></imm32.<>	<imm32.< td=""><td><imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<></td></imm32.<>	<imm32< td=""><td><imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<></td></imm32<>	<imm32.< td=""><td><imm32.< td=""></imm32.<></td></imm32.<>	<imm32.< td=""></imm32.<>
5		ı	10Dn <	10Dn <	10Dn <	10Dn <	10Dn <	10Dn <	10Dn <	10Dn <		10Dn <	10Dn <	10Dn <	10Dn <	10Dn <	10Dn <	00Dn <	ooDn ^		o0Dn <	ooDn ^	ooDn v	00Dn <		00Dn <		00Dn <	00Dn <	00Dn <	00Dn <		00Dn <	10Dn <	10Dn <	10Dn <	10Dn <	10Dn <		10Dn <	10Dn <
4		0 10Dn								l	1 10Dn			l						0 00Dn					1 00Dn		1 00Dn					0 00Dn							1 10Dn		
က		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111
2		1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1101	1101	1101	1101	1101	1101	1101	1101	1101	1101	1101	1101	1101	1101	1101	1101	1101	1101	1101	1101	1101	1101	1101	1101
_		1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111
For		D2	D2	D2	D2	D2	D2	D2	D2	D2	D2	D2	D2	D2	D2	D2	D2	4	D4	D4	D4	D4	D4	D4	D4	D4	D4	D4	D4	D4	D4	D4	D4	D4	D4	D4	D4	D4	D4	D4	D4
e Oycle		*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
Cod		4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
_ H		<u> </u>	-		-	-	'					-	-	<u>'</u>	'	-		*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	-	-	-				'	H
Flag		-	,	-			,	-				-		,	,	-		*	*	*	*	*	*	*	*	*	*	*	*	*		*	*	-				-	-	,	
Flag Code Cycle For VF CF NF 75 Size		-	-	-				-				-	-			-		*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*						-		
ation																																									
Operation		n D	nD d	p Dn	p Dn	p Dn	n D D	p Dn	D D	n D D	p Dn	p Dn	nD d	D D	D D	nD d	p Dn	ے	۵	ے	u	_	_	ے	_	ے	u	u	u	<u>u</u>	ے	u	u								
		ext) o	ext) o	ext) o	ext) o	ext) o	ext) o	ext) o	ext) o	ext) o	ext) o	ext) o	ext) o	ext) o	ext) o	ext) o	ext) o	1	1	1	1 أ	1	_ _	1	1 أ	1	ر ر	راب	ر پ ر	1	1	_ ر	_ ر	_	_	_	ے	_	ر	_	ا ا
		sign	sign_	sign_	sign_	sign_	sign	sign_	sign	sign	sign	sign_	sign_	sign	sign	sign_	sign	op Dr	op Dr	op Dr	op Dr	op Dr	op Dr	op Dr	op Dr	op Dr	op Dr	op Dr	op Dr	op Dr	op Dr	op Dr	op Dr	op Dr	op Dr	op Dr	op Dr	op Dr	op Dr	op Dr	op Dr
		imm16(sign_ext) op Dn	imm16(sign_ext) op Dn	imm16(sign_ext) op Dn	imm16(sign_ext) op Dn	imm16(sign_ext) op Dn	imm16(sign_ext) op Dn	imm16(sign_ext) op Dn	imm16(sign_ext) op Dn	imm16(sign_ext) op Dn	imm16(sign_ext) op Dn	imm16(sign_ext) op Dn	imm16(sign_ext) op Dn	imm16(sign_ext) op Dn	imm16(sign_ext) op Dn	imm16(sign_ext) op Dn	imm16(sign_ext) op Dn	imm32 op Dn → Dn	imm32 op Dn → Dn	imm32 op Dn → Dn	imm32 op Dn → Dn	imm32 op Dn → Dn	imm32 op Dn → Dn	imm32 op Dn → Dn	imm32 op Dn → Dn	imm32 op Dn → Dn	imm32 op Dn → Dn	imm32 op Dn → Dn	imm32 op Dn → Dn	imm32 op Dn → Dn	imm32 op Dn → Dn	imm32 op Dn → Dn	imm32 op Dn → Dn	imm32 op Dn	imm32 op Dn	imm32 op Dn	imm32 op Dn	imm32 op Dn	imm32 op Dn	imm32 op Dn	imm32 op Dn
\vdash		<u>.</u> ⊑	_≟.	ï	.⊑	.⊑	_≛.	ï	⊑	⊑	.⊑	ī	.⊑	⊑	_≛.	Ľ.	.⊑	.⊑	.⊑	_≟.	Ľ.	.⊑	⊑	_≟.	ï	Ŀ	ī	ī	ï	.⊑	.⊑	ī	ï	.⊑	.⊑	ï	.⊑	.⊑	ï	⊑	.⊑
٦ic		۵	Du	Dn	Du	Du	۵	Dn	۵	۵	Du	Dn	Dn	Du	Du	Du	۵	Du	Du	Du	Dn	۵n	۵n	Du	Dn	Dn	Dn	Dn	Dn	۵	۵	Dn	Dn	Du	Du	Dn	Du	Du	Dn	۵	۵
Mnemonic		UDF20 imm16,Dn	UDF21 imm16,Dn	UDF22 imm16,Dn	UDF23 imm16,Dn	UDF24 imm16,Dn	UDF25 imm16,Dn	UDF26 imm16,Dn	UDF27 imm16,Dn	UDF28 imm16,Dn	UDF29 imm16,Dn	UDF30 imm16,Dn	UDF31 imm16,Dn	UDF32 imm16,Dn	UDF33 imm16,Dn	UDF34 imm16,Dn	UDF35 imm16,Dn	UDF00 imm32,Dn	UDF01 imm32,Dn	UDF02 imm32,Dn	UDF03 imm32,Dn	UDF04 imm32,Dn	UDF05 imm32,Dn	UDF06 imm32,Dn	UDF07 imm32,Dn	UDF08 imm32,Dn	UDF09 imm32,Dn	UDF10 imm32,Dn	UDF11 imm32,Dn	UDF12 imm32,Dn	UDF13 imm32,Dn	UDF14 imm32,Dn	UDF15 imm32,Dn	UDF20 imm32,Dn	UDF21 imm32,Dn	UDF22 imm32,Dn	UDF23 imm32,Dn	UDF24 imm32,Dn	UDF25 imm32,Dn	UDF26 imm32,Dn	UDF27 imm32,Dn
ĮΣ		F20 ii	F21 ii	F22 ii	F23 ii	F24 ii	F25 ii	F26 ii	1F27 ii	F28 ii	F29 ii	F30 ii	F31 ii	F32 ii	F33 ii	F34 ii	F35 ir	F00 ii	F01 ii	F02 ii	F03 ii	F04 ii	F05 ii	F06 ir	F07 ii	F08 ii	F09 ii	1F10 ii	F11 ii	F12 ii	F13 ii	ìF14 ii	F15 ii	F20 ii	F21 ii	F22 ii	F23 ii	1F24 ii	F25 ii	7F26 ir)F27 ii
۵		19	3	В	9	9	3	B	3	3	9	9	9	3	3	3	3	3	9	3	9	3	3	3	9	9	В	B	В	9	9	В	В	9	9	9	3	9	9	3	
Group		UDFnn																																							

	Group	Mnemonic	Operation	VF CF	Flag VF CF NF ZF	Code ZF Size	Cycle	For -mat 1	2	3	4	Machine Code 5	ne Code	8	6	10	11	12 13	14	Notes	Page
VEX.5D (WINTERS OF DITTOR SECTION CONTROL SECTION CONTR													:								
Interest op Din	UDFnn		imm32 op Dn			9 -			1101			<imm32< td=""><td>:</td><td>:</td><td>:</td><td>:</td><td>:</td><td>^</td><td></td><td>Not used for AM30</td><td>99</td></imm32<>	:	:	:	:	:	^		Not used for AM30	99
ministry op Dh		UDF29 imm32,Dn	imm32 op Dn			9 -	*		1101			<imm32< td=""><td>:</td><td>:</td><td>:</td><td>:</td><td>:</td><td>^:::</td><td></td><td>Not used for AM30</td><td>66</td></imm32<>	:	:	:	:	:	^ :::		Not used for AM30	66
Interest of photons		UDF30 imm32,Dn	imm32 op Dn			9 -		-	1101			<imm32< td=""><td> ;</td><td>:</td><td>:</td><td> :</td><td>:</td><td>^::</td><td></td><td>Not used for AM30</td><td>99</td></imm32<>	;	:	:	:	:	^ ::		Not used for AM30	99
ministrate or Difference and go Difference and		UDF31 imm32,Dn	imm32 op Dn	-		9 -			1101			<imm32< td=""><td>:</td><td>i</td><td>i</td><td>:</td><td>i</td><td>^</td><td></td><td>Not used for AM30</td><td>66</td></imm32<>	:	i	i	:	i	^		Not used for AM30	66
ministrop Din		UDF32 imm32,Dn	imm32 op Dn			9 -			1101			<imm32< td=""><td>:</td><td>:</td><td>:</td><td>:</td><td>:</td><td>^:::</td><td></td><td>Not used for AM30</td><td>66</td></imm32<>	:	:	:	:	:	^ :::		Not used for AM30	66
mimigran, and op Dn. — Dr. 1 1 1 1 1 1 1 1 1		UDF33 imm32,Dn	imm32 op Dn	-	-	9 -	*		1101			<imm32< td=""><td>:</td><td>:</td><td>:</td><td>:</td><td>:</td><td>^</td><td></td><td>Not used for AM30</td><td>66</td></imm32<>	:	:	:	:	:	^		Not used for AM30	66
minicipal condition		UDF34 imm32,Dn	imm32 op Dn			9 -			1101			<imm32< td=""><td>:</td><td>i</td><td>:</td><td>:</td><td>i</td><td>^</td><td></td><td>Not used for AM30</td><td>99</td></imm32<>	:	i	:	:	i	^		Not used for AM30	99
minificacio and op Dn - Dn 1 1 1 1 1 1 1 1 1		UDF35 imm32,Dn	imm32 op Dn			9 -			1101			<imm32< td=""><td></td><td>:</td><td>:</td><td>:</td><td>:</td><td>^::</td><td></td><td>Not used for AM30</td><td>66</td></imm32<>		:	:	:	:	^ ::		Not used for AM30	66
miniglaciae, act of or Dn - Dn 1 1 1 1 1 1 1 1 1	UDFUnr	UDFU00 imm8,Dn	imm8(zero_ext) op Dn →Dn	*					1001			<>									100
minigizaro, exi) op Dn -Dn -Dn 1		UDFU01 imm8,Dn	imm8(zero_ext) op Dn →Dn			_*	*		1001			<>									100
Immigrator ext) op Dn -Dn Dn Dn Dn Dn Dn Dn		UDFU02 imm8,Dn	imm8(zero_ext) op Dn →Dn			რ *			1001			<>									100
immê[zero_ext] op DhDh 1		UDFU03 imm8,Dn	imm8(zero_ext) op Dn →Dn						1001	l		<>									100
mm/g/cac_axi) op DnDn		UDFU04 imm8,Dn	imm8(zero_ext) op Dn →Dn					_	1001	l		<>									100
Imm8[seao, axi op DnDn 1		UDFU05 imm8,Dn	imm8(zero_ext) op Dn →Dn			۳ *			1001			<>									100
Imm8igleaco, and op Dn -Dn		UDFU06 imm8,Dn	imm8(zero_ext) op Dn →Dn						1001			<>									100
imm8[cenc_ext] op DnDn 1 1 1111 1001 1000 0100 cinm8		UDFU07 imm8,Dn	imm8(zero_ext) op Dn →Dn			ო *	*		1001			<>									100
immé(cao, ext) op DnDn r <td></td> <td>UDFU08 imm8,Dn</td> <td>imm8(zero_ext) op Dn →Dn</td> <td></td> <td></td> <td>_</td> <td></td> <td></td> <td>1001</td> <td></td> <td>100</td>		UDFU08 imm8,Dn	imm8(zero_ext) op Dn →Dn			_			1001												100
imm8(zero_ext) op DnDn		UDFU09 imm8,Dn	imm8(zero_ext) op Dn →Dn			_*			1001			<>									100
imm8(zero_ext) op Dn →Dn imm8(zero_ext) op Dn imm8(zero		UDFU10 imm8,Dn	imm8(zero_ext) op Dn →Dn						1001			<>									100
mm8[zero_ext) op Dn →Dn mm8[zero_ext) op Dn mm0 mm0 mm8[zero_ext) op Dn mm0 mm8[zero_		UDFU11 imm8,Dn	imm8(zero_ext) op Dn →Dn						1001			<imm8></imm8>									100
imm8[zero_ext) op DnDn r <td></td> <td>UDFU12 imm8,Dn</td> <td>imm8(zero_ext) op Dn →Dn</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>1001</td> <td></td> <td></td> <td><></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>100</td>		UDFU12 imm8,Dn	imm8(zero_ext) op Dn →Dn						1001			<>									100
imm8(zero_ext) op Dn →Dn r <td></td> <td>UDFU13 imm8,Dn</td> <td>imm8(zero_ext) op Dn →Dn</td> <td>-</td> <td>_</td> <td>m *</td> <td>\neg</td> <td></td> <td>1001</td> <td></td> <td></td> <td><></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>100</td>		UDFU13 imm8,Dn	imm8(zero_ext) op Dn →Dn	-	_	m *	\neg		1001			<>									100
imm8[Zero_ext) op DnDn		UDFU14 imm8,Dn	imm8(zero_ext) op Dn →Dn			-		_	1001	- 1	- 1	<									100
imm8(zero_ext) op Dh -		UDFU15 imm8,Dn	imm8(zero_ext) op Dn →Dn						1001			<imm8></imm8>									100
imm8(zero_ext) op Dn		UDFU20 imm8,Dn	imm8(zero_ext) op Dn			ო	*		1001			<								Not used for AM30	100
imm8(zero_ext) op Dh -		UDFU21 imm8,Dn	imm8(zero_ext) op Dn						1001			<>								Not used for AM30	100
imm8(zero_ext) op Dn		UDFU22 imm8,Dn	imm8(zero_ext) op Dn						1001											Not used for AM30	100
imm8(zero_ext) op Dn		UDFU23 imm8,Dn	imm8(zero_ext) op Dn						1001			<								Not used for AM30	100
imm8(zero_ext) op Dn -		UDFU24 imm8,Dn	imm8(zero_ext) op Dn			ი	\neg		1001	- 1	- 1	<>								Not used for AM30	100
imm8(zero_ext) op Dn -		UDFU25 imm8,Dn	imm8(zero_ext) op Dn			°	*	_	1001			<>								Not used for AM30	100
imm8(zero_ext) op Dn -		UDFU26 imm8,Dn	imm8(zero_ext) op Dn			ი			1001			<								Not used for AM30	100
imm8(zero_ext) op Dn -		UDFU27 imm8,Dn	imm8(zero_ext) op Dn		•	ი			1001			<>								Not used for AM30	100
imm8(zero_ext) op Dn -		UDFU28 imm8,Dn	imm8(zero_ext) op Dn			\dashv			1001	- 1	- 1	<								Not used for AM30	100
imm8(zero_ext) op Dn -		UDFU29 imm8,Dn	imm8(zero_ext) op Dn	-					1001											Not used for AM30	100
imm8(zero_ext) op Dn -		UDFU30 imm8,Dn	imm8(zero_ext) op Dn		-		*	-	1001			<								Not used for AM30	100
imm8(zero_ext) op Dn -		UDFU31 imm8,Dn	imm8(zero_ext) op Dn	1		ი	\neg	_	1001	- 1										Not used for AM30	100
imm8(zero_ext) op Dn -		UDFU32 imm8,Dn	imm8(zero_ext) op Dn			ი		-	1001											Not used for AM30	100
imm8(zero_ext) op Dn - - - - 3 * D1 1111 1001 1111 1111 1101 1111 111		UDFU33 imm8,Dn	imm8(zero_ext) op Dn			ი	\neg		1001		- 1	<>								Not used for AM30	100
imm8(zero_ext) op Dn		UDFU34 imm8,Dn	imm8(zero_ext) op Dn		•	e -	\neg	-	1001											Not used for AM30	100
		UDFU35 imm8,Dn	imm8(zero_ext) op Dn		_	ი	*		1001	- 1	- 1	<>								Not used for AM30	100

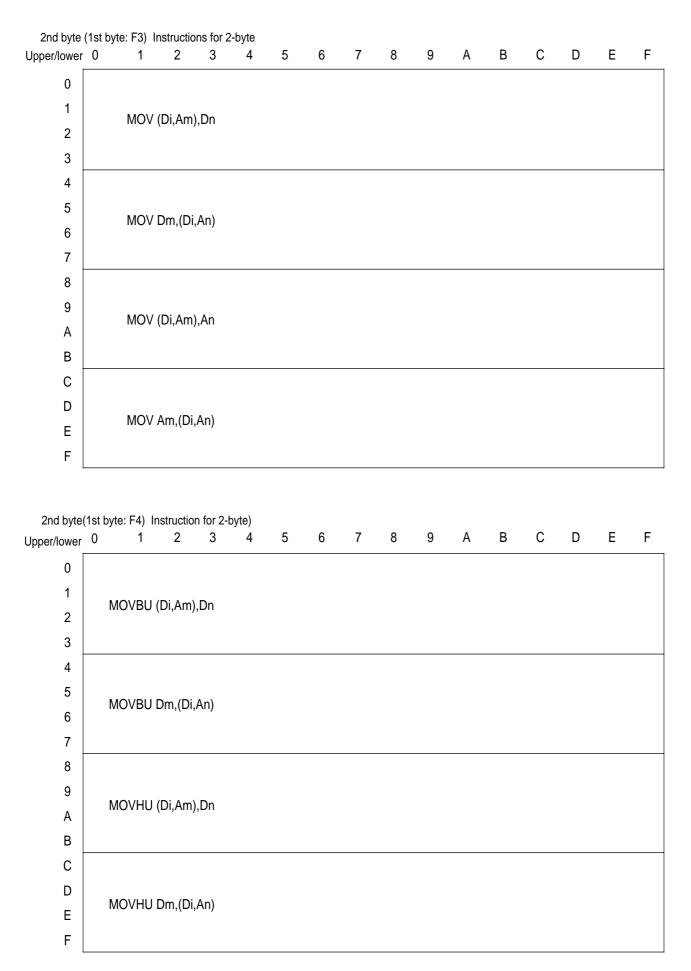
r gg gg	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	5
																	130	130	130	130	130	130	130	130	130	130	130	130	130	130	130	130								
																	Not used for AM30	d for AM	d for AM	d for AM	d for AM	d for AM	d for AM	d for AM	d for AM	d for AM	d for AM	d for AM	Not used for AM30	d for AM	d for AM	d for AM								
																	Not use	Not used for AM30	Not used for AM30	Not used for AM30	Not used for AM30	Not used for AM30	Not used for AM30	Not used for AM30	Not used for AM30	Not used for AM30	Not used for AM30	Not used for AM30	Not use	Not used for AM30	Not used for AM30	Not used for AM30								
_																																								İ
1																																								
2																																								
!																																	^::	1:	^ ::	^: :	^:::	^::	^:	
=																																	:	i	:	i	:	i	i	
2																																	:	i	:	i	:	i	i	
0																																	:	;	:	:	:	:	:	
	^:	<u>^</u>	^::	^:	^:	<u>^</u>	1	^:	^:	^:	^:	^:::	^::	^ <u>:</u>	<u>^</u>	^:	^::	^:	^:	^::	^:	^:	^:	^:	^:	<u>^:</u>	^:	^:	^:	^ <u>:</u>	<u>^</u>	^:	:	i	:	:	:	:	i	
0																																								
		:	:	•	:	:	:	:		:		:	:		:	:	:	:		:	:	•	•	:	•			1	i	•		•	:	•		:	:	i	i	
٥	<imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< p=""></imm16<>	<imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm16< p=""></imm16<></td><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< p=""></imm16<>	<imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<></td></imm16<>	<imm16< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm16<>	<imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<>	<imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<>	<imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<></td></imm32<>	<imm32< td=""><td><imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<></td></imm32<>	<imm32< td=""><td><imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<></td></imm32<>	<imm32< td=""><td><imm32< td=""><td></td></imm32<></td></imm32<>	<imm32< td=""><td></td></imm32<>	
	1				l			l						l								- 1				- 1				- 1	- 1									П
	01Dn	01Dn	01Dn	01Dn	01Dn	01Dn	01Dn	01Dn	01Dn	01Dn	01Dn	01Dn	01Dn	01Dn	01Dn	01Dn	11Dn	11Dn	11Dn	11Dn	11Dn	11Dn	11Dn	11Dn	11Dn	11Dn	11Dn	11Dn	11Dn	11Dn	11Dn	11Dn	01Dn	01Dn	01Dn	01Dn	01Dn	01Dn	01Dn	
	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	
1	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1011	1101	1101	1101	1101	1101	1101	1101	
	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	ı
-mat	D2 1	D2 1	D2 1	D2 ,	D2 1	D2 1	D2 1	D2 1	D2 1	D2 1	D2 1	D2 1	D2 1	D2 1	D2 1	D2 1	D2 1	D2 1	D2 1	D2 1	D2 1	D2 1	D2 ,	D2 1	D2 1	D2 1	D2 1	D2 1	D2 1	D2 1	D2 1	D2 1	D4 1	, ,	D4	D4	D4 1	D4	7	
9.		*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	* 4	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	* 9	*	*	* 9	* 9	* 9	* 9	+
ZF Siz	4	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	- 4	- 4	4	- 4	- 4	- 4	4	- 4	- 4	- 4	- 4	4	4	- 4	- 4	- 4	* 6	*	*	*	*	*	*	ł
VF CF NF ZF Size -m	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*		٠		-			٠	٠							•	٠	*	*	*	*	*	*	*	ļ
<u>.</u>	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*			'	-	-	1		'	-	1	-	-		-	- 1		*	*	*	*	*	*	*	H
																																								t
	١	ڃ	u	Ľ	ء	ء	٤	٤	ء	ء	u	u	u	ڃ	ء	٤																								
	u ↑	n ↓	n → D	U ↑ U	u(n ↓	u(Q ↑ u(Q ↑ u	n ↓	u ← u	n → D	u(Q ↑ u	n ↓	Q ↑ u(u(u	u L)u	'n	u	Ę	'n	n(u L	u C	u	u	'n	u C	Ľ								
	xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [xt) op [→ Dn	u□	Du	u□	→ Dn	Du	D	
	zero_e;	zero_e)	zero_e;	zero_e;	zero_e)	zero_e)	zero_e)	zero_e;	zero_e;	zero_e)	zero_e;	zero_e;	zero_e)	zero_e;	zero_e;	zero_e;	zero_e;	zero_e;	zero_e;	zero_e;	zero_e;	zero_e;	zero_e;	zero_e;	zero_e;	zero_e)	zero_e;	zero_e;	zero_e;	zero_e;	zero_e;	zero_e;	- UD do	- DD do	- uO do	- Du	- uO do	- DD do	- nO do	
	imm16(zero_ext) op Dn → Dn	imm16(zero_ext) op Dn → Dn	imm16(zero_ext) op Dn → Dn	imm16(zero_ext) op Dn → Dn	imm16(zero_ext) op Dn → Dn	imm16(zero_ext) op Dn → Dn	imm16(zero_ext) op Dn → Dn	imm16(zero_ext) op Dn → Dn	imm16(zero_ext) op Dn → Dn	imm16(zero_ext) op Dn → Dn	imm16(zero_ext) op Dn → Dn	imm16(zero_ext) op Dn → Dn	imm16(zero_ext) op Dn → Dn	imm16(zero_ext) op Dn → Dn	imm16(zero_ext) op Dn → Dn	imm16(zero_ext) op Dn → Dn	imm16(zero_ext) op Dn	imm16(zero_ext) op Dn	imm16(zero_ext) op Dn	imm16(zero_ext) op Dn	imm16(zero_ext) op Dn	imm16(zero_ext) op Dn	imm16(zero_ext) op Dn	imm16(zero_ext) op Dn	imm16(zero_ext) op Dn	imm16(zero_ext) op Dn	imm16(zero_ext) op Dn	imm16(zero_ext) op Dn	imm16(zero_ext) op Dn	imm16(zero_ext) op Dn	imm16(zero_ext) op Dn	imm16(zero_ext) op Dn	imm32 op Dn → Dn	imm32 op Dn → Dn	imm32 op Dn → Dn	imm32 op Dn → Dn	imm32 op Dn → Dn	imm32 op Dn → Dn	imm32 op Dn → Dn	
																																								t
	UDFUnn UDFU00 imm16,Dn	UDFU01 imm16,Dn	UDFU02 imm16,Dn	UDFU03 imm16,Dn	UDFU04 imm16,Dn	UDFU05 imm16,Dn	UDFU06 imm16,Dn	UDFU07 imm16,Dn	UDFU08 imm16,Dn	UDFU09 imm16,Dn	UDFU10 imm16,Dn	UDFU11 imm16,Dn	UDFU12 imm16,Dn	UDFU13 imm16,Dn	UDFU14 imm16,Dn	UDFU15 imm16,Dn	UDFU20 imm16,Dn	UDFU21 imm16,Dn	UDFU22 imm16,Dn	UDFU23 imm16,Dn	UDFU24 imm16,Dn	UDFU25 imm16,Dn	UDFU26 imm16,Dn	UDFU27 imm16,Dn	UDFU28 imm16,Dn	UDFU29 imm16,Dn	UDFU30 imm16,Dn	UDFU31 imm16,Dn	UDFU32 imm16,Dn	UDFU33 imm16,Dn	UDFU34 imm16,Dn	UDFU35 imm16,Dn	UDFU00 imm32,Dn	UDFU01 imm32,Dn	UDFU02 imm32,Dn	UDFU03 imm32,Dn	UDFU04 imm32,Dn	UDFU05 imm32,Dn	UDFU06 imm32,Dn	
	J00 im	J01 im	J02 imi	J03 im	704 im	J05 im	J06 im	J07 imi	J08 im	mi 60C	J10 imi	J11 imi	U12 imi	713 imi	714 im	J15 imi	J20 imi	J21 imi	U22 imi	J23 imi	U24 imi	J25 imi	U26 imi	U27 imi	U28 im	U29 imi	U30 imi	U31 im	U32 imi	U33 imi	U34 imi	J35 im	J00 imi	J01 im	J02 imi	J03 imi	J04 imi	J05 imi	J06 im	
	UDFL	UDFL	UDFL	UDFL	JE	UDFL	UDFL	UDFL	Jan	Jen	UDFL	UDFL	UDFL	UDFL	UPFL	UDFL	UDFL	UDFL	UDFL	UDFL	UDFL	UDFL	UDFL	UDFL	UDFL	UDFL	JA	UDFL	UDFL	UDFL	JA	UDFL	UDFL	UDFL	UDFL	UDFL	UDFL	UDFL	UDFL	
5	PFUn																																							

data registers address registers address registers address registers address registers address registers and by concessor status word program counter to program of the pipline, memory referred by () address to program of the pipline, memory referred by () address to program of the counter to program of the pipline, memory referred by () address to program of the counter to program of the pipline, memory referred by () address to program of the counter to program of the pipline, memory referred by () address to program of the counter to program of the pipline, memory registers bit inverted control (choest-highest) to program of the pipline, memory registers bit inverted control (choest-highest) to program of the pipline installation to program of the pipline installation to program of the counter to program o	INSTRUCTION SET		Z C L	■ Instance of the state of the
nulpiy data registers multiply data registers multiply data registers multiply data registers processor status word program counter stack pointer loop instruction registers loop address register registers loop address	Description		Flag	Instructions replaced to other instructions by Assembler
mutdress registers	Dn,Dm,Di	data registers	l changes	Format or Mchine Codeare not written
multiply/divide ragister 0 always 1 processor status word 1 always 1 processor status word 1 always 1 processor status word 1 always 1 processor status word 1 always 1 processor status word 1 always 1 processor status word 1 always 1 processor status word 1 always 1 processor status word 1 always 1 processor status word 1 always 1 processor status word 1 always 1 processor status word 1 always 1 processor status 1 always 1	An,Am	address registers		usable CodeSize and Cycles are written
processor status word stack word stack border stack pointer stack pointer loop instruction registers loop instruction registers loop instruction registers loop instruction registers loop instruction registers loop instruction registers loop instruction registers stack pointer loop instruction registers loop instruction registers stack pointer loop instruction registers stack pointer loop instruction registers loop in memory referred by () address loopical AND lover 15th data in memory referred by () address loopical AND lover 15th data in memory referred by () address loopical AND lover 15th data in memory referred by () address loopical AND lover 15th data in memory referred by () address loopical AND lover 15th data in memory referred by () address loopical AND lover 15th data in memory referred by () address loopical AND lover 15th data in memory referred by () address loopical AND lover 15th data in memory referred by () address loopical AND lover 25th data in memory referred by () address loopical AND lover 35th data in memory referred by () address loopical AND lover 15th data in memory referred by () address loopical AND lover 15th data in memory referred by () address loopical AND lover 32th data in memory referred by () address loopical AND lover 15th defined by users with cache loopical AND lover 15th data defined by users loopical AND lover 15th data defined by users loopical AND lover 15th data defined by users loopical AND lover 25th data and loopical AND lover 32th data are in MDR and lover 32th in hexadecimal (loxer defined in lover) loopical AND lover 32th data are in MDR and lover 32th in hexadecimal (loxer defined in lover) loopical AND loopical AND loopical AND lover 32th data are in MDR and lover 32th in hexadecimal (loxer defined in lover) loopical AND loopical AND loopical AND loopical AND loopical AND loopical AND loopical AND loopical AND loopical AND loopical AND loopical AND loopical AND loopical AND loopical AND loopical AND loopical AND loopical AND loopical AND loopical AND loopical	MDR	multiply/divide register		
suck program counter by program counter coop address registers loop instruction registers loop address registers loop address registers loop address registers loop address registers loop address registers coop address registers loop address registers loop address registers lower 2b bits or 32 bits) coop address (16 or 32 bits) coop address (16 or 32 bits) coop address (16 or 32 bits) coop address registers lower 32-bit data in memory referred by () address loop and offer data in memory referred by () address loop and nower 32-bit data in memory referred by () address loop and nower 32-bit data in memory referred by () address loop and nower 32-bit data in memory referred by () address loop and nower 32-bit data in memory referred by () address loop and nower 32-bit data in memory referred by () address loop and nower 32-bit data in memory referred by () address loop and nower 32-bit data in memory referred by () address loop and nower 32-bit data in memory referred by () address loop and nower 32-bit data in memory referred by () address loop and nower 32-bit data in memory referred by () address loop and nower 32-bit data are in MDR and lower 32-bit in hexadecimal (loov with only by look) loop and nower 32-bit data are in MDR and lower 32-bit in hexadecimal (loov with only by look) loop and nower 32-bit data are in MDR and lower 32-bit in hexadecimal (loov with only by look)	PSW	processor status word		MOVB Reg,Mem , MOVH Reg,Mem,
stack pointer loop instruction registers loop or instruction regis	PC	program counter		ASR Dn , LSR Dn , RTS
Loop instruction registers	SP	stack pointer	* defined by users	
Immurition Propagator Propagator	LIR	loop instruction registers		 Instructions replaced to multiple instructions by Assembler
Immediate value(8, 16 or 32 bits) ■ CodeSize Usable CodeSiz	LAR	loop address registers		Format or Mchine Code are not written
Subside Commun (8, 16 or 32 bits) byte: Subside Commun (8, 16 or 32 bits) byte: Subside Commun (8, 16 or 32 bits) byte: Subside Commun (8, 16 or 32 bits) byte: Subside Commun (8, 16 or 32 bits) byte: Subside Commun (8, 16 or 32 bits) byte: Subside Commun (8, 16 or 32 bits) byte: Subside Commun (8, 16 or 32 bits) byte: Subside Commun (8, 16 or 32 bits) byte: Subside Commun (8, 16 or 32 bits) byte: Subside Commun (10 ower 15chi data in memory referred by () address byte: Subside Commun (10 ower 15chi data in memory referred by () address byte: Subside Commun (10 ower 15chi data in memory referred by () address byte: Subside Commun (10 ower 15chi data in memory referred by () address byte: Subside Commun (10 ower 15chi data are in MDR and lower 32-bit in memory referred byte: Subside Commun (10 ower 15chi data are in MDR and lower 32-bit in regater byte: Subside Commun (10 ower 15chi data are in MDR and lower 32-bit in regater byte: Subside Commun (10 ower 15chi data are in MDR and lower 32-bit in regater byte: Subside Commun (10 ower 15chi data are in MDR and lower 32-bit in regater byte: Subside Commun (10 ower 15chi data are in MDR and lower 32-bit in regater byte: Subside Commun (10 ower 15chi data are in MDR and lower 32-bit in regater byte: Subside Commun (10 ower 15chi data are in MDR and lower 32-bit in regater byte: Subside Commun (10 ower 15chi data are in MDR and lower 32-bit in regater byte: Subside Commun (10 ower 15chi data are in MDR and lower 32-bit in regater byte: Subside Commun (10 ower 15chi data are in MDR and lower 32-bit in regater byte: Subside Commun (10 ower 15chi data are in MDR and lower 32-bit in regater byte: Subside Commun (10 ower 15chi data are in MDR and lower 32-bit in regater byte: Subside Commun (10 ower 15chi data are in MDR and lower 32-bit in regater byte: Subside Commun (10 ower 15chi data are in MDR and lower 32-bit in regater byte: Subside Commun (10 ower 15chi data are in MDR and lowe	imm8,imm16,imm32		■ CodeSize	usable CodeSize and Cycles are written
Signature address (16 or 32 bits)	d8,d16,d32 displace	ment(8, 16 or 32 bits)	byte:	
Cycles Ower 8-bit data in memory referred by () address Cycles	abs16,abs32	absolute address (16 or 32 bits)		MOVB Mem,Reg , MOVH Reg,Mem ,
lower 16bit data in memory referred by () address lower 32-bit data in memory referred by () address registers bit location((lowest/highest) logical AND logical AND logical AND logical AND logical OR exclusive OR bit inverted operation defined by users bit shift(right/left) performs a bit shift for specified value overflow flags carry flags negative flags temporary registers move reflects operation result sign-extend carch data defined whose upper 32-bit data are in MDR and lower 32-bit in register Dn within "{}". hexadecimal(hexadecimal following to Ox.)	mem8(An) lower 8-t	bit data in memory referred by () address	■ Cycles	JSR (An) , JSR label
registers nsb bit location((lowest/highest) logical AND logical A	mem16(An)	lower 16bit data in memory referred by ()address	Cycles may be changed the status of the pipline, memory space	
registers bit location(lowest/highest) logical AND logical AND logical OR exclusive OR bit inverted operation defined by users bit shift(right/left) performs a bit shift for specified value overflow flags carry flags negative flags zero flags temporary registers move reflects operation result sign-extend c-ext) cetch data defined whose upper 32-bit data are in MDR and lower 32-bit in register Dn within "{}". hexadecimal(hexadecimal following to Ox.)	mem32(An)	lower 32-bit data in memory referred by () address	to access.	
bit location(lowesthighest) logical AND logical AND logical OR exclusive OR bit inverted operation defined by users bit shift(right/left) performs a bit shift for specified value overflow flags carry flags negative flags zero flags temporary registers move reflects operation result sign-extend cext) certow within "{}". hexadecimal(hexadecimal following to Ox.)	reds	registers	Cycles are calculated on those conditions;	
logical AND logical OR exclusive OR bit inverted operation defined by users bit shift(right/left) performs a bit shift for specified value overflow flags carry flags negative flags zero flags temporary registers move reflects operation result sign-extend cext) zero-extend 64-bit data defined whose upper 32-bit data are in MDR and lower 32-bit in register Dn within "{}". hexadecimal(hexadecimal following to Ox.)	dsmdsl.	bit location(lowest/highest)	(1) no pipeline installation	
exclusive OR exclusive OR bit inverted operation defined by users bit shift(right/left) performs a bit shift for specified value overflow flags carry flags negative flags zero flags temporary registers move reflects operation result sign-extend ext) zero-extend c-ext) de4-bit data defined whose upper 32-bit data are in MDR and lower 32-bit in register Dn within "{}". hexadecimal(hexadecimal following to Ox.)	•ở	logical AND	(2) Instruction queue: 2 cycles	
bit inverted operation defined by users bit shift(right/left) performsa bit shift for specified value overflow flags carry flags reary flags temporary registers move reflects operation result sign-extend ext) cero-extend cext) cero-extend de-bit data defined whose upper 32-bit data are in MDR and lower 32-bit in register Dn within "{}". hexadecimal(hexadecimal following to 0x.)		logical OR	data load/store: 1 cycle	
bit inverted operation defined by users bit shift(right/left) performsa bit shift for specified value overflow flags carry flags regative flags zero flags temporary registers move reflects operation result sign-extend ext) certo-extend cext) certo-extend de-bit data defined whose upper 32-bit data are in MDR and lower 32-bit in register Dn within "{}". hexadecimal(hexadecimal following to 0x.)	. <	exclusive OR	(ROM/RAM/ internal flash:	
operation defined by users bit shift(right/left) performsa bit shift for specified value overflow flags carry flags regative flags zero flags temporary registers move reflects operation result sign-extend ext) cero-extend cext) de-bit data defined whose upper 32-bit data are in MDR and lower 32-bit in register Dn within "{}". hexadecimal(hexadecimal following to 0x.)	1	bit inverted	Instructions: access to internal ROM/RAM space	
bit shift(right/left) performsa bit shift for specified value overflow flags carry flags negative flags zero flags temporary registers move reflects operation result sign-extend ext) sign-extend c-ext) de-bit data defined whose upper 32-bit data are in MDR and lower 32-bit in register Dn within "{}". hexadecimal(hexadecimal following to 0x.)	do	operation defined by users	data: access to internal RAM space	
performs bit shift for specified value overflow flags carry flags recordings zero flags zero flags temporary registers move reflects operation result sign-extend ext) zero-extend c-ext) zero-extend de-bit data defined whose upper 32-bit data are in MDR and lower 32-bit in register Dn within "{}". hexadecimal(hexadecimal following to 0x.)	^(`\	bit shift(right/left)	with cache	
overflow flags carry flags negative flags zero flags temporary registers move reflects operation result sign-extend ext) sign-extend c-ext) de-bit data defined whose upper 32-bit data are in MDR and lower 32-bit in register Dn within "{}". hexadecimal(hexadecimal following to 0x.)		performsa bit shift for specified value	Instructions/data :access to cachable area and hit the	chache)
carry flags negative flags zero flags temporary registers move reflects operation result sign-extend ext) sign-extend 5.Dn} 64-bit data defined whose upper 32-bit data are in MDR and lower 32-bit in register Dn within "{}". hexadecimal(hexadecimal following to 0x.)	VF	overflow flags		
negative flags zero flags temporary registers move reflects operation result sign-extend ext) zero-extend c-ext) de-bit data defined whose upper 32-bit data are in MDR and lower 32-bit in register Dn within "()". hexadecimal(hexadecimal following to 0x.)	Q.	carry flags	Please see the LSI manuals for how the pipeline installation affer	sts the cycles.
zero flags temporary registers move reflects operation result sign-extend ext) zero-extend c-ext) de4-bit data defined whose upper 32-bit data are in MDR and lower 32-bit in register Dn within "()". hexadecimal(hexadecimal following to 0x.)	ΝF	negative flags	If using extended instructions, the users define the cycles.	
temporary registers move reflects operation result ext) sign-extend _ext) zero-extend 4-bit data defined whose upper 32-bit data are in MDR and lower 32-bit in register Dn within "{}". hexadecimal(hexadecimal following to 0x.)	ZF	zero flags		
move reflects operation result sign-extend ext) zero-extend c.Dn} 64-bit data defined whose upper 32-bit data are in MDR and lower 32-bit in register Dn within "()". hexadecimal(hexadecimal following to 0x.)	temp	temporary registers	■ Format	
_ext) _ext) ;,Dn}		move	Please refer to Chapter 1 Overview	
_ext) _ext) ;,Dn}		reflects operation result		
_ext) ;,Dn}	(sign_ext)	sign-extend		
('Du')	(zero_ext)	zero-extend		
	{MDR,Dn}	64-bit data defined whose upper 32-bit data are in MDR and lower 32-bit ir		
		register Dn within "{}".		
	0x	hexadecimal(hexadecimal following to 0x.)		

1st byte																
Upper/Low	er 0	1	2	3	4	5	6	7	8	9	Α	В	С	D	Е	F
0		1OV),(abs16	MOVBU D0,(abs16)	MOVHU D0,(abs16)	CLR D1	MOV D1,(abs16	MOVBU 6)D1,(abs16	MOVHU D1,(abs16)	CLR D2	MOV D2,(abs16	MOVBU D2,(abs16	MOVHU D2,(abs16)	CLR D3	MOV D3,(abs16	MOVBU D3,(abs16)	MOVHU D3,(abs16)
1		EXT	ΓB Dn			EXT	BU Dn			EXT	H Dn			EXTI	HU Dn	
2	Α	DD ir	mm8,Ar	n		MOV ir	mm16, <i>F</i>	∖n		ADD ir	nm8,Dr	า	ı	MOV in	nm16,D	n
3	M	OV (a	abs16),l	Dn	М	OVBU	(abs16	5),Dn	М	OVHU	(abs16)),Dn		MOV	SP,An	
4	INC D0	INC A0	MOV D0,(d8,SP)	MOV A0,(d8,SP)	INC D1	INC A1	MOV D1,(d8,SP	MOV () A1,(d8,SP)	INC D2	INC A2	MOV D2,(d8,SP)	MOV A2,(d8,SP)	INC D3	INC A3	MOV D3,(d8,SP)	MOV A3,(d8,SP)
5		IN	C4 An			ASI	L2 Dn			MOV (d	8,SP),[On	N	ЛОV (d	8,SP), <i>F</i>	۸n
6		MOV	Dm,(A	n)												
7		MOV	′ (Am),[On												
8		MOV	Dm,Dr	n (If m=	nMOV	, imm8	3,Dn)									
9		MOV	′ Am,Ar	(If m=r	nMOV,	, imm8	3,An)									
Α		СМР	Dm,Dr	n (If m=	n, CM	P imm	8,Dn)									
В		CMP	Am,An	(If m=r	n, CMF	imm8	,An)									
С		BGT (d8,PC)	BGE (d8,PC)	BLE (d8,PC)	BCS (d8,PC)	BHI (d8,PC)	BCC (d8,PC)	BLS (d8,PC)	BEQ (d8,PC)	BNE (d8,PC)	BRA (d8,PC)	NOP	JMP (d16,PC)	CALL (d16,PC)	MOVM (SP),regs	MOVM regs,(SP)
D	LLT	LGT	LGE	LLE	LCS	LHI	LCC	LLS	LEQ	LNE	LRA	SETLB	JMP (d32,PC)	CALL (d32,PC)	RETF	RET
Е		ADD	Dm,Dn													
F			C	ode extension	on (2-byte)				Code ex (3-byte)	tension	Code ext	ension	Code exte (6-byte)	ension	Code exten (7-byte)	sion

2nd byte (1st byte:F0) Instruction for 2-byte 5 6 7 8 9 A B C D E F 2 Upper/lower 0 MOV (Am),An 0 1 MOV Am,(An) 2 3 4 MOVBU (Am),Dn 5 MOVBU Dm,(An) 6 MOVHU (Am),Dn 7 MOVHU Dm,(An) 8 BSET Dm,(An) 9 BCLR Dm,(An) Α В С D Ε F RETS RTI TRAP CALLS (An) JMP (An)

per/lower	0 1	2	3	4	5	6	7	8	9	Α	В	С	D	Ε	F
0	SUB	Dm,Dn													
1	SUB	Am,Dn													
2	SUB	Dm,An													
3	SUB .	Am,An													
4	ADDO	Dm,D	n												
5	ADD	Am,Dn													
6	ADD	Dm,An													
7	ADD	Am,An													
8	SUBC	Dm,D	n												
9	CMP	Am,Dn													
Α	CMP	Dm,An													
В															
С															
D	MOV	Am,Dn													
Е	MOV	Dm,An													
F 2nd byte(1	Ist byte:F2) Ins				5	6	7	0	0	Λ	D	<u> </u>	n		
F 2nd byte(1	0 1	2	3	yte 4	5	6	7	8	9	A	В	С	D	E	F
F 2nd byte(1	0 1 AND	2 Dm,Dn	3		5	6	7	8	9	A	В	С	D	E	F
F 2nd byte(1 per/lower 0 1	0 1 AND OR D	2 Dm,Dn m,Dn	3		5	6	7	8	9	A	В	С	D	E	F
F 2nd byte(1 per/lower 0 1 2	0 1 AND OR D XOR	2 Dm,Dn Dm,Dn Dm,Dn	3		5	6	7	8	9	A	В	С	D	E	F
F 2nd byte(1 per/lower 0 1 2 3	0 1 AND OR D XOR NOT	2 Dm,Dn Dm,Dn Dm,Dn	3		5	6	7	8	9	A	В	С	D	E	F
F 2nd byte(1) per/lower 0 1 2 3 4	0 1 AND OR D XOR NOT MUL	2 Dm,Dn Dm,Dn Dm,Dn Dn Dn	3		5	6	7	8	9	A	В	С	D	E	F
F 2nd byte(1) per/lower 0 1 2 3 4 5	0 1 AND OR D XOR NOT MUL	2 Dm,Dn Dm,Dn Dm,Dn Dn Dm,Dn J Dm,D	3		5	6	7	8	9	A	В	C	D	E	F
F 2nd byte(1) per/lower 0 1 2 3 4 5 6	0 1 AND OR D XOR NOT MUL DIV D	2 Dm,Dn Dm,Dn Dm,Dn Dn Dm,Dn J Dm,D	3 1 On		5	6	7	8	9	A	В	C	D	E	
F 2nd byte(1per/lower 0 1 2 3 4 5 6 7	0 1 AND OR D XOR NOT MUL DIV D	2 Dm,Dn Dm,Dn Dn Dn Dm,Dn J Dm,D Dm,Dn Dm,Dn	3 1 On				7	8	9	A	В	C	D	E	
F 2nd byte(1) per/lower 0 1 2 3 4 5 6 7 8	0 1 AND OR D XOR NOT MUL MULU DIV D ROL	2 Dm,Dn Dm,Dn Dn Dm,Dn Dm,Dn Dm,Dn Dm,Dn	3 1 On			6 PR Dn	7	8	9	A	В	C	D	E	F
F 2nd byte(1per/lower 0 1 2 3 4 5 6 7	0 1 AND OR D XOR NOT MUL DIV D DIVU ROL ASL I	2 Dm,Dn Dm,Dn Dn Dm,Dn J Dm,Dn Dm,Dr Dm,Dr	3 On				7	8	9	A	В	C	D	E	
F 2nd byte(1) per/lower 0 1 2 3 4 5 6 7 8 9	0 1 AND OR D XOR NOT MUL DIV D ROL ASL I	Dm,Dn Dm,Dn Dm,Dn Dm,Dn Dm,Dn Dm,Dn Dm,Dn Dm,Dn Dm,Dn Dm,Dn Dm,Dn	3 On				7	8	9	A	В	C	D	E	F
Per/lower 0 1 2 3 4 5 6 7 8 9 A	0 1 AND OR D XOR NOT MUL DIV D ROL ASL I	2 Dm,Dn Dm,Dn Dn Dm,Dn J Dm,Dn Dm,Dr Dm,Dr	3 On				7	8	9	A	В	C	D	E	
F 2nd byte(1) per/lower 0 1 2 3 4 5 6 7 8 9 A B	0 1 AND OR D XOR NOT MUL DIV D ROL ASL I	Dm,Dn Dm,Dn Dm,Dn Dm,Dn Dm,Dn Dm,Dn Dm,Dn Dm,Dn Dm,Dn Dm,Dn Dm,Dn Dm,Dn	3 On				7	8	9	A	В	C	D	E	F
F 2nd byte(1) per/lower 0 1 2 3 4 5 6 7 8 9 A B C	0 1 AND OR D XOR NOT MUL DIV D ROL ASL I LSR ASR	2 Dm,Dn Dm,Dn Dm,Dn Dm,Dn Dm,Dn Dm,Dn Dm,Dn Dm,Dn Dm,Dn Dm,Dn Dm,Dn	3 On		RC			8	9	A	В	C	D	E	F



Upper/lower	0	1	2	3	4	5	6	7	8	9	Α	В	С	D	Ε	F
0		UDF2	0 Dm,[On												
1		UDF2	1 Dm,[On												
2		UDF2	2 Dm,[On												
3		UDF2	3 Dm,[On												
4		UDF2	4 Dm,[On												
5		UDF2	5 Dm,[On												
6		UDF2	6 Dm,E	On												
7		UDF2	7 Dm,[)n												
8		UDF2	8 Dm,[)n												
9		UDF2	9 Dm,[On												
Α		UDF3	0 Dm,[On												
В		UDF3	1 Dm,[On												
С		UDF3	2 Dm,[On												
D		UDF3	3 Dm,[On												
E		UDF3	4 Dm,[On												
F		UDF3	5 Dm,[)n												

2nd byte	(1st by	te: F6) I	nstruction	on for 2	-byte)											
Upper/lower	0	1	2	3	4	5	6	7	8	9	Α	В	С	D	Ε	F
0		UDFO	00 Dm,l	Dn												
1		UDFO)1 Dm,I	Dn												
2		UDF)2 Dm,[Dn												
3		UDF)3 Dm,l	Dn												
4		UDFO)4 Dm,[Dn												
5		UDFO)5 Dm,l	Dn												
6		UDFO)6 Dm,l	Dn												
7		UDFO)7 Dm,l	Dn												
8		UDF)8 Dm,l	Dn												
9		UDF)9 Dm,l	Dn												
Α		UDF1	10 Dm,I	Dn												
В		UDF1	l1 Dm,l	Dn												
С		UDF1	12 Dm,I	Dn												
D		UDF1	13 Dm,I	Dn												
E		UDF1	14 Dm,[Dn												
F		UDF1	15 Dm,[Dn												

2nd byte (1st byte:F8) Instruction for 3-byte

Upper/lower	0	1	2	3	4	5	6	7	8	9	Α	В	С	D	Е	F
0		MOV ((d8,An	n),Dn												
1		MOV I	Dm,(d	3,An)												
2		MOV (d8,An	n),An												
3		MOV	Am,(d8	3,An)												
4		MOVE	8U (d8	,Am),Dn												
5		MOVE	BU Dm	,(d8,An)												
6		MOVH	1U (d8	,Am),Dr	l											
7		MOVH	łU Dm	,(d8,An))											
8		_												_		
9		ľ	MOVBU 00,(d8,SP)	MOVHU D0,(d8,SP)			MOVBU D1,(d8,SP)	MOVHU D1,(d8,SP)			MOVBU D2,(d8,SP)	MOVHU D2,(d8,SP)			MOVBU I D3,(d8,SP)	MOVHU 03,(d8,SP)
Α																
В									M	OVBU	(d8,SP)),Dn	М	OVHU (d8,SP),	Dn
С	Д	ASL im	m8,Dn			LSR in	nm8,Dn	ı		ASR ir	nm8,Dr	۱				
D				•												
Е	А	ND im	m8,Dn			OR in	nm8,Dn		BVC (d8,PC)	BVS (d8,PC)	BNC (d8,PC)	BNS (d8,PC)		BTST ir	nm8,Dr	1
F	М	Sb) VO	,An),S	Р	N	10V SI	P,(d8,A	n)						·	ADD imm8,SP	

2nd byte (1st byte:F9) Instruction for 3-byte

Upper/lower	r 0 1	2	3	4	5	6	7	8	9	Α	В	С	D	Е	F
0	UDF00 i	mm8,Dr	1	UE	FU00	imm8,l	Dn	U	IDF20 ir	nm8,Dı	า*	UI	DFU20	imm8,l	On*
1	UDF01 i	mm8,Dr	1	UE	FU01	imm8,l	Dn	U	IDF21 ir	nm8,Dı	า*	UI	DFU21	imm8,l	On*
2	UDF02 i	mm8,Dr	1	UE	FU02	imm8,l	Dn	U	IDF22 ir	nm8,Dı	า*	UI	DFU22	imm8,l	On*
3	UDF03 i	mm8,Dr	1	UE	FU03	imm8,l	Dn	U	IDF23 ir	nm8,Dı	n*	UI	DFU23	imm8,I	On*
4	UDF04 i	mm8,Dr	1	UE	FU04	imm8,l	Dn	U	IDF24 ir	nm8,Dı	า*	UI	DFU24	imm8,l	On*
5	UDF05 ii	mm8,Dr	1	UE	FU05	imm8,l	Dn	U	IDF25 ir	nm8,Dı	า*	UI	DFU25	imm8,l	On*
6	UDF06 i	mm8,Dr	1	UE	FU06	imm8,l	Dn	U	IDF26 ir	nm8,Dı	า*	UI	DFU26	imm8,I	On*
7	UDF07 i	mm8,Dr	1	UE	FU07	imm8,l	Dn	U	IDF27 ir	nm8,Dı	า*	UI	DFU27	imm8,l	On*
8	UDF08 i	mm8,Dr	١	UE	FU08	imm8,l	Dn	U	IDF28 ir	nm8,Dı	า*	UI	DFU28	imm8,I	On*
9	UDF09 i	mm8,Dr	1	UE	FU09	imm8,l	Dn	U	IDF29 ir	nm8,Dı	า*	UI	DFU29	imm8,I	On*
Α	UDF10 i	mm8,Dr	١	UE	FU10	imm8,l	Dn	U	IDF30 ir	nm8,Dı	า*	UI	DFU30	imm8,I	On*
В	UDF11 i	mm8,Dr	1	UE	FU11	imm8,l	Dn	U	IDF31 ir	nm8,Dı	า*	UI	DFU31	imm8,I	On*
С	UDF12 i	mm8,Dr	1	UE	FU12	imm8,l	Dn	U	IDF32 ir	nm8,Dı	n*	UI	DFU32	imm8,I	On*
D	UDF13 i	mm8,Dr	1	UE	FU13	imm8,l	Dn	U	IDF33 ir	nm8,Dı	า*	UI	DFU33	imm8,l	On*
Е	UDF14 i	mm8,Dr	1	UE	FU14	imm8,l	Dn	U	IDF34 ir	nm8,Dı	า*	UI	DFU34	imm8,I	On*
F	UDF15 i	mm8,Dr	1	UE	FU15	imm8,l	Dn	U	IDF35 ir	nm8,Dı	า*	UI	DFU35	imm8,l	On*



^{*:} Installed for AM30/AM32. Not used for AM30.

2nd byte (1st byte: F4) Instruction for 4-byte

pper/lower	0	1	2	3	4	5	6	7	8	9	Α	В	С	D	Е	F
0		MOV	(d16,Aı	m),Dn												
1		MOV	Dm,(d1	6,An)												
2		MOV	(d16,Aı	m),An												
3		MOV	Am,(d1	6,An)												
4		MOVI	BU (d16	5,Am),[On											
5		MOVI	BU Dm,	,(d16,A	n)											
6		MOV	HU (d16	3,Am),[On											
7		MOVI	HU Dm	,(d16,A	n)											
8	MOV A0,(abs16)				MOV A1,(abs16)				MOV A2,(abs16)				MOV A3,(abs16)			
9	MOV A0,(d16,SP)	MOV D0,(d16,SP)	MOVBU D0,(d16,SP)I	MOVHU D0,(d16,SP)	MOV A1,(d16,SP)	MOV D1,(d16,SP)	MOVBU D1,(d16,SP	MOVHU) D1,(d16,SP)	MOV A2,(d16,SP)	MOV D2,(d16,SP)	MOVBU D2,(d16,SP)	MOVHU D2,(d16,SP)	MOV A3,(d16,SP)	MOV D3,(d16,SP)	MOVBU D3,(d16,SP)	MOVHU)D3,(d16,SP)
Α	N	ЛОV (а	bs16),A	ın				•								
В	М	OV (d1	16,SP), <i>F</i>	٩n	Ν	1 OV (d1	16,SP)	,Dn	МС	VBU (d16,SP),Dn	МО	VHU (d	d16,SP),Dn
С	A	ADD im	m16,Dr	n					(CMP im	nm16,D	n				
D	P	ADD im	m16,Ar	1					(CMP in	nm16,A	ın				
E	P	AND im	m16,Dr	n		OR im	m16,D	n		XOR in	nm16,E)n		BTST i	mm16,	Dn
F	BS	ET imr	m8,(d8,	An)	ВС	CLR im	m8,(d8	B,An)	ВТ	ST imi	n8,(d8	An)	AND imm16,PSW	OR imm16,PSW	ADD imm16,SP	CALLS (d16,PC)

2nd byte (1st byte: FB) Instruction for 4-byte 1

Upper/lower 0

3 5 F 6 0 UDF00 imm16,Dn UDFU00 imm16,Dn UDF20 imm16,Dn* UDFU20 imm16,Dn* UDF01 imm16,Dn UDFU01 imm16,Dn UDF21 imm16,Dn* UDFU21 imm16,Dn* 1 2 UDF22 imm16,Dn* UDFU22 imm16,Dn* UDF02 imm16,Dn UDFU02 imm16,Dn 3 UDF03 imm16,Dn UDFU03 imm16,Dn UDF23 imm16,Dn* UDFU23 imm16,Dn* 4 UDF04 imm16,Dn UDFU04 imm16,Dn UDF24 imm16,Dn* UDFU24 imm16,Dn* 5 UDF05 imm16,Dn UDFU05 imm16,Dn UDF25 imm16,Dn* UDFU25 imm16,Dn* 6 UDF06 imm16,Dn UDFU06 imm16,Dn UDF26 imm16,Dn* UDFU26 imm16,Dn* 7 UDF07 imm16,Dn UDFU07 imm16,Dn UDF27 imm16,Dn* UDFU27 imm16,Dn* 8 UDF08 imm16,Dn UDFU08 imm16,Dn UDF28 imm16,Dn* UDFU28 imm16,Dn* 9 UDF09 imm16,Dn UDFU09 imm16,Dn UDF29 imm16,Dn* UDFU29 imm16,Dn* Α UDFU30 imm16,Dn* UDF10 imm16,Dn UDFU10 imm16,Dn UDF30 imm16,Dn* В UDF11 imm16,Dn UDFU11 imm16,Dn UDF31 imm16,Dn* UDFU31 imm16,Dn* C UDF12 imm16,Dn UDFU12 imm16,Dn UDF32 imm16,Dn* UDFU32 imm16,Dn* D UDF13 imm16,Dn UDFU13 imm16,Dn UDF33 imm16,Dn* UDFU33 imm16,Dn* Ε UDF14 imm16,Dn UDFU14 imm16,Dn UDF34 imm16,Dn* UDFU34 imm16,Dn* F UDF15 imm16,Dn UDFU15 imm16,Dn UDFU35 imm16,Dn* UDF35 imm16,Dn*

7

9

Α

В

С

D

Ε



^{*:} Installed for AM31/AM32. Not used for AM30.

2nd byte (1st byte: FC) Instruction for 6-byte

Upper/lower	r 0	1	2	3	4	5	6	7	8	9	Α	В	С	D	Е	F
0		MOV	(d32,A	m),Dn												
1		MOV	Dm,(d	32,An)												
2		MOV	(d32,A	m),An												
3		MOV	Am,(d3	32,An)												
4		MOVI	BU (d3	2,Am),l	Dn											
5		MOVI	BU Dm	,(d32, <i>P</i>	\n)											
6		MOVI	HU (d3	2,Am),	Dn											
7		MOVI	HU Dm	,(d32, <i>F</i>	An)											
8	MOV A0,(abs32)	MOV D0,(abs32)	MOVBU D0,(abs32)	MOVHU D0,(abs32)	MOV A1,(abs32)	MOV D1,(abs32)	MOVBU D1,(abs32)	MOVHU D1,(abs32)	MOV A2,(abs32)	MOV D2,(abs32)	MOVBU D2,(abs32)	MOVHU D2,(abs32)	MOV A3(abs32)	MOV D3,(abs32)	MOVBU D3,(abs32)	MOVHU D3,(abs32)
9	MOV A0,(d32,SP)	MOV D0,(d32,SP)	MOVBU D0,(d32,SP)	MOVHU D0,(d32,SP)	MOV A1,(d32,SP)	MOV D1,(d32,SP)	MOVBU D1,(d32,SP)	MOVHU D1,(d32,SP)	MOV A2,(d32,SP)	MOV D2,(d32,SP)	MOVBU D2,(d32,SP)	MOVHU D2,(d32,SP)	MOV A3,(d32,SP)	MOV D3,(d32,SP	MOVBU D3,(d32,SP)	MOVHU D3,(d32,SP)
Α	N	IOV (a	bs32),/	٩n		MOV	(abs32	2),Dn	М	OVBU (abs32)	,Dn	М	DVHU	(abs32)),Dn
В	М	OV (d3	32,SP),	An	l N	MOV (d	32,SP),	,Dn	МС	OVBU (d32,SP),Dn	МО	VHU (d32,SP),Dn
С	A	NDD im	m32,D	n		SUB in	nm32,D)n		CMP in	nm32,[)n	ı	MOV in	nm32,E)n
D	ADD imm32,An SUB imm32,An CMP imm32,An MOV imm32,An							\n								
Е	A	ND im	m32,D	n		OR im	m32,D	n		XOR im	m32,D	n	E	BTST in	mm32,[On
F															ADD imm32,SP	CALLS (d32,PC)

2nd byte (1st byte: FD) Instruction for 6-byte

Upper/lower 0 7 9 В С D Ε F 5 6 8 Α 0 UDF00 imm32,Dn UDFU00 imm32,Dn UDF20 imm32,Dn* UDFU20 imm32,Dn* 1 UDF01 imm32,Dn UDFU01 imm32,Dn UDF21 imm32,Dn* UDFU21 imm32,Dn* 2 UDF02 imm32,Dn UDF22 imm32,Dn* UDFU02 imm32,Dn UDFU22 imm32,Dn* 3 UDF03 imm32,Dn UDFU03 imm32,Dn UDF23 imm32,Dn* UDFU23 imm32,Dn* 4 UDF04 imm32,Dn UDFU04 imm32,Dn UDF24 imm32,Dn* UDFU24 imm32,Dn* 5 UDF05 imm32,Dn UDFU05 imm32,Dn UDF25 imm32,Dn* UDFU25 imm32,Dn* 6 UDFU26 imm32,Dn* UDF06 imm32,Dn UDFU06 imm32,Dn UDF26 imm32,Dn* 7 UDF07 imm32,Dn UDFU07 imm32,Dn UDF27 imm32,Dn* UDFU27 imm32,Dn* 8 UDF08 imm32,Dn UDFU08 imm32,Dn UDF28 imm32,Dn* UDFU28 imm32,Dn* 9 UDF09 imm32,Dn UDFU09 imm32,Dn UDF29 imm32,Dn* UDFU29 imm32,Dn* Α UDF10 imm32,Dn UDFU10 imm32,Dn UDF30 imm32,Dn* UDFU30 imm32,Dn* В UDF11 imm32,Dn UDFU11 imm32,Dn UDF31 imm32,Dn* UDFU31 imm32,Dn* С UDF12 imm32,Dn UDFU12 imm32,Dn UDF32 imm32,Dn* UDFU32 imm32,Dn* D UDF13 imm32,Dn UDFU13 imm32,Dn UDF33 imm32,Dn* UDFU33 imm32,Dn* Ε UDF14 imm32,Dn UDFU14 imm32,Dn UDF34 imm32,Dn* UDFU34 imm32,Dn* F UDF15 imm32,Dn UDFU15 imm32,Dn UDF35 imm32,Dn* UDFU35 imm32,Dn*



^{*:} Installed for AM31/AM32. Not used for AM30.

	e (1st byte:															
Upper/lower	0	1	2	3	4	5	6	7	8	9	Α	В	С	D	Е	F
0	BSET imm8, B((abs32) (a	CLR imm8, E bs32) (BTST imm8 (abs32)													
1																
2																
3																
4																
5																
6																
7																
8	BSET imm8, BC (abs16)* (a	CLR imm8, B	BTST imm8, (abs16)*													
9		•														
Α																
В																
С																
D																
Е																
F																
							* - 1	netalled	l for ΔN	//32 No	nt used	for AM	30/AM	31		
		1	AM3	0	AN	131	. 11	istance	1101 / 11	VIOZ. INC	n uscu	IOI / IV	00// ((1)	J 1 .		
and but	. (1 at by tag	1 7	AM3		AN	131	. 11	istance	10174	VIOZ. INC	n uscu	IOI AIV	00// ((V))	J1.		
2nd byte Upper/lowe	e (1st byte: er 0	F7) re:		map	4	umun	6	7	8	9	А	В	C	D	E	F
Upper/lowe	e (1st byte: er 0		served		unnu	131 11111111 5									Е	F
Upper/lowe	e (1st byte: er 0		served	map	unnu	umun									Е	F
Upper/lowe 0 1	e (1st byte: er 0		served	map	unnu	umun									E	F
Upper/lowe 0 1 2	e (1st byte: er 0		served	map	unnu	umun									Е	F
Upper/lowe 0 1 2 3	e (1st byte: er 0		served	map	unnu	umun									Е	F
Upper/lowe 0 1 2 3 4	e (1st byte: er 0		served	map	unnu	umun									Е	F
Upper/lowe 0 1 2 3 4 5	e (1st byte: er 0		served	map	unnu	umun									Е	F
Upper/lowe 0 1 2 3 4 5 6	e (1st byte: er ()		served	map	unnu	umun									E	F
Upper/lowe 0 1 2 3 4 5 6 7	e (1st byte: er ()		served	map	unnu	umun									E	F
Upper/lowe 0 1 2 3 4 5 6 7 8	e (1st byte: er ()		served	map	unnu	umun									E	F
Upper/lowe 0 1 2 3 4 5 6 7 8 9	e (1st byte: er 0		served	map	unnu	umun									E	F
Upper/lowe 0 1 2 3 4 5 6 7 8 9 A	e (1st byte: er ()		served	map	unnu	umun									E	F
Upper/lowe 0 1 2 3 4 5 6 7 8 9 A B	e (1st byte: er ()		served	map	unnu	umun									E	F
Upper/lower 0 1 2 3 4 5 6 7 8 9 A B C	e (1st byte: er ()		served	map	unnu	umun									E	F
Upper/lowe 0 1 2 3 4 5 6 7 8 9 A B C	e (1st byte: er ()		served	map	unnu	umun									E	F
Upper/lower 0 1 2 3 4 5 6 7 8 9 A B C	e (1st byte: er ()		served	map	unnu	umun									E	F

5

Index

		BRA	label	. 81
Α		BSET	Dm,(An)	. 66
ADD	Am,An		imm8,(abs16)	. 67
	Am,Dn		imm8,(abs32)	. 67
	Dm,Dn		imm8,(d8,An)	. 67
	Dm,Dn	BTST	imm16,Dn	. 65
	imm16,An 47		imm32,Dn	. 65
	imm16,Dn 47		imm8,(abs16)	. 65
	imm16,SP		imm8,(abs32)	. 65
	imm32,An 47		imm8,(d8,An)	. 65
	imm32,Dn 47		imm8,Dn	. 65
	imm32,SP	BVC	label	. 81
	imm8, An	BVS	label	. 81
	imm8,Dn 47			
	imm8,SP	С		
ADDC	Dm,Dn	CALL	label	. 85
AND	Dm,Dn 59	CALLS	(An)	. 87
	imm16,Dn 59		label	. 88
	imm16,PSW 60	CLR	Dn	. 45
	imm32,Dn 59	CMP	Am,An	. 58
	imm8,Dn 59		Am,Dn	. 58
ASL	Dm,Dn		Dm,An	. 58
	imm8,Dn 77		Dm,Dn	. 58
ASL2	Dn 78		imm16,An	. 58
ASR	Dm,Dn 70		imm16,Dn	. 58
	Dn		imm32,An	. 58
	imm8,Dn 71		imm32,Dn	. 58
			imm8,An	. 58
В			imm8,Dn	. 58
BCC	label 81			
BCLR	Dm,(An) 68	D		
	imm8,(abs16) 69	DIV	Dm,Dn	. 54
	imm8,(abs32) 69	DIVU	Dm,Dn	. 55
	imm8,(d8,An)			
BCS	label 81	Е		
BEQ	label	EXT	Dn	. 40
BGE	label	EXTB	Dn	. 41
BGT	label 81	EXTBU	Dn	. 42
BHI	label 81	EXTH	Dn	. 43
BLE	label 81	EXTHU	Dn	. 44
BLS	label 81			
BLT	label 81	1		
BNC	label 81	INC	Dn	. 56
BNE	label 81		An	
BNS	label 81	INC4	An	

J				(d8,Am),An	. 27
JMР	(An)	84		(d8,Am),Dn	. 27
V1.11	label			(d8,Am),SP	. 27
JSR	(An)			(d16,SP),An	. 27
3510	label			(d16,SP),Dn	. 27
		75		(d32,SP),An	. 27
ı				(d32,SP),Dn	. 27
L				(d8,SP),An	. 27
LCC				(d8,SP),Dn	. 27
LCS				(Di,Am),An	. 27
LEQ				(Di,Am),Dn	. 27
LGE		82		Dm,(abs16)	. 28
LGT		82		Dm,(abs32)	. 28
LHI		82		Dm,An	. 26
LLE		82		Dm,(An)	
LLS		82		Dm,(d16,An)	
LLT		82		Dm,(d16,SP)	
LNE		82		Dm,(d32,An)	
LRA		82		Dm,(d32,SP)	
LSR	Dm,Dn	73		Dm,(d8,An)	
	Dn	75		Dm,(d8,SP)	
	imm8,Dn	74		Dm,(Di,An)	
				Dm,Dn	
M				Dm,MDR	
MOV	(abs16),An	27		Dm,PSW	
	(abs16),Dn			imm16,An	
	(abs32),An			imm16,Dn	
	(abs32),Dn			imm32,An	
	Am,(abs16)				
	Am,(abs32)			imm32,Dn	
	Am,An			imm8,An	
	Am,(An)			imm8,Dn	
	Am,(d16,An)			MDR,Dn	
	Am,(d16,SP)			PSW,Dn	
	Am,(d32,An)			SP,An	
				SP,(d8,An)	
	Am,(d32,SP)		MOVB	(abs16),Dn	
	Am,(d8,An)			(abs32),Dn	
	Am,(d8,SP)			(Am),Dn	
	Am,(Di,An)			(d16,Am),Dn	
	Am,Dn			(d32,Am),Dn	. 32
	Am,SP			(d8,Am),Dn	. 32
	(Am),An			(d16,SP),Dn	. 32
	(Am),Dn			(d32,SP),Dn	. 32
	(d16,Am),An			(d8,SP),Dn	. 32
	(d16,Am),Dn			(Di,Am),Dn	. 32
	(d32,Am),An			Dm,(abs16)	. 33
	(d32,Am),Dn	27			

	D (1.20)	22		D., (40 A.)	27
	Dm,(abs32)			Dm,(d8,An)	
	Dm,(An)			Dm,(d8,SP)	
	Dm,(d16,An)		MOMM	Dm,(Di,An)	
	Dm,(d16,SP)		MOVHU	J (abs16),Dn	
	Dm,(d32,An)			(abs32),Dn	
	Dm,(d32,SP)			(Am),Dn	
	Dm,(d8,An)			(d16,Am),Dn	
	Dm,(d8,SP)			(d32,Am),Dn	
	Dm,(Di,An)	. 33		(d8,Am),Dn	. 34
MOVBU	(abs16),Dn	. 30		(d16,SP),Dn	
	(abs32),Dn	. 30		(d32,SP),Dn	. 34
	(Am),Dn	. 30		(d8,SP),Dn	
	(d16,Am),Dn	. 30		(Di,Am),Dn	. 34
	(d32,Am),Dn	. 30		Dm,(abs16)	. 35
	(d8,Am),Dn	. 30		Dm,(abs32)	. 35
	(d16,SP),Dn	. 30		Dm,(An)	. 35
	(d32,SP),Dn	. 30		Dm,(d16,An)	. 35
	(d8,SP),Dn	. 30		Dm,(d16,SP)	. 35
	(Di,Am),Dn	. 30		Dm,(d32,An)	. 35
	Dm,(abs16)	. 31		Dm,(d32,SP)	. 35
	Dm,(abs32)	. 31		Dm,(d8,An)	. 35
	Dm,(An)	. 31		Dm,(d8,SP)	. 35
	Dm,(d16,An)	. 31		Dm,(Di,An)	. 35
	Dm,(d16,SP)	. 31	MOVM	regs,(SP)	. 39
	Dm,(d32,An)	. 31		(SP),regs	. 39
	Dm,(d32,SP)	. 31	MUL	Dm,Dn	. 52
	Dm,(d8,An)	. 31	MULU	Dm,Dn	. 53
	Dm,(d8,SP)	. 31			
	Dm,(Di,An)	. 31	Ν		
MOVH	(abs16),Dn	. 36	NOP		. 97
	(abs36),dn	. 36	NOT	Dn	. 64
	(Am),Dn	. 36			
	(d16,Am),Dn	. 36	0		
	(d32,Am),Dn	. 36	OR	Dm,Dn	. 61
	(d8,Am),Dn	. 36		imm16,Dn	. 61
	(d16,SP),Dn	. 36		imm16,PSW	. 62
	(d32,SP),Dn	. 36		imm32,Dn	. 61
	(d8,SP),Dn	. 36		imm8,Dn	. 61
	(Di,Am),Dn	. 36			
	Dm,(abs16)	. 37	R		
	Dm,(abs32)		ROL	Dn	80
	Dm,(An)	. 37	ROR	Dn	
	Dm,(d16,An)		RET	DII	
	Dm,(d16,SP)		RETF		
	Dm,(d32,An)		RETS		
	Dm,(d32,SP)		RTI		
			1/11		. ,,

RTS		94
S		
SETLB		83
SUB	Am,An	49
	Am,Dn	49
	Dm,An	49
	Dm,Dn	49
	imm32,An	50
	imm32,Dn	50
SUBC	Dm,Dn	51
T TRAP		96
U		
UDFnn	Dm,Dn	98
UDFnn	imm,Dn	99
UDFUnn	imm,Dn	00
Χ		
XOR	Dm,Dn	63
	imm16,Dn	63
	imm32,Dn	63

MN1030/MN103S Series Instruction Manual

January, 2003 4th Edition

Issued by Matsushita Electric Industrial Co., Ltd.

© Matsushita Electric Industrial Co., Ltd.

Semiconductor Company, Matsushita Electric Industrial Co., Ltd.

Nagaokakyo, Kyoto 617-8520, Japan Tel: (075) 951-8151

http://www.panasonic.co.jp/semicon/

SALES OFFICES

■ NORTH AMERICA

●U.S.A. Sales Office:

Panasonic Industrial Company

[PIC]

• New Jersey Office:

Two Panasonic Way Secaucus, New Jersey 07094 U.S.A. Tel: 1-201-348-5257 Fax:1-201-392-4652

• Chicago Office:

1707 N. Randall Road Elgin, Illinois 60123-7847 U.S.A. Tel: 1-847-468-5720 Fax:1-847-468-5725

• Milpitas Office:

1600 McCandless Drive Milpitas, California 95035 U.S.A. Tel: 1-408-942-2912 Fax:1-408-946-9063

Atlanta Office:

1225 Northbrook Parkway Suite 1-151 Suwanee, GA 30024 U.S.A.

Tel: 1-770-338-6953 Fax:1-770-338-6849

• San Diego Office:

9444 Balboa Avenue, Suite 185, San Diego, California 92123 U.S.A.

Tel: 1-619-503-2903 Fax:1-858-715-5545

●Canada Sales Office:

Panasonic Canada Inc. [PCI]

5770 Ambler Drive 27 Mississauga, Ontario, L4W 2T3 CANADA

Tel: 1-905-238-2315 Fax:1-905-238-2414

■ LATIN AMERICA

●Mexico Sales Office:

Panasonic de Mexico, S.A. de C.V. [PANAMEX]

Amores 1120 Col. Del Valle Delegacion Benito Juarez C.P. 03100 Mexico, D.F. MEXICO

Tel: 52-5-488-1000 Fax:52-5-488-1073

· Guadalajara Office:

SUCURSAL GUADALAJARA

Av. Lazaro Cardenas 2305 Local G-102 Plaza Comercial Abastos; Col. Las Torres Guadalajara, Jal. 44920 MEXICO

Tel: 52-3-671-1205 Fax:52-3-671-1256

●Brazil Sales Office:

Panasonic do Brasil Ltda. [PANABRAS]

Caixa Postal 1641, Sao Jose dos Campos, Estado de Sao Paulo

Tel: 55-12-335-9000 Fax:55-12-331-3789

■ EUROPE

●Europe Sales Office:

Panasonic Industrial Europe GmbH [PIE]

• U.K. Sales Office:

Willoughby Road, Bracknell, Berks., RG12 8FP, THE UNITED KINGDOM

Tel: 44-1344-85-3671 Fax:44-1344-85-3853

• Germany Sales Office:

Hans-Pinsel-Strasse 2 85540 Haar, GERMANY Tel: 49-89-46159-119 Fax:49-89-46159-195

■ ASIA

●Singapore Sales Office:

Panasonic Semiconductor of South Asia [PSSA] 300 Beach Road, #16-01, The Concourse, Singapore 199555 THE REPUBLIC OF SINGAPORE

Tel: 65-6390-3688 Fax:65-6390-3689

●Malaysia Sales Office:

Panasonic Industrial Company (M) Sdn. Bhd. [PICM]

Head Office:

Tingkat 16B, Menara PKNS Petaling Jaya, No.17, Jalan Yong Shook Lin 46050 Petaling Jaya, Selangor Darul Ehsan, MALAYSIA

Tel: 60-3-7951-6601 Fax:60-3-7954-5968

• Penang Office:

Suite 20-07,20th Floor, MWE Plaza, No.8, Lebuh Farquhar,10200 Penang, MALAYSIA

Tel: 60-4-201-5113 Fax:60-4-261-9989

Johore Sales Office:

Menara Pelangi, Suite8.3A, Level8, No.2, Jalan Kuning Taman Pelangi, 80400 Johor Bahru, Johor, MALAYSIA Tel: 60-7-331-3822 Fax:60-7-355-3996

●Thailand Sales Office:

Panasonic Industrial (THAILAND) Ltd.

PICTI

252-133 Muang Thai-Phatra Complex Building, 31st Fl. Rachadaphisek Rd., Huaykwang, Bangkok 10320, THAILAND

Tel: 66-2-693-3428 Fax:66-2-693-3422

●Philippines Sales Office:

[PISP]

Panasonic Indsutrial Sales Philippines Division of Matsushita Electric Philippines Corporation

102 Laguna Boulevard,Bo.Don Jose Laguna Technopark, Santa. Rosa, Laguna 4026 PHILIPPINES

Tel: 63-2-520-8615 Fax:63-2-520-8629

●India Sales Office:

National Panasonic India Ltd.

[NPI]

E Block, 510, International Trade Tower Nehru Place, New Delhi_110019 INDIA

Tel: 91-11-629-2870 Fax:91-11-629-2877

●Indonesia Sales Office:

P.T.MET & Gobel

[M&G]

JL. Dewi Sartika (Cawang 2) Jakarta 13630, INDONESIA Tel: 62-21-801-5666 Fax:62-21-801-5675

●China Sales Office:

Panasonic Industrial (Shanghai) Co., Ltd. [PI(SH)

Floor 6, Zhong Bao Mansion, 166 East Road Lujian Zui, PU Dong New District, Shanghai, 200120 CHINA

Tel: 86-21-5866-6114 Fax:86-21-5866-8000

Panasonic Industrial (Tianjin) Co., Ltd. [PI(TJ Room No.1001, Tianjin International Building 75, Nanjin

Road, Tianjin 300050, CHINA Tel: 86-22-2313-9771 Fax:86-22-2313-9770

Panasonic SH Industrial Sales (Shenzhen) Co., Ltd.

[PSI(SZ)]

7A-107, International Bussiness & Exhibition Centre, Futian Free Trade Zone, Shenzhen 518048, CHINA Tel: 86-755-8359-8500 Fax:86-755-8359-8516

Panasonic Shun Hing Industrial Sales (Hong Kong)
Co., Ltd. [PSI(HK)]

11th Floor, Great Eagle Center 23 Harbour Road, Wanchai, HONG KONG

Tel: 852-2529-7322 Fax:852-2865-3697

●Taiwan Sales Office:

Panasonic Industrial Sales (Taiwan) Co.,Ltd. [PIST]
• Head Office:

nead Office:

6F, 550, Sec. 4, Chung Hsiao E. RD. Taipei, 110, TAIWAN Tel: 886-2-2757-1900 Fax:886-2-2757-1906

• Kaohsiung Office:

6th Floor, Hsin Kong Bldg. No.251, Chi Hsien 1st Road Kaohsiung 800, TAIWAN

Tel: 886-7-346-3815 Fax:886-7-236-8362

●Korea Sales Office:

Panasonic Industrial Korea Co., Ltd. [PIKL]

Kukje Center Bldg. 11th Fl., 191 Hangangro 2ga, Youngsan-ku, Seoul 140-702, KOREA

Tel: 82-2-795-9600 Fax:82-2-795-1542

220103