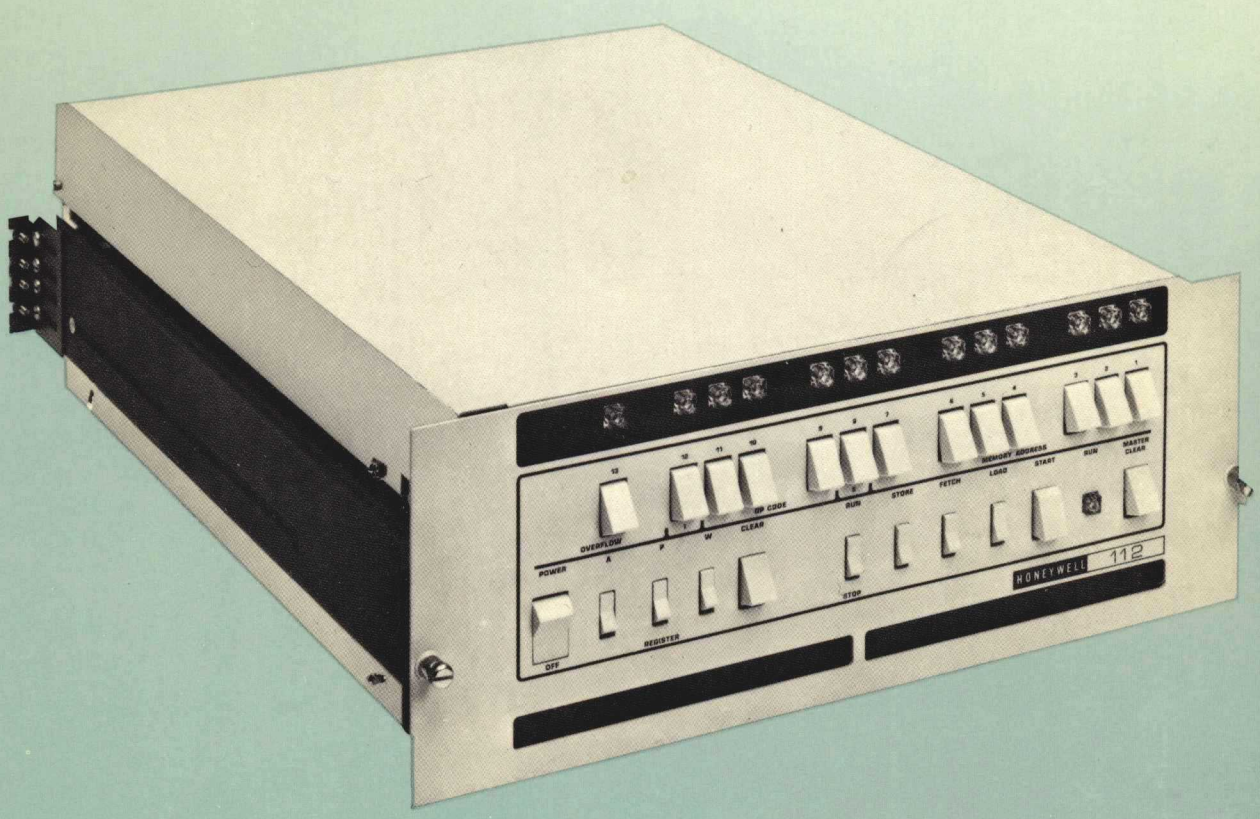
PROGRAMMERS REFERENCE MANUAL



H112 Digital Controller

Honeywell

Doc. No. 70130072242A
M-1164

H112 DIGITAL CONTROLLER

PROGRAMMERS REFERENCE MANUAL

October 1969

Honeywell

COMPUTER CONTROL DIVISION

Printed in U.S.A.

Published by the Publications Department,
Honeywell Inc., Computer Control Division

CONTENTS

CONTENTS (Cont)

SECTION IV
SAP-12 ASSEMBLER

CONTENTS (Cont)

CONTENTS (Cont)

ILLUSTRATIONS

CONTENTS (Cont)

TABLES

SECTION I
INTRODUCTION

The Honeywell H112 Controller is a low cost processor which features high speed, efficient use of memory, and the adaptability needed to tailor the unit to a wide variety of on-line, real-time control, data collection, and data-reduction applications. Honeywell has configured the H112 to meet the varied needs of control and systems engineers. This unit can be tailored to practically any application by the use of standard, off-the-shelf, plug-in modules. Memory is field expandable from 4 to 8K, using a standard 4K plug-in module. The plug-in approach also allows a user to share one control panel among several H112 installations by unplugging the panel once each unit is on-line.

SCOPE

The H112 Programmers Reference Manual is divided into seven sections with additional appendices which provide quick reference data. Section I provides leading particulars on the controller hardware. Section II contains a brief hardware block diagram description and machine instructions and functions. Section III contains a brief discussion of I/O signals and controller interface modes. Section IV discusses basic assembler specifications, the processing involved, and output formats for printed and punched data. Section V contains a brief table of machine instructions and all pseudo operations available to the programmer. Section VI provides programming instructions and includes examples. The operating instructions contained in Section VII provide the user with procedures for loading tapes, operating via the control panel or other devices, and communication between operator and controller when using the Utility Debug Program.

H112 CONTROLLER CHARACTERISTICS

Type:	Parallel, binary, stored program	
Addressing:	Single-word addressing with single-level indirect addressing	
Number System:	Two's complement	
Circuitry:	Integrated, DTL	
Typical Execution Times:	Load Accumulator	3.39 μs
	Add	7.63 μs
	Jump Unconditional	3.39 μs
	Clear Accumulator	2.54 μs
Instruction Complement:	Five instruction types, making up 37 standard instructions; skip instructions are microprogrammable	
Word Length:	12 bits	

Control Panel:	Optional control panel is plug-in. The panel has facilities for display of the A, P, or W registers as well as RUN/STOP, STORE, FETCH, START, LOAD, and MASTER CLEAR functions. Interlocks prevent accidental entry in the run mode.
Input/Output Rate:	One 12-bit word/3.39 μs or 295K words/sec (Direct Data Channel Option)
Input/Output Bus:	Party line with priority interrupt structure. Optional bidirectional direct data channels (2).
Memory:	Coincident current, random access, ferrite core; 12-bit word length; one or two 4K modules, field expandable to 8K. Memory cycle time 1.69 μs
Programming:	Standard Software includes: Assembler Debug/Utility Loader Math Library Diagnostics
Peripherals and Options:	10-cps ASR-33 paper tape reader/punch/printer High speed paper tape reader (400 cps) A second 4K memory module for 8K memory capability Real Time Clock Analog Input/Output Digital Input/Output Special options, such as magnetic tape, paper tape punch, and communication interfaces, are easily applied using standard Honeywell logic modules.
Physical Characteristics:	Power consumption: 200W, 115V ±10% single phase, 48 to 62 Hz Dimensions: 19-inch rack mount, 7 inches high by 26 inches deep Environment: Room ambient (less I/O devices): 0° to 50°C Storage, -65° to 150°C

REFERENCES

The publications listed below will assist programmers, engineers and technicians in installing, using, and servicing the H112 Controller:

H112 Installation and Interface, Doc. No. 70130072243

H112 Central Processor Description, Doc. No. 70130072244

SECTION II
MACHINE CHARACTERISTICS

H112 CONTROLLER BASIC OPERATION

The H112 system is comprised of the basic functional blocks shown in Figure 2-1. A brief description of each function follows.

Registers Available to the Programmer

A Register. -- The A register is the accumulator used in the H112 system. Its function is to perform arithmetic operations in conjunction with the serial adder and to contain the results. The A register (12-bit) also functions as an interface for I/O operations.

W Register. -- The W register is a 12-bit working register used for memory data interface and instruction execution.

P Register. -- The P register is the program counter and normally contains the address of the next instruction to be executed. In the standard 4K memory, the register has 12 bits. In the 8K memory controller (option), the register has 13 bits.

B Register. -- The B register is the single-bit bank register used with the 8K memory option to select memory banks; it is the most significant bit in all indirect pointer words.

Z Register. -- The Z register is a two-bit register used with the 8K memory option. After an interrupt, the Z register stores the B register bit and bit 13 of the P register so that the original program may be restored after the interrupt.

OV Register. -- The OV register is the overflow flip-flop (F/F) which stores the A register carry bit. It is used with Add and Shift instructions. With Add instructions, the OV F/F (register) will be set if two negative numbers of any size are added, or if a negative and positive number are added and the result is positive or zero. Shift instructions cause the OV F/F to be set if any logical 1's are shifted out of the A register.

Registers Unavailable to the Programmer

Y Register - The Y register is the memory address register. It is a 12-bit register for the standard 4K memory and 13-bit for the 8K memory option. This register stores the effective memory address during accessing operations.

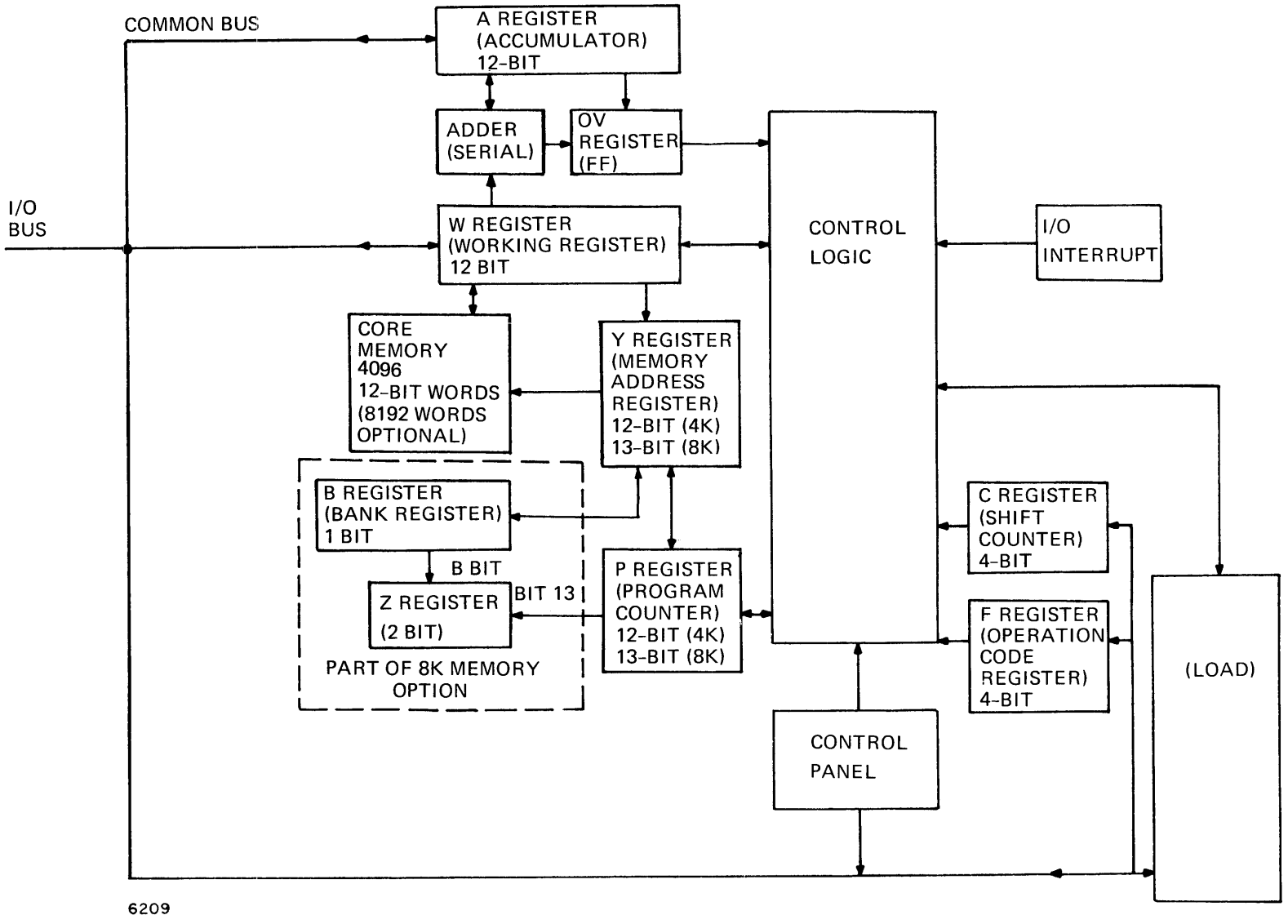


Figure 2-1. H112 Controller — System Block Diagram

C Register. -- The C register is a 4-bit shift counter which stores the required number of right shifts required when using shift instructions. Additionally, it serves as a counter during the execution of instructions involving the serial adder.

F Register. -- The F register is a 4-bit storage register which contains the operation code of the instruction being processed.

Core Memory

The 12-bit, 4K memory module used in the H112 is the standard Honeywell ICM-160 core memory unit.

In the basic controller, one module provides 4096 12-bit words of memory, with an additional 4096-word module available as an option. Direct addressing can access 256 storage words. Single-level indirect addressing accesses 4096 words. All storage and retrieval operations are in parallel, with all 12 bits of each word available for data.

Input/Output Characteristics

The standard I/O structure in the H112 is a single cable, duplex, "party line" system. Six I/O instructions, plus reassignable, programmed priority interrupts, make the H112's parallel transfer I/O bus capable of an unlimited variety of applications. Normal transfers are 12 bits in parallel to and from the accumulator; two device-controlled bidirectional Direct Data Channels are optional.

I/O characteristics have been carefully tailored to provide maximum interface flexibility with the widest possible interface tolerances. I/O signals are true in the ground state and are relatively immune to noise; the timing requirements have been chosen to allow ample time for a device to respond; strobe pulses are wide to eliminate capacitive losses. Timing is consistent for all I/O transfers, the I/O instructions all being of the same length. A stall instruction is included as part of the standard instruction complement to allow convenient synchronization (with very low latency time) to external devices.

Two direct data channels are available as options. These bidirectional channels are completely controlled by the peripheral devices, with no software required for devices to directly access memory. The address registers and range counters are external and are not stored in memory. A more detailed discussion of the DDC mode is provided in Section III.

Control Panel

The plug-in control panel used with the H112 allows considerable savings to large scale users. A user may buy and use only as many panels as he requires for startup and service, unplugging the panel once service is completed and then using it with a different controller. On large-scale unattended installations, such as oil well monitoring or pipeline remote control and monitoring, the panel can be part of the control engineer's and service technician's equipment. Once the panel is detached, unauthorized personnel cannot interfere with the controller operation, thus eliminating a problem common to widely scattered systems.

Panel Controls and Indicators (see Figure 2-2). --

Displays - Indicators are provided to display the contents of the A, P, or W registers. The control panel displays the contents of the A register following an LDA or INA instruction when the machine is running, with the overflow flop as bit 13. The other registers may be displayed when the machine is halted and the RUN switch is in the STOP position.

Switch Register - 13 switches are provided to allow entry of data for bits 1-13, which correspond to the display indicators.

Interlock - The RUN/STOP switch must be in the STOP position in order to use the panel.

REGISTER Select - Switches A, P, and W select the desired register for display or data entry.

CLEAR - The CLEAR switch clears the selected register.

RUN/STOP - The STOP position of the RUN/STOP switch halts the controller. Once in the stopped mode, registers can be selected, displayed, and modified; in this mode, the controller can be stepped using the START switch. The controller reads the instruction from memory, executes the instruction, increments the P register, and halts each time the START switch is depressed.

STORE - To store data, the memory location is entered in the P register, the STORE switch is depressed, and data is placed in the W register. The START switch is depressed to store the contents of the W register in the location specified by the program counter. The program counter is then automatically incremented.

FETCH - To retrieve data from a memory location, the location is entered in the P register and then the FETCH and START switches are depressed. The data is displayed in the W register. The P register is incremented.

LOAD - Load format tapes are loaded by setting the P register to the starting location, setting the RUN/STOP switch to RUN, depressing the LOAD switch, and then the START switch.

RUN - The RUN indicator, when lighted, signifies that the controller is running the stored program.

START - In addition to the Store, Load, etc functions, the START switch, when depressed, causes the controller to begin program execution at the address specified by the P register if the RUN/STOP switch is set to run.

MASTER CLEAR - The MASTER CLEAR switch clears main registers, plus control addressing, and I/O F/F's, but does not affect the A and W registers.

POWER/OFF - controls operating power to controller.

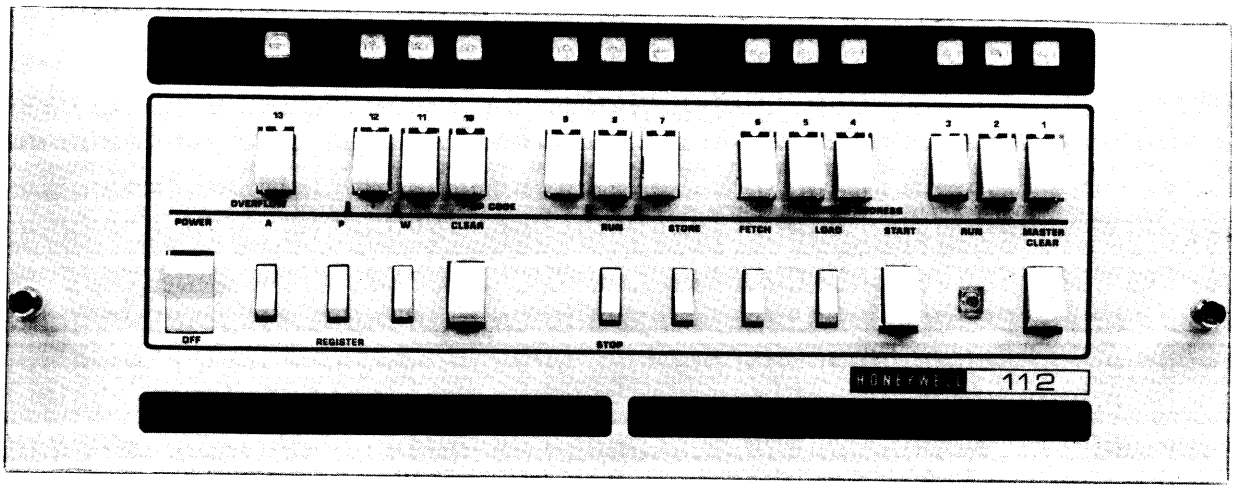


Figure 2-2. H112 Controller Control Panel

Load Mode. -- The H112 controller may be placed in the load mode either with or without a control panel. A control panel enables the operator to set the starting address of the loader to any desired point by inserting that location in the P register. Without a panel, loading will always begin at location zero.

Auxiliary Control

In addition to the standard control panel, or when full displays are not required for a specific application, plug-in connections are available for external control of the following functions: Run/Stop, Start, Master Clear, Load, and single instruction execution.

WORD ORGANIZATION

Two types of internal words are used by the H112: data and instruction. Each is a 12-bit word type and is discussed below.

Data Word Organization (Figure 2-3)

Data is represented by a 12-bit binary number. Bit 12 is the sign bit with 0 indicating a positive (or zero) number and a 1 indicating a negative number. Negative numbers are represented in two's complement. The data word can also be considered as a 12-bit binary integer (unsigned).

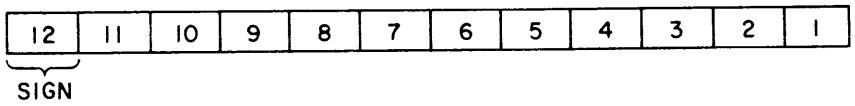


Figure 2-3. Data Word Organization

Instruction Word Organization (Figure 2-4)

Instructions are represented by a 12-bit binary number. The instruction word is divided into functional parts as shown below.

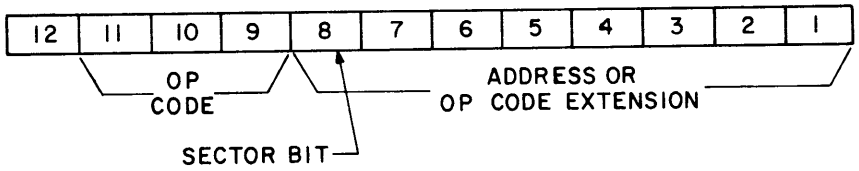


Figure 2-4. Instruction Word Organization

ADDRESSING

All memory reference instructions (Figure 2-5) refer to an effective address which is a function of the status of the machine, location of the instruction, and selected address modification mode. The memory reference instruction is organized as shown in Figure 2-5.

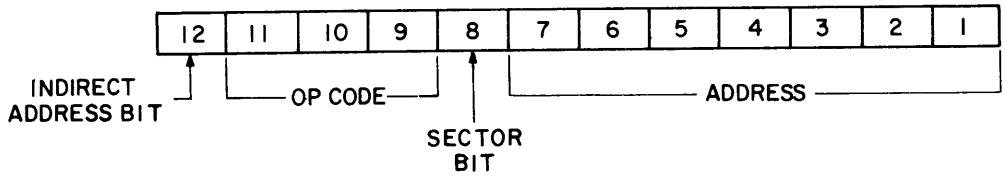


Figure 2-5. Memory Reference Instruction

Each of the seven H112 memory reference instructions (LDA, STA, ADD, ANA, IRS, JMP, JST) contains a seven bit address field, allowing 128 locations ($0 - 177_8$) to be directly referenced. As a result, memory is divided into sectors of 128 words each. The sectors begin at addresses 0, 200_8, 400_8, 600_8, etc. A 4K H112 memory contains 32 sectors numbered $0 - 37_8$; an 8K H112 memory contains 64 sectors numbered $0 - 77_8$.

Each memory reference instruction contains a sector bit. This bit determines the sector in which the direct address is located. If the bit is ZERO, the direct address is located in a primary sector (sector 0 or 40_8); if the bit is ONE, the direct address is located in the same sector as the instruction.

Each memory reference instruction contains an indirect bit. If this bit is a ZERO, the instruction operates on the contents of the direct address, or the effective address equals the direct address. If the indirect bit is a ONE, the instruction operates on the contents of the location specified by the direct address, or the effective address equals the contents of the location specified by the direct address. Indirect addressing adds 1.69 μs to the instruction execution times.

INSTRUCTION REPERTOIRE

Instructions for the H112 are divided into five groups: Memory Reference, Input/Output, Shift, Skip, and Generic. Each group is discussed in detail in the following.

Memory Reference Instructions

Memory reference instructions refer to an effective address which is a function of the machine status, location of the instruction, and address mode selected. The op code contained in bits 9, 10, and 11 of the word (see Figure 2-5) is an octal digit. A description of each op code is provided in the following paragraphs. The execution times listed for each instruction are without indirect addressing; indirect addressing adds 1.69 μs. Note that the operation of some instructions is modified under interrupt conditions.

LDA (Load A). --

Op Code:	1
Execution time:	3.39 μs
Function:	The contents of the memory location at the effective address are loaded into the accumulator. The previous contents of the accumulator are lost.

STA (Store A). --

Op Code:	2
Execution time:	3.39 μs
Function:	The contents of the accumulator are deposited in the memory cell at the effective address. The previous contents of the memory cell are lost. The contents of the accumulator is unaltered.

JMP (Jump). --

Op Code:	3
Execution time:	3.39 μs
Function:	The P register is loaded with the effective address which causes program control to transfer to that address. When this instruction is not indirectly addressed, P13 is not altered. When it is indirectly addressed, the contents of the bank register are

transferred to P13 and P13 to the bank register. The operation of this instruction is substantially altered when executed for the first time in the indirect addressing mode following the execution of an ITR (interrupt return) instruction. See the description of that instruction.

ADD (Add). --

Op Code: 4

Execution time: 7.63 μs

Function: The contents of the memory cell at the effective address are added to the contents of the accumulator and the sum deposited in the accumulator. If addition is considered to be an unsigned binary add, the overflow flop is set if there is a carry out of the most significant bit (which indicates that the register has overflowed). The overflow flop is useful for double precision arithmetic. If addition is considered to be in a two's complement fashion, the overflow flop is set if: (a) two negative numbers of any size are added, (b) a negative and a positive number are added and the result is positive or zero (positive $\geq$ magnitude of negative). The overflow flop is not set if: (a) two positive numbers of any size are added, (b) a negative and a positive number are added and the sum is negative (magnitude of negative > positive). Note that if the result does not set the overflow flop, the previous contents of the flip-flop are not altered.

ANA (And A). --

Op Code: 5

Execution time: 7.63 μs

Function: The contents of the memory cell at the effective address are logically ANDed with the contents of the accumulator and the results placed into the accumulator. The final value of any accumulator bit is a 1 if and only if both the initial accumulator bit in this position and the corresponding bit in the memory location are 1's.

IRS (Increment, Replace, and Skip). --

Op Code: 6

Execution time: 9.33 μs

Function: The contents of the memory cell at the effective address are incremented by 1 and the result returned to the same cell. If, after the addition, the result is 0, the next instruction is skipped.

JST (Jump and Store P). --

Op Code: 7

Execution time: 4.66 μs

Function: The address of the next sequential instruction is stored in the memory cell at the effective address. The next instruction executed is the contents of the memory cell following the effective address. Thus, the P register plus 1 (lower 12 bits only) is stored in the memory cell at the effective address and then the P register is loaded with the effective address plus 1. This instruction is used in interrupt response routines and has a different action under those conditions. (See the section on interrupts.)

When this instruction is not indirectly addressed, P 13 is not altered. When it is indirectly addressed, the contents of the bank register are transferred to P13 and P13 to the bank register.

The operation of this instruction is substantially altered when executed for the first time in the indirect addressing mode following the execution of an ITR instruction. See the description of that instruction.

Input/Output Instructions

The format of the I/O instruction word is shown in Figure 2-6. Most I/O instructions require an address which specifies not only the device to be used but also the nature of the command. The op codes listed below are the entire 12-bit instruction words read as four octal digits with the address (if present) as the second two digits, XX. Note that the stall instruction (STL) is described on page 2-14.

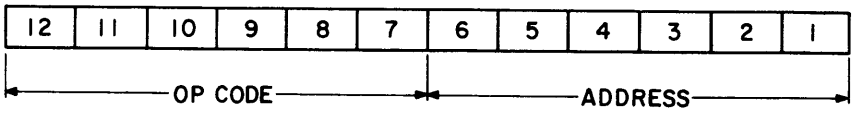


Figure 2-6. I/O Instructions

INA (Input Transfer to Accumulator). --

Op Code: 40XX

Execution time: 4.66 μs

Description: If the addressed device is ready and responds, the accumulator is cleared, the bit pattern on the I/O bus from the addressed device is transferred into the accumulator and the next sequential

instruction is skipped. If the addressed device is not ready or does not respond, the accumulator is cleared but the data is not transferred into the machine; the next sequential instruction is executed.

SKS (Skip if Set). --

Op Code: 41XX
Execution time: 4.66 μs
Description: If the device condition specified in the address field is true (set) and the device responds, the next sequential instruction is skipped. If the condition is not true (not set) or the device does not respond, the next sequential instruction is executed.

OTA (Output Transfer From Accumulator). --

Op Code: 42XX
Execution time: 4.66 μs
Description: If the addressed device is ready and responds, the contents of the accumulator are transferred to the device and the next sequential instruction is skipped. If the addressed device is not ready or does not respond, the transfer does not take place and the next sequential instruction is executed. In either case, the contents of the accumulator are unaltered.

OCP (Output Control Pulse). --

Op Code: 43XX
Execution time: 4.66 μs
Description: Upon execution of this instruction, a command pulse is delivered to the device (or device function) specified by the address field. The execution of this command is not contingent upon a device ready response; the following instruction will never be skipped. Addresses 00 and 01 are not available for OCP instructions. See SMK instruction.

SMK (Set Mask). --

Op Codes: 4300, 4301
Execution time: 4.66 μs
Description: These instructions transfer the contents of the accumulator to the interrupt mask bits of the individual peripheral devices. Generally, a peripheral device is assigned to one accumulator bit of one of the instructions. A 1 in the accumulator bit permits

an interrupt while a 0 not only inhibits further interrupts but also temporarily removes an interrupt request. Because of circuit speeds, this instruction may not be effective until after the next instruction. This instruction is not contingent upon device ready and the following instruction is never skipped.

Shift Instructions

The shift instructions shift the accumulator to the right the number of places specified by the shift count (Figure 2-7). The shift count, read as a positive integer, specifies the number of places to be shifted. A shift count of 01 causes a shift of 1 place; 02 causes a shift of 2 places, etc. A shift count of 00 causes a shift of 16 places. The op codes below are the entire 12-bit instruction words read as four octal digits with the shift count of YX. The variable N in the execution times below is the number of shifts specified.

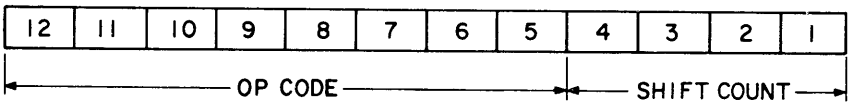


Figure 2-7. Shift Instructions

LGR (Logical Right Shift). --

Op Code: 01YX (Y = 0 or 1)
Execution time: 3.8 + N (0.424) μs
Description: The accumulator is shifted to the right with 0's filling from the left. If any 1's are shifted out of the accumulator the overflow flop is set. If no 1's are shifted out, the status of the overflow flop is unaltered.

RAR (Rotate A Right). --

Op Code: 01YX (Y = 4 or 5)
Execution time: 3.8 + N (0.424) μs
Description: The accumulator is rotated with the data out of bit 1 being shifted into bit 12. The status of the overflow flop is unaltered.

ARS (Arithmetic Right Shift). --

Op Code: 01YX (Y = 2 or 3)
Execution Time: 3.8 + N (0.424) μs
Description: The accumulator is shifted to the right with the sign being spread from the left. If any 1's are shifted out of the accumulator, the

overflow flop will be set. If no 1's are shifted out, the status of the overflow flop is unaltered.

Skip Instructions

The entire 12 bits of the skip instruction word is op code. The op code is microprogrammed as shown in Figure 2-8.

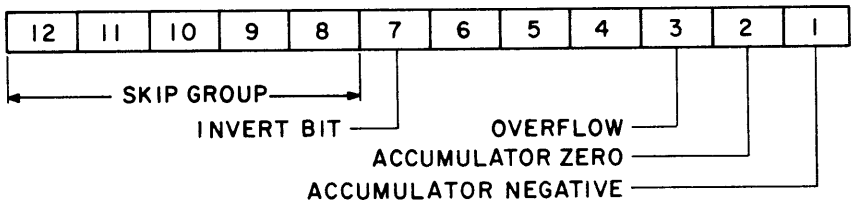


Figure 2-8. Skip Instructions

If the invert bit is a 0 and any condition is true for which the corresponding bit in the instruction is set, the next sequential instruction is skipped. If the invert bit is a 1, the next instruction is skipped unless at least one condition is true for which the corresponding bit is set.

If the bit which tests the overflow flop is a 1 and the overflow flop is set, the flop is reset after execution of the instruction. The No Operation Instruction is formed by testing no conditions. The following instructions are the only commands documented, although all combinations function as described. The op codes listed below are the 12-bit instruction words read as four octal digits.

NOP (No Operation). --

Op Code: 0200
Execution time: 3.39 μs
Description: No operation

SMI (Skip if Accumulator is Minus). --

Op Code: 0201
Execution time: 3.39 μs
Description: This instruction skips if bit 12 of the accumulator is a 1.

SZE (Skip if Accumulator is Zero). --

Op Code: 0202
Execution time: 3.39 μs
Description: This instruction skips if the accumulator contains a 0 in each bit.

SMZ (Skip If Minus or Zero). --

Op Code: 0203
Execution time: 3.39 μs
Description: This instruction skips if the accumulator is either negative or zero.

SOV (Skip If the Overflow Flop Is Set). --

Op Code: 0204
Execution time: 3.39 μs
Description: This instruction skips if the overflow flop is set and also resets the overflow flop if set.

SKP (Skip Unconditionally). --

Op Code: 0300
Execution time: 3.39 μs
Description: Skip next instruction.

SPL (Skip If Accumulator Is Plus). --

Op Code: 0301
Execution time: 3.39 μs
Description: This instruction skips if bit 12 of the accumulator is 0.

SNZ (Skip If Accumulator Is Non-Zero). --

Op Code: 0302
Execution time: 3.39 μs
Description: This instruction skips if there is a 1 in any accumulator bit position.

SPN (Skip If Positive And Non-Zero). --

Op Code: 0303
Execution time: 3.39 μs
Description: This instruction skips if the accumulator is both positive and non-zero.

SNO (Skip On No Overflow). --

Op Code: 0304
Execution time: 3.39 μs
Description: This instruction skips if the overflow flop is reset and resets the overflow flop if set.

Generic Instructions

The entire instruction word is op code and the 12-bit word is read as four octal digits in the instructions below.

HLT (Halt). --

Op Code: 0000
Execution time: 2.54 μs + wait until restart
Description: The machine is halted in a non-interruptable condition until manually restarted.

OCA (One's Complement Accumulator). --

Op Code: 0003
Execution time: 7.63 μs
Description: The logical state of each bit of the accumulator is inverted.

TCA (Two's Complement Accumulator). --

Op Code: 0005
Execution time: 7.63 μs
Description: The accumulator is 1's complemented and then incremented.

STL (Stall On Line). --

Op Code: 0021
Execution time: 3.39 μs + wait
Description: If the KSTAL- line is positive when this instruction is executed, the machine stalis. When the KSTAL- line goes to ground, the machine starts at the next instruction. Execution of the next instruction starts within 450 to 900 ns of the ground transition of KSTAL-. If the instruction is executed when the line is ground, the instruction is treated as a NOP.

TBA (Transfer Bank Register to Accumulator). --

Op Code: 0022
Execution time: 3.39 μs
Description: The entire accumulator is cleared, the contents of the bank register are loaded into the accumulator bit 1 and bit 13 of the P register is loaded into the bank register. This instruction operates with the 8K memory option and is treated as an CRA in 4K. If TBA is in the last location in a bank, the accumulator bit 1 is set to the next bank.

ITS (Interrupt Save). --

Op Code: 0024

Execution time: 3.39 μs

Description: The contents of P13 and the bank register are copied into the Z register upon interrupt in the 8K version. This instruction copies the Z register into the accumulator such that the bank register is copied into bit 1 and P13 into bit 3. The entire accumulator is cleared before the Z register is copied. This instruction is treated as a CRA in the 4K version.

TOA (Transfer Overflow To Accumulator). --

Op Code: 0030

Execution time: 3.39 μs

Description: This instruction clears the accumulator, transfers the overflow flop to bit 1, and then clears the overflow flop if it was set.

TAB (Transfer Accumulator to Bank Register). --

Op Code: 0041

Execution time: 2.54 μs

Description: The contents of accumulator bit 1 are transferred to the bank register. This instruction operates with the 8K memory option and is treated as a NOP in 4K.

ITR (Interrupt Return). --

Op Code: 0042

Execution time: 2.54 μs

Description: The contents of the accumulator are copied into the Z register and the Z register copied into the bank register and P13 on the next indirect jump or jump store instruction. Bit 1 of the accumulator is placed into the bank register and bit 3 goes to P13. The normal transfers of P13 to the bank register and the bank to P13 are inhibited on the indirect jump following this instruction. This instruction is supplied in the 8K version, and is treated as a NOP in 4K.

ENB (Enable Interrupts). --

Op Code: 0044

Execution time: 2.54 μs

Description: Interrupts are permitted after the execution of the next sequential instruction.

INH (Inhibit Interrupts). --

Op Code: 0050
Execution time: 2.54 μs
Description: No interrupts are permitted after the execution of this instruction.

CRA (Clear Accumulator). --

Op Code: 0060
Execution time: 2.54 μs
Description: The accumulator is reset to all 0's.

INTERRUPTS

Interruptible Conditions

An interrupt is recognized by the machine at the end of the execution of an instruction if interrupts are enabled. Interrupts are also contingent on the following:

a. A halt due to the execution of an HLT instruction is not interruptible.

b. A stall after the execution of the STL instruction and before the KSTAL- line goes to ground is not interruptible.

c. The direct data channel has priority over the interrupt. A DDC request is honored before an interrupt request.

d. An interrupt disables further interrupts until an ENB instruction is executed.

e. Master clear disables interrupts until an ENB is executed.

f. The machine must be running to be interruptible.

Response to an Interrupt

When an interrupt occurs, the machine does the following, instead of executing the next instruction as specified by the P register:

a. Inhibit further interrupts.

b. P13 and the bank register are placed in the Z register for eventual storage (8K only).

c. The bank register and P13 are cleared (8K only).

d. The instruction in location 00002 is executed.

Programming an Interrupt

4K Memory. -- The instruction in location 00002 must be a JST or indirect JST. The JST saves the address of the next sequential instruction in the interrupted program. Once entered, the interrupt subroutine may determine the interrupt source by polling the peripheral devices, save the A register and overflow flop, and then execute the procedure associated with the interrupting device. If it is desirable to have the interrupt routine itself be

interruptible for higher priority requests, the I/O devices may be remasked by use of the SMK instruction and interrupts enabled. It is then necessary to inhibit interrupts at the end of the interrupt subroutine in order to restore the original I/O mask. In either case (interruptible or non-interruptible subroutines), the method of exiting is to restore the overflow flop and the A register, enable interrupts, and finally do an indirect jump to restore the program counter. The indirect jump always takes place because the ENB does not allow a new interrupt until after the execution of the next instruction.

Note that, if the original interrupt had occurred just after the execution of a JMP or JST command, the return takes place to the location specified by the jump command.

4K Or 8K Memory. -- The following procedure permits the saving of the bank register and P13. This procedure may be used in a machine with 4K memory because the instructions which are included only in the 8K version are treated as NOPs or CRAs in 4K.

As in the procedure described for the 4K memory, the instruction in location 00002 must be a JST or indirect JST. After polling the I/O devices, the A register and overflow flop are saved. An ITS (interrupt save) instruction may be executed to transfer the Z register to the A register which may then be stored. The Z register saved the contents of the bank register and P13 when the interrupt request was honored. As in the 4K procedure, the I/O may be remasked, depending on the desirability of having an interruptible subroutine.

When the interrupt subroutine has been executed and the I/O mask restored (if necessary) with interrupts inhibited, the Z register is loaded with the previously saved contents of P13 and the bank register by executing an ITR instruction. This instruction also prepares for a modified action on the next indirect jump. After the overflow flop and the A register have been restored and interrupts enabled, the return is made by an indirect jump. The normal action of the indirect jump in the 8K version is to transfer the bank register to P13 and vice versa; however, due to the action of the ITR instruction, such transfers are inhibited and the contents of the Z register are loaded into the bank register and P13, restoring full machine context.

As in the 4K version, an interrupt just after a JMP or JST causes the eventual return to the location specified by that jump command.

MEMORY ORGANIZATION

Extended Addressing (8K Memory)

A program written to run in the 4K version of the machine runs in either bank of an 8K machine with the factors listed below to be considered. Since a direct memory reference instruction with the sector bit set references the sector containing the instruction, no action out of the ordinary takes place. When a direct memory reference instruction references a primary sector, the sector referenced is the first sector of the bank in which the instruction is contained. The factors to be considered have to do with indirect addressing and interrupts and are as follows:

a. Each pointer or indirect word is prefixed by the bank register as the most significant bit of the pointer. If the bank register is made equal to P13, each pointer references the bank in which the instruction is contained, just as in a 4K machine.

b. The interrupt location is in the lower bank and the effective address of the JST or indirect JST in the interrupt location is within the lower bank. If the interrupt handler is to be in the upper bank, a small routine is needed to jump from the lower bank to the upper. The interrupt return can be from either bank.

The instructions used to communicate between banks are TAB (transfer A to bank register) and TBA (transfer bank to A register). These instructions are treated as a NOP and CRA in 4K. The TBA and TAB instructions can store and load the bank register into and out of bit 1 of the accumulator.

To reference data in the upper bank, the bank bit is set to a 1 and an indirect reference is made. The data is acquired from the given address in the upper bank regardless of the location of the instruction. The contents of the bank register are unaltered. To move from one bank to another, the bank bit is established and an indirect JMP or JST is executed. In this case, the address of the next instruction is constructed from the bank bit and the remainder of the effective address. The previous contents of P13 are loaded into the bank register. In this manner, the full 13 bits of the P register are saved on JST commands.

Memory Wrap

In the 4K machine, the memory location after 7777_8 is 0000_8. The P register, pointers, and skip instructions increment in this fashion. A JST to 7777_8 causes the next instruction to be taken from 0000_8 and likewise a JST in 7777_8 stores 0000_8 as the return address.

In the 8K machine, the memory location after 07777_8 is 10000_8. A JST to location 07777_8 (effective address) causes the instruction in 10000_8 to be executed next. A JST from 07777_8 causes 0000_8 to be stored and the bank register to be set.

The P register increments from 17777_8 to 00000_8. JST instructions to 17777_8 cause the instruction in 00000_8 to be executed and JST's from 17777_8 cause 0000_8 to be stored and resets the bank bit.

Since pointer overflows do not alter the contents of the bank register, a pointer increments to the bank boundary and resets to the beginning of the bank, i.e., 07777_8 to 0000_8 and 17777_8 to 10000_8.

DIRECT DATA CHANNEL

The DDC option provides for I/O rates of up to a word every 3.39 μs. Up to two bidirectional subchannels may be used simultaneously. The subchannels are in the peripheral device adapters while the DDC option is in the mainframe.

The DDC is entirely controlled by the peripheral device. The location of the data in memory and the number of words to be transferred are contained in hardware registers in the subchannels. The direction of transfer and the time at which the transfers take place are under the control of the subchannel. There are no instructions in the mainframe

pertaining to this option; however, there may be I/O instructions to load the address and range counters and to initiate the transfer via the peripheral device.

DDC transfers are not constrained to one bank or the other and can increment across either bank boundary. A DDC request has higher priority than an interrupt request. The DDC channel transfers only at the completion of the execution of the present instruction and is not inhibitable. DDC transfers do not occur while the machine is not running, while it is halted due to a HLT instruction, or while the machine is stalled due to an STL instruction. If the DDC requests occur fast enough, the execution of the program can be completely locked out.

POWER FAILURE AND INITIALIZATION

Power Failure

The standard machine is designed to preserve the contents of all core locations if the power fails or is turned off while the machine is running, stopped, halted, or in the stall mode. The register contents are lost. If power is restored, the machine comes on in the stopped mode with the P register and critical control F/F's cleared.

Power Failure Interrupt

As an option, a power failure interrupt option may be added which causes an interrupt when the power fails. The time between the interrupt and when memory shutdown occurs for lack of power is a minimum of 1 millisecond. The interrupt also occurs on power turn-off. The machine comes up cleared and not running when power returns. The option is an omni-block in the peripheral option area.

SECTION III
INPUT/OUTPUT CHARACTERISTICS

The H112 Controller normally communicates with an external system and I/O devices by way of the programmed I/O bus where data transfers are under program control. Load mode transfers also occur over the I/O bus but are entirely under the control of the controller hardware with no program in execution. DDC transfers occur between the channel in the controller and subchannels in the peripheral devices over the DDC bus under the control of the subchannel (although the transfer may originate with a command from the program to the peripheral device over the I/O bus).

The control panel exercises a macro control over the machine and serves to reset the machine, start the program, etc. Certain functions are also available to the system via the machine control interface whether or not the panel is included in the machine.

See H112 Interface and Installation Manual for further details.

PROGRAM I/O BUS INTERFACE REQUIREMENTS

The program I/O bus consists of 30 signals on a μ-PAC connector as shown in Table 3-1. All levels are μ-PAC (0 to +6 volts) levels. The controller drives, on each output line, 16 receiver loads, each consisting of a standard μ-PAC gate input in parallel with 6.8K to +6V and a diode with anode to ground. Each input line to the controller appears as one receiver load; however, peripheral devices load the input lines in parallel. Each line may appear as up to 17 loads to a particular device including the device's own load and the controller's input load. The I/O bus timings are guaranteed only when driven and received with circuits such as on the CS-517 PAC, which contains six receiver circuits and six driver circuits.

All signals into and out of the controller are true (represent logical 1) when at ground and are false (represent 0) when at +6V. Signal timings are guaranteed to be correct at the input or output of a peripheral device when the device is driven by a standard cable whose total length is less than 25 feet.

Each peripheral device is to load the bus with no more than one load and to drive the bus with only one driver.

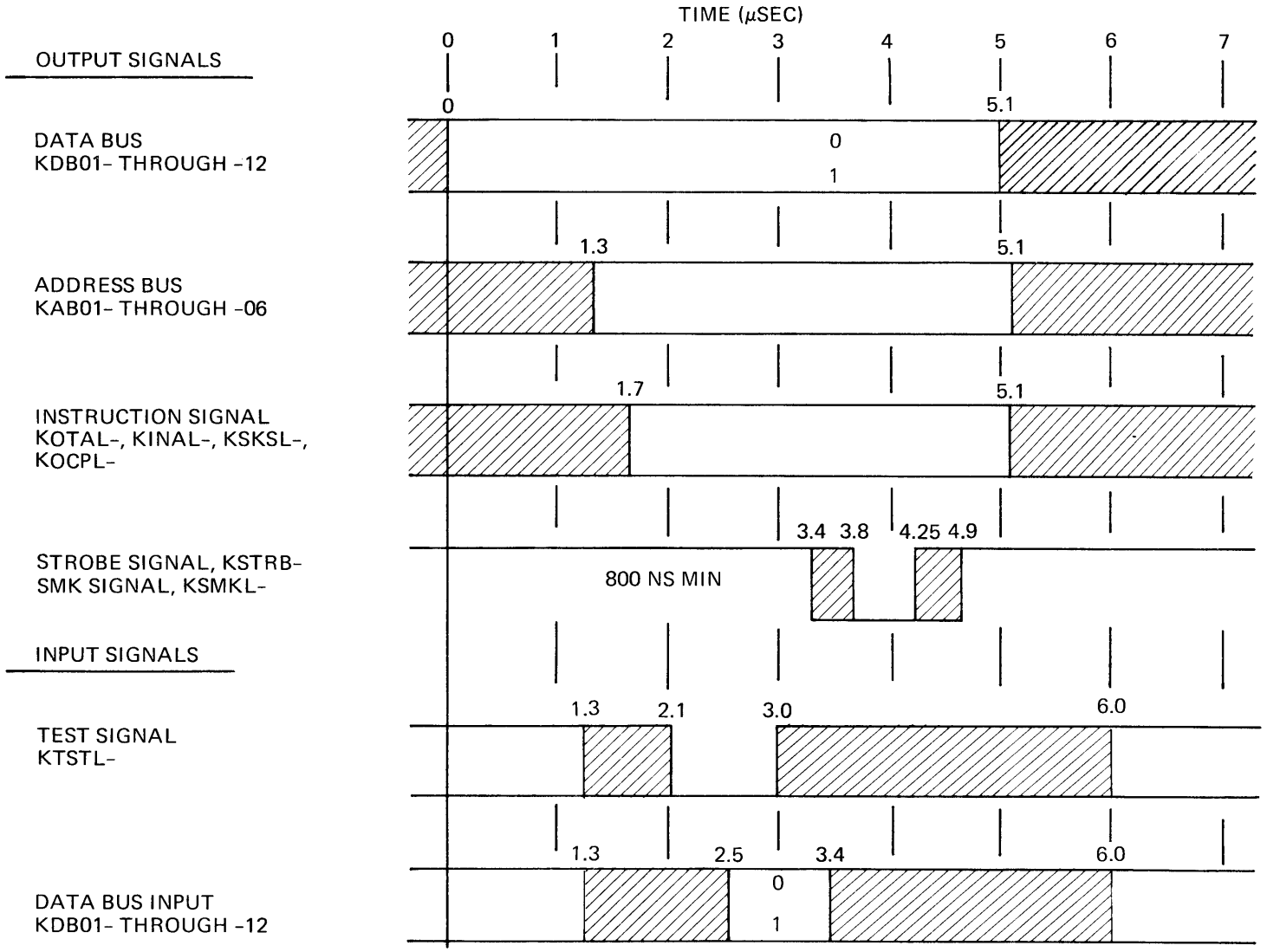
Signal edges are guaranteed to fall within the specified time but edges are not unambiguous. They may cross gate threshold more than once during transition times.

SIGNAL DESCRIPTIONS

A description of the I/O signals is provided in the paragraphs which follow. Signal timing for OTA, INA, OCP, SKS, and SMK is shown in Figure 3-1.

Table 3-1.
I/O Bus Signals

Pin	Signal Name	Function
1	KDB01-	(Data Bus - 12 lines)
2	KDB02-	
3	KDB03-	
4	KDB04-	
5	KDB05-	
6	KDB06-	
7	KDB07-	
8	KDB08-	
9	KDB09-	
10	KDB10-	
11	KDB11-	
12	KDB12-	
13	KAB01-	(Address Bus - 6 lines)
14	KAB02-	
15	KAB03-	
16	KAB04-	
17	KAB05-	
18	KAB06-	
19	KOTAL-	OTA (Output Transfer From Accumulator) Instruction
20	KINAL-	INA (Input Transfer to Accumulator) Instruction
21	KOCPL-	OCP (Output Control Pulse) Instruction
22	KSKSL-	SKS (Skip If Set) Instruction
23	KSMKL-	SMK (Set Mask) Instruction
24	KSTRB-	Strobe Signal
25	KINTL-	Interrupt Signal
26	KLOAD-	Load Signal
27	KTSTL-	Test Line
28	KSTAL-	Stall Line
29		(Spare)
30	KXCLR-	Master Clear
31	KGND-	Ground
32	KPWFL-	Power Failure



NOTE:

LOGICAL 0 = +6V
LOGICAL 1 = 0V

Figure 3-1. Signal Timing for I/O Instruction Execution

KDB01- through KDB12-

These 12 output signals from the controller represent the contents of the accumulator during execution of OTA and OCP commands. During INA's, the addressed peripheral device places data on this bus for eventual gating into the accumulator. Each device must permit these lines to be false except when data is being transferred to the controller in response to an INA.

KOTAL-, KINAL-, KOCPL-, KSKSL-

Each output line indicates that the execution of the corresponding instruction is in progress. The OCP line pulses during execution of SMK.

KAB01- through KAB06-

These six output signals are the least significant bits of the W register and present the address portion of the I/O instruction.

KTSTL-

This input signal is made true by the peripheral device in response to address and instruction signals if the device is ready for a data transfer. This signal is used during an INA or OTA instruction or if a skip is to occur during an SKS instruction. The line serves no function during OCP. Each device must permit this line to be false except during the period when it is addressed.

KSTRB-

This output signal indicates that a true test signal has been recognized during either an INA, OTA, or SKS signal, that the next instruction will be skipped, and that a data transfer, if any, will take place.

During OCP and OTA commands, this signal defines the time when the address and data lines are stable. This signal is also active, but redundant during execution of an SMK.

KINTL-

Interrupt signal is an input signal to the controller. This line is made true to interrupt the program and held true until the interrupt is recognized.

KSMKL-

This output signal is true during the execution of either an SMK0 or SMK1 instruction. The peripheral device uses this line with KAB01± and the appropriate data bus bit to set or reset the mask F/F.

KXCLR-

This line is true during the operation of the master clear pushbutton on the panel, activation of the remote clear line, and during power-on and during power-off conditions.

KLOAD-

This line is true when the load function is activated.

KSTAL-

A true pulse which must be greater than 500 ns on this line restarts the controller after the execution of an STL instruction. If this line is true during execution of the STL instruction the machine does not stop.

KPWFL-

A true level on this output line indicates that operation of the controller is terminated because of a power failure or turn-off operation. The line goes true at least 1 millisecond before the memory supply is turned off. This signal is supplied only with the power failure interrupt option.

INTERRUPTS

The interrupt facility enables an external device to change the normal program sequence with low response latency. An interrupt request occurs when one of the devices on the common interrupt line (KINTL) grounds the line. If interrupts have been enabled by the controller program in progress, the request is honored on completion of the current instruction and control is transferred to the dedicated location 00002. Additional interrupts will be inhibited until the execution of an ENB instruction. The device which causes the interrupt maintains the interrupt line at ground until the interrupt is acknowledged. This may be done by data transfer or by separate instruction depending on the devices' purpose in causing the interrupt.

The ability to selectively inhibit or enable interrupts on a device basis is desirable in some real time systems. To provide this facility each device has a mask flip-flop in addition to the interrupt flip-flop. The mask flip-flop is set and reset by the set mask instruction (SMK00 or SMK01). Interrupts are permitted from a particular device only when the mask flip-flop is set and are inhibited when reset.

DIRECT DATA CHANNEL

The DDC allows an external device to interface directly with core memory. Control of DDC transfers are entirely under the control of the external device; the H112 central processor suspends all normal operations when a DDC cycle occurs.

The DDC mode is entered between normal program instructions. When the external device requests the DDC mode, the controller completes the present instruction. The

contents of all registers are maintained throughout the DDC operation and the original program data is not lost. DDC operation is inhibited when the machine is halted, stalled or stopped. Because the DDC mode is a high priority mode, the normal program execution is delayed. Normal program operation resumes upon completion of the DDC operation.

LOAD MODE

Load mode transfers load 3-bit data into the H112 controller for storage in core memory. Transfers are via the programmed I/O bus under H112 hardware control. Four consecutive octal characters are stored at each 12-bit memory location.

The load mode may be initiated by either the control panel LOAD and START switches or via machine control interface. The KLOAD signal line initiates the load mode on transition to true state (ground) and remains there during the transfer. When the stop character is detected (KDB07 true), KLOAD becomes false (high) and the load mode terminates. The controller accepts one octal digit upon transition of KDB09 from false (+6) to the true (ground) state.

SECTION IV
SAP-12 ASSEMBLER

SYSTEM ASSEMBLY PROGRAM

Assembly Language

SAP-12 is the system assembly program used with the H112 Controller. It is a programming aid which translates the symbolic (source) program into the machine language (code) which is compatible with the hardware. SAP-12 provides symbolic programming while maintaining the characteristics, flexibility, speed, and conciseness of machine language programming, and permits the assignment of symbolic addresses to storage locations.

Assembly Procedure

The SAP-12 assembly program paper tape is first loaded into a Honeywell H316 or DDP-516 General Purpose computer via a paper tape reader. After the SAP-12 program has been loaded, the source program may be read into the computer via paper tape, magnetic tape or cards.

The source program is read twice under the control of the SAP-12 program. During the first pass, a table of symbols is established. During the second pass, object program assembly takes place and an assembly listing printout is generated. This is followed by a post processor cross-reference listing printout.

A description of the source program line processing follows. (Refer to Figure 4-1.) Program assembly takes place during pass 2. A line is first read from the tape and stored in a buffer (part of memory). SAP-12 calls the subroutines used for reading and storing one line of input. The line is separated into its subfields and the operation menemonic is examined. The nature of the indicated operation (normal or pseudo) determines the subroutines used to process the operation field. For normal operations, SAP-12 determines the specified machine operation code and places this in the appropriate location of the instruction word being assembled. For pseudo operations, analytical routines are used which modify the assembly process, allocate storage, or define data words. The variable field is then processed. Defined data is converted to binary; the SAP-12 symbol table is searched to evaluate symbols; calculations are performed to evaluate expressions. If the operation field specifies a normal machine operation, the resultant value forms the address field of the instruction being assembled.

The assembled object program is punched on an "object tape." This tape is punched in blocks of from 4 to 38 12-bit words. The words are object program words converted to

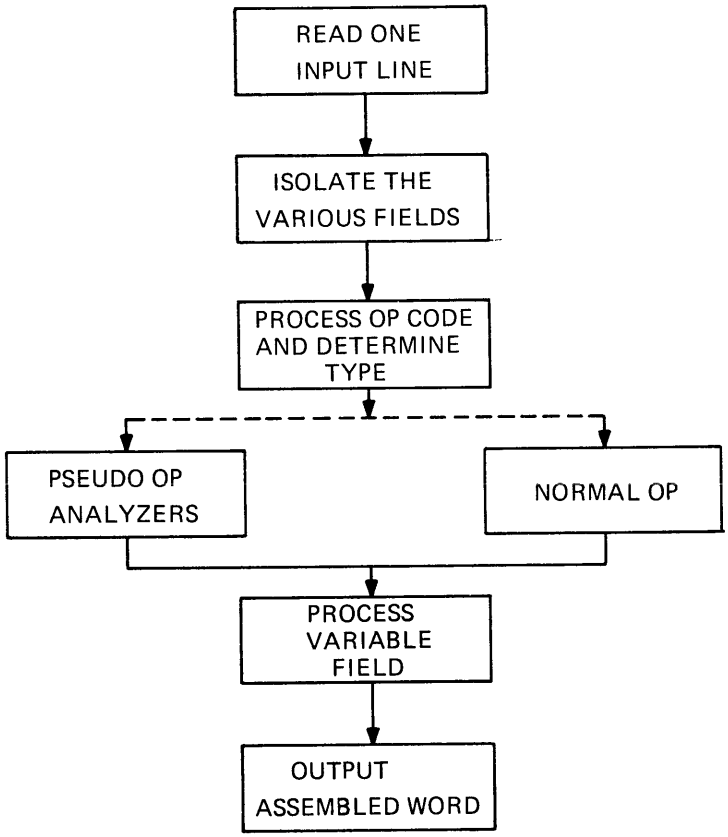


Figure 4-1. Processing of One Line

(printable) ASCII characters. The object tape block formats, types, program printing format and other details are discussed in later paragraphs.

ADDRESSING

The H112 controller memory is divided into banks of 4096 words. Each bank is divided into sectors of 128 words. Each of the seven memory reference instructions (LDA, STA, ADD, ANA, IRS, JMP, JST) has a 7-bit address field, allowing direct reference to words contained in a single sector. The sector bit determines the sector which contains the required word. If the sector bit is 0, the address is in the primary sector; if it is 1, the address is in the sector containing the instruction. A memory reference instruction may, therefore, address 256 words: 128 in the primary sector and 128 in the current sector.

The assembler maintains a base location counter (BLC) in addition to the normal location counter. The BLC is normally set to location 0004 when the assembler is initialized. When the assembler must create an indirect link, it places the full 12-bit address in the word specified by the BLC, creates a memory reference instruction which indirectly references that word, and increments the BLC.

Through the use of the SETB pseudo operation, the programmer should set the BLC to some location in memory. (See example Section 6-3). This is necessary to place the indirect links for a given sector within that sector. If the BLC is incremented across a sector boundary, no further indirect links are created until another SETB pseudo operation is encountered, and any instructions requiring cross-sector references are marked with an error flag. An error flag is also printed if a cross-sector address is required on an indirect memory reference instruction, or if a SETB region is exceeded.

Because the execution of instructions with addresses in another bank requires the programmer to perform certain actions (see TAB, TBA machine instructions), a warning flag is placed on the listing at each instruction requiring a cross-bank address.

LANGUAGE STRUCTURE

Source Program

Programs written in the SAP-12 source language consist of a sequence of symbolic instructions or statements known as source lines. Figure 4-2 shows a typical symbolic instruction written on a SAP-12 source program coding form. The instruction shown represents one source line.

As indicated on the coding form, symbolic instructions consist of four main fields. Each field is variable in length and is delimited by a space. For convenience when key-punching cards, forms are available which provide fixed fields. (See Figure 4-2.) This format also contains an extra field used for entering the card sequence number; however, it is not required as part of a symbolic instruction.

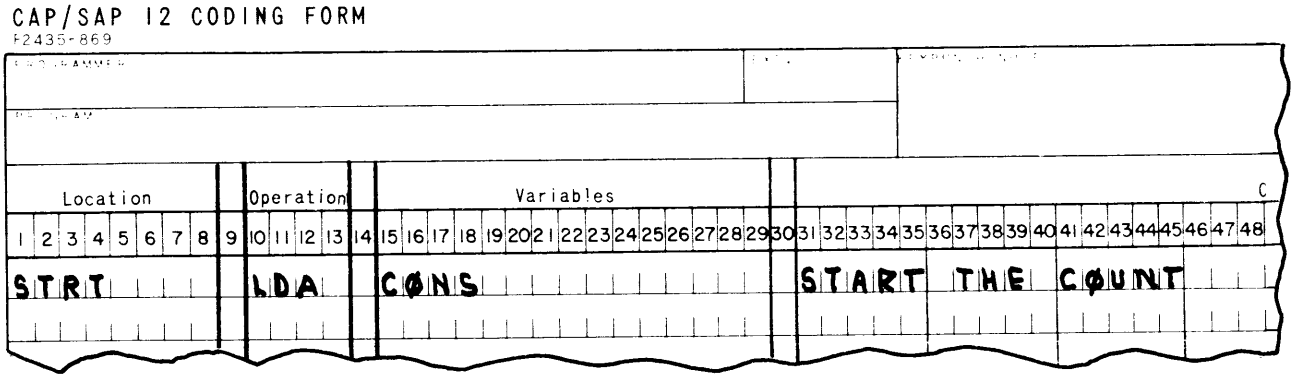


Figure 4-2. Source Program Coding Form

Source Program Format

Each source language statement consists of five fields: line number (optional), location, operation, variable and comment. Each of these fields is delimited by a space character. The five fields must be punched in the above order. A line is terminated at 80 characters when the input is cards or mag tape, and at an X-OFF when the input is paper tape. Carriage returns, line feeds, and rubout characters are ignored on paper tape.

Line Number Field (Not Shown on Figure 4-2). -- The line number field is optional and is allowed only for paper tape. It is provided for compatibility with input prepared for the Honeywell 1648 time sharing system. The line number is a number less than 32768. It must be terminated by exactly one space. The occurrence of more than one space will indicate the absence of the location field.

Location Field. -- The location field may be used to assign a symbolic address or "label" to an instruction so that the instruction can be referred to elsewhere in the program. The symbolic address in the location field must be a legal symbol. SAP-12 assigns memory addresses to the symbolic locations when assembling the object program. If the first character of the location field is an asterisk (*), the entire statement is regarded as comments.

Operation Field. -- The operation field is analogous to the operation-code portion of a machine language instruction. The contents of the operation field may be either a machine language instruction mnemonic, or one of the pseudo-operation menemonics provided in the SAP-12 repertoire. Operation mnemonics are either three or four characters in length.

In addition to specifying an operation, the operation field also specifies that indirect addressing is desired if an asterisk (*) immediately follows the operation-code mnemonic.

Variable Field. -- The variable field is normally used to specify an address or a shift count. When used with a SAP-12 pseudo-operation, the significance of the variable field depends upon the nature of the pseudo-operation.

Comments Field. -- The comments field may be used for any comments the programmer cares to write. This field has no effect on the assembler, but is printed out on the symbolic assembly listing.

Sequence Number Field. -- The sequence number field is actually contained within the comments field and its use is optional. It is used for the assembler sequence check on cards (SEQC) pseudo-operation.

The source line shown on the form in Figure 4-2 is an instruction located at the symbolic location STRT. When executed the instruction will load a constant, located at the symbolic location CONS, into the A register.

Symbols

Symbols generally represent memory addresses and may appear in both the location and the variable field of symbolic statements. The programmer defines a symbol by placing it in the location field of a statement, thus giving the statement a symbolic address. The assembly program keeps track of the location of statement in the source program by stepping a location counter by one for each instruction. When a symbol appears in the location field, it is normally assigned the current value of the location counter. This is not true of some pseudo-ops; see EQU, SET and FORM. The first such occurrence constitutes the definition of the symbol, and any subsequent occurrence in the location field causes an error flag (M) to be printed. If a symbol appears in a variable field and never appears in a location field, it is an undefined symbol and will cause an error flag (U) to be printed. The value of an undefined symbol is zero.

Symbols consist of one to eight characters from among the 36-character set of the letters of the alphabet and the 10 digits. The first character in any symbol must be alphabetic.

Expressions

Expressions appear only in the variable field and may be either simple (composed of a single element) or compound (composed of two or more elements separated by operators).

Elements. -- An element may be:

a. A binary number less than 2^{12} (coded as B'n....n)
b. An octal number less than 10000 (coded as O'n....n)
c. A decimal number less than 4096 (coded as D'n....n)
d. An asterisk (representing the current contents of the location counter)
e. A character pair (coded as C'aa)

f. An address constant (coded as a symbol)

g. An absolute number (coded as n....n; evaluated according to the base established by the previous BASE pseudo operation)

h. A single ASCII character, (coded as A'a and assembled with leading binary zeroes).

Operators. -- An operator may be:

a. Blank (terminates expression)

b. Asterisk (multiply)

c. Slash (divide)

d. Plus (add)

e. Minus (subtract)

f. Parentheses (evaluate expression within parentheses first)

The following is an example of a valid expression:

```
TAB  EQU 2+*-REL/(XX*3)
```

This reads "TAB equals the value of REL divided by the value of XX times 3, subtracted from the current location plus 2."

Strings

A string is valid only in a DATA statement. It is coded as follows:

S'daaa....aaad

a. S' defines the String.

b. d.....d are the delimiter characters which start and end the string.

c. aaa....aaa is the string.

The characters in the string are stored two per word, left-justified. The last word is space-filled if necessary. The delimiter characters may be any valid character.

An example of a string is:

```
S'$WHAT'S UP, DOC ?$
```

The dollar signs delimit the string which may contain any character except the delimiter character.

Source Preparation

The first statement in each program should be a comment line (* in column 1) containing a $ in column 2, and the program name in columns 3-8. This statement will cause a Header block to be punched on the object tape. This statement will also allow the program to be called as a source library tape by a COPY pseudo operation, when source is punched from Pass 1 using pseudo-op PCH.

The last statement in the program must be an END statement.

Paper Tape

The ASR 33/35 requires that each line be punched as follows:

a. one line feed character

b. source statement

c. one carriage return character

d. one X-OFF character

e. one or more rubout characters for ASR-33, two or more rubout characters for ASR-35.

SAP-12 recognizes the X-OFF character as the line terminator; all line feed, carriage return, and rubout characters are ignored.

Holding down the control key and depressing the H key causes the previous character to be deleted. Repeated operation will delete multiple characters. Operating the X key with the control key will cause the entire current line to be deleted.

Cards

Each source statement must start on a new card and must be completed on that card. The statement scan is under the control of the CARD pseudo operation.

Magnetic Tape

SAP-12 accepts unblocked card images on magnetic tape. They are processed in the same manner as cards.

Asterisk Conventions

The conventions for use of the asterisk are summarized below.

a. An asterisk in the first character of the location field: treat the entire statement as remarks.

b. An asterisk appended to instruction mnemonic: set the indirect address flag.

c. An asterisk as an element in an expression: current value of the location counter.

d. A single asterisk as an operator in an expression: multiply.

e. An asterisk - asterisk (*-*) or a double asterisk (**) as a symbolic address: assemble an address of zero (address is modified by another instruction).

f. A triple asterisk (***) as an operation code: assemble the instruction as a memory reference with an op code of 0 (op-code is modified by another instruction).

ASSEMBLY LISTING

The assembler produces a side-by-side listing of the assembled object code and the source input (see Figure 4-3). The format of this listing is as follows:

```
0169  00500  1325 00525          STA   LMY          RESTORE SHIFTED MULTIPLIER
0170  00501  0726 00526          LDA   LP           PICK UP H,O,W, OR PRODUCT
0171  00502  0304                SNO                IF MULTIPLIER BIT WAS ONE, ADD IN
0172  00503  2324 00524          ADD   LMD          MULTIPLICAND,  NOTE TA
0173  00504  2324 00524          ADD   LMD          MULTIPLICAND,  NOTE THAT THE OVF
0174                    *                           HAS BEEN RESET AND OVERFLOW CANNOT
0175                    *                           OCCUR DURING THE ADD
0176  00505  0101                LGR   1            SHIFT H,O,W, OF PRODUCT RIGHT ONE
0177  00506  1326 00526          STA   LP           AND RESTORE IT
0178  00507  0030                TOA                PICK UP THE BIT SHIFTED OFF AND
0179  00510  0141                RAR   1            MAKE IT M,S,B, OVF IS AGAIN RESET
0180  00511  2327 00527          ADD   LP+1         ADD IN L,O,W, OF PRODUCT
0181  00512  0101                LGR   1            SHIFT NEW L,O,W, RIGHT ONE AND
0182  00513  1327 00527          STA   LP+1         RESTORE IT.  THE SHIFT NEVER SETS OVF
0183  00514  3327 00527          IRS   MCNT         INCREMENT ITERATION COUNTER
0184  00515  1676 00476          JMP   MPLP         RECYCLE IF NOT DONE
0185  00516  5651 00451          JMP*  MPYS         RETURN
0186                    * IF BOTH OPERANDS WERE LNN, RETURN WITH ZERO PRODUCT, ZERO ACC,
0187                    * AND WITH OVF SET.
0188  00517  0060      MLNN      CRA                CLEAR
0189  00520  0003                OCA                ONE'S COMPLIMENT
0190  00521  0101                LGR   1            SET OVF
0191  00522  0060                CRA                CLEAR ACC
0192  00523  5651 00451          JMP*  MPYS         RETURN
0193  00524           LMD       BSS   1
0194  00525           LMY       BSS   1
0195  00526           LP        BSS   1
0196  00527           MCNT      BSS   1
0197  00530  7765      M0013     DATA  -0'0013
```

HONEYWELL

COMPUTER CONTROL DIVISION

Figure 4-3. Sample Assembly Listing

Page Heading

Each page is headed by a line consisting of the first comment card encountered.

Page Title

Each page may, under control of a pseudo operation (TTL), have a title line between the heading line and the body of the page.

Body of Page

The assembler produces one line for each object word produced by the assembler, plus lines created by comment and space cards. Multiple lines are generated by certain pseudo operations and are printed under control of the GENR pseudo operation.

Typical Listed Line

The typical listed line consists of six fields:

a. Line number (decimal)

b. Error flags (one to four alphabetic characters)

c. Assembled location (5-digit octal number)

d. Assembled H112 word (4-digit octal number)

e. Absolute address (5-digit octal number)

f. Reproduced source input (may be controlled by TABS)

Error Flag Definitions

M Multiply defined symbol; the same symbol appears in the location field of two or more statements. Every reference to this symbol is flagged. Because the symbol table uses a hash-coding technique, it is possible for two different symbols to look the same to the assembler. This will result in a spurious "M" flag. To correct this condition, merely change one of the two symbols.

A The pass 1 and pass 2 values assigned to this tag are not equal. Example:

```
LOC     BSS     VAL
TAG     LDA     P5
VAL     EQU     10
```

"TAG" line will be flagged with A, because "VAL" is not defined in pass 1 (=0) while it is 10 in pass 2.

R Core wrap-around; instruction addresses a location beyond the top of memory. High order bits of address will be masked to force core-wrap; i.e., the address will be calculated modulo the memory size.

S The address of a memory reference instruction is in neither the current sector nor a primary sector. An indirect link could not be made.

X There is no room in the current SETB region to store the indirect link required by this statement; link not generated.

W The most recent SETB pseudo operation specified a location neither in the current sector nor a primary sector; therefore, the indirect link required by this statement cannot be generated.

B The address specified by this statement is not in the bank in which this statement resides.

F A format error has been found in a FORM or TABS pseudo operation.

U A symbol specified in the variable field of this statement is undefined.

I The operation code of this statement is invalid.

L The location field of this statement is blank; a symbol is required.

P A pushdown list is exhausted. Possible causes are too many nested parentheses in this expression or SETB list overflowed.

T A number has been truncated in order to fit it into its field as defined in a FORM pseudo operation.

E An expression is invalid. The field is set to 0.

Fatal Errors

Three fatal errors will abort the assembly;

PTBL full! — symbol pointer table is full.

STBL full! — symbol table itself is full.

SRCL ERROR — assembler self-check error.

OBJECT TAPE

The object paper tape is punched in blocks consisting of from 4 to 38 words. Each 12-bit word is punched as two 6-bit characters. Each 6-bit character is added to 240_8 to produce an 8-bit ASCII character in the range 240_8 - 337_8. All the characters in this range are printable. The tape feed character is 001_8 which is not printable. This tape format is compatible with tape produced by the Honeywell Series 16 time sharing system.

Tape Format

The paper tape format for the object tape is shown in Figure 4-4. As shown, one vertical column of holes is a frame. One frame corresponds to one typed character or typing function. The holes which make up the frame are numbered as channels 1 through 8. Sprocket holes are used for tape drive.

Each channel may be regarded as a binary bit. A hole in the tape (filled in circle - see Figure 4-5) represents a logical 1. The absence of a punched hole (plain circle) represents a logical 0. The binary bits are grouped, as shown in Figure 4-5, to form a 3-bit octal number. Note the examples shown on the figure for a line feed and carriage return. They are read as 212_8 and 215_8, respectively. Although these two characters do not actually print, they do control the format of the printed output.

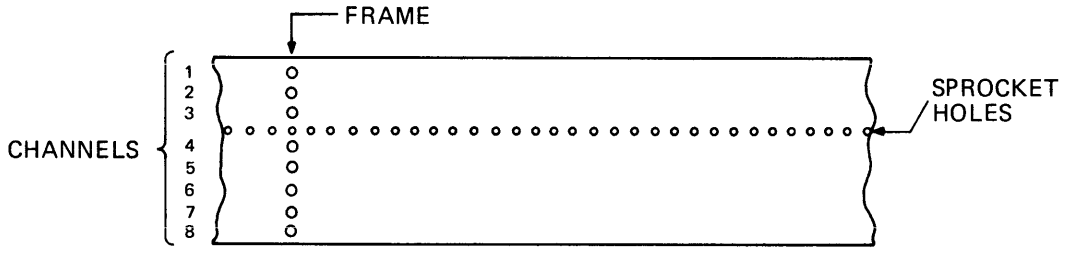


Figure 4-4. Paper Tape Format

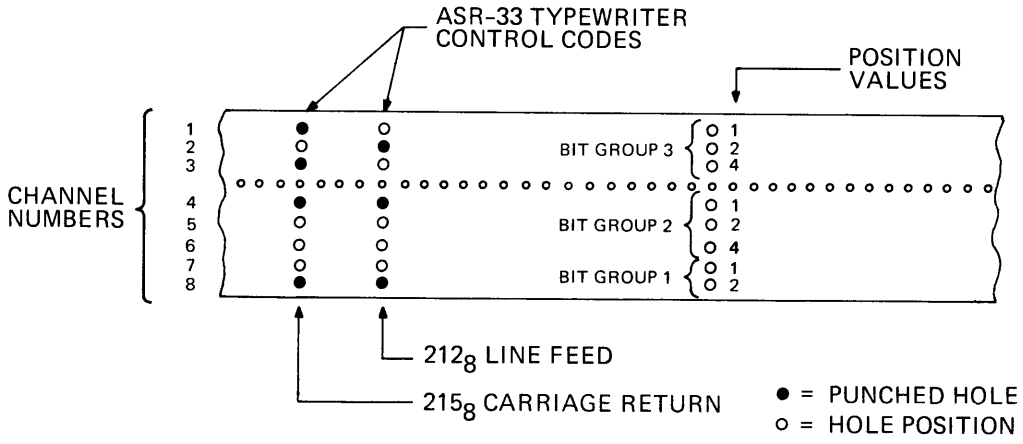


Figure 4-5. ASCII Punched Tape Format

Typical Block Format

Each block has the following format.

Word 1: left character: block type (0-4) converted to ASCII (240_8-244_8).
right character: data block length in 12-bit words (0-34) converted to ASCII (240_8-302_8); the data block length is the number of words after word 1 and before the checksum.

Words 2-(n-3):
data block (if any); each 12-bit word is converted to two ASCII characters.

Word n-2: checksum of words 1 through n-3; converted to two printable ASCII characters (240_8-337_8).

Word n-1: left character: ASCII carriage return (215_8).
right character: ASCII line feed (212_8).

Word n: left character: X-OFF (223_8).
right character: tape feed (001_8).

Block Types

Block type 0 - header (Figure 4-6)

Word 1: left character: block type (=0)
right character: data block length (=3)

Words 2-4: 6 character program name

Word 5: checksum of words 1-4

Word 6: carriage return and line feed

Word 7: X-OFF and tape feed

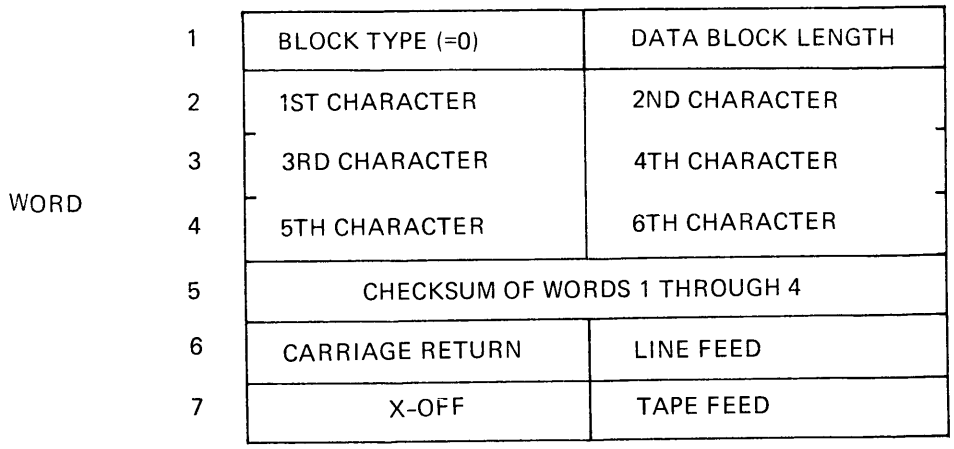


Figure 4-6. Block Type Zero Format

Block Type 1 - Data Block (Figure 4-7). --

Word 1: left character: block type (=1)
right character: data block length (=3-34)

Word 2-3: 13-bit location of first word to be loaded (right-justified in a 24-bit double word).

Word 4 through n-3: data words to be loaded (1-32 words)

Word n-2: checksum of words 1 through n-3

Word n-1: carriage return and line feed

Word n: X-OFF and tape feed

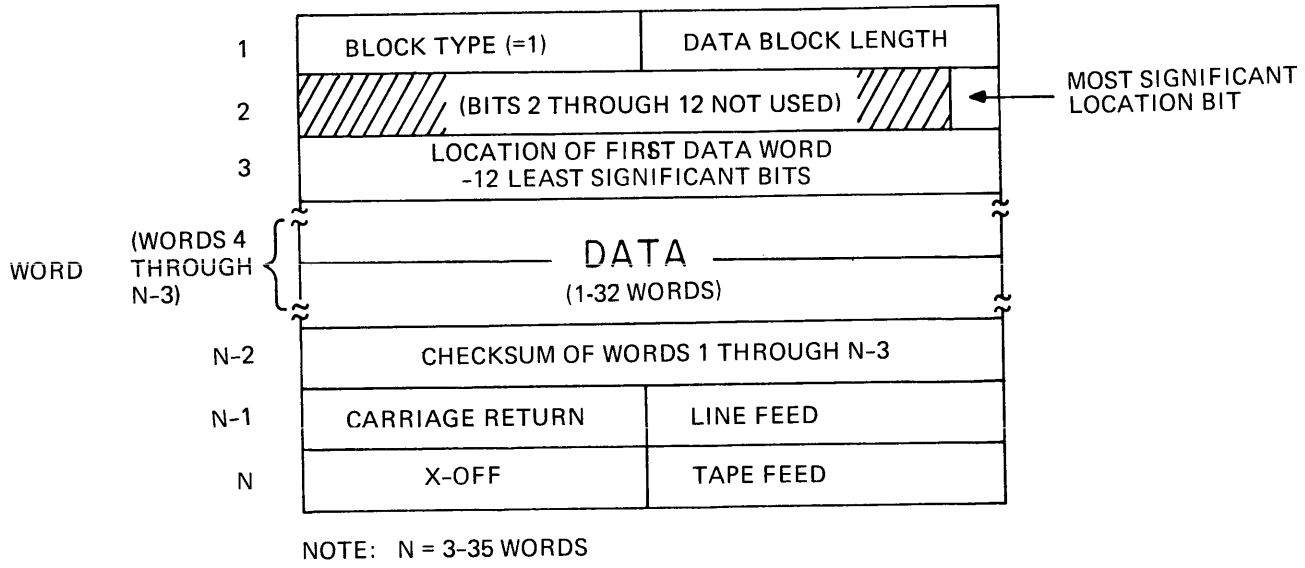


Figure 4-7. Block Type One Format

Block Type 2 - Fill Block (Figure 4-8). --

Word 1: left character: block type (=2)
right character: data block length (=4)

Words 2-3: 13-bit location of first word to be filled

Word 4: number of words to be filled

Word 5: data word; all words are filled with this value

Word 6: checksum of words 1-5

Word 7: carriage return and line feed

Word 8: X-OFF and tape feed

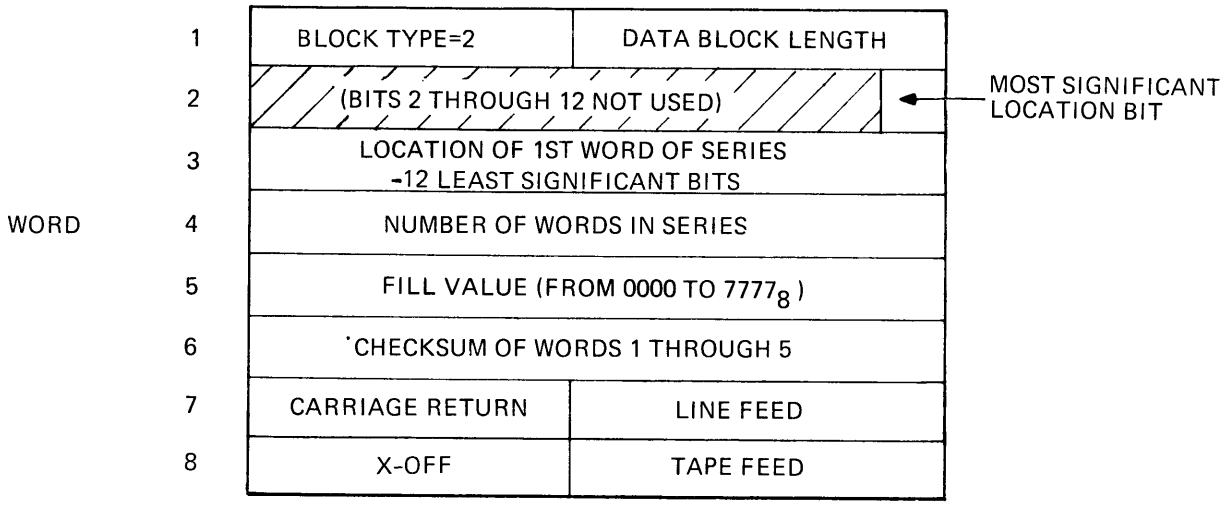


Figure 4-8. Block Type Two Format

Block Type 3 - Transfer Block (Figure 4-9). --

Word 1: left character: block type (=3)
right character: data block length (=2)

Word 2-3: 13-bit location at which to start program execution

Word 4: checksum of words 1-3

Word 5: carriage return and line feed

Word 6: X-OFF and tape feed

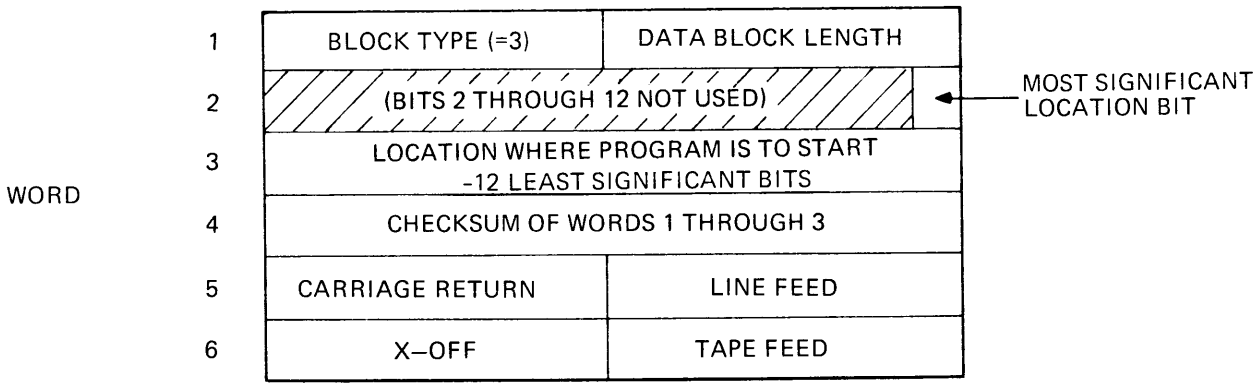


Figure 4-9. Block Type Three Format

Block Type 4 - End-of-File Block (Figure 4-10). --

Word 1: left character: block type (=4)
right character: data block length (=0)

Word 2: checksum of word 1

Word 3: carriage return and line feed

Word 4: X-OFF and tape feed

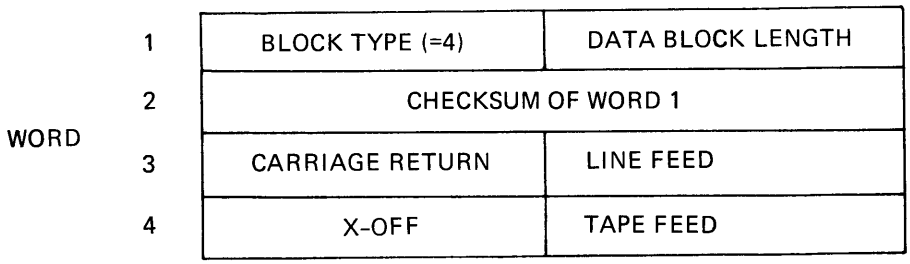


Figure 4-10. Block Type Four Format

POST PROCESSOR LISTING OUTPUT

The post processor listing output is printed immediately after the assembly listing if requested by the POST pseudo operation. It is a list of all symbols defined during program assembly, their locations and all references to the symbols. The format of the list is described below and is shown in Figure 4-11. Each page is headed with the same line as the assembly listing. There is one line containing the following fields for each symbol defined:

a. Sector which contains the symbol.

b. Location of the symbol, (or value, in the case of EQU and SET).

c. Symbol.

d. Reference List: the location of each reference to the symbol.

NOTE:

POST increases the size of the symbol table by a factor of about 3.

```
                    TEST OF SAP-12 NUMBER 1                                    SECTOR: 47    PAGE:   12
                  C R O S S    R E F E R E N C E    T A B L E

00  00001  AA          00046, 00211, 00214, 00216, 00216, 00235, 00425, 01020,
40  10001  B2          01026,
02  00426  BAD
00  00000  BB
01  00204  BEGIN       00205, 00205, 01017,
00  00054  BK
00  00004  BLOCK       00054, 00054, -
00  00062  BSET        00062,
00  00004  CC          00016, 00046, 00236,
01  00220  CSET        00200,
00  00016  DATA1
00  00012  DD          00001, 00001, 00002, 00002, 00033, 00040, 00425,
02  00432  DSET        00400,
00  00001  DTESTA      00207,
00  00002  DTESTB
00  00014  FF          00016, 00236,
00  00011  FILLBLK
01  00237  H1234567
01  00233  HELP        00403, 00406, 01016,
04  01021  HIBLK       01015,
02  00427  LABEL       01013,
02  00430  LABEL2
02  00431  LABEL3
04  01001  LOWLBL      01014,
02  00445  MULDEF      00235,
02  00402  NEXTSECT    00241, 00401,
02  00407  OVER11      00410, 00411, 00412, 00413, 00414, 00415, 00416, 00417, 00420, 00424, 00425, 00425, 11610,
00  00017  Q
00  00033  Q1
00  00046  Q3
00  00055  Q4
02  00410  SDF
01  00243  THISSECT    00242, 00400, 10000,
01  00234  WHY         00404, 00427, 00430, 01015,
40  10003  XBANK       10002,
47  11610  XYZ         01000, 10002,
```

Figure 4-11. Sample Post Processor Listing

SECTION V
BASIC PROGRAMMING

This section contains information for coding H112 machine instructions and pseudo-operations used in programming the H112 with the SAP-12 assembler language.

MACHINE INSTRUCTIONS

There are five general types of H112 machine instructions: memory reference (MR), input/output (I/O), shift (SH), skip (SK) and generic (G).

A summary of H112 machine instructions is provided in Table 5-1. Refer to Section II for a detailed explanation of each instruction.

Memory Reference Instructions

The 7 memory reference instruction codes are: LDA, STA, ADD, ANA, IRS, JMP, JST. They are coded as follows:

LOCATION:	blank or any valid symbol.
OPERATION:	one of 7 codes above, followed by an asterisk (*) if indirect addressing is desired.
VARIABLE:	any valid expression which will result in a 7 bit address in either the current sector or a primary sector.
COMMENTS:	ignored.

Input/Output Instructions

The 5 input/output instruction codes are: OTA, INA, SKS, OCP, SMK. They are coded follows:

LOCATION:	blank or any valid symbol.
OPERATION:	one of 5 codes above.
VARIABLE:	any valid expression which will result in a six bit device address for all except SMK which requires a one bit result.
COMMENTS:	ignored.

Shift Instructions

The 3 shift instruction codes are: LGR, RAR, ARS. They are coded as follows:

LOCATION:	same.
OPERATION:	one of 3 codes above.
VARIABLE:	any valid expression which will result in a four bit shift count.
COMMENTS:	ignored.

Skip Instructions

The 10 skip instruction codes are: SPL, SMI, SZE, SNZ, SOV, SNO, SPN, SMZ, NOP, SKP. They are coded as follows:

LOCATION: same.
OPERATION: one of 10 codes above
VARIABLE: ignored.
COMMENTS: ignored.

Variations of these can be created by using the FORM pseudo-up. Refer to page 2-12.

Generic Instructions

The 12 generic instruction codes are: CRA, OCA, TCA, TOA, ENB, INH, STL, HLT, TAB, TBA, ITS, ITR. They are coded as follows:

LOCATION: same.
OPERATION: one of 12 codes above.
VARIABLE: ignored.
COMMENTS: ignored.

Table 5-1.
Machine Instructions

Op Code				Execution
Octal	Mnemonic	Instruction	Type	Time (μs)
1	LDA	Load A	MR	3.39
2	STA	Store A	MR	3.39
3	JMP	Jump	MR	3.39
4	ADD	Add	MR	7.63
5	ANA	AND to A	MR	7.63
6	IRS	Increment, Replace, and Skip	MR	9.33
7	JST	Jump and Store P	MR	4.66
40XX	INA	Input Transfer to Accumulator	IO	4.66
41XX	SKS	Skip If Set	IO	4.66
42XX	OTA	Output Transfer From Accumulator	IO	4.66
43XX	OCP	Output Control Pulse	IO	4.66
4300, 4301	SMK	Set Mask	IO	4.66
010X, 011X	LGR	Logical Right Shift	SH	3.8+N(0.424)
012X, 013X	ARS	Arithmetic Right Shift	SH	3.8+N(0.424)
014X, 015X	RAR	Rotate A Right	SH	3.8+N(0.424)
0200	NOP	No Operation	SK	3.39
0201	SMI	Skip If Accumulator is Negative	SK	3.39

Table 5-1. (Cont)
Machine Instructions

Op Code				
Octal	Mnemonic	Instruction	Type	Execution Time (μs)
0202	SZE	Skip If Accumulator is Zero	SK	3.39
0203	SMZ	Skip If Zero Or Negative	SK	3.39
0204	SOV	Skip If Overflow F/F is Set	SK	3.39
0300	SKP	Skip Unconditionally	SK	3.39
0301	SPL	Skip if Accumulator is Positive	SK	3.39
0302	SNZ	Skip if Accumulator is Non-Zero	SK	3.39
0303	SPN	Skip if Positive and Non-Zero	SK	3.39
0304	SNO	Skip if Overflow F/F is Reset	SK	3.39
0000	HLT	Halt	G	2.54
0003	OCA	One's Complement Accumulator	G	7.63
0005	TCA	Two's Complement Accumulator	G	7.63
0021	STL	Stall On Line	G	3.39 + (Wait)
0022	TBA	Transfer Bank Register to Accumulator	G	3.39
0024	ITS	Interrupt Save	G	3.39
0030	TOA	Transfer Overflow to Accumulator	G	3.39
0041	TAB	Transfer Accumulator to Bank Register	G	2.54
0042	ITR	Interrupt Return	G	2.54
0044	ENB	Enable Interrupts	G	2.54
0050	INH	Inhibit Interrupts	G	2.54
0060	CRA	Clear Interrupts	G	2.54

PSEUDO OPERATIONS

The 21 pseudo operations included in this section supplement standard instructions. They allow the programmer to execute functions which do not have any counterparts in machine language. Pseudo operations are divided into the following six groups of operations:

a. Data defining pseudo operations
b. Storage allocation pseudo operation
c. Symbol defining pseudo operations
d. Assembly controlling pseudo operations
e. List controlling pseudo operations
f. Special pseudo operation

Each group is discussed in detail in paragraphs which follow. Read Section 4-5 to 4-6 for expression types.

Data Defining Pseudo Operations

Two data defining pseudo operations, DATA and BASE, are described below.

DATA. -- The DATA pseudo operation defines constants and strings, and reserves the proper number of storage locations to hold them. It is coded as follows:

LOCATION:	blank or any valid symbol; if present, the symbol is assigned the address of the first location reserved.
OPERATION:	DATA
VARIABLE:	a list of strings and/or expressions, separated by commas.
COMMENTS:	ignored

Each expression in the list causes one word to be reserved and filled with the value of the expression. Each string causes a number of words sufficient to hold the string (two characters/word) to be reserved and filled. A string or expression may be repeated by preceding it by a repetition constant. This is an expression indicating the number of repetitions, followed by a period.

Example:

```
TABLE       DATA        3.7,(AA-FF).C'AB,O'77
```

will assemble 3 decimal 7's, "AA-FF" worth of 'AB', and 1 octal 77.

BASE. -- The BASE pseudo operation defines the radix (base) of any numeric elements in succeeding statements which are not prefixed by D', B', or O'. It is coded as follows:

LOCATION:	ignored
OPERATION:	BASE
VARIABLE:	n (n = 2, 3....., 10)
COMMENTS:	ignored

The assembler is initialized to BASE 10.

Storage Allocation Pseudo Operation

The BSS (Block Starting With Symbol) storage allocation pseudo operation reserves a block of storage and fills it with a specified value. BSS is coded as follows:

LOCATION:	blank or any valid symbol; if present, the symbol is assigned the address of the first location reserved.
OPERATION:	BSS
VARIABLE:	n, f (two valid expressions) n = number of locations to be reserved f = value with which to fill the reserved locations. If f is omitted, the reserved locations retain their previous values when the program is loaded into memory.
COMMENTS:	ignored

Symbol Defining Pseudo Operations

The three symbols defining pseudo operations, EQU, SET, and FORM, are as follows:

EQU (Equals). -- The EQU pseudo operation is used for permanently defining a value for a symbol for reference by other SAP-12 operations. It is coded as follows:

LOCATION:	must contain a symbol.
OPERATION:	EQU
VARIABLE:	any absolute expression. Any symbol used in this field must have been previously defined.
COMMENTS:	ignored.

SET. -- The SET pseudo operation is used to define and redefine a value for a symbol. A SET is coded as follows:

LOCATION:	must contain a symbol.
OPERATION:	SET
VARIABLE:	any absolute expression.
COMMENTS:	ignored.

Example:

```
AA      EQU     D'20
BB      EQU     D'30
FACT    SET     D'10
VAL1    EQU     (AA+BB)/FACT
FACT    SET     D'5
VAL2    EQU     (AA+BB)/FACT
```

VAL1 is set equal to 5_{10}; VAL2 to 10_{10}.

FORM. -- FORM is used to define a 12-bit word in any format desired and, optionally, to fill any part with a constant value. It is coded as follows:

LOCATION:	must contain a symbol.
OPERATION:	FORM
VARIABLE:	$n_1, n_2, n_3, \ldots, n_{12}$
COMMENTS:	ignored.

When FORM is specified, the location symbol becomes a "pseudo-op" used to break up a string of numbers into the fields defined by $n_1, n_2, n_3, \ldots, n_{12}$.

Example:

```
HALF    FORM    6,6
```

This establishes "HALF" as a pseudo-op which, when called as an operation, creates a data word with a value in the upper 6 bits and a value in the lower 6.

Example:

```
location    HALF    6,5
```

will generate a data word: 0605_8.

Optionally, the FORM may establish that parts or all of the field should be pre-filled with a certain value.

Example:

```
TABC        FORM        1,5,6=O'77
```

defines TABC as a pseudo-op breaking a field down into a 1-bit portion, a 5-bit portion, and a 6-bit portion filled with binary 1's:

```
location    TABC        1,9
```

will generate: 5177_8.

Assembly Controlling Pseudo Operations

Eight assembly controlling pseudo operations, ORG, END, SETB, COPY, PCH, CARD, SEQC, and POST, are described below.

ORG (Origin). -- The ORG pseudo operation sets the location counter to a specified value. It is coded as follows:

LOCATION:	ignored.
OPERATION:	ORG.
VARIABLE:	any absolute expression. Any symbol used in this field must have been previously defined.
COMMENTS:	ignored.

The ORG pseudo operation performs the following functions:

a. The expression in the variable field is evaluated.

b. The location counter is set to the value thus determined.

END. -- The END pseudo operation is used to terminate the current assembly pass. It is coded as follows:

LOCATION:	ignored.
OPERATION:	END.
VARIABLE:	$\binom{\text{YES}}{\text{NO}}$, expression
COMMENTS:	ignored.

If the first subfield of the variable field contains YES, the assembler will punch an End of File block on the object tape. The second subfield defines the address to which control should be transferred at the end of the loading process. If the second field is present a transfer block will be punched on the object tape.

The END pseudo-operation causes SAP-12 to perform the following functions:

a. The current block of assembly output information is terminated.

b. Transfer and End of File blocks are punched on the object tape if specified.

c. The current pass is terminated.

The END pseudo-operation must be the last statement in the source program.

SETB (Set Base Sector)

The SETB pseudo operation sets the base location counter to a new location. It is coded as follows:

LOCATION:	ignored.
OPERATION:	SETB
VARIABLE:	k, n (two valid expressions) k = location at which base location counter is to be set. n = number of locations in the block reserved for this SETB.
COMMENTS:	ignored.

The location specified by the first expression in a SETB instruction should be defined as the beginning of a block of storage of length equal to the second expression in the SETB. All indirect links generated in the assembler are stored in this block until another SETB is encountered. A SETB will normally have a corresponding BSS with location field = k and variable field = n.

A special case of SETB is as follows:

```
SETB          !
```

This allows the programmer to switch between a SETB in the current sector and a SETB in the primary sector. (See Figure 6-2.)

COPY. -- The COPY pseudo operation directs the assembler to copy symbolic statements from a symbolic punched paper tape. It is coded as follows:

LOCATION:	ignored.
OPERATION:	COPY
VARIABLE:	asterisk or program name (1 - 6 letters)
COMMENTS:	ignored.

If the variable field is an asterisk, the assembler stops immediately and requests the operator to load more paper tape. This option is used when a source language paper tape is in two or more pieces.

If the variable field contains a program name, the assembler stores the name until the end of the current pass. At that time, it requests the operator to load a symbolic library paper tape. The assembler reads that tape, copying subroutines whose names are in the list, ignoring all others.

Subroutines are named by a special statement containing an asterisk in column 1, a dollar sign in column 2, and the name in columns 3-8. This statement must be the first statement in the subroutine.

PCH (Punch). -- The PCH pseudo operation directs the assembler to punch a copy of the source program onto paper tape. It is coded as follows:

LOCATION: ignored.
OPERATION: PCH
VARIABLE: YES or NO
COMMENTS: ignored.

If the variable is YES, all statements are copied until a PCH NO statement is encountered, at which time, punching stops. The assembler is normally in the PCH NO mode. The output is formatted according to the current TABS. PCH does not operate when the ASR is the input device.

NOTE

The PCH statement itself will never be punched.

CARD (Card Length). -- The CARD pseudo operation gives the assembler the length in characters of the longest source statement. It is coded as follows:

LOCATION: ignored
OPERATION: CARD
VARIABLE: n (n = 1-80, decimal)
COMMENTS: ignored

The CARD pseudo operation speeds up the assembler by terminating the image scan. The entire statement is printed on the listing regardless of the value on a CARD statement. The assembler assumes 80 column images until a CARD statement is encountered. CARD operates on all input media.

SEQC (Sequence Check On Cards). -- The SEQC pseudo operation notifies the assembler that the input cards have a sequence number punched in columns 73 to 80 which should be checked. The contents of symbolic instructions containing the SEQC pseudo operation are as follows:

LOCATION: ignored
OPERATION: SEQC
VARIABLE: YES or NO
COMMENTS: ignored

When YES, this pseudo operation directs the assembler to perform a sequence check of columns 73 to 80 while reading the source cards. Any sequence errors are flagged. Cards with blanks in columns 73 to 80 are regarded as not in error. The assembler is normally in SEQC NO mode. SEQC YES operates only if CARD pseudo-op is 80.

POST (Post Processor Request). -- The POST pseudo operation directs the assembler to produce a post processor listing. It is coded as follows:

LOCATION: ignored
OPERATION: POST
VARIABLE: ignored
COMMENTS: ignored

The POST pseudo operation (if included) must be the first non-comment statement in the program. Otherwise, it is ignored. The use of POST expands the symbol table.

List Controlling Pseudo Operations

Six list controlling pseudo operations, LIST, GENR, TABS, EJCT, SPAC, and TTL are described below.

LIST. -- The LIST pseudo operation directs the assembler to print a side-by-side listing of the program being assembled. It is coded as follows:

LOCATION:	ignored.
OPERATION:	LIST
VARIABLE:	YES or NO
COMMENTS:	ignored.

When YES, the LIST pseudo operation causes the source program and its octal representation to be listed on the on-line typewriter or printer. The assembler then continues to operate in the listing mode until a LIST NO pseudo operation is encountered. The assembler is normally in the LIST YES mode.

When NO, this pseudo operation directs the assembler to suppress the listing of the program being assembled.

GENR (Print Generated Lines). -- The GENR pseudo operation directs SAP-12 to print generated object code lines on the assembly listing. It is coded as follows:

LOCATION:	ignored.
OPERATION:	GENR
VARIABLE:	YES or NO
COMMENTS:	ignored.

When YES, the GENR pseudo operation causes the printout routine to print one line for each generated word produced by the assembler. Generated words include indirect links and multiple words created as a result of a DATA statement. The assembler is normally in the GENR YES mode.

When NO, this pseudo operation directs the assembler to suppress generated words on the assembly listing.

TABS (Set Tabs On Listing). -- The TABS pseudo operation allows the programmer to set tab stops for the four fields of symbolic output: location, operation, variable, and comments. It allows reformatting of the source which may have been punched free form. It is coded as follows:

LOCATION:	ignored.
OPERATION:	TABS
VARIABLE:	L, O, V, C
COMMENTS:	ignored.

L = print column to begin location field (default = 1)

O = print column to begin operation field (default = 12)

V = print column to begin variable field (default = 19)

C = print column to begin comment field (default = 34)

NOTE

Depending on the value of C, the comment field may be truncated to fit on the page.

EJCT (Eject Page). -- The EJCT pseudo operation directs the assembler to begin or resume listing on a new page. It is coded as follows:

LOCATION:	ignored.
OPERATION:	EJCT
VARIABLE:	ignored.
COMMENTS:	ignored.

The EJCT pseudo operation causes the printout routine to generate the necessary commands to advance the listing one page and continue listing on a new page. This pseudo operation is ignored if the most recent LIST pseudo operation was NO.

An automatic EJCT is performed whenever: the page line count is reached; a TTL pseudo operation is encountered; a new sector is begun.

SPAC (Space One Line). -- The SPAC pseudo operation directs the assembler to space a number of lines before resuming printing. It is coded as follows:

LOCATION:	ignored.
OPERATION:	SPAC
VARIABLE:	an expression whose value is the number of lines to be spaced.
COMMENTS:	ignored.

The SPAC pseudo operation causes the printout routine to generate the number of blank lines specified in the variable field. A blank variable field produces one space. Should the printout routine reach the bottom of a page while spacing, the printout skips to the top of a new page and further spacing is discontinued.

TTL (Page Title). -- The TTL pseudo operation directs the assembler to print a title line beneath the heading line at the top of each page of the assembly printout. It is coded as follows:

LOCATION:	ignored.
OPERATION:	TTL
VARIABLE:	a string containing from 1 to 60 characters which make up the title line to be printed.
COMMENTS:	none.

The TTL operation causes the printout routine to store the contents of the variable field in a title buffer. This buffer is printed at the top of each page beneath the heading

line and above the first line of the assembly listing. This title buffer is normally set to blank when the assembler is initialized.

TTL causes a page eject.

Special Pseudo Operation

The special pseudo operation "***" is an undefined operation code. This pseudo operation is provided as a programmer convenience to indicate an operation to be filled in at run time. The special pseudo operation is coded as follows:

LOCATION:	blank or any valid symbol.
OPERATION:	***
VARIABLE:	any valid expression which will result in a 7-bit address in either the current sector or a primary sector.
COMMENTS:	ignored.

This pseudo operation is assembled as a memory reference instruction with an operation code of zero. A fourth asterisk in the operation field causes the indirect bit to be set in the generated word.

SECTION VI
PROGRAMMING THE H112

This section serves as a guide for programmers who have never programmed a small computer. The more experienced programmer will be introduced to the programming features of the H112. The programming examples shown in this section were carefully selected to demonstrate basic programming considerations. Most programs written for the H112 will use a number of the routines shown.

PROGRAMMING ORGANIZATION

The effort involved in producing a working program in a small machine is dependent upon several considerations and practices. It is possible to reduce that effort by careful attention to organization.

1. The program should be organized into subroutines, each of which is a logical entity. Each subroutine should be entirely contained in one sector of core memory. A subroutine should never cross a sector boundary.

2. A hierarchical chart should be generated, illustrating the "program tree" of subroutines. This chart should show the subroutines called by each routine and the common symbols used by each (a common symbol is one which is used in more than one routine). The size of each routine should be included.

3. Using the information contained on the hierarchical chart, the programmer should group small subroutines which use common symbols into "units". Each unit should be contained in a single sector. Symbols which are unique to one subroutine or which are common only to the subroutines in one sector should refer to locations in that sector.

4. The allocation of memory may change during the debuggin process. It is, therefore, especially important that rigorous documentation procedures be established and followed. In general, every line of code should have a comment; every group of instructions that form a procedure should be clearly labeled as such.

5. Subroutines should be checked out individually. The routines at the bottom of the "program tree" should be checked out first. These are the routines that do not call other routines. By debugging in this manner, the problems of reallocation of memory are generally reduced, since once a sector unit has been checked out, it does not require alteration again.

Each of the seven H112 memory reference instructions (LDA, STA, ADD, ANA, IRS, JMP, JST) contains a seven bit address field. This allows 128 locations ($0\text{-}177_8$) to be directly referenced. As a result, memory is divided into sectors of 128 words each. The sectors begin at addresses 0, 200_8, 400_8, 600_8, etc. A 4K H112 memory contains 32 sectors numbered $0\text{-}37_8$; an 8K H112 memory contains 64 sectors numbered $0\text{-}77_8$.

Each memory reference instruction contains a sector bit. This bit determines the sector in which the direct address is located. If the bit is ZERO, the direct address is located in a primary sector (sector 0 or 40_8); if the bit is ONE, the direct address is located in the same sector as the instruction.

Each memory reference instruction contains an indirect bit. If this bit is a ZERO, the instruction operates on the contents of the direct address; i.e., the effective address equals the direct address. If the indirect bit is a ONE, the instruction operates on the contents of the location specified by the direct address; i.e., the effective address equals the contents of the location specified by the direct address.

In a 4K machine, or when the bank bit in an 8K machine is ZERO, memory referencing occurs as shown in Table 6-1.

Table 6-1.
Memory Referencing in 4K Mode

Example No.	Current Program Location	Address assembled in the Instruction (7-bits)	[plus]	Sector Bit of:	[and]	Indirect Bit of:	[yields]	Effective Address of:
1	204	134		1		0		334
2	204	134		0		0		134
3	204	134		1		1		(all 12 bits at location 334)
4	204	134		0		1		(all 12 bits at location 134)

SECTOR PROGRAMMING

In example 1, the sector bit is a logical "1". This designates the 7-bit address as being within the current sector. Thus, the effective address is the present sector starting address (200) plus the 7-bit address (134), or 334.

In example 2, the sector bit is a "0". This designates the address as being in the primary sector, starting with memory location 0000. Hence, the effective address 134 is referenced.

In example 3, the indirect bit, in addition to the sector bit, is a "1". This indicates indirect memory referencing. Indirect addressing requires that the machine go to the address, as specified in example 1, and read the full 12-bit word as the effective address for the instruction. Indirect addressing takes longer than direct referencing but allows reference to any sector.

Example 4 also shows indirect addressing. However, the machine will obtain the 12-bit effective address from location 134.

In an H112 with 8K memory, the bank bit is included in the calculation of the effective address. If the instruction is located in the lower 4K and the bank bit is ZERO, memory referencing occurs exactly as shown in Table 6-1. If the instruction is located in the upper 4K and the bank bit is ONE, memory referencing occurs in a similar manner to that shown

in Table 6-1 except that 10000_8 (4096_{10}) is added to each address. For example, if the instruction location is 10204_8, the instruction address is 134_8, and the sector and indirect bits are both ONE (see example 3), then the effective address is all 12 bits at location 10334_4.

If the instruction is in the lower bank and the bank bit is ONE, all direct addresses are in the lower bank, but all indirect addresses are increased by 10000_8. Using example 3 under this condition, the instruction location is 204_8, the instruction address is 134_8, and the effective address is the contents of location 334_8 plus 10000_8. If 334_8 contains 1234_8, the effective address is 11234_8.

If the instruction is in the upper bank and the bank bit is ZERO, all direct addresses are in the upper bank, but all indirect addresses are in the lower bank. Again referring to example 3, the instruction location is 10204_8, the instruction address is 134_8, and the effective address is the contents of location 10334_8. If 10334_8 contains 1234_8, the effective address is 1234_8.

Sector Zero

Sector zero and sector 40_8 are the primary sectors and certain uses are reserved for these sectors. This is due to the addressing structure of the machine and the fact that upon receipt of an interrupt, the machine will go to location 00002 for interrupt instructions. Sector zero is generally used for storage of frequently referenced data and for start-up entries. Typical content for sector zero is shown in Figure 6-1.

Octal Memory Location	Symbolic Location	Operation	Variables	
		ORG	0	
		SET B	ENDSECT,N	Primary Sector Indirect Link Block
0000		JMP*	*+1	Start After Reset
0001		DATA	START	Start Address
0002		JST*	*+1	Executed on Interrupt
0003		DATA	INTR	Interrupt Routine Address
	*			
	D1	DATA	1	Commonly Used Constants
	D2	DATA	2	
	*			
	M7777	DATA	0'7777	
	*			
	AA	DATA	A	Commonly Used Parameters
	BA	DATA	B	
	*			
	XA	DATA	X	
	NA	DATA	N	
	ENDSECT	BSS	N	
0177				(Last Address in Sector Zero)

Figure 6-1. Typical Sector Zero Allocation

Automatic Sectorization

Because a memory reference instruction can directly address only those locations which are either in its own sector or in the primary sector, it is necessary to create "indirect links" to locations in other sectors. To assist the programmer in this task, the assembler creates these links whenever possible. For example, if a memory reference instruction in location 342_8 requires an address of 401_8, the instruction may be assembled with an address of 007_8, a sector bit of zero and an indirect bit of 1. Location 007_8 will have 0401_8 placed in it by the assembler; this is the "indirect link" for the instruction in 342_8.

The programmer must give the assembler a block of storage in which to place the links which it creates. This is accomplished with the SETB pseudo operation. The most efficient method is to place all links in the primary sector; however, a large program may require more links than can fit in the primary sector. In addition, the programmer will normally wish to reserve part of the primary sector for common parameters and constants. In this case, some links must be placed in the sector containing the corresponding memory reference instructions.

A SETB instruction should reference a corresponding BSS instruction. Normal practice is to place a SETB as the first instruction in a sector, and the corresponding BSS as the last instruction in that sector. A useful estimate in allocating storage is that a block of approximately 100_{10} words of code will require about 25 words of parameters, constants and indirect links.

The assembler retains two base location counters: the most recent SETB location in the primary sector and the most recent SETB location in the current sector. The use of an exclamation point in the variable field of an SETB enables the programmer to swap between the two. (See Figure 6-2).

Subroutines

Coding subroutines in the H112 is simplified by using instructions which perform multiple functions. The JST instruction jumps to the subroutine and stores a return address; the IRS instruction increments a cell and replaces it in memory.

When a single parameter is to be passed to a subroutine, the usual method is to pass it in the A-register. As an example, a subroutine which spaces the ASR a variable number of lines is shown in Figure 6-3.

As an alternative, or when more than one parameter is required by the subroutine, a parameter vector may be listed below the JST, as shown in the print subroutines, Figure 6-4.

Note that the above examples of subroutines require that the subroutines reside in the same sector as the calling program. If they are in different sectors, indirect addressing (JST*) is required.

CAP/SAP 12 CODING FORM
F2435-869

PROGRAMMER | EXT. | KEYPUNCH NOTES | DATE | PAGE OF
PROGRAM | CHARGE NUMBER

Location	Operation	Variables	Comments	Sequence Number
	ØRG	4	START IN SECTØR ZERØ	
	SETB	BS1,100		
BS1	BSS	100		
	ØRG	ø'1400		
	SETB	BS14,25	SET BASE IN CURRENT SECTØR	
	LDA	X	IF X IS IN ANØTHER SECTØR, THE LINK WILL	
*			BE GENERATED IN THE BS14 BLØCK	
	SETB	!	SWAP TØ PRIMARY SECTØR BASE	
	LDA	Y	THIS LINK WILL BE IN THE BS1 BLØCK	
	SETB	!	SWAP TØ CURRENT SECTØR BASE	
	LDA	I	THIS LINK WILL BE IN THE BS14 BLØCK	
BS14	BSS	25		

Figure 6-2. SETB Pseudo Operation

CAP/SAP 12 CODING FORM
F2435-869

PROGRAMMER | EXT. | KEYPUNCH NOTES | DATE | PAGE
PROGRAM | CHARGE NUMBER

Location	Operation	Variables	Comments	Sequence Number
*	MAIN	PRØGRAM		
	≀			
	LDA	NLINS	PICK UP NUMBER ØF LINES	
	JST	SPACE	GØ SPACE	
	≀		(RETURN)	
*	SPACE SUBRØUTINE			
*				
SPACE	DATA	**	ENTRY PØINT (CØNTAINS RETURN ADDRESS)	
	TCA		CØMPLEMENT CØUNT	
	STA	SPACN	SET CØUNT	
	LDA	LFEED	PICK UP ASCII LINE FEED CHARACTER	
SPACI	ØTA	0,2	ØUTPUT ØNE CHARACTER TØ ASR	
	JMP	*-1	WAIT FØR ASR NØT BUSY	
	IRS	SPACN	INCREMENT CØUNT	
	JMP	SPACI	NØT ZERØ, ØUTPUT ANØTHER	
	JMP*	SPACE	ZERØ, RETURN TØ MAIN PRØGRAM	
SPACN	BSS	1	STØRAGE FØR CØUNT	
LFEED	DATA	Ø'212	ASCII LINE FEED	

Figure 6-3. ASR Space Subroutine

CAP/SAP 12 CODING FORM

F2435-869

PROGRAMMER | EXT. | KEYPUNCH NOTES | DATE | PAGE 01

PROGRAM | CHARGE NUMBER

Location	Operation	Variables	Comments	Sequence Number
*	MAIN	PROGRAM		
	{			
	JST	PRINT	GO PRINT	
	DATA	D'3, D'5, MESG	NO OF SPACES, NO OF CHRS., ADRS OF MESSAGE	
	{		(RETURN)	
*	PRINT	SUBROUTINE		
*				
PRINT	DATA	**	ENTRY POINT	
	LDA*	PRINT	PICK UP NO OF SPACES	
	JST	SPACE	GO SPACE	
	IRS	PRINT	INCREMENT RETURN ADDRESS	
	LDA*	PRINT	PICK UP NO OF CHRS. TO PRINT	
	TCA		COMPLEMENT COUNT	
	STA	PRINN	SET COUNT	
	IRS	PRINT	INCREMENT RETURN ADDRESS	
	LDA*	PRINT	PICK UP ADDRESS OF MESSAGE	
	STA	PRINA	SET ADDRESS	
PRINI	LDA*	PRINA	PICK UP ONE CHARACTER OF MESSAGE	
	OTA	02	OUTPUT ON ASR	
	JMP	*-1	WAIT FOR ASR NOT BUSY	
	IRS	PRINA	INCREMENT ADDRESS	
	IRS	PRINN	INCREMENT COUNT	

Figure 6-4. Print Subroutine

CAP/SAP 12 CODING FORM

F2435-869

PROGRAMMER | EXT. | KEYPUNCH NOTES | DATE | PAGE OF

PROGRAM | CHARGE NUMBER

Location	Operation	Variables	Comments	Sequence Number
	JMP	PRINI	NØT ZERØ, PRINT ANØTHER	
	IRS	PRINT	ZERØ, INCREMENT RETURN ADDRESS	
	JMP*	PRINT	RETURN	
PRINN	BSS	1	STØRAGE FØR CØUNT	
PRINA	BSS	1	STØRAGE FØR ADDRESS	

Figure 6-4. Print Subroutine (Cont)

Programming With Expanded (8K) Memory

The full 12-bit word of the H112 allows only 4K of the memory to be indirectly addressed; however, in the 8K machine, the programmer uses the bank register (B register bit) to indicate whether the upper or lower 4K memory bank is used. Thus, the programmer must be careful with indirect addressing when programming the 8K machine, having regard for memory bank boundaries.

A routine which copies a block of data across the bank boundary is provided as an example in Figure 6-5, using non-jump instructions. Assume that the routine is in the lower bank (zero), and that the data is in the upper bank (one).

The inefficiency of the above routine demonstrates the problems which may be encountered when communicating across banks, and indicates why this should be avoided whenever possible.

Notice that the routine would be more efficient if the destination address were in the same sector as the routine, allowing direct address. More efficient programming for a similar routine is shown in Figure 6-6.

A jump instruction to a subroutine across the bank boundary is shown in Figure 6-7.

A programmer may wish to write a general subroutine which can be assembled with any main program. In such a case, it may be necessary for the subroutine to determine the bank in which it is residing. This may be accomplished by executing two TBA instructions when the subroutine is entered. The first TBA will transfer to the A-register the bank bit of the bank in which the calling program resides, and will transfer (P13) of the current routine into the bank register. A second TBA will transfer this bank bit to the A-register.

CARRY VS OVERFLOW DURING ADDITION

The overflow bit, contained in the overflow (OV) register, is actually a carry bit. An addition results in a carry bit if:

1. Both numbers being added are negative
2. Numbers being added are different in signs and a positive number results.

The following subroutine demonstrates the use of the carry bit to determine whether or not overflow occurred during addition (see Figure 6-8).

CAP/SAP 12 CODING FORM

12435-869

PROGRAMMER | DATE | KEYPUNCH NOTES | PAGE

PROGRAM | CHARGE NUMBER

Location	Operation	Variables	Comments	Sequence Number
*	MØVE	BLØCK ØF STØRAGE	FRØM BANK 1 TØ BANK 0	
*				
	LDA	ADRB1	SET INITIAL ADDRESS IN BANK 1	
	STA	A1	*	
	LDA	ADRB0	SET INITIAL ADDRESS IN BANK 0	
	STA	A0	*	
	LDA	NMBR	SET NUMBER ØF WØRDS TØ BE MØVED	
	TCA		*	
	STA	CØUNT	*	
*				
LØØP	LDA	D1	SET BANK REGISTER TØ 1	
	TAB		*	
	LDA*	A1	PICK UP A WØRD	
	STA	TEMP	SAVE IT	
	CRA		SET BANK REGISTER TØ ZERØ	
	TAB		*	
	LDA	TEMP	RESTØRE WØRD	
	STA*	A0	PUT IT AWAY	
	IRS	A1	INCREMENT SØURCE ADDRESS	
	IRS	A0	INCREMENT DESTINATIØN ADDRESS	
	IRS	CØUNT	INCREMENT CØUNT	
	JMP	LØØP	NØT DØNE, GET ANØTHER WØRD	

Figure 6-5. Inefficient 8K Programming

CAP/SAP 12 CODING FORM
F2435-869

PROGRAMMER | EXT. | KEYPUNCH NOTES | DATE | PAGE 0
PROGRAM | CHARGE NUMBER

Location	Operation	Variables	Comments	Sequence Number
D1	DATA	1	CONSTANT 1	
A0	BSS	1	STORAGE FOR DESTINATION ADDRESS	
A1	BSS	1	STORAGE FOR SOURCE ADDRESS	
COUNT	BSS	1	COUNTER	
ADRB1	DATA	SOURCE	12 BITS OF UPPER BANK ADDRESS	
ADRB0	DATA	DEST	LOWER BANK ADDRESS	
NMBR	DATA	N	NUMBER OF WORDS	

Figure 6-5. Inefficient 8K Programming (Cont)

CAP/SAP 12 CODING FORM
E2435-869

PROGRAMMER | EXT. | KEYPUNCH NOTES | DATE | PAGE OF
PROGRAM | CHARGE NUMBER

Location	Operation	Variables	Comments	Sequence Number
*	MØVE	BLØCK ØF STØRAGE	FRØM BANK 1 TØ BANK 0	
*				
	LDA	ADRB1	SET INITIAL ADDRESS IN BANK 1	
	STA	A1	*	
	LDA	ADRB0	PICK UP INITIAL ADDRESS IN BANK 0	
	ANA	M177	MASK TO 7 BITS	
	ADD	STA1	ADD SKELETØN STØRE INSTRUCTIØN	
	ADD	SBIT	ADD SECTØR BIT	
	STA	VSTA	SET STØRE INSTRUCTIØNS	
	LDA	NMBR	SET NUMBER ØF WØRDS TØ MØVE	
	TCA		*	
	STA	CØUNT	*	
	LDA	D1	SET BANK REGISTER TØ UPPER	
	TAB		*	
*				
LØØP	LDA*	A1	PICK UP A WØRD	
VSTA	***	**	STØRE IT	
	IRS	VSTA	INCREMENT DESTINATIØN ADDRESS	
	IRS	A1	INCREMENT SØURCE ADDRESS	
	IRS	CØUNT	INCREMENT CØUNT	
	JMP	LØØP	NØT DØNE, GET ANØTHER WØRD	
SBIT	DATA	Ø'200	SECTØR BIT FØR MEMØRY REFERENCE	

Figure 6-6. Efficient 8K Programming

CAP/SAP 12 CODING FORM
F2435-869

PROGRAMMER | EXT. | KEYPUNCH NOTES | DATE | PAGE 0
PROGRAM | CHARGE NUMBER

Location	Operation	Variables	Comments	Sequence Number
STAI	STA	**	SKELETØN STØRE INSTRUCTIØN	
D1	DATA	1	CØNSTANT 1	
ADRB1	DATA	SØURCE	12 BITS ØF UPPER BANK ADDRESS	
ADRB0	DATA	DEST	LØWER BANK ADDRESS (IN THIS SECTØR)	
M177	DATA	Ø'177	ADDRESS MASK	
NMBR	DATA	N	NUMBER ØF WØRDS	
A1	BSS	1	STØRAGE FØR SØURCE ADDRESS	
CØUNT	BSS	1	CØUNTER	
DEST	BSS	N	BLØCK ØF STØRAGE TØ BE FILLED	

Figure 6-6. Efficient 8K Programming (Cont)

CAP/SAP 12 CODING FORM
F2435-869

PROGRAMMER | EXT. | KEYPUNCH NOTES | DATE | PAGE
PROGRAM | CHARGE NUMBER

Location	Operation	Variables	Comments	Sequence Number
*	MAIN	PROGRAM IN BANK ZERO		
*	}			
	LDA	D1	SET BANK TO 1	
	TAB		*	
	JST*	SUBR	GO EXECUTE	
	DATA	3	PARAMETER	
	}			
	TRA		RETURN. ESTABLISH YOUR OWN BANK AGAIN.	
	}			
*	SUBROUTINE IN BANK 1			
*				
SUBR	DATA	**	ENTRY	
	LDA*	SUBR	PICK UP PARAMETER	
	STA	A0	*	
	IRS	SUBR	INCREMENT RETURN	
	TBA		SAVE CALLING BANK	
	STA	BANK	*	
	}		}(EXECUTE)	
	LDA	BANK	RESTORE CALLING BANK	
	TAB		*	
	JMP*	SUBR	RETURN	

Figure 6-7. Jump Instructions Across Memory Bank Boundaries

CAP/SAP 12 CODING FORM

F2435-869

PROGRAMMER | DATE | PAGE 0

PROGRAM | CHARGE NUMBER

Location	Operation	Variables	Comments	Sequence Number
*				
A0	BSS 1	1	STORAGE FOR PARAMETER	
BANK	BSS 1	1	STORAGE FOR CALLING BANK	

Figure 6-7. Jump Instructions Across Memory Bank Boundaries (Cont)

CAP/SAP 12 CODING FORM
F2435-869

PROGRAMMER | EXT. | KEYPUNCH NOTES | DATE | PAGE
PROGRAM | CHARGE NUMBER

Location	Operation	Variables	Comments	Sequence Number
	TTL	ADD WITH ØVERFLØW TEST		
*	JST	ADDØT		
*	DATA	(FIRST NUMBER)		
*	DATA	(SECØND NUMBER)		
*	ERRØR	RETURN		
*	NØRMAL	RETURN		
ADDØT	DATA	**	ENTRY	
	TØA		CLEAR CARRY BIT	
	LDA*	ADDØT	PICK UP FIRST NUMBER	
	STA	ADDA	*	
	IRS	ADDØT	INCREMENT RETURN ADDRESS	
	LDA*	ADDØT	PICK UP SECØND NUMBER	
	STA	ADDB	*	
	ANA	M4000	MASK ØUT ALL BUT BIT 12 (SIGN)	
	ADD	ADDA	ADD TØ FIRST NUMBER	
	SNØ		SKIP IF NØ CARRY	
	JMP	NEG2	CARRY=BØTH NEGATIVE	
	SMI		NØ CARRY+MINUS=ØPPØSITE SIGNS	
	JMP	PØS2	NØ CARRY+PLUS=BØTH PØSITIVE	
*				
	LDA	ADDA	ØPPØSITE SIGNS CANNØT ØVERFLØW	
	ADD	ADDB	ADD THEM TØGETHER	
	JMP	ADDNR	RETURN	
*				

Figure 6-8. Addition and Overflow Detection

CAP/SAP 12 CODING FORM

F2435-869

PROGRAMMER | EXT. | KEYPUNCH NOTES | DATE | PAGE OF

PROGRAM | CHARGE NUMBER

Location	Operation	Variables	Comments	Sequence Number
P0S2	LDA	ADDA	B0TH P0SITIVE	
	ADD	ADDB	ADD THEM T0GETHER	
	SMI		IF NEGATIVE, HAVE 0VERFL0W	
	JMP	ADDNR	G00D, N0RMAL RETURN	
	JMP	ADDER	0VERFL0W, ERR0R RETURN	
*				
NEG2	LDA	ADDA	B0TH NEGATIVE	
	ADD	ADDB	ADD THEM T0GETHER	
	SPL		IF P0SITIVE, HAVE 0VERFL0W	
*				
ADDNR	IRS	ADD0T	N0RMAL RETURN, STEP TWICE	
ADDER	IRS	ADD0T	ERR0R RETURN, STEP 0NCE	
	JMP*	ADD0T	RETURN	
*				
M400	DATA	0'4000	MASK IN BIT 12 (SIGN)	
ADDA	BSS	1	ST0RAGE F0R FIRST NUMBER	
ADDB	BSS	1	ST0RAGE F0R SEC0ND NUMBER	

Figure 6-8. Addition and Overflow Detection (Cont)

Normal I/O Process

The normal I/O process is the noninterrupt exchange of data between the H112 and other equipment, such as the ASR and high-speed tape reader. The normal I/O process could occur when outputting data to the ASR or inputting data as in the program loader. Programming examples for the high-speed tape reader and ASR-33 are shown in Figures 6-9 and 6-10, respectively.

Interrupts

The interrupt facility enables an external device to change the normal program sequence with low response latency. An interrupt request occurs when one of the devices on the common interrupt line (KINTL) grounds the line. If interrupts have been enabled by the controller program in progress, the request is honored on completion of the current instruction and control is transferred to the dedicated location 00002. Additional interrupts will be inhibited until the execution of an ENB instruction.

Since it is generally desirable, after servicing an interrupt, to continue the interrupted program as though no interrupt processing intervened, the contents of all registers affected by the servicing routine must be saved before they are changed and restored before control is returned to the interrupted program. The program counter is saved during the execution of the JST or JST* which must be in location 00002. This instruction stores in its effective address the location of the next command of the main (interrupted) program. The accumulator is stored by an STA instruction. The overflow flip-flop is transferred to the accumulator with a TOA and then stored with an STA.

The next task to be executed is the determination of the interrupting device. Each device is polled in turn with an SKS, skip if device not interrupting, instruction. If a particular device is interrupting, the SKS will not skip, and control can be transferred to the interrupt service routine. The order in which the devices are polled determines priorities in cases of simultaneous interrupt. Therefore, devices with higher priority should be polled first. Another consideration in the ordering of device polling is the relative frequency with which the devices will cause interrupts. Those devices causing frequent interrupts should be polled early in the list to minimize SKS processing time.

Once the interrupting device has been determined, the interrupt service routine for that device may be executed. If the interrupt was calling for a data transfer, the transfer may now take place. The interrupt request logic in the device is normally reset by the data transfer; however, if the interrupt does not call for a transfer, (a real time clock pulse interrupt for example), the execution of an instruction may be required to reset the interrupt.

After the interrupt service routine is completed, control is usually returned to the main (interrupted) program. Before this is done, however, the various registers of the machine must be restored to their condition just prior to the interrupt. The status of the overflow may be restored by clearing the flip-flop with a TOA, loading the value saved into the A-register with an LDA, and shifting it into the overflow flip-flop with an LGR 01. The contents of the A-register may be restored with an LDA from the location where the accumulator was saved.

CAP/SAP 12 CODING FORM
F2435-869

PROGRAMMER | EXT. | KEYPUNCH NOTES | DATE | PAGE 0
PROGRAM | CHARGE NUMBER

Location	Operation	Variables	Comments	Sequence Number
	ØCP	04	START HSPTR	
	INA	05	INPUT A CHARACTER FRØM HSPTR IF READY	
	JMP	*-1	DELAY UNTIL READY	
	ØCP	05	STØP HSPTR	

Figure 6-9. High-Speed Tape Reader Subroutine

CAP/SAP 12 CODING FORM
F2435-869

PROGRAMMER | EXT. | KEYPUNCH NOTES | DATE | PAGE 0
PROGRAM | CHARGE NUMBER

Location	Operation	Variables	Comments	Sequence Number
* THIS	ROUTINE WILL TURN ON		THE ASR PAPER TAPE READER	
* AND	INPUT	THE CHARACTERS		
	SKS	02	WAIT FOR THE ASR NOT BUSY	
	JMP	*-1		
	LDA	XON	LOAD ACCUMULATOR WITH X-ON CHARACTER	
	OTA	02	OUTPUT TO ASR (X-ON TURNS ON READER)	
	JMP	*-1	WAIT FOR NOT BUSY	
	~			
	INA	01	INPUT FROM ASR PAPER TAPE READER	
	JMP	*-1	WAIT FOR READY	
	~			
XON	DATA	Ø'221	X-ON CHARACTER	
* TO TURN	OFF	READER, THERE MUST BE AN X-OFF CHARACTER ON THE TAPE		

Figure 6-10. Normal ASR-33 I/O Process Subroutine

To complete the restoration of the machine context, interrupts are enabled with an ENB and the P register is restored with a JMP* through the location which is the effective address of the instruction in the interrupt location 00002. Because the ENB does not immediately enable interrupts, but delays the action until the end of the next instruction, the JMP* will be completed even though another interrupt may have occurred during the service routine. In this case, the interrupt will be waiting when the ENB is executed and will be recognized at the completion of the JMP*. The entire procedure above of saving and restoring content will again be executed without a single line of the main program code being processed. However, when all of the interrupts have been serviced, control will be returned to the main program with the original register contents restored.

INTERRUPTIBLE INTERRUPTS

The above discussion presents a method of handling interrupts in which the entire interrupt service routine is processed while interrupts are inhibited. In some applications which have one or more interrupts that require immediate servicing, it may be desirable to have the interrupt service routines be interruptible themselves. This may be the case if the system has some interrupt service routines which are longer in processing time than can be tolerated by the interrupts which require immediate service. Two additional factors must be considered in the programming for interruptible interrupt service routines. The first is that a common area for the storage of the program counter, A register and overflow flip-flop is not sufficient. If service routine "A" interrupts the main program and stores its parameters in a set of locations, then interrupt "B" occurs, service routine "B" must store the parameters of "A", yet the parameters of the main program must not be destroyed. Service routine "A" will be resumed after "B" is completed and when "A" is completed, control will be returned to the main program. A convenient way of providing this storage in each routine is first to determine the interrupt source in the conventional manner with the SKS-JMP list before storing the accumulator and overflow flip-flop. Once the interrupting device has been determined, and the service routine entered, the service routine may store the accumulator and the overflow flip-flop in local storage. The return information must be moved from where the instruction in location 00002 stored the program counter to storage local to the service routine, so that the next interrupt will not destroy the previous return address.

The second factor which must be considered when programming interruptible interrupt service routines is masking. Provision is made in the interrupt logic of each device to selectively inhibit interrupts on a device basis by use of the SMK instructions. By masking off the interrupt from the device being serviced and all other devices which are less important, the problem of a double device interrupt is solved and a priority sequence is provided.

This service routine may have interrupted a lower priority routine with a mask configuration it would be desirous to restore. This may be accomplished by maintaining an image of the mask bits in a common depository in memory. An interrupt routine may first retrieve this mask image, save it locally for future restoration, then AND in a field of ONE's for higher priority devices and ZERO's for lower priority devices including the device

being serviced. This mask is then outputted with an SMK instruction and also stored in the common depository.

Once both of the above tasks have been accomplished, interrupts may be enabled with an ENB instruction.

On exiting from such an interruptible service routine, interrupts are first inhibited with an INH instruction. The former mask may be restored with an SMK instruction and placed in the common depository. The accumulator and overflow flip-flop are restored in the usual manner from local storage and interrupts are enabled. Control is returned with a JMP* through the locally stored program counter.

Interrupts In An 8K Machine

Several other factors must be considered when programming an 8K machine. The hardware is such that upon interrupt recognition, the bank register and P13 are stored in the Z register and then the bank register and P13 are cleared. Thus if location 00002 contains a JST*, the effective address will be in the lower 4K bank and unless the bank register is modified by the service routine, all indirect references will be in the lower 4K bank. The contents of the Z register may be saved by using an ITS instruction to transfer Z to A for storage. When the service routine is complete and context is to be restored, the Z register can be loaded from the A register with an ITR instruction. This instruction also readies logic in the controller such that on the first JMP* following the ITR, the contents of Z is transferred to the bank register and P13. This JMP* should be the instruction which transfers control to the interrupted program. See Figure 6-11 for an example of an 8K non-interruptible service routine.

Generating a System Object Tape

It is often useful to assemble sections of a large program as separate subprograms. Since the loader tape does not contain a facility for linking these programs, a transfer vector in the primary sector must be used. See programming examples in Figure 6-12. The three small programs shown are assembled by three separate assemblies, but are punched on the same piece of object tape; hence, the entire tape is loaded as if it were one program. As each subroutine is loaded, it fills the proper entry in the transfer vector with its own starting address. It is also reasonable to allocate each subroutine a few locations in the primary sector for indirect links. Any common parameters should be defined in all programs.

CAP/SAP 12 CODING FORM

F2435-869

PROGRAMMER | EXT. | KEYPUNCH NOTES | DATE | PAGE 01

PROGRAM | CHARGE NUMBER

Location	Operation	Variables	Comments	Sequence Number
*			NØT INTERRUPTING	
*				
IRTN	DATA	**	LØCATIØN ØF WHERE INTERRUPT ØCCURED	
*			(12-BIT ADDRESS)	
	STA	SACC	SAVE ACCUMULATØR	
	TØA		SAVE ØVERFLØW BIT	
	STA	SØVFB	*	
	ITS		SAVE BANK INDICATØR AND P13	
	STA	BNKP13	*	
*				
*	DETER	MINE SØURCE ØF	INTERRUPT	
*				
	SKS	IPTRDR	IS IT PAPER TAPE READER	
	JMP	PTRDR	YES, GØ PRØCESS IT	
	SKS	IMTAPE	IS IT MAGNETIC TAPE	
	JMP	MAGTAP	YES, GØ PRØCESS IT	
	SKS	ICDRDR	IS IT THE CARD READER	
	JMP	CRDRDR	YES, GØ PRØCESS IT	
	SKS	ITTYPE	IS IT THE TELETYPE	
	JMP	TELTYP	YES, GØ PRØCESS IT	

Figure 6-11. Typical Interrupt Routine

CAP/SAP 12 CODING FORM

F2435-869

PROGRAMMER | EXT. | KEYPUNCH NOTES | DATE | PAGE 01

PROGRAM | CHARGE NUMBER

Location	Operation	Variables	Comments	Sequence Number
*	EXAMPLE ØF AN INTERRUPT RØUTINE HANDLER/EXECUTIVE			
*	WHICH RESIDES IN LØWER 4K (BANK ZERØ)			
*				
	ØRG	2		
	JST*	*+1	INTERRUPT LOCATIØN 0002	
	DATA	IRTN	RØUTINE WHICH PRØCESSES INTERRUPTS	
*				
*	HARDWARE PARAMETERS			
*				
*				
*		INA TYPES		
*				
IRDR	EQU	5	READ PAPER TAPE	
*				
*				
*		SKS TYPES		
*				
ICDR	EQU	7	CØNDITIØN CØDE FØR CARD READER-	
*			NØT INTERRUPTING	
IMTAPE	EQU	5	CØNDITIØN CØDE FØR MAGNETIC TAPE-	
*			NØT INTERRUPTING	
IPTRDR	EQU	4	CØNDITIØN CØDE FØR PAPER TAPE READER-	
*			NØT INTERRUPTING	
ITTYPE	EQU	3	CØNDITIØN CØDE FØR TELETYPE-	

Figure 6-11. Typical Interrupt Routine (Cont)

CAP/SAP 12 CODING FORM
12435-869

PROGRAMMER | EXT. | KEYPUNCH NOTES | DATE | PAGE 0
PROGRAM | CHANGE NUMBER

Location	Operation	Variables	Comments	Sequence Number
*	RESTORE AND RETURN TO		WHERE INTERRUPT OCCURRED	
*				
IEXIT	TOA		TURN OFF OVERFLOW BIT	
	LDA	SOVFB	SAVED OVERFLOW STATUS	
	LGR	1	OVERFLOW BIT RESTORED	
	LDA	BNKP13	RESTORE BANK INDICATOR AND P13	
	ITR		*	
	LDA	SACC	RESTORE ACCUMULATOR	
	ENB		ALLOW INTERRUPTS	
	JMP*	IRTN	RETURN TO WHERE INTERRUPT OCCURRED	
*				
*	PAPER TAPE READER I/O HANDLER			
*				
PTRDR				
	INA	IRDR	READ THE FRAME	
	{			
	}			
	JMP	IEXIT	PROCESSING COMPLETE	
*				

Figure 6-11. Typical Interrupt Routine (Cont)

CAP/SAP 12 CODING FORM

E2435-869

PROGRAMMER | EXT. | KEYPUNCH NOTES | DATE | PAGE

PROGRAM | CHARGE NUMBER

Location	Operation	Variables	Comments	Sequence Number
*$	MAIN	PROGRAM	FIRST CARD OF MAIN PROGRAM	
*				
	ORG	0	MAIN PROGRAM ORIGIN AT ZERO	
	JMP*	*+1	JUMP TO START OF MAIN PROGRAM	
	DATA	START	ADDRESS OF START	
	JST*	*+1	JUMP ON INTERRUPT	
	DATA	INTR	ADDRESS OF INTERRUPT ROUTINE	
*				
SUBR1	BSS	1	STORAGE FOR ADDRESS OF SUBROUTINE 1	
SUBR2	BSS	1	STORAGE FOR ADDRESS OF SUBROUTINE 2	
*				
	SETB	*,N	SPACE FOR SECTORIZED LINKS	
	BSS	N	*	
*				
	ORG	O'200	MAIN PROGRAM START	
START	EQU	*		
	JST*	SUBR1	CALL TO SUBROUTINE 1	
	JST*	SUBR2	CALL TO SUBROUTINE 2	

Figure 6-12. Programming Examples

CAP/SAP 12 CODING FORM

F2435-869

PROGRAMMER | EXT. | KEYPUNCH NOTES | DATE | PAGE 0

PROGRAM | CHARGE NUMBER

Location	Operation	Variables	Comments	Sequence Number
	(			
	END	NØ, START	END WITHØUT END FILE BLØCK	
*$SUBR1			FIRST CARD ØF SUBRØUTINE 1	
*				
	ØRG	4	SET TØ START ØF TRANSFER VECTØR	
SUBR1	DATA	START	DEFINE ADDRESS ØF THIS SUBRØUTINE	
SUBR2	BSS	1	STØRAGE FØR ADDRESS ØF SUBRØUTINE 2	
*				
	ØRG	Ø'1000	ØRIGIN ABØVE MAIN PRØGRAM	
START	EQU	*	START ØF SUBRØUTINE 1	
	(			
	JST*	SUBR2	CALL TØ SUBRØUTINE 2	
	)			
	END	NØ	END WITHØUT END FILE BLØCK	

Figure 6-12. Programming Examples (Cont)

CAP/SAP 12 CODING FORM

82435-869

PROGRAMMER | EXT. | KEYPUNCH NOTES | DATE | PAGE

PROGRAM | CHARG. NUMBER

Location	Operation	Variables	Comments	Sequence Number
*$SUBR2			FIRST CARD ØF SUBRØUTINE 2	
*				
	ØRG	4	SET TØ START ØF TRANSFER VECTØR	
SUBR1	BSS	1	STØRAGE FØR ADDRESS ØF SUBRØUTINE 1	
SUBR2	DATA	START	DEFINE ADDRESS ØF THIS SUBRØUTINE	
*				
	ØRG	Ø'2000	ØRIGIN ABØVE SUBRØUTINE 1	
START	EQU	*	START ØF SUBRØUTINE 2	
	END	YES	END WITH FILE BLØCK	

Figure 6-12. Programming Examples (Cont)

SECTION VII
OPERATING INSTRUCTIONS

GENERAL OPERATION

The normal operations required in programming the H112 controller are listed below:

1. Write source program -- see Sections II and IV through VI.
2. Punch the source program into cards or paper tape or generate magnetic tape.
3. Load the SAP-12 tape into an H316 or DDP-516 General Purpose Digital Computer (instructions provided in this section).
4. SAP-12 reads the source program into the H316 or DDP-516 and generates an object program tape, assembly printout, and post processor listing (if requested).
5. Read the loader program tape into the H112 Controller. The controller load mode may be initiated either with or without the use of the H112 control panel (described in this section).
6. Execute loader program which loads the assembled object program tape into the H112 Controller core memory.
7. The H112 controller program may now be exercised and the system may be considered operational. If it is desired to use the H112 Utility/Debug program for program checkout, etc., the following is required.
8. Read the Utility/Debug tape into the H112 Controller by executing the loader.
9. Communicate with the controller via the Utility/Debug program and ASR (I/O typewriter) (described fully in this section).

SOURCE PROGRAM ASSEMBLY

The first step in a source program assembly consists of loading the SAP-12 tape into the H316 or DDP-516. The source program tape or card deck is then read and assembled into the object program under the control of SAP-12. A punched object tape and printouts are produced. Detailed instructions follow:

1. Load the SAP-12 self-loading tape into memory via an ASR or high speed paper tape reader.
   a. Set the MA/SI/RUN switch to SI.
   b. Press MSTR CLEAR.
   c. Press REGISTER select button P/Y.
   d. Mount the SAP-12 assembler tape in the proper reader device.
   e. Set the switch register to 000001_8; the switch register is the set of 16 indicator/switches which correspond to the 16-bit computer word.
   f. Set the MA/SI/RUN switch to RUN.

g. Press START. The SAP-12 tape will load automatically and stop when completed. Remove and store the SAP-12 tape.

h. Install the source program into the appropriate reader device.

i. Set P/Y to 1000_8 - press START to initiate the first pass. The assembler will type "H112 ASSEMBLER SAP-12" on the I/O typewriter and begin reading the source program.

2. The first pass by the SAP-12 program is executed.

If any COPY statements are included in the source program, the COPY pseudo operation directs the assembler to copy symbolic statements from a symbolic punched paper tape. It is coded as follows in the source program:

LOCATION:	ignored
OPERATION:	COPY
VARIABLE:	asterisk or program name (1-6 letters)
COMMENTS:	ignored

If the variable field is an asterisk, the assembler stops immediately and requests the operator to load more paper tape by typing "MORE SOURCE" on the I/O typewriter. This option is used when a source language paper tape is in two or more pieces.

If the variable field contains a program name, the assembler stores the name in a list until the end of the current pass. At that time, it requests the operator to load a symbolic library paper tape, by typing "LOAD LIBRARY TAPE." The assembler reads that tape, copying subroutines whose names are in the list, ignoring all others. Missing subroutines will be noted on the log.

The subroutine is named by a special statement containing an asterisk in column 1, a dollar sign in column 2, and the name in columns 3-8. This statement must be the first statement in the subroutine.

At the end of pass 1, the SAP-12 assembler types "RELOAD SOURCE" on the ASR, requesting the operator to reload the source program card deck or tape. Then Sense Switch 1 must be flipped up and down.

If the input is on magnetic tape, or if an intermediate magnetic tape is used, the assembler proceeds from pass 1 to pass 2 without halt or comment. If the only I/O device available is an ASR, pass 2 must be executed twice: once to list; again to punch the object tape.

3. The second pass progresses and the object tape is punched by the I/O typewriter or punch.

If any COPY statements were included in the program, the assembler will, after all the original source is read in, ask the operator to mount and read in another tape as in pass 1. The assembly listing is automatically typed by the I/O typewriter or printer.

4. The post-processor cross-reference listing is typed by the ASR or printer immediately after the assembly listing if requested by the POST pseudo operation in the source program.

5. At the end of an assembly, SAP-12 types "END OF JOB" on the ASR. If another program is to be assembled, the operator loads the source and flips SS1 up and down.

OBJECT TAPE LOADING

H112 Paper Tape Loader Program

The H112 Paper Tape Loader is a program which loads assembled object program paper tapes into the H112 core memory. The loader program reads the object program tape, which is in the ASCII format, and formats 12-bit words for memory; the object program word block types control the type of loader operation desired. The processing of each block type is discussed in detail in this section. General loader operation is described below.

General Loader Operation. -- Each assembled block, on the object tape, consists of data block length, block type, data block (if any), checksum, carriage return, line feed, X-OFF, and a tape feed character. All printable characters are in the ASCII format; each character was previously added to 240_8 to produce a printable ASCII character. The format of object tape word blocks is shown in detail in Section IV.

The loader program subtracts 240_8 from each 8-bit object tape character. The result is a 6-bit character (two octal digits). Two such 6-bit characters are then combined to form a 12-bit word. The block type is read from word 1, examined for validity, and used to select the proper subroutine for processing the block. The data block length is read, examined for validity, and then stored in a counter. The proper subroutine, selected by the block type, is executed to process the data block. The checksum is then validated. The carriage return, line feed, X-OFF and tape feed characters are bypassed; they are not read into memory. At this point the processing of the block is complete and it is stored in the H112 core memory.

Detailed Loader Operation. -- Each block type and the resulting Paper Tape Loader program operation are listed below. Refer to Section IV for a detailed discussion of block types.

Block Type 0 - Header. -- The loader program does not load the header into the H112 core memory. If the I/O typewriter is available, the program name (header) is typed; otherwise, the header block is ignored.

Block Type 1 - Data Block. -- The loader program first reads the two words containing the 13-bit location for the first data word to be loaded. The data words are then sequentially loaded into memory starting at the location specified.

Block Type 2 - Fill Block. -- The loader program first reads the two words containing the 13-bit location for the first "fill" loading operation. The program then reads the number of words to be filled, starting at the specified location. The program then reads the fill value, ranging from 0000 to 7777_8, to be sequentially read into the core memory locations beginning at the specified address.

Block Type 3 - Transfer Block. -- The loader program reads the two words containing the program start address and loads it into memory.

Block Type 4 - End-Of-File Block. -- The loader program may perform one of two actions, based on whether or not a transfer block had been previously read. If a transfer block had been previously read, control is transferred to that location specified. If no transfer block had been read, the loader program halts. Another object tape may then be placed in the reader. Loading may resume by pressing the START button on the H112 control panel.

LOADER OPERATION

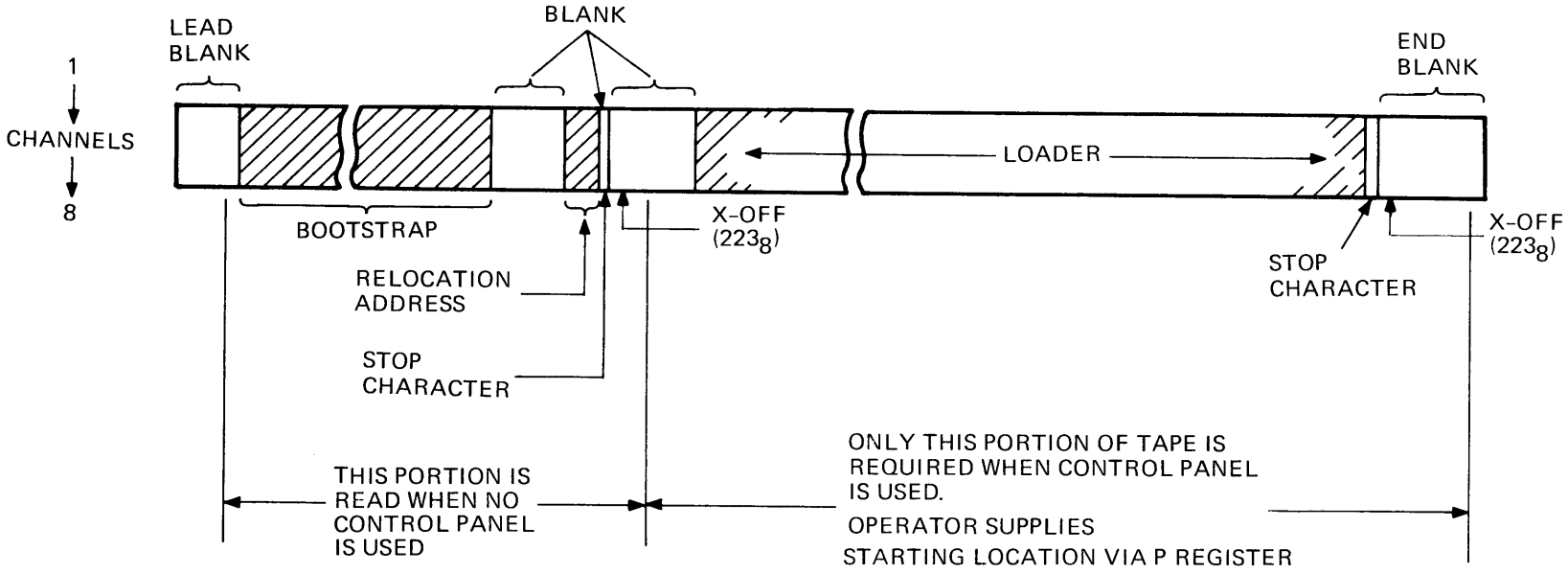
The loading process is under the control of the loader program which is punched on tape. Loading may occur with or without a control panel permanently installed; however, a portable panel providing basic machine control is available (model no. 112-19). Panel functions START, STOP, RUN, LOAD and MASTER CLEAR must be provided if the control panel is not on the machine.

The loader tape format is shown in Figure 7-1. The tape is punched with characters to be read as octal integers. In addition, control functions are provided. The H112 reads each tape character as shown in Figure 7-2 under the control of the bootstrap portion of the tape.

Loading With Control Panel

A description of Control Panel controls and indicators is provided in Section II. The Control Panel allows the operator to set the load mode starting address via the P register. Proceed as follows:

1. Mount tape at blank just prior to loader portion.
2. Press MASTER CLEAR.
3. Set LOAD switch to ON.
4. Press REGISTER select switch P.
5. Using the panel switch register, set the P register to the first location of the sector into which the loader is to be loaded; P register bits 1 through 7 must be 0's.
6. Press START. The loader will automatically be read and will stop at the stop character. If the I/O typewriter (ASR-33) is used as the loading device, the START/STOP/FREE lever must be moved to the START position. See Appendix F for detailed operation of the ASR in the load mode.
7. Press MASTER CLEAR.
8. Set LOAD switch to OFF.
9. Set RUN/STOP switch to RUN.
10. Set P register to the first location of the loader (same location established in step 5, above).
11. Mount the object tape to be loaded into the reader.



NOTES:

1. LOADER CHARACTERS = $0_8 - 7_8$ (ASCII CODES 260 - 267 OR 020_8 - 027_8)
2. LOAD MODE IGNORES BLANK TAPE. STOP IS BY STOP CHARACTER ONLY.
3. FOUR TAPE CHARACTERS = ONE 12-BIT MEMORY WORD.
4. RELOCATION ADDRESS IS TWO TAPE CHARACTERS FOLLOWED BY STOP CODE.
   ADDRESS BITS: TAPE CHARACTER 1 = BITS 1-3 OF SECTOR NUMBER;
   TAPE CHARACTER 2 = BITS 4-6 OF SECTOR NUMBER
5. A PUNCHED CHANNEL 7 INDICATES STOP CHARACTER (SUCH AS "H" -ASCII 310).

6206

Figure 7-1. Loader Tape Format

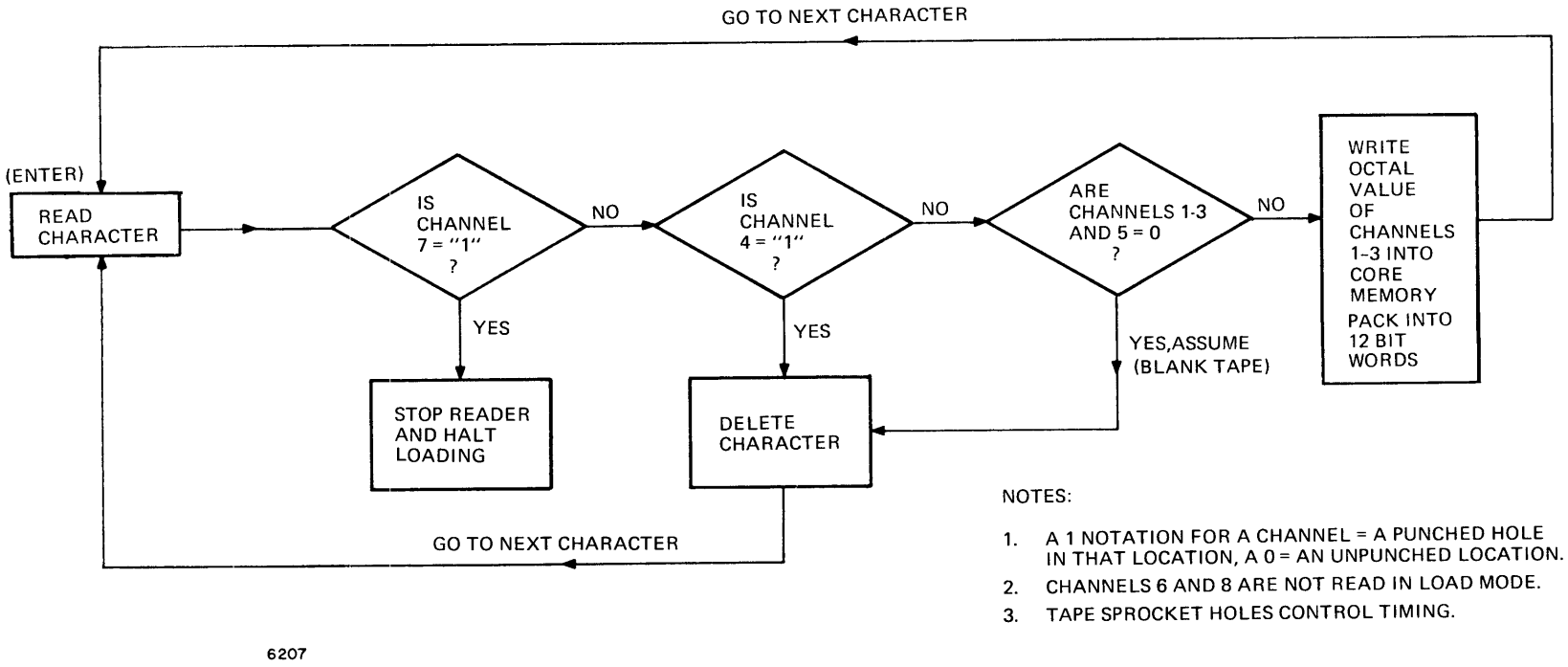


Figure 7-2. Bootstrap Program Control

12. Press START.

    a. If a high-speed paper tape reader is used, the object tape will automatically load and then stop.

    b. If the I/O typewriter paper tape reader is used, it is necessary to initiate tape reader operation by momentarily placing the function switch to the START position. The tape will stop at the X-OFF character.

13. Repeat steps 11 and 12 for additional object tapes.

14. To execute the loaded object tape programs, press START (assuming the object tape contains a transfer block).

Loading Without Control Panel

Loading without the control panel is made possible by the bootstrap loader portion of the H112 loader tape. The bootstrap loader occupies a section of core memory in sector zero, starting with core location zero. This is followed by the relocation address which contains the sector into which the loader is relocated by the bootstrap loader. The relocation address consists of two octal characters followed by the stop code. Note that the two octal characters for the relocation address appear in reverse order; tape character 1 is bits 1-3 of sector number; tape character 2 is bits 4-6 of sector number.

The loader may be loaded into sector zero or in another sector. A procedure for each is provided below.

Loader to be Resident in Sector Zero. --

1. Mount tape at blank between relocation address stop character and loader.
2. Turn on the tape reading device.
3. Press MASTER CLEAR.
4. Set the LOAD switch to ON.
5. Press START; the loader will be loaded. (If ASR, move start lever to START position.)
6. Press MASTER CLEAR.
7. Set the LOAD switch to OFF.
8. Set the RUN/STOP switch to RUN.
9. Mount the object tape to be loaded into the tape reading device.
10. Press START.

    a. If a high-speed paper tape reader is used, the object tape will automatically load and then stop.

    b. If the I/O typewriter paper tape reader is used, it is necessary to initiate tape reader operation by momentarily placing the function switch to the START position. The tape will stop at the X-OFF character.

11. Repeat steps 9 and 10 for additional tapes.

12. To execute loaded object programs, press START (assumming the object tape contains a transfer block).

Loader to be Resident in Sector Other than Zero. --

1. Mount the tape at the lead blank position.
2. Turn on the tape reading device.
3. Press MASTER CLEAR.
4. Set LOAD switch to ON. (Move START lever on ASR to the START position.)
5. Press START: the bootstrap and relocation address portion of the tape will be loaded.
6. Press MASTER CLEAR.
7. Set the LOAD switch to OFF.
8. Set the RUN/STOP switch to RUN.
9. Press START: the loader will be loaded.
10. Mount the object tape to be loaded into the tape reading device.
11. Press START.
    a. If a high-speed paper tape reader is used, the object tape will automatically load and then stop.
    b. If the I/O typewriter paper tape reader is used, it is necessary to initiate tape reader operation by momentarily placing the function switch to the START position. The tape will stop at the X-OFF character.
12. Repeat steps 10 and 11 for additional object tapes.
13. To execute the loaded object tape programs, press START (assuming the object tape contains a transfer block).

NOTE

The relocation address for the loader may be changed. First, prepare a duplicate tape. The relocation address may then be replaced with a new section of tape with the desired address. The relocation address may also be changed by locating the two relocation characters, on the reproduced loader tape, and by repunching those two characters with "8" on the I/O typewriter keyboard; the coded 8 causes a punched hole in channel 4, indicating "delete". Repunch the desired characters in the blank space preceding the "delete" characters. Take care that the two characters are punched in the proper order (bits 1-3 first, then bits 4-6).

H112 DEBUG UTILITY

The H112 Debug Utility program is a programming aid which allows the programmer to communicate with the H112 central processor during the program checkout process. Program functions include 11 basic operator commands and program responses. Program responses generally include memory dump operations via I/O typewriter typed or punched outputs, access to memory for program changes, and entry of data via I/O typewriter or tape reader.

General Operation

The general operation described in this section occurs after the H112 Debug Utility tape has been loaded into the central processor. The program initiates operator action by typing a carriage return, a line feed, and a question mark; this indicates that the program is awaiting an operator command. Each operator command consists of a single mnemonic alphabetic character. Most commands are then followed by one or more octal parameters. The operator command is normally terminated by a carriage return; however, in the event that a typing error is made, the operator types a slash mark. The program then types a carriage return, line feed, and question mark and is ready for a new command and the command previously typed is ignored. When command is terminated by a carriage return, the program executes the command. When program execution has been completed, the program types the carriage return, line feed, and question mark. A new command may then be entered.

A list of commands and program response is provided below:

A = Access a memory location
B = Breakpoint set
C = Compare to memory
D = Dump core in octal
E = Enter into memory
G = Go after breakpoint
J = Jump to
M = Mnemonic core dump
P = Punch memory
R = Reproduce memory
S = Search memory

The Debug Utility program may reside in any part of memory. Two versions of the Debug Utility are provided:

Minimum version — allows the use of the A, D, J and P commands only.

Full version — allows the use of all commands.

Detailed Operation

The following paragraphs contain the required operator command entries and the resulting program response. General operation previously described applies to all program functions following.

Dump Core in Octal on I/O Typewriter. --

Operator Command:

Daaaaa, bbbbb (CR)

aaaaa and bbbbb are addresses of from one to five octal digits. If bbbbb is omitted, it is assumed equal to aaaaa.

Program Response:

A listing of core from location aaaaa through bbbbb is produced on the I/O typewriter. The listing is printed with eight locations on a line, with each line preceded by the location of

the first word on that line. The operator may terminate the core dump operation by pressing the BREAK key on the I/O typewriter.

Mnemonic Core Dump on I/O Typewriter. --

Operator Command:

Maaaaa, bbbbb (CR)

aaaaa and bbbbb are addresses of from one to five octal digits. If bbbbb is omitted, it is assumed equal to aaaaa.

Program Response:

A listing of core from location aaaaa through bbbbb is typed by the I/O typewriter. The listing is typed with one location on each line.

Each line will consist of:

a. The location (five octal digits).

b. The mnemonic operation code (three alphabetic characters).

c. An asterisk if the indirect bit is set.

d. The contents of the location (four octal digits).

The operator may terminate the core dump by pressing the BREAK key on the I/O typewriter.

Access a Memory Location. --

Operator Command:

Aaaaaa (CR)

aaaaa is an address of from one to five octal digits.

Program Response:

The program types the location contents of the location (octal), the instruction mnemonic†, * (asterisk)† and waits for an operator response.

Operator Response:

The operator may type one of the following:

a. cccc (CR), which changes the contents of the location to cccc; the program then types the next location, its contents and waits for the operator response.

b.† aaa*ccc(CR), which changes the contents of the location to the instruction specified by mnemonic aaa, indirect address indicator *, and address ccc; the program then types the next location, its contents and waits for the operator response.

NOTE

† This option is not available in the minimum version of the Debug Utility.

c. Carriage return, which will cause the program to advance to the next location without changing the contents of the current location.

d. Slash, which will cause the program to exit from the access routine and wait for the next command.

Enter Into Memory. --

Operator Command:

Eaaaaa, bbbbb, xxxx (CR)

aaaaa and bbbbb are addresses of from one to five octal digits; xxxx is a number of from one to four octal digits; if bbbbb is omitted, it is assumed equal to aaaaa; if xxxx is omitted, it is assumed equal to zero.

Program Response:

All memory cells from aaaaa through bbbbb are set to xxxx.

Punch Memory. --

Operator Command:

Paaaaa, bbbbb (CR)

aaaaa and bbbbb are addresses of from one to five octal digits; if bbbbb is omitted, it is assumed equal to aaaaa.

Program Response:

The contents of memory from location aaaaa through bbbbb are punched on the I/O typewriter in the object format. See the object tape format described in Section IV.

The operator may terminate the memory dump by pressing the BREAK key on the I/O typewriter.

Compare to Memory. --

Operator Command:

Cx (CR)

x is 0 for I/O typewriter, 1 for high speed paper tape reader.

Program Response:

An object tape (or one punched by a "P" command) is compared with the memory locations to which it corresponds and any differences are printed on the I/O typewriter. The address of the memory location, the contents of the tape, and the current contents of memory are printed.

Breakpoint Set. --

Operator Command:

Baaaaa (CR)

aaaaa is an address of from one to five octal digits.

Program Response:

A breakpoint is set at location aaaaa. When the program to be executed reaches location aaaaa, it jumps into the Debug routine. The I/O typewriter prints the location and the contents of the A register, and waits for another command. The instruction at location aaaaa is not executed. A command of B0 cancels any outstanding breakpoints.

Go After Breakpoint. --

Operator Command:

Gxxxx (CR)

xxxx is a number of from one to four octal digits.

Program Response:

Execution is continued at the location of the breakpoint with xxxx in the A register; if xxxx is omitted, the previous contents of the A-register are restored.

Jump to Location. --

Operator Command:

Jaaaaa, xxxx (CR)

aaaaa is an address of from one to five octal digits; xxxx is a number of from one to four octal digits; if either is omitted, 0 is assumed.

Program Response:

xxxx is placed in the A register and execution begins at aaaaa.

Search Memory. --

Operator Command:

Saaaaa, bbbbb, xxxx, yyyy (CR)

aaaaa and bbbbb are addresses of from one to five octal digits; xxxx is a number of from one to four octal digits; yyyy is a mask.

Program Response:

All memory locations from location aaaaa through location bbbbb are compared to xxxx; both numbers are masked by (ANDed with) yyyy before the compare; all locations which match are printed on the I/O typewriter; the operator may terminate the operation by pressing the BREAK key on the I/O typewriter.

Reproduce Memory. --

Operator Command:

Raaaaa, bbbbb, ccccc (CR)

aaaaa, bbbbb, and ccccc are addresses of from one to five octal digits.

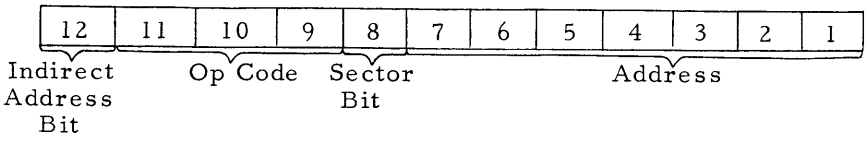
Program Response:

Memory locations from aaaaa through bbbbb are copied into corresponding locations starting with ccccc.

APPENDIX A
MACHINE INSTRUCTION CODES

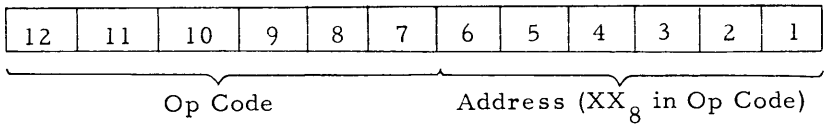
The H112 controller instruction codes are shown in this appendix. Observe the word format shown for each of the five instruction types; the word format shown identifies the op code bit locations within the 12-bit type. Execution time is given in microseconds and quarter cycles. Each quarter cycle is 423.7 ns ±0.1%.

Op Code			Execution Time (μs)	Quarter Cycles
Octal	Mnemonic	Instruction		
Memory Reference Instructions:				



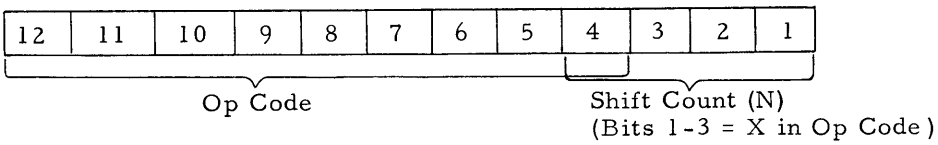
1	LDA	Load A (Direct)	3.39	8
	LDA*	Load A (Indirect)	5.09	12
2	STA	Store A (Direct)	3.39	8
	STA*	Store A (Indirect)	5.09	12
3	JMP	Jump (Direct)	3.39	8
	JMP*	Jump (Indirect)	5.09	12
4	ADD	Add (Direct)	7.63	18
	ADD*	Add (Indirect	9.33	22
5	ANA	AND A (Direct)	7.63	18
	ANA*	AND A (Indirect)	9.33	22
6	IRS	Increment, Replace and Skip (Direct)	9.33	22
	IRS*	Increment, Replace and Skip (Indirect)	11.02	26
7	JST	Jump and Store P (Direct)	4.66	11
	JST*	Jump and Store P (Indirect)	6.36	15

Input/Output Instructions:



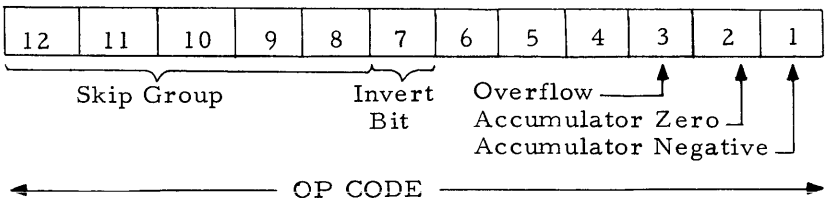
40XX	INA	Input Transfer to Accumulator	4.66	11
41XX	SKS	Skip If Set	4.66	11
42XX	OTA	Output Transfer From Accumulator	4.66	11
43XX	OCP	Output Control Pulse	4.66	11
4300 4301	SMK	Set Mask	4.66	11

Shift Instructions:



010X 011X	LGR	Logical Right Shift	3.8+N(0.424)	9+N
012X 013X	ARS	Arithmetic Right Shift	3.8+N(0.424)	9+N
014X 015X	RAR	Rotate A Right	3.8+N(0.424)	9+N

Skip Instructions:

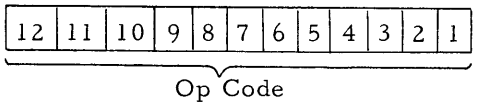


0200	NOP	No Operation	3.39	8
0201	SMI	Skip If Accumulator is Minus	3.39	8
0202	SZE	Skip If Accumulator is Zero	3.39	8
0203	SMZ	Skip If Minus or Zero	3.39	8
0204	SOV	Skip If Overflow Flop is Set	3.39	8
0300	SKP	Skip Unconditionally	3.39	8
0301	SPL	Skip If Accumulator is Plus	3.39	8

Skip Instructions (Cont):

0302	SNZ	Skip If Accumulator is Non-Zero	3.39	8
0303	SPN	Skip If Positive and Non-Zero	3.39	8
0304	SNO	Skip On No Overflow	3.39	8

Generic Instructions:



0000	HLT	Halt	2.54 + Wait	6+ Wait
0003	OCA	One's Complement Accumulator	7.63	18
0005	TCA	Two's Complement Accumulator	7.63	18
0021	STL	Stall On Line	3.39 + Wait	8+ Wait
0022	TBA	Transfer Bank Register to Accumulator	3.39	8
0024	ITS	Interrupt Save	3.39	8
0030	TOA	Transfer Overflow to Accumulator	3.39	8
0041	TAB	Transfer Accumulator to Bank Register	2.54	6
0042	ITR	Interrupt Return	2.54	6
0044	ENB	Enable Interrupts	2.54	6
0050	INH	Inhibit Interrupts	2.54	6
0060	CRA	Clear Accumulator	2.54	6

APPENDIX B
LOAD MODE CODES

Octal Bit Configuration In Core Memory	Paper Tape Format Channel No.									ASCII Character	I/O Typewriter Key To Produce ASCII Character
	8	7	6	5	4		3	2	1		
0	●	○	●	●	○	:	○	○	○	260	0
1	●	○	●	●	○	↑	○	○	●	261	1
2	●	○	●	●	○	(Sprocket Holes)	○	●	○	262	2
3	●	○	●	●	○		○	●	●	263	3
4	●	○	●	●	○		●	○	○	264	4
5	●	○	●	●	○		●	○	●	265	5
6	●	○	●	●	○		●	●	○	266	6
7	●	○	●	●	○	↓	●	●	●	267	7
(Blank)	○	○	○	○	○	:	○	○	○	000	"Here Is"
(Delete)	●	○	●	●	●	:	○	○	○	270	8
(Halt)	●	●	○	○	●	:	○	○	○	310	H

NOTE:

In Paper Tape Format,

○ = Unpunched location (0)

● = Punched location (1)

APPENDIX C
INTERNAL 6 BIT CODE

The internal 6 bit code is related to ASCII code and is used for the H112 object tape. ASCII is compatible with teletype ASR-33 and ASR-35 typing equipment. The table below shows the ASCII codes and associated characters which they represent. The internal 6 bit code allows packing of two characters per H112 memory word. An alphanumeric string in a DATA statement produces two 6 bit characters per word in this internal set. The internal 6 bit code, the ASCII code, and the corresponding character are listed below.

Character	Internal 6 Bit	ASCII
Space	00	240
!	01	241
" (quote)	02	242
#	03	243
$	04	244
%	05	245
&	06	246
' (apostrophe)	07	247
(	10	250
)	11	251
*	12	252
+	13	253
,	14	254
-	15	255
.	16	256
/	17	257
0	20	260
1	21	261
2	22	262
3	23	263
4	24	264
5	25	265
6	26	266
7	27	267
8	30	270
9	31	271

Character	Internal 6 Bit	ASCII
:	32	272
;	33	273
<	34	274
=	35	275
>	36	276
?	37	277
@	40	300
A	41	301
B	42	302
C	43	303
D	44	304
E	45	305
F	46	306
G	47	307
H	50	310
I	51	311
J	52	312
K	53	313
L	54	314
M	55	315
N	56	316
O	57	317
P	60	320
Q	61	321
R	62	322
S	63	323
T	64	324
U	65	325
V	66	326
W	67	327
X	70	330
Y	71	331
Z	72	332
Line Feed	73	212
Carriage Return	74	215
X-ON	75	221
X-OFF	76	223
Rubout	77	377

APPENDIX D
INSTRUCTION WORD IDENTIFICATION
(BY OCTAL WORD FORMAT)

Octal Word Format	Op Code Mnemonic	Instruction Type
0000	HLT	Generic
0003	OCA	Generic
0005	TCA	Generic
0021	STL	Generic
0022	TBA	Generic
0024	ITS	Generic
0030	TOA	Generic
0041	TAB	Generic
0042	ITR	Generic
0044	ENB	Generic
0050	INH	Generic
0060	CRA	Generic
010X 011X	LGR	Shift
012X 013X	ARS	Shift
014X 015X	RAR	Shift
0200	NOP	Skip
0201	SMI	Skip
0202	SZE	Skip
0203	SMZ	Skip
0204	SOV	Skip
0300	SKP	Skip
0301	SPL	Skip
0302	SNZ	Skip
0303	SPN	Skip
0304	SNO	Skip
04XX 05XX 06XX 07XX	LDA	Memory Reference (Direct)

Octal Word Format	Op Code Mnemonic	Instruction Type
10XX 11XX 12XX 13XX	STA	Memory Reference (Direct)
14XX 15XX 16XX 17XX	JMP	Memory Reference (Direct)
20XX 21XX 22XX 23XX	ADD	Memory Reference (Direct)
24XX 25XX 26XX 27XX	ANA	Memory Reference (Direct)
30XX 31XX 32XX 33XX	IRS	Memory Reference (Direct)
34XX 35XX 36XX 37XX	JST	Memory Reference (Direct)
40XX	INA	Input/Output
41XX	SKS	Input/Output
42XX	OTA	Input/Output
43XX (See Note 3)	OCP	Input/Output
4300 4301	SMK	Input/Output
44XX 45XX 46XX 47XX	LDA*	Memory Reference (Indirect)
50XX 51XX 52XX 53XX	STA*	Memory Reference (Indirect)
54XX 55XX 56XX 57XX	JMP*	Memory Reference (Indirect)
60XX 61XX 62XX 63XX	ADD*	Memory Reference (Indirect)
64XX 65XX 66XX 67XX	ANA*	Memory Reference (Indirect)

Octal Word Format	Op Code Mnemonic	Instruction Type
70XX 71XX 72XX 73XX	IRS*	Memory Reference (Indirect)
74XX 75XX 76XX 77XX	JST*	Memory Reference (Indirect)

NOTES:

1. "X" indicates an octal digit.
2. See Appendix A for description and timing.
3. OCP word format cannot be 4300 or 4301.
4. First octal word format shown for memory reference instructions is the primary word format for the associated instruction code. The three additional word formats shown result from sector bit and most significant address bit combinations. (See memory reference instruction format in Appendix A.)

APPENDIX E
INSTRUCTION WORD FORMAT LIST
(BY OP CODE MNEMONIC)

Op Code Mnemonic	Octal Word Format	Instruction Type
ADD	20XX 21XX 22XX 23XX	Memory Reference (Direct)
ADD*	60XX 61XX 62XX 63XX	Memory Reference (Indirect)
ANA	24XX 25XX 26XX 27XX	Memory Reference (Direct)
ANA*	64XX 65XX 66XX 67XX	Memory Reference (Indirect)
ARS	012X 013X	Shift
CRA	0060	Generic
ENB	0044	Generic
HLT	0000	Generic
INA	40XX	Input/Output
INH	0050	Generic
IRS	30XX 31XX 32XX 33XX	Memory Reference (Direct)
IRS*	70XX 71XX 72XX 73XX	Memory Reference (Indirect)
ITR	0042	Generic
ITS	0024	Generic
JMP	14XX 15XX 16XX 17XX	Memory Reference (Direct)
JMP*	54XX 55XX 56XX 57XX	Memory Reference (Indirect)

Op Code Mnemonic	Octal Word Format	Instruction Type
JST	34XX 35XX 36XX 37XX	Memory Reference (Direct)
JST*	74XX 75XX 76XX 77XX	Memory Reference (Indirect)
LDA	04XX 05XX 06XX 07XX	Memory Reference (Direct)
LDA*	44XX 45XX 46XX 47XX	Memory Reference (Indirect)
LGR	010X 011X	Shift
NOP	0200	Skip
OCA	0003	Generic
OCP	43XX (See Note 3)	Input/Output
OTA	42XX	Input/Output
RAR	014X 015X	Shift
SKP	0300	Skip
SKS	41XX	Input/Output
SMI	0201	Skip
SMK	4300 4301	Input/Output
SMZ	0203	Skip
SNO	0304	Skip
SNZ	0302	Skip
SOV	0204	Skip
SPL	0301	Skip
SPN	0303	Skip
STA	10XX 11XX 12XX 13XX	Memory Reference (Direct)
STA*	50XX 51XX 52XX 53XX	Memory Reference (Indirect)

Op Code Mnemonic	Octal Word Format	Instruction Type
STL	0021	Generic
SZE	0202	Skip
TAB	0041	Generic
TBA	0022	Generic
TCA	0005	Generic
TOA	0030	Generic

NOTES:

1. "X" indicates an octal digit.
2. See Appendix A for description and timing.
3. OCP word format cannot be 4300 or 4301.
4. First octal word format shown for memory reference instructions is the primary word format for the associated instruction code. The three additional word formats shown result from sector bit and most significant address bit combinations. (See memory reference instruction format in Appendix A.)

APPENDIX F
(MODEL 112-25) I/O TYPEWRITER (ASR-33)

The ASR-33 Teletype Unit is the basic I/O device for the H112 controller. It is a versatile device that receives data from the computer at a 10-cps rate for printing on a roll of paper or punching on a paper tape punch. It also transmits data from a paper tape reader or from the keyboard to the computer at a maximum rate of 10 cps. In the LOCAL mode it may be used off-line for paper tape generation, paper tape reproduction, and printer listing.

KEYBOARD AND CARRIAGE FEATURES

The ASR-33 keyboard is similar to that of a standard typewriter. The keyboard contains four rows of keys that generate an 8-bit internal code (see Figure F-1 and Table F-1). Letters and numerals are transmitted without use of the shift key. All letters are printed as capitals. The shift key is only used for special punctuation marks, and symbols which correspond to upper case positions on certain typewriters. Control functions, such as X-ON (which turns on the paper tape reader), X-OFF (which turns off the paper tape reader), and BELL, are generated by holding down the control key (CTRL) while the particular function key is being pressed (Q, S, and G keys, respectively, in the examples given above).

The unit can print up to 72 characters per line. A line feed and a carriage return must be executed after the last character to be printed in each line. The keyboard is mechanically interlocked (for all keys except SHIFT, CTRL, and REPT) to prevent more than one key from being depressed at a time. The keyboard does not lock in the upper case position, thus the operator must hold the SHIFT key depressed to produce the special upper case characters.

ON-LINE OPERATING MODES

There are three basic modes of operation when the I/O typewriter is on-line: input, output, and load. The control logic automatically resets to the input mode at the end of each output transfer operation and when a master clear is performed. To enter the output mode it is only necessary to give the OTA associated with the output transfer to this device. Load mode is detailed later.

Input Mode

The input mode is used to transfer information from the reader or keyboard of the ASR-33 to the controller. Printed copy is always produced if the 8-bit character is printable. If it is a control character, a specific control function is performed. There are 256 possible 8-bit characters that can be read by the reader and transferred to the controller. When the X-OFF character is read, the reader stops after reading the character following the X-OFF

unless that character is the X-ON. The control logic will always be in the input mode except during the time a character is being transferred out. When the character has been transferred to the ASR-33, the control logic automatically reverts to the input mode.

Output Mode

The output mode is used to transfer information from the controller to the printer and/or punch of the ASR-33. Printed copy is always produced if the 8-bit character is printable. If it is a control character, a specific control function is performed. Any of the 256 possible 8-bit characters will be punched, if the punch is turned on, though certain characters will cause a control action by the ASR-33 which affects reception of additional characters (i.e., X-ON starts the paper tape reader and WRU triggers the answer-back drum). Control of the punch is manual only.

Character Format

The character format is exclusively 8-bit ASCII (see Table F-1 for codes). If the internal 6 bit code is to be transferred to the I/O typewriter, a code translation to ASCII must be made.

Load Mode

The load mode is used to transfer information from the ASR-33 to the controller without the use of a software program. The load function is controlled by hardware in the controller and is initiated by pressing the LOAD pushbutton followed by the START pushbutton on the control panel and moving the motion control lever on the reader housing assembly of the ASR-33 to its START position.

Transfers are made using a 3-bit character. Four such characters are packed into a 12-bit memory location of the controller with the first digit in bits 1-3, the second in bits 4-6, the third in bits 7-9, and the fourth in bits 10-12. The next four 3-bit characters on the tape are packed into the following memory location, etc., until all characters on the tape have been read. The loading of characters terminates when the controller receives a character with a 1 in channel 7. If the reader on the ASR-33 is to stop at this point, the next character on the tape must be X-OFF. If there is no X-OFF on the tape, the reader continues to slew tape until the end of the tape has been reached. (See Figure F-2 for load mode format.)

The load mode is especially useful when the memory does not contain a loader, or when the only available panel is the remote operators panel. The load lever on the high speed paper tape reader must be up while the ASR-33 is operated in the load mode.

The load mode may also be used with the keyboard, however, the octal representation of the instruction must be typed in reverse order, e.g. to load 3126 into a location, 6213 is typed.

OPERATION

Off-line operation includes the following combination of functions:

a. Keyboard to printer (normal typewriter operation)

b. Keyboard to printer and punch

c. Reader to printer

d. Reader to printer and punch

During on-line operation, as in off-line, the printer always functions whenever the reader or punch is being used and printable characters are being transferred.

Punch Control

The punch is controlled by manually operating the punch ON and OFF pushbuttons on the punch housing. With the ON button depressed, any input to or output from the ASR-33 causes tape to be punched. Tape leader can be generated by depressing the HERE IS key on the keyboard. This causes generation of a burst of 20 blank characters. Extra leader can be generated by repeated operation of this key. The tape can be back-spaced one character position at a time by depressing the B.SP. pushbutton. Tape insertion in and removal from the punch mechanism is facilitated by depressing the REL. pushbutton.

Reader Control

The reader is controlled either manually or through the program. When under manual control, the reader is operated with the START/STOP/FREE lever on the reader housing. The first character to be read when the lever is moved from the STOP to the START position is the one initially placed over the read pins. The FREE position of the lever is used to allow tape to be drawn freely through the reader mechanism.

The reader is started under program control by first outputting an X-ON character and waiting for the reader to begin reading tape. The reader stops (both under program and manual control) upon recognition of an X-OFF character. The X-OFF character and the character immediately following are transmitted to the interface buffer before the reader stops. The reader stops during both manual and programmed operation if the manual control lever is moved to the STOP position or if the reader runs out of tape before processing an X-OFF character. A reader which has stopped because it has read an X-OFF character can be started again simply by moving the manual control lever to its START position or outputting an X-ON character.

PROGRAMMING

The instructions assigned to the I/O typewriter are as follows:

SKS 03 — Skip If I/O Typewriter NOT Interrupting

This instruction determines whether the I/O typewriter has caused an interrupt on the standard interrupt line.

SKS 01 — Skip If I/O Typewriter Ready

This instruction determines whether the I/O typewriter interface buffer is ready to transfer a new character to the controller. The device is ready as soon as the last bit in the character has been received in the interface buffer from the ASR-33 and remains ready until the character has been transferred to the controller. This instruction is used in connection with input transfers.

SKS 02 — Skip If I/O Typewriter NOT Busy

This instruction determines whether the I/O typewriter interface buffer is in the process of transferring a character to or receiving a character from the ASR-33. In the input mode, the device is busy from the time the first bit of the character enters the interface buffer until the time the entire character is ready for transfer to the controller. In the output mode the device is busy from the time a character has been transferred from the controller to the interface buffer until the time the last bit in the character has been received by the ASR-33. This test is used in connection with output transfers.

INA 01 — Input a Character From I/O Typewriter, If Device Is Ready

This instruction transfers an 8 bit character from the I/O typewriter interface buffer to the least significant 8-bits of the A register in the controller and skips the next instruction in the program if the device is ready. The transfer must take place within 4 ms after the device has become ready or the interrupt has occurred. Completion of the instruction will reset the ready flip-flop and the interrupt. The A register is cleared before the transfer takes place. If the device is not ready, the A register is cleared, no transfer takes place and the next instruction in the program is executed. (See Figure F-1 for input format.)

OTA 02 — Output a Character To I/O Typewriter If Device Is NOT Busy

This instruction transfers the least significant 8 bits of the A register in the controller to the I/O typewriter interface buffer and skips the next instruction in the program if the device is NOT busy. Completion of this instruction will reset the interrupt. If the device is busy, no transfer takes place and the next instruction in the program is executed. (See Figure F-1 for output format.)

OCP 03 — Reset I/O Typewriter Interrupt

This instruction resets the I/O typewriter interrupt request logic if no transfer action is required as a result of the interrupt. The interrupt logic is normally reset by the appropriate input or output instruction.

SMK 00 — Set Standard Interrupt Mask

This instruction sets the standard interrupt mask F/F in the I/O typewriter interface if the accumulator bit 4 is a 1, and resets it if the accumulator bit 4 is a 0. There is no skip test associated with this instruction and the next instruction in the program is always executed. Masking off an interrupt will not reset the interrupt F/F.

Program Examples

The first example shows coding which may be used for inputting a character under program control:

```
--------
INA 01      Input character from buffer, if ready.
JMP *-1     Delay until ready.
--------
```

Since the device is always reset to the input mode at the end of any operation, no mode setting command is necessary. If it is not desirable to wait in an input loop for a period of time, a test instruction may precede the INA instruction. Failure of the test could cause a jump to another part of the program which could be executed prior to trying the test again.

Under interrupt control the program would automatically enter a subroutine with the above two instructions contained in it, immediately input the character and exit from the subroutine without delay. The INA resets the interrupt at the end of the transfer.

If the input is to be immediately followed by an output command, the built-in test will inhibit outputting until the device becomes NOT busy. A delay is built into the interface which holds the busy line true for such time as it takes to determine whether a new character is coming in from the ASR-33. This delay is at most 15 ms.

The second example shows coding which may be used for outputting a character under program control:

```
OTA 02      Output character to I/O typewriter if not busy
JMP *-1     If busy, try again
```

Because the OTA 02 will detect a busy condition, the controller may wait in the OTA-JMP loop. If the interface is busy transferring the previous character to the ASR-33, the transfer will take place when the previous transfer is completed. The interface may also be busy due to a transfer from the ASR-33 to the interface. This may occur if break key or other keys are operated. The controller will then wait in the OTA-JMP loop for the duration of the operation of the key. The SKS 02 may be used to detect the busy condition and other codes may be executed while waiting for the ASR-33 to be not busy.

Under interrupt control, the program would automatically enter a subroutine with the above loop contained in it. If the device is not busy, the output transfer from the controller will occur and the OTA will reset the interrupt.

The device automatically reverts to the input mode when an output transfer has been completed (i.e., a character has been sent to the ASR-33) and a delay is built into the interface to hold the busy line true for such time as it takes to make sure that the ASR-33 has completely received and processed the character. This delay is at most 10 ms. When the interrupt is used it always occurs at the end of this delay.

When the program has sent its last character to the interface buffer, a final interrupt occurs when the transfer has been completed. This must be reset using the OCP 03 command. Normally the INA 01 and the OTA 02 reset the interrupts.

In some cases it is desirable that the operator be able to terminate a lengthy output by use of the BREAK key. To accomplish this, the following coding may be used:

```
----------------------
OUTP LDA DATA OTA 02     Output character to I/O typewriter, if not busy
             JMP INPT    Try inputting if device is ready
----------------------
INPT INA 01              Input character if ready
   JMP OUTP              Try outputting if device is not ready
----------------------
```

The interface continues in the output mode, as usual, until the last bit of the character it was attempting to output has been sent and it has returned to the input mode. At this instant the continuous spacing (i.e., break in the line) enters the interface and appears as an incoming character. By using the loop described above the interface accepts the all zero character, sets READY or initiates an interrupt when the entire character has been received, and waits for the character to be transferred to the controller. When the transfer is completed, the program is then able to test for an all zero character and refrain from outputting the next character if the test is true. Several such characters may enter and be tested in this manner before the break key is released. When the break key is released an indeterminate character is produced.

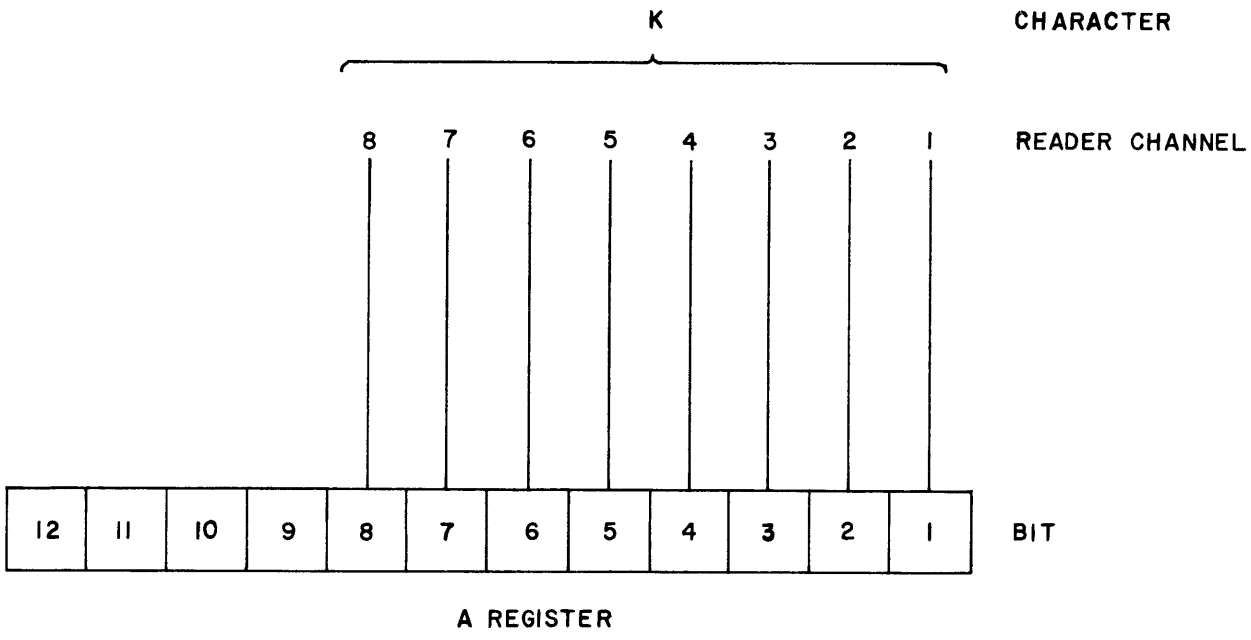


Figure F-1. Bit Assignments During Programmed Input Transfers

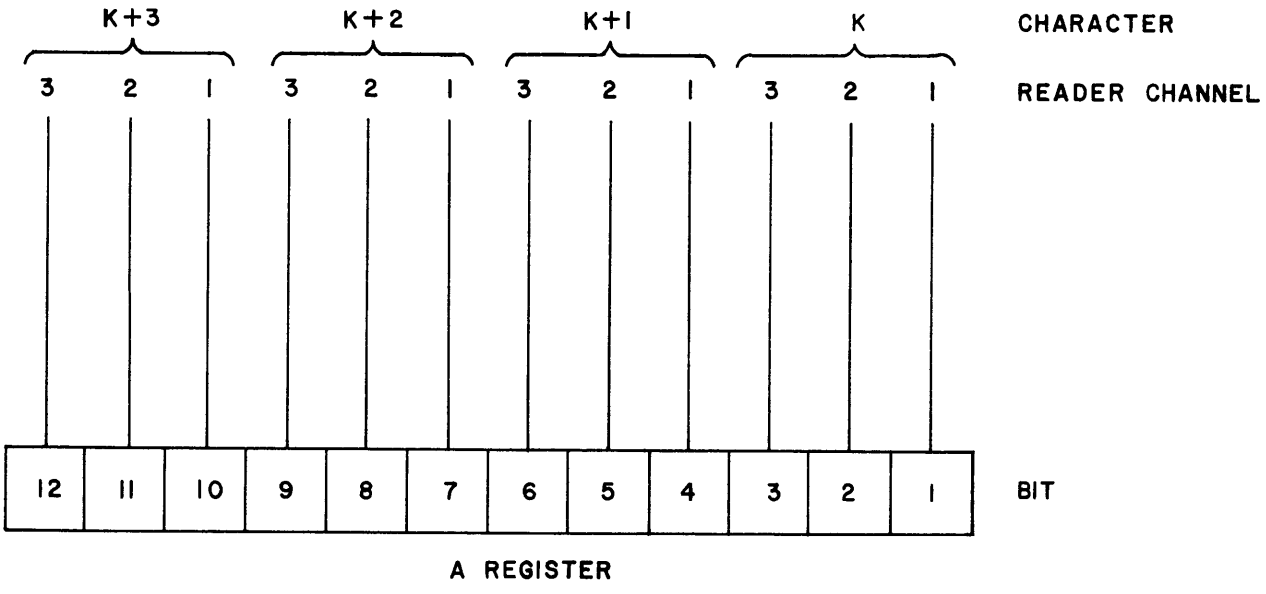


Figure F-2. Bit Assignments During Load Mode Input Transfers

Table F-1.
ASR-33 Character and Symbol Codes

Code	Character Printed	Code	Character Printed	Code	Character Printed	Code	Character Printed
000	Null*	040	Space	100	@	140	@
001	Null	041	!	101	A	141	A
002	Null	042	"	102	B	142	B
003	Null	043	#	103	C	143	C
004	Null	044	$	104	D	144	D
005	WRU	045	%	105	E	145	E
006	Null	046	&	106	F	146	F
007	BELL	047	'	107	G	147	G
010	Null	050	(	110	H	150	H
011	Null	051	)	111	I	151	I
012	LF	052	*	112	J	152	J
013	Null	053	+	113	K	153	K
014	Null	054	,	114	L	154	L
015	Null	055	-	115	M	155	M
016	Null	056	.	116	N	156	N
017	Null	057	/	117	O	157	O
020	Null	060	0	120	P	160	P
021	Null	061	1	121	Q	161	Q
022	Null	062	2	122	R	162	R
023	Null	063	3	123	S	163	S
024	Null	064	4	124	T	164	T
025	Null	065	5	125	U	165	U
026	Null	066	6	126	V	166	V
027	Null	067	7	127	W	167	W
030	Null	070	8	130	X	170	X
031	Null	071	9	131	Y	171	Y
032	Null	072	:	132	Z	172	Z
033	Null	073	;	133	[	173	[
034	Null	074	<	134	\	174	Null
035	Null	075	=	135	]	175	Null
036	Null	076	>	136	↑	176	Null
037	Null	077	?	137	←	177	Null

*Whenever the HERE-IS key is depressed, the answer-back drum is activated, producing a burst of 20 characters of all zeros.

Table F-1. (Cont)
ASR-33 Character and Symbol Codes

Code	Character Printed	Key Depressed		
		Lower Case	Simult. Control	Simult. Shift and Control
200	Null			P
201	Null		A	
202	Null		B	
203	Null		C	
204	Null		D	
205	WRU		E	
206	Null		F	
207	BELL		G	
210	Null		H	
211	Null		I	
212	LF	LF	J	
213	Null		K	
214	Null		L	
215	CR	CR	M	
216	Null		N	
217	Null		O	
220	Null		P	
221	X-ON		Q	
222	TAPE		R	
223	X-OFF		S	
224	Null		T	
225	Null		U	
226	Null		V	
227	Null		W	
230	Null		X	
231	Null		Y	
232	Null		Z	
233	Null			K
234	Null			L
235	Null			M
236	Null			N
237	Null			O

Table F-1. (Cont)
ASR-33 Character and Symbol Codes

Code	Character Printed	Key Depressed			
		Lower Case	Simult. Shift	Simult. Control	Simult. Shift and Control
240	Space	Space Bar		Space Bar	
241	!		1		1
242	"		2		2
243	#		3		3
244	$		4		4
245	%		5		5
246	&		6		6
247	'		7		7
250	(		8		8
251	)		9		9
252	*		:		:
253	+		;		;
254	,	,		,	
255	-	-(minus)		-(minus)	
256	.	.		.	
257	/	/		/	
260	0	0		0	
261	1	1		1	
262	2	2		2	
263	3	3		3	
264	4	4		4	
265	5	5		5	
266	6	6		6	
267	7	7		7	
270	8	8		8	
271	9	9		9	
272	:	:		:	
273	;	;			
274	<		,		,
275	=		-	Alt Mode	-
276	>		.		.
277	?		/	Rub Out	?

Table F-1. (Cont)
ASR-33 Character and Symbol Codes

Code	Character Printed	Key Depressed	
		Lower Case	Simult. Shift
300	@		P
301	A	A	
302	B	B	
303	C	C	
304	D	D	
305	E	E	
306	F	F	
307	G	G	
310	H	H	
311	I	I	
312	J	J	
313	K	K	
314	L	L	
315	M	M	
316	N	N	
317	O	O	
320	P	P	
321	Q	Q	
322	R	R	
323	S	S	
324	T	T	
325	U	U	
326	V	V	
327	W	W	
330	X	X	
331	Y	Y	
332	Z	Z	
333	[		K
334	\		L
335	]		M
336	↑		N
337	←		O

Table F-1. (Cont)
ASR-33 Character and Symbol Codes

Code	Character Printed	Code	Page Printer	Lower Case
340	@	360	P	
341	A	361	Q	
342	B	362	R	
343	C	363	S	
344	D	364	T	
345	E	365	U	
346	F	366	V	
347	G	367	W	
350	H	370	X	
351	I	371	Y	
352	J	372	Z	
353	K	373	[	
354	L	374	Null	
355	M	375	Null	
356	N	376	Null	
357	O	377	Null	Rub Out

NOTES:

1. Whenever the BREAK key is depressed, a 000 code is generated as long as the key is held depressed. However, when the key is released, an indeterminate character is produced.
2. The symbols appearing in the Character Printed column indicate the reaction of the printer to codes received on the line in the output mode and to codes generated by the reader or keyboard in the input mode. Null indicates no printing and no spacing.
3. The lack of an entry in the Key Depressed column indicates the inability of the keyboard to produce that code.
4. The punch perforates all codes transmitted in the input or output mode when it is turned on.

APPENDIX G
HIGH SPEED PAPER TAPE READER (MODEL 112-20/22)

The high speed paper tape reader, together with its interface logic, transmits data to the controller at a maximum rate of 400 cps. Opaque paper or Mylar tapes, 0.0025 to 0.005 inch thick, can be used with no adjustments required. Tapes are loaded by positioning the RUN/LOAD/OFF power switch in either the LOAD or OFF position, lifting the LOAD lever on the reader housing, threading the tape through the reader with sprocket side inwards, lowering the LOAD lever, and returning the power switch to the RUN position. The pinch roller and brake solenoids are energized in the RUN position only.

OPERATING MODES

In addition to the normal input mode (described below under PROGRAMMING) wherein characters on the tape are transferred to the controller under program control, the reader may operate in the load mode, wherein characters are transferred to the controller without the use of a program. The load function is controlled by hardware in the controller and is initiated, when the tape has been loaded as described above, by pressing the LOAD pushbutton followed by the START pushbutton on the control panel. This starts the reader.

Transfers are made using a 3-bit character instead of an 8-bit character. Four such characters are packed into a 12-bit memory location of the controller with the first one in bits 1-3, the second in bits 4-6, the third in bits 7-9, and the fourth in bits 10-12. The next four 3-bit characters on the tape are packed into the following memory location, etc., until all characters on the tape have been read. The reader stops whenever it detects a character with a 1 in channel 7. (See Appendix B for load mode format.)

The load mode is especially useful when the memory does not contain a loader or when the only available panel is the remote operators panel. The ASR-33 keyboard or reader must not be used while the high speed reader is in the load mode.

The interface guarantees that the tape will stop after the end of the character just read but before the beginning of the next character in both the normal input and load modes.

PROGRAMMING

The instructions assigned to the high speed paper tape reader are as follows:

SKS 04 — Skip If High Speed Paper Tape Reader NOT Interrupting

This instruction determines whether the reader has caused the interrupt on the standard interrupt line.

SKS 05 — Skip If High Speed Paper Tape Reader Is Ready

This instruction determines whether the reader is ready to transfer a new character to the controller. The device is ready as soon as the control logic has received the sprocket pulse and remains ready until the character has been transferred to the controller.

INA 05 — Input A Character From The High Speed Paper Tape Reader, If Device Is Ready.

This instruction transfers an 8-bit character from the reader to the least significant 8 bits of the A register in the controller and skips the next instruction in the program, if the device is ready. The transfer must take place within 1 ms after the device has become ready or the interrupt has occurred. Completion of the instruction will reset the ready flip-flop and the interrupt. If the device is not ready, the A register is cleared, no transfer takes place, and the next instruction in the program is executed.

OCP 04 — Start High Speed Paper Tape Reader.

This instruction initializes the control logic and starts tape motion. The start response (time to advance one character) is less than 10 ms.

OCP 05 — Stop High Speed Paper Tape Reader.

This instruction resets the control logic and stops tape motion. The tape will stop between characters if this instruction is given within 1 ms after the device is ready or an interrupt has occurred.

SMK 00 — Set Standard Interrupt Mask.

This instruction sets the standard interrupt mask F/F in the high speed paper tape interface if the accumulator bit 3 is a 1 and resets it if the accumulator bit 3 is a 0. There is no skip test associated with this instruction and the next instruction in the program is always executed. Masking off an interrupt will not reset the interrupt F/F.

Program Examples

The example shows coding which may be used for inputting a character under program control:

```
--------
OCP 04      Start tape reader
INA 05      Input character from reader, if ready
JMP *-1     Delay until ready
--------
OCP 05      Stop tape reader
--------
```

Under interrupt control the program would enter a routine with the INA/JMP *-1 routine contained in it, immediately input the character and exit from the subroutine without delay. The routine should contain coding to determine whether a stop character had been read in and if so to stop the tape with an OCP 05. The controller has less than 1 ms to respond to the interrupt before data is lost and the ability to stop before next character is lost. The INA resets the interrupt at the end of the transfer.

Honeywell

COMPUTER CONTROL DIVISION, FRAMINGHAM, MASS. 01701

Printed in U.S.A.