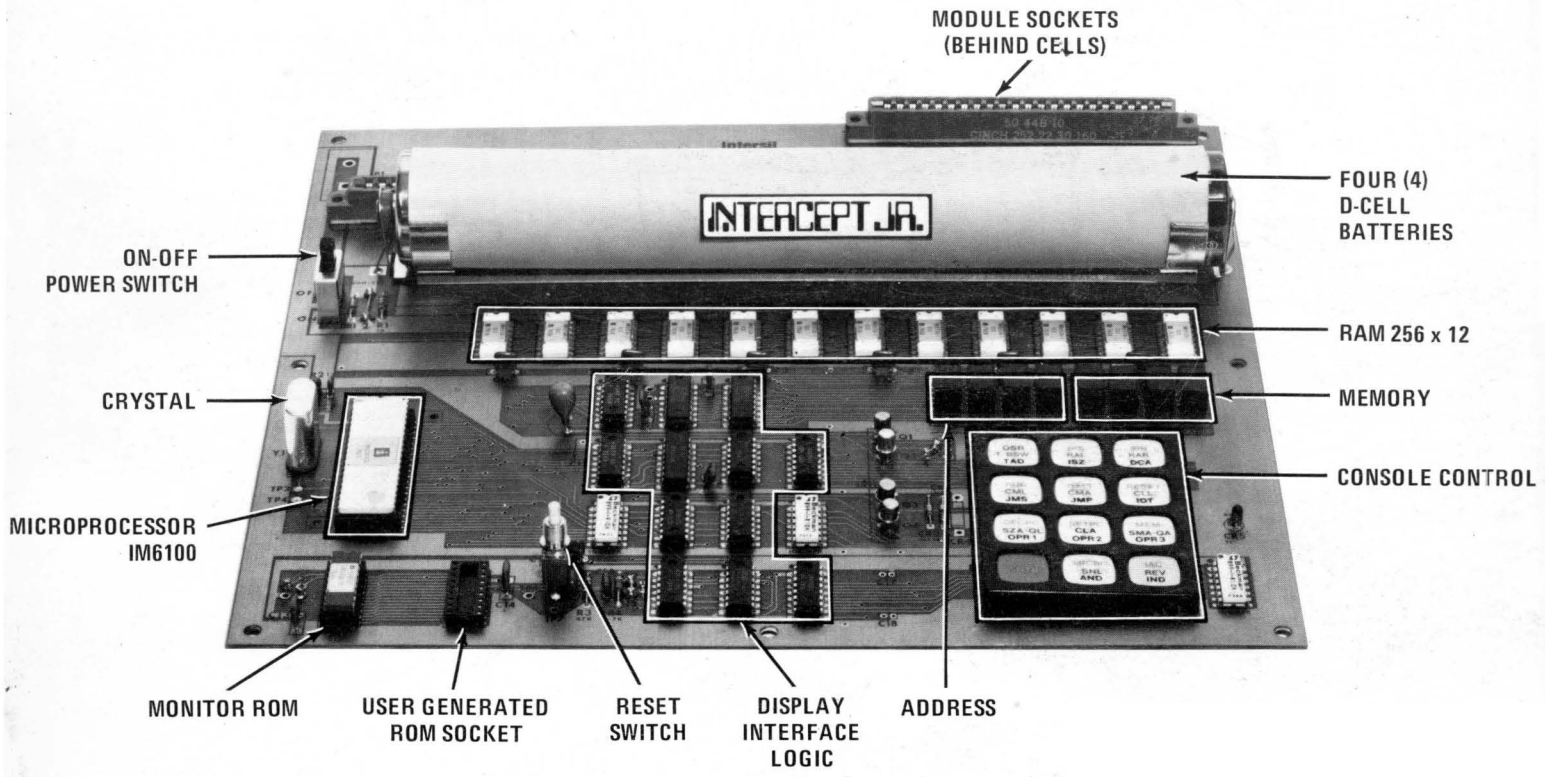
$5.00

INTERCEPT JR. TUTORIAL SYSTEM FROM INTERSIL

Intersil

OWNER'S HANDBOOK



Rev. A-November 15, 1976

INTERSIL CMOS/LSI

Intersil cannot assume responsibility for use of any circuitry described other than circuitry entirely embodied in an Intersil product. No other circuit patent licenses are implied. Intersil reserves the right to change without notice at any time the circuitry and specifications.

TABLE OF CONTENTS

APPENDICES

FIGURES

TABLES

CHAPTER 1
INTRODUCTION

The theoretical principles underlying digital computers were first enunciated by Charles Babbage in 1833, but the technology available at the time was not equal to the task of actually building a working machine. John Von Neumann developed the stored program concept at the Institute for Advanced Studies at Princeton University and, since then electronic computers have undergone several reiterations from the early vacuum tube machines to transistorization to integrated circuit systems, and now the age of LSI is evident. Architectural advances from the first use of hardware index registers, microprogrammed control, interrupt processing, direct memory access channels, and distributed processing have been numerous, but the history of digital computers has yet to be fully written.

In the late 1930's and early 1940's, wartime requirements and the development of vacuum tubes led to the construction of extremely expensive and complex digital computers used mainly to speed up numerical calculations. As the technology progressed, computers became faster, smaller and less expensive. Advances in hardware architecture and programming languages evolved rapidly. As a result, the 1960's saw significant increases in the application of business and data processing computers.

The first minicomputer, the PDP-8*, was introduced by Digital Equipment Corporation in 1965 and made dedicated applications for digital computers possible. This first minicomputer, costing approximately $50,000, was considered so inexpensive that it found itself being used in universities, laboratories, and in numerous process control applications. Many versions of this machine were brought out in succeeding years.

Computers, big and small, must all have a processor, main memory and input/output. Decreasing hardware costs and increasing sophistication of processing technology led to multiplicity of computer architecture. The early 1970's saw the microprocessor, the heart of a computer, enter the scene. Its function is to accept data from the user, process it according to instructions provided by the user, and stored in memory, and return usable results to the user in some convenient fashion.

LSI techniques, with their high density capability, have enabled semiconductor manufacturers to produce processing units and memory devices on single monolithic silicon chips. Input and output devices which constitute the man/machine interface, have remained relatively bulky.

* Trademark of Digital Equipment Corporation, Maynard, Mass.

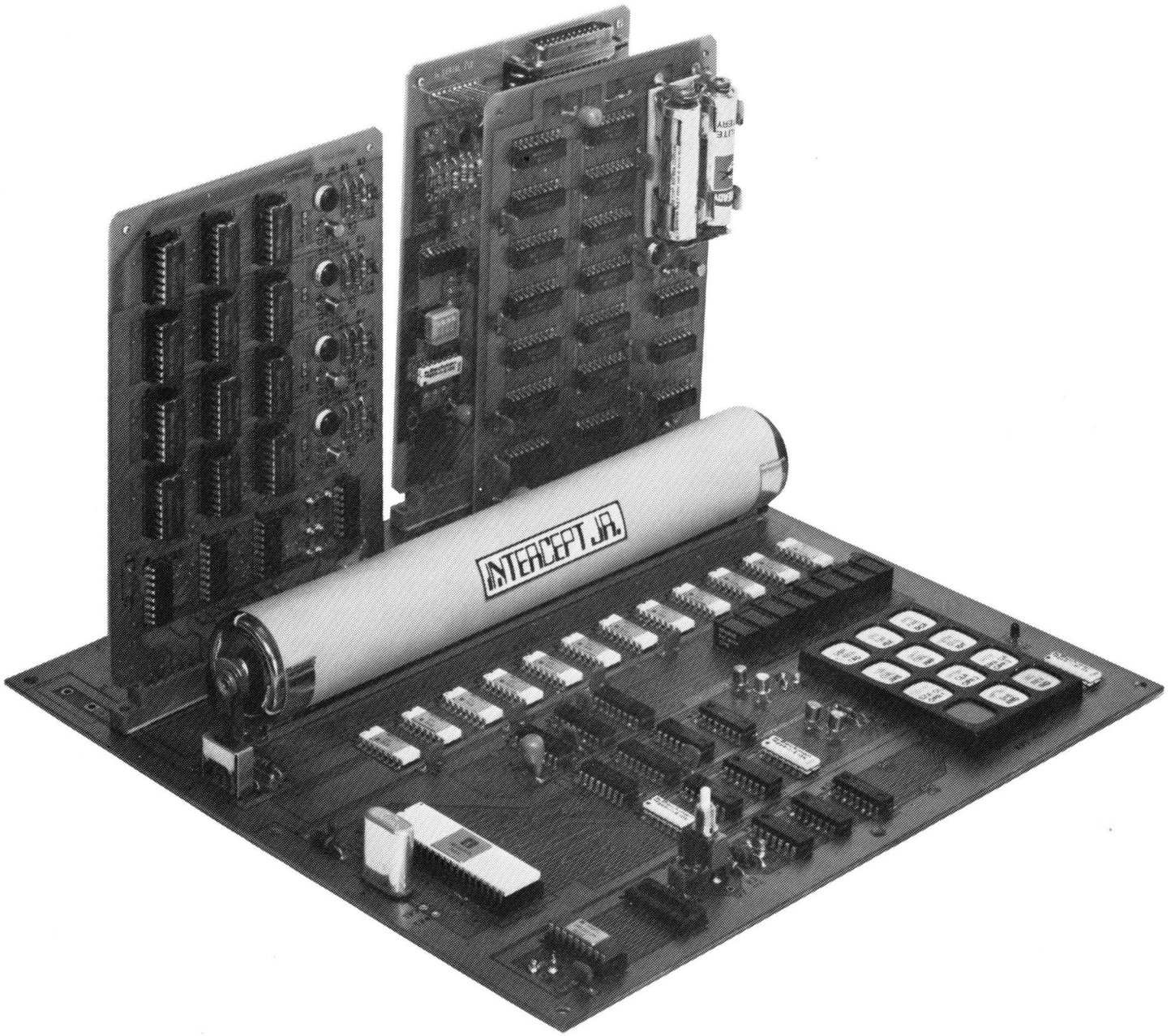


FIGURE 1-1

The INTERCEPT JR. TUTORIAL SYSTEM, pictured in Figure 1-1, recognizes the instruction set of Digital Equipment Corporation's PDP-8/E and is designed with a modular concept to enable the user to purchase only those modules which meet his requirements. The design permits the user to participate in the future of digital computers by yielding an understanding of the microprocessor and related component functions as well as programming fundamentals.

Large Scale Integration (LSI) of Intersil's digital CMOS components results in the system being battery operable and, thereby, yields the flexibility of a portable system. Experience can be gained with the components required for a classical computer architecture--a processor, or central processing unit (CPU), memory and input/output. The IM6100 microprocessor serves as the CPU and memory is available in the form of CMOS RAM, ROM and bipolar P/ROM. Input/output can be experienced in its simplest form via the keyboard and LED displays or can be studied in greater detail by utilizing the JR. SERIAL I/O MODULE.

This Owner's Handbook presents a step-by-step learning experience for the INTERCEPT JR. TUTORIAL SYSTEM. Chapter 2 entitled "Working With The Intercept Jr. Module" instructs the user in the fundamentals of the basic module--the start-up and the selection of a function. The console control, or keyboard, is discussed in detail. Chapter 3, "Programming Fundamentals", presents the user with simple programming examples and the ability to progress to more complex problems. Chapters 4, 5, 6 and 7 explain the hardware aspects of the four modules via pictorial representation, text and the corresponding schematics. Chapter 8 discusses the monitor ROM program, presents the flow chart and listing, and, thereby, gives the user a greater degree of programming insight. The Appendices contain fundamental information on number systems, two's complement arithmetic, an introduction to logic, and other miscellaneous information that will be of interest to the user.

It is Intersil Incorporated's opinion that the INTERCEPT JR. TUTORIAL SYSTEM will enable you to embark upon a truly rewarding educational experience. The microprocessor has resulted in a natural evolutionary step in electronic circuitry design. This is only the beginning. We sincerely wish that your participation in this evolution will be rewarding to you.

CHAPTER 2

WORKING WITH THE INTERCEPT JR. MODULE

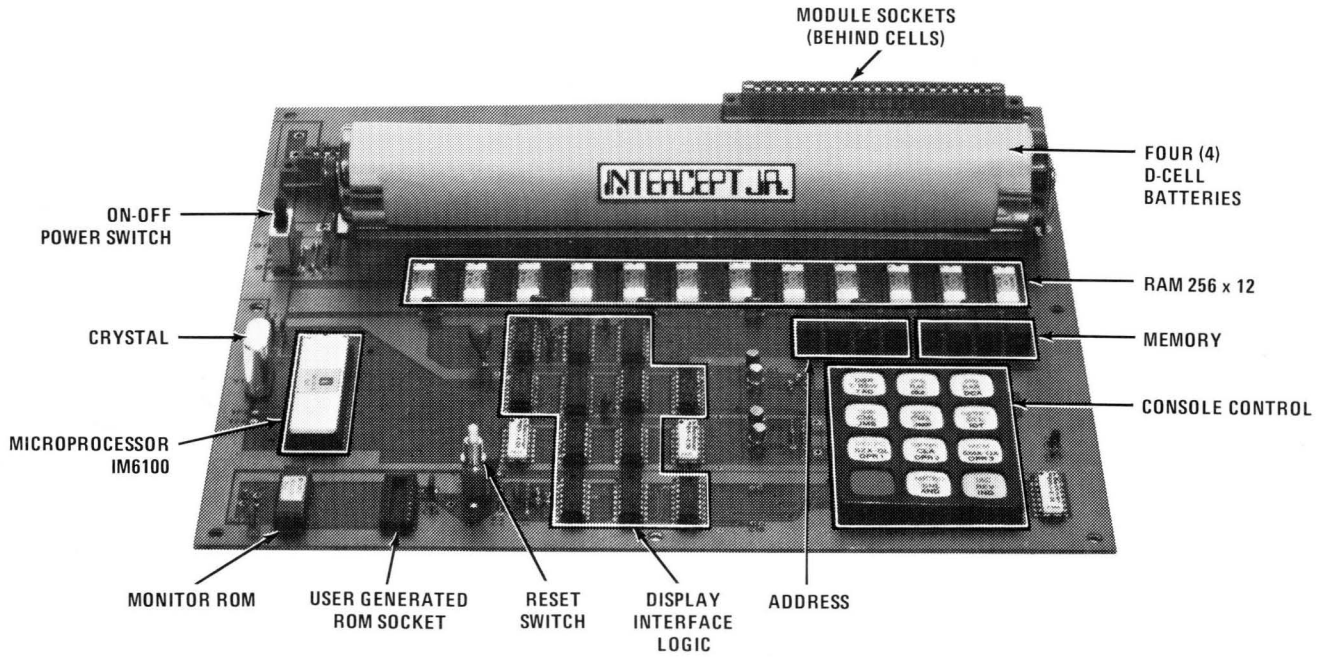


FIGURE 2-1

Figure 2-1 provides a pictorial representation of the INTERCEPT JR. MODULE with the pertinent components discussed in this chapter highlighted.

INTERCEPT JR. START-UP

Turn the module "ON" with the "ON-OFF" power switch. Power is provided by the four (4) D-Cell batteries which must be inserted, with the sleeve, in the module. When facing the module, with the keyboard in front and connectors on the far side, the left hand

battery clip is positive and the right hand battery clip is negative. BATTERY REVERSAL WILL DAMAGE THE SYSTEM. The module does a power-up RESET so that it will always come up halted with the Program Counter, PC (ADDRESS) equal to 7777. The CONSOLE CONTROL timer will be active so that the ADDRESS and MEMORY displays will be valid provided a "BLANK DISPLAY" is not in effect. The information displayed will be PC = 7777 in the ADDRESS and the MEMORY data in that location will be 5366. This instruction branches the microprocessor to routines which save registers, initialize the RAM stack and search for keyboard depressions. If the display does not illuminate, press 

to turn it on.

RESET SWITCH

The RESET SWITCH does a complete hardware reset of the microprocessor and can be used at any time for this purpose. Therefore, it is not necessary to turn power off to reset the microprocessor. When switching the module OFF it is recommended that the RESET SWITCH be depressed while the power is turned off. This keeps the microprocessor from running during the power down process thereby eliminating the possibility of writing bad data into the RAM as the voltage level goes lower than the minimum specified. If the RAM data is <u>not</u> required to be preserved, use of the RESET SWITCH is not required during power-off.

ENTERING THE CONTROL MODE

The operator will now enter the control mode by pressing the control key, CNTRL, on the KEYBOARD. This key will cause the module to enter what is referred to as an undefined control mode when the CONSOLE CONTROL timer is enabled, or at any point during the execution of a control function. This state is referred to as undefined as we have not yet chosen a CONSOLE CONTROL function to be performed. We may leave this state, without choosing any function, by pressing what we will refer to as the SHIFT key, S, or the key designated 

<u>IAC</u> REV IND. This will take the module out of the control mode and place it in the user mode, be it running, or halted. In the user mode, the module is either waiting for or executing user programs.

SELECTING A FUNCTION

After pressing the CNTRL key, we are now ready to choose a function to be performed. This is accomplished by pressing any of the ten (10) function keys which are described below.

SHIFT

As SHIFT, this is not a function key and pressing it will cause the module to return to the user mode. This key also has special meaning for certain functions and its use is described with each of those functions. This key is color-coded yellow.

SETPC

This function allows the user to control the Program Counter, PC, in the module for purposes of depositing words, or examining words or conditions. Following the activation of this key, the user will load an octal number into the PC by entering the digits on keys 0-7. The digits will be displayed in ADDRESS and will be entered from the right, shifting the previously entered digits to the left. Any number of digits may be entered until the display contains the value desired. Then the SHIFT key is pressed to indicate that the value is entered and that we wish to return to user mode. A CNTRL key will enter the value displayed into the PC and will return the state to undefined control mode. Note that leading zeros may be needed to clear the display before entering the desired octal numbers.

DECREMENT PC, DECPC

This function will decrement the value of the PC by one and return the module to user mode. This function is useful when examining sequences of memory locations.

DEPOSIT DATA INTO MEMORY, MEM

This function allows the user to enter instructions and data in the RAM as well as set the values of the internal registers of the module by depositing the data into memory locations used by the monitor program to save and update the data in these registers. After a closure of the MEM key, the user will proceed to enter digits on to MEMORY with keys 0-7 as he did for SETPC. The new digits will be displayed on MEMORY, entering from the right and shifting to the left. When the MEMORY display contains the desired value, the user will deposit it in the RAM by pressing either DECPC, or MEM. If

DECPC is pressed, the MEMORY display will be deposited into RAM in the memory location addressed by the ADDRESS display. The ADDRESS display will be decremented and the RAM information in the decremented address will be displayed in MEMORY. If MEM is pressed, the value shown in the MEMORY display will be deposited into the RAM in the memory location addressed by the ADDRESS display, the ADDRESS display will be incremented and the next word in RAM will be displayed in MEMORY. Successive depressions of MEM will increment the memory ADDRESS. Digits can now be entered from the right, as before. If the user wishes to skip a location, he presses MEM again. This will retain the value of that location in RAM and the ADDRESS will move to the next location. By pressing SHIFT, the user will deposit the value of MEMORY into the location specified by ADDRESS, the module will exit the control mode and enter the user mode. If the user presses CNTRL, the value shown in MEMORY will be deposited and the module will enter the undefined control mode. RAM locations 0000 and 0140-0177 are reserved for the MONITOR and should not be modified (see Chapter 8).

RUN

This function will set the microprocessor Run flip flop to RUN and will exit the control mode. The module will come out in the user mode at the PC point specified during control mode, running.

HALT

This function will clear the RUN flip flop in the microprocessor so that the module will come out of the control mode halted.

RESET

This function will be a complete software RESET of the module. All internal microprocessor flags are initialized, the accumulator and link are cleared and the PC is set to 7777. It will also remove a BLANK DISPLAY status.

SIN

This function, referred to as Single Instruction, will cause the module to perform, in the user mode, a single instruction. Following this, the possible changes of state can be observed by inspecting the contents of the appropriate memory locations. Due to the MONITOR program structure, the user can not single step through ROM-P/ROM locations or JMP.-1 instruction (see Chapter 8). SIN may be successively depressed to single step through a program.

DIS

This function will BLANK and RESTORE the ADDRESS and MEMORY display thereby conserving power. The BLANK/RESTORE function is achieved by depressing CNTRL followed by DIS to BLANK the display and then CNTRL followed by DIS to RESTORE the display. A blanked display will carry over from a power-down but will be cleared by a software RESET (depression of CNTRL and RESET). The RESET switch does not affect display status.

BIN LOADER

This function will activate the firmware loader which will load BINary tapes using the 6953-PIEART, JR. SERIAL I/O MODULE. This loader will return to the halted user mode when data has finished loading.

MICRO

This function will place the CONSOLE CONTROL at the control of the MICROINTERPRETER in the MONITOR ROM. The MICROINTERPRETER functions are elaborated on in the next section.

MICROINTERPRETER FUNCTIONS

Pressing CNTRL followed by MICRO causes INTERCEPT JR. to execute the microinterpreter routines which are resident in the MONITOR ROM. These routines will interpret key closures as opcode bits, relative address bits, page bits, address mode bits, and microinstruction bits according to the specific sequence in which the keys are depressed.

This enables the user to rapidly enter programs via the keyboard without constantly referring to the instruction format listings. The user should be familiar with the use of the instructions and the rules for combining micro-instructions in order to make the most efficient use of the microinterpreter.

MEMORY REFERENCE INSTRUCTIONS

In the MICRO mode, if any of the keys marked AND, TAD, ISZ, DCA, JMS, or JMP are pressed, the MEMORY display at the current memory ADDRESS will show 0000, 1000, 2000, 3000, 4000, or 5000, respectively.

All the following key closures are interpreted as address bits. The numerical keys may be depressed as many times as desired, entering octal address digits from right to left. If the resulting relative address is outside the page boundary (0-177$_8$ is the allowable relative addressing range), the displays will flash.

Depression of the SHIFT key will stop the flashing and clear the address field. The user should again attempt to enter a valid address.

At any time after the opcode is entered and the display is not flashing, thereby representing a valid address, depression of the SHIFT key will set the indirect bit of the instruction (add 4 to the next-to-most significant octal digit). After entering the instruction, depressing CNTRL will advance the ADDRESS counter. SHIFT may be pressed repeatedly to advance the ADDRESS counter.

INPUT/OUTPUT TRANSFER (IOT) INSTRUCTIONS

In the MICRO mode, depression of the IOT key will cause 6000 to be entered into the currently addressed memory location. Subsequent numeric key depressions are performed to enter the required device address and control bits into the IOT instruction.

Depressing CNTRL will advance the ADDRESS counter to the next location. Depressing SHIFT will cause the ADDRESS counter to step.

OPERATE INSTRUCTIONS

Operate instructions are divided into three groups of operate microinstructions. Thus, in the MICRO mode, the desired microinstruction group is selected by depressing the keys marked OPR1, OPR2 or OPR3. This will enter 7000, 7400 or 7401, respectively, into the MEMORY

display. If no additional keys are depressed and the address counter is advanced, these instructions, which are all NO OPERATION, NOP, will be entered. Further key depressions will set various bits in the instruction enabling the user to select valid microinstruction combinations. The microinterpreter does not check for illegal microinstruction combinations so the user must be careful about the combinations being selected. The tables show the more useful combinations. The user should become familiar with the rules of combinations and logical execution sequence in order to create microinstructions not shown in the tables.

On the CONSOLE CONTROL, in general, the designations in red are associated with OPR1 microinstructions, and the designations in green, except for -QA and -QL, are associated with OPR2 microinstructions. The -QA and -QL designations stand for MQA and MQL which are OPR3 microinstructions.

Conditional skip microinstructions in the OPR2 group may have their skip condition inverted by pressing the REV key while setting the microinstruction bits.

Rotate instructions in the OPR1 group may be changed from a single bit rotate to a two bit rotate by pressing the key with T/BSW designation on it. (This key is used for both two bit rotates as well as Byte SWap.)

The CLA command is used in all the operate microinstruction groups and the key may be pressed in any of these groups.

LEAVING MICRO MODE

Depressing the CNTRL key twice puts the user back into the undefined control state and free to choose the next function.

PROGRAM EDITING AND CORRECTION

If an instruction is entered incorrectly, the user must exit MICRO by depressing CNTRL twice. This will result in advancing the ADDRESS counter by one. Decrementing the ADDRESS counter by one is achieved by pressing DECPC. The user must then reenter the MICRO mode by pressing CNTRL and MICRO. Now the correct instruction is reentered in full.

The program may be examined location by location by successively pressing DECPC or MEM from the undefined control state. DECPC results in stepping backward through memory, and MEM results in stepping forward. These two keys may be pressed without going through the undefined control state in order to go backwards and forwards through the program in any sequence.

Memory data may be changed at will while stepping back and forth through the program simply by depressing the numeric keys in any desired fashion.

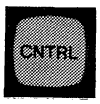
When editing in the MICRO mode, an instruction may be changed by entering a new sequence of keys. If an instruction is correct, the address counter may be stepped simply by pressing the yellow SHIFT key (immediately after CNTRL has been pressed to step the address) as many times as desired.

TABLE 2-1

TABLE OF INSTRUCTION CODES

KEYS DEPRESSED LEFT TO RIGHT	MEMORY OCTAL CODE	OPERATION
	-	Enter MICROINSTRUCTION Mode

MEMORY REFERENCE INSTRUCTIONS

KEYS DEPRESSED LEFT TO RIGHT	MNEMONIC	MEMORY OCTAL CODE	OPERATION
	AND*	0000	Logical AND
nn....n		00nn or 01nn	Depress numeric keys as required for valid address
		04nn or 05nn	Depress IND key if INDirect MRI is required
			Advances ADDRESS counter to next location
	TAD	1000	Binary ADD
	ISZ	2000	Increment and Skip if Zero

* The sequence of key depressions required to enter the opcode, address field, indirect bit (if necessary) and advance the address counter is shown in full for this case. The same sequence is true for the other memory reference instructions, but only the initial operation of entering the opcode is shown for the remainder to avoid duplication

STN RAR DCA	DCA	3000	Deposit and Clear Accumulator
RUN CML JMS	JMS	4000	JuMp to Subroutine
HALT CMA JMP	JMP	5000	JuMP

MEMORY REFERENCE INSTRUCTION FORMAT

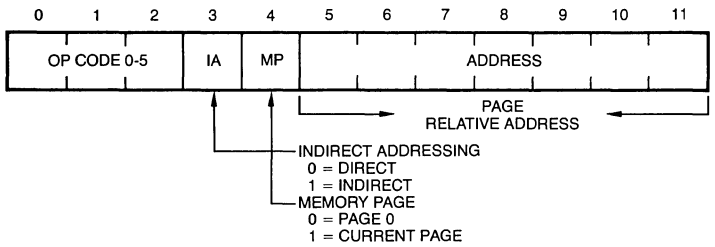
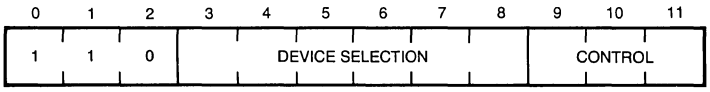


FIGURE 2-2

MICROPROCESSOR INPUT/OUTPUT TRANSFER (IOT) INSTRUCTIONS

KEYS DEPRESSED LEFT TO RIGHT	MNEMONIC	MEMORY OCTAL CODE	OPERATION
RESET IOT	SKON	6000	Skip if Interrupt on
RESET IOT + OSR T/BSW TAD	ION	6001	Interrupt Turn on
RESET IOT + DIS RAL ISZ	IOT	6002	Interrupt Turn off
RESET IOT + STN RAR DCA	SRQ	6003	Skip if INT Request
RESET IOT + RUN CML JMS	GTF	6004	Get Flags

RESET CLL IOT + HALT CMA JMP	RTF	6005	Return Flags
RESET CLL IOT + RESET CLL IOT	SGT	6006	Operation is Determined by External Device, if Any
RESET CLL IOT + SETPC CLA OPR2	CAF	6007	Clear All Flags



IOT INSTRUCTION FORMAT

FIGURE 2-3

DEVICE INPUT/OUTPUT TRANSFER (IOT) INSTRUCTION

KEYS DEPRESSED LEFT TO RIGHT	MNEMONIC	MEMORY OCTAL CODE	OPERATION
RESET CLL IOT	As applicable	6000	
n, n, n, n...n		6nnn	Depress numeric keys as required to enter specific address and control bits

GROUP 1 OPERATE MICROINSTRUCTIONS

KEYS DEPRESSED LEFT TO RIGHT	MNEMONIC	MEMORY OCTAL CODE	OPERATION
DECPC SZA-QL OPR1	NOP	7000	No operation
DECPC SZA-QL OPR1 + IAC REV IND	IAC	7001	Increment Accumulator

	RAL	7004	Rotate Accumulator Left
	RTL	7006	Rotate Two Left The T in T/BSW indicates bit 10 is set to give two shifts. Key may be pressed before RAL if desired.
	RAR	7010	Rotate Accumulator Right
	RTR	7012	Rotate Two Right Except for OPR1 key, order of depression is irrelevant.
	BSW	7002	Byte Swap Only bit 10 set giving byte swap function.
	CML	7020	Complement Link
	CMA	7040	Complement Accumulator
	CIA	7041	Complement and Increment Accumulator Logical execution sequence is CMA, IAC, but keys may be pressed in IAC, CMA order.
	CLL	7100	Clear Link
	CLL RAL	7104	Clear Link-Rotate Accumulator Left Logical sequence first clears link, then rotates.

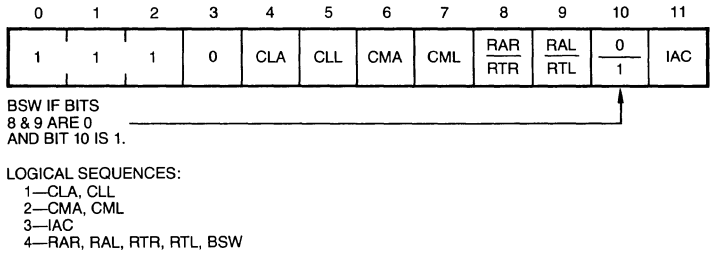
	CLL RTL	7106	Clear Link-Rotate Two Left
	CLL RAR	7110	Clear Link-Rotate Accumulator Right
	CLL RTR	7112	Clear Link-Rotate Two Right
	STL	7120	Set the Link Logical sequence first clears, then complements link.
	CLA	7200	Clear Accumulator Common to all groups, so not colored.
	CLA IAC	7201	Clear Accumulator-Increment Accumulator Loads accumulator with 1.
	GTL	7204	Get the Link Accomplished by rotating it into cleared accumulator.
	CLA CLL	7300	Clear Accumulator-Clear Link
	STA	7240	Set the Accumulator Sets accumulator to all ones.

Example of microprogrammed instruction to set accumulator to octal six.

Logical sequence: CLA CLL CML IAC RTL

Key sequence: 

Octal instruction: 7327

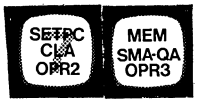
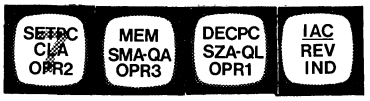


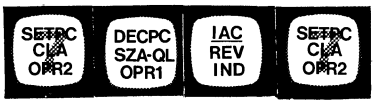
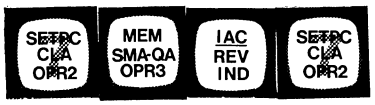
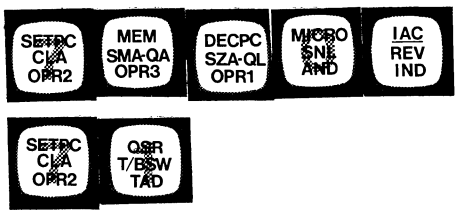
GROUP 1 MICROINSTRUCTION FORMAT

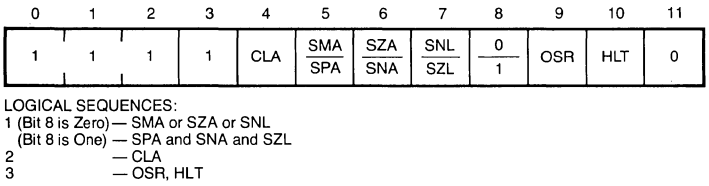
FIGURE 2-4

GROUP 2 OPERATE MICROINSTRUCTIONS

KEYS DEPRESSED LEFT TO RIGHT	MNEMONIC	MEMORY OCTAL CODE	OPERATION
SETPC CLA OPR2	NOP	7400	No operation
SETPC CLA OPR2 + HALT CMA JMP	HLT	7402	Halt
SETPC CLA OPR2 + OSR T/BSW TAD	OSR	7404	Or with Switch Register
SETPC CLA OPR2 + IAC REV IND	SKP	7410	Skip REV key sets bit 8 giving the AND condition of skips specified in bits 5, 6, 7. This results in unconditional skip.
SETPC CLA OPR2 + MICRO SNL AND	SNL	7420	Skip on Non-Zero Link
SETPC CLA OPR2 + MICRO SNL AND + IAC REV IND	SZL	7430	Skip on Zero Link REV reverses selected skip condition by setting bit 8.
SETPC CLA OPR2 + DECPC SZA-QL OPR1	SZA	7440	Skip On Zero Accumulator

	SNA	7450	Skip on Non-Zero Accumulator
	SZA SNL	7460	Skip on Zero Accumulator, or Skip on Non-Zero Link, or both OR'ed skip conditions.
	SNA SZL	7470	Skip on Non-Zero Accumulator, and Skip on Zero Link AND'ed skip conditions.
	SMA	7500	Skip on Minus Accumulator
	SPA	7510	Skip on Positive Accumulator
	SMA SNL	7520	Skip on Minus Accumulator, or Skip on Non-Zero Link, or both OR'ed skip conditions.
	SPA SZL	7530	Skip on Positive Accumulator and Skip on Zero Link AND'ed skip conditions.
	SMA SZA	7540	Skip on Minus Accumulator or Skip on Zero Accumulator or both. OR'ed skip conditions.
	SPA SNA	7550	Skip on Positive Accumulator and Skip on Non-Zero Accumulator AND'ed skip conditions.
	SMA SZA SNL	7560	Skip on Minus Accumulator or Skip on Zero Accumulator or Skip on Non-Zero Link or all OR'ed skip conditions.

	SPA SNA SZL	7570	Skip on Positive Accumulator and Skip on Non-Zero Accumulato and Skip on Zero Link REV AND'ed skip conditions.
	CLA	7600	Clear Accumulator Common to all groups.
	LAS	7604	Load Accumulator with Switch Register Logical sequence clears AC then loads it with switch register.
	SZA CLA	7640	Skip on Zero Accumulator then Clear Accumulator
	SNA CLA	7650	Skip on Non-Zero Accumulator then Clear Accumulator Order of key depression is irrelevant.
	SMA CLA	7700	Skip on Minus Accumulator then Clear Accumulator
	SPA CLA	7710	Skip on Positive Accumulator then Clear Accumulator
	SPA SNA SZL CLA OSR	7774	Skip on Positive Accumulator and Skip on Non-Zero Accumulato and Skip on Zero Link, then clear accumulator and load accumulator with the content of the switch register

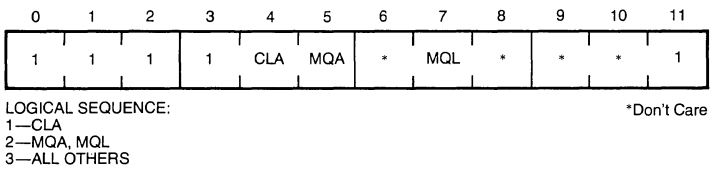


GROUP 2 MICROINSTRUCTION FORMAT

FIGURE 2-5

GROUP 3 OPERATE MICROINSTRUCTIONS

KEYS DEPRESSED LEFT TO RIGHT	MNEMONIC	MEMORY OCTAL CODE	OPERATION
MEM SMA-QA OPR3	NOP	7401	No operation
MEM SMA-QA OPR3 + DECPC SZA-QL OPR1	MQL	7421	MQ Register Load
MEM SMA-QA OPR3 + MEM SMA-QA OPR3	MQA	7501	MQ Register into Accumulator
MEM SMA-QA OPR3 + MEM SMA-QA OPR3 + DECPC SZA-QL OPR1	SWP	7521	Swap Accumulator and MQ Register
MEM SMA-QA OPR3 + SETPC CLA OPR2	CLA	7601	Clear Accumulator Common to all groups.
MEM SMA-QA OPR3 + SETPC CLA OPR2 + DECPC SZA-QL OPR1	CAM	7621	Clear Accumulator and MQ Register
MEM SMA-QA OPR3 + SETPC CLA OPR2 + MEM SMA-QA OPR3	ACL	7701	Clear Accumulator and Load MQ Register into Accumulator
MEM SMA-QA OPR3 + SETPC CLA OPR2 + DECPC SZA-QL OPR1 + MEM SMA-QA OPR3	CLA SWP	7721	Clear Accumulator and Swap Accumulator and MQ Register



GROUP 3 MICROINSTRUCTION FORMAT

FIGURE 2-6

CHAPTER 3
INTERCEPT JR. PROGRAMMING EXAMPLES

INTRODUCTION

The reader who is not familiar with elementary programming techniques, two's complement arithmetic and octal coding, should study Appendix A and the IM6100 brochure for a description of the instruction set before continuing with this section. The MONITOR program will be used to illustrate the use of various techniques.

EXAMPLE 1 - INCREMENTING MEMORY DATA

```
          :
          :
6400      2140    INCAC,    ISZ SAVAC   /Increment data in
                                          0140₈, location SAVAC
6401      7000              NOP         /In case location contained
          :                               7777
          :
```

This technique uses the ISZ instruction to directly increment memory data without needing to bring it into the AC first. Note the use of the NOP in case the data was 7777 and a skip was performed. In applications where the programmer knows this cannot happen, the NOP could be omitted.

EXAMPLE 2 - DECREMENTING MEMORY DATA

```
          :
          :
6403      7340    DECPC,    CLA CLL CMA /Set AC to -1
6404      1000              TAD SAVPC   /Add data in SAVPC
                                          (location 0000)
6405      3000              DCA SAVPC   /Restore decremented data
          :
          :
```

Note the use of the microinstruction combination CLA CLL CMA to clear the AC and the link and then to complement the AC, resulting in 7777 in the AC and 0 in L. By adding the contents of location SAVPC to the AC in two's complement arithmetic, a decrement is effectively performed. Note that the logical sequence of micro-instruction execution is chosen for usefulness. It would be of no value to complement the AC first and then to clear it.

EXAMPLE 3 - PROGRAMMING TIME DELAYS

```
        :
6203    3145    DCA       SAVE      /Store AC in SAVE and clear AC
6204    1233    TAD       TK1       /Get Time constant #1
6205    3144    DCA       TIME      /Store in timer
6206    2144    ISZ       TIME      /Time out 4 ms at 3.33 MHz
6207    5206    JMP.-1              /Jump back one location
        :
6233    7400    TK1,      7400      /-256
        :
```

This sequence is part of SWDB, the switch debounce routine described in Chapter 8. The AC is cleared (incidentally while depositing in SAVE), and the constant TK1 is fetched from the current page address 6233. It is stored in the page 0 location 0144 and ISZ instructions are successively executed until the timer goes to zero and the jump-back instruction is skipped. The delay produced may be calculated by counting the number of major states in each instruction executed and multiplying by the state time. Thus, ISZ requires 16 states and JMP requires 10, so these 26 states are gone through a total of 256 times, for a total of 6656 states. Adding in the states for the DCA, TAD and DCA (11 + 10 + 11 = 32) we have 6688 states. With a 3.3 MHz clock rate, the state timer is 600 ns so the delay is (0.6 x 6688) microseconds = 4012.8 microseconds or approximately 4 milliseconds. At 4 MHz, if the constant is not changed, the delay will be reduced by the factor 5/6 to 3.33 ms.

It is also instructive to note that the location TIME is in page 0, whereas the constant TK1 is stored in the current page (page 31). In this case, RAM happens to be available only in page 0 and 1 and by keeping TIME in page 0, the ISZ instruction in page 31 was able to directly reference the location TIME in page 0. Obviously, ISZ instructions may only reference RAM locations.

EXAMPLE 4A - ADDRESSING MODES

The user should note that a characteristic of page addressing results in the octal coding for two memory reference instructions on different pages being identical when their operands are in the same relative location on the respective pages.

```
      :
      :
0020  5225                    /JMP to location 25 on current
      :                        page, for example to 0025
0220  5225                    /JMP to location 25 on current
      :                        page, for example to 0225
      :
```

The user should enter these instructions. By using the SIN, single instruction key, to execute the instruction, the user will see how the addresses are referenced.

Note that memory reference instructions can reference 400_8 locations directly, 200_8 on page 0, and 200_8 on the page containing this instruction. If the instruction happens to be on page 0, then only locations 0 to 177_8 are directly addressable.

EXAMPLE 4B - ADDRESSING MODES

```
      :
      :
0020  5625                    /JMP indirect via 0025
      :
      :
0025  0010                    /Pointer to 0010₈
      :
      :
0220  5625                    /JMP indirect via 0225
      :
      :
0225  0010                    /Pointer to 0010₈
      :
      :
```

Now, by using the single step key at locations 0020 or 0220, the address should change to 0010 showing that an indirect reference has been made.

The pointer (location containing the effective address) can contain a full 12 bits of address, so the program can branch anywhere in the 4K address space by jumping indirect.

When constants and pointer addresses are stored in page 0, references may be made to them from any page, avoiding the necessity of storing them on each page that needs them.

EXAMPLE 5 - INDIRECT ADDRESSING USED IN TABLE MANIPULATION

This example is taken from the UDCS routine described in Chapter 8. It is a common technique of passing program control to one of several possible sequences by adding an index to a base address.

At the point that the following sequence is entered, the accumulator contains an octal number from 0 to 13 which stands for the routines MICRO, BIN, BLK, SIN, RUN, HALT, RESET, SETPC, DECPC, DEP, INCAC and UDCS respectively.

```
      :
      :
6123    1330            TAD GOTO     /add base address to constant
6124    3147            DCA POINT    /store pointer in POINT
6125    1547            TAD I POINT  /get routine starting address
6126    3147            DCA POINT    /phase starting address in POINT
6127    5547            JMP I POINT  /go to the routine
6130    6131    GOTO,   GOTO +1      /base address
6131    6633            MICRO        )
6132    7600            BIN          )
6133    6566            BLK          )
6134    7301            SIN          )
6135    6411            RUN          )   TABLE OF ROUTINE
6136    6407            HALT         )   STARTING ADDRESSES
6137    6414            RESET        )
6140    6600            SETPC        )
6141    6403            DECPC        )
6142    6524            DEP          )
6143    6400            INCAC        )
6144    6117            UDCS         )
      :
      :
```

Note that location 6130, labeled GOTO contains base address 6131, so by adding a number from 0_8 to 13_8 to 6131, a number from 6131 to 6144 is obtained. This number is stored in POINT.

Now, the effective starting address is obtained by executing a TAD indirect through POINT, for example contents of POINT used as operand address. Thus, is AC contained 3_8, then 6134 would be stored in POINT, and TAD I POINT would place 7301 in the AC to be again stored in POINT. This time an indirect jump through POINT loads 7301 into the program counter.

Of course, POINT had to be stored in RAM and since pages 0 and 1 are in RAM, POINT was chosen to be in page 0, in order that the upper ROM pages could reference it. It can be seen that indirect addressing makes writing programs easier in mixed RAM-ROM memory where memory references cannot be easily confined to small relative address displacements. See Table 3-1 for a list of pages and their memory locations.

TABLE 3-1

PAGE	MEMORY LOCATIONS
0	0-177
1	200-377
2	400-577
3	600-777
4	1000-1177
5	1200-1377
6	1400-1577
7	1600-1777
10	2000-2177
11	2200-2377
12	2400-2577
13	2600-2777
14	3000-3177
15	3200-3377
16	3400-3577
17	3600-3777
20	4000-4177
21	4200-4377
22	4400-4577
23	4600-4777
24	5000-5177
25	5200-5377
26	5400-5577
27	5600-5777
30	6000-6177
31	6200-6377
32	6400-6577
33	6600-6777
34	7000-7177
35	7200-7377
36	7400-7577
37	7600-7777

EXAMPLE 6 - THE JMS INSTRUCTION AND INDIRECT ADDRESSING

A very important use of indirect addressing is in returning to a main program from a subroutine. Appendix A shows how two programs may be linked using JMP instructions. The JMS instruction's usefulness lies in the fact that only one copy of a subroutine need be stored, for example in page 0, and a program anywhere in main memory may call it. INTERCEPT JR. uses a "last-in-first-out" (LIFO) or "pushdown" stack in page 0 to store subroutine return addresses. This allows nesting of subroutines and calling subroutines stored in the MONITOR ROM by linking through RAM. For further details refer to INTERSIL Applications Bulletin M008.

Our example will demonstrate the use of the JMS instruction in RAM, and the use of indirect addressing to return.

To enter the program, use the microinterpreter as described in Chapter 2.

```
0020    7240            CLA CMA     /AC set to 7777
0021    4100            JMS 0100    /Jump to subroutine starting
                                     at 0100
0022    7240            CLA CMA     /AC set to 7777
0023    7402            HLT
 .
 .
 .
0100    0000                        /This location will contain
                                     return address
0101    7200            CLA         /AC set to 0000
0102    5500            JMP I 0100  /Return to main program
```

Single step through this program (by successive depressions of SIN key after initial "CNTRL" "SIN" sequence at program starting address) and the program sequencing will be seen to go from 0020 - 0021 - 0100 - 0101 - 0102 - 0022 - 0023. In between, it will be instructive to look at location 0140 where the AC is saved by the MONITOR. The AC will initially be set to 7777, then the subroutine clears it, and then the main program again sets it to 7777. The JMS instruction stores the return address, namely 0022 in location 0100 so that upon executing the JMP indirect via 0100, the main program can be rejoined in sequence.

If a 1K RAM option card is available, the user could relocate the main program in an upper page and execute the same program provided the subroutine remained in page 0. The subroutine could be moved to a page different from page 0 or the main program's page but then an indirect JMS would have to be executed. We can illustrate this in page 0 as follows:

```
0020    7240            CLA CMA     /AC = 7777
0021    4424            JMS I 0024  /Jump via pointer in 0024
0022    7240            CLA CMA     /AC = 7777
0023    7402            HLT
0024    0100                        /pointer address
 .
 .
 .
0101    7200            CLA         /AC = 0000
0102    5500            JMP I 0100  /Return
```

An extra location to store the pointer is needed.

EXAMPLE 7 - AUTOINDEXING

Example 3 showed how a simple loop could be programmed using the ISZ and JMP instructions.

The IM6100 treats memory locations 0010 through 0017, in page 0, in a unique manner. Whenever an instruction makes an indirect reference to any of these locations, the content of the location is incremented before it is used as an operand. These locations can, therefore, be used in indexing applications. The incrementation is done automatically, provided the location was referenced indirectly, without needing ISZ or TAD and IAC instructions, so this feature is known as autoindexing. When these locations are addressed directly, they act as any other location.

Since the autoindex location is incremented before it is used as an operand, it must be set to one less than the first value desired.

```
0010                                    /Autoindex location
 :
 :
0200     7200           CLA             /Clear AC to 0000
0201     1212           TAD 0212        /Get # of locations to be cleared
0202     7041           CMA IAC         /2's complement of AC
0203     3212           DCA 0212        /Store in loop counter
0204     1213           TAD 0213        /Get "starting address -1"
0205     3010           DCA 0010        /Store in autoindex location
0206     3410           DCA I 0010      /Clear location pointed to by 0010
0207     2212           ISZ 0212        /Increment loop counter
0210     5206           JMP 0206        /Jump back two places
0211     7402           HLT             /Stop.  All locations cleared
0212     0100           CONSTANT        /# of locations to be cleared
0213     0277           START-1         /Starting address (0300) -1
```

Note that the autoindex location supplies successive memory address pointers until the counter goes to zero and the program halts. The program will clear locations 0300 to 0377.

EXAMPLE 8 - ADDRESS FIELD MODIFICATION

Instructions and program data may be stored in the same memory. Thus, it is possible to treat instructions as data or data as instructions if this would be of any use.

A powerful programming technique involves performing arithmetic on memory reference instructions in order to alter the location being referenced. In this case, the instruction is treated as an operand and incremented, decremented, etc. Logical operations such as masking certain bits may also be useful. Such techniques are useful when manipulating large data tables. Example 5 has shown one technique of manipulating jump address pointers.

Consider the following example:

```
0200    7300        CLA CLL     /Clear AC and L
0201    1213        TAD 0213    /Get # of data items
0202    7041        CMA IAC     /2's complement of constant
0203    3213        DCA 0213    /Store TALLY
0204    7240        CLA CMA     /AC = 7777
0205    0300        AND 0300    /AND contents of 0300 with AC
0206    7450        SNA
0207    5215        JMP 0215
0210    2205        ISZ 0205    /Increment address field
0211    2213        ISZ 0213    /Increment TALLY
0212    5204        JMP 0204    /Jump back to check next item
0213    0100        TALLY       /Constant giving # of items to
                                 be checked
0214    0777        MASK        /Used to mask off opcode bits
0215    1205        TAD 0205    /Get instruction referencing
                                 zero data item
0216    0214        AND 0214    /Zero opcode bits
0217    3221        DCA 0221    /Store address of zero item
0220    7402        HALT        /Halt
0221                            /Address of zero item
```

This program checks data stored in locations 0300_8 to 0377_8, when it encounters a zero data item in the list, it stores the address of this item in 0221 and stops.

Location 0213 initially contains the number of items stored starting in location 0300. The program replaces this number with its negative by two's complementing it. Successive data items are then read, AND'ing with 7777 in the AC. Note that if the AND leaves a non-zero AC, the AND instruction is incremented, stepping to the next item. A logical operation is done with this instruction to strip off the opcode bits when and if a zero data item is eventually detected. For this purpose, the mask 0777 is stored in 0214.

On powering up most locations will be non-zero, so the user can put a zero anywhere he chooses to check Example 8 operation. This technique of modifying instructions is a dangerous one to use in many situations because programs may be unintentionally changed because of an undiscovered "bug". (Modern concepts of structured programming discourage the use of this technique, but it is included because in some microprocessor applications, it might save memory locations.) For example, in this case, every time the program is rerun, locations 0205 and 0213 must be initialized.

EXAMPLE 9 - USING CONDITIONAL SKIPS

Group 2 microinstructions are primarily conditional skips and may be used to test conditions other than the number of passes that have been made through a loop. That is, the program may be made to loop an indefinite number of times until a specific condition is present in the accumulator or link bit. When two or more skip conditions are microprogrammed into a single instruction, the resulting condition on which the decision will be based is the logical OR of the individual conditions when bit 8 is 0, or, when bit 8 is 1, the decision will be based on the logical AND.

In the last example, the SNA instruction was used to skip on non-zero accumulator. The loop would continue as long as the next instruction was skipped and when the AC became zero, the program would jump out of the loop.

Very often conditional skips are used along with Group 1 operate microinstructions. The Group 1 instructions are used to manipulate the AC and L with shift, rotate, set, clear operations to set up these registers for testing with conditional skip instructions. This is used extensively in the MONITOR program, for example, in the routine called HEX (see listing of MONITOR and Chapter 8).

The following segment of code is in the MONITOR locations 6503-6507.

```
      .
      .
      .
6503     7006              RTL
6504     7420              SNL
6505     5310              JMP.+3
6506     7325              CLA CLL CML IAC RAL
6507     5564              RETURN
      .
      .
      .
```

This segment shows how a rotate is used to "set up" the link and bit 0 of the AC for a test with skip instructions. AC (0) can be tested with instructions such as SMA, skip if AC is less than 0, SPA, skip if AC is greater than or equal to 0, and their combinations, and the Link can be tested with instructions such as SZL, skip if Link = 0, SNL, skip if Link = 1. Combinations are possible which test these bits in one instruction, for example, SMA SNL, skip if AC is less than 0 OR if Link = 1, or SPA SZL, skip if AC is greater than or equal to 0 and Link = 0.

The user should note that SMA SNL will produce a skip on minus AC OR non-zero link OR both, whereas SPA SZL will produce a skip on plus AC AND zero link (both conditions must be present for a skip).

The example also shows how microprogrammed combinations of micro-instructions may be used to set various constants into the AC.

In this case, the AC is set to 0003. The sequence is as follows: AC and L are cleared, L is set, the AC is incremented and by shifting left one, the L is shifted into the LSB and the former LSB is shifted into bit 10, so the AC contains 0003 and the L contains 0.

Refer to the HEX routine for more examples of this nature.

EXAMPLE 10 - FLOWCHARTING A PROGRAM

Flowcharts may be used to represent hardware operation as well as to represent an algorithm to be implemented in software.

As an example of an algorithm, or computational procedure, we shall work out a program to compute the product of two octal numbers.

PROBLEM: Compute the product of two octal numbers

ASSUMPTION: The numbers are positive integers and their product does not exceed 4095_{10} or 7777_8. The 1st operand is not zero.

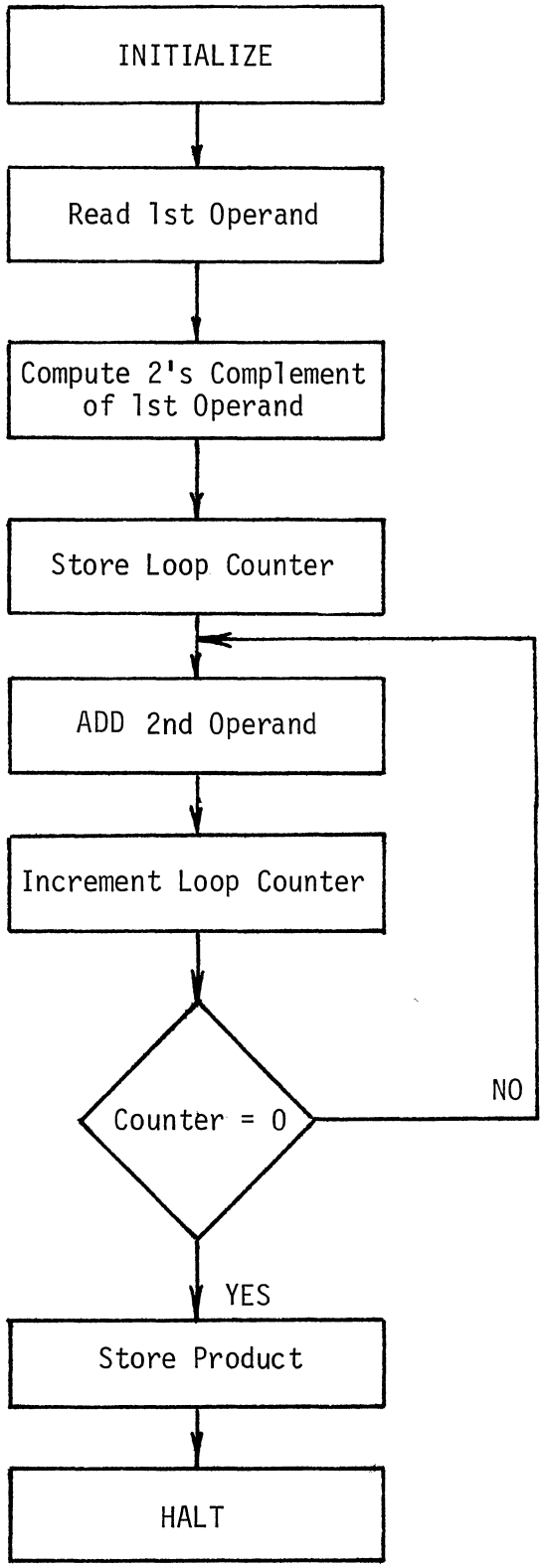
SOLUTION: Many different multiplication algorithms exist. We shall choose a simple, inefficient one which is easy to understand and flowchart.

Add one number repeatedly to itself using a second number to determine the number of additions.

The program will make use of a memory reference instruction known as "Increment and Skip on Zero". The ISZ instruction adds a 1 to the referenced data word and then examines the result of the addition. If the result is not zero, the program continues in sequence, performing the instruction following the ISZ. If the result is zero, the instruction following the ISZ is skipped (by incrementing the Program Counter again). In either case, the result of the addition replaces the original data word in memory.

By computing the 2's complement of one operand (data word) and referencing it with the ISZ instruction, we can repeatedly add the second operand to itself until the desired product is obtained. At this point, the counter becomes zero and the loop exit is taken.

After entering the program as shown, data may be entered into locations 0032 and 0033, the Program Counter is set to the starting address, and the program is run.


INITIALIZE
Read 1st Operand
Compute 2's Complement of 1st Operand
Store Loop Counter
ADD 2nd Operand
Increment Loop Counter
Counter = 0
NO
YES
Store Product
HALT

```
0020        7300        CLA   CLL
0021        1032        TAD   0032
0022        7041        CMA   IAC
0023        3031        DCA   0031
0024        1033        TAD   0033
0025        2031        ISZ   0031
0026        5024        JMP   0024
0027        3031        DCA   0031
0030        7402        HLT
0031                    loop counter and final product
0032                    1st OPERAND
0033                    2nd OPERAND
```

PROGRAM TO MULTIPLY TWO OCTAL NUMBERS TOGETHER

```
CNTRL       SETPC       0     0     2     0
CNTRL       MICRO     OPR1      CLA       CLL
CNTRL       TAD         0     0     3     2
CNTRL       OPR1       CMA       IAC
CNTRL       DCA         0     0     3     1
CNTRL       TAD         0     0     3     3
CNTRL       ISZ         0     0     3     1
CNTRL       JMP         0     0     2     4       enter program
CNTRL       DCA         0     0     3     1
CNTRL       OPR2      HALT
CNTRL
CNTRL       SETPC       0     0     3     2
CNTRL       MEM         1st OPERAND
            MEM         2nd OPERAND
CNTRL       SETPC       0     0     2     0       execute program
CNTRL       RUN         Display shows product
```

For example, if 1st operand is 0004 and 2nd operand is 0010, the display will show 0040. The user will also find it instructive to load small numbers as operands and single-step through the program to verify that the program follows the flowchart. Thus, set the PC to 0020, then press "CNTRL", "SIN" and then press the "SIN" key repeatedly. Each time it is pressed, the program executes one SINgle instruction. At any point, the user may set the PC to 0140 to examine the contents of the accumulator (this is explained further in Chapter 8) and resume execution of single instructions by resetting the PC to the last address he had stopped at and continuing with SIN key depressions.

The user will find it useful to rewrite the program to make the assumptions less restrictive. For example, a check could be included to test for a zero 1st operand and, if the test was true, the product zero could be immediately calculated. Tests for negative operands could be included and/or checks for arithmetic overflow.

EXAMPLE 11 - BIT MANIPULATION

Often, it is necessary to set, clear or determine the status of individual bits in a word. For example, a peripheral interface may be returning the status of various devices, and the processor must take action conditional on the status of these flags.

There are several methods. In one, the AC is rotated until the desired bit is in the link and then group 2 operate microinstructions are used to skip conditionally on the link status. This technique is illustrated in Example 9. Another method is to AND a mask word with the AC, zeroing out all bits except the one to be tested and then testing the AC for zero.

This technique will be illustrated with an example from the SIN routine in the MONITOR.

```
          :
          :
7343      1400      INDB,   TAD I SAVPC     /Get the instruction
7344      0355              AND LOT         /Mask out indirect bit
7345      7650              SNA CLA         /Test; is bit set
7346      5564              RETURN          /No; return with true
                                             address in TIME
7347      1544              TAD I TIME      /Yes; get true address
7350      3144              DCA TIME        /Place it in TIME
7351      5564              RETURN          /Return with true address
          :                                  in TIME
          :
7355      0400      LOT,    0400            /AND mask word
```

This routine INDB, determines the effective address referenced by an instruction and places it in location TIME. By AND'ing the instruction with 0400, the AC will be non-zero if the indirect bit, bit 3, is set and zero if this bit is zero.

The methods for setting and clearing bits are similar. One can rotate the bit into the link and then use group 1 microinstructions to clear or set the link. This has the advantage that rotates may be combined with link bit operations in one instruction.

To clear a bit, one can AND the word in AC with a word containing one's everywhere except in the desired bit position. To set a bit, one can add a word containing zero's everywhere except in the desired bit position. This technique is used by the bit set routines in the MICROINTERPRETER, ROM locations 7246-7300, listing lines 1006-1045.

The next example shows the use the MQ register in logical operations. It will be seen that this register may also be used in bit manipulation operations.

EXAMPLE 12 - LOGICAL OPERATIONS

Boolean operations play an important role in computer logic. We have seen examples of how the AND instruction can be used to mask out selected bits.

The NOT or logical complement operation is easily performed by placing the logical data word in the accumulator and executing a CMA, complement AC, instruction.

The inclusive OR operation is performed by placing one logical operand into the MQ register (executing an MQL - load MQ from AC), loading the second logical operand into the AC, then executing an MQA instruction (contents of the MQ are OR'ed with contents of the AC).

Any Boolean operation may be synthesized using combinations of the basic AND, OR and NOT operations.

EXAMPLE 13 - I/O PROGRAMMING

Chapter 7 and Chapter 8 give examples of I/O instructions as used in INTERCEPT JR.

There are three methods by which information may be transferred between INTERCEPT JR. and peripheral devices:

1) DMA I/O transfer
2) Interrupt I/O transfer
3) Programmed I/O transfer

The first method involves Direct Memory Access, DMA, by an I/O devices and allows for high speed transfers of blocks of data at essentially the memory cycle rate. The transfer is controlled without processor intervention on a "cycle stealing" basis. That is, the I/O device requests a DMA cycle and the processor grants it at the end of the current instruction. (See Figure 17 of the IM6100 brochure.) The processor tri-states its bus drivers and from that point on, as long as the DMA REQ line is active, the device controls the DX bus and data transfers on the bus. Typical DMA using devices are disks, tapes and CRT screen refresh circuits.

INTERCEPT JR. primarily uses the last two methods. Both of these require CPU intervention. Interrupt transfers use the interrupt system to service one or more peripheral devices simultaneously, permitting processing to be performed concurrently with data I/O operations.

Both methods use the AC as a data buffer for transfers in both directions.

Interrupt programming is especially useful in real time systems which are required to respond to real time events. The time spent waiting for a change in device status is greatly reduced or even eliminated. This is done by writing I/O handling routines which are separate from the main program and using the interrupting capability of I/O devices to enter these routines only when the I/O device is either ready to perform a data transfer or requires CPU intervention. Thus, as long as the device does not request an interrupt, the mainline program may continue to run and time is not wasted "polling" I/O devices for changes in status.

In INTERCEPT JR., the control panel timer generates interrupt requests at periodic intervals. The display refresh routine that periodically drives the LED displays is an example of an I/O handling routine. When the main program is interrupted, a method of returning to it after servicing the interrupt request is necessary. INTERCEPT JR. saves the current content of the PC in location 0000_8 of the memory and fetches the next instruction from location 0001_8 if an external I/O device requests an interrupt.

In the case of a control panel interrupt, the return address is stored in location 0000_8 of panel memory. This is the same as 0000_8 of page 0 memory in INTERCEPT JR.; status word bit 0 differentiates between panel and user memory. The CPU resumes operation at location 7777_8 of panel memory, for example 7777_8 of MONITOR ROM ISD002 with status word bit 0 = 0.

For further details on device interrupts and CP interrupts, refer to the IM6100 and IM6101 data sheets.

The third, and slowest method, that of programmed data transfer, is also the simplest, needing a minimum of hardware support. The INTERCEPT JR. PIEART board uses this technique. The processor, upon recognizing an I/O instruction, opcode 6_8, places the instruction on the DX bus during $IOT_A \cdot LXMAR$. The selected device communicates with the CPU through four control lines-- C_0, C_1, C_2 and SKP. The control line SKP, when low during an IOT, causes the CPU to skip the next sequential instruction.

The INPIE, TALK, LISN, READ routines of the MONITOR should be studied to see the use of IOT's in programmed data transfer.

For example, the print to TTY routine is as follows:

```
          :
          :
7466      6163      TALK,   SKIP2       /Skip on clear Xmit buffer
7467      5266              JMP.-1      /Xmit buffer not yet clear
7470      6161              WRITE1      /Write AC to buffer
7471      3144              DCA TIME    /Store data for possible
                                        recovery
7472      5564              RETURN
```

Note the use of the SKIP2 instruction to implement a "wait" loop. When the condition is satisfied, the loop is exited. The device must activate the SKP line back to the CPU in order for the CPU to skip the next instruction.

The WRITE1 instruction is another IOT used to write the AC to the UART. (See Chapter 7 for device address codes and command codes.) Refer to the IM6100 and IM6101 data sheets for more information.

The next chapter describes dedicated IOT instructions used in INTERCEPT JR. namely, 6400, Load Display, 6402, Enable/Disable CP Timer, 6403, IOT CPREQ, 6406, IOT Reset, 6407, IOT RUN. The experienced user may use these to shut off the timer and perhaps use subroutines in the MONITOR for his own purposes, for instance, display information other than the USERPC and its contents.

EXAMPLE 14 - TELETYPE I/O USING MONITOR CALLS

The following program makes use of the MONITOR ROM PIE-UART subroutines by calling them via the software stack mechanism.

The control panel interrupt requests must be shut off to prevent timing difficulties.

```
0100      7340      Set AC to 7777
0101      6402      Disable CP request timer
0102      4161      CALL
0103      7445      PIE initialization routine in PIE
0104      4161      CALL READ from
0105      7501      Teletype routine
0106      4161      CALL TALK, the print
0107      7466      to TTY routine
0110      5104      Jump back for next character
```

Note that the stack mechanism requires that the CALL instruction (JMS 0161) be followed by the entry address of the subroutine.

EXAMPLE 15 - PRINTING UNDER KEYPAD CONTROL

The following program will print ASCII characters on a Teletype under control of the INTERCEPT JR. board.

Refer to Appendix F for the ASCII character set.

```
100     7340            STA STL     /Disable
101     6402            IOT TIMER   /Control panel timer
102     4161            CALL        /Initialize
103     7445            INPIE       /PIEART interface
104     7300    BACK,   CLA CLL
105     4161            CALL        /Wait for keypad
106     6110            CLKPD       /To clear
107     4161            CALL        /Read octal
110     6425            HEX         /Data from keypad
111     7004            RAL         /Shift three places
112     7006            RTL         /Left and swap bytes
113     7002            BSW         /To determine leading
                                     code digit
114     1131            TAD K0002   /MSB of ASCII code
                                     always one
115     7500            SMA         /Is 2nd ASCII digit
                                     4,5,6,7?
116     7001            IAC         /No, 1st digit must
                                     therefore be 3
117     7002            BSW         /Yes, 1st digit must be 2
120     3132            DCA TEMP1   /Store temporarily
121     4161            CALL        /Wait for clear
122     6110            CLKPD       /Keypad
123     4161            CALL        /Read 2nd octal
124     6425            HEX         /Digit
125     1132            TAD TEMP1   /Assemble ASCII character
126     4161            CALL        /Transmit character
127     7466            PRINT       /To printer
130     5104            JMP BACK    /Go back for next character
131     0002    K0002,  0002
132     0000    TEMP1,  0000
```

Appendix F shows that the 8-bit ASCII character codes have the property that if the left octal digit is 2, the second octal digit is 4, 5, 6 or 7, and if the left octal digit is 3, then the second octal digit is 0, 1, 2 or 3.

This program allows the user to enter characters as two successive octal digits.

Note that this assumes the eighth (parity) bit is always set.

EXAMPLE 16 - PROGRAM TO DEMONSTRATE I/O TO 6957 AUDVIS MODULE

```
0225    7201            CLA IAC         /Set AC=0001
0226    6402            ENDIS TIMER     /Shut off CP timer
0227    7000            NOP
0230    7000            NOP
0231    7604    READ,   LAS             /Load keypad to AC
0232    7450            SNA             /Key depressed?
0233    5231            JMP READ        /No, go back to try
                                         again
0234    6401            LD DISPLAY      /Display AC on LED
                                         register
0235    6404            CLOCK           /Click speaker
0236    5231            JMP READ
```

The first two instructions shut off the control panel interrupt timer. The three instruction loop in locations 231, 232, and 233 cause the processor to wait until a key is depressed, and when this occurs, to load the LED register with the AC and CLICK the speaker.

While a key is depressed, the processor executes the instructions

```
LAS             (15 major states)
SNA             (10 major states)
LD DISPLAY      (17 major states)
CLOCK           (17 major states)
JMP READ        (10 major states)
```

continuously, and the speaker "clicks" merge into a high pitched beep. The fundamental frequency of this "beep" is easily calculated by counting the number of major states in the above instruction sequence, multiplying by twice the clock period and taking the reciprocal of this number.

In this case, there are 69 major states; and, assuming a 2.56 MHz crystal, the clock period is 390 ns, and the "beep" frequency is $1 / (69 \times 2 \times 390 \times 10^{-6}) \simeq 18$ KHz.

Now change the instruction in location 0236 to 5230. This adds a NOP, or 10 more major states to the loop, decreasing the frequency of the beep. By placing 5227 in location 0236, the frequency is lowered further. This program enables the user to find out which DX line each key is connected to.

Instead of a beep, the program can be made to click on each key depression by replacing the two NOPs with 4161 and 6110. This calls the CLKPD subroutine which waits for a clear (fully released) keypad before returning to the calling program.

The action of the HEX program which encodes key depressions in order to generate MONITOR program subroutine starting addresses may be easily seen by replacing the three instruction keypad read loop in locations 231, 232 and 233 with the sequence 7000, 4161, and 6425. As before, 4161 is a JMS to the top of the RAM subroutine stack and 6425 is the starting address of the HEX routine. Descriptions of these programs may be found in Chapter 8, and a discussion of the software stack may be found in Applications Bulletin M008 of the IM6100 databook.

The program just entered should have looked like this:

```
0225    7201
0226    6402
0227    4161
0230    6110
0231    7000
0232    4161
0233    6425
0234    6401
0235    6404
0236    5227
```

CHAPTER 4
INTERCEPT JR. MODULE

INTRODUCTION

As shown on the schematic, all memory and I/O devices are connected to the IM6100 DX bus. The twelve (12) bit bus carries time-multiplexed addresses and data from memory and I/O devices.

Timing information must be provided to strobe data on and off the bus and select lines are needed to enable the proper devices.

The MONITOR ROM and 256 X 12 RAM are mapped in upper and lower areas of the 4K address space, and it is necessary to select the proper devices during memory I/O.

The keyboard commands must be interpreted after making sure switch bounce does not cause erroneous operation.

The ADDRESS and MEMORY display digits are multiplexed in order to reduce the number of decoder/drivers required.

The IM6100 microprocessor used in the INTERCEPT JR. is the commercial temperature range device and a 2.46 MHz crystal is used in order to ensure operation of the system as battery voltage falls from 6 V to 4.5 V.

TYING ON TO THE DX BUS

The DX bus carries addresses and data at different times. All peripherals and memory address inputs, peripherals and memory data inputs and outputs are connected to the bus. All elements connected to the bus are, therefore, tri-state devices.

Data strobes and device signals must be generated in order to demultiplex data from the bus or multiplex data onto the bus.

The MONITOR ROM, a 1024 X 12 device is mask-programmed at the factory to decode the lower ten (10) bits as an address, and the upper two (2) bits as a chip enable. For example, the MONITOR ROM, as supplied by the factory, has the upper two bits mask programmed to 11 to select the ROM for 6000 to 7777.

When data is read out, the chip puts its data out onto the DX bus. Thus the DX pins on the 6312 are bidirectional (addresses in and data out).

The RAM is a 256 X 12 array implemented in CMOS.

The A_0-A_7 address inputs and the I/O data pins are connected to the DX bus.

ADDRESS DEMULTIPLEXING

Both the ROM chips and the RAM chips have internal address latches. These latches are loaded from the address inputs when the strobe input STR is driven low. When STR is low, the latches are not affected.

When the processor places memory address data on the bus, it drives the signal LXMAR at pin 10 low. This signal, Load External Memory Address Register, is intended to strobe the memory address latches. Note that the chip does not have to be selected in order to latch address information.

DATA DEMULTIPLEXING

After the CPU places a memory address on the bus, a data transfer must take place either into the CPU from memory or from the CPU to memory. The direction is indicated by the XTC line. The various SELECT lines are activated during the data-in and data-out phases of the memory cycle. XTC is high for the first half of a memory cycle (when memory read operations may be performed) and low for the second half (when memory may be written into). Thus XTC may be directly connected to OEH, Output Enable Active High, of the ROM chips and WE, Write Enable Active Low, of the RAM chips to enable these chips for reading or writing. During XTC high, of course, the RAM may be selected for reading. The memory outputs will not be activated unless the chip has been selected as well as had its output enabled. Otherwise, many chips would be activated at the same time.

Obviously, it would be undesirable to simultaneously read from several devices onto the same DX lines at once.

For this reason, the active low chip select pins on the RAM chips and OEL, Output Enable Active Low, on the IM6312's are connected to the SEL line. This line may be strapped to either the "MEM SEL" line or the AND'ed combination of "MEM SEL" and "CP SEL". These are active low signals generated by the CPU to select user memory, MEM SEL, or control panel memory, CP SEL. With only the Intersil provided control panel ROM in the system, the jumpers should provide the combination AND signal. This combination signal will select memory when either MEM SEL or CP SEL goes low.

Another aspect to be considered is how addressable memory space is partitioned. In the INTERCEPT JR., the MONITOR ROM occupies the highest 1K of the basic 4K address space and the RAM occupies the lowest 256 words of this space. It is possible to program 256 word pages of the 4K address space for RAM into the IM6312 ROM such that it will generate an RSEL, RAM SELECT, signal by decoding the high

order four bits of the address. These fields must obviously be aligned with page boundaries. RSEL is connected to CS_3 of the IM6524's. In the IM6312-002 MONITOR ROM, RSEL is activated by "0000" on DX0, DX1, DX2 and DX3.

RSEL allows random mapping of double page RAM fields within the 4K address space. Note that the base page, or at least the first 16 locations mut be writable in order for the autoincrement instructions and interrupt instructions to work. Also note that the highest location (7777) should normally be in ROM as it is used as a pointer to power up initialization routines. See Figure 8-1 for a memory map.

Normally the RAM area does not overlap with the ROM area, therefore, one of the RAM chip select pins is kept permanently low by a jumper to GND so that selection depends only on the chip select connected to the SEL line. RAM VCC is always present for data retention.

The mapping of RAM into ROM space is of significance should the user generate a ROM to be placed in the spare socket which requires this feature. In such a case, the RAM chip select jumper must be connected to the appropriate RSEL pin. The ROM is mask programmed to generate RSEL appropriately.

Please refer to the IM6312 data sheet for further details.

KEYBOARD INPUT

The INTERCEPT JR. uses a 12 switch keyboard which is an ideal situation as there are 12 DX lines. Each key is connected through a 3-state inverting buffer to the corresponding DX line.

When the CPU executes an OSR instruction, OR Switch Register with accumulator contents, it activates the SW SEL, Switch Select, line and OR's the DX bus with the accumulator. SW SEL is used to enable the keyboard buffers thereby giving the means to read the keyboard.

Naturally, it must not respond to illegal key closures (illegal combinations, bouncing, or too many keys being depressed, etc.). These conditions are checked by the firmware, to be described later.

To improve noise immunity, the inputs to the buffers are pulled up to VCC via 10K resistors in a DIP package. This is done to the DX bus as well, because lines floating at threshold are sensitive to noise.

DIGITAL DISPLAY OUTPUT

The INTERCEPT JR. has two display registers, each with four decimal (BCD) digits.

Each register is driven by a type 4511 CMOS BCD-TO-7 segment latch/ decoder/driver and four transistors that enable successive digits in turn (E2, F2, Q1, Q2, Q3, Q4)*.

The CPU loads the BCD latch with a digit each, and the 34042 quad CMOS latch (D2) with a single bit and this enables two particular digits to display the decoded contents of the BCD latches. In the next cycle, the BCD latches get loaded with the contents of the two adjacent digits and the bit shifts one position in the quad latch, enabling the next digits, and so on. The CPU can blank the displays under keyboard control in order to conserve battery power.

The data in the AC is loaded into the display latches by 'LOAD DISPLAY' at IOTA · $\overline{XTC}$ · $\overline{DEVSEL}$ The 'LOAD DISPLAY' command is generated by IOT decoding circuitry to be described in the next section.

The 2N2222 transistors, when turned on by the shifting bit, connect the LED common cathode to a low voltage. The drivers source current to individual segments, lighting these up for the time that the bit keeps that digit selected (nominally 8 ms at 4 MHz).

IOT PROCESSING

The INTERCEPT JR. uses Programmed Data Transfer techniques for all I/O operations. This technique uses the IM6100 IOT instructions, which have an octal opcode of 6, to initiate peripheral I/O operations. These operations could be sensing of peripheral device status flags, for example, "is TTY ready", or controlling device operation, for example, "move disk head to next track", or a data transfer operation, for example, "read character". The nature of the operation depends entirely on the device interface circuitry.

The IM6100 also has the capability for INTERRUPT data transfers and DMA data transfer, but these are unused in the INTERCEPT JR. except for console interrupts described in the next section.

When the IM6100 fetches an IOT instruction, it executes an IOTA cycle, during which the entire IOT instruction is placed on the DX bus during LXMAR time. This means external address registers, such as the ones on board memory chips, will all be loaded with the IOT instruction. In order not to have a memory chip respond falsely, the CPU suppresses the MEM SEL signal, and activates the DEV SEL, Device Select, signal. The device address and control information present in bits 3-11 of the IOT instruction are decoded and the DEV SEL signal is used by the peripheral to enable the selected functions.

* These designations are used to identify the devices on the schematic and on the assembled board.

The 340175 CMOS quad latch (D3) is strobed by $\overline{\text{LXMAR}}$ to latch DX3 and DX9, DX10, DX11 from the bus. The 74C42 CMOS BCD to decimal decoder (E3) is fed with AX11, AX10, AX9 and $\overline{\text{AX3}}$. The AX3 line acts as an enable to the decoder and must be high in order for the D input to the decoder, which is the most significant bit, to be low.

This means that all device addresses in this system should be of the form 1XXXXX. The 74C42 is a control decoder and only eight of its outputs, corresponding to the possible permutations of the three bit control field in the IOT instruction, may be used. Of these eight, only five, corresponding to IOT's with DX3 high and 0, 2, 3, 6 and 7_8 in their control field, are used. For simplicity we shall assume a device address of 100000 or 40_8.

These IOT instructions will now be described:

LOAD DISPLAY, or 6400 is gated along with XTC and DEVSEL through an OR, the 34025 NOR (F3) followed by the 34069 inverter (F4), into the Load Enable pins of the display drivers. During IOTA · $\overline{\text{XTC}}$ · $\overline{\text{DEVSEL}}$ time, this control function will load the latches in the display drivers (E2, F2) and the 34042 quad latch (D2) which drives the multiplexing transistors.

IOT RESET, or 6406 is gated along with DEVSEL through the two NOR's (E4) to generate an active low RESET. RESET is also generated on power-up, when the one input of the 34001 NOR gate (E4) is pulled high by the charging .47 microfarad capacitor. The RESET line driven low will clear the IM6100 accumulator, load 7777_8 into the program counter, and halt the CPU, besides resetting external logic. RESET is activated on power-up through the RC circuit, at any time by pressing the RESET switch or under program control. The RESET line into the IM6100 is sampled at T1 time of the last cycle of an instruction, and the worst case response time is 14 μsec at 4 MHz. The IOT RESET is a software simulation of the direct RESET line needing approximately a dozen instructions. Including the time needed to debounce the keypad, executing the routine, etc., the response time is many milliseconds. Thus the CPU does not actually do a RESET; it is made to clear all registers initialize the PC to 7777 and is then halted.

IOT RUN, or 6407 from the control decoder is gated along with DEVSEL. When enabled by XTC, the RUN/HLT line is driven by a negative going pulse. Each such pulse causes the CPU to alternatively run and halt by changing the state of the internal RUN/HLT flip flop.

IOT CPREQ, or 6403 is gated with DEVSEL through the 34025 NOR (F3) and 34069 inverter (F4) into the active low direct set input of the DFF 74C74 (F5). During IOTA time, DEVSEL will set the DFF and provided that INTGNT is not active and holding off the 34011 NAND (E5), a CPREQ will be issued. The 74C74 is reset by CPSEL.

CP TIMER EN/DIS, or 6402 is an IOT instruction that is used to turn the control panel interrupt timer or or off under program control. The CP timer circuit is formed by two gates (34001 NOR at E4 and 34011 inverter at E5) and an RC circuit (6.8 K R4 and .47 microfarad C16) and as long as pin 8 of the NOR at E5 is low, the oscillator is enabled, running and clocking the DFF at F5 at a 30 Hz rate. Thus, CP REQuests are issued at a 30 Hz rate (the DFF being reset by CPSEL in between). When IOT instruction 6402 is executed, during IOTA · $\overline{\text{DEVSEL}}$ · $\overline{\text{XTC}}$ time, clock input pin 3 of the 74C74 DFF at F5 is driven low and the rising edge of DEVSEL clocks in the data on DX11 into the flip flop. At this time, the IM6100 is driving the DX bus with the accumulator so if AC11 is high, the DFF is set, and if AC11 is low, the DFF is cleared. If the DFF is set, the CP timer is disabled by holding pin 10 of the NOR gate at E4 at a low. If the DFF is cleared, this gate is allowed to toggle and the timer runs. Note that during normal operation, the CP timer is running, and CPREQ and CPSEL are being generated.

The reason that CPREQ is not activated unless INTGNT is inactive is that control panel interrupt requests have higher priority than device interrupt requests or even DMA requests. Since INTERCEPT JR. uses main memory for both control panel as well as user routines, interrupt return addresses are saved in location 0000_8. Thus, if CPREQ were allowed to be active at all times, the user's device interrupt return address could be destroyed by a CPREQ. INTGNT is activated only by INTREQ and is reset by executing the first IOT instruction in the interrupt service routine. At this time, the CPREQ is allowed to get through, as long as the IOT did not disable the CP timer. If the user is implementing an interrupting device interface with PIE interrupts enabled, a single IOT would be used to reset INTGNT, disable CPREQ and get an interrupt vector from the PIE. At the conclusion of the service routine, CPREQ would be re-enabled under program control.

The monitor firmware will be more fully discussed in Chapter 8. For a more detailed discussion of the control panel capabilities of the IM6100, refer to the IM6100 brochure. INTERCEPT JR. uses the same memory address space for control panel, monitor functions and user memory. The monitor keeps track of memory use by periodically examining a status word. See the discussion on the monitor program for further details.

OPTIONS

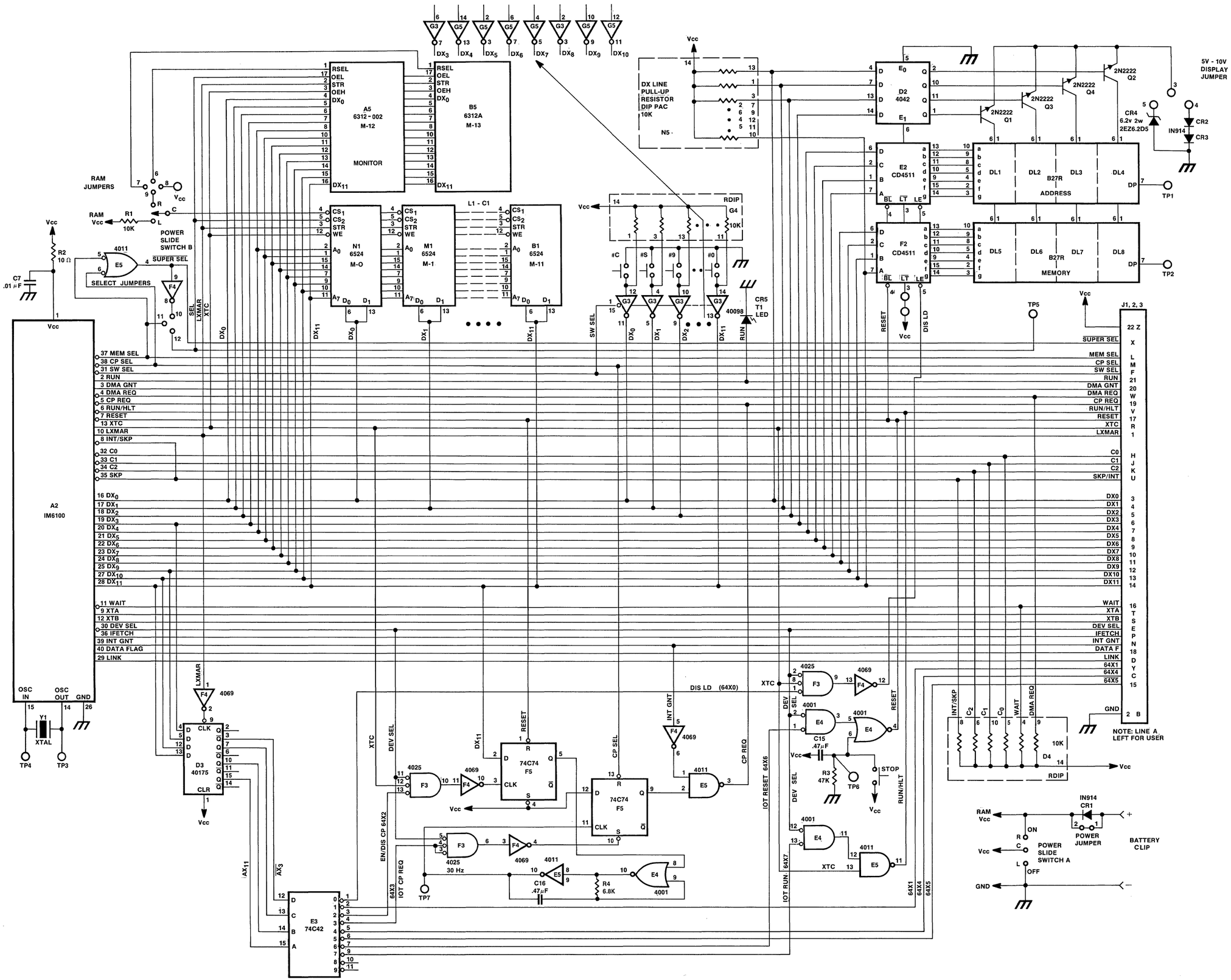
The user may put another IM6312 ROM in the second socket provided on the INTERCEPT JR. board. Extra decoders are not required. The second ROM could contain user and/or factory generated programs such as floating point math routines, I/O handlers, diagnostics programs, utilities, etc.

The following chapters will describe the optional boards that may be plugged into the 6950-INTERCEPT JR. to expand its capabilities The three connectors on the 6950 board are in parallel and bring out the DX bus, IM6100 control lines, select lines, power connections and unused IOT control lines from the 74C42 decoder (E3).

The basic 256 words of RAM may be disabled by tying chip select high through the jumper option pins provided. This is done when the 6951-M1KX12 JR. RAM MODULE board is to be mapped into the lower 1K field in 0000_8 to 1777_8.

The information in this manual and in the IM6100 Family brochure should help the user to design his own I/O interface boards if required.

INTERCEPT JR. MODULE SCHEMATIC


A2
IM6100
A5
6312 - 002
M-12
MONITOR
B5
6312A
M-13
N1
6524
M-O
M1
6524
M-1
B1
6524
M-11
L1 - C1
DX LINE
PULL-UP
RESISTOR
DIP PAC
10K
N5
RDIP
G4
10K
RAM
JUMPERS
RAM
Vcc
R1
10K
POWER
SLIDE
SWITCH B
SUPER SEL
SELECT JUMPERS
4011
E5
F4
R2
10 Ω
C7
.01 μF
Y1
XTAL
TP4
TP3
OSC
IN
OSC
OUT
GND
D3
40175
E3
74C42
F5
74C74
F3
4025
4069
E4
4001
E5
4011
C16
.47μF
R4
6.8K
C15
.47μF
R3
47K
TP6
TP7
CR5
T1
LED
40098
RUN
D2
4042
E2
CD4511
F2
CD4511
DL1
DL2
DL3
DL4
B27R
ADDRESS
DL5
DL6
DL7
DL8
B27R
MEMORY
DP
TP1
TP2
TP5
2N2222
Q1
2N2222
Q2
2N2222
Q3
2N2222
Q4
CR4
6.2v 2w
2EZ6.2D5
IN914
CR2
CR3
5V - 10V
DISPLAY
JUMPER
J1, 2, 3
22 Z
X
L
M
F
21
20
W
19
V
17
R
1
H
J
K
U
3
4
5
6
7
8
9
10
11
12
13
14
16
T
S
E
P
N
18
D
Y
C
15
2 B
SUPER SEL
MEM SEL
CP SEL
SW SEL
RUN
DMA GNT
DMA REQ
CP REQ
RUN/HLT
RESET
XTC
LXMAR
C0
C1
C2
SKP/INT
DX0
DX1
DX2
DX3
DX4
DX5
DX6
DX7
DX8
DX9
DX10
DX11
WAIT
XTA
XTB
DEV SEL
IFETCH
INT GNT
DATA F
LINK
64X1
64X4
64X5
GND
37 MEM SEL
38 CP SEL
31 SW SEL
2 RUN
3 DMA GNT
4 DMA REQ
5 CP REQ
6 RUN/HLT
7 RESET
13 XTC
10 LXMAR
8 INT/SKP
32 C0
33 C1
34 C2
35 SKP
16 DX0
17 DX1
18 DX2
19 DX3
20 DX4
21 DX5
22 DX6
23 DX7
24 DX8
25 DX9
27 DX10
28 DX11
11 WAIT
9 XTA
12 XTB
30 DEV SEL
36 IFETCH
39 INT GNT
40 DATA FLAG
29 LINK
INT/SKP
C2
C1
C0
WAIT
DMA REQ
10K
D4
RDIP
NOTE: LINE A
LEFT FOR USER
IN914
CR1
POWER
JUMPER
BATTERY
CLIP
RAM
Vcc
ON
POWER
SLIDE
SWITCH A
OFF
SEL
LXMAR
XTC
DX0
DX11
SW SEL
CP SEL
CP REQ
INT GNT
RESET
DIS LD (64X0)
EN/DIS CP 64X2
IOT CP REQ
64X3
IOT RESET 64X6
IOT RUN 64X7
DEV SEL
STOP
RUN/HLT
AX11
AX3

CHAPTER 5

JR. RAM MODULE

INTRODUCTION

The JR. RAM MODULE, 6951-M1KX12, pictured in Figure 5-1, allows the user to expand the complexity and size of the programs that may be written.

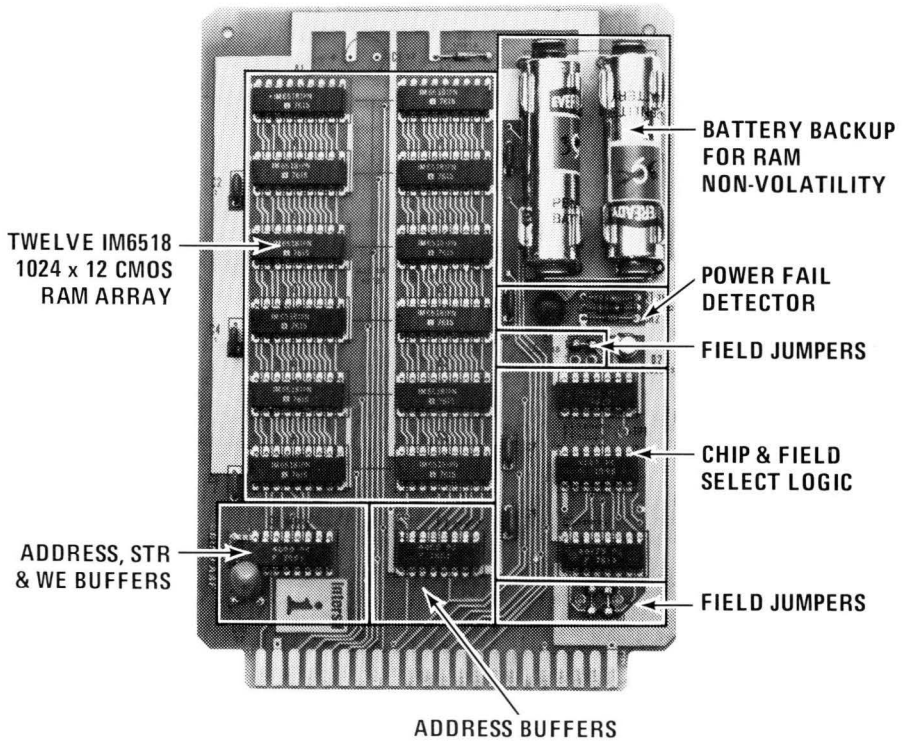


FIGURE 5-1

The board is fully nonvolatile using penlite chips to retain the RAM chips in the low power data retention mode. Thus, the user may write programs on a board, unplug it and use a different board without losing programs. The board may be mapped into memory space according to several jumper options. The board may also be configured as either an Instruction Field or a Data Field by jumper option. (Refer to the IM6100 brochure and Applications Bulletin M007)

DISCUSSION

Twelve (12) IM6518 CMOS RAM chips are used to implement the 1024 X 12 array for this board. The IM6518 is organized as 1024 X 1 with separate data-in and data-out pins and ten (10) address pins. (Refer to the IM6508/18 data sheet for further information.) INTERCEPT JR. uses a single bus for all address and data I/O, therefore, the DI and DO pins on the RAM chips are both connected to the respective DX line. The ten (10) address lines are buffered using ten gates from

CHAPTER 5

JR. RAM MODULE

INTRODUCTION

The JR. RAM MODULE, 6951-M1KX12, pictured in Figure 5-1, allows the user to expand the complexity and size of the programs that may be written.

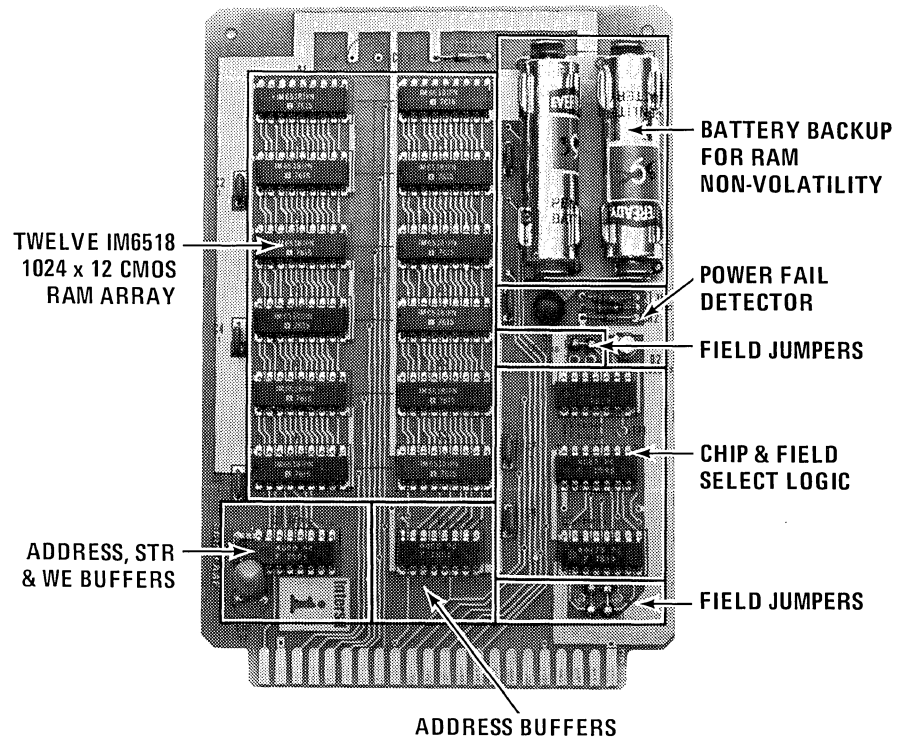


FIGURE 5-1

The board is fully nonvolatile using penlite chips to retain the RAM chips in the low power data retention mode. Thus, the user may write programs on a board, unplug it and use a different board without losing programs. The board may be mapped into memory space according to several jumper options. The board may also be configured as either an Instruction Field or a Data Field by jumper option. (Refer to the IM6100 brochure and Applications Bulletin M007)

DISCUSSION

Twelve (12) IM6518 CMOS RAM chips are used to implement the 1024 X 12 array for this board. The IM6518 is organized as 1024 X 1 with separate data-in and data-out pins and ten (10) address pins. (Refer to the IM6508/18 data sheet for further information.) INTERCEPT JR. uses a single bus for all address and data I/O, therefore, the DI and DO pins on the RAM chips are both connected to the respective DX line. The ten (10) address lines are buffered using ten gates from

two 34050 hex CMOS buffers (G1 and G2). Two gates are used to buffer LXMAR and XTC. These two signals, as previously explained in the discussion of the 6950 board, strobe memory addresses into the RAM chips and enable the chip for data write operations.

The $\overline{\text{SUP SEL}}$ signal is also buffered. This signal selects the RAM for both control panel and main memory use.

The two most significant bits of address are latched in the 340175 quad D-type latch (G3). This latch provides both true and complemented outputs, and, by connecting the appropriate jumpers to the 34023 three input NAND (F3), the 1K RAM field provided by the board may be mapped into any of the four 1K fields of the total 4K memory space addressable by the IM6100 microprocessor.

Since the highest 1K field is occupied by the MONITOR ROM and 256 words of RAM are provided in the lower 1K field by the 6950 module, normally the jumper should be placed to map the RAM into one of the middle 1K areas, for example 2000_8-3777_8 or 4000_8-5777_8.

If these two fields are being allocated for the PROM board, 6952, the RAM may be mapped into the 1K base field in which it will overlay the 256 words provided in the 6950 board. This will provide the additional 768 words that would otherwise be unobtainable.

Table 5-1 provides the jumper connections for different mappings.

TABLE 5-1

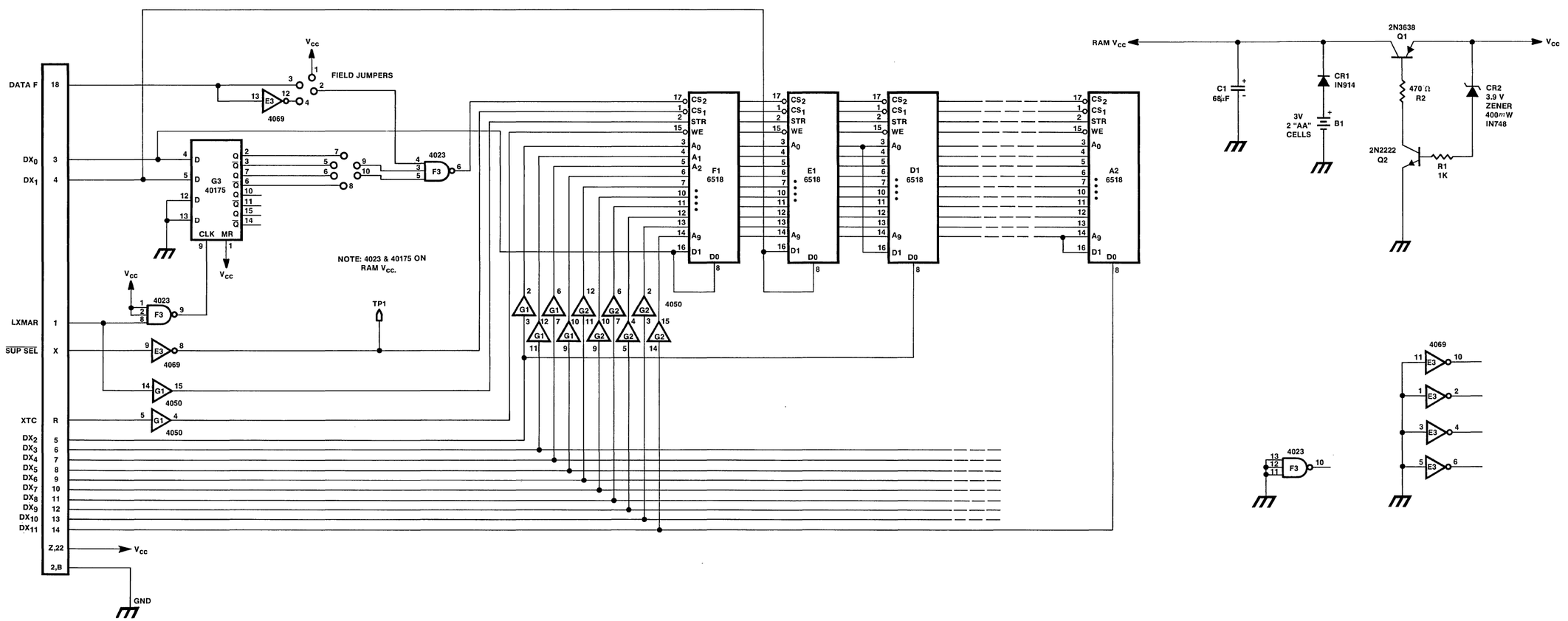
Desired Mapping	Strap* Pins 9, 10
0-1777	To Pins 5 & 8
2000-3777	To Pins 5 & 6
4000-5777	To Pins 7 & 8
6000-7777	To Pins 7 & 6

* These strapping option pins are numbered and located between the 340175 at G3 and the connector pins. As an example, for mapping 2000-3777, pins 9 and 5 should be strapped and pins 10 and 6 must be strapped, or alternatively, pins 9 and 6 strapped together and pins 10 and 5 strapped together.

The board may also be configured to be either an instruction field or a data field by an appropriate jumper connected to the DATAF pin. Normally, the field jumper from test point 2 is connected to V_{CC} and distinctions are not made between IF and DF. These distinctions are usually required only in extended memory systems (Refer to Applications Bulletin M007)

The RAM on this board may be made nonvolatile by using two "AA" type penlite cells in the chips provided. If V_{CC} from the "D" cells falls below 3.9 volts, the zener diode CR2 turns off, turning off transistor Q2, which in turn cuts off the series transistor Q1. Diode CR1 becomes forward biased, and the "AA" cells power the RAM array in the data retention mode.

JR. RAM MODULE SCHEMATIC


DATA F
DX0
DX1
LXMAR
SUP SEL
XTC
DX2
DX3
DX4
DX5
DX6
DX7
DX8
DX9
DX10
DX11
Z,22
2,B
GND
Vcc
FIELD JUMPERS
4069
G3
40175
CLK MR
4023
F3
NOTE: 4023 & 40175 ON
RAM Vcc.
TP1
4050
F1
6518
E1
6518
D1
6518
A2
6518
CS2
CS1
STR
WE
A0
A1
A2
A9
D1
D0
RAM Vcc
2N3638
Q1
C1
68µF
CR1
IN914
3V
2 "AA"
CELLS
B1
470 Ω
R2
CR2
3.9 V
ZENER
400mW
IN748
2N2222
Q2
R1
1K

CHAPTER 6
JR. P/ROM MODULE

INTRODUCTION

The JR. P/ROM MODULE, 6952-P2KX12, pictured in Figure 6-1, enables user developed programs to be stored in user programmable read only memory.

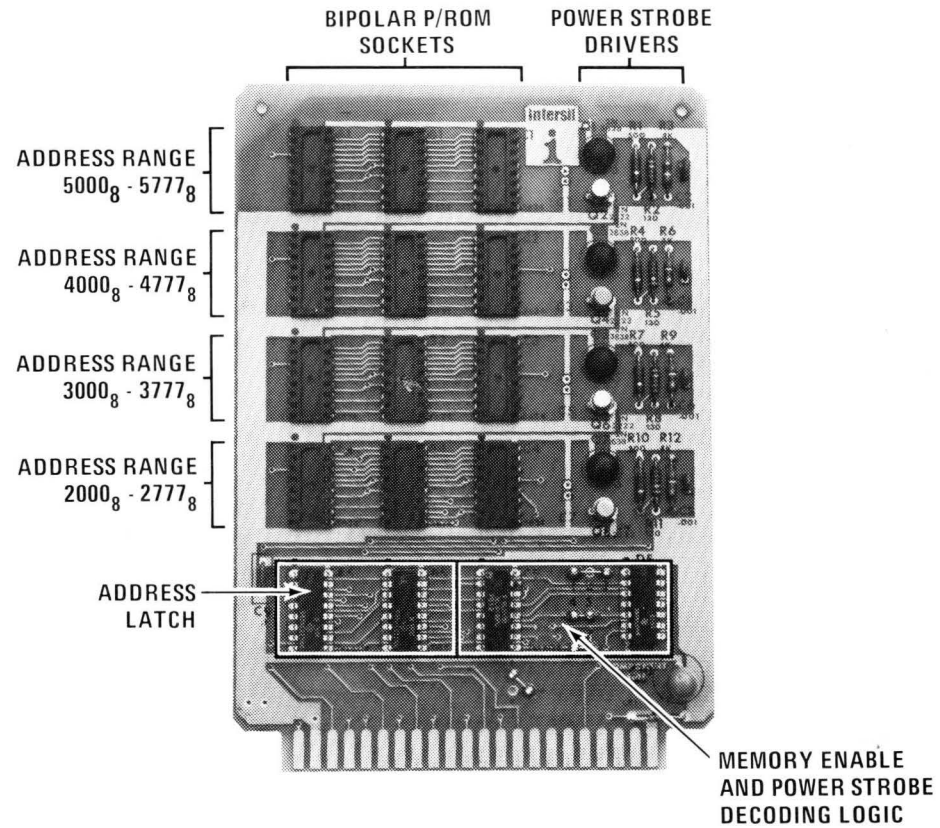


Figure 6-1

The user has the option of utilizing the IM5623, 256 X 4, or IM5624, 512 X 4, three-state output Avalanche Induced Migration (AIM) programmable bipolar P/ROMs to obtain from 256 to 2048 words of program. Power dissipation is minimized by supplying power, via the POWER STROBE DRIVERS, only to those P/ROMs which are enabled. ADDRESS LATCH, MEMORY ENABLE AND POWER STROBE DECODING LOGIC are pictured in Figure 6-1.

The figure shows the address range for IM5624, 512 X 4 P/ROMs. For the user's convenience, the address range for the IM5623, 256 X 4, P/ROM and IM5624 are shown in TABLE 6-1. The user should change address range, as required, when mixing IM5623 and IM5624 on a given module.

TABLE 6-1

ADDRESS RANGE IN OCTAL IM5623/IM5624

IM5623 (256 X 4)	IM5624 (512 X 4)
2000-2377	2000-2777
3000-3377	3000-3777
4000-4377	4000-4777
5000-5377	5000-5777

DISCUSSION

This text should be used in conjunction with the enclosed schematic for a complete understanding of the 6952-P2KX12 JR. P/ROM MODULE.

The memory address is latched from the DX bus by the two 74LS174 hex latches when they are strobed by LXMAR.

The lower nine bits of the address go to the address inputs of all the twelve P/ROMs, which are arranged in a matrix of four rows of three.

The higher order three bits of the address are decoded by the 74LS138, and it generates a chip enable to the appropriate row of P/ROMs. This chip enable is also used to turn on the two transistors in the appropriate power strobe circuit in order to connect V_{CC} (less a $V_{CE(SAT)}$) to the power pins of the enabled row of P/ROMs. There is no delay penalty in power strobing because the bipolar P/ROMs are much faster than required by the CMOS processor. The average power dissipation is reduced to approximately 5% of the non-strobed case. With the chip enable high, the P/ROM outputs are in a high impedance state permitting XTC to be used as one of the signals enabling the 74LS138 decoder. The P/ROM outputs, therefore, may be directly connected to the DX bus. The XTC line signals the read and write phases of the memory cycle. Thus, XTC when high, enables decoder pin G1 during the time that the address is latched into the 74LS174's, and remains enabled during the time the address is decoded, the P/ROMs are enabled, strobed and accessed. XTC goes low during the second half of the memory cycle, disabling the P/ROMs.

Decoder pin G2A is enabled only during the $\overline{SUP}$ $\overline{SEL}$ time, that is, when either MEMSEL or CPSEL is active. Therefore, the memory is really powered only for three clock cycles.

The uppermost 1K of memory is in the monitor ROM on the processor board, so the decoder does not use the pins for a decoded zero and one.

In the event that extended memory is used, the DATAF (DATA Field) pin is jumpered to the G2B enable pin of the 74LS138 decoder. This signal is normally low, enabling the decoder, and is activated to the high state during the execute phase of indirectly addressed AND, TAD, ISZ and DCA instructions (see IM6100 data sheet) so that data transfers are controlled by the Data Field, DF, and not the Instruction Field, IF, when addressing more than 4K words. Otherwise, the G2B pin may be left grounded by a jumper.

Table 6-1 shows the address space occupied by the P/ROMs. The user must supply at least three P/ROMs and can use them anywhere in the address space provided.

JR. P/ROM MODULE SCHEMATIC

POWER STROBE REPEATED 4 TIMES

POWER STROBE REPEAT

POWER STROBE REPEAT

POWER STROBE REPEAT

Vcc

R10 500Ω

R11 130Ω

R12 4K

Q7 2N3638

Q8 2N2222

C_8 .001 μF

C_7

IN

OUT

D5 74LS04

J1

XTC R

SUP SEL X

DATA F 18

Vcc 22, Z

Gnd 2, B

DX_0 3

DX_1 4

DX_2 5

DX_3 6

DX_4 7

DX_5 8

DX_6 9

DX_7 10

DX_8 11

DX_9 12

DX_{10} 13

DX_{11} 14

LXMAR 1

INSTUCTION FIELD JUMPER

C5 74LS138

MONITOR JUMPERS

74LS174 A5

74LS174 B5

74LS04

5624 A4

5624 C4

5624 B4

2000 - 2777

5624 A3

5624 C3

5624 B3

3000 - 3777

5624 A2

5624 C2

5624 B2

4000 - 4777

5624 A1

5624 C1

5624 B1

5000 - 5777

CHAPTER 7

JR. SERIAL I/O MODULE

INTRODUCTION

The JR. SERIAL I/O MODULE, 6953-PIEART, pictured in Figure 7-1, allows the user to communicate with a 110 baud full duplex terminal with either an EIA RS-232C type differential voltage interface or a 20 mA current loop interface.

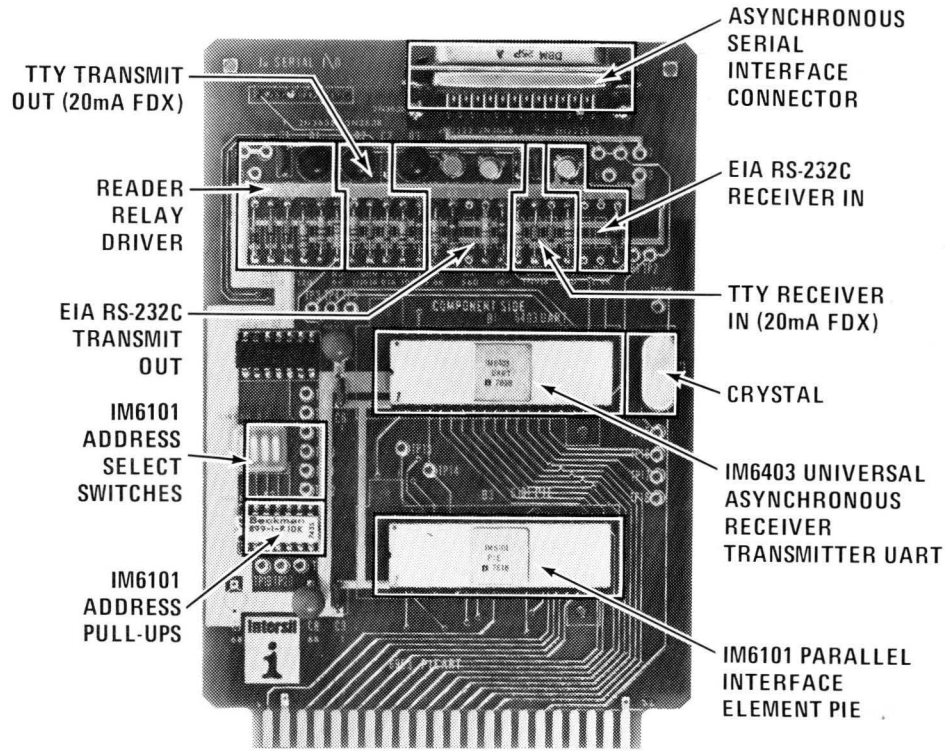


Figure 7-1

This board uses two CMOS LSI chips, the IM6101 Programmable Interface Element (PIE) and the IM6403 Universal Asynchronous Receiver/ Transmitter (UART). The MONITOR ROM provided with the 6950-INTERCEPT JR. MODULE contains a bootstrap loader for loading programs from the 6953-PIEART using BIN formatted media, such as paper tape punched out by the 6950-INTERCEPT JR. via the 6953-PIEART and an ASR-33 Teletype using the Memory Dump routines contained in the MONITOR ROM. This allows the user to create programs, dump them out on paper tape and use them at a later date by simply reading the tape back in.

DISCUSSION

The data sheets on the PIE and UART should be studied in order to fully understand the description of the operation of this module.

It will also be beneficial to study the listing of the PIE-UART routines in the MONITOR ROM. These routines are listed in line numbers 1171 through 1511.

The PIE address used is 00111, therefore, all IOT instructions to the PIE are of the form 616X or 617X in octal.

By using a UART, the amount of code required to do serial I/O is considerably reduced because bit timing is taken care of by the UART. Also, the programs become insensitive to the CPU clock frequency. Both the PIE (B3) and the UART (B1) are general purpose programmable devices and, therefore, need to be programmed or initialized to specific system requirements.

Some functions are programmed by hardwired pin connections and others by MONITOR ROM firmware routines.

The DIP switch is set up to program the PIE SEL 3-7 inputs to the address 00111. It also grounds CNTRL pin 2 of the 6403 UART selecting the internal 11 stage divider. This divider's output is the 16X clock used by the receiver register and transmitter register. The 6403 is designed to be directly clocked by a crystal. The crystal used is a TV colorburst crystal of 3,579,545 Hz. When this is divided by 2^{11} and 16, the baud rate of 109.2 Hz is within the tolerance limits of a 110 baud Teletype interface. The DIP package of 10K resistors (A3) pulls up the SEL 5, 6, 7 inputs and the PIE series priority input pin 3. The PIE control registers A and B and the vector register are initialized by the INPIE routine in firmware. Table 7-1 shows the constants loaded into these registers.

TABLE 7-1

CONTROL REGISTER A

0	1	2	3	4	5	6	7	8	9	10	11
FL4	FL3	FL2	FL1	WP2	•	WP1	•	IE4	IE3	IE2	IE1
1	1	1	0	1	0	0	0	0	0	0	0

FL 2, 3, 4 — bits set high cause the unused FLAG outputs 2, 3, 4 to be at high level

FL 1 — bit set low causes FLAG output 1 (Reader Run Relay Flag) to be at low level

WP 2 — set high means positive WRITE POLARITY or positive pulses at WRITE output 2 (used to load the UART CONTROL REGISTER)

WP 1 — set low causes negative pulses at WRITE output 1 (used to load the UART TRANSMITTER BUFFER REGISTER from the data inputs)

IE 1, 2, 3, 4 — set at 0 disables all PIE interrupts.

TABLE 7-2

CONTROL REGISTER B

0	1	2	3	4	5	6	7
SL4	SL3	SL2	SL1	SP4	SP3	SP2	SP1
0	0	1	1	0	1	1	1

NOTE:

1. Sense input S4 is not used, therefore, SL4 and SP4 bits are irrelevant.

2. SL 3 = 0 and SP 3 = 1 program the SENSE3 flip flop to be set by a positive going edge. SENSE3 is connected to the serial data input of the UART and is used for start bit detection.

3. SL 2 = 1 and SP 2 = 1 program the SENSE2 flip flop to be set by a high level. SENSE2 is connected to the TRANSMITTER BUFFER REGISTER EMPTY (TBRE) output of the UART which indicates that the UART transmitter is ready for new data. The TBRE signal is a high level.

4. SL 1 = 1 and SP 1 = 1 program the SENSE1 flip flop to be set by a high level. SENSE1 is connected to the DATA READY (DR) output of the UART, which is a high level indicating that a character has been received and transferred to the receiver buffer register.

TABLE 7-3

VECTOR REGISTER

0	1	2	3	4	5	6	7	8	9	10	11
INTERRUPT VECTOR										VPR1	
0	0	0	0	0	0	0	0	0	0	0	0

NOTE: The PIE interrupts are disabled in this application, and the sense flip flops are tested by the firmware with SKIP instructions.

The PIE's READ2 output is unused and the READ1 output is connected to the UART RECEIVER REGISTER DISABLE (RRD) and DATA RECEIVED RESET (DRR, an active low input) so that when a received character is ready, R1 which is normally high (keeping the RECEIVER REGISTER disabled) pulses low during IOTA·$\overline{\text{DEVSEL}}$, transferring the receiver data to the IM6100 via the DX bus while simultaneously clearing the DR flag in readiness for the next character.

The UART is also initialized both via hardwired connections and under program control.

STATUS FLAGS DISABLE (SFD pin 16) is grounded to enable all UART status flags. The UART CONTROL REGISTER bits are loaded from the DX bus as shown in Table 7-4.

TABLE 7-4

DX Lines	0	1	2	3	4
Designations	PI	SBS	EPE	CLS1	CLS2
Constant	1	1	1	1	1

PI = 1 — PARITY INHIBIT - Parity generation and checking is inhibited and PARITY ERROR (PE) output is forced low.

SBS = 1 — STOP BIT SELECT - In conjunction with CLS1 and CLS2, this selects two (2) stop bits.

EPE = 1 — EVEN PARITY ENABLE - Irrelevant as parity is inhibited.

CLS1 = 1)
CLS2 = 1) — CHARACTER LENGTH SELECTED - These bits select on eight-bit character.

All unused pins are brought out to test points, to facilitate experiments by the user.

The UART TBR parallel data input bus and RBR parallel data output bus are connected to DX4-11.

The serial input and output pins of the UART go to both EIA-RS-232C and 20 mA current loop interface drivers and receivers.

Table 7-5 shows the connector and jumper options for the two interfaces.

Serial output bits from the UART cause the push-pull EIA driver to switch between V_{CC} and -12 volt (-12 volt must be provided externally) and transistor Q2 to supply 25 mA nominally (5 volt ÷ (R5 + R4)) to the current loop interface.

Briefly, the PIEART interface works as follows once the interface is initialized. When transmitting to a terminal, the IM6100 executes a waiting loop using a SKIP on SENSE2 instruction followed by a jump back. SENSE2 as shown in Table 7-3 is set when the TRANSMITTER BUFFER is empty. When the character has been transmitted, the waiting loop is exited and a WRITE1 instruction is executed writing a new character into the UART transmit buffer. The PIE strobes the DX bus at the proper time when this instruction is performed.

When receiving from a terminal, the IM6100 resets the SENSE3 flip flop by executing a SKIP on SENSE3 instruction. This flip flop senses the start bit of a character. The READER RUN flag is set by executing a SET FLAG 1 instruction to the PIE. Now the interface is ready for a character from either a tape reader or a keyboard and a wait loop is entered. This loop is exited when a start bit is detected and the READER RUN flag is cleared just in case the data source was a reader. This stops the reader from advancing until the CPU is ready for another character. Another wait loop is entered and this time it is exited when the DATA RECEIVED flag goes true, setting the SENSE1 flip flop. The accumulator may then be cleared and a READ1 command executed. This causes the PIE to enable the UART receiver buffer onto the DX bus, simultaneously clearing the DR flag.

When reading BIN tape, the above transmit and receive program sequences are called as subroutines, while the main program performs functions such as testing characters for a rubout, accumulating checksums, testing for leader-trailer, etc. (Refer to MONITOR description)

Whenever SKIP on SENSE flip flop instructions are executed, the PIE will test the state of the desired flip flop and, if it has been set, it will assert the SKP/INT output causing the IM6100 to skip the next instruction. The sense flip flop is then cleared. For more details, refer to the PIE data sheet.

TABLE 7-5

20 mA LOOP/EIA RS232-C CONNECTOR PINOUTS

OPTION	STANDARD CONNECTION	MODIFIED CONNECTION
Voltage Change Option	+5 VDC on V_{CC} Connect points #1 and #2	+10 VDC on V_{CC} Cut between points #1 and #2 and connect points #1 and #3
Driver/Receiver Change Option	20 mA loop Connect points #4 and #5	EIA RS232-C Cut between points #4 and #5 and connect points #5 and #6
EIA Earth Ground Option	No EIA Earth ground	To connect Earth ground, tie points #7 and #8 together

CONNECTOR PINOUTS

20 mA Loop

Pin	Signal
1	XMIT+
2	KEY
3	XMIT-
4	RCVE+
5	RCVE-
6	RDR+
7	RDR-
8	-12 VDC
9	N.C.
10	N.C.

EIA RS232-C

Pin	Signal
1	Earth Ground
2	XMIT
3	RCVE
7	Signal Ground
18	-12 VDC

All others are N.C.

Pins 5 (Clear to Send)
6 (Data Set Ready)
8 (Received Line Signal Detector)
may have to be tied to VCC with some terminals

In order to use the module, it must first be connected to a serial ASCII 110 baud tape reader, typically an ASR-33 Teletype equipped with the reader. The connection is done by a cable connecting the 20 mA loop connector pins to the Teletype terminal strip (Figures 7 and 8 of Intersil Applications Bulletin M005 "Teletype Interface for the IM6100 Microprocessor"). The external -12 VDC supply is connected and the Teletype is turned to the LINE position.

Note that the Teletype must be equipped for 20 mA full duplex operation and should have a reader run relay installed.

To read BIN format tape, the tape is placed in the reader, the key is put in the START position and the sequence CNTRL 1 is pressed on the INTERCEPT JR.

As explained on page 2-5, this function will activate the loader. At the end of the load sequence, the machine is halted showing the AC (SAVAC location 0140) whose contents represent the checksum and should be zero for a valid load.

To dump memory onto tape, the starting and ending address of the block should be entered into locations 0176 and 0177 and the program run starting at location 7510. Naturally, the tape punch should be turned on.

Chapter 8 page 14 describes these routines in more detail.

Table 7-6 lists the PIE-UART instructions as used by the MONITOR. These instructions are also listed in the program listing on page 8-16C.

TABLE 7-6

PIE-UART INSTRUCTIONS

6160	READ1	(Reset UART Data Received Flag and read received character)
6170	READ2	(Generate read strobe 2) - Not used
6161	WRITE1	(Load UART Transmit Buffer)
6171	WRITE2	(Load UART Control Register)
6162	SKIP1	(Test state of sense FF1; skip if set by UART Data Received Flag)
6163	SKIP2	(Test state of sense FF2; skip if set by UART Transmit Buffer Empty Flag)
6172	SKIP3	(Test state of sense FF3; skip if set by START bit)
6173	SKIP4	(Test state of sense FF4; skip if set) - Not used

6164	RCRA	(Read control register A)
6165	WCRA	(Write control register A)
6175	WCRB	(Write control register B)
6174	WVR	(Write vector register)
6166	SFLAG1	(Set FLAG 1) - Reader Relay Flag - ON
6176	SFLAG3	(Set FLAG 3)
6167	CFLAG1	(Clear FLAG 1) - Reader Run Relay Flag - OFF
6177	CFLAG3	(Clear FLAG 3)

In addition to these, the IM6100 internal IOT instruction 6007_8 or CAF (Clear All Flags) clears the sense flip-flop thus clearing all interrupt requests.

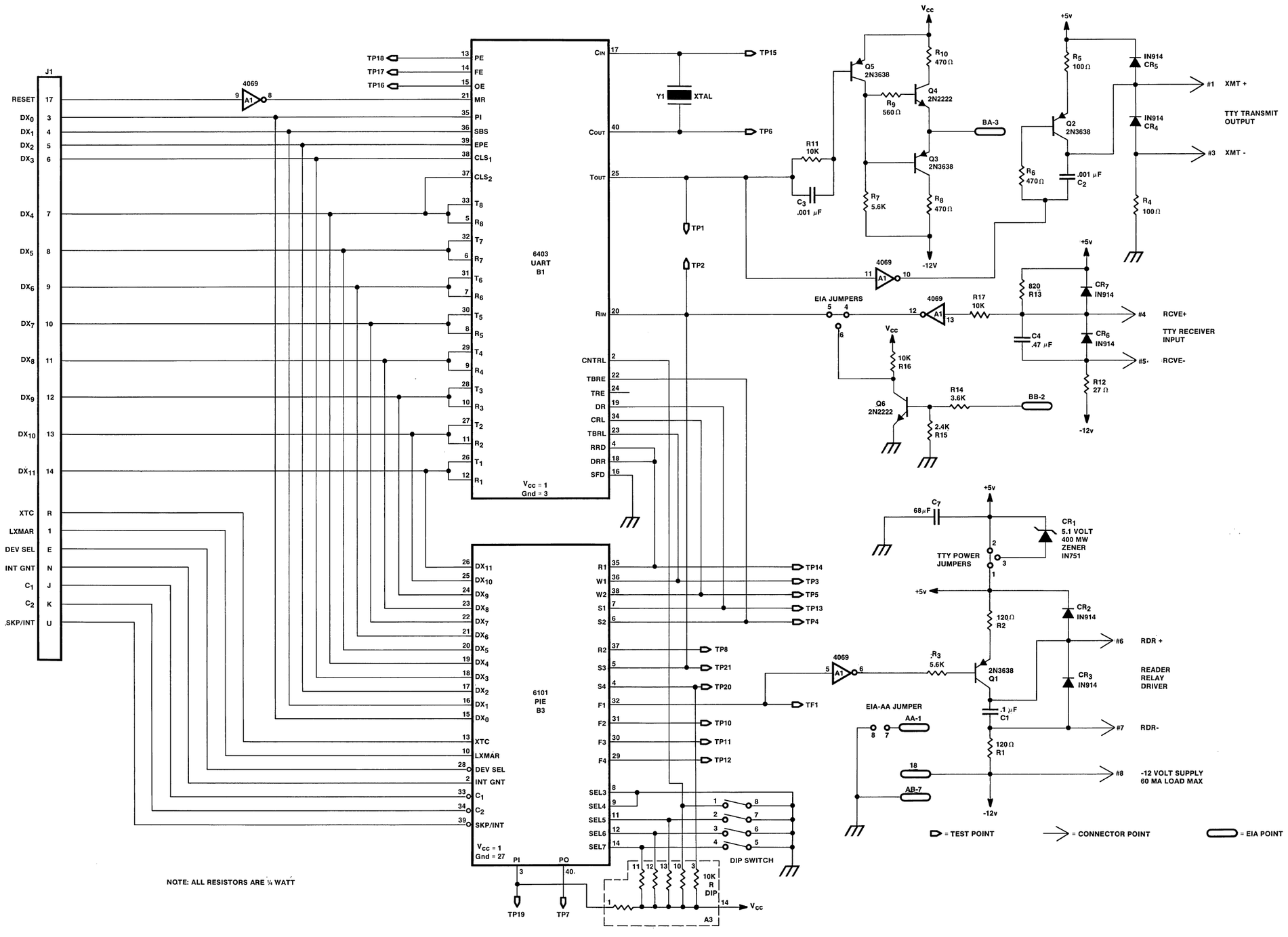
The serial I/O module is typically used with the INTERCEPT JR. BINARY LOADER and MEMORY DUMP routines in order to read BIN format tape and dump a block of memory onto BIN formatted tape.

The PIE-UART interface is initialized only when the BIN and DUMP programs are used. The user has access to these routines via the software subroutine call stacking mechanism in case the serial port is to be used for other purposes, such as printing characters on the Teletype.

The user may also write his own code in RAM for interface utilization and handling Teletype I/O.

Example 14 in Chapter 3 shows how the MONITOR subroutine may be called to implement Teletype keyboard and printer operation.

JR. PIEART SERIAL I/O MODULE SCHEMATIC



NOTE: ALL RESISTORS ARE ¼ WATT

CHAPTER 8

INTERCEPT JR. TUTORIAL SYSTEM MONITOR PROGRAM

The MONITOR uses main memory to store control panel routines in order to keep the system inexpensive. The IM6100 architecture, however, will allow control panel programs to exist in separate memory totally transparent to the user.

Figure 8-1 shows the memory allocation map for INTERCEPT JR.

The MONITOR uses several locations in page 0. These are listed in the program.

Some of these locations, SAVAC, SAVMQ, SAVFL in locations 0140_8, 0141_8, 0142_8, are used by the MONITOR to store IM6100 registers and flags and enable the user to conveniently examine and alter these registers.

Most other locations are used as temporary workspace by the MONITOR routines. Locations 167 to 177 are used as a software stack for subroutine return addresses.

The stack is initialized on power-up and on every pass through the control panel interrupt service routine.

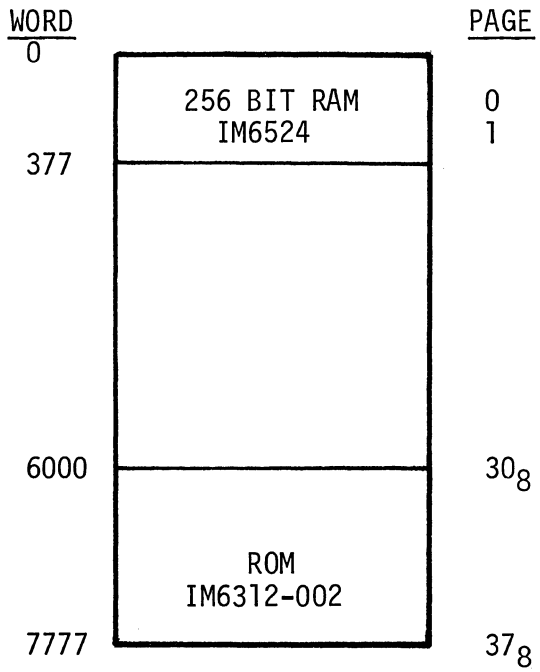
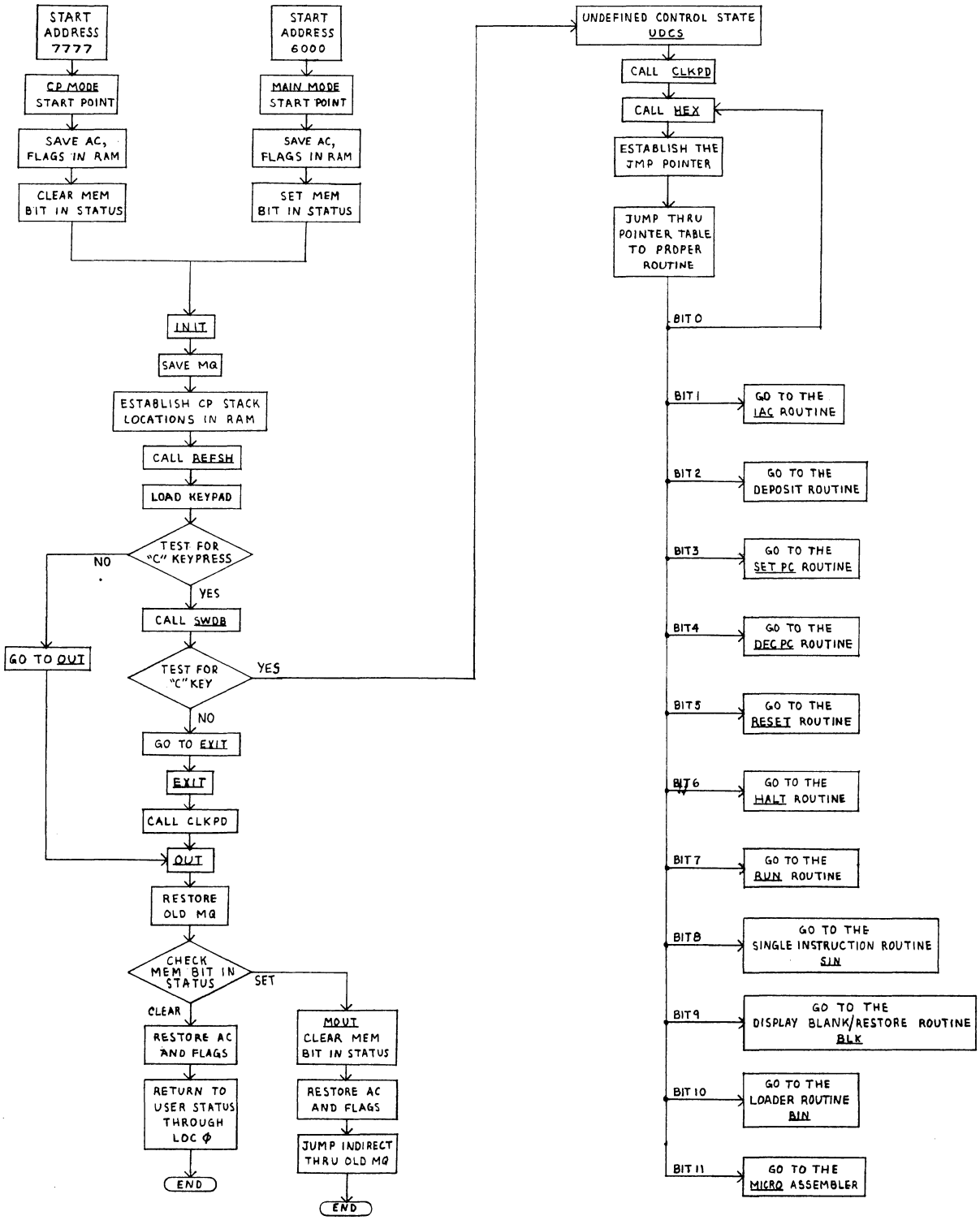
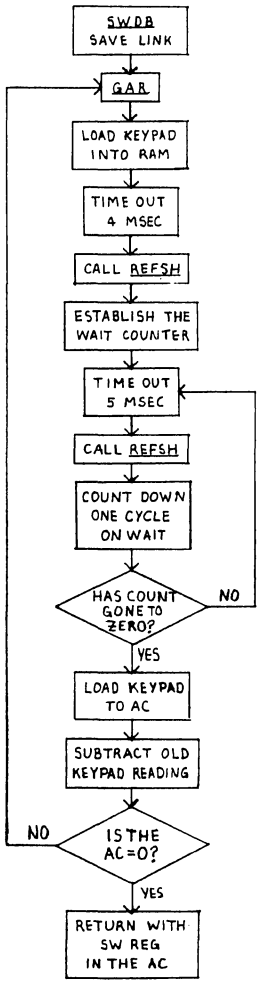
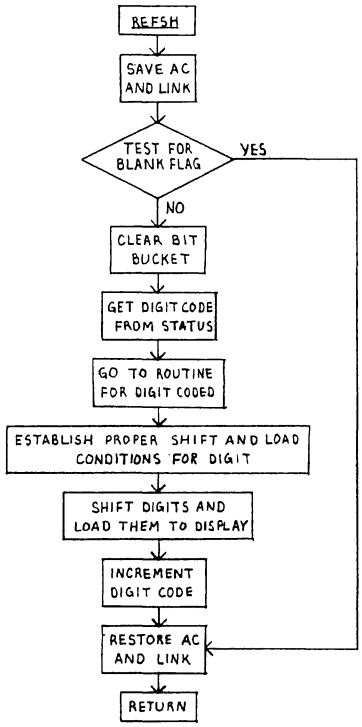
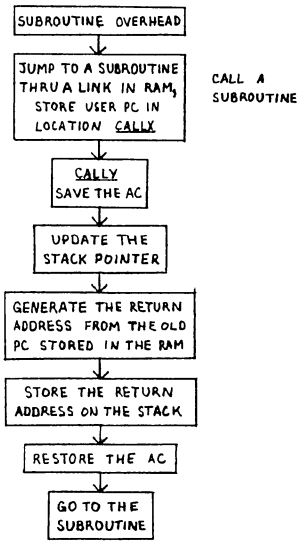


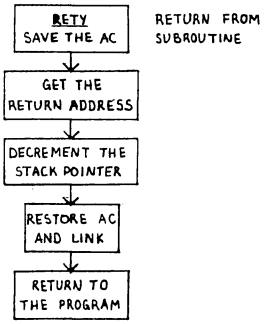
FIGURE 8-1

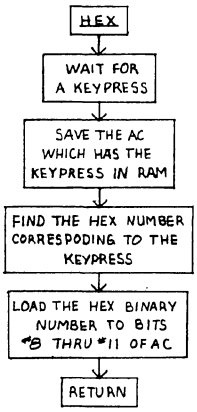

START ADDRESS 7777
CP MODE START POINT
SAVE AC, FLAGS IN RAM
CLEAR MEM BIT IN STATUS
START ADDRESS 6000
MAIN MODE START POINT
SAVE AC, FLAGS IN RAM
SET MEM BIT IN STATUS
INIT
SAVE MQ
ESTABLISH CP STACK LOCATIONS IN RAM
CALL REFSH
LOAD KEYPAD
TEST FOR "C" KEYPRESS
NO
YES
CALL SWDB
GO TO OUT
TEST FOR "C" KEY
YES
NO
GO TO EXIT
EXIT
CALL CLKPD
OUT
RESTORE OLD MQ
CHECK MEM BIT IN STATUS
SET
CLEAR
RESTORE AC AND FLAGS
RETURN TO USER STATUS THROUGH LOC 0
END
MQUT
CLEAR MEM BIT IN STATUS
RESTORE AC AND FLAGS
JUMP INDIRECT THRU OLD MQ
END
UNDEFINED CONTROL STATE
UDCS
CALL CLKPD
CALL HEX
ESTABLISH THE JMP POINTER
JUMP THRU POINTER TABLE TO PROPER ROUTINE
BIT 0
BIT 1
GO TO THE IAC ROUTINE
BIT 2
GO TO THE DEPOSIT ROUTINE
BIT 3
GO TO THE SET PC ROUTINE
BIT 4
GO TO THE DEC PC ROUTINE
BIT 5
GO TO THE RESET ROUTINE
BIT 6
GO TO THE HALT ROUTINE
BIT 7
GO TO THE RUN ROUTINE
BIT 8
GO TO THE SINGLE INSTRUCTION ROUTINE SIN
BIT 9
GO TO THE DISPLAY BLANK/RESTORE ROUTINE BLK
BIT 10
GO TO THE LOADER ROUTINE BIN
BIT 11
GO TO THE MICRO ASSEMBLER

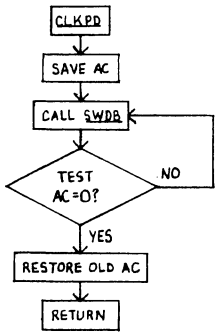


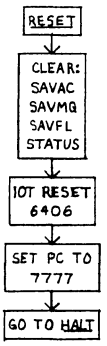






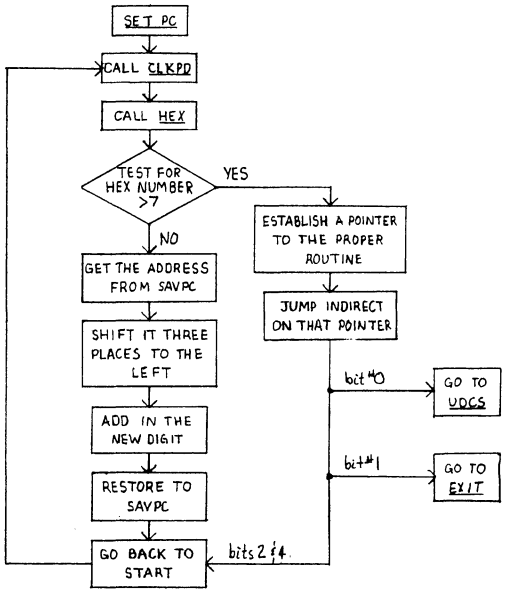


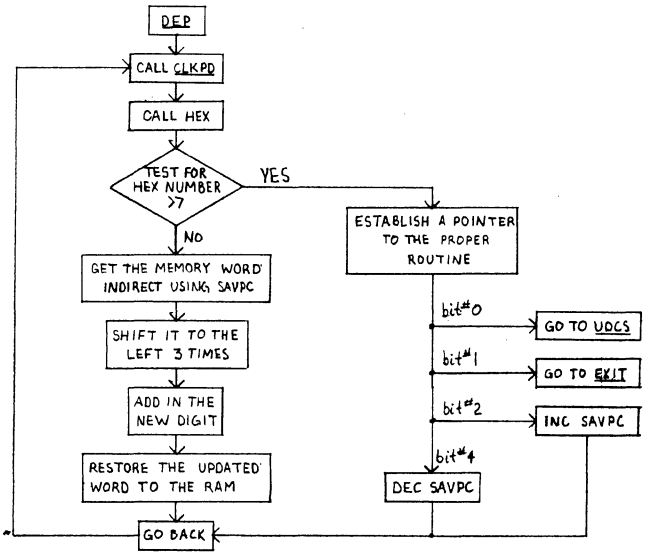


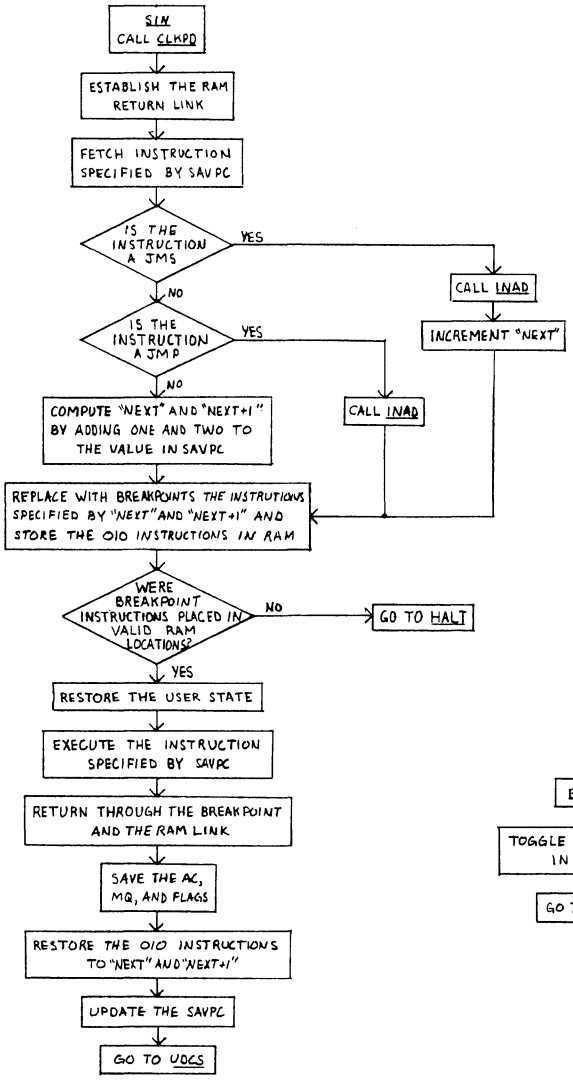


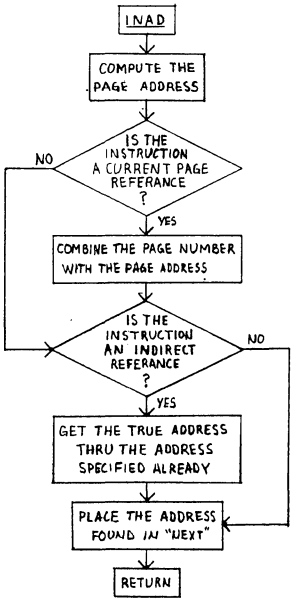
VALUE TABLE FOR HEX

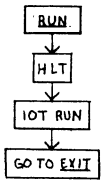
BUTTON	SW REG BIT #	HEX VALUE
RED	0	B
YELLOW	1	A
"MEM"	2	9
"DEC PC"	4	8
7	3	7
6	5	6
5	6	5
4	7	4
3	8	3
2	9	2
1	10	1
0	11	0

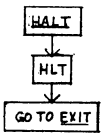

SET PC
CALL CLKPD
CALL HEX
TEST FOR HEX NUMBER >7
YES
NO
ESTABLISH A POINTER TO THE PROPER ROUTINE
JUMP INDIRECT ON THAT POINTER
GET THE ADDRESS FROM SAVPC
SHIFT IT THREE PLACES TO THE LEFT
ADD IN THE NEW DIGIT
RESTORE TO SAVPC
GO BACK TO START
bits 2 & 4
bit #0
GO TO UDCS
bit #1
GO TO EXIT

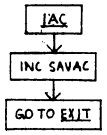

DEP
CALL CLKPD
CALL HEX
TEST FOR HEX NUMBER >7
YES
NO
ESTABLISH A POINTER TO THE PROPER ROUTINE
GET THE MEMORY WORD INDIRECT USING SAVPC
SHIFT IT TO THE LEFT 3 TIMES
ADD IN THE NEW DIGIT
RESTORE THE UPDATED WORD TO THE RAM
GO BACK
bit #0
GO TO UDCS
bit #1
GO TO EXIT
bit #2
INC SAVPC
bit #4
DEC SAVPC

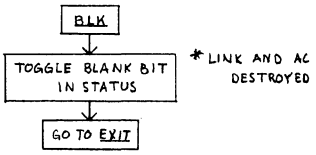

SIN
CALL CLKPD
ESTABLISH THE RAM RETURN LINK
FETCH INSTRUCTION SPECIFIED BY SAVPC
IS THE INSTRUCTION A JMS
YES
NO
CALL INAD
INCREMENT "NEXT"
IS THE INSTRUCTION A JMP
YES
NO
CALL INAD
COMPUTE "NEXT" AND "NEXT+1" BY ADDING ONE AND TWO TO THE VALUE IN SAVPC
REPLACE WITH BREAKPOINTS THE INSTRUTIONS SPECIFIED BY "NEXT" AND "NEXT+1" AND STORE THE OIO INSTRUCTIONS IN RAM
WERE BREAKPOINT INSTRUCTIONS PLACED IN VALID RAM LOCATIONS?
NO
GO TO HALT
YES
RESTORE THE USER STATE
EXECUTE THE INSTRUCTION SPECIFIED BY SAVPC
RETURN THROUGH THE BREAKPOINT AND THE RAM LINK
SAVE THE AC, MQ, AND FLAGS
RESTORE THE OIO INSTRUCTIONS TO "NEXT" AND "NEXT+1"
UPDATE THE SAVPC
GO TO UDCS

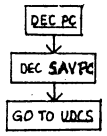

INAD
COMPUTE THE PAGE ADDRESS
IS THE INSTRUCTION A CURRENT PAGE REFERANCE ?
NO
YES
COMBINE THE PAGE NUMBER WITH THE PAGE ADDRESS
IS THE INSTRUCTION AN INDIRECT REFERANCE ?
NO
YES
GET THE TRUE ADDRESS THRU THE ADDRESS SPECIFIED ALREADY
PLACE THE ADDRESS FOUND IN "NEXT"
RETURN

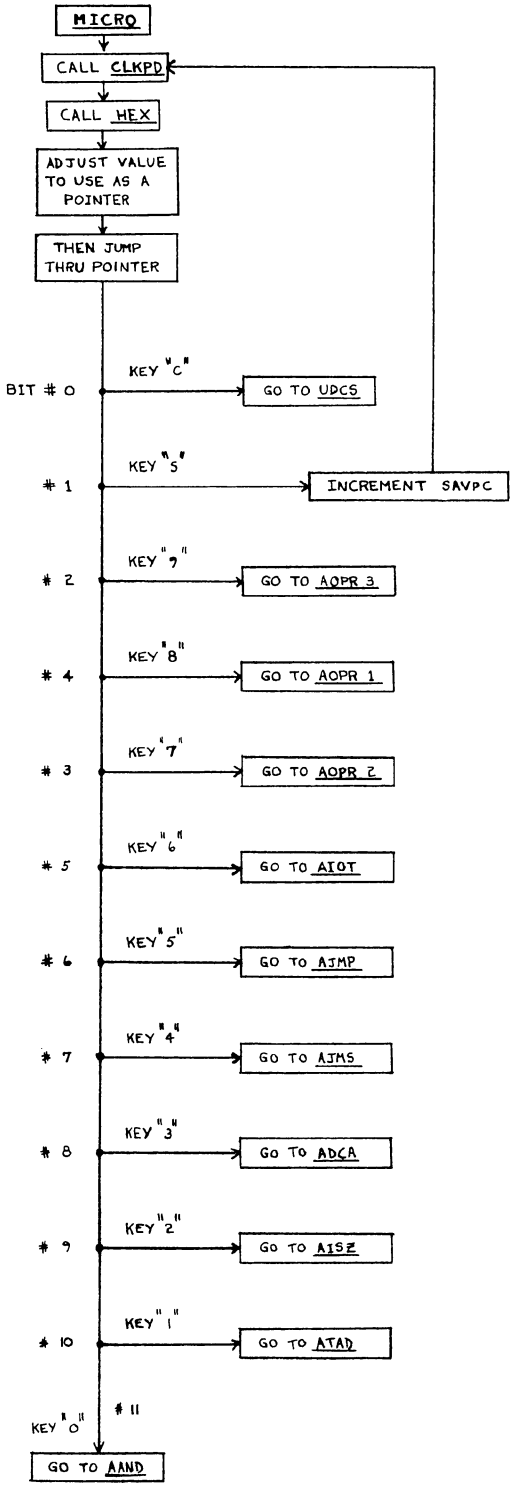

RUN
HLT
IOT RUN
GO TO EXIT

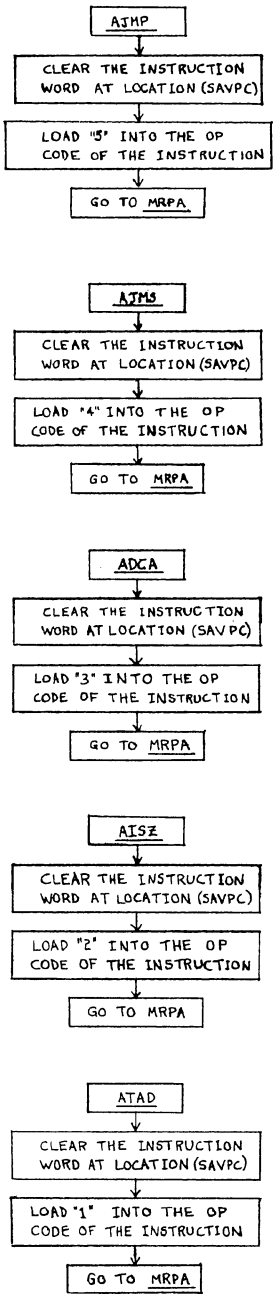

HALT
HLT
GO TO EXIT

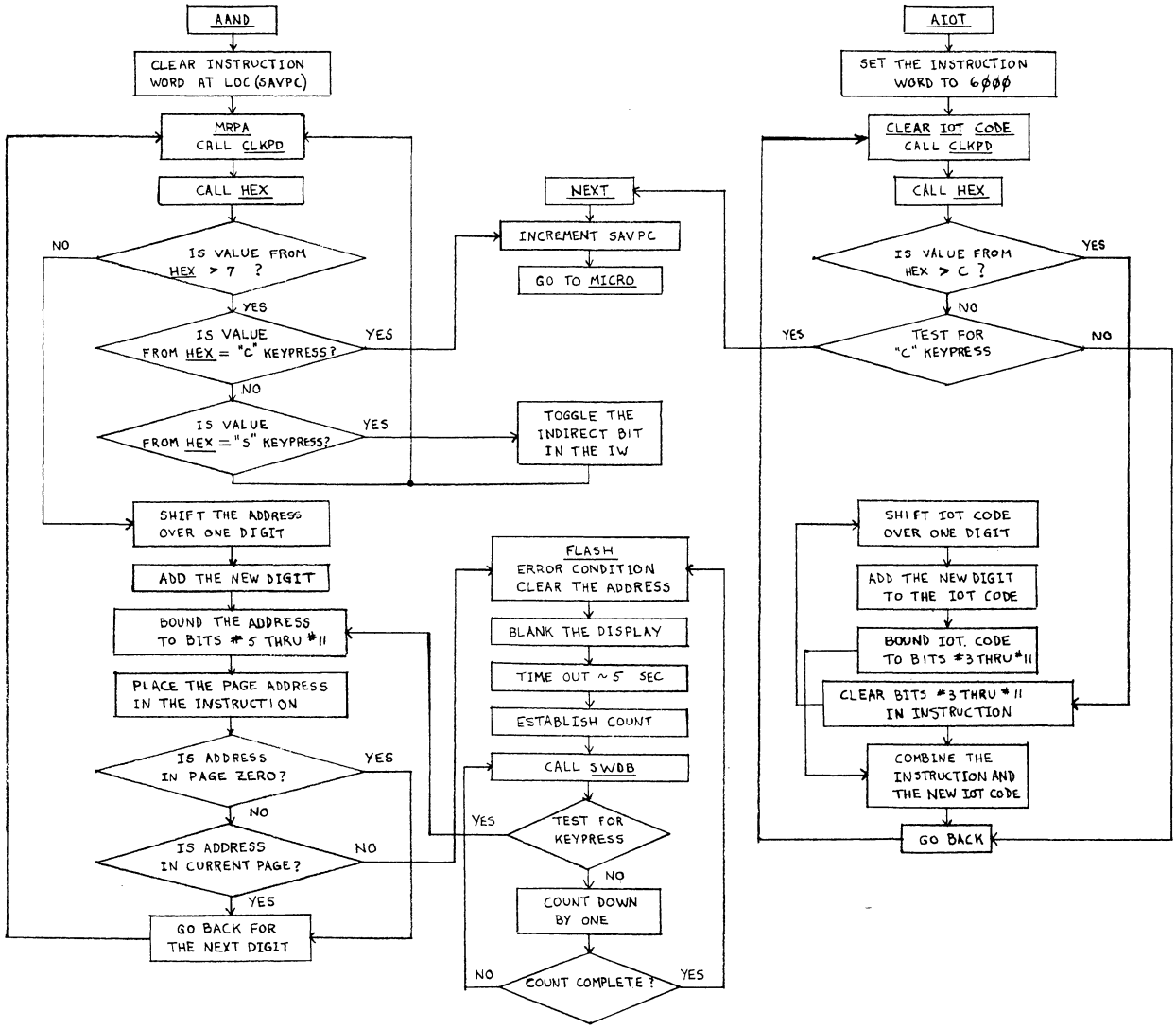

IAC
INC SAVAC
GO TO EXIT

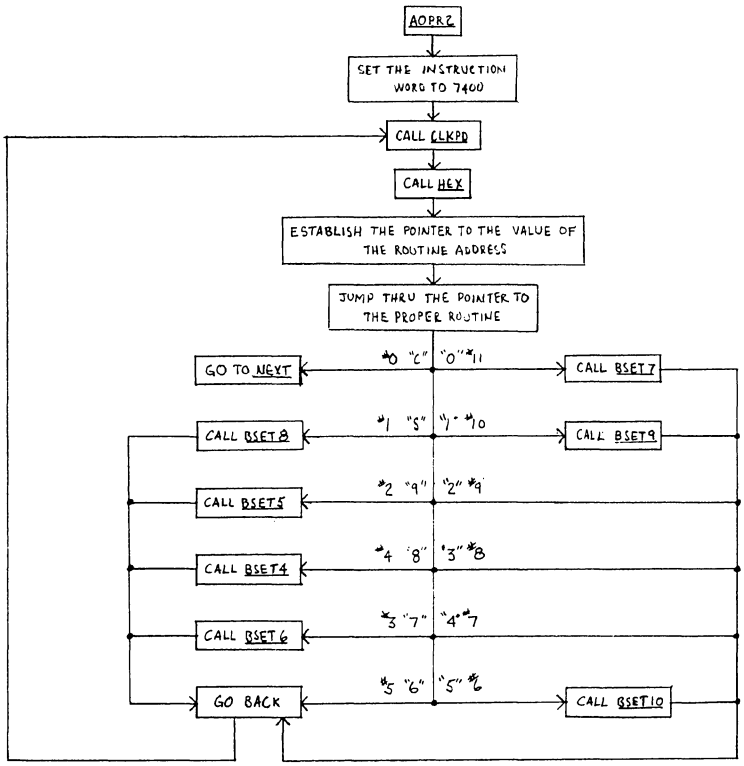

BLK
TOGGLE BLANK BIT IN STATUS
GO TO EXIT
* LINK AND AC DESTROYED

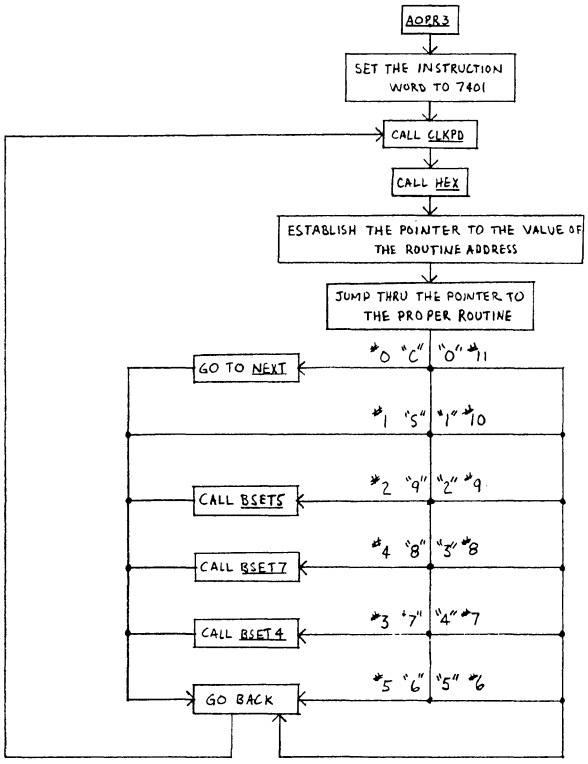

DEC PC
DEC SAVPC
GO TO UDCS

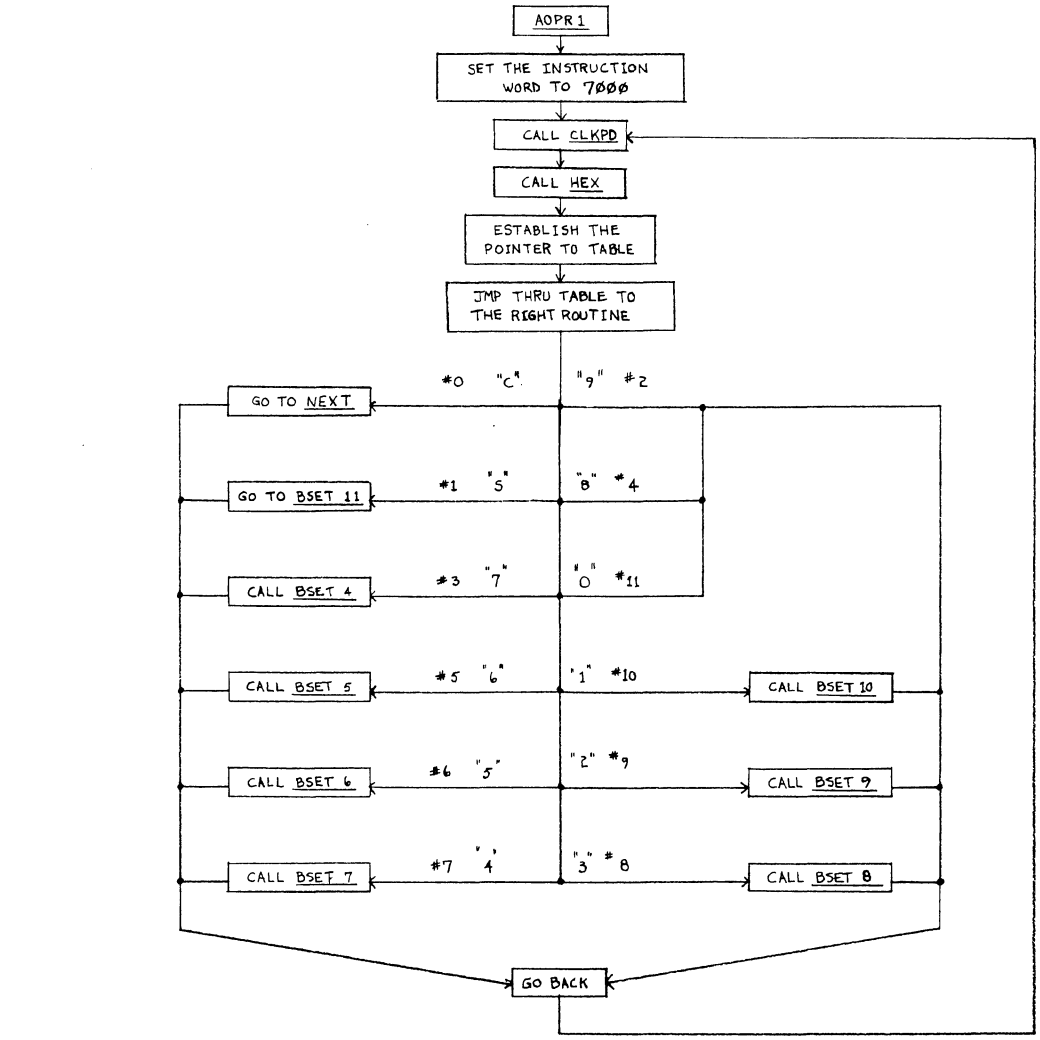

MICRO
CALL CLKPD
CALL HEX
ADJUST VALUE TO USE AS A POINTER
THEN JUMP THRU POINTER
BIT # 0
KEY "C"
GO TO UDCS
1
KEY "5"
INCREMENT SAVPC
2
KEY "7"
GO TO AOPR 3
4
KEY "8"
GO TO AOPR 1
3
KEY "7"
GO TO AOPR 2
5
KEY "6"
GO TO AIOT
6
KEY "5"
GO TO AJMP
7
KEY "4"
GO TO AJMS
8
KEY "3"
GO TO ADCA
9
KEY "2"
GO TO AISZ
10
KEY "1"
GO TO ATAD
KEY "0"
11
GO TO AAND

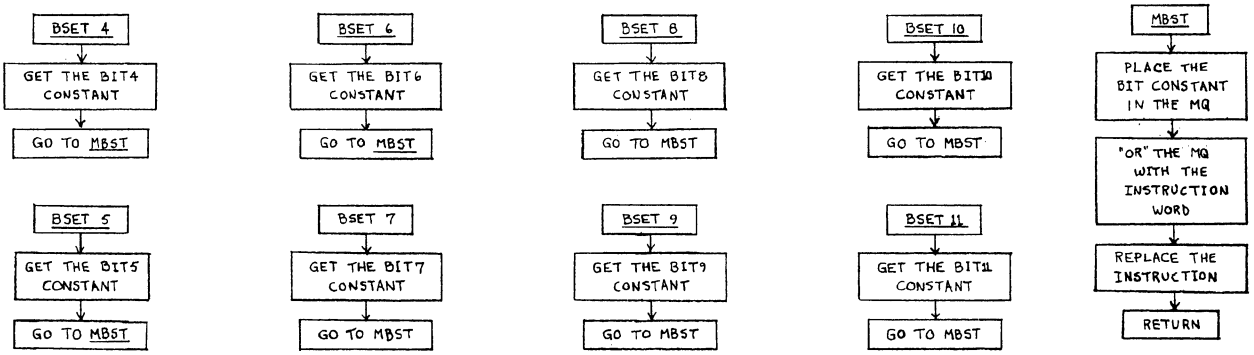

AJMP
CLEAR THE INSTRUCTION WORD AT LOCATION (SAVPC)
LOAD "5" INTO THE OP CODE OF THE INSTRUCTION
GO TO MRPA
AJMS
CLEAR THE INSTRUCTION WORD AT LOCATION (SAVPC)
LOAD "4" INTO THE OP CODE OF THE INSTRUCTION
GO TO MRPA
ADCA
CLEAR THE INSTRUCTION WORD AT LOCATION (SAVPC)
LOAD "3" INTO THE OP CODE OF THE INSTRUCTION
GO TO MRPA
AISZ
CLEAR THE INSTRUCTION WORD AT LOCATION (SAVPC)
LOAD "2" INTO THE OP CODE OF THE INSTRUCTION
GO TO MRPA
ATAD
CLEAR THE INSTRUCTION WORD AT LOCATION (SAVPC)
LOAD "1" INTO THE OP CODE OF THE INSTRUCTION
GO TO MRPA

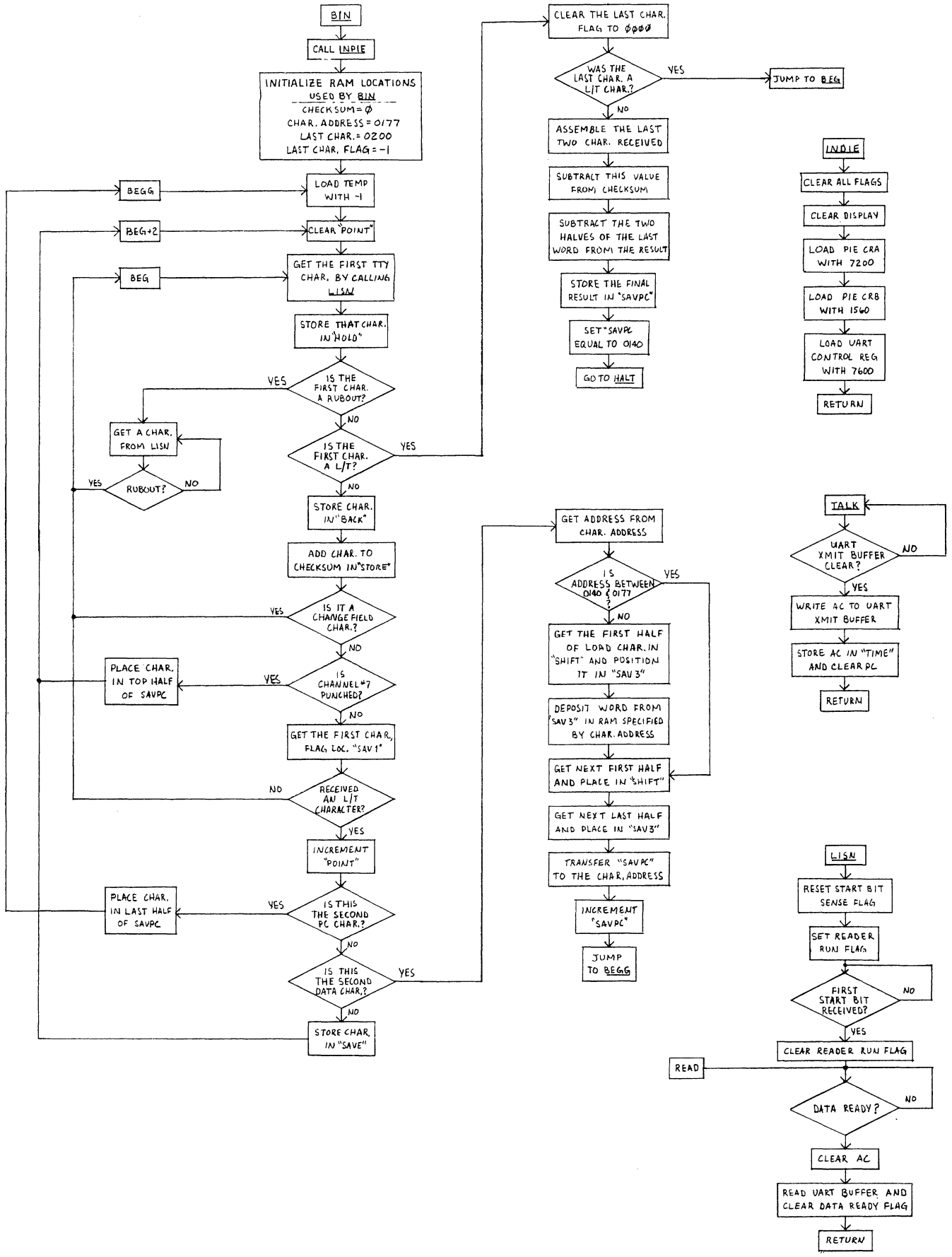

AAND
CLEAR INSTRUCTION WORD AT LOC (SAVPC)
MRPA
CALL CLKPD
CALL HEX
IS VALUE FROM HEX > 7 ?
NO
YES
IS VALUE FROM HEX = "C" KEYPRESS?
YES
NO
IS VALUE FROM HEX = "S" KEYPRESS?
YES
TOGGLE THE INDIRECT BIT IN THE IW
SHIFT THE ADDRESS OVER ONE DIGIT
ADD THE NEW DIGIT
BOUND THE ADDRESS TO BITS #5 THRU #11
PLACE THE PAGE ADDRESS IN THE INSTRUCTION
IS ADDRESS IN PAGE ZERO?
YES
NO
IS ADDRESS IN CURRENT PAGE?
NO
YES
GO BACK FOR THE NEXT DIGIT
NEXT
INCREMENT SAVPC
GO TO MICRO
FLASH ERROR CONDITION CLEAR THE ADDRESS
BLANK THE DISPLAY
TIME OUT ~5 SEC
ESTABLISH COUNT
CALL SWDB
YES
TEST FOR KEYPRESS
NO
COUNT DOWN BY ONE
NO
COUNT COMPLETE ?
YES
AIOT
SET THE INSTRUCTION WORD TO 6000
CLEAR IOT CODE
CALL CLKPD
CALL HEX
IS VALUE FROM HEX > C ?
YES
NO
YES
TEST FOR "C" KEYPRESS
NO
SHIFT IOT CODE OVER ONE DIGIT
ADD THE NEW DIGIT TO THE IOT CODE
BOUND IOT. CODE TO BITS #3 THRU #11
CLEAR BITS #3 THRU #11 IN INSTRUCTION
COMBINE THE INSTRUCTION AND THE NEW IOT CODE
GO BACK

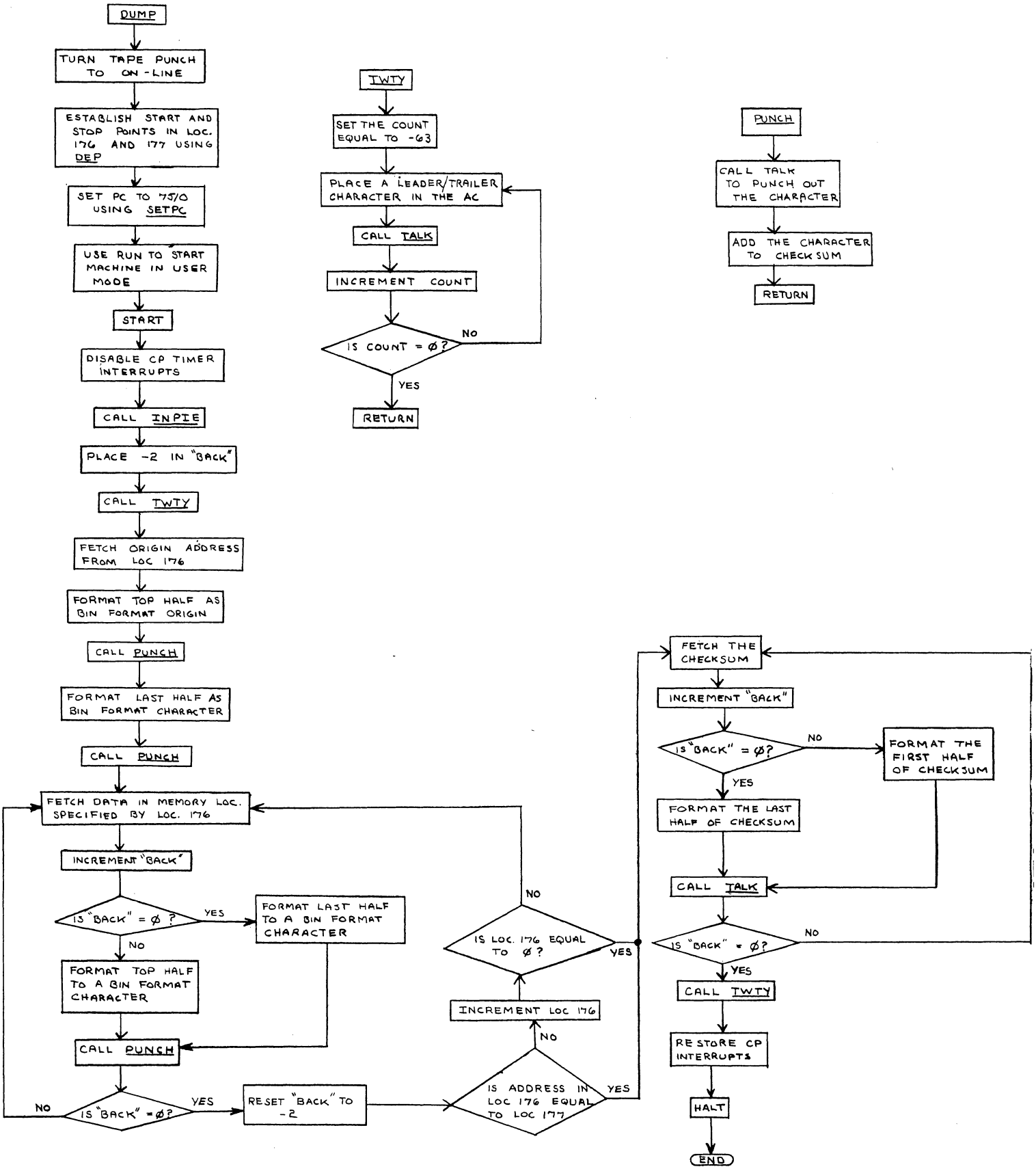

AOPR2
SET THE INSTRUCTION WORD TO 7400
CALL CLKPD
CALL HEX
ESTABLISH THE POINTER TO THE VALUE OF THE ROUTINE ADDRESS
JUMP THRU THE POINTER TO THE PROPER ROUTINE
#0 "C" "0" #11
GO TO NEXT
CALL BSET7
#1 "5" "1" #10
CALL BSET8
CALL BSET9
#2 "9" "2" #9
CALL BSET5
#4 "8" "3" #8
CALL BSET4
#3 "7" "4" #7
CALL BSET6
#5 "6" "5" #6
GO BACK
CALL BSET10


AOPR3
SET THE INSTRUCTION WORD TO 7401
CALL CLKPD
CALL HEX
ESTABLISH THE POINTER TO THE VALUE OF THE ROUTINE ADDRESS
JUMP THRU THE POINTER TO THE PROPER ROUTINE
#0 "C" "0" #11
GO TO NEXT
#1 "5" "1" #10
#2 "9" "2" #9
CALL BSET5
#4 "8" "3" #8
CALL BSET7
#3 "7" "4" #7
CALL BSET4
#5 "6" "5" #6
GO BACK


AOPR 1
SET THE INSTRUCTION WORD TO 7000
CALL CLKPD
CALL HEX
ESTABLISH THE POINTER TO TABLE
JMP THRU TABLE TO THE RIGHT ROUTINE
#0 "C"
"9" #2
GO TO NEXT
GO TO BSET 11
#1 "5"
"8" #4
CALL BSET 4
#3 "7"
" " 0 #11
CALL BSET 5
#5 "6"
"1" #10
CALL BSET 10
CALL BSET 6
#6 "5"
"2" #9
CALL BSET 9
CALL BSET 7
#7 "4"
"3" #8
CALL BSET 8
GO BACK


BSET 4
GET THE BIT4 CONSTANT
GO TO MBST
BSET 6
GET THE BIT6 CONSTANT
GO TO MBST
BSET 8
GET THE BIT8 CONSTANT
GO TO MBST
BSET 10
GET THE BIT10 CONSTANT
GO TO MBST
MBST
PLACE THE BIT CONSTANT IN THE MQ
"OR" THE MQ WITH THE INSTRUCTION WORD
REPLACE THE INSTRUCTION
RETURN
BSET 5
GET THE BIT5 CONSTANT
GO TO MBST
BSET 7
GET THE BIT7 CONSTANT
GO TO MBST
BSET 9
GET THE BIT9 CONSTANT
GO TO MBST
BSET 11
GET THE BIT11 CONSTANT
GO TO MBST


BIN
CALL INPIE
INITIALIZE RAM LOCATIONS USED BY BIN
CHECKSUM = Ø
CHAR. ADDRESS = 0177
LAST CHAR. = 0200
LAST CHAR. FLAG = -1
BEGG
LOAD TEMP WITH -1
BEG+2
CLEAR "POINT"
BEG
GET THE FIRST TTY CHAR. BY CALLING LISN
STORE THAT CHAR. IN "HOLD"
IS THE FIRST CHAR. A RUBOUT?
YES
NO
GET A CHAR. FROM LISN
RUBOUT?
YES
NO
IS THE FIRST CHAR. A L/T?
YES
NO
STORE CHAR. IN "BACK"
ADD CHAR. TO CHECKSUM IN "STORE"
IS IT A CHANGE FIELD CHAR.?
YES
NO
IS CHANNEL #7 PUNCHED?
YES
NO
PLACE CHAR. IN TOP HALF OF SAVPC
GET THE FIRST CHAR., FLAG, LOC. "SAV 1"
RECEIVED AN L/T CHARACTER?
NO
YES
INCREMENT "POINT"
IS THIS THE SECOND PC CHAR.?
YES
NO
PLACE CHAR. IN LAST HALF OF SAVPC
IS THIS THE SECOND DATA CHAR.?
YES
NO
STORE CHAR. IN "SAVE"
CLEAR THE LAST CHAR. FLAG TO ØØØØ
WAS THE LAST CHAR. A L/T CHAR.?
YES
NO
JUMP TO BEG
ASSEMBLE THE LAST TWO CHAR. RECEIVED
SUBTRACT THIS VALUE FROM CHECKSUM
SUBTRACT THE TWO HALVES OF THE LAST WORD FROM THE RESULT
STORE THE FINAL RESULT IN "SAVPC"
SET "SAVPC" EQUAL TO 0140
GO TO HALT
GET ADDRESS FROM CHAR. ADDRESS
IS ADDRESS BETWEEN 0140 & 0177?
YES
NO
GET THE FIRST HALF OF LOAD CHAR. IN "SHIFT" AND POSITION IT IN "SAV 3"
DEPOSIT WORD FROM "SAV 3" IN RAM SPECIFIED BY CHAR. ADDRESS
GET NEXT FIRST HALF AND PLACE IN "SHIFT"
GET NEXT LAST HALF AND PLACE IN "SAV 3"
TRANSFER "SAVPC" TO THE CHAR. ADDRESS
INCREMENT "SAVPC"
JUMP TO BEGG
INDIE
CLEAR ALL FLAGS
CLEAR DISPLAY
LOAD PIE CRA WITH 7200
LOAD PIE CRB WITH 1540
LOAD UART CONTROL REG WITH 7600
RETURN
TALK
UART XMIT BUFFER CLEAR?
NO
YES
WRITE AC TO UART XMIT BUFFER
STORE AC IN "TIME" AND CLEAR PC
RETURN
LISN
RESET START BIT SENSE FLAG
SET READER RUN FLAG
FIRST START BIT RECEIVED?
NO
YES
CLEAR READER RUN FLAG
READ
DATA READY?
NO
CLEAR AC
READ UART BUFFER AND CLEAR DATA READY FLAG
RETURN


DUMP
TURN TAPE PUNCH TO ON-LINE
ESTABLISH START AND STOP POINTS IN LOC. 176 AND 177 USING DEP
SET PC TO 7510 USING SETPC
USE RUN TO START MACHINE IN USER MODE
START
DISABLE CP TIMER INTERRUPTS
CALL INPIE
PLACE -2 IN "BACK"
CALL TWTY
FETCH ORIGIN ADDRESS FROM LOC 176
FORMAT TOP HALF AS BIN FORMAT ORIGIN
CALL PUNCH
FORMAT LAST HALF AS BIN FORMAT CHARACTER
CALL PUNCH
FETCH DATA IN MEMORY LOC. SPECIFIED BY LOC. 176
INCREMENT "BACK"
IS "BACK" = Ø ?
YES
FORMAT LAST HALF TO A BIN FORMAT CHARACTER
NO
FORMAT TOP HALF TO A BIN FORMAT CHARACTER
CALL PUNCH
IS "BACK" = Ø?
NO
YES
RESET "BACK" TO -2
IS ADDRESS IN LOC 176 EQUAL TO LOC 177
NO
YES
INCREMENT LOC 176
IS LOC. 176 EQUAL TO Ø ?
NO
YES
FETCH THE CHECKSUM
INCREMENT "BACK"
IS "BACK" = Ø?
NO
YES
FORMAT THE FIRST HALF OF CHECKSUM
FORMAT THE LAST HALF OF CHECKSUM
CALL TALK
IS "BACK" = Ø?
NO
YES
CALL TWTY
RESTORE CP INTERRUPTS
HALT
END
TWTY
SET THE COUNT EQUAL TO -63
PLACE A LEADER/TRAILER CHARACTER IN THE AC
CALL TALK
INCREMENT COUNT
IS COUNT = Ø?
NO
YES
RETURN
PUNCH
CALL TALK TO PUNCH OUT THE CHARACTER
ADD THE CHARACTER TO CHECKSUM
RETURN

Referring to Figure 8-2, the INTERCEPT JR. MAIN FLOW CHART, the MONITOR is entered on power-up or on every CPREQ through location 7777 of control panel memory and the return address is saved in location 0000. Since in INTERCEPT JR. control panel routines and user programs share the same memory, bit 0 of the status word, Figure 8-3 is used to keep track of who is executing currently, the MONITOR or the user. For example, the programmer may disable the CP TIMER and yet use MONITOR SUBROUTINES in the "user" mode.

Thus the MONITOR updates the register save locations, CP/user mode bit 0 in the status word, and goes on to the initialization routines. The CP subroutine stack is established. (Refer to Applications Bulletin M008 for a description of software stack operation with the IM6100.) Returns from subroutine calls should normally leave AC, MQ and L unchanged.

The Display Refresh subroutine, REFSH, is executed 100-200 times a second in order to keep the display flicker-free.

Next, the keypad is tested for depression of the CNTRL key. If this is not detected, the monitor goes to the exit point, restores registers and flags and returns via the pointer in location 0000 or in the old MQ according to the memory mode status bit. The return via MQ feature is provided as a convenience in writing user mode programs that do not use MQ for any other purpose.

If a CNTRL key depression is detected, the switch debounce routine, SWDB, is called, and the test for CNTRL is made again. In case the test fails, the routine waits for the keypad to become inactive, by calling CLKPD, and exits as before. If the CNTRL key is definitely detected, the MONITOR enters the undefined control state and subsequent key depressions will have to be detected and analyzed. The MONITOR waits for the keypad to clear, by calling CLKPD, and calls HEX, a routine which generates starting addresses for the subroutines that are used to service each of the different key depressions that define a control state. Figure 8-4 shows the connections between the keys and the DX bus, and the control state selected by the key.

The MONITOR is directed to the proper service routine, and may or may not need further data (more key depressions, external conditions, status word bit settings, etc.) to properly execute the routine.

We shall now study some frequently called subroutines in the MONITOR ROM, REFSH, SWDB, CLKPD, HEX and EXIT.

0	1	2	3	4	5
MEM FLAG					

6	7	8	9	10	11
BLANK FLAG			BIT BUCKET	DISPLAY CODE #1	DISPLAY CODE #2

0 - CP/MAIN MEMORY MODE BIT

6 - DISPLAY BLANK = 1

9 - CATCHES "OVERFLOW" FROM BIT 10. IS PERIODICALLY TESTED AND CLEARED.

10 & 11 - SELECTS ONE OUT OF FOUR LED DIGITS IN DISPLAY

FIGURE 8-3 STATUS WORD

DX LINE	0	1	2	3	4	5
KEYBOARD						
CONTROL STATE	CNTRL	SHIFT	MEMory data deposit	SETPC	DECPC	RESET
VALUE RETURNED BY HEX	0013_8 or B_{16}	0012_8 or A_{16}	0011_8 or 9_{16}	0007_8 or 7_{16}	0010_8 or 8_{16}	0006_8 or 6_{16}

DX LINE	6	7	8	9	10	11
KEYBOARD						
CONTROL STATE	HALT	RUN	SINgle instruction execute	DISplay blank/ restore	binary loader	MICRO interpreter
VALUE RETURNED BY HEX	0005_8 or 5_{16}	0004_8 or 4_{16}	0003_8 or 3_{16}	0002_8 or 2_{16}	0001_8 or 1_{16}	0000_8 or 0_{16}

FIGURE 8-4

REFSH - ROM locations 6236-6373, listing line numbers 237-346, flow chart Figure 8-1

This routine first saves the AC and LINK as it uses them and then looks at the display blank flag, bit # 6 of the status word, Figure 8-3. It does this by doing a byte swap, bringing bit # 6 into bit 0 and testing for a negative sign. If the blank flag is set, all zeros are loaded to the display, resulting in disabling all the multiplexing transistors and thus blanking the display. The routine would return in this case to the calling program after restoring the AC and L.

If the flag was clear, the display must be refreshed. Bits 10 and 11 of the status word encode the digit to be driven. Bit 9, the "bit bucket", is cleared every time a refresh is performed in order to prevent other bits in the status word from being affected when the status word is incremented to select the next digit to be refreshed. Constants stored in ROM are used as "AND masks" to clear bit 9 and select the digit code. The digit code is added to a base address to generate a pointer to one of four routines, DIG0, DIG1, DIG2, DIG3, that set up constants and loop counters with which to enter the LOAD routine. LOAD uses a mask to select the particular digit to be displayed from the user PC and data at the user PC. Then LOAD uses the loop counter constants to rotate these digits into the proper position and adds the multiplexer select bit (stored in TEMP). Figure 8-5 shows the format of this IOT word.

0	1	2	3	4	5	6	7	8	9	10	11
D4	D3	D2	D1	BCD Address				BCD Memory data			

FIGURE 8-5

SWDB - ROM locations 6200-6235, listing line numbers 190-230, flow chart, Figure 8-1

This routine reads the keypad into the accumulator, waits for 25 milliseconds, and again reads the keypad to see if it matches the first reading, thus indicating the end of switch bounce. If the readings do not match another 25 milliseconds timeout is allowed. During the timeout, the display is refreshed approximately every five milliseconds.

CLKPD - ROM locations 6110-6116, listing line numbers 350-361, flow chart Figure 8-1

This routine calls SWDB in order to timeout bounces, and checks for a zero reading from the keypad (indicating keypad clear) as long as required then returns to the calling program.

HEX - ROM locations 6425-6523, listing line numbers 434-514, flow chart Figure 8-1

This routine determines whick key was pressed and generates a different number for each key. These numbers are used by the UNDEFINED CONTROL STATE routine to generate starting addresses to the control state routines for each key.

EXIT - ROM locations 6061-6076, listing line numbers 152-170, flow chart Figure 8-1

This routine is entered when no keypad activity can be detected. The routine waits for the keypad to clear by executing CLKPD, then restores all registers and flags from RAM save locations. The memory mode bit in the status word is checked to make sure that the routine was entered by the control panel MONITOR.

There is another entry point to this routine called OUT which is used if no keypad activity was detected even before key debouncing is needed, indicating the keypad was already clear. By entering at OUT, CLKPD does not have to be called, saving at least the 25 milliseconds it takes to execute SWDB.

If EXIT was entered in the main mode, the routine clears the memory mode bit, restores flags and registers and exits indirect via the contents of the SAVMQ location. This feature is provided to enable the user to store his return address in the MQ and not have to alter other registers. This is useful when writing programs that use subroutines in the MONITOR. The MONITOR itself rarely uses MQ for anything (for example, routine MBST at 7273_8).

CONTROL STATE SERVICE ROUTINES

Five of the control states possible through key depressions require extremely simple service routines. These five along with the symbolic starting address are:

INCREMENT AC	INCAC
DECREMENT PC	DECPC
HALT	HALT
RUN	RUN
RESET	RESET

These routines are stored in ROM locations 6400-6424, listed on lines 393-430 and the flow charts are in Figure 8-2.

These routines are each a few instructions long and self-explanatory. They modify the RAM save locations.

The control panel program when executing the EXIT routine restores all flags and registers in the IM6100 from these RAM save locations.

The RUN routine uses the IOT RUN, 6407, command described in Chapter 4.

The RESET routine clears all save locations, executes the IOT RESET command, 6406, sets the PC to 7777 and goes to the HALT routine.

Except for DECPC, the above routines, when complete, branch to the EXIT routine described previously by jumping indirect via the location labeled UG. DECPC, upon completion, jumps indirect via BUG which is the starting address of UDCS, returning INTERCEPT JR. to the undefined control state. This enables the user to pick the next control state without again pressing the CNTRL key.

DEPOSIT INTO MEMORY, MEM, ROM locations 6524-6556, listing line numbers 516-550, flow chart Figure 8-2

This routine with starting address at DEP may be executed repeatedly when a sequence of numbers is entered from the keypad. It begins by calling CLKPD, then HEX. The value passed on by HEX is tested for being greater than 7. If it is not greater than 7, it is interpreted to be an octal digit to be deposited into memory by shifting it into the rightmost digit. This is done by getting the current memory data indirect via 0000_8, SAVPC, shifting left three bits, while clearing the link each time so that zeros are shifted into the LSB, then adding the new digit. The updated data word is restored via the pointer in SAVPC and the routine jumps back to the beginning for the next digit from HEX.

If the digit is greater than 7, it is not to be entered into memory, but rather a pointer is computed to force a branch to the proper routine to be executed next. This is done by adding the contents of TAB, 6534_8, to the value returned by HEX, 10, 11, 12, 13, resulting in 6544_8, 6545_8, 6546_8,

6547_8. These locations contain pointers to routines DCI, PCI, EXIT and UDCS, respectively.

In other words, pressing DECPC at this time results in routine DCI being executed, pressing MEM results in routine PCI being executed, pressing the yellow key results in the EXIT routine being executed and pressing the CNTRL key results in UDCS being executed, meaning a return to undefined control state.

Routine DCI decrements the PC by adding -1, 7777_8, to it, and returns to DEP to get the next digit, indicating the contents of the decremented memory location may now be altered.

Routine PCI increments the PC when key MEM is pressed and returns to DEP so that data may be entered into the incremented memory location.

These routines allow the user to step forwards and backwards through memory and alter data at will, as long as the memory area being addressed is not in ROM. ROM may be examined but not altered.

BLANK FLAG TOGGLE, DIS, ROM locations 6566-6575, listing line numbers 564-579, flow chart Figure 8-2

This routine is executed when the key marked DIS RAL ISZ is pressed when in the undefined control state. Bit #6 in the status word, Figure 8-3, is called the blank flag, and this routine toggles it every time it is executed, therefore, allowing the user to shut off the display to conserve power and to turn it back on. The routine clears the AC and L, gets the status word, shifts bit #6 into the link (by doing a byte swap and left shift), complements the link, shifts it back, swaps bytes again, restores status and goes to EXIT.

SET PROGRAM COUNTER, SETPC, ROM locations 6600-6632, listing line numbers 578-612, flow chart Figure 8-2

This routine, like DEP, accepts octal digits from the keypad. It begins by calling CLKPD, and then HEX to get a valid number from a key depression. The value is checked for being over 7. If not, the routine goes on to GOON, which loads the digit into the rightmost octal position in the PC and jumps back to SETPC to pick up a new key depression.

If the value returned by HEX is greater than 7, a base address in location ADJT is added to it, and the sum is used as an indirect pointer back to SETPC (if the DECPC or MEM keys are pressed) to EXIT (if yellow key is pressed) or to UDCS (if CNTRL is pressed).

MICROINTERPRETER, MICRO, ROM locations 6633-7300, listing line numbers 613-1045, flow chart Figure 8-2

Routine MICRO calls HEX and gets an index to compute a pointer to the routines servicing the individual keys (see Example 5 in Chapter 3 for a detailed description).

Pressing the SHIFT key causes AINC to be executed, incrementing SAVPC. Pressing any of the keys with memory reference instruction opcodes on them causes routines ATAD, AISZ, ADCA, AJMS or AJMP to be executed. These routines load the opcode into the AC and jump to AAND. (Note that the opcodes are sometimes stored as constants, and sometimes are instructions located elsewhere in the same page.) This results in the AC being placed into the location being addressed by the user. The MONITOR, therefore, displays the address and opcode selected by the user.

Routine MRPA continues to scan digits entered from the keypad and checks to see if they are address digits, 0-7, a CNTRL key depression (routine NEXT is executed in which the user PC is incremented, and control returns to MICRO to interpret the next instruction) or a SHIFT key depression (in which case routine ZONK is entered in order to set indirect bit 3). This is done by rotating the bit into the link, setting it and rotating back. Control is passed back to MRPA so it makes no difference if the indirect bit is set before or after the address bits are supplied.

Address digits are shifted into the address field from right to left and the resulting address is checked for validity (in page 0 or in current page). If the address is outside valid page boundaries, then the program branches to routine FLASH. If the address is valid, MRPA is re-entered to get the next digit. Routine FLASH flashes the display to indicate an invalid address field.

The routine blanks the display using IOT instruction 6400 and times out approximately (4096 X (16 + 10) X 10) or 1064960 states. This takes over half a second at 3.33 or 4 MHz.

The routine then checks to see if the keypad has been depressed. If it has, the address field is loaded with the new digits. If it has not, the routine continues to time out a different constant, TKB.

Routine AIOT is entered if in the MICRO mode, key IOT is pressed. An opcode of 6 is entered into the AC with a microprogrammed combination of Group I microinstructions and the routine collects digits from the keypad, while checking for a CNTRL key entry.

Detection of a CNTRL causes a branch to NEXT which increments SAVPC and returns to MICRO as before. Octal digits are shifted into the device address and control fields of the IOT instruction from right to left.

Routine AOPR1 is entered when an operate group 1 instruction is to be loaded via the keypad. The routine starts by loading 7000 into the user addressed location, then calling CLKPD and HEX as further digits are expected.

A table of jump addresses is used as described in Example 5, Chapter 3 to branch to the proper routine.

The branches either cause the program to ignore the key and look for the next key depression, AOPR1 + 2, or to call an appropriate bit set subroutine, JA10-JA4. The bit set routines are used by routines in all three operate groups so they are coded as subroutines that may be nested in the MONITOR stack.

The bit set routines work by reading a constant, AAA-AAG, corresponding to the appropriate bit being set into the AC, then jumping to the MBST routine. This routine stores the constant temporarily in MQ, clears the AC, gets the instruction in its current state, updates it by OR'ing in the MQ, replaces it at the user addressed location and returns.

This procedure is followed by all the operate group micro-instruction service routines.

In other words, a table of jump addresses is used to compute a branch to either a bit set routine or back to the keypad reading sequence.

SINGLE INSTRUCTION EXECUTE, SIN, ROM locations 7301-7444, listing line numbers 1047-1170, flow chart Figure 8-2

This routine is useful in program development as a single instruction at a time may be executed allowing intermediate results to be examined under MONITOR control. This routine may only be used to single step through programs in RAM and not in ROM because software "breakpoints" are implemented by replacing the instruction at a breakpoint with a jump to the breakpoint processing subroutine and this requires writing into the memory.

SIN first initializes page 0 locations 0152 and 0153 labeled STORE and SHIFT to contain the instruction JMP I SHIFT and the address 7427. Then it checks the instruction to see if it is a JMP or JMS. It does this by extracting the opcode bits with an AND mask, adding -4000 or -5000 to them and checking for AC = 0.

It also checks the address mode bits with routines INAD and INDB and computes the effective address for the next memory reference. This address is in location TIME. In case of a JMS, location TIME is incremented (to point to the next instruction to be fetched which follows the location where the return address is to be stored).

Routines INAD and INDB determine whether the current page bit and indirect bit are set by masking of all other bits and testing for a non-zero AC. If the page bit is set, the current page number is obtained by masking off other bits. This page number is concatenated with the page address. If the indirect bit is set, the effective address is fetched and replaced in TIME. In any event, when location EXEC + 4 is reached, TIME contains the address of the next instruction to be fetched. Now the program gets the contents of this location, NEXT, and the next sequential one (NEXT + 1) and saves them in SAVE and SAV1. The contents of these two locations are replaced by the instruction JMS BACK, which is 4151, a JMS to page 0 location 0151 labeled BACK. Then both these locations are tested to see if the instruction was actually placed there, that is, if RAM exists there. The program does this by reading the locations back, adding the two's complement of 4151_8 to them and checking for a zero AC.

If the locations were indeed loaded correctly, the program proceeds to restore the MQ, LINK and AC and performs an indirect jump via SAVPC, executing the instruction specified by the user.

This instruction is executed, and, when the user program fetches the next instruction, it turns out to be the JMS BACK breakpoint placed by the MONITOR, so the user program stores the return address in BACK, 0151, and executes the instruction in location 0152 which happens to be the JMP I SHIFT which was placed there earlier. Thus, control is returned to the SIN routine at the point 7427 labeled RET. The routine saves away the AC, L and MQ again, restores the two instructions at the breakpoints, updates the user PC using the address stored in BACK and returns to the undefined control state.

The reason for storing JMS BACK in two successive locations can now be seen to provide for the case when the single instruction to be executed may skip the next location.

A limitation of this program is that JMP.-1, JMS.-1 and JMS.-2 instructions cannot be single stepped. There is not much application for a JMS.-1 or a JMS.-2, so the real limitation is with the instruction JMP.-1.

What happens is that one breakpoint will be placed in the location of the actual instruction that is to be performed. Thus, the effective address referred to as NEXT will be the location containing JMP.-1 and the location NEXT + 1 will actually be the location containing JMP.-1. As these two locations are replaced by the JMS BACK, the program in attempting to perform the JMP.-1 will immediately see the breakpoint. Control will return without any action having been taken and the state of the machine when restored will be identical to what it was before. The effect is that of not performing the instruction.

To get around this limitation, when writing skip and test loops, always provide an additional NOP so that the JMP will not be a ".-1".

Example:

Address	Instruction
A	NOP
A + 1	SKIP on condition
A + 2	JMP.-2

This limitation affects only the single instruction function and does not apply to running in normal mode.

This limitation applies to the TAD, ISZ and DCA memory reference instructions when they try to reference a *+2 or *+1 location. (There is not much application for a program that uses instructions referencing the next sequential location, and especially, alters it).

The instruction TAD *+2 will add the breakpoint instruction 4151 to the contents of the AC.

The instruction ISZ*+2 will increment the value 4151 to 4152 and then the original datum is restored so there is no net effect when single stepping this instruction.

The instruction DCA*+2 is useful in the INTERCEPT JR. to display a result when the location following this instruction contains the HALT instruction 7402. However, when single stepping this instruction, the DCA will write over the breakpoint instruction, then the original content is restored, so there is no net effect. It is recommended that the sequence

```
DCA*+3

NOOP

HALT
```

is used to display data in programs when single stepping is desired. Alternatively, the user, after single stepping through the previous part of the program, can depress CNTRL RUN for the display sequence.

PIE INITIALIZE, INPIE, PRINT TO TTY, TALK, RECEIVE FROM TTY KEYBOARD OR READER, LISN

These routines in ROM locations 7445-7507, listed in lines 1170-1248 are described in Chapter 7 on the PIEART board. See Figure 8-2 for the flow chart.

INTERCEPT JR. BINARY LOADER, BIN, ROM locations 7600-7755, listing line numbers 1249-1385, flow chart Figure 8-2

This loader uses the PIEART interface board. The BIN format is described in Applications Bulletin M003, and the Teletype modifications are described in Applications Bulletin M005. The routine initializes the PIE-UART checksum and RAM locations it uses, then gets a character by calling LISN. The character is checked for being a rubout (all channels punched) or part of leader-trailer (only channel 8 punched), and if it is either, the program branches to RUM or LTC respectively. RUM continues to scan characters until another rubout is detected at which point it returns to BEG, the beginning of the character processing program. Thus the system ignores text enclosed by rubouts.

LTC checks if the character is a first, LT character or not. If so, the load routine is ended, the checksum computed, the SAVAC location placed in the address display and the machine is halted showing the checksum.

If the character received was neither a rubout nor an LT character, the program updates the checksum, checks for a "change field" character (if it is, it is ignored and the next character is processed), checks for "origin" data (if so, it gets the address data in two successive characters) and checks to make sure the starting address does not fall in the range 0140-0177 which is used by the MONITOR. If the address falls in this range, the RAM is <u>not</u> loaded. Data is loaded by routine DL2 only when conditions are valid.

INTERCEPT JR. MEMORY DUMP, DUMP, ROM locations 7510-7576, listing line numbers 1410-1511, flow chart Figure 8-2

This program requires that the first and last locations, of a block of memory to be dumped on tape, should be entered in locations 0176 and 0177, and the program run starting at location 7510.

The program uses the leader-trailer routine contained in locations 7757-7765 and 6173-6176. It will punch out a BIN formatted tape complete with leader-trailer and checksum.

The program disables the CP request timer, initializes the PIE-UART, calls routine TWTY in the leader-trailer program to punch 63 LT characters. (Note that TWTY uses a constant KM63, which happens to be an instruction located in address 7723 that conveniently lies in the same page and has the numerical value required. This programming device saves valuable memory locations.)

The program next punches out the origin address, user entered in 0176, in two successive ASCII characters along with the channel 7 punch.

The data is also punched out using two characters per 12 bit word. The program counts the 1st and 2nd characters by looking at location BACK which is loaded with 7776 and incremented as a character is output. After two characters, the location becomes zero and the ISZ that incremented it will skip the BSW that is used to position the 2nd half of the character.

After every data item is transmitted, the address is checked to see if the end of the block has been reached.

As each character is punched (by calling the PUNCH routine, which in turn calls TALK, then jumps to LINKER) the checksum is updated in location SAV5 (by routine LINKER).

After the last data item has been punched, the checksum is punched by CHSUM and routine TWTY is again called to punch out the leader-trailer tape.

Finally, the CP request timer is restored and the processor halted.

```
MICROINTER


BEGIN PASS 1


END OF PASS 1
BEGIN PASS 2

    1                                       / THE MONITOR PROGRAM FOR THE
    2                                       / INTERCEPT JR.
    3
    4                                       / THE PAGE ZERO LOCATIONS
    5
    6          0000   *0000
    7
    8  0000   0000   SAVPC,  0
    9
   10
   11          0140   *0140
   12
   13  0140   0000   SAVAC,  0
   14  0141   0000   SAVMQ,  0
   15  0142   0000   SAVFL,  0
   16  0143   0000   STATUS, 0
   17
   18  0144   0000   TIME,   0
   19  0145   0000   SAVE,   0
   20  0146   0000   HOLD,   0
   21  0147   0000   POINT,  0
   22  0150   0000   TEMP,   0
   23
   24  0151   0000   BACK,   0
   25  0152   0000   STORE,  0
   26  0153   0000   SHIFT,  0
   27  0154   0000   SAV1,   0
   28  0155   0000   SAV2,   0
   29  0156   0000   SAV3,   0
   30  0157   0000   SAV4,   0
   31  0160   0000   SAV5,   0
   32
   33
   34          0161   *0161
   35
   36  0161   0000   CALLX,  0               / THE PAGE ZERO LOCATIONS FOR THE
   37                                         / CP MONITOR STACK
   38
   39
   40          0164   *0164
   41
   42  0164   0000   RETX,   0
   43  0165   0000   STACK,  0
   44  0166   0000   AC,     0
   45
   46                                       / THE LOCATIONS 167 TO 177 ARE THE
   47                                       / STACK POINTER TABLE LOCATIONS
   48
   49                                       / THE PROGRAM MACROS
   50
   51          4161   CALL=JMS CALLX
   52          5564   RETURN=JMP I RETX
   53
   54                                       / THE CP ENTRY POINT
   55
   56          7777   *7777
   57
   58  7777   5366           JMP CPMODE      / GO TO THE CPMODE START POINT
   59
   60          7766   *7766
   61
   62  7766   3140   CPMODE, DCA SAVAC       / SAVE THE AC
   63  7767   6004           GTF
   64  7770   3142           DCA SAVFL       / SAVE THE FLAGS
   65  7771   1143           TAD STATUS      / GET THE STATUS WORD
   66  7772   7104           CLL RAL         / POSITION BIT #0
   67  7773   7110           CLL RAR         / CLEAR THE MEM FLAG BIT IN STATUS
   68  7774   3143           DCA STATLS      / RESTORE STATUS
   69  7775   5776           JMP I 7776      / GO TO INIT
   70  7776   6007           INIT            / THE COMMON START POINTER
   71
   72
   73                                       / THE USER START POINT
   74
   75
   76          6000   *6000
   77
   78  6000   3140   START,  DCA SAVAC       / SAVE THE AC
   79  6001   6004           GTF             / GET THE FLAGS
   80  6002   3142           DCA SAVFL       / SAVE THE FLAGS
   81  6003   1143           TAD STATUS      / GET THE STATUS WORD
   82  6004   7104           CLL RAL         / POSITION BIT #0
   83  6005   7130           CLL CML RAR     / SET THE MEM FLAG BIT IN STATLS
   84  6006   3143           DCA STATUS      / RESTORE STATUS
   85
   86  6007   7521   INIT,   SWP            / GET THE MQ
   87  6010   3141           DCA SAVMQ       / SAVE THE MQ
   88
   89
   90
   91                                       / ESTABLISH THE CP SUBROUTINE STACK
   92
   93  6011   1234           TAD JMPI
   94  6012   3162           DCA CALLX+1
   95  6013   1235           TAD KCALLY
   96  6014   3163           DCA CALLX+2
   97  6015   1236           TAD KRETY
   98  6016   3164           DCA RETX
   99  6017   1237           TAD BASE
  100  6020   3165           DCA STACK       / THIS WILL CLEAR THE STACK POINTER
  101
  102                                       / GO ON WITH THE PROGRAM
  103
  104  6021   4161           CALL            / CALL THE FIRST DISPLAY REFRESH
  105  6022   6236           REFSH
  106  6023   7604           LAS             / LOAD THE KEYPAD TO THE AC
  107  6024   7500           SMA             / TEST; IS THE "C" KEY PRESSED
  108  6025   5263           JMP OUT         / NO; GO TO OUT
  109  6026   4161           CALL            / YES; CALL A SWDB TO BE SURE
  110  6027   6200           SWDB
  111  6030   7500           SMA             / TEST; IS THERE A VALID "C" KEYPRESS
  112  6031   5261           JMP EXIT        / NO; GO TO EXIT
  113  6032   5633           JMP I .+1       / YES; GO TO THE UNDEFINED CONTROL
  114  6033   6117           UCCS            / STATE
  115
  116
  117                                       / THIS IS THE SUBROUTINE STACK
  118                                       / ROM OVERHEAD
  119
  120  6034   5563   JMPI,   JMP I CALLX+2
  121  6035   6040   KCALLY, CALLY
  122  6036   6051   KRETY,  RETY
  123  6037   0167   BASE,   STACK+2
  124
  125
  126  6040   3166   CALLY,  DCA AC          / SAVE THE AC
  127  6041   2165           ISZ STACK       / UPDATE THE STACK POINTER
  128  6042   1161           TAD CALLX       / CALLX HAS THE RETURN ADDRESS
  129  6043   7001           IAC             / INCREMENT THE RETURN ADDRESS
  130  6044   3565           DCA I STACK     / SAVE ON THE LIFO STACK
  131  6045   1561           TAD I CALLX     / GET THE USER SUBROUTINE
  132  6046   3161           DCA CALLX       / PUT IT IN CALLX
  133  6047   1166           TAD AC          / RESTORE THE AC
  134  6050   5561           JMP I CALLX     / GO TO THE SUBROUTINE
  135
  136
  137  6051   3166   RETY,   DCA AC          / SAVE THE AC
  138  6052   1565           TAD I STACK     / GET THE RETURN ADDRESS FROM THE STACK
  139  6053   3161           DCA CALLX       / AND PUT IT IN CALLX
  140  6054   7060           CMA CML         / COMPLIMENT THE AC AND LINK
  141  6055   1165           TAD STACK       / DECREMENT THE STACK POINTER AND
  142                                       / RESTORE THE LINK
  143  6056   3165           DCA STACK       / RESTORE THE STACK POINTER
  144  6057   1166           TAD AC          / RESTORE THE AC
  145  6060   5561           JMP I CALLX     / RETURN TO THE PROGRAM
  146
  147
  148
  149                                       / WE NOW CONTINUE WITH THE CP
  150                                       / PROGRAM
  151
  152
  153                                       / THIS IS THE EXIT POINT
  154
  155  6061   4161   EXIT,   CALL            / WAIT FOR THE KEYPAD TO CLEAR
  156  6062   6110           CLKPD
  157  6063   7300   OUT,    CLA CLL         / CLEAR THE AC AND THE LINK
  158  6064   1141           TAD SAVMQ       / GET THE MQ
  159  6065   7421           MQL             / RESTORE THE MQ
  160  6066   1143           TAD STATUS      / GET THE STATUS WORD
  161  6067   7710           SPA CLA         / TEST; IS THE MEM FLAG SET?
  162  6070   5277           JMP MOUT        / YES; GO TO THE MEMORY EXIT ROUTINE
  163  6071   1142           TAD SAVFL       / GET THE FLAGS
  164  6072   7104           CLL RAL         / RESTORE THE LINK
  165  6073   7200           CLA             / CLEAR THE AC
  166  6074   1140           TAD SAVAC       / GET THE AC
  167  6075   6001           ION             / RESTORE THE IENFF AND COME OUT
                                             / OF THE CP MODE
  168
  169  6076   5400           JMP I SAVPC     / RESTORE THE PC
  170
  171
  172  6077   1143   MCLT,   TAD STATUS      / GET THE STATUS WORD
  173  6100   7104           CLL RAL         / POSITION BIT #0
  174  6101   7110           CLL RAR         / CLEAR THE MEM FLAG IN STATUS
  175  6102   3143           DCA STATLS      / RESTORE THE STATUS WORD
  176  6103   1142           TAD SAVFL       / GET THE FLAGS
  177  6104   6005           RTF             / RESTORE THE FLAGS
  178  6105   7200           CLA             / CLEAR THE AC
  179  6106   1140           TAD SAVAC       / RESTORE THE OLD AC
  180  6107   5541           JMP I SAVMQ     / RETURN THRU THE OLD MQ
  181
  182
  183
  184
  185
  186                                       /  THE MONITOR SUBROUTINES
  187
  188
  189
  190
  191                                       / THE SWITCH DEBOUNCE AND
  192                                       / DISPLAY REFRESH ROUTINE
  193
  194
  195          6200   *6200
  196
  197  6200   7210   SWCG,   CLA RAR         / POSITION THE LINK IN A CLEARED AC
  198  6201   3155           DCA SAV2        / SAVE THE LINK IN SAV2
  199  6202   7604   GAR,    LAS             / LOAD THE SWITCH REGISTER (KEYPAD)
                                             / TO THE AC
  200
  201  6203   3145           DCA SAVE        / STORE IT IN SAVE
  202  6204   1233           TAD TK1         / GET THE TIME CONSTANT #1
  203  6205   3144           DCA TIME        / STORE IT IN THE TIMER
  204  6206   2144           ISZ TIME        / TIME OUT 4 MILLISECONDS AT 2.5 MHZ
  205  6207   5206           JMP .-1         / JUMP BACK ONE PLACE
  206  6210   4161           CALL            / CALL THE DISPLAY REFRESH
  207  6211   6236           REFSH
  208  6212   1235           TAD TCNT        / GET THE WAIT COUNT
  209  6213   3157           DCA SAV4        / PLACE THE COUNT IN THE COUNTER
  210  6214   1234           TAD TK2         / GET THE TIME CONSTANT #2
  211  6215   3144           DCA TIME        / PLACE IN THE TIMER
  212  6216   2144           ISZ TIME        / TIME OUT 5 MILLISECONDS AT 2.5 MHZ
  213  6217   5216           JMP .-1         / JMP BACK ONE PLACE
  214  6220   4161           CALL            / REFRESH THE DISPLAY
  215  6221   6236           REFSH
  216  6222   2157           ISZ SAV4        / COUNT DOWN THE COUNTER
  217  6223   5214           JMP .-7         / GO BACK FOR ANOTHER WAIT CYCLE
  218  6224   7604           LAS             / LOAD THE KEYPAD A SECOND TIME
  219  6225   7041           CIA             / NEGATE THE VALUE
  220  6226   1145           TAD SAVE        / ADD THE FIRST READING
  221  6227   7440           SZA             / TEST; ARE THE VALUES THE SAME
  222  6230   5202           JMP GAR         / NO; GO BACK AND DO IT AGAIN
  223  6231   1145           TAD SAVE        / YES; GET THE READING INTO THE
                                             / AC
  224
  225  6232   5564           RETURN          / RETURN TO THE PROGRAM
  226
  227
  228  6233   7500   TK1,    7500            / -192
  229  6234   7420   TK2,    7420            / -240
  230  6235   7774   TCNT,   7774            / -4
  231
  232
  233
  234
  235                                       / THE DISPLAY REFRESH SUBROUTINE
  236
  237  6236   3151   REFSH,  DCA BACK        / SAVE THE AC IN BACK
  238  6237   7010           RAR             / POSITION THE LINK
  239  6240   3146           DCA HOLD        / SAVE THE LINK IN HOLD
  240  6241   1143           TAD STATUS      / GET THE STATUS WORD
  241  6242   7002           BSW             / POSITION THE BLANK FLAG, BIT #6
  242  6243   7700           SMA CLA         / TEST; IS THE BLANK FLAG SET,
                                             / CLEAR THE AC
  243
  244  6244   5252           JMP .+6         / NO; SKIP TO THE DISPLAY REFRESH
  245  6245   6400           6400            / YES; LOAD ZEROS TO THE DISPLAY
  246  6246   1146   THRU,   TAD HOLD        / GET THE LINK
  247  6247   7104           CLL RAL         / RESTORE THE LINK
  248  6250   1151           TAD BACK        / RESTORE THE AC
  249  6251   5564           RETURN          / RETURN TO THE PROGRAM
  250
  251
  252  6252   1143           TAD STATUS      / GET THE STATUS WORD
  253  6253   0372           AND MSK1        / CLEAR THE BIT BUCKET
  254  6254   3143           DCA STATUS      / RESTORE THE STATUS WORD
  255  6255   1143           TAD STATLS      / GET THE STATLS WORD
  256  6256   0373           AND MSK2        / MASK OUT THE DIGIT CODE
  257  6257   1262           TAD ADJ         / ADJUST FOR THE TABLE
  258  6260   3147           DCA POINT
  259  6261   5547           JMP I POINT     / GO TO THE PROPER ROUTINE THRU
  260                                        / THE TABLE
  261  6262   6263   ADJ,    TABLE
  262  6263   5267   TABLE,  JMP DIGO        / GO TO THE PROPER ROUTINE
  263  6264   5301           JMP DIG1
  264  6265   5312           JMP DIG2
  265  6266   5323           JMP DIG3
  266
  267  6267   7332   DIGO,   CLA CLL CML RTR / SET THE AC EQUAL TO 2000
  268  6270   7012           RTR             / SET THE AC EQUAL TO 0400
  269  6271   3150           DCA TEMP        / PLACE IN THE TEMP
  270  6272   1365           TAD KM8         / SET THE AC EQUAL TO -8
  271  6273   3147           DCA POINT       / STORE IN POINT
  272  6274   1366           TAD KM4         / SET THE AC EQUAL TO -4
  273  6275   3152           DCA STORE       / PLACE IN STORE
  274  6276   1360           TAD MASKC       / GET MASK 7000
  275  6277   3153           DCA SHIFT       / PLACE IT IN SHIFT
  276  6300   5333           JMP LOAD        / GO TO LOAD
  277
  278  6301   1333   DIG1,   TAD LOAD        / SET THE AC EQUAL TO 1000
  279  6302   3150           DCA TEMP        / PLACE IN THE IOT WORD
  280  6303   1371           TAD KM11        / SET THE AC EQUAL TO -11
  281  6304   3147           DCA POINT       / PLACE IN POINT
  282  6305   1367           TAD KM7         / SET THE AC EQUAL TO -7
  283  6306   3152           DCA STORE       / PLACE IN STORE
  284  6307   1361           TAD MASK1       / GET THE MASK 0700
  285  6310   3153           DCA SHIFT       / PLACE IN SHIFT
  286  6311   5333           JMP LOAD        / GO TO LOAD
  287
  288  6312   7332   DIG2,   CLA CLL CML RTR / SET THE AC EQUAL TO 2000
  289  6313   3150           DCA TEMP        / PLACE THE BIT CODE IN THE IOT WORD
  290  6314   1364           TAD KM14        / SET THE AC EQUAL TO -14
  291  6315   3147           DCA POINT       / PLACE IT IN POINT
  292  6316   1370           TAD KM10        / SET THE AC EQUAL TO -10
  293  6317   3152           DCA STORE       / PLACE IT IN STORE
```

```
6320  1362          TAD MASK2       / GET MASK 0070
6321  3153          DCA SHIFT       / PLACE IT IN SHIFT
6322  5333          JMP LOAD        / GO TO LOAD

6323  7330  DIG3,   CLA CLL CML RAR / SET THE AC EQUAL TO 4000
6324  3150          DCA TEMP        / PLACE IT IN THE IOT WORD
6325  1366          TAD KM4         / SET THE AC EQLAL TO -4
6326  3147          DCA POINT       / PLACE IT IN POINT
6327  7240          CLA CMA         / SET THE AC EQUAL TO -1
6330  3152          DCA STORE       / PLACE IT IN STORE
6331  1363          TAD MASK3       / GET MASK 0007
6332  3153          DCA SHIFT       / PLACE IT IN SHIFT

6333  1000  LOAD,   TAD SAVPC      / GET THE USER PC
6334  0153          AND SHIFT       / MASK OUT THE DESIRED DIGIT
6335  2147          ISZ POINT       / ENTER THE SHIFT LOOP
6336  5340          JMP .+2
6337  5342          JMP .+3         / SHIFT TO POSITION FOR DISPLAY #1
6340  7004          RAL
6341  5335          JMP .-4
6342  1150          TAD TEMP        / COMBINE WITH THE IOT WORD
6343  3150          DCA TEMP        / RESTORE THE IOT WORD
6344  1400          TAD I SAVPC     / GET THE MEMORY DATA THRU THE
                                    / USER PC
6345  0153          AND SHIFT       / MASK OUT THE DESIRED DIGIT
6346  2152          ISZ STORE       / ENTER THE SHIFT LOOP
6347  5351          JMP .+2
6350  5353          JMP .+3
6351  7004          RAL             / SHIFT TO THE PROPER LOCATION FOR
                                    / DISPLAY #2
6352  5346          JMP .-4
6353  1150          TAD TEMP        / COMBINE WITH THE IOT WORD
6354  6400          6400            / LOAD IT TO THE DISPLAY
6355  2143          ISZ STATUS      / UPDATE THE DIGIT CODE
6356  7300          CLA CLL
6357  5246          JMP THRU        / GO TO THE EXIT POINT OF THE ROUTINE

6360  7000  MASK0,  7000
6361  0700  MASK1,  0700
6362  0070  MASK2,  0070
6363  0007  MASK3,  0007

6364  7761  KM14,   7761
6365  7767  KM8,    7767
6366  7773  KM4,    7773
6367  7770  KM7,    7770
6370  7765  KM10,   7765
6371  7764  KM11,   7764

6372  7773  MSK1,   7773
6373  0003  MSK2,   0003


                                    / THE WAIT FOR CLEAR KEYPAD ROUTINE

      6110  *6110

6110  3154  CLKPD,  DCA SAV1        / SAVE THE AC IN SAV1
6111  4161          CALL            / CALL THE SWITCH DEBOUNCE
6112  6200          SWDB
6113  7440          SZA             / TEST; IS THE KEYPAD CLEAR?
6114  5311          JMP .-3         / NO; GO BACK UNTIL IT IS
6115  1154          TAD SAV1        / YES; RESTORE THE AC
6116  5564          RETURN          / GO BACK TO THE PROGRAM


                                    / THE UNDEFINED CONTROL STATE

6117  4161  UDCS,   CALL           / WAIT FOR THE KEYPAD TO CLEAR
6120  6110          CLKFD
6121  4161          CALL            / LOOK FOR A HEX DIGIT FROM THE
                                    / KEYPAD
6122  6425          HEX
6123  1330          TAD GOTO        / ADJUST A POINTER TO THE TABLE
6124  3147          DCA POINT       / PLACE THE POINTER IN POINT
6125  1547          TAD I POINT     / GET THE ROUTINE ADDRESS
6126  3147          DCA POINT       / PLACE THAT ADDRESS IN POINT
6127  5547          JMP I POINT     / GO TO THAT ROUTINE
6130  6131  GOTO,   GOTO+1
6131  6633          MICRC
6132  7600          BIN
6133  6566          BLK
6134  7301          SIN             / THIS IS THE TABLE OF ROUTINE
6135  6411          RUN             / ADDRESSES
6136  6407          HALT
6137  6414          RESET
6140  6600          SETPC
6141  6403          DECPC
6142  6524          DEP
6143  6400          INCAC
6144  6117          UDCS


                                    / THE INCREMENT AC ROUTINE

      6400  *6400

6400  2140  INCAC,  ISZ SAVAC       / INCREMENT THE AC
6401  7000          NOP             / IN CASE AC WAS 7777
6402  5746          JMP I UG        / GO TO EXIT


                                    / THE DECREMENT PC ROUTINE

6403  7340  DECPC,  CLA CLL CMA     / SET THE AC EQUAL TO -1
6404  1000          TAD SAVPC       / ADD THE USER PC
6405  3400          DCA SAVPC       / RESTORE THE DECREMENTED USER PC
6406  5747          JMP I BUG       / GO TO UNDEFINED CONTROL STATE


                                    / THE HALT SUBROUTINE

6407  7402  HALT,   HLT             / CLEAR THE RUN FLIP/FLOP
6410  5746          JMP I UG        / GO TO EXIT


                                    / THE RUN ROUTINE

6411  7402  RUN,    HLT             / CLEAR THE RUN FLIP/FLOP
6412  6407          6407            / IOT RUN COMMAND
6413  5746          JMP I UG        / GO TO EXIT


                                    / THE RESET ROUTINE

6414  7300  RESET,  CLA CLL         / CLEAR THE AC AND LINK
6415  3140          DCA SAVAC       / CLEAR THE USER AC
6416  3142          DCA SAVFL       / CLEAR THE FLAGS
6417  3141          DCA SAVMQ       / CLEAR THE MQ
6420  3143          DCA STATUS      / CLEAR THE STATUS WORD
6421  6406          6406            / IOT RESET COMMAND
6422  7040          CMA             / SET THE AC EQLAL TO 7777
6423  3000          DCA SAVPC       / SET THE PC EQUAL TO 7777
6424  5207          JMP HALT        / GO TO THE HALT ROUTINE TO CLEAR
                                    / THE RUN FLIP/FLOP


                                    / THE HEX DIGIT ROUTINE
                                    / TAKES A KEYPRESS FROM THE KEYPAD
                                    / AND CONVERTS IT TO A HEX BINARY
                                    / DIGIT FROM 0 TO 12

6425  4161  HEX,    CALL            / GET A KEYPAD VALUE
6426  6200          SWDB
6427  7450          SNA             / TEST FOR A KEYPRESS
6430  5225          JMP HEX         / NO; GO BACK UNTIL THERE IS A
                                    / KEYPRESS
6431  7004          RAL             / YES; POSITION BITS #0 AND #1
6432  7420          SNL             / TEST FOR A "C" KEYPRESS, BIT #0
6433  5237          JMP .+4         / NO; GO ON
6434  7307          CLA CLL IAC RTL / YES; SET THE AC EQUAL TO 0004
6435  1323          TAD K0007       / ADJUST THE AC TO 0013, HEX = B
6436  5564          RETURN          / GO BACK TO THE PROGRAM
6437  7500          SMA             / TEST FOR A "S" KEYPRESS, BIT #1
6440  5244          JMP .+4         / NO; GO ON
6441  7325          CLA CLL CML IAC RAL / YES; SET THE AC EQUAL TO 0003
6442  1323          TAD K0007       / ADJUST THE AC TO 0012, HEX = A
6443  5564          RETURN          / GO BACK TO THE PROGRAM
6444  7006          RTL             / POSITION BITS #2 AND #3
6445  7420          SNL             / TEST FOR A "9" KEYPRESS, BIT #2
6446  5252          JMP .+4         / NO; GO ON
6447  7305          CLA CLL IAC RAL / YES; SET THE AC EQUAL TO 0002
6450  1323          TAD K0007       / ADJUST THE AC TO 0011, HEX = 9
6451  5564          RETURN          / GO BACK TO THE PROGRAM
6452  7500          SMA             / TEST FOR "7" KEYPRESS, BIT #3
6453  5257          JMP .+4         / NO; GO ON
6454  7327          CLA CLL CML IAC RTL / YES; SET THE AC EQUAL TO 0006
6455  7001          IAC             / SET THE AC EQLAL TO 0007, HEX = 7
6456  5564          RETURN          / GO BACK TO THE PROGRAM
6457  7006          RTL             / POSITION BITS #4 AND #5
6460  7420          SNL             / TEST FOR "8" KEYPRESS, BIT #4
6461  5265          JMP .+4         / NO; GO ON
6462  7301          CLA CLL IAC     / YES; SET THE AC EQUAL TO 0001
6463  1323          TAD K0007       / ADJUST THE AC TO 0010, HEX = 8
6464  5564          RETURN          / GO BACK TO THE PROGRAM
6465  7500          SMA             / TEST FOR "6" KEYPRESS, BIT #5
6466  5271          JMP .+3         / NO; GO ON
6467  7327          CLA CLL CML IAC RTL / YES; SET THE AC EQUAL TO 0006
                                    / HEX = 6
6470  5564          RETURN          / GO BACK TO THE PROGRAM
6471  7006          RTL             / POSITION BITS #6 AND #7
6472  7420          SNL             / TEST FOR "5" KEYPRESS, BIT #6
6473  5277          JMP .+4         / NO; GO ON
6474  7307          CLA CLL IAC RTL / YES; SET THE AC EQUAL TO 0004
6475  7001          IAC             / SET THE AC EQLAL TO 0005, HEX = 5
6476  5564          RETURN          / RETURN TO THE PROGRAM
6477  7500          SMA             / TEST FOR A "4" KEYPRESS, BIT #7
6500  5303          JMP .+3         / NO; GO ON
6501  7307          CLA CLL IAC RTL / YES; SET THE AC EQUAL TO 0004,
                                    / HEX = 4
6502  5564          RETURN          / GO BACK TO THE PROGRAM
6503  7006          RTL             / POSITION BITS #8 AND #9
6504  7420          SNL             / TEST FOR "3" KEYPRESS, BIT #8
6505  5310          JMP .+3         / NO; GO ON
6506  7325          CLA CLL CML IAC RAL / YES; SET THE AC EQUAL TO 0003,
                                    / HEX = 3
6507  5564          RETURN          / GO BACK TO THE PROGRAM
6510  7500          SMA             / TEST FOR A "2" KEYPRESS, BIT #9
6511  5314          JMP .+3         / NO; GO ON
6512  7305          CLA CLL IAC RAL / YES; SET THE AC EQUAL TO 0002,
                                    / HEX = 2
6513  5564          RETURN          / GO BACK TO THE PROGRAM
6514  7006          RTL             / POSITION BITS #10 AND #11
6515  7420          SNL             / TEST FOR A "1" KEYPRESS
6516  5321          JMP .+3         / NO; GO ON
6517  7301          CLA CLL IAC     / YES SET THE AC EQUAL TO 0001,
                                    / HEX = 1
6520  5564          RETURN          / GO BACK TO THE PROGRAM
6521  7300          CLA CLL         / DEFAULT CONDITION; IT MUST HAVE BEEN
                                    / THE "0" KEY SO SET THE AC EQUAL TO
                                    / 0000, HEX = 0
6522  5564          RETURN          / RETURN TO THE PROGRAM

6523  0007  K0007,  0007


                                    / THE DEPOSIT INTO MEMORY ROUTINE

6524  4161  DEP,    CALL            / WAIT FOR THE KEYPAD TO CLEAR
6525  6110          CLKFD
6526  4161          CALL            / GET A HEX VALUE FROM THE KEYPAD
6527  6425          HEX
6530  3150          DCA TEMP        / PLACE IT IN TEMP
6531  1150          TAD TEMP        / GET THE VALUE FROM TEMP
6532  0333          AND SNERD       / MASK OUT BIT #8
6533  7650  SNERD,  SNA CLA         / TEST; IS VALUE > 7
6534  5350          JMP PON         / NO; GO TO LOAD MEMORY
6535  1150          TAD TEMP        / YES; GET THE VALUE
6536  1343          TAD TAB         / ADJUST TO POINT AT THE TABLE
6537  3150          DCA TEMP        / AND PLACE IT IN A POINTER LOCATION
6540  1550          TAD I TEMP      / GET THE ADDRESS OF THE ROUTINE
                                    / FROM THE TABLE
6541  3150          DCA TEMP        / AND PLACE IT IN TEMP
6542  5550          JMP I TEMP      / GO TO THE PROPER ROUTINE

6543  6534  TAB,    TAB-7
6544  6557          DCI
6545  6563          PCI
6546  6061  UG,     EXIT
6547  6117  BUG,    UDCS


                                    / THE INCREMENT AC ROUTINE

6550  1400  PCA,    TAD I SAVPC     / GET THE MEMORY DATA THRU SAVPC
6551  7104          CLL RAL         / SHIFT IT OVER ONE DIGIT
6552  7104          CLL RAL
6553  7104          CLL RAL
6554  1150          TAD TEMP        / ADD IN THE NEW DIGIT
6555  3400          DCA I SAVPC     / RESTORE THE DATA TO THE MEMORY
6556  5324          JMP DEP         / GO GET THE NEXT DIGIT

6557  7360  DCI,    CLA CLL CML CMA / SET THE AC AND LINK
6560  1000          TAD SAVPC       / DECREMENT THE PC
6561  3000          DCA SAVPC       / RESTORE THE PC
6562  5324          JMP DEP         / GO GET THE NEXT DIGIT


6563  2000  PCI,    ISZ SAVPC       / INCREMENT THE PC
6564  7000          NOP             / IN CASE THE PC WAS 7777
6565  5324          JMP DEP         / GO GET THE NEXT DIGIT


                                    / THE BLANK FLAG TOGGLE ROUTINE

6566  7300  BLK,    CLA CLL         / CLEAR THE AC AND LINK
6567  1143          TAD STATUS      / GET THE STATUS WORD
6570  7002          BSW             / POSITION THE BLANK FLAG, BIT #6
6571  7004          RAL             / PUT THE FLAG INTO THE LINK
6572  7030          CML RAR         / TOGGLE THE FLAG AND RESTORE
6573  7002          BSW             / RESTORE THE POSITION OF AC
6574  3143          DCA STATUS      / RESTORE THE STATUS WORD
6575  5746          JMP I UG        / GO TO EXIT


                                    / THE SET PROGRAM COUNTER (PC)
                                    / SUBROUTINE

      6600  *6600

6600  4161  SETPC,  CALL            / WAIT FOR THE KEYPAD TO CLEAR
6601  6110          CLKFD
6602  4161          CALL            / GET A HEX VALUE FROM THE KEYPAD
6603  6425          HEX
6604  3150          DCA TEMP        / STORE IN TEMP
6605  1150          TAD TEMP        / GET THE VALUE FROM TEMP
6606  0207          AND MASK        / MASK OUT BIT #8
6607  7650  MASK,   SNA CLA         / TEST; IS VALUE > 7
6610  5224          JMP GOON        / NO; GO ON TO LOAD THE PC
6611  1150          TAD TEMP        / YES; GET THE VALUE
6612  1217          TAD ADJT        / ADJUST TO POINT AT THE TABLE
6613  3150          DCA TEMP        / PLACE IN A POINTER WORD
6614  1550          TAD I TEMP      / GET THE JUMP ADDRESS FROM THE TABLE
```

```
6615 3150          DCA TEMP          / PLACE THAT ADDRESS IN THE POINTER
6616 5550          JMP I TEMP        / GO TO THE PROPER ROUTINE

6617 6610  ADJT,   ADJT-7
6620 6600          SETPC
6621 6600          SETPC
6622 6061          EXIT
6623 6117          UDCS

6624 1000  GOON,   TAD SAVPC         / GET THE PC
6625 7104          CLL RAL           / SHIFT IT OVER ONE DIGIT
6626 7104          CLL RAL
6627 7104          CLL RAL
6630 1150          TAD TEMP          / PLACE THE NEW DIGIT
6631 3000  K3000,  DCA SAVPC        / PLACE IN THE PC LOCATION
6632 5200          JMP SETPC         / GO BACK FOR A NEW DIGIT

                                     / THIS IS THE MICRO ASSEMBLER
                                     / PROGRAM WHICH IS ENTERED FROM THE
                                     / MONITOR BY DEPRESSING THE
                                     / MONITOR SELECT KEY

6633 4161  MICRC,  CALL              / WAIT FOR A CLEAR KEYBOARD
6634 6110          CLKPD
6635 4161          CALL              / GET A HEX VALUE FROM THE KEYPAD
6636 6425          HEX
6637 1244          TAD XEQ           / ADJUST A POINTER TO THE TABLE
6640 3147          DCA POINT         / PLACE IN THE POINTER
6641 1547          TAD I POINT       / GET THE JUMP ADDRESS FROM THE TABLE
6642 3147          DCA POINT         / PLACE THIS ADDRESS IN THE POINTER
6643 5547          JMP I POINT       / GO TO THE PROPER ROUTINE

6644 6645  XEQ,    XEQ+1
6645 6701          AAND              / THIS IS THE TABLE OF JUMP ADDRESSES
6646 6664          ATAD
6647 6666          AISZ
6650 6671          ADCA
6651 6673          AJMS
6652 6676          AJMP
6653 7027          AIOT
6654 7153          ACPR2
6655 7071          ACPR1
6656 7204          ACPR3
6657 6661          AINC
6660 6117          UDCS

6661 2000  AINC,   ISZ SAVPC         / INCREMENT THE USER PC
6662 5233          JMP MICRC         / GO ASSEMBLE THE NEXT INSTRUCTION
6663 5233          JMP MICRC         / IN CASE THE USER PC WAS 7777

6664 1224  ATAD,   TAD GOON          / SET THE AC EQUAL TO 1000
6665 5301          JMP AAND          / GO TO MRPA

6666 1270  AISZ,   TAD K2000         / SET THE AC EQUAL TO 2000
6667 5301          JMP AAND          / GO TO MRPA
6670 2000  K2000,  2000

6671 1231  ADCA,   TAD K3000         / SET THE AC EQUAL TO 3000
6672 5301          JMP AAND          / GO TO MRPA

6673 1275  AJMS,   TAD K4000         / SET THE AC EQUAL TO 4000
6674 5301          JMP AAND          / GO TO MRPA
6675 4000  K4000,  4000

6676 1300  AJMP,   TAD K5000         / SET THE AC EQUAL TO 5000
6677 5301          JMP AAND          / GO TO MRPA
6700 5000  K5000,  5000

6701 3400  AAND,   DCA I SAVPC       / PLACE THE OP CODE INTO THE INSTRUCTION
6702 3156          DCA SAV3          / CLEAR THE ABSOLUTE ADDRESS
6703 4161  MRPA,   CALL              / WAIT FOR A CLEAR KEYPAD
6704 6110          CLKPD
6705 4161          CALL              / GET A HEX VALUE FROM THE KEYPAD
6706 6425          HEX
6707 3150          DCA TEMP          / PLACE IN TEMP
6710 1150          TAD TEMP          / GET THE VALUE
6711 0312          AND ZOT           / MASK OUT BIT #8
6712 7650  ZOT,    SNA CLA           / TEST; IS THE VALUE > 7
6713 5344          JMP TCZ           / NO; GO TO LOAD THE ADDRESS
6714 1150          TAD TEMP          / YES; GET THE VALUE AGAIN
6715 1325          TAD BOOB          / ADD -11
6716 7650          SNA CLA           / TEST; IS THERE A "C" KEYPRESS
6717 5336          JMP NEXT          / YES; GO ASSEMBLE THE NEXT INSTRUCTION
6720 1150          TAD TEMP          / NO; GET THE VALUE
6721 1326          TAD BOOB2         / ADD -10
6722 7650          SNA CLA           / TEST; IS THERE A "S" KEYPRESS
6723 5327          JMP ZONK          / YES; SET THE INDIRECT BIT
6724 5303          JMP MRPA          / NO; GO BACK AND GET ANOTHER VALUE

6725 7765  BOOB,   7765
6726 7766  BOOB2,  7766              / -11 AND -10

6727 1400  ZONK,   TAD I SAVPC       / GET THE INSTRUCTION
6730 7106          CLL RTL           / CLEAR THE LINK AND ROTATE THE AC TWICE
                                     / LEFT
6731 7006          RTL               / PLACE THE INDIRECT BIT IN THE LINK
6732 7132          CLL CML RTR       / SET THE INDIRECT BIT AND ROTATE
6733 7012          RTR               / BACK
6734 3400          DCA I SAVPC       / RESTORE THE INSTRUCTION
6735 5303          JMP MRPA          / GO GET THE NEXT VALUE

6736 2000  NEXT,   ISZ SAVPC         / INCREMENT THE USER PC
6737 5233          JMP MICRO         / GO TO MICRO
6740 5233          JMP MICRC         / IN CASE PC EQUAL 7777

6741 0177  TUC,    0177
6742 7600  TUC1,   7600
6743 7400  TUC2,   7400

6744 1156  TCZ,    TAD SAV3          / GET THE ABSOLUTE ADDRESS
6745 7104          CLL RAL           / SHIFT IT OVER ONE DIGIT
6746 7104          CLL RAL           /
6747 7104          CLL RAL
6750 1150          TAD TEMP          / ADD IN THE NEW DIGIT
6751 3156          DCA SAV3          / RESTORE THE ABSOLUTE ADDRESS
6752 1400  PLP,    TAD I SAVPC       / GET THE ABSOLUTE ADDRESS
6753 0343          AND TUG2          / MASK OUT THE OP CODE AND INDIRECT
                                     / BIT
6754 3400          DCA I SAVPC       / REPLACE THE INSTRUCTION
6755 1156          TAD SAV3          / GET THE ABSOLUTE ADDRESS
6756 0341          AND TUC           / MASK OUT THE PAGE ADDRESS
6757 1400          TAD I SAVPC       / COMBINE WITH THE INSTRUCTION
6760 3400          DCA I SAVPC       / PLACE THE INSTRUCTION IN MEMORY
6761 1156          TAD SAV3          / GET THE ABSOLUTE ADDRESS
6762 0342          AND TUG1          / MASK OUT BITS #0 THRU #4
6763 7450          SNA               / TEST; IS THE ABSOLUTE ADDRESS IN
                                     / PAGE ZERO
6764 5303          JMP MRPA          / YES; GO GET THE NEXT DIGIT
6765 3146          DCA HOLD          / NO; PLACE THE BITS #0 THRU #4 IN HOLD
6766 1000          TAD SAVPC         / GET THE USER PC
6767 0342          AND TUG1          / MASK OUT BITS #0 THRU #4
6770 7041          CIA               / NEGATE THE VALUE
6771 1146          TAD HOLD          / ADD THE ABSOLUTE PAGE NUMBER
6772 7640          SZA CLA           / TEST; IS THE ABSOLUTE ADDRESS IN THE
                                     / CURRENT PAGE
6773 5377          JMP FLASH         / NO; GO TO ERROR
6774 4161          CALL              / SET THE CURRENT PAGE BIT
6775 7255          BSET4
6776 5303          JMP MRPA          / GO GET THE NEXT DIGIT

6777 3156  FLASH,  DCA SAV3          / ERROR CONDITION. FLASH THE DISPLAY
                                     / TO INDICATE AN ATTEMPT TO LOAD AN
                                     / ADDRESS THAT IS NOT IN THE CURRENT
                                     / PAGE OR PAGE ZERO. CLEAR THE ABSOLUTE
                                     / ADDRESS.
7000 4161          CALL              / WAIT FOR A CLEAR KEYPAD
7001 6110          CLKPD
7002 6400          6400              / BLANK THE DISPLAY
7003 1223          TAD TKA           / GET THE TIME CONSTANT #A
7004 3144          DCA TIME          / PLACE IT IN THE TIMER
7005 3150          DCA TEMP          / CLEAR TEMP
7006 2150          ISZ TEMP          / COUNT OUT TEMP 4096 TIMES
7007 5206          JMP .-1
7010 2144          ISZ TIME          / COUNT OUT TIME
7011 5206          JMP .-3
7012 1224          TAD TK9           / GET TIME CONSTANT #B
7013 3160          DCA SAV5          / PLACE IN SAV5
7014 4161          CALL              / GET A KEYPAD VALUE
7015 6200          SWDB
7016 7640          SZA CLA           / TEST; IS THERE A KEYBOARD PRESS
7017 5625          JMP I SCUP        / YES; GO TO GET THE PROPER ADDRESS
7020 2160          ISZ SAV5          / NO; COUNT TIME
7021 5214          JMP .-5           / GO BACK TO SWDB
7022 5626          JMP I ZOK         / GO BACK TO FLASH THE DISPLAY

7023 7765  TKA,    7765              / -0012  (-10)
7024 7757  TK9,    7757              / -0010  (-16)

7025 6752  SCUP,   PUP
7026 6777  ZOK,    FLASH

7027 7333  AIOT,   CLA CLL CML IAC RTR / SET THE AC EQUAL TO 6000
7030 3400          DCA I SAVPC       / PLACE IT IN THE INSTRUCTION
7031 3156          DCA SAV3          / CLEAR THE ABSOLUTE ADDRESS
7032 4161          CALL              / WAIT FOR A CLEAR KEYPAD
7033 6110          CLKPD
7034 4161          CALL              / GET A VALUE FROM THE KEYPAD
7035 6425          HEX
7036 3150          DCA TEMP          / PLACE IN TEMP
7037 1150          TAD TEMP          / GET THE VALUE
7040 0241          AND C70           / MASK OUT BIT #8
7041 7650  CCB,    SNA CLA           / TEST; IS THE VALUE > 7
7042 5253          JMP SCB           / NO; GO TO ADDRESS LOAD
7043 1150          TAD TEMP          / YES; GET THE VALUE
7044 1250          TAD SNCT          / ADD -11
7045 7650          SNA CLA           / TEST; IS IT A "C" KEYPRESS
7046 5720          JMP I SCT         / YES; GO TO NEXT
7047 5232          JMP AIOT+3        / NO; DEFAULT, GO GET THE NEXT DIGIT

7050 7765  SNCT,   7765              / -0013  (-11)
7051 7000  SCC,    7000
7052 0777  CUS,    0777

7053 1400  SCB,    TAD I SAVPC       / GET THE INSTRUCTION
7054 0251          AND SOC           / MASK OUT BITS #0 THRU #2
7055 3400          DCA I SAVPC       / PLACE BACK IN THE MEMORY
7056 1156          TAD SAV3          / GET THE ABSOLUTE ADDRESS
7057 7104          CLL RAL           / ROTATE IT OVER ONE DIGIT
7060 7104          CLL RAL
7061 7104          CLL RAL
7062 1150          TAD TEMP          / ADD IN THE NEW DIGIT
7063 0252          AND CUS           / BOUND THE ADDRESS TO BITS #3 THRU #11
7064 3156          DCA SAV3          / PLACE THE NEW ADDRESS IN SAV3
7065 1156          TAD SAV3          / GET THE NEW ADDRESS
7066 1400          TAD I SAVPC       / COMBINE WITH THE INSTRUCTION
7067 3400          DCA I SAVPC       / PLACE THE NEW INSTRUCTION INTO
                                     / THE MEMORY
7070 5232          JMP AIOT+3        / GO GET THE NEXT DIGIT

                                     / THE OPERATE GROUPS ASSEMBLY
                                     / ROUTINES

7071 1251  ACPR1,  TAD SOC           / SET THE AC EQUAL TO 7000
7072 3400          DCA I SAVPC       / PLACE THIS IN THE INSTRUCTION
7073 4161          CALL              / WAIT FOR THE KEYBOAD TO CLEAR
7074 6110          CLKPD
7075 4161          CALL              / GET A HEX VALUE FROM THE KEYPAD
7076 6425          HEX
7077 1304          TAD GUM           / ADJUST A POINTER TO THE TABLE
7100 3147          DCA POINT         / PLACE IN THE POINTER
7101 1547          TAD I POINT       / GET THE JUMP ADDRESS FROM THE TABLE
7102 3147          DCA POINT         / PLACE IN THE POINTER
7103 5547          JMP I POINT       / GO TO THE PROPER ROUTINE

7104 7105  GUM,    GUM+1             / THE TABLE ADJUSTMENT VALUE
7105 7073          ACPR1+2           / THE TABLE OF JUMP ADDRESSES
7106 7143          JA1C
7107 7140          JA9
7110 7135          JA8
7111 7132          JA7
7112 7127          JA6
7113 7124          JA5
7114 7121          JA4
7115 7073          ACPR1+2
7116 7073          ACPR1+2
7117 7146          BSET11
7120 6736  SO1     NEXT

7121 4161  JA4,    CALL              / GO TO THE APROPRIATE BIT SET ROUTINE
7122 7255          BSET4
7123 5273          JMP ACPR1+2

7124 4161  JA5,    CALL
7125 7257          BSET5
7126 5273          JMP ACPR1+2

7127 4161  JA6,    CALL
7130 7261          BSET6
7131 5273          JMP ACPR1+2

7132 4161  JA7,    CALL
7133 7263          BSET7
7134 5273          JMP ACPR1+2

7135 4161  JA8,    CALL
7136 7265          BSET8
7137 5273          JMP ACPR1+2

7140 4161  JA9,    CALL
7141 7267          BSET9
7142 5273          JMP ACPR1+2

7143 4161  JA10,   CALL
7144 7271          BSET10
7145 5273          JMP ACPR1+2

7146 1400  BSET11, TAD I SAVPC       / GET THE INSTRUCTION
7147 7010          RAR               / POSITION BIT #11 IN THE LINK
7150 7124          CLL CML RAL       / SET THE LINK AND ROTATE THE WORD BACK
7151 3400          DCA I SAVPC       / RESTORE TO THE MEMORY
7152 5273          JMP ACPR1+2

7153 1366  ACPR2,  TAD ZOL2          / SET THE AC EQUAL TO 7400
7154 3400          DCA I SAVPC       / STORE IN THE INSTRUCTION
7155 4161          CALL              / WAIT FOR THE KEYBOAD TO CLEAR
```

```
 897  7156  6110          CLKPD
 898  7157  4161          CALL                    / GET A HEX VALLE FROM THE KEYPAD
 899  7160  6425          HEX
 900  7161  1367          TAD GUB                 / ADJUST A POINTER TO THE TABLE
 901  7162  3147          DCA POINT               / PLACE IN THE POINTER
 902  7163  1547          TAD I POINT             / GET THE JUMP ADDRESS FROM THE TABLE
 903  7164  3147          DCA POINT               / PLACE IN THE POINTER
 904  7165  5547          JMP I POINT             / GO TO THE PROPER ROUTINE
 905
 906
 907  7166  7400  ZCL2,   7400                  / OPERATE GROUP 2 OP CODE
 908
 909  7167  7170  GUB,    GUB+1                 / THE TABLE ADJUSTMENT VALUE
 910  7170  6156          JB7                     / THE TABLE OF JUMP ADDRESSES
 911  7171  6164          JB9
 912  7172  7155          ACPR2+2
 913  7173  7155          ACPR2+2
 914  7174  7155          ACPR2+2
 915  7175  6167          JB10
 916  7176  7155          ACPR2+2
 917  7177  6145          JP4
 918  7200  6153          JB6
 919  7201  6150          JB5
 920  7202  6161          JB9
 921  7203  6736          NEXT
 922
 923        6145  *6145
 924
 925  6145  4161  JB4,    CALL                    / GO TO THE APROPRIATE BIT SET ROUTINE
 926  6146  7255          BSET4
 927  6147  5772          JMP I TOK               / GO BACK TO ACPR2
 928
 929  6150  4161  JB5,    CALL
 930  6151  7257          BSET5
 931  6152  5772          JMP I TOK
 932
 933  6153  4161  JB6,    CALL
 934  6154  7261          BSET6
 935  6155  5772          JMP I TOK
 936
 937  6156  4161  JB7,    CALL
 938  6157  7263          BSET7
 939  6160  5772          JMP I TOK
 940
 941  6161  4161  JB8,    CALL
 942  6162  7265          BSET8
 943  6163  5772          JMP I TOK
 944
 945  6164  4161  JB9,    CALL
 946  6165  7267          BSET9
 947  6166  5772          JMP I TOK
 948
 949  6167  4161  JB10,   CALL
 950  6170  7271          BSET10
 951  6171  5772          JMP I TOK
 952
 953  6172  7155  TOK,    ACPR2+2                 / POINTER TO GO BACK
 954
 955
 956
 957        7204  *7204
 958
 959  7204  1217  ACPR3,  TAD ZCL3                / SET THE AC EQUAL TO 7401
 960  7205  3400          DCA I SAVPC             / PLACE THIS IN THE INSTRUCTION
 961  7206  4161          CALL                    / WAIT FOR THE KEYBOAD TO CLEAR
 962  7207  6110          CLKPD
 963  7210  4161          CALL                    / GET A HEX VALUE FROM THE KEYPAD
 964  7211  6425          HEX
 965  7212  1220          TAD BOB                 / ADJUST A POINTER TO THE TABLE
 966  7213  3147          DCA POINT               / PLACE IN THE POINTER
 967  7214  1547          TAD I POINT             / GET THE JUMP ADDRESS FROM THE TABLE
 968  7215  3147          DCA POINT               / PLACE IN THE POINTER
 969  7216  5547          JMP I POINT             / GO TO THE PROPER ROUTINE
 970
 971
 972
 973  7217  7401  ZCL3,   7401                  / OPERATE GROUP 3 OP CODE
 974
 975  7220  7221  BOB,    BOB+1
 976  7221  7206          ACPR3+2                 / TABLE OF JUMP ADDRESSES
 977  7222  7206          ACPR3+2
 978  7223  7206          ACPR3+2
 979  7224  7206          ACPR3+2
 980  7225  7206          ACPR3+2
 981  7226  7206          ACPR3+2
 982  7227  7206          ACPR3+2
 983  7230  7235          JC4
 984  7231  7243          JC7
 985  7232  7240          JC5
 986  7233  7206          ACPR3+2
 987  7234  6736          NEXT
 988
 989
 990  7235  4161  JC4,    CALL
 991  7236  7255          BSET4
 992  7237  5206          JMP ACPR3+2
 993
 994  7240  4161  JC5,    CALL
 995  7241  7257          BSET5
 996  7242  5206          JMP ACPR3+2
 997
 998  7243  4161  JC7,    CALL
 999  7244  7263          BSET7
1000  7245  5206          JMP ACPR3+2
1001
1002
1003
1004
1005
1006                                            / THE BIT SET SUBROUTINES
1007
1008
1009  7246  0002  AAB,    0002                  / SET BIT #10
1010  7247  0004  AAE,    0004                  / SET BIT #9
1011  7250  0010  AAC,    0010                  / SET BIT #8
1012  7251  0020  AAD,    0020                  / SET BIT #7
1013  7252  0040  AAE,    0040                  / SET BIT #6
1014  7253  0100  AAF,    0100                  / SET BIT #5
1015  7254  0200  AAG,    0200                  / SET BIT #4
1016
1017  7255  1254  BSET4,  TAD AAG               / GET THE PROPER SET WORD
1018  7256  5273          JMP MBST
1019
1020  7257  1253  BSET5,  TAD AAF
1021  7260  5273          JMP MBST
1022
1023  7261  1252  BSET6,  TAD AAE
1024  7262  5273          JMP MBST
1025
1026  7263  1251  BSET7,  TAD AAD
1027  7264  5273          JMP MBST
1028
1029  7265  1250  BSET8,  TAD AAC
1030  7266  5273          JMP MBST
1031
1032  7267  1247  BSET9,  TAD AAB
1033  7270  5273          JMP MBST
1034
1035  7271  1246  BSET10, TAD AAA
1036  7272  5273          JMP MBST
1037
1038
1039  7273  7521  MEST,   SWP                     / PLACE THE SET CONSTANT IN THE MQ
1040  7274  7300          CLA CLL                 / CLEAR THE AC
1041  7275  1400          TAD I SAVPC             / GET THE INSTRUCTION
1042  7276  7501          MQA                     / OR IN THE SET CONSTANT
1043  7277  3400          DCA I SAVPC             / REPLACE THE INSTRUCTION
1044  7300  5564          RETURN                  / GO BACK TO THE PROGRAM
1045
1046
1047
1048
1049  7301  4161  SIN,    CALL                    / WAIT FOR A CLEAR KEYPAD
1050  7302  6110          CLKPD
1051  7303  7300          CLA CLL                 / CLEAR THE AC
1052  7304  1323          TAD KJMP                / GET THE INSTRUCTION "JUMP I SHIFT"
1053  7305  3152          DCA STORE               / PLACE IT IN PAGE ZERO
1054  7306  1326          TAD PLUM                / GET THE RETURN ADDRESS
1055  7307  3153          DCA SHIFT               / PLACE IT IN SHIFT
1056  7310  1400          TAD I SAVPC             / GET THE INSTRUCTION TO BE PREFORMED
1057  7311  0324          AND MK                  / MASK OUT THE OPCODE
1058  7312  1325          TAD KIT                 / ADD -4000
1059  7313  765C          SNA CLA                 / TEST; IS THE INSTRUCTION A JMS
1060  7314  5356          JMP EJMS                / YES; GO TO THE JMS ROUTINE
1061  7315  14CC          TAD I SAVPC             / NO; GET THE INSTRUCTION
1062  7316  0324          AND MK                  / MASK OUT THE OP CODE
1063  7317  1327          TAD KAT                 / ADD -5000
1064  7320  765C          SNA CLA                 / TEST; IS THE INSTRUCTION A JMP
1065  7321  5363          JMP EJMP                / YES; GO TO THE JMP ROUTINE
1066  7322  5366          JMP EXEC                / NO; GO TO THE EXECUTE ROUTINE
1067
1068  7323  5553  KJMP,   JMP I SHIFT
1069  7324  7000  MK,     7000
1070  7325  4CCC  KIT,    4000
1071  7326  7427  PLUM,   RET
1072  7327  3000  KAT,    3000
1073
1074
1075  7330  14CC  TAAD,   TAD I SAVPC            / GET THE INSTRUCTION
1076  7331  0352          AND NOT                 / MASK OUT THE PAGE ADDRESS
1077  7332  3144          DCA TIME                / PLACE IN TIME
1078  7333  14CC          TAD I SAVPC             / GET THE INSTRUCTION
1079  7334  0354          AND GUT                 / MASK OUT THE CURRENT PAGE BIT
1080  7335  765C          SNA CLA                 / TEST; IS THIS BIT SET
1081  7336  5343          JMP INDB                / NO; NO  GO TO INDB
1082  7337  1000          TAD SAVPC               / YES; GET THE CURRENT ADDRESS
1083  7340  0353          AND PUC                 / MASK OUT THE PAGE NUMBER
1084  7341  1144          TAD TIME                / COMBINE WITH THE PAGE ADDRESS
1085  7342  3144          DCA TIME                / PLACE IN TIME
1086
1087  7343  1400  INDB,   TAD I SAVPC             / GET THE INSTRUCTION
1088  7344  0355          AND LOT                 / MASK OUT THE INDIRECT BIT
1089  7345  765C          SNA CLA                 / TEST; IS THE BIT SET
1090  7346  5564          RETURN                  / NO; RETURN WITH THE TRUE ADDRESS
1091                                              / IN LOCATION TIME
1092  7347  1544          TAD I TIME              / YES; GET THE TRUE ADDRESS
1093  7350  3144          DCA TIME                / PLACE IT IN TIME
1094  7351  5564          RETURN                  / RETURN WITH THE TRUE ADDRESS IN TIME
1095
1096  7352  C177  NC1,    C177
1097  7353  7600  PLC,    7600
1098  7354  C200  GL1,    0200
1099  7355  04CC  LO1,    0400
1100  7356  4161  EJMS,   CALL                    / GET THE JMS ADDRESS
1101  7357  7330          INAD
1102  7360  2144          ISZ TIME                / INCREMENT IT TO "NEXT"
1103  7361  7000          NOP                     / FOR WRAP AROUND
1104  7362  5371          JMP .+7                 / GO TO EXEC
1105
1106  7363  4161  EJMP,   CALL                    / GET THE JMP ADDRESS
1107  7364  7330          INAD
1108  7365  5371          JMP . +4                / GO TO EXEC
1109
1110  7366  1000  EXEC,   TAD SAVPC              / GET THE CURRENT ADDRESS
1111  7367  7001          IAC                     / INCREMENT IT TO NEXT
1112  7370  3144          DCA TIME                / PLACE IT IN TIME
1113
1114                                              / NOW WE CONTINUE ON WITH "NEXT"
1115                                              / IN TIME
1116
1117  7371  1144          TAD TIME                / GET THE NEXT ADDRESS
1118  7372  7001          IAC                     / INCREMENT IT
1119  7373  3145          DCA SAVE                / PLACE IT IN SAVE
1120  7374  1544          TAD I TIME              / GET THE NEXT INSTRUCTION
1121  7375  3154          DCA SAV1                / STORE IT IN SAV1
1122  7376  1545          TAD I SAVE              / GET THE "NEXT+1" INSTRUCTION
1123  7377  3155          DCA SAV2                / PLACE IT IN SAV2
1124  7400  1226          TAD TAL                 / GET THE INSTRUCTION "JMS BACK"
1125  7401  3544          DCA I TIME              / PLACE IT IN THE NEXT LOCATION
1126  7402  1226          TAD TAL                 / GET THE INSTRUCTION AGAIN
1127  7403  3545          DCA I SAVE              / PLACE IT IN THE NEXT+1 LOCATION
1128  7404  1226          TAD TAL                 / ONCE MORE
1129  7405  7C41          CIA                     / NEGATE IT
1130  7406  1544          TAD I TIME              / TEST FOR RAM
1131  7407  7640          SZA CLA                 / TEST; DID IT GET PLACED?
1132  7410  5625          JMP I HOT               / NO; THERE IS NO RAM THERE
1133  7411  1226          TAD TAL                 / YES; ALSO TEST THE NEXT+1 LOCATION
1134  7412  7041          CIA
1135  7413  1545          TAD I SAVE              / ADD THE NEXT+1 LOCATION
1136  7414  7640          SZA CLA                 / TEST; DID IT GET PLACED?
1137  7415  5625          JMP I HOT               / NO; THERE IS NO RAM THERE
1138
1139                                              / EVERYTHING IS OK SO WE CAN
1140                                              / NOW EXECUTE THE INSTRUCTION
1141
1142  7416  1141          TAD SAVMC               / RESTORE THE MQ
1143  7417  7421          MCL
1144  7420  1142          TAD SAVFL               / RESTORE THE LINK
1145  7421  7004          RAL
1146  7422  7200          CLA
1147  7423  1140          TAD SAVAC               / RESTORE THE AC
1148
1149  7424  5400          JMP I SAVPC             / GO EXECUTE THE SINGLE INSTRUCTION
1150
1151
1152
1153  7425  6407  HOT,    HALT                    / HALT ROUTINE POINTER
1154  7426  4151  TAL,    JMS BACK
1155
1156  7427  3140  RET,    DCA SAVAC
1157  7430  6004          GTF                     / SAVE THE AC AND FLAGS
1158  7431  3142          DCA SAVFL
1159  7432  7521          SWP                     / GET THE MQ
1160  7433  3141          DCA SAVMQ               / SAVE IT
1161  7434  1154          TAD SAV1                / GET THE ORIGINAL FIRST INSTRUCTION
1162  7435  3544          DCA I TIME              / REPLACE IT
1163  7436  1155          TAD SAV2                / GET THE OTHER
1164  7437  3545          DCA I SAVE              / REPLACE IT
1165  7440  7340          CLA CLL CMA             / SET THE AC EQUAL TO -1
1166  7441  1151          TAD BACK                / DECREMENT THE RETURN PC
1167  7442  3000          DCA SAVPC               / UPDATE THE USER PC
1168  7443  5644          JMP I .+1
1169  7444  6117          UCDS
1170
1171
1172
1173                                              / THE FOLLOWING ROUTINES USE
1174                                              / THE PIE-UART INTERFACE.
1175
1176                                              / THE PIE-UART INSTRUCTIONS
1177
1178        6160  READ1=6160
1179        6170  READ2=6170
1180        6161  WRITE1=6161
1181        6171  WRITE2=6171
1182        6162  SKIP1=6162
1183        6163  SKIP2=6163
1184        6172  SKIP3=6172
1185        6173  SKIP4=6173
1186        6164  RCRA=6164
1187        6165  WCRA=6165
1188        6175  WCRB=6175
1189        6174  WVR=6174
1190        6166  SFLAG1=6166
1191        6176  SFLAG3=6176
1192        6167  CFLAG1=6167
1193        6177  CFLAG3=6177
1194
1195
1196
```

```
                                        / THE PIE INITALIZE ROUTINE
                                        /
7445 6007  INPIE,  CAF                  / CLEAR ALL FLAGS
7446 6400          6400                 / CLEAR THE DISPLAY
7447 1263          TAD KCRA             / GET THE CRA WORD
7450 6165          WCRA                 / LOAD IT TO CRA IN PIE
7451 7300          CLA CLL
7452 1264          TAD KCRB             / GET THE CRB WORD
7453 6175          WCRB                 / LOAD IT TO THE CRB WORD IN PIE
7454 7300          CLA CLL
7455 1265          TAD KTTY             / GET THE TTY-UART CONTROL WORD
7456 6171          WRITE2               / LOAD IT TO UART CONTROL WORD
7457 7300          CLA CLL              /
7460 6174          WWB                  / WRITE ALL ZEROS INTO THE VECTOR WORD
7461 3160          DCA SAV5             / CLEAR SAV5
7462 5564          RETURN               / GO BACK TO THE PROGRAM
7463 7200  KCRA,   7200
7464 1560  KCRB,   1560
7465 7600  KTTY,   7600
                                        / THE PRINT TO TTY ROUTINE
7466 6163  TALK,   SKIP2                / SKIP ON CLEAR XMIT BUFFER
7467 5266          JMP .-1              / XMIT BUFFER NOT YET CLEAR
7470 6161          WRITE1               / WRITE THE AC TC THE UART
7471 3144          DCA TIME             / CLEAR THE AC AND STORE THE DATA
                                        / IN TIME FOR POSSIBLE RECOVERY
7472 5564          RETURN               / GO BACK TO THE PROGRAM
                                        / LISN IS THE ROUTINE TO GET A
                                        / CHARACTER FROM THE TTY KEYBOARD
                                        / OR READER.
7473 6172  LISN,   SKIP3                / RESET THE START BIT SENSE FLAG
7474 7000          NOP
7475 6166          SFLAG1               / SET THE READER RUN FLAG
7476 6172          SKIP3                / WAIT FOR THE FIRST START BIT
7477 5276          JMP .-1              / NOT YET
75C0 6167          CFLAG1               / CLEAR THE READER RUN FLAG
7501 6162  READ,   SKIP1                / WAIT FOR DATA READY FLAG
7502 5301          JMP .-1
7503 7200          CLA                  / CLEAR THE AC
7504 616C          READ1                / READ THE UART BUFFER AND ERROR
                                        / FLAGS, CLEAR THE DATA READY FLAG.
7505 0307          AND TTYM             / CLEAR OUT THE UNWANTED BITS
7506 5564          RETURN               / GO BACK TO THE PROGRAM
7507 0377  TTYM,   0377
                                        / THE INTERJEPT JR. BIN LOADER
                                        / USING THE PIE-UART INTERFACE
        7600       *7600
7600 4161  BIN,    CALL                 / INITIALIZE THE PIE-UART
7601 7445          INPIE
7602 3152          DCA STORE            / CLEAR THE CHECKSUM
7603 1350          TAD K177             / SET THE PC EQUAL TO 177
7604 3160          DCA SAV5
7605 1350          TAD K177             / SET THE AC EQUAL TO 200
7606 7001          IAC                  / SET THE AC EQUAL TO 200
7607 3151          DCA BACK             / STORE IN BACK
7610 7240          CLA CMA              / SET THE AC EQUAL TO 7777
7611 3154          DCA SAV1             / SET SAV1
7612 7240  BEGG,   CMA CLA              / SET AC TO 7777
7613 3150          DCA TEMP             / SET TEMP
7614 3147          DCA POINT            / CLEAR POINT
7615 4161  BEG,    CALL
7616 7473          LISN                 / GET THE FIRST TTY CHARACTER
7617 3146          DCA HOLD             / STORE THE CHAR IN HOLD FOR
                                        / SAFE KEEPING
7620 1146          TAD HOLD             / GET THE CHAR
7621 1351          TAD KRUB             / ADD -377
7622 7700          SMA CLA              / TEST; IS IT A RUBOUT?
7623 532C          JMP RUM              / YES; GO TO RUBOUT ROUTINE
7624 1146          TAD HOLD             / NO; GET THE CHAR
7625 1353          TAD KCH8             / ADD -200
7626 765C          SNA CLA              / TEST; WAS IT A LEADER-TRAILER
7627 5326          JMP LTC              / YES; GO TO THE LT ROUTINE
763C 1146          TAD HOLD             / GET THE CHARACTER
7631 3151          DCA BACK             / PLACE IT IN THE LAST CHARACTER HOLE
7632 1146          TAD HOLD             / GET THE CHARACTER
7633 1152          TAD STORE            / ADD TO THE CHECKSUM SUBTOTAL
7634 3152          DCA STORE            / PLACE IN CHECKSUM
7635 1146          TAD HOLD             / GET THE CHAR
7636 1352          TAD KFD              / ADD -277
7637 7700          SMA CLA              / TEST; IS IT A CHANGE FIELD CHAR
7640 5215          JMP BEG              / YES; IGNORE IT
7641 1146          TAD HOLD             / GET THE CHARACTER
7642 0354          AND KLONG            / MASK OUT CHANNEL 7
7643 764C  KLING,  SZA CLA              / TEST; IS IT AN ORGIN?
7644 5311          JMP PCL              / YES; GO TO LOAD THE PC
7645 1154          TAD SAV1             / NO; GET THE FIRST CHAR FLAG
7646 7640          SZA CLA              / TEST; HAVE WE GOTTEN A LT YET?
7647 5215          JMP BEG              / NO; IGNORE THE DATA
7650 2147          ISZ POINT            / YES; IS THIS A SECOND PC CHAR?
7651 7410          SKP                  / NO; GO ON
7652 5305          JMP PCL2             / YES; LOAD THE SECOND HALF OF THE PC
7653 2150          ISZ TEMP             / TEST; IS THIS A SECOND DATA CHAR
7654 5260          JMP DL2              / YES; GO TO SECOND DATA LOAD
7655 1146          TAD HOLD             / NO; GET THE CHARACTER
7656 3145          DCA SAVE             / STORE IT IN SAVE
7657 5214          JMP REGG+2           / GO BACK FOR THE SECOND PART
766C 1160  DL2,    TAD SAV5             / GET THE ADDRESS INTO THE AC
7661 1353          TAD KCH8             / ADD -177
7662 7700          SMA CLA              / TEST; IS PC > 177
7663 527C          JMP .+5              / YES; OK TO LOAD THE RAM
7664 116C          TAD SAV5             / NO; GET THE ADDRESS
7665 1243          TAD KLING            / ADD -140
7666 7700          SMA CLA              / TEST; IS PC < 14C
7667 5274          JMP .+5              / NO; BYPASS THE LOAD OF RAM
7670 1153          TAD SHIFT            / YES; GET THE FIRST HALF OF THE
                                        / LOAD CHARACTER
7671 7CC2          BSW                  / POSITION IT
7672 1156          TAD SAV3             / GET THE SECOND HALF
7673 3560          DCA I SAV5           / PLACE THE WORD IN THE RAM
7674 1145          TAD SAVE             / GET THE NEXT FIRST HALF
7675 3153          DCA SHIFT            / PLACE IT IN SHIFT
7676 1146          TAD HOLD             / GET THE NEXT SECOND HALF
7677 3156          DCA SAV3             / PLACE IT IN SAV3
7700 1000          TAD SAVPC            / GET THE ADDRESS
7701 316C          DCA SAV5             / PLACE IT IN THE LOAD POINTER
7702 2000          ISZ SAVPC            / INCREMENT THE PC
7703 5212          JMP BEGG             / GO GET THE NEXT CHARACTER
77C4 5212          JMP BEGG
7705 1146  PCL2,   TAD HOLD             / GET THE CHARACTER
7706 1000          TAD SAVPC            / PLACE IN THE LAST HALF OF PC
7707 3000          DCA SAVPC            / RESTORE
7710 5212          JMP REGG             / GET ANOTHER CHARACTER
7711 1354  PCL,    TAD KLONG            / SET THE AC TC 0100
7712 7040          CMA                  / SET THE AC TO 7677
7713 0146          AND HOLD             / GET THE CHARACTER MINUS CHANNEL 7
7714 7002          BSW                  / POSITION THE BITS
7715 3CC0          DCA SAVPC            / PLACE AS THE FIRST HALF OF PC
7716 7040          CMA                  / SET THE AC TC 7777
7717 5214          JMP REGG+2           / GO GET THE SECOND HALF OF THE PC
7720 4161  RUM,    CALL                 / GET ANOTHER CHARACTER FROM THE
                                        / UART
7721 7473          LISN
7722 1351          TAD KRUB             / ADD -377
7723 7700  KM63,   SMA CLA              / TEST; IS IT A RUBOUT?
7724 5215          JMP BEG              / YES; CONTINUE WITH THE LOADING
7725 5322          JMP RUM              / NO; IGNORE UNTIL A RUBOUT OCCURS
7726 3154  LTC,    DCA SAV1             / CLEAR THE LF FLAG
7727 1151          TAD BACK             / GET THE LAST CHAR
7730 1353          TAD KCH8             / ADD -200
7731 765C          SNA CLA              / TEST; IS IT A LT CHAR?
7732 5215          JMP BEG              / YES; CHARACTER RECEIVED WAS NOT THE
                                        / FIRST LT
7733 1153          TAD SHIFT            / NO; THE CHARACTER RECEIVED WAS
                                        / A FIRST LT CHARACTER AND
                                        / THEREFORE THIS SHOULD END THE LOAD
                                        / ROUTINE. GET THE FIRST HALF OF THE
                                        / LAST FIRST CHARACTER
7734 7002          BSW                  / POSITION IT
7735 1156          TAD SAV3             / ADD THE LAST HALF OF THE LAST CHARACTER
7736 7041          CIA                  / NEGATE THE VALUE
7737 1152          TAD STORE            / ADD THE CHECKSUM
7740 7041          CIA                  / NEGATE THE RESULT
7741 1153          TAD SHIFT            / NOW ADJUST FOR THE SECOND
                                        / TIME WITH THE CHECKSUM
7742 1156          TAD SAV3
7743 3140          DCA SAVAC            / STORE IN THE AC
7744 1355          TAD K140             / GET THE SAVAC ADDRESS
7745 3000          DCA SAVPC            / PLACE IN THE PC
7746 5747          JMP I  .+1
7747 6407          HALT                 / GO TO THE HALT ROUTINE
7750 0177  K177,   0177
7751 7401  KRUB,   7401
7752 75CC  KFD,    7500
7753 760C  KCH8,   760C
7754 0100  KLONG,  0100
7755 C14C  K14C,   C14C
                                        / THIS IS THE LEADER-TRAILER
                                        / ROUTINE FOR THE MEMORY DUMP
                                        / PROGRAM.  PLEASE NOTE THAT
                                        / IT LINKS THROUGH TWO PAGES.
        7757       *7757
7757 1323  TWTY,   TAD KM63             / SET THE AC TO -63
7760 3145          DCA SAVE             / PLACE IT IN SAVE
7761 1353          TAD KCH8             / GET THE CHARACTER YOU WANT TC
                                        / PRINT
7762 4161          CALL                 / PRINT IT OUT CN THE TTY
7763 7466          TALK
7764 5765          JMP I .+1
7765 6173          LINK                 / GC THROUGH THE LINK TO THE
                                        / REST OF THE SUBROUTINE
        6173       *6173
6173 2145  LINK,   ISZ SAVE             / COUNT OUT THE NUMBER OF LOOPS
6174 5776          JMP I .+2            / NOT DONE YET.  GC DO ANOTHER
6175 5564          RETURN               / COMPLETE. GO BACK TO PROGRAM
6176 7761          TWTY+2               / RETURN TO THIS POINT
                                        / THIS IS THE MEMORY DUMP
                                        / PROGRAM WHICH CAN BE IMPLIMENTED
                                        / BY STARTING THE INTERCEPT
                                        / JUNIOR RUNNING IN THE NORMAL
                                        / MODE WITH A STARTING ADDRESS
                                        / 7510.  TO SET THE ADDRESSES
                                        / OF THE RANGE OF MEMORY YOU
                                        / WANT PUNCHED OUT IN BIN FORMAT
                                        / PLACE THE ADDRESS OF THE FIRST
                                        / LOCATION YOU WANT PUNCHED IN
                                        / LOCATION 176 AND THE LAST
                                        / LOCATIONS ADDRESS IN 177.
                                        / THE PROGRAM WILL THEN PUNCH OUT ALL
                                        / MEMORY LOCATIONS IN BETWEEN WITH
                                        / LEADER-TRAILER AND A CHECKSUM.
        7510       *7510
7510 7340  DUMP,   CLA CLL CMA          / SET THE AC TC 7777
7511 6402          6402                 / DISABLE THE CP REQUEST TIMER
7512 4161          CALL                 / INITIALIZE THE PIE;UART
7513 7445          INPIE
7514 7344          CLA CLL CMA RAL      / SET THE AC EQUAL TO 7776
7515 3151          DCA BACK             / PLACE THIS IN BACK
7516 4161          CALL                 / PRINT OUT 63 LEADER-TRAILER CHARACTERS
7517 7757          TWTY
7520 1176          TAD 176              / PRINT OUT THE ORGIN
7521 7002          BSW                  / POSITION FOR THE FIRST BITS
7522 0371          AND HALF             / MASK OUT THE BITS
7523 1372          TAD ORGIN            / ADD THE CHANNEL 7 PUNCH
7524 4161          CALL                 / PUNCH IT OUT ON THE TTY
7525 7573          PUNCH
7526 1176          TAD 176              / GET THE STARTING ADDRESS AGAIN
7527 0371          AND HALF             / MASK OUT THE LAST HALF
7530 4161          CALL                 / PUNCH IT OUT ON THE TTY
7531 7573          PUNCH
7532 1576  DATAL,  TAD I 176            / PRINT OUT THE DATA
7533 2151          ISZ BACK             / TEST; IS THIS FIRST OR SECOND
7534 7002          BSW                  / FIRST; POSITION THE BITS
7535 0371          AND HALF             / MASK OUT THE BITS
7536 4161          CALL                 / PUNCH OUT ON THE TTY
7537 7573          PUNCH
7540 1151          TAD BACK             / GET THE FIRST FLAG
7541 7640          SZA CLA              / TEST; IS THIS THE FIRST?
7542 5332          JMP DATAL            / YES; GO BACK AND DO THE LAST HALF
7543 7344          CLA CLL CMA RAL      / NO; RESET BACK TO 7776
7544 3151          DCA BACK
7545 1176          TAD 176              / GET THE ADDRESS
7546 7041          CIA                  / NEGATE IT
7547 1177          TAD 177              / ADD THE LAST ADDRESS
7550 765C          SNA CLA              / TEST; ARE THEY THE SAME?
7551 5354          JMP .+3              / YES; GO TO END OF DUMP
7552 2176          ISZ 176              / NO; INCREMENT THE ADDRESS
7553 5332          JMP DATAL            / GO GET THE NEXT WORD.  NOTE
                                        / THAT IF THE ADDRESS WAS 7777 THEN THE
                                        / DUMP PROGRAM WILL DEFAULT TO END
7554 1160  CHSUM,  TAD SAV5             / GET THE CHECKSUM AND PUNCH IT OUT
7555 2151          ISZ BACK             / TEST; IS THIS FIRST OR LAST
7556 7002          BSW                  / FIRST; POSITION THE BITS
7557 0371          AND HALF             / MASK OUT THE BITS
7560 4161          CALL                 / PUNCH OUT THE CHECKSUM
7561 7466          TALK
7562 1151          TAD BACK             / GET THE FIRST FLAG
7563 7640          SZA CLA              / TEST IF FIRST OF SECOND HALF
7564 5354          JMP CHSUM            / FIRST HALF, GO BACK FOR SECOND
7565 4161          CALL                 / PUNCH OUT LEADER-TRAILER
7566 7757          TWTY
7567 6402          6402                 / RESTORE THE CP REQUEST TIMER
7570 7402          HLT                  / END THE PROGRAM
7571 0077  HALF,   0077
7572 0100  ORGIN,  0100
```

```
1497
1498   7573  4161   PUNCH,  CALL                / OUTPUT THE AC TO THE TTY
1499   7574  7466           TALK
1500   7575  5776           JMP I .+1           / LINK TO THE REST OF THE SUBROUTINE
1501   7576  6374           LINKER
1502
1503
1504         6374   *6374
1505
1506   6374  1144   LINKER, TAD TIME            / RECOVER THE CHARACTER
1507   6375  1160           TAD SAV5            / ADD TO THE CHECKSUM
1508   6376  3160           DCA SAV5            / UPDATE THE NEW CHECKSUM
1509   6377  5564           RETURN              / GO BACK TO THE PROGRAM
1510
1511

END OF PASS 2

   0 ERRORS DETECTED
```

SYMBOL TABLE

AAA	7246	AAB	7247	AAC	7250	AAD	7251	AAE	7252	AAF	7253	AAG	7254	AAND	6701
AC	0166	ACCA	6671	ADJT	6617	ACJ	6262	AINC	6661	AIDT	7027	AISZ	6666	AJMP	6676
AJMS	6673	ADPR1	7C71	ADPR2	7153	ADPR3	7204	ATAD	6664	BACK	0151	BASE	6037	BEGG	7612
BEG	7615	BIN	7600	BLK	6566	BOB	7220	BOOB2	6726	BOOB	6725	BSET10	7271	BSET11	7145
BSET4	7255	BSET5	7257	BSET6	7261	BSET7	7263	BSET8	7265	BSET9	7267	BUG	6547	CALLX	0161
CALLY	6040	CALL	4161	CFLAG1	6167	CFLAG3	6177	CHSUM	7554	CLKPD	6110	CCB	7041	CPMODE	7766
CUS	7052	DATAL	7532	DCI	6557	DECPC	6403	DEP	6524	DIGO	6267	DIG1	6301	DIG2	6312
DIG3	6323	EL2	7660	DUMP	7510	EJMP	7363	EJMS	7356	EXEC	7366	EXIT	6051	FLASH	6777
GAR	6202	GCCN	6624	GOTO	6130	GUE	7167	GUM	7104	GUT	7354	HALF	7571	HALT	6407
HEX	6425	HOLD	0146	HOT	7425	INAD	7330	INCAC	6400	INDB	7343	INIT	6007	INPIE	7445
JAL0	7143	JA4	7121	JA5	7124	JA6	7127	JA7	7132	JA8	7135	JA9	7140	JB10	6167
JB4	6145	JB5	6150	JB6	6153	JB7	6156	JB8	6161	JB9	6164	JC4	7235	JC5	7240
JC7	7243	JMPI	6034	KAT	7327	KCALLY	6035	KCH8	7753	KCRA	7463	KCRB	7464	KFD	7752
KIT	7325	KJMP	7323	KLING	7643	KLCNG	7754	KM10	6370	KM11	6371	KM14	6364	KM4	6366
KM63	7723	KM7	6367	KMB	6365	KRETY	6036	KRUB	7751	KTTY	7465	K0007	6523	K140	7755
K177	7750	K2000	6670	K3000	6631	K4000	6675	K5000	6700	LINKER	6374	LINK	6173	LISN	7473
LCAD	6333	LOT	7355	LTC	7726	MASK0	6360	MASK1	6361	MASK2	6362	MASK3	6363	MASK	6607
MBST	7273	MICRO	6633	MK	7324	MCUT	6077	MRPA	6703	MSK1	6372	MSK2	6373	NEXT	6736
NOT	7352	ORGIN	7572	OUT	6063	PCI	6563	PCL2	7705	PCL	7711	PLIM	7326	POINT	0147
PCA	6550	PUD	7353	PUNCH	7573	PUP	6752	RCRA	6164	READ1	6160	READ2	6170	READ	7501
REFSH	6236	RESET	6414	RETURN	5564	RETX	0164	RETY	6051	RET	7427	RUM	7720	RUN	6411
SAVAC	0140	SAVE	0145	SAVFL	0142	SAVMQ	0141	SAVPC	0000	SAV1	0154	SAV2	0155	SAV3	0156
SAV4	0157	SAV5	0160	SETPC	6600	SFLAG1	6166	SFLAG3	6176	SHIFT	0153	SIN	7301	SKIP1	6162
SKIP2	6163	SKIP3	6172	SKIP4	6173	SNERD	6533	SNOT	7050	SCB	7053	SOC	7051	SOT	7120
SOUP	7025	STACK	0165	START	6000	STATUS	0143	STORE	0152	SWDB	6200	TABLE	6263	TAB	6543
TALK	7466	TAL	7426	TCNT	6235	TEMP	0150	THRU	6246	TIME	0144	TKA	7023	TKB	7024
TK1	6233	TK2	6234	TOK	6172	TCZ	6744	TTYM	7507	TUG1	6742	TUG2	6743	TUG	6741
TWTY	7757	UDCS	6117	LG	6546	WCRA	6165	WCRE	6175	WRITE1	6161	WRITE2	6171	WWR	6174
XEC	6644	ZOK	7C26	ZOL2	7166	ZOL3	7217	ZONK	6727	ZOT	6712				

CHAPTER 9

INTERCEPT JR. AUDIO CARD

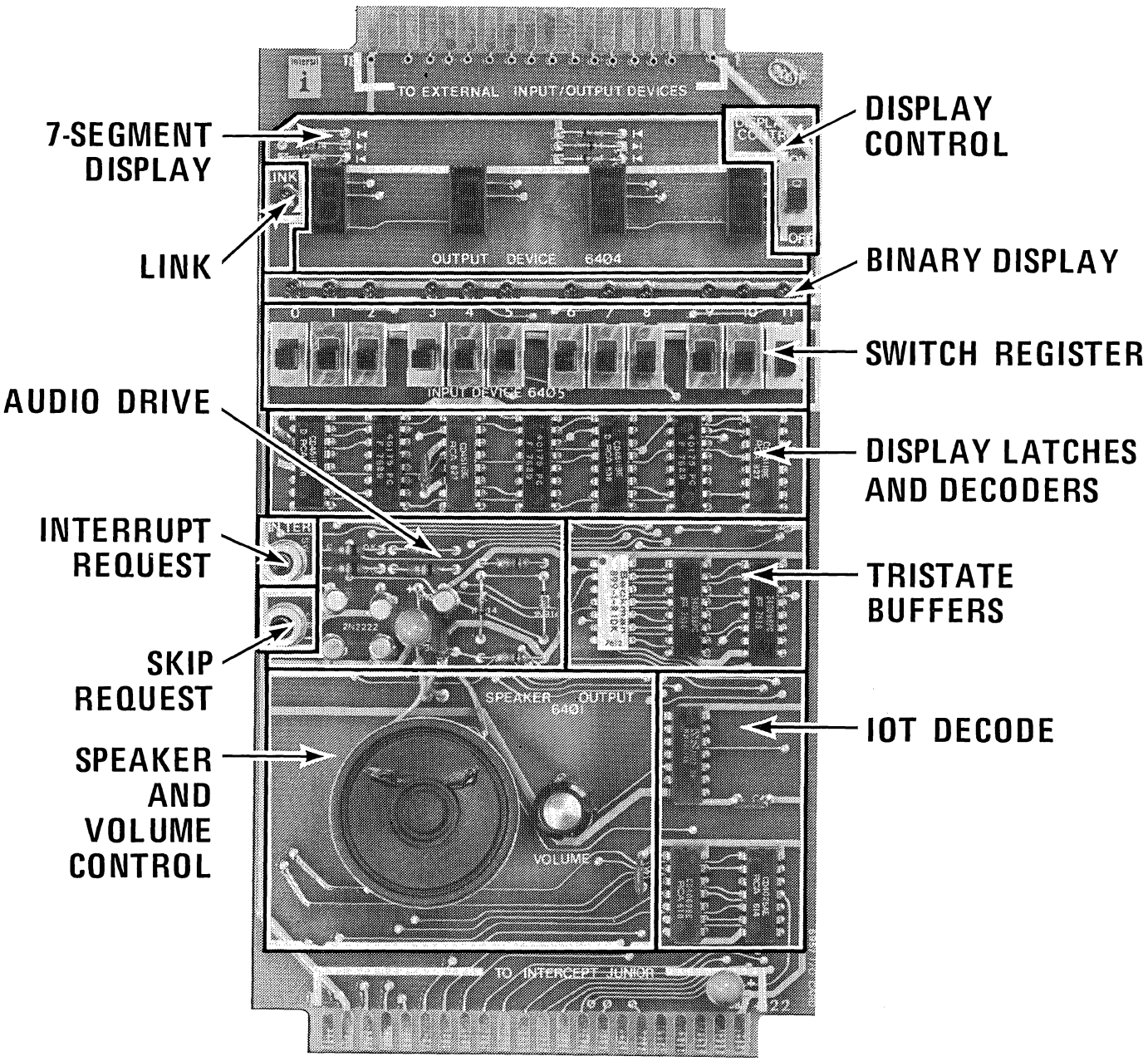


FIGURE 9-1

INTRODUCTION

The INTERCEPT JR. AUDIO MODULE, 6957-AUD/VIS, pictured in Figure 9-1, is used in microprocessor tutorial courses developed by INTERSIL INC.

The user can "click" the speaker or produce tones by controlling the rate at which the speaker clicks; the user can read a switch register and load data to an LED display register in either binary or in both binary and octal.

DISCUSSION

The AUDIO card makes use of the three unused IOT instruction codes 64 X 1, 64 X 4 and 64 X 5 brought out to connector pins Y, C and 15 of the INTERCEPT JR. module.

The card should be plugged in with the LED display on top and the speaker below using the card edge connector designated "to INTERCEPT JUNIOR".

The switch register is connected to the DX bus via two 340098 three-state hex buffers. The LED binary register is driven by three 74C175 quad D-latches with their inputs connected to the DX bus. The true outputs of the latches drive three 4511 BCD to 7 segment decoder drivers. The D input of each of the 4511's is grounded so that the seven segment display can only display in octal. The display can be blanked by pulling the blanking inputs on the 4511's low via the Display Control Switch S_{12}.

All the switch outputs are pulled up to V_{CC} via the 10K resistor pack.

IOT 6401 along with DEVSEL and XTC drives a 4025 three input NOR so that during IOTA·$\overline{\text{DEVSEL}}$·$\overline{\text{XTC}}$ the 74C74 flip-flop is clocked by the execution of this instruction. The flip-flop toggles every time it is clocked as its $\overline{Q}$ output is connected back to the D input. This turns the transistors in the push-pull driver alternately ON or OFF, charging and discharging the 68 microfarad capacitor through the speaker voice coil and producing an audible click.

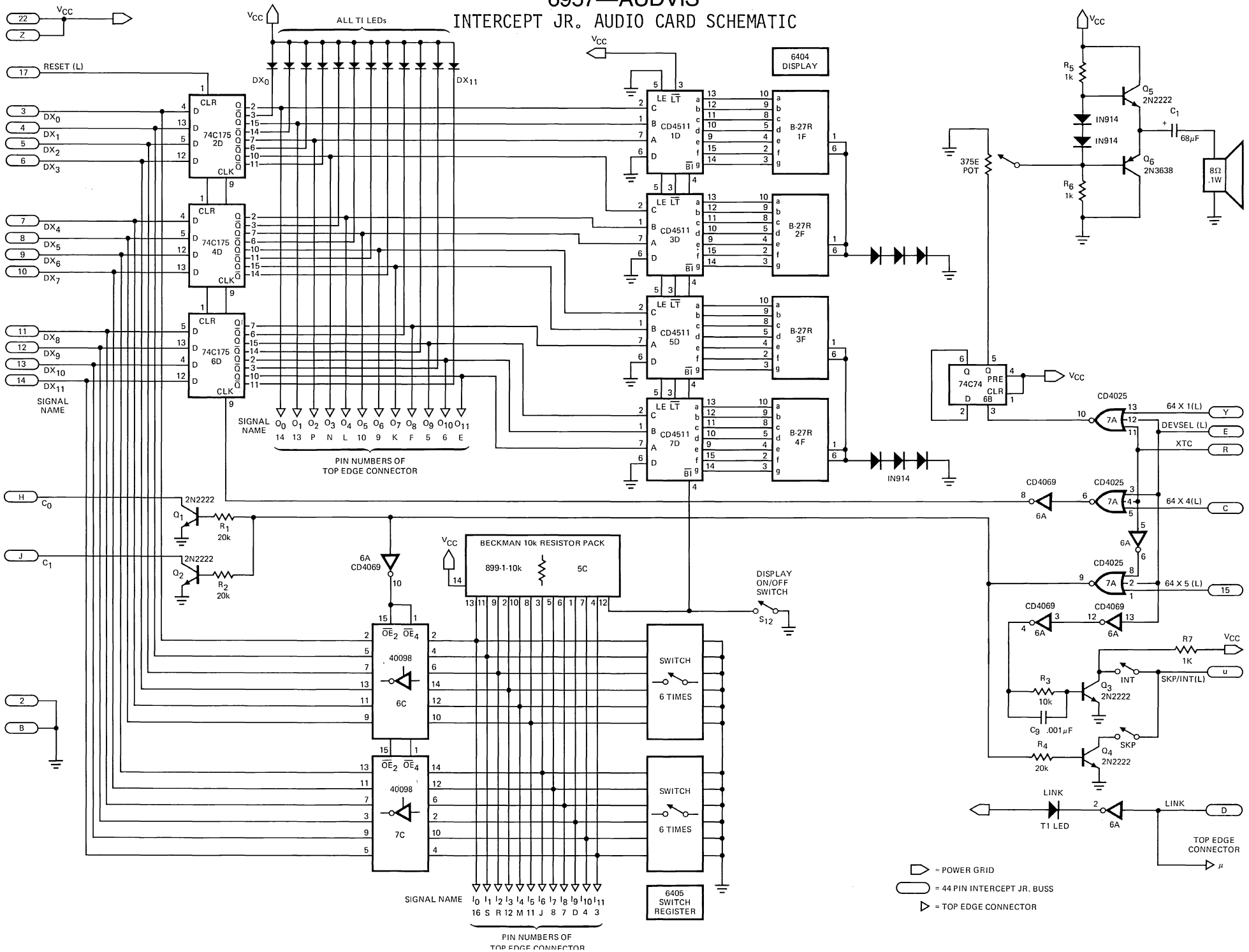
IOT 6404 is also an output instruction and thus is gated with DEVSEL and XTC to produce a load pulse (inverted by a 4069) to the three quad D-latches connected to the DX bus. The latches will thus store the contents of the AC which are placed on the bus by the IM6100 during IOTA·$\overline{\text{DEVSEL}}$·$\overline{\text{XTC}}$.

IOT 6405 is an input instruction and is decoded along with DEVSEL and XTC to produce a strobe pulse at IOTA·$\overline{\text{DEVSEL}}$·XTC time. This pulse is inverted by a 4069 and enables the tristate buffers onto the DX bus and also turns ON the two 2N2222 transistors driving the C_0 and C_1 lines. The IM6100 thus reads the DX bus during IOTA·$\overline{\text{DEVSEL}}$·XTC and loads the data into the accumulator.

The INTREQ and SKP lines to the IM6100 are multiplexed onto the same line. The data read strobe generated by an IOT 6405 enables the SKP line so that depression of the SKP switch will drive the SKP line low. The INTREQ line is always enabled except during $\overline{\text{DEVSEL}}$ time. Actually, the SKP line is sampled only during $\overline{\text{DEVSEL}}$·XTC, but for simplicity, interrupt requests are disabled even during $\overline{\text{DEVSEL}}$·$\overline{\text{XTC}}$. In any case, the INTREQ line is sampled only during the last cycle of an instruction execution during the first major state time.

The LINK bit drives an LED diode directly via a 4069.

6957—AUDVIS
INTERCEPT JR. AUDIO CARD SCHEMATIC



APPENDIX A

INTERCEPT JR. PROGRAMMING FUNDAMENTALS

NUMBER SYSTEMS

INTERCEPT JR., as most digital computers, uses the binary system. Representation of binary numbers by positional notation is analogous to the common representation of decimal numbers by assigning ten different "weights" to each position. Any number of n digits may be written as the string of digits.

$$C_{n-1}\ C_{n-2}\cdots C_1\ C_0$$

where C's can range from 0 to 9.

This actually stands for

C_{n-1} followed by (n-1) zeros +

C_{n-2} followed by (n-2) zeros +

C_{n-3} followed by (n-3) zeros +

$\vdots$

C_1 followed by 1 zero +

C_0

or $C_{n-1}\ (10)^{n-1} + C_{n-2}\ (10)^{n-2} + \ldots C_1\ (10)^1 + C_0\ (10)^0$

For example, 1234 is 1000 + 200 + 30 + 4 or $1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4$.

Similarly, in the binary system, any number may be represented by a string of coefficients

$$B_{n-1}\ B_{n-2}\cdots B_0\ B_1$$

which stands for

$$B_{n-1}\ (2)^{n-1} + B_{n-2}\ (2)^{n-2} + B_1\ (2)^1 + B_0\ (2)^0$$

where the B's may be 0 or 1.

The "radix", or base of the binary system is 2, whereas it is 0 in the case of the decimal system.

The reason that the binary system is universally used in digital computers is that it is very convenient and easy to provide for two states in digital circuits and this makes a binary representation of digital system states very practical.

Other physical systems may be easier to describe in a number system with a different number of states. To illustrate, consider this puzzle:

> Given a scale balance, how many different weights would be needed to balance any object that could weigh a whole (integer) number of pounds up to 1000 pounds?

One way of looking at this puzzle is by imagining that the object is placed on one pan of the balance and the weights are added, or taken off, the other side until the pan balances.

Since a weight could either be on the pan, or not on the pan, we have two possible states for each weight--on or off the pan. By now, it may be intuitively apparent that we could take a group of weights, in ascending powers of two, and using them or not using them on the pan, we could balance any weight up to the sum total of all weights.

It takes ten binary digits, or "bits" for short, to represent any number from 0 to 1023. Our problem is solved by having ten weights, weighing 1, 2, 4, 8, 16, 32, 64, 128, 256, 512 pounds each.

We have gone from 1000 to 10 by making the binary connection. Can we do better? Actually, we can, by going just a little further along the same train of thought.

What if we were allowed to place the weight on the opposite pan, along side the object? This adds a third possible state to each weight. Now it can either be on the "normal" side, on the "object" side, or not on the balance at all. We have a three-valued, or a ternary system.

By putting the object on the "object" side, we are effectively giving it a negative weight, as it acts to force the "normal" side of the pan upwards. So each physical weight has three mathematical "weights" assigned to it, 0 +1, -1 according to where it is--off, on or opposite side of the pan.

It can be proved, but it should also be intuitively apparent that now we need weights in ascending powers of three. The weights required are 1, 3, 9, 27, 81, 243, 729 so we have bettered our previous score by three.

Digital circuits are composed of vast quantities of two-state or bi-stable devices known as flip-flops.

In theory, binary numbers may be used to describe the condition of these flip-flops.

A system with 12 flip-flops could be represented by a 12 bit number, for example 1 0 1 1 0 0 1 1 1 0 0 1, where each bit represents the set or reset state of a particular flip-flop. Binary numbers are unwieldy to handle because of the long strings involved, so often a simplification is introduced.

Consider the numbers 0 through 15 written in their binary equivalent.

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

Observe that in the "units", or 2^0 position, the state changes, or "toggles" most often, for example every time the number increments. In the next or 2^1 position, the bits toggle every two increments, and in the 2^2 position, every four times, etc.

Or, looking at this another way, the one bit groups 0 and 1 alternate every time, the two bit groups 00, 01, 10, 11 recur every fourth time, the three bit groups

000)
001)
010)
011)
100)
101)
110)
111)

recur every eight times, and so on.

Thus, to shorten binary numbers, we could encode these groups. The tradeoff is between the length of the number string, and the number of symbols required.

We could encode the two bit groups with four symbols.

00	0
01	1
10	2
11	3

Then we could represent the same number we had before as shown:

```
10 11 00 11 10 01
 2  3  0  3  2  1
```

The string has half the number of digits, but it took twice the number of symbols.

We could go further, and take three bit groups. Let us encode the group as follows:

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

The previous example is split up into three bit groups as shown:

```
101 100 111 001
 5   4   7   1
```

We now have one-third the number of digits in the string, but instead of only three times the number of symbols, we have four times the number of symbols. This is because the number of symbols is doubled every time we increase the group size by one more.

Proceeding further, using four bit groups. We will run out of numerals, so we will use some alphabetic characters to help encode the group.

0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

Our 12 bit number may now be represented by three of the above codes:

	1011	0011	1001
or	B	3	9

So, we have doubled the number of symbols to sixteen but reduced the length of our string only by one, from four to three. The code itself has also become a little unwieldy because the number of different symbols.

As a matter of fact, this representation by four bit groups is known as the hexadecimal system (base 16 system) and is widely used.

The system of representation with three bit groups encoded with the eight symbols 0 through 7 is known as the octal number system and is also in wide use.

We shall adopt the octal numbering system for INTERCEPT JR.

It should be evident by now that the choice is based purely on convenience and consistency with the available literature as almost all digital computers are fundamentally binary machines.

At this point, it is instructive to turn your machine ON. Press the CNTRL key and the MEM key and then keep pressing the MEM key. The address display will increment in an octal progression. By watching the addresses increment, the user can become familiar with the octal system.

To recapitulate, conversion from binary to octal is done by taking groups of three bits, starting from the least significant bit, filling in a zero or zeros to the most significant group, if necessary, and writing down the octal equivalent for each group.

Conversion from octal to binary is done by directly writing down three bits from each octal number.

INTERCEPT JR. uses two's complement arithmetic in its processing logic.

The processor performs binary addition between two operands but binary subtraction is best done adding the "negative" of one operand to the other. This requires an extra symbol to indicate the sign of the number. To avoid this, a form of representation known as two's complement has been devised to represent negative numbers.

To illustrate the concept involved, let us look at a couple of analogies in the decimal system. In the decimal system, we could use the "ten's complement" to represent negative decimal numbers. Consider a ruler marked off in centimeters. If zero is your reference point, then "1" would be the same as +1. To measure "-1", you would have to turn the ruler around. Or, you could slide it so that the number 10 was opposite your reference point. Now, the point "-1" is marked by the number 9. That is, 9 is the 10's complement of 1. Similarly, 8 is the ten's complement of 2, 7 is the ten's complement of 3, etc., these numbers representing -1, -2, -3, etc.

To subtract one number from another, we add their ten's complement together and ignore the carry.

Thus 8 - 3 is given by adding 8 and 7 to get 15 and ignoring the carry to finally give 5.

The one's complement, by definition is obtained by subtracting each digit from one. In the binary system, this is particularly easy. All one has to do is to invert the bits.

	00000101
One's complement	11111010
Add 1 to get 2's complement	11111011

By taking the two's complement again, we get the negative of a negative number, so we should get the original number back again.

	11111011
One's complement	00000100
Add 1 to get 2's complement	00000101

Thus 8 - 3 in binary is 1000 - 0011, or taking two's complement 1000 + (1100 + 1) = 1000 + 1101 = 10101 or 1010 = 5 neglecting the high order carry.

ARITHMETIC PROGRAMMING EXAMPLE #1

ADDRESS	MEMORY	SYMBOL
0020	7200	CLA
0021	1026	TAD 0026
0022	1027	TAD 0027
0023	3025	DCA 0025
0024	7402	HLT

We shall enter this program with INTERCEPT JR. via the keyboard to practice binary arithmetic. At this point, it is sufficient to know that CLA stands for clear accumulator, TAD for binary ADD, DCA for deposit into memory and HLT for halt.

The octal numbers on the left are successively numbered memory locations or "addresses" and the numbers on the right are the octal representations for the binary data that will be stored in these memory locations.

Each location can store four octal digits, that is, twelve binary digits, or bits. Each location may be thought of as a row of twelve flip-flops that are set or reset according to the data to be stored.

To enter this program, turn on the machine and perform the following sequence of key depressions:

CONTRL SETPC 0 0 2 0

This enters the starting address.

CONTRL MEM 7 2 0 0

This enters the first instructions.

MEM This increments the address.

CNTRL MEM 1 0 2 6

This enters the second instruction in location 0021.

MEM Increments address.

Finally, after HLT is entered, press MEM twice to step the address to 0026.

Now, enter an octal number, for example 7, into this address, step MEM again and enter a second octal number, for example 10. These are the operands in location 0026 and 0027. The program will add them and place the result in lcoation 0025. In this case, 0017 will be seen in this location.

Now, to run the program, we have to get back to the beginning, so press

CONTRL SETPC 0 0 2 0

The display will show the address and the instruction.

Press

CONTRL RUN

The program will be executed, and the processor will halt, showing the result in location 0025. Note that if the sum of the two numbers is greater than 7777 in octal, a carry out or overflow will occur from the most significant bit position, setting the LINK bit in the processor.

Practice different addition problems on paper, in binary, then in octal and check them on INTERCEPT JR.

At this point, think of all the numbers as unsigned positive integers. Now, think of the numbers as signed two's complement numbers.

Write down two numbers, and subtract one from the other using this program to add one operand to the two's complement of the other operand.

EXAMPLE:

000	111	000	111	or	0707
111	000	111	000	or	7070
added together					
111	111	111	111	or	7777

If 7070 is considered as a negative number, then, by taking the two's complement, we get:

000	111	000	111	+	1 =
000	111	001	000	or	0710

That is 0707 - 0710 = 7777. This can be seen to be true because 0710 is just one greater than 0707 and 7777 is obviously -1 (two's complement = 0000 + 1).

Since data entry and readout in INTERCEPT JR. is in octal, it may be convenient to work in eight's complement notation. This is a direct extension of the previously described technique:

EXAMPLE: Subtract 3456_8 from 7142_8

A)	7's complement of 3456_8	4321_8
B)	add 1 to get 2's complement	4322_8
C)	add to 7142	7142_8
		13464_8
D)	discard high order carry to give answer =	3464_8

ARITHMETIC PROGRAMMING EXAMPLE #2

We shall now explore in greater detail the advantages of two's complement arithmetic.

As explained previously, to form the two's complement of a binary number, the one's complement is taken and one is added to it.

In general, this works with any radix (base). That is, the complement with respect to the largest single digit integer in the system is taken, and one is added to it to give the radix complement.

In particular, the binary system lets us implement the one's complement in a simple way. You simply invert the bits. This is very easily done in almost all modern digital logic families because inversion is a function basic to them all.

In many circuits, the true and inverted levels are available at the same time (flip-flops) and one just has to select the desired level.

The addition of 1 is also easy to do because the capability to add must be present anyway.

Thus 2's complement conversion, in conjunction with standard adders, allows the processor to do signed arithmetic. Multiplication and division are done by programming suitable algorithms, that is, computational sequences. We shall study these techniques later in this text.

The following program computes the two's complement of a binary number. Bear in mind that the number is entered in octal, and the two's complement is also displayed in octal.

```
ADDRESS     MEMORY      SYMBOLIC

 0020        7200      CLA / Clear accumulator
 0021        1026      TAD 0026 / Read data in 26
 0022        7040      CMA / Complement accumulator
 0023        1027      TAD 0027 / Add one
 0024        3026      DCA 0026 / Deposit into 26
 0025        7402      HLT / Halt
 0026                      / Result and user entered data
 0027        0001      0001 / DATA = 1
```

Enter this program exactly as explained in example one. Notice that this program uses data supplied by the user as well as data contained in the program itself

What if we would like to use both the program examples together?

If you look at the memory addresses used by the two programs, they look very similar. In fact, we just wrote over the first example and effectively lost it.

Let us assume we want to keep both example 1 and example 2 in memory. Since we wrote over example 1, let us choose to relocate it. Go back to the listing for that example and take a look at the symbolic code.

The octal numbers following the mnemonics are memory addresses that are referenced by the instructions.

All we have to do is to alter the memory references according to where the program is going to be relocated and make sure that the data is entered in the proper addresses where the program expects to find it.

Let us move the program up to twenty locations. All addresses must have twenty added to them and the three memory reference instructions must also have twenty added to them.

```
0040    7200
0041    1046
0042    7040
0043    1047
0044    3046
0045    7402
0046
0047    0001
```

Enter this as explained before. Naturally, the data must be entered into 46 and the result will also be in 46. Run a few examples and check out the program

Now we can do signed arithmetic by using one program for addition and the other for calculating two's complement. The only problem is, each time we have to enter data, execute, read data and then repeat the process for the other program.

ARITHMETIC PROGRAMMING EXAMPLE #3

This brings us to the concept of linking programs together and passing parameters between them.

For example, we can link the two programs into a single program that subtracts one number from another and displays the result in two's complement notation.

One program must give the other program the number to be converted and receive the two's complement result from it so it can finish the addition, for example subtraction of the original number, and display the result.

We pass parameters by storing data where both programs can reference it. We pass control by using unconditional branch or jump instructions to change the flow of the program. A jump instruction specifies the location from which the next instruction is to be fetched.

```
ADDRESS      MEMORY      SYMBOLIC

 0020         7200        CLA
 0021         1026        TAD 0026
 0022         7040        CMA
 0023         1027        TAD 0027
 0024         3046        DCA 0046
 0025         5040        JMP 0040
 0026                     / DATA 1
 0027         0001        0001

 0040         7200        CLA
 0041         1046        TAD 0046
 0042         1047        TAD 0047
 0043         3045        DCA 0045
 0044         7402        HLT
 0045                     / RESULT DISPLAYED : DATA 2 - DATA 1
 0046                     / DATA STORED BY FIRST PART OF PROGRAM
 0047                     / DATA 2
```

Note that location 24 is changed to store the two's complement of DATA 1 in location 46 instead of 26. Location 25 contains a JUMP to 0040 instead of a HLT. This causes the computer to fetch the next instruction from location 40 and, thus, execute the second segment of the program which finally halts showing the result of the subtraction in location 45.

Note that we could have relocated the data in 27 elsewhere and filled the space from 25 to 40 with NOP instruction, No Operation. This would have let the computer ripple down to the second segment of the program but would have been wasteful of memory space and not permitted the introduction of the JMP instruction.

A further simplification would have been to use the instruction CMA IAC or 7041 in location 0022. This would have eliminated the TAD instruction in 0023 and the data stored in 0027. CMA IAC

complements the accumulator, then increments it in the same memory cycle. This is an example of the use of combinations of micro-instructions. When using such combinations, the "logical sequence" of execution of the microinstructions must be carefully studied. In this example, for instance, CMA must be performed before the IAC. Refer to the IM6100 brochure for details on logical sequences.

Additional Reference: "Introduction to Programming", Digital Equipment Corporation Software Distribution Centers - 146 Main Street, Maynard, MA 01754 or 1400 Terrabella Road, Mountain View, CA 94040

ADDITION AND MULTIPLICATION TABLES

Addition

Multiplication

Binary Scale

0 + 0 = 0	0 X 0 = 0
0 + 1 = 1 + 0 = 1	0 X 1 = 1 X 0 = 0
1 + 1 = 10	1 X 1 = 1

Octal Scale

0	01	02	03	04	05	06	07
1	02	03	04	05	06	07	10
2	03	04	05	06	07	10	11
3	04	05	06	07	10	11	12
4	05	06	07	10	11	12	13
5	06	07	10	11	12	13	14
6	07	10	11	12	13	14	15
7	10	11	12	13	14	15	16

1	02	03	04	05	06	07
2	04	06	10	12	14	16
3	06	11	14	17	22	25
4	10	14	20	24	30	34
5	12	17	24	31	36	43
6	14	22	30	36	44	52
7	16	25	34	43	52	61

APPENDIX B
INTRODUCTION TO LOGIC

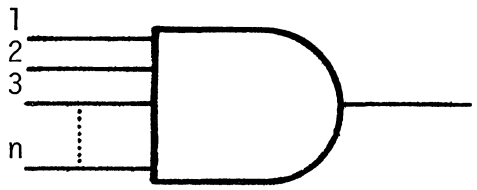
INTRODUCTION

This appendix briefly reviews truth tables as applied to simple logic elements, both combinatorial and sequential. Timing diagrams and state diagrams are illustrated using flip-flops as examples.

TRUTH TABLES

AND FUNCTION

Symbol for AND gate

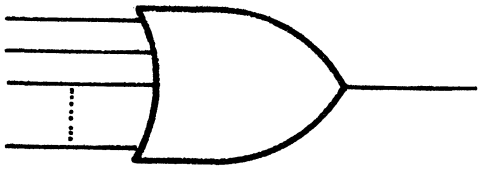


Output is true only if all inputs are true, that is, input 1 AND input 2 AND...AND input N

Input 1	Input 2	.....	Input N	Output
0	1	.....	1	0
0	0	.....	1	0
0	0	.....	0	0
1	0	.....	1	0
1	0	.....	0	0
		⋮		
1	1	all 1's	1	1

This table shows a conventional positive logic AND gate, with 1 representing logic high or true, and 0 representing logic low or false. Thus, only <u>one</u> combination of the inputs gives a <u>high</u> output.

OR FUNCTION SYMBOL FOR OR GATE



Output is true if at least one of the inputs is true, for example Input 1 OR Input 2 OR...Input N OR any combination of true inputs yields a true output.

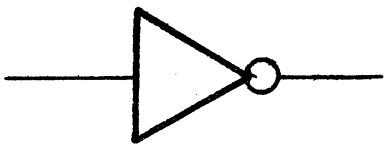
Input 1	Input 2	.....	Input N	Output
0	0	all 0's	0	0
0	1	.....	0	1
1	1	.....	0	1
0	0	.....	1	1

Here, only one of the 2^N possible input combinations namely all 0's will yield a false or low output.

NOT FUNCTION

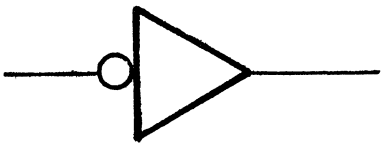
Symbol for inverter

A)



OR

B)



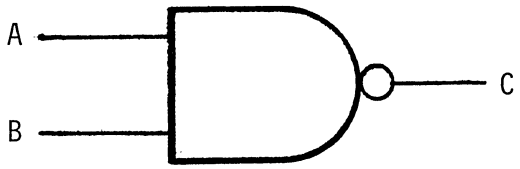
Output is logical inversion of input.

Input	Output
1	0
0	1

The position of the "bubble" tells you what the active level of the input is expected to be by the designer. Quite often, it is drawn as in A above.

NAND FUNCTION

Symbol for NAND gate

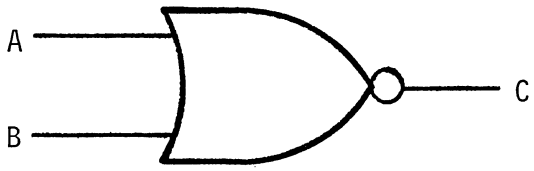


A	B	C
1	1	0
0	1	1
1	0	1
0	0	1

This is the same as an AND gate followed by an inverter.

NOR FUNCTION

Symbol for NOR gate

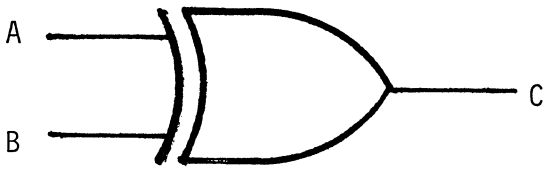


A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

This is the same as an OR gate followed by an inverter.

EXCLUSIVE-OR FUNCTION

Symbol for EX-OR gate

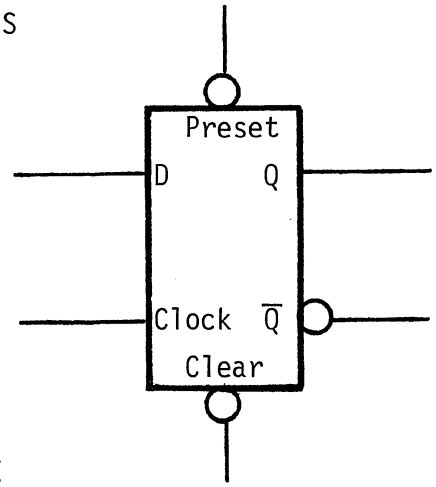


A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

Output is true if Input A OR B but not BOTH are true. Note that this gate can be used to detect the fact that the inputs are identical. Thus, it is used quite often in digital comparators.

D-TYPE FLIP-FLOPS

Symbol



TRUTH TABLE

Input				Output	
D	Clock	Clear	Preset	Q	$\overline{Q}$
X	X	0	0	1	1
X	X	0	1	0	1
X	X	1	0	1	0
X	0	1	1	STABLE	
X	1	1	1	STABLE	
X	↓	1	1	STABLE	
0	↑	1	1	0	1
1	↑	1	1	1	0

The truth table for a D flip-flop is complicated by the sequential nature of this logic device. Strictly speaking, truth tables should represent combinatorial logic properties only.

In this case, the truth table also shows the edge-triggered action of the flip-flop with ↓ representing the negative going edge and ↑ the positive going edge. 0 and 1 show stable levels.

The table is really a hybrid of a combinatorial truth table and a state table.

This flip-flop is a synchronous storage element. In other words, it stores data using a clock signal to synchronize the operation. In this case, the device is positive-going edge triggered, or simply, positive edge triggered.

The bottom two lines show that as long as the clear and present inputs are high, the positive clock edge loads the flip-flop with the data at D such that the Q output reflects the D input. The $\overline{Q}$ output is always supposed to be the inverse of the Q input.

All other conditions of the clock--high, low, or negative edge, have no effect and the outputs remain stable (at the value loaded on the previous positive edge).

The D flip-flop thus delays data by one clock period. Note that during the preceding discussion, the clear and preset inputs were assumed high.

These inputs are asynchronous, and so can change the outputs regardless of the clock or data input.

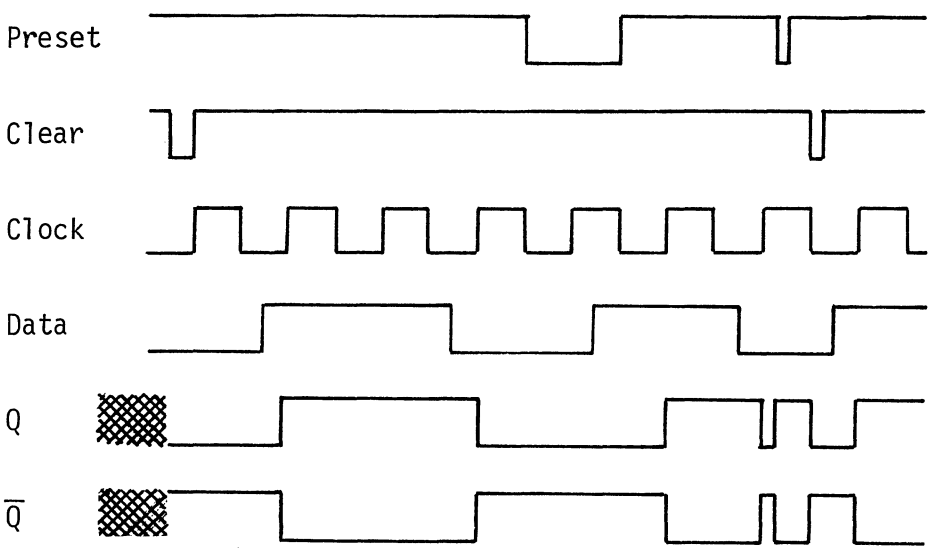
The bubbles indicate active low operation.

When both asynchronous, or "direct" inputs are low, both Q and $\overline{Q}$ go high, so this condition is normally forbidden.

In sequential circuits, other time related parameters are generally specified. Thus data inputs generally have to meet setup and hold times with respect to the active edge of the clock, or "interrogating" edge. A setup time is the time the data must be present before the active edge, and the hold time is the time for which it must continue to be present--"held", after the active edge in order for proper operation. Sequential device operation can be much better understood using another graphical technique known as a timing diagram. Such diagrams bring out the time-sequential interactions in these devices much more clearly. The next section will deal with timing diagrams.

TIMING DIAGRAMS

Shown below is a timing diagram for a D flip-flop.



STATE DIAGRAMS

Sequential circuits inherently contain storage elements each of which may be in one of two stable states. Each "state" of a digital system, as explained in the section on truth tables, could be represented by a binary number. The system changes states in response to internal and/or external conditions. The state transition may be synchronous to a clock pulse train or asynchronous. Asynchronous sequential circuits will not be covered in detail in this book, and we shall deal only with clocked logic.

State tables and state transition diagrams are additional tools of analysis and design that digital engineers use.

As an example, we shall show the state table and state transition diagram for the J-K flip-flop.

Q_n	J_n	K_n	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

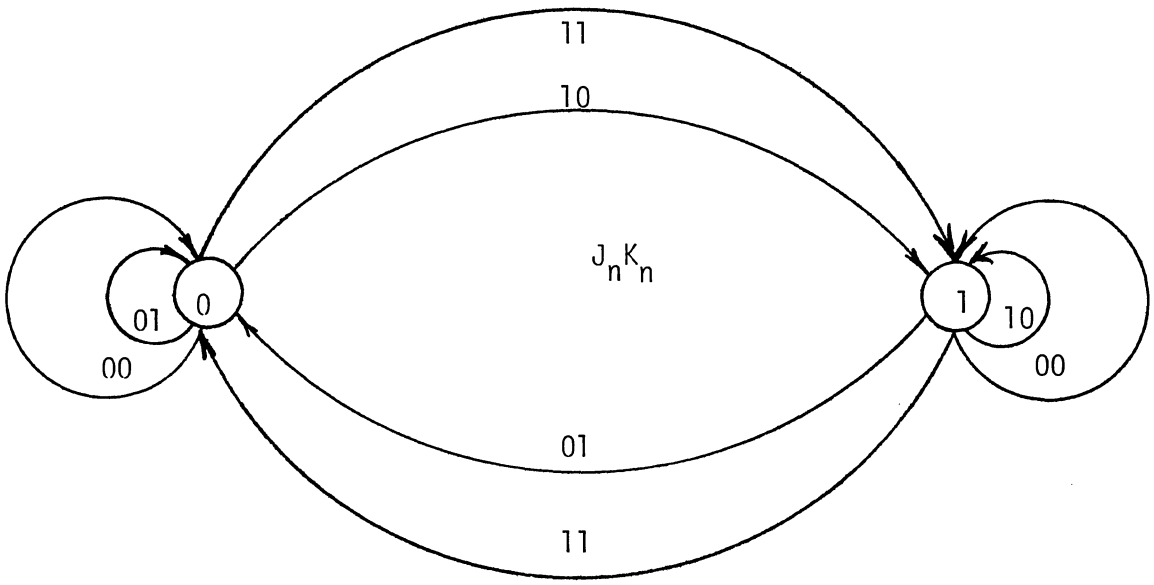
The transition, if any, from Q_n to Q_{n+1} (Q at time t_n and Q at time t_{n+1}) is triggered by the negative going edge of the clock.

In words, when J and K are zero, the outputs do not change. When J and K are both one, the output toggles at every clock pulse and when J and K are at opposite levels, Q follows J (and $\overline{Q}$ follows K).

Another form of the state table shows this relationship:

	$J_n=0, K_n=0$	$J_n=0, K_n=1$	$J_n=1, K_n=0$	$J_n=1, K_n=1$
Present State Q_n	Next State Q_{n+1}	Next State Q_{n+1}	Next State Q_{n+1}	Next State Q_{n+1}
0	0	0	1	1
1	1	0	1	0

The number of inputs and outputs in a digital system are not related to the number of states. They only determine the number of paths along which a change of state may occur. In this specific case, the output is also the state.



The state diagram shows the different states of a digital system and the conditions necessary to cause the system to change states.

Information that is not shown on a state transition diagram is presented in other visual aids such as timing diagrams.

Thus, in general, a complex system must be studied with the aid of many different tools in order to gain insight into the operation of the system from many different angles.

Digital systems may be "hardwired" or programmable. Hardwired digital systems have many logic devices scattered at random and many operations are done in parallel.

This "random logic" consists of such standard SSI and MSI functions as counters, multiplexers, decoders, latches, registers, etc.

Programmable logic systems usually have denser, more regularly arrayed chips such as ROMs, PROMs, RAMs, FPLAs, microprocessors, etc. and substitute many sequential operations for a single parallel operation, though this is not always the case.

Such systems replace the "randomness" in the logic with random bit patterns in the memory components. Programmable logic systems are gaining popularity with the advent of inexpensive LSI storage and processor devices.

APPENDIX C

OCTAL-DECIMAL INTEGER CONVERSION TABLE

0000 to 0777 (Octal) | 0000 to 0511 (Decimal)

	0	1	2	3	4	5	6	7
0000	0000	0001	0002	0003	0004	0005	0006	0007
0010	0008	0009	0010	0011	0012	0013	0014	0015
0020	0016	0017	0018	0019	0020	0021	0022	0023
0030	0024	0025	0026	0027	0028	0029	0030	0031
0040	0032	0033	0034	0035	0036	0037	0038	0039
0050	0040	0041	0042	0043	0044	0045	0046	0047
0060	0048	0049	0050	0051	0052	0053	0054	0055
0070	0056	0057	0058	0059	0060	0061	0062	0063
0100	0064	0065	0066	0067	0068	0069	0070	0071
0110	0072	0073	0074	0075	0076	0077	0078	0079
0120	0080	0081	0082	0083	0084	0085	0086	0087
0130	0088	0089	0090	0091	0092	0093	0094	0095
0140	0096	0097	0098	0099	0100	0101	0102	0103
0150	0104	0105	0106	0107	0108	0109	0110	0111
0160	0112	0113	0114	0115	0116	0117	0118	0119
0170	0120	0121	0122	0123	0124	0125	0126	0127
0200	0128	0129	0130	0131	0132	0133	0134	0135
0210	0136	0137	0138	0139	0140	0141	0142	0143
0220	0144	0145	0146	0147	0148	0149	0150	0151
0230	0152	0153	0154	0155	0156	0157	0158	0159
0240	0160	0161	0162	0163	0164	0165	0166	0167
0250	0168	0169	0170	0171	0172	0173	0174	0175
0260	0176	0177	0178	0179	0180	0181	0182	0183
0270	0184	0185	0186	0187	0188	0189	0190	0191
0300	0192	0193	0194	0195	0196	0197	0198	0199
0310	0200	0201	0202	0203	0204	0205	0206	0207
0320	0208	0209	0210	0211	0212	0213	0214	0215
0330	0216	0217	0218	0219	0220	0221	0222	0223
0340	0224	0225	0226	0227	0228	0229	0230	0231
0350	0232	0233	0234	0235	0236	0237	0238	0239
0360	0240	0241	0242	0243	0244	0245	0246	0247
0370	0248	0249	0250	0251	0252	0253	0254	0255
0400	0256	0257	0258	0259	0260	0261	0262	0263
0410	0264	0265	0266	0267	0268	0269	0270	0271
0420	0272	0273	0274	0275	0276	0277	0278	0279
0430	0280	0281	0282	0283	0284	0285	0286	0287
0440	0288	0289	0290	0291	0292	0293	0294	0295
0450	0296	0297	0298	0299	0300	0301	0302	0303
0460	0304	0305	0306	0307	0308	0309	0310	0311
0470	0312	0313	0314	0315	0316	0317	0318	0319
0500	0320	0321	0322	0323	0324	0325	0326	0327
0510	0328	0329	0330	0331	0332	0333	0334	0335
0520	0336	0337	0338	0339	0340	0341	0342	0343
0530	0344	0345	0346	0347	0348	0349	0350	0351
0540	0352	0353	0354	0355	0356	0357	0358	0359
0550	0360	0361	0362	0363	0364	0365	0366	0367
0560	0368	0369	0370	0371	0372	0373	0374	0375
0570	0376	0377	0378	0379	0380	0381	0382	0383
0600	0384	0385	0386	0387	0388	0389	0390	0391
0610	0392	0393	0394	0395	0396	0397	0398	0399
0620	0400	0401	0402	0403	0404	0405	0406	0407
0630	0408	0409	0410	0411	0412	0413	0414	0415
0640	0416	0417	0418	0419	0420	0421	0422	0423
0650	0424	0425	0426	0427	0428	0429	0430	0431
0660	0432	0433	0434	0435	0436	0437	0438	0439
0670	0440	0441	0442	0443	0444	0445	0446	0447
0700	0448	0449	0450	0451	0452	0453	0454	0455
0710	0456	0457	0458	0459	0460	0461	0462	0463
0720	0464	0465	0466	0467	0468	0469	0470	0471
0730	0472	0473	0474	0475	0476	0477	0478	0479
0740	0480	0481	0482	0483	0484	0485	0486	0487
0750	0488	0489	0490	0491	0492	0493	0491	0495
0760	0496	0497	0498	0499	0500	0501	0502	0503
0770	0504	0505	0506	0507	0508	0509	0510	0511

Octal Decimal
10000- 4096
20000- 8192
30000-12288
40000-16384
50000-20480
60000-24576
70000-28672

1000 to 1777 (Octal) | 0512 to 1023 (Decimal)

	0	1	2	3	4	5	6	7
1000	0512	0513	0514	0515	0516	0517	0518	0519
1010	0520	0521	0522	0523	0524	0525	0526	0527
1020	0528	0529	0530	0531	0532	0533	0534	0535
1030	0536	0537	0538	0539	0540	0541	0542	0543
1040	0544	0545	0546	0547	0548	0549	0550	0551
1050	0552	0553	0554	0555	0556	0557	0558	0559
1060	0560	0561	0562	0563	0564	0565	0566	0567
1070	0568	0569	0570	0571	0572	0573	0574	0575
1100	0576	0577	0578	0579	0580	0581	0582	0583
1110	0584	0585	0586	0587	0588	0589	0590	0591
1120	0592	0593	0594	0595	0596	0597	0598	0599
1130	0600	0601	0602	0603	0604	0605	0606	0607
1140	0608	0609	0610	0611	0612	0613	0614	0615
1150	0616	0617	0618	0619	0620	0621	0622	0623
1160	0624	0625	0626	0627	0628	0629	0630	0631
1170	0632	0633	0634	0635	0636	0637	0638	0639
1200	0640	0641	0642	0643	0644	0645	0646	0647
1210	0648	0649	0650	0651	0652	0653	0654	0655
1220	0656	0657	0658	0659	0660	0661	0662	0663
1230	0664	0665	0666	0667	0668	0669	0670	0671
1240	0672	0673	0674	0675	0676	0677	0678	0679
1250	0680	0681	0682	0683	0684	0685	0686	0687
1260	0688	0689	0690	0691	0692	0693	0694	0695
1270	0696	0697	0698	0699	0700	0701	0702	0703
1300	0704	0705	0706	0707	0708	0709	0710	0711
1310	0712	0713	0714	0715	0716	0717	0718	0719
1320	0720	0721	0722	0723	0724	0725	0726	0727
1330	0728	0729	0730	0731	0732	0733	0734	0735
1340	0736	0737	0738	0739	0740	0741	0742	0743
1350	0744	0745	0746	0747	0748	0749	0750	0751
1360	0752	0753	0754	0755	0756	0757	0758	0759
1370	0760	0761	0762	0763	0764	0765	0766	0767
1400	0768	0769	0770	0771	0772	0773	0774	0775
1410	0776	0777	0778	0779	0780	0781	0782	0783
1420	0784	0785	0786	0787	0788	0789	0790	0791
1430	0792	0793	0794	0795	0796	0797	0798	0799
1440	0800	0801	0802	0803	0804	0805	0806	0807
1450	0808	0809	0810	0811	0812	0813	0814	0815
1460	0816	0817	0818	0819	0820	0821	0822	0823
1470	0824	0825	0826	0827	0828	0829	0830	0831
1500	0832	0833	0834	0835	0836	0837	0838	0839
1510	0840	0841	0842	0843	0844	0845	0846	0847
1520	0848	0849	0850	0851	0852	0853	0854	0855
1530	0856	0857	0858	0859	0860	0861	0862	0863
1540	0864	0865	0866	0867	0868	0869	0870	0871
1550	0872	0873	0874	0875	0876	0877	0878	0879
1560	0880	0881	0882	0883	0884	0885	0886	0887
1570	0888	0889	0890	0891	0892	0893	0894	0895
1600	0896	0897	0898	0899	0900	0901	0902	0903
1610	0904	0905	0906	0907	0908	0909	0910	0911
1620	0912	0913	0914	0915	0916	0917	0918	0919
1630	0920	0921	0922	0923	0924	0925	0926	0927
1640	0928	0929	0930	0931	0932	0933	0934	0935
1650	0936	0937	0938	0939	0940	0941	0942	0943
1660	0944	0945	0946	0947	0948	0949	0950	0951
1670	0952	0953	0954	0955	0956	0957	0958	0959
1700	0960	0961	0962	0963	0964	0965	0966	0967
1710	0968	0969	0970	0971	0972	0973	0974	0975
1720	0976	0977	0978	0979	0980	0981	0982	0983
1730	0984	0985	0986	0987	0988	0989	0990	0991
1740	0992	0993	0994	0995	0996	0997	0998	0999
1750	1000	1001	1002	1003	1004	1005	1006	1007
1760	1008	1009	1010	1011	1012	1013	1014	1015
1770	1016	1017	1018	1019	1020	1021	1022	1023

2000 to 2777 (Octal) | 1024 to 1535 (Decimal)

	0	1	2	3	4	5	6	7
2000	1024	1025	1026	1027	1028	1029	1030	1031
2010	1032	1033	1034	1035	1036	1037	1038	1039
2020	1040	1041	1042	1043	1044	1045	1046	1047
2030	1048	1049	1050	1051	1052	1053	1054	1055
2040	1056	1057	1058	1059	1060	1061	1062	1063
2050	1064	1065	1066	1067	1068	1069	1070	1071
2060	1072	1073	1074	1075	1076	1077	1078	1079
2070	1080	1081	1082	1083	1084	1085	1086	1087
2100	1088	1089	1090	1091	1092	1093	1094	1095
2110	1096	1097	1098	1099	1100	1101	1102	1103
2120	1104	1105	1106	1107	1108	1109	1110	1111
2130	1112	1113	1114	1115	1116	1117	1118	1119
2140	1120	1121	1122	1123	1124	1125	1126	1127
2150	1128	1129	1130	1131	1132	1133	1134	1135
2160	1136	1137	1138	1139	1140	1141	1142	1143
2170	1144	1145	1146	1147	1148	1149	1150	1151
2200	1152	1153	1154	1155	1156	1157	1158	1159
2210	1160	1161	1162	1163	1164	1165	1166	1167
2220	1168	1169	1170	1171	1172	1173	1174	1175
2230	1176	1177	1178	1179	1180	1181	1182	1183
2240	1184	1185	1186	1187	1188	1189	1190	1191
2250	1192	1193	1194	1195	1196	1197	1198	1199
2260	1200	1201	1202	1203	1204	1205	1206	1207
2270	1208	1209	1210	1211	1212	1213	1214	1215
2300	1216	1217	1218	1219	1220	1221	1222	1223
2310	1224	1225	1226	1227	1228	1229	1230	1231
2320	1232	1233	1234	1235	1236	1237	1238	1239
2330	1240	1241	1242	1243	1244	1245	1246	1247
2340	1248	1249	1250	1251	1252	1253	1254	1255
2350	1256	1257	1258	1259	1260	1261	1262	1263
2360	1264	1265	1266	1267	1268	1269	1270	1271
2370	1272	1273	1274	1275	1276	1277	1278	1279
2400	1280	1281	1282	1283	1284	1285	1286	1287
2410	1288	1289	1290	1291	1292	1293	1294	1295
2420	1296	1297	1298	1299	1300	1301	1302	1303
2430	1304	1305	1306	1307	1308	1309	1310	1311
2440	1312	1313	1314	1315	1316	1317	1318	1319
2450	1320	1321	1322	1323	1324	1325	1326	1327
2460	1328	1329	1330	1331	1332	1333	1334	1335
2470	1336	1337	1338	1339	1340	1341	1342	1343
2500	1344	1345	1346	1347	1348	1349	1350	1351
2510	1352	1353	1354	1355	1356	1357	1358	1359
2520	1360	1361	1362	1363	1364	1365	1366	1367
2530	1368	1369	1370	1371	1372	1373	1374	1375
2540	1376	1377	1378	1379	1380	1381	1382	1383
2550	1384	1385	1386	1387	1388	1389	1390	1391
2560	1392	1393	1394	1395	1396	1397	1398	1399
2570	1400	1401	1402	1403	1404	1405	1406	1407
2600	1408	1409	1410	1411	1412	1413	1414	1415
2610	1416	1417	1418	1419	1420	1421	1422	1423
2620	1424	1425	1426	1427	1428	1429	1430	1431
2630	1432	1433	1434	1435	1436	1437	1438	1439
2640	1440	1441	1442	1443	1444	1445	1446	1447
2650	1448	1449	1450	1451	1452	1453	1454	1455
2660	1456	1457	1458	1459	1460	1461	1462	1463
2670	1464	1465	1466	1467	1468	1469	1470	1471
2700	1472	1473	1474	1475	1476	1477	1478	1479
2710	1480	1481	1482	1483	1484	1485	1486	1487
2720	1488	1489	1490	1491	1492	1493	1494	1495
2730	1496	1497	1498	1499	1500	1501	1502	1503
2740	1504	1505	1506	1507	1508	1509	1510	1511
2750	1512	1513	1514	1515	1516	1517	1518	1519
2760	1520	1521	1522	1523	1524	1525	1526	1527
2770	1528	1529	1530	1531	1532	1533	1534	1535

3000 to 3777 (Octal) | 1536 to 2047 (Decimal)

	0	1	2	3	4	5	6	7
3000	1536	1537	1538	1539	1540	1541	1542	1543
3010	1544	1545	1546	1547	1548	1549	1550	1551
3020	1552	1553	1554	1555	1556	1557	1558	1559
3030	1560	1561	1562	1563	1564	1565	1566	1567
3040	1568	1569	1570	1571	1572	1573	1574	1575
3050	1576	1577	1578	1579	1580	1581	1582	1583
3060	1584	1585	1586	1587	1588	1589	1590	1591
3070	1592	1593	1594	1595	1596	1597	1598	1599
3100	1600	1601	1602	1603	1604	1605	1606	1607
3110	1608	1609	1610	1611	1612	1613	1614	1615
3120	1616	1617	1618	1619	1620	1621	1622	1623
3130	1624	1625	1626	1627	1628	1629	1630	1631
3140	1632	1633	1634	1635	1636	1637	1638	1639
3150	1640	1641	1642	1643	1644	1645	1646	1647
3160	1648	1649	1650	1651	1652	1653	1654	1655
3170	1656	1657	1658	1659	1660	1661	1662	1663
3200	1664	1665	1666	1667	1668	1669	1670	1671
3210	1672	1673	1674	1675	1676	1677	1678	1679
3220	1680	1681	1682	1683	1684	1685	1686	1687
3230	1688	1689	1690	1691	1692	1693	1694	1695
3240	1696	1697	1698	1699	1700	1701	1702	1703
3250	1704	1705	1706	1707	1708	1709	1710	1711
3260	1712	1713	1714	1715	1716	1717	1718	1719
3270	1720	1721	1722	1723	1724	1725	1726	1727
3300	1728	1729	1730	1731	1732	1733	1734	1735
3310	1736	1737	1738	1739	1740	1741	1742	1743
3320	1744	1745	1746	1747	1748	1749	1750	1751
3330	1752	1753	1754	1755	1756	1757	1758	1759
3340	1760	1761	1762	1763	1764	1765	1766	1767
3350	1768	1769	1770	1771	1772	1773	1774	1775
3360	1776	1777	1778	1779	1780	1781	1782	1783
3370	1784	1785	1786	1787	1788	1789	1790	1791
3400	1792	1793	1794	1795	1796	1797	1798	1799
3410	1800	1801	1802	1803	1804	1805	1806	1807
3420	1808	1809	1810	1811	1812	1813	1814	1815
3430	1816	1817	1818	1819	1820	1821	1822	1823
3440	1824	1825	1826	1827	1828	1829	1830	1831
3450	1832	1833	1834	1835	1836	1837	1838	1839
3460	1840	1841	1842	1843	1844	1845	1846	1847
3470	1848	1849	1850	1851	1852	1853	1854	1855
3500	1856	1857	1858	1859	1860	1861	1862	1863
3510	1864	1865	1866	1867	1868	1869	1870	1871
3520	1872	1873	1874	1875	1876	1877	1878	1879
3530	1880	1881	1882	1883	1884	1885	1886	1887
3540	1888	1889	1890	1891	1892	1893	1894	1895
3550	1896	1897	1898	1899	1900	1901	1902	1903
3560	1904	1905	1906	1907	1908	1909	1910	1911
3570	1912	1913	1914	1915	1916	1917	1918	1919
3600	1920	1921	1922	1923	1924	1925	1926	1927
3610	1928	1929	1930	1931	1932	1933	1934	1935
3620	1936	1937	1938	1939	1940	1941	1942	1943
3630	1944	1945	1946	1947	1948	1949	1950	1951
3640	1952	1953	1954	1955	1956	1957	1958	1959
3650	1960	1961	1962	1963	1964	1965	1966	1967
3660	1968	1969	1970	1971	1972	1973	1974	1975
3670	1976	1977	1978	1979	1980	1981	1982	1983
3700	1984	1985	1986	1987	1988	1989	1990	1991
3710	1992	1993	1994	1995	1996	1997	1998	1999
3720	2000	2001	2002	2003	2004	2005	2006	2007
3730	2008	2009	2010	2011	2012	2013	2014	2015
3740	2016	2017	2018	2019	2020	2021	2022	2023
3750	2024	2025	2026	2027	2028	2029	2030	2031
3760	2032	2033	2034	2035	2036	2037	2038	2039
3770	2040	2041	2042	2043	2044	2045	2046	2047

4000 to 4777 (Octal) | 2048 to 2559 (Decimal)

	0	1	2	3	4	5	6	7
4000	2048	2049	2050	2051	2052	2053	2054	2055
4010	2056	2057	2058	2059	2060	2061	2062	2063
4020	2064	2065	2066	2067	2068	2069	2070	2071
4030	2072	2073	2074	2075	2076	2077	2078	2079
4040	2080	2081	2082	2083	2084	2085	2086	2087
4050	2088	2089	2090	2091	2092	2093	2094	2095
4060	2096	2097	2098	2099	2100	2101	2102	2103
4070	2104	2105	2106	2107	2108	2109	2110	2111
4100	2112	2113	2114	2115	2116	2117	2118	2119
4110	2120	2121	2122	2123	2124	2125	2126	2127
4120	2128	2129	2130	2131	2132	2133	2134	2135
4130	2136	2137	2138	2139	2140	2141	2142	2143
4140	2144	2145	2146	2147	2148	2149	2150	2151
4150	2152	2153	2154	2155	2156	2157	2158	2159
4160	2160	2161	2162	2163	2164	2165	2166	2167
4170	2168	2169	2170	2171	2172	2173	2174	2175
4200	2176	2177	2178	2179	2180	2181	2182	2183
4210	2184	2185	2186	2187	2188	2189	2190	2191
4220	2192	2193	2194	2195	2196	2197	2198	2199
4230	2200	2201	2202	2203	2204	2205	2206	2207
4240	2208	2209	2210	2211	2212	2213	2214	2215
4250	2216	2217	2218	2219	2220	2221	2222	2223
4260	2224	2225	2226	2227	2228	2229	2230	2231
4270	2232	2233	2234	2235	2236	2237	2238	2239
4300	2240	2241	2242	2243	2244	2245	2246	2247
4310	2248	2249	2250	2251	2252	2253	2254	2255
4320	2256	2257	2258	2259	2260	2261	2262	2263
4330	2264	2265	2266	2267	2268	2269	2270	2271
4340	2272	2273	2274	2275	2276	2277	2278	2279
4350	2280	2281	2282	2283	2284	2285	2286	2287
4370	2288	2289	2290	2291	2292	2293	2294	2295
4370	2296	2297	2298	2299	2300	2301	2302	2303
4400	2304	2305	2306	2307	2308	2309	2310	2311
4410	2312	2313	2314	2315	2316	2317	2318	2319
4420	2320	2321	2322	2323	2324	2325	2326	2327
4430	2328	2329	2330	2331	2332	2333	2334	2335
4440	2336	2337	2338	2339	2340	2341	2342	2343
4450	2344	2345	2346	2347	2348	2349	2350	2351
4460	2352	2353	2354	2355	2356	2357	2358	2359
4470	2360	2361	2362	2363	2364	2365	2366	2367
4500	2368	2369	2370	2371	2372	2373	2374	2375
4510	2376	2377	2378	2379	2380	2381	2382	2383
4520	2384	2385	2386	2387	2388	2389	2390	2391
4530	2392	2393	2394	2395	2396	2397	2398	2399
4540	2400	2401	2402	2403	2404	2405	2406	2407
4550	2408	2409	2410	2411	2412	2413	2114	2415
4560	2416	2417	2418	2419	2420	2421	2422	2423
4570	2424	2425	2426	2427	2428	2429	2430	2431
4600	2432	2433	2434	2435	2436	2437	2438	2439
4610	2440	2441	2442	2443	2444	2445	2446	2447
4620	2448	2449	2450	2451	2452	2453	2454	2455
4630	2456	2457	2458	2459	2460	2461	2462	2463
4640	2464	2465	2466	2467	2468	2469	2470	2471
4650	2472	2473	2474	2475	2476	2477	2478	2479
4660	2480	2481	2482	2483	2484	2485	2486	2487
4670	2488	2489	2490	2491	2492	2493	2494	2495
4700	2496	2497	2498	2499	2500	2501	2502	2503
4710	2504	2505	2506	2507	2508	2509	2510	2511
4720	2512	2513	2514	2515	2516	2517	2518	2519
4730	2520	2521	2522	2523	2524	2525	2526	2527
4740	2528	2529	2530	2531	2532	2533	2534	2535
4750	2536	2537	2538	2539	2540	2541	2542	2543
4760	2544	2545	2546	2547	2548	2549	2550	2551
4770	2552	2553	2554	2555	2556	2557	2558	2259

5000 to 5777 (Octal) | 2560 to 3071 (Decimal)

	0	1	2	3	4	5	6	7
5000	2560	2561	2562	2563	2564	2565	2566	2567
5010	2568	2569	2570	2571	2572	2573	2574	2575
5020	2576	2577	2578	2579	2580	2581	2582	2583
5030	2584	2585	2586	2587	2588	2589	2590	2591
5040	2592	2593	2594	2595	2596	2597	2598	2599
5050	2600	2601	2602	2603	2604	2605	2606	2607
5060	2608	2609	2610	2611	2612	2613	2614	2615
5070	2616	2617	2618	2619	2620	2621	2622	2623
5100	2624	2625	2626	2627	2628	2629	2630	2631
5110	2632	2633	2634	2635	2636	2637	2638	2639
5120	2640	2641	2642	2643	2644	2645	2646	2647
5130	2648	2649	2650	2651	2652	2653	2654	2655
5140	2656	2657	2658	2659	2660	2661	2662	2663
5150	2664	2665	2666	2667	2668	2669	2670	2671
5160	2672	2673	2674	2675	2676	2677	2678	2679
5170	2680	2681	2682	2683	2684	2685	2686	2687
5200	2688	2689	2690	2691	2692	2693	2694	2695
5210	2696	2697	2698	2699	2700	2701	2702	2703
5220	2704	2705	2706	2707	2708	2709	2710	2711
5230	2712	2713	2714	2715	2716	2717	2718	2719
5240	2720	2721	2722	2723	2724	2725	2726	2727
5250	2728	2729	2730	2731	2732	2733	2734	2735
5260	2736	2737	2738	2739	2740	2741	2742	2743
5270	2744	2745	2746	2747	2748	2749	2750	2751
5300	2752	2753	2754	2755	2756	2757	2758	2759
5310	2760	2761	2762	2763	2764	2765	2766	2767
5320	2768	2769	2770	2771	2772	2773	2774	2775
5330	2776	2777	2778	2779	2780	2781	2782	2783
5340	2784	2785	2786	2787	2788	2789	2790	2791
5350	2792	2793	2794	2795	2796	2797	2798	2799
5360	2800	2801	2802	2803	2804	2805	2806	2807
5370	2808	2809	2810	2811	2812	2813	2814	2815
5400	2816	2817	2818	2819	2820	2821	2822	2823
5410	2824	2825	2826	2827	2828	2829	2830	2831
5420	2832	2833	2834	2835	2836	2837	2838	2839
5430	2840	2841	2842	2843	2844	2845	2846	2847
5440	2848	2849	2850	2851	2852	2853	2854	2855
5450	2856	2857	2858	2859	2860	2861	2862	2863
5460	2864	2865	2866	2867	2868	2869	2870	2871
5470	2872	2873	2874	2875	2876	2877	2878	2879
5500	2880	2881	2882	2883	2884	2885	2886	2887
5510	2888	2889	2890	2891	2892	2893	2894	2895
5520	2896	2897	2898	2899	2900	2901	2902	2903
5530	2904	2905	2906	2907	2908	2909	2910	2911
5540	2912	2913	2914	2915	2916	2917	2918	2919
5550	2920	2921	2922	2923	2924	2925	2926	2927
5560	2928	2929	2930	2931	2932	2933	2934	2935
5570	2936	2937	2938	2939	2940	2941	2942	2943
5600	2944	2945	2946	2947	2948	2949	2950	2951
5610	2952	2953	2954	2955	2956	2957	2958	2959
5620	2960	2961	2962	2963	2964	2965	2966	2967
5630	2968	2969	2970	2971	2972	2973	2974	2975
5640	2976	2977	2978	2979	2980	2981	2982	2983
5650	2984	2985	2986	2987	2988	2989	2990	2991
5660	2992	2993	2994	2995	2996	2997	2998	2999
5670	3000	3001	3002	3003	3004	3005	3006	3007
5700	3008	3009	3010	3011	3012	3013	3014	3015
5710	3016	3017	3018	3019	3020	3021	3022	3023
5720	3024	3025	3026	3027	3028	3029	3030	3031
5730	3032	3033	3034	3035	3036	3037	3038	3039
5740	3040	3041	3042	3043	3044	3045	3046	3047
5750	3048	3049	3050	3051	3052	3053	3054	3055
5760	3056	3057	2058	3059	3060	3061	3062	3063
5770	3064	3065	3066	3067	3068	3069	3070	3071

6000 to 6777 (Octal) | 3072 to 3583 (Decimal)

	0	1	2	3	4	5	6	7
6000	3072	3073	3074	3075	3076	3077	3078	3079
6010	3080	3081	3082	3083	3084	3085	3086	3087
6020	3088	3089	3090	3091	3092	3093	3094	3095
6030	3096	3097	3098	3099	3100	3101	3102	3103
6040	3104	3105	3106	3107	3108	3109	3110	3111
6050	3112	3113	3114	3115	3116	3117	3118	3119
6060	3120	3121	3122	3123	3124	3125	3126	3127
6070	3128	3129	3130	3131	3132	3133	3134	3135
6100	3136	3137	3138	3139	3140	3141	3142	3143
6110	3144	3145	3146	3147	3148	3149	3150	3151
6120	3152	3153	3154	3155	3156	3157	3158	3159
6130	3160	3161	3162	3163	3164	3165	3166	3167
6140	3168	3169	3170	3171	3172	3173	3174	3175
6150	3176	3177	3178	3179	3180	3181	3182	3183
6160	3184	3185	3186	3187	3188	3189	3190	3191
6170	3192	3193	3194	3195	3196	3197	3198	3199
6200	3200	3201	3202	3203	3204	3205	3206	3207
6210	3208	3209	3210	3211	3212	3213	3214	3215
6220	3216	3217	3218	3219	3220	3221	3222	3223
6230	3224	3225	3226	3227	3228	3229	3230	3231
6240	3232	3233	3234	3235	3236	3237	3238	3239
6250	3240	3241	3442	3243	3244	3245	3246	3247
6260	3248	3249	3250	3251	3252	3253	3254	3255
6270	3256	3257	3258	3259	3260	3261	3262	3263
6300	3264	3265	3266	3267	3268	3269	3270	3871
6310	3272	3273	3274	3275	3276	3277	3278	3279
6320	3280	3281	3282	3283	3284	3285	3286	3287
6330	3288	3289	3290	3291	3292	3293	3294	3295
6340	3296	3297	3298	3299	3300	3301	3302	3003
6350	3304	3305	3306	3307	3308	3309	3310	3311
6360	3312	3313	3314	3315	3316	3317	3318	3319
6370	3320	3321	3322	3323	3324	3325	3326	3327
6400	3328	3329	3330	3331	3332	3333	3334	3335
6410	3336	3337	3338	3339	3340	3341	3342	3343
6420	3344	3345	3346	3347	3348	3349	3350	3351
6430	3352	3353	3354	3355	3356	3357	3358	3359
6440	3360	3361	3362	3363	3364	3365	3366	3367
6450	3368	3369	3370	3371	3372	3373	3374	3375
6460	3376	3377	3378	3379	3380	3381	3382	3383
6470	3384	3385	3386	3387	3388	3389	3390	3391
6500	3392	3393	3394	3395	3396	3397	3398	3399
6510	3400	3401	3402	3403	3404	3405	3406	3407
6520	3408	3409	3410	3411	3412	3413	3414	3415
6530	3416	3417	3418	3419	3420	3421	3422	3423
6540	3424	3425	3426	3427	3428	3429	3430	3431
6550	3432	3433	3434	3435	3436	3437	3438	3439
6560	3440	3441	3442	3443	3444	3445	3446	3447
6570	3448	3449	3450	3451	3452	3453	3454	3455
6600	3456	3457	3458	3459	3460	3461	3462	3463
6610	3464	3465	3466	3467	3468	3469	3470	3471
6620	3472	3473	3474	3475	3476	3477	3478	3479
6630	3480	3481	3482	3483	3484	3485	3486	3487
6640	3488	3489	3490	3491	3492	3493	3494	3495
6650	3496	3497	3498	3499	3500	3501	3502	3503
6660	3504	3505	3506	3507	3508	3509	3510	3511
6670	3512	3513	3514	3515	3516	3517	3518	3519
6700	3520	3521	3522	3523	3524	3525	3526	3527
6710	3528	3529	3530	3531	3532	3533	3534	3535
6720	3536	3537	3538	3539	3540	3541	3542	3543
6730	3544	3545	3546	3547	3548	3549	3550	3551
6740	3552	3553	3554	3555	3556	3557	3558	3559
6750	3560	3561	3562	3563	3564	3655	3566	3567
6760	3568	3569	3570	3571	3572	3573	3574	3575
6770	3576	3577	3578	3579	3580	3581	3582	3583

7000 to 7777 (Octal) | 3584 to 4095 (Decimal)

	0	1	2	3	4	5	6	7
7000	3584	3585	3586	3587	3588	3589	3590	3591
7010	3592	3593	3594	3595	3596	3597	3598	3599
7020	3600	3601	3602	3603	3604	3605	3606	3607
7030	3608	3609	3610	3611	3612	3613	3614	3615
7040	3616	3617	3618	3619	3620	3621	3622	3623
7050	3624	3625	3626	3627	3628	3629	3630	3631
7060	3632	3633	3634	3635	3636	3637	3638	3639
7070	3640	3641	3642	3643	3644	3645	3646	3647
7100	3648	3649	3650	3651	3652	3653	3654	3655
7110	3656	3657	3658	3659	3660	3661	3662	3663
7120	3664	3665	3666	3667	3668	3669	3670	3671
7130	3672	3673	3674	3675	3676	3677	3678	3679
7140	3680	3681	3682	3683	3684	3685	3686	3687
7150	3688	3689	3690	3691	3692	3693	3694	3695
7160	3696	3697	3698	3699	3700	3701	3702	3703
7170	3704	3705	3706	3707	3708	3709	3710	3711
7200	3712	3713	3714	3715	3716	3717	3718	3719
7210	3720	3721	3722	3723	3724	3725	3726	3727
7220	3728	3729	3730	3731	3732	3733	3734	3735
7230	3736	3737	3738	3739	3740	3741	3742	3743
7240	3744	3745	3746	3747	3748	3749	3750	3751
7250	3752	3753	3754	3755	3756	3757	3758	3759
7260	3760	3761	3762	3763	3764	3765	3766	3767
7270	3768	3769	3770	3771	3772	3773	3774	3775
7300	3776	3777	3778	3779	3780	3781	3782	3783
7310	3784	3785	3786	3787	3788	3789	3790	3791
7320	3792	3893	3794	3795	3796	3797	3798	3799
7330	3800	3801	3802	3803	3804	3805	3806	3807
7340	3808	3809	3810	3811	3812	3813	3814	3815
7350	3816	3817	3818	3819	3820	3821	3822	3823
7360	3824	3825	3826	3827	3828	3829	3830	3831
7370	3832	3833	3834	3835	3836	3837	3838	3839
7400	3840	3841	3842	3843	3844	3845	3846	3847
7410	3848	3849	3850	3851	3852	3853	3854	3855
7420	2856	3857	3858	3859	3860	3861	3862	3863
7430	3864	3865	3866	3867	3868	3869	3870	3871
7440	3872	3873	3874	3875	3876	3877	3878	3879
7450	3880	3881	3882	3883	3884	3885	3886	3887
7460	3888	3889	3890	3891	3892	3893	3894	3895
7470	3896	3897	3898	3899	3900	3901	3902	3903
7500	3904	3905	3906	3907	3908	3909	3910	3911
7510	3912	3913	3914	3915	3916	3917	3918	3919
7520	3920	3921	3922	3923	3924	3925	3926	3927
7530	3928	3929	3930	3931	3932	3933	3934	3935
7540	3936	3937	3938	3939	3940	3941	3942	3943
7550	3944	3945	3946	3947	3948	3949	3950	3951
7560	3952	3953	3954	3955	3956	3957	3958	3959
7570	3960	3961	3962	3963	3964	3965	3966	3967
7600	3968	3969	4970	3971	3972	3973	3974	3975
7610	3976	3977	3978	3979	3980	3981	3982	3983
7620	3984	3985	3986	3987	3988	3989	3990	3991
7630	3992	3993	3994	3995	3996	3997	3998	3999
7640	4000	4001	4002	4003	4004	4005	4006	4007
7650	4008	4009	4010	4011	4012	4013	4014	4015
7660	4016	4017	4018	4019	4020	4021	4022	4023
7670	4024	4025	4026	4027	4028	4029	4030	4031
7700	4032	4033	4034	4035	4036	4037	4038	4039
7710	4040	4041	4042	4043	4044	4045	4046	4047
7720	4048	4049	4050	4051	4052	4053	4054	4055
7730	4056	4057	4058	4059	4060	4061	4062	4063
7740	4064	4065	4066	4067	4068	4069	4070	4071
7750	4072	4073	4074	4075	4076	4077	4078	4079
7760	4080	4081	4082	4083	4084	4085	4086	4087
7770	4088	4089	4090	4091	4092	4093	4094	4095

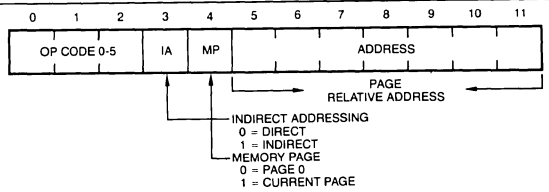
APPENDIX D
INSTRUCTION SUMMARY
AND
BIT ASSIGNMENTS

BASIC INSTRUCTIONS

MNEMONIC	OCTAL CODE	OPERATION	NO. OF STATES DIR	IND	AUTO
AND	0000	Logical AND	10	15	16
TAD	1000	Binary ADD	10	15	16
ISZ	2000	Increment, and skip if zero	16	21	22
DCA	3000	Deposit and clear AC	11	16	17
JMS	4000	Jump to subroutine	11	16	17
JMP	5000	Jump	10	15	16
IOT	6000	In/out transfer	17	—	—
OPR	7000	Operate	10/15*	—	—

*For ROTATES and OSR

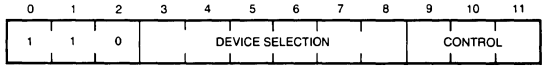
MEMORY REFERENCE INSTRUCTION FORMAT



PROCESSOR IOT INSTRUCTIONS

MNEMONIC	OCTAL CODE	OPERATION	NO. OF STATES
SKON	6000	Skip if interruption on	17
ION	6001	Interrupt turn on	17
IOF	6002	Interrupt turn off	17
SRQ	6003	Skip if INT request	17
GTF	6004	Get flags	17
RTF	6005	Return flags	17
SGT	6006	Operation is determined by external devices, if any	17
CAF	6007	Clear all flags	17

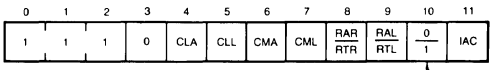
BIT ASSIGNMENTS IOT



GROUP I OPERATE MICROINSTRUCTIONS

MNEMONIC	OCTAL CODE	OPERATION	LOG SEQ.	NO. OF STATES
NOP	7000	No operation	1	10
IAC	7001	Increment accumulator	3	10
RAL	7004	Rotate accumulator left	4	15
RTL	7006	Rotate two left	4	15
RAR	7010	Rotate accumulator right	4	15
RTR	7012	Rotate two right	4	15
BSW	7002	Byte swap	4	15
CML	7020	Complement link	2	10
CMA	7040	Complement accumulator	2	10
CIA	7041	Complement and increment accumulator	2,3	10
CLL	7100	Clear link	1	10
CLL RAL	7104	Clear link—rotate accum. left	1,4	15
CLL RTL	7106	Clear link—rotate two left	1,4	15
CLL RAR	7110	Clear link—rotate accum. right	1,4	15
CLL RTR	7112	Clear link—rotate two right	1,4	15
STL	7120	Set the link	1,2	10
CLA	7200	Clear accumulator	1	10
CLA IAC	7201	Clear accumulator —Increment accumulator	1,3	10
GLT	7204	Get the link	1,4	15
CLA CLL	7300	Clear accumulator—clear link	1	10
STA	7240	Set the accumulator	1,2	10

BIT ASSIGNMENTS GROUP 1

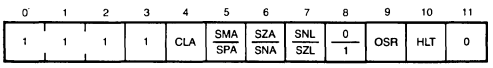


LOGICAL SEQUENCES:
1—CLA, CLL
2—CMA, CML
3—IAC
4—RAR, RAL, RTR, RTL, BSW

GROUP 2 OPERATE MICROINSTRUCTIONS

MNEMONIC	OCTAL CODE	OPERATION	LOG SEQ	NO. OF STATES
NOP	7400	No operation	1	10
HLT	7402	Halt	3	10
OSR	7404	Or with switch register	3	15
SKP	7410	Skip	1	10
SNL	7420	Skip on non-zero link	1	10
SZL	7430	Skip on zero link	1	10
SZA	7440	Skip on zero accumulator	1	10
SNA	7450	Skip on non-zero accumulator	1	10
SZA SNL	7460	Skip on zero accum, or skip on non-zero link, or both	1	10
SNA SZL	7470	Skip on non-zero accum. and skip on zero link	1	10
SMA	7500	Skip on minus accumulator	1	10
SPA	7510	Skip on positive accumulator	1	10
SMA SNL	7520	Skip on minus accum. or skip on non-zero link or both	1	10
SPA SZL	7530	Skip on positive accum. and skip on zero link	1	10
SMA SZA	7540	Skip on minus accum. or skip on zero accum. or both	1	10
SPA SNA	7550	Skip on positive accum. and skip on non-zero accum.	1	10
SMA SZA SNL	7560	Skip on minus accum. or skip on zero accum. or skip on non-zero link or all	1	10
SPA SNA SZL	7570	Skip on positive accum. and skip on non-zero accum. and skip on zero link	1	10
CLA	7600	Clear accumulator	2	10
LAS	7604	Load accumulator with switch register	1,3	15
SZA CLA	7640	Skip on zero accum. then clear accum.	1,2	10
SNA CLA	7650	Skip on non-zero accum. then clear accumulator	1,2	10
SMA CLA	7700	Skip on minus accum. then clear accumulator	1,2	10
SPA CLA	7710	Skip on positive accum. then clear accumulator	1,2	10

BIT ASSIGNMENTS GROUP 2

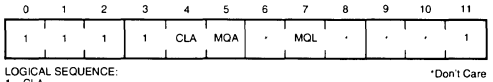


LOGICAL SEQUENCES:
1 (Bit 8 is Zero) — SMA or SZA or SNL
(Bit 8 is One) — SPA and SNA and SZL
2 — CLA
3 — OSR, HTL

GROUP 3 OPERATE MICROINSTRUCTIONS

MNEMONIC	OCTAL CODE	OPERATION	LOG SEQ	NO. OF STATES
NOP	7401	No operation	3	10
MQL	7421	MQ register load	2	10
MQA	7501	MQ register into accumulator	2	10
SWP	7521	Swap accum. and MQ register	3	10
CLA	7601	Clear accumulator	1	
CAM	7621	Clear accum. and MQ register	3	10
ACL	7701	Clear accum. and load MQ register into accumulator	3	10
CLA SWP	7721	Clear accum. and swap accum. and MQ register	3	10

BIT ASSIGNMENTS GROUP 3



LOGICAL SEQUENCE:
1—CLA
2—MQA, MQL
3—ALL OTHERS

*Don't Care

APPENDIX E

GLOSSARY

ABSOLUTE ADDRESS: A binary number that is permanently assigned as the address of a memory storage location.

ACCESS TIME: The time required to locate an off-line storage location.

ACCESSING DATA: The process of locating the off-line storage location with which data is to be transferred.

ACCUMULATOR: A 12-bit register in which the result of an operation is formed; abbreviation: AC.

ADDRESS: A label, name, or number which designates a location where information is stored.

ADDRESSING: The term given to the act of selecting a word in memory.

ALGORITHM: A prescribed set of well-defined rules or processes for the solution of a problem in a finite number of steps.

ALPHANUMERIC: Pertaining to a character set that contains both letters and numerals, and usually other characters.

ARGUMENT:
1. A variable or constant which is given in the call of a subroutine as information to it.
2. A variable upon whose value the value of a function depends.
3. The known reference factor necessary to find an item in a table or array (i.e. the index).

ARITHEMETIC AND LOGIC UNIT (ALU): The unit which performs both arithmetic and logic operations.

ARITHMETIC UNIT: The component of a computer where arithmetic and logical operations are performed.

ASCII: An abbreviation for American Standard Code for Information Interchange.

ASSEMBLE: To translate from a symbolic program to a binary program by substituting binary operation codes for symbolic operation codes and absolute or relocatable addresses for symbolic addresses.

ASSEMBLER: A program which translates symbolic op-codes into machine language and assigns memory locations for variables and constants.

AUTO-INDEXING: When one of the absolute locations from 0010 through 0017 is addressed indirectly, the content of that location is incremented by one, rewritten in that same location, and used as the effective address of the current instruction.

AUXILLARY STORAGE: Storage that supplements memory such as disk or tape.

BASE ADDRESS: A given address from which an absolute address is derived by combination with a relative address, synonymous with address constant.

BINARY: Pertaining to the number of system with a radix of two.

BINARY CODE: A code that makes use of exactly two distinct characters, 0 and 1.

BIT: A binary digit. In the IM6100 microprocessor each word is composed of 12 bits.

BLOCK: A set of consecutive machine words, characters, or digits handled as a unit, particularly with reference to I/O.

BOOTSTRAP: A technique or device designed to bring a program into the computer from an input device.

BRANCH: A point in a routine where one of two or more choices is made under control of the routine.

BUFFER: A storage area.

BUG: A mistake in the design or implementation of a program resulting in erroneous results.

BYTE: A group of binary digits usually operated upon as a unit.

CALL: To transfer control to a specified routine.

CALLING SEQUENCE: A specified set of instructions and data necessary to set up and call a given routine.

CENTRAL PROCESSING UNIT: The unit of a computing system that includes the circuits controlling the interpretation and execution of instructions--the computer proper, excluding I/O and other peripheral devices.

CHARACTER: A single letter, numeral, or symbol used to represent information.

CLEAR: To erase the contents of a storage location by replacing the contents, normally with zeros or spaces; to set to zero.

CODING: To write instructions for a computer using symbols meaningful to the computer, or to an assembler, compiler or other language processor.

COMMAND: A user order to a computer system, usually given through a Teletype keyboard.

COMMAND DECODER: That part of a computer system which interprets used commands. Also called command-string decoder.

COMPATIBILITY: The ability of an instruction or source language to be used on more than one computer.

COMPILE: To produce a binary-coded program from a program written in source (symbolic) language, by selecting appropriate subroutines from a subroutine library, as directed by the instructions or other symbols of the source program. The linkage is supplied for combining the subroutines into a workable program, and the subroutine and linkage are translated into binary code.

COMPILER: A program which translates statements and formulas written in a source language into a machine language program, e.g. a FORTRAN Compiler. Usually generates more than one machine instruction for each statement.

COMPLEMENT: (One's) To replace all 0 bits with 1 bits and vice versa. (Two's) To form the one's complement and add 1.

CONDITIONAL ASSEMBLY: Assembly of certain parts of a symbolic program only if certain conditions have been met.

CONDITIONAL SKIP: Depending upon whether a condition within the program is met, control may transfer to another point in the program.

CONSOLE: Usually the external front side of a device where controls and indicators are available for manual operation of the device.

CONVERT:
1. To change numerical data from one radix to another.
2. To transfer data from one recorded format to another.

CORE MEMORY: The main high-speed storage of a computer in which binary data is represented by the switching polarity of magnetic cores.

COUNT: The successive increase or decrease of a cumulative total of the number to times an event occurs.

COUNTER: A register or storage location (variable) used to represent the number of occurrences of an operation.

CURRENT LOCATION COUNTER: A counter kept by an assembler to determine the address assigned to an instruction or constant being assembled.

CURRENT PAGE: The page of memory "pointed to" or addressed by the Program Counter. The page we are on.

CYCLE TIME: The length of time it takes the computer to reference one word of memory.

DATA: A general term used to denote any or all facts, numbers, letters and symbols. It connotes basic elements of information which can be processed or produced by a computer.

DATA BREAK: A facility which permits I/O transfers to occur on a cycle-stealing basis without disturbing program execution.

DEBUG: To detect, locate and correct mistakes in a program.

DEVICE FLAGS: One-bit registers which record the current status of a device.

DIGITAL COMPUTER: A device that operates on discrete data, performing sequences of arithmetic and logical operations on this data.

DIRECT ADDRESS: An address that specifies the location of an instruction operand.

DOUBLE PRECISION: Pertaining to the use of two computer words to represent one number. In the IM6100 a double precision result is stored in 24 bits.

DUMP: To copy the contents of all or part of core memory, usually onto an external storage medium.

EFFECTIVE ADDRESS: The address actually used in the execution of a computer instruction.

EXECUTE: To carry out an instruction or run a program on the computer.

EXTERNAL STORAGE: A separate facility or device on which data usable by the computer is stored (such as paper tape, tape or disk.

FIELD:

1. One or more characters treated as a unit.
2. A specified area of a record used for a single type of data.
3. A division of memory on a IM6100 computer referring to a 4K section of core.

FILE: A collection of related records treated as a unit.

FLAG: A variable or register used to record the status of a program or device. In the latter case, also called a device flag.

FLIP-FLOP: A device with two stable states.

FLOATING POINT: A number system in which the position of the radix point is indicated by one part of the number (the exponent) and another part represents the significant digits (the mantissa), I/O.

FLOWCHART: A graphical representation of the operations required to carry out a data processing operation.

HARDWARE: Physical equipment, e.g., mechanical, electrical or electronic devices.

HEAD: A component that reads, records or erases data on a storage device.

INDIRECT ADDRESS: An address in a computer instruction which indicates a location where the address of the referenced operand is to be found.

INITIALIZE: To set counters, switches, and addresses to zero or other starting values at the beginning of, or at prescribed points in, a computer routine.

INSTRUCTION: A command which causes the computer or system to perform an operation. Usually one line of a source program.

INSTRUCTION FETCH (IFETCH): The act of completing an instruction address to memory and returning to the Microprocessor with the instruction.

INSTRUCTION REGISTER (IR): The register which holds the instruction when it is obtained, or received, from memory.

INTERNAL STORAGE: The storage facilities forming an integral physical part of the computer and directly controlled by the computer. Also called main memory.

INTERPRETER: A program that translates and executes source language statements at run time.

I/O: Abbreviation for input/output.

JOB: A unit of code which solves a problem, i.e. a program and all its related subroutines and data.

JUMP: A departure from the normal sequence of executing instructions in a computer.

K: An abbreviation for the prefix kilo, i.e. 1000 in decimal notation.

LABEL: One or more characters used to identify a source language statement or line.

LANGUAGE, ASSEMBLY: The machine-oriented programming language used by an assembly system.

LANGUAGE, COMPUTER: A systematic means of communicating instructions and information to the computer.

LANGUAGE, MACHINE: Information that can be directly processed by the computer, expressed in binary notation.

LANGUAGE, SOURCE: A computer language such as PAL III or FOCAL in which programs are written and which require extensive translation in order to be executed by the computer.

LEADER: The blank section of tape at the beginning of the tape.

LEAST SIGNIFICANT DIGIT: The right-most digit of a number.

LIBRARY ROUTINES: A collection of standard routines which can be incorporated into larger programs.

LINE FEED: The Teletype operation which advances the paper by one line.

LINE NUMBER: In source languages such as FOCAL, BASIC, and FORTRAN, a number which begins a line of the source program for purposes of identification. A numeric label.

LINK:
1. A one-bit register in the IM6100.
2. An address pointer generated automatically by the PAL-D or MACRO-8 Assembler to indirectly address an off-page symbol.
3. An address pointer to the next element of a list, or the next block number of a file.

LIST:
1. A set of items.
2. To print out a listing on the line printer or Teletype.

LOAD: To place data into internal storage.

LOCATION: A place in storage or memory where a unit of data or an instruction may be stored.

LOOP: A sequence of instructions that is executed repeatedly until a terminal condition prevails.

MACHINE LANGUAGE PROGRAMMING: In this text, synonymous with assembly language programming. This term is also used to mean the actual binary machine instructions.

MACRO INSTRUCTION: An instruction in a source language that is equivalent to a specified sequence of machine instructions.

MANUAL INPUT: The entry of data by hand into a device at the time of processing.

MANUAL OPERATION: The processing of data in a system by direct manual techniques.

MASK: A bit pattern which selects those bits from a word of data which are to be used in some subsequent operation.

MASS STORAGE: Pertaining to a device such as disk or tape which stores large amounts of data readily accessible to the central processing unit.

MATRIX: A rectangular array of elements. Any table can be considered a matrix.

MEMORY:
1. The alterable storage in a computer.
2. Pertaining to a device in which data can be stored and from which it can be retrieved.

MEMORY ADDRESS REGISTER (MAR): The register which contains the address where information is to be read from memory or written (stored) into memory.

MEMORY PAGING: A system by which a memory is subdivided in order to permit addressing with a limited number of binary bits.

MEMORY PROTECTION: A method of preventing the contents of some part of main memory from being destroyed or altered.

MICROCOMPUTER: A complete small computing system that usually sells for less than $5,000 and whose main processor building blocks are made of semiconductor integrated circuits. In function and structure it is similar to a minicomputer, with the main difference being price, size, speed and computing power.

MICROPROCESSOR: The semiconductor central processing unit (CPU) and one of the principal components of the microcomputer. The elements of the microprocessor are frequently contained on a single chip or within the same package but sometimes distributed over several chips. Microprocessors can contain registers, an arithmetic logic unit, a PLA, and associated timing and control logic.

MINICOMPUTER: A computer whose main frame sells for less than $25,000. Usually it is a parallel binary system with 8, 12 16, 18, or 24-bit word lengths incorporating semiconductor or magnetic memory offering 4K words to 32K words of storage. A naked minicomputer is one without cabinet, console and power supplies and consists of as little as a single PC card selling for less than $1,000.

MONITOR: The master control program that observes, supervises, controls or verifies the operation of a system.

MQ REGISTER: A register which is program accessible and interacts with the Accumulator.

NESTING:
1. Including a program loop inside loop. Special rules apply to the nesting of FORTRAN DO-loops.
2. Algebraic nesting, such as (A+B* (C+D)), where execution proceeds from the innermost to the outermost level.

NORMALIZE: To adjust the exponent and mantissa of a floating-point number so that the mantissa appears in a prescribed format.

OBJECT PROGRAM: The binary coded program which is the output after translation of a source language program.

OCTAL: Pertaining to the number system with a radix of eight.

OFF-LINE: Pertaining to equipment or devices not under direct control of the computer, or processes performed on such devices.

ON-LINE: Pertaining to equipment or devices under direct control of the computer and to programs which respond directly and immediately to user commands.

OPERAND:
1. A quantity which is affected, manipulated or operated upon.
2. The address, or symbolic name, portion of an assembly language instruction.

OPERATOR: The symbol or code which indicates an action (or operation) to be performed, e.g. + or TAD.

OR: (Inclusive) A logical operation such that the result is true if either or both operands are true, and false if both operands are false. (Exclusive) A logical operation such that the result is true if either operand is true, and false if either or both operands are false. When neither case is specifically indicated, Inclusive OR is assumed.

ORIGIN: The absolute address of the beginning of a section of code.

OUTPUT: Information transferred from the internal storage of a computer to output devices or external storage.

OVERFLOW: A condition that occurs when a mathematical operation yields a result whose magnitude is larger than the program is capable of handling.

PAGE: A 128-word section of IM6100 memory beginning at an address which is a multiple of 200.

PASS: One complete cycle during which a body of data is processed. An assembler usually requires two passes during which a source program is translated into binary code.

PATCH: To modify a routine in a rough or expedient way.

PERIPHERAL EQUIPMENT: In a data processing system, any unit of equipment distinct from the central processing unit which may provide the system with outside storage or communication.

POINTER ADDRESS: Address of a memory location containing the actual (effective) address of desired data.

PRIORITY INTERRUPT: An interrupt which is given preference over other interrupts within the system.

PROCEDURE: The course of action taken for the solution of a problem.

PROGRAM COUNTER (PC): The register which contains, at any given time, the address in memory of the next instruction.

PROGRAMMED LOGIC ARRAY (PLA): That section of the Microprocessor which correctly sequences the Microprocessor for the appropriate instruction.

PSEUDO-OP: See Pseudo-operation.

PSEUDO-OPERATION: An instruction to the assembler; an operation code that is not part of the computer's hardware command repertoire.

PUSHDOWN LIST: A list that is constructed and maintained so that the next item to be retrieved is the item most recently stored in the list.

QUEUE: A waiting list. In time-sharing, the monitor maintains a queue of user programs waiting for processing time.

RADIX: The base of a number system; the number of digits symbols required by a number system.

RANDOM ACCESS: A storage device in which the addressability of data is effectively independent of the location of the data. Synonymous with direct access.

RANDOM ACCESS MEMORY: A memory whose content can be predetermined, stored indefinitely, changed at will and retrieved at random

READ ONLY MEMORY: A memory whose content, once predetermined, is permanent and can not be changed.

REAL-TIME: Pertaining to computation performed while the related physical process is taking place so that results of the computation can be used in guiding the physical process.

RECORD: A collection of related items of data treated as a unit.

RECURSIVE SUBROUTINE: A subroutine capable of calling itself.

REGISTER: A device capable of storing a specified amount of data, usually one word.

RELATIVE ADDRESS: The number that specified the difference between the actual address and a base address.

RELOCATABLE: Used to describe a routine whose instructions are written so that they can be located and executed in different parts of core memory.

RESPONSE TIME: Time between initialing an operation from a remote terminal and obtaining the result. Includes transmission time to and from the computer, processing time and access time for files employed.

RESTART: To resume execution of a program.

ROUTINE: A set of instructions arranged in proper sequence to cause the computer to perform a desired task. A program or subprogram.

RUN: A single, continuous execution of a program.

SEGMENT:
1. That part of a long program which may be resident in memory at any one time.
2. To divide a program into two or more segments or to store part of a routine on an external storage device to be brought into core as needed.

SERIAL ACCESS: Pertaining to the sequential or consecutive transmission of data to or from memory, as with paper tape: contract with random access.

SHIFT: A movement of bits to the left or right frequently performed in the accumulator.

SIMULATE: To represent the function of a device, system or program with another device, system or program.

SINGLE STEP: Operation of a computer in such a manner that only one instruction is executed each time the computer is started.

SOFTWARE: The collection of programs and routines associated with a computer.

SOURCE LANGUAGE: See Language, source.

SOURCE PROGRAM: A computer program written in a source language.

STATEMENT: An expression or instruction in source language.

STORAGE ALLOCATION: The assignment of blocks of data and instructions to specified blocks of storage.

STORAGE CAPACITY: The amount of data that can be contained in a storage device.

STORAGE DEVICE: A device in which data can be entered, retained and retrieved.

STORE: To enter data into a storage device.

STRING: A connected sequence of entities such as characters in a command string.

SUBROUTINE, CLOSED: A subroutine not stored in the main part of a program, such a subroutine is normally called or entered with a JMS instruction and provision is made to return control to the main routine at the end of the subroutine.

SUBROUTINE, OPEN: A subroutine that must be relocated and inserted into a routine at each place it is used.

SUBSCRIPT: A number or set of numbers used to specify a particular item in an array.

SWAPPING: In a time-sharing environment, the action of either temporarily bringing a user program into core or storing it on the system device.

SWITCH: A device or programming technique for making selections.

SYMBOL TABLE: A table in which symbols and their corresponding values are recorded.

SYMBOLIC ADDRESS: A set of characters used to specify a memory location within a program.

SYMBOLIC EDITOR: A system library program which helps users in the preparation and modification of source language programs by adding, changing or deleting lines of text.

SYSTEM: A combination of software and hardware which performs specific processing operations.

TABLE: A collection of data stored for ease of reference, generally as an array.

TEMPORARY REGISTER (TEMP): A register which is used primarily as a latch for the result and ALU operation before it is sent to the destination register to avoid race conditions.

TEMPORARY STORAGE: Storage locations reserved for immediate results.

TERMINAL: A peripheral device in a system through which data can enter or leave the computer.

TIMESHARING: A method of allocating central processor time and other computer resources to multiple users so that the computer, in effect, processes a number of programs simultaneously.

TIME QUANTUM: In time-sharing, a unit of time allotted to each user by the monitor.

TOGGLE: To use switches to enter data into the computer memory.

TRANSLATE: To convert from one language to another.

TRUNCATION: The reduction of precision by dropping one or more of the least significant digits, e.g. 3.141592 truncated to four decimal digits is 3.141.

UNDERFLOW: A condition that occurs when a floating point operation yields a result whose magnitude is smaller than the program is capable of expressing.

USER: Programmer or operator of a computer.

VARIABLE: A symbol whose value changes during execution of a program.

WORD: With the IM6100, a 12-bit unit of data which may be stored in one addressable location.

WRITE: To transfer information from memory to a peripheral device or to auxiliary storage.

ZERO PAGE: The first page in the subdivided memory.

ZOMBIE: Appearance assumed by programmer attempting to debug undocumented object code.

APPENDIX F
ASCII CHARACTER CODES

CHARACTER CODES

8-bit ASCII CODE	6-bit CODE	CHARACTER REPRESENTATION	REMARKS
240	40		space (non-printing)
241	41	!	exclamation point
242	42	"	quotation marks
243	43	#	number sign
244	44	$	dollar sign
245	45	%	percent
246	46	&	ampersand
247	47	'	apostrophe or acute accent
250	50	(	opening parenthesis
251	51	)	closing parenthesis
252	52	*	asterisk
253	53	+	plus
254	54	,	comma
255	55	-	minus sign or hyphen
256	56	.	period or decimal point
257	57	/	slash
260	60	0	
261	61	1	
262	62	2	
263	63	3	
264	64	4	
265	65	5	
266	66	6	
267	67	7	
270	70	8	
271	71	9	
272	72	:	colon
273	73	;	semicolon
274	74	<	less than
275	75	=	equals
276	76	>	greater than
277	77	?	question mark

8-bit ASCII CODE	6-bit CODE	CHARACTER REPRESENTATION	REMARKS
300	00	@	at sign[1]
301	01	A	
302	02	B	
303	03	C	
304	04	D	
305	05	E	
306	06	F	
307	07	G	
310	10	H	
311	11	I	
312	12	J	
313	13	K	
314	14	L	
315	15	M	
316	16	N	
317	17	O	
320	20	P	
321	21	Q	
322	22	R	
323	23	S	
324	24	T	
325	25	U	
326	26	V	
327	27	W	
330	30	X	
331	31	Y	
332	32	Z	
333	33	[	opening bracket, SHIFT/K
334	34	\	backslash, SHIFT/L
335	35	]	closing bracket, SHIFT/M
336	36	↑	up arrow
337	37	←	back arrow[2]

Footnotes:

(1) In 6-bit code, 00_8 represents CARRIAGE RETURN

(2) In 6-bit code, 37_8 represents TAB

CONTROL CODES

8-bit ASCII CODE	CHARACTER NAME		REMARKS
000	null		Ignored in ASCII input
200	leader/trailer		Leader/trailer code precedes and follows the data portion of binary files
203	CTRL/C	(1)	IFDOS break character, forces return to Keyboard Monitor, echoed as ↑C
207	BELL		CTRL/G
211	TAB		CTRL/I, horizontal tabulation
212	LINE FEED	(2)	Used as a control character by the Command Decoder and ODT
213	VT		CTRL/K, vertical tabulation
214	FORM		CTRL/L, form feed
215	RETURN		Carriage return, generally echoed as carriage return followed by a line feed
217	CTRL/O		Break Character, used conventionally to suppress Teletype output, echoed as ↑O
225	CTRL/U		Delete current input line, echoes as ↑U
232	CTRL/Z	(3)	End-of-File character for all ASCII and binary files (in relocatable binary files CTRL/Z is not a terminator if it occurs before the trailer code)
233	ESC		Escape replaces ALTMODE on some terminals Considered equivalent to ALTMODE
375	ALTMODE		Special break character for Teletype input
376	PREFIX		PREFIX replaces ALTMODE on some terminals. Considered equivalent to ALTMODE
377	RUBOUT		Key is labeled DELETE on some terminals Deletes the previous character typed

(1) IFDOS break character--does not affect INTERCEPT JR. MONITOR

(2) OCTAL DEBUGGING TECHNIQUE program as supplied on IM6312 ROM

(3) Applies to IFDOS (INTERSIL FLOPPY DISK OPERATING SYSTEM)

APPENDIX G
LOADING CONSTANTS INTO THE ACCUMULATOR

MNEMONIC	DECIMAL CONSTANT	OCTAL CODE	INSTRUCTIONS COMBINED
K0000 =	0	7300	CLA CLL
K0001 =	1	7301	CLA CLL IAC
K0002 =	2	7305	CLA CLL IAC RAL
		(or)	
K0002 =	2	7326	CLA CLL CML RTL
K0003 =	3	7325	CLA CLL CML IAC RAL
K0004 =	4	7307	CLA CLL IAC RTL
K0006 =	6	7327	CLA CLL CML IAC RTL
K0100 =	64	7203	CLA IAC BSW
K2000 =	1024	7332	CLA CLL CML RTR
K3777 =	2047	7350	CLA CLL CMA RAR
K4000 =	-0	7330	CLA CLL CML RAR
K5777 =	-1025	7352	CLA CLL CMA RTR
K6000 =	-1024	7333	CLA CLL CML IAC RTL
K7775 =	-3	7346	CLA CLL CMA RTL
K7776 =	-2	7344	CLA CLL CMA RAL
K7777 =	-1	7340	CLA CLL CMA

APPENDIX H

KEY BOARD TENNIS PROGRAM WITH INTERCEPT JR.

DEMO PROGRAM: "PING"

IN 'PING', THE PLAYER PLAYS AGAINST THE MACHINE. THE COMPUTER "SERVES" FROM THE LEFT, AND THE "BALL" TRAVELS ALONG THE LED'S UNTIL IT REACHES BIT 11, THE RIGHTMOST LED.

IF THE PLAYER PRESSES THE YELLOW BUTTON (IAC), THE BALL WILL BE RETURNED WITH A 'CLICK'. THE MACHINE WILL RETURN THE BALL AND THE SEQUENCE IS REPEATED.

IN ORDER TO ADD EXCITEMENT TO THE GAME, EACH TIME THE PLAYER RETURNS THE BALL, IT SPEEDS UP.

WHEN THE PLAYER MISSES, BY PRESSING THE BUTTON TOO SOON OR TOO LATE, THE MACHINE BUZZES, DELAYS, THEN SERVES AT THE SLOWEST RATE.

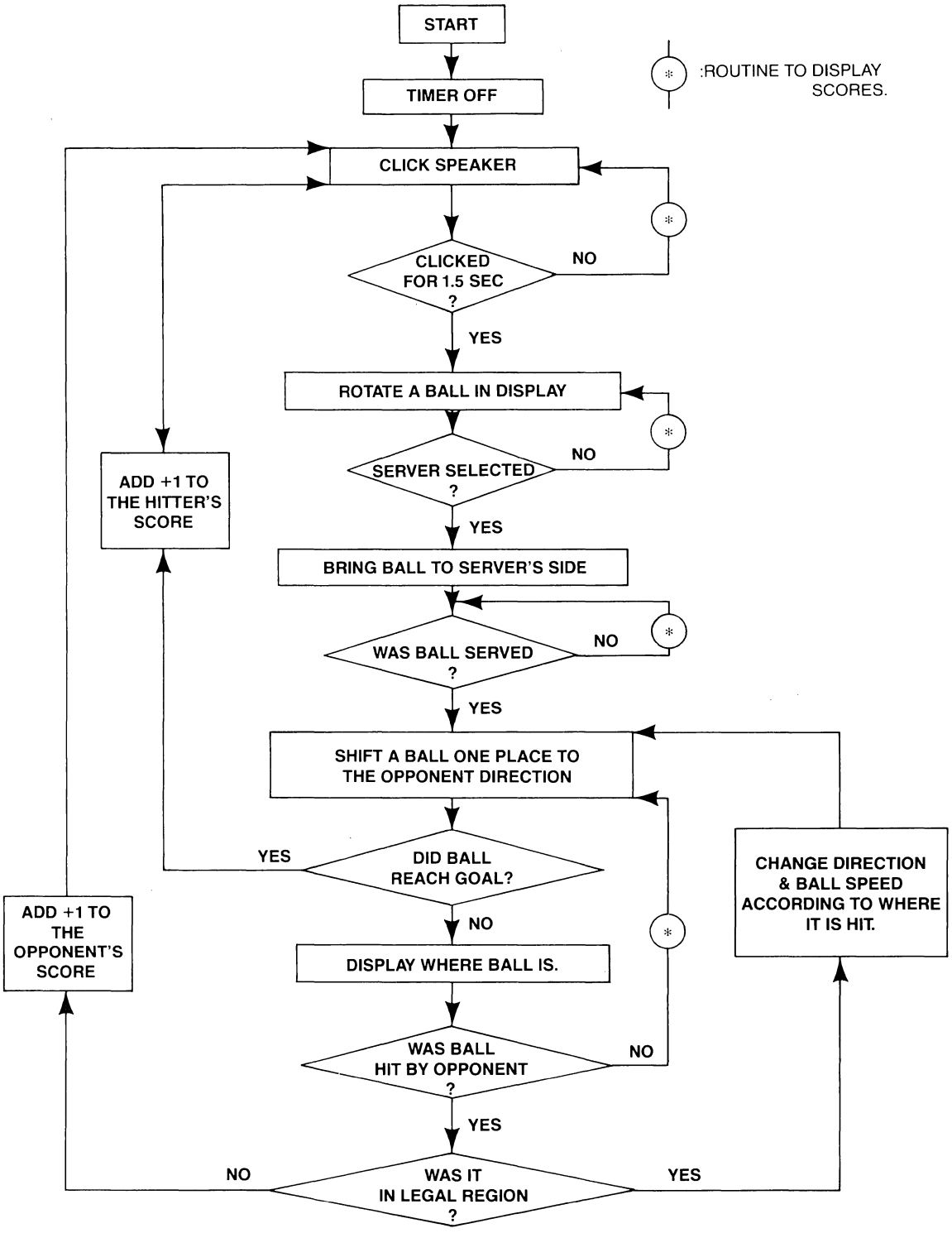
HAVE FUN!

(NOTE: THE CONTENTS OF LOCATION 0262 DETERMINE THE ORIGINAL SPEED OF THE BALL, AND LOCATION 0263 DETERMINES HOW FAST IT SPEEDS UP.)

"PING"

$ADDRESS_8$	$CONTENTS_8$	$ADDRESS_8$	$CONTENTS_8$	$ADDRESS_8$	$CONTENTS_8$
0201	7300	0223	7320	0245	1263
0202	1262	0224	6404	0246	3264
0203	3264	0225	6401	0247	7004
0204	7330	0226	2265	0250	3265
0205	6401	0227	5223	0251	1264
0206	6404	0230	7010	0252	3266
0207	3265	0231	2265	0253	1265
0210	1264	0232	5231	0254	6404
0211	3266	0233	7440	0255	7450
0212	7604	0234	5230	0256	5204
0213	7440	0235	5201	0257	2266
0214	5236	0236	6401	0260	5255
0215	2266	0237	7300	0261	5247
0216	5212	0240	1265	0262	0000
0217	1265	0241	7010	0263	1000
0220	7010	0242	7440	0264	—
0221	7440	0243	5223	0265	—
0222	5206	0244	1264	0266	—

FLOWCHART FOR KEYBOARD TENNIS PROGRAM WITH INTERCEPT JR.



```
                /KEY BOARD TENNIS WITH INTERCEPT JR.
                /
                /
                /
                /
                /
                /
                /
                /
                /RULES:
                /       START AT LOCATION #200.
                /       SINCE JR IS WAITING FOR SIGN OF STARTER,
                /       PRESS IAC OR CTR WHCIHEVER STARTS FIRST
                /       TO PREPARE FOR SERVICE.
                /       THEN, SERVE THE BALL BY PRESSING THE KEY.
                /       THE OPPONENT MUST PRESS KEY BEFORE BALL
                /       HITTING THE SIDE BUT IN THE NEAREST 2 BITS
                /       NOT TO LOSE POINTS.
                /       SCORE IS +1 FOR ONE SUCCESSFUL GOAL AND
                /       +1 BY THE OPPONENT'S FAULT.  THE HIGHEST
                /       SCORE WHICH CAN BE HANDLED IS 99.
                /
                /
                /DEFINITIONS:
                /
                /
          6400          WRITED=6400     /WRITE DISPLAY.
          6401          CLICK=6401      /CLICK SPEAKER.
          6402          TIMER=6402      /TIMER ON OR OFF.
          6404          WRITES=6404     /WRITE DISPLAY OF I/O BRD.

                /SUBPROGRAMS:
          0020  *20
00020     0000  KEY,     0       /TO DETECT KEY BOARD.
00021     7300          CLA CLL
00022     7604          LAS             /LOAD AC WITH SR.
00023     7004          RAL             /CTR=4000.
00024     7430          SZL             /CTR KEY PRESSED?
00025     5040          JMP ID0         /YES.
00026     7004          RAL             /IAC=2000.
00027     7620          SNL CLA         /IAC KEY PRESSED?
00030     5420          JMP I KEY       /NEITHER PRESSED.
00031     7140          CLL CMA         /PUT ALL 1'S IN AC.
00032     3120          DCA ID          /ID=1 FOR IAC PLAYER.
00033     7604  GO,     LAS             /STOP FURTHER EXECUTION
00034     7640          SZA CLA         /UNTIL KEY IS RELEASED.
00035     5033          JMP .-2
00036     2020          ISZ KEY         /TO GET OUT OF WAITING LOOP.
00037     5420          JMP I KEY
00040     7300  ID0,    CLL CLA
00041     3120          DCA ID          /ID=0 FOR CTR PLAYER.
00042     5033          JMP GO          /RETURN.
                /
                /
00043     0000  SHOW,    0       /SUBROUTINE TO DISPLAY SCORES:
00044     7300          CLA CLL
00045     3117          DCA DIGIT2      /CLEAR REGISTER DIGIT2.
00046     1124          TAD SCORE1      /BRING IAC PLAYER'S SCORE.
00047     4074          JMS DECIML      /CONVERT OCTAL TO DECIMAL.
00050     1116          TAD DIGIT1      /1ST DECIMAL DIGIT.
00051     3121          DCA SAVE1       /STORE IT IN SAVE1.
00052     1117          TAD DIGIT2      /STORE 2ND DECIMAL DIGIT
00053     3122          DCA SAVE10      /IN SAVE10.
00054     3117          DCA DIGIT2      /CLEAR DIGIT2.
00055     1125          TAD SCORE2      /BRING CTR PLAYER'S SCORE.
00056     4074          JMS DECIML      /CONVERT IT INTO DECIMAL NO.
00057     1116          TAD DIGIT1      /SHIFT 1ST DECIMAL NO. INTO
00060     7106          CLL RTL         /2ND BITE FROM RIGHT.
00061     7006          RTL
00062     1121          TAD SAVE1       /JOIN TO IAC PLAYER'S SCORE.
00063     1133          TAD K4000       /SET BIT #0 TO DISPLAY THEM.
00064     4106          JMS DELAY       /DISPLAY 1ST DECIMAL DIGITS
00065     1117          TAD DIGIT2      /OF BOTH PLAYER'S SCORES.
00066     7106          CLL RTL         /SHIFT 2ND DECIMAL NO.
00067     7006          RTL
00070     1122          TAD SAVE10      /JOIN TO IAC PLAYER'S SCORE.
00071     1132          TAD K2000       /SET BIT #1.
00072     4106          JMS DELAY       /DISPLAY 2ND DECIMAL DIGITS
00073     5443          JMP I SHOW      /OF BOTH SCORES & RETURN.
                /
                /
00074     0000  DECIML,  0       /SUBROUTINE TO CONVERT OCTAL TO DECIMAL.
00075     7100          CLL             /KILL LINK BIT.
00076     1127          TAD M12         /AC=-12.
00077     7420          SNL             /NO MORE 2ND DECIMAL DIGITS?
00100     5103          JMP OUT         /IF NOT, OUTPUT RESULTS.
```

```
00101  2117          ISZ DIGIT2       /IF YES, COUNT THE DIGITS.
00102  5075          JMP DECIML+1     /EXHAUST 10TH DIGIT.
00103  1130  OUT,    TAD P12          /ADD 10 TO COMPENSATE.
00104  3116          DCA DIGIT1       /STORE IT.
00105  5474          JMP I DECIML
                     /
                     /
00106  0000  DELAY,  0        /SUBROUTINE TO DISPLAY & DELAY TIME.
00107  6400          WRITED           /DISPLAY AC CONTENTS.
00110  7300          CLA CLL
00111  1131          TAD M7000        /TO COUNT 512.
00112  3123          DCA TEMP
00113  2123          ISZ TEMP         /COMPLETED COUNTING?
00114  5113          JMP .-1          /NOT YET.
00115  5506          JMP I DELAY      /YES.
                     /
                     /
                     /DATA (1):
                     /
00116  0000          DIGIT1, 0
00117  0000          DIGIT2, 0
00120  0000          ID,     0
00121  0000          SAVE1,  0
00122  0000          SAVE10, 0
00123  0000          TEMP,   0
00124  0000          SCORE1, 0
00125  0000          SCORE2, 0
00126  7774          M4,     -4
00127  7766          M12,    -12
00130  0012          P12,    0012
00131  7000          M7000,  7000
00132  2000          K2000,  2000
00133  4000          K4000,  4000
00134  7700          K7700,  7700

                     /PROGRAM FOR GAME STARTS HERE:
       0200  *200    /STARTING ADDRESS.
00200  7301          CLA CLL IAC      /AC=1 FOR TIMER OFF.
00201  6402          TIMER            /AC MUST BE 0 FOR TIMER ON.
00202  7200          CLA
00203  3124          DCA SCORE1       /INITIAL SCORE.
00204  3125          DCA SCORE2
00205  1134  DISPLY, TAD K7700       /CLICK SPEAKER 64 TIMES
00206  3361          DCA COUNT        /IN 1.5 SEC FOR STARTING SIGN.
00207  6401          CLICK
00210  4043          JMS SHOW         /TO KEEP DISPLAYING.
00211  2361          ISZ COUNT
00212  5207          JMP .-3
00213  7301          CLA CLL IAC      /AC=1.
00214  6404  RLEFT,  WRITES          /DISPLAY AC.
00215  3362          DCA SR           /SAVE DISPLAY BIT.
00216  4355          JMS BOARD        /CHECK KEY COMMAND.
00217  4043          JMS SHOW         /40 MS TIME DELAY
00220  4043          JMS SHOW         /TO KEEP DISPLAYING.
00221  1362          TAD SR           /BRING DISPLAY BIT BACK.
00222  7004          RAL              /SHIFT LEFT ONE.
00223  7420          SNL              /REACHED TO EDGE?
00224  5214          JMP RLEFT        /NOT YET.
00225  7010          RAR              /YES.
00226  6404  RRIGHT, WRITES          /DISPLAY.
00227  3362          DCA SR           /SAVE IT.
00230  4355          JMS BOARD        /CHECK KEY INPUT.
00231  4043          JMS SHOW         /40 MS TIME DELAY TO
00232  4043          JMS SHOW         /DISPLAY.
00233  1362          TAD SR
00234  7010          RAR              /SHIFT RIGHT ONE.
00235  7420          SNL              /REACHED TO EDGE?
00236  5226          JMP RRIGHT       /IF NOT, KEEP SHIFTING.
00237  7004          RAL              /IF YES, CHANGE DIRECTION.
00240  5214          JMP RLEFT
00241  1120  START,  TAD ID          /CHCK WHICH PLAYER FIRST.
00242  7650          SNA CLA          /THE FOLLOWING ROUTINE
00243  5251          JMP .+6          /BRINGS BALL TO THE
00244  7101          CLL IAC          /PLAYER'S SIDE.
00245  6404          WRITES
00246  3362          DCA SR           /SAVE DISPLAY BIT.
00247  1364          TAD LEFT         /LEFT=RAL.
00250  5255          JMP .+5
00251  1133          TAD K4000
00252  6404          WRITES
00253  3362          DCA SR           /SAVE DISPLAY BIT.
00254  1365          TAD RIGHT        /RIGHT=RAR.
00255  3266          DCA ROTATE       /DEFINE SHIFT DIRECTION.
00256  4020          JMS KEY          /GAME STARTED?
00257  5261          JMP .+2          /NOT YET.
00260  5263          JMP .+3          /YES, STARTED.
00261  4043          JMS SHOW         /TO KEEP DISPLAYING.
00262  5256          JMP .-4          /CHECK KEY AGAIN.
00263  1274          TAD SPEED+1      /INITIALIZE SPEED.
00264  3273          DCA SPEED
```

```
00265  1362           TAD SR          /BRING DISPLAY BIT TO SHIFT.
00266  0000  ROTATE,  0               /RAL OR RAR IS STORED HERE.
00267  7430           SZL             /SUCCEEDED TO GOAL?
00270  5341           JMP SCORE       /IF YES, SCORE & CLICK.
00271  6404           WRITES          /DISPLAY NEW SHIFTED BIT.
00272  3362           DCA SR          /SAVE DISPLAY BIT.
00273  4043  SPEED,   JMS SHOW        /THIS IS ONLY FOR SERVER.
00274  4043           JMS SHOW        /20 MS TIME DELAY.
00275  4043           JMS SHOW        /20 MS.
00276  4043           JMS SHOW        /20 MS.
00277  4043           JMS SHOW        /FASTEST=40 MS, SLOWEST=100 MS.
00300  4020           JMS KEY         /OPPONENT KEY PRESSED?
00301  5265           JMP ROTATE-1    /NOT YET, SO KEEP SHIFTING.
00302  1120           TAD ID
00303  7640           SZA CLA         /WHICH PLAYER RECEIVED BALL?
00304  5321           JMP CTR         /CTR SIDE.
00305  1362           TAD SR          /IAC SIDE.
00306  1131           TAD M7000       /DETERMINE RETURN SPEED.
00307  7440           SZA             /HIT BALL AT 2ND BIT?
00310  5314           JMP A1          /NO.
00311  1131  EASY,    TAD M7000       /IF YES, GIVE EASY BALL.
00312  3273           DCA SPEED       /M7000="NOP".
00313  5330           JMP CHANGE      /RETURN THE BALL.
00314  7710  A1,      SPA CLA         /IS IT FAULT OR BEST BIT?
00315  5351           JMP FAULT       /HIT IN WRONG REGION.
00316  1363  DFFCLT,  TAD JMPDFF      /IT WAS BEST HIT.  SO, RETURN
00317  3273           DCA SPEED       /BALL FASTEST.
00320  5330           JMP CHANGE
00321  1362  CTR,     TAD SR
00322  1126           TAD M4          /AC=-4.
00323  7450           SNA             /HIT AT 2ND BIT.
00324  5311           JMP EASY        /YES.
00325  7700           SMA CLA         /IS IT FAULT HIT?
00326  5351           JMP FAULT       /YES.
00327  5316           JMP DFFCLT      /NO, IT WAS BEST HIT.
00330  1120  CHANGE,  TAD ID          /CHANGE DIRECTION.
00331  7650           SNA CLA
00332  5335           JMP .+3
00333  1364           TAD LEFT
00334  5336           JMP .+2
00335  1365           TAD RIGHT
00336  3266           DCA ROTATE      /DEFINE NEW DIRECTION.
00337  7100           CLL             /CLEAR USELESS LINK BIT.
00340  5265           JMP ROTATE-1    /SHIFT TO THE DIRECTION.
00341  7300  SCORE,   CLA CLL
00342  1120           TAD ID
00343  7650           SNA CLA         /WHICH SCORED?
00344  5347           JMP .+3
00345  2124           ISZ SCORE1      /IAC SIDE.
00346  5205           JMP DISPLY
00347  2125           ISZ SCORE2      /CTR SIDE.
00350  5205           JMP DISPLY
00351  1120  FAULT,   TAD ID          /CHECK WHO WAS AGAINST RULE.
00352  7040           CMA             /GIVE POINT TO THE OPPONENT.
00353  3120           DCA ID
00354  5342           JMP SCORE+1
                /
00355  0000  BOARD,   0       /SUBROUTINE BOARD.
00356  4020           JMS KEY         /CHECK KEY.
00357  5755           JMP I BOARD     /IF NO INPUT, RETURN TO LOOP.
00360  5241           JMP START       /IF SIGN, START GAME.
                /
                /
                /DATA (2):
                /
00361  0000           COUNT,  0
00362  0000           SR,     0
00363  5276           JMPDFF, 5276
00364  7004           LEFT,   7004
00365  7010           RIGHT,  7010
                /
                /
00366  7000  NOP
```

```
A1       0314
BOARD    0355
CHANGE   0330
CLICK    6401
COUNT    0361
CTR      0321
DECIML   0074
DELAY    0106
DFFCLT   0316
DIGIT1   0116
DIGIT2   0117
DISPLY   0205
EASY     0311
FAULT    0351
GO       0033
ID       0120
ID0      0040
JMPDFF   0363
KEY      0020
K2000    0132
K4000    0133
K7700    0134
LEFT     0364
M12      0127
M4       0126
M7000    0131
OUT      0103
P12      0130
RIGHT    0365
RLEFT    0214
ROTATE   0266
RRIGHT   0226
SAVE1    0121
SAVE10   0122
SCORE    0341
SCORE1   0124
SCORE2   0125
SHOW     0043
SPEED    0273
SR       0362
START    0241
TEMP     0123
TIMER    6402
WRITED   6400
WRITES   6404
```

10900 N. Tantau Ave., Cupertino, Calif. 95014, (408) 996-5000, TWX 910-338-0228