



INTEL CORP. 3065 Bowers Avenue, Santa Clara, California 95051 • (408) 246-7501

MCS^{T.M.}-4 Assembly Language Programming Manual

PRELIMINARY EDITION

December 1973

-- TABLE OF CONTENTS --
The INTELLEC 4 Microcomputer System
Programming Manual

| | <u>PAGE NO.</u> |
|--|-----------------|
| 1.0 INTRODUCTION | 1-1 |
| 2.0 COMPUTER ORGANIZATION | 2-1 |
| 2.1 WORKING REGISTERS (INDEX REGISTERS) | 2-2 |
| 2.2 THE ACCUMULATOR | 2-3 |
| 2.3 MEMORIES | 2-3 |
| 2.3.1 READ-ONLY MEMORY | 2-3 |
| 2.3.2 PROGRAM RANDOM ACCESS MEMORY | 2-4 |
| 2.3.3 DATA RANDOM ACCESS MEMORY | 2-5 |
| 2.4 THE STACK | 2-7 |
| 2.5 INPUT/OUTPUT | 2-9 |
| 2.6 COMPUTER PROGRAM REPRESENTATION IN MEMORY | 2-10 |
| 2.7 MEMORY ADDRESSING | 2-13 |
| 2.7.1 DIRECT ADDRESSING | 2-13 |
| 2.7.2 SAME PAGE ADDRESSING | 2-14 |
| 2.7.3 INDIRECT ADDRESSING | 2-15 |
| 2.7.4 IMMEDIATE ADDRESSING | 2-16 |
| 2.7.5 PROGRAM RAM ADDRESSING | 2-16 |
| 2.7.6 DATA RAM ADDRESSING | 2-16 |
| 2.7.7 SUBROUTINES AND USE OF THE STACK FOR ADDRESSING | 2-17 |
| 2.8 CARRY BIT | 2-20 |
| 3.0 THE 4004 INSTRUCTION SET | 3-1 |
| 3.1 ASSEMBLY LANGUAGE | 3-1 |
| 3.1.1 HOW ASSEMBLY LANGUAGE IS USED . . . | 3-1 |
| 3.1.2 STATEMENT MNEMONICS | 3-4 |

-- TABLE OF CONTENTS -- (Continued)

The INTELLEC 4 Microcomputer System
Programming Manual

| | <u>PAGE No.</u> |
|---|-----------------|
| 3.1.3 LABEL FIELD | 3-5 |
| 3.1.4 CODE FIELD | 3-6 |
| 3.1.5 OPERAND FIELD | 3-7 |
| 3.1.6 COMMENT FIELD | 3-11 |
| 3.2 DATA STATEMENTS | 3-12 |
| 3.2.1 TWO'S COMPLEMENT | 3-12 |
| 3.2.2 CONSTANT DATA | 3-15 |
| 3.3 INDEX REGISTER INSTRUCTIONS | 3-16 |
| 3.3.1 INC INCREMENT REGISTER | 3-17 |
| 3.3.2 FIN FETCH INDIRECT | 3-18 |
| 3.4 INDEX REGISTER TO ACCUMULATOR INSTRUCTIONS | 3-20 |
| 3.4.1 ADD ADD REGISTER TO ACCUMULATOR WITH CARRY | 3-21 |
| 3.4.2 SUB SUBTRACT REGISTER FROM ACCUM- ULATOR WITH BORROW | 3-22 |
| 3.4.3 LD LOAD ACCUMULATOR | 3-24 |
| 3.4.4 XCH EXCHANGE REGISTER AND ACCUM- ULATOR | 3-25 |
| 3.5 ACCUMULATOR INSTRUCTIONS | 3-26 |
| 3.5.1 CLB CLEAR BOTH | 3-27 |
| 3.5.2 CLC CLEAR CARRY | 3-27 |
| 3.5.3 IAC INCREMENT ACCUMULATOR | 3-28 |
| 3.5.4 CMC COMPLEMENT CARRY | 3-29 |
| 3.5.5 CMA COMPLEMENT ACCUMULATOR | 3-29 |
| 3.5.6 RAL ROTATE ACCUMULATOR LEFT THROUGH CARRY | 3-30 |
| 3.5.7 RAR ROTATE ACCUMULATOR RIGHT THROUGH CARRY | 3-31 |
| 3.5.8 TCC TRANSMIT CARRY AND CLEAR | 3-32 |
| 3.5.9 DAC DECREMENT ACCUMULATOR | 3-32 |

-- TABLE OF CONTENTS -- (Continued)

The INTELLEC 4 Microcomputer System
Programming Manual

| | <u>PAGE No.</u> |
|---|-----------------|
| 3.5.10 TCS TRANSFER CARRY SUBTRACT | 3-33 |
| 3.5.11 STC SET CARRY | 3-34 |
| 3.5.12 DAA DECIMAL ADJUST ACCUMULATOR | 3-34 |
| 3.5.13 KBP KEYBOARD PROCESS | 3-35 |
| 3.6 IMMEDIATE INSTRUCTIONS | 3-36 |
| 3.6.1 FIM FETCH IMMEDIATE | 3-36 |
| 3.6.2 LDM LOAD ACCUMULATOR | 3-37 |
| 3.7 TRANSFER OF CONTROL INSTRUCTIONS | 3-38 |
| 3.7.1 JUN JUMP UNCONDITIONALLY | 3-38 |
| 3.7.2 JIN JUMP INDIRECT | 3-40 |
| 3.7.3 JCN JUMP ON CONDITION | 3-41 |
| 3.7.4 ISZ INCREMENT AND SKIP IF ZERO | 3-43 |
| 3.8 SUBROUTINE LINKAGE COMMANDS | 3-45 |
| 3.8.1 JMS JUMP TO SUBROUTINE | 3-45 |
| 3.8.2 BBL BRANCH BACK AND LOAD | 3-46 |
| 3.9 NOP INSTRUCTION NO OPERATION | 3-48 |
| 3.10 MEMORY SELECTION INSTRUCTIONS | 3-48 |
| 3.10.1 DCL DESIGNATE COMMAND LINE | 3-48 |
| 3.10.2 SRC SEND REGISTER CONTROL | 3-50 |
| 3.11 INPUT/OUTPUT RAM INSTRUCTIONS | 3-53 |
| 3.11.1 RDM READ DATA RAM DATA CHARACTER . . | 3-54 |
| 3.11.2 RD _n READ DATA RAM STATUS CHARACTER . | 3-54 |
| 3.11.3 RDR READ ROM PORT | 3-55 |
| 3.11.4 WRM WRITE DATA RAM CHARACTER | 3-56 |
| 3.11.5 WR _n WRITE DATA RAM STATUS CHARACTER | 3-57 |
| 3.11.6 WMP WRITE RAM PORT | 3-58 |

-- TABLE OF CONTENTS -- (Continued)

The INTELLEC 4 Microcomputer System
Programming Manual

| | <u>PAGE No.</u> |
|--|-----------------|
| 3.11.7 WRR WRITE ROM PORT | 3-59 |
| 3.11.8 ADM ADD DATA RAM TO ACCUMULATOR WITH CARRY | 3-60 |
| 3.11.9 SBM SUBTRACT DATA RAM FROM MEMORY WITH BORROW | 3-61 |
| 3.11.10 WPM WRITE PROGRAM RAM | 3-62 |
| 3.12 PSEUDO INSTRUCTION | 3-65 |
| 3.12.1 EQUATE FUNCTION | 3-65 |
| 3.12.2 ORIGIN FUNCTION | 3-66 |
| 4.0 PROGRAMMING TECHNIQUES | 4-1 |
| 4.1 CROSSING PAGE BOUNDARIES | 4-1 |
| 4.2 SUBROUTINES | 4-3 |
| 4.3 BRANCH TABLE PSEUDOSUBROUTINE | 4-5 |
| 4.4 LOGICAL OPERATIONS | 4-8 |
| 4.4.1 LOGICAL "AND" | 4-8 |
| 4.4.2 LOGICAL "OR" | 4-9 |
| 4.4.3 LOGICAL "XOR" EXCLUSIVE-OR | 4-11 |
| 4.5 MULTI-DIGIT ADDITION | 4-13 |
| 4.6 MULTI-DIGIT SUBTRACTION | 4-15 |
| 4.7 DECIMAL ADDITION | 4-18 |
| 4.8 DECIMAL SUBTRACTION | 4-20 |
| 4.9 FLOATING POINT NUMBERS | 4-24 |
| APPENDIX "A" INSTRUCTION SUMMARY | A-1 |
| APPENDIX "B" INSTRUCTION MACHINE CODES | B-1 |
| APPENDIX "C" ASCII TABLES | C-1 |
| APPENDIX "D" BINARY-DECIMAL-HEXADECIMAL CONVERSION TABLES | D-1 |

-- TERMS AND ABBREVIATIONS --

| | |
|----------------|--|
| Address | A 12 bit number assigned to a read-only-memory or program random-access memory location corresponding to its sequential position. |
| Bit | The smallest unit of information which can be represented. (A bit may be in one of two states, 0 or 1). |
| Byte | A group of 8 contiguous bits occupying a single memory location. |
| Character | A group of 4 contiguous bits of data. |
| Instruction | The smallest single operation that the computer can be directed to execute. |
| Object Program | A program which can be loaded directly into the computer's memory and which requires no alteration before execution. An object program is usually on paper tape, and is produced by assembling a source program. Instructions are represented by binary machine code in an object program. |
| Program | A sequence of instructions which, taken as a group, allow the computer to accomplish a desired task. |
| Source Program | A program which is readable by a programmer but which must be transformed into object program format before it can be loaded into the computer and executed. Instructions in an assembly language source program are represented by their assembly language mnemonic. |
| System Program | A program written to help in the process of creating user programs. |
| User Program | A program written by the user to make the computer perform any desired task. |
| nnnB | nnn represents a number in binary format. |
| nnnH | nnn represents a number in hexadecimal format. |

Note: All numbers in this manual are assumed to be decimal unless otherwise specified.

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | R | P | 0 |
|---|---|---|---|---|---|---|

A representation of a byte in memory. Bits which are fixed as 0 or 1 are indicated by 0 or 1; bits which may be either 0 or 1 in different circumstances are represented by letters; thus RP represents a three-bit field which contains one of the eight possible combinations of zeroes and ones.

1.0 INTRODUCTION

This manual has been written to help the reader program the INTEL 4004 microcomputer in assembly language, and to show how it is economical and practical to do so. Accordingly, this manual assumes that the reader has a good understanding of logic, but may be unfamiliar with programming concepts.

For those readers who do understand programming concepts, several features of the INTEL 4004 microcomputer are described below. They include:

- 4 bit parallel CPU on a single chip.
- 46 instructions, including conditional branching, subroutine capability, and binary and decimal arithmetic modes.
- Direct addressing for 32,768 bits of read-only memory, 5120 bits of data random-access memory and 32,768 bits of program random-access memory.
- Sixteen 4-bit index registers and a three 12-bit register stack.

INTEL 4004 microcomputer users will have widely differing programming needs. Some users may wish to write a few short programs, while other users may have extensive programming requirements.

For the user with limited programming needs, two system programs resident on the INTELLEC 4 (Intel's development system for the MCS-4 microcomputer) are provided; they are an Assembler and a System Monitor. Use of the INTELLEC 4 and its system programs is described in the INTELLEC 4 Operator's Manual.

For the user with extensive programming needs, cross assemblers are available which allow programs to be generated on a computer having a FORTRAN compiler whose word size is 32 bits or greater, limiting INTELLEC 4 use to final checkout of programs only.

2.0 COMPUTER ORGANIZATION

This section provides the programmer with a functional overview of the 4004 computer. Information is presented in this section at a level that provides a programmer with necessary background in order to write efficient programs.

To the programmer, the computer is represented as consisting of the following parts:

- (1) Sixteen working registers which serve as temporary storage for data, and provide the means for addressing memory.
- (2) The accumulator, in which data is processed.
- (3) Memories, which may hold program instructions or data (or sometimes both), and which must be addressed location by location in order to access stored information.
- (4) The stack, which is a device used to facilitate execution of subroutines, as described later in this section.
- (5) Input/Output, which is the interface between a program and the outside world.

2.1 WORKING REGISTERS (INDEX REGISTERS)

The 4004 provides the programmer with sixteen 4-bit registers. These may be referenced individually by the integers 0 through 15, or as 8 register pairs by the even integers from 0 through 14. The register pairs may also be referenced by the symbols 0P through 7P. These correspondences are shown as follows:

INDIVIDUAL REGISTER REFERENCE

| | | | | |
|-------------|---|----|----|---------------|
| Register 0 | → | 0 | 1 | ← Register 1 |
| Register 2 | → | 2 | 3 | ← Register 3 |
| Register 4 | → | 4 | 5 | ← Register 5 |
| Register 6 | → | 6 | 7 | ← Register 7 |
| Register 8 | → | 8 | 9 | ← Register 9 |
| Register 10 | → | 10 | 11 | ← Register 11 |
| Register 12 | → | 12 | 13 | ← Register 13 |
| Register 14 | → | 14 | 15 | ← Register 15 |

REGISTER PAIR REFERENCE

| | | | |
|------------------------|---|----|----|
| Register Pair 0 or 0P | → | 0 | 1 |
| Register Pair 2 or 1P | → | 2 | 3 |
| Register Pair 4 or 2P | → | 4 | 5 |
| Register Pair 6 or 3P | → | 6 | 7 |
| Register Pair 8 or 4P | → | 8 | 9 |
| Register Pair 10 or 5P | → | 10 | 11 |
| Register Pair 12 or 6P | → | 12 | 13 |
| Register Pair 14 or 7P | → | 14 | 15 |

2.2 THE ACCUMULATOR

The accumulator is a special 4-bit register in which data may be transformed by program instructions.

2.3 MEMORIES

The 4004 can be used with three different types of memory which have different organizations and characteristics, and are used for different purposes. These are described below.

2.3.1 READ-ONLY MEMORY

Read-only memory (ROM) is used for storing program instructions and constant data which is never changed by the program. This is because the program can read locations in ROM, but can never alter (write) ROM locations.

ROM may be visualized as in Figure 2-1, as a sequence of bytes, each of which may store 8 bits (two hexadecimal digits). Up to 4096 bytes of ROM may be present, and an individual byte is addressed by its sequential number between 0 and 4095.

ROM is further divided into pages, each of which contains 256 bytes. Thus locations 0 through 255 comprise page 0 of ROM, location 256 through 511 comprise page 1 and so on.

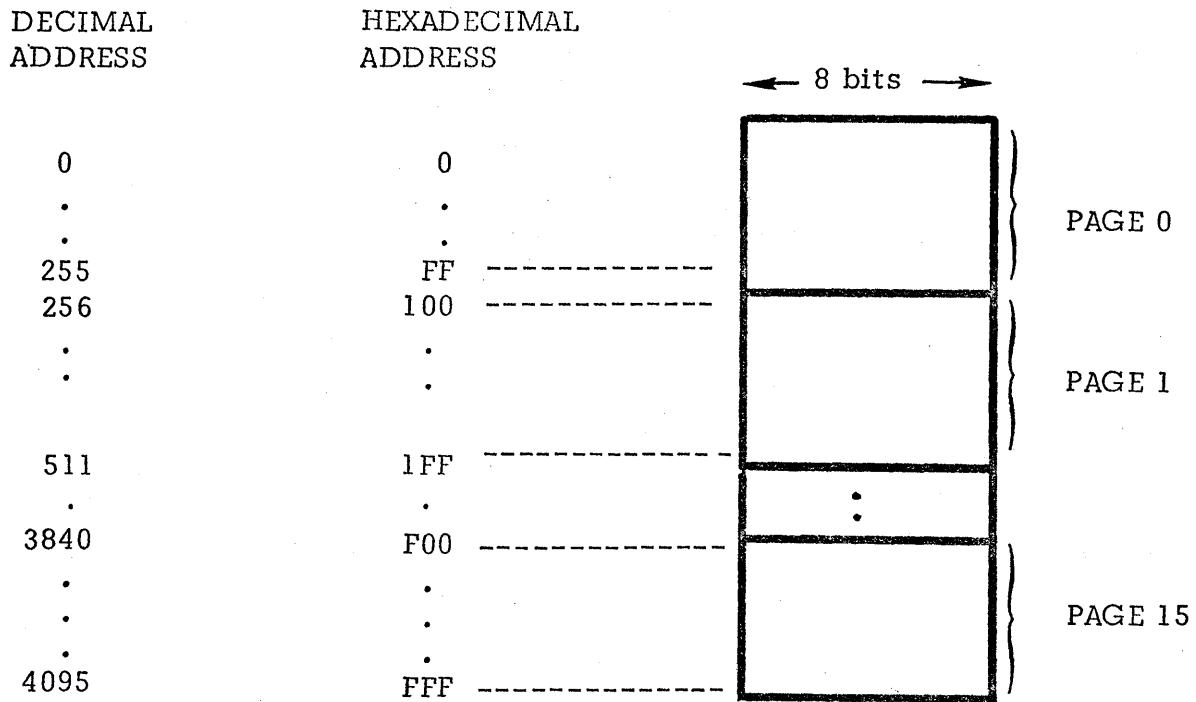


FIGURE 2-1.
ROM ORGANIZATION

As described in Section 3, certain instructions function differently when located in the last byte (or bytes) of a page than when located elsewhere.

2.3.2 PROGRAM RANDOM ACCESS MEMORY

Program random access memory (RAM) is organized exactly like ROM. 4096 locations are always available, which are used to hold program instructions or data. Unlike ROM, however, program RAM locations can be altered by program instructions.

2.3.3 DATA RANDOM ACCESS MEMORY

As its name implies, data random access memory (DATA RAM) is used for the temporary storage of data by programs.

Figures 2-2 and 2-3 show how this memory is addressed:

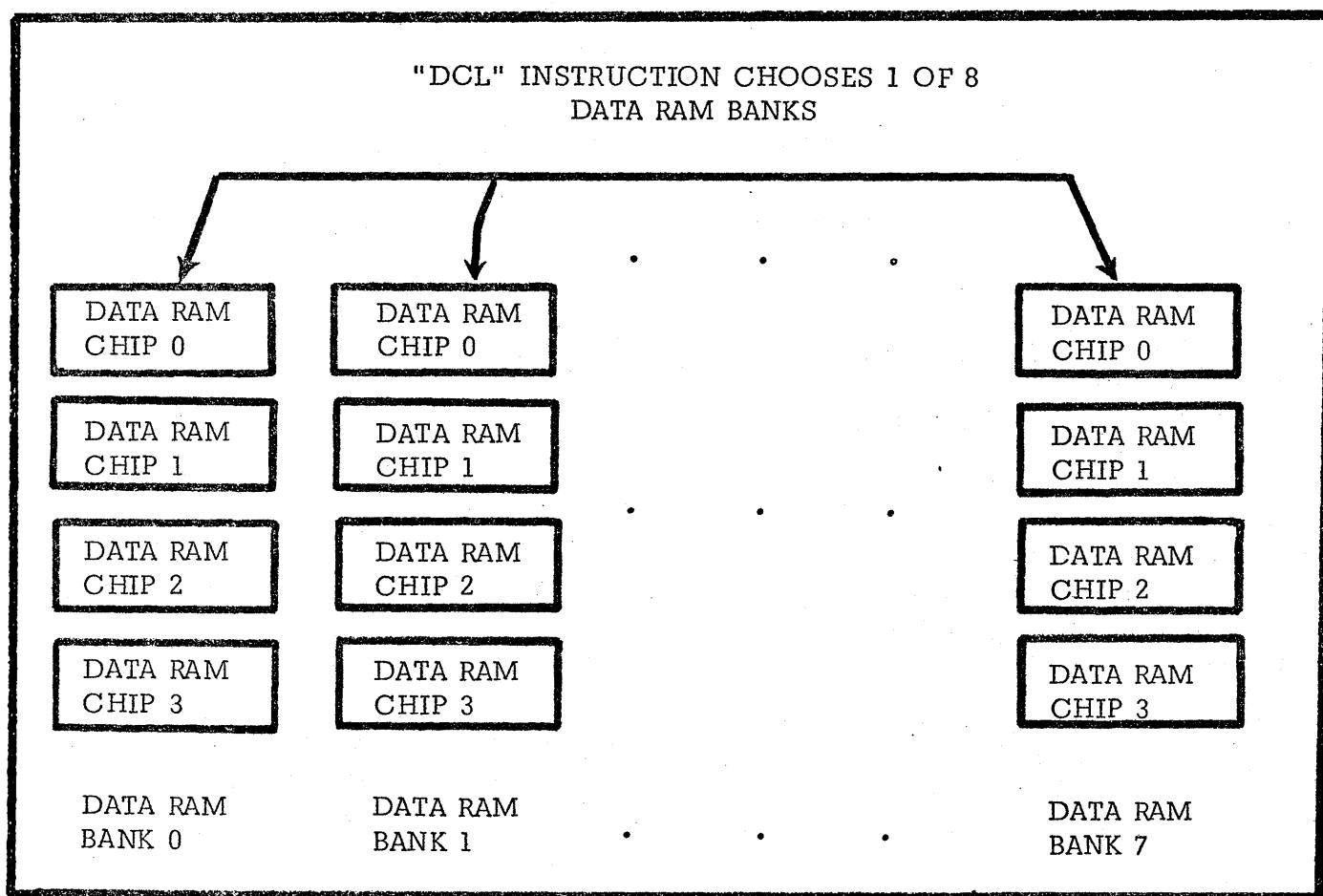


FIGURE 2-2.
DATA RAM BANK ORGANIZATION.

| | | |
|--|---|--|
| Decimal Addresses Hexadecimal Addresses | 16 Directly Addressable 4-bit characters per DATA RAM Register. | 4 Specially Addressable 4 bit status characters per DATA RAM Register. |
|--|---|--|

| | |
|-------|-------|
| 0-15 | 00-0F |
| 16-31 | 10-1F |
| 32-47 | 20-2F |
| 48-63 | 30-3F |

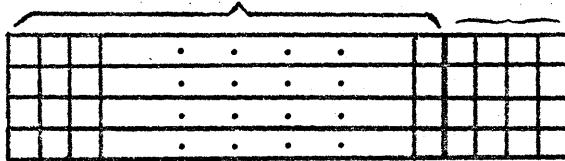


FIGURE 2-3.
DATA RAM CHIP 0 ORGANIZATION

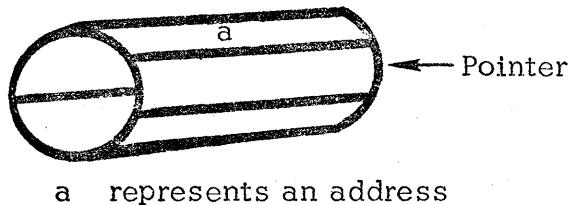
In order to address a 4 bit character of DATA RAM, the programmer first uses a "DCL" instruction as described in Section 3.10.1 to choose one of a maximum of eight DATA RAM BANKS. An eight bit address is then sent via an "SRC" instruction as described in Section 3.10.2 which chooses one of four DATA RAM CHIPS within the DATA RAM BANK, one of four 16-character DATA RAM REGISTERS within the DATA RAM CHIP, and one of 16 4-bit characters within the DATA RAM REGISTER. Within any particular DATA RAM BANK, then, addresses 0-63 indicate which of the 64 directly addressable characters of DATA RAM CHIP 0 is to be addressed. Addresses 64-127 correspond to the characters of CHIP 1, addresses 128-191 correspond to CHIP 2, and addresses 192-255 correspond to CHIP 3.

In addition, each DATA RAM REGISTER has four 4-bit STATUS characters associated with it. These status characters may be read and written like the data characters, but are accessed by special instructions as described in Section 3.

2.4 THE STACK

The stack consists of three 12-bit registers used to hold addresses of program instructions. Since programs are always run in ROM or program RAM, the stack registers will always refer to ROM locations or program RAM locations.

Stack operations consist of writing an address to the stack, and reading an address from the stack. In order to understand these operations, it may be helpful to visualize the stack as three registers on the surface of a cylinder, as shown below:



a represents an address

Each stack register is adjacent to the other two stack registers. The 4004 keeps a pointer to the next stack register available.

Writing An Address To The Stack:

To perform a stack write operation;

- (1) The address is written into the register indicated by the pointer.
- (2) The pointer is advanced to the next sequential register.

Any register may be used to hold the first address written to the stack. More than three addresses may be written to the stack; however, this will cause a corresponding number of previously stored addresses to be overwritten and lost. This is illustrated in Figure 2-4.

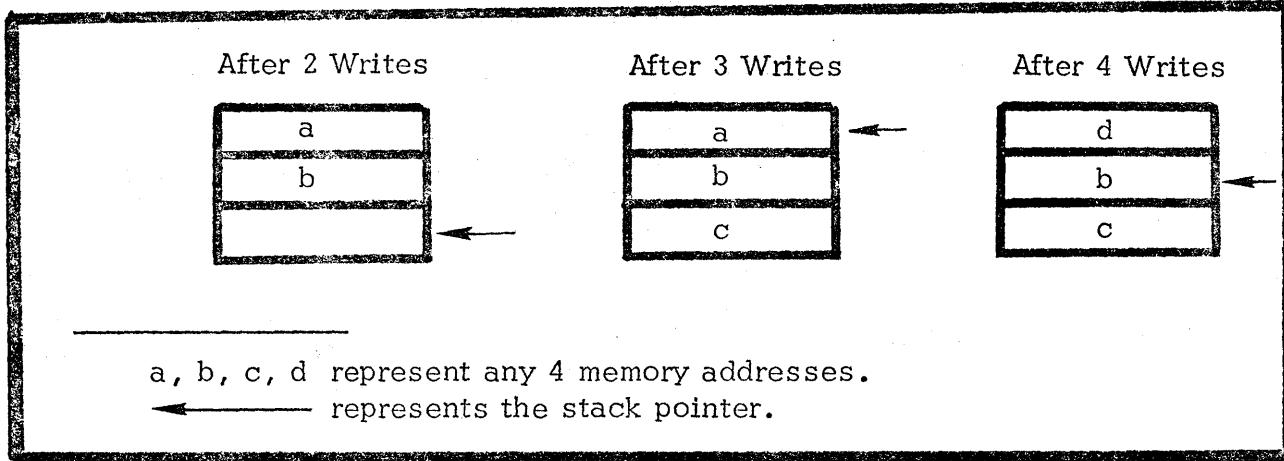


FIGURE 2-4.
STACK WRITE OPERATIONS

Storing the fourth address (d) overwrites the first address stored (a).

Reading An Address From The Stack:

To perform a stack read operation;

- (1) The pointer is backed up one register.
- (2) The memory address indicated by the pointer is read.

The address read remains in the stack undisturbed. Thus, if 4 addresses are written to the stack and then three reads are performed, the stack will appear as in Figure 2-5.

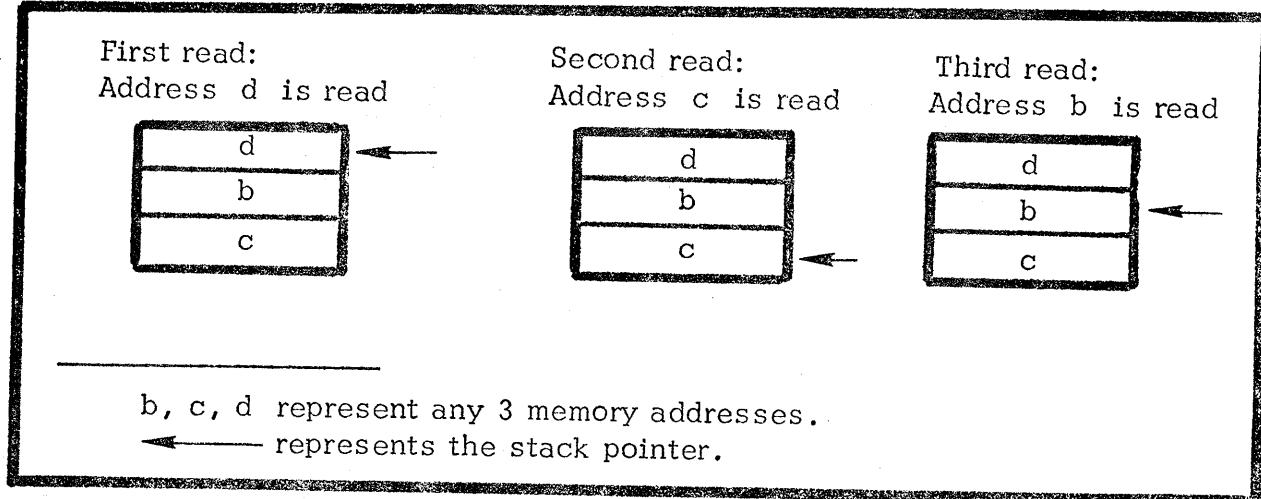


FIGURE 2-5.
STACK READ OPERATIONS.

Section 2.7.7 describes how the stack is used by programs.

2.5 INPUT/OUTPUT

Programs communicate with the outside world via 4-bit input or output ports. The operation of these ports is controlled by special I/O instructions described in Section 3.

These ports are physically located on the same devices which hold ROMs and DATA RAMs; therefore, they are referred to as ROM ports or RAM ports. These are totally separate from the instruction or data locations provided in ROM or RAM, and should not be confused with them. The ports associated with RAMs may be used only for output.

2.6 COMPUTER PROGRAM REPRESENTATION IN MEMORY

A computer program consists of a sequence of instructions. Each instruction performs an elementary operation such as the movement of data, an arithmetic operation on data, or a change in instruction execution sequence. Instructions are described individually in Section 3.

A program will be stored in Read-Only Memory or Program Random Access Memory. It will appear as a sequence of hexadecimal digits which represent the instructions of the program. The memory address of the instruction being executed is recorded in a 12-bit register called the Program Counter, and thus it is possible to track a program as it is being executed. After each instruction is executed, the program counter is advanced to the address of the next instruction. Program execution proceeds sequentially unless a transfer-of-control instruction (jump or skip) is executed, which causes the program counter to be set to a specified address. Execution then continues sequentially from this new address in memory.

Upon examining the contents of a ROM or program RAM memory location, there is no way of telling whether a byte contains an encoded instruction or constant data. For example, the hexadecimal code F2 has been selected to represent the instruction IAC (increment accumulator). Thus, the hex value F2 stored in a memory byte could represent either the instruction IAC or the hex data value F2.

It is up to the programmer to insure that data is not misinterpreted as an instruction code, but this is simply done as follows:

Every program has a starting memory address, which is the memory address of the location holding the first instruction to be executed. Just before the first instruction is executed, the program counter will automatically be set to this address, and this procedure will be repeated for every instruction in the program. 4004 instructions may require 8 or 16 bits for their encoding; in each case the program counter is set to the corresponding address as shown in Figure 2-6.

| MEMORY ADDRESS (Hexadecimal) | INSTRUCTION NUMBER | PROGRAM COUNTER CONTENTS (Hexadecimal) |
|------------------------------------|-----------------------|--|
| 13A | 1 | 13A |
| 13B | 2 | 13B |
| 13C | | |
| 13D | 3 | 13D |
| 13E | | |
| 13F | 4 | 13F |
| 140 | 5 | 140 |
| 141 | | |

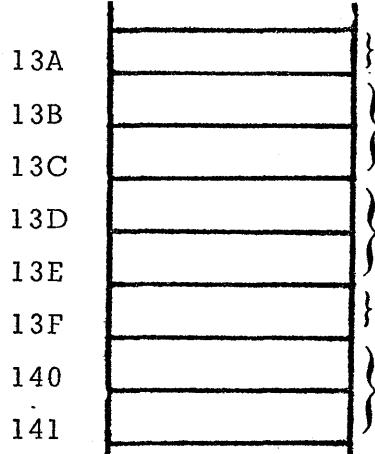


FIGURE 2-6.

PROGRAM COUNTER CONTENTS AS INSTRUCTIONS ARE EXECUTED.

In order to avoid errors, the programmer must be sure that a byte of constant data does not follow an instruction when another instruction is expected. Referring to Figure 2-6, an instruction is expected in location 13FH, since instruction 4 is to be executed after instruction 3. If location 13FH held constant data, the program would not execute correctly. Therefore, when writing a program, do not place constant data in between adjacent instructions that are to be executed consecutively.

A class of instructions (referred to as transfer-of-control instructions) cause program execution to branch to an instruction other than the next sequential instruction. The memory address specified by the transfer of control instruction must be the address of another instruction; if it is the address of a memory location holding data, the program will not execute correctly. For example, referring to Figure 2-6, suppose instruction 2 specifies a jump to location 140H, and instructions 3 and 4 were replaced by data. Then following execution of instruction 2, the program counter would be set to 140H and the program would execute correctly. But if, in error, instruction 2 were to specify a jump to 13EH, an error would result since this location now holds data. Even if instructions 3 and 4 were not altered, a jump to location 13EH would cause an error, since this is not the first byte of the instruction.

Upon reading Section 3, you will see that it is easy to avoid writing an assembly language program with jump instructions which have erroneous memory addresses. Information on this subject is given here rather to help the programmer who is debugging programs by entering hexadecimal codes directly into program RAM (Programs usually exist in ROM, and therefore cannot be altered in this manner).

2.7 MEMORY ADDRESSING

By now it will have become apparent that addressing specific memory bytes constitutes an important part of any computer program. There are a number of ways in which this can be done, as described in the following subsections.

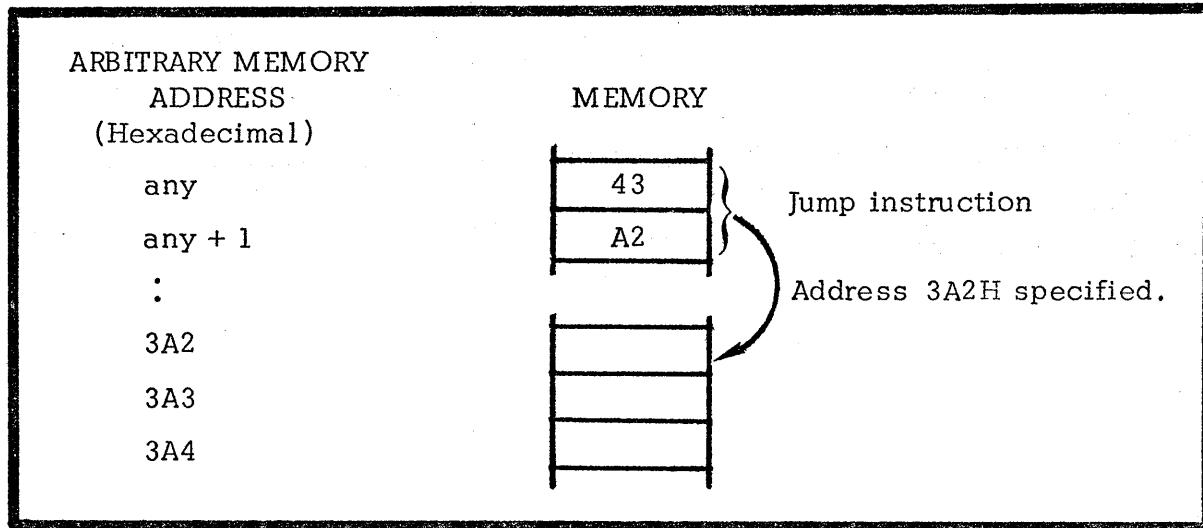
2.7.1 DIRECT ADDRESSING

With direct addressing, as the name implies, an instruction provides an exact memory address. The following instruction provides an example of direct addressing:

"Jump to location 3A2H"

This instruction is represented by 4 hexadecimal digits in ROM or program RAM. The first digit is a 4, signifying a jump instruction, while the final 3 digits specify the address.

This instruction would appear in memory as follows:



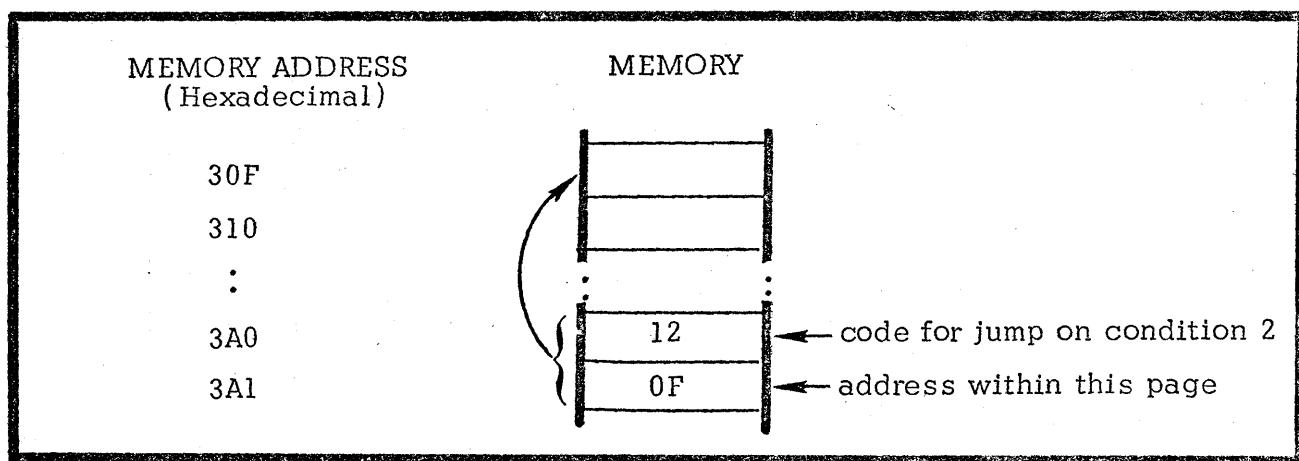
2.7.2 SAME PAGE ADDRESSING

Some instructions supply two hexadecimal digits which replace the lowest 8 bits of the program counter, addressing a ROM or program RAM location on the same page as the instruction being executed. (Two addresses are on the same page if the highest order hexadecimal digit of their addresses are equal. See Section 2.3.1).

The following instruction provides an example of same page addressing:

"Jump on condition 2 to location 3BH of this page."

This instruction would appear in memory as follows:



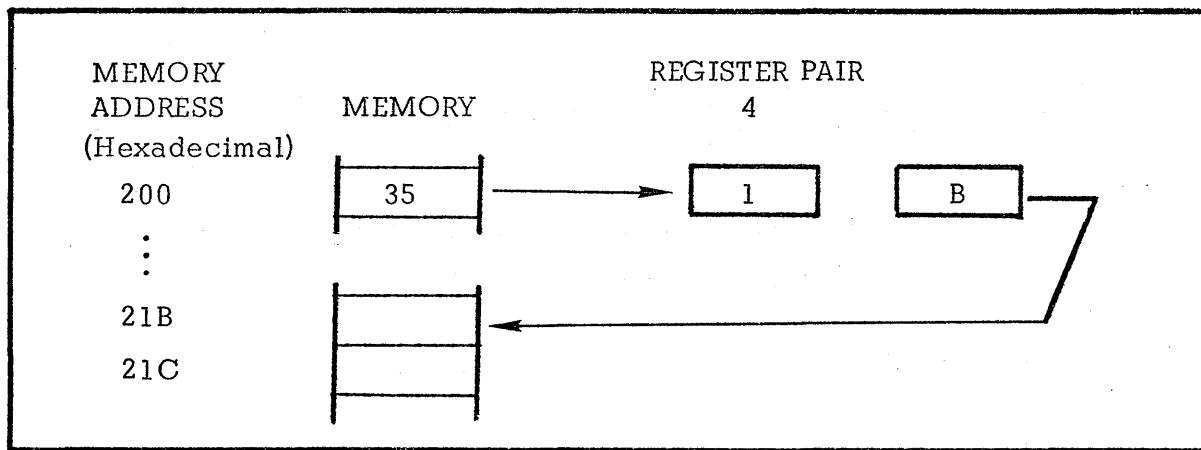
The identical encoding 120FH if located at location 501H, would cause a jump to memory address 50FH.

2.7.3 INDIRECT ADDRESSING

With indirect addressing, an instruction specifies a register pair which in turn holds an 8 bit value used for same page addressing (Section 2.7.2). Suppose that registers 4 and 5 hold the 4-bit hexadecimal numbers 1 and B, respectively. Then the instruction:

"Jump indirect to contents of register pair 4"

would appear as follows:



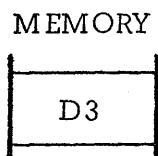
The 3 indicates a "jump indirect" instruction; the 5 indicates that the address indicated on this page is held in register pair 4. If register pair 4 had held the hex numbers 3 and C, a jump to location 23CH would have occurred.

2.7.4 IMMEDIATE ADDRESSING

An immediate instruction is one that provides its own data. The following is an example of immediate addressing:

"Load the accumulator with the hexadecimal number 3".

This instruction would be coded in memory as follows:



The digit D signifies a "load accumulator immediate" instruction; the digit 3 is the number to be loaded.

2.7.5 PROGRAM RAM ADDRESSING

When a program stores an 8 bit value into a program RAM location a special sequence of instructions must be used as described in Section 3.11.10 (the WPM instruction).

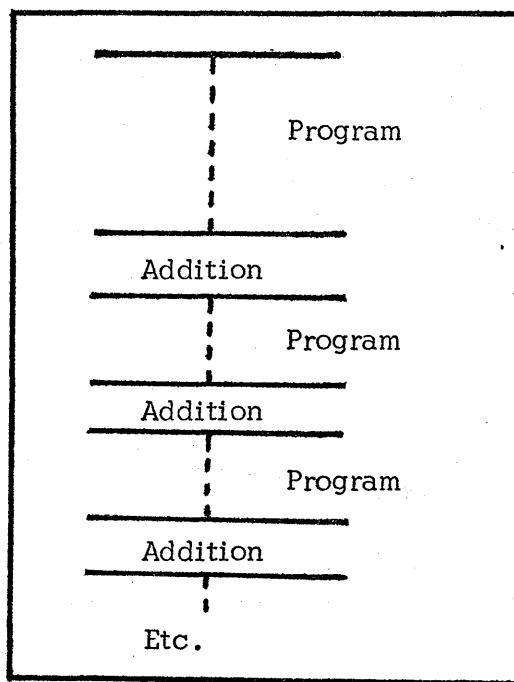
2.7.6 DATA RAM ADDRESSING

To address a location in DATA RAM, the DCL and SRC instructions must be used as described in Sections 2.3.3, 3.10.1, and 3.10.2. When the DCL has chosen a specific DATA RAM bank, the address of the specific character is held in a register pair accessed by the SRC instruction.

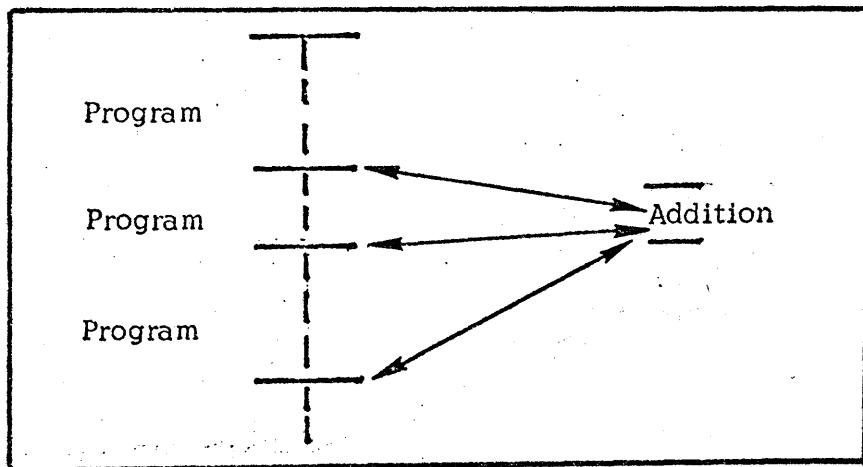
2.7.7 SUBROUTINES AND USE OF THE STACK FOR ADDRESSING

Before understanding the purpose or effectiveness of the stack, it is necessary to understand the concept of a subroutine.

Consider a frequently used operation such as addition. The 4004 provides instructions to add one character of data to another, but what if you wish to add numbers outside the range of 0 to 15 (the range of one character)? Such addition will require a number of instructions to be executed in sequence. It is quite possible that this addition routine may be required many times within one program; to repeat the identical code every time it is needed is possible, but very wasteful of memory:

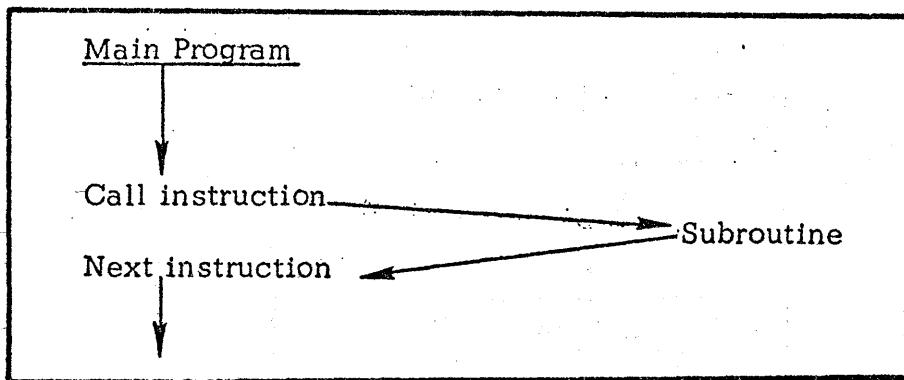


A more efficient means of accessing the addition routine would be to store it once, and find a way of accessing it when needed:



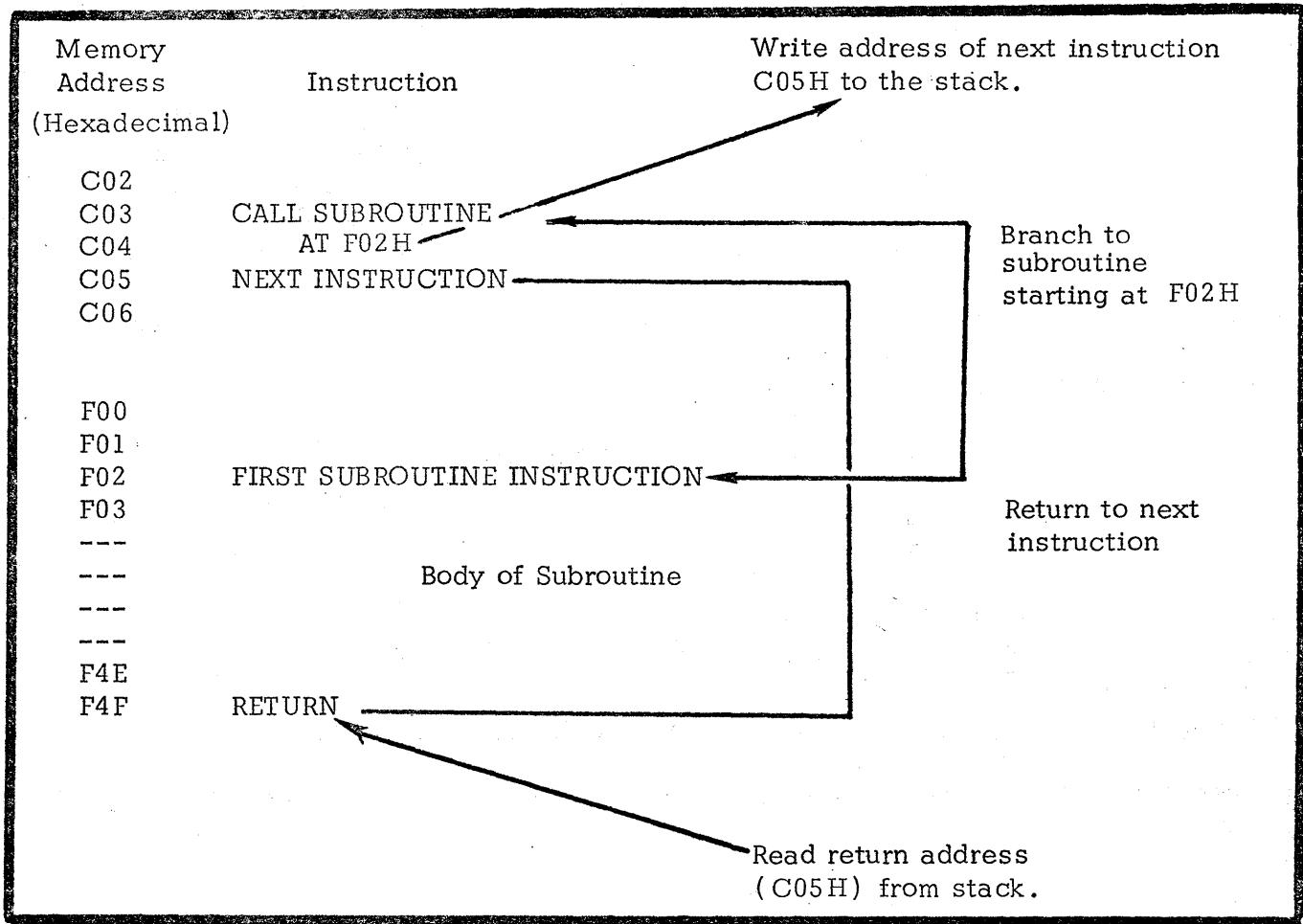
A frequently accessed routine such as the addition above is called a subroutine, and the 4004 provides instructions that call subroutines and return from subroutines.

When a subroutine is executed, the sequence of events may be depicted as follows:



The arrows indicate the execution sequence.

When the "Call" instruction is executed, the address of the "next" instruction is written to the stack (see Section 2.4), and the subroutine is executed. The last executed instruction of a subroutine will always be a special "Return Instruction", which reads an address from the stack into the program counter, and thus causes program execution to continue at the "Next" instruction as illustrated on the next page.



Since the stack provides three registers, subroutines may be nested up to three deep; for example, the addition subroutine could itself call some other subroutine, and so on. An examination of the sequence of write and read stack operations will show that the return path will always be identical to the call path, even if the same subroutine is called at more than one level; however, an attempt to nest subroutines to a depth of more than 3 will cause the program to fail, since some addresses will have been overwritten.

2.8 CARRY BIT

To make programming easier, a carry bit is provided by the 4004 to reflect the results of data operations. The descriptions of individual instructions in Section 3 specify which instructions affect the carry bit and whether the execution of the instruction is dependent in any way on the prior status of the carry bit. The carry bit is "set" if 1 and "reset" if 0.

Certain data operations can cause an overflow out of the high-order 3-bit. For example, addition of two hexadecimal digits can give rise to an answer that does not fit in one digit:

| 3 | 2 | 1 | 0 | Bit Number |
|----|-------|---|---|------------|
| A | 1 | 0 | 1 | 0 |
| +7 | 0 | 1 | 1 | 1 |
| | <hr/> | | | |
| | 1 | 0 | 0 | 1 |
| | | | | Carry = 1 |

An operation that results in a carry out of bit 3 will set the carry bit.

An operation that could have resulted in a carry out of bit 3 but did not will reset the carry bit.

3.0 THE 4004 INSTRUCTION SET

This section describes the 4004 assembly language instruction set.

For the reader who understands assembly language, Appendix A provides a complete summary of the 4004 instructions.

For the reader who is not completely familiar with assembly language, this section describes individual instructions with examples and machine code equivalents.

3.1 ASSEMBLY LANGUAGE

3.1.1 HOW ASSEMBLY LANGUAGE IS USED

Upon examining the contents of read-only memory or program random-access memory, a program would appear as a sequence of hexadecimal digits which are interpreted by the machine as instruction codes, addresses, or constant data. It is possible to write a program as a sequence of digits (just as they appear in memory), but that is slow and expensive. For example, several instructions reference memory to address another instruction:

| HEXADECIMAL MEMORY ADDRESS | |
|-------------------------------|----|
| 332 | F0 |
| 333 | 43 |
| 334 | 56 |
| . | . |
| . | . |
| 354 | 20 |
| 355 | FF |
| 356 | 60 |

The above program operates as follows:

Byte 332H specifies that the accumulator and carry bit are to be cleared.

Bytes 333H and 334H specify that program execution is to continue at location 356H.

Byte 356H specifies that register 0 is to be incremented.

Now suppose that an error discovered in the program logic necessitates placing a new instruction after byte 332H. Program code would have to change as follows:

| <u>HEXADECIMAL MEMORY ADDRESS</u> | <u>OLD CODE</u> | <u>NEW CODE</u> |
|---------------------------------------|-----------------|-----------------|
| 332 | F0 | F0 |
| 333 | 43 | New Instruction |
| 334 | 56 | 43 |
| 335 | | 57 |
| . | . | . |
| 354 | 20 | |
| 355 | FF | 20 |
| 356 | 60 | FF |
| 357 | | 60 |

Note that many instructions have been moved and as a result some must be changed to reflect the new addresses of instructions. The potential for making mistakes is very high and is aggravated by the complete unreadability of the program.

Writing programs in assembly language is the first and most significant step towards economical programming; it provides a readable notation for instructions, and separates the programmer from a need to know or specify absolute memory addresses.

Assembly language programs are written as a sequence of instructions which are converted to executable hexadecimal code by a special program called an ASSEMBLER.

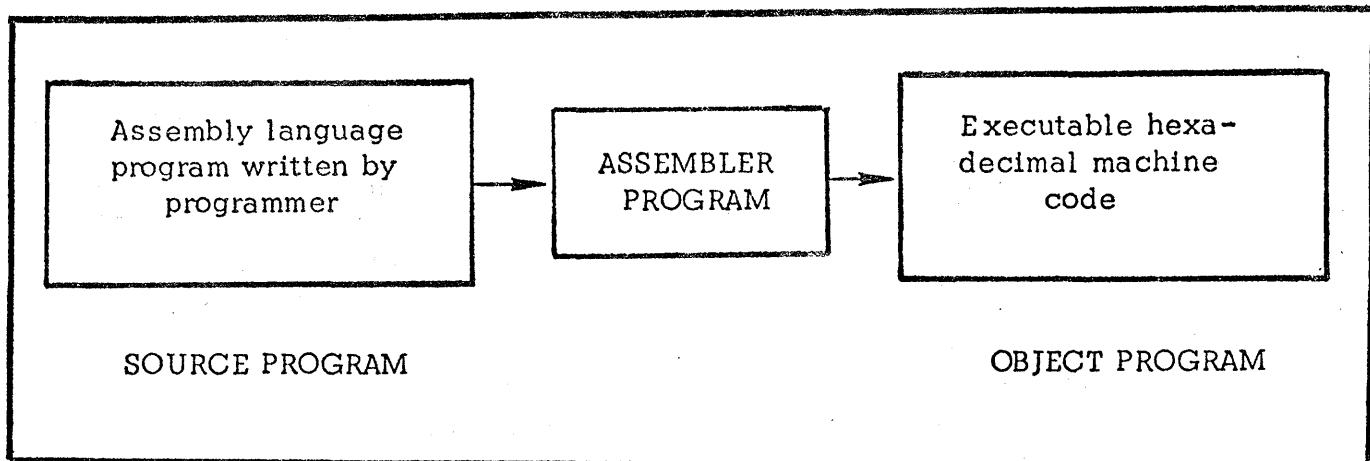


FIGURE 3-1.

ASSEMBLER PROGRAM CONVERTS ASSEMBLY LANGUAGE SOURCE PROGRAM TO HEXADECIMAL OBJECT PROGRAM

As illustrated in Figure 3-1, the assembly language program generated by a programmer is called a SOURCE PROGRAM. The assembler converts the SOURCE PROGRAM into an equivalent OBJECT PROGRAM, which consists of a sequence of hexadecimal codes that can be loaded into ROM or program RAM and executed.

For example:

| <u>Source Program</u> | is converted by the Assembler to | <u>One Possible Version of the Object Program</u> |
|-----------------------|-------------------------------------|---|
| NOW, CLB | | F0 |
| JUN NXT | | 4356 |
| : | | : |
| FIM 0 255 | | 20FF |
| NXT, INC 0 | | 60 |

Now if a new instruction must be added, only one change is required. Even the reader who is not yet familiar with assembly language will see how simple the addition is:

```
NOW,    CLB
       ( New instruction inserted here )
JUN     NXT
      :
FIM     0 255
NXT,    INC   0
```

The assembler takes care of the fact that a new instruction will shift the rest of the program in memory.

3.1.2 STATEMENT MNEMONICS

Assembly language instructions must adhere to a fixed set of rules as described in this section. An instruction has four separate and distinct parts or FIELDS.

Field 1 is the LABEL field. It is the instruction location's label or name, and it is used to reference the instruction.

Field 2 is the CODE field. It defines the operation that is to be performed by the instruction.

Field 3 is the OPERAND field. It provides any address or data information needed by the CODE field.

Field 4 is the COMMENT field. It is present for the programmer's convenience and is ignored by the assembler. The programmer uses comment fields to describe the operation and thus make the program more readable.

The assembler uses free fields; that is, any number of blanks may separate fields.

Before describing each field in detail, here are some general examples:

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> | <u>COMMENT</u> |
|--------------|-------------|----------------|--|
| CMI, | CLB | | / Clear accumulator and carry. |
| LAB, | INC | 3 | / Increment register 3. |
| | JUN | CMI | / Jump to instruction CMI. |
| FCH, | FIM | OP 255 | / Load hex FF (decimal 255) into / register pair 0. |

3.1.3 LABEL FIELD

This is an optional field. If present, the first character of a label must be a letter of the alphabet. The remaining characters may be letters or decimal digits. The label field must end with a comma, immediately following the last character of the label. Labels may be any length, but should be unique in the first three characters; the assembler cannot always distinguish between labels whose first three characters are identical. If no label is present, at least one blank must begin the line.

Here are some examples of valid label fields:

CM0,
NUL,
EGO,

Here are some invalid labels:

4GE, does not begin with a letter.
AGE valid label, but label field does not end with a comma.
A/A, contains invalid character.

The following label has more than 3 characters:

STROB

The assembler may not be able to differentiate this label from others beginning with the characters STR.

Since labels serve as instruction addresses, they cannot be duplicated. For example, the sequence:

| | | |
|------|-----|-----|
| NOW, | JUN | NXT |
| | --- | |
| | --- | |
| NXT, | INC | 2 |
| | --- | |
| | --- | |
| NXT, | CLB | |

is ambiguous; the assembler cannot determine which NXT address is referenced by the JUN instruction.

3.1.4 CODE FIELD

This field contains a code which identifies the machine operation (add, subtract, jump, etc.) to be performed: hence the term operation code or op-code. The instructions described in Sections 3.3 thru 3.11, are each identified by a mnemonic label which must appear in the code field. For example, since the "jump unconditionally" instruction is identified by the letters "JUN", these letters must appear in the code field to identify the instruction as "jump unconditionally".

There must be at least one space following the code field. Thus:

LAB, JUN AWY

is legal, but:

LAB, JUNAWY

is illegal.

3.1.5 OPERAND FIELD

This field contains information used in conjunction with the code field to define precisely the operation to be performed by the instruction. Depending upon the code field, the operand field may be absent or may consist of one item or two items separated by blanks.

There are five types of information [(a) through (e) below] that may be requested as items of an operand field, and the information may be specified in five ways [(1) through (5) below].

The five ways of specifying information are as follows:

- (1) A decimal number.

Example:

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> | <u>COMMENT</u> |
|--------------|-------------|----------------|--|
| ABC, | LDM | 14 | /Load accumulator with decimal /14 (1110 binary). |

- (2) The current program counter. This is specified as the character '*' and is equal to the address of the first byte of the current instruction.

Example:

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> |
|--------------|-------------|----------------|
| GO, | JUN | *+6 |

If the instruction above is being assembled at location 213, it will cause program control to be transferred to address 219.

- (3) Labels that have been assigned a decimal number by the assembler.
 (See Section 3.12.1 for the equate procedure).

Example:

Suppose label VAL has been equated to the number 42, and ZER has been equated to the number 0. Then the following instructions all load register pair zero with the hexadecimal value 2A (decimal 42):

| LABEL | CODE | OPERAND |
|-------|------|---------|
| A1, | FIM | 0 42 |
| A2, | FIM | ZER 42 |
| A3 | FIM | ZER VAL |

- (4) Labels that appear in the label field of another instruction.

Example:

| LABEL | CODE | OPERAND | COMMENT |
|-------|------|---------|-------------------------------|
| LP1, | JUN | LP2 | / Jump to instruction at LP2. |
| | | --- | |
| | | --- | |
| LP2, | CMA | | |

- (5) Arithmetic expressions involving data types (1) to (4) above connected by the operators + (addition) and - (subtraction). These operators treat their arguments as 12-bit quantities, and generate 12-bit quantities as their result. If a value is generated which exceeds the number of bits available for it in an instruction, the value is truncated on the left.

For example, if VAL refers to hexadecimal address FFE, the instruction:

JUN VAL

is encoded as 4FFEH; a 4-bit operation code and 12 bit value. However, the instruction:

JUN VAL + 9

will be encoded as 4007H, where the value 1007H has been truncated on the left to 12 bits (three hex digits) giving a value of 007H.

Using some or all of the above data specifications, the following five types of information may be requested:

- (a) A register to serve as the source or destination in a data operation. Methods 1, 3, or 5 may be used to specify the register, but the specification must finally evaluate to one of the decimal numbers 0 to 15.

Example:

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> |
|--------------|-------------|----------------|
| I1, | INC | 4 |
| I2, | INC | R4 |
| I3, | INC | 16-12 |

Assuming label R4 has been equated to 4, all the above instructions will increment register 4.

- (b) A register pair to serve as the source or destination in a data operation. The specification must evaluate to one of the even decimal numbers from 0 through 14 (corresponding to register pair designators 0P through 7P).

Example:

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> |
|--------------|-------------|----------------|
| I1, | SRC | 1P |
| I2, | SRC | 2 |
| I3, | SRC | RG2 |

Assuming label RG2 has been equated to 2, all of the above instructions refer to register pair 1P (registers 2 and 3).

- (c) Immediate data, to be used directly as a data item.

Example:

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> | <u>COMMENT</u> |
|--------------|-------------|----------------|---|
| AC1, | LDM | DATA | /Load the accumulator with /the value of DATA. |

DATA could take any of the following forms:

19
12 + 72 - 3
VAL (where VAL has been equated to a number).

- (d) A 12 bit address, or the label of another location in memory.

Example:

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> | <u>COMMENT</u> |
|--------------|-------------|----------------|---|
| HR, | J UN | OVR | /Jump to instruction at OVR. |
| | J UN | 513 | /Jump to hex address 201 (decimal 513). |

- (e) A condition code for use by the JCN (jump on condition) instruction.
This must evaluate to a number from 0 to 15.

Example:

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> |
|--------------|-------------|----------------|
| | JCN | 4 LOC |
| | JCN | 2+2 LOC |

The above instructions cause program control to be transferred to address LOC if condition 4 (accumulator zero) is true.

3.1.6 COMMENT FIELD

The only rule governing this field is that it must begin with a slash (/). It is terminated by the end of the line.

A comment field may appear alone on a line:

```
LOC,      CLB      /This is a comment  
/This is a comment line
```

3.2 DATA STATEMENTS

This section describes ways in which data can be specified in and interpreted by a program. Any 4 bit character in DATA RAM contains one of the 16 possible combinations of zeros and ones.

Arithmetic instructions assume that the DATA RAM characters upon which they operate are in a special format called "two's complement", and the operations performed on these bytes are called "two's complement arithmetic".

3.2.1 TWO'S COMPLEMENT

When a character is interpreted as a signed two's complement number, the low order 3 bits supply the magnitude of the number, while the high order bit is interpreted as the sign of the number (0 for positive numbers, 1 for negative).

The range of positive numbers that can be represented in signed two's complement notation is, therefore, from 0 to 7:

| | | |
|---|---|-----------|
| 0 | = | 0 0 0 0 B |
| 1 | = | 0 0 0 1 B |
| | : | |
| 6 | = | 0 1 1 0 B |
| 7 | = | 0 1 1 1 B |

To change the sign of a number represented in two's complement, the following rules are applied:

- (a) Invert each bit of the number (producing the so-called one's complement).
- (b) Add one to the result, ignoring any carry out of the high order bit position.

Example: Produce the two's complement representation of -6.
Following the rules above,

$$+6 = 0110 B$$

Invert each bit: 1 0 0 1 B
Add one : 1 0 1 0 B

Therefore, the two's complement representation of -6 is the hexadecimal number 'A'. (Note that the sign bit is set, indicating a negative number.)

Example: What is the value of the hexadecimal number 'C' interpreted as a signed two's complement number? The high order bit is set, indicating that this is a negative number. To obtain its value, again invert each bit and add one. (This is equivalent to subtracting one from the number and inverting each bit).

$$\begin{array}{ll} \text{CH} & = 1100 \text{ B} \\ \text{Invert each bit} & : 0011 \text{ B} \\ \text{Add one} & : 0100 \text{ B} \end{array}$$

Thus, the value of CH is -4.

The range of negative numbers that can be represented in signed two's complement notation is from -1 to -8.

$$\begin{array}{ll} -1 & = 1111 \text{ B} \\ -2 & = 1110 \text{ B} \\ & : \\ & : \\ -7 & = 1001 \text{ B} \\ -8 & = 1000 \text{ B} \end{array}$$

To perform the subtraction 6-3, the following operations are performed:

Take the two's complement of 3 = 1101 B

Add the result to the minuend:

$$\begin{array}{r} 6 = 0110 \text{ B} \\ +(-3) = \underline{1101 \text{ B}} \\ 0011 \text{ B} = 3, \text{ the correct answer} \end{array}$$

When a data character is interpreted as an unsigned two's complement number, its value is considered positive and in the range 0 to 15.

$$\begin{array}{l} 0 = 0000 \text{ B} \\ 1 = 0001 \text{ B} \\ \vdots \\ 7 = 0111 \text{ B} \\ 8 = 1000 \text{ B} \\ \vdots \\ 15 = 1111 \text{ B} \end{array}$$

Two's complement arithmetic is still valid. When performing an addition operation, the carry bit is set when the result is greater than 15. When performing subtraction, the carry bit is set when the result is positive. If the carry bit is reset, the result is negative and present in its two's complement form.

Example: Subtract 3 from 10 using unsigned two's complement arithmetic.

$$\begin{array}{r} 10 = 1010 \text{ B} \\ -3 = \underline{1101} \text{ B} \\ \hline 1 | 0111 \text{ B} \\ \text{Carry} = 1 \end{array}$$

Since the carry bit is set, the result (7) is correct and positive.

Example: Subtract 15 from 12 using unsigned two's complement arithmetic.

$$\begin{array}{r} 12 = 1100 \text{ B} \\ -15 = \underline{0001} \text{ B} \\ \hline 0 | 1101 \text{ B} = -3 \\ \text{Carry} = 0 \end{array}$$

Since the carry bit is reset, the result is negative and in its two's complement form.

WHY TWO'S COMPLEMENT?

Using two's complement notation for negative numbers, any subtraction problem becomes a sequence of bit inversions and additions. Therefore, fewer circuits are needed to perform subtraction.

3.2.2 CONSTANT DATA

Eight-bit data values can be assembled into ROM or program RAM locations by writing a blank code field and an operand field beginning with a positive number. If the operand is greater than 8 bits, it will be truncated on the left.

Example: Assume that label VAL has been equated to 14, and the label LOC appears on an instruction assembled at hexadecimal location 34B.

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> | <u>ASSEMBLED DATA</u> |
|--------------|-------------|----------------|-----------------------|
| C1, | | 0 + VAL | 0E |
| C2, | | 4095 | FF |
| C3, | | 0+LOC | 4B |

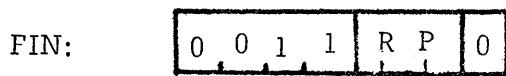
The following are invalid data statements:

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> | <u>COMMENT</u> |
|--------------|-------------|----------------|--------------------------------|
| C4, | | ABC | /Does not begin with a number. |
| C5, | | -18 | /Number is not positive. |

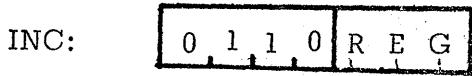
3.3 INDEX REGISTER INSTRUCTIONS

This section describes two instructions which involve index registers or register pairs.

These instructions occupy one byte as follows:



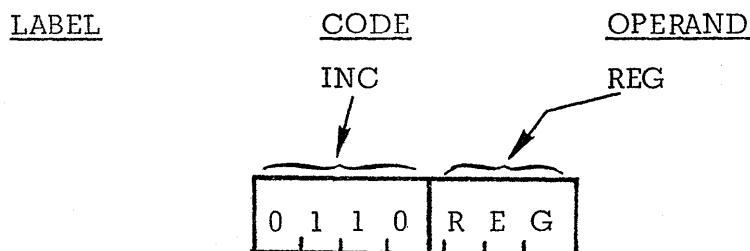
- 000 for register pair 0 or 0P.
001 for register pair 2 or 1P.
010 for register pair 4 or 2P.
011 for register pair 6 or 3P.
100 for register pair 8 or 4P.
101 for register pair 10 or 5P.
110 for register pair 12 or 6P.
111 for register pair 14 or 7P.



- 0000 for register 0 1000 for register 8
0001 for register 1 1001 for register 9
0010 for register 2 1010 for register 10
0011 for register 3 1011 for register 11
0100 for register 4 1100 for register 12
0101 for register 5 1101 for register 13
0110 for register 6 1110 for register 14
0111 for register 7 1111 for register 15

3.3.1 INC INCREMENT REGISTER

Format:



Description:

The index register indicated by REG is incremented by one.

The carry bit is not affected.

Example: If register 3 contains the number 6, the instruction:

INC 3

will cause register 3 to contain the number 7.

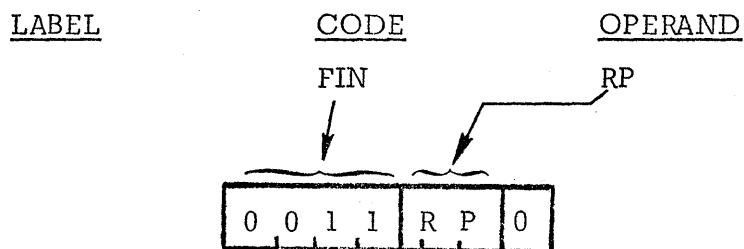
If register 8 contains the number 15 (1111 binary), the instruction:

INC 8

will cause register 8 to contain 0, leaving the carry bit unchanged.

3.3.2 FIN FETCH INDIRECT

Format:



Description:

The contents of registers 0 and 1 are concatenated to form the lower 8 bits of a ROM or program RAM address. The upper 4 bits of the address are assumed equal to the upper 4 bits of the address at which the FIN instruction is located (that is, the address of the FIN instruction and the address referenced by register 0 and 1 are on the same page). The 8 bits at the designated address are loaded into the register pair specified by RP. The 8 bits at the designated address are unaffected; the contents of registers 0 and 1 are unaffected unless RP = 0.

The carry bit is not affected.

Example: Suppose a program in memory appears as follows:

| DECIMAL ADDRESS | HEXADECIMAL ADDRESS | INSTRUCTION | ASSEMBLED DATA |
|--------------------|------------------------|-------------|-------------------|
| 603 | 25B | 110 | 6E |
| : | : | : | : |
| 681 | 2A9 | FIN 7P | 3E |

If register 0 contains the hex digit 5 and register 1 contains the hex digit B when the FIN instruction is executed, the 8 bits located at hex address 25B will be loaded into register pair 7P. Thus register 14 will contain the hex digit 6 (0110 binary) and register 15 will contain the hex digit E (1110 binary).

If registers 0 and 1 had contained CH and 4 when the FIN was executed, the 8 bits at hex address 2C4 would have been loaded into registers 14 and 15.

NOTE: If a FIN instruction is located in the last location of a page, the upper 4 bits of the designated address will be assumed equal to the upper 4 bits of the next page.

Thus if the instruction:

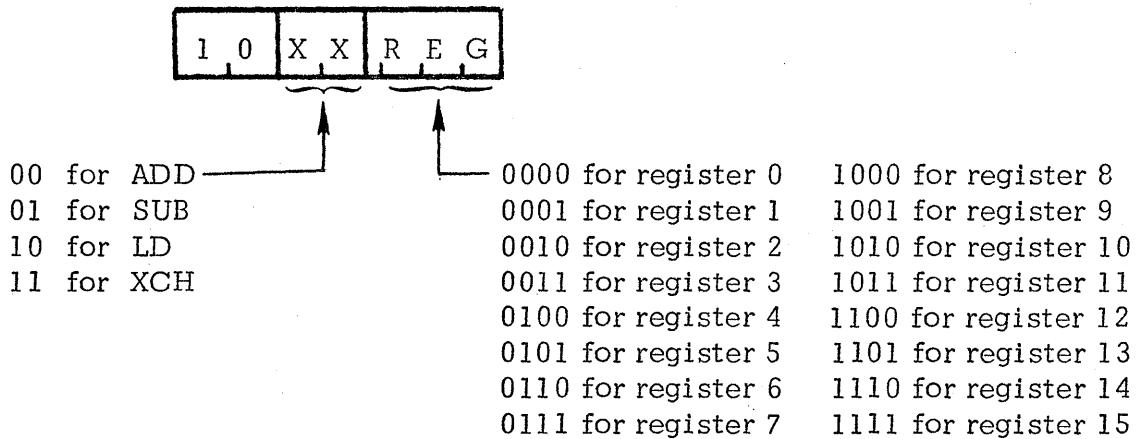
FIN 7P

is located at decimal address 511 (hex 1FF) and registers 0 and 1 contain 3 and CH, the 8 bits at hex address 23C (not 13C) will be loaded into registers 14 and 15.

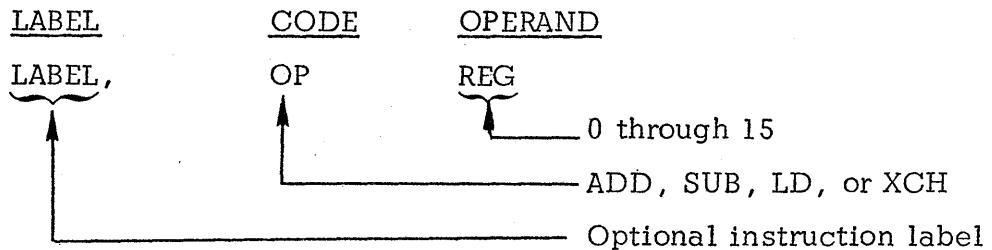
This is dangerous programming practice and should be avoided whenever possible.

3.4 INDEX REGISTER TO ACCUMULATOR INSTRUCTIONS

This section describes instructions which involve an operation between an index register and the accumulator. Instructions in this class occupy one byte as follows:

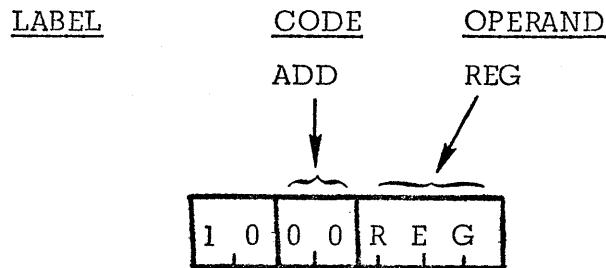


The general assembly language instruction format is:



3.4.1 ADD ADD REGISTER TO ACCUMULATOR WITH CARRY

Format:



Description:

The contents of the index register REG plus the contents of the carry bit are added to the accumulator. The result is kept in the accumulator; the contents of REG are unchanged. The carry bit is set if there is a carry out of the high-order bit position, and reset if there is no carry.

Example:

Suppose the accumulator contains 6, register 14 contains 9, and the carry bit = 0.

Then the instruction:

ADD 14

will perform the following operation:

$$\begin{array}{rcl} \text{Accumulator} & = & 0110 \text{ B} \\ \text{Register 14} & = & 1001 \text{ B} \\ \text{Carry} & = & \underline{\quad 0 \quad} \\ \hline 0 & 1111 \text{ B} & = \text{Result} = 15 \\ & \downarrow & \\ & \quad \quad \quad \text{= Carry} & \end{array}$$

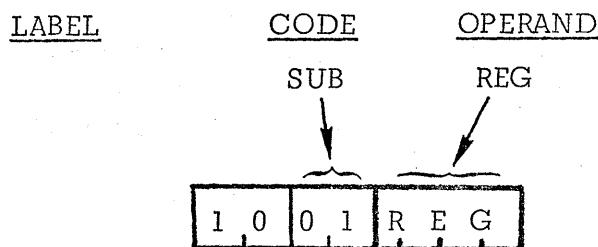
The accumulator contains 15 and the carry bit is reset. If the carry bit had been one at the start of the previous operation, the following would have occurred:

| | | |
|-------------|---|-----------------------------------|
| Accumulator | = | 0 1 1 0 B |
| Register 14 | = | 1 0 0 1 B |
| Carry | = | <u>1</u> |
| | | <u>1</u> 0 0 0 0 B = Result = 0 |
| | | _____ = Carry |

The accumulator would contain 0, while the carry bit would be set.

3.4.2 SUB SUBTRACT REGISTER FROM ACCUMULATOR WITH BORROW

Format:



Description:

The contents of index register REG are subtracted with borrow from the accumulator. The result is kept in the accumulator; the contents of REG are unchanged. A borrow from the previous subtraction is indicated by the carry bit being equal to one at the beginning of this instruction. If the carry bit equals zero at the beginning of this instruction it is assumed that no borrow occurred from the previous subtraction.

This instruction sets the carry bit if there is no borrow out of the high order bit position, and resets the carry bit if there is a borrow.

The subtract with borrow operation is actually performed by complementing each bit of the contents of REG and adding the resulting value plus the complement of the carry bit to the accumulator.

Note: This instruction may be used to subtract numbers greater than 4 bits in length. The carry bit must be complemented by the program between each required subtraction operation. For an example of this, see Section 4.8.

Example: In order to perform a normal subtraction, the carry bit should = 0. Suppose the accumulator contains 6, register 10 contains 2, and the carry bit = 0. Then the instruction:

SUB 10

will perform the following operation:

Accumulator = 0110 B
 Register 10 = 0010
 Complemented = 1101 B
 Complement of carry = 1

$$\begin{array}{r} 1 \\ \hline 0100 \end{array} \text{B} = \text{Result} = 4$$

Carry = 1 indicating no borrow

Had the carry bit been = 1, the operation would have produced the following:

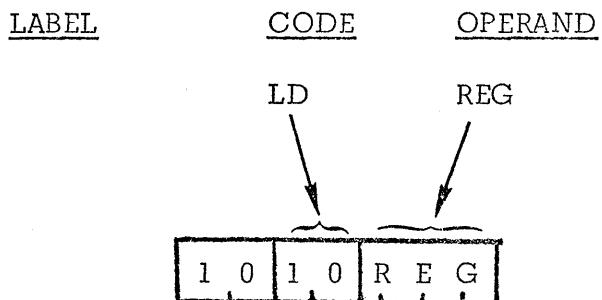
Accumulator 10 = 0110 B
 Complement of register 14 = 1101 B
 Complement of carry = 0

$$\begin{array}{r} 1 \\ \hline 0011 \end{array} \text{B} \text{ Result} = 3$$

Carry = 1 indicating no borrow.

3.4.3 LD LOAD ACCUMULATOR

Format:



Description:

The contents of REG are stored into the accumulator, replacing the previous contents of the accumulator. The contents of REG are unchanged. The carry bit is not affected.

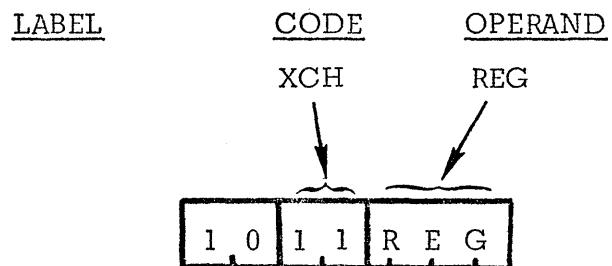
Example: If register 12 contains 0100 B, the instruction

LD 12

will cause the accumulator also to contain 0100 B.

3.4.4 XCH EXCHANGE REGISTER AND ACCUMULATOR

Format:



Description:

The contents of the register specified by REG are exchanged with the contents of the accumulator.

The carry bit is not affected.

Example: If the accumulator contains 1100 B, and register 0 contains 0011 B, then the instruction

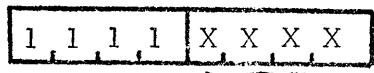
XCH 0

will cause the accumulator to contain 0011 B and register 0 to contain 1100 B.

3.5 ACCUMULATOR INSTRUCTIONS

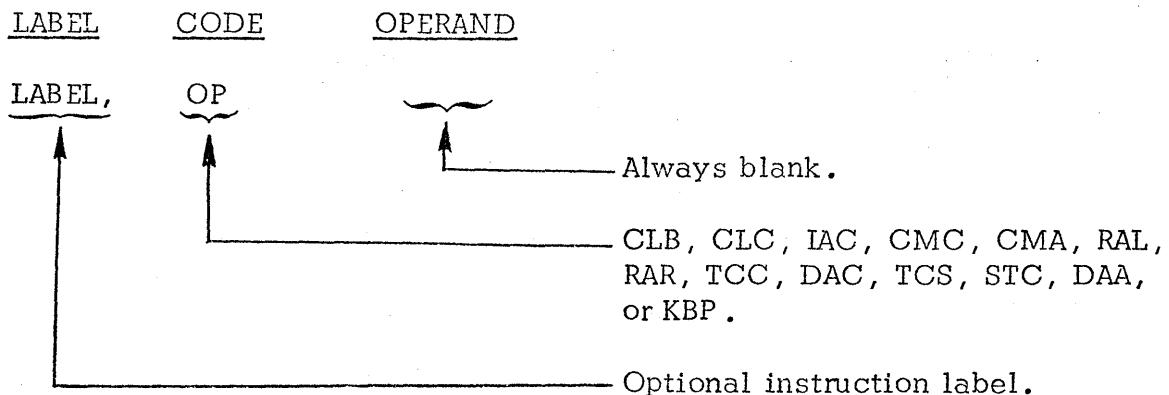
This section describes instructions which operate only on the contents of the accumulator and/or the carry bit.

Instructions in this class occupy one byte as follows:



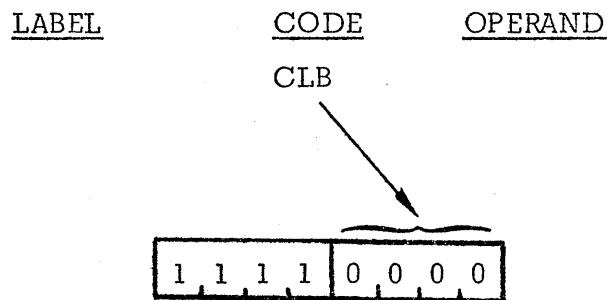
| | |
|--------------|--------------|
| 0000 for CLB | 0110 for RAR |
| 0001 for CLC | 0111 for TCC |
| 0010 for IAC | 1000 for DAC |
| 0011 for CMC | 1001 for TCS |
| 0100 for CMA | 1010 for STC |
| 0101 for RAL | 1011 for DAA |
| | 1100 for KBP |

The general assembly language instruction format is:



3.5.1 CLB CLEAR BOTH

Format:

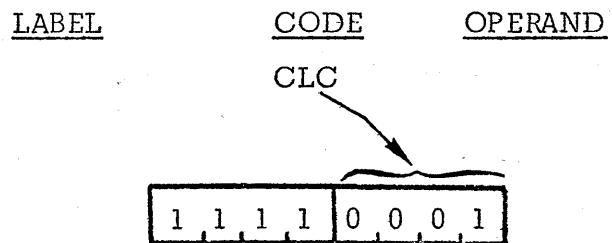


Description:

The accumulator is set to 0000 B, and the carry bit is reset.

3.5.2 CLC CLEAR CARRY

Format:

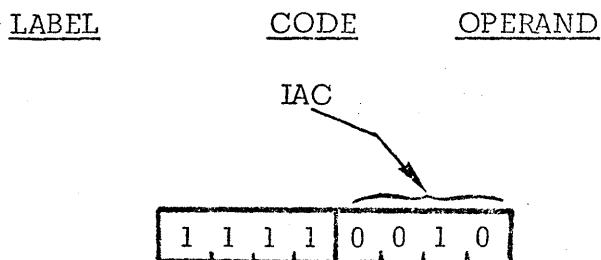


Description:

The carry bit is reset to 0.

3.5.3 IAC INCREMENT ACCUMULATOR

Format:



Description:

The contents of the accumulator are incremented by one. The carry bit is set if there is a carry out of the high order bit position, and reset if there is no carry.

Example: If the accumulator contains 1001B, the instruction IAC will perform the following operation:

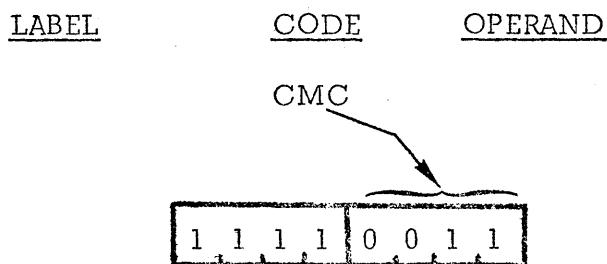
$$\begin{array}{r} \text{Accumulator} = 1001\text{B} \\ + \underline{0001}\text{B} \\ \hline 0 \boxed{1} 010\text{B} = \text{New contents of accumulator.} \\ \text{Carry} = 0 \end{array}$$

If the accumulator contains 1111B, the instruction IAC will perform the following operation:

$$\begin{array}{r} \text{Accumulator} = 1111\text{B} \\ + \underline{0001}\text{B} \\ \hline 1 \boxed{0} 000\text{B} = \text{New contents of accumulator.} \\ \text{Carry} = 1 \end{array}$$

3.5.4 CMC COMPLEMENT CARRY

Format:

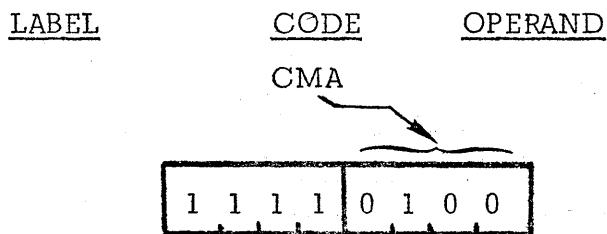


Description:

If the carry bit = 0, it is set to 1. If the carry bit is = 1, it is set to 0.

3.5.5 CMA COMPLEMENT ACCUMULATOR

Format:



Description:

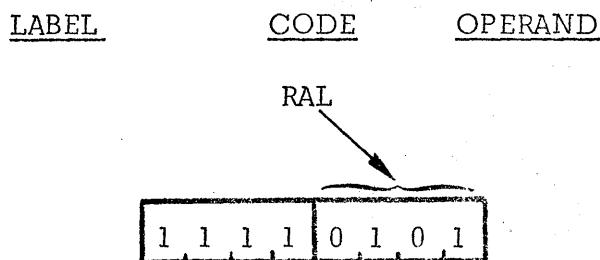
Each bit of the contents of the accumulator is complemented (producing the so-called one's complement).

The carry bit is not affected.

Example: If the accumulator contains 0110B, the instruction CMA will cause the accumulator to contain 1001B.

3.5.6 RAL ROTATE ACCUMULATOR LEFT THROUGH CARRY

Format:



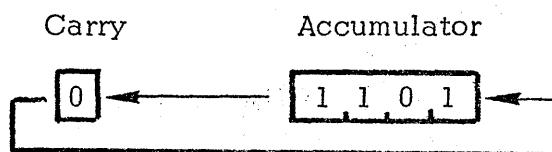
Description:

The contents of the accumulator are rotated one bit position to the left.

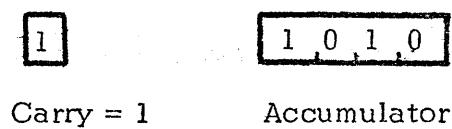
The high-order bit of the accumulator replaces the carry bit, while the carry bit replaces the low-order bit of the accumulator.

Example: Suppose the accumulator contains 1101 B, and the carry bit = 0.

Before RAL is executed:

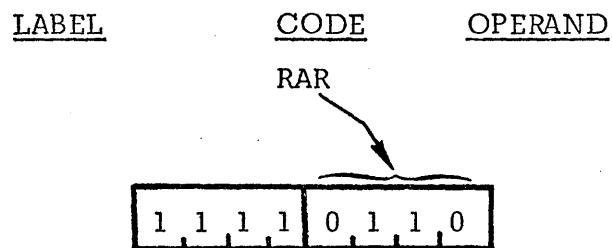


After RAL is executed:



3.5.7 RAR ROTATE ACCUMULATOR RIGHT THROUGH CARRY

Format:



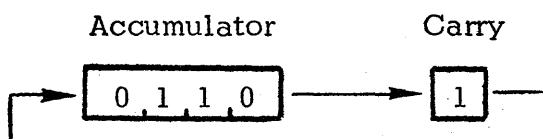
Description:

The contents of the accumulator are rotated one bit position to the right.

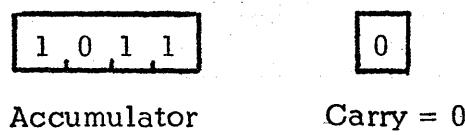
The low-order bit of the accumulator replaces the carry bit, while the carry bit replaces the high-order bit of the accumulator.

Example: Suppose the accumulator contains 0110B, and the carry bit = 1

Before RAR is executed:

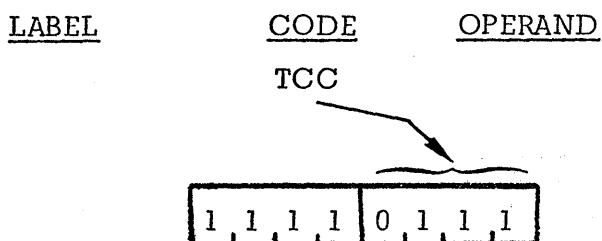


After RAR is executed:



3.5.8 TCC TRANSMIT CARRY AND CLEAR

Format:

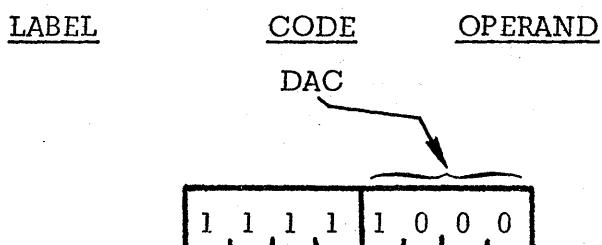


Description:

If the carry bit = 0, the accumulator is set to 0000B. If the carry bit = 1, the accumulator is set to 0001B. In either case, the carry bit is then reset.

3.5.9 DAC DECREMENT ACCUMULATOR

Format:



Description:

The contents of the accumulator are decremented by one. The carry bit is set if there is no borrow out of the high-order bit position, and reset if there is a borrow.

Example: If the accumulator contains 1001B, the instruction DAC will perform the following operation:

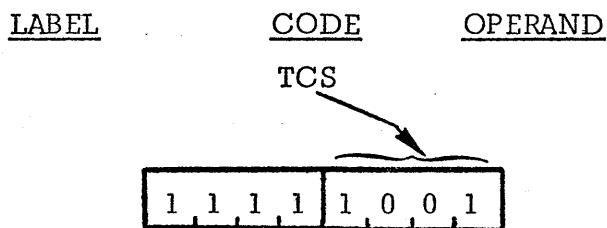
$$\begin{array}{rcl}
 \text{Accumulator} & = & 1001 \text{B} \\
 + (-1) & = & \underline{1111 \text{B}} \\
 1 | & 1000 \text{B} & = \text{New contents of accumulator} \\
 & \swarrow & \text{Carry} = 1 \text{ indicating no borrow.}
 \end{array}$$

If the accumulator contains 0000, the instruction DAC will perform the following:

$$\begin{array}{rcl}
 \text{Accumulator} & = & 0000 \text{B} \\
 + (-1) & = & \underline{1111 \text{B}} \\
 0 | & 1111 \text{B} & = \text{New contents of accumulator} \\
 & \swarrow & \text{Carry} = 0 \text{ indicating a borrow.}
 \end{array}$$

3.5.10 TCS TRANSFER CARRY SUBTRACT

Format:



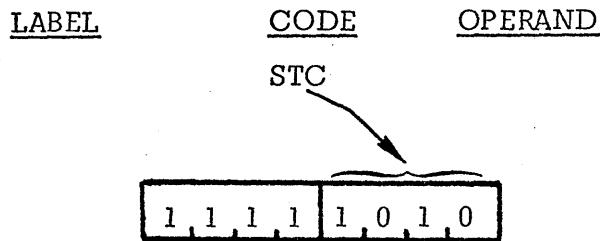
Description:

If the carry bit = 0, the accumulator is set to 9. If the carry bit = 1, the accumulator is set to 10. In either case, the carry bit is then reset.

NOTE: This instruction is used when subtracting decimal numbers greater than 4 bits in length. For an example of this, see Section 4.8.

3.5.11 STC SET CARRY

Format:

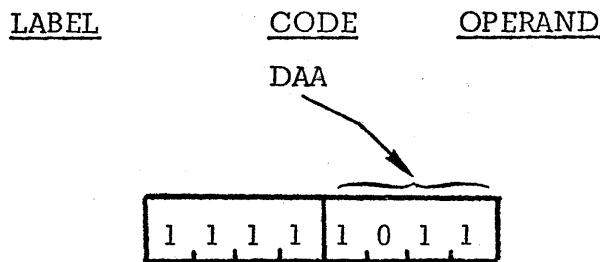


Description:

The carry bit is set to 1.

3.5.12 DAA DECIMAL ADJUST ACCUMULATOR

Format:



Description:

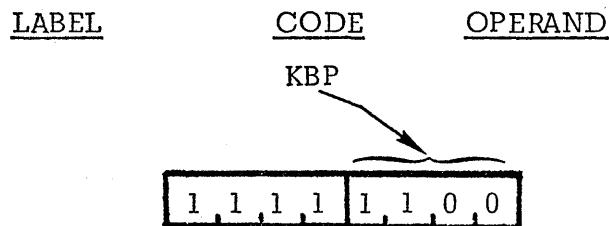
If the contents of the accumulator are greater than 9, or if the carry bit = 1, the accumulator is incremented by 6. Otherwise, the accumulator is not affected.

If the result of incrementing the accumulator produces a carry out of the high order bit position, the carry bit is set. Otherwise the carry bit is unaffected (in particular, it is not reset).

NOTE: This instruction is used when adding decimal numbers. For an example of this, see Section 4.7.

3.5.13 KBP KEYBOARD PROCESS

Format:



Description:

If the accumulator contains 0000B, it remains unchanged. If one bit of the accumulator is set, the accumulator is set to a number from 1 to 4 indicating which bit was set.

If more than one bit of the accumulator is set, the accumulator is set to 1111B.

This process is summarized as follows:

| BINARY CONTENTS OF ACCUMULATOR BEFORE KBP | BINARY CONTENTS OF ACCUMULATOR AFTER KBP |
|---|--|
| 0000 | 0000 |
| 0001 | 0001 |
| 0010 | 0010 |
| 0100 | 0011 |
| 1000 | 0100 |
| 0011 | 1111 |
| 0101 | 1111 |
| 0110 | 1111 |
| 0111 | 1111 |
| 1001 | 1111 |
| 1010 | 1111 |
| 1011 | 1111 |
| 1100 | 1111 |
| 1101 | 1111 |
| 1110 | 1111 |
| 1111 | 1111 |

The carry bit is not affected.

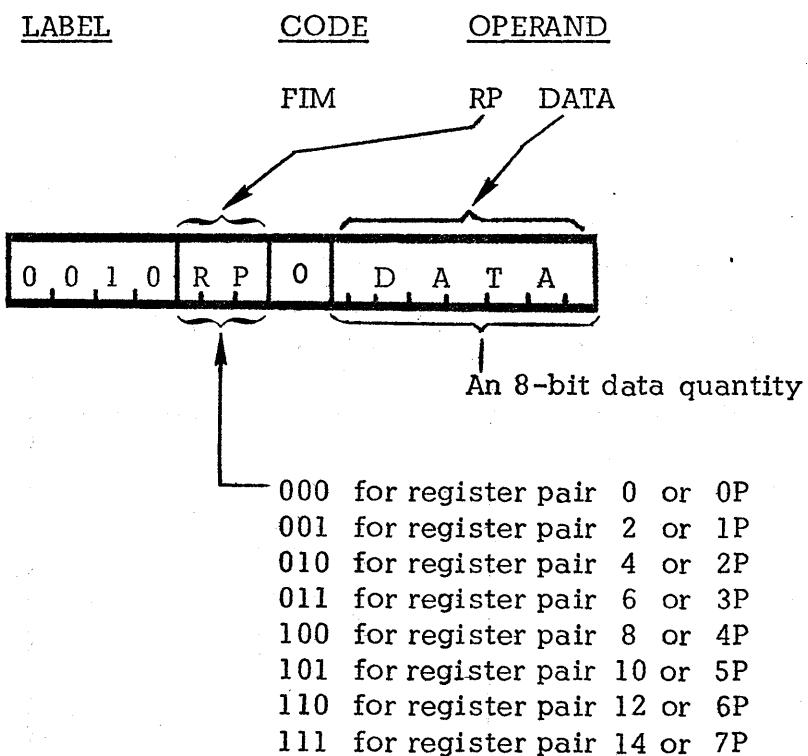
3.6 IMMEDIATE INSTRUCTIONS

This section describes two instructions which use data that is part of the instruction itself.

3.6.1 FIM FETCH IMMEDIATE

The FIM instruction occupies two bytes.

Format:



Description:

The 8 bits of immediate data are loaded into the register pair specified by RP. The carry bit is not affected.

Example: The instruction

FIM 2 254

will cause register 2 to contain 15, and register 3 to contain 14. This is because 254 decimal is encoded as FE hexadecimal; the upper four bits are loaded into register 2 and the lower four bits are loaded into register 3.

The instruction:

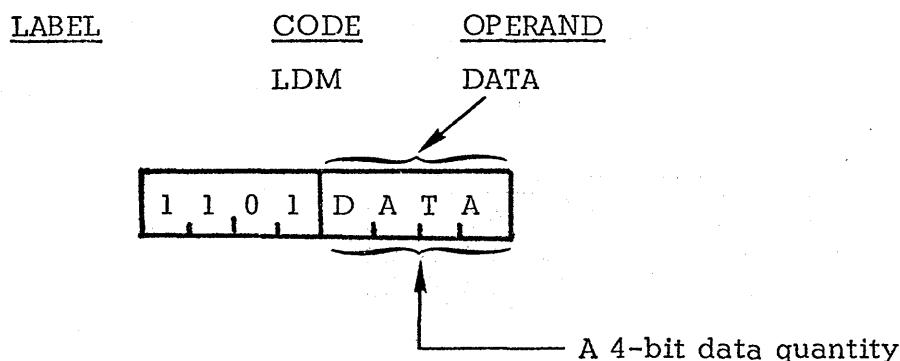
FIM 1P 6

will cause register 2 to contain 0, and register 3 to contain 6.

3.6.2 LDM LOAD ACCUMULATOR IMMEDIATE

The LDM instruction occupies one byte.

Format:



Description:

The 4 bits of immediate data are loaded into the accumulator.

The carry bit is not affected.

Example: The instruction:

LDM 0

will clear the accumulator.

The instruction:

LDM 15

will set each bit of the accumulator.

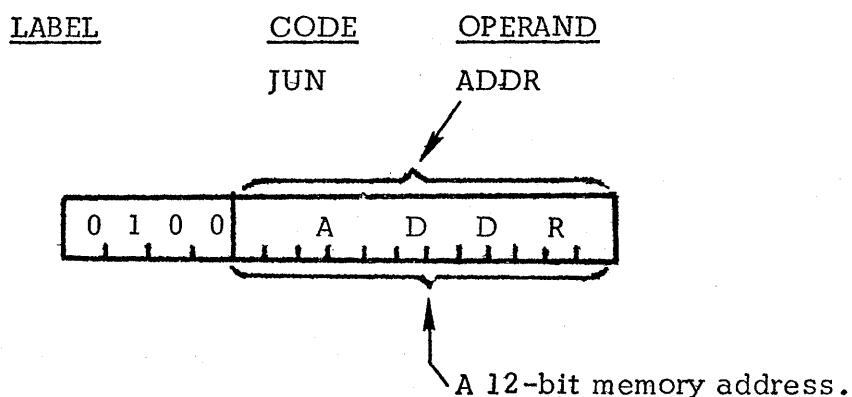
3.7 TRANSFER OF CONTROL INSTRUCTIONS

This section describes instructions which alter the normal execution sequence of instructions.

3.7.1 JUN JUMP UNCONDITIONALLY

The JUN instruction occupies two bytes:

Format:



Description:

Program execution is transferred to the instruction at location ADDR, which may be anywhere in memory. (If the JUN is located in ROM, ADDR is a ROM address; if located in program RAM, ADDR is a program RAM address).

The carry bit is not affected.

NOTE: This instruction and the JMS instruction (Section 3.8.1), use a 12 bit address, and can reference any memory location. Their operation is not influenced by their position within a page of memory, whereas some other instruc-

tions are. Therefore, only a JUN or JMS instruction should be used to transfer control from one page of memory to another.

Example:

| Arbitrary Memory Address (Hex) | Label | Code | Operand | Assembled Data |
|--------------------------------|-------|------|---------|----------------|
| 360 | | JUN | LRG | 43E0 |
| 362 | AD, | ADD | 1 | 82 |
| 370 | LAC, | LDM | 3 | D3 |
| 371 | | JUN | AD | 4362 |
| 3E0 | LRG, | FIM | 0P 4 | 2004 |
| 3E2 | | JUN | LAC | 4370 |

Normally, program instructions are executed sequentially. A 12-bit register called the program counter holds the address of the instruction to be executed. The JUN instruction replaces the program counter contents, causing program execution to continue at that address.

Thus the execution sequence of this example is as follows:

The JUN instruction at 360H replaces the contents of the program counter with 3E0H. The next instruction executed is the FIM at location LRG which loads register 0 with the value 0, and register 1 with the value 4. The JUN at 3E2H is then executed.

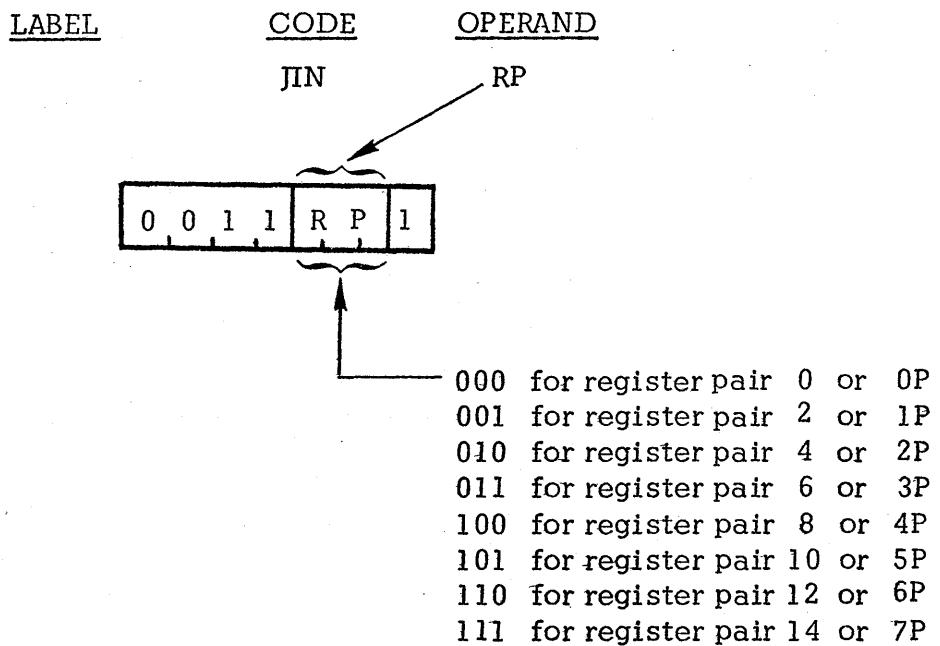
The program counter is set to 370H, and the LDM at this address loads the accumulator with the value 3. The JUN at 371H sets the program counter to 362H, where the ADD instruction adds the contents of register 1 plus the carry bit to the accumulator.

From here, normal program execution continues at location 363H.

3.7.2 JIN JUMP INDIRECT

The JIN instruction occupies one byte.

Format:



Description:

The 8 bits held in the register pair specified by RP are loaded into the lower 8 bits of the program counter. The highest 4 bits of the program counter are unchanged. Therefore, program execution continues at this address on the same page of memory in which the JIN instruction is loaded.

The carry bit is not affected.

Example:

| Hexadecimal Memory Address | Code | Operand | Assembled Data |
|-------------------------------|------|---------|----------------|
| 3E4 | FIM | OP 21 | 2015 |
| 3E6 | JIN | OP | |

The FIM instructions loads register 0 with the value 1 and register 1 with the value 5. The JIN instruction then causes a jump to hexadecimal location 315.

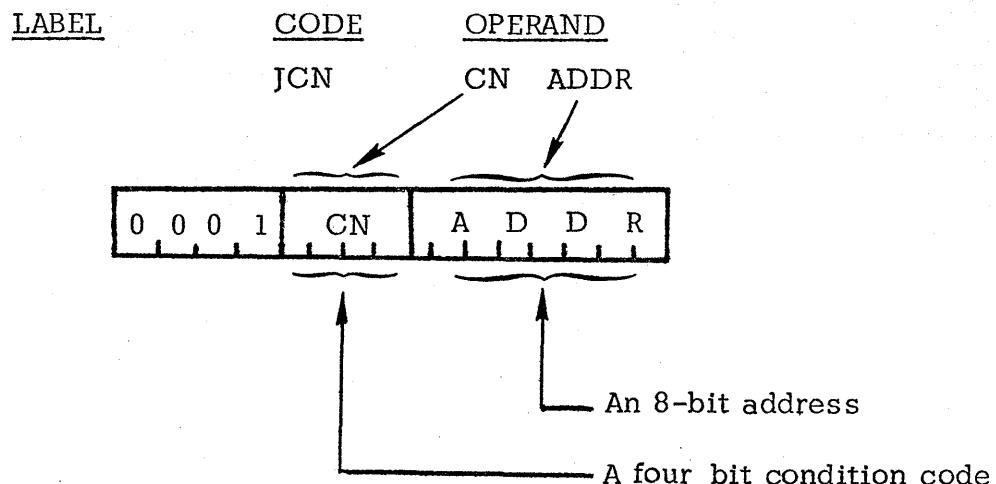
NOTE: If the JIN instruction is located in the last location of a page in memory, the highest 4 bits of the program counter are incremented by one, causing control to be transferred to the corresponding location on the next page.

If the above example, the JIN had been located at address 255 decimal (0FF hexadecimal), control would have been transferred to address 115 hexadecimal, not 015 hexadecimal. This is dangerous programming practice, and should be avoided whenever possible.

3.7.3 JCN JUMP ON CONDITION

The JCN instruction occupies two bytes.

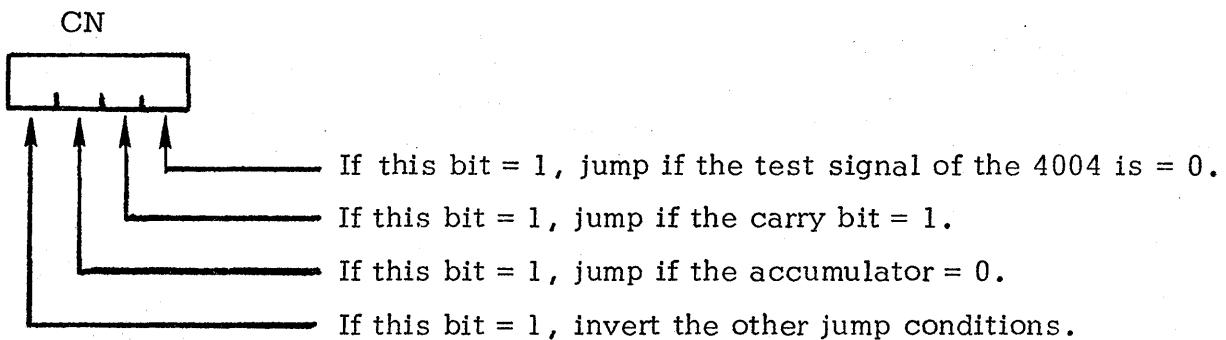
Format:



Description:

If the condition specified by CN is false, no action occurs and program execution continues with the next sequential instruction. If the condition specified by CN is true, the 8 bits specified by ADDR replace the lower 8 bits of the program counter. The highest 4 bits of the program counter are unchanged. Therefore, program execution continues at the specified address on the same page of memory in which the JCN instruction is located. The carry bit is not affected.

The condition code is specified in the assembly language statement as a decimal value from 0 to 15, which is represented in the assembled instruction as the corresponding 4 bit hexadecimal digit. Each bit of the condition code has a meaning, as follows:



More than one condition at a time may be tested. If the leftmost bit of the condition code is zero, a jump occurs if any of the remaining specified conditions is true (an "or" condition). If the leftmost bit is one, a jump occurs if the logical inverse of the "or" condition is true. In Boolean notation, the equation for the jump condition is as follows: $JUMP = \overline{C_1} \bullet ((ACC = 0) \bullet C_2 + (\text{carry} = 1) \bullet C_3 + \overline{\text{TEST}} \bullet C_4) + C_1 \bullet ((ACC \neq 0) + \overline{C_2}) \bullet ((\text{carry} = 0) + \overline{C_3}) \bullet (\overline{\text{TEST}} + \overline{C_4})$

Example:

| <u>Hexadecimal Memory Address</u> | <u>Label</u> | <u>Code</u> | <u>Operand</u> | <u>Assembled Data</u> |
|---------------------------------------|--------------|-------------|----------------|-----------------------|
| 302 | LOC, | LDM | 4 | D4 |
| : | | | : | : |
| 38B | | JCN | 6 LOC | 1602 |
| 38D | | | --- | |

The condition code is encoded as 0110B. Therefore, the JCN will cause a jump to address 302H if the accumulator = 0, or if the carry bit = 1. If neither of these is true, program execution continues with the instruction at location 38DH.

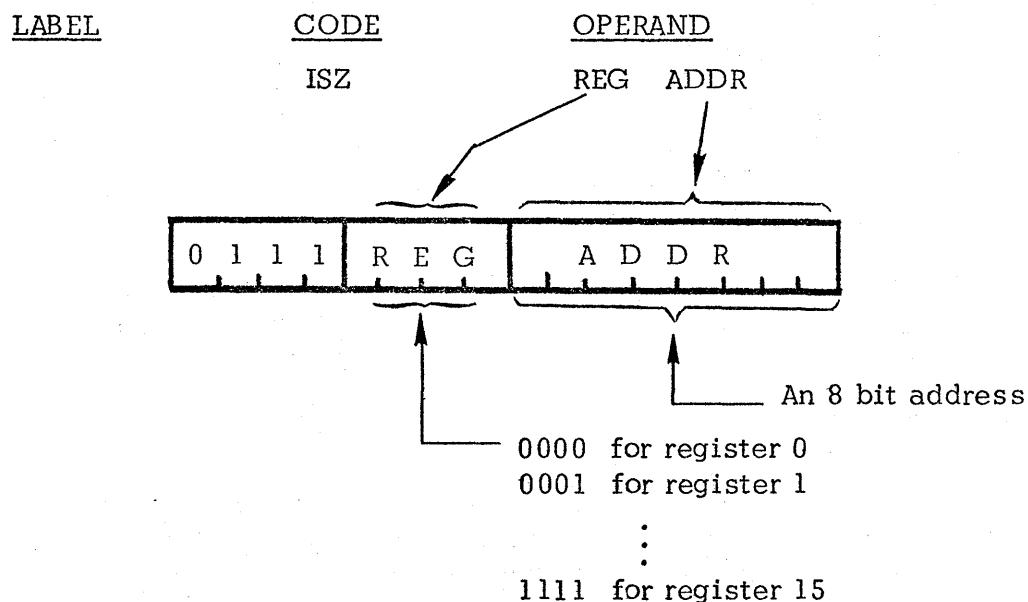
NOTE: If the JCN instruction is located in the last two locations of a page in memory and the jump condition is true, the highest 4 bits of the program counter are incremented by 1, causing control to be transferred to the corresponding location on the next page.

If in the above example, the JCN had been located at addresses 254 and 255 decimal (0FE and 0FF hexadecimal) a true condition would have caused jump to location 102 hexadecimal rather than 002 hexadecimal. This is dangerous programming practice, and should be avoided whenever possible.

3.7.4 ISZ INCREMENT AND SKIP IF ZERO

The ISZ instruction occupies two bytes.

Format:



Description:

The index register specified by REG is incremented by one. If the result is 0000B, program execution continues with the next sequential instruction. If the result does not equal 0000B, the 8 bits specified by ADDR replace the lowest 8 bits of the program counter. The highest 4 bits of the program counter are unchanged. Therefore, program execution continues at the specified address on the same page of memory in which the ISZ instruction is located.

The carry bit is not affected.

Example:

| <u>Hexadecimal Memory Address</u> | <u>Label</u> | <u>Code</u> | <u>Operand</u> | <u>Assembled Data</u> |
|---------------------------------------|--------------|-------------|----------------|-----------------------|
| 30F | | FIM | 0P 0 | 2000 |
| 311 | LP, | XCH | 2 | B2 |
| : | : | : | | |
| 31A | | ISZ | 0 LP | 7011 |
| 31C | | -- | | |

The FIM instruction loads registers 0 and 1 with 0.

The XCH is then executed. Program execution continues until the ISZ is reached. Register 0 is incremented to contain 1, and, since this result is non-zero, program control is transferred back to location 311H. This process continues until register 0 = 1111B. Then the ISZ increments register 0 producing a result of 0000B, and execution continues with the instruction at 31CH.

NOTE: If the ISZ instruction is located in the last two locations of a page in memory and the incrementation produces a non-zero result, the highest 4 bits of the program counter are incremented by 1, causing control to be transferred to the corresponding location on the next page.

If in the above example, the ISZ had been located at decimal addresses 1022 and 1023 (3FE and 3FF hexadecimal), control would have been transferred to location 411 hexadecimal and the XCH and remaining instructions would have been executed only once. Thus, this is dangerous programming practice, and should be avoided whenever possible.

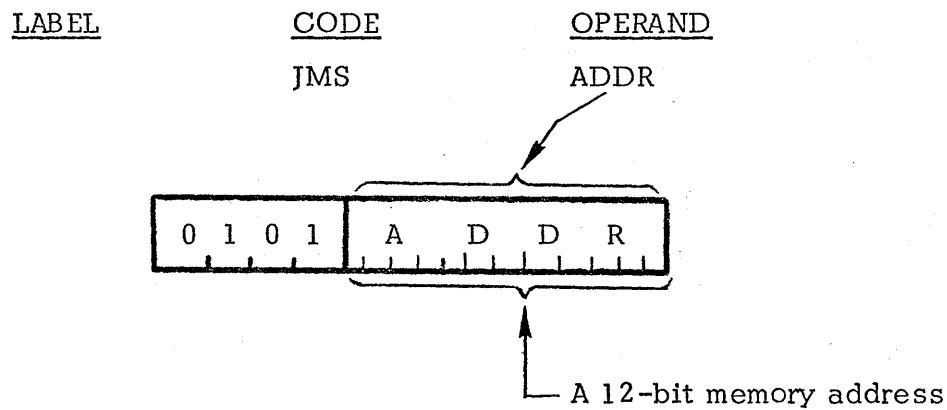
3.8 SUBROUTINE LINKAGE COMMANDS

This section describes the commands which call and cause return from subroutines. They cause a transfer of program control and use the address stack (see Sections 2.4 and 2.7.7).

3.8.1 JMS JUMP TO SUBROUTINE

The JMS instruction occupies two bytes.

Format:



Description:

The address of the instruction immediately following the JMS is written to the address stack for later use by a BBL instruction. Program execution continues at memory address ADDR, which may be on any page.

The carry bit is not affected.

NOTE: Since the JMS uses a 12 bit memory address, it operates the same wherever it is located in memory, and can reference any address in memory. For this reason, only a JMS or JUN instruction should be used to transfer program control from one page of memory to another.

Example:

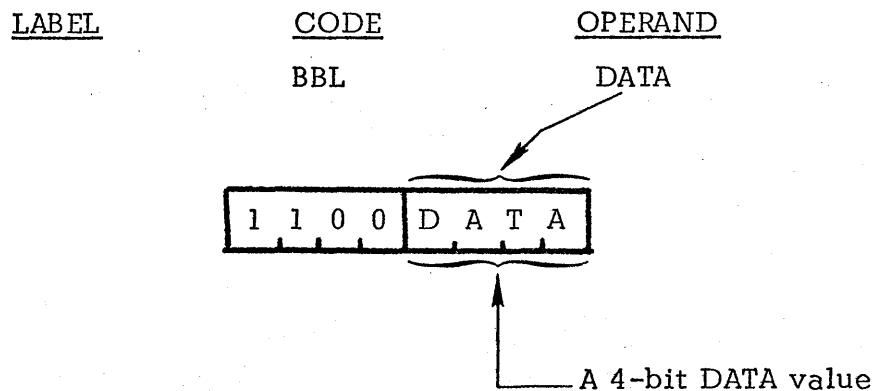
| <u>Hexadecimal Memory Address</u> | <u>Label</u> | <u>Code</u> | <u>Operand</u> | <u>Assembled Data</u> |
|---------------------------------------|--------------|-------------|----------------|-----------------------|
| 011 | | JMS | SUB | 53A0 |
| 013 | | XCH | 0 | D0 |
| | | ⋮ | | |
| 3A0 | SUB, | INC | 1 | 61 |
| | | ⋮ | | |
| | | BBL | 6 | C6 |

The JMS instruction causes the 12 bit address 013H (the address of the instruction following the JMS) to be written to the address stack. Execution continues with the INC instruction at SUB, and proceeds sequentially from this point.

3.8.2 BBL BRANCH BACK AND LOAD

The BBL instruction occupies one byte.

Format:



Description:

The 4 bits of immediate data encoded in the instruction are loaded into the accumulator. Then the last 12 bit address saved on the address stack (by a JMS instruction)

is read from the stack and placed in the program counter. Thus, execution continues with the instruction immediately following the last JMS instruction.

The carry bit is not affected.

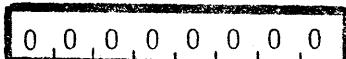
Example: In the example of Section 3.8.1, the BBL instruction loads the value 6 into the accumulator. The address 013 is read into the program counter, and program execution proceeds with the XCH instruction.

3.9 NOP INSTRUCTION NO OPERATION

This instruction occupies one byte.

Format:

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> |
|--------------|-------------|----------------|
| | NOP | |



A horizontal rectangle divided into eight equal-width segments, each containing a '0'. This represents the binary byte for the NOP instruction.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

Description:

No operation is performed. The program counter is incremented by one and execution continues with the next sequential instruction.

The carry bit is not affected.

3.10 MEMORY SELECTION INSTRUCTIONS

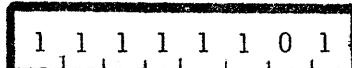
This section describes instructions which specify DATA RAM data and status characters, RAM output ports and ROM input and output ports to be operated on by I/O and RAM instructions described in Section 3.11.

3.10.1 DCL DESIGNATE COMMAND LINE

The DCL instruction occupies one byte.

Format:

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> |
|--------------|-------------|----------------|
| | DCL | |



A horizontal rectangle divided into eight equal-width segments. The first seven segments contain '1' and the eighth segment contains '0'. This represents the binary byte for the DCL instruction.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
|---|---|---|---|---|---|---|---|

Description:

As described in Section 2.3.3 there may be up to 8 DATA RAM BANKS, each of which consists of four DATA RAM units. The DCL instruction uses the rightmost 3 bits of the accumulator to determine which of the 8 DATA RAM BANKS will be referenced during subsequent operations.

The selection is made as follows:

| RIGHTMOST 3 BITS OF ACCUMULATOR | DATA RAM BANK SELECTED |
|------------------------------------|---------------------------|
| 000 | 0 |
| 001 | 1 |
| 010 | 2 |
| 011 | 3 |
| 100 | 4 |
| 101 | 5 |
| 110 | 6 |
| 111 | 7 |

This choice remains in effect until the next DCL is executed, or an external RESET signal is received. A RESET causes DATA RAM BANK 0 to be selected.

The carry bit is not affected.

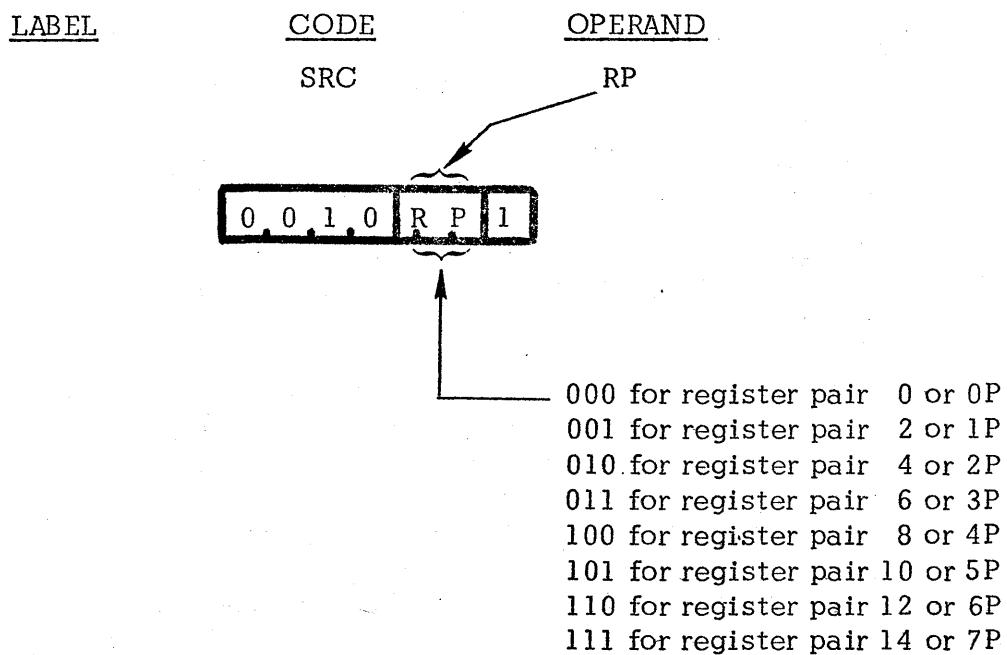
Example: The following instructions will select DATA RAM BANK 3:

LDM 3 /Load accumulator with 0011 B
DCL

3.10.2 SRC SEND REGISTER CONTROL

The SRC instruction occupies one byte.

Format:



Description:

The 8 bits contained in the register pair specified by RP are used as an address. This address may designate a particular DATA RAM data character, a DATA RAM status character, a RAM output port, or a ROM input/output port. (A description of these elements appears in Section 2). In fact, the address designates all of these simultaneously; it is up to the programmer to then write the correct I/O or RAM instruction (described in Section 3.11) to access the proper entity.

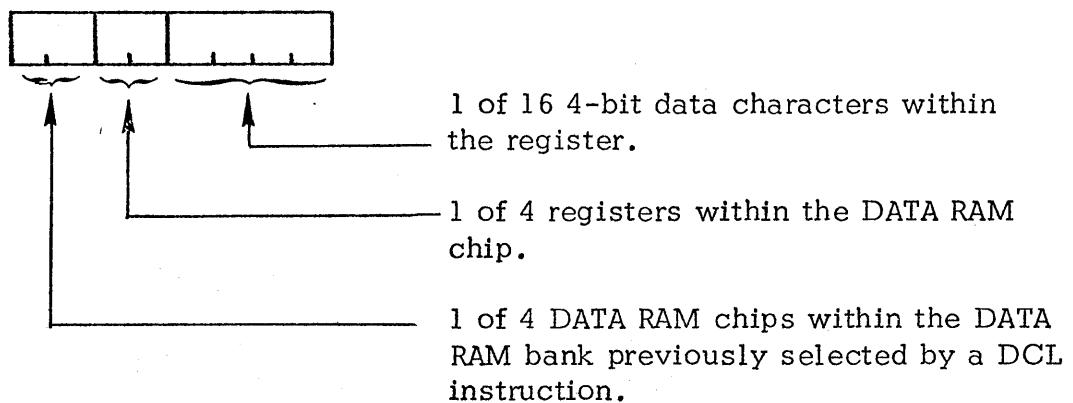
The address sent by the SRC remains in effect until changed by a subsequent SRC.

The **only** DATA RAM bank which receives the SRC address is the one selected by the last previous DCL instruction.

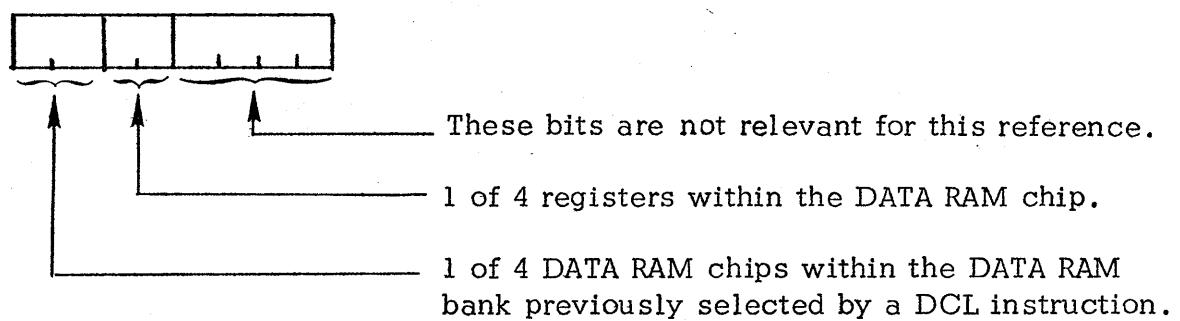
The carry bit and the contents of the register pair are unaffected.

The 8 bits of the address sent by the SRC are interpreted as follows:

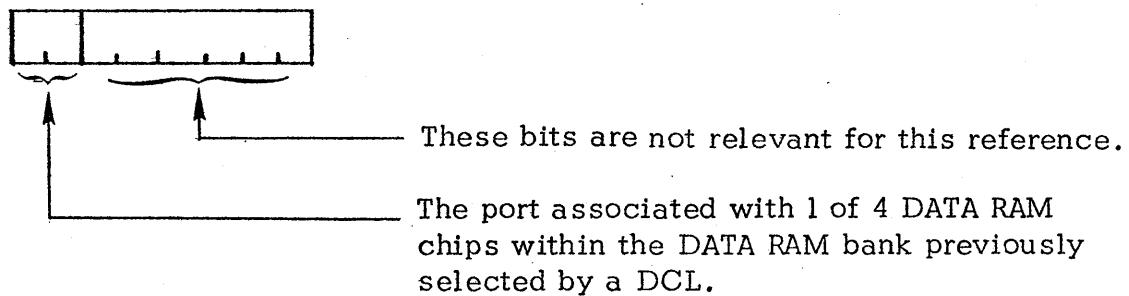
- (1) When referencing a DATA RAM data character:



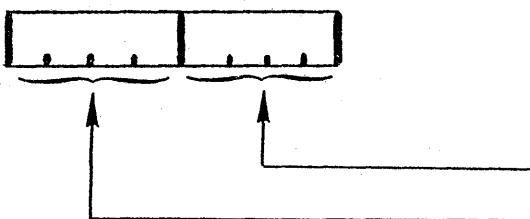
- (2) When referencing a DATA RAM status character:



- (3) When referencing a RAM output port:



(4) When referencing a ROM input or output port:



These bits are not relevant for this reference.
The port associated with 1 of 16 ROM's.

Example: The instructions:

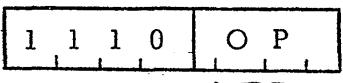
| | | |
|-----|----|-----|
| FIM | 1P | 180 |
| SRC | 1P | |

will cause the eight bit value 10110100B to be used as an address. Subsequent instructions could then reference DATA RAM data character number 4 of register 3 of chip 2, any of the status characters associated with DATA RAM register 3 of chip 2, RAM output port number 2 (the port associated with DATA RAM chip 2), or ROM port number 11 (the port associated with ROM number 11). The address remains in effect until another SRC instruction is executed.

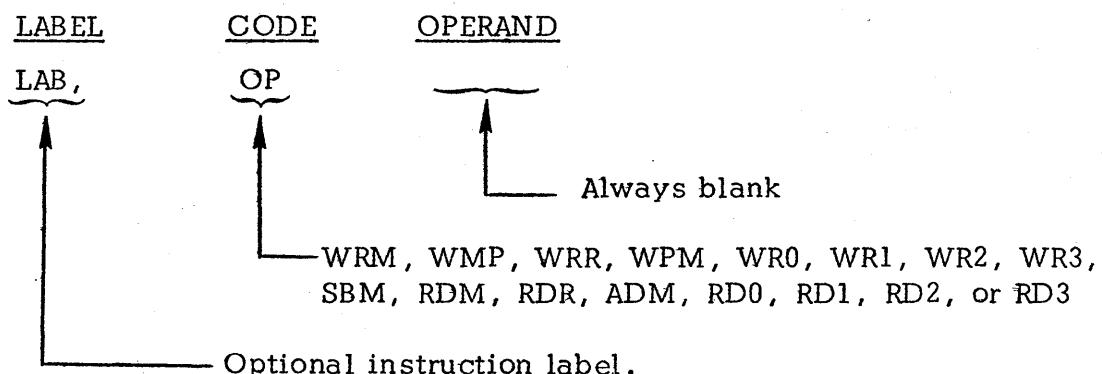
3.11 INPUT/OUTPUT AND RAM INSTRUCTIONS

This section describes instructions which access DATA RAM characters or perform input or output operations. One instruction, WPM, allows the programmer to read or write 8-bit program RAM locations. These instructions use addresses selected by the DCL and SRC instructions described in Section 3.1.0.

Instructions in this class occupy one byte as follows:

|  | |
|---|--------------|
| 0000 for WRM | 1000 for SBM |
| 0001 for WMP | 1001 for RDM |
| 0010 for WRR | 1010 for RDR |
| 0011 for WPM | 1011 for ADM |
| 0100 for WR0 | 1100 for RD0 |
| 0101 for WR1 | 1101 for RD1 |
| 0110 for WR2 | 1110 for RD2 |
| 0111 for WR3 | 1111 for RD3 |

The general assembly language instruction format is:

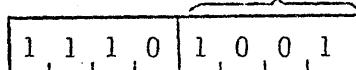


3.11.1 RDM READ DATA RAM DATA CHARACTER

Format:

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> |
|--------------|-------------|----------------|
|--------------|-------------|----------------|

RDM



Description:

The DATA RAM data character specified by the last SRC instruction is loaded into the accumulator. The carry bit and the data character are not affected.

Example:

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> |
|--------------|-------------|----------------|
|--------------|-------------|----------------|

| | | |
|-----|----|---|
| FIM | 2P | 5 |
| SRC | 2P | |
| RDM | | |

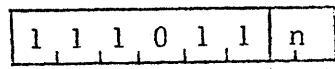
The above instructions will read the contents of DATA RAM data character number 5 of register 0 of chip 0 of the currently selected DATA RAM bank into the accumulator.

3.11.2 RDn READ DATA RAM STATUS CHARACTER

Format:

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> |
|--------------|-------------|----------------|
|--------------|-------------|----------------|

RDn



$n = 0, 1, 2, \text{ or } 3$

Description:

The DATA RAM status character whose number from 0 to 3 is specified by n, associated with the DATA RAM register specified by the last SRC instruction, is loaded into the accumulator.

The carry bit and the status character are not affected.

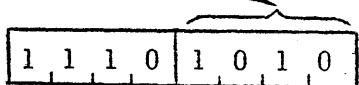
Example:

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> |
|--------------|-------------|----------------|
| | FIM | 2P 5 |
| | SRC | 2P |
| | RD3 | |

The above instructions will read the contents of DATA RAM status character 3 of register 0 of chip 0 of the currently selected DATA RAM bank into the accumulator.

3.11.3 RDR READ ROM PORT

Format:

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> |
|--------------|-------------|--|
| | RDR |  A diagram showing the RDR code pointing to a memory location containing binary data. The memory location is represented as a horizontal box divided into two sections by a vertical line. The left section contains the binary digits 1, 1, 1, 0. The right section contains the binary digits 1, 0, 1, 0. An arrow points from the 'RDR' label to the right section of the memory location. |

Description:

The ROM port specified by the last SRC instruction is read. When using the 4001 ROM, each of the 4 lines of the port may be an input or an output line; the data on the input lines is transferred to the corresponding bits of the accumulator. Any output lines cause either a 0 or a 1 to be transferred to the corresponding bits of the accumulator. Whether a 0 or a 1 is transferred is a function of the hardware, not under control of the programmer.

The carry bit is not affected.

Example:

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> |
|--------------|-------------|----------------|
| | FIM | 3P 160 |
| | SRC | 3P |
| | RDR | |

The above instructions will read the contents of the port associated with ROM number ten into the accumulator. If the leftmost I/O line is an output line and the remaining I/O lines are input lines containing 010B, the accumulator will contain either 1010B or 0010B.

NOTE: On the INTELLEC 4, a ROM port may be used for either input or output. If programs tested on the INTELLEC 4 are to be run later with a 4001 ROM, the programmer must be careful not to use one port for both functions.

3.11.4 WRM WRITE DATA RAM CHARACTER

Format:

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> |
|--------------|-------------|----------------|
| | WRM | |

↓

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|

Description:

The contents of the accumulator are written into the DATA RAM data character specified by the last SRC instruction.

The carry bit and the accumulator are not affected.

Example:

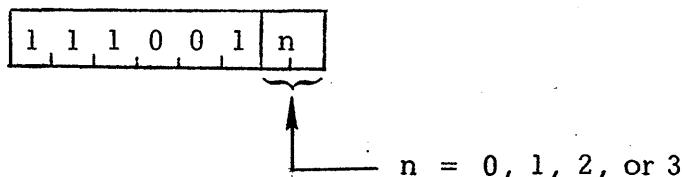
| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> |
|--------------|-------------|----------------|
| FIM | OP 180 | |
| SRC | OP | |
| LDM | 15 | |
| WRM | | |

The above instruction will cause DATA RAM data character number 4 of register 3 of chip 2 of the DATA RAM bank selected by the last DCL instruction to contain 15 (1111B).

3.11.5 WRn WRITE DATA RAM STATUS CHARACTER

Format:

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> |
|--------------|-------------|----------------|
| WRn | | |



Description:

The contents of the DATA RAM status character whose number from 0 to 3 is specified by n, associated with the DATA RAM register specified by the last SRC instruction, are replaced by the contents of the accumulator.

The carry bit and the accumulator are not affected.

Example:

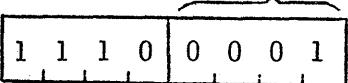
| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> |
|--------------|-------------|----------------|
| FIM | 0P | 0 |
| SRC | 0P | |
| LDM | 2 | |
| WRL | | |

The above instructions will write the value 2 into status character 1 of DATA RAM register 0 of chip 0 of the currently selected DATA RAM bank.

3.11.6 WMP WRITE RAM PORT

Format:

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> |
|--------------|-------------|----------------|
| | WMP | |

↓


Description:

The contents of the accumulator are written to the output port associated with the DATA RAM chip selected by the last SRC instruction. This value will stay at the output port until overwritten.

The carry bit and the accumulator are unchanged.

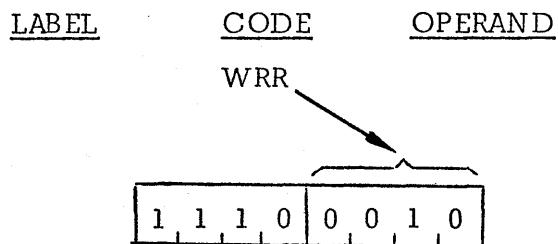
Example:

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> |
|--------------|-------------|----------------|
| FIM | 3P | 64 |
| SRC | 3P | |
| LDM | 6 | |
| WMP | | |

The above instructions will write the value 6 to the output port associated with DATA RAM chip 2 of the currently selected DATA RAM bank.

3.11.7 WRR WRITE ROM PORT

Format:



Description:

The contents of the accumulator are written to the output port associated with the ROM selected by the last SRC instruction. This value will stay at the output port until overwritten.

The carry bit and the accumulator are unchanged.

Example:

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> |
|--------------|-------------|----------------|
| FIM | 4P | 64 |
| SRC | 4P | |
| LDM | 15 | |
| WRR | | |

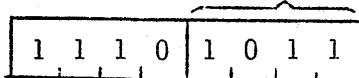
The above instructions will write the value 15 to the output port associated with ROM number 4.

3.11.8 ADM ADD DATA RAM TO ACCUMULATOR WITH CARRY

Format:

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> |
|--------------|-------------|----------------|
|--------------|-------------|----------------|

ADM



Description:

The DATA RAM data character specified by the last SRC instruction, plus the carry bit, are added to the accumulator.

The carry bit will be set if the result generates a carry, and will be reset otherwise.

The data character is not affected.

Example:

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> |
|--------------|-------------|----------------|
| FIM | OP 0 | |
| SRC | OP | |
| ADM | | |

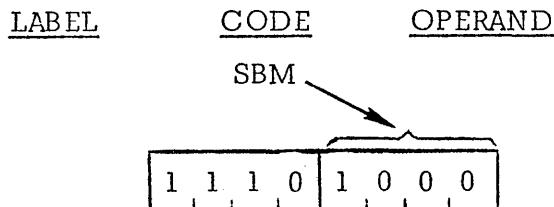
If the carry bit = 0, the accumulator contains 10, and DATA RAM data character 0 of register 0 of chip 0 contains 7, the ADM will perform the following operation:

| | | |
|----------------|---|----------------|
| Accumulator | = | 1 0 1 0 B |
| Data character | = | 0 1 1 1 B |
| Carry bit | = | <u> 0</u> |

1 0 0 0 1 B = New contents of accumulator.
Carry bit will be set.

3.11.9 SBM SUBTRACT DATA RAM FROM MEMORY WITH BORROW

Format:



Description:

The value of the DATA RAM character specified by the last SRC instruction is subtracted from the accumulator with borrow. The data character is unaffected. A borrow from the previous subtraction is indicated by the carry bit being equal to one at the beginning of this instruction. No borrow from the previous subtraction is indicated by the carry bit being equal to zero at the beginning of this instruction.

This instruction sets the carry bit if the result generates no borrow, and resets the carry bit if the result generates a borrow.

The subtract with borrow operation is actually performed by complementing each bit of the data character and adding the resulting value plus the complement of the carry bit to the accumulator.

NOTE: When this instruction is used to subtract numbers greater than 4 bits in length, the carry bit must be complemented by the program between each required subtraction operation. For an example of this, see Section 4.8.

Example:

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> |
|--------------|-------------|----------------|
| FIM | 1P 1 | |
| SRC | 1P | |
| SBM | | |

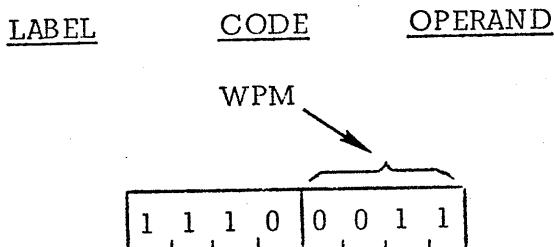
If the carry bit = 1, the accumulator contains 7, and DATA RAM character 1 of register 0 of chip 0 contains 5, the SBM will perform the following operation:

$$\begin{array}{rcl} \text{Accumulator} & = & 0111 \text{ B} \\ \text{Data character} = 0101 & \text{B} & \\ \text{Complemented} & = & 1010 \text{ B} \\ \text{Complement of Carry} & = & \underline{0} \\ \hline 0 & 0001 \text{ B} & = \text{New contents of accumulator.} \end{array}$$

Carry bit will be reset indicating a borrow.

3.11.10 WPM WRITE PROGRAM RAM

Format:



Description:

This is a special instruction which may be used to write the contents of the accumulator into a half byte of program RAM, or read the contents of a half byte of program RAM into a ROM input port where it can be accessed by a program.

The carry bit is unaffected.

NOTE: Two WPM instructions must always appear in close succession; that is, each time one WPM instruction references a half byte of program RAM as indicated by an SRC address, another WPM must access the other half byte before the SRC address is altered. An internal counter keeps track of which half-byte is being accessed. If only one WPM occurs, this counter will be out of sync with the program and errors will occur. In this situation, a RESET pulse must be used to re-initialize the machine.

NOTE: A WPM instruction requires an SRC address to access program RAM. Whenever a WPM is executed, the DATA RAM which happens to correspond to this SRC address will also be written. If data needed later in the program is being held in such a DATA RAM, the programmer must save it elsewhere before executing the WPM instruction.

Storing Data Into Program RAM:

A program must perform the following actions in order to store eight bits of data into a program RAM location:

- (1) The value 1 must be written to ROM port number 14. This is a "write enable" signal, permitting the store operation to work.
- (2) The highest 4 bits of the program RAM address to be accessed must be written to ROM port number 15.
- (3) The lowest 8 bits of the program RAM address to be accessed must be sent out by an SRC instruction.

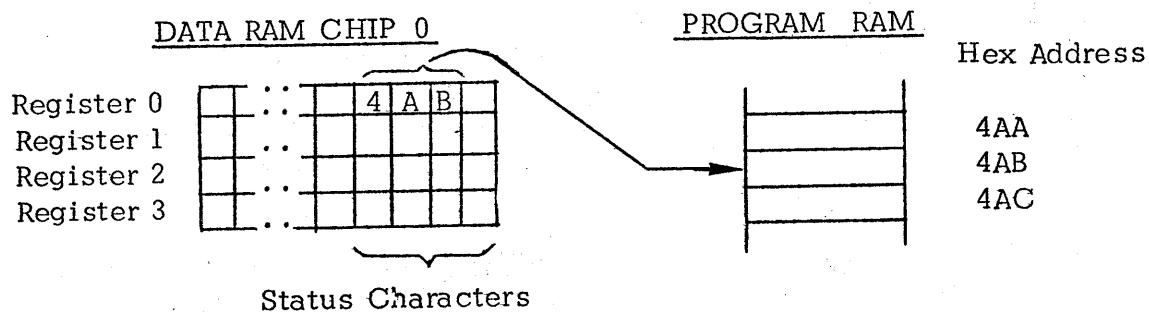
- (4) The higher 4 bits of data to be written must be loaded into the accumulator and written with the first WPM; the lower 4 bits of data must then be loaded into the accumulator and written with the second WPM.
- (5) The value 0 must be written to ROM port number 14, clearing the "write enable".

Reading Data From Program RAM:

A program must perform the following actions in order to read eight bits of data from a program RAM location:

- (1) The highest 4 bits of the program RAM address to be accessed must be written to ROM port 15.
- (2) The lowest 8 bits of the program RAM address to be accessed must be sent out by an SRC instruction.
- (3) Two WPM instructions in succession must be executed. The first reads the leftmost 4 bits of the program RAM location into ROM port 14; the second reads the rightmost 4 bits of the program RAM location into ROM port 15.

Example: The following routines access a program RAM location whose address is held in status characters 0, 1, and 2 of DATA RAM register 0 of DATA RAM chip 0.



Routine STR stores the contents of registers 2 and 3 into the addressed location; routine FCH reads the contents of the addressed location into registers 2 and 3.

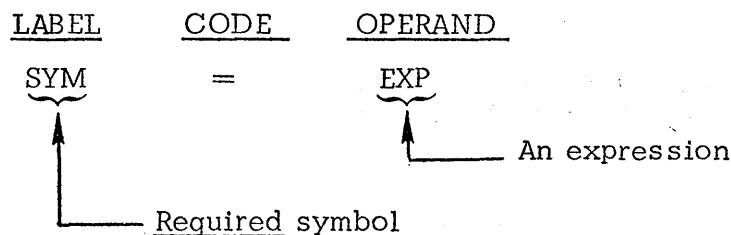
| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> | <u>COMMENT</u> |
|--------------|-------------|----------------|--------------------------------------|
| STR, | FIM | OP 224 | |
| | SRC | OP | / Select ROM port 14. |
| | LDM | 1 | |
| | WRR | | / Turn on write enable. |
| | JMS | COM | / Routine COM sets up PRAM address. |
| | LD | 2 | / High 4 data bits to accumulator. |
| | WPM | | / Write to PRAM |
| | LD | 3 | / Low 4 data bits to accumulator. |
| | WPM | | |
| | FIM | OP 224 | |
| | SRC | OP | / Select ROM port 14. |
| | CLB | | |
| | WRR | | / Turn off write enable. |
| | BBL | 0 | / Return to program |
| <hr/> | | | |
| COM, | FIM | OP 0 | |
| | SRC | OP | / Select DATA RAM chip 0 register 0. |
| | RD1 | | / Read middle 4 bits of address. |
| | XCH | 10 | / Save in register 10. |
| | RD2 | | / Read lowest 4 bits of address. |
| | XCH | 11 | / Save in register 11. |
| | RD0 | | / Read highest 4 bits of address. |
| | FIM | OP 240 | |
| | SRC | OP | / Select ROM port 15. |
| | WRR | | / Write high address |
| | SRC | 5P | / Write middle + low address |
| | BBL | 0 | / Return to STR or FCH |
| <hr/> | | | |
| FCH | JMS | COM | / Routine COM sets up PRAM address. |
| | WPM | | / PRAM data to ROM port 14. |
| | WPM | | / PRAM data to ROM port 15. |
| | FIM | OP 224 | |
| | SRC | OP | / Select port 14. |
| | RDR | | / Read to accumulator. |
| | XCH | 2 | / Save in register 2. |
| | INC | 0 | |
| | SRC | OP | / Select port 15. |
| | RDR | | / Read to accumulator |
| | XCH | 3 | / Save in register 3. |
| | BBL | 0 | / Return to program |

3.12 PSEUDO INSTRUCTION

This section describes the functions of the pseudo instruction recognized by the assembler. The pseudo instruction is indicated by the character = (equal sign) written in the code field of an assembler statement. No executable object code is generated by the pseudo instruction. It acts merely to provide the assembler with information to be used subsequently while generating object code.

3.12.1 EQUATE FUNCTION

Format:



Description:

The symbol SYM is assigned the value EXP by the assembler. Whenever the symbol SYM is encountered subsequently by the assembler, this value will be used.

Example: The statements

CZ = 10
JCN CZ ADDR

are equivalent to the statement

JCN 10 ADDR

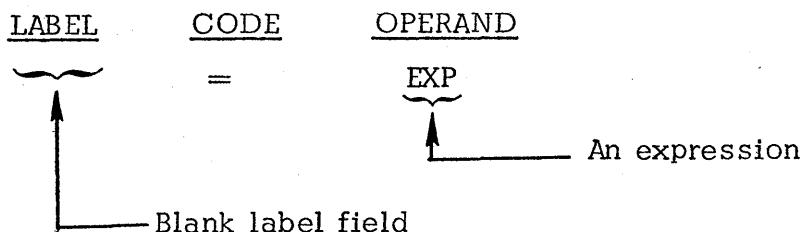
The statements

DAT = 5
LDM DAT

will load the value 5 into the accumulator.

3.12.2 ORIGIN FUNCTION

Format:



Description:

The assembler's location counter, is set to the value of EXP. The next machine instruction or data byte generated will be assembled at address EXP.

NOTE: The equal sign may appear in the first position of the line.

Example:

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> |
|--------------|-------------|----------------|
| = | 0 | |
| | JUN | LO |
| | = | 512 |
| LO, | LDM | 7 |

The JUN instruction will be assembled in locations 0 and 1 of ROM or program RAM. The location counter is then set to 512, causing the LDM instruction to be assembled at location 512, the first location on the second memory page. The JUN will therefore cause a jump to location 512.

NOTE: The pseudo instruction also makes it possible to assemble constant data values into a program. For a description of how to do this, see Section 3.2.2.

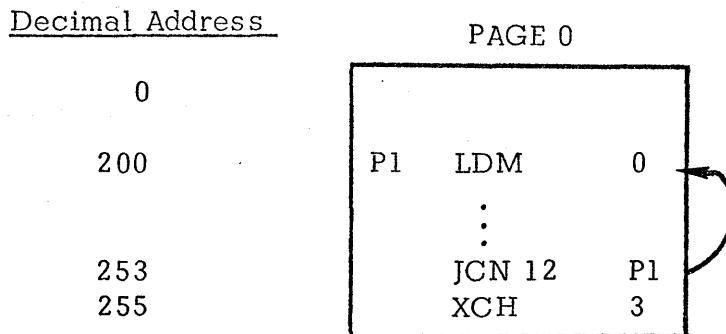
4.0 PROGRAMMING TECHNIQUES

This section describes some techniques which may be of help to the programmer.

4.1 CROSSING PAGE BOUNDARIES

As described in Section 2, programs are held in either ROM or program RAM, both of which are divided into pages. Each page consists of 256 8-bit locations. Addresses 0 through 255 comprise the first page, 256-511 comprise the second page, and so on.

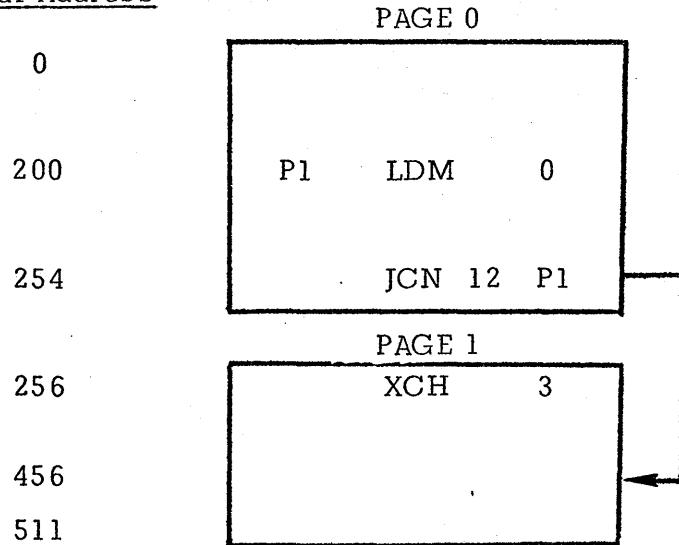
In general, it is good programming practice to never allow program flow to cross a page boundary except by using a JUN or JMS instruction. The following example will show why this is true. Suppose a program in memory appears as below:



If the accumulator is non-zero when the JCN is executed, program control will be transferred to location 200, as the programmer intended.

Suppose now that an error discovered in the program requires that a new instruction be inserted somewhere between locations 200 and 253. The program would now appear as follows:

Decimal Address



Since the JCN is now located in the last two locations of a page, it functions differently. Now if the accumulator is non-zero when the JCN is executed, program control will be erroneously transferred to location 456, causing invalid results.

Since both the JUN and JMS instructions use 12-bit addresses to directly address locations on any page of memory, only these instructions should be used to cross page boundaries.

4.2 SUBROUTINES

Frequently, a group of instructions must be repeated many times in a program. The group may be written "n" times if it is needed at "n" different points in a program, but better economy can be obtained by using subroutines.

A subroutine is coded like any other group of assembly language statements, and is referred to by its name, which is the label of the first instruction. The programmer references a subroutine by writing its name in the operand field of a JMS instruction. When the JMS is executed, the address of the next sequential instruction after the JMS is written to the address stack (see Section 2.4), and program execution proceeds with the first instruction of the subroutine. When the subroutine has completed its work, a BBL instruction is executed, which loads a value into the accumulator and causes an address to be read from the stack into the program counter, causing program execution to continue with the instruction following the JMS. Thus, one copy of a subroutine may be called from many different points in memory, preventing duplication of code. Note also that since the address stack and the JMS instruction use 12-bit addresses, calling programs and subroutines may be located anywhere in ROM or control program RAM (they need not be on the same page in memory).

Example: Subroutine IN increments an 8 bit number passed in index register 0 and 1 and then returns to the instruction following the last JMS instruction executed.

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> | |
|--------------|-------------|----------------|-------------------------------------|
| IN, | XCH | 1 | / Reg 1 to Accum. |
| | IAC | | / Increment value and produce carry |
| | XCH | 1 | / Restore reg 1. |
| | JCN | 10 NC | / Jump if Carry = 0. |
| | INC | 0 | / Increment high order 4 bits |
| NC, | BBL | 0 | / Return |

Assume IN appears as follows:

Arbitrary Memory

Address

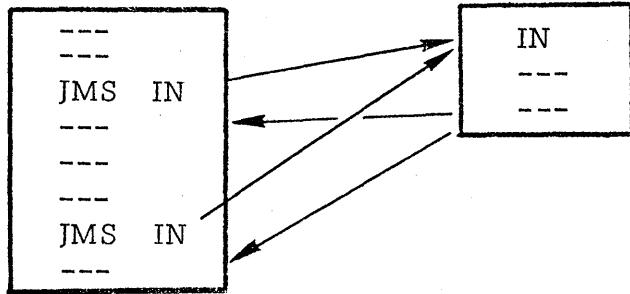
(Hex)

3C0

3C2

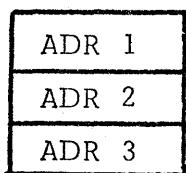
401

403

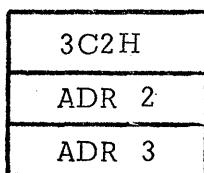


When the first JMS is executed, address 3C2H is written to the address stack, and control is transferred to IN. Execution of the BBL statement will cause the address 3C2H to be read from the stack and placed in the program counter, causing execution to continue at 3C2H (since the JMS occupies two bytes).

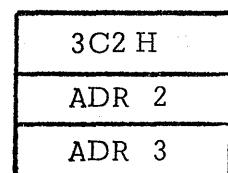
Address Stack
Before JMS



Stack While IN
Is Executing



Stack After BBL
Is Performed.



When the second JMS is executed, address 403H is written to the stack, and control is again transferred to IN. This time, the BBL will cause execution to resume at 403H.

Note that IN could have called another subroutine during its execution, causing another address to be written to the stack. This can occur only up to three levels, however, since the stack can hold only three addresses. Beyond this point, some addresses will be overwritten and BBL's will transfer program control to incorrect addresses.

4.3 BRANCH TABLE PSEUDOSUBROUTINE

Suppose a program consists of several separate routines, any of which may be executed depending upon some initial condition (such as a bit set in the accumulator). One way to code this would be to check each condition sequentially and branch to the routines accordingly as follows:

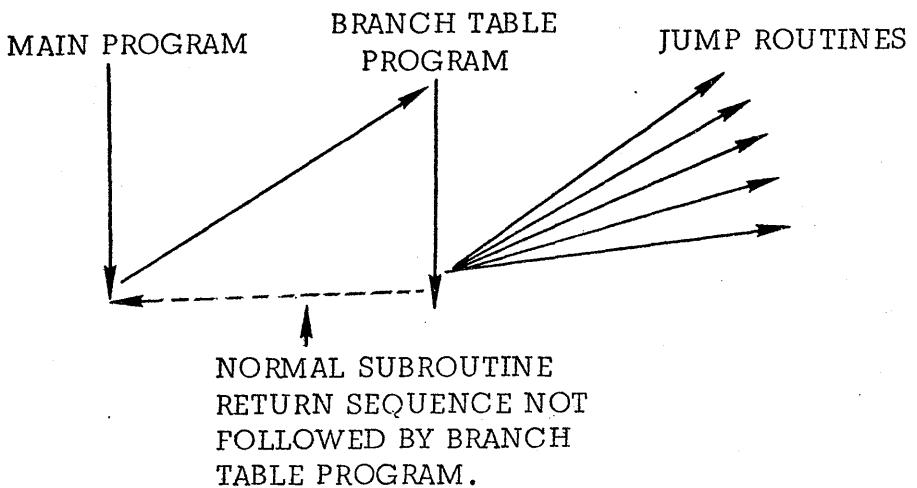
```
CONDITION = CONDITION 1 ?
IF YES BRANCH TO ROUTINE 1
CONDITION = CONDITION 2 ?
IF YES BRANCH TO ROUTINE 2
:
BRANCH TO CONDITION N
```

A sequence as above is inefficient, and can be improved by using a branch table.

The logic at the beginning of the branch table program computes an index into the branch table. The branch table itself consists of a list of starting addresses for the routines to be selected. Using the table index, the branch table program loads the selected routine's starting address into a register pair and executes a "jump indirect" to that address. For example, consider a program that executes one of five routines depending upon which bit (possibly none) of the accumulator is set:

```
Jump to routine 0 if accumulator = 0000 B
Jump to routine 1 if accumulator = 0001 B
Jump to routine 2 if accumulator = 0010 B
Jump to routine 3 if accumulator = 0100 B
Jump to routine 4 if accumulator = 1000 B
```

A program that provides the above logic is given at the end of this section. The program is termed a "pseudosubroutine" because it is treated as a subroutine by the programmer, (i.e., it appears just once in memory), but it is entered via a regular "jump" instruction rather than via a JMS instruction. This is possible because the branch routines control subsequent execution, and will never return to the instruction following JMS;



| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> | |
|--------------|---|----------------|--|
| ST, | KBP | | / Convert Accum to branch table index. |
| | IAC | | / If = 1111B, ERROR |
| | JCN | 4 ERR | / Jump if IAC produced zero. |
| | DAC | | / O.K., restore accumulator. |
| | FIM | OP BTL | / Regs 0 and 1 = address of branch table. |
| | CLC | | / Carry = 0 |
| | ADD | 1 | / Add index to branch table address |
| | XCH | 1 | / Store back in reg 1 |
| | JCN | 10 NC | / Jump if no carry |
| | INR | 0 | / If carry, increment reg 0. |
| NC, | FIN | OP | / Regs 0 and 1 = address of routine. |
| | JIN | OP | / Jump to correct routine. |
| | | ⋮ | |
| BTL, | 0 + RT0 0 + RT1 0 + RT2 0 + RT3 0 + RT4 | | / Branch table. Each entry is an 8-bit address |
| | | ⋮ | |
| ERR, | --- | | / Error handling routine. |

NOTE: Since FIM, FIN, and JIN operate with 8-bit addresses, routines ST, BTL, and RT0 through RT4 must all reside in the same page of memory.

If the accumulator held 0100B when location ST was reached, the KBP would convert it to 0011B. The 8 bit address at BTL + 3 would therefore be loaded into registers 0 and 1, and the JIN would cause program control to be transferred to routine RT3.

4.4 LOGICAL OPERATIONS

This section gives three subroutines which produce the logical operations "AND", "OR", and "XOR" (exclusive -OR).

4.4.1 LOGICAL "AND"

The AND function of two bits is given by the following truth table:

| | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

Since any bit ANDed with a zero produces a zero, and any bit ANDed with a one remains unchanged, the AND function is often used to zero groups of bits.

The following subroutine produces the AND, bit by bit, of the two 4-bit quantities held in index registers 0 and 1. The result is placed in register 0, while register 1 is set to 0. Index registers 2 and 3 are also used.

For example, if register 0 = 1110B and register 1 = 0011B, register 0 will be replaced with 0010B.

$$\begin{array}{r} 1110 \text{ B} \\ \text{AND } 0011 \text{ B} \\ \hline 0010 \text{ B} \end{array}$$

The subroutine produces the AND of two bits by placing the bits in the leftmost position of the accumulator and register 2, respectively, and zeroing the rightmost three bits of the accumulator and register 2. Register 2 is then added to the accumulator, and the resulting carry is equal to the AND of the two bits.

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> | |
|--------------|-------------|----------------|-------------------------------------|
| AND , | FIM | 1P 11 | / REG 2 = 0 , REG 3 = 11 |
| L1 , | LDM | 0 | / GET BIT OF REG 0; SET ACC = 0 |
| | XCH | 0 | / REG 0 DATA TO ACC; REG 0 = 0 |
| | RAL | | / 1st 'AND' BIT TO CARRY |
| | XCH | 0 | / SAVE SHIFTED DATA IN REG 0; ACC=0 |
| | INC | 3 | / DONE IF REG 3 = 0 |
| | XCH | 3 | / REG 3 TO ACC |
| | JCN | 4 L2 | / RETURN IF ACC = 0 |
| | XCH | 3 | / OTHERWISE RESTORE ACC AND REG3 |
| | RAR | | / BIT OF REG 0 IS ALONE IN ACC |
| | XCH | 2 | / SAVE 1st 'AND' BIT IN REG 2 |
| | XCH | 1 | / GET BIT OF REG 1 |
| | RAL | | / LEFT BIT TO CARRY |
| | XCH | 1 | / SAVE SHIFTED DATA IN REG 1 |
| | RAR | | / 2ND 'AND' BIT TO ACC |
| | ADD | 2 | / 'ADD' GIVES 'AND' OF THE 2 BITS |
| | JUN | L1 | / IN CARRY |
| L2 , | BBL | 0 | / RETURN TO MAIN PROGRAM. |

4.4.2 LOGICAL "OR"

The OR function of two bits is given by the following truth table:

| | | |
|---|---|---|
| | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 1 | 1 |

Since any bit ORed with a one produces a one, and any bit ORed with a zero remains unchanged, the OR function is often used to set groups of bits to one.

The following subroutine produces the OR, bit by bit, of the two 4-bit quantities held in index registers 0 and 1. The result is placed in register 0, while register 1 is set to 0. Index registers 2 and 3 are also used.

For example, if register 0 = 0100B and register 1 = 0011B, register 0 will be replaced with 0111B.

$$\begin{array}{r}
 & 0100 \text{ B} \\
 \text{OR} & \underline{0011 \text{ B}} \\
 & 0111 \text{ B}
 \end{array}$$

The subroutine produces the OR of two bits by placing the bits in the leftmost position of the accumulator and register 2, respectively, and zeroing the rightmost three bits of the accumulator and register 2. Register 2 is then added to the accumulator. If the resulting carry = 1, the OR of the two bits = 1. If the resulting carry = 0, the OR of the two bits is equal to the leftmost bit of the accumulator.

| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> | |
|--------------|-------------|----------------|-------------------------------------|
| OR, | FIM | 1P 11 | / REG 2 = 0, REG 3 = 11 |
| L1, | LDM | 0 | / GET BIT OF REG 0; SET ACC = 0 |
| | XCH | 0 | / REG 0 DATA TO ACC; REG 0 = 0 |
| | RAL | | / 1st 'OR' BIT TO CARRY |
| | XCH | 0 | / SAVE SHIFTED DATA IN REG 0; ACC=0 |
| | INC | 3 | / DONE IF REG 3 = 0 |
| | XCH | 3 | / REG 3 TO ACC |
| | JCN | 4 L2 | / RETURN IF ACC = 0 |
| | XCH | 3 | / OTHERWISE RESTORE ACC AND REG3 |
| | RAR | | / BIT OF REG 0 IS ALONE IN ACC |
| | XCH | 2 | / SAVE 1st 'OR' BIT IN REG 2 |
| | LDM | 0 | / GET BIT IN REG 1; SET ACC = 0 |
| | XCH | 1 | |
| | RAL | | / LEFT BIT TO CARRY |
| | XCH | 1 | / SAVE SHIFTED DATA IN REG 1 |
| | RAR | | / 2ND 'OR' BIT TO ACC |
| | ADD | 2 | / PRODUCE THE OR OF THE BITS |
| | JCN | 2 L1 | / JUMP IF CARRY = 1 BECAUSE 'OR'=1 |
| | RAL | | / OTHERWISE 'OR' = LEFT BIT OF |
| | JUN | L1 | / ACCUMULATOR |
| L2, | BBL | 0 | / TRANSMIT TO CARRY BY RAL |

4.4.3 LOGICAL "XOR" EXCLUSIVE-OR

The XOR (exclusive -OR) function of two bits is given by the following truth table:

| | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

Since the exclusive OR of two equal bits produces a zero and the exclusive OR of two unequal bits produces a one, the exclusive OR function can be used to test two quantities for equality. If the quantities differ in any bit position, a one will be produced in the result.

The following subroutine produces the exclusive- OR of the two 4-bit quantities held in index registers 0 and 1. The result is placed in register 0, while register 1 is set to 0. Index registers 2 and 3 are also used.

For example if register 0 = 0011B and register 1 = 0010B, register 0 will be replaced with 0001B.

$$\begin{array}{r} 0011 \text{ B} \\ \text{XOR } \underline{0010 \text{ B}} \\ 0001 \text{ B} \end{array}$$

The subroutine produces the XOR of two bits by placing the bits in the leftmost position of the accumulator and register 2, respectively, and zeroing the rightmost three bits of the accumulator and register 2. Register 2 is then added to the accumulator. The XOR of the two bits is then equal to the leftmost bit of the accumulator.

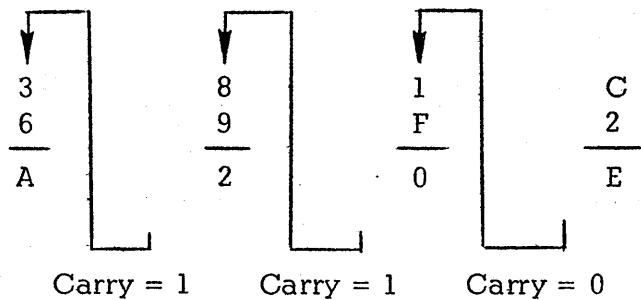
| <u>LABEL</u> | <u>CODE</u> | <u>OPERAND</u> | |
|--------------|-------------|----------------|---------------------------------------|
| XOR, L1, | FIM | 1P 11 | / REG 2 = 0 , REG 3 = 11 |
| | LDM | 0 | / GET BIT OF REG 0; SET ACC =0 |
| | XCH | 0 | / REG 0 DATA TO ACC; REG 0=0 |
| | RAL | | / 1ST XOR BIT TO CARRY |
| | XCH | 0 | / SAVE SHIFTED DATA IN REG 0; ACC = 0 |
| | INC | 3 | / DONE IF REG 3 = 0 |
| | XCH | 3 | / REG 3 TO ACC |
| | JCN | 4 L2 | / RETURN IF ACC = 0. |
| | XCH | 3 | / OTHERWISE RESTORE ACC & REG 3 |
| | RAR | | / BIT OF REG 0 IS ALONE IN ACC |
| | XCH | 2 | / SAVE 1ST XOR BIT IN REG 2 |
| | LDM | 0 | / GET BIT IN REG 1; SET ACC = 0 |
| | XCH | 1 | |
| | RAL | | / LEFT BIT TO CARRY |
| | XCH | 1 | / SAVE SHIFTED DATA IN REG 1 |
| | RAR | | / 2ND 'XOR' BIT TO ACC |
| | ADD | 2 | / PRODUCE THE XOR OF THE BITS |
| | RAL | | / XOR = LEFT BIT OF ACCUM; TRANSMIT |
| JUN | L1 | | / TO CARRY BY RAL. |
| L2, | BBL | 0 | |

4.5 MULTI-DIGIT ADDITION

The carry bit may be used to add unsigned data quantities of arbitrary length. Consider the following addition of two 4-digit hexadecimal numbers:

$$\begin{array}{r} 381C \\ + 69F2 \\ \hline A20E \end{array}$$

This addition may be performed by setting the carry bit = 0, adding the two low-order digits of the numbers, then adding the resulting carry to the two next higher order digits, and so on:



The following subroutine will perform a sixteen digit addition, making these assumptions:

The two numbers to be added are stored in DATA RAM chip 0, registers 0 and 1.

The numbers are stored with the least significant digit first (in character 0).

The result will be stored least significant digit first in register 1, replacing the contents of register 1.

Index register 8 will count the number of digits (up to 16) which have been added.

DATA RAM CHIP 0 BEFORE ADDITION Status Chars.

Register 0

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C | 1 | 8 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | F | 9 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | | | | | | | | | | | | | | | | |
| 3 | | | | | | | | | | | | | | | | |

DATA RAM CHIP 0 AFTER ADDITION

Register 0

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C | 1 | 8 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | E | 0 | 2 | A | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | | | | | | | | | | | | | | | | |
| 3 | | | | | | | | | | | | | | | | |

| | | | | |
|------|-----|----|-----|--|
| AD, | FIM | 2P | 0 | / REG PAIR 2P = RAM CHIP 0 OF / REG 0 |
| | FIM | 3P | 16 | / REG PAIR 3P = RAM CHIP 0 OF / REG 1 |
| | CLB | | | / SET CARRY = 0 |
| | XCH | 8 | | / SET DIGIT COUNTER = 0 |
| AD1, | SRC | 2P | | / SELECT RAM REG 0 |
| | RDM | | | / READ DIGIT TO ACCUMULATOR |
| | SRC | 3P | | / SELECT RAM REG 1 |
| | ADM | | | / ADD DIGIT + CARRY TO ACCUMU- / LATOR |
| | WRM | | | / WRITE RESULT TO REG 1 |
| | INC | 5 | | / ADDRESS NEXT CHAR. OF RAM / REG 0 |
| | INC | 7 | | / ADDRESS NEXT CHAR OF RAM / REG 1 |
| | ISZ | 8 | AD1 | / BRANCH IF DIGIT COUNTER < 16 (NON ZERO) |
| OVR, | BBL | 0 | | |

When location OVR is reached, RAM register 1 will contain the sum of the two 16 digit numbers arranged from low order digit to high order digit. (The reason multi-digit numbers are arranged this way is that it is easier to add numbers from low order to high order digit, and it is easier to increment addresses than to decrement them.

The first time through the program loop, index register pair 2 (index register 4 and 5) contains 0 and index register pair 3 (index registers 6 and 7) contains 16, referencing the first data characters of DATA RAM registers 0 and 1, respectively.

On succeeding repetitions of the loop, index registers 5 and 7 are incremented, referencing sequential data characters, until all 16 digits have been added.

4.6 MULTI-DIGIT SUBTRACTION

The carry bit may be used to subtract unsigned data quantities of arbitrary length. Consider the following subtraction of two 4-digit hexadecimal numbers:

$$\begin{array}{r} 5 \ 4 \ B \ A \\ - 1 \ 4 \ F \ 6 \\ \hline 3 \ F \ C \ 4 \end{array}$$

This subtraction may be performed by first setting the carry bit = 1. Then for each pair of digits, the program must complement the carry bit and perform the subtraction. By this process, the carry bit will adjust the differences, taking into account any borrows which may have occurred.

This process applied to the above subtraction proceeds as follows:

- (1) Set carry bit = 1.
- (2) Complement carry bit. Carry now = 0.
- (3) Subtract low order digits:

$$\begin{array}{r} A = 1010B \\ \overline{6} = 1001B \\ \hline \text{carry} = \underline{\quad\quad\quad} \\ \hline 1 | 0100B = 4 \end{array}$$

(4) Complement resulting carry. Carry now = 0.

(5) Subtract next digits:

$$\begin{array}{r} B = 1011 \text{ B} \\ \overline{F} = 0000 \text{ B} \\ \hline \text{carry} = \underline{\quad} 1 \\ \hline 0 \boxed{1} 100 \text{ B} = \text{CH} \end{array}$$

(6) Complement resulting carry. Carry now = 1.

(7) Subtract next digits:

$$\begin{array}{r} 4 = 0100 \text{ B} \\ \overline{4} = 1011 \text{ B} \\ \hline \text{carry} = \underline{\quad} 0 \\ \hline 0 \boxed{1} 111 \text{ B} = \text{FH} \end{array}$$

(8) Complement resulting carry. Carry now = 1.

(9) Subtract next digits:

$$\begin{array}{r} 5 = 0101 \text{ B} \\ \overline{1} = 1110 \text{ B} \\ \hline \text{carry} = \underline{\quad} 0 \\ \hline 1 \boxed{0} 011 \text{ B} = 3 \end{array}$$

Thus the correct result, 3FC4H, is obtained. The following subroutine will perform a sixteen digit subtraction, making these assumptions:

As in the example of Section 4.2, the two numbers are stored in DATA RAM chip 0, registers 0 and 1 (register 1 containing the subtrahend). The numbers are stored with the least significant digit in character 0, and the result is stored back into register 1. Index register 8 will count the number of digits (up to 16) which have been subtracted.

| | | | | |
|-------|-----|----|-----|--|
| SB, | FIM | 2P | 0 | / REG PAIR 2P = RAM CHIP 0 / REG 0 |
| | FIM | 3P | 16 | / REG PAIR 3P = RAM CHIP 0 / REG 1 |
| | CLB | | | |
| | XCH | 8 | | / SET DIGIT COUNTER = 0 |
| | STC | | | / SET CARRY = 1 |
| SB1 , | CMC | | | / COMPLEMENT CARRY BIT |
| | SRC | 2P | | / SELECT RAM REG 0 |
| | RDM | | | / READ DIGIT TO ACCUMULATOR |
| | SRC | 3P | | / SELECT RAM REG 1 |
| | SBM | | | / SUBTRACT DIGIT AND CARRY / FROM ACCUMULATOR |
| | WRM | | | / WRITE RESULT TO REG 1 |
| | INC | 5 | | / ADDRESS NEXT CHAR. OF RAM / REG 0 |
| | INC | 7 | | / ADDRESS NEXT CHAR. OF RAM / REG 1 |
| | ISZ | 8 | SB1 | / BRANCH IF DIGIT COUNTER / < 16 (NON-ZERO). |
| OV, | BBL | 0 | | |

When location OV is reached, RAM register 1 will contain the difference of the two 16 digit numbers. Note that the carry bit from the previous subtraction is complemented by the CMC instruction each time through the program loop.

4.7 DECIMAL ADDITION

Each 4 bit data quantity may be treated as a decimal number as long as it represents one of the decimal digits from 0 through 9, and does not contain any of the bit patterns representing the hexadecimal digits A through F. In order to preserve this decimal interpretation when performing addition, the value 6 must be added to the accumulator whenever an addition produces a result between 10 and 15. This is because each 4 bit data quantity can hold 6 more combinations of bits than there are decimal digits.

The DAA (decimal adjust accumulator) instruction is provided for this purpose. Also, to permit addition of multi-digit decimal numbers, the DAA adds 6 to the accumulator whenever the carry bit is set indicating a decimal carry from previous additions. The carry bit is unaffected unless the addition of 6 produces a carry, in which case the carry bit is set.

To perform the decimal addition:

$$\begin{array}{r} 469 \\ + 329 \\ \hline 798 \end{array}$$

the process works as follows.

- (1) Clear the carry and add the lowest-order digits

$$\begin{array}{r} 9 = 1001 \text{ B} \\ 9 = 1001 \text{ B} \\ \text{carry} = \underline{\quad} 0 \\ \hline 1 \underline{0} 0010 \text{ B} \\ \text{Carry} = 1 \end{array}$$

- (2) Perform a DAA operation, which will add 6 to the accumulator. Since no carry is produced by this operation, the carry bit is left unaffected, remaining = 1.

$$\begin{array}{r}
 \text{Accum.} = 0010 \text{ B} \\
 6 = 0110 \text{ B} \\
 \text{Carry} = \underline{0} \\
 \hline
 0 | 1000 \text{ B} = 8
 \end{array}$$

- (3) Add the next two digits.

$$\begin{array}{r}
 6 = 0110 \text{ B} \\
 2 = 0010 \text{ B} \\
 \text{Carry} = \underline{1} \\
 \hline
 0 | 1001 \text{ B} = 9
 \end{array}$$

↓
Carry = 0

- (4) Perform a DAA operation. Since the accumulator is not greater than 9 and the carry is not set, no action occurs.

- (5) Add the next two digits:

$$\begin{array}{r}
 4 = 0100 \text{ B} \\
 3 = 0011 \text{ B} \\
 \text{Carry} = \underline{0} \\
 \hline
 0 | 0111 \text{ B} = 7
 \end{array}$$

↓
Carry = 0

- (6) Perform a DAA operation. Again, no action occurs. Thus the correct decimal result 798 is generated in three 4 bit data characters.

A subroutine which adds two 16 digit decimal numbers, then, is exactly analogous to the 16 digit hexadecimal addition subroutine of Section 4.2, and may be produced by inserting the instruction DAA after the ADM instruction of that example.

4.8 DECIMAL SUBTRACTION

Each 4 bit data quantity may be treated as a decimal number as long as it represents one of the decimal digits 0 through 9. The TCS (transfer carry subtract) and DAA (decimal adjust accumulator) may be used to subtract two decimal numbers and produce a decimal number. In fact, the TCS instruction permits subtraction of multi-digit decimal numbers.

The process consists of generating the ten's complement of the subtrahend digit (the difference between the subtrahend digit and 10 decimal), and adding the result to the minuend digit. For instance, to subtract 2 from 7, the ten's complement of 2 ($10-2=8$) is added to 7, producing 15 decimal which, when truncated to a 4 bit quantity gives 5 (the required result). If a borrow was generated by the previous subtraction, the 9's complement of the subtrahend digit is produced to compensate for the borrow.

In detail, the procedure for subtracting one multi-digit decimal number from another is as follows:

- (1) Set the carry bit = 1 indicating no borrow.
- (2) Use the TCS instruction to set the accumulator to either 9 or 10 decimal.
- (3) Subtract the subtrahend digit from the accumulator, producing either the 9's or 10's complement.
- (4) Set the carry bit = 0.
- (5) Add the minuend digit to the accumulator.
- (6) Use the DAA instruction to make sure the result in the accumulator is in decimal format, and to indicate a borrow in the carry bit if one occurred.
Save this result.
- (7) If there are more digits to subtract, go to step 2.
Otherwise stop.

Example: Perform the decimal subtraction

$$\begin{array}{r} 51 \\ - 38 \\ \hline 13 \end{array}$$

- (1) Set carry = 1.
- (2) TCS sets accumulator = 1010B and carry = 0.
- (3) Subtract the subtrahend digit 8 from the accumulator.

$$\begin{array}{r} \text{Accumulator} = 1010 \text{ B} \\ \underline{-} \quad \underline{8} = 0111 \text{ B} \\ \text{Carry} = \underline{\underline{1}} \\ \hline 0010 \text{ B} \end{array}$$

- (4) Set carry = 0.
- (5) Add minuend digit 1 to accumulator.

$$\begin{array}{r} \text{Accumulator} = 0010 \text{ B} \\ 1 = 0001 \text{ B} \\ \text{Carry} = \underline{\underline{0}} \\ \hline \end{array}$$

0 | 0011 B = 3

← Carry = 0

- (6) DAA leaves accumulator = 3 = first digit of result, and carry = 0, indicating that a borrow occurred.
- (7) TCS sets accumulator = 1001B and carry = 0.
- (8) Subtract the subtrahend digit 3 from the accumulator.

$$\begin{array}{r} \text{Accumulator} = 1001 \text{ B} \\ \underline{-} \quad \underline{3} = 1100 \text{ B} \\ \text{Carry} = \underline{\underline{1}} \\ \hline 0110 \text{ B} \end{array}$$

(9) Set carry = 0.

(10) Add minuend digit 5 to accumulator.

$$\begin{array}{r} \text{Accumulator} = 0110 \text{ B} \\ 5 = 0101 \text{ B} \\ \text{Carry} = \underline{0} \\ \hline 01011 \text{ B} \\ \text{Carry} = 0 \end{array}$$

(11) DAA adds 6 to accumulator and sets carry = 1, indicating that no borrow occurred.

$$\begin{array}{r} \text{Accumulator} = 1011 \text{ B} \\ 6 = \underline{0110} \text{ B} \\ \hline 10001 \text{ B} = 1 = \text{Second digit of result.} \\ \text{Carry} = 1 \end{array}$$

Therefore the result of subtracting 38 from 51 is 13.

The following subroutine will subtract one 16 digit decimal number from another, using the following assumptions.

The minuend is stored least significant digit first in DATA RAM chip 0, register 0.

The subtrahend is stored least significant digit first in DATA RAM chip 0, register 1.

The result will be stored least significant digit first in DATA RAM chip 0, register 0, replacing the minuend.

Index register 8 will count the number of digits (up to 16) which have been subtracted.

| | | | | |
|-------|-----|----|-----|-----------------------------------|
| SD , | FIM | 2P | 0 | / REG PAIR 2P = RAM CHIP 0, REG 0 |
| | FIM | 3P | 16 | / REG PAIR 3P = RAM CHIP 0 |
| | | | | / REG 1 |
| | CLB | | | |
| | XCH | 8 | | / SET DIGIT COUNTER = 0 |
| | STC | | | / SET CARRY = 1 |
| SD1 , | TCS | | | / ACCUMULATOR = 9 OR 10 |
| | SRC | 3P | | / SELECT RAM REG 1 |
| | SBM | | | / PRODUCE 9's OR 10's |
| | | | | / COMPLEMENT |
| | CLC | | | / SET CARRY = 0 |
| | SRC | 2P | | / SELECT RAM REG 0 |
| | ADM | | | / ADD MINUEND TO ACCUMU- |
| | | | | / LATOR |
| | DAA | | | / ADJUST ACCUMULATOR |
| | WRM | | | / WRITE RESULT TO REG 0 |
| | INC | 5 | | / ADDRESS NEXT CHAR. OF RAM |
| | | | | / REG 0 |
| | INC | 7 | | / ADDRESS NEXT CHAR. OF RAM |
| | | | | / REG 1 |
| | ISZ | 8 | SD1 | / BRANCH IF DIGIT COUNTER < 16 |
| | | | | / (NON-ZERO). |
| DN , | BBL | 0 | | |

4.9 FLOWING POINT NUMBERS

The structure of DATA RAM chips is fully described in Section 2.3.3.

One use to which a 16-character DATA RAM register and its 4 status characters can be put is to store a 16 digit decimal floating point number.

Such a number can be represented in the form:

$$\pm . \text{DDDDDDDDDDDDDDDD} * 10^{\text{EE}}$$

The 16 data characters of a RAM register could then be used to store the digits of the number, two status characters could be used to hold the digits of the exponent, while the remaining two status characters would hold the signs of the number and its exponent.

If a value of one is chosen to represent minus and a value of zero is chosen to represent plus, status characters 0 and 1 hold the exponent digits, status character 2 holds the exponent sign and status character 3 holds the number's sign, then the number

$$+.1234567890812489 \times 10^{-23}$$

would appear in a RAM register as follows:

RAM CHIP

RAM REGISTER 0
RAM REGISTER 1
RAM REGISTER 2
RAM REGISTER 3

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 8 | 1 | 2 | 4 | 8 | 9 | 2 | 3 | 1 | 0 |
| | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | |

DATA CHARACTERS

Status Characters

APPENDIX "A"

-- INSTRUCTION SUMMARY --

This appendix provides a summary of 4004 instructions. Abbreviations used are as follows:

| ABBREVIATION | DESCRIPTION |
|----------------|---|
| A | The accumulator. |
| A _n | Bit n in the accumulator, where n may have any value from 0 to 3. |
| ADDR | A read-only memory or program random-access memory address. |
| carry | The carry bit. |
| PC | The 12 bit Program Counter. |
| PCH | The high-order 4 bits of the Program Counter. |
| PCL | The low-order 4 bits of the Program Counter. |
| PCM | The middle 4 bits of the Program Counter. |
| RAM | Random-access memory. |
| REG | Any index register from 0 to 15. |
| R0 | Index register 0. |
| R1 | Index register 1. |
| ROM | Read-only memory. |
| RP | Any index register pair from 0P to 7P. |
| STK | The address stack. |
| value | The number obtained by complementing each bit of value. |

(Continued):

| ABBREVIATION | DESCRIPTION |
|--------------|--|
| X:Y | The value obtained by concatenating the values X and Y. |
| [] | An optional field enclosed by brackets. |
| () | Contents of register or memory enclosed by parentheses. |
| ← | Replace value on left hand side of arrow with value on right hand side of arrow. |

A.1 INDEX REGISTER INSTRUCTIONS

Format:

[LABEL,] FIN RP
-- or --
[LABEL,] INC REG

| Code | Description | |
|------|---------------------------------|---|
| FIN | $(RP) \leftarrow ((PCH:R0:RI))$ | Load RP with 8 bits of ROM data addressed by register pair 0. |
| INC | $(REG) \leftarrow (REG) + 1$ | Increment register REG. |

A.2 INDEX REGISTER TO ACCUMULATOR INSTRUCTIONS

Format:

[LABEL,] CODE REG

| Code | Description | |
|------|---|--|
| ADD | $(A) \leftarrow (A) + (REG) + (\text{carry})$ | Add REG plus carry bit to accumulator. |
| SUB | $(A) \leftarrow (A) + (\overline{REG}) + (\overline{\text{carry}})$ | Subtract REG from accumulator with borrow. |
| LD | $(A) \leftarrow (REG)$ | Load accumulator from REG. |
| XCH | $(A) \leftrightarrow (REG)$ | Exchange contents of accumulator and REG. |

A.3 ACCUMULATOR INSTRUCTIONS

Format:

[LABEL] CODE

| Code | Description |
|------|---|
| CLB | (A) \leftarrow 0, (carry) \leftarrow 0 |
| CLC | (carry) \leftarrow 0 |
| IAC | (A) \leftarrow (A) + 1 |
| CMC | (carry) \leftarrow $\overline{\text{carry}}$ |
| CMA | (A) \leftarrow $\overline{(A)}$ |
| RAL | $A_{n+1} \leftarrow A_n$, (carry) $\leftarrow A_3$, $A_0 \leftarrow$ (carry) |
| RAR | $A_n \leftarrow A_{n+1}$, (carry) $\leftarrow A_0$, $A_3 \leftarrow$ (carry) |
| TCC | (A) \leftarrow 0, $A_0 \leftarrow$ (carry), (carry) \leftarrow 0 |
| DAC | (A) \leftarrow (A) - 1 |
| TCS | If (carry) = 0, (A) $\leftarrow 9_{10}$ If (carry) = 1, (A) $\leftarrow 10_{10}$ (carry) \leftarrow 0 |
| STC | (carry) \leftarrow 1 |
| DAA | If (A) $> 9_{10}$ or (carry) $= 1$, (A) $\leftarrow (A) + 6$ |
| KBP | Convert $A_3\ A_2\ A_1\ A_0$ |
| | Convert accumulator from 1 of n code to binary value. |

A.4 IMMEDIATE INSTRUCTIONS

Format:

[LABEL,] FIM RP DATA
--- or ---
[LABEL,] LDM DATA

| Code | Description | |
|------|-------------|--|
| FIM | (RP) ← DATA | Load 8 bit immediate DATA into register pair RP. |
| LDM | (A) ← DATA | Load 4-bit immediate DATA into the accumulator. |

A.5 TRANSFER OF CONTROL INSTRUCTIONS

Format:

[LABEL,] JCN CN ADDR
--- or ---
[LABEL,] JIN RP
--- or ---
[LABEL,] ISZ REG
--- or ---
[LABEL,] JUN ADDR

| Code | Description | |
|------|--|--|
| JUN | (PCH: PCM: PCL) ← ADDR | Jump to location ADDR. |
| JIN | (PCM: PCL) ← (RP) | Jump to the address in register pair RP. |
| JCN | If CN true, (PCM:PCL) ← ADDR If CN false, (PL) ← (PL) + 2 | Jump to ADDR if condition true. |
| ISZ | (REG) ← (REG) + 1 If result = 0, (PL) ← (PL) + 2 If result = 1, (PCM:PCL) ← ADDR | Increment REG. If zero, skip. If non zero, jump to ADDR |

A.6 SUBROUTINE LINKAGE INSTRUCTIONS

Format:

[LABEL,] JMS ADDR

-- or --

[LABEL,] BBL DATA

| Code | Description | |
|------|---------------------------|--|
| JMS | (STK) ← (PC), (PC) ← ADDR | Call subroutine and push return address onto stack. |
| BBL | (PC) ← (STK), (A) ← DATA | Return from subroutine and load accumulator with immediate DATA. |

A.7 NOP INSTRUCTION

Format:

[LABEL,] NOP

| Code | Description |
|------|-----------------------|
| NOP | ----- No operation |

A.8 MEMORY SELECTION INSTRUCTIONS

Format:

[LABEL,] SRC RP

--- or ---

[LABEL,] DCL

| Code | Description | |
|------|--|--|
| SRC | DATA BUS ← (RP) | Contents of RP select a RAM or ROM address to be used by I/O and RAM instructions. |
| DCL | CPU ← A ₂ : A ₁ : A ₀ | Select a particular RAM bank. |

A.9 I/O AND RAM INSTRUCTIONS

Format:

[LABEL,] CODE

| Code | Description |
|------|---|
| WRM | (RAM) ← A Write accumulator to RAM. |
| WMP | RAM output port ← (A) Write accumulator to RAM output port. |
| WRR | ROM output port ← (A) Write accumulator to ROM output port. |
| WPM | (PRAM) ← (A) Write accumulator to Program RAM. |
| WRn | RAM status character n ← (A) Write accumulator to RAM status character n (n = 0, 1, 2 or 3). |
| RDM | (A) ← RAM Load accumulator from RAM. |
| RDR | (A) ← ROM input port Load accumulator from ROM input port. |
| RDn | (A) ← RAM status character n Load accumulator from RAM status character n (n = 0, 1, 2, or 3). |
| ADM | (A) ← (A) + (RAM) + (carry) Add RAM data plus carry to accumulator. |
| SBM | (A) ← (A) + (RAM) + (carry) Subtract RAM data from accumulator with borrow. |

APPENDIX "B"

-- INSTRUCTION MACHINE CODES --

In order to help the programmer examine memory when debugging programs, this appendix provides the assembly language instruction represented by each of the 256 possible instruction code bytes.

Where an instruction occupies two bytes, only the first (code) byte is given.

| DEC | OCTAL | HEX | MNEMONIC | COMMENT |
|-----|-------|-----|----------|---------|
| 0 | 000 | 00 | NOP | |
| 1 | 001 | 01 | --- | |
| 2 | 002 | 02 | --- | |
| 3 | 003 | 03 | --- | |
| 4 | 004 | 04 | --- | |
| 5 | 005 | 05 | --- | |
| 6 | 006 | 06 | --- | |
| 7 | 007 | 07 | --- | |
| 8 | 010 | 08 | --- | |
| 9 | 011 | 09 | --- | |
| 10 | 012 | 0A | --- | |
| 11 | 013 | 0B | --- | |
| 12 | 014 | 0C | --- | |
| 13 | 015 | 0D | --- | |
| 14 | 016 | 0E | --- | |
| 15 | 017 | 0F | --- | |
| 16 | 020 | 10 | JCN | CN = 0 |
| 17 | 021 | 11 | JCN | CN = 1 |
| 18 | 022 | 12 | JCN | CN = 2 |
| 19 | 023 | 13 | JCN | CN = 3 |
| 20 | 024 | 14 | JCN | CN = 4 |
| 21 | 025 | 15 | JCN | CN = 5 |
| 22 | 026 | 16 | JCN | CN = 6 |
| 23 | 027 | 17 | JCN | CN = 7 |
| 24 | 030 | 18 | JCN | CN = 8 |
| 25 | 031 | 19 | JCN | CN = 9 |
| 26 | 032 | 1A | JCN | CN = 10 |
| 27 | 033 | 1B | JCN | CN = 11 |
| 28 | 034 | 1C | JCN | CN = 12 |
| 29 | 035 | 1D | JCN | CN = 13 |
| 30 | 036 | 1E | JCN | CN = 14 |
| 31 | 037 | 1F | JCN | CN = 15 |
| 32 | 040 | 20 | FIM | 0P |
| 33 | 041 | 21 | SRC | 0P |
| 34 | 042 | 22 | FIM | 1P |
| 35 | 043 | 23 | SRC | 1P |
| 36 | 044 | 24 | FIM | 2P |
| 37 | 045 | 25 | SRC | 2P |
| 38 | 046 | 26 | FIM | 3P |
| 39 | 047 | 27 | SRC | 3P |
| 40 | 050 | 28 | FIM | 4P |

| DEC | OCTAL | HEX | MNEMONIC | COMMENT |
|-----|-------|-----|----------|---|
| 41 | 051 | 29 | SRC | 4P |
| 42 | 052 | 2A | FIM | 5P |
| 43 | 053 | 2B | SRC | 5P |
| 44 | 054 | 2C | FIM | 6P |
| 45 | 055 | 2D | SRC | 6P |
| 46 | 056 | 2E | FIM | 7P |
| 47 | 057 | 2F | SRC | 7P |
| 48 | 060 | 30 | FIN | 0P |
| 49 | 061 | 31 | JIN | 0P |
| 50 | 062 | 32 | FIN | 1P |
| 51 | 063 | 33 | JIN | 1P |
| 52 | 064 | 34 | FIN | 2P |
| 53 | 065 | 35 | JIN | 2P |
| 54 | 066 | 36 | FIN | 3P |
| 55 | 067 | 37 | JIN | 3P |
| 56 | 070 | 38 | FIN | 4P |
| 57 | 071 | 39 | JIN | 4P |
| 58 | 072 | 3A | FIN | 5P |
| 59 | 073 | 3B | JIN | 5P |
| 60 | 074 | 3C | FIN | 6P |
| 61 | 075 | 3D | JIN | 6P |
| 62 | 076 | 3E | FIN | 7P |
| 63 | 077 | 3F | JIN | 7P |
| 64 | 100 | 40 | JUN | Second hex digit is part of jump address. |
| 65 | 101 | 41 | JUN | |
| 66 | 102 | 42 | JUN | |
| 67 | 103 | 43 | JUN | |
| 68 | 104 | 44 | JUN | |
| 69 | 105 | 45 | JUN | |
| 70 | 106 | 46 | JUN | |
| 71 | 107 | 47 | JUN | |
| 72 | 110 | 48 | JUN | |
| 73 | 111 | 49 | JUN | |
| 74 | 112 | 4A | JUN | |
| 75 | 113 | 4B | JUN | |
| 76 | 114 | 4C | JUN | |
| 77 | 115 | 4D | JUN | |
| 78 | 116 | 4E | JUN | |
| 79 | 117 | 4F | JUN | |
| 80 | 120 | 50 | JMS | |
| 81 | 121 | 51 | JMS | |

| DEC | OCTAL | HEX | MNEMONIC | COMMENT |
|-----|-------|-----|----------|---------|
| 82 | 122 | 52 | JMS | |
| 83 | 123 | 53 | JMS | |
| 84 | 124 | 54 | JMS | |
| 85 | 125 | 55 | JMS | |
| 86 | 126 | 56 | JMS | |
| 87 | 127 | 57 | JMS | |
| 88 | 130 | 58 | JMS | |
| 89 | 131 | 59 | JMS | |
| 90 | 132 | 5A | JMS | |
| 91 | 133 | 5B | JMS | |
| 92 | 134 | 5C | JMS | |
| 93 | 135 | 5D | JMS | |
| 94 | 136 | 5E | JMS | |
| 95 | 137 | 5F | JMS | |
| 96 | 140 | 60 | INC 0 | |
| 97 | 141 | 61 | INC 1 | |
| 98 | 142 | 62 | INC 2 | |
| 99 | 143 | 63 | INC 3 | |
| 100 | 144 | 64 | INC 4 | |
| 101 | 145 | 65 | INC 5 | |
| 102 | 146 | 66 | INC 6 | |
| 103 | 147 | 67 | INC 7 | |
| 104 | 150 | 68 | INC 8 | |
| 105 | 151 | 69 | INC 9 | |
| 106 | 152 | 6A | INC 10 | |
| 107 | 153 | 6B | INC 11 | |
| 108 | 154 | 6C | INC 12 | |
| 109 | 155 | 6D | INC 13 | |
| 110 | 156 | 6E | INC 14 | |
| 111 | 157 | 6F | INC 15 | |
| 112 | 160 | 70 | ISZ 0 | |
| 113 | 161 | 71 | ISZ 1 | |
| 114 | 162 | 72 | ISZ 2 | |
| 115 | 163 | 73 | ISZ 3 | |
| 116 | 164 | 74 | ISZ 4 | |
| 117 | 165 | 75 | ISZ 5 | |
| 118 | 166 | 76 | ISZ 6 | |
| 119 | 167 | 77 | ISZ 7 | |
| 120 | 170 | 78 | ISZ 8 | |
| 121 | 171 | 79 | ISZ 9 | |
| 122 | 172 | 7A | ISZ 10 | |

| DEC | OCTAL | HEX | MNEMONIC | COMMENT |
|-----|-------|-----|----------|---------|
| 123 | 173 | 7B | ISZ | 11 |
| 124 | 174 | 7C | ISZ | 12 |
| 125 | 175 | 7D | ISZ | 13 |
| 126 | 176 | 7E | ISZ | 14 |
| 127 | 177 | 7F | ISZ | 15 |
| 128 | 200 | 80 | ADD | 0 |
| 129 | 201 | 81 | ADD | 1 |
| 130 | 202 | 82 | ADD | 2 |
| 131 | 203 | 83 | ADD | 3 |
| 132 | 204 | 84 | ADD | 4 |
| 133 | 205 | 85 | ADD | 5 |
| 134 | 206 | 86 | ADD | 6 |
| 135 | 207 | 87 | ADD | 7 |
| 136 | 210 | 88 | ADD | 8 |
| 137 | 211 | 89 | ADD | 9 |
| 138 | 212 | 8A | ADD | 10 |
| 139 | 213 | 8B | ADD | 11 |
| 140 | 214 | 8C | ADD | 12 |
| 141 | 215 | 8D | ADD | 13 |
| 142 | 216 | 8E | ADD | 14 |
| 143 | 217 | 8F | ADD | 15 |
| 144 | 220 | 90 | SUB | 0 |
| 145 | 221 | 91 | SUB | 1 |
| 146 | 222 | 92 | SUB | 2 |
| 147 | 223 | 93 | SUB | 3 |
| 148 | 224 | 94 | SUB | 4 |
| 149 | 225 | 95 | SUB | 5 |
| 150 | 226 | 96 | SUB | 6 |
| 151 | 227 | 97 | SUB | 7 |
| 152 | 230 | 98 | SUB | 8 |
| 153 | 231 | 99 | SUB | 9 |
| 154 | 232 | 9A | SUB | 10 |
| 155 | 233 | 9B | SUB | 11 |
| 156 | 234 | 9C | SUB | 12 |
| 157 | 235 | 9D | SUB | 13 |
| 158 | 236 | 9E | SUB | 14 |
| 159 | 237 | 9F | SUB | 15 |
| 160 | 240 | A0 | LD | 0 |
| 161 | 241 | A1 | LD | 1 |
| 162 | 242 | A2 | LD | 2 |
| 163 | 243 | A3 | LD | 3 |

| DEC | OCTAL | HEX | MNEMONIC | COMMENT |
|-----|-------|-----|----------|---------|
| 164 | 244 | A4 | LD | 4 |
| 165 | 245 | A5 | LD | 5 |
| 166 | 246 | A6 | LD | 6 |
| 167 | 247 | A7 | LD | 7 |
| 168 | 250 | A8 | LD | 8 |
| 169 | 251 | A9 | LD | 9 |
| 170 | 252 | AA | LD | 10 |
| 171 | 253 | AB | LD | 11 |
| 172 | 254 | AC | LD | 12 |
| 173 | 255 | AD | LD | 13 |
| 174 | 256 | AE | LD | 14 |
| 175 | 257 | AF | LD | 15 |
| 176 | 260 | B0 | XCH | 0 |
| 177 | 261 | B1 | XCH | 1 |
| 178 | 262 | B2 | XCH | 2 |
| 179 | 263 | B3 | XCH | 3 |
| 180 | 264 | B4 | XCH | 4 |
| 181 | 265 | B5 | XCH | 5 |
| 182 | 266 | B6 | XCH | 6 |
| 183 | 267 | B7 | XCH | 7 |
| 184 | 270 | B8 | XCH | 8 |
| 185 | 271 | B9 | XCH | 9 |
| 186 | 272 | BA | XCH | 10 |
| 187 | 273 | BB | XCH | 11 |
| 188 | 274 | BC | XCH | 12 |
| 189 | 275 | BD | XCH | 13 |
| 190 | 276 | BE | XCH | 14 |
| 191 | 277 | BF | XCH | 15 |
| 192 | 300 | C0 | BBL | 0 |
| 193 | 301 | C1 | BBL | 1 |
| 194 | 302 | C2 | BBL | 2 |
| 195 | 303 | C3 | BBL | 3 |
| 196 | 304 | C4 | BBL | 4 |
| 197 | 305 | C5 | BBL | 5 |
| 198 | 306 | C6 | BBL | 6 |
| 199 | 307 | C7 | BBL | 7 |
| 200 | 310 | C8 | BBL | 8 |
| 201 | 311 | C9 | BBL | 9 |
| 202 | 312 | CA | BBL | 10 |
| 203 | 313 | CB | BBL | 11 |
| 204 | 314 | CC | BBL | 12 |

| DEC | OCTAL | HEX | MNEMONIC | COMMENT |
|-----|-------|-----|----------|---------|
| 205 | 315 | CD | BBL | 13 |
| 206 | 316 | CE | BBL | 14 |
| 207 | 317 | CF | BBL | 15 |
| 208 | 320 | D0 | LDM | 0 |
| 209 | 321 | D1 | LDM | 1 |
| 210 | 322 | D2 | LDM | 2 |
| 211 | 323 | D3 | LDM | 3 |
| 212 | 324 | D4 | LDM | 4 |
| 213 | 325 | D5 | LDM | 5 |
| 214 | 326 | D6 | LDM | 6 |
| 215 | 327 | D7 | LDM | 7 |
| 216 | 330 | D8 | LDM | 8 |
| 217 | 331 | D9 | LDM | 9 |
| 218 | 332 | DA | LDM | 10 |
| 219 | 333 | DB | LDM | 11 |
| 220 | 334 | DC | LDM | 12 |
| 221 | 335 | DD | LDM | 13 |
| 222 | 336 | DE | LDM | 14 |
| 223 | 337 | DF | LDM | 15 |
| 224 | 340 | E0 | WRM | |
| 225 | 341 | E1 | WMP | |
| 226 | 342 | E2 | WRR | |
| 227 | 343 | E3 | WPM | |
| 228 | 344 | E4 | WR0 | |
| 229 | 345 | E5 | WR1 | |
| 230 | 346 | E6 | WR2 | |
| 231 | 347 | E7 | WR3 | |
| 232 | 350 | E8 | SBM | |
| 233 | 351 | E9 | RDM | |
| 234 | 352 | EA | RDR | |
| 235 | 353 | EB | ADM | |
| 236 | 354 | EC | RD0 | |
| 237 | 355 | ED | RD1 | |
| 238 | 356 | EE | RD2 | |
| 239 | 357 | EF | RD3 | |
| 240 | 360 | F0 | CLB | |
| 241 | 361 | F1 | CLC | |
| 242 | 362 | F2 | IAC | |
| 243 | 363 | F3 | CMC | |
| 244 | 364 | F4 | CMA | |
| 245 | 365 | F5 | RAL | |

| DEC | OCTAL | HEX | MNEMONIC | COMMENT |
|-----|-------|-----|----------|---------|
| 246 | 366 | F6 | RAR | |
| 247 | 367 | F7 | TCC | |
| 248 | 370 | F8 | DAC | |
| 249 | 371 | F9 | TCS | |
| 250 | 372 | FA | STC | |
| 251 | 373 | FB | DAA | |
| 252 | 374 | FC | KBP | |
| 253 | 375 | FD | DCL | |
| 254 | 376 | FE | --- | |
| 255 | 377 | FF | --- | |

APPENDIX "C"

-- ASCII TABLE --

The 4004 uses a seven-bit ASCII code, which is the normal 8 bit ASCII code with the parity (high order) bit always reset.

| Graphic or Control | ASCII (Hexadecimal) |
|--------------------|---------------------|
| NULL | 00 |
| SOM | 01 |
| EOA | 02 |
| EOM | 03 |
| EOT | 04 |
| WRU | 05 |
| RU | 06 |
| BELL | 07 |
| FE | 08 |
| H.Tab | 09 |
| Line Feed | 0A |
| V. Tab | 0B |
| Form | 0C |
| Return | 0D |
| SO | 0E |
| SI | 0F |
| DCO | 10 |
| X-On | 11 |
| Tape Aux. On | 12 |
| X-Off | 13 |
| Tape Aux. Off | 14 |
| Error | 15 |
| Sync | 16 |
| LEM | 17 |
| S0 | 18 |
| S1 | 19 |
| S2 | 1A |
| S3 | 1B |
| S4 | 1C |
| S5 | 1D |
| S6 | 1E |
| S7 | 1F |

| Graphic or Control | ASCII Hexadecimal |
|--------------------|-------------------|
| ACK | 7C |
| Alt. Mode | 7D |
| Rubout | 7F |
| ! | 21 |
| " | 22 |
| # | 23 |
| \$ | 24 |
| % | 25 |
| & | 26 |
| - | 27 |
| (| 28 |
|) | 29 |
| * | 2A |
| + | 2B |
| : | 2C |
| - | 2D |
| : | 2E |
| / | 2F |
| : | 3A |
| : | 3B |
| < | 3C |
| = | 3D |
| > | 3E |
| ? | 3F |
| [| 5B |
| / | 5C |
|] | 5D |
| ↑ | 5E |
| ← | 5F |
| @ | 40 |
| blank | 20 |
| 0 | 30 |
| 1 | 31 |
| 2 | 32 |
| 3 | 33 |
| 4 | 34 |
| 5 | 35 |
| 6 | 36 |
| 7 | 37 |
| 8 | 38 |
| 9 | 39 |

| Graphic or Control | ASCII Hexadecimal |
|--------------------|-------------------|
| A | 41 |
| B | 42 |
| C | 43 |
| D | 44 |
| E | 45 |
| F | 46 |
| G | 47 |
| H | 48 |
| I | 49 |
| J | 4A |
| K | 4B |
| L | 4C |
| M | 4D |
| N | 4E |
| O | 4F |
| P | 50 |
| Q | 51 |
| R | 52 |
| S | 53 |
| T | 54 |
| U | 55 |
| V | 56 |
| W | 57 |
| X | 58 |
| Y | 59 |
| Z | 5A |

APPENDIX "D"

-- BINARY-DECIMAL-HEXADECIMAL CONVERSION TABLES --

HEXADECIMAL ARITHMETIC

ADDITION TABLE

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F | 10 |
| 2 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F | 10 | 11 |
| 3 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F | 10 | 11 | 12 |
| 4 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F | 10 | 11 | 12 | 13 |
| 5 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F | 10 | 11 | 12 | 13 | 14 |
| 6 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F | 10 | 11 | 12 | 13 | 14 | 15 |
| 7 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 8 | 09 | 0A | 0B | 0C | 0D | 0E | 0F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 9 | 0A | 0B | 0C | 0D | 0E | 0F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| A | 0B | 0C | 0D | 0E | 0F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| B | 0C | 0D | 0E | 0F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A |
| C | 0D | 0E | 0F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B |
| D | 0E | 0F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C |
| E | 0F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D |
| F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D | 1E |

MULTIPLICATION TABLE

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 2 | 04 | 06 | 08 | 0A | 0C | 0E | 10 | 12 | 14 | 16 | 18 | 1A | 1C | 1E |
| 3 | 06 | 09 | 0C | 0F | 12 | 15 | 18 | 18 | 1E | 21 | 24 | 27 | 2A | 2D |
| 4 | 08 | 0C | 10 | 14 | 18 | 1C | 20 | 24 | 28 | 2C | 30 | 34 | 38 | 3C |
| 5 | 0A | 0F | 14 | 19 | 1E | 23 | 28 | 2D | 32 | 37 | 3C | 41 | 46 | 4B |
| 6 | 0C | 12 | 18 | 1E | 24 | 2A | 30 | 36 | 3C | 42 | 48 | 4E | 54 | 5A |
| 7 | 0E | 15 | 1C | 23 | 2A | 31 | 38 | 3F | 46 | 4D | 54 | 5B | 62 | 69 |
| 8 | 10 | 18 | 20 | 28 | 30 | 38 | 40 | 48 | 50 | 58 | 60 | 68 | 70 | 78 |
| 9 | 12 | 1B | 24 | 2D | 36 | 3F | 48 | 51 | 5A | 63 | 6C | 75 | 7E | 87 |
| A | 14 | 1E | 28 | 32 | 3C | 46 | 50 | 5A | 64 | 6E | 78 | 82 | 8C | 96 |
| B | 16 | 21 | 2C | 37 | 42 | 4D | 58 | 63 | 6E | 79 | 84 | 8F | 9A | A5 |
| C | 18 | 24 | 30 | 3C | 48 | 54 | 60 | 6C | 78 | 84 | 90 | 9C | A8 | B4 |
| D | 1A | 27 | 34 | 41 | 4E | 58 | 68 | 75 | 82 | 8F | 9C | A9 | B6 | C3 |
| E | 1C | 2A | 38 | 46 | 54 | 62 | 70 | 7E | 8C | 9A | A8 | B6 | C4 | D2 |
| F | 1E | 2D | 3C | 48 | 5A | 69 | 78 | 87 | 96 | A5 | B4 | C3 | D2 | E1 |

POWERS OF TWO

| <u>2^n</u> | <u>n</u> | <u>2^{-n}</u> |
|---------------------------|-----------------------|---|
| 1 | 0 | 1.0 |
| 2 | 1 | 0.5 |
| 4 | 2 | 0.25 |
| 8 | 3 | 0.125 |
| 16 | 4 | 0.062 5 |
| 32 | 5 | 0.031 25 |
| 64 | 6 | 0.015 625 |
| 128 | 7 | 0.007 812 5 |
| 256 | 8 | 0.003 906 25 |
| 512 | 9 | 0.001 953 125 |
| 1 024 | 10 | 0.000 976 562 5 |
| 2 048 | 11 | 0.000 488 281 25 |
| 4 096 | 12 | 0.000 244 140 625 |
| 8 192 | 13 | 0.000 122 070 312 5 |
| 16 384 | 14 | 0.000 061 035 156 25 |
| 32 768 | 15 | 0.000 030 517 578 125 |
| 65 536 | 16 | 0.000 015 258 789 062 5 |
| 131 072 | 17 | 0.000 007 629 394 531 25 |
| 262 144 | 18 | 0.000 003 814 697 265 625 |
| 524 288 | 19 | 0.000 001 907 348 632 812 5 |
| 1 048 576 | 20 | 0.000 000 953 674 316 406 25 |
| 2 097 152 | 21 | 0.000 000 476 837 158 203 125 |
| 4 194 304 | 22 | 0.000 000 238 418 579 101 562 5 |
| 8 388 608 | 23 | 0.000 000 119 209 209 550 781 25 |
| 16 777 216 | 24 | 0.000 000 059 604 644 775 390 625 |
| 33 554 432 | 25 | 0.000 000 029 802 322 387 495 312 5 |
| 67 108 864 | 26 | 0.000 000 014 901 161 193 847 656 25 |
| 134 217 728 | 27 | 0.000 000 007 450 580 596 923 828 125 |
| 268 435 456 | 28 | 0.000 000 003 725 290 298 461 914 062 5 |
| 536 870 912 | 29 | 0.000 000 001 862 645 149 230 957 031 25 |
| 1 073 741 824 | 30 | 0.000 000 000 931 322 574 615 478 515 625 |
| 2 147 483 648 | 31 | 0.000 000 000 465 661 287 307 739 257 812 5 |
| 4 294 967 296 | 32 | 0.000 000 000 232 830 643 653 869 628 906 25 |
| 8 589 934 592 | 33 | 0.000 000 000 116 415 321 826 934 814 453 125 |
| 17 179 869 184 | 34 | 0.000 000 000 058 207 661 913 467 407 226 562 5 |
| 34 359 738 368 | 35 | 0.000 000 000 029 103 831 456 733 703 613 281 25 |
| 68 719 476 736 | 36 | 0.000 000 000 014 551 915 228 366 851 806 640 625 |
| 137 438 953 472 | 37 | 0.000 000 000 007 275 957 614 183 425 903 320 312 5 |
| 274 877 906 944 | 38 | 0.000 000 000 003 637 978 807 091 712 951 660 156 25 |
| 549 755 813 888 | 39 | 0.000 000 000 001 818 989 403 545 856 475 830 078 125 |
| 1 099 511 627 776 | 40 | 0.000 000 000 000 909 494 701 772 928 237 915 039 062 5 |
| 2 199 023 255 552 | 41 | 0.000 000 000 000 454 747 350 886 464 118 957 519 531 25 |
| 4 398 046 511 104 | 42 | 0.000 000 000 000 227 373 675 443 232 059 478 759 765 625 |
| 8 796 093 022 206 | 43 | 0.000 000 000 000 113 686 837 721 616 029 739 379 882 812 5 |
| 17 592 188 044 416 | 44 | 0.000 000 000 000 056 843 418 860 808 014 869 689 941 406 25 |
| 35 184 372 088 832 | 45 | 0.000 000 000 000 028 421 709 430 404 007 434 844 970 703 125 |
| 70 368 744 177 664 | 46 | 0.000 000 000 000 014 210 854 715 202 003 717 422 485 351 562 5 |
| 140 737 488 355 328 | 47 | 0.000 000 000 000 007 105 427 357 601 001 858 711 242 675 781 25 |
| 281 474 976 710 656 | 48 | 0.000 000 000 000 003 552 713 678 800 500 929 355 621 337 890 625 |
| 562 949 953 421 312 | 49 | 0.000 000 000 000 001 776 356 839 400 250 464 677 810 668 945 312 5 |
| 1 125 899 906 842 624 | 50 | 0.000 000 000 000 000 888 178 419 700 125 232 338 905 334 472 656 25 |
| 2 251 799 813 685 248 | 51 | 0.000 000 000 000 000 444 089 209 850 062 616 169 452 667 236 328 125 |
| 4 503 599 627 370 496 | 52 | 0.000 000 000 000 000 222 044 604 925 031 308 084 726 333 618 164 062 5 |
| 9 007 199 254 740 992 | 53 | 0.000 000 000 000 000 111 022 302 462 515 654 042 363 166 809 082 031 25 |
| 18 014 398 509 481 984 | 54 | 0.000 000 000 000 000 055 511 151 231 257 827 021 181 583 404 541 015 625 |
| 36 028 797 018 963 968 | 55 | 0.000 000 000 000 000 027 755 575 615 628 913 510 590 791 702 270 507 812 5 |
| 72 057 594 037 927 936 | 56 | 0.000 000 000 000 000 013 877 787 807 814 456 755 295 395 851 135 253 906 25 |
| 144 115 188 075 855 872 | 57 | 0.000 000 000 000 000 006 938 893 903 907 228 377 647 697 925 567 626 953 125 |
| 288 230 376 151 711 744 | 58 | 0.000 000 000 000 000 003 469 446 951 953 614 188 823 848 962 783 813 476 562 5 |
| 576 460 752 303 423 488 | 59 | 0.000 000 000 000 000 001 734 723 475 976 807 094 411 924 481 391 906 738 281 25 |
| 1 152 921 504 604 846 976 | 60 | 0.000 000 000 000 000 000 867 361 737 988 403 547 205 962 240 695 953 369 140 625 |
| 2 305 843 009 213 693 952 | 61 | 0.000 000 000 000 000 000 433 680 868 994 201 773 602 981 120 347 976 684 570 312 5 |
| 4 611 686 018 427 387 904 | 62 | 0.000 000 000 000 000 000 218 840 434 497 100 886 801 490 560 173 988 342 283 156 25 |
| 9 223 372 036 854 773 808 | 63 | 0.000 000 000 000 000 000 106 420 217 248 550 443 400 743 280 066 994 171 142 578 125 |

TABLE OF POWERS OF SIXTEEN₁₀

| | 16^n | n | 16^{-n} | | | | | |
|--|---------------------|----|-----------|-------|-------|-------|---|------------|
| | 1 | 0 | 0.10000 | 00000 | 00000 | 00000 | x | 10^{-10} |
| | 16 | 1 | 0.62500 | 00000 | 00000 | 00000 | x | 10^{-1} |
| | 256 | 2 | 0.39062 | 50000 | 00000 | 00000 | x | 10^{-2} |
| | 4096 | 3 | 0.24414 | 06250 | 00000 | 00000 | x | 10^{-3} |
| | 65536 | 4 | 0.15258 | 78906 | 25000 | 00000 | x | 10^{-4} |
| | 1048576 | 5 | 0.95367 | 43164 | 06250 | 00000 | x | 10^{-5} |
| | 16777216 | 6 | 0.59604 | 64477 | 53906 | 25000 | x | 10^{-6} |
| | 268435456 | 7 | 0.37252 | 90298 | 46191 | 40625 | x | 10^{-7} |
| | 4294967296 | 8 | 0.23283 | 06436 | 53869 | 62891 | x | 10^{-8} |
| | 68719476736 | 9 | 0.14551 | 91522 | 83668 | 51807 | x | 10^{-9} |
| | 1099511627776 | 10 | 0.90949 | 47017 | 72928 | 23792 | x | 10^{-10} |
| | 17592186044416 | 11 | 0.56843 | 41886 | 08080 | 14870 | x | 10^{-11} |
| | 281474976710656 | 12 | 0.35527 | 13678 | 80050 | 09294 | x | 10^{-12} |
| | 4503599627370496 | 13 | 0.22204 | 46049 | 25031 | 30808 | x | 10^{-13} |
| | 72057594037927936 | 14 | 0.13877 | 78780 | 78144 | 56755 | x | 10^{-14} |
| | 1152921504606846976 | 15 | 0.86736 | 17379 | 88403 | 54721 | x | 10^{-15} |

TAB'II OF POWERS OF 10₁₆

| | 10^n | n | 10^{-n} | | | | | |
|--|------------------|----|-----------|------|------|------|---|------------|
| | 1 | 0 | 1.0000 | 0000 | 0000 | 0000 | | |
| | A | 1 | 0.1999 | 9999 | 9999 | 999A | | |
| | 64 | 2 | 0.28F5 | C28F | 5C28 | F5C3 | x | 16^{-1} |
| | 3E8 | 3 | 0.4189 | 3748 | C6A7 | EF9E | x | 16^{-2} |
| | 2710 | 4 | 0.68DB | 8BAC | 710C | B296 | x | 16^{-3} |
| | 186A0 | 5 | 0.A7C5 | AC47 | 1B47 | 8423 | x | 16^{-4} |
| | F4240 | 6 | 0.10C6 | F7A0 | B5ED | 8D37 | x | 16^{-5} |
| | 989680 | 7 | 0.1AD7 | F29A | BCAF | 4858 | x | 16^{-6} |
| | 5F5E100 | 8 | 0.2AF3 | 1DC4 | 6118 | 73BF | x | 16^{-7} |
| | 3B9AC00 | 9 | 0.44B8 | 2FA0 | 9B5A | 52CC | x | 16^{-8} |
| | 2540BE400 | 10 | 0.6DF3 | 7F67 | 5EF6 | EADF | x | 16^{-9} |
| | 174876E800 | 11 | 0.AFEB | FF0B | CB24 | AAFF | x | 16^{-10} |
| | E8D4A51000 | 12 | 0.1197 | 9981 | 2DEA | 1119 | x | 16^{-11} |
| | 9164E72A000 | 13 | 0.1C25 | C268 | 4976 | 81C2 | x | 16^{-12} |
| | 5AF3107A4000 | 14 | 0.2D09 | 370D | 4257 | 3604 | x | 16^{-13} |
| | 38D7EA4C68000 | 15 | 0.480E | BE7B | 9D58 | 566D | x | 16^{-14} |
| | 2386526FC10000 | 16 | 0.734A | CA5F | 6226 | F0AE | x | 16^{-15} |
| | 16345785D8A0000 | 17 | 0.B877 | AA32 | 36A4 | B449 | x | 16^{-16} |
| | DE0B6B3A7640000 | 18 | 0.1272 | 5DD1 | D243 | ABA1 | x | 16^{-17} |
| | BAC7230489E80000 | 19 | 0.1D83 | C94F | B6D2 | AC35 | x | 16^{-18} |

HEXADECIMAL-DECIMAL INTEGER CONVERSION

The table below provides for direct conversions between hexadecimal integers in the range 0-FFF and decimal integers in the range 0-4095. For conversion of larger integers, the table values may be added to the following figures:

| <u>Hexadecimal</u> | <u>Decimal</u> | <u>Hexadecimal</u> | <u>Decimal</u> |
|--------------------|----------------|--------------------|----------------|
| 01 000 | 4 096 | 20 000 | 131 072 |
| 02 000 | 8 192 | 30 000 | 196 608 |
| 03 000 | 12 288 | 40 000 | 262 144 |
| 04 000 | 16 384 | 50 000 | 327 680 |
| 05 000 | 20 480 | 60 000 | 393 216 |
| 06 000 | 24 576 | 70 000 | 458 752 |
| 07 000 | 28 672 | 80 000 | 524 288 |
| 08 000 | 32 768 | 90 000 | 589 824 |
| 09 000 | 36 864 | A0 000 | 655 360 |
| 0A 000 | 40 960 | B0 000 | 720 896 |
| 0B 000 | 45 056 | C0 000 | 786 432 |
| 0C 000 | 49 152 | D0 000 | 851 968 |
| 0D 000 | 53 248 | E0 000 | 917 504 |
| 0E 000 | 57 344 | F0 000 | 983 040 |
| 0F 000 | 61 440 | 100 000 | 1 048 576 |
| 10 000 | 65 536 | 200 000 | 2 097 152 |
| 11 000 | 69 632 | 300 000 | 3 145 728 |
| 12 000 | 73 728 | 400 000 | 4 194 304 |
| 13 000 | 77 824 | 500 000 | 5 242 880 |
| 14 000 | 81 920 | 600 000 | 6 291 456 |
| 15 000 | 86 016 | 700 000 | 7 340 032 |
| 16 000 | 90 112 | 800 000 | 8 388 608 |
| 17 000 | 94 208 | 900 000 | 9 437 184 |
| 18 000 | 98 304 | A00 000 | 10 485 760 |
| 19 000 | 102 400 | B00 000 | 11 534 336 |
| 1A 000 | 106 496 | C00 000 | 12 582 912 |
| 1B 000 | 110 592 | D00 000 | 13 631 488 |
| 1C 000 | 114 688 | E00 000 | 14 680 064 |
| 1D 000 | 118 784 | F00 000 | 15 728 640 |
| 1E 000 | 122 880 | 1 000 000 | 16 777 216 |
| 1F 000 | 126 976 | 2 000 000 | 33 554 432 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|-----|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 000 | 0000 | 0001 | 0002 | 0003 | 0004 | 0005 | 0006 | 0007 | 0008 | 0009 | 0010 | 0011 | 0012 | 0013 | 0014 | 0015 |
| 010 | 0016 | 0017 | 0018 | 0019 | 0020 | 0021 | 0022 | 0023 | 0024 | 0025 | 0026 | 0027 | 0028 | 0029 | 0030 | 0031 |
| 020 | 0032 | 0033 | 0034 | 0035 | 0036 | 0037 | 0038 | 0039 | 0040 | 0041 | 0042 | 0043 | 0044 | 0045 | 0046 | 0047 |
| 030 | 0048 | 0049 | 0050 | 0051 | 0052 | 0053 | 0054 | 0055 | 0056 | 0057 | 0058 | 0059 | 0060 | 0061 | 0062 | 0063 |
| 040 | 0064 | 0065 | 0066 | 0067 | 0068 | 0069 | 0070 | 0071 | 0072 | 0073 | 0074 | 0075 | 0076 | 0077 | 0078 | 0079 |
| 050 | 0080 | 0081 | 0082 | 0083 | 0084 | 0085 | 0086 | 0087 | 0088 | 0089 | 0090 | 0091 | 0092 | 0093 | 0094 | 0095 |
| 060 | 0096 | 0097 | 0098 | 0099 | 0100 | 0101 | 0102 | 0103 | 0104 | 0105 | 0106 | 0107 | 0108 | 0109 | 0110 | 0111 |
| 070 | 0112 | 0113 | 0114 | 0115 | 0116 | 0117 | 0118 | 0119 | 0120 | 0121 | 0122 | 0123 | 0124 | 0125 | 0126 | 0127 |
| 080 | 0128 | 0129 | 0130 | 0131 | 0132 | 0133 | 0134 | 0135 | 0136 | 0137 | 0138 | 0139 | 0140 | 0141 | 0142 | 0143 |
| 090 | 0144 | 0145 | 0146 | 0147 | 0148 | 0149 | 0150 | 0151 | 0152 | 0153 | 0154 | 0155 | 0156 | 0157 | 0158 | 0159 |
| 0A0 | 0160 | 0161 | 0162 | 0163 | 0164 | 0165 | 0166 | 0167 | 0168 | 0169 | 0170 | 0171 | 0172 | 0173 | 0174 | 0175 |
| 0B0 | 0176 | 0177 | 0178 | 0179 | 0180 | 0181 | 0182 | 0183 | 0184 | 0185 | 0186 | 0187 | 0188 | 0189 | 0190 | 0191 |
| 0C0 | 0192 | 0193 | 0194 | 0195 | 0196 | 0197 | 0198 | 0199 | 0200 | 0201 | 0202 | 0203 | 0204 | 0205 | 0206 | 0207 |
| 0D0 | 0208 | 0209 | 0210 | 0211 | 0212 | 0213 | 0214 | 0215 | 0216 | 0217 | 0218 | 0219 | 0220 | 0221 | 0222 | 0223 |
| 0E0 | 0224 | 0225 | 0226 | 0227 | 0228 | 0229 | 0230 | 0231 | 0232 | 0233 | 0234 | 0235 | 0236 | 0237 | 0238 | 0239 |
| 0F0 | 0240 | 0241 | 0242 | 0243 | 0244 | 0245 | 0246 | 0247 | 0248 | 0249 | 0250 | 0251 | 0252 | 0253 | 0254 | 0255 |

HEXADECIMAL-DECIMAL INTEGER CONVERSION (Cont.)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|-----|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 100 | 0256 | 0257 | 0258 | 0259 | 0260 | 0261 | 0262 | 0263 | 0264 | 0265 | 0266 | 0267 | 0268 | 0269 | 0270 | 0271 |
| 110 | 0272 | 0273 | 0274 | 0275 | 0276 | 0277 | 0278 | 0279 | 0280 | 0281 | 0282 | 0283 | 0284 | 0285 | 0286 | 0287 |
| 120 | 0288 | 0289 | 0290 | 0291 | 0292 | 0293 | 0294 | 0295 | 0296 | 0297 | 0298 | 0299 | 0300 | 0301 | 0302 | 0303 |
| 130 | 0304 | 0305 | 0306 | 0307 | 0308 | 0309 | 0310 | 0311 | 0312 | 0313 | 0314 | 0315 | 0316 | 0317 | 0318 | 0319 |
| 140 | 0320 | 0321 | 0322 | 0323 | 0324 | 0325 | 0326 | 0327 | 0328 | 0329 | 0330 | 0331 | 0332 | 0333 | 0334 | 0335 |
| 150 | 0336 | 0337 | 0338 | 0339 | 0340 | 0341 | 0342 | 0343 | 0344 | 0345 | 0346 | 0347 | 0348 | 0349 | 0350 | 0351 |
| 160 | 0352 | 0353 | 0354 | 0355 | 0356 | 0357 | 0358 | 0359 | 0360 | 0361 | 0362 | 0363 | 0364 | 0365 | 0366 | 0367 |
| 170 | 0368 | 0369 | 0370 | 0371 | 0372 | 0373 | 0374 | 0375 | 0376 | 0377 | 0378 | 0379 | 0380 | 0381 | 0382 | 0383 |
| 180 | 0384 | 0385 | 0386 | 0387 | 0388 | 0389 | 0390 | 0391 | 0392 | 0393 | 0394 | 0395 | 0396 | 0397 | 0398 | 0399 |
| 190 | 0400 | 0401 | 0402 | 0403 | 0404 | 0405 | 0406 | 0407 | 0408 | 0409 | 0410 | 0411 | 0412 | 0413 | 0414 | 0415 |
| 1A0 | 0416 | 0417 | 0418 | 0419 | 0420 | 0421 | 0422 | 0423 | 0424 | 0425 | 0426 | 0427 | 0428 | 0429 | 0430 | 0431 |
| 1B0 | 0432 | 0433 | 0434 | 0435 | 0436 | 0437 | 0438 | 0439 | 0440 | 0441 | 0442 | 0443 | 0444 | 0445 | 0446 | 0447 |
| 1C0 | 0448 | 0449 | 0450 | 0451 | 0452 | 0453 | 0454 | 0455 | 0456 | 0457 | 0458 | 0459 | 0460 | 0461 | 0462 | 0463 |
| 1D0 | 0464 | 0465 | 0466 | 0467 | 0468 | 0469 | 0470 | 0471 | 0472 | 0473 | 0474 | 0475 | 0476 | 0477 | 0478 | 0479 |
| 1E0 | 0480 | 0481 | 0482 | 0483 | 0484 | 0485 | 0486 | 0487 | 0488 | 0489 | 0490 | 0491 | 0492 | 0493 | 0494 | 0495 |
| 1F0 | 0496 | 0497 | 0498 | 0499 | 0500 | 0501 | 0502 | 0503 | 0504 | 0505 | 0506 | 0507 | 0508 | 0509 | 0510 | 0511 |
| 200 | 0512 | 0513 | 0514 | 0515 | 0516 | 0517 | 0518 | 0519 | 0520 | 0521 | 0522 | 0523 | 0524 | 0525 | 0526 | 0527 |
| 210 | 0528 | 0529 | 0530 | 0531 | 0532 | 0533 | 0534 | 0535 | 0536 | 0537 | 0538 | 0539 | 0540 | 0541 | 0542 | 0543 |
| 220 | 0544 | 0545 | 0546 | 0547 | 0548 | 0549 | 0550 | 0551 | 0552 | 0553 | 0554 | 0555 | 0556 | 0557 | 0558 | 0559 |
| 230 | 0560 | 0561 | 0562 | 0563 | 0564 | 0565 | 0566 | 0567 | 0568 | 0569 | 0570 | 0571 | 0572 | 0573 | 0574 | 0575 |
| 240 | 0576 | 0577 | 0578 | 0579 | 0580 | 0581 | 0582 | 0583 | 0584 | 0585 | 0586 | 0587 | 0588 | 0589 | 0590 | 0591 |
| 250 | 0592 | 0593 | 0594 | 0595 | 0596 | 0597 | 0598 | 0599 | 0600 | 0601 | 0602 | 0603 | 0604 | 0605 | 0606 | 0607 |
| 260 | 0608 | 0609 | 0610 | 0611 | 0612 | 0613 | 0614 | 0615 | 0616 | 0617 | 0618 | 0619 | 0620 | 0621 | 0622 | 0623 |
| 270 | 0624 | 0625 | 0626 | 0627 | 0628 | 0629 | 0630 | 0631 | 0632 | 0633 | 0634 | 0635 | 0636 | 0637 | 0638 | 0639 |
| 280 | 0640 | 0641 | 0642 | 0643 | 0644 | 0645 | 0646 | 0647 | 0648 | 0649 | 0650 | 0651 | 0652 | 0653 | 0654 | 0655 |
| 290 | 0656 | 0657 | 0658 | 0659 | 0660 | 0661 | 0662 | 0663 | 0664 | 0665 | 0666 | 0667 | 0668 | 0669 | 0670 | 0671 |
| 2A0 | 0672 | 0673 | 0674 | 0675 | 0676 | 0677 | 0678 | 0679 | 0680 | 0681 | 0682 | 0683 | 0684 | 0685 | 0686 | 0687 |
| 2B0 | 0688 | 0689 | 0690 | 0691 | 0692 | 0693 | 0694 | 0695 | 0696 | 0697 | 0698 | 0699 | 0700 | 0701 | 0702 | 0703 |
| 2C0 | 0704 | 0705 | 0706 | 0707 | 0708 | 0709 | 0710 | 0711 | 0712 | 0713 | 0714 | 0715 | 0716 | 0717 | 0718 | 0719 |
| 2D0 | 0720 | 0721 | 0722 | 0723 | 0724 | 0725 | 0726 | 0727 | 0728 | 0729 | 0730 | 0731 | 0732 | 0733 | 0734 | 0735 |
| 2E0 | 0736 | 0737 | 0738 | 0739 | 0740 | 0741 | 0742 | 0743 | 0744 | 0745 | 0746 | 0747 | 0748 | 0749 | 0750 | 0751 |
| 2F0 | 0752 | 0753 | 0754 | 0755 | 0756 | 0757 | 0758 | 0759 | 0760 | 0761 | 0762 | 0763 | 0764 | 0765 | 0766 | 0767 |
| 300 | 0768 | 0769 | 0770 | 0771 | 0772 | 0773 | 0774 | 0775 | 0776 | 0777 | 0778 | 0779 | 0780 | 0781 | 0782 | 0783 |
| 310 | 0784 | 0785 | 0786 | 0787 | 0788 | 0789 | 0790 | 0791 | 0792 | 0793 | 0794 | 0795 | 0796 | 0797 | 0798 | 0799 |
| 320 | 0800 | 0801 | 0802 | 0803 | 0804 | 0805 | 0806 | 0807 | 0808 | 0809 | 0810 | 0811 | 0812 | 0813 | 0814 | 0815 |
| 330 | 0816 | 0817 | 0818 | 0819 | 0820 | 0821 | 0822 | 0823 | 0824 | 0825 | 0826 | 0827 | 0828 | 0829 | 0830 | 0831 |
| 340 | 0832 | 0833 | 0834 | 0835 | 0836 | 0837 | 0838 | 0839 | 0840 | 0841 | 0842 | 0843 | 0844 | 0845 | 0846 | 0847 |
| 350 | 0848 | 0849 | 0850 | 0851 | 0852 | 0853 | 0854 | 0855 | 0856 | 0857 | 0858 | 0859 | 0860 | 0861 | 0862 | 0863 |
| 360 | 0864 | 0865 | 0866 | 0867 | 0868 | 0869 | 0870 | 0871 | 0872 | 0873 | 0874 | 0875 | 0876 | 0877 | 0878 | 0879 |
| 370 | 0880 | 0881 | 0882 | 0883 | 0884 | 0885 | 0886 | 0887 | 0888 | 0889 | 0890 | 0891 | 0892 | 0893 | 0894 | 0895 |
| 380 | 0896 | 0897 | 0898 | 0899 | 0900 | 0901 | 0902 | 0903 | 0904 | 0905 | 0906 | 0907 | 0908 | 0909 | 0910 | 0911 |
| 390 | 0912 | 0913 | 0914 | 0915 | 0916 | 0917 | 0918 | 0919 | 0920 | 0921 | 0922 | 0923 | 0924 | 0925 | 0926 | 0927 |
| 3A0 | 0928 | 0929 | 0930 | 0931 | 0932 | 0933 | 0934 | 0935 | 0936 | 0937 | 0938 | 0939 | 0940 | 0941 | 0942 | 0943 |
| 3B0 | 0944 | 0945 | 0946 | 0947 | 0948 | 0949 | 0950 | 0951 | 0952 | 0953 | 0954 | 0955 | 0956 | 0957 | 0958 | 0959 |
| 3C0 | 0960 | 0961 | 0962 | 0963 | 0964 | 0965 | 0966 | 0967 | 0968 | 0969 | 0970 | 0971 | 0972 | 0973 | 0974 | 0975 |
| 3D0 | 0976 | 0977 | 0978 | 0979 | 0980 | 0981 | 0982 | 0983 | 0984 | 0985 | 0986 | 0987 | 0988 | 0989 | 0990 | 0991 |
| 3E0 | 0992 | 0993 | 0994 | 0995 | 0996 | 0997 | 0998 | 0999 | 1000 | 1001 | 1002 | 1003 | 1004 | 1005 | 1006 | 1007 |
| 3F0 | 1008 | 1009 | 1010 | 1011 | 1012 | 1013 | 1014 | 1015 | 1016 | 1017 | 1018 | 1019 | 1020 | 1021 | 1022 | 1023 |

HEXADECIMAL-DECIMAL INTEGER CONVERSION (Cont.)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|-----|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 400 | 1024 | 1025 | 1026 | 1027 | 1028 | 1029 | 1030 | 1031 | 1032 | 1033 | 1034 | 1035 | 1036 | 1037 | 1038 | 1039 |
| 410 | 1040 | 1041 | 1042 | 1043 | 1044 | 1045 | 1046 | 1047 | 1048 | 1049 | 1050 | 1051 | 1052 | 1053 | 1054 | 1055 |
| 420 | 1056 | 1057 | 1058 | 1059 | 1060 | 1061 | 1062 | 1063 | 1064 | 1065 | 1066 | 1067 | 1068 | 1069 | 1070 | 1071 |
| 430 | 1072 | 1073 | 1074 | 1075 | 1076 | 1077 | 1078 | 1079 | 1080 | 1081 | 1082 | 1083 | 1084 | 1085 | 1086 | 1087 |
| 440 | 1088 | 1089 | 1090 | 1091 | 1092 | 1093 | 1094 | 1095 | 1096 | 1097 | 1098 | 1099 | 1100 | 1101 | 1102 | 1103 |
| 450 | 1104 | 1105 | 1106 | 1107 | 1108 | 1109 | 1110 | 1111 | 1112 | 1113 | 1114 | 1115 | 1116 | 1117 | 1118 | 1119 |
| 460 | 1120 | 1121 | 1122 | 1123 | 1124 | 1125 | 1126 | 1127 | 1128 | 1129 | 1130 | 1131 | 1132 | 1133 | 1134 | 1135 |
| 470 | 1136 | 1137 | 1138 | 1139 | 1140 | 1141 | 1142 | 1143 | 1144 | 1145 | 1146 | 1147 | 1148 | 1149 | 1150 | 1151 |
| 480 | 1152 | 1153 | 1154 | 1155 | 1156 | 1157 | 1158 | 1159 | 1160 | 1161 | 1162 | 1163 | 1164 | 1165 | 1166 | 1167 |
| 490 | 1168 | 1169 | 1170 | 1171 | 1172 | 1173 | 1174 | 1175 | 1176 | 1177 | 1178 | 1179 | 1180 | 1181 | 1182 | 1183 |
| 4A0 | 1184 | 1185 | 1186 | 1187 | 1188 | 1189 | 1190 | 1191 | 1192 | 1193 | 1194 | 1195 | 1196 | 1197 | 1198 | 1199 |
| 4B0 | 1200 | 1201 | 1202 | 1203 | 1204 | 1205 | 1206 | 1207 | 1208 | 1209 | 1210 | 1211 | 1212 | 1213 | 1214 | 1215 |
| 4C0 | 1216 | 1217 | 1218 | 1219 | 1220 | 1221 | 1222 | 1223 | 1224 | 1225 | 1226 | 1227 | 1228 | 1229 | 1230 | 1231 |
| 4D0 | 1232 | 1233 | 1234 | 1235 | 1236 | 1237 | 1238 | 1239 | 1240 | 1241 | 1242 | 1243 | 1244 | 1245 | 1246 | 1247 |
| 4E0 | 1248 | 1249 | 1250 | 1251 | 1252 | 1253 | 1254 | 1255 | 1256 | 1257 | 1258 | 1259 | 1260 | 1261 | 1262 | 1263 |
| 4F0 | 1264 | 1265 | 1266 | 1267 | 1268 | 1269 | 1270 | 1271 | 1272 | 1273 | 1274 | 1275 | 1276 | 1277 | 1278 | 1279 |
| 500 | 1280 | 1281 | 1282 | 1283 | 1284 | 1285 | 1286 | 1287 | 1288 | 1289 | 1290 | 1291 | 1292 | 1293 | 1294 | 1295 |
| 510 | 1296 | 1297 | 1298 | 1299 | 1300 | 1301 | 1302 | 1303 | 1304 | 1305 | 1306 | 1307 | 1308 | 1309 | 1310 | 1311 |
| 520 | 1312 | 1313 | 1314 | 1315 | 1316 | 1317 | 1318 | 1319 | 1320 | 1321 | 1322 | 1323 | 1324 | 1325 | 1326 | 1327 |
| 530 | 1328 | 1329 | 1330 | 1331 | 1332 | 1333 | 1334 | 1335 | 1336 | 1337 | 1338 | 1339 | 1340 | 1341 | 1342 | 1343 |
| 540 | 1344 | 1345 | 1346 | 1347 | 1348 | 1349 | 1350 | 1351 | 1352 | 1353 | 1354 | 1355 | 1356 | 1357 | 1358 | 1359 |
| 550 | 1360 | 1361 | 1362 | 1363 | 1364 | 1365 | 1366 | 1367 | 1368 | 1369 | 1370 | 1371 | 1372 | 1373 | 1374 | 1375 |
| 560 | 1376 | 1377 | 1378 | 1379 | 1380 | 1381 | 1382 | 1383 | 1384 | 1385 | 1386 | 1387 | 1388 | 1389 | 1390 | 1391 |
| 570 | 1392 | 1393 | 1394 | 1395 | 1396 | 1397 | 1398 | 1399 | 1400 | 1401 | 1402 | 1403 | 1404 | 1405 | 1406 | 1407 |
| 580 | 1408 | 1409 | 1410 | 1411 | 1412 | 1413 | 1414 | 1415 | 1416 | 1417 | 1418 | 1419 | 1420 | 1421 | 1422 | 1423 |
| 590 | 1424 | 1425 | 1426 | 1427 | 1428 | 1429 | 1430 | 1431 | 1432 | 1433 | 1434 | 1435 | 1436 | 1437 | 1438 | 1439 |
| 5A0 | 1440 | 1441 | 1442 | 1443 | 1444 | 1445 | 1446 | 1447 | 1448 | 1449 | 1450 | 1451 | 1452 | 1453 | 1454 | 1455 |
| 5B0 | 1456 | 1457 | 1458 | 1459 | 1460 | 1461 | 1462 | 1463 | 1464 | 1465 | 1466 | 1467 | 1468 | 1469 | 1470 | 1471 |
| 5C0 | 1472 | 1473 | 1474 | 1475 | 1476 | 1477 | 1478 | 1479 | 1480 | 1481 | 1482 | 1483 | 1484 | 1485 | 1486 | 1487 |
| 5D0 | 1488 | 1489 | 1490 | 1491 | 1492 | 1493 | 1494 | 1495 | 1496 | 1497 | 1498 | 1499 | 1500 | 1501 | 1502 | 1503 |
| 5E0 | 1504 | 1505 | 1506 | 1507 | 1508 | 1509 | 1510 | 1511 | 1512 | 1513 | 1514 | 1515 | 1516 | 1517 | 1518 | 1519 |
| 5F0 | 1520 | 1521 | 1522 | 1523 | 1524 | 1525 | 1526 | 1527 | 1528 | 1529 | 1530 | 1531 | 1532 | 1533 | 1534 | 1535 |
| 600 | 1536 | 1537 | 1538 | 1539 | 1540 | 1541 | 1542 | 1543 | 1544 | 1545 | 1546 | 1547 | 1548 | 1549 | 1550 | 1551 |
| 610 | 1552 | 1553 | 1554 | 1555 | 1556 | 1557 | 1558 | 1559 | 1560 | 1561 | 1562 | 1563 | 1564 | 1565 | 1566 | 1567 |
| 620 | 1568 | 1569 | 1570 | 1571 | 1572 | 1573 | 1574 | 1575 | 1576 | 1577 | 1578 | 1579 | 1580 | 1581 | 1582 | 1583 |
| 630 | 1584 | 1585 | 1586 | 1587 | 1588 | 1589 | 1590 | 1591 | 1592 | 1593 | 1594 | 1595 | 1596 | 1597 | 1598 | 1599 |
| 640 | 1600 | 1601 | 1602 | 1603 | 1604 | 1605 | 1606 | 1607 | 1608 | 1609 | 1610 | 1611 | 1612 | 1613 | 1614 | 1615 |
| 650 | 1616 | 1617 | 1618 | 1619 | 1620 | 1621 | 1622 | 1623 | 1624 | 1625 | 1626 | 1627 | 1628 | 1629 | 1630 | 1631 |
| 660 | 1632 | 1633 | 1634 | 1635 | 1636 | 1637 | 1638 | 1639 | 1640 | 1641 | 1642 | 1643 | 1644 | 1645 | 1646 | 1647 |
| 670 | 1648 | 1649 | 1650 | 1651 | 1652 | 1653 | 1654 | 1655 | 1656 | 1657 | 1658 | 1659 | 1660 | 1661 | 1662 | 1663 |
| 680 | 1664 | 1665 | 1666 | 1667 | 1668 | 1669 | 1670 | 1671 | 1672 | 1673 | 1674 | 1675 | 1676 | 1677 | 1678 | 1679 |
| 690 | 1680 | 1681 | 1682 | 1683 | 1684 | 1685 | 1686 | 1687 | 1688 | 1689 | 1690 | 1691 | 1692 | 1693 | 1694 | 1695 |
| 6A0 | 1696 | 1697 | 1698 | 1699 | 1700 | 1701 | 1702 | 1703 | 1704 | 1705 | 1706 | 1707 | 1708 | 1709 | 1710 | 1711 |
| 6B0 | 1712 | 1713 | 1714 | 1715 | 1716 | 1717 | 1718 | 1719 | 1720 | 1721 | 1722 | 1723 | 1724 | 1725 | 1726 | 1727 |
| 6C0 | 1728 | 1729 | 1730 | 1731 | 1732 | 1733 | 1734 | 1735 | 1736 | 1737 | 1738 | 1739 | 1740 | 1741 | 1742 | 1743 |
| 6D0 | 1744 | 1745 | 1746 | 1747 | 1748 | 1749 | 1750 | 1751 | 1752 | 1753 | 1754 | 1755 | 1756 | 1757 | 1758 | 1759 |
| 6E0 | 1760 | 1761 | 1762 | 1763 | 1764 | 1765 | 1766 | 1767 | 1768 | 1769 | 1770 | 1771 | 1772 | 1773 | 1774 | 1775 |
| 6F0 | 1776 | 1777 | 1778 | 1779 | 1780 | 1781 | 1782 | 1783 | 1784 | 1785 | 1786 | 1787 | 1788 | 1789 | 1790 | 1791 |

HEXADECIMAL-DECIMAL INTEGER CONVERSION (Cont.)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|-----|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 700 | 1792 | 1793 | 1794 | 1795 | 1796 | 1797 | 1798 | 1799 | 1800 | 1801 | 1802 | 1803 | 1804 | 1805 | 1806 | 1807 |
| 710 | 1808 | 1809 | 1810 | 1811 | 1812 | 1813 | 1814 | 1815 | 1816 | 1817 | 1818 | 1819 | 1820 | 1821 | 1822 | 1823 |
| 720 | 1824 | 1825 | 1826 | 1827 | 1828 | 1829 | 1830 | 1831 | 1832 | 1833 | 1834 | 1835 | 1836 | 1837 | 1838 | 1839 |
| 730 | 1840 | 1841 | 1842 | 1843 | 1844 | 1845 | 1846 | 1847 | 1848 | 1849 | 1850 | 1851 | 1852 | 1853 | 1854 | 1855 |
| 740 | 1856 | 1857 | 1858 | 1859 | 1860 | 1861 | 1862 | 1863 | 1864 | 1865 | 1866 | 1867 | 1868 | 1869 | 1870 | 1871 |
| 750 | 1872 | 1873 | 1874 | 1875 | 1876 | 1877 | 1878 | 1879 | 1880 | 1881 | 1882 | 1883 | 1884 | 1885 | 1886 | 1887 |
| 760 | 1888 | 1889 | 1890 | 1891 | 1892 | 1893 | 1894 | 1895 | 1896 | 1897 | 1898 | 1899 | 1900 | 1901 | 1902 | 1903 |
| 770 | 1904 | 1905 | 1906 | 1907 | 1908 | 1909 | 1910 | 1911 | 1912 | 1913 | 1914 | 1915 | 1916 | 1917 | 1918 | 1919 |
| 780 | 1920 | 1921 | 1922 | 1923 | 1924 | 1925 | 1926 | 1927 | 1928 | 1929 | 1930 | 1931 | 1932 | 1933 | 1934 | 1935 |
| 790 | 1936 | 1937 | 1938 | 1939 | 1940 | 1941 | 1942 | 1943 | 1944 | 1945 | 1946 | 1947 | 1948 | 1949 | 1950 | 1951 |
| 7A0 | 1952 | 1953 | 1954 | 1955 | 1956 | 1957 | 1958 | 1959 | 1960 | 1961 | 1962 | 1963 | 1964 | 1965 | 1966 | 1967 |
| 7B0 | 1968 | 1969 | 1970 | 1971 | 1972 | 1973 | 1974 | 1975 | 1976 | 1977 | 1978 | 1979 | 1980 | 1981 | 1982 | 1983 |
| 7C0 | 1984 | 1985 | 1986 | 1987 | 1988 | 1989 | 1990 | 1991 | 1992 | 1993 | 1994 | 1995 | 1996 | 1997 | 1998 | 1999 |
| 7D0 | 2000 | 2001 | 2002 | 2003 | 2004 | 2005 | 2006 | 2007 | 2008 | 2009 | 2010 | 2011 | 2012 | 2013 | 2014 | 2015 |
| 7E0 | 2016 | 2017 | 2018 | 2019 | 2020 | 2021 | 2022 | 2023 | 2024 | 2025 | 2026 | 2027 | 2028 | 2029 | 2030 | 2031 |
| 7F0 | 2032 | 2033 | 2034 | 2035 | 2036 | 2037 | 2038 | 2039 | 2040 | 2041 | 2042 | 2043 | 2044 | 2045 | 2046 | 2047 |
| 800 | 2048 | 2049 | 2050 | 2051 | 2052 | 2053 | 2054 | 2055 | 2056 | 2057 | 2058 | 2059 | 2060 | 2061 | 2062 | 2063 |
| 810 | 2064 | 2065 | 2066 | 2067 | 2068 | 2069 | 2070 | 2071 | 2072 | 2073 | 2074 | 2075 | 2076 | 2077 | 2078 | 2079 |
| 820 | 2080 | 2081 | 2082 | 2083 | 2084 | 2085 | 2086 | 2087 | 2088 | 2089 | 2090 | 2091 | 2092 | 2093 | 2094 | 2095 |
| 830 | 2096 | 2097 | 2098 | 2099 | 2100 | 2101 | 2102 | 2103 | 2104 | 2105 | 2106 | 2107 | 2108 | 2109 | 2110 | 2111 |
| 840 | 2112 | 2113 | 2114 | 2115 | 2116 | 2117 | 2118 | 2119 | 2120 | 2121 | 2122 | 2123 | 2124 | 2125 | 2126 | 2127 |
| 850 | 2128 | 2129 | 2130 | 2131 | 2132 | 2133 | 2134 | 2135 | 2136 | 2137 | 2138 | 2139 | 2140 | 2141 | 2142 | 2143 |
| 860 | 2144 | 2145 | 2146 | 2147 | 2148 | 2149 | 2150 | 2151 | 2152 | 2153 | 2154 | 2155 | 2156 | 2157 | 2158 | 2159 |
| 870 | 2160 | 2161 | 2162 | 2163 | 2164 | 2165 | 2166 | 2167 | 2168 | 2169 | 2170 | 2171 | 2172 | 2173 | 2174 | 2175 |
| 880 | 2176 | 2177 | 2178 | 2179 | 2180 | 2181 | 2182 | 2183 | 2184 | 2185 | 2186 | 2187 | 2188 | 2189 | 2190 | 2191 |
| 890 | 2192 | 2193 | 2194 | 2195 | 2196 | 2197 | 2198 | 2199 | 2200 | 2201 | 2202 | 2203 | 2204 | 2205 | 2206 | 2207 |
| 8A0 | 2208 | 2209 | 2210 | 2211 | 2212 | 2213 | 2214 | 2215 | 2216 | 2217 | 2218 | 2219 | 2220 | 2221 | 2222 | 2223 |
| 8B0 | 2224 | 2225 | 2226 | 2227 | 2228 | 2229 | 2230 | 2231 | 2232 | 2233 | 2234 | 2235 | 2236 | 2237 | 2238 | 2239 |
| 8C0 | 2240 | 2241 | 2242 | 2243 | 2244 | 2245 | 2246 | 2247 | 2248 | 2249 | 2250 | 2251 | 2252 | 2253 | 2254 | 2255 |
| 8D0 | 2256 | 2257 | 2258 | 2259 | 2260 | 2261 | 2262 | 2263 | 2264 | 2265 | 2266 | 2267 | 2268 | 2269 | 2270 | 2271 |
| 8E0 | 2272 | 2273 | 2274 | 2275 | 2276 | 2277 | 2278 | 2279 | 2280 | 2281 | 2282 | 2283 | 2284 | 2285 | 2286 | 2287 |
| 8F0 | 2288 | 2289 | 2290 | 2291 | 2292 | 2293 | 2294 | 2295 | 2296 | 2297 | 2298 | 2299 | 2300 | 2301 | 2302 | 2303 |
| 900 | 2304 | 2305 | 2306 | 2307 | 2308 | 2309 | 2310 | 2311 | 2312 | 2313 | 2314 | 2315 | 2316 | 2317 | 2318 | 2319 |
| 910 | 2320 | 2321 | 2322 | 2323 | 2324 | 2325 | 2326 | 2327 | 2328 | 2329 | 2330 | 2331 | 2332 | 2333 | 2334 | 2335 |
| 920 | 2336 | 2337 | 2338 | 2339 | 2340 | 2341 | 2342 | 2343 | 2344 | 2345 | 2346 | 2347 | 2348 | 2349 | 2350 | 2351 |
| 930 | 2352 | 2353 | 2354 | 2355 | 2356 | 2357 | 2358 | 2359 | 2360 | 2361 | 2362 | 2363 | 2364 | 2365 | 2366 | 2367 |
| 940 | 2368 | 2369 | 2370 | 2371 | 2372 | 2373 | 2374 | 2375 | 2376 | 2377 | 2378 | 2379 | 2380 | 2381 | 2382 | 2383 |
| 950 | 2384 | 2385 | 2386 | 2387 | 2388 | 2389 | 2390 | 2391 | 2392 | 2393 | 2394 | 2395 | 2396 | 2397 | 2398 | 2399 |
| 960 | 2400 | 2401 | 2402 | 2403 | 2404 | 2405 | 2406 | 2407 | 2408 | 2409 | 2410 | 2411 | 2412 | 2413 | 2414 | 2415 |
| 970 | 2416 | 2417 | 2418 | 2419 | 2420 | 2421 | 2422 | 2423 | 2424 | 2425 | 2426 | 2427 | 2428 | 2429 | 2430 | 2431 |
| 980 | 2432 | 2433 | 2434 | 2435 | 2436 | 2437 | 2438 | 2439 | 2440 | 2441 | 2442 | 2443 | 2444 | 2445 | 2446 | 2447 |
| 990 | 2448 | 2449 | 2450 | 2451 | 2452 | 2453 | 2454 | 2455 | 2456 | 2457 | 2458 | 2459 | 2460 | 2461 | 2462 | 2463 |
| 9A0 | 2464 | 2465 | 2466 | 2467 | 2468 | 2469 | 2470 | 2471 | 2472 | 2473 | 2474 | 2475 | 2476 | 2477 | 2478 | 2479 |
| 9B0 | 2480 | 2481 | 2482 | 2483 | 2484 | 2485 | 2486 | 2487 | 2488 | 2489 | 2490 | 2491 | 2492 | 2493 | 2494 | 2495 |
| 9C0 | 2496 | 2497 | 2498 | 2499 | 2500 | 2501 | 2502 | 2503 | 2504 | 2505 | 2506 | 2507 | 2508 | 2509 | 2510 | 2511 |
| 9D0 | 2512 | 2513 | 2514 | 2515 | 2516 | 2517 | 2518 | 2519 | 2520 | 2521 | 2522 | 2523 | 2524 | 2525 | 2526 | 2527 |
| 9E0 | 2528 | 2529 | 2530 | 2531 | 2532 | 2533 | 2534 | 2535 | 2536 | 2537 | 2538 | 2539 | 2540 | 2541 | 2542 | 2543 |
| 9F0 | 2544 | 2545 | 2546 | 2547 | 2548 | 2549 | 2550 | 2551 | 2552 | 2553 | 2554 | 2555 | 2556 | 2557 | 2558 | 2559 |

HEXADECIMAL-DECIMAL INTEGER CONVERSION (Cont.)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| A00 | 2560 | 2561 | 2562 | 2563 | 2564 | 2565 | 2566 | 2567 | 2568 | 2569 | 2570 | 2571 | 2572 | 2573 | 2574 | 2575 |
| A10 | 2576 | 2577 | 2578 | 2579 | 2580 | 2581 | 2582 | 2583 | 2584 | 2585 | 2586 | 2587 | 2588 | 2589 | 2590 | 2591 |
| A20 | 2592 | 2593 | 2594 | 2595 | 2596 | 2597 | 2598 | 2599 | 2600 | 2601 | 2602 | 2603 | 2604 | 2605 | 2606 | 2607 |
| A30 | 2608 | 2609 | 2610 | 2611 | 2612 | 2613 | 2614 | 2615 | 2616 | 2617 | 2618 | 2619 | 2620 | 2621 | 2622 | 2623 |
| A40 | 2624 | 2625 | 2626 | 2627 | 2628 | 2629 | 2630 | 2631 | 2632 | 2633 | 2634 | 2635 | 2636 | 2637 | 2638 | 2639 |
| A50 | 2640 | 2641 | 2642 | 2643 | 2644 | 2645 | 2646 | 2647 | 2648 | 2649 | 2650 | 2651 | 2652 | 2653 | 2654 | 2655 |
| A60 | 2656 | 2657 | 2658 | 2659 | 2660 | 2661 | 2662 | 2663 | 2664 | 2665 | 2666 | 2667 | 2668 | 2669 | 2670 | 2671 |
| A70 | 2672 | 2673 | 2674 | 2675 | 2676 | 2677 | 2678 | 2679 | 2680 | 2681 | 2682 | 2683 | 2684 | 2685 | 2686 | 2687 |
| A80 | 2688 | 2689 | 2690 | 2691 | 2692 | 2693 | 2694 | 2695 | 2696 | 2697 | 2698 | 2699 | 2700 | 2701 | 2702 | 2703 |
| A90 | 2704 | 2705 | 2706 | 2707 | 2708 | 2709 | 2710 | 2711 | 2712 | 2713 | 2714 | 2715 | 2716 | 2717 | 2718 | 2719 |
| AA0 | 2720 | 2721 | 2722 | 2723 | 2724 | 2725 | 2726 | 2727 | 2728 | 2729 | 2730 | 2731 | 2732 | 2733 | 2734 | 2735 |
| AB0 | 2736 | 2737 | 2738 | 2739 | 2740 | 2741 | 2742 | 2743 | 2744 | 2745 | 2746 | 2747 | 2748 | 2749 | 2750 | 2751 |
| AC0 | 2752 | 2753 | 2754 | 2755 | 2756 | 2757 | 2758 | 2759 | 2760 | 2761 | 2762 | 2763 | 2764 | 2765 | 2766 | 2767 |
| AD0 | 2768 | 2769 | 2770 | 2771 | 2772 | 2773 | 2774 | 2775 | 2776 | 2777 | 2778 | 2779 | 2780 | 2781 | 2782 | 2783 |
| AE0 | 2784 | 2785 | 2786 | 2787 | 2788 | 2789 | 2790 | 2791 | 2792 | 2793 | 2794 | 2795 | 2796 | 2797 | 2798 | 2799 |
| AF0 | 2800 | 2801 | 2802 | 2803 | 2804 | 2805 | 2806 | 2807 | 2808 | 2809 | 2810 | 2811 | 2812 | 2813 | 2814 | 2815 |
| B00 | 2816 | 2817 | 2818 | 2819 | 2820 | 2821 | 2822 | 2823 | 2824 | 2825 | 2826 | 2827 | 2828 | 2829 | 2830 | 2831 |
| B10 | 2832 | 2833 | 2834 | 2835 | 2836 | 2837 | 2838 | 2839 | 2840 | 2841 | 2842 | 2843 | 2844 | 2845 | 2846 | 2847 |
| B20 | 2848 | 2849 | 2850 | 2851 | 2852 | 2853 | 2854 | 2855 | 2856 | 2857 | 2858 | 2859 | 2860 | 2861 | 2862 | 2863 |
| B30 | 2864 | 2865 | 2866 | 2867 | 2868 | 2869 | 2870 | 2871 | 2872 | 2873 | 2874 | 2875 | 2876 | 2877 | 2878 | 2879 |
| B40 | 2880 | 2881 | 2882 | 2883 | 2884 | 2885 | 2886 | 2887 | 2888 | 2889 | 2890 | 2891 | 2892 | 2893 | 2894 | 2895 |
| B50 | 2896 | 2897 | 2898 | 2899 | 2900 | 2901 | 2902 | 2903 | 2904 | 2905 | 2906 | 2907 | 2908 | 2909 | 2910 | 2911 |
| B60 | 2912 | 2913 | 2914 | 2915 | 2916 | 2917 | 2918 | 2919 | 2920 | 2921 | 2922 | 2923 | 2924 | 2925 | 2926 | 2927 |
| B70 | 2928 | 2929 | 2930 | 2931 | 2932 | 2933 | 2934 | 2935 | 2936 | 2937 | 2938 | 2939 | 2940 | 2941 | 2942 | 2943 |
| B80 | 2944 | 2945 | 2946 | 2947 | 2948 | 2949 | 2950 | 2951 | 2952 | 2953 | 2954 | 2955 | 2956 | 2957 | 2958 | 2959 |
| B90 | 2960 | 2961 | 2962 | 2963 | 2964 | 2965 | 2966 | 2967 | 2968 | 2969 | 2970 | 2971 | 2972 | 2973 | 2974 | 2975 |
| BA0 | 2976 | 2977 | 2978 | 2979 | 2980 | 2981 | 2982 | 2983 | 2984 | 2985 | 2986 | 2987 | 2988 | 2989 | 2990 | 2991 |
| BB0 | 2992 | 2993 | 2994 | 2995 | 2996 | 2997 | 2998 | 2999 | 3000 | 3001 | 3002 | 3003 | 3004 | 3005 | 3006 | 3007 |
| BC0 | 3008 | 3009 | 3010 | 3011 | 3012 | 3013 | 3014 | 3015 | 3016 | 3017 | 3018 | 3019 | 3020 | 3021 | 3022 | 3023 |
| BDD0 | 3024 | 3025 | 3026 | 3027 | 3028 | 3029 | 3030 | 3031 | 3032 | 3033 | 3034 | 3035 | 3036 | 3037 | 3038 | 3039 |
| BE0 | 3040 | 3041 | 3042 | 3043 | 3044 | 3045 | 3046 | 3047 | 3048 | 3049 | 3050 | 3051 | 3052 | 3053 | 3054 | 3055 |
| BFO | 3056 | 3057 | 3058 | 3059 | 3060 | 3061 | 3062 | 3063 | 3064 | 3065 | 3066 | 3067 | 3068 | 3069 | 3070 | 3071 |
| C00 | 3072 | 3073 | 3074 | 3075 | 3076 | 3077 | 3078 | 3079 | 3080 | 3081 | 3082 | 3083 | 3084 | 3085 | 3086 | 3087 |
| C10 | 3088 | 3089 | 3090 | 3091 | 3092 | 3093 | 3094 | 3095 | 3096 | 3097 | 3098 | 3099 | 3100 | 3101 | 3102 | 3103 |
| C20 | 3104 | 3105 | 3106 | 3107 | 3108 | 3109 | 3110 | 3111 | 3112 | 3113 | 3114 | 3115 | 3116 | 3117 | 3118 | 3119 |
| C30 | 3120 | 3121 | 3122 | 3123 | 3124 | 3125 | 3126 | 3127 | 3128 | 3129 | 3130 | 3131 | 3132 | 3133 | 3134 | 3135 |
| C40 | 3136 | 3137 | 3138 | 3139 | 3140 | 3141 | 3142 | 3143 | 3144 | 3145 | 3146 | 3147 | 3148 | 3149 | 3150 | 3151 |
| C50 | 3152 | 3153 | 3154 | 3155 | 3156 | 3157 | 3158 | 3159 | 3160 | 3161 | 3162 | 3163 | 3164 | 3165 | 3166 | 3167 |
| C60 | 3168 | 3169 | 3170 | 3171 | 3172 | 3173 | 3174 | 3175 | 3176 | 3177 | 3178 | 3179 | 3180 | 3181 | 3182 | 3183 |
| C70 | 3184 | 3185 | 3186 | 3187 | 3188 | 3189 | 3190 | 3191 | 3192 | 3193 | 3194 | 3195 | 3196 | 3197 | 3198 | 3199 |
| C80 | 3200 | 3201 | 3202 | 3203 | 3204 | 3205 | 3206 | 3207 | 3208 | 3209 | 3210 | 3211 | 3212 | 3213 | 3214 | 3215 |
| C90 | 3216 | 3217 | 3218 | 3219 | 3220 | 3221 | 3222 | 3223 | 3224 | 3225 | 3226 | 3227 | 3228 | 3229 | 3230 | 3231 |
| CA0 | 3232 | 3233 | 3234 | 3235 | 3236 | 3237 | 3238 | 3239 | 3240 | 3241 | 3242 | 3243 | 3244 | 3245 | 3246 | 3247 |
| CB0 | 3248 | 3249 | 3250 | 3251 | 3252 | 3253 | 3254 | 3255 | 3256 | 3257 | 3258 | 3259 | 3260 | 3261 | 3262 | 3263 |
| CC0 | 3264 | 3265 | 3266 | 3267 | 3268 | 3269 | 3270 | 3271 | 3272 | 3273 | 3274 | 3275 | 3276 | 3277 | 3278 | 3279 |
| CD0 | 3280 | 3281 | 3282 | 3283 | 3284 | 3285 | 3286 | 3287 | 3288 | 3289 | 3290 | 3291 | 3292 | 3293 | 3294 | 3295 |
| CE0 | 3296 | 3297 | 3298 | 3299 | 3300 | 3301 | 3302 | 3303 | 3304 | 3305 | 3306 | 3307 | 3308 | 3309 | 3310 | 3311 |
| CF0 | 3312 | 3313 | 3314 | 3315 | 3316 | 3317 | 3318 | 3319 | 3320 | 3321 | 3322 | 3323 | 3324 | 3325 | 3326 | 3327 |

HEXADECIMAL-DECIMAL INTEGER CONVERSION (Cont.)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|-----|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| D00 | 3328 | 3329 | 3330 | 3331 | 3332 | 3333 | 3334 | 3335 | 3336 | 3337 | 3338 | 3339 | 3340 | 3341 | 3342 | 3343 |
| D10 | 3344 | 3345 | 3346 | 3347 | 3348 | 3349 | 3350 | 3351 | 3352 | 3353 | 3354 | 3355 | 3356 | 3357 | 3358 | 3359 |
| D20 | 3360 | 3361 | 3362 | 3363 | 3364 | 3365 | 3366 | 3367 | 3368 | 3369 | 3370 | 3371 | 3372 | 3373 | 3374 | 3375 |
| D30 | 3376 | 3377 | 3378 | 3379 | 3380 | 3381 | 3382 | 3383 | 3384 | 3385 | 3386 | 3387 | 3388 | 3389 | 3390 | 3391 |
| D40 | 3392 | 3393 | 3394 | 3395 | 3396 | 3397 | 3398 | 3399 | 3400 | 3401 | 3402 | 3403 | 3404 | 3405 | 3406 | 3407 |
| D50 | 3408 | 3409 | 3410 | 3411 | 3412 | 3413 | 3414 | 3415 | 3416 | 3417 | 3418 | 3419 | 3420 | 3421 | 3422 | 3423 |
| D60 | 3424 | 3425 | 3426 | 3427 | 3428 | 3429 | 3430 | 3431 | 3432 | 3433 | 3434 | 3435 | 3436 | 3437 | 3438 | 3439 |
| D70 | 3440 | 3441 | 3442 | 3443 | 3444 | 3445 | 3446 | 3447 | 3448 | 3449 | 3450 | 3451 | 3452 | 3453 | 3454 | 3455 |
| D80 | 3456 | 3457 | 3458 | 3459 | 3460 | 3461 | 3462 | 3463 | 3464 | 3465 | 3466 | 3467 | 3468 | 3469 | 3470 | 3471 |
| D90 | 3472 | 3473 | 3474 | 3475 | 3476 | 3477 | 3478 | 3479 | 3480 | 3481 | 3482 | 3483 | 3484 | 3485 | 3486 | 3487 |
| DA0 | 3488 | 3489 | 3490 | 3491 | 3492 | 3493 | 3494 | 3495 | 3496 | 3497 | 3498 | 3499 | 3500 | 3501 | 3502 | 3503 |
| DB0 | 3504 | 3505 | 3506 | 3507 | 3508 | 3509 | 3510 | 3511 | 3512 | 3513 | 3514 | 3515 | 3516 | 3517 | 3518 | 3519 |
| DC0 | 3520 | 3521 | 3522 | 3523 | 3524 | 3525 | 3526 | 3527 | 3528 | 3529 | 3530 | 3531 | 3532 | 3533 | 3534 | 3535 |
| DD0 | 3536 | 3537 | 3538 | 3539 | 3540 | 3541 | 3542 | 3543 | 3544 | 3545 | 3546 | 3547 | 3548 | 3549 | 3550 | 3551 |
| DE0 | 3552 | 3553 | 3554 | 3555 | 3556 | 3557 | 3558 | 3559 | 3560 | 3561 | 3562 | 3563 | 3564 | 3565 | 3566 | 3567 |
| DF0 | 3568 | 3569 | 3570 | 3571 | 3572 | 3573 | 3574 | 3575 | 3576 | 3577 | 3578 | 3579 | 3580 | 3581 | 3582 | 3583 |
| E00 | 3584 | 3585 | 3586 | 3587 | 3588 | 3589 | 3590 | 3591 | 3592 | 3593 | 3594 | 3595 | 3596 | 3597 | 3598 | 3599 |
| E10 | 3600 | 3601 | 3602 | 3603 | 3604 | 3605 | 3606 | 3607 | 3608 | 3609 | 3610 | 3611 | 3612 | 3613 | 3614 | 3615 |
| E20 | 3616 | 3617 | 3618 | 3619 | 3620 | 3621 | 3622 | 3623 | 3624 | 3625 | 3626 | 3627 | 3628 | 3629 | 3630 | 3631 |
| E30 | 3632 | 3633 | 3634 | 3635 | 3636 | 3637 | 3638 | 3639 | 3640 | 3641 | 3642 | 3643 | 3644 | 3645 | 3646 | 3647 |
| E40 | 3648 | 3649 | 3650 | 3651 | 3652 | 3653 | 3654 | 3655 | 3656 | 3657 | 3658 | 3659 | 3660 | 3661 | 3662 | 3663 |
| E50 | 3664 | 3665 | 3666 | 3667 | 3668 | 3669 | 3670 | 3671 | 3672 | 3673 | 3674 | 3675 | 3676 | 3677 | 3678 | 3679 |
| E60 | 3680 | 3681 | 3682 | 3683 | 3684 | 3685 | 3686 | 3687 | 3688 | 3689 | 3690 | 3691 | 3692 | 3693 | 3694 | 3695 |
| E70 | 3696 | 3697 | 3698 | 3699 | 3700 | 3701 | 3702 | 3703 | 3704 | 3705 | 3706 | 3707 | 3708 | 3709 | 3710 | 3711 |
| E80 | 3712 | 3713 | 3714 | 3715 | 3716 | 3717 | 3718 | 3719 | 3720 | 3721 | 3722 | 3723 | 3724 | 3725 | 3726 | 3727 |
| E90 | 3728 | 3729 | 3730 | 3731 | 3732 | 3733 | 3734 | 3735 | 3736 | 3737 | 3738 | 3739 | 3740 | 3741 | 3742 | 3743 |
| EA0 | 3744 | 3745 | 3746 | 3747 | 3748 | 3749 | 3750 | 3751 | 3752 | 3753 | 3754 | 3755 | 3756 | 3757 | 3758 | 3759 |
| EB0 | 3760 | 3761 | 3762 | 3763 | 3764 | 3765 | 3766 | 3767 | 3768 | 3769 | 3770 | 3771 | 3772 | 3773 | 3774 | 3775 |
| EC0 | 3776 | 3777 | 3778 | 3779 | 3780 | 3781 | 3782 | 3783 | 3784 | 3785 | 3786 | 3787 | 3788 | 3789 | 3790 | 3791 |
| ED0 | 3792 | 3793 | 3794 | 3795 | 3796 | 3797 | 3798 | 3799 | 3800 | 3801 | 3802 | 3803 | 3804 | 3805 | 3806 | 3807 |
| EE0 | 3808 | 3809 | 3810 | 3811 | 3812 | 3813 | 3814 | 3815 | 3816 | 3817 | 3818 | 3819 | 3820 | 3821 | 3822 | 3823 |
| EF0 | 3824 | 3825 | 3826 | 3827 | 3828 | 3829 | 3830 | 3831 | 3832 | 3833 | 3834 | 3835 | 3836 | 3837 | 3838 | 3839 |
| F00 | 3840 | 3841 | 3842 | 3843 | 3844 | 3845 | 3846 | 3847 | 3848 | 3849 | 3850 | 3851 | 3852 | 3853 | 3854 | 3855 |
| F10 | 3856 | 3857 | 3858 | 3859 | 3860 | 3861 | 3862 | 3863 | 3864 | 3865 | 3866 | 3867 | 3868 | 3869 | 3870 | 3871 |
| F20 | 3872 | 3873 | 3874 | 3875 | 3876 | 3877 | 3878 | 3879 | 3880 | 3881 | 3882 | 3883 | 3884 | 3885 | 3886 | 3887 |
| F30 | 3888 | 3889 | 3890 | 3891 | 3892 | 3893 | 3894 | 3895 | 3896 | 3897 | 3898 | 3899 | 3900 | 3901 | 3902 | 3903 |
| F40 | 3904 | 3905 | 3906 | 3907 | 3908 | 3909 | 3910 | 3911 | 3912 | 3913 | 3914 | 3915 | 3916 | 3917 | 3918 | 3919 |
| F50 | 3920 | 3921 | 3922 | 3923 | 3924 | 3925 | 3926 | 3927 | 3928 | 3929 | 3930 | 3931 | 3932 | 3933 | 3934 | 3935 |
| F60 | 3936 | 3937 | 3938 | 3939 | 3940 | 3941 | 3942 | 3943 | 3944 | 3945 | 3946 | 3947 | 3948 | 3949 | 3950 | 3951 |
| F70 | 3952 | 3953 | 3954 | 3955 | 3956 | 3957 | 3958 | 3959 | 3960 | 3961 | 3962 | 3963 | 3964 | 3965 | 3966 | 3967 |
| F80 | 3968 | 3969 | 3970 | 3971 | 3972 | 3973 | 3974 | 3975 | 3976 | 3977 | 3978 | 3979 | 3980 | 3981 | 3982 | 3983 |
| F90 | 3984 | 3985 | 3986 | 3987 | 3988 | 3989 | 3990 | 3991 | 3992 | 3993 | 3994 | 3995 | 3996 | 3997 | 3998 | 3999 |
| FA0 | 4000 | 4001 | 4002 | 4003 | 4004 | 4005 | 4006 | 4007 | 4008 | 4009 | 4010 | 4011 | 4012 | 4013 | 4014 | 4015 |
| FB0 | 4016 | 4017 | 4018 | 4019 | 4020 | 4021 | 4022 | 4023 | 4024 | 4025 | 4026 | 4027 | 4028 | 4029 | 4030 | 4031 |
| FC0 | 4032 | 4033 | 4034 | 4035 | 4036 | 4037 | 4038 | 4039 | 4040 | 4041 | 4042 | 4043 | 4044 | 4045 | 4046 | 4047 |
| FD0 | 4048 | 4049 | 4050 | 4051 | 4052 | 4053 | 4054 | 4055 | 4056 | 4057 | 4058 | 4059 | 4060 | 4061 | 4062 | 4063 |
| FE0 | 4064 | 4065 | 4066 | 4067 | 4068 | 4069 | 4070 | 4071 | 4072 | 4073 | 4074 | 4075 | 4076 | 4077 | 4078 | 4079 |
| FF0 | 4080 | 4081 | 4082 | 4083 | 4084 | 4085 | 4086 | 4087 | 4088 | 4089 | 4090 | 4091 | 4092 | 4093 | 4094 | 4095 |



West: 17291 Irvine Blvd., Suite 262/(714)838-1126, TWX: 910-595-1114/Tustin, California 92680

Mid-America: 800 Southgate Office Plaza/501 West 78th St./ (612)835-6722, TWX: 910-576-2867/Bloomington, Minnesota 55437

Northeast: 2 Militia Drive, Suite 4/(617)861-1136, Telex: 92-3493/Lexington, Massachusetts 02173

Mid-Atlantic: 21 Bala Avenue/(215)664-6636/Bala Cynwyd, Pennsylvania 19004

Europe: Intel Office/Vester Farimagsgade 7/45-1-11 5644, Telex: 19567/DK 1606 Copenhagen V

Orient: Intel Japan Corp./Han-Ei 2nd Building/1-1, Shinjuku, Shinjuku-Ku/03-354-8251, Telex: 781-28426/Tokyo 160

© 1973/Printed in U.S.A./MCS-030-1273-150