Pyntel4004

Release ENV_VERSION

Andrew Shapton

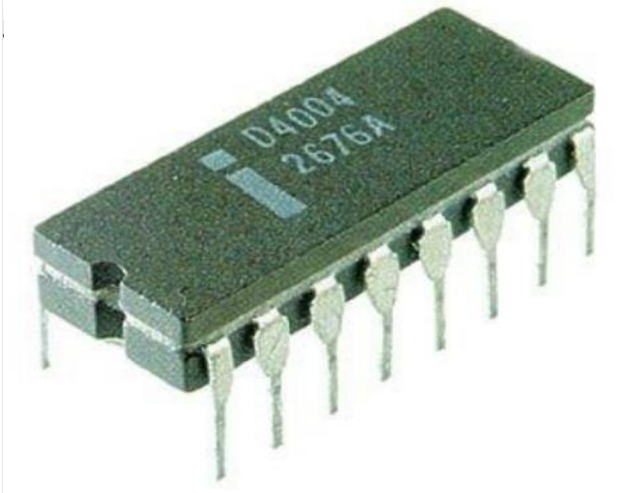
Jun 28, 2022

CONTENTS

CHAPTER

ONE

INTEL 4004 CHIP HISTORY



In 1969 Busicom contracted Intel to design a set of chips to be used in a new high perfomance calculator. Ted Hoff, Federico Faggin and Stan Mazor came up with a design that involved four different chips. The CPU was eventually to be called a microprocessor.

Later Intel negotiated for a return of the rights for the chips, which had gone to Busicom in the original contract.

The 4000 Family (A.K.A. Busicom Chip Set / MCS-4 Chip Set)

The 4000 family consisted of four different chips:

- a 2048-bit ROM with a 4-bit programmable input-output port (*4001*)
- a 4-registers x 20-locations x 4-bit RAM data memory with a 4-bit output port (*4002*)
- an input-output expansion chip, consisting of a static shift register with serial input and serial and parallel output (*4003*)
- a 4-bit CPU chip (*4004*)

Other chips in the 4xxx family were:

- an 8-bit address latch for access to standard memory chips, and one built-in 4-bit chip select and I/O port (4008)
- a program and I/O access converter to standard memory and I/O chips (4009)
- an 8192-bit(1024 × 8) ROM w/ 4-bit I/O Ports (4208)
- a general purpose Bye I/O port (4211))
- a keyboard/display interface (4269)

- a memory interface (combined functions of 4008 and 4009)(4289)
- an 8k mask-programming ROM (4308)
- a 16384-bit (2048 × 8) Static ROM (4316)
- a 2048-bit (256 × 8) EPROM (4702)
- a 5.185 MHz Clock Generator Crystal for 4004/4201A or 4040/4201A (4801)

All the chips were packaged in 16-pin, dual-in-line packages. This package restriction was imposed by Intel's management, who at the time considered any package with more that 16 pins uneconomical, despite the fact that 40-pin packages were widely used by other semiconductor companies.

This unfortunate choice considerably constrained the performance of the system. Address and data had to be multiplexed onto the pins (one of the claims of `Patent number US3821785`), causing a major penalty in the instruction cycle execution.

The instruction cycle of 10.8 microseconds could have been easily reduced to 4 microseconds by a more appropriate package choice.

The 4000-family was completed by March 1971, in production by June 1971 and introduced to the general market in November 1971 with the name MCS-4.

1.1 MCS-4

The MCS-4 is a microprogrammable computer set designed for applications such as test systems, peripherals, terminals, billing machines, measuring machines, numeric and process control.

The 4004 CPU, 4003 Shift Register, and 4002 RAM are standard building blocks. The 4001 ROM contains the custom microprogram and is implemented as a metal mask according to customer specifications.

MCS-4 systems readily interface to switches, keyboards, displays, teletypewriters, printers, readers, A-D converters and other popular peripherals.

A system built with the MCS-4 micro computer set can have up to 4k * 8-bit ROM words, 8192 * 4-bit RAM characters, and 128 I/O lines without requiring any interface logic. By adding a few gates, the MCS-4 can have up to 48 RAM and ROM packages in any combination, and 192 I/O lines. The minimum configuration consists of one CPU and one 256 * 8-bit ROM.

The MCS-4 has a very powerful instruction set that allows both binary and decimal arithmetic. It includes conditional branching, jump to subroutine, and provides for the efficient use of ROM look-up tables by indirect fetching.

The Intel MCS-4 micro computer set (4001/2/3/4) is fabricated with Silicon Gate Technology . This low threshold technology allows the design and production of higher performance MOS circuits and provides a higher functional density on a monolithic chip than conventional MOS technologies.

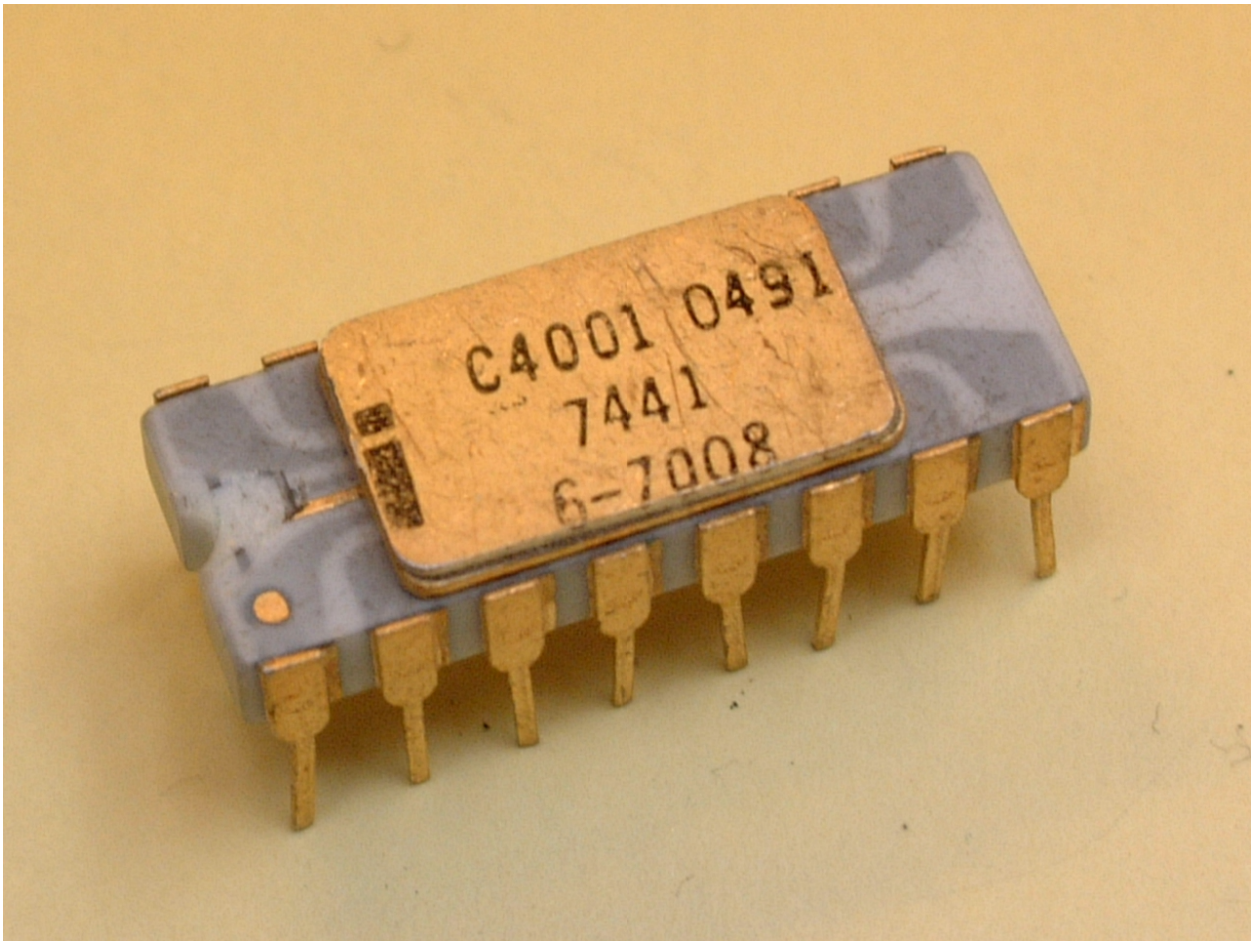
1.2 Busicom 141-PF

In the case of the Busicom 141-PF (also marketed as the NCR-18-36), the ROM contained the custom microprogramming to allow the MCS-4 chipset to operate as a calculator.

CHAPTER

TWO

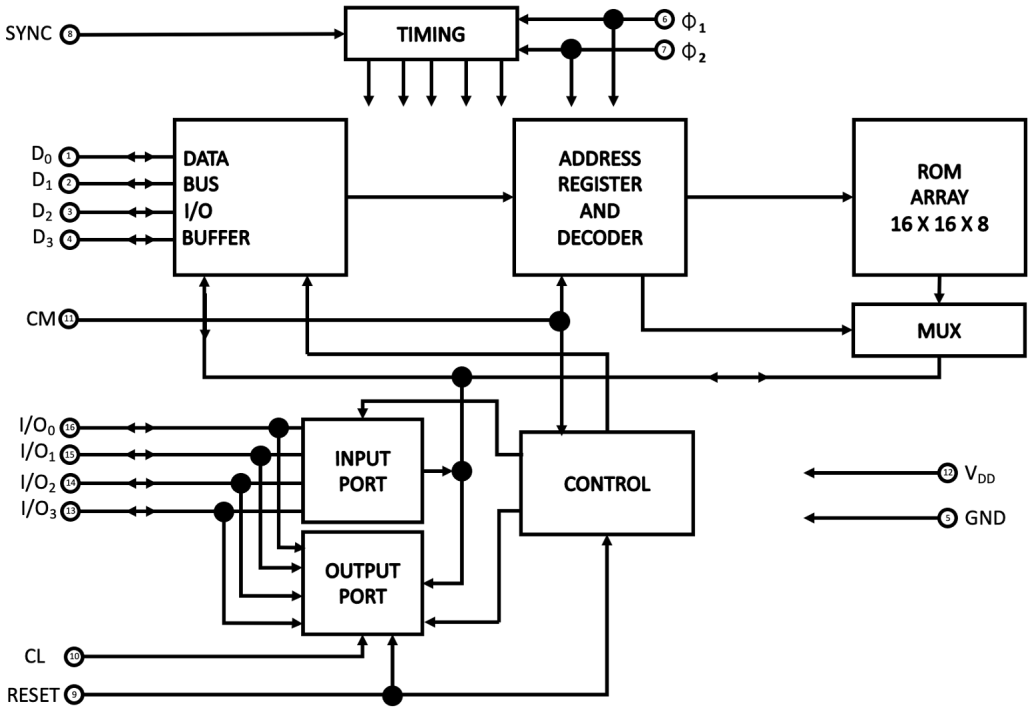
THE INTEL 4001 CHIP



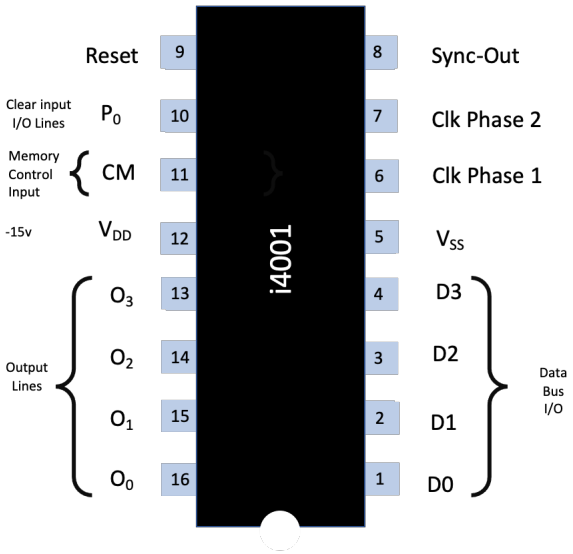
The Intel 4001 chip was introduced in 1971 as part of the Intel 4000 family; a fully decoded static Random Access Memory chip, fabricated with P-channel silicon gate MOS technology

It is a 2048-bit metal mask programmable ROM providing custom microprogramming capability for the MCS-4 micro computer set. It is organised as 256 x 8-bit words.

Logically, the Intel 4001 is set out as shown:



The circled numbers relate to the pins as shown below:



Address and data are transferred in and out by time multiplexing on 4 data bus lines. Timing is internally generated using two clock signals ϕ_1 and ϕ_2, and a SYNC signal supplied by the 4004. Addressed are received from the CPU on three time periods following SYNC, and select 1 out of 256 words and 1 out of 16 ROMs.

For that purpose, each ROM is identified #0, 1, 2, through 15 by metal option. A Command Line (CM) is also provided and its scope is to select a ROM bank (group of 16 ROM's).

During the two time periods (M_1 & M_2) following the addressing time, information is transferred from the ROM to the data bus lines.

A second mode of operation of the ROM is as an Input/output control device. In that mode, a ROM chip will route information to and from data bus lines in and out of 4 I/O external lines. Each chip has the capability to identify itself for an I/O port operation, recognise an I/O port instruction and decide whether it is an Input or Output operation and

execute the instruction.

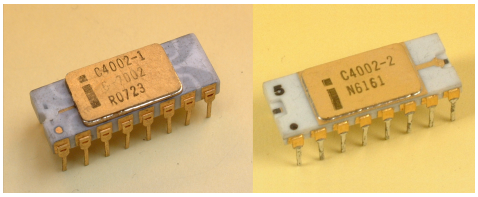
An external signal (CL) will asynchronously clear the output register during normal operation. All internal flip flops (including the output register) will be reset when the RESET line goes low (negative voltage).

Each I/O pin can be uniquely chosen as either an input or output port by metal option when ordering. An example order form can be downloaded `here` Direct or inverted input or output is optional. An on-chip resistor at the input pins connected to either V_{dd} or V_{ss} is also optional (see ordering information on page 12).

CHAPTER

THREE

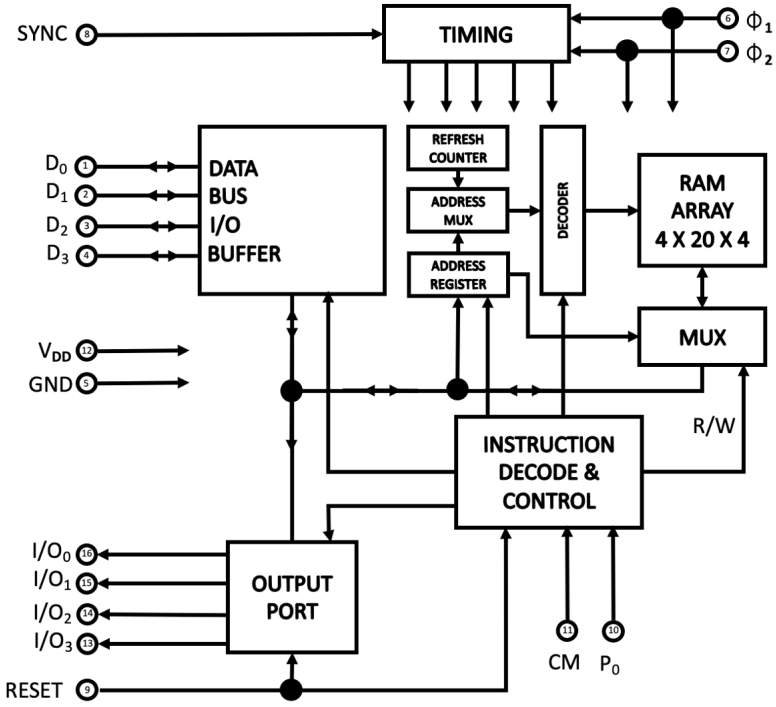
THE INTEL 4002 CHIP



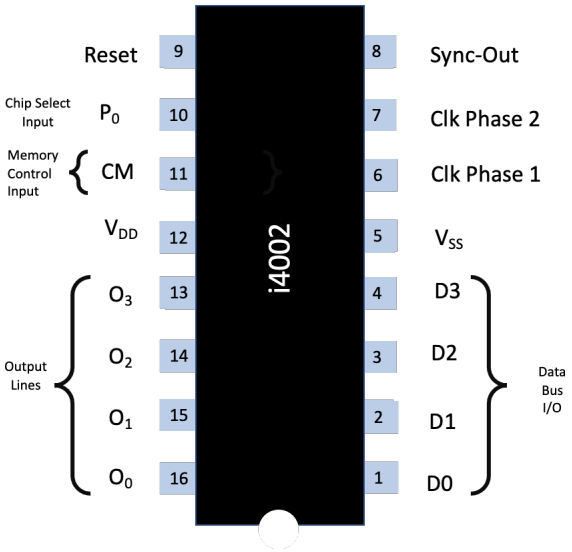
The Intel 4002 chip was introduced in 1971 as part of the Intel 4000 family; a 320-bit MOS RAM and 4-bit output port, fabricated with P-channel silicon gate MOS technology

The 4002 was designed to be used with other MCS-4/40 devices such as the 4004 CPU. The chip was available in two different metal options 4002-1 and 4002-2 this was to make it possible to extend the chip selection so that 4pcs of 4002 chips could be connected to the 4004 CPU without any external chip selection logic. Although produced by Intel, National Semiconductors was the only second source.

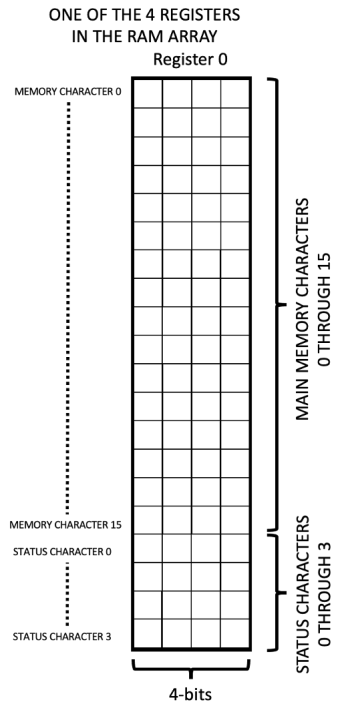
Logically, the Intel 4002 is set out as shown:



The circled numbers relate to the pins as shown below:

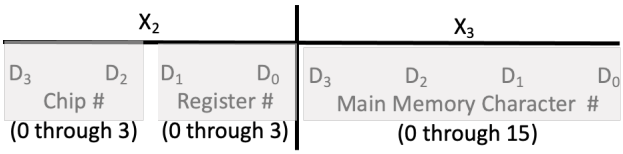


The 4002 performs two functions. As a RAM, it stores 320 bits arranged in 4 registers of 20 x 4-bit characters each (16 main memory characters and 4 status characters).



In the RAM mode, the operation is as follows: When the CPU executes an *SRC* instruction, it will send out the contents of the designated index register pair during X_2 and X_2 as an address to RAM, and will activate 1 CM-RAM line at X_2 for the *previously selected RAM bank* (see basic instruction cycle on page 5).

The data at X_2 and X_3 is interpreted as shown below:



As a vehicle for communication with peripheral devices, it is provided with 4 output lines and associated control logic to perform output operations.

The status character locations (0 through 3) are selected by the OPA portion of one of the I/O and RAM instructions.

For chip selection, the 4002 is available in two metal options, 4002-1 and 4002-2. An extra pin, P_0, (which may be hard wired to either V_{DD} or V_{SS}) is also available for chip selection.

The chip number is assigned as follows:

Chip #	4002 Option	P_0	D_3 @ X_2	D_2 @ X_2
0	4002-1	GND	0	0
1	4002-1	V_{DD}	0	1
2	4002-2	GND	1	0
3	4002-2	V_{DD}	1	1

Timing is internally generated using two clock signals X_1 and X_2, and a SYNC signal provided by the 4004. Internal refresh circuitry maintains data levels in the cells.

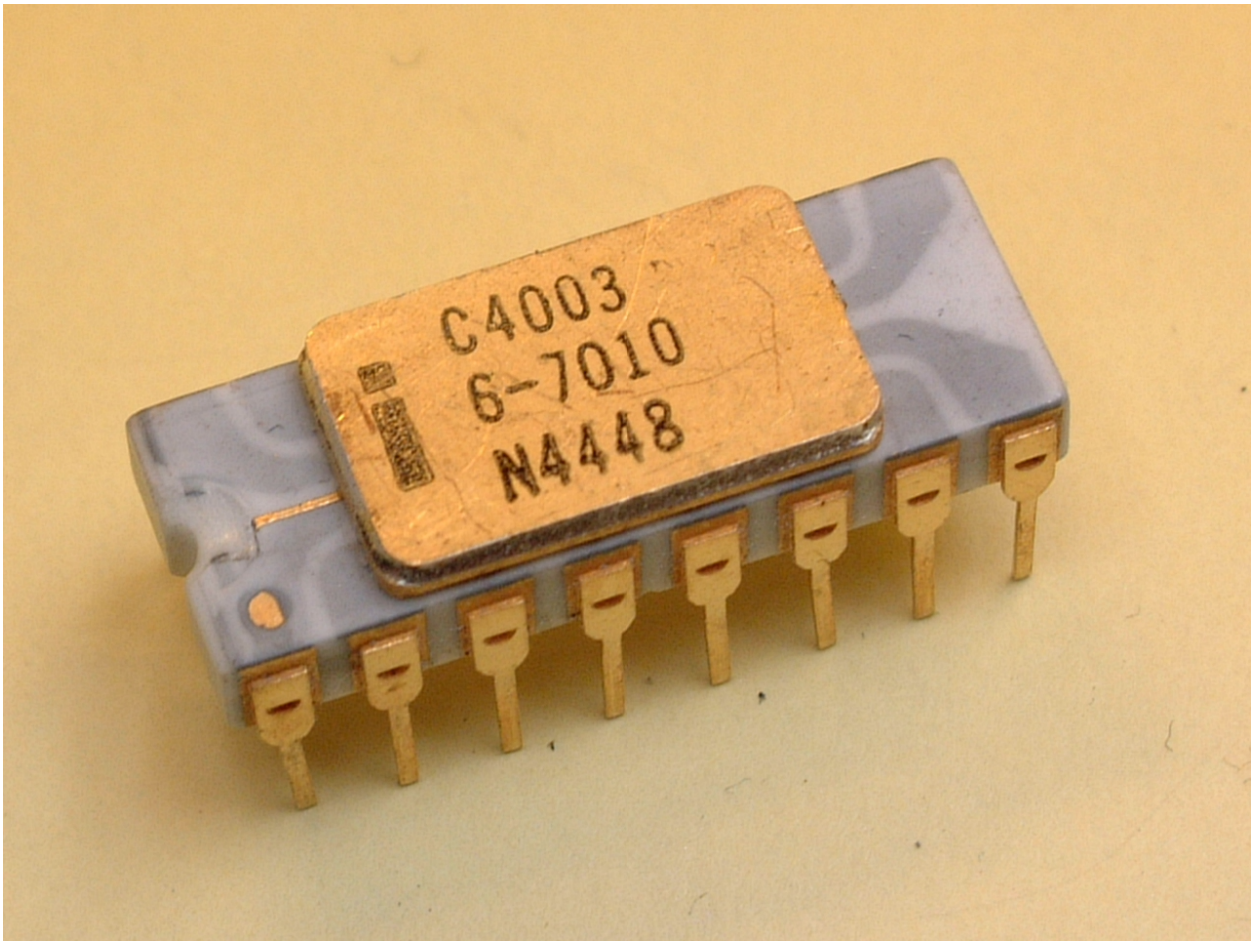
All communications with the system is through the data bus. The I/O port permits data out of the system. When the external RESET signal goes low, the memory and all static flip-flops (including the output registers) will be cleared. To fully clear the memory, the RESET signal must be maintained for at least 32 memory cycles (32 x 8 clock periods).

Note: Previously Selected Ram Bank Bank switching is accomplished by the CPU after receiving a "*DCL*" (designate command line) instruction. Prior to the execution of the DCL instruction the desired CM-RAM$_i$ code has been stored in the accumulator (for example, through an LDM instruction). During DCL the CM-RAM$_1$ code is transferred from the accumulator to the CM-RAM register. The RAM bank is then selected starting with the next instruction.

CHAPTER

FOUR

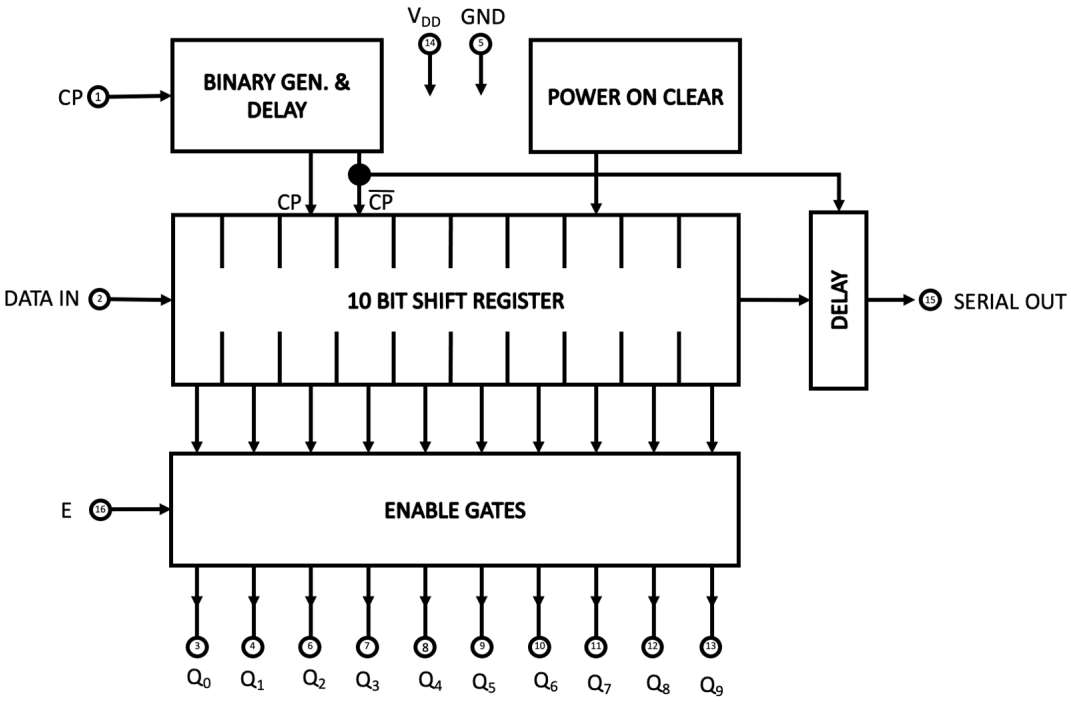
THE INTEL 4003 CHIP



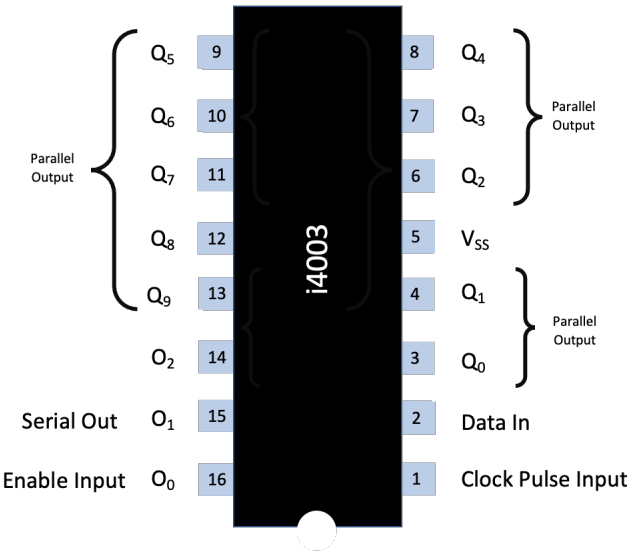
The Intel 4003 chip was introduced in 1971 as part of the Intel 4000 family; 10-bit Serial-in/Parallel-out, Serial-out Shift Register, fabricated with P-channel silicon gate MOS technology.

The 4003 was designed to be used with other MCS-4/40 devices such as the 4004 CPU. Although produced by Intel, National Semiconductors was the only second source.

Logically, the Intel 4003 is set out as shown:



The circled numbers relate to the pins as shown below:



The 4003 is a 10-bit static shift register with serial-in, parallel-out and serial-out data.

Its function is to increase the number of output lines to interface with I/O devices such as keyboards, displays, printers, teletypes, switchers, readers, A-D converters, etc.

Data is loaded serially and is available in parallel on 10 output lines which are accessed through enable logic. When enabled (E = low), the shift register contents is read out; when not enabled (E = high), the parallel-out lines are at V_{SS}. The serial-out line is not affected by the enable logic.

Data is also available serially permitting an indefinite number of similar devices to be cascaded together to provide shift register length multiples of 10.

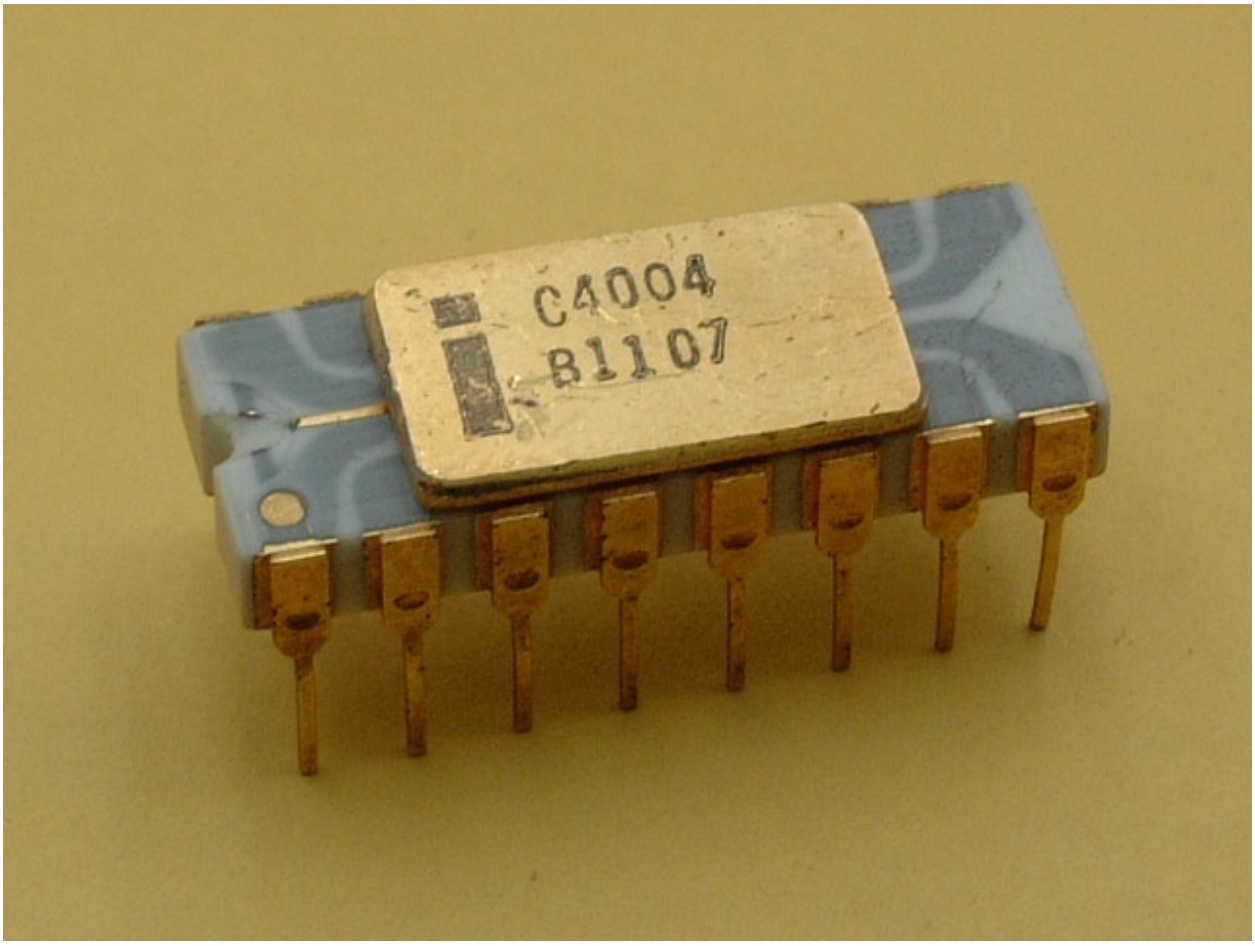
The data shifting is controlled by the CP signal. An internal power-on-clear circuit (`Patent number US3821785`)

will clear the shift register ($Q_i = V_{SS}$) between the application of a supply voltage and the first CP signal.

CHAPTER

FIVE

THE INTEL 4004 CHIP



5.1 Instruction Set Format

5.1.1 Machine instructions

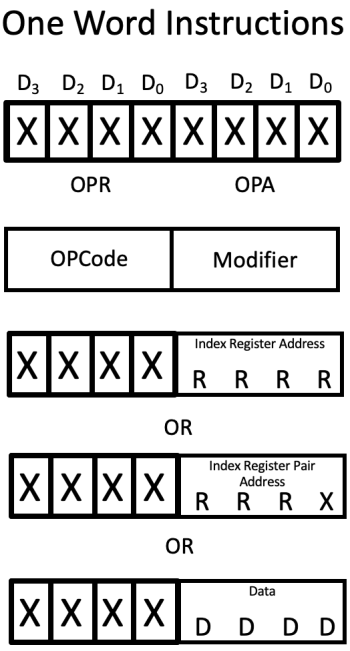
The Intel 4004 chip Machine Instructions consist of:

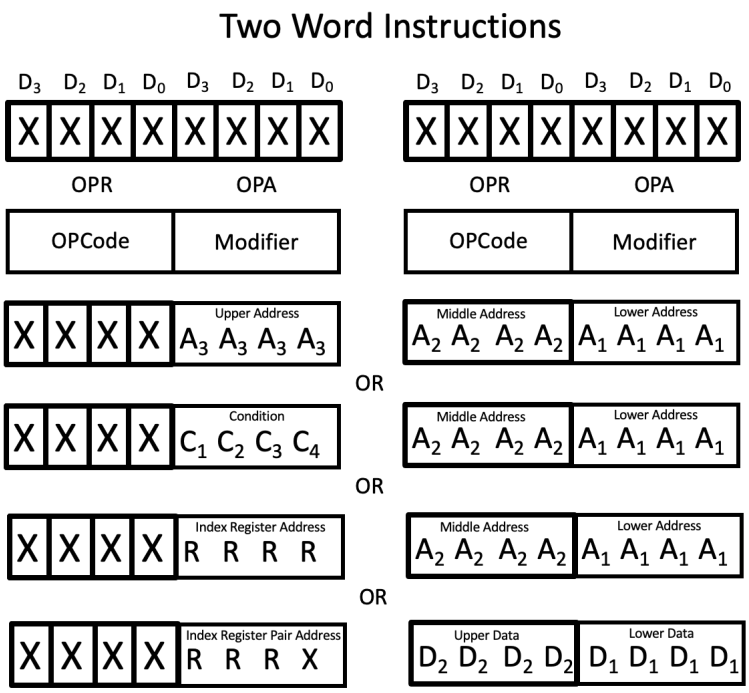
- 1 word instructions - 8 bits requiring 8 clock periods (instruction cycle)
- 2 word instructions - 16 bits requiring 16 clock periods (2 instruction cycles)

Each instruction is divided into two 4 bit fields. The upper 4 bits is the **OPR** field containing the operation code. The lower 4 bits is the **OPA** field containing the modifier.

For 2 word instructions, the second word contains the address information or data.

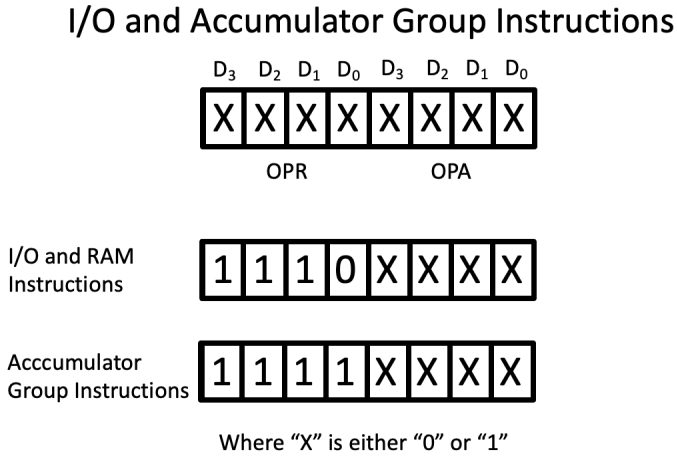
The upper 4 bits (OPR) will always be fetched befor the lower 4 bits (OPA) during M_1 and M_2 times respectively.





5.1.2 Input/Output, RAM, and Accumulator Group instructions

In these instructions (which are all 1 word),the OPR contains a 4 bit code which identifies either the I/O instruction or the accumulator group instruction, and the OPA contains a 4 bit code which identifies the operation to be performed. The table below illustrates the contents of each 4 bit field:



The Intel 4004 chip was introduced in 1971 as part of the Intel 4000 family; 4-bit central processing unit (CPU), fabricated with P-channel silicon gate MOS technology.

The 4004 was designed to be used with other members of the MCS-4/40 family (4001, 4002, 4003).

The *packaging* of the Intel 4004 (and the Second Source manufacturers) is shown below:

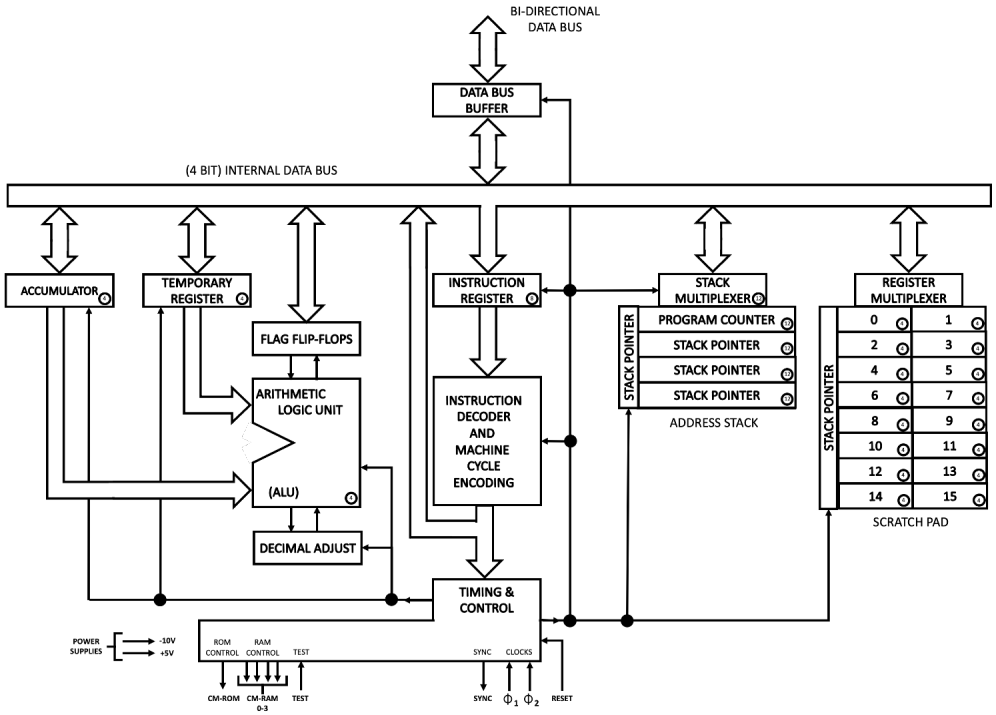
Manufacturer	Model	Package
Manufacturer	Model	Package
Intel	C4004	16-pin Ceramic DIP
Intel	D4004	16-pin Ceramic DIP
Intel	P4004	16-pin Plastic DIP
National Semiconductor	INS4004D	16-pin Ceramic DIP
National Semiconductor	INS4004J	16-pin side-brazed Ceramic DIP
Hitachi	HD35404	16-pin DIP
Microsystems International	MF7114	

Internally, the 4004 is a 4-bit microprocessor with 8-bit instructions. It is clocked at a frequency of 500KHz - 740KHz. It contains 4096x8-bit ROM and 1280x4-bit RAM, with 2,300 transistors at a 10 micron definition. There are 45 instructions (46 including NOP) with a 4 level stack and sixteen 4-bit (or eight 8-bit) registers

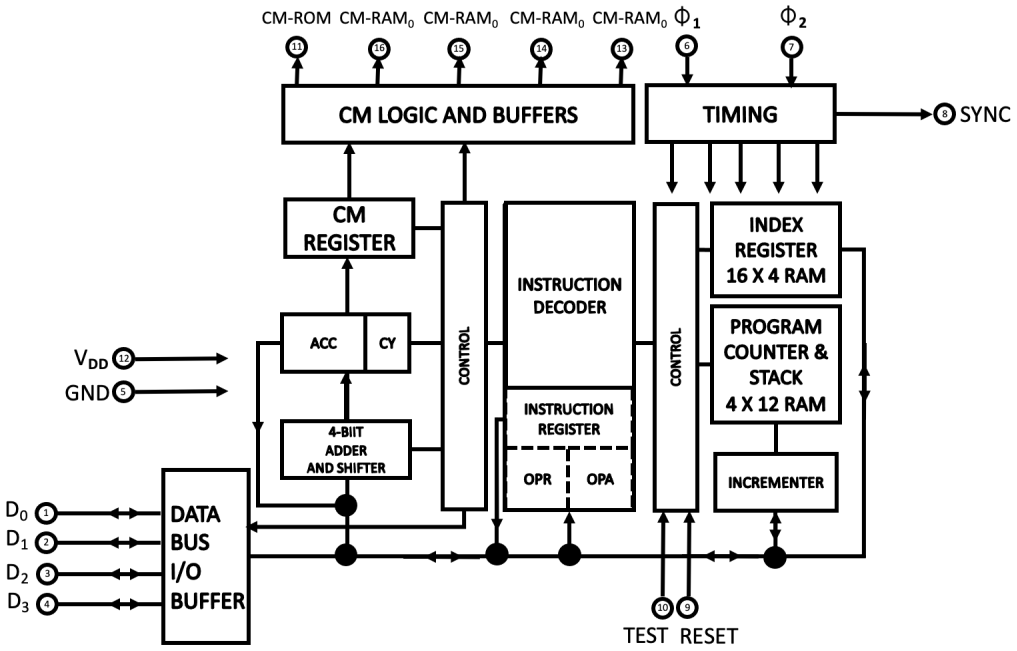
For more detail, see the *hardware characteristics* or the *instruction format*.

Logically, the Intel 4004 is set out as shown:

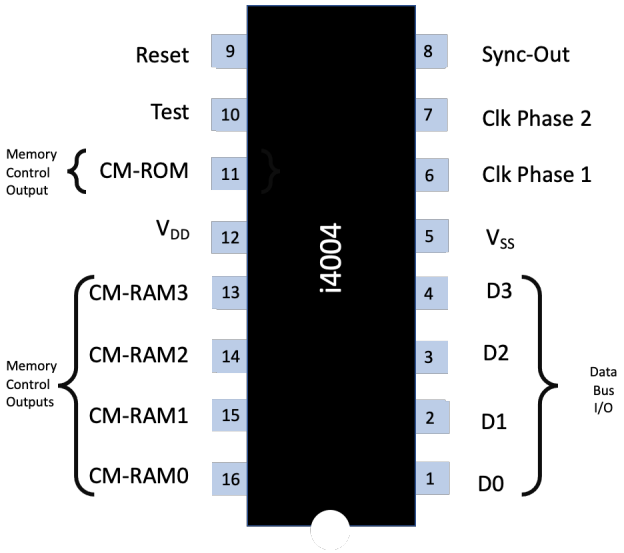
Internal



including external



The circled numbers relate to the pins as shown below:



Pins described as follows:

Pin(s)	Description
D0 - D3	Bi-Directional Data Bus. All address and data communication between the processor and the RAM and ROM chips occurs on these 4 lines
V_{SS}	Most positive voltage
Clk-Phase 1 Clk-Phase 2	2 phase clock inputs
SYNC	SYNC Output. SYnchronisation signal generated by the processor and sent to the ROM and RAM chips. It indicates the beginning of an instruction cycle
RESET	RESET input. A logic '1' at this input clears all flags and status registers and resets the program counter to zero. To completely clear all address and index registers, RESET must be applied for 64 clock cycles (8 macxhine cycles)
TEST	TEST input. The logical state of this signal can be tested with the JCN instruction
CM-ROM	CM-ROM Output. This is the ROM selection signal sent out by the processor when data is required from program memory
V_{DD}	V_{DD} - 15 +/-5% main supply voltage
CM-RAM0 - CM-RAM3	CM-RAM Output. These are the bank selection signals for the 4001 and 4002 RAM chips in the system

The CPU consists of the following components:

	Component
a	4-bit adder
b	64-bit (16 x 4) index register
c	48-bit Program Counter
d	Stack (nesting up to 3 levels if possible)
e	Address incrementer
f	8-bit instruction register and decoder
g	Control logic

Information flows between the 4004 and the other chips through a 4-line data bus. One 4004 may be combined with up to 48 ROM (4001) and RAM (4002) chips in any combination.

A typical machine cycle starts with the CPU sending a synchronisation signal (SYNC) to the ROMs and RAMs. Next, 12 bits of ROM address are sent to the data bus using three clock cycles (@ 0.75Mhz). The address is then incremented by one and stored in the Program Counter.

The selected ROM sends back 8 bits of instruction or data during the following two clock cycles.

This information is stored in two registers: OPR and OPA. The next three clock cycles are used to execute the instruction. (See Basic Instruction Cycle on Page 5.)

The ROM bank is controlled by a command ROM control signal (CM-ROM) and up to four RAM banks are controlled by four RAM control signals (CM-RAM $_0$, CM-RAM $_1$, CM-RAM $_2$, CM-RAM $_3$)

Bank switching is accomplished by the execution of a "*DCL*" instruction.

An input test signal (TEST) is used in conjunction with the jump on condition ("*JCN*") instruction. An external RESET signal is used to clear all registers and flip-flops. To fully clear all registers, the RESET signal must be applied for at least 8 memory cycles (8 x 8 clock periods). After RESET the program will start from "0" step and CM-RAM $_0$ will be selected.

The instruction repertoire of the 4004 consists of :

Instruction Type	Number
Machine instructions (5 of which are double length)	16
Accumulator group instructions	14
Input/output and RAM instructions	16
Total	45
No-Operation	1
TOTAL	**46**

Note: Bank Switching Bank switching is accomplished by the CPU after receiving a "DCL" (designate command line) instruction. Prior to the execution of the DCL instruction the desired CM-RAM $_i$ code has been stored in the accumulator (for example, through an LDM instruction). During DCL the CM-RAM $_i$ code is transferred from the accumulator to the CM-RAM register. The RAM bank is then selected starting with the next instruction.

CHAPTER

SIX

MCS-4 SYSTEM INTERCONNECTIONS

The MCS-4 uses a 10.8 μ sec instruction cycle. The CPU (4004) generates a synchronisation (SYNC) signal, indicating the start of an instruction cycle, and sends it to the ROMs (4001) and RAMs (4002).

Basic instruction execution requires 8 or 16 cycles of a 750 kHz clock. In a typical sequence, the CPU sends 12 bits of address to the ROMs in three cycles (A_1, A_2, A_3). The selected ROM sends back 8 bits of instruction (OPR, OPA) to the CPU in the next two cycles (M_1, M_2). The instruction is then interpreted and executed in the final 3 cycles (X_1, X_2, X_3).

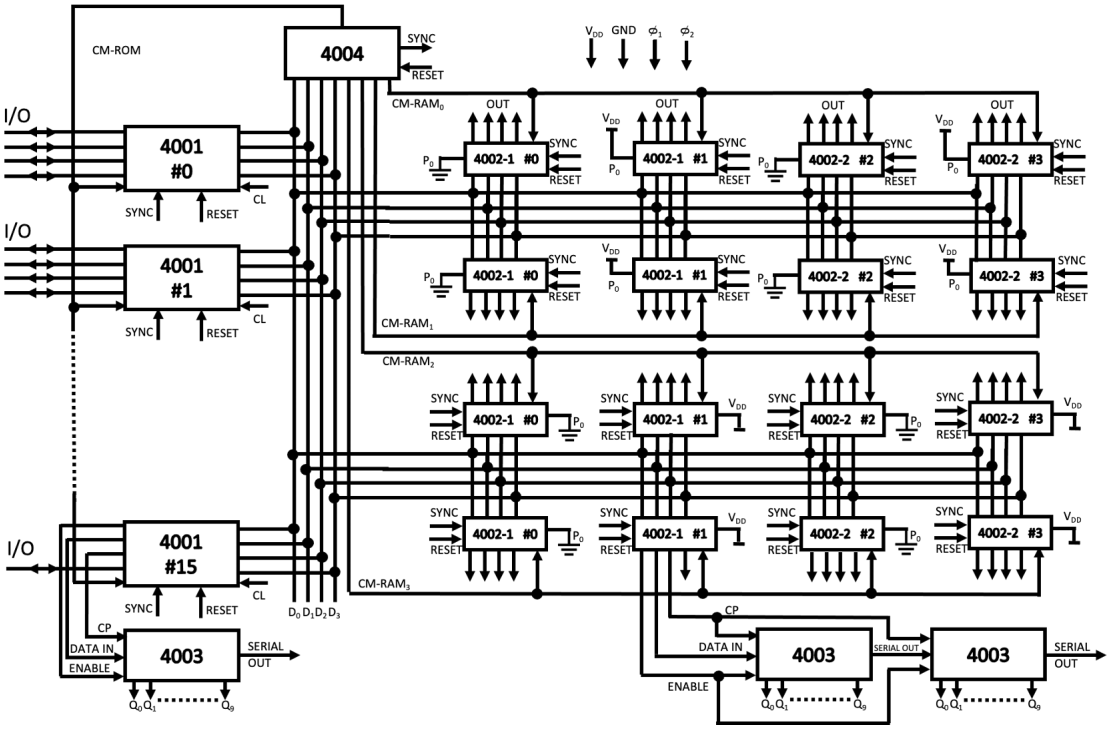
The CPU, RAMs and ROMs can be controlled by an external RESET line. While RESET is activated the contents of the registers and flip-flops are cleared. After REST, the CPU will start from address 0 and CM-RAM $_0$ is selected.

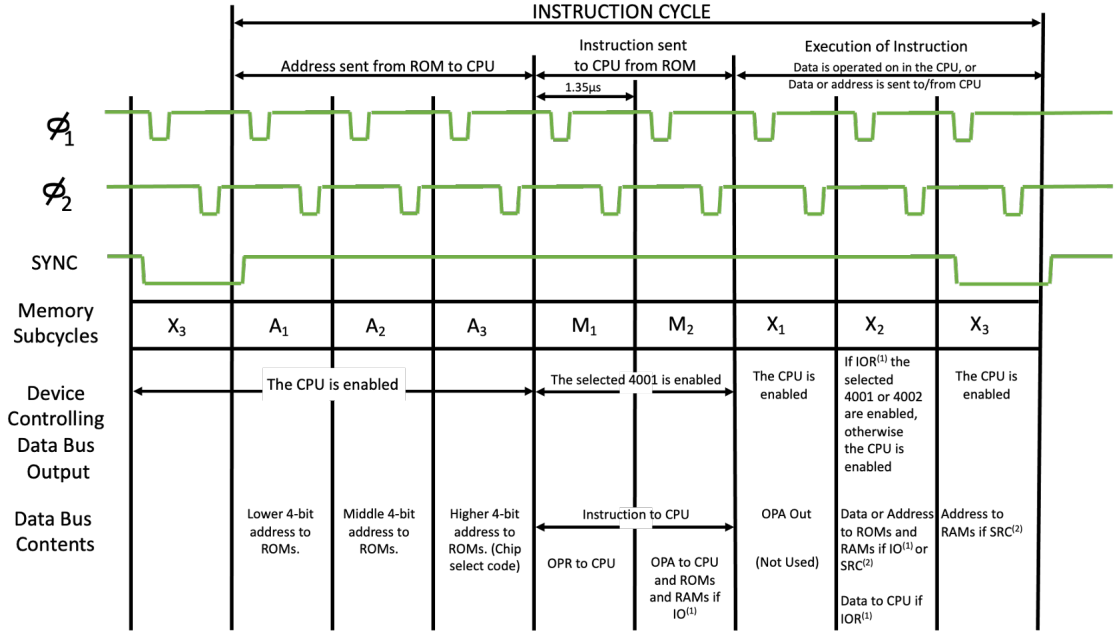
The **MCS-4** can have up to 4K x 8-bit ROM words, 1280 x 4-bit RAM characters and 128 I/O lines, without requiring any interface logic. By adding a few extra gates, the **MCS-4** can have up to 48 RAM and ROM packages in any combination and 192 I/O lines.

The 4001, 4002, and 4004 are interconnected by a 4-line data bus (D_0, D_1, D_2, D_3) used for all information flow between the chips except for the control signals sent by the CPU on 6 additional lines. The interconnection of the MCS-4 system is shown below. Note that an expanded configuration is shown. The minimum system configuration consists of one CPU (4004), and one ROM (4001). The timing diagram below shows the activity on the data bus during each clock period, and how a basic instruction rate is subdivided.

Each data bus output buffer has 3 possible states - “1”, “0”, and “floating”. At any given time, only one output buffer is allowed to drive a data line, therefore, all the other buffers must be in a floating condition. However, more than one input buffer per data line can receive data at the same time.

The **MCS-4** has a very powerful *instruction set* that allows both binary and decimal arithmetic. It allows conditional branching, jump to subroutine and provides for the efficient use of ROM look up tables by indirect fetching. Typically, 2 8-bit numbers can be added in 850 μ secs.





(1) IO instructions control the flow of information between the accumulator in the CPU, I/O lines in ROMs and RAMs and RAM storage. **IOR** stands for I/O Read. In this case, the CPU will receive data from RAM storage locations or I/O input lines of 4001

(2) The SRC instruction designates the chip number and address for a following I/O instruction.

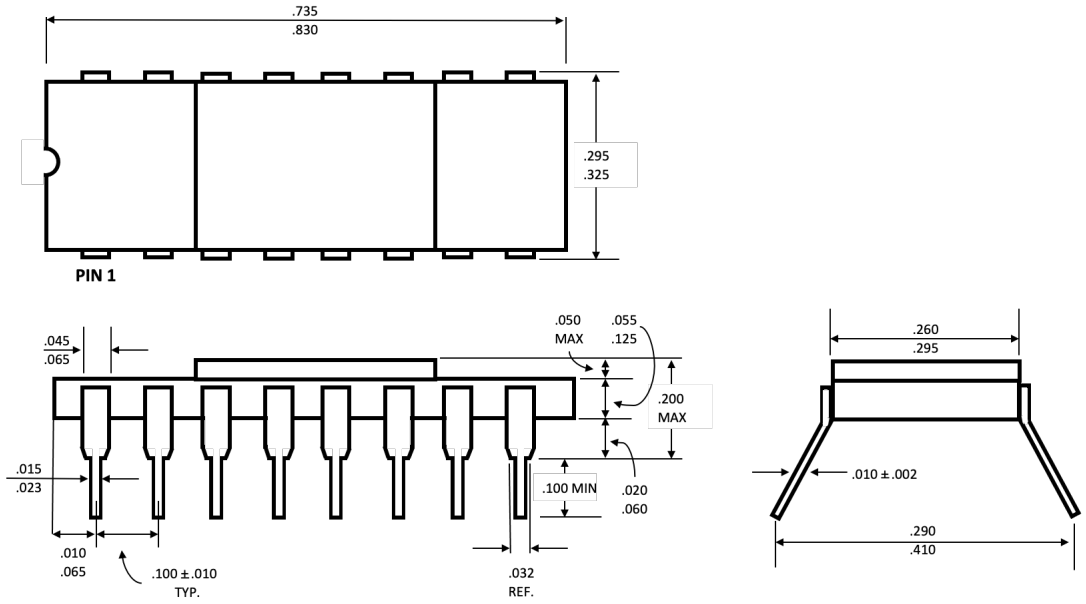
CHAPTER

SEVEN

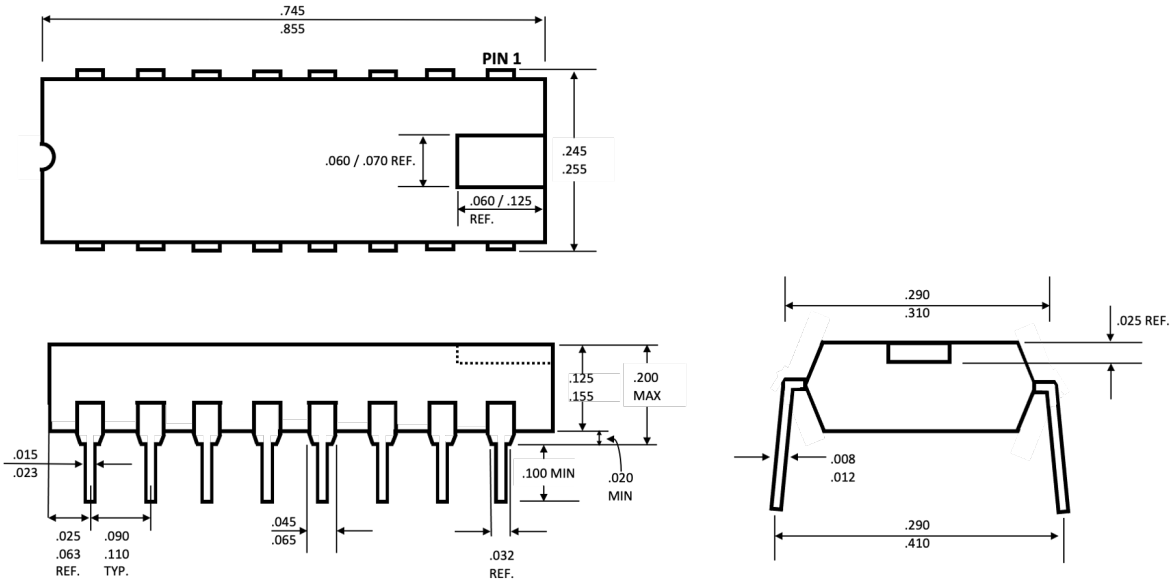
CHIP PACKAGING AND CHARACTERISTICS

Each of the chips in the MCS-4 series have the same packaging dimensions depending on the construction:

Ceramic Packaging



Plastic Packaging



CHAPTER

EIGHT

MCS-4 CHIPSET HARDWARE CHARACTERISTICS

8.1 4001 Hardware Characteristics

Absolute Maximum Ratings

Ambient Temperature Under Bias	0 ° C to +70 ° C
Storage Temperature	-55 ° C to +125 ° C
Input Voltage and Supply Voltage with respect to V_{SS}	+0.5 to -20 V
Power Dissipation	1.0 W

Note that stresses above those listed in “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

D.C. and Operating Characteristics

T_A = 0^o C to 70^o C

V_{SS} - D_{DD} = 15V $\pm$ 5%

$t_{\phi PW}$ = $t_{\phi D1}$ = 400nsec

logic “0” is defined as the more positive voltage (V_{IH}, V_{OH})

logic “1” is defined as the more negative voltage (V_{IL}, V_{OL}); unless otherwise specified.

SUPPLY Current

Symbol	Parameter	Min	Limit Typica
I_{DD}	Average Supply Current		15
	Input Characteristics		
I_{LI}	Input Leakage Current		
V_{IH}	Input High Voltage (except clocks)	V_{SS} -1.5	
V_{IL}	Input Low Voltage (except clocks)	V_{DD}	
V_{IHC}	Input High Voltage Clocks	V_{SS} -1.5	

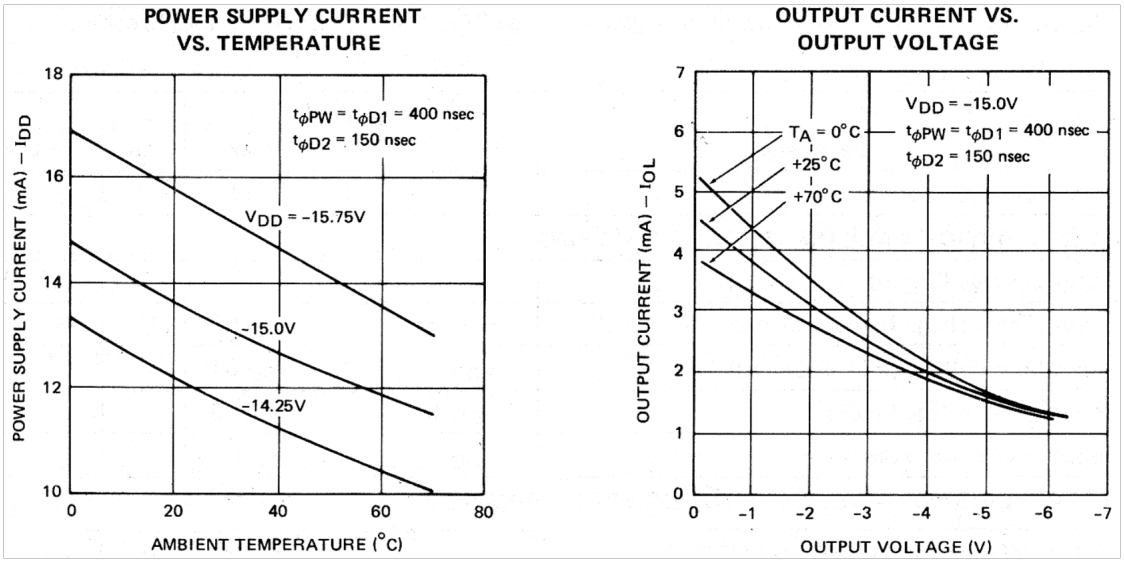
Table 1 – continued from previous page

Symbol	Parameter	Min	Limit Typica
V_{ILC}	Input Low Voltage Clocks	V_{DD}	
Output Characteristics - All outputs except I/O Pins			
I_{LO}	Data Bus Output Leakage Current		
V_{OH}	Output High Voltage	V_{SS}-0.5V	V_{SS}
I_{OL}	Data Lines Sinking Current	8	15
V_{OL}	Output Low Voltage, Data Bus, CM, Sync	V_{SS}-12	
R_{OH}	Output Resistance, Data Line 0 Level		150
I/O Input Characteristics			
I_{LI}	Input Leakage Current		
V_{IH}	Input High Voltage	V_{SS} -1.5V |	
I_{IL}	Input Low Voltage, Inverting Input	V_{DD}	
V_{IL}	Input Low Voltage, Non-Inverting Input	V_{DD}	
V_{IL}	CL Low Voltage	V_{DD}	
R_I	Input Resistance, if used	10	18
R_1 [1]	Input Resistance, if used	15	25
I/O Output Characteristics			
V_{OH}	Output High Voltage	V_{SS} -1.5V	
R_{OH}	I/O Output “0” Resistance		1.2
I_{OL}	I/O Output “1” Sink current	2.5	5
I_{OL} [2]	I/O Output “1” Sink current	0.8	3
V_{OL}	I/O Output Low Voltage	V_{DD} -12	
Capacitance			
C_ϕ	Clock Capacitance		8
C_{DB}	Data Bus Capacitance		9.5
C_{IN}	Input Capacitance		
C_{OUT}	Output Capacitance		

Note: [1] R_1 is large signal equivalent resistance to $(V_{SS} - 4.85)$ V

[2] For Transistor-transistor logic (TTL) compatibility, use $12k\Omega$ external resistor to V_{DD}

Typical D.C. Characteristics



A.C. Characteristics

T_A = 0^o C to 70^o C

V_{SS} - D_{DD} = 15V $\pm$ 5%

Symbol	Parameter	Min	Limit Typical	Max	Unit	Test Conditions
t_{CY}	Clock Period	1.35		2.0	μsec	
$t_{\phi R}$	Clock Rise Times			50	ns	
$t_{\phi F}$	Clock Fall Times			50	ns	
$t_{\phi PW}$	Clock Width	380		480	ns	
$t_{\phi D1}$	Clock Delay $t_{\phi 1}$ to $t_{\phi 2}$	400		550	ns	
$t_{\phi D2}$	Clock Delay $t_{\phi 2}$ to $t_{\phi 1}$	150			ns	
t_W	Data-In, CM, SYNC Write Time	350	100		ns	
t_H [1,3]	Data-In, CM, SYNC Hold Time	40	20		ns	
t_{OS} [2]	Set Time (Reference)	0			ns	
t_{ACC}	Data-Out Access Time Data Lines SYNC CM-ROM CM-RAM			930 930 930 930	ns	C_{OUT} = 500pF Data Lines 500pF SYNC 160pF CM-ROM 50pF CM-RAM
t_{OH}	Data-Out Hold Time	50	150		ns	C_{OUT} = 20pF
t_{IS}	I/O Input Set-Time	50			ns	
t_{IH}	I/O Input Hold-Time	100			ns	
t_D	I/O Output Delay			1500	ns	C_{OUT} = 100pF
t_D [4]	I/O Output Lines Delay on Clear			1500	ns	C_{OUT} = 100pF

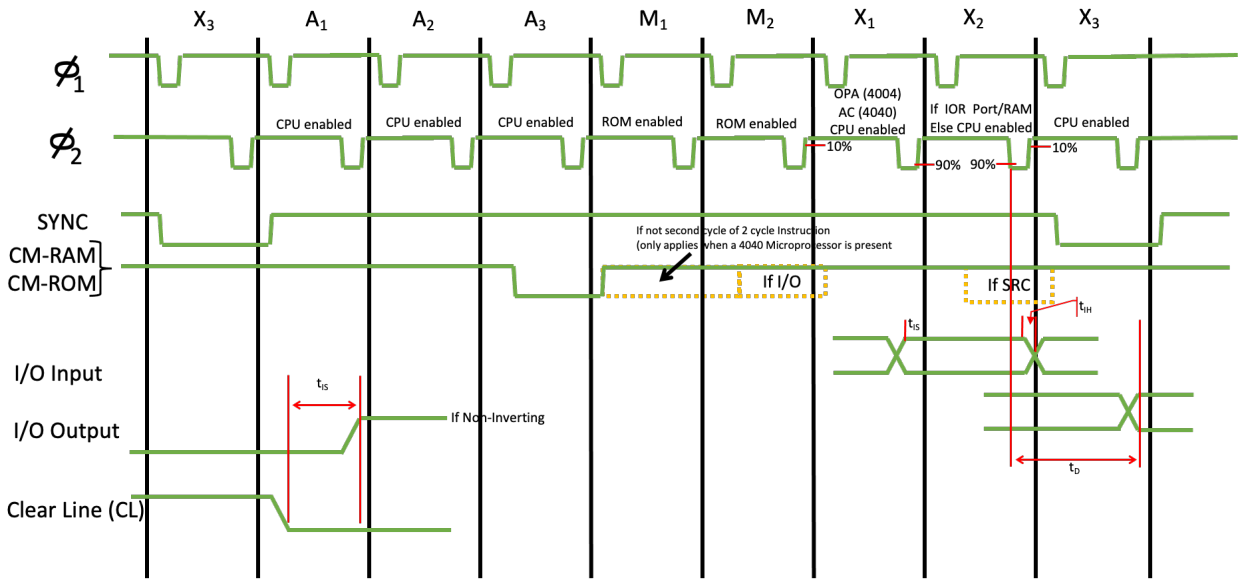
Note: [1] t_H measured with $t_{\phi R}$ = 10nsec

[2] T_{ACC} is Data Bus, SYNC and CM-line output access time referred to the ϕ_2 trailing edge which clocks these lines out. t_{OS} is the same output access time referred to the leading edge of the next ϕ_2 clock pulse.

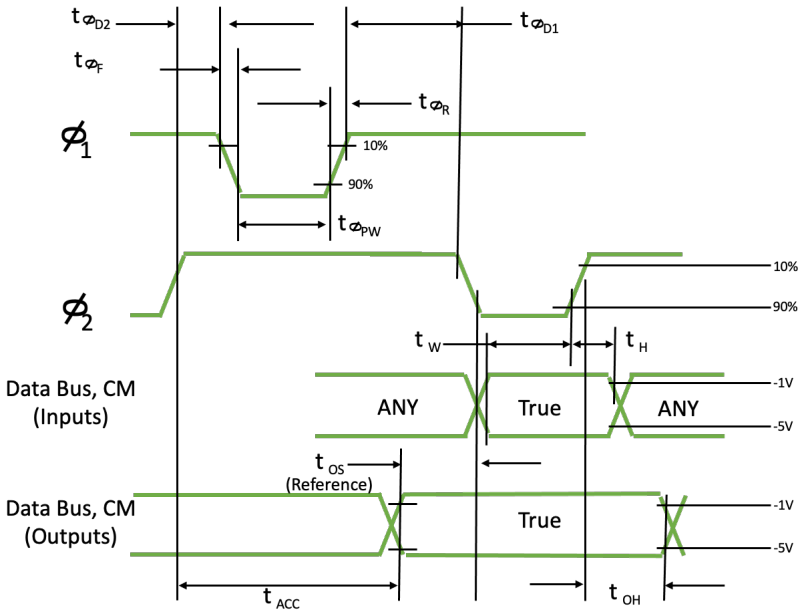
[3] All MCS-40 components which may transmit instruction or data to the 4004 at M_2 and X_2 always enter a float state until the 4004 takes over the data bus at X_1 and X_3 time. Therefore, the T_H requirement is always insured since each component contributes $10\mu A$ of leakage current and 10pF of capacitance, which guarantees that the data bus cannot change faster than $1V/\mu sec$

[4] CL on the 4001 is used to asynchronously clear the input flip-flops associated with the I/O lines.

4001 Timing Diagram



4001 Timing Diagram Detail



8.2 4002 Hardware Characteristics

Absolute Maximum Ratings

Ambient Temperature Under Bias	0 ° C to +70 ° C
Storage Temperature	-55 ° C to +125 ° C
Input Voltage and Supply Voltage with respect to V_{SS}	+0.5 to -20 V
Power Dissipation	1.0 W

Note that stresses above those listed in "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

D.C. and Operating Characteristics

T_A = 0^o C to 70^o C

V_{SS} - D_{DD} = 15V $\pm$ 5%

$t_{\phi PW}$ = $t_{\phi D1}$ = 400nsec

logic "0" is defined as the more positive voltage (V_{IH}, V_{OH})

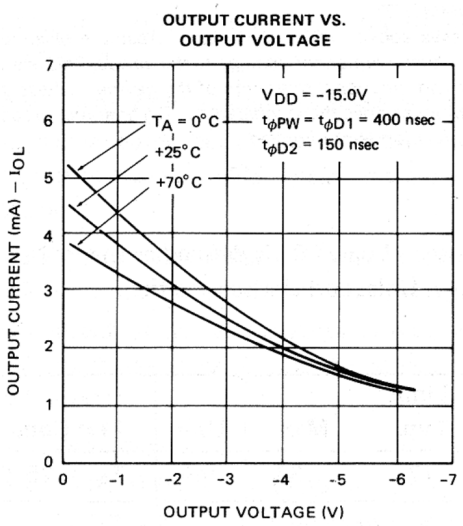
logic "1" is defined as the more negative voltage (V_{IL}, V_{OL}); unless otherwise specified.

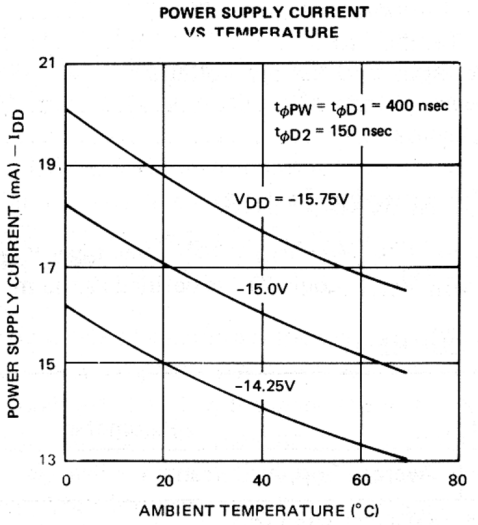
SUPPLY Current

Symbol	Parameter	Min	Limit Typical	Max	Unit	Test Conditions
I_{DD}	Average Supply Current		17	33	mA	T_A = 25^o C
Input Characteristics						
I_{LI}	Input Leakage Current			10	μA	V_{IL} - V_{DD}
V_{IH}	Input High Voltage (except clocks)	V_{SS} -1.5		V_{SS} +0.3	V	
V_{IL}	Input Low Voltage (except clocks)	V_{DD}		V_{SS} -5.5	V	
V_{IHC}	Input High Voltage Clocks	V_{SS} -1.5		V_{SS} +0.3	V	
V_{ILC}	Input Low Voltage Clocks	V_{DD}		V_{SS} -13.4	V	
Output Characteristics - All outputs except I/O Pins						
I_{LO}	Data Bus Output Leakage Current			10	μA	V_{OUT} = -12V
V_{OH}	Output High Voltage	V_{SS}- 0.5V	V_{SS}		V	Capacitive Load
I_{OL}	Data Lines Sinking Current	8	15		mA	V_{OUT} = V_{SS}
V_{OL}	Output Low Voltage, Data Bus, CM, Sync	V_{SS}-12		V_{SS}-6.5	V	I_{OL} = 0.5mA
R_{OH}	Output Resistance, Data Line 0 Level		150	250	Ω	V_{OUT} = V_{SS} - 0.5V
I/O Output Characteristics						
V_{OH}	Output High Voltage	V_{SS} -1.5V			V	I_{OUT} = 0
R_{OH}	I/O Output “0” Resistance		1.2	2	$k\Omega$	V_{OUT} - V_{SS} - 0.5V
I_{OL}	I/O Output “1” Sink current	2.5	5		μA	V_{OUT} - V_{SS} - 0.5V
I_{OL} [1]	I/O Output “1” Sink current	0.8	3		μA	V_{OUT} - V_{SS} - 4.85V
V_{OL}	I/O Output Low Voltage	V_{DD} -12		V_{SS} -6.5	V	I_{OUT} = 50μA
Capacitance						
C_ϕ	Clock Capacitance		8	15	pF	V_{IN} - V_{SS}
C_{DB}	Data Bus Capacitance		7	10	pF	V_{IN} - V_{SS}
C_{IN}	Input Capacitance			10	pF	V_{IN} - V_{SS}
C_{OUT}	Output Capacitance			10	pF	V_{IN} - V_{SS}

Note: [1] For Transistor-transistor logic (TTL) compatibility, use $12k\Omega$ external resistor to V_{DD}

Typical D.C. Characteristics





A.C. Characteristics

T_A = 0^o C to 70^o C

V_{SS} - D_{DD} = 15V $\pm$ 5%

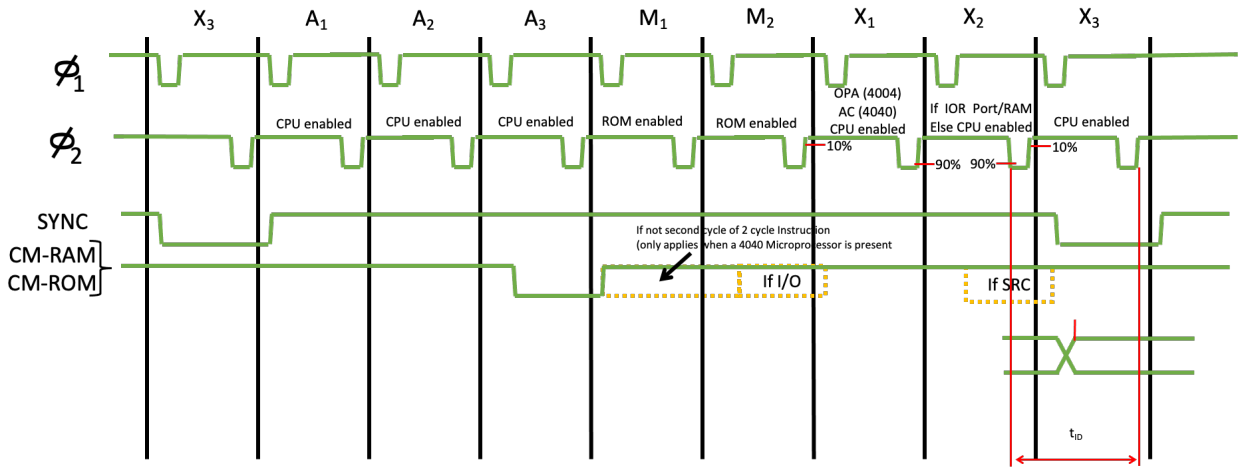
Sym-bol	Parameter	Min	Limit Typi-cal	Max	Unit	Test Conditions
t_{CY}	Clock Period	1.35		2.0	μsec	
$t_{\phi R}$	Clock Rise Times			50	ns	
$t_{\phi F}$	Clock Fall Times			50	ns	
$t_{\phi PW}$	Clock Width	380		480	ns	
$t_{\phi D1}$	Clock Delay $t_{\phi 1}$ to $t_{\phi 2}$	400		550	ns	
$t_{\phi D2}$	Clock Delay $t_{\phi 2}$ to $t_{\phi 1}$	150			ns	
t_W	Data-In, CM, SYNC Write Time	350	100		ns	
t_H [1,3]	Data-In, CM, SYNC Hold Time	40	20		ns	
t_{OS} [2]	Set Time (Reference)	0			ns	
t_{ACC}	Data-Out Access Time Data Lines SYNC CM-ROM CM-RAM			930 930 930 930	ns	C_{OUT} = 500pF Data Lines 500pF SYNC 160pF CM-ROM 50pF CM-RAM
t_{OH}	Data-Out Hold Time	50	150		ns	C_{OUT} = 20pF
t_D	I/O Output Delay			1500	ns	C_{OUT} = 100pF

Note: [1] t_H measured with $t_{\phi R}$ = 10nsec

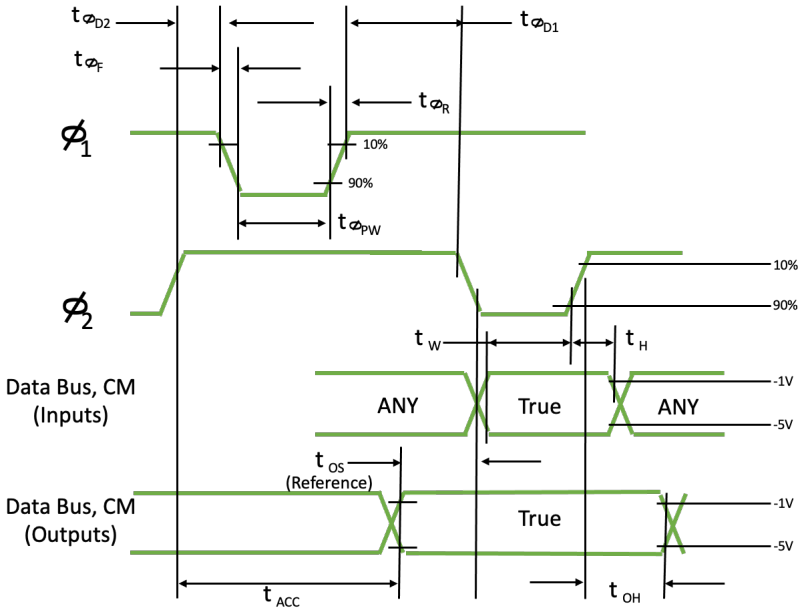
[2] T_{ACC} is Data Bus, SYNC and CM-line output access time referred to the ϕ_2 trailing edge which clocks these lines out. t_{OS} is the same output access time referred to the leading edge of the next ϕ_2 clock pulse.

[3] All MCS-40 components which may transmit instruction or data to the 4004 at M_2 and X_2 always enter a float state until the 4004 takes over the data bus at X_1 and X_3 time. Therefore, the T_H requirement is always insured since each component contributes $10\mu A$ of leakage current and 10pF of capacitance, which guarantees that the data bus cannot change faster than $1V/\mu sec$

4002 Timing Diagram



4002 Timing Diagram Detail



8.3 4003 Hardware Characteristics

Absolute Maximum Ratings

Ambient Temperature Under Bias	0 ° C to +70 ° C
Storage Temperature	-55 ° C to +125 ° C
Input Voltage and Supply Voltage with respect to V_{SS}	+0.5 to -20 V
Power Dissipation	1.0 W

Note that stresses above those listed in “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

D.C. and Operating Characteristics

T_A = 0^o C to 70^o C

V_{SS} - D_{DD} = 15V $\pm$ 5%

$t_{\phi PW}$ = $t_{\phi D1}$ = 400nsec

$t_{\phi D2}$ = 400nsec ; unless otherwise specified.

logic “0” is defined as the more positive voltage (V_{IH}, V_{OH})

logic “1” is defined as the more negative voltage (V_{IL}, V_{OL})

SUPPLY Current

Symbol	Parameter	Min	Limit Typical [1]	Max	Unit	Test Conditions
I_{DD}	Average Supply Current		5.0	8.5	mA	T_{WL} = T_{WH} = $8\mu s$; T_A = 25^o C
Input Characteristics						
I_{LI}	Input Leakage Current			10	μA	V_{IL} - V_{DD}
V_{IH}	Input High Voltage	V_{SS} -1.5		V_{SS} +0.3		
V_{IL}	Input Low Voltage	V_{DD}		V_{SS} -4.2	V	
I/O Output Characteristics						
I_{OH}	Parallel Out Pins Sinking Current, “1” Level	0.6	1.0		μA	I_{OUT} = 0
I_{OL}	Serial Out Pins Sinking Current, “1” Level	1.0	2.0		μA	V_{OUT} - V_{SS} - 0.5V
V_{OL}	I/O Output Low Voltage	V_{SS} -11	V_{SS} -7.5	V_{SS} -6.5	V	I_{OUT} = $50\mu A$
R_{OH}	Parallel Out Pins Resistance, “0” Level		400	750	$k\Omega$	I_{OUT} = 0
R_{OH}	Serial Out Resistance, “0” Level		650	1200	$k\Omega$	V_{OUT} - V_{SS} - 0.5V

Capacitance

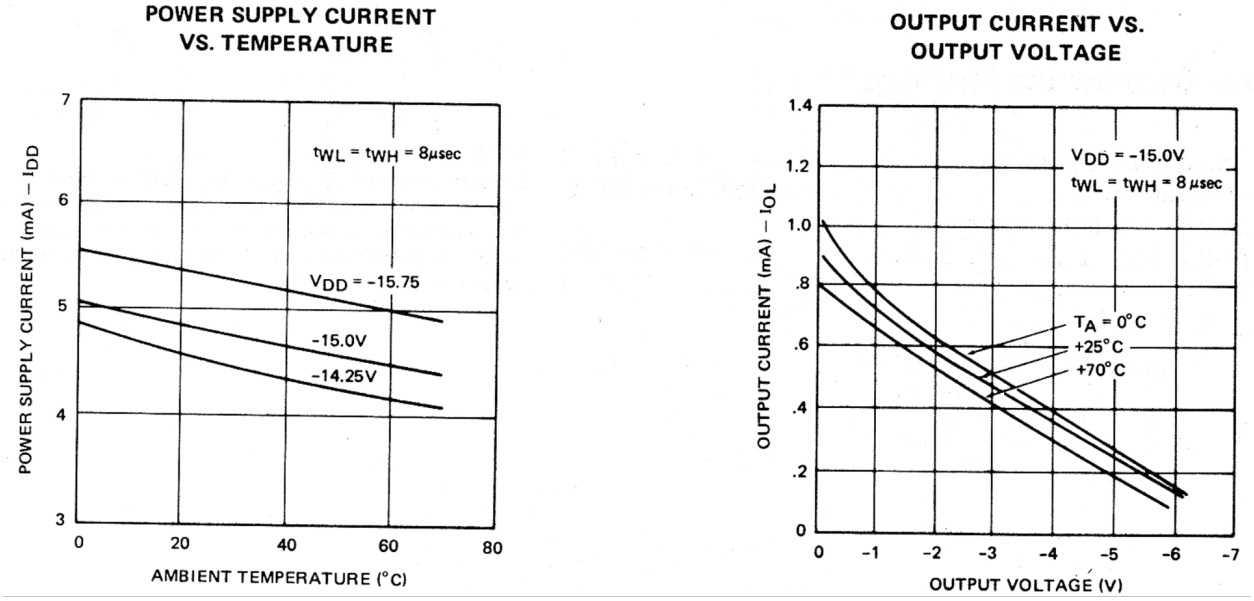
f = 1MHz; V_{IN} = 0V; T_A = 25^oC Unmeasured Pins Grounded

Symbol	Test	Typ.	Max	Unit
C_{IN}	Input Capacitance	5	10	pF

Note: [1] Typical values are to T_A = 25^o C and Nominal Supply Voltages

[2] For Transistor-transistor logic (TTL) compatibility, use $12k\Omega$ external resistor to V_{DD}

Typical D.C. Characteristics



A.C. Characteristics

T_A = 0^o C to 70^o C

V_{DD} = 15V $\pm$ 5%

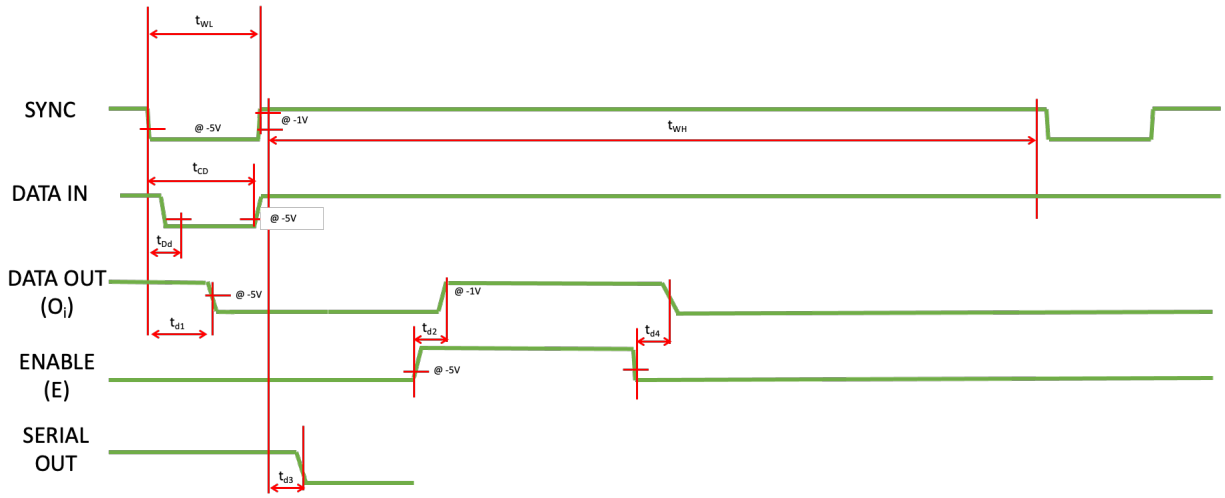
V_{SS} = GND

Symbol	Parameter	Min	Limit Typical	Max	Unit	Test Conditions
t_{WL}	CP Low Width	6		10,000	μsec	
t_{WH} [1]	CP High Width	6			μsec	
t_{CD}	Clock-On to Clock-Off Time	3			μsec	
t_{Dd}[2]	CP to Data Set Delay			250	ns	
t_{d1}	CP to Data Out Delay	250		1750	ns	
t_{d2}	Enable to Data Out Delay			350	ns	C_{OUT} = 20pF
t_{d3}	CP to Serial Out Delay	200		1250	ns	C_{OUT} = 20pF
t_{d4}	Enable to Data Out Delay	40		1.0	μsec	C_{OUT} = 20pF

Note: [1] t_{WH} can be any time greater than $6\mu s$

[2] Data can occur prior to CP

4003 Timing Diagram



8.4 4004 Hardware Characteristics

Absolute Maximum Ratings

Ambient Temperature Under Bias	0 ° C to +70 ° C
	Available with Operating Temp of -40 ° C to +85 ° C
Storage Temperature	-55 ° C to +100 ° C
Input Voltage and Supply Voltage with respect to V_{SS}	+0.5 to -20 V
Power Dissipation	1.0 W

Note that stresses above those listed in “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

D.C. and Operating Characteristics

T_A = 0^o C to 70^o C

V_{SS} - D_{DD} = 15V $\pm$ 5%

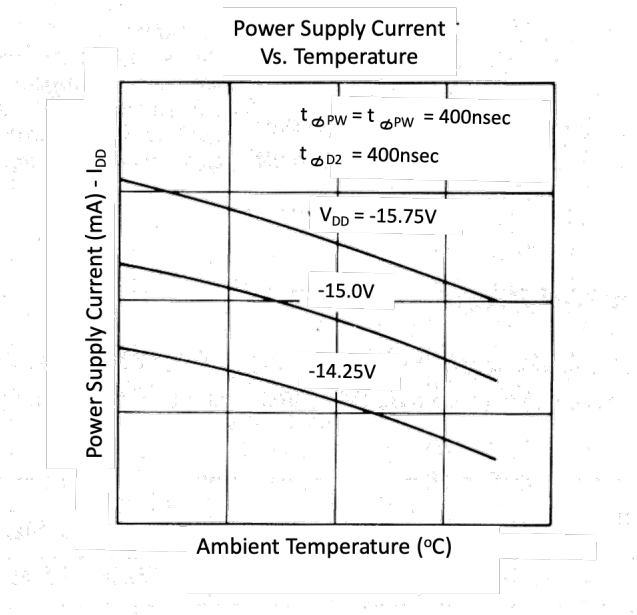
$t_{\phi PW}$ = $t_{\phi D1}$ = 400nsec

logic “0” is defined as the more positive voltage (V_{IH}, V_{OH})

logic “1” is defined as the more negative voltage (V_{IL}, V_{OL}); unless otherwise specified.

Symbol	Parameter	Min	Limit Typical	Max	Unit	Test Conditions
Input Characteristics						
I_{DD}	Average Supply Current		30	40	mA	T_A = 25^o C
Input Characteristics						
I_{LI}	Input Leakage Current			10	μA	V_{IL} - V_{DD}
V_{IH}	Input High Voltage (except clocks)	V_{SS} -1.5		V_{SS} +0.3	V	
V_{IL}	Input Low Voltage (except clocks)	V_{DD}		V_{SS} -5.5	V	
V_{ILO}	Input Low Voltage	V_{DD}		V_{SS} -4.2	V	4004 Test input
V_{IHC}	Input High Voltage Clocks	V_{SS} -1.5		V_{SS} +0.3	V	
V_{ILC}	Input Low Voltage Clocks	V_{DD}		V_{SS} -13.4	V	
Output Characteristics						
I_{LO}	Data Bus Output Leakage Current			10	μA	V_{OUT} = -12V
V_{OH}	Output High Voltage	V_{SS}- 0.5V	V_{SS}		V	Capacitance Load
I_{OL}	Data Lines Sinking Current	8	15		mA	V_{OUT} = V_{SS}
I_{OL}	CM-ROM Sinking Current	6.5	12		mA	V_{OUT} = V_{SS}
I_{OL}	CM-RAM Sinking Current	2.5	6		mA	V_{OUT} = V_{SS}
V_{OL}	Output Low Voltage, Data Bus, CM, Sync	V_{SS}-12		V_{SS}-6.5	V	I_{OL} = 0.5mA
R_{OH}	Output Resistance, Data Line 0 Level		150	250	Ω	V_{OUT} = V_{SS} - 0.5V
R_{OH}	CM-ROM Output Resistance, Data Line 0 Level		320	600	Ω	V_{OUT} = V_{SS} - 0.5V
R_{OH}	CM-RAM Output Resistance, Data Line 0 Level		1.1	1.8	$k\Omega$	V_{OUT} = V_{SS} - 0.5V
Capacitance						
C_{ϕ}	Clock Capacitance		14	20	pF	V_{IN} - V_{SS}
C_{DB}	Data Bus Capacitance		7	10	pF	V_{IN} - V_{SS}
C_{IN}	Input Capacitance			10	pF	V_{IN} - V_{SS}
C_{OUT}	Output Capacitance			10	pF	V_{IN} - V_{SS}

Typical D.C. Characteristics



A.C. Characteristics

T_A = 0^o C to 70^o C

V_{SS} - D_{DD} = 15V $\pm$ 5%

Symbol	Parameter	Min	Limit Typical	Max	Unit	Test Conditions
t_{CY}	Clock Period	1.35		2.0	μsec	
$t_{\phi R}$	Clock Rise Times			50	ns	
$t_{\phi F}$	Clock Fall Times			50	ns	
$t_{\phi PW}$	Clock Width	380		480	ns	
$t_{\phi D1}$	Clock Delay $t_{\phi 1}$ to $t_{\phi 2}$	400		550	ns	
$t_{\phi D2}$	Clock Delay $t_{\phi 2}$ to $t_{\phi 1}$	150			ns	
t_W	Data-In, CM, SYNC Write Time	350	100		ns	
t_H [1,3]	Data-In, CM, SYNC Hold Time	40	20		ns	
t_H [3]	Data Bus Hold Time in M_2 - X_1 and X_2 - X_3 transition		40	20	ns	
t_{OS} [2]	Set Time (Reference)	0			ns	
t_{ACC}	Data-Out Access Time Data Lines Data Lines SYNC CM-ROM CM-RAM			930 700 930 930 930	ns	C_{OUT} = 500pF Data Lines 200pF Data Lines 500pF SYNC 160pF CM-ROM 50pF CM-RAM
t_{OH}	Data-Out Hold Time	50	150		ns	C_{OUT} = 50pF

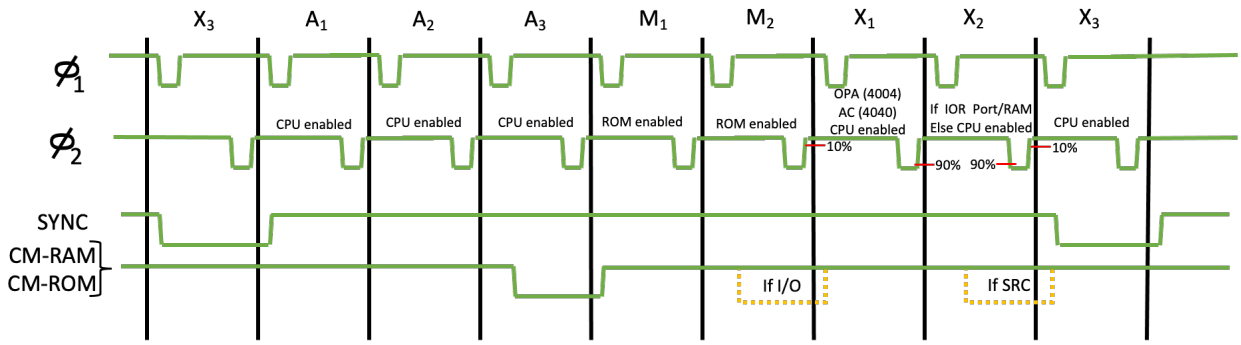
Note: [1] t_H measured with $t_{\phi R}$ = 10nsec

[2] T_{ACC} is Data Bus, SYNC and CM-line output access time referred to the ϕ_2 trailing edge which clocks these lines out. t_{OS} is the same output access time referred to the leading edge of the next ϕ_2 clock pulse.

[3] All MCS-40 components which may transmit instruction or data to the 4004 at M_2 and X_2 always enter a float state until the 4004 takes over the data bus at X_1 - X_3 time. Therefore, the T_H requirement is always insured since each component contributes $10\mu A$ of leakage current and 10pF of capacitance, which guarantees that the data bus cannot change faster than $1V/\mu sec$

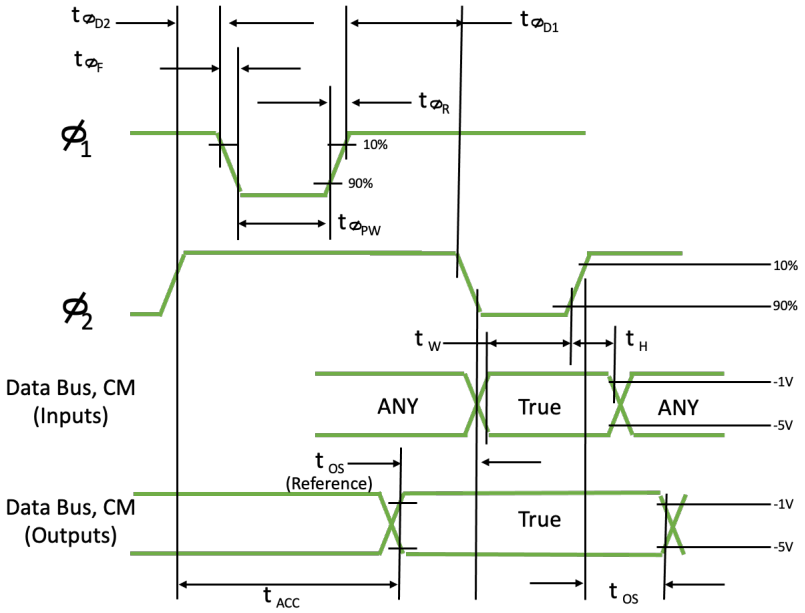
[4] $C_{DATABUS}$ = 200pF if 4008 and 4009 or 4298 is used.

4004 Timing Diagram



1 **IOR** stands for I/O Read. In this case, the CPU will receive data from RAM storage locations or I/O input lines of 4001.
2 The successor to the 4004, the intel 4040, has a superset of the 4004's instruction set, and can also respond to interrupts.

4004 Timing Diagram Detail



The multiple chips in the MCS-4 chipset have their own unique hardware characteristics:

- *4001*
- *4002*
- *4003*
- *4004*

CHAPTER

NINE

OVERVIEW OF PYNTEL4004

Pyntel4004 consists of two components:

- an Intel 4004 assembler/disassembler
- an Intel 4004 emulator to run assembled code

Hardware emulation

The Intel 4004 emulator mimics the hardware of an original Intel 4004 processor and its' support chips through software.

Each instruction in Pyntel4004 acts on a virtual processor in the same way as the original hardware implementations of the instructions would act upon the real hardware.

The intention is to test the assembled code on a real Intel 4004 chip to verify this..

Usage

In order to use these tools, a source file must first be prepared in i4004 assembly language.

```
/ Example program
    org    ram
    fim    2  254
    end
```

This file should then be assembled into 4004 machine code.

In order to do this, the CLI package should be installed:

pip install pyntel4004-cli

The full instructions for Pyntel4004-CLI should be read, however, a basic summary is below: ### Basic Usage.

4004 <*command*> <*options*> <*arguments*>

<*command*> - *asm* Assemble the input file - *dis* Disassemble the input file - *exe* Execute the object file

<*options*> - **-h**, **–help**: Show help. - **-v**, **–version**: Show the version and exit.

asm options.

- **-i**, **–input** <*input file*>: assembly language source file.
- **-o**, **–output** <*output file*>: object code output file.
- **-e**, **–exec**: execute the assembled program if successful assembly.

- **-t, –type <*extension*>: Type of output required. (multiple output types can be specified)**
  - *bin* will deliver a binary file of machine code
  - *obj* will deliver an object module which can be loaded back into the disassembler for debugging
  - *h* will deliver a c-style header file that can be used in a RetroShield Arduino to run the code on a real 4004
  - *ALL* will deliver all of the above<details>New in 0.0.1-alpha.2<summary>Changelog</summary></details>
- **-c**, **–config** <*config file*>: use the specified config file<details>New in 0.0.1-alpha.2<summary>Changelog</summary></details>
- **-q**, **–quiet**: Quiet mode on *
- **-m**, **–monitor**: Start monitor*
- **-h**, **–help**: Show help.

*Mutually exclusive parameters

dis options.

- **-o**, **–object** <*object file*>: object code or binary input file.
- **-l**, **–labels**: show the label table (only available in .OBJ files)<details>New in 0.0.1-alpha.2<summary>Changelog</summary></details>
- **-c**, **–config** <*config file*>: use the specified config file<details>New in 0.0.1-alpha.2<summary>Changelog</summary></details>
- **-b**, **–byte**: number of bytes to disassemble (between 1 and 4096).
- **-h**, **–help**: Show help.

  It is the user's responsibility to understand that if a byte count causes the disassembler to end up midway through a 2-byte instruction, that last instruction will not be disassembled correctly.

exe options.

- **-o**, **–object** <*object file*>: object code or binary input file.
- **-c**, **–config** <*config file*>: use the specified config file<details>New in 0.0.1-alpha.2<summary>Changelog</summary></details>
- **-q**, **–quiet**: Quiet mode on
- **-h**, **–help**: Show help.

9.1 Error Messages

Error messages are displayed when there are issues with either the supplied command, or issues with the source code itself. The errors are raised as exceptions, with an exception type together with an information message

9.2 Errors

Command | Exception | Options | Message |

|---|—|----|–| | asm | BadParameter | **|Invalid Parameter Combination: --quiet and --monitor cannot be used together | | asm | BadOptionUsage | --type |Invalid output type specified | | asm | BadOptionUsage | --type |Cannot specify 'ALL' with any others| |dis|** BadOptionUsage| –inst | Instructions should be between 1 and 4096 |

Special Error Message

Exception | Message |

|----------|———| | CoreNotInstalled| Pyntel4004 core is not installed - use pip install Pyntel4004

9.3 Configuration Files

<details>New in 0.0.1-alpha.2<summary>Changelog</summary></details>
 Pyntel4004-cli configuration files are specified using the [TOML](http://toml.io/) notation. This is a notation which favours humans over machines, so it is easy to understand and write the configuration you want.

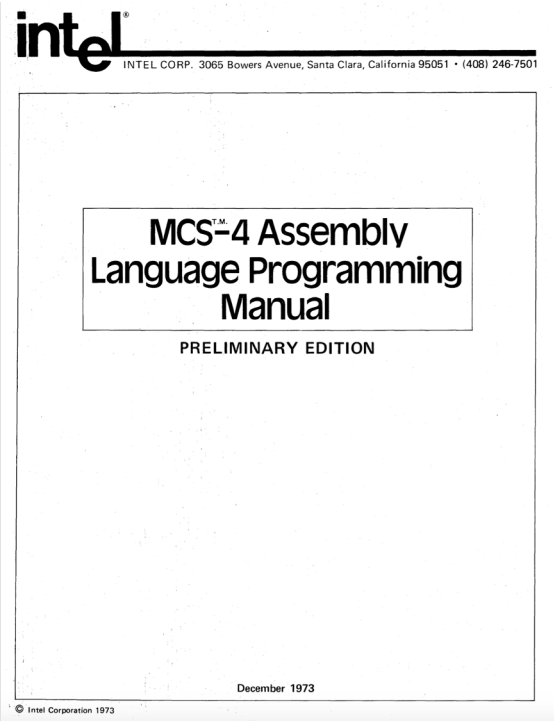
 Example Configuration File - example2.toml

``` # Configuration for Pyntel4004-cli.

title = “Configuration file for example2.asm”

[asm] input = “example2.asm” output = “example2” type = [“BIN”, “H”] exec = true monitor = true quiet = true

[dis] object = “examples/example2.obj” inst = 6 labels = true

[exe] object = “examples/example2.obj” quiet = true ```

The configuration file has 4 sections:

This MUST be first

i) The title - simply a description of what the configuration file is for. Note that any comments (lines starting with a `#` can be added anywhere for readability).

(in no particular order)

ii) `[asm]` section containing directives for the assembly of a specific program source file

iii) `[dis]` section containing directives for the disassembly of a specific object module

iv) `[exe]` section containing directives for the execution of a specific object module

The valid configuration tokens are shown in the example above - they mirror the options that can be specified on the command line.

ANY of the configuration tokens can be overriden simply by specifying them on the command line.
```

CHAPTER

TEN

MCS-4 ASSEMBLY LANGUAGE PROGRAMMING MANUAL



10.1 Acknowledgements

Datasheets provided by chipdb.org

bitsavers.org provided: MCS-4 Assembly Language Manual MCS-4 Users Manual Intellec-4 manual

Second Source information provided by wikichip.org

10.2 Glossary of Terms

Term	Definition
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 was usually on paper tape, and is produced by assembling a source program, however the Pyntel4004 Assembler can produce object code to be loaded into an emulator or directly on to a board simulating an MCS-4. Instructions are represented by binary machine code in an object program.
Program	A sequence of instructions which are taken as a group to 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.
0xnn	nnn represents a number in hexadecimal format.

Note

All numbers in this document are assumed to be decimal unless otherwise specified.

Note

A representation of a byte in memory. Bits which are fixed are indicated by 0 or 1; bits vvhich 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.

10.3 Introduction

This document 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 32768 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.

10.4 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 *here*

(5) Input/Output which is the interface between a program and the outside world.

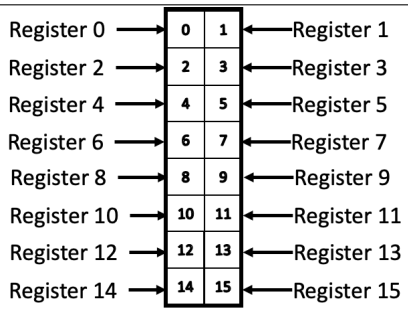
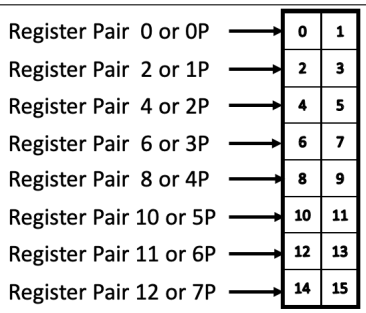
10.5 Working (Index) Registers

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 Registers	Register Pairs
	

10.6 Accumulator

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

10.7 Memories

10.7.1 Program Random Access Memory (PRAM)

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

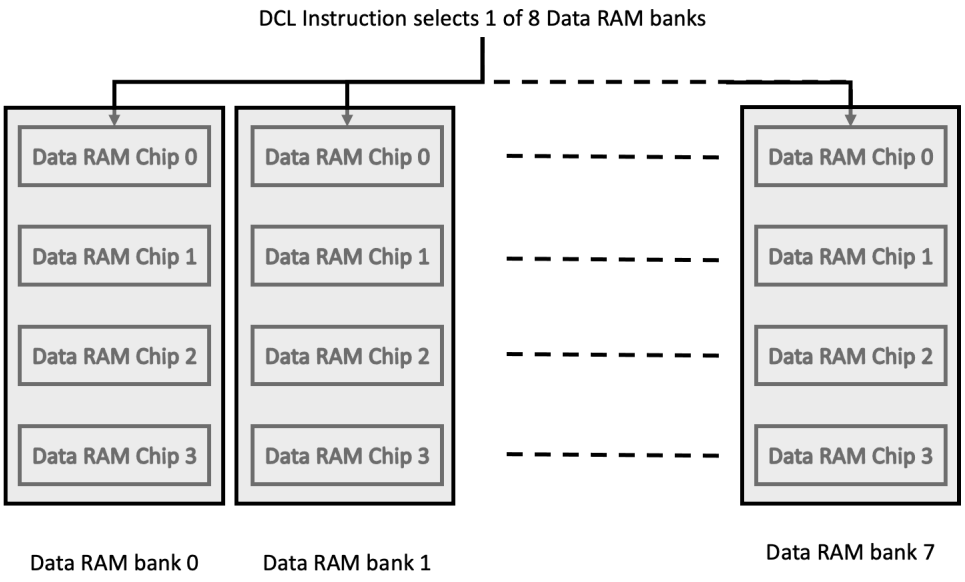
Unlike ROM, however, program RAM locations can be altered by program instructions.

10.7.2 Data Random Access Memory (RAM)

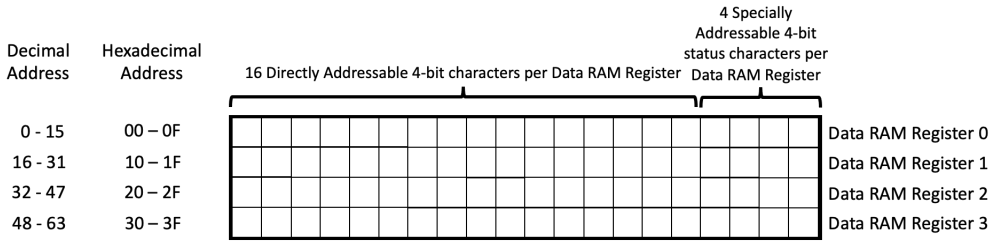
As its name implies, **data random access memory** (DATA RAM) is used for the temporary storage of data by programs.

The RAM is laid out as shown below:

Data RAM bank organisation



RAM chip organisation



In order to address a 4-bit character of DATA RAM, the programmer first uses a "*DCL*" instruction to choose one of a maximum of eight DATA RAM BANKS.

An eight bit address is then sent via an "*SRC*" instruction 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 *here* and *here*

10.7.3 Read-Only Memory (ROM)

Read-only memory (ROM) is used for storing program instructions and constant data which is never changed by the program.

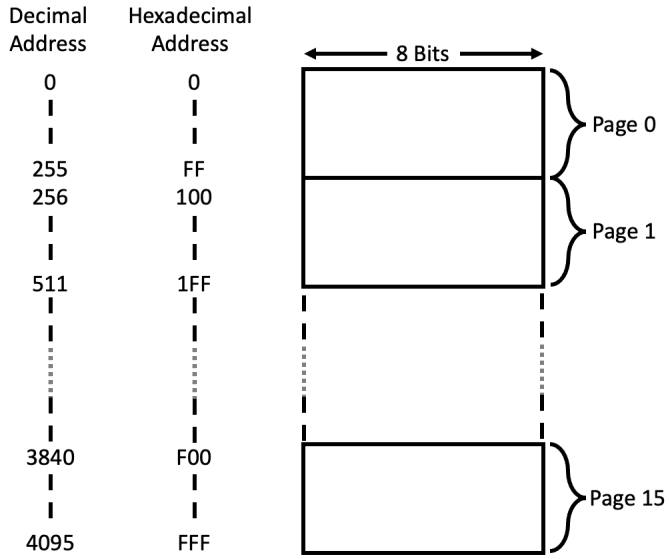
This is because the program can read locations in ROM, but can never alter (write) ROM locations.

ROM may be visualized as below; 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, locations 256 through 511 comprise page 1 and so on.



Note: Instruction Positioning

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

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

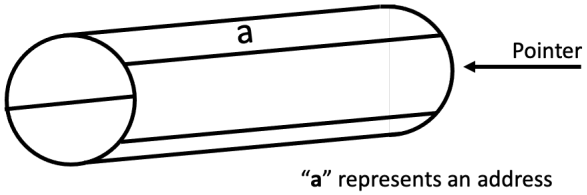
These are :

- *ROM (Read Only Memory)*
- *PRAM (Program Random Access Memory)*
- *RAM (Data Random Access Memory)*

10.8 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:



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

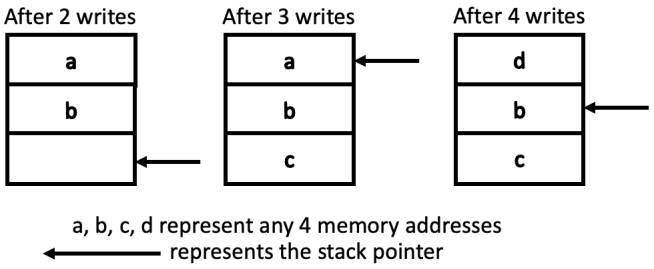
10.8.1 Writing An Address To The Stack

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 below:



Note:

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

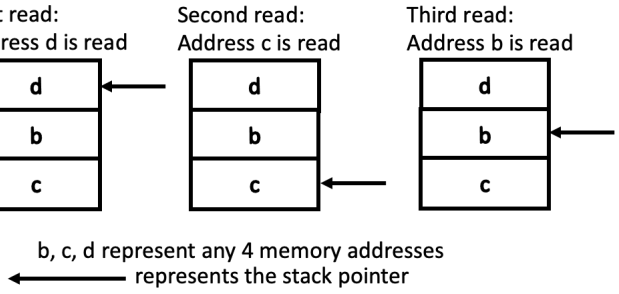
10.8.2 Reading An Address From The Stack

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 below:



The stack is used by programs as described *here*.

10.9 Input and 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 *here*. 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.

10.10 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 in *groups* or *individually*.

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 ensure 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 the diagram below.

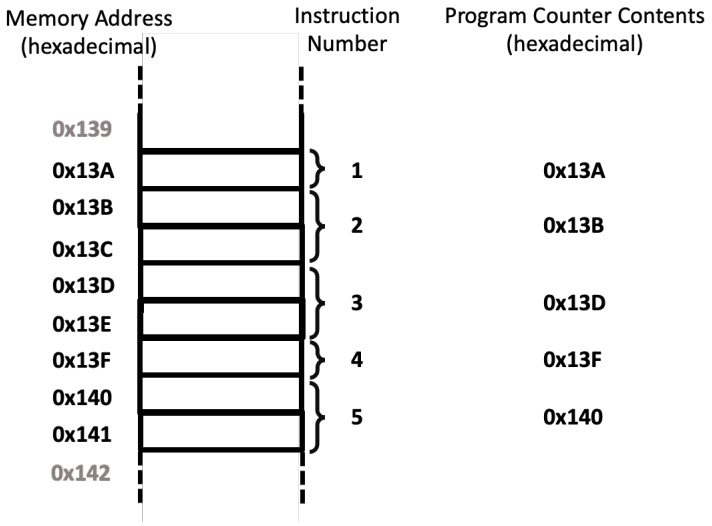
In order to avoid errors, the programmer must be sure that a byte of constant data does not follow an instruction when another instruction is expected. Referring to the diagram, an instruction is expected in location 0x13F, since instruction 4 is to be executed after instruction 3.

If location 0x13F 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) causes 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 the diagram below, suppose instruction 2 specifies a jump to location 0x140 and instructions 3 and 4 were replaced by data. Then following execution of instruction 2, the program counter would be set to 0x140 and the program would execute correctly. But if, in error, instruction 2 were to specify a jump to 0x13E, an error would result since this location now holds data. Even if instructions 3 and 4 were not altered, a jump to location 0x13E would cause an error, since this is not the first byte of the instruction.

Upon reading *the instruction summary*, 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

Note: Programs usually exist in ROM, and therefore cannot be altered in this manner.



10.11 Memory Addressing

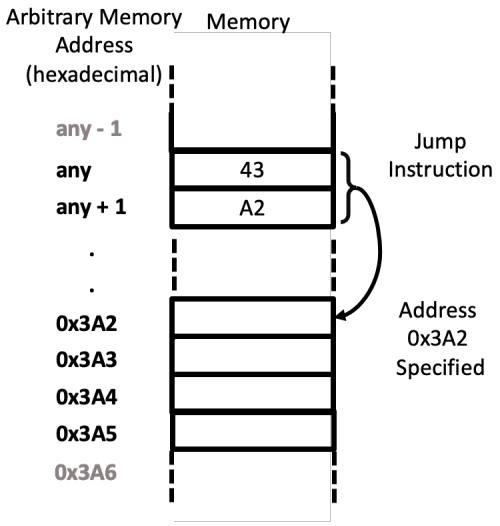
10.11.1 Direct Addressing

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 3A2
```

This instruction is represented by 4 hexadecimal digits in RQM 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:



10.11.2 Same Page Addressing

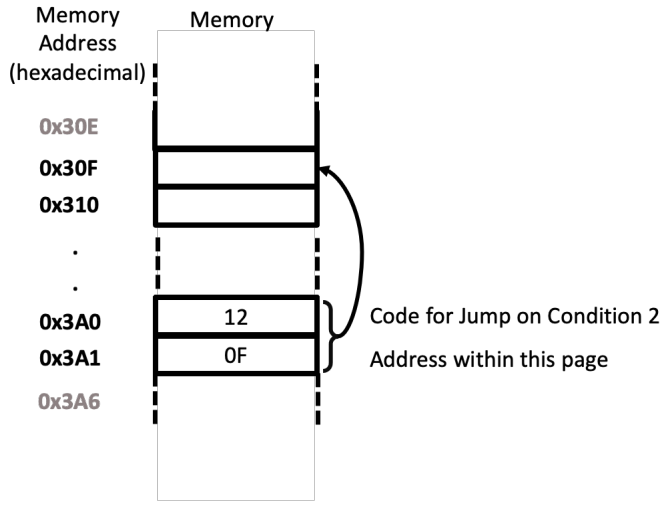
Some instructions supply two hexadecimal digits which replace the lowest 8 bits of the program counter, addressing a ROM or program RAM location on the same page as the instruction being executed.

Note: (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 0F of this page
```

This instruction would appear in memory as follows:



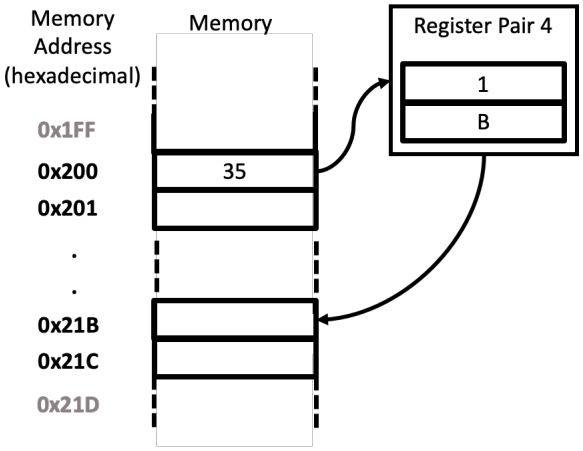
The identical instruction encoding 0x120F, if located at location 0x501, would cause a jump to memory address 0x50F.

10.11.3 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
```

This instruction would appear in memory as follows:



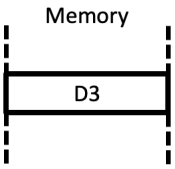
The 3 indicates a "jump indirect" instruction; the 5 indicates that the address indicated on this page is held in register pair 4. If register pair 4 had held the hex numbers 3 and C, a jump to location 0x23C would have occurred.

10.11.4 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 appear in memory as follows:



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

10.11.5 Program RAM Addressing

When a program stores an 8 bit value into a program RAM location, a special sequence of instructions using the *WPM* instruction.

10.11.6 Data RAM Addressing

To address a location in DATA RAM, the *DCL* and *SRC* instructions must be used as described *here*.

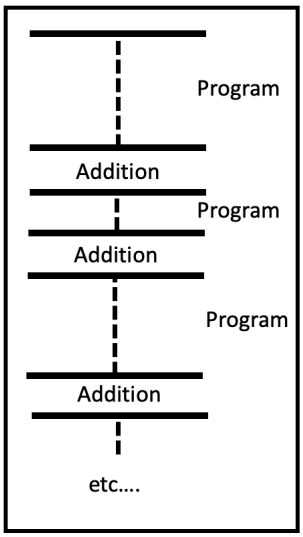
When the DCL has chosen a specific DATA RAM bank, the address of the specific character is held in a register pair accessed by the SRC instruction.

10.11.7 Subroutines and use of the Stack for Addressing

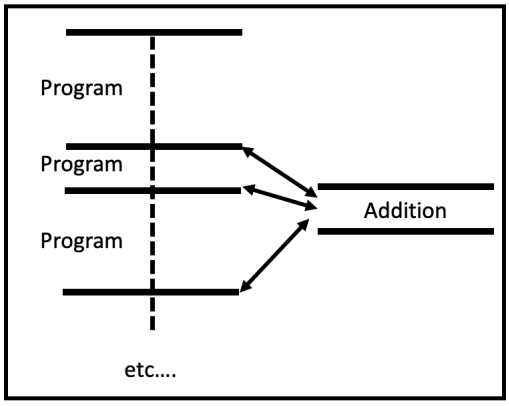
Before understanding the purpose or effectiveness of the stack, it is necessary to understand the concept of a subroutine.

Consider a frequently used operation such as addition.

The 4004 provides instructions to add one character of data to another, but what if there was a requirement 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:

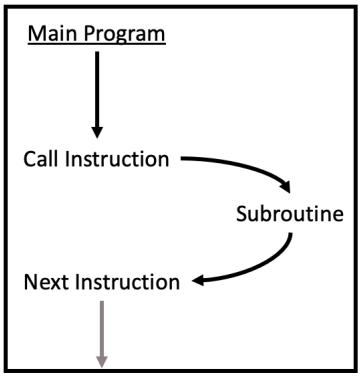


A more efficient means of accessing the addition routine would be to store it once, and find a way of accessing it when needed:



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.

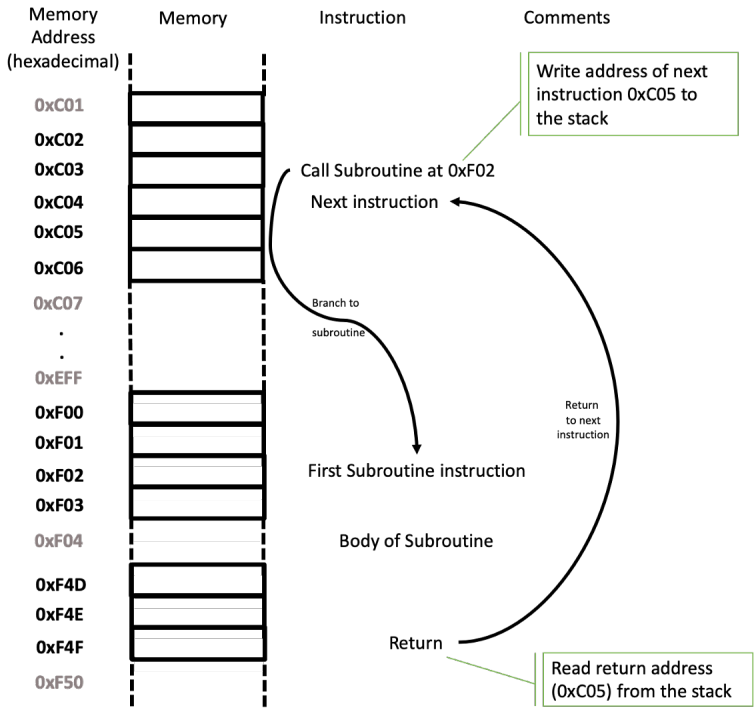
When a subroutine is executed, the sequence of events may be depicted as follows:



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 below:



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.

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 below

- *Direct Addressing*
- *Same Page Addressing*
- *Indirect Addressing*
- *Immediate Addressing*
- *Program RAM Addressing*
- *Data RAM Addressing*
- *Subroutines and the use of the stack for Addressing*

10.12 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* 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.

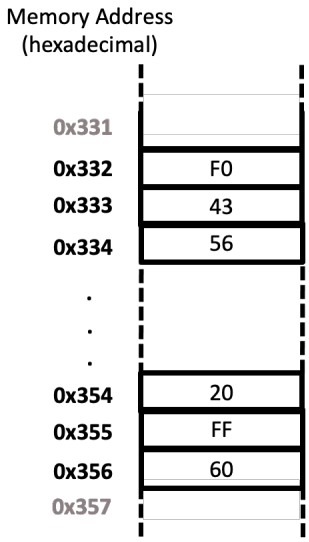
Certain data operations can cause an overflow out of the high-order 3-bit. For example, addition of two hexadecimal digits can give rise to an answer that does not fit in one digit:

```
            3 2 1 0     Bit number

    A       1 0 1 0
+   7       0 1 1 1
            -------
    1       0 0 0 1
carry
```

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.

10.13 The 4004 Instruction Set

10.13.1 How Assembly Language is Used



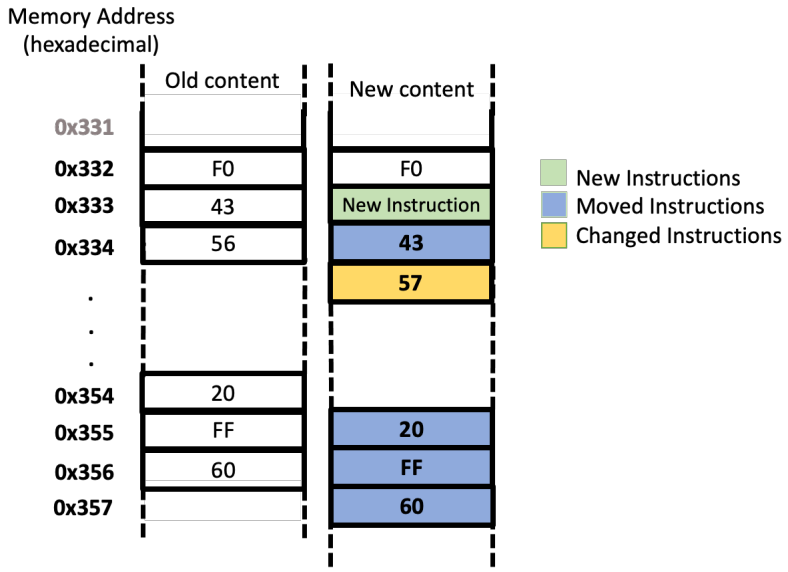
Upon examining the contents of read-only memory or program random-access memory, a program would appear as a sequence of hexaaecimal digits which are interpreted by the machine as instruction codes, addresses, or constant data. It is possible to write a program as a sequence of digits (just as they appear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

For example, in the example to the right, several instructions reference memory to address another instructions.

The example program works as follows:

- Byte 0x332 specifies that the accumulator and carry bit are to be cleared.
- Bytes 0x333 and 0x334 specify that program execution is to continue at location 0x356.
- Byte 0x356 specifies that register 0 is to be incremented.

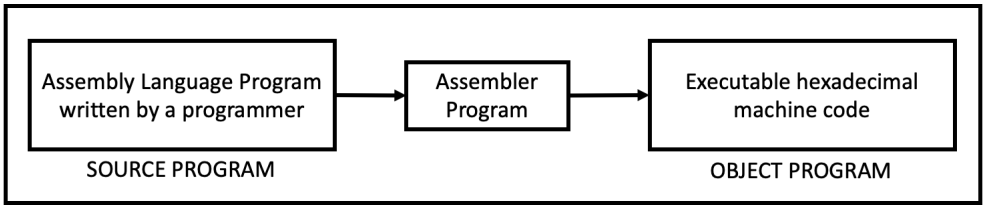
Now suppose that an error discovered in the program logic necessitates placing a new instruction after byte 0x332. Program code would have to change as follows:



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.

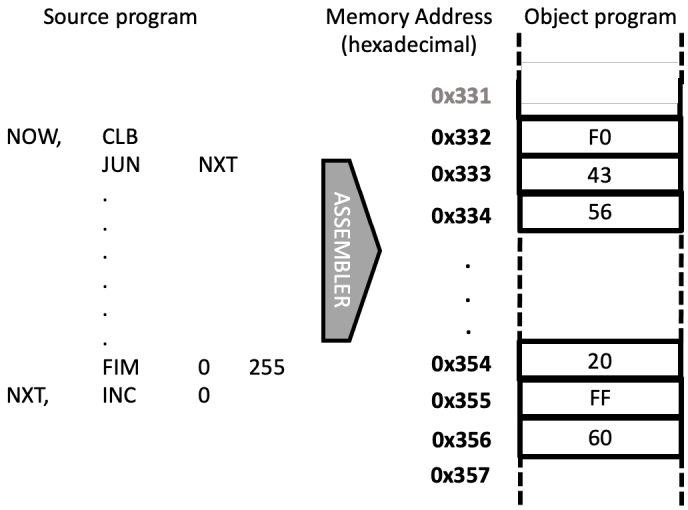
Writing programs in assembly language is the first and most significant step towards economical programming; it provides a readable notation for instructions, and separates the programmer from a need to know or specify absolute memory addresses.

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

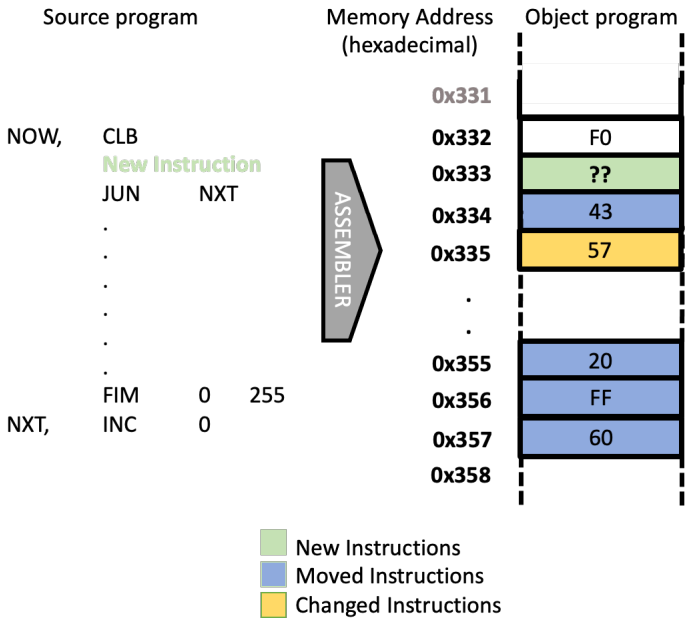


As illustrated above, the assembly language program generated by a programmer is called a **SOURCE PROGRAM**. The assembler converts the SOURCE PROGRAM into an equivalent **OBJECT PROGRAM**, which consists of a sequence of hexadecimal codes that can be loaded into ROM or program RAM and executed.

For example:



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:



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

10.13.2 Statement Mnemonics

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

Field	Name	Description
1	LABEL	It is the instruction location's label or name, and it is used to reference the instruction.
2	CODE	It defines the operation that is to be performed by the instruction.
3	OPERAND	It provides any address or data information needed by the CODE field.
4	COMMENT	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.

Some examples are shown below:

```
CMI     CLB             / Clear accumulator and carry

LAB,    INC     3       / Increment register 3

        JUN     CMI     / Jump to instruction labelled "CMI"

FCH,    FIM     0P 255  / Load 0xFF (decimal 255) into register pair 0
```

10.13.3 Label Field

This is an optional field. If present, it must start with 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.

Some examples of legal label field values are:

```
CM0,
NUL,
EGO,
```

Some examples of illegal label field values are:

```
4GE,        / Does not begin with a letter
AGE         / Valid characters, but does not end with a comma
A/A,        / Contains invalid characters
```

The following label has more than 3 characters:

```
STROB,
```

Whilst this is legal, care must be taken not to have more than one label with the first 3 characters identical.

For example, the following labels are indistinguishable from one another and will result in unpredictable behaviour:

```
LABEL,
LAB2,
LAB6
LABEL29,
```

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 detennine which **NXT** address is referenced by the **JUN** instruction.

10.13.4 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.

10.13.5 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:

```
ABC,    LDM     14      / Load accumulator with decimal 14 (1100b).
```

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

Example:

```
GO,    LDM     *+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 (the equate instruction).

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):

```
A1,     FIM  0      42
A2,     FIM  ZER    42
A3,     FIM  ZER    VAL
```

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

Example:

```
LP1,    JUN  LP2    / Jump to label 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 0xFFE, the instruction:

```
JUN VAL
```

is encoded as 0x4FFE; a 4-bit operation code and 12 bit value.

However, the instruction:

```
JUN VAL + 9
```

will be encoded as 0x4007, where the value 0x007 has been truncated on the left to 12 bits (three hex digits) giving a value o 0x007.

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:

```
I1,     INC 4
I2,     INC R4
I3,     INC 16 - 12
```

Assuming *R4* has been equated to 4, then the above instructions will **ALL** 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:

```
I1,     SRC 1P
I2,     INC 2
I3,     INC RG2
```

Assuming label *RG2* has been equated to 2, then the above instructions will **ALL** increment register pair 2 (i.e. registers 2 and 3).

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

Example:

```
AC1,    DATA    / Load the value of DATA into the accumulator
```

The value of *DATA* could be one 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:

```
HR, JUN     OVR / Jump to instruction at OVR.
    JUN     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:

```
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.

10.13.6 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
```

For the reader who understands assembly language, refer to the *summary of the 4004 instruction set*.

For the reader who is not completely familiar with assembly language, refer to the *individual instructions* with examples and machine code equivalents.

More detailled information is contained within the sections below:

- *How Assembly Language is Used*
- *Statement Mnemonics*
- *Label Field*
- *Code Field*
- *Operand Field*
- *Comment Field*

10.14 Data Statements

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” •

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:

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

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

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

Example 1:

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

```
    + 6             =   0 1 1 0
Invert each bit     =   1 0 0 1
Add 1               =   1 0 1 0
```

Therefore, the two' s complement representation of -6 is the hexadecimal number '0x0A'. (Note that the sign bit is set, indicating a negative number.)

Example 2:

To interpret the value 0x0C 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 f:um the number and inverting each bit.

```
      C             =   1 1 0 0
Invert each bit     =   0 0 1 1
Add 1               =   0 1 0 0
```

Thus, the value of 0x0C is - 4.

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

Decimal	Binary
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000

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

```
-   Take the two's complement of 3  =   1 1 0 1
-   Add the result to the minuend:
                                6   =   0 1 1 0
                          + (-3)    =   1 1 0 1
                                        -------
                                        0 0 1 1 = 3
```

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.

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

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 1:

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

```
10  =   1 0 1 0
-3  =   1 1 0 1
       -------
    1   0 1 1 1
```

(continues on next page)

(continued from previous page)

```
carry

Since the carry bit is **set**, the result is correct and positive
```

Example 2:

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

```
 12  =   1 1 0 0
-15  =   0 0 0 1
         -------
     0   1 1 0 1 = -3
 carry

 Since the carry bit is **reset**, the result is negative and in its two's compliment␣
↪form.
```

To summarise Two's complement, below is a number line showing all the 4-bit representations from +7 to -8.

Decimal	Binary
7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0	0000
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000

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.

10.15 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 a label VAL has been equated to 14, and the label LOC appears on an instruction assembled at location 0x034B

```
                                                    Assembled Data

    LDM     0           / Statement for context
C1,         0 + VAL                                     0x0E
C2,         4095                                        0xFF
C3,         0 + LOC                                     0x4B
```

The following are invalid data statements

```
    LDM     0           / Statement for context
C4,         ABC         / Does not begin with a number
C5,         -18         / Number is not positive
```

10.16 Instruction Summary

10.16.1 Index Register Instructions

FIN

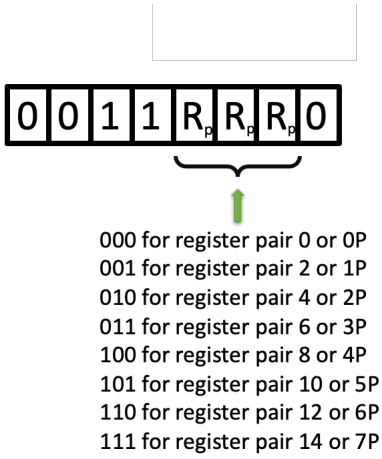
Name	Fetch Indirect from ROM
Function	8 bits of immediate data are loaded into the register pair specified by RP.
Syntax	FIN RPp
Assembled	
Binary	0010RRR0
Decimal	48, then incrementing by 2 until 62 (1st word)
Hexadecimal	0x30, then incrementing by 2 until 0x3E (1st word)
Symbolic	$((P_H) . (R_0) . (R_1)) \Rightarrow RP_p$ OR $((P_{H+1}) . (R_0) . (R_1)) \Rightarrow RP_p$
Execution	1 word, 8-bit code but with an execution time of 21.6 μ sec
Side-effects	Not Applicable, unless RP0 is the designated target register pair, in which case, RP0 will contain the data at the memory location referenced by RP0 at the start of the instruction
Implemented	fin

Detailed 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 registers 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 = O.

The carry bit is not affected.

The target register pair is defined as part of the opcode as detailed below.



Example program

```
/ Example program
        org     ram
        fin     7p
        end
```

(Assume that address 0x25B contains the data 0x6E (spread over 2 words)).

If register 0 contains 0x5 and register 1 contains 0xB, when the FIN instruction is executed, the 8 bits located at hex address 0x25B will be loaded into register pair 7P. Thus register 14 will contain 0x6, and register 15 will contain 0xE.

Notes

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 (0x1FF) and registers 0 and 1 contain 3 and 0xC, the 8 bits at address **0x23C** (not 0x13C) will be loaded into registers 14 and 15.

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

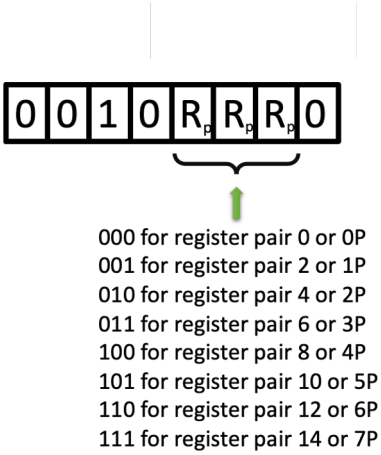
INC

Name	Increment Register
Function	Increments a specified register by 1.
Syntax	INC(R)
Assembled	
Binary	0110RRRR
Decimal	96, then incrementing by 1 until 111
Hexadecimal	0x60, then incrementing by 1 until 0x6F
Symbolic	(RRRR) + 1 ⟹ RRRR
Execution	1 word, 8-bit code and an execution time of 10.3 μ sec
Side-effects	Not Applicable
Implemented	inc

Detailed Description

The address contained within the specified register pair designates either a particular DATA RAM data character, a DATA RAM status character, a RAM output port, or a ROM input/output port. However, the address designates all of these simultaneously; it is up to the programmer to then write the correct I/O or RAM instruction to access the proper entity.

The disassembly of the instruction below shows how the register pair are represented in the opcode.



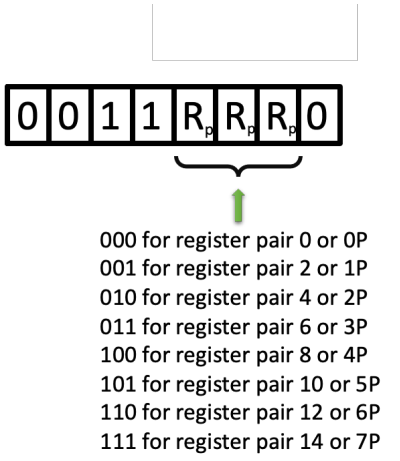
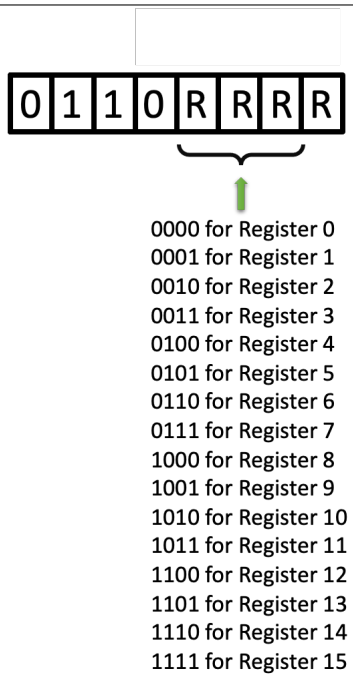
The register specified in the lower 4 bits of the instruction is incremented by 1. The carry bit will remain unchanged. If the register specified contains a value of 0b1111 and an INC instruction is applied, the register will contain a value of 0b0000, but the carry bit will remain unchanged

Example program

```
/ Example program
/ Loads the Accumulator with a value of 2
/ places that value in Register 6
/ increments Register 6
/ Register 6 contains a value of 3
       org    ram
       ld     2
       xch    6
       inc    6
       end
```

The index register instructions involve index registers or register pairs.

These instructions occupy one byte as follows:

FIN	INC
	

Code	Description
FIN	Load RP with 8 bits of ROM data addressed by register pair 0.
INC	Increment register REG.

10.16.2 Index Register To Accumulator Instructions

ADD

Name	Add register to accumulator with carry
Function	Add a value from a specified register to the accumulator, respecting the carry flag.
Syntax	ADD R
Assembled	
Binary	1000 R
Decimal	128, then incrementing by 1 until 143
Hexadecimal	0x80, then incrementing by 1 until 0x8F
Symbolic	(RRRR) + (ACC) + (CY) ⟹ ACC, CY
Execution	1 word, 8-bit code and an execution time of 10.8 μ sec
Side-effects	Depending on the result, the carry bit is reset or set.
Implemented	add

Detailed Description

The 4 bit content of the designated index register is added to the content of the accumulator with carry. The result is stored in the accumulator. The carry/link is set to 1 if a sum greater than 15 was generated to indicate a carry out; otherwise, the carry/link is set to 0. The 4 bit content of the index register is unaffected.

Example programs

```
/ Example program 1
        org    ram
        ldm    9
        xch    12
        ldm    6
        clc
        add    12
        end
```

In this example, the accumulator contains a value of 6, register 12 contains a value of 9, and the carry bit is 0.

Performing an ADD 12 (add the value of the accumulator to that in register 12) does the following:

```
Accumulator    =   0 1 1 0
Register 12    =   1 0 0 1
Carry          =         0
                  ---------
Result       0     1 1 1 1
          Carry
```

The accumulator contains 15 and the carry bit is reset.

```
/ Example program 2
        org     ram
        ldm     9
        xch     12
        ldm     6
        stc
        add     12
        end
```

In this example, the accumulator contains a value of 6, register 12 contains a value of 9, and the carry bit is 1 - note the STC instruction replacing the CLC instruction.

Performing an ADD 12 (add the value of the accumulator to that in register 12) does the following:

```
Accumulator    =    0 1 1 0
Register 12    =    1 0 0 1
Carry          =          1
                   ---------
Result       1     0 0 0 0
           Carry
```

The accumulator contains 0 and the carry bit is set.

SUB

Name	Subtract index register from accumulator with borrow
Function	Subtract a value in an index register from the accumulator, respecting the carry flag.
Syntax	SUB R
Assembled	
Binary	1001 R
Decimal	144, then incrementing by 1 until 159
Hexadecimal	0x90, then incrementing by 1 until 0x9F
Symbolic	$(ACC) + \overline{(RRRR)} + \overline{(CY)} \Longrightarrow ACC, CY$
Execution	1 word, 8-bit code and an execution time of 10.8 μ sec
Side-effects	Depending on the result, the carry bit is reset or set.
Implemented	sub

Detailed Description

The contents of index register R are subtracted with borrow from the accumulator. The result is kept in the accumulator; the contents of R 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.

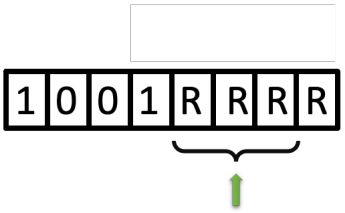
This instruction sets the carry bit if there is no borrow out of the high order bit position, and resets the carry bit if there is a borrow.

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

Notes

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 *"Decimal Subtraction"*:.

The disassembly of the instruction below shows how the register is represented in the opcode:



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

Example programs

In order to perform a normal subtraction, the carry bit should be zero. If the accumulator contains 6, register 10 contains 2, and the carry bit is zero.

This is the set-up for the operation *6 - 2*, giving the answer *4* in the accumulator.

```
/ Example program 1
        org     ram
        ldm     2
        xch     10
        ldm     6
        clc
        sub     10
        end
```

The *sub* operation above is carried out as follows:

```
   Accumulator    =   0 1 1 0
~  Register 10    =   1 1 0 1     ( register 10 = 0 0 1 0)
~  Carry          =         1     ( carry = 0)
```

(continues on next page)

(continued from previous page)

```
                  ---------
   Result      1     0 1 0 0
             Carry indicates no borrow
```

The accumulator contains 4 and the carry bit is reset.

In this second example, if the accumulator contains 6, register 10 contains 2, and the carry bit is one:

```
/ Example program 2
        org    ram
        ldm    2
        xch    10
        ldm    6
        stc
        sub    10
        end
```

The *sub* operation above is carried out as follows:

```
   Accumulator    =   0 1 1 0
~  Register 10    =   1 1 0 1    ( register 10 = 0 0 1 0)
~  Carry          =         0    ( carry = 1)
                  ---------
    Result      1     0 0 1 1
             Carry indicates no borrow
```

The accumulator contains 3 and the carry bit is reset.

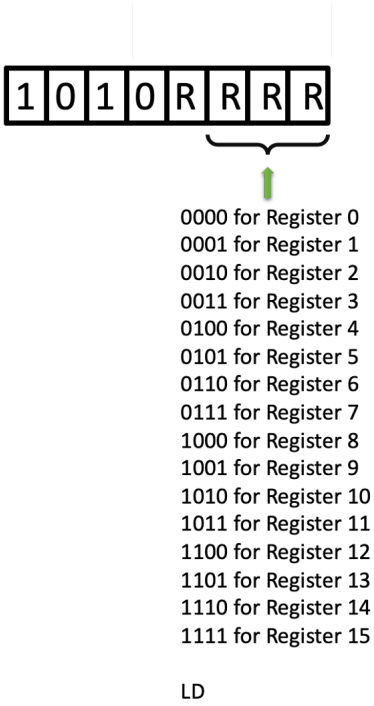
LD

Name	Load index register to Accumulator
Function	The 4 bit content of the designated index register (RRRR) is loaded into accumulator. The previous contents of the accumulator are lost. The 4 bit content of the index register and the carry/link bit are unaffected..
Syntax	LD(R)
Assembled	
Binary	1010 RRRR
Decimal	160, then incrementing by 1 until 175 (1st word).
Hexadecimal	0xA0, then incrementing by 1 until 0xAF (1st word).
Symbolic	(RRRR) ⟹ ACC
Execution	1 word, 8-bit code and an execution time of 10.3 μ sec
Side-effects	Not Applicable
Implemented	ld

Detailed 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 and the accumulator are not affected.



Example program

The example program will load the contents of register 11 into the accumulator.

If register 11 contains the value 9 (1001b), then after this program is executed, the accumulator will contain 9 also.

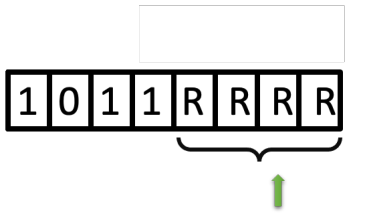
```
/ Example program
LD    11
END
```

XCH

Name	Exchange index register and accumulator
Function	The contents of the register specified by REG are exchanged with the contents of the accumulator. The carry bit is not affected.
Syntax	XCH(R)
Assembled	
Binary	1011 RRRR
Decimal	176, then incrementing by 1 until 191 (1st word).
Hexadecimal	0xB0, then incrementing by 1 until 0xBF (1st word).
Symbolic	(ACC) ➡ ACBR (RRRR) ➡ ACC (ACBR) ➡ RRRR
Execution	1 word, 8-bit code and an execution time of 10.3 μ sec
Side-effects	Not Applicable
Implemented	xch

Detailed Description

The contents of the register specified by REG are exchanged with the contents of the accumulator. The carry bit is not affected.



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

Example program

If the accumulator contains 1100 and register 0 contains 0011 then the instruction XCH 0 will cause the accumulator to contain 0011 and register 0 to contain 1100.

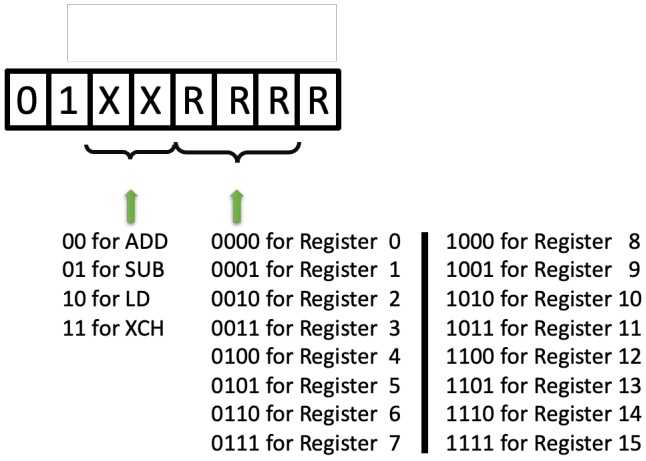
```
/ Example program
XCH   0
END
```

Note

ACBR is the **Accumulator Buffer Register (4-bit)**

This section describes instructions which involve an operation between an index register and the accumulator.

Instructions in this class occupy one byte as follows:



Code	Description
ADD	Add REG plus carry bit to the accumulator.
SUB	Subtract REG from accumulator with borrow.
LD	Load accumulator from REG.
XCH	Exchange the contents of accumulator and REG.

10.16.3 Accumulator Instructions

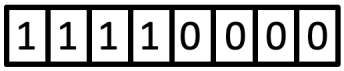
CLB

Name	Clear Both
Function	Clear both the accumulator and carry bit
Syntax	CLB
Assembled	
Binary	11110000
Decimal	240
Hexadecimal	0xF0
Symbolic	0 ➡ ACC 0 ➡ CY
Execution	1 word, 8-bit code and an execution time of 10.3 μ sec
Side-effects	Carry bit and Accumulator are zeroed
Implemented	clb

Detailed Description

The accumulator is set to 0 and the carry bit is reset.

The opcode for this instruction does not contain any additional data:



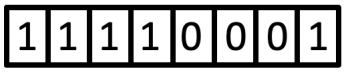
CLC

Name	Clear Carry
Function	Clear the carry bit
Syntax	CLC
Assembled	
Binary	11110001
Decimal	241
Hexadecimal	0xF1
Symbolic	0 ➡ CY
Execution	1 word, 8-bit code and an execution time of 10.3 μ sec
Side-effects	Carry bit is reset
Implemented	clc

Detailed Description

The carry bit is reset.

The opcode for this instruction does not contain any additional data:



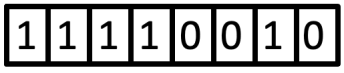
IAC

Name	Increment accumulator
Function	The content of the accumulator is incremented by 1.
Syntax	IAC
Assembled	
Binary	11110010
Decimal	242
Hexadecimal	0xF2
Symbolic	(ACC) + 1 ⟹ ACC
Execution	1 word, 8-bit code and an execution time of 10.8 μ sec
Side-effects	No overflow sets the carry/link to 0; Overflow sets the carry/link to a 1.
Implemented	iac

Detailed 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.

The opcode for this instruction does not contain any additional data:



Examples

Example 1

If the accumulator contains 9, then the IAC operation will be as follows:

```
Accumulator    =    1 0 0 1
          +                1
                  ---------
 Result       0     1 0 1 0
            Carry
```

Example 2

If the accumulator contains 15, then the IAC operation will be as follows:

```
Accumulator   =   1 1 1 1
         +              1
                  ---------
 Result      1    0 0 0 0
           Carry
```

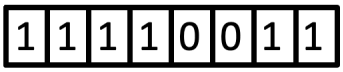
CMC

Name	Complement Carry
Function	The carry/link content is complemented.
Syntax	CMC
Assembled	
Binary	11110011
Decimal	243
Hexadecimal	0xF3
Symbolic	$(\overline{CY}) \Rightarrow CY$
Execution	1 word, 8-bit code and an execution time of 10.3 μ sec
Side-effects	Carry bit is complemented
Implemented	cmc

Detailed Description

The contents of the carry is complemented, i.e. if the carry bit is 1, it is set to zero. If it is zero, it is set to 1.

The opcode for this instruction does not contain any additional data:



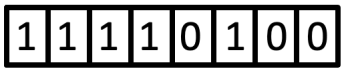
CMA

Name	Complement Accumulator
Function	Perform one's complement on the accumulator
Syntax	CMA
Assembled	
Binary	11110100
Decimal	244
Hexadecimal	0xF4
Symbolic	$\overline{a_3}\ \overline{a_2}\ \overline{a_1}\ \overline{a_0} \Rightarrow ACC$
Execution	1 word, 8-bit code and an execution time of 10.3 μ sec
Side-effects	N/A
Implemented	cma

Detailed Description

The contents of the carry is complemented, i.e. if the carry bit is 1, it is set to zero. If it is zero, it is set to 1.

The opcode for this instruction does not contain any additional data:



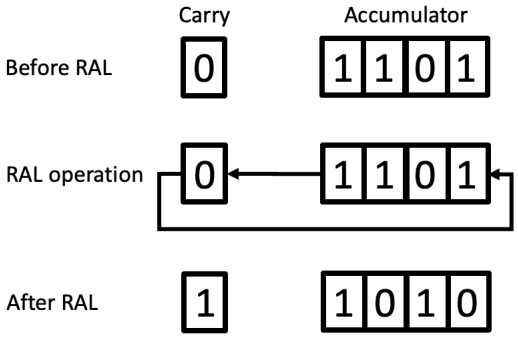
RAL

Name	Rotate Accumulator left through Carry
Function	The content of the accumulator and carry/link are rotated left.
Syntax	RAL
Assembled	
Binary	11110101
Decimal	245
Hexadecimal	0xF5
Symbolic	$(CY) \rightarrow a_0$ $a_i \rightarrow a_{i+1}$ $a_3 \rightarrow CY$
Execution	1 word, 8-bit code and an execution time of 10.3 μ sec
Side-effects	As described
Implemented	ral

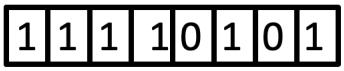
Detailed Description

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 as shown in the example below:



The opcode for this instruction does not contain any additional data:



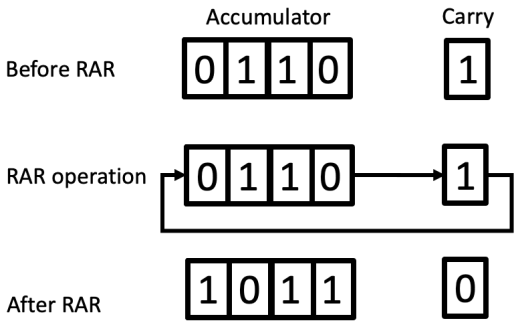
RAR

Name	Rotate Accumulator right through Carry
Function	The content of the accumulator and carry/link are rotated right.
Syntax	RAR
Assembled	
Binary	11110110
Decimal	246
Hexadecimal	0xF6
Symbolic	$a_0 \Rightarrow CY$ $a_i \Rightarrow a_{i-1}$ $CY \Rightarrow a_3$
Execution	1 word, 8-bit code and an execution time of 10.3 μ sec
Side-effects	As described
Implemented	rar

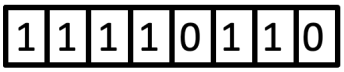
Detailed Description

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.



The opcode for this instruction does not contain any additional data:



TCC

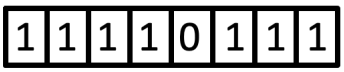
Name	Transmit Carry and Clear
Function	The accumulator is cleared. The least significant position of the accumulator is set to the value of the carry/link. The carry/link is set to 0.
Syntax	TCC
Assembled	
Binary	11110111
Decimal	247
Hexadecimal	0xF7
Symbolic	
Execution	1 word, 8-bit code and an execution time of 10.3 μ sec
Side-effects	Carry bit is reset
Implemented	tcc

Detailed Description

If the carry bit is zero, the accumulator is set to 0000. If the carry bit is one, the accumulator is set to 0001.

In either case, the carry bit is then reset.

The opcode for this instruction does not contain any additional data:



DAC

Name	Decrement accumulator
Function	The content of the accumulator is decremented by 1.
Syntax	DAC
Assembled	
Binary	11111000
Decimal	248
Hexadecimal	0xF8
Symbolic	
Execution	1 word, 8-bit code and an execution time of 10.8 μ sec
Side-effects	No borrow sets the carry/link to 1; Borrow sets the carry/link to a 0.
Implemented	iac

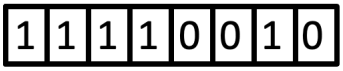
Detailed Description

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.

Note

Subtracting a number is carried out using the complement of the number and adding. Therefore subtracting 1 becomes adding -1.

The opcode for this instruction does not contain any additional data:



Examples

Example 1

If the accumulator contains 9, then the DAC operation will be as follows:

```
Accumulator    =    1 0 0 1
    + (-1)          1 1 1 1
                   ---------
 Result       1     1 0 0 0
           Carry (indicating no borrow)
```

Example 2

If the accumulator contains 0, then the DAC operation will be as follows:

```
Accumulator    =    0 0 0 0
    + (-1)          1 1 1 1
                   ---------
 Result       0     1 1 1 1
           Carry (indicating a borrow)
```

TCS

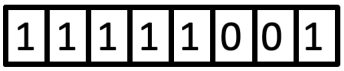
Name	Transmit Carry and Subtract
Function	The accumulator is set to 9 if the carry/link is 0. The accumulator is set to 10 if the carry/link is a 1. The carry/link is set to 0.
Syntax	TCS
Assembled	
Binary	11111001
Decimal	249
Hexadecimal	0xF9
Symbolic	1011 ➡ ACC If (CY) = 0 1010 ➡ ACC If (CY) = 1 0 ➡ CY
Execution	1 word, 8-bit code and an execution time of 10.3 μ sec
Side-effects	Carry bit is reset
Implemented	tcs

Detailed 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.

The opcode for this instruction does not contain any additional data:



This instruction is used when subtracting decimal numbers greater than 4 bits in length. For an example of this, see *here*

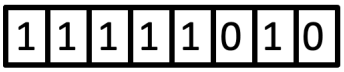
STC

Name	Set Carry
Function	Set the carry bit
Syntax	STC
Assembled	
Binary	11111010
Decimal	250
Hexadecimal	0xFA
Symbolic	1 ➡ CY
Execution	1 word, 8-bit code and an execution time of 10.3 μ sec
Side-effects	Carry bit is set
Implemented	stc

Detailed Description

The carry bit is set.

The opcode for this instruction does not contain any additional data:



DAA

Name	Decimal Adjust Accumulator
Function	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.
Syntax	DAA
Assembled	
Binary	11111011
Decimal	251
Hexadecimal	0xFB
Symbolic	(ACC) + (0000 or 0110) ➡ ACC
Execution	1 word, 8-bit code and an execution time of 10.3 μ sec
Side-effects	As Described
Implemented	daa

Detailed 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.

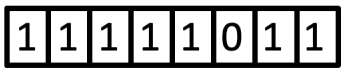
If the result of incrementing the accumulator produces a carry out of the high order bit position, the cary bit is set.

Otherwise the carry bit is unaffected (**in particular it is not reset**).

Notes

This instruction is used when adding decimal numbers. For an example of this see *Decimal Addition*:

The opcode for this instruction does not contain any additional data:



KBP

Name	Keyboard Process
Function	If the accumulator contains OOOOB, 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.
Syntax	KBP
Assembled	
Binary	11111100
Decimal	252
Hexadecimal	0xFC
Symbolic	(ACC) ⟹ KBP ROM ⟹ ACC
Execution	1 word, 8-bit code and an execution time of 10.3 μ sec
Side-effects	Not Applicable
Implemented	kbp

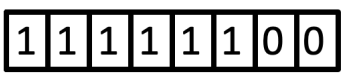
Detailed Description

A code conversion is performed on the accumulator content, from 1 out of n to binary code. If the accumulator content has more than one bit on, the accumulator will be set to 15 (to indicate error). The carry/link is unaffected.

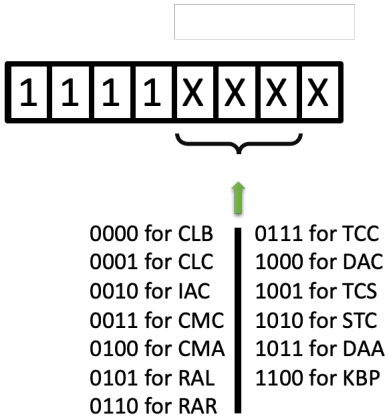
The conversion table is shown below:

Accumulator before KBP	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 opcode for this instruction does not contain any additional data:



Accumulator instructions operate only on the contents of the accumulator and/or the carry bit. Instructions in this class occupy one byte as follows:



Code	Description
CLB	Clear both the accumulator and carry.
CLC	Clear carry.
IAC	Increment accumulator.
CMC	Complement carry.
CMA	Complement each bit of the accumulator.
RAL	Rotate accumulator left through carry.
RAR	Rotate accumulator right through carry.
TCC	Transmit the value of the carry to the accumulator then clear carry.
DAC	Decrement accumulator.
TCS	Adjust accumulator for decimal subtract.
STC	Set carry.
DAA	Adjust accumulator for decimal add.
KBP	Convert accumulator from 1 of n code to a binary value.

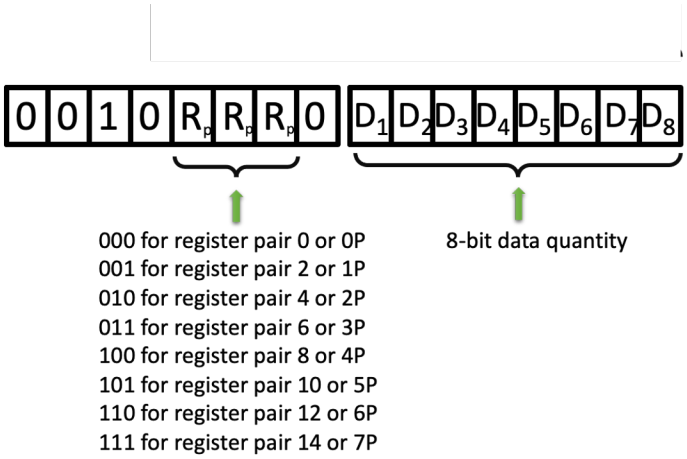
10.16.4 Immediate Instructions

FIM

Name	Fetch Immediate
Function	8 bits of immediate data are loaded into the register pair specified by RP.
Syntax	FIM RPp Data
Assembled	
Binary	0010RRR0 DDDDDDDD
Decimal	32, then incrementing by 2 until 46 (1st word)
Hexadecimal	0x20, then incrementing by 2 until 0x2E (1st word)
Symbolic	$D_1D_2D_3D_4 \Rightarrow R_p\,R_p\,R_p\,R_p$ $D_5D_6D_7D_8 \Rightarrow R_{p+1}\,R_{p+1}\,R_{p+1}\,R_{p+1}$
Execution	2 words, 8-bit code and an execution time of 21.6 μ sec
Side-effects	Not Applicable
Implemented	fim

Detailed Description

The 8 bits of immediate data (word 2) are loaded into the named register pair. The register pairs are defined within the opcode as shown below:



Example program

```
/ Example program
        org     ram
        fim     2  254
        end
```

This will load the 8-bit decimal value 254 into the register pair 2 & 3.

After execution, register 2 will contain the upper 4 bits of the value 254, with register 3 containing the lower 4 bits i.e. 15 and 14 respectively.

This is because decimal 254 is represented as 0xFE, so register 2 will contain 0xF (decimal 15), while register 3 will contain 0xE (decimal 14).

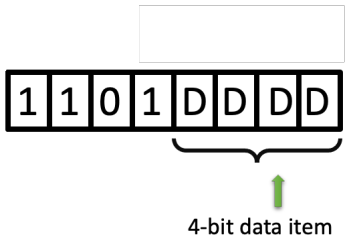
LDM

Name	Branch Back and Load
Function	The 4 bits of immediate data encoded in the instruction are loaded into the accumuator, then execution continues with the most recent address on the stack. contents of the accumulator. The carry bit is not affected.
Syntax	LDM(D)
Assembled	
Binary	1100 DDDD
Decimal	208, then incrementing by 1 until 223 (1st word).
Hexadecimal	0xD0, then incrementing by 1 until 0xDF (1st word).
Symbolic	DDDD ➡ ACC
Execution	1 word, 8-bit code and an execution time of 10.3 μ sec
Side-effects	Not Applicable
Implemented	ldm

Detailed Description

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

The carry bit is not affected.



Example Program

```
/ Example program
        ldm   0
        ldm   9
        ldm   15
        end
```

The above program will first clear the accumulator (setting all 4 bits to 0), then load the value 9 into the accumulator, then finally, set all the accumulator's 4 bits by loading the value 15.

There are two instructions which use data that is part of the instruction itself.

Code	Description
FIM	Load 8 bit immediate DATA into register pair RP.
LDM	Load 4-bit immediate DATA into the accumulator.

10.16.5 Transfer Of Control Instructions

JUN

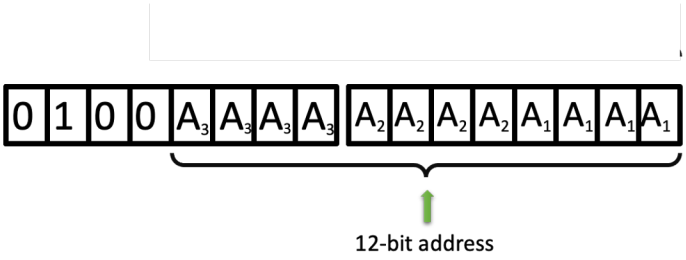
Name	Jump Unconditionally
Function	Jump to any address within the memory space.
Syntax	JUN (address)
Assembled	
Binary	0010RAAAA AAAAAAAA
Decimal	64, then incrementing by 1 until 79 (1st word)
Hexadecimal	0x40, then incrementing by 2 until 0x4F (1st word)
Symbolic	$A_3 A_3 A_3 A_3 \rightarrow P_H$ $A_2 A_2 A_2 A_2 \rightarrow P_M$ $A_1 A_1 A_1 A_1 \rightarrow P_L$
Execution	2 words 16-bit code and an execution time of 21.6 μ sec
Side-effects	Not Applicable
Implemented	jun

Detailed 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, nor the contents of the register pair are not affected.

The disassembly of the instruction below shows how the register pair is represented in the opcode.



This instruction and the JMS instruction , 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 instructions are.

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

Example program snippet for illustration

```
Arbitrary Memory
Address   (Hex)

  0x360               jun LRG
  0x362         AD,   add 1



  0x370         LAC,  ldm 3
  0x371               jun AD



  0x3EO         LRG,  fim 0p, 4
  0x3E2               jun LAC
                      end
```

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 the above example is as follows:

The **jun** instruction at 0x360 replaces the contents of the program counter with 0x3EO. 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 0x3E2 is then executed. The program counter is set to 0x370, and the **ldm** at this address loads the accumulator with the value 3.

The **jun** at 0x371 sets the program counter to 0x362, 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 0x363.

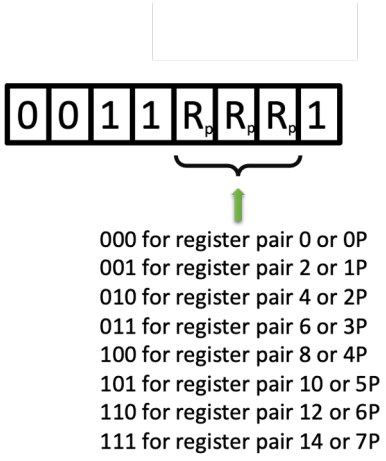
JIN

Name	Jump Indirect
Function	Jump to an address within this page of ROM.
Syntax	JIN(Rp)
Assembled	
Binary	0011RPp1
Decimal	49, then incrementing by 2 until 63 (1st word)
Hexadecimal	0x31, then incrementing by 2 until 0x3F (1st word)
Symbolic	$\{P_H \rightarrow P_H\}$ unchanged $R_p R_p R_p R_p \rightarrow P_M$ $R_{p+1} R_{p+1} R_{p+1} R_{p+1} \rightarrow P_L$
Execution	1 words 8-bit code and an execution time of 10.3 μ sec
Side-effects	Not Applicable
Implemented	jin

Detailed 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.



Example program

```
/ Example program
        org     0x3e4
        fim     0p 21
        jin     0p
        end
```

The FIM instruction loads register 0 with the value 1 and register 1 with the value 5. The JIN instruction then causes a jump to location 0x315.

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 (0x0FF) then control would have been transferred to address 0x115, not 0x015.

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

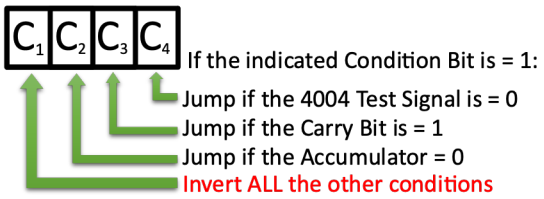
JCN

Name	Jump Conditional
Function	Jump if satisfying a set of conditions
Syntax	JCN
Assembled	
Binary	0001 CCCC AAAAAAAA
Decimal	16 - 31 (1st word)
Hexadecimal	0x10 - 0x21 (1st word)
Symbolic	If $C_1C_2C_3C_4$ is true: $A_2A_2A_2A_2$ ⟹ PM $A_1A_1A_1A_1$ ⟹ PL PH unchanged if $C_1C_2C_3C_4$ is false: (PH) ⟹ PH (PM) ⟹ PM (PL + 2) ⟹ PL
Execution	1 word, 8-bit code and an execution time of 10.8 μ sec
Side-effects	Not Applicable
Implemented	jcn

Detailed Description

If the condition specified by C_x is false, no action occurs and program execution continues with the next sequential instruction. If the condition specified by C_x is true, the 8 bits specified by AAAA 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. 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:

$$JUMP = \overline{C_1} * ((ACC = 0) * C_2 + (carry = 1) * C_3 + \overline{TEST} * C_4 +$$

$$C_1 * (((ACC <> 0) + \overline{C_2}) + ((carry = 0) + \overline{C_3}) * (TEST + \overline{C_4})$$

Test Pin 10

A special capability of the JCN instruction is being able to read the status of Pin 10 on the processor.

Pin 10 (shown below toward the top left of the layout) is the "TEST" pin.

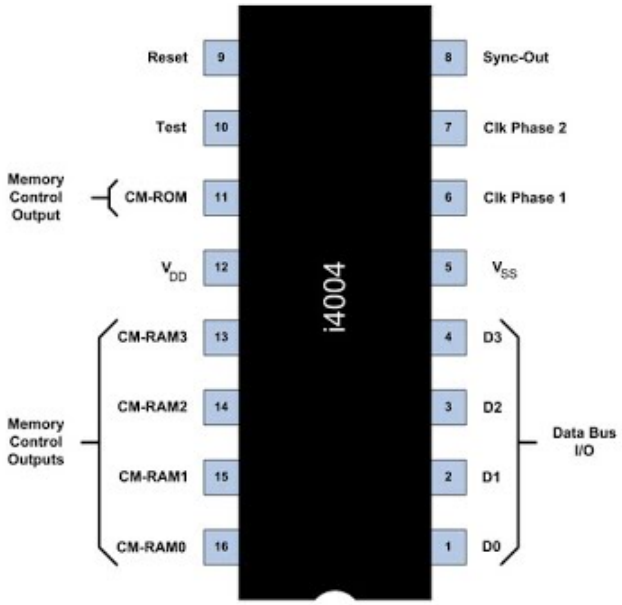
Since this is a software implementation of the i4004, a directive has been introduced to the assembler to allow simulation of Pin 10.

The directive 'pin' can set the pin 10 to be 1 or 0 depending oon the supplied value e.g.

pin 1

pin 0

The subsequent value of Pin 10 will be used until it is changed again, or the program ends.



Example program

```
/ Example program
        org     ram
        ldm     10
        jcn     4     done
        dac
        jun     loop
 done,  end
```

Loads the value 10 into the accumulator, and decrements it by one. Once the value of the accumulator is zero, the program ends.

Notes

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

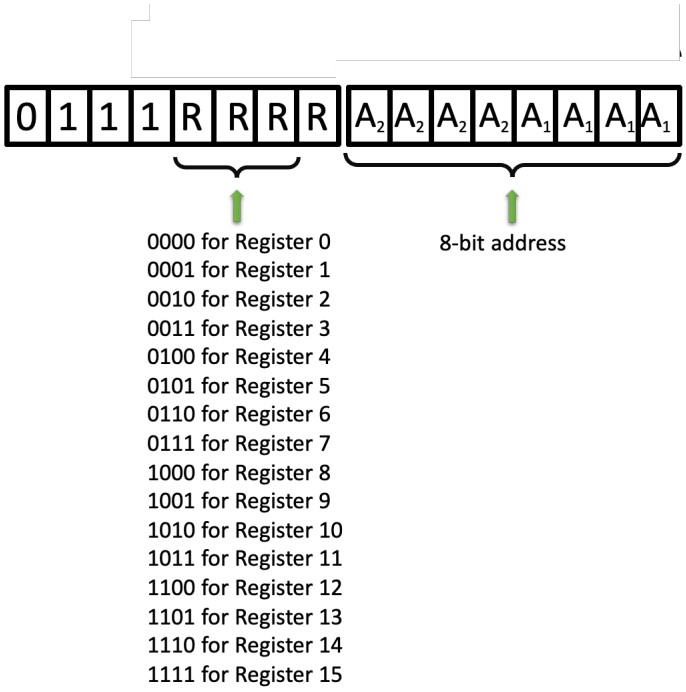
ISZ

Name	Increment index register skip if zero
Function	The content of the designated index register is incremented by 1. The accumulator and carry/link are unaffected. If the result is zero, the next instruction after ISZ is executed. If the result is different from 0, program control is transferred to the instruction located at the 8 bit address A2A2A2A2, A1A1A1A1 on the same page (ROM) where the ISZ instruction is located.
Syntax	ISZ(R, 8-bit address)
Assembled	
Binary	0111R A2A2A2A2, A1A1A1A1
Decimal	112, then incrementing by 1 until 127 (1st word)
Hexadecimal	0x70, then incrementing by 1 until 0x7F (1st word)
Symbolic	(RRRR) + 1 ⟹ RRRR If result = 0 (P_H) ⟹ P_H (P_M) ⟹ P_M (P_L+2) ⟹ P_L If result <> 0 (P_H) ⟹ P_H (A_2 A_2 A_2 A_2) ⟹ P_M (A_1 A_1 A_1 A_1) ⟹ P_L
Execution	2 words, 8-bit code and an execution time of 10.8 μ sec
Side-effects	Not Applicable
Implemented	isz

Detailed Description

The index register specified by REG is incremented by one. If the result is 0000, program execution continues with the next sequential instruction. If the result does not equal 0000 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



NOTE: If ISZ is located on words 254 and 255 of a ROM page, when ISZ is executed and the result is not zero, program control is transferred to the 8-bit address located on the next page in sequence and not on the same page where ISZ is located. .. rubric:: Example program

```
/ Example program
        org     ram
        fim     0p
lp      xch     2
        isz     0    lp
        end
```

The FIM instruction loads registers 0 and 1 with O. 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 labelled “lp”. This process continues until register 0 = 1111. Then the ISZ increments register 0 producing a result of OOOO, and execution continues with the instruction at after the ISZ (which is the END).

Instructions which alter the normal execution sequence of instructions.

Code	Description
JUN	Jump to location ADDR.
JIN	Jump to the address in register pair RP.
JCN	Jump to ADDR if condition true.
ISZ	Increment REG. If zero, skip. If non zero, jump to ADDR

10.16.6 Subroutine Linkage Instructions

JMS

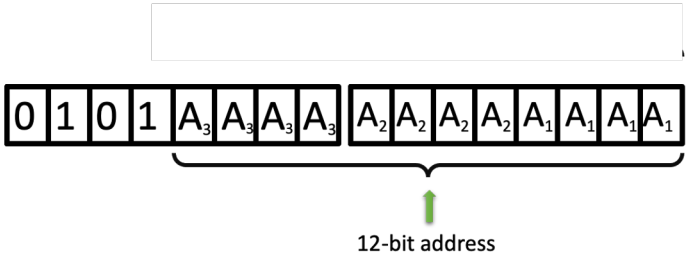
Name	Jump to Subroutine
Function	Jump to a subroutine.
Syntax	JMS (address)
Assembled	
Binary	0101AAAA AAAAAAAA
Decimal	80, then incrementing by 1 until 95 (1st word)
Hexadecimal	0x50, then incrementing by 2 until 0x5F (1st word)
Symbolic	$A_3 A_3 A_3 A_3 \rightarrow P_H$ $A_2 A_2 A_2 A_2 \rightarrow P_M$ $A_1 A_1 A_1 A_1 \rightarrow P_L$
Execution	2 words 16-bit code and an execution time of 21.6 μ sec
Side-effects	Not Applicable
Implemented	jms

Detailed 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.

The disassembly of the instruction below shows how the register pair is represented in the opcode.



This instruction and the JUN instruction , 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 instructions are.

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

Example program snippet for illustration

```
        jms lab
        xch 0


lab,    inc 1


        bbl 6
```

Normally, program instructions are executed sequentially.

A 12-bit register called the **program counter** holds the address of the instruction to be executed. The JMS instruction replaces the program counter contents, causing program execution to continue at that address, whilst also placing the address of the next instruction on the stack.

Thus the execution sequence of the above example is as follows:

The **jms** instruction replaces the contents of the program counter with the address of the label *lab*. The next instruction executed is **inc**.

Additional instructions are then executed, then the **bbl** instruction.

The **bbl** instruction then retrieves the topmost address from the stack (the address of the **xch** instruction), sets the program counter to that address.

From here, normal program execution continues at that location.

BBL

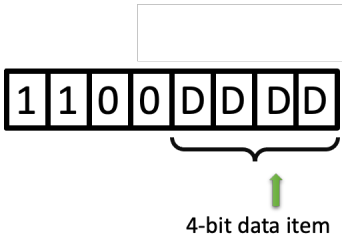
Name	Load Accumulator Immediate
Function	The 4 bits of data, DDDD stored in the OPA field of the insruction word | br| are loaded into the accumulator. The previous contents of the acummulator | br | are lost. The carry/link bit is unaffected.
Syntax	LDM(D)
Assembled	
Binary	1101 DDDD
Decimal	192, then incrementing by 1 until 207 (1st word).
Hexadecimal	0xC0, then incrementing by 1 until 0xCF (1st word).
Symbolic	DDDD ⟹ ACC
Execution	1 word, 8-bit code and an execution time of 10.3 μ sec
Side-effects	Not Applicable
Implemented	bbl

Detailed Description

The program counter (address stack) is pushed down one level. Program control transfers to the next instruction following the last jump to subroutine (*JMS*) instruction.

The 4 bits of data DDDD stored in the OPA portion of the instruction are loaded to the accumulator.

BBL is used to return from a subroutine to main program. The carry bit is not affected.



Note

In the example *here*, 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.

This section describes the commands which call and cause return from subroutines. They cause a transfer of program control and use the address stack XXXX(see Sections 2.4 and 2.7•7)XXX

Code	Description
JMS	Call subroutine and push return address onto stack.
BBL	Return from subroutine and load accumulator with immediate DATA.

10.16.7 Nop Instructions

NOP

Name	No Operation
Function	No operation performed
Syntax	NOP
Assembled	
Binary	0000 0000
Decimal	0
Hexadecimal	0x00
Symbolic	Not Applicable
Execution	1 word, 8-bit code and an execution time of 10.8 μ sec
Side-effects	Not Applicable
Implemented	nop

Description

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

Example program

```
/ Example program
        org    ram
        nop
        end
```

The program does nothing, since the NOP operation is the only operator in the program.

Notes

The NOP instruction is useful for padding out memory positions for those operators that function differently at the page boundary, such that they do not end at a page boundary.

This instruction occupies one byte.

Code	Description
NOP	No Operation

10.16.8 Memory Selection Instructions

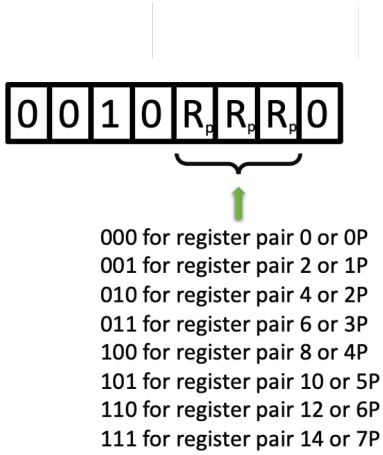
SRC

Name	Send Register Control
Function	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.
Syntax	SRC(RPp)
Assembled	
Binary	0010RPp1
Decimal	33, then incrementing by 2 until 47 (1st word)
Hexadecimal	0x21, then incrementing by 2 until 0x2F (1st word)
Symbolic	$(RP_p) \Rightarrow DB(x2)$ $(RP_{p+1}) \Rightarrow DB(x3)$
Execution	1 words, 8-bit code and an execution time of 10.3 μ sec
Side-effects	Not Applicable
Implemented	src

Detailed Description

The address contained within the specified register pair designates either a particular DATA RAM data character, a DATA RAM status character, a RAM output port, or a ROM input/output port. However, the address designates all of these simultaneously; it is up to the programmer to then write the correct I/O or RAM instruction to access the proper entity.

The disassembly of the instruction below shows how the register pair are represented in the opcode.

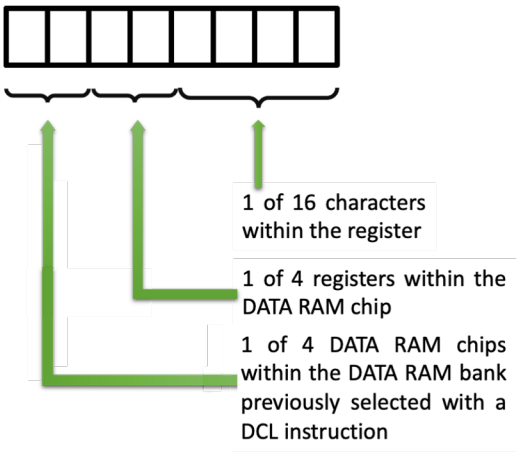


The address sent by the SRC remains in effect until changed by a subsequent SRC.

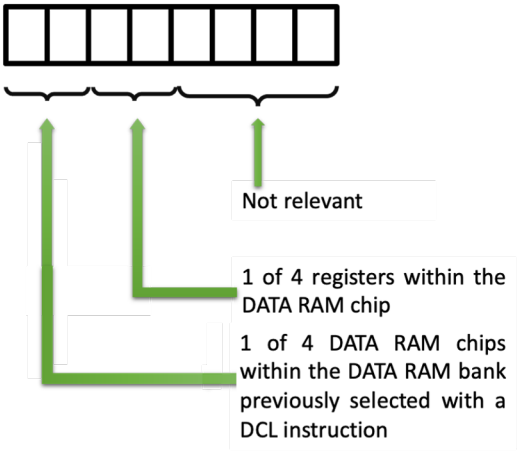
The only DATA RAM bank which receives the SRC address is the one selected by the last previous DCL instruction.

The 8 bits of the address sent by the SRC are interpreted in one of four ways, depending on the context as follows:

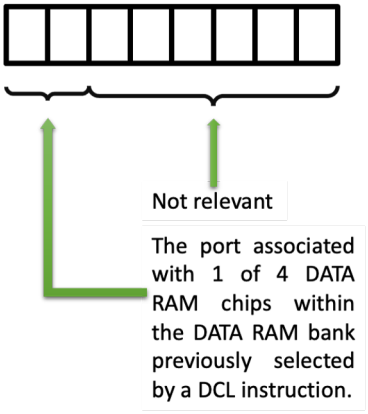
When referring to a DATA RAM Character



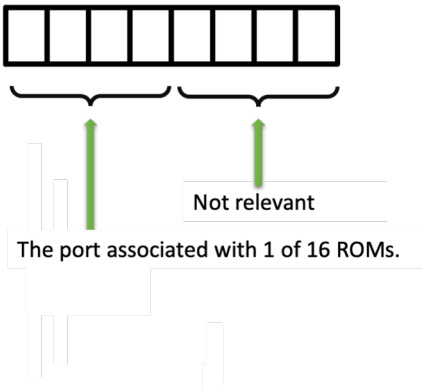
When referring to a DATA RAM Status Character



When referring to a DATA RAM Output Port



When referring to a ROM I/O Port



Example program

```
/ Example program
        org     ram
        fim     1p  180
        src     1p
        end
```

DCL

Name	Designate Command Line
Function	Select a RAM bank
Syntax	DCL
Assembled	
Binary	11111101
Decimal	253
Hexadecimal	0xFD
Symbolic	a0 ➡ CM0 a1 ➡ CM1 a2 ➡ CM2
Execution	1 word, 8-bit code and an execution time of 10.3 μ sec
Side-effects	Not Applicable
Implemented	dcl

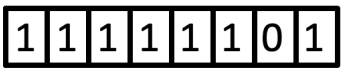
Detailed Description

The content of the three least significant accumulator bits is transferred to the comand control register within the CPU. This instruction provides RAM bank selection when multiple RAM banks are used, since there could be up to 8 RAM banks.

(If no DCL instruction is sent out, RAM Bank number zero is automatically selected after application of at least one RESET).

DCL remains latched until it is changed.

The opcode for this instruction does not contain any additional data:



The least significant 3 bits of the accumulator determine which RAM bank is selected (detailled in the table below, along with the bits of the command register).

Accumulator	CM-RAM $_i$ enabled				RAM Bank
0x000	CM-RAM $_0$				0
0x001		CM-RAM $_1$			1
0x010			CM-RAM $_2$		2
0x100				CM-RAM $_3$	3
0x011		CM-RAM $_1$	CM-RAM $_2$		4
0x101		CM-RAM $_1$		CM-RAM $_3$	5
0x110			CM-RAM $_2$	CM-RAM $_3$	6
0x111		CM-RAM $_1$	CM-RAM $_2$	CM-RAM $_3$	7

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 *here*.

Code	Description
SRC	Contents of RP select a RAM or ROM address to be used by I/O and RAM instructions.
DCL	Select a particular RAM bank.

10.16.9 Io And Ram Instructions

WRM

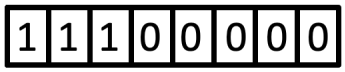
Name	Write accumulator into RAM character
Function	The accumulator content is written into the previously selected RAM main memory character location. The accumulator and carry/link are unaffected.
Syntax	WRM
Assembled	
Binary	11100000
Decimal	224
Hexadecimal	0xE0
Symbolic	(ACC) ⟹ M
Execution	1 word, 8-bit code and an execution time of 10.3 μ sec
Side-effects	Not Applicable
Implemented	wrm

Detailed 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.

The opcode for this instruction does not contain any additional data:



Example program

The example program will cause the 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).

```
/ Example program
        FIM   0P  180
        SRC   0P
        LDM   15
        WRM
```

WMP

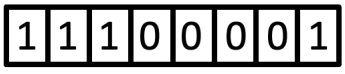
Name	Write RAM Port
Function	Write to a specified RAM port
Syntax	WMP
Assembled	
Binary	11100001
Decimal	225
Hexadecimal	0xE1
Symbolic	(ACC) ⟹ RAM Output Register
Execution	1 word, 8-bit code and an execution time of 10.3 μ sec
Side-effects	Not Applicable
Implemented	wmp

Detailed Description

The contents of the accumulator are written to the output port associated with the DATA RAM chip selected by the last SRC instruction. The data is available on the output pins until a new WMP is executed on the same RAM chip. The LSB bit of the accumultor appears on O0, (Pin 16), of the 4002.

The carry bit and the accumulator are unchanged.

The opcode for this instruction does not contain any additional data:



Example Program

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

```
/ Example program
      FIM         3P  64
      SRC         3P
      LDM         6
      WMP
```

WRR

Name	Write ROM Port
Function	Write to a specified ROM port
Syntax	WRR
Assembled	
Binary	11100010
Decimal	226
Hexadecimal	0xE2
Symbolic	(ACC) ➡ ROM Output Lines
Execution	1 word, 8-bit code and an execution time of 10.3 μ sec
Side-effects	Not Applicable
Implemented	wrr

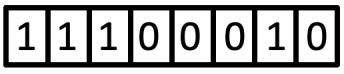
Detailed Description

The content of the accumulator is transferred to the ROM output port of the previously selected ROM chip. The data is available on the output pins until a new WRR is executed on the same chip. The LSB bit of the accumulator appears on I/O 0, (pin 16), of the 4001.

No operation is performed on I/O lines coded as inputs.

The carry bit and the accumulator are unchanged.

The opcode for this instruction does not contain any additional data:



Example Program

The example program will write the value 15 to the output port associated with the ROM chip 2.

```
/ Example program
      FIM       4P  64
      SRC       4P
      LDM       15
      WRR
```

WPM

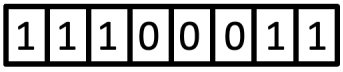
Name	Write accumulator into RAM character
Function	Read/Write half a byte to Program RAM from accumulator.
Syntax	WPM
Assembled	
Binary	11100011
Decimal	227
Hexadecimal	0xE3
Symbolic	(ACC) ➡ (PRAM)
Execution	1 word, 8-bit code and an execution time of 10.3 μ sec
Side-effects	Not Applicable
Implemented	wpm

Detailed 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 not affected.

The opcode for this instruction does not contain any additional data:



Notes

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 ounter 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.

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 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:

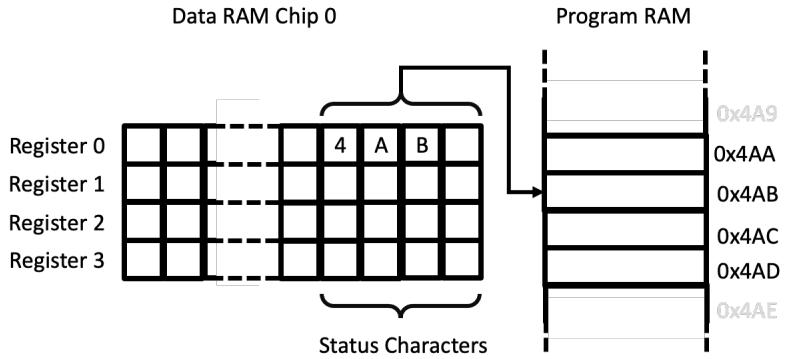
(1) The highest 4 bits of the program RAM address to be accessed must be written to ROM port 15.

(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 Program

The following program writes to 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, shown below.




```
/ Example program
      FIM       0P  180
      SRC       0P
      LDM       15
      WRM

      FIM       0P    224
      SRC       0P          / Select ROM port 14.
      LDM       1
      WRR                   / Turn on write enable.
```

(continues on next page)

(continued from previous page)

```
                        / Set up PRAM address.
                        /
    FIM         0P    0
    SRC         0P          / 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         0P    240
    SRC         0P          / Select ROM port 15.
    WRR                     / Write high address.
    SRC         5P          / Write middle + low address (RP5)
                        /
    LD          2           / High 4 data bits to accumulator.
    WPM                     / Write to PRAM
    LD          3           / Low 4 data bits to accumulator.
    WPM                     / Write to PRAM

    FIM         0P    224
    SRC         0P          / Select ROM port 14.
    CLB
    WRR                     / Turn off write enable.
```

WRn

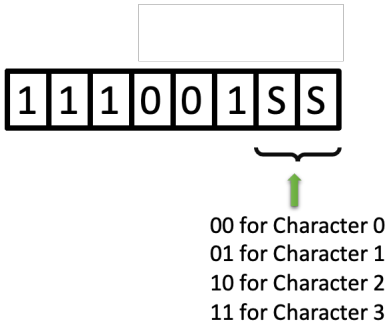
Name	Write Data Ram Status Character
Function	The content of the accumulator is written into the RAM status character n of the previously selected RAM register.
Syntax	WR0, WR1, WR2, WR3
Assembled	
Binary	11100100, 11100101, 11100110, 11100111,
Decimal	228, 229, 230, 231
Hexadecimal	0xE4, 0xE5, 0xE6, 0xE7
Symbolic	(ACC) ⟹ MSn
Execution	1 word, 8-bit code and an execution time of 10.3 μ sec
Side-effects	Not Applicable
Implemented	wrn

Detailed 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 tbe last SRC instruction, are replaced by the contents of the accumulator.

The carry bit and the accumulator are not affected.

The DATA RAM status character is encoded in the opcode as shown below:



Example program

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

```
/ Example program
FIM   0P  0
SRC   0P
LDM   2
WR1
```

RDM

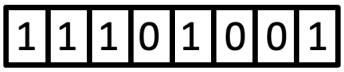
Name	Read RAM character
Function	The content of the previously selected RAM main memory character is transferred to the accumulator.
Syntax	WRM
Assembled	
Binary	11101001
Decimal	233
Hexadecimal	0xE9
Symbolic	(M) ➡ ACC
Execution	1 word, 8-bit code and an execution time of 10.3 μ sec
Side-effects	Not Applicable
Implemented	rdm

Detailed 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.

The opcode for this instruction does not contain any additional data:



Example program

The example 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.

```
/ Example program
        FIM   2P  5
        SRC   2P
        RDM
```

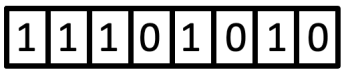
RDR

Name	Read ROM Port
Function	The data present at the input lines of the previously selected ROM chip is transferred to the accumulator.
Syntax	RDR
Assembled	
Binary	11101010
Decimal	234
Hexadecimal	0xEA
Symbolic	(ROM input lines) ➡ ACC
Execution	1 word, 8-bit code and an execution time of 10.8 μ sec
Side-effects	Not Applicable
Implemented	rdr

Detailed 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.

The opcode for this instruction does not contain any additional data:



Example

The following instructions will read the contents of the port associated with ROM number 10 into the accumulator.

```
/ Example
        FIM   3P    160
        SRC   3P
        RDR
```

The *rdr* operation above is carried out as follows:

```
   Accumulator    =   1 0 1 0
Data Character    =   0 1 1 1
Carry             =         0
                      -------
   Result      1      0 0 0 1
```

The accumulator contains 1 and the carry bit is set.

If the leftmost I/O line is an output line and the remaining I/O lines are input lines containing 010b, then the accumulator will contain either 1010b or 0010b.

I_3 O_2 O_1 I_0	(ACC)
1 X X 0 1	1 (1 or 0) (1 or 0) 0
ROM pins	

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.

Note

Whether a 0 or a 1 is transferred is a function of the hardware and not under control of the programmer. That is to say, when a 4001 ROM chip is ordered, it is required to determine **at that stage** what the functionality of the pins should be. Once ordered, the decision cannot be reverted. An order form can be downloaded `here`

RDn

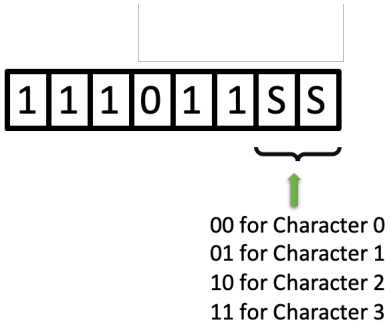
Name	Read Data from Ram Status Character
Function	The 4-bits of status character n for the previously selected RAM register are transferred to the accumulator.
Syntax	RD0, RD1, RD2, RD3
Assembled	
Binary	11101100, 11101101, 11101110, 11101111,
Decimal	236, 237, 238, 239
Hexadecimal	0xEC, 0xED, 0xEE, 0xEF
Symbolic	MSn ⟹ ACC
Execution	1 word, 8-bit code and an execution time of 10.3 μ sec
Side-effects	Not Applicable
Implemented	rdn

Detailed 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.

The DATA RAM status character is encoded in the opcode as shown below:



Example program

The example program 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.

```
/ Example program
FIM   2P  5
SRC   2P
RD3
```

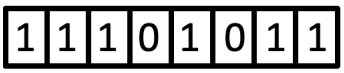
ADM

Name	Addd DATA RAM to accumulator with carry
Function	The content of the previously selected RAM main memory character is added to the accumulator with carry.
Syntax	ADM
Assembled	
Binary	11101011
Decimal	235
Hexadecimal	0xEB
Symbolic	(M) + (ACC) + (CY) ⟹ ACC, CY
Execution	1 word, 8-bit code and an execution time of 10.8 μ sec
Side-effects	Depending on the result, the carry bit is reset or set.
Implemented	*adm*

Detailed 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, otherwise. the data character is not affected.

The opcode for this instruction does not contain any additional data:



Example

In this example, the carry bit = 0, the accumulator contains a value of 10, and DATA RAM character 0 of register 0 of chip 0 contains 7.

```
/ Example
        FIM   0P    0
        SRC   0P
        ADM
```

The *adm* operation above is carried out as follows:

```
   Accumulator    =   1 0 1 0
Data Character    =   0 1 1 1
Carry             =         0
                      -------
   Result      1      0 0 0 1
```

The accumulator contains 1 and the carry bit is set.

SBM

Name	Subtract DATA RAM from memory with borrow
Function	The content of the previously selected RAM character is subtracted from the accumulator with borrow.
Syntax	SBM
Assembled	
Binary	11101000
Decimal	232
Hexadecimal	0xE8
Symbolic	$(\overline{M}) + (ACC) + (\overline{CY}) \Rightarrow ACC, CY$
Execution	1 word, 8-bit code and an execution time of 10.8 μ sec
Side-effects	Depending on the result, the carry bit is reset or set.
Implemented	sbm

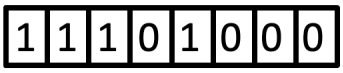
Detailed Description

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.

Notes

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 *"Decimal Subtraction"*:.

The opcode for this instruction does not contain any additional data:



Example

In order to perform a normal subtraction, the carry bit should be zero.

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

```
/ Example
        FIM   1P    1
        SRC   1P
        SBM
```

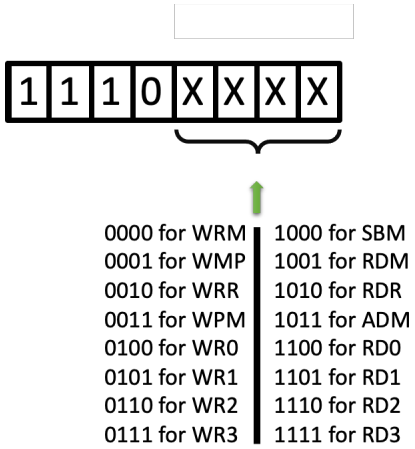
The *sbm* operation above is carried out as follows:

```
      Accumulator    =   0 1 1 1
~  Data Character    =   1 0 1 0     ( Character = 0 1 0 1)
~  Carry             =         0     ( carry = 1)
                       -------
      Result       0     0 0 0 1
              Carry indicates a borrow
```

The accumulator contains 1 and the carry bit is reset.

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.

Instructions in this class occupy one byte as follows:



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

This is a summary of 4004 instructions.

Abbreviations used are as follows:

Abbreviation	Description
A	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
$\overline{\text{value}}$	The number obtained by complementing each bit of "value".
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.

Table 1: Instruction Summary

Group	Definition
Index Register Instructions	Instructions which involve index registers or register pairs.
Index Register to Accumulator Instructions	Instructions which involve an operation between an index register and the accumulator. Instructions in this class occupy one byte.
Accumulator Instructions	Instructions which operate only on the contents of the accumulator and/or the carry bit. Instructions in this class occupy one byte.
Immediate Instructions	Instructions which use data that is part of the instruction itself.
Transfer Of Control Instructions	Instructions which alter the normal execution sequence of instructions.
Subroutine Linkage Instructions	Instructions which call and cause return from subroutines. They cause a transfer of program control and use the address stack.
No-Operation Instruction	This instruction occupies one byte.
Memory Selection Instructions	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
Input/Output and RAM Instructions	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.

10.17 Instruction Machine Codes

In order to help the programmer examine memory when debugging programs, this list 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.

Table 2: Instruction Machine Codes

Decimal	Octal	Hex	Mnemonic	Parameter	Comment
0	0	0	*NOP*		
1	1	1	Not Used		
2	2	2	Not Used		
3	3	3	Not Used		
4	4	4	Not Used		
5	5	5	Not Used		
6	6	6	Not Used		
7	7	7	Not Used		
8	10	8	Not Used		
9	11	9	Not Used		
10	12	A	Not Used		
11	13	B	Not Used		
12	14	C	Not Used		
13	15	D	Not Used		
14	16	E	Not Used		
15	17	F	Not Used		
16	20	10	*JCN*	0	CN=0
17	21	11	*JCN*	1	CN=1
18	22	12	*JCN*	2	CN=2
19	23	13	*JCN*	3	CN=3
20	24	14	*JCN*	4	CN=4
21	25	15	*JCN*	5	CN=5
22	26	16	*JCN*	6	CN=6
23	27	17	*JCN*	7	CN=7
24	30	18	*JCN*	8	CN=8
25	31	19	*JCN*	9	CN=9
26	32	1A	*JCN*	10	CN=10
27	33	1B	*JCN*	11	CN=11
28	34	1C	*JCN*	12	CN=12
29	35	1D	*JCN*	13	CN=13
30	36	1E	*JCN*	14	CN=14
31	37	1F	*JCN*	15	CN=15
32	40	20	*FIM*	0P	
33	41	21	*SRC*	0	
34	42	22	*FIM*	1P	
35	43	23	*SRC*	1	
36	44	24	*FIM*	2P	
37	45	25	*SRC*	2	
38	46	26	*FIM*	3P	
39	47	27	*SRC*	3	
40	50	28	*FIM*	4P	

continues on next page

Table 2 – continued from previous page

Decimal	Octal	Hex	Mnemonic	Parameter	Comment
41	51	29	*SRC*	4	
42	52	2A	*FIM*	5P	
43	53	2B	*SRC*	5	
44	54	2C	*FIM*	6P	
45	55	2D	*SRC*	6	
46	56	2E	*FIM*	7P	
47	57	2F	*SRC*	7	
48	60	30	*FIN*	0	
49	61	31	*JIN*	0	
50	62	32	*FIN*	1	
51	63	33	*JIN*	1	
52	64	34	*FIN*	2	
53	65	35	*JIN*	2	
54	66	36	*FIN*	3	
55	67	37	*JIN*	3	
56	70	38	*FIN*	4	
57	71	39	*JIN*	4	
58	72	3A	*FIN*	5	
59	73	3B	*JIN*	5	
60	74	3C	*FIN*	6	
61	75	3D	*JIN*	6	
62	76	3E	*FIN*	7	
63	77	3F	*JIN*	7	
64	100	40	*JUN*		ψ
65	101	41	*JUN*		ψ
66	102	42	*JUN*		ψ
67	103	43	*JUN*		ψ
68	104	44	*JUN*		ψ
69	105	45	*JUN*		ψ
70	106	46	*JUN*		ψ
71	107	47	*JUN*		ψ
72	110	48	*JUN*		ψ
73	111	49	*JUN*		ψ
74	112	4A	*JUN*		ψ
75	113	4B	*JUN*		ψ
76	114	4C	*JUN*		ψ
77	115	4D	*JUN*		ψ
78	116	4E	*JUN*		ψ
79	117	4F	*JUN*		ψ
80	120	50	*JMS*		ψ
81	121	51	*JMS*		ψ
82	122	52	*JMS*		ψ
83	123	53	*JMS*		ψ
84	124	54	*JMS*		ψ
85	125	55	*JMS*		ψ
86	126	56	*JMS*		ψ
87	127	57	*JMS*		ψ
88	130	58	*JMS*		ψ
89	131	59	*JMS*		ψ

continues on next page

Table 2 – continued from previous page

Decimal	Octal	Hex	Mnemonic	Parameter	Comment
90	132	5A	*JMS*		ψ
91	133	5B	*JMS*		ψ
92	134	5C	*JMS*		ψ
93	135	5D	*JMS*		ψ
94	136	5E	*JMS*		ψ
95	137	5F	*JMS*		ψ
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	
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	

continues on next page

Table 2 – continued from previous page

Decimal	Octal	Hex	Mnemonic	Parameter	Comment
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*		
161	241	A1	*LD*		
162	242	A2	*LD*		
163	243	A3	*LD*		
164	244	A4	*LD*		
165	245	A5	*LD*		
166	246	A6	*LD*		
167	247	A7	*LD*		
168	250	A8	*LD*		
169	251	A9	*LD*		
170	252	AA	*LD*		
171	253	AB	*LD*		
172	254	AC	*LD*		
173	255	AD	*LD*		
174	256	AE	*LD*		
175	257	AF	*LD*		
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	

continues on next page

Table 2 – continued from previous page

Decimal	Octal	Hex	Mnemonic	Parameter	Comment
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	
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*		

continues on next page

Table 2 – continued from previous page

Decimal	Octal	Hex	Mnemonic	Parameter	Comment
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*		
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	Not Used		
255	377	FF	Not Used		

ψ Second hexadecimal digit is part of the jump address.

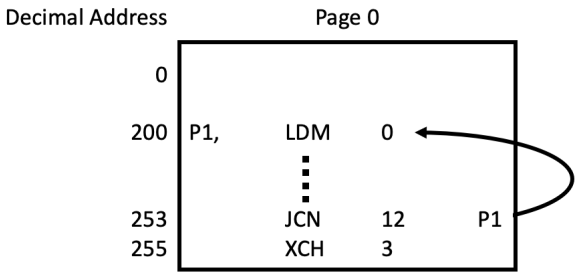
10.18 Programming Techniques

10.18.1 Crossing Page Boundaries

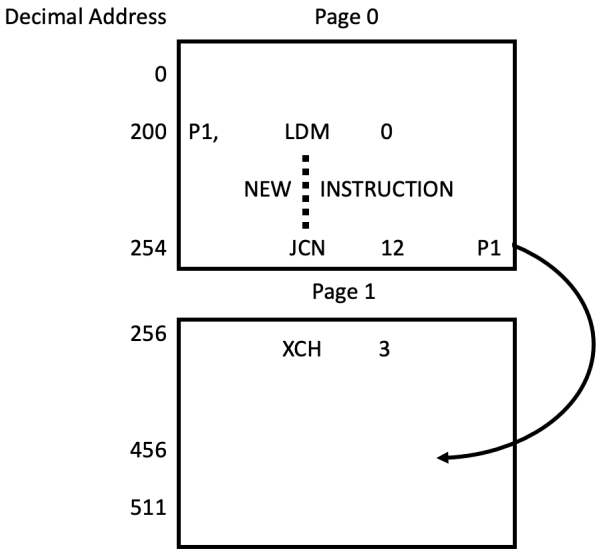
As described in Section 2, programs are held in either ROM or program RAM, both of which are divided into pages. Each page consists of 256 8 -bit locations. Addresses 0 through 255 comprise the first page,256-511 comprise the second page, and so on.

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:



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.

10.18.2 Subroutines

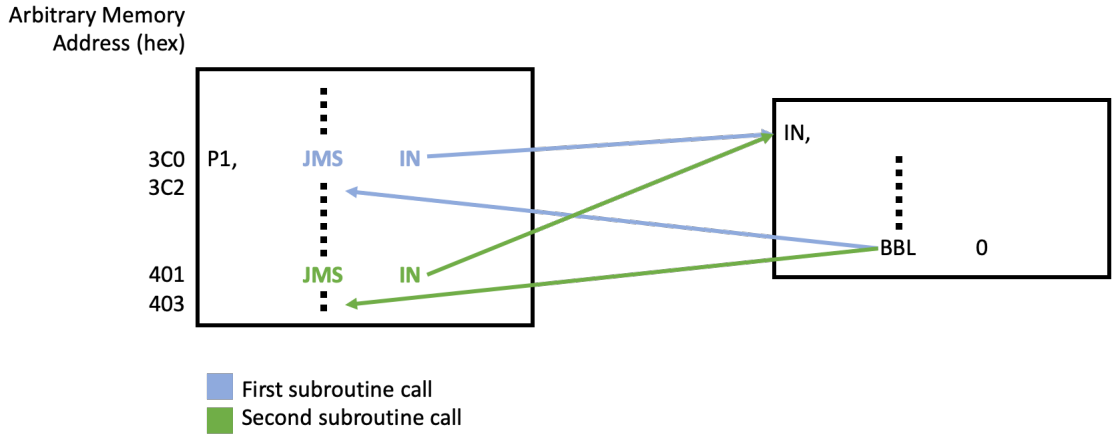
Frequently, a group of instructions must be repeated many times in a program. The group may be written "n" times if it is needed at "n" different points in a program, but better economy can be obtained by using subroutines. A subroutine is coded like any other group of assembly language statements, and 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:

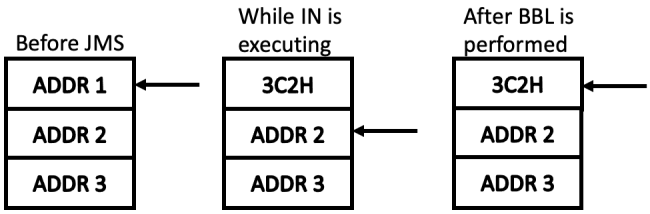
Subroutine IN increments an 8 bit number passed in index register 0 and 1 and then returns to the instruction following the last JMS instruction executed.

```
IN,     XCH     1   / Register 1 to accumulator
        IAC         / Increment value and produce carry
        XCH     1   / Restore register 1
        JCN 10  NC  / Jump if Carry is zero
        INC     0   / Increment high order 4 bits
NC,     BBL     0   / returns
```

Assume IN appears as follows:



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).



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 BBLs will transfer program control to incorrect addresses.

10.18.3 Branch Table Pseudosubroutines

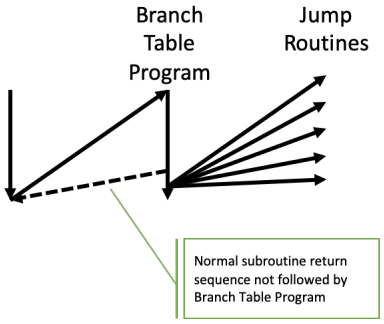
Suppose a program consists of several separate routines, any of which may be executed depending upon some initial condition (such as a bit set in the accumulator). One way to code this would be to check each condition sequentially and branch to the routines accordingly as follows:

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

A program that provides the above logic is given below. 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.




```
ST,     KBP             / Convert Accum to branch table index
        IAC             / If accumulator = 1111, Error
        JCN 4   ERR     / Jump if IAC produced zero
        DAC             / OK, restore accumulator
        FIM 0   BTL     / Registers 0 and 1 are the address of the branch table
        CLC             / Clear Carry
```

(continues on next page)

(continued from previous page)

```
        ADD 1           / Add index to the branch table address
        XCH 1           / Store back in register 1
        JCN 10  NC      / Jump if no carry
        INC 0           / If carry, increment register 0
NC,     FIN 0P          / Registers 0 and 1 (address of routine)
        JIN 0P          / Jump to correct routine
        .
        .
        .
        .
BTL,    0 + RT0         / Branch table.
        0 + RT1         / Each entry is an 8-bit address
        0 + RT2         / of the specific routine to call
        0 + RT3
        0 + RT4
        .
        .
        .
        .
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 01OO when location ST was reached, the KBP would convert it to 0011. 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.

10.18.4 Logical Operations

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 = 1110 and register 1 = 0011, register 0 will be replaced with 0010.

```
        1 1 1 0
AND     0 0 1 1
        -------
        0 0 1 0
```

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 right-most 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.

```
AND,    FIM     1P  11      / Register 2 = 0, Register 3 = 11
L1,     LDM     0           / Get bit of Register 0, Set accumulator = 0
        XCH     0           / Set Register 0 to accumulator; Register 0 = 0
        RAL                 / Move first 'AND' bit to carry
        XCH     0           / Save shifted data in Register 0, set accumulator = 0
        INC     3           / Done if Register 3 = 0
        XCH     3           / Register 3 to accumulator
        JCN     4   L2      / Return if accumulator = 0
        XCH     3           / Otherwise, restore accumulator and Register 3
        RAR                 / Bit of Register 0 is alone in the accumulator
        XCH     2           / Save first 'AND' bit in Register 2
        XCH     1           / Get bit of Register 1
        RAL                 / Move leftmost bit to carry
        XCH     1           / Save shifted data in Register 1
        RAR                 / Move second 'AND' bit to carry
        ADD     2           / 'ADD' gives the 'AND' of the bits in carry
        JUN     L1
L2,     BBL     0           / Return to main program
```

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 is set to 0100 and register 1 to 0011, register 0 will be replaced with 0111.

```
        0 1 0 0
AND     0 0 1 1
        -------
        0 1 1 1
```

The subroutine produces the OR of two bits by placing the bits in the leftmost position of the accumuiator 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.

```
OR,     FIM     1P  11      / Register 2 = 0, Register 3 = 11
L1,     LDM     0           / Get bit of Register 0, Set accumulator = 0
        XCH     0           / Set Register 0 to accumulator; Register 0 = 0
        RAL                 / Move first 'OR' bit to carry
        XCH     0           / Save shifted data in Register 0, set accumulator = 0
        INC     3           / Done if Register 3 = 0
        XCH     3           / Register 3 to accumulator
        JCN     4   L2      / Return if accumulator = 0
        XCH     3           / Otherwise, restore accumulator and Register 3
        RAR                 / Bit of Register 0 is alone in the accumulator
        XCH     2           / Save first 'OR' bit in Register 2
        LDM     0           / Get bit in Register 1, set accumulator = 0
        XCH     1           / Get bit of Register 1
        RAL                 / Move leftmost bit to carry
        XCH     1           / Save shifted data in Register 1
        RAR                 / Move second 'OR' bit to carry
        ADD     2           / 'ADD' gives the 'OR' of the bits in carry
        JCN     2   L1      / Jump if carry = 1 because 'OR' = 1
        RAL                 / Otherwise, 'OR' leftmost bit of the accumulator,
                            / transmit to carry by RAL
        JUN     L1
L2,     BBL     0           / Return to main program
```

Logical XOR

The XOR 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 = 0011 and register 1 = 0010, register 0 will be replaced with 0001.

```
        0 0 1 1
AND     0 0 1 0
        -------
        0 0 0 1
```

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.

```
XOR,    FIM     1P  11      / Register 2 = 0, Register 3 = 11
L1,     LDM     0           / Get bit of Register 0, Set accumulator = 0
        XCH     0           / Set Register 0 to accumulator; Register 0 = 0
        RAL                 / Move first 'XOR' bit to carry
        XCH     0           / Save shifted data in Register 0, set accumulator = 0
        INC     3           / Done if Register 3 = 0
        XCH     3           / Register 3 to accumulator
        JCN     4   L2      / Return if accumulator = 0
        XCH     3           / Otherwise, restore accumulator and Register 3
        RAR                 / Bit of Register 0 is alone in the accumulator
        XCH     2           / Save first 'XOR' bit in Register 2
        LDM     0           / Get bit in Register 1, set accumulator = 0
        XCH     1
        RAL                 / Move leftmost bit to carry
        XCH     1           / Save shifted data in Register 1
        RAR                 / Move second 'XOR' bit to carry
        ADD     2           / 'ADD' gives the 'XOR' of the bits in carry
        RAL                 / Otherwise, 'XOR' leftmost bit of the accumulator,
                            / transmit to carry by RAL
        JUN     L1
L2,     BBL     0           / Return to main program
```

There are three subroutines which produce basic logical operations:

- *AND*
- *OR*
- *XOR (eXclusive OR)*

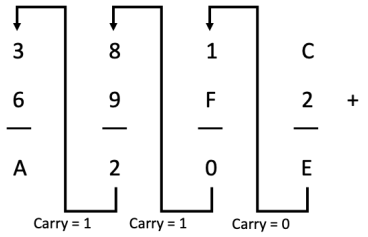
10.18.5 Multi-Digit Addition

The carry bit may be used to add unsigned data quantities of arbitrary length.

Consider the following addition of two 4-digit hexadecimal numbers:

```
3 8 1 C
6 9 F 2 +
-------
A 2 0 E
```

This addition may be performed by setting the carry bit = 1, then 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.

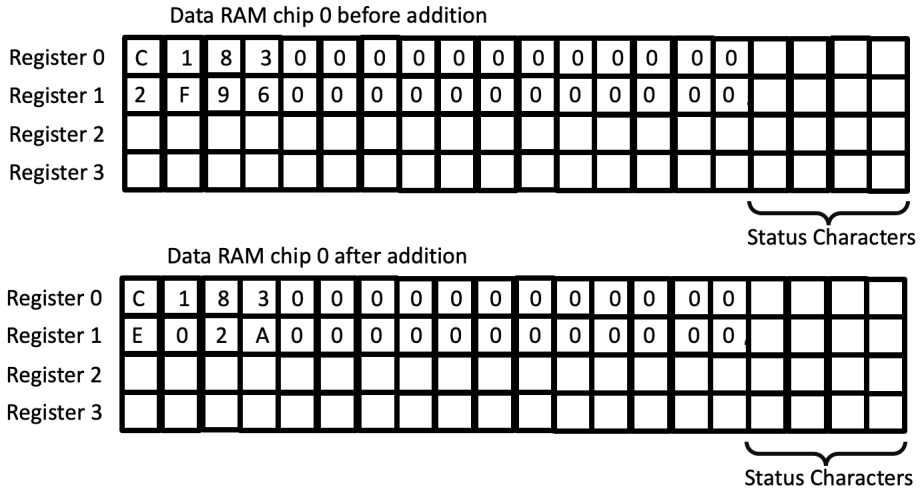
```
          9  =   1 0 0 1
          9  =   1 0 0 1
      Carry  =         0
               ---------
Result           0 0 1 0
Carry       1
```

```
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 ACCUMULATOR
      WRM                / WRITE RESULT TO REG 1
      INC   5            / ADDRESS NEXT CHARACTER OF RAM REG 0
      INC   7            / ADDRESS NEXT CHARACTER 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 repititions of the loop, index registers 5 and 7 are incremented, referenecing sequential data characters, until all 16 digits have been added.



A variant of the subroutine is below - this time for an arbitary number of 16 digit numbers. The only difference is the addition of an DAA instruction.

```
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               / SELECT RAM REG 1
      ADM               / ADD DIGIT + CARRY TO ACCUMULATOR
      DAA               / ADJUST FOR DECIMAL
      WRM               / WRITE RESULT TO REG 1
      INC   5           / ADDRESS NEXT CHARACTER OF RAM REG 0
      INC   7           / ADDRESS NEXT CHARACTER OF RAM REG 1
      ISZ   8  AD1      / BRANCH IF DIGIT COUNTER < 16 (NON ZERO)
OVR,  BBL   0
```

10.18.6 Multi-Digit Subtraction

The carry bit may be used to add unsigned data quantities of arbitrary length.

Consider the following subtraction of two 4-digit hexadecimal numbers:

```
5 4 B A
1 4 F 6 -
-------
3 F C 4
```

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.

The process as applied to the above subtraction is as follows:

(1) Set carry bit = 1

(2) Complement carry bit (carry is now 0)

(3) Subtract low order digits

```
     A      1 0 1 0
    ~6      1 0 0 1
~carry             1
           -------
     1     0 1 0 0      =      0x04
  carry
```

(4) Complement resulting carry bit (carry is now 0)

(5) Subtract next digits

```
     B      1 0 1 0
    ~F      0 0 0 0
~carry            1
           -------
     0      1 1 0 0      =      0x0C
  carry
```

(6) Complement resulting carry bit (carry is now 1)

(7) Subtract next digits

```
     4      0 1 0 0
    ~4      1 0 1 1
~carry            0
           -------
     0      1 1 1 1      =      0x0F
  carry
```

(8) Complement resulting carry bit (carry is now 1)

(9) Subtract next digits

```
     5      0 1 0 1
    ~1      1 1 1 0
~carry            0
           -------
     1      0 0 1 1      =      0x03
  carry
```

Thus, the correct result (0x3FC4) is obtained.

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

- The two numbers to be subtracted are stored in DATA RAM chip 0, registers 0 and 1.
- Register 1 contains the subtrahend.
- 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 subtracted.

```
SB,   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
      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 + CARRY FROM ACCUMULATOR
      WRM               / WRITE RESULT TO REG 1
```

(continues on next page)

(continued from previous page)

```
      INC   5           / ADDRESS NEXT CHARACTER OF RAM REG 0
      INC   7           / ADDRESS NEXT CHARACTER 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 arranged from low order digit to high order digit.

Note: Carry Bit

The carry bit from the previous subtraction is complemented by the **CMC** instruction each time through the loop.

10.18.7 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 perfonning 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.

Example: Perform the decimal addition

```
4 6 9
3 2 9 +
  ---
7 9 8
```

1 Clear the carry and add the lowest-order digits

```
          9  =   1 0 0 1
          9  =   1 0 0 1
     Carry   =         0
               ---------
Result           0 0 1 0
Carry      1
```

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 (i.e. 1)

```
 Accumulator  =   0 0 1 0
          6   =   0 1 1 0
     Carry    =         0
               ---------
```

(continues on next page)

(continued from previous page)

```
Result            1 0 0 0 = 8
Carry       1

(since the DAA produced no carry, the bit is unaffected)
```

3 Add the next two digits

```
          6  =   0 1 1 0
          2  =   0 0 1 0
     Carry   =         1
               ---------
Result            1 0 0 1 = 9
Carry       0
```

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

5 Add the next two digits

```
          4  =   0 1 0 0
          3  =   0 0 1 1
     Carry   =         0
               ---------
Result            0 1 1 1 = 7
Carry       0
```

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

Example Code (subroutine)

A subroutine which adds two 16 digit decimal numbers can be found *here*:

10.18.8 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. 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 to 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 to 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.

7 Save this resuIt.

8 If there are more digits to subtract, goto step 2, otherwise stop.

Example: Perform the decimal subtraction

```
5 1
3 8 -
---
1 3
```

1 Set the carry bit to 1

2 TCS sets accumulator = 1010b and carry = O

3 Subtract the subtrahend digit 8 from the accumulator

```
Accumulator    =   1 0 1 0
        ~ 8    =   0 1 1 1
    ~ Carry    =         1
                 ---------
Result             0 0 1 0
```

4 Set the carry bit to 0

5 Add minuend digit 1 to accumulator

```
Accumulator    =   0 0 1 0
          1    =   0 1 1 1
      Carry    =         0
                 ---------
Result             0 0 1 1
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

```
Accumulator    =   1 0 0 1
        ~ 3    =   1 1 0 0
    ~ Carry    =         1
                 ---------
Result             0 1 1 0
```

9 Set carry = 0

10 Add minuend digit 5 to accumulator

```
Accumulator    =   0 1 1 0
          5    =   0 1 0 1
```

(continues on next page)

(continued from previous page)

```
      Carry    =          0
                 ---------
Result             1 0 1 1
Carry          0
```

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

```
Accumulator    =    1 0 1 1
          6    =    0 1 1 0
                 ---------
Result             0 0 0 1
Carry          1
```

Therefore the result of subtracting 38 from 51 is 13.

Example Code (subroutine)

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 O.
- 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

        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 ACCUMULATOR
        DAA                     / ADJUST ACCUMULATOR
        WRM                     / WRITE RESULT TO REG 0
        INC     5               / ADDRESS NEXT CHARACTER OF RAM REG 0
        INC     7               / ADDRESS NEXT CHARACTER OF RAM REG 1
        ISZ     8       SD1     / BRANCH IF DIGIT COUNTER < 16 (NON-ZERO)
DN      BBL     0
```

Glossary

Term	Definition
minuend	The number that the subtrahend is being subtracted from
subtrahend	The number that is being subtracted

10.18.9 Floating 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:

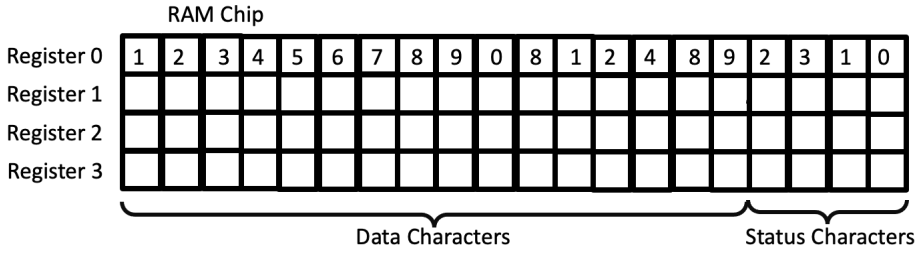
$\pm$ .DDDDDDDDDDDDDDDD * 10 $^{\pm \text{ EE}}$

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

$\pm$ .1234567890812489 * 10 $^{-23}$

would appear in a RAM register as follows:



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

> *Crossing Page Boundaries Subroutines Branch Table Pseudosubroutines Logical Operations Floating Point Numbers Crossing Page Boundaries* ***Multi Digit Subtraction Decimal Addition Decimal Subtraction***

10.19 Powers Of Two

2^n	n	2^{-n}
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	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 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	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 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
8 796 093 022 208	43	0.000 000 000 000 113 686 837 721 616 029 739 379 882 812 5
17 592 186 044 416	44	0.000 000 000 000 056 843 418 860 808 014 869 689 941 406 25

continues on next page

Table 3 – continued from previous page

2^n	n	2^{-n}
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
1 152 921 504 606 846 976	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	0.000 000 000 000 000 000 216 840 434 497 100 886 801 490 560 173 988 342 285 156 25
9 223 372 036 854 775 808	63	0.000 000 000 000 000 000 108 420 217 248 550 443 400 745 280 086 994 171 142 578 125

10.20 Powers Of Sixteen

16^n	n	16^{-n}
1	0	$1.000\ 00000\ 00000\ 00000 \times 10^{0}$
16	1	$0.625\ 00000\ 00000\ 00000 \times 10^{-1}$
256	2	$0.390\ 62500\ 00000\ 00000 \times 10^{-2}$
4 096	3	$0.244\ 14062\ 50000\ 00000 \times 10^{-3}$
65 536	4	$0.152\ 58789\ 06250\ 00000 \times 10^{-4}$
1 048 576	5	$0.953\ 67431\ 64062\ 50000 \times 10^{-6}$
16 777 216	6	$0.596\ 04644\ 77539\ 06250 \times 10^{-7}$
268 435 456	7	$0.372\ 52902\ 98461\ 91406 \times 10^{-8}$
4 294 967 296	8	$0.232\ 83064\ 36538\ 69628 \times 10^{-9}$
68 719 476 736	9	$0.145\ 51915\ 22836\ 68518 \times 10^{-10}$
1 099 511 627 776	10	$0.909\ 49470\ 17729\ 28237 \times 10^{-12}$
17 592 186 044 416	11	$0.568\ 43418\ 86080\ 80148 \times 10^{-13}$
281 474 976 710 656	12	$0.355\ 27136\ 78800\ 50092 \times 10^{-14}$
4 503 599 627 370 496	13	$0.222\ 04460\ 49250\ 31308 \times 10^{-15}$
72 057 594 037 927 936	14	$0.138\ 77787\ 80781\ 44567 \times 10^{-16}$
1 152 921 504 606 846 976	15	$0.867\ 36173\ 79884\ 03547 \times 10^{-18}$

10.21 Powers Of 10 $_{16}$

10^{n}	n	10^{-n}
1	0	1.0000 0000 0000 0000
A	1	0.1999 9999 9999 999A
64	2	0.28F5 C285 5C28 F5C3 x 16^{-1}
3E8	3	0.4189 374B C6A7 EF9E x 16^{-2}
2710	4	0.68DB 8BAC 710c b296 x 16^{-3}
1 86A0	5	0.A7C5 AC47 1B47 8423 x 16^{-4}
F 4240	6	0.10C6 F7A0 B5ED 9D37 x 16^{-4}
98 9680	7	0.1A07 F29A BCAF 4858 x 16^{-5}
5F5 E100	8	0.2AF3 1DC4 6118 73BF x 16^{-6}
3B9A CA00	9	0.44B8 2FA0 9B5A 52CC x 16^{-7}
2 540B E400	10	0.6DF3 7F67 5EF6 EADF x 16^{-8}
17 4876 E800	11	0.AFEB FF0B CB24 AAFF x 16^{-9}
E8 D4A5 1000	12	0.1197 9981 2DEA 1119 x 16^{-9}
918 4E72 A000	13	0.1C25 C268 4976 81C2 x 16^{-10}
5AF3 107A 4000	14	0.2D09 370D 4257 3604 x 16^{-11}
3 8D7E A4C6 8000	15	0.480E BE7B 9D58 566D x 16^{-12}
23 86F2 6FC1 0000	16	0.734A CASF 6226 F0AE x 16^{-13}
163 4578 5D8A 0000	17	0.B877 AA32 36A4 B449 x 16^{-14}
DE0 B6B3 A764 0000	18	0.1272 5DD1 D243 ABA1 x 16^{-14}
8AC7 2304 89E8 0000	19	0.1D83 C94F B6D2 AC35 x 16^{-15}

10.22 Hexadecimal Decimal Integer Conversion

The table below provldes for direct conversions between hexadecimal integers in the range O-FFF and decimal integers in the range 0-4095.

For conversion of larger integers, the table values may be added to the following figures:

Hex	Decimal	Hex	Decimal
1 000	4 096	20 000	131 072
2 000	8 192	30 000	196 608
3 000	12 288	40 000	262 144
4 000	16 384	50 000	327 680
5 000	20 480	60 000	393 216
6 000	24 576	70 000	458 752
7 000	28 672	80 000	524 288
8 000	32 768	90 000	589 824
9 000	36 864	A0 000	655 360
A 000	40 960	B0 000	720 896
B 000	45 056	C0 000	786 432

continues on next page

Table 4 – continued from previous page

Hex	Decimal	Hex	Decimal
C 000	49 152	D0 000	851 968
D 000	53 248	E0 000	917 504
E 000	57 344	F0 000	983 040
F 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	13 631 408
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	000	001	002	003	004	005	006	007	008	009	010	011	012	013	014	015
010	016	017	018	019	020	021	022	023	024	025	026	027	028	029	030	031
020	032	033	034	035	036	037	038	039	040	041	042	043	044	045	046	047
030	048	049	050	051	052	053	054	055	056	057	058	059	060	061	062	063
040	064	065	066	067	068	069	070	071	072	073	074	075	076	077	078	079
050	080	081	082	083	084	085	086	087	088	089	090	091	092	093	094	095
060	096	097	098	099	100	101	102	103	104	105	106	107	108	109	110	111
070	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
080	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
090	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
0A0	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
0B0	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
0C0	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
0D0	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
0E0	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
0F0	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255
100	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271
110	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287
120	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303
130	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319
140	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335
150	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351
160	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367
170	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383
180	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399
190	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415

continues on next page

Table 5 – continued from previous page

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1A0	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431
1B0	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447
1C0	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463
1D0	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479
1E0	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495
1F0	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511
200	512	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527
210	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543
220	544	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559
230	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575
240	576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591
250	592	593	594	595	596	597	598	599	600	601	602	603	604	605	606	607
260	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623
270	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639
280	640	641	642	643	644	645	646	647	648	649	650	651	652	653	654	655
290	656	657	658	659	660	661	662	663	664	665	666	667	668	669	670	671
2A0	672	673	674	675	676	677	678	679	680	681	682	683	684	685	686	687
2B0	688	689	690	691	692	693	694	695	696	697	698	699	700	701	702	703
2C0	704	705	706	707	708	709	710	711	712	713	714	715	716	717	718	719
2D0	720	721	722	723	724	725	726	727	728	729	730	731	732	733	734	735
2E0	736	737	738	739	740	741	742	743	744	745	746	747	748	749	750	751
2F0	752	753	754	755	756	757	758	759	760	761	762	763	764	765	766	767
300	768	769	770	771	772	773	774	775	776	777	778	779	780	781	782	783
310	784	785	786	787	788	789	790	791	792	793	794	795	796	797	798	799
320	800	801	802	803	804	805	806	807	808	809	810	811	812	813	814	815
330	816	817	818	819	820	821	822	823	824	825	826	827	828	829	830	831
340	832	833	834	835	836	837	838	839	840	841	842	843	844	845	846	847
350	848	849	850	851	852	853	854	855	856	857	858	859	860	861	862	863
360	864	865	866	867	868	869	870	871	872	873	874	875	876	877	878	879
370	880	881	882	883	884	885	886	887	888	889	890	891	892	893	894	895
380	896	897	898	899	900	901	902	903	904	905	906	907	908	909	910	911
390	912	913	914	915	916	917	918	919	920	921	922	923	924	925	926	927
3A0	928	929	930	931	932	933	934	935	936	937	938	939	940	941	942	943
3B0	944	945	946	947	948	949	950	951	952	953	954	955	956	957	958	959
3C0	960	961	962	963	964	965	966	967	968	969	970	971	972	973	974	975
3D0	976	977	978	979	980	981	982	983	984	985	986	987	988	989	990	991
3E0	992	993	994	995	996	997	998	999	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
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

continues on next page

Table 5 – continued from previous page

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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
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

continues on next page

Table 5 – continued from previous page

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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
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

continues on next page

Table 5 – continued from previous page

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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
BD0	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
BF0	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
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

continues on next page

Table 5 – continued from previous page

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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

10.23 Hexadecimal Arithmetic

Table 6: Hexadecimal Addition

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

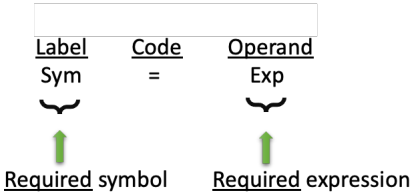
Table 7: Hexadecimal Multiplication

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	1B	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	5B	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	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

10.24 Pseudo Instructions

10.24.1 Equate

The "equate" pseudo instruction is indicated by the character **=** (equals 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.



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.

The statements

```
CZ   =    10
JCN CZ   ADDR
```

are equivalent to the statement

```
JCN 10   ADDR
```

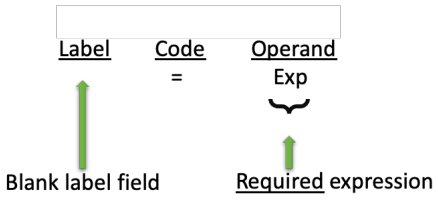
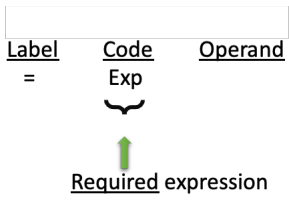
The statements

```
DAT =    5
    LDM DAT
```

will load the value 5 into the accumulator

10.24.2 Origin

Two forms of the instruction are acceptable:

Form 1	Form 2
	

As shown above, the equals sign may appear in the "label" or the "code" field.

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'.

```
Label   Code    Operand
=       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 !!!!

There are two pseudo instructions which recognised by the assembler:

Pseudo Instruction	Description
Equate	Assign a label to an expression.
Origin	Determine where the next instruction will be located.

CHAPTER

ELEVEN

INTEL 4004 OP-CODES

Table 1: Intel 4004 processor Op-Codes

Instruction	Mnemonic	1st byte	2nd byte	Modifiers
No Operation	*NOP*	00000000		
Jump Conditional	*JCN*	0001CCCC	AAAAAAAA	C, A
Fetch Immediate	*FIM*	0010RRR0	DDDDDDDD	RP, D
Send Register Control	*SRC*	0010RRR1		RP
Fetch Indirect	*FIN*	0011RRR0		RP
Jump Indirect	*JIN*	0011RRR1		RP
Jump Unconditional	*JUN*	0100AAAA	AAAAAAAA	A
Jump to Subroutine	*SRC*	0101AAAA	AAAAAAAA	A
Increment	*INC*	0110RRRR		R
Increment and Skip	*ISZ*	0111RRRR	AAAAAAAA	R, A
Add	*ADD*	1000RRRR		R
Subtract	*SUB*	1001RRRR		R
Load	*LD*	1010RRRR		R
Exchange	*XCH*	1011RRRR		R
Branch Back and Load	*BBL*	1100DDDD		D
Load Immediate	*LDM*	1101DDDD		D
Write Main Memory	*WRM*	11100000		
Write RAM Port	*WMP*	11100001		
Write Program RAM	*WPM*	11100011		
Write ROM Port	*WRR*	11100010		
Write Status Char 0	*WR0*	11100100		
Write Status Char 1	*WR1*	11100101		
Write Status Char 2	*WR2*	11100110		
Write Status Char 3	*WR3*	11100111		
Subtract Main Memory	*SBM*	11101000		
Read Main Memory	*RDM*	11101001		
Read ROM Port	*RDR*	11101010		
Add Main Memory	*ADM*	11101011		
Read Status Char 0	*RD0*	11101100		
Read Status Char 1	*RD1*	11101101		
Read Status Char 2	*RD2*	11101110		
Read Status Char 3	*RD3*	11101111		
Clear Both	*CLB*	11110000		
Clear Carry	*CLC*	11110001		
Increment Accumulator	*IAC*	11110010		
Complement Carry	*CMC*	11110011		

continues on next page

Table 1 – continued from previous page

Instruction	Mnemonic	1st byte	2nd byte	Modifiers
Complement Accumulator	*CMA*	11110100		
Rotate Left	*RAL*	11110101		
Rotate Right	*RAR*	11110110		
Transfer Carry and Clear	*TCC*	11110111		
Decrement Accumulator	*DAC*	11111000		
Transfer Carry Subtract	*TCS*	11111001		
Set Carry	*STC*	11111010		
Decimal Adjust Accumulator	*DAA*	11111011		
Keyboard Process	*KBP*	11111100		
Designate Command Line	*DCL*	11111101		

Note: Modifiers

- A = Address
- C = Condition
- D = Data
- R = Register
- RP = Register Pair

CHAPTER

TWELVE

THE 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.

Table 1: ASCII table

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
S0	0E
S1	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
83	1B
S4	1C
S5	1D

continues on next page

Table 1 – continued from previous page

Graphic or Control	ASCII (Hexadecimal)
S6	1E
S7	1F
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
A	41
B	42
C	43
D	44

continues on next page

Table 1 – continued from previous page

Graphic or Control	ASCII (Hexadecimal)
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

CHAPTER

THIRTEEN

INDICES AND TABLES

- genindex
- modindex
- search