Rev 9-28-21

ASSEMBLY Programming for

MCS-8 + i8086

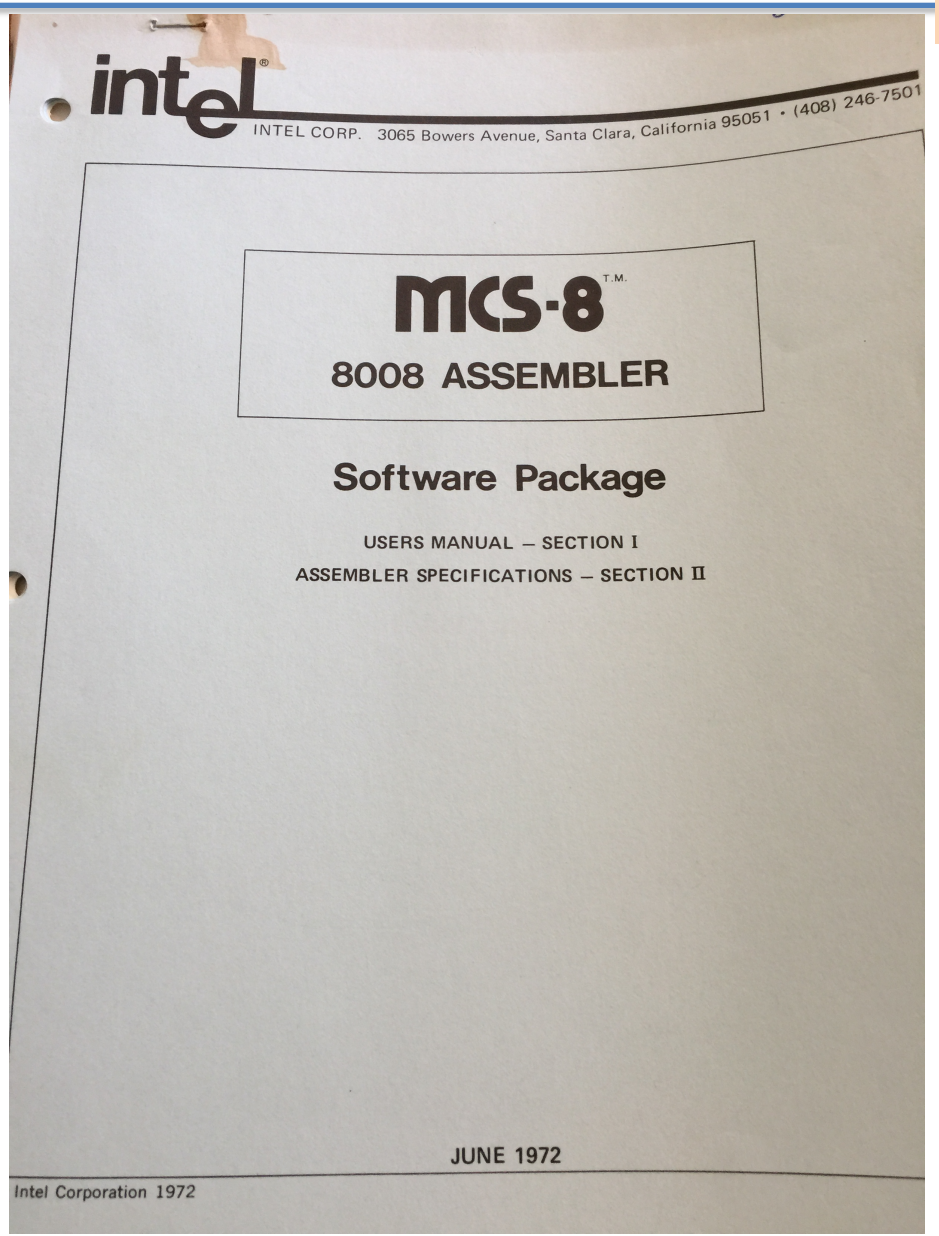
Dr Jeff Drobman

website drjeffsoftware.com/classroom.html

email jeffrey.drobman@csun.edu

i8008 Assembler

MCS-8



intel®

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

MCS-8™

8008 ASSEMBLER

Software Package

USERS MANUAL — SECTION I

ASSEMBLER SPECIFICATIONS — SECTION II

JUNE 1972

Intel Corporation 1972

Assembly Language

Examples of legal symbols:

```
MAT     START2
MIKE    Z148
TED24   RONA3Z
*
```

Numeric Constants

Two types of numeric constants are recognized by the Assembler: decimal and octal. A decimal number is represented by one to five digits (0-9) within the range of 0 to 16383. An octal number contains from one to five digits (0-7) followed by the letter B. The range of octal numbers is 0 to 37777B.

Copy Data Table

8080/8085

```
; memcpy --
; Copy a block of memory from one location to another.
;
; Entry registers
;       BC - Number of bytes to copy
;       DE - Address of source data block
;       HL - Address of target data block
;
; Return registers
;       BC - Zero

            org     1000h       ;Origin at 1000h
memcpy      public
loop        mov     a,b         ;Test BC,
            ora     c           ;If BC = 0,
            rz                  ;Return
            ldax    d           ;Load A from (DE)
            mov     m,a         ;Store A into (HL)
            inx     d           ;Increment DE
            inx     h           ;Increment HL
            dcx     b           ;Decrement BC
            jmp     loop        ;Repeat the loop
            end
```

6800

```
; memcpy --
; Copy a block of memory from one location to another.
;
; Entry parameters
;       cnt - Number of bytes to copy
;       src - Address of source data block
;       dst - Address of target data block

cnt         dw      $0000
src         dw      $0000
dst         dw      $0000

memcpy      public
            ldab    cnt+1       ;Set B = cnt.L
            beq     check       ;If cnt.L=0, goto check
loop        ldx     src         ;Set IX = src
            ldaa    ix          ;Load A from (src)
            inx                 ;Set src = src+1
            stx     src
            ldx     dst         ;Set IX = dst
            staa    ix          ;Store A to (dst)
            inx                 ;Set dst = dst+1
            stx     dst
            decb                ;Decr B
            bne     loop        ;Repeat the loop
            stab    cnt+1       ;Set cnt.L = 0
check       tst     cnt+0       ;If cnt.H=0,
            beq     done        ;Then quit
            dec     cnt+0       ;Decr cnt.H
            decb                ;Decr B
            bra     loop        ;Repeat the loop
done        rts                 ;Return
```

Boot Code Listing

MCS-8

```
LOC     OBJECT CODE      SOURCE STATEMENTS
0                        *TTY TAPE READER CONTROL
0                        *PROGRAMMER: DR. PHIL TAI, MCS, INTEL CORP.
0                        *PROGRAM: A0800-00
0                        *DATE: MAY 22,1972
0                                ORC 0
0    6    1              BEGIN  LAI 1                    SUPPRESS TTY
2   85                           OUT 12B                 OUTPUT 2
3  168                           XRA                     CLEAR AC
4   87                           OUT 13B                 OUTPUT 3 - TAPE R
                         EADER CONTROL
5   70   11    0                 CAL TAPE                CALL FOR TAPE REA
                         DER CONT. RT.
8   68    0    0                 JMP BEGIN
11   6    1              TAPE   LAI 1                    TAPE READER ENABL
                         E CODE
13  87                           OUT 13B                 OUTPUT 3 - ENABLE
                          TAPE READER
14  70   58    0                 CAL TTYD1               TAPE READER CONTR
                         OL DELAY
17   0                   TTY    HLT                     WAIT FOR TTY STAR
                         T PULSE
18  70   65    0                 CAL TTYD2               TTY DELAY - 4.468
                          MSEC.
21 168                           XRA                     TAPE READER DISAB
                         LE CODE
22  87                           OUT 13B                 OUTPUT 3, DISABLE
                          TAPE READER
23  65                           INP 0B                  INPUT 0, READ STA
                         RT PULSE
24  22 255                       LCI 255                 COMPLEMENT TTY ST
                         ART PULSE
26 170                           XRC                     EXCLUSIVE-OR REC.
                          C
27  85                           OUT 12B                 OUTPUT 2, OUTPUT
```

```
                         START PULSE
28  38 248                       LEI 248                 TTY DATA SAMPLING
                          COUNTER
30  70   58    0          TTYIN  CAL TTYD1               TTY DELAY - 9.012
                          MSEC.
33  65                           INP 0B                  READ TTY DATA INP
                         UT
34  22 255                       LCI 255                 COMPLEMENT TTY DA
                         TA
36 170                           XRC
37  85                           OUT 12B                 OUTPUT 2, TTY DAT
                         A OUT
38  26                           RAR                     STORE TTY DATA
39 193                           LAB                     LOAD TTY DATA TO
                         REC. B
40  26                           RAR
41 200                           LBA                     LOAD AC TO REC. B
42  32                           INE                     E = E + 1
43  72   30    0                 JFZ TTYIN               JUMP IF ZERO F/F
                         IS NOT SET
46 193                           LAB                     LOAD REC. B TO AC
47  83                           OUT 11B                 OUTPUT 1, TTY CHA
                         RACTER
48  20 128                       SUI 128                 REMOVE PARITY BIT
50 200                           LBA                     STORE TTY INPUT D
```

i8080 Code: RAM Test

1972 MCS-8

```
*1101 RAM TEST
*PROGRAM: A0802
*DATE: MAY 8,1972
*PROGRAMMER: DR. PHIL TAI, MCS, INTEL CORP.
*
*
*
*

BEGIN   ORG 0
        LAI 0
        OUT 10B
        OUT 11B
        OUT 12B
        OUT 13B
        LBI 8
        LCI 0
        LHI 8
        LLI 0
LM1     XRA
LM2     LMA
        INL
        CPL
        JFZ LM2
        INH
        LAI 12
        CPH
        JFZ LM1
        LHI 8
REPT4   LAB
        OUT 10B
```

Header

ORG 0
XRA
INL
INH
OUT

```
                LOAD 0 TO AC
                WRITE TO OUTPUT 0
                WRITE TO OUTPUT 1
                WRITE TO OUTPUT 2
                WRITE TO OUTPUT 3
                LOAD 8 TO REG. B
                LOAD 0 TO REG. C
                LOAD 8 TO REG H
                LOAD 0 TO REG. L
                CLEAR AC
                LOAD AC TO MEMORY
                L = L + 1
                AC - L
                JUMP IF AC IS NOT ZERO
                H = H + 1
                LOAD 12 TO AC
                AC-H
                JUMP IF AC IS NOT ZERO

                LOAD REG. B TO AC
```

```
REPT3   LLC             LOAD REG. C TO L
        LAC             LOAD REG. C TO AC
        OUT 13B
        LAI 255         LOAD 255 TO AC
        LMA             LOAD AC TO MEMORY
        CPM             AC-M
        JFZ ERROR       JUMP IF AC IS NOT ZERO
REPT2   LAH             LOAD REG. H TO AC
        OUT 10B
REPT5   XRA             CLEAR AC
        INL             L = L + 1
        CPL             AC - L
        JTZ REPT1       JUMP IF AC=0
        LAL             LOAD REG. L TO AC
        OUT 11B
        XRA             CLEAR AC
        CPM             AC-M
        JFZ ERROR       JUMP IF AC IS NOT ZERO
        JMP REPT5
```

Sample Flowchart

Search for a Char

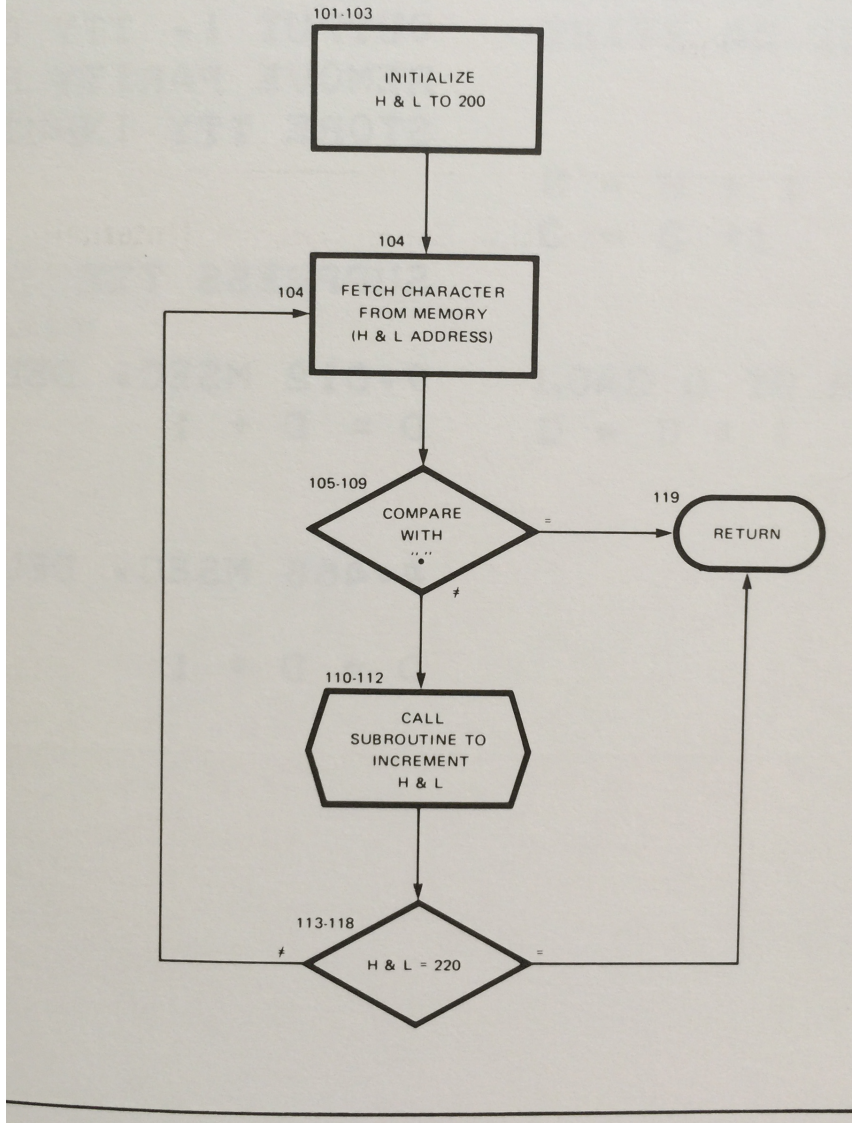


Figure 21. Subroutine to Search For Period

Sample Code

Search for a Char

MCS-8

PROGRAMMING EXAMPLES

Sample Program to Search A String Of Characters In Memory Locations 200-219 For A Period (.)

MNEMONIC	OPERAND	EXPLANATION	BYTES	LOCATION	ROM CODE	COMMENT
Start: LLI	200	Load L with 200	2	100 101	00110110 11001000	(200)
LHI	0	Load H with 0	2	102 103	00101110 00000000	(0)
Loop: LAM		Fetch Character from Memory	1	104	11000111	ASC II
CPI	"."	Compare it with period	2	105 106	00111100 00101110	ASC II (•)
JTZ	Found	If equal go to return	3	107 108 109	01101000 01110111 00000000	(119)
CAL	INCR	Call increment H&L subroutine	3	110 111 112	01000110 00111100 00000000	(60)
LAL		Load L to A	1	113	11000110	
CPI	220	Compare it with 220	2	114 115	00111100 11011100	(220)
JFZ	Loop	If unequal go to loop	3	116 117 118	01001000 01101000 00000000	(104)
Found: RET		Return	1	119	00000111	
INCR: INL		Increment L	1	60	00110000	
RFZ		Return if not zero	1	61	00001011	
INH		Increment H	1	62	00101000	
RET		Return	1	63	00000111	

Interrupts

ADDR.	LOCATION	PC CONTENTS	
N – 1	INTR. N – 1	N	(INTERRUPT ARRIVES HERE)
N	INSTR. N		
N + 1	INSTR. N + 1		

USER SUPPLIES ALTERNATE INSTRUCTION (RESTART OR CALL TO SRT), RELEASES INTERRUPT,
PC IS SAVED IN STACK (VALUE = N)

SUBROUTINE FOR HANDLING INTERRUPT:

S	INSTR. S			
S + 1	INSTR. S + 1			
S + 2				
S + K	RETURN	–	STACK POPS	– WITH VALUE N

AFTER COMPLETION OF SUBROUTINE, 8008 RETURNS TO EXECUTE ORIGINALLY REQUESTED INSTRUCTION, WHICH BLOCKING ADVANCE OF PC HAS SAVED.

Figure 5. 8008 Interrupt

x86 Assembly Language

i8086 Code

John Stephenson, Analyst programmer
magnetic tape reels.

Answered 9h ago

Set low byte A = all 1's or all 0's

Several methods:

```
        MOV   AL,FF
        OR    AL,FF
        XOR   AL,AL
        NOT   AL
```

MOV

AH	AL

AX

UPPER CASE!

i8086 Code

Subroutine to Sum Array Elements

Save

Loop

Restore

```
ADD_N_NUMBERS_FROM_ARRAY: // we suppose CX will hold how many we want
PUSH SI          // may not be needed, saves si value
PUSH BX          // may not be needed, saves BX value
LEA SI, MYDATA   // we move the memory address to SI
CLD              // clean the direction flag so goes forward.
XOR BX,BX        // clean the AX (make it zero)
ANFR1
LODSW                    // copies each data to AX
ADD BX,AX        // makes BX += AX (so holds the total sum)
LOOP ANFR1       // goes to the upper loop until CX = 0
MOV AX,BX
POP BX                   // restores bx
POP SI           // restores si, (each of this required if the push was done)
RET
```

BX = 0

AX	
AH	AL
BH	BL
BX	

i8086 Code

Subroutine to Sum Array Elements

should be called like this.

CALL sub

Load Im

```
MOV CX,10   //where 10 is in decimal format and its absolute value (in some compilers requires #)
CALL ADD_N_NUMBERS_FROM_ARRAY
```

to do it from just 10 different constant values have no much sense (yu can do manually the sum) but will be

```
XOR AX,AX
ADD AX,VAL1
ADD AX,VAL2
ADD AX,VAL3
ADD AX,VAL4
ADD AX,VAL5
ADD AX,VAL6
ADD AX,VAL7
ADD AX,VAL8
ADD AX,VAL9
ADD AX,VAL10
```

Loop unrolling

CSUN
CALIFORNIA STATE UNIVERSITY NORTHRIDGE
COMP122

x86 Assembly

DR JEFF SOFTWARE
INDIE APP DEVELOPER

x86 assembly language

From Wikipedia, the free encyclopedia

x86 Assembly Language is a family of backward-compatible assembly languages, which provide some level of compatibility all the way back to the Intel 8008 introduced in April 1972. x86 assembly languages are used to produce object code for the x86 class of processors. Like all assembly languages, it uses short mnemonics to represent the fundamental instructions that the CPU in a computer can understand and follow. Compilers sometimes produce assembly code as an intermediate step when translating a high level program into machine code. Regarded as a programming language, assembly coding is *machine-specific* and *low level*. Assembly languages are more typically used for detailed and time critical applications such as small real-time embedded systems or operating system kernels and device drivers.

Using the flags register [ec

Flags are heavily used for comparis
conditional jump instructions can be

```
    cmp eax, ebx
    jne do_something
    ; ...
do_something:
    ; do something here
```

x86 Assembly

DOS

"Hello world!" program for DOS in MASM style assembly [ed

Using interrupt 21h for output – other samples use libc's printf to print to stdout.[

```
.model small
.stack 100h

.data
msg db  'Hello world!$'

.code
start:
    mov ah, 09h    ; Display the message
    lea dx, msg
    int 21h
    mov ax, 4C00h  ; Terminate the executable
    int 21h

end start
```

x86 Assembly

Windows

"Hello world!" program for Windows in NASM style assembly

```
; Image base = 0x00400000
%define RVA(x) (x-0x00400000)
section .text
push dword hello
call dword [printf]
push byte +0
call dword [exit]
ret

section .data
hello db "Hello world!"

section .idata
dd RVA(msvcrt_LookupTable)
dd -1
dd 0
dd RVA(msvcrt_string)
dd RVA(msvcrt_imports)
times 5 dd 0 ; ends the descriptor table
```

```
msvcrt_string dd "msvcrt.dll", 0
msvcrt_LookupTable:
dd RVA(msvcrt_printf)
dd RVA(msvcrt_exit)
dd 0

msvcrt_imports:
printf dd RVA(msvcrt_printf)
exit dd RVA(msvcrt_exit)
dd 0

msvcrt_printf:
dw 1
dw "printf", 0
msvcrt_exit:
dw 2
dw "exit", 0
```

printf

x86 Assembly


```
                        ; _memcpy(dst, src, len)
                        ; Copy a block of memory from one location to another.
                        ;
                        ; Entry stack parameters
                        ;       [BP+6] = len, Number of bytes to copy
                        ;       [BP+4] = src, Address of source data block
                        ;       [BP+2] = dst, Address of target data block
                        ;
                        ; Return registers
                        ;       AX = Zero

0000:1000                       org     1000h       ; Start at 0000:1000h

0000:1000               _memcpy proc
0000:1000 55                    push    bp          ; Set up the call frame
0000:1001 89 E5                 mov     bp,sp
0000:1003 06                    push    es          ; Save ES
0000:1004 8B 4E 06              mov     cx,[bp+6]   ; Set CX = len
0000:1007 E3 11                 jcxz    done        ; If len = 0, return
0000:1009 8B 76 04              mov     si,[bp+4]   ; Set SI = src
0000:100C 8B 7E 02              mov     di,[bp+2]   ; Set DI = dst
0000:100F 1E                    push    ds          ; Set ES = DS
0000:1010 07                    pop     es

0000:1011 8A 04         loop    mov     al,[si]     ; Load AL from [src]
0000:1013 88 05                 mov     [di],al     ; Store AL to [dst]
0000:1015 46                    inc     si          ; Increment src
0000:1016 47                    inc     di          ; Increment dst
0000:1017 49                    dec     cx          ; Decrement len
0000:1018 75 F7                 jnz     loop        ; Repeat the loop

0000:101A 07            done    pop     es          ; Restore ES
0000:101B 5D                    pop     bp          ; Restore previous call frame
0000:101C 29 C0                 sub     ax,ax       ; Set AX = 0
0000:101E C3                    ret                 ; Return
0000:101F                       end proc
```

x86 Assembly

The above routine is a rather cumbersome way to copy blocks of data. The 8086 provides dedicated instructions for copying strings of bytes. These instructions assume that the source data is stored at DS:SI, the destination data is stored at ES:DI, and that the number of elements to copy is stored in CX. The above routine requires the source and the destination block to be in the same segment, therefore DS is copied to ES. The loop section of the above can be replaced by:

```
0000:1011 FC                cld                 ; Copy towards higher addresses
0000:1012 F2    loop        repnz               ; Repeat until CX = 0
0000:1013 A4                movsb               ; Move the data block
```