Microprocessor and Assembly Language Lab

Lab Material 7_1 for CSE 312 (M&AL Lab)

Dr. Shah Murtaza Rashid Al Masud
Associate Prof.
Dept. of CSE, UAP

FLOW CONTROL INSTRUCTIONS

For assembly language-programs to carry out useful tasks, there must be a way to make decisions and repeat sections of code. In this lab we show how these things can be accomplished with the jump and loop instructions.

The jump and loop instructions transfer control to another part of the program. This transfer can be unconditional or can depend on a particular combination of status flag settings.

After introducing the jump instructions, we'll use them to implement high-level language decision and looping structures. This application will make it much easier to convert a pseudo-code algorithm to assembly code,

Today we will see how jump and loop instructions can be used in assembly language programming to make decisions or repeat sections of code

In this experiment you will practice how to control the flow of an assembly language program using the compare instruction, the different jump instructions and the loop instructions.

JUMP Instruction

The jump instructions are used to transfer the flow of the process to the indicated operator.

There are two types of Jump instructions:

Unconditional Jump

Conditional Jump

Conditional Jumps

Jump_instruction destination_label

- **If the condition for the jump is true,** ***the next instruction to be executed is the one at destination_label***
- **If the condition is false,** ***the instruction immediately following the jump is done next***

Conditional Jumps

There are three categories of conditional jumps-

- **Signed conditional jumps: for *signed interpretation***
- **Unsigned conditional jumps: for *unsigned interpretation***
- **Single-flag jumps: operate on *settings of individual flags***

The JMP Instruction: unconditional Jump Instruction

JMP destination

- JMP can be used to get around the range restriction of a conditional jump.

Table: Different Types of Jump Instructions

Type	Instruction		Meaning (jump if)	Condition
Unconditional	JMP		unconditional	None
Comparisons	JA	jnbe	above (not below or equal)	CF = 0 and ZF = 0
	JAE	jnb	above or equal (not below)	CF = 0
	JB	jnae	below (not above or equal)	CF = 1
	JBE	jna	below or equal (not above)	CF = 1 or ZF = 1
	JE	jz	equal (zero)	ZF = 1
	JNE	jnz	not equal (not zero)	ZF = 0
	JG	jnle	greater (not lower or equal)	ZF = 0 and SF = OF
	JGE	jnl	greater or equal (not lower)	SF = OF
	JL	jnge	lower (not greater or equal)	(SF xor OF) = 1 i.e. SF ≠ OF
	JLE	jng	lower or equal (not greater)	(SF xor OF or ZF) = 1
	JCXZ	loop	CX register is zero	(CF or ZF) = 0
Carry	JC		Carry	CF = 1
	JNC		no carry	CF = 0
Overflow	JNO		no overflow	OF = 0
	JO		overflow	OF =1
Parity Test	JNP	jpo	no parity (parity odd)	PF = 0
	JP	jpe	parity (parity even)	PF = 1
Sign Bit	JNS		no sign	SF = 0
	JS		sign	SF = 1
Zero Flag	JZ		zero	ZF = 1
	JNZ		non-zero	ZF = 0

Signed Conditional Jump

Symbol	Description	Condition for Jumps
JG/JNLE	Jump if greater than	ZF=0 and SF=OF
	Jump if not less than or equal to	
JGE/JNL	Jump if greater than or equal to	SF=OF
	Jump if not less than or equal to	
JL/JNGE	Jump if less than	SF<>OF
	Jump if not greater than or equal	
JLE/JNG	Jump if less than or equal	ZF=1 or SF<>OF
	Jump if not greater than	

Unsigned Conditional Jump

Symbol	Description	Condition for Jumps
JA/JNBE	Jump if above	CF=0 and ZF=0
	Jump if not below or equal	
JAE/JNB	Jump if above or equal	CF=0
	Jump if not below	
JB/JNAE	Jump if below	CF=1
	Jump if not above or equal	
JBE/JNA	Jump if equal	CF=1 or ZF=1
	Jump if not above	

Single Flag Conditional Jump

Symbol	Description	Condition for Jumps
JE/JZ	Jump if equal	ZF=1
	Jump if equal to or zero	
JNE/JNZ	Jump if not equal	ZF=0
	Jump if not zero	
JC	Jump if carry	CF=1
JNC	Jump if no carry	CF=0
JO	Jump if overflow	OF=1
JNO	Jump if no overflow	OF=0
JS	Jump if sign negative	SF=1
JNS	Jump if nonnegative sign	SF=0
JP/JPE	Jump if parity even	PF=1
JNP/JPO	Jump if parity odd	PF=0

Compare(CMP) instruction:

The compare instruction is used to compare two numbers.

The jump condition is often provided by the CMP (compare) instruction.

The compare instruction **subtracts** its source operand from its destination operand and sets the value of the **status flags** according to the subtraction result.

The result of the subtraction is not stored anywhere.

CMP Instruction

CMP destination, source

- **Does the** ***compare by subtracting the source from the destination***
- **The result is** ***not stored***
- ***Only the flags are affected***
- **The operands of CMP** ***may not both be memory locations***
- **Destination operand** ***may not be a constant***

High-level Language Branching Structures

- IF-THEN

Syntax

```
IF condition is true
        THEN
        execute true branch statements
END_IF
```

The CMP (compare) Instruction

- **CMP destination, source**
- CMP is just like SUB, except that destination is not changed.
- CMP AX, BX ; AX = 7FFFh, BX = 0001h

  JG BELOW ; AX – BX = 7FFEh
- The jump condition for JG is satisfied because ZF = SF = OF = 0, so control transfers to label BELOW.

Interpreting the Conditional Jumps

- CMP AX, BX

  JG BELOW
- If AX is greater than BX (in a signed sense), then JG (jump if greater than) transfers to BELOW.
- DEC AX

  JL THERE
- If the contents of AX, in a signed sense, is less than 0, control transfers to THERE.

Signed Versus Unsigned Jumps

- CMP AX, BX ; AX = 7FFFh, BX = 8000h
  JA BELOW
- 7FFFh > 8000h in a signed sense, the program does not jump to BELOW.
- 7FFFh < 8000h in an unsigned sense, and we are using the unsigned jump JA.

Suppose AX and BX contain signed numbers. Write some code to put the biggest one in CX.

```
        MOV CX, AX      ; put AX in CX
        CMP BX, CX      ; is BX bigger?
        JLE NEXT        ; no, go on
        MOV CX, BX      ; yes, put BX in CX
NEXT:
```

```
.MODEL SMALL
.STACK 100H
.DATA
.CODE
MAIN PROC

    ;Suppose AX and BX contain signed numbers.
    ;Write  some code to put the biggest one in CX

    MOV AX, 5
    MOV BX, 3

    CMP AX, BX
    JG Label1                ; AX > BX

    MOV CX,BX

    Label1:
    MOV CX,AX




    MOV AH, 4CH
    INT 21H

MAIN ENDP
```

edit: E:\SayaStation\Assembly Language Programming\Flow Control Instruction\1.asm

original source code

```
.MODEL SMALL
.STACK 100H
.DATA
.CODE
MAIN PROC

;Suppose AX and BX contain signed numl
;Write  some code to put the biggest c

MOV AX, 3
MOV BX, 5

CMP AX,BX
JG Label1          ; AX < BX
MOV CX,BX

Label1:
MOV CX,AX


MAIN ENDP
```

emulator: 1.exe_

registers

	H	L
AX	00	03
BX	00	05
CX	00	05
DX	00	00
CS	0720	
IP	000C	
SS	0710	
SP	0100	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

```
07200: B8 184
07201: 03 003
07202: 00 000 NULL
07203: BB 187
07204: 05 005
07205: 00 000 NULL
07206: 3B 059 ;
07207: C3 195
07208: 7F 127
07209: 02 002
0720A: 8B 139
0720B: CB 203
0720C: 8B 139
0720D: C8 200
0720E: 90 144 É
0720F: 90 144 É
07210: 90 144 É
07211: 90 144 É
07212: 90 144 É
07213: 90 144 É
07214: 90 144 É
07215: 90 144 É
```

```
MOV CX, BX
MOV CX, AX
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```

```
.MODEL SMALL
.STACK 100H
.DATA
.CODE
MAIN PROC

    ;Suppose AX and BX contain signed numbers.
    ;Write  some code to put the biggest one in CX

    MOV AX, 3
    MOV BX, 5

    CMP AX,BX
    JG Label1           ; AX < BX
    MOV CX,BX
    JMP RETURN ;unconditional jump

    Label1:
    MOV CX,AX


    RETURN:
    MOV AH, 4CH
    INT 21H




MAIN ENDP
```

emulator: 1.exe_

file math debug view external virtual devices virtual drive help

Load | reload | step back | single step | run | step delay ms: 0

registers

	H	L
AX	00	03
BX	00	05
CX	00	05
DX	00	00
CS	0720	
IP	000C	
SS	0710	
SP	0100	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

0720:000C

```
07200: B8 184 ╕
07201: 03 003 ♥
07202: 00 000 NULL
07203: BB 187 ╗
07204: 05 005 ♣
07205: 00 000 NULL
07206: 3B 059 ;
07207: C3 195 ├
07208: 7F 127 ⌂
07209: 04 004 ♦
0720A: 8B 139 ï
0720B: CB 203 ╦
0720C: EB 235 δ
0720D: 02 002 ☻
0720E: 8B 139 ï
0720F: C8 200 ╚
07210: B4 180 ┤
07211: 4C 076 L
07212: CD 205 ═
07213: 21 033 !
07214: 90 144 É
07215: 90 144 É
```

0720:000C

```
MOV AX, 00003h
MOV BX, 00005h
CMP AX, BX
JNLE 0Eh
MOV CX, BX
JMP 010h
MOV CX, AX
MOV AH, 04Ch
INT 021h
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```

screen | source | reset | aux | vars | debug | stack | flags

original source code

```
.MODEL SMALL
.STACK 100H
.DATA
.CODE
MAIN PROC

;Suppose AX and BX contain signed numl
;Write  some code to put the biggest (

MOV AX, 3
MOV BX, 5

CMP AX,BX
JG Label1          ; AX < BX
MOV CX,BX
JMP RETURN ;unconditional jump

Label1:
MOV CX,AX


RETURN:
MOV AH, 4CH
INT 21H




MAIN ENDP
```

emulator: 1.exe_

file math debug view external virtual devices virtual drive help

Load | reload | step back | single step | run | step delay ms: 0

registers

	H	L
AX	00	03
BX	00	05
CX	00	05
DX	00	00
CS	0720	
IP	0010	
SS	0710	
SP	0100	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

0720:0010

```
07200: B8 184 ╕
07201: 03 003 ♥
07202: 00 000 NULL
07203: BB 187 ╗
07204: 05 005 ♣
07205: 00 000 NULL
07206: 3B 059 ;
07207: C3 195 ├
07208: 7F 127 ⌂
07209: 04 004 ♦
0720A: 8B 139 ï
0720B: CB 203 ╦
0720C: EB 235 δ
0720D: 02 002 ☻
0720E: 8B 139 ï
0720F: C8 200 ╚
07210: B4 180 ┤
07211: 4C 076 L
07212: CD 205 ═
07213: 21 033 !
07214: 90 144 É
07215: 90 144 É
```

0720:0010

```
MOV AH, 04Ch
INT 021h
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```

screen | source | reset | aux | vars | debug | stack | flags

```
.MODEL SMALL
.STACK 100H
.DATA
.CODE
MAIN PROC

;Suppose AX and BX contain signed numl
;Write  some code to put the biggest (

MOV AX, 3
MOV BX, 5

CMP AX,BX
JG Label1          ; AX < BX
MOV CX,BX
JMP RETURN ;unconditional jump

Label1:
MOV CX,AX


RETURN:
MOV AH, 4CH
INT 21H




MAIN ENDP
```

Now alter the value in AX and BX

edit: E:\SayaStation\Assembly Language Programming\Flow Control Instruction\1.asm

file edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator convertor options help about

```
.MODEL SMALL
.STACK 100H
.DATA
.CODE
MAIN PROC

    ;Suppose AX and BX contain signed numbers.
    ;Write  some code to put the biggest one in CX

    MOV AX, 5
    MOV BX, 3

    CMP AX,BX
    JG Label1           ; AX < BX
    MOV CX,BX
    JMP RETURN ;unconditional jump

    Label1:
    MOV CX,AX


    RETURN:
    MOV AH, 4CH
    INT 21H




MAIN ENDP
```

Now alter the value in AX and BX

emulator: 1.exe_

file math debug view external virtual devices virtual drive help

Load | reload | step back | single step | run | step delay ms: 0

registers

	H	L
AX	00	05
BX	00	03
CX	01	14
DX	00	00
CS	0720	
IP	0008	
SS	0710	
SP	0100	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

0720:0008

```
07200: B8 184
07201: 05 005
07202: 00 000 NULL
07203: BB 187
07204: 03 003
07205: 00 000 NULL
07206: 3B 059 ;
07207: C3 195
07208: 7F 127
07209: 04 004
0720A: 8B 139
0720B: CB 203
0720C: EB 235
0720D: 02 002
0720E: 8B 139
0720F: C8 200
07210: B4 180
07211: 4C 076 L
07212: CD 205 =
07213: 21 033 !
07214: 90 144 É
07215: 90 144 É
```

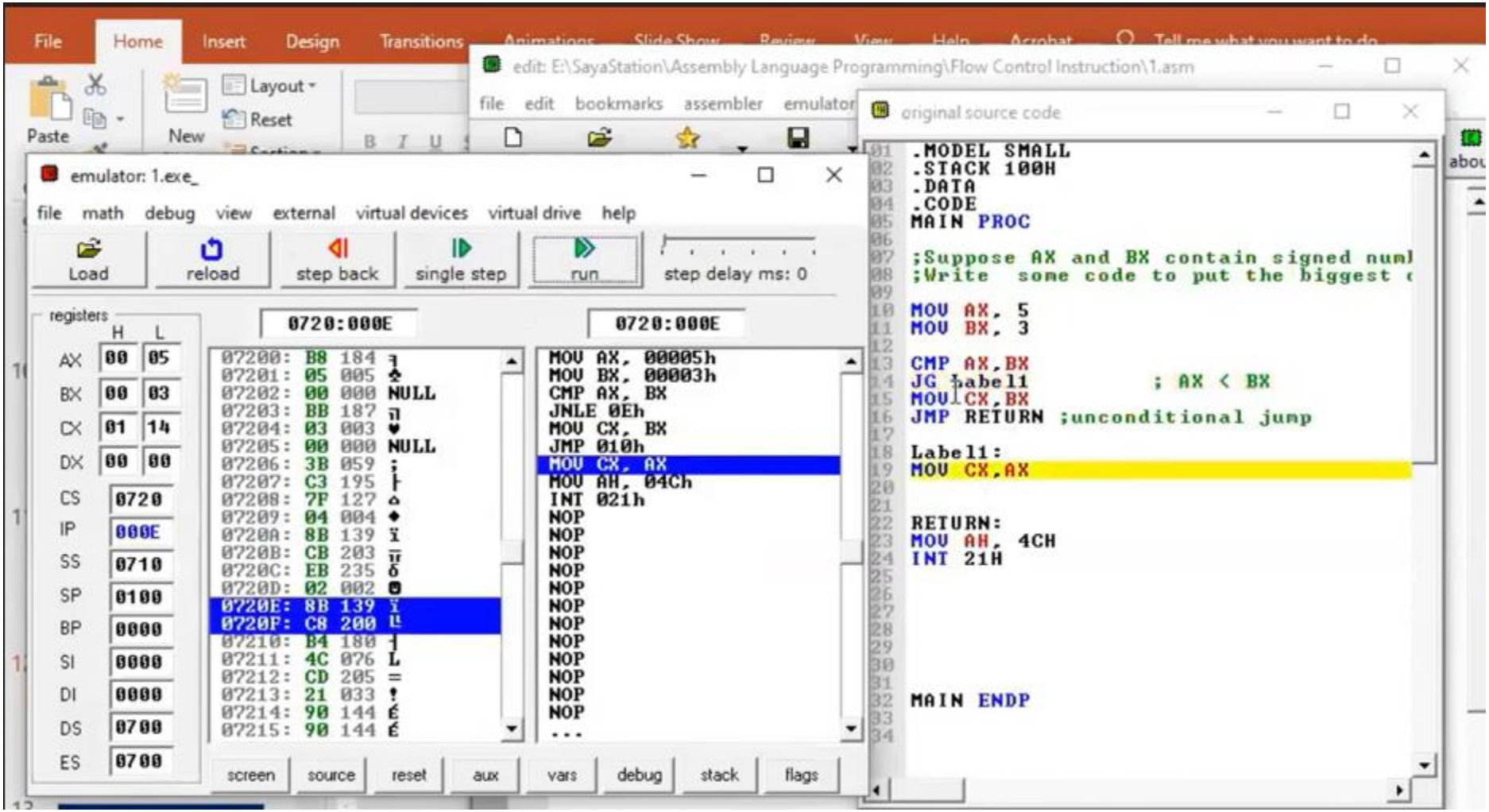
0720:0008

```
MOV AX, 00005h
MOV BX, 00003h
CMP AX, BX
JNLE 0Eh
MOV CX, BX
JMP 010h
MOV CX, AX
MOV AH, 04Ch
INT 021h
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```

screen | source | reset | aux | vars | debug | stack | flags

```
.MODEL SMALL
.STACK 100H
.DATA
.CODE
MAIN PROC

;Suppose AX and BX contain signed numl
;Write  some code to put the biggest

MOV AX, 5
MOV BX, 3

CMP AX,BX
JG Label1          ; AX < BX
MOV CX,BX
JMP RETURN ;unconditional jump

Label1:
MOV CX,AX


RETURN:
MOV AH, 4CH
INT 21H




MAIN ENDP
```

Now alter the value in AX and BX



Another example

- IF-THEN-ELSE

Syntax

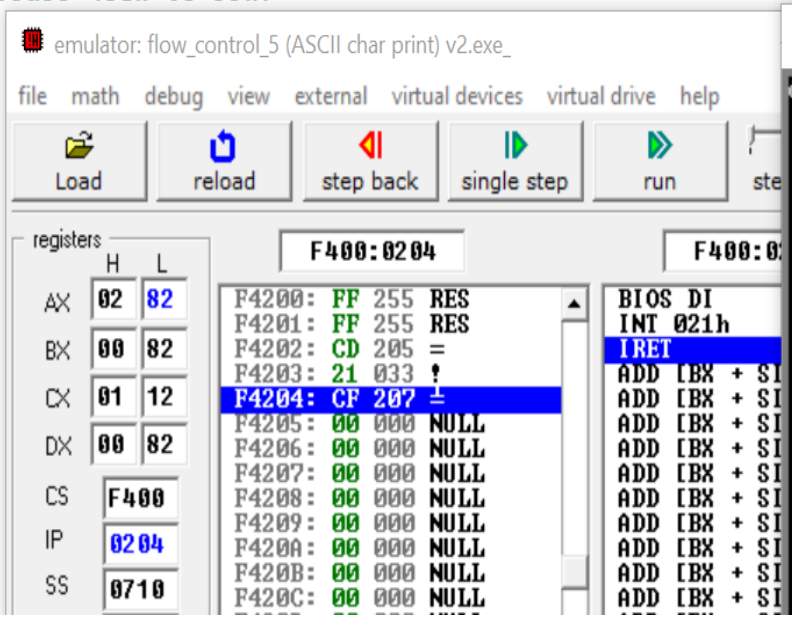
```
IF condition is true
        THEN
        execute true branch statements
ELSE
        execute false branch statements
END_IF
```

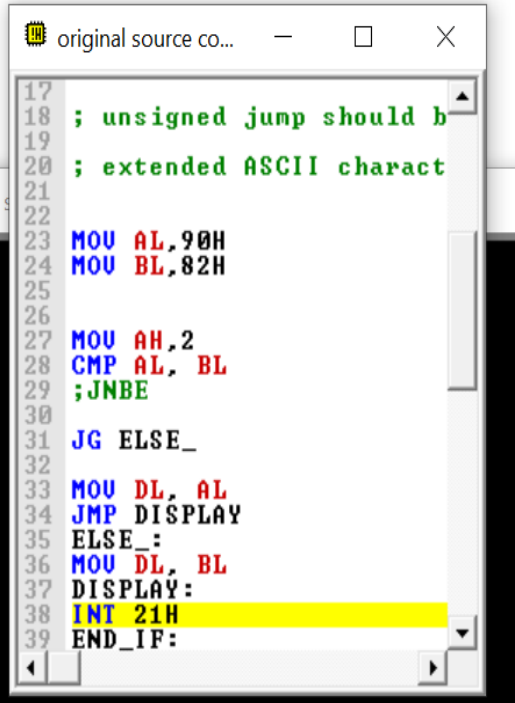
Another example

```
.MODEL SMALL
.STACK 100H
.DATA
.CODE
MAIN PROC

    ;Suppose AL and BL contain extended ASCII characters.
    ;Display the one that comes first in the character  sequence.

    ;unsigned jumps should be used when comparing
    ;extended ASCII character  codes (80H to FFH)


    MOV AL, 90H
    MOV BL, 82H

    CMP AL,BL
    JB PRINTAL      ; AL<BL

    MOV AH, 2
    MOV DL, BL
    INT 21H

    JMP RETURN


    PRINTAL:
    MOV AH, 2
    MOV DL, AL
    INT 21H

    RETURN:
    MOV AH, 4CH
    INT 21H
```

Another example

```
.DATA SEGMENT

.CODE SEGMENT

MAIN PROC

   ; suppose AL and BL contains extended ASCII characters
   ; display the one that comes first in character sequence

   ; unsigned jump should be used when comparing

   ; extended ASCII character codes (80H to FFH)

        MOV AL,90H
        MOV BL,82H

        MOV AH,2
CMP AL, BL
;JNBE

JG ELSE_

MOV DL, AL
JMP DISPLAY
ELSE_:
MOV DL, BL
DISPLAY:
INT 21H
END_IF:

        MAIN ENDP

        END MAIN
```





Extended ASCII from 80h to FFh

ascii codes

00:	null	20:	spa	40:	@	60:	`	C0:	Ç	A0:	á	C0:	└	E0:	α
01:	☺	21:	!	41:	A	61:	a	81:	ü	A1:	í	C1:	┴	E1:	β
02:	☻	22:	"	42:	B	62:	b	82:	é	A2:	ó	C2:	┬	E2:	Γ
03:	♥	23:	#	43:	C	63:	c	83:	â	A3:	ú	C3:	├	E3:	π
04:	♦	24:	$	44:	D	64:	d	84:	ä	A4:	ñ	C4:	─	E4:	Σ
05:	♣	25:	%	45:	E	65:	e	85:	à	A5:	Ñ	C5:	┼	E5:	σ
06:	♠	26:	&	46:	F	66:	f	86:	å	A6:	ª	C6:	╞	E6:	µ
07:	beep	27:	'	47:	G	67:	g	87:	ç	A7:	º	C7:	╟	E7:	τ
08:	back	28:	(	48:	H	68:	h	88:	ê	A8:	¿	C8:	╚	E8:	Φ
09:	tab	29:	)	49:	I	69:	i	89:	ë	A9:	⌐	C9:	╔	E9:	Θ
0A:	newl	2A:	*	4A:	J	6A:	j	8A:	è	AA:	¬	CA:	╩	EA:	Ω
0B:	♂	2B:	+	4B:	K	6B:	k	8B:	ï	AB:	½	CB:	╦	EB:	δ
0C:	♀	2C:	,	4C:	L	6C:	l	8C:	î	AC:	¼	CC:	╠	EC:	∞
0D:	cret	2D:	-	4D:	M	6D:	m	8D:	ì	AD:	¡	CD:	═	ED:	φ
0E:	♫	2E:	.	4E:	N	6E:	n	8E:	Ä	AE:	«	CE:	╬	EE:	ε
0F:	☼	2F:	/	4F:	O	6F:	o	8F:	Å	AF:	»	CF:	╧	EF:	∩
10:	►	30:	0	50:	P	70:	p	90:	É	B0:	░	D0:	╨	F0:	≡
11:	◄	31:	1	51:	Q	71:	q	91:	æ	B1:	▒	D1:	╤	F1:	±
12:	↕	32:	2	52:	R	72:	r	92:	Æ	B2:	▓	D2:	╥	F2:	≥
13:	‼	33:	3	53:	S	73:	s	93:	ô	B3:	│	D3:	╙	F3:	≤
14:	¶	34:	4	54:	T	74:	t	94:	ö	B4:	┤	D4:	╘	F4:	⌠
15:	§	35:	5	55:	U	75:	u	95:	ò	B5:	╡	D5:	╒	F5:	⌡
16:	▬	36:	6	56:	V	76:	v	96:	û	B6:	╢	D6:	╓	F6:	÷
17:	↨	37:	7	57:	W	77:	w	97:	ù	B7:	╖	D7:	╫	F7:	≈
18:	↑	38:	8	58:	X	78:	x	98:	ÿ	B8:	╕	D8:	╪	F8:	°
19:	↓	39:	9	59:	Y	79:	y	99:	Ö	B9:	╣	D9:	┘	F9:	∙
1A:	→	3A:	:	5A:	Z	7A:	z	9A:	Ü	BA:	║	DA:	┌	FA:	·
1B:	←	3B:	;	5B:	[	7B:	{	9B:	¢	BB:	╗	DB:	█	FB:	√
1C:	∟	3C:	<	5C:	\	7C:	|	9C:	£	BC:	╝	DC:	▄	FD:	²
1D:	↔	3D:	=	5D:	]	7D:	}	9D:	¥	BD:	╜	DD:	▌	FE:	■
1E:	▲	3E:	>	5E:	^	7E:	~	9E:	₧	BE:	╛	DE:	▐	FF:	res
1F:	▼	3F:	?	5F:	_	7F:	⌂	9F:	ƒ	BF:	┐	DF:	▀		

- CASE

Syntax

```
CASE expression
        values_1: statements_1
        values_2: statements_2
        .
        .
        values_n: statements_n
END_CASE
```

Example

If AX contains a negative number, put -1 in BX; if AX contains 0, put 0 in BX; if AX contains a positive number, put 1 in BX

Pseudocode Algorithm	Assembly Code

CASE AX
  <0: put -1 in BX
  =0: put 0 in BX
  >0: put 1 in BX
END_CASE
	CMP AX,0
	JL NEGATIVE
	JE ZERO
	JG POSITIVE
	NEGATIVE:
	MOV BX, -1
	JMP END_CASE
	ZERO:
	MOV BX,0
	JMP END_CASE
	POSITIVE:
	MOV BX,1
	END_CASE:

Another Example

```
.MODEL SMALL
.STACK 100H
.DATA
.CODE
MAIN PROC

    ;If AX contains a negative number, put -1 in BX;
    ;if AX contains  0,put 0 in BX;
    ;if AX contains a positive number, put 1 in BX

    MOV AX, -1

    CMP AX,0
    JL NEGATIVE
    JE ZERO
    JG POSITIVE

    NEGATIVE:
    MOV BX,-1
    JMP RETURN

    ZERO:
    MOV BX,0
    JMP RETURN

    POSITIVE:
    MOV BX,1


    RETURN:
    MOV AH,4CH
    INT 21H
```

Another Example

```
.MODEL SMALL
.STACK 100H
.DATA
.CODE
MAIN PROC

    ;If AX contains a negative number, put -1 in BX;
    ;if AX contains  0,put 0 in BX;
    ;if AX contains a positive number, put 1 in BX

    MOV AX, -1

    CMP AX,0
    JL NEGATIVE     ; AX<0
    JE ZERO         ; AX=0
    JG POSITIVE     ; AX>0

    NEGATIVE:
    MOV BX,-1
    JMP RETURN

    ZERO:
    MOV BX,0
    JMP RETURN

    POSITIVE:
    MOV BX,1


    RETURN:
    MOV AH,4CH
    INT 21H
```

Another Example

```
.MODEL SMALL
.STACK 100H
.DATA
.CODE
MAIN PROC

;If AX contains a negative num
;if AX contains  0,put 0 in BX
;if AX contains a positive num

MOV AX, -1

CMP AX,0
JL NEGATIVE    ; AX<0
JE ZERO        ; AX=0
JG POSITIVE    ; AX>0

NEGATIVE:
MOV BX,-1
JMP RETURN

ZERO:
MOV BX,0
JMP RETURN
```

```
07200: B8 184 ╗   MOV AX, 0FFFFh
07201: FF 255 RES CMP AX, 00000h
07202: FF 255 RES JL 0Ch
07203: 3D 061 =   JZ 011h
07204: 00 000 NULL JNLE 016h
07205: 00 000 NULL MOV BX, 0FFFFh
07206: 7C 124 ¦   JMP 019h
07207: 04 004 ♦   MOV BX, 00000h
07208: 74 116 t   JMP 019h
07209: 07 007 BEEP MOV BX, 00001h
0720A: 7F 127 ⌂   MOV AH, 04Ch
0720B: 0A 010 NEWL INT 021h
0720C: BB 187 ╗   NOP
0720D: FF 255 RES NOP
0720E: FF 255 RES NOP
0720F: EB 235 δ   NOP
07210: 08 008 BACK NOP
07211: BB 187 ╗   NOP
07212: 00 000 NULL NOP
07213: 00 000 NULL NOP
07214: EB 235 δ   NOP
07215: 03 003 ♥   ...
```

Another Example

```
.DATA SEGMENT

.CODE SEGMENT

MAIN PROC

    ; .....

        MOV AX, -1
        CMP AX, 0
JL NEGATIVE
JE ZERO
JG POSITIVE
NEGATIVE:
MOV BX, -1
JMP END_CASE
ZERO:
MOV BX,0
JMP END_CASE
POSITIVE:
MOV BX,1
END_CASE:


        MOV AX,4C00H

        INT 21H

       MAIN ENDP

       END MAIN
```

emulator: flow_control_7 NEG Potive Zero.exe_

file math debug view external virtual devices virtual drive help

Load | reload | step back | single step | run | step delay m

registers

	H	L
AX	FF	FF
BX	FF	FF
CX	01	1E
DX	00	00
CS	0720	
IP	000F	
SS	0710	
SP	0100	
BP	0000	
SI	0000	
DI	0000	
DS	0700	

0720:000F

```
07200: B8 184 ╕
07201: FF 255 RES
07202: FF 255 RES
07203: 3D 061 =
07204: 00 000 NULL
07205: 00 000 NULL
07206: 7C 124 ¦
07207: 04 004 ♦
07208: 74 116 t
07209: 07 007 BEEP
0720A: 7F 127 ⌂
0720B: 0A 010 NEWL
0720C: BB 187 ╗
0720D: FF 255 RES
0720E: FF 255 RES
0720F: EB 235 δ
07210: 08 008 BACK
07211: BB 187 ╗
07212: 00 000 NULL
07213: 00 000 NULL
07214: EB 235 δ
07215: 03 003 ♥
```

0720:000F

```
MOV AX, 0FFFFh
CMP AX, 00000h
JL 0Ch
JZ 011h
JNLE 016h
MOV BX, 0FFFFh
JMP 019h
MOV BX, 00000h
JMP 019h
MOV BX, 00001h
MOV AX, 04C00h
INT 021h
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```

original source co...

```
15
16
17 MOV AX, -1
18 CMP AX, 0
19 JL NEGATIVE
20 JE ZERO
21 JG POSITIVE
22 NEGATIVE:
23 MOV BX, -1
24 JMP END_CASE
25 ZERO:
26 MOV BX,0
27 JMP END_CASE
28 POSITIVE:
29 MOV BX,1
30 END_CASE:
31
32
33
34 MOV AX,4C00H
35
36 INT 21H
37
```

Another Example

emu8086 - assembler and microp

file edit bookmarks assembler emulator math ascii codes

new open examples save compile emula

```
01 include 'emu8086.inc'
02 org 100h
03
04    mov Cx,5h
05    mov Bx,6h
06
07    cmp Cx,Bx
08
09    je a ; jumping instruction
10
11    mov al,'*'
12    mov ah, 0eh
13    int 10h
14
15    a: ; landing instruction
16
17 ret
```

original source code

```
01 include 'emu8086.inc'
02 org 100h
03
04 mov Cx,5h
05 mov Bx,6h
06
07 cmp Cx,Bx
08
09 je a ; jumping instruction
10
11 mov al,'*'
12 mov ah, 0eh
13 int 10h
14
15 a: ; landing instruction
16
17 ret
18
19
```

emulator: noname.com_

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 200

registers H L

0700:0100

0700:0100

Another Example (print the asterisk)

More Example (jae and print the asterisk)

emu8086 - assembler and microp

file edit bookmarks assembler emulator math ascii codes

new open examples save compile emula

```
01 include 'emu8086.inc'
02 org 100h
03
04    mov Cx,5h
05    mov Bx,6h
06
07    cmp Cx,Bx
08
09    jae a ; jumping instruction
10
11    mov al,'*'
12    mov ah, 0eh
13    int 10h
14
15    a: ; landing instruction
16
17 ret
```

```
01 include 'emu8086.inc'
02 org 100h
03
04 mov Cx,5h
05 mov Bx,6h
06
07 cmp Cx,Bx
08
09 jae a ; jumping instruction
10
11 mov al,'*'
12 mov ah, 0eh
13 int 10h
14
15 a: ; landing instruction
16
17 ret
18
19
```

emulat

file math debug view external virtu

emulator screen (80x25 chars)

*

More Example (jbe and not satisfied the condition)

emu8086 - assembler and microp

file edit bookmarks assembler emulator math ascii codes

new open examples save compile emula

```
01 include 'emu8086.inc'
02 org 100h
03
04    mov Cx,5h
05    mov Bx,6h
06
07    cmp Cx,Bx
08
09    jbe a ; jumping instruction
10
11    mov al,'*'
12    mov ah, 0eh
13    int 10h
14
15    a: ; landing instruction
16
17 ret
```

```
01 include 'emu8086.inc'
02 org 100h
03
04 mov Cx,5h
05 mov Bx,6h
06
07 cmp Cx,Bx
08
09 jbe a ; jumping instruction
10
11 mov al,'*'
12 mov ah, 0eh
13 int 10h
14
15 a: ; landing instruction
16
17 ret
18
19
```

emulator noname.com_

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 200

registers

	H	L
AX	00	00
BX	00	06
CX	00	05

F400:0154

F400:0154

message

PROGRAM HAS RETURNED CONTROL
TO THE OPERATING SYSTEM

AL