CHAPTER 6

FLOW CONTROL INSTRUCTIONS

IBM CHARACTER DISPLAY

```
TITLE     IBM CHARACTER DISPLAY
.MODEL SMALL
.STACK    100H
.CODE
MAIN      PROC
     MOV  AH, 2              ; display char function
     MOV  CX, 256            ; no. of chars to display
     MOV  DL, 0              ; DL has ASCII code of null char
PRINT_LOOP:
     INT  21h                ; display a char
     INC  DL                 ; increment ASCII code
     DEC  CX                 ; decrement counter
     JNZ  PRINT_LOOP         ; keep going if CX not 0
; DOS exit
     MOV  AH, 4CH
     INT  21H
MAIN ENDP
     END MAIN
```

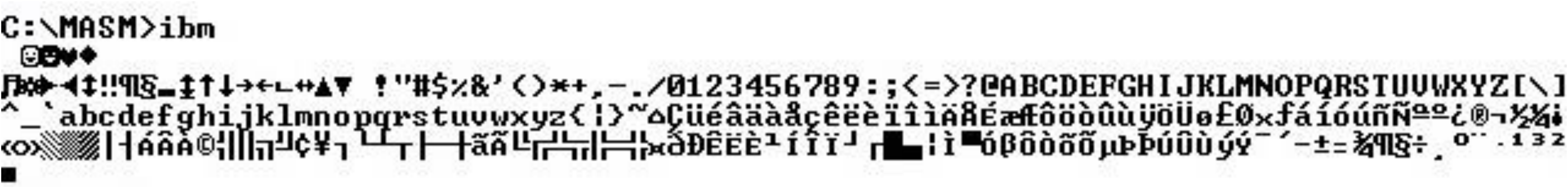
Label

0	<NUL>	32	<SPC>	64	@	96	`	128	Ä	160	†	192	¿	224	‡
1	<SOH>	33	!	65	A	97	a	129	Å	161	°	193	¡	225	·
2	<STX>	34	"	66	B	98	b	130	Ç	162	¢	194	¬	226	‚
3	<ETX>	35	#	67	C	99	c	131	É	163	£	195	√	227	„
4	<EOT>	36	$	68	D	100	d	132	Ñ	164	§	196	ƒ	228	‰
5	<ENQ>	37	%	69	E	101	e	133	Ö	165	•	197	≈	229	Â
6	<ACK>	38	&	70	F	102	f	134	Ü	166	¶	198	∆	230	Ê
7	<BEL>	39	'	71	G	103	g	135	á	167	ß	199	«	231	Á
8	<BS>	40	(	72	H	104	h	136	à	168	®	200	»	232	Ë
9	<TAB>	41	)	73	I	105	i	137	â	169	©	201	…	233	È
10	<LF>	42	*	74	J	106	j	138	ä	170	™	202		234	Í
11	<VT>	43	+	75	K	107	k	139	ã	171	´	203	À	235	Î
12	<FF>	44	,	76	L	108	l	140	å	172	¨	204	Ã	236	Ï
13	<CR>	45	-	77	M	109	m	141	ç	173	≠	205	Õ	237	Ì
14	<SO>	46	.	78	N	110	n	142	é	174	Æ	206	Œ	238	Ó
15	<SI>	47	/	79	O	111	o	143	è	175	Ø	207	œ	239	Ô
16	<DLE>	48	0	80	P	112	p	144	ê	176	∞	208	–	240	
17	<DC1>	49	1	81	Q	113	q	145	ë	177	±	209	—	241	Ò
18	<DC2>	50	2	82	R	114	r	146	í	178	≤	210	“	242	Ú
19	<DC3>	51	3	83	S	115	s	147	ì	179	≥	211	”	243	Û
20	<DC4>	52	4	84	T	116	t	148	î	180	¥	212	‘	244	Ù
21	<NAK>	53	5	85	U	117	u	149	ï	181	µ	213	’	245	ı
22	<SYN	54	6	86	V	118	v	150	ñ	182	∂	214	÷	246	ˆ
23	<ETB>	55	7	87	W	119	w	151	ó	183	∑	215	◊	247	˜
24	<CAN>	56	8	88	X	120	x	152	ò	184	∏	216	ÿ	248	¯
25	<EM>	57	9	89	Y	121	y	153	ô	185	π	217	Ÿ	249	˘
26	<SUB>	58	:	90	Z	122	z	154	ö	186	∫	218	⁄	250	˙
27	<ESC>	59	;	91	[	123	{	155	õ	187	ª	219	€	251	˚
28	<FS>	60	<	92	\	124	|	156	ú	188	º	220	‹	252	¸
29	<GS>	61	=	93	]	125	}	157	ù	189	Ω	221	›	253	˝
30	<RS>	62	>	94	^	126	~	158	û	190	æ	222	ﬁ	254	[illegible]

0	<NUL>	32	<SPC>	64	@	96	`	128	Ä	160	†	192	¿	224	‡
1	<SOH>	33	!	65	A	97	a	129	Å	161	°	193	¡	225	·
2	<STX>	34	"	66	B	98	b	130	Ç	162	¢	194	¬	226	‚
3	<ETX>	35	#	67	C	99	c	131	É	163	£	195	√	227	„
4	<EOT>	36	$	68	D	100	d	132	Ñ	164	§	196	ƒ	228	‰
5	<ENQ>	37	%	69	E	101	e	133	Ö	165	•	197	≈	229	Â
6	<ACK>	38	&	70	F	102	f	134	Ü	166	¶	198	∆	230	Ê
7	<BEL>	39	'	71	G	103	g	135	á	167	ß	199	«	231	Á
8	<BS>	40	(	72	H	104	h	136	à	168	®	200	»	232	Ë
9	<TAB>	41	)	73	I	105	i	137	â	169	©	201	…	233	È
10	<LF>	42	*	74	J	106	j	138	ä	170	™	202		234	Í
11	<VT>	43	+	75	K	107	k	139	ã	171	´	203	À	235	Î
12	<FF>	44	,	76	L	108	l	140	å	172	¨	204	Ã	236	Ï
13	<CR>	45	-	77	M	109	m	141	ç	173	≠	205	Õ	237	Ì
14	<SO>	46	.	78	N	110	n	142	é	174	Æ	206	Œ	238	Ó
15	<SI>	47	/	79	O	111	o	143	è	175	Ø	207	œ	239	Ô
16	<DLE>	48	0	80	P	112	p	144	ê	176	∞	208	–	240	
17	<DC1>	49	1	81	Q	113	q	145	ë	177	±	209	—	241	Ò
18	<DC2>	50	2	82	R	114	r	146	í	178	≤	210	“	242	Ú
19	<DC3>	51	3	83	S	115	s	147	ì	179	≥	211	”	243	Û
20	<DC4>	52	4	84	T	116	t	148	î	180	¥	212	‘	244	Ù
21	<NAK>	53	5	85	U	117	u	149	ï	181	µ	213	’	245	ı
22	<SYN	54	6	86	V	118	v	150	ñ	182	∂	214	÷	246	ˆ
23	<ETB>	55	7	87	W	119	w	151	ó	183	∑	215	◊	247	˜
24	<CAN>	56	8	88	X	120	x	152	ò	184	∏	216	ÿ	248	¯
25	<EM>	57	9	89	Y	121	y	153	ô	185	π	217	Ÿ	249	˘
26	<SUB>	58	:	90	Z	122	z	154	ö	186	∫	218	⁄	250	˙
27	<ESC>	59	;	91	[	123	{	155	õ	187	ª	219	€	251	˚
28	<FS>	60	<	92	\	124	|	156	ú	188	º	220	‹	252	¸
29	<GS>	61	=	93	]	125	}	157	ù	189	Ω	221	›	253	˝
30	<RS>	62	>	94	^	126	~	158	û	190	æ	222	ﬁ	254	
31	<US>	63	?	95	_	127	<DEL>	159	ü	191	ø	223	ﬂ	255	ˇ

IBM Character Display

- IBM.ASM



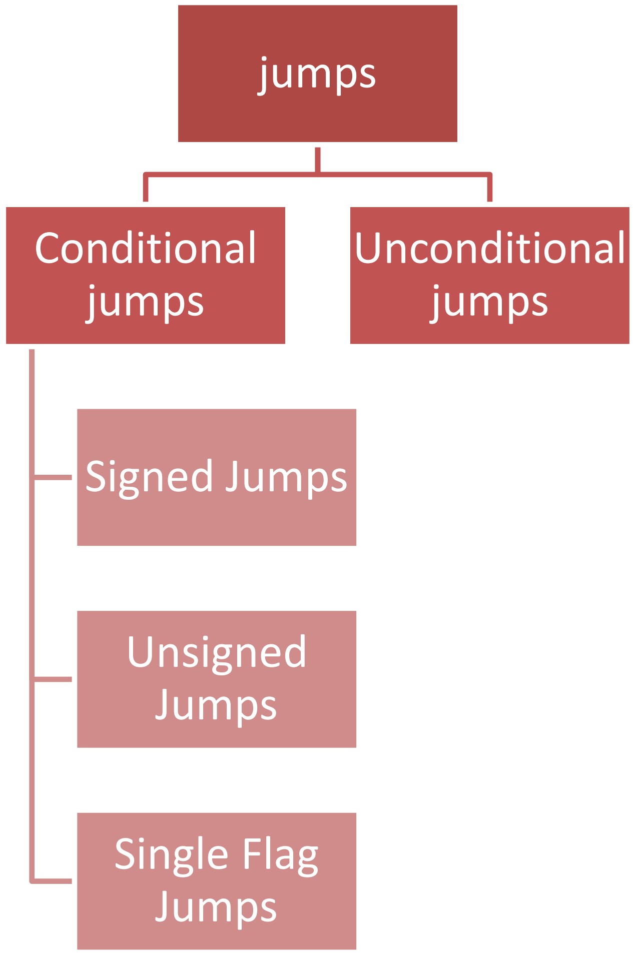
```
C:\MASM>ibm
 ☺☻♥♦
♫☼►◄↕‼¶§▬↨↑↓→←∟↔▲▼ !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]
^_`abcdefghijklmnopqrstuvwxyz{¦}~⌂ÇüéâäàåçêëèïîìÄÅÉæÆôöòûùÿÖÜø£Ø×ƒáíóúñÑªº¿®¬½¼¡
«»░▒▓│┤ÁÂÀ©╣║╗╝¢¥┐└┴┬├─┼ãÃ╚╔╩╦╠═╬¤ðÐÊËÈıÍÎÏ┘┌█▄¦Ì▀ÓßÔÒõÕµþÞÚÛÙýÝ¯´-±‗¾¶§÷¸°¨·¹³²
■
```

Conditional Jumps

- **Jxxx destination_label**
- In IBM.ASM, the CPU executes JNZ PRINT_LOOP by inspecting ZF.
- If ZF = 0, control transfers to PRINT_LOOP
- If ZF = 1, it goes on to execute MOV AH, 4CH
- Jump instructions themselves do not affect the flags.

Range of a conditional jump

- destination_label must precede the jump instruction by no more than 126 bytes, or follow it by no more than 127 bytes.


jumps
Conditional jumps
Unconditional jumps
Signed Jumps
Unsigned Jumps
Single Flag Jumps

The CMP (compare) Instruction

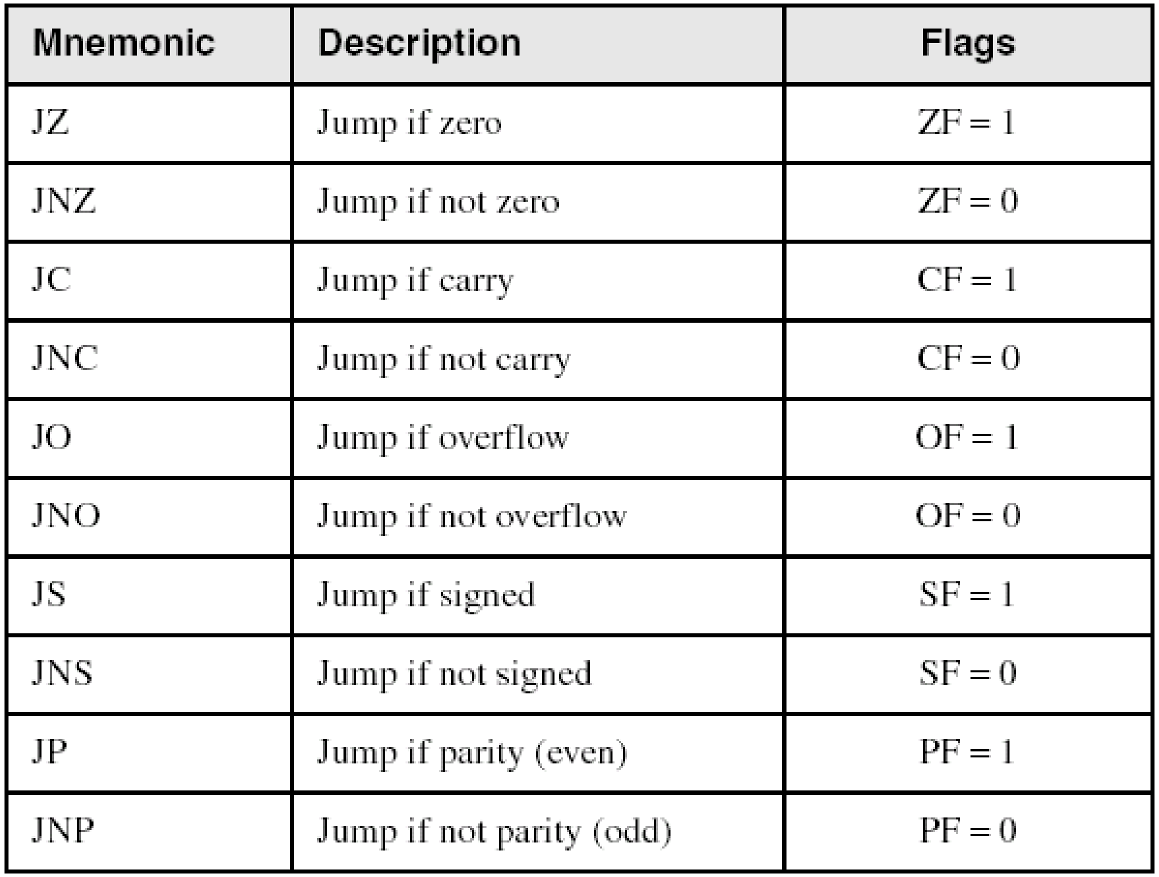
- **CMP destination, source**
- CMP is just like SUB, except that destination is not changed.
-
```
CMP     AX, BX      ; AX = 7FFFh, BX = 0001h
JG      BELOW       ; AX – BX = 7FFEh
```
- The jump condition for JG is satisfied because ZF = SF = OF = 0, so control transfers to label BELOW.

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.

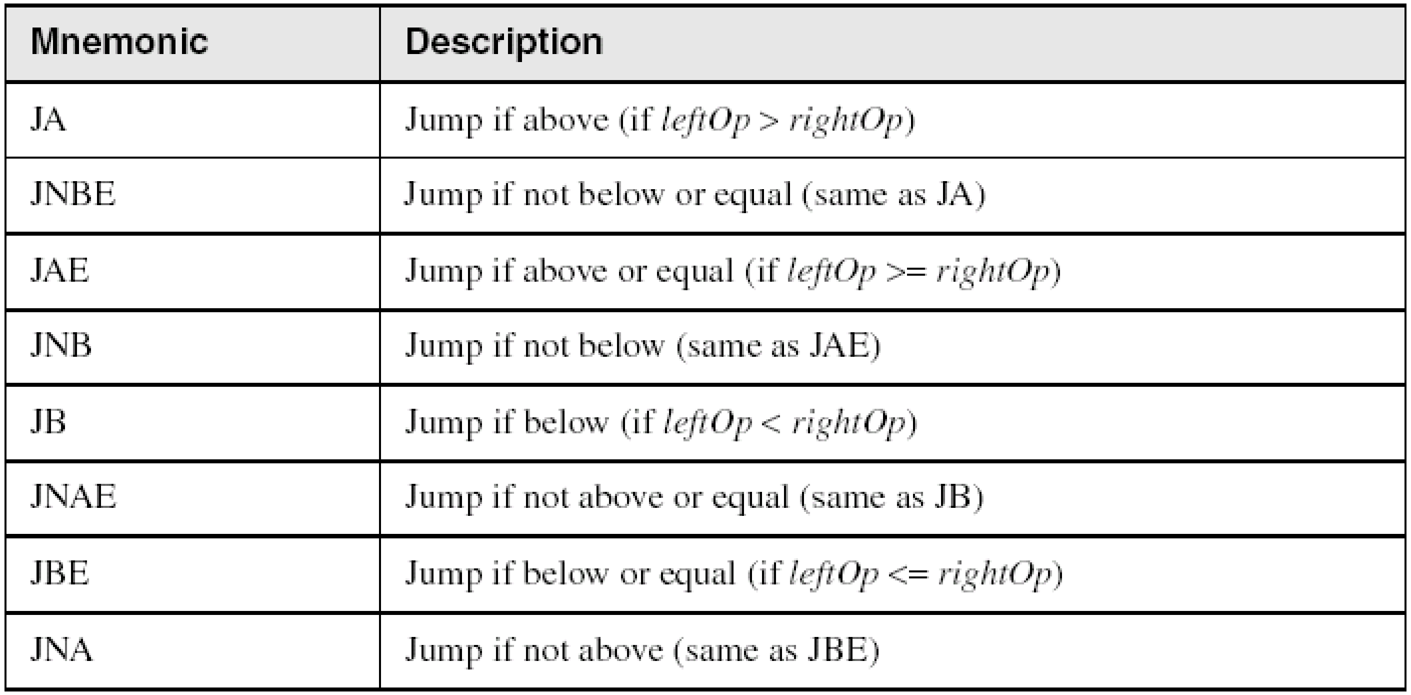
Jumps Based on Specific Flags



Mnemonic	Description	Flags
JZ	Jump if zero	ZF = 1
JNZ	Jump if not zero	ZF = 0
JC	Jump if carry	CF = 1
JNC	Jump if not carry	CF = 0
JO	Jump if overflow	OF = 1
JNO	Jump if not overflow	OF = 0
JS	Jump if signed	SF = 1
JNS	Jump if not signed	SF = 0
JP	Jump if parity (even)	PF = 1
JNP	Jump if not parity (odd)	PF = 0

JCXZ

Jumps Based on Unsigned Comparisons



Mnemonic	Description
JA	Jump if above (if *leftOp* > *rightOp*)
JNBE	Jump if not below or equal (same as JA)
JAE	Jump if above or equal (if *leftOp* >= *rightOp*)
JNB	Jump if not below (same as JAE)
JB	Jump if below (if *leftOp* < *rightOp*)
JNAE	Jump if not above or equal (same as JB)
JBE	Jump if below or equal (if *leftOp* <= *rightOp*)
JNA	Jump if not above (same as JBE)

JCXZ

Jumps Based on Signed Comparisons

Mnemonic	Description
JG	Jump if greater (if *leftOp* > *rightOp*)
JNLE	Jump if not less than or equal (same as JG)
JGE	Jump if greater than or equal (if *leftOp* >= *rightOp*)
JNL	Jump if not less (same as JGE)
JL	Jump if less (if *leftOp* < *rightOp*)
JNGE	Jump if not greater than or equal (same as JL)
JLE	Jump if less than or equal (if *leftOp* <= *rightOp*)
JNG	Jump if not greater (same as JLE)

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:
```

The JMP Instruction

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

Unconditional Jump

```
TOP:
; body of the loop
        DEC   CX              ; decrement counter
        JNZ   TOP             ; keep looping if CX > 0
        MOV   AX, BX
```

; the loop body contains so many instructions that label TOP is out of range for JNZ (more than 126 bytes before JMP TOP)

Unconditional Jump

```
TOP:
; body of the loop
        DEC   CX               ; decrement counter
        JNZ   BOTTOM           ; keep looping if CX > 0
        JMP   EXIT
BOTTOM:
        JMP   TOP
EXIT:
        MOV  AX, BX
```

High level language constructs

IF-THEN

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

Replace the number in AX by its absolute value.

```
IF AX < 0
   THEN
        replace AX by –AX
END_IF
```

Replace the number in AX by its absolute value.

```
; if AX < 0
    CMP AX, 0        ; AX < 0 ?
    JNL END_IF   ; no, exit
; then
    NEG AX       ; yes, change sign
END IF:
```

IF-THEN-ELSE

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

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

```
IF AL <= BL
   THEN
        display the character in AL
   ELSE
        display the character in BL
END_IF
```

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

```
        MOV     AH, 2           ; prepare to display
; if AL <= BL
        CMP     AL, BL          ; AL <= BL?
        JNBE    ELSE_           ; no, display char in BL
; then                          ; AL <= BL
        MOV     DL, AL          ; move char to be displayed
        JMP     DISPLAY         ; go to display
ELSE_:                          ; BL < AL
        MOV     DL, BL
DISPLAY:
        INT     21h             ; display it
END_IF
```

```
CMP AL,BL

JB  next
MOV DL,BL
INT 21H
JMP exit

next:
MOV DL,AL
INT 21H
JMP exit

exit:
```

ELSE is a reserved word

Needed to skip false branch (not needed in high level language)

CASE

```
CASE expression
   value 1 : statements_1
   value 2 : statements_2
   .
   .
   .
   value n : statements_n
END_CASE
```

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.

```
CASE AX
   <0 : put –1 in BX
   =0 : put 0 in BX
   >0 : put 1 in BX
END_CASE
```

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.

```
; case AX
        CMP     AX, 0           ; test AX
        JL      NEGATIVE        ; AX < 0
        JE      ZERO            ; AX = 0
        JG      POSITIVE        ; AX > 0
NEGATIVE:
        MOV     BX, -1          ; put -1 in BX
        JMP     END_CASE        ; and exit
ZERO:
        MOV     BX, 0           ; put -0in BX
        JMP     END_CASE        ; and exit
POSITIVE:
        MOV     BX, 1           ; put 1 in BX
END_CASE:
```

Only one cmp is needed as jump instructions don’t affect the flags

If AL contains 1 or 3, display “o”; If AL contains 2 or 4, display “e”.

```
CASE AL
    1, 3 :      display “o”
    2, 4 :      display “e”
END_CASE
```

If AL contains 1 or 3, display “o”; If AL contains 2 or 4, display “e”.

```
; case AL
; 1,3 :
        CMP     AL, 1           ; AL = 1?
        JE      ODD             ; yes, display ‘o’
        CMP     AL ,3           ; AL = 3?
        JE      ODD             ; yes, display ‘o’
; 2,4 :
        CMP     AL, 2           ; AL = 2?
        JE      EVEN            ; yes, display ‘e’
        CMP     AL, 4           ; AL = 4?
        JE      EVEN            ; yes, display ‘e’
        JMP     END_CASE        ; not 1..4
```

If AL contains 1 or 3, display “o”; If AL contains 2 or 4, display “e”.

```
ODD:                    ; display ‘o’
        MOV  DL, ‘o’    ; get ‘o’
        JMP  DISPLAY    ; go to display
EVEN:                   ; display ‘e’
        MOV  DL, ‘e’    ; get ‘e’
DISPLAY:
        MOV  AH, 2
        INT  21H        ; display char
END_CASE:
```

```
Mov AH,2
Mov DL,'0'
int 21h
jmp end_case
```

Branches with Compound Conditions

- Some times the branching in an IF or CASE takes from;
- **condition_1 AND condition_2**

or

condition_1 OR condition_2

AND Conditions

- **condition_1 AND condition_2**
- An AND condition is true if and only if condition_1 and condition_2 are both true.
- If either condition is false, then the whole thing is false.

Read a character, and if it’s an uppercase letter, display it.

```
Read a character (into AL)
IF (‘A’ <= character) and (character <= ‘Z’)
   THEN
        display character
END_IF
```

Read a character, and if it's an uppercase letter, display it.

```
; read a character
             MOV     AH, 1                  ; prepare to read
             INT     21H                    ; char in AL
; if ('A' <= char) and (char >= 'Z')
             CMP     AL, 'A'                ; char >= 'A'?
  JL END_IF  JNGE    END_IF                 ; no, exit
             CMP     AL, 'Z'                ; char <= 'Z'?
  JG end_if  JNLE    END_IF                 ; no, exit
; then display char
             MOV     DL, AL                 ; get char
             MOV     AH, 2                  ; prepare to display
             INT     21H                    ; display char
END_IF:
```

OR Conditions

- **condition_1 OR condition_2**
- condition_1 OR condition_2 is true if at least one of the conditions is true.
- It is only false when both conditions are false.

Read a character, and if it is “y” or “Y”, display it; otherwise, terminate the program.

```
Read a character (into AL)
IF (character = ‘y’) or (character = ‘Y’)
   THEN
        display it
   ELSE
        terminate the program
END_IF
```

Read a character, and if it is “y” or “Y”, display it; otherwise, terminate the program.

```
; read a character
        MOV  AH, 1          ; prepare to read
        INT  21H            ; char in AL
; if (character = ‘y’) or (character = ‘Y’)
        CMP  AL, ‘y’        ; char = ‘y’?
        JE   THEN           ; yes, go to display it
        CMP  AL, ‘Y’        ; char = ‘Y’?
        JE   THEN           ; yes, go to display it
        JMP  ELSE_          ; no, terminate
```

Read a character, and if it is “y” or “Y”, display it; otherwise, terminate the program.

```
THEN:
        MOV  AH, 2          ; prepare to display
        MOV  DL, AL         ; get char
        INT  21H            ; display it
        JMP  END_IF         ; end exit
ELSE_:
        MOV  AH, 4CH
        INT  21H            ; DOS exit
END_IF:
```

Looping Structures

- A loop is a sequence of instructions that is repeated .
- The number of times to repeat may be known in advance

    or

- Depend on some condition

FOR LOOP

Loop statements are repeated a known number of times;

```
FOR  loop_count  times  DO
   statements
END_FOR
```

The LOOP instruction

- **LOOP destination_label**

```
; initialize CX to loop_count
TOP:
          ; body of the loop
          LOOP TOP
```

The LOOP instruction

- The counter for the loop is the register CX which is initialized to loop_count.
- Execution of the LOOP instruction causes CX to be decremented automatically.
- If CX is not 0, control transfers to destination_label.
- If CX = 0, the next instruction after LOOP is done.

Write a count-controlled loop to display a row of 80 stars.

```
FOR 80 times DO
   display ‘*’
END_FOR
```

Write a count-controlled loop to display a row of 80 stars.

```
        MOV  CX, 80        ; number of stars to display
        MOV  AH, 2         ; display character function
        MOV  DL, ‘*’       ; character to display
TOP:
        INT  21h           ; display a star
        LOOP TOP           ; repeat 80 times
```

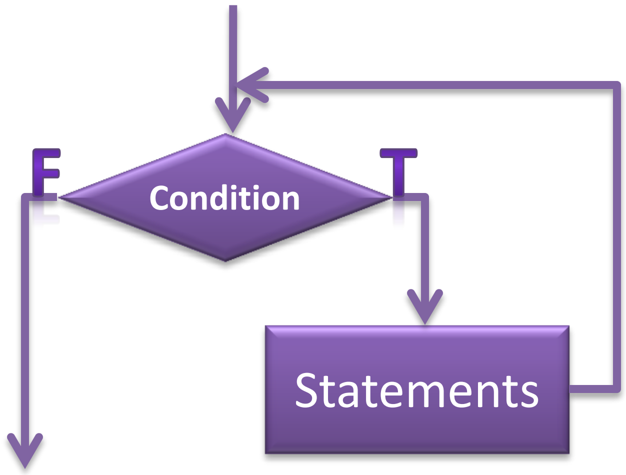
The instruction JCXZ (jump if CX is zero)

- **JCXZ destination_label**

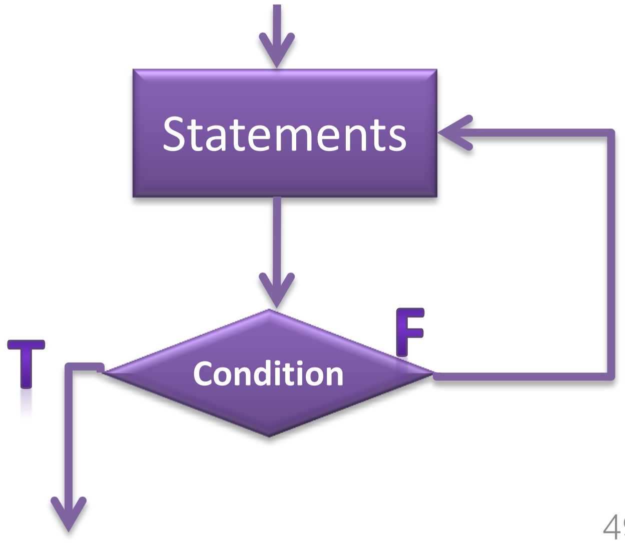
```
        JCXZ  SKIP
TOP:
              ; body of the loop
              LOOP TOP
SKIP:
```

WHILE LOOP and REPEAT LOOP

```
WHILE  condition  DO
   statements
END_WHILE
```



```
REPEAT
   statements
UNTIL  condition
```



Write some code to count the number of characters in an input line.

```
Initialize count to 0
read a character
WHILE  character <> carriage_return  DO
   count = count + 1
    read a character
END_WHILE
```

Write some code to count the number of characters in an input line.

```
        MOV   DX, 0           ; DX counts characters
        MOV   AH, 1           ; prepare to read
        INT   21H             ; character in AL
WHILE_:
        CMP   AL, 0DH         ; CR?
        JE    END_WHILE       ; yes, exit
        INC   DX              ; not CR, increment count
        INT   21H             ; read a character
        JMP   WHILE_          ; loop back
END_WHILE_:
```

carries return mane hosse enter button cap diyesi kina

Write some code to read characters until a blank is read.

```
REPEAT
    read a character
UNTIL  character is a blank
```

```
        MOV   AH, 1          ; prepare to read
REPEAT:
        INT   21H            ; char in AL
; until
        CMP   AL, ‘ ‘        ; a blank?
        JNE   REPEAT         ; no, keep reading
```

```
Mov ah,1

lebel:

int 21h
cmp al,' '
je exit
jmp lebel

exit:
```

Programming with High-Level Structures

- CAP.ASM

```
Type a line of text:
THE QUICK BROWN FOX JUMPED.
First capital = B Last capital = X
```

If no capital letter entered, display
“No capital letter entered”

Read and process a line of text

```
Read a character
WHILE character is not a carriage return DO
    IF character is a capital letter (‘A’ <= character AND character <= ‘Z’)
          THEN
                    IF character precedes first capital
                              THEN    first capital = character
                    END IF
                    IF character follows last capital
                              THEN    last capital = character
                    END IF
    END IF
Read a character
END_WHILE
```

Display the results

```
IF no capitals were typed,
   THEN
        display "No capitals"
   ELSE
        display first capital and last capital
END_IF
```

LAST

ASCII Character Table

FIRST

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
[illegible]	[illegible]	[illegible]	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

CAP.ASM

1(4)

```
TITLE      FIRST AND LAST CAPITALS
.MODEL     SMALL
.STACK     100H
.DATA
PROMPT     DB   'Type a line of text', 0DH,
   0AH, '$'
NOCAP_MSG DB   0DH, 0AH, 'No capitals $'
CAP_MSG    DB   0DH, 0AH, 'First capital =
   '
FIRST      DB   '['
   DB ' Last capital = '
LAST DB   '@ $'

.CODE
MAIN PROC
; initialize DS
   MOV      AX, @DATA
   MOV      DS, AX
```

CAP.ASM 2(4)

```
    ; display opening message
      MOV AH, 9        ; display string function
      LEA DX, PROMPT   ; get opening message
      INT 21H          ; display it
    ; read and process a line of text
      MOV AH, 1        ; read char function
      INT 21H          ; char in AL
   WHILE_:
    ; while character is not a carriage return do
      CMP AL, 0DH                ; CR?
      JE  END_WHILE    ; yes, exit
    ; if character is a capital letter
      CMP AL, 'A'                ; char >= 'A'?
JLE or JNGE      END_IF          ; not a capital letter
      CMP AL, 'Z'                ; chat <= 'Z'?
JGE or JNLE      END_IF          ; not a capital letter
    ; then
```

CAP.ASM 3(4)

```
; if character precedes first capital
   CMP AL, FIRST     ; char < first capital?
   JNL CHECK_LAST    ; no, >=
; then first capital = character
   MOV FIRST, AL     ; FIRST = char
; end_if
CHECK_LAST:
; if character follows last capital
   CMP AL, LAST      ; char > last capital?
   JNG END_IF              ; no, <=
; then last capital = character
   MOV LAST, AL      ; LAST = char
; end_if
END_IF:
; read a character
   INT 21H           ; char in AL
   JMP WHILE_              ; repeat loop
END_WHILE:
```

CAP.ASM 4(4)

```
; display results
  MOVAH, 9        ; display string function
; if no capitals were typed
  CMPFIRST, '['; first = '['
  JNECAPS         ; no, display results
; then
  LEADX, NOCAP_MSG   ; no capitals
  JMPDISPLAY
CAPS:
  LEADX, CAP_MSG     ; capitals
DISPLAY:
  INT21H          ; display message
; end_if
; dos exit
  MOVAH, 4CH
  INT21H
MAIN ENDP
  END MAIN
```