
Unconditional Instruction and Conditional Processing

Contents of Lecture:

- ❖ Introduction
- ❖ JMP Instruction
- ❖ Conditional Jump
- ❖ IF statements

References for Lecture:

KIP R. IRVINE, *Assembly Language for x86 Processors, 7th Edition, Chapter 4: Data Transfer, Addressing and Arithmetic*

KIP R. IRVINE, *Assembly Language for x86 Processors, 7th Edition, Chapter 6, Conditional Processing*

Note:

Conditional and Unconditional Processing answers the following questions:

How do I write an IF statement in assembly language?

How do I write loops (for, while, do...while) statement in assembly language?

In this lecture will answer the first question.

Introduction:

- ❖ By default, the CPU loads and executes programs sequentially. But the current instruction might be conditional, meaning that it transfers control to a new location in the program based on the values of CPU status flags (Zero, Sign, Carry, etc.).
- ❖ Assembly language programs use conditional instructions to implement high-level statements such as IF statements and loops.
- ❖ Each of the conditional statements involves a possible transfer of control (jump) to a different memory address.
- ❖ A transfer of control, or branch, is a way of altering the order in which statements are executed.
- ❖ There are two basic types of transfers:
 - ✓ **Unconditional Transfer:**
 - Control is transferred to a new location in all cases; a new address is loaded into the instruction pointer, causing execution to continue at the new address.
 - The JMP instruction does this.

✓ **Conditional Transfer:**

- The program branches if a certain condition is true.
- A wide variety of conditional transfer instructions can be combined to create conditional logic structures.
- The CPU interprets true/false conditions based on the contents of the ECX and Flags registers

JMP Instruction:

- ❖ The JMP instruction causes an unconditional transfer to a destination, identified by a code label that is translated by the assembler into an offset.
- ❖ The syntax is:
 JMP destination ; *where destination is target-Label*
- ❖ When the CPU executes an unconditional transfer, the offset of destination is moved into the instruction pointer, causing execution to continue at the new location.

❖ **Example:**

```

top:
    .statements
    .
    jmp top

```

Conditional Jump:**Conditional Structures**

- ❖ There are no explicit high-level logic structures in the x86 instruction set, but you can implement them using a combination of comparisons and jumps.
- ❖ Two steps are involved in executing a conditional statement:
 - **First**, an operation such as CMP, AND, or SUB modifies the CPU status flags.
 - **Second**, a conditional jump instruction tests the flags and causes a branch to a new address.

Jcond Instruction

- ❖ A conditional jump instruction branches to a destination label when a status flag condition is *true*. Otherwise, if the flag condition is *false*, the instruction immediately following the conditional jump is executed.
- ❖ Syntax:
 Jcond destination

- ❖ CPU status flags are most commonly set by **arithmetic, comparison, and Boolean** instructions.

- ✓ Arithmetic instructions: Add, Sub, inc, dec and Neg
- ✓ Comparison instructions: Cmp and Test
- ✓ Boolean instructions: AND, OR, XOR and NOT

❖ **CMP Instruction:**

- ✓ Compares the destination operand to the source operand
 - ✓ Nondestructive subtraction of source from destination (destination operand is not changed)
- ✓ Syntax:

CMP destination, source

- ✓ The CMP (compare) instruction performs an implied subtraction of a source operand from a destination operand. Neither operand is modified
- ✓ CMP uses the same operand combinations as the MOV instruction
- ✓ The CMP instruction changes the Overflow, Sign, Zero, Carry, Auxiliary Carry, and Parity flags according to the value the destination operand would have had if actual subtraction had taken place.

- ✓ When two **unsigned operands** are compared, the Zero and Carry flags indicate the following relations between operands:

CMP Results	ZF	CF
Destination < source	0	1
Destination > source	0	0
Destination = source	1	0

- ✓ When two **signed operands** are compared, the Sign, Zero, and Overflow flags indicate the following relations between operands:

CMP Results	Flags
Destination < source	SF ≠ OF
Destination > source	SF = OF
Destination = source	ZF = 1

- ✓ **Example:** destination = source
 - mov ax,1000h
 - mov cx,1000h
 - cmp cx,ax ; ZF = 1 and CF = 0

- ✓ **Example:** destination < source
 - mov al,5
 - cmp al,10 ; ZF = 0 and CF = 1
- ✓ **Example:** destination > source
 - mov SI,105
 - cmp SI,0 ; ZF = 0, CF = 0
- ✓ CMP used to create conditional logic structures
- ✓ When follow CMP with a conditional jump instruction, the result is the assembly language equivalent of an **IF statement**
- ❖ Conditional jump instructions evaluate the flag states, using them to determine whether or not jumps should be taken
- ❖ Types of Conditional Jump Instructions
 - Jumps based on specific flag values
 - Jumps based on equality between operands or the value of (E)CX
 - Jumps based on comparisons of unsigned operands
 - Jumps based on comparisons of signed operands

Jumps Based on Specific Flag Values.

Mnemonic	Description	Flags / Registers
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

Jumps Based on Equality.

Mnemonic	Description
JE	Jump if equal ($leftOp = rightOp$)
JNE	Jump if not equal ($leftOp \neq rightOp$)
JCXZ	Jump if CX = 0
JECXZ	Jump if ECX = 0
JRCXZ	Jump if RCX = 0 (64-bit mode)

Jumps Based on Unsigned Comparisons.

Mnemonic	Description
JA	Jump if above (if $leftOp > rightOp$)
JNBE	Jump if not below or equal (same as JA)
JAЕ	Jump if above or equal (if $leftOp \geq rightOp$)
JNB	Jump if not below (same as JAЕ)
JB	Jump if below (if $leftOp < rightOp$)
JNAЕ	Jump if not above or equal (same as JB)
JBE	Jump if below or equal (if $leftOp \leq rightOp$)
JNA	Jump if not above (same as JBE)

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 $leftOp \geq rightOp$)
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 $leftOp \leq rightOp$)
JNG	Jump if not greater (same as JLE)

IF.....Then.....Command:

- ❖ General form of IF.....Then..... command:

```
IF condition is True then
    Execute True branch statements
End_IF
```

- ❖ Mean if condition is true will execute the statements else nothing will happened.

- ❖ **Example:**

Write part of code do the following:

```
IF AX < 0 then
    Replace AX with -AX
End_IF
```

Code in assembly language

IF...THEN.....ELSE.....ENDIF Command:

- ❖ General form of IF...THEN.....ELSE.....ENDIF command:

```
IF Condition is True then
    Execute True_Branch statements
ELSE
    Execute False_Branch statements
End_IF
```

- ❖ Mean if condition is true will execute the true statements else execute the false statements.

❖ **Example:**

Write part of code do the following:

```
IF  AL <=  BL THEN
    DISPLAY    AL
ELSE
    DISPLAY  BL
END_IF
```

Code in assembly language:

CASE Command:

- ❖ General form of Case command:

```
CASE      EXPRESSION
    VALUE_1  :      STATEMENT_1
    VALUE_2  :      STATEMENT_2
    :
    VALUE_N   :      STATEMENT_N
END_CASE
```

- ❖ Used when program has more paths can go throw.

- ❖ **Example:**

Write part of code do the following:

```
CASE AX
    < 0   :      PUT  -1   IN BX
    = 0   :      PUT   0   IN BX
    > 0   :      PUT   1   IN BX
END_CASE
```

Code in assembly language:

Example:

- ❖ Write part of code do the following:

```
CASE    AL
1,3:   DISPLAY "0"
2,4:   DISPLAY "E"
END_CASE
```

- ❖ Code in assembly language:

Example:

- ❖ Write part of code do the following:

```
Read a Character into AL
If ( 'A' <= character AND character <= 'Z' ) then
    Display character
End_IF
```

- ❖ Code in assembly language:

Example:

❖ Write part of code do the following:

Read character from keyboard into AL

IF (character = 'y' OR character = 'Y') then

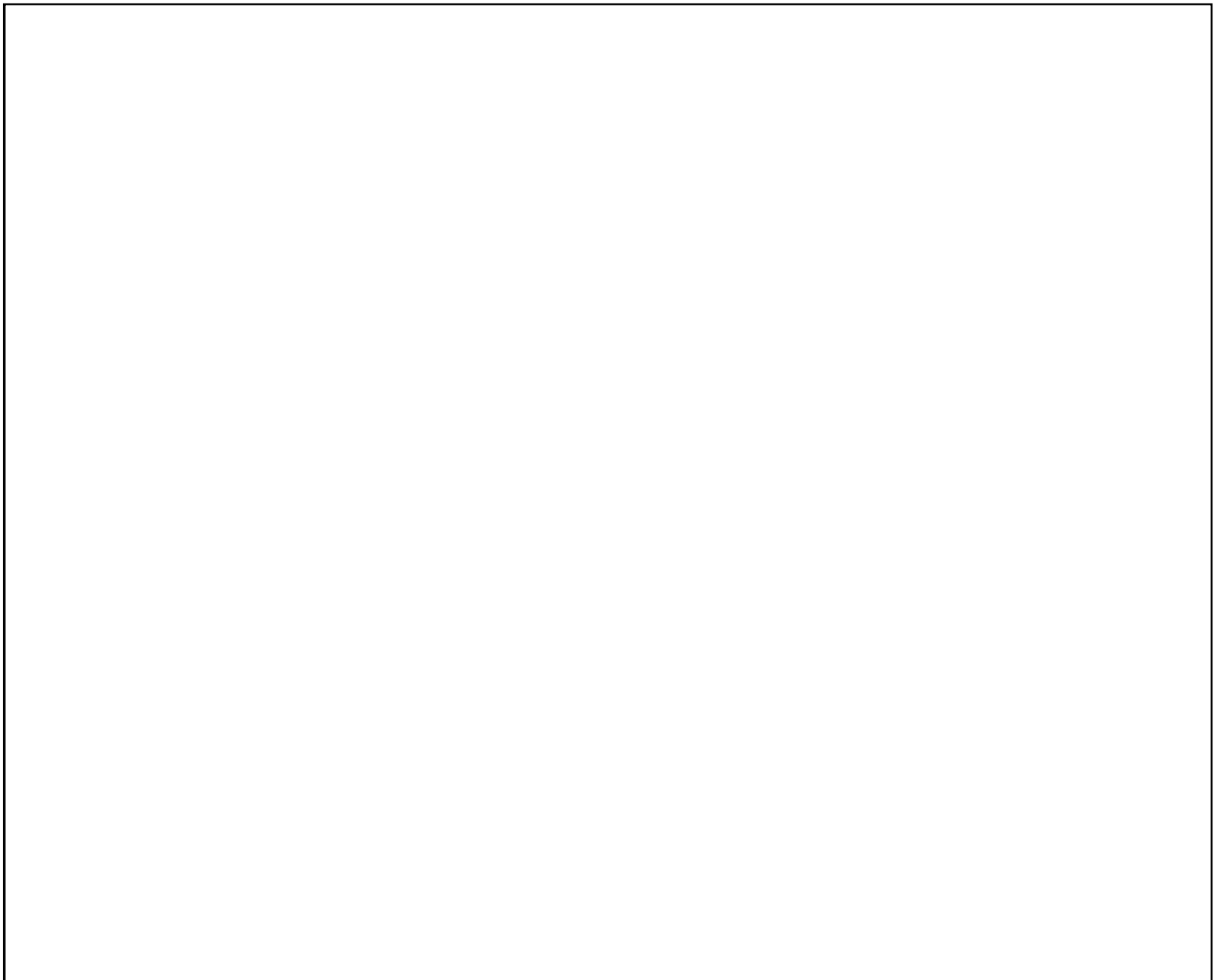
Display character

Else

Terminate the program

End_IF

❖ Code in assembly language:



ASCII Code (7-bit)

American Standard Code for Information Interchange

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

Source: www.LookupTables.com