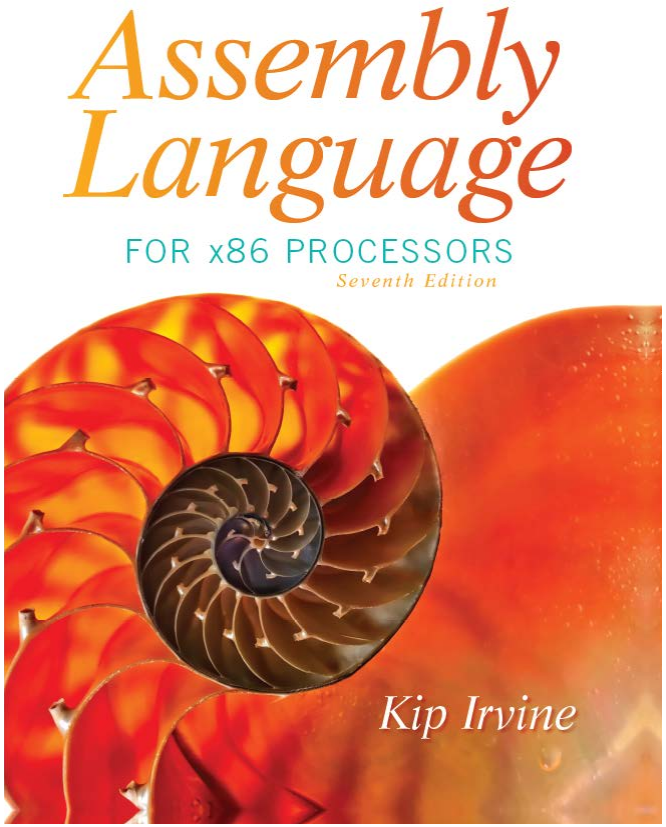
Assembly Language for x86 Processors

Seventh Edition



Chapter 6

Conditional Processing

Chapter Overview

- **Boolean and Comparison Instructions**
- Conditional Jumps
- Conditional Loop Instructions
- Conditional Structures
- Application: Finite-State Machines
- Conditional Control Flow Directives

Boolean and Comparison Instructions

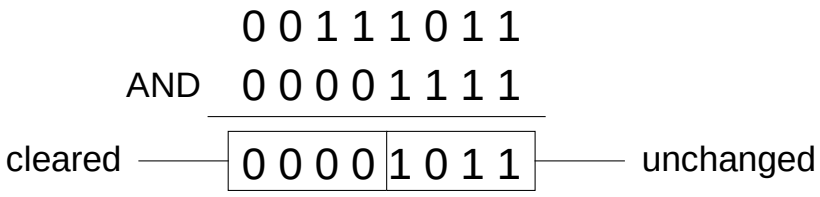
- AND Instruction
- OR Instruction
- XOR Instruction
- NOT Instruction
- Applications
- TEST Instruction
- CMP Instruction

AND Instruction

- Performs a Boolean AND operation between each pair of matching bits in two operands
- Syntax:

```
AND destination, source
```

(same operand types as MOV)



AND

x	y	x ∧ y
0	0	0
0	1	0
1	0	0
1	1	1

OR Instruction

- Performs a Boolean OR operation between each pair of matching bits in two operands
- Syntax:

```
OR destination, source
```

```
    0 0 1 1 1 0 1 1
OR  0 0 0 0 1 1 1 1
    -----------------
    0 0 1 1 1 1 1 1
```

unchanged — set

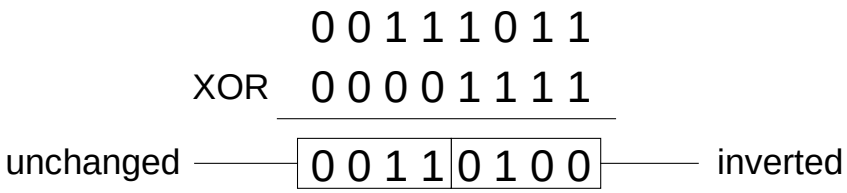
OR

x	y	x ∨ y
0	0	0
0	1	1
1	0	1
1	1	1

XOR Instruction

- Performs a Boolean exclusive-OR operation between each pair of matching bits in two operands
- Syntax:

```
XOR destination, source
```



XOR

x	y	x ⊕ y
0	0	0
0	1	1
1	0	1
1	1	0

XOR is a useful way to toggle (invert) the bits in an operand.

NOT Instruction

- Performs a Boolean NOT operation on a single destination operand
- Syntax:

NOT ***destination***

NOT 0 0 1 1 1 0 1 1

1 1 0 0 0 1 0 0 ——— inverted

NOT

X	¬X
F	T
T	F

Application

- Task: Convert a letter in AL from lowercase to uppercase.

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	&#127;	DEL

Application

‘a’ → 61h → 0 1 1 0 0 0 0 1 ← given

‘A’ → 41h → 0 1 0 0 0 0 0 1 ← target

- Solution: Use the AND instruction to clear bit 5.

```
mov al,'a'     ; AL = 01100001b
and al,11011111b     ; AL = 01000001b
```

Conditional Jumps

- Jumps Based On . . .
  - Specific flags
  - Equality
  - Unsigned comparisons
  - Signed comparisons
- Applications

Jcond Instruction

- A conditional jump instruction branches to a label when specific register or flag conditions are met
- Specific jumps:

  JB, JC - jump to a label if the Carry flag is set

  JE, JZ - jump to a label if the Zero flag is set

  JS - jump to a label if the Sign flag is set

  JNE, JNZ - jump to a label if the Zero flag is clear

  JECXZ - jump to a label if ECX = 0

Application--1

- Task: Jump to a label if an integer is even.
- Solution: AND the lowest bit with a 1. If the result is Zero, the number was even.

```
mov ax,wordVal
and ax,1      ; low bit set?
jz  EvenValue ; jump if Zero flag set
```

Application--2

- Task: Jump to a label if the value in AL is not zero.
- Solution: OR the byte with itself, then use the JNZ (jump if not zero) instruction.

```
or  al,al
jnz IsNotZero ; jump if not zero
```

ORing any number with itself does not change its value.

TEST Instruction

- Performs a nondestructive AND operation between each pair of matching bits in two operands
- TEST sets the zero flag, ZF, when the result of the AND operation is zero. If two operands are equal, their bitwise AND is zero when both are zero. TEST also sets the sign flag, SF, when the most significant bit is set in the result, and the parity flag, PF, when the number of set bits is even

TEST -- Examples

- Example1: jump to a label if either bit 0 or bit 1 in AL is set.

```
test al,00000011b
jnz  ValueFound
```

- Example2: jump to a label if neither bit 0 nor bit 1 in AL is set.

```
test al,00000011b
jz   ValueNotFound
```

CMP Instruction (1 of 3)

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

  CMP *destination, source*
- Example: destination == source

```
mov al,5
cmp al,5        ; Zero flag set
```

- Example: destination < source

```
mov al,4
cmp al,5        ; Carry flag set
```

CMP Instruction (2 of 3)

- Example: destination > source

```
mov al,6
cmp al,5        ; ZF = 0, CF = 0
```

(both the Zero and Carry flags are clear)

CMP Instruction (3 of 3)

- The comparisons shown here are performed with signed integers.
- Example: destination > source

```
mov al,5
cmp al,-2     ; Sign flag == Overflow flag
```

- Example: destination < source

```
mov al,-1
cmp al,5      ; Sign flag != Overflow flag
```

Applications (1 of 4)

- Task: Jump to a label if unsigned EAX is greater than EB X
- Solution: Use CMP, followed by J A

```
cmp eax,ebx
ja  Larger
```

- Task: Jump to a label if signed EAX is greater than EBX
- Solution: Use CMP, followed by JG

```
cmp eax,ebx
jg  Greater
```

Applications (2 of 4)

- Jump to label L1 if unsigned EAX is less than or equal to Val1

```
cmp eax,Val1
jbe L1              ; below or equal
```

- Jump to label L1 if signed EAX is less than or equal to Val1

```
cmp eax,Val1
jle L1
```

Applications (3 of 4)

- Compare unsigned AX to BX, and copy the larger of the two into a variable named Large

```
    mov Large,bx
    cmp ax,bx
    jna Next
    mov Large,ax
Next:
```

- Compare signed AX to BX, and copy the smaller of the two into a variable named Small

```
    mov Small,ax
    cmp bx,ax
    jnl Next
    mov Small,bx
Next:
```

Applications (4 of 4)

- Task: Jump to label L1 if bits 0, 1, and 3 in AL are all set.
- Solution: Clear all bits except bits 0, 1,and 3. Then compare the result with 00001011 binary.

```
and al,00001011b     ; clear unwanted bits
cmp al,00001011b     ; check remaining bits
je  L1 ; all set? jump to L1
```