Cheatography

x86 assembly

by Kzderrick via cheatography.com/26001/cs/7101/

Jump Instructions

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

Data Types

BYTE 8-bit unsigned integer.

SBYTE 8-bit signed integer.

WORD 16-bit unsigned integer

SWORD 16-bit signed integer

DWORD 32-bit unsigned integer.

SDWORD 32-bit signed integer.

Arrays - (from book pg 120-124)

```
.data
    arrayB BYTE 10h,20h,30h
.code
    mov ESI,OFFSET arrayB
    mov AL,[ESI]
    inc ESI
--------OFFSET-------------
.data
    arrayW WORD 1000h,2000h,3000h
.code
    mov ESI,OFFSET arrayW
    mov AX,[ESI]
```

LOOP

```
.data
    mov ax,0
    mov ecx,5
.code
    L1:
        inc ax
    loop L1
```

ASCII

Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
32	20	040		Space	64	40	100	@	@	96	60	140	`	`
33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

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.

destination < source ....carry flag set

destination > source.... ZF=0, CF=0

Registers

32-Bit	16-Bit	8-bit (high)	8 bit (low)
EAX	AX	AH	AL
EBX	BX	BH	BL
ECX	CX	CH	CL
EDX	DX	DH	DL
ESI	SI		
EDI	DI		
EBP	BP		
ESP	SP		

FLAGS

• The Carry flag (CF) is set when the result of an unsigned arithmetic operation is too large to fit into the destination.

• The Overflow flag (OF) is set when the result of a signed arithmetic operation is too large or too small to fit into the destination.

• The Sign flag (SF) is set when the result of an arithmetic or logical operation generates a negative result.

FLAGS (cont)

• The Zero flag (ZF) is set when the result of an arithmetic or logical operation generates a result of zero.

• The Auxiliary Carry flag (AC) is set when an arithmetic operation causes a carry from bit 3 to bit 4 in an 8-bit operand.

• The Parity flag (PF) is set if the least-significant byte in the result contains an even number of 1 bits. Otherwise, PF is clear. In general, it is used for error checking when there is a possi- bility that data might be altered or corrupted.

TEST Instruction

Performs a nondestructive AND operation between each pair of matching bits in two operands.No operands are modified, but the Zero flag is affected.

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

test al,00000011b

jnz ValueFound

ROT SHIFT

SAL (shift arithmetic left) is identical to SHL.

SAR (shift arithmetic right) performs a right arithmetic shift on the destination operand.

ROL (rotate) shifts each bit to the left

The highest bit is copied into both the Carry flag and into the lowest bit

No bits are lost

ROR (rotate right) shifts each bit to the right

The lowest bit is copied into both the Carry flag and into the highest bit

No bits are lost

RCL (rotate carry left) shifts each bit to the left

Shifts a destination operand a given number of bits to the right

The bit positions opened up by the shift are filled by the least significant bits of the source operand

The source operand is not affected

Copies the Carry flag to the least significant bit

By **Kzderrick**

cheatography.com/kzderrick/

Not published yet.

Last updated 13th May, 2016.

ROT SHIFT (cont)

Copies the most significant bit to the Carry flagRCR (rotate carry right) shifts each bit to the right
Copies the Carry flag to the most significant bit
Copies the least significant bit to the Carry flag
Shifts a destination operand a given number of bits to the left
The bit positions opened up by the shift are filled by the most significant bits of the source operand
The source operand is not affected
Syntax:
SHLD destination, source, count

OR, XOR, AND, NOT

OR destination, source. 0,0=0. 0,1=1, 1,0=1. 1,1 = 1
XOR destination, source. 0,0=0. 0,1=1, 1,0=1. 1,1 = 0
NOT destination...invert all
AND destination, source 0,0=0. 0,1=0, 1,0=0. 1,1 = 1

Procedures

```
SumOf PROC
    add EAX, EBX
    add EAX, ECX
    ret
SumOf ENDP
END main
```

Irvine 32 lib procedures

```
DumpMemm
  mov ESI,OFFSET array
  mov ECX,LENGTHOF array
  mov EBX,TYPE array
  call DumpMem
ReadChar
    char BYTE ?
    .code
    call ReadChar
    mov char, AL
Random32
```

Irvine 32 lib procedures (cont)

```
  call Random32
  mov randVal, EAX
ReadDec ;same for dec, hex, int
  intVal DWORD ?
.code
  call ReadDec
  mov intVal,eax
ReadString
.data
  buffer BYTE 21 DUP(0) ; input
buffer
  byteCount DWORD ? ; holds
counter
.code
  mov EDX,OFFSET buffer ; point to
the buffer
  mov ECX,SIZEOF buffer ; specify
max characters
  call ReadString ; input the
string
  mov byteCount, EAX ; number of
characters

WriteChar
  mov AL, 'A'
  call WriteChar
WriteDec,hex,int
  mov eax,295
  call WriteDec
WriteString
  .data
  prompt BYTE "Enter your name: ",0
.code
  mov edx,OFFSET prompt
  call WriteString
```

Unsigned Comparison

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)

PUSHAD POPAD

The PUSHAD instruction pushes all of the 32-bit general-purpose registers on the stack in the following order: EAX, ECX, EDX, EBX, ESP (value before executing PUSHAD), EBP, ESI, and EDI.The POPAD instruction pops the same registers off the stack in reverse order.

Signed

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)

MUL IMUL

```
In 32-bit mode, MUL (unsigned
multiply) instruction multiplies an
8-, 16-, or 32-bit operand by
either AL, AX, or EAX
.data
val1 WORD 2000h
val2 WORD 100h
.code
mov ax,val1
mul val2 ; DX:AX = 00200000h, CF=1
IMUL (signed integer multiply )
multiplies an 8-, 16-, or 32-bit
signed operand by either AL, AX, or
EAX
Preserves the sign of the product
by sign-extending it into the upper
half of the destination register
mov eax,4823424
mov ebx,-423
imul ebx ; EDX:EAX =
FFFFFFFF86635D80h, OF=0
```