Addition

Mnemonic	Operands	Size	Operation	Flags
ADD	destination, source	B/W	(dest) ← (dest) + (src)	AF, CF, OF, PF, SF, ZF

Examples

```
add     ax, 0f123h
add     dx, si
add     ax, [5678h]
add     w[3215h], 8adch
add     [3598h], di
add     bx, [bp]
```

dx = dx + si

addition

0008h

```
sub     ax, 0f123h
sub     ax, [bx]
```

subtraction

MULTIPLICATION

Mnemonic	Operands	Size	Operation
MUL	SOURCE	B	AX ← (AL) · (SRC)
MUL	SOURCE	W	DX:AX ← (AX) * (SRC)

Flags:
CF OF

Addressing Modes:

register, register
register, memory
↑ destination
↑ source

Examples:

mul bx
mul cl
mul b[1234h]
mul w[di]

DX:AX = AX*BX
AX = AL*CL

- If the most significant part of result is > 0, both CF and OF are set to 1
  = 0, " " " " " 0

DIVISION

Mnemonic	Operands	Size	Operation
DIV	SOURCE	B	(AX) / (SRC)
DIV	source	W	(DX:AX) / (SRC)

for B:
Results:
- quotient: AL
- remainder: AH

quotient: AX
remainder: DX

Addressing modes:

register, register
register, memory
↑ destination ↑ source

Examples:

```
div bx
div cl
div w[1234h]
div b[di]
```

Logical Operations

Mnemonic	Operand	Size	Operation	Flags
AND	destination, source	B/W	(DST) ← (DST AND SRC)	CF = 0 OF = 0 AF = ? PF SF ZF

Addressing Modes

register, immediate
register, register
register, memory
memory, immediate
memory, register

Examples

```
and   dh, 0fh
and   ax, cx
and   bl, [123h]
and   [5000h], cx
and   ax, [bx]
```

```
not   dh
not   ax
not   b[1234h]
not   w[1234h]
not   b[di]
```

bitwise NOT

```
or    dh, 0fh
or    ax, cx
or    ax, [bp]
```

bitwise OR

```
rcl   ax, 1
rcl   w[bx], cl
```

rotate destination with carry flag

```
xor   dh, 0fh
xor   ax, cx
```

bitwise exclusive or

Compare Instruction (cont)

condition	flag Conditions	
(dest) > (src)	SF = OF, ZF = 0	signed numbers
(dest) < (src)	SF <> OF, ZF = 0	
(dest) = (src)	ZF = 1	

(dest) > (src)	CF = 0, ZF = 0	unsigned numbers
(dest) < (src)	CF = 1, ZF = 0	
(dest) = (src)	ZF = 1	

JUMP Instructions

same instr.

- JC — jump on carry
- JB — jump on below
- JNAE — jump on NOT above or equal

Example:

```
JC  lp4
```

- displacement is short range

Conditional Jumps (cont)

JE	jump on equal	short range jump
JZ	jump on zero	
JNC	jump on no carry	
JAE	jump on above or equal	
JNB	jump on not below	

Unconditional Jump

JMP	short-label	(IP) ← short-label
JMP	near-label	(IP) ← near-label
JMP	far-label	(CS:IP) ← far label

examples

```
jmp here
jmp short here
jmp far here
```

JUMPS

Jump Range	Jump Distance in Bytes
Short	Forward 127, backward 128
Near	Anywhere within a 64K segment
Far	Anywhere within memory (Jump to other segments possible

- Jumps take place based on the condition of various status flags
- Instructions are available to compare, and Jump
- Compare Instruction:

Mnemonic	Operands	Size	Operation	Flags
CMP	destination, source	B/W	(dest) - (src)	AF, CF, OF, PF, SF, ZF

Examples:

```
cmp  ah, 0f1h
cmp  dx, si
cmp  w[si], 123h
```

Dos Interrupt Services

- **Interrupt:**
  - suspension of a process caused by an event
  - process can be resumed
  - events can be external, internal to CPU, software, hardware
  - example:
    - → A peripheral can generate an electrical signal called an interrupt to set a hardware flag within the processor
    - → "Timer" device can generate an interrupt after a specified interval of time.

- **Interrupt Service Routine:**
  - If the processor detects an interrupt, it saves the current value of CS:IP and loads a new value which is the address of special function called "Interrupt Service Routine".

- **INT** : Dos interrupt instruction

- **Interrupt Vector:** holds addresses of interrupt service routines

INT Instruction

INT type

directs CPU to cause a software interrupt sequence to be performed.

operations

1. CPU stores the flag register
2. CPU pushes CS:IP on the stack
3. CPU replaces the CS:IP with the contents of the interrupt vector entry (indexed by Type)
4. Program execution resumes at the interrupt service routine (handler)
5. A return at the end of handler, causes the original program execution to resume

Examples of INT

INT 21h dos interrupt

Depending on what code we put in AH DOS performs some action:

02h → display a character given in DL

0Ah → input a string

Read and print a character 5 times

```
code segment
  mov cx,5d ; set counter to 5
more:
  mov  ah,01h ;
  int  21h
  mov  dl,al
  mov  ah,02h ; display a character
  int  21h
  dec  cx
  jnz  more
  int  20h ; exit to dos
code ends
```

Hello World Program

```
code segment
  mov bx,msg1
  mov cx,11d
  mov ah,02h
more:
  mov  dl,[bx]
  int  21h
  inc  bx
  dec  cx
  jnz  more
  int  20h ; exit to dos
msg1:
  db 'hello world'
code ends
```

Compiling High Level Statements

The program **val = (461*y) div 4 + (200*m+2) div 5 + d** is translated to the following instructions:

	Abstract Instruction	**A86 Instuctions**
0	push-addr-var val	PUSH offset VAL
1	push-num 461	PUSH 461
2	push-val-var y	PUSH Y
3	*	POP CX POP AX MULT CX PUSH AX
4	push-num 4	PUSH 4
5	div	MOV DX,0 POP CX POP AX DIV CX PUSH AX
6	push-num 200	PUSH 200
7	push-val-var m	PUSH M
8	*	POP CX POP AX MULT CX PUSH AX
9	push-num 2	PUSH 2
10	+	POP CX POP AX ADD AX,CX PUSH AX
11	push-num 5	PUSH 5
12	div	MOV DX,0 POP CX POP AX DIV CX PUSH AX
13	+	POP CX POP AX ADD AX,CX PUSH AX
14	push-val-var d	PUSH D
15	+	POP CX POP AX ADD AX,CX PUSH AX
16	assign	POP AX POP BX MOV [BX],AX
17	stop	INT 20h

IF Statement

if *expr* then *stm*

will be translated as:

code for *expr*
POP AX CMP AX,0 JZ OUTLABEL
code for *stm*
OUTLABEL:

WHILE LOOP

while expression do stm

will be translated as follows:

TESTLABEL:
code for *expr*
POP AX CMP AX,0 JZ OUTLABEL
code for *stm*
JMP TESTLABEL
OUTLABEL:

```
begin
   read a    ;
   while( 100 - a ) begin
      print a  ;
      a = a + 1
   end
end
```

```
code segment
  call myread       ; read a
  mov  va,cx        ;
labl1:              ; while  begin
  push 100d         ;
  push va           ;
  pop  cx           ;
  pop  ax           ;
  sub  ax,cx        ;
  push ax           ;
  pop  ax           ;
  cmp  ax,0         ;
  jz   labl2        ;
  push va           ;
  pop  ax           ;
  call myprint      ;
  push offset va    ;
  push va           ;
  push 1            ;
  pop  cx           ;
  pop  ax           ;
  add  ax,cx        ;
  push ax           ;
  pop  ax           ;
  pop  bp           ;
  mov  [bp],ax      ;
  jmp  labl1        ; jump to while
beginnning
labl2:              ;
  int  20h        ; exit to dos
```

```
.....
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;reads a nonnegative integer
and puts
;it in cx register
.....
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
myread:
   MOV  CX,0
morechar:
   mov  ah,01h
   int  21H
   mov  dx,0
   mov  dl,al
   mov  ax,cx
   cmp  dl,0D
   je   myret
   sub  dx,48d
   mov  bp,dx
   mov  ax,cx
   mov  cx,10d
   mul  cx
   add  ax,bp
   mov  cx,ax
   jmp  morechar
myret:
   ret
```

```
.....
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; number in AX register is
printed
.....
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
myprint:
   mov    si,10d
   xor    dx,dx
   push   ''    ; push newline
   mov    cx,1d
nonzero:
   div    si
   add    dx,48d
   push   dx
   inc    cx
   xor    dx,dx
   cmp    ax,0h
   jne    nonzero
   writeloop:
   pop    dx
   mov    ah,02h
   int    21h
   dec    cx
   jnz    writeloop
   ret
.....
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Variables are put in this area
.....
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
va      dw ?  ;  variable a
code ends
```