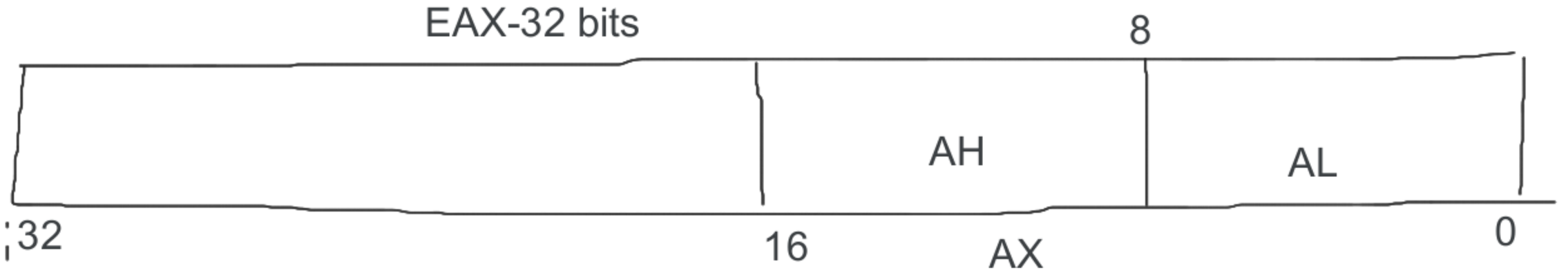


| OR /
IN OPCODE

88|77

Integer Addition and Subtraction Instructions

Courtesy: UMBC and JBLearning

Basic Instructions

- Add
- Sub
- Inc
- Dec
- Neg

FLAGS Register

Bit #	Abbreviation	Description	Category
0	CF	Carry flag	Status
1		Reserved	
2	PF	Parity flag	Status
3		Reserved	
4	AF	Adjust flag	Status
5		Reserved	
6	ZF	Zero flag	Status
7	SF	Sign flag	Status
8	TF	Trap flag (single step)	Control
9	IF	Interrupt enable flag	Control
10	DF	Direction flag	Control
11	OF	Overflow flag	Status
12-13	IOPL	I/O privilege level (286+ only), always 1 on 8086 and 186	System
14	NT	Nested task flag (286+ only), always 1 on 8086 and 186	System
15		Reserved, always 1 on 8086 and 186, always 0 on later models	

EFLAGS, RFLAGS Register

Bit #	Abbreviation	Description	Category
16	RF	Resume flag (386+ only)	System
17	VM	Virtual 8086 mode flag (386+ only)	System
18	AC	Alignment check (486SX+ only)	System
19	VIF	Virtual interrupt flag (Pentium+)	System
20	VIP	Virtual interrupt pending (Pentium+)	System
21	ID	Able to use CPUID instruction (Pentium+)	System
22-31		RESERVED	
32-63		RESERVED	

FLAG register is of 16 bits and each of the bit indicate particular flag
EFLAGS - 32 bit ----> first 16 bit are same as flag reg(16 bit)

`add` Instruction

- Format: `add` *`destination, source`*
- The integer at *source* is added to the integer at *destination* and the sum replaces the old value at *destination.*
- Source content is not altered.
- SF, ZF, OF, CF, PF, and AF flags are set according to the value of the result of the operation.
  - Example: CF = 1 if there is a carry out of the sum.

For add instruction

SF----->sign flag
ZF----->Zero flag
OF----->Overflow flag
CF----->Carry out flag
PF----->Parity flag
AF----->Adjust flag

SF
After any operation if the MSB is 1, then it indicates that the number is negative. And this flag is set to 1

ZF
If the total register is zero, then only the Z flag is set

PF
This is even parity flag. When result has even number of 1, it will be set to 1, otherwise 0 for odd number of 1s

CY
This is carry bit. If some operations are generating carry after the operation this flag is set to 1

OF
The overflow flag is set to 1 when the result of a signed operation is too large to fit.

Addition Example

- *Before*
  EAX: 00000075
  ECX: 000001A2
- *Instruction*
  `add  eax, ecx`
- *After*
  EAX: 00000217
  ECX: 000001A2
  SF=0 ZF=0 CF=0 OF=0

EAX--> 00000075
ECX--> 000001A2
---

2 1 7

7+A-->7+10-->11

`sub` Instruction

- Format: `sub` *`destination, source`*
- The integer at *source* is subtracted from the integer at *destination* and the difference replaces the old value at *destination*.
- Source content is not altered.
- SF, ZF, OF, CF, PF, and AF flags are set according to the value of the result of the operation.
  - Example: ZF = 1 if the difference is zero.

Subtraction Example

- *Before*
  doubleword at Dbl: 00000100
- *Instruction*
  ```
  sub   Dbl, 2
  ```
- *After*
  Dbl: 000000FE
  SF=0 ZF=0 CF=0 OF=0

Instruction Encoding

- Opcode depends on operand types
- The *ModR/M* byte distinguishes between
  - operand types
  - `add`, `sub` and other operations for certain operand types
- A small immediate operand is sometimes encoded as a byte even in a 32-bit instruction.

Instruction Encoding

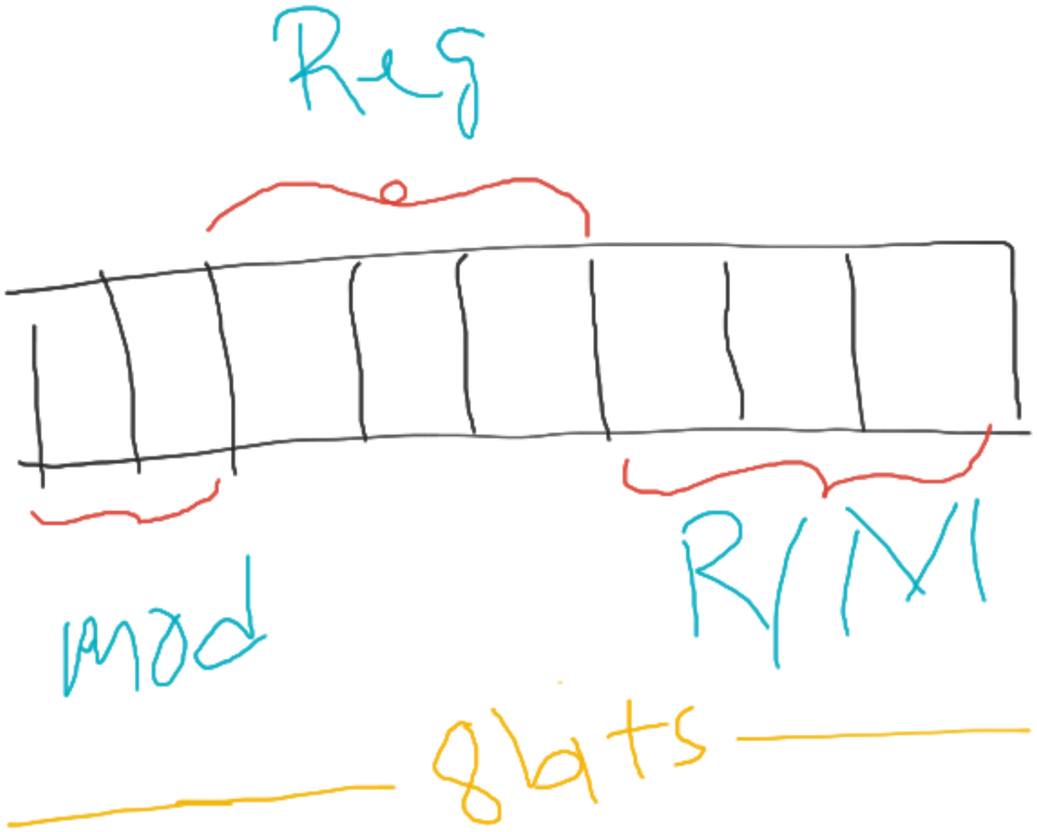
- 66 prefix : for word operands
- 4x prefix: for 64 bit

Dest	Src	Add	Sub	#bytes
Register 8	Immediate byte	80	80	3
Register 16	Immediate byte	83	83	3
Register 32	Immediate byte	83	83	3
Register 16	Immediate word	81	81	4
Register 32	Immediate doubleword	81	81	6
AL	Immediate byte	04	2C	2
AX	Immediate word	05	2D	3
EAX	Immediate doubleword	05	2D	5

ModR/M->Reg Field spec

- Distinguishes between instructions for the same opcode

	000	001	010	011	100	101	110	111
80,81	ADD	OR	ADC	SBB	AND	SUB	XOR	CMP
D0,D1	ROL	ROR	RCL	RCR	SHL	SHR		SAR
F6,F7	TEST		NOT	NEG	MUL	IMUL	DIV	IDIV
FE,FF	INC	DEC					PUSH	



Instruction Encoding - Examples

```
00000000  80 C2 14      add DL, 20
00000003  80 EA 1E      sub DL, 30
00000006  66| 83 C3 14      add BX, 20
0000000A  66| 83 EB 1E      sub BX, 30
0000000E  83 C1 64      add ECX, 100
00000011  83 E9 19      sub ECX, 25
00000014  81 C1 000003FF    add ECX, 1023
0000001A  81 E9 000003FF    sub ECX, 1023
00000020  81 C2 009CFB5D    add EDX, 10287965
00000026  81 EA 009CFB5D    sub EDX, 10287965

0000002C  04 14         add AL, 20
0000002E  2C 14         sub AL, 20
00000030  66| 05 012C       add AX, 300
00000034  66| 2D 012C       sub AX, 300
00000038  05 00233EEB       add EAX, 2309867
0000003D  2D 00233EEB       sub EAX, 2309867
```

ebx-32 bit - double word

bx- 16bit- word

c2- 11 000 010

EA- 11 101 010

Increment and Decrement Instructions

- `inc` *`destination`*
  - Adds 1 to *destination*
- `dec` *`destination`*
  - Subtracts 1 from *destination*
- Each sets same flags as `add` or `sub` except for CF, which isn't changed

Instruction Encoding

- 66 prefix : for word operands
- 4x prefix: for 64 bit

Dest	Inc	Dec	#bytes
Register 8	FE	FE	2
AX	40	48	1
CX	41	49	1
DX	42	4A	1
BX	43	4B	1
SP	44	4C	1
Mem byte	FE	FE	2+
Mem word	FF	FF	2+
Mem DWORD	FF	FF	2+

Instruction Encoding

```
00000042  FE C0          inc AL
00000044  FE C8          dec AL
00000046  FE C1          inc CL
00000048  FE C9          dec CL

0000004A  66| 40         inc AX
0000004C  66| 48         dec AX
0000004E  66| 41         inc CX
00000050  66| 49         dec CX
00000052  66| 42         inc DX
00000054  66| 4A         dec DX
00000056  66| 43         inc BX
00000058  66| 4B         dec BX
0000005A  66| 44         inc SP
0000005C  66| 4C         dec SP
0000005E  66| 45         inc BP
00000060  66| 4D         dec BP
```

```
00000062  40             inc EAX
00000063  48             dec EAX
00000064  43             inc EBX
00000065  4B             dec EBX

00000066  FE 05 000001CE R inc bytenum
0000006C  FE 0D 000001CE R dec bytenum
00000072  66| FF 05        inc wnum
      000001D3 R
00000079  66| FF 0D        dec wnum
      000001D3 R
00000080  FF 05 000001CF R inc dwnum
00000086  FF 0D 000001CF R dec dwnum
```

`neg` Instruction

- `neg` *`destination`*
- Negates (takes the 2's complement of) its operand
  - A positive value gives a negative result
  - A negative value will become positive
  - Zero remains o
- Affects same flags as `add` and `sub`

Instruction Encoding

- 66 prefix : for word operands
- 4x prefix: for 64 bit

Dest	Op	#bytes
Register 8	F6	2
Register 16	F7	2
Register 32	F7	2
Mem byte	F6	2+
Mem word	F7	2+
Mem DWORD	F7	2+
Register 64	F7	2
Mem QWORD	F7	2+

Instruction Encoding

```
0000008C  F6 D8          neg AL
0000008E  F6 DE          neg DH

00000090  66| F7 D8          neg AX
00000093  66| F7 D9          neg CX
00000096  66| F7 DA          neg DX
00000099  66| F7 DD          neg BP

0000009C  F7 D8          neg EAX
0000009E  F7 DB          neg EBX

000000A0  8D 1D 000001CE R lea EBX, bytenum
000000A6  8D 0D 000001D3 R lea ECX, wnum
000000AC  8D 15 000001CF R lea EDX, dwnum

000000B2  F6 1D 000001CE R neg bytenum
000000B8  F6 1B          neg BYTE PTR [EBX]

000000BA  66| F7 1D          neg wnum
      000001D3 R
000000C1  66| F7 19          neg WORD PTR [ECX]

000000C4  F7 1D 000001CF R neg dwnum
000000CA  F7 1A          neg DWORD PTR [EDX]
```

memory direct

register indirect

```
.data
wordop word 100
```

1>determine opcode and number of bytes of instruction

add ax, var

solution:

```
.data
var word 10

.code

add ax,var
```

opcode:66 03

bytes: 7 bytes

https://en.wikipedia.org/wiki/FLAGS_register

2> determine after values and SF,ZF,OF,CF

Before values of eax and ebx

EAX: FF 0D F9 75

EBX:01 C0 A1 92

Instruction

ADD EAX,EBX

solution:

```
mov eax, 001C0A192H
mov ebx,0FFODF975H
add  eax,ebx
```

after values eax=0CE9B07,ebx=01C0A192

sf=0

zf=0

of=0

cf=1