80x86 Assembly Language SUMMARY

Hexadecimal System

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

- It uses 16 digits
- "h" at the end indicates a hexadecimal number

$$\begin{tikzcd}\boxed{1234h} \arrow[d, leftrightarrow] \\ \boxed{1\text{x}16^3+2\text{x}16^2+3\text{x}16^1+4\text{x}16^0=4660}\end{tikzcd}$$

Hexadecimal System

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

- Can be used as abbreviation for binary

$$\begin{tikzcd} \boxed{1001\ 0101\ 1100} \arrow[d, leftrightarrow] \\ \boxed{95Ch} \end{tikzcd}$$

$$\begin{tikzcd} \boxed{DEADBEEFh} \arrow[d, leftrightarrow] \\ \boxed{?} \end{tikzcd}$$

Truth Table

- is used to describe Boolean functions
  - lists all possible values of the inputs to the function

A	$\underline{A}$
1	0
0	1

NOT

A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

AND

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

OR

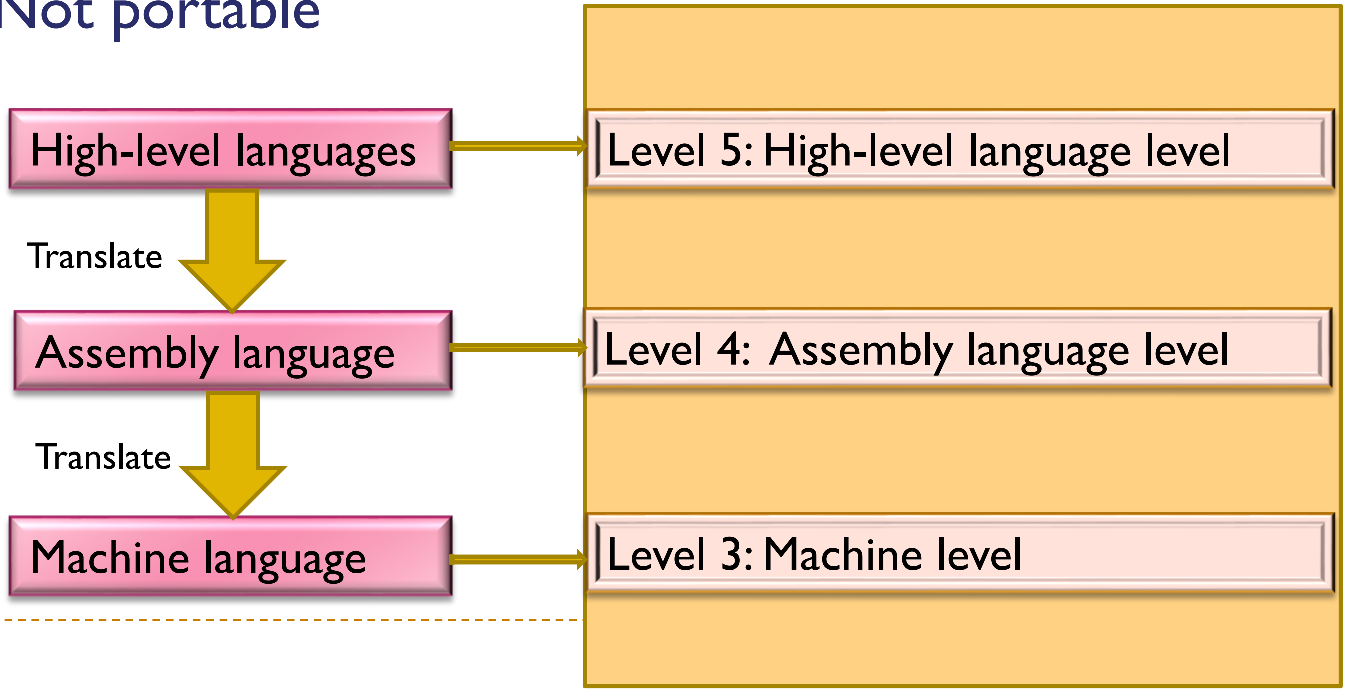
A	B	AXB
0	0	0
0	1	1
1	0	1
1	1	0

XOR

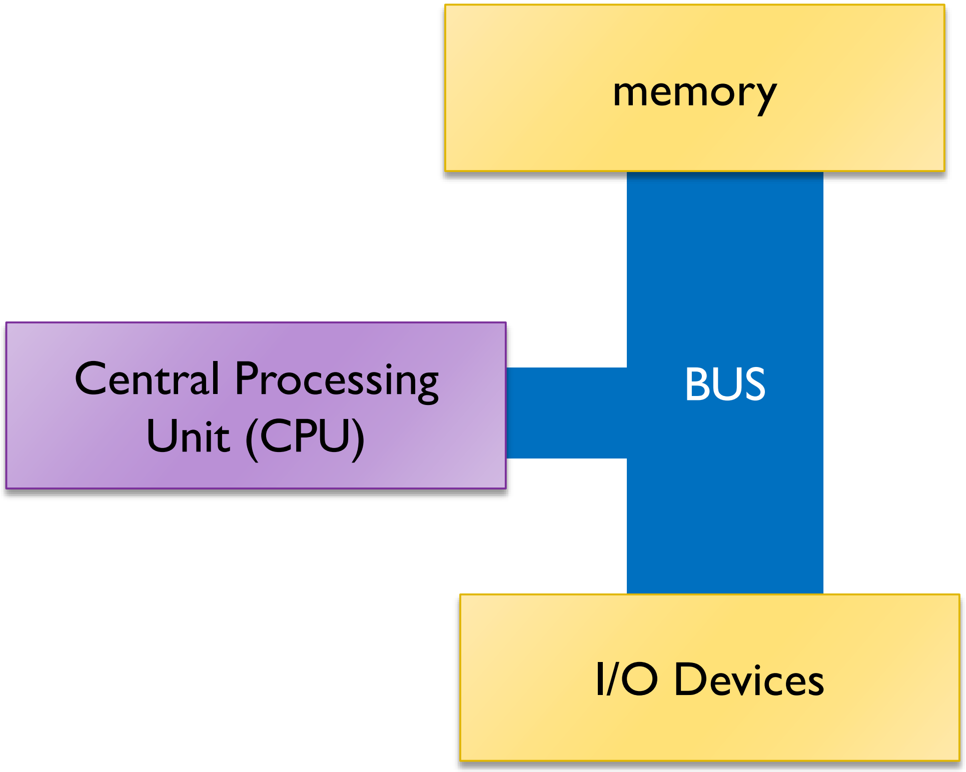
- Need 2^n rows in a truth-table for a function of *n* variables

What is Assembly Language

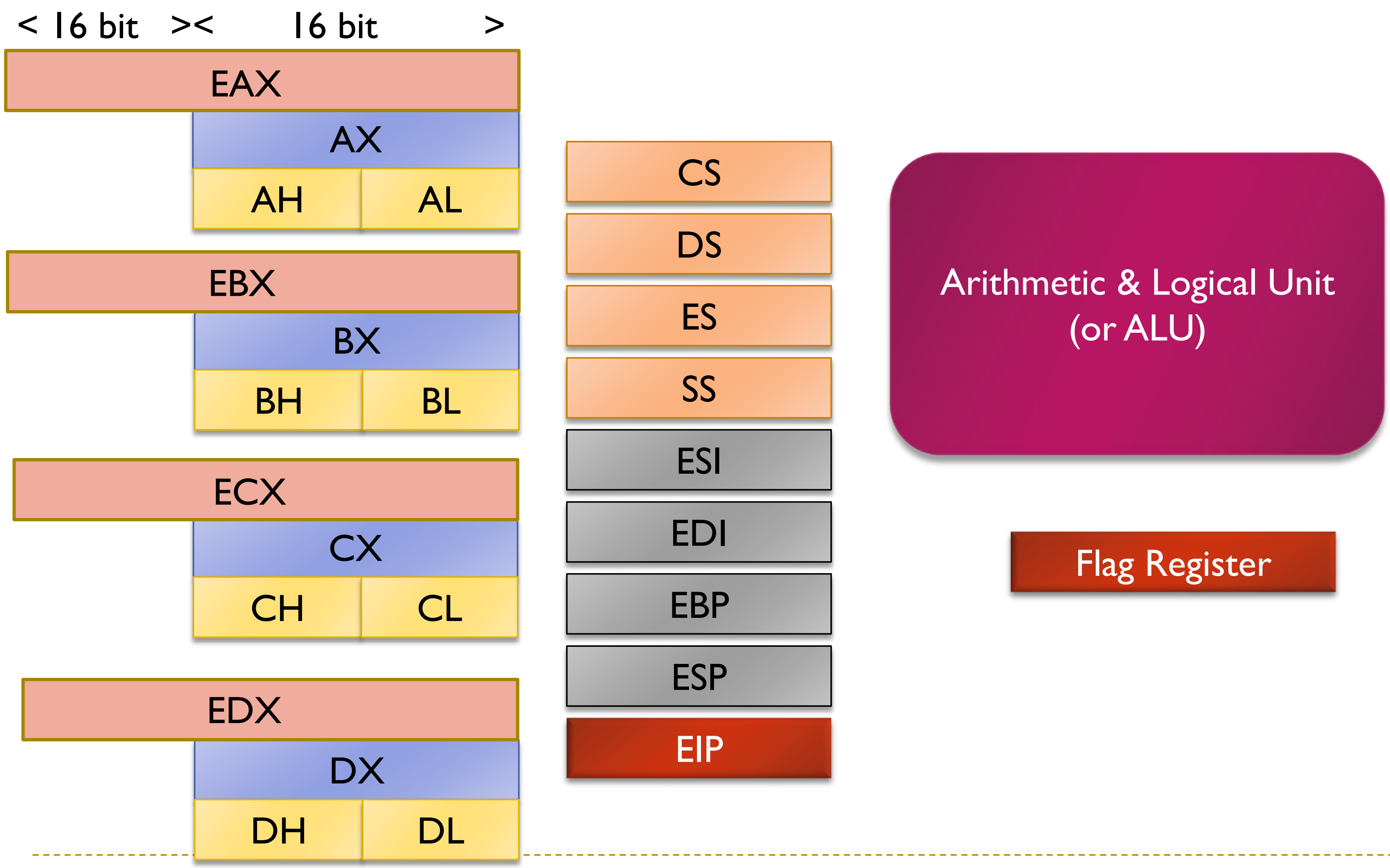
- a low level programming language.
  - Symbolic representation of machine binary code used for programming a specific computer architecture.
  - Based on instructions, registers, memory locations and some other features defined by the hardware manufacturer
  - Not portable



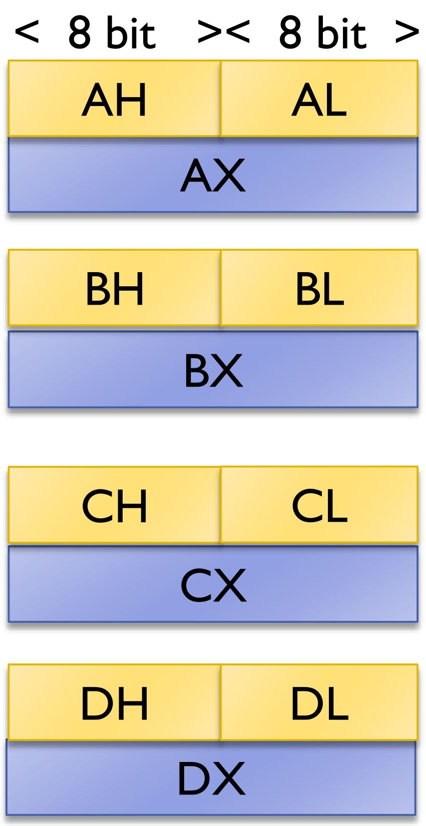
Simple Computer Model



Inside the CPU (80x86 CPU)



General Purpose Registers

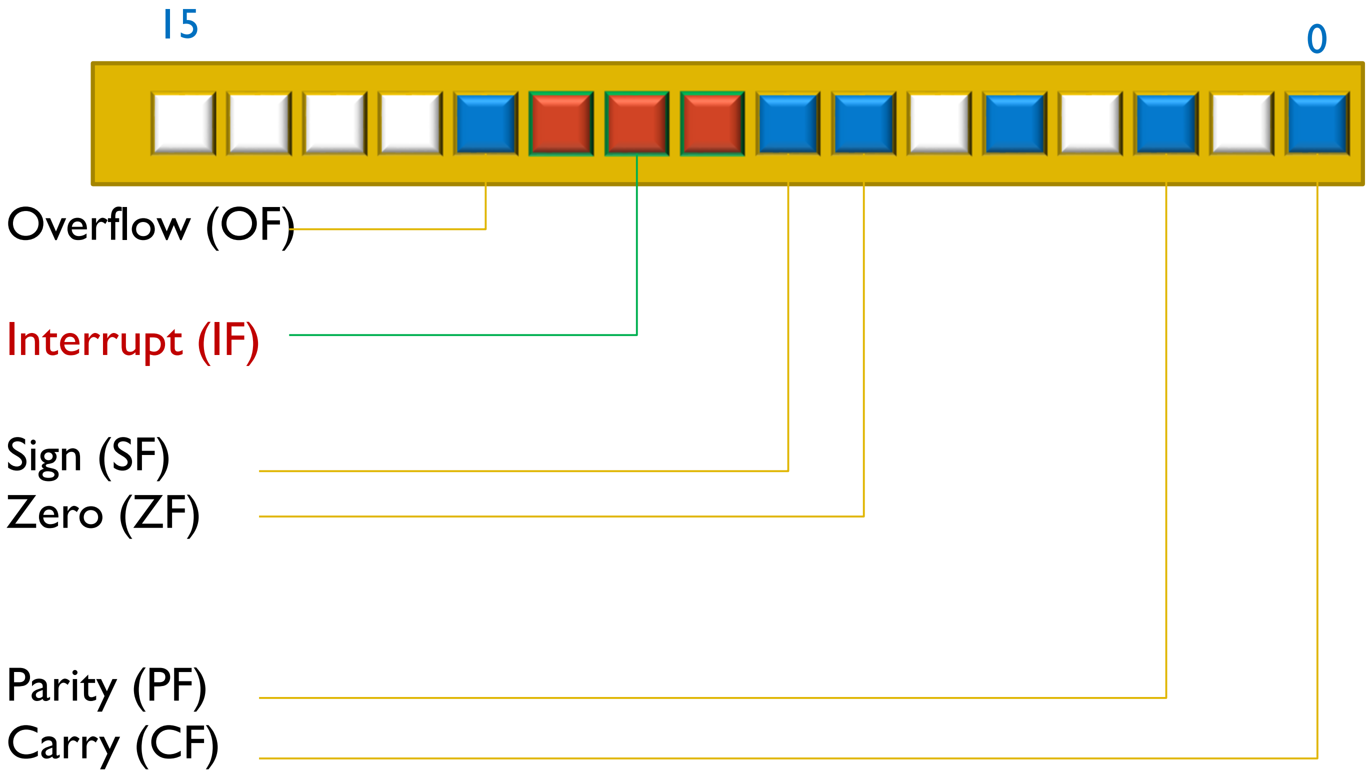


- 16-bit registers for general purpose but some may have specific usage
- AX, BX, CX, DX are made of two separate 8-bit registers (-H/-L), which can be used separately.
- -H represents high byte
- -L represents low byte

Special Purpose Registers

EIP	Instruction pointer, is used to hold address of the next instruction to be executed (We called it Program Counter)
Flag Register	Determine the current status of the CPU

Flag Register



Status Flags

Flags	1	0
Overflow Flag (OF)	There is a signed overflow (e.g. 125+5 is not in range -128—127)	otherwise
Interrupt Flag (IF)	Enable interrupts	Disable interrupts
Sign Flag (SF)	When result is negative	otherwise
Zero Flag (ZF)	When result is zero	otherwise
Parity Flag (PF)	Even number of 1s in result (e.g. the result is 01010101b)	otherwise
Carry Flag (CF)	There is an unsigned overflow (e. g. 255+5 is not in range 0-255)	otherwise

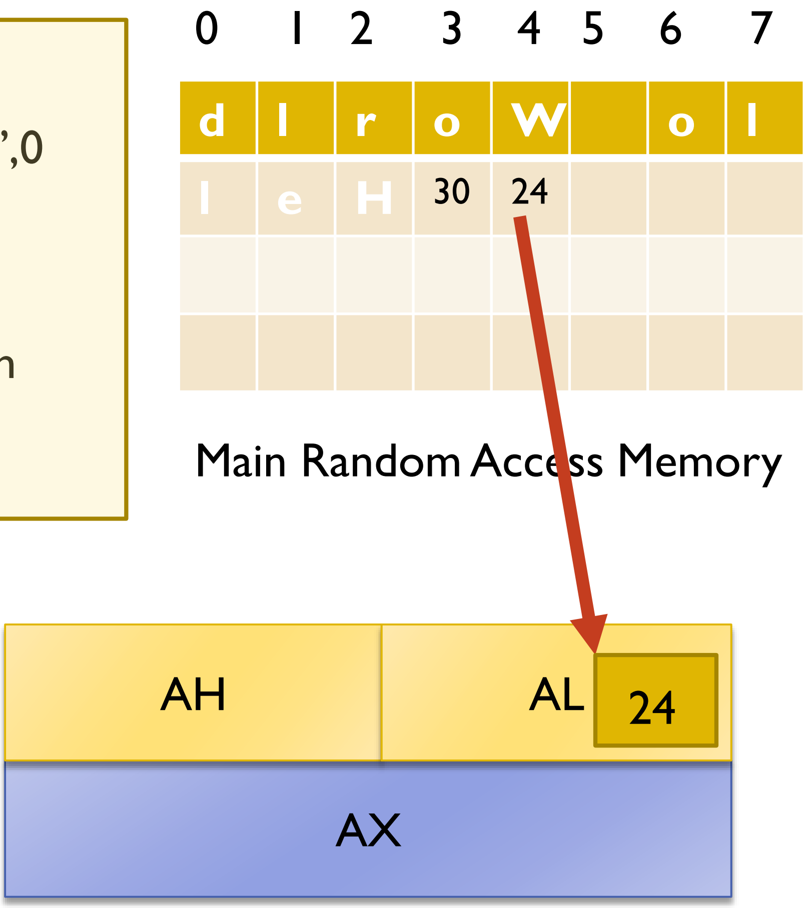
Data

```
.data
        myString db "Hello World",0
        myNumber db 24h
```

Note: Variables and Strings are store in main random access memory.

```
.code
        mov al, myNumber
        …..
        …..
end
```

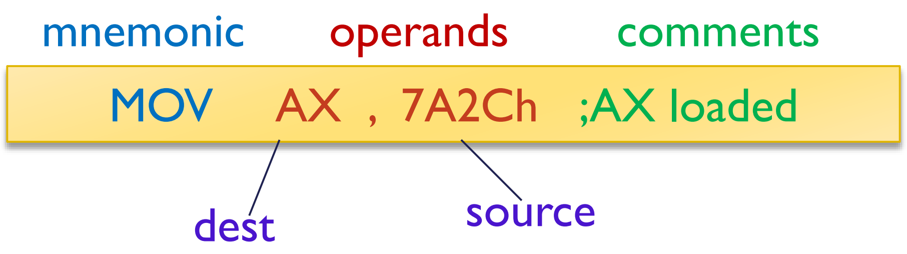
Note oprerands in the code work with the CPU registers.



Main Random Access Memory

The EAX CPU Register

Instruction Format



```
MOV    AX , 7A2Ch   ;AX loaded
```

Instruction Set

- Data moving instructions.
- Arithmetic and Logic
  - add, subtract, increment, decrement, convert byte/word and compare.
  - AND, OR, exclusive OR, shift/rotate and test.
- Control transfer
  - conditional, unconditional, call subroutine and return from subroutine.
- I/O instructions
- Other
  - setting/clearing flag bits, stack operations, software interrupts, etc.

Basic Instructions we covered

Data moving

MOV destination , source

JMP label

JZ label

Loop

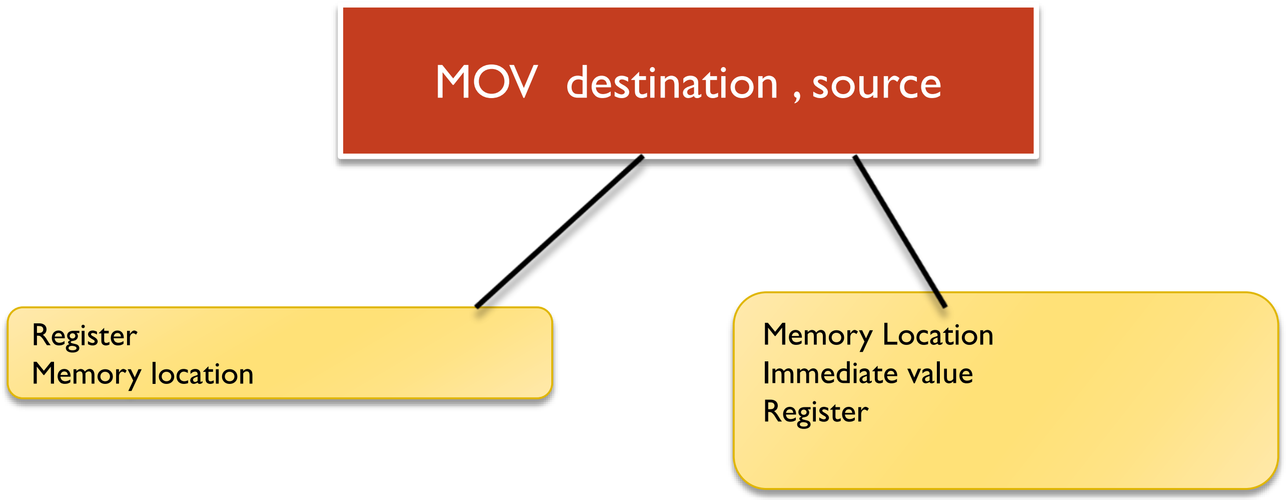
control

ADD dest, source
SUB dest, source
MUL source
DIV source

CMP dest, source

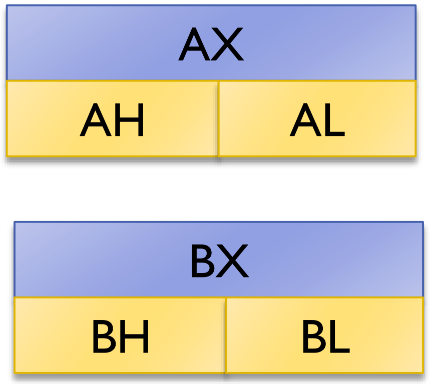
Arithmetic and logic

MOV Instruction



Example

```
MOV   AL, 4Fh     ; Move (copy) value 4F to AL
MOV   AH, 3Ah     ; Move (copy) value 3A to AH
MOV   BX, AX      ; Move (copy) contents of AX into BX
```



Arithmetic Instructions: ADD, SUB, MUL, DIV

```
ADD dest, source
SUB dest, source
DIV  source
MUL source
```

- ADD --- add *source* to *dest*
- SUB --- subtract *source* from *dest*
- Result is always stored in *dest*

- *MUL ---- multiply source*
  - *If the source is 8bit then the high order value is left in AH the low order in AL*
  - *If the source is 16bit then high order value is left in DX and the low order in AX*
- *DIV ---- divide source*
  - *If the source is 8bit the quotient is placed in AH the remainder is in AL*
  - *If the source is 16bit the quotient is placed in AX the remainder is in AX*

Examples

```
ADD AL,74H        ;Add number 74H to content of AL
```

```
ADD DX, BX        ;Add contents of BX to contents of DX
```

```
MOV CL, 01110011b         ;115 decimal
MOV BL, 01001111b         ; 79 decimal
ADD CL, BL                ; Result in CL = 11000010 = 194 decimal

; Do addition once more
ADD CL, BL                ; Result in CL = ?????
```

CMP

CMP dest, source

Reg,	M
M	Immediate
	Reg

- Only affects **flags**. No result is stored.
  - **CMP** – subtract source from dest for flags only

If	dest > source	then	SF=0
If	dest = source	then	ZF=1
If	dest < source	then	SF =1

- Used for conditional flow control

Program Flow Control

- **JMP** instruction
  - Unconditional jumps that transfers control to another point in the program.
  - JMP syntax

```
JMP label
```

  - Declare a label

```
label:  MOV AH, 013h

label2:
        ADD AL, BL
```

    - A label cannot start with a number

Conditional Flow Control

Jump instructions that test Flags

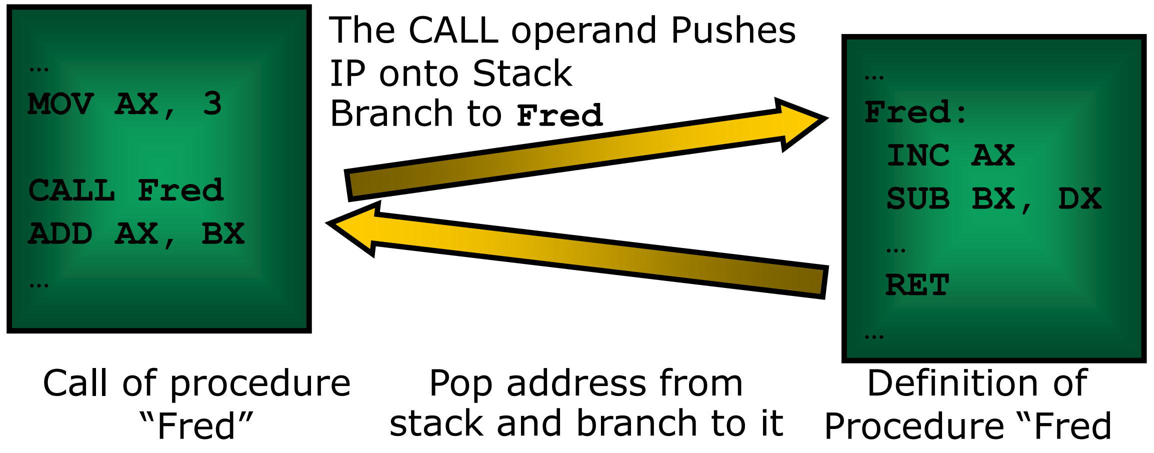
You only need to know

Instruction	Function	Flag
JZ	Jump if Zero	ZF=1
JC	Jump if Carry	CF=1
JS	Jump if Sign	SF=1
JO	Jump if Overflow	OF=1
JNZ	Jump if Not Zero	ZF=0
JNC	Jump if Not Carry	CF=0
JNS	Jump if Not Sign	SF=0
JNO	Jump if Not Overflow	OF=0

Example

```
MOV  AL, 20
MOV  BL, 50

CMP  AL, BL

JZ  equal

MOV  CL, 04Eh
JMP stop

equal:
MOV   CL, 059h

Stop:
RET

END
```

Procedures


The CALL operand Pushes IP onto Stack Branch to Fred
Pop address from stack and branch to it
Call of procedure "Fred"
Definition of Procedure "Fred

```
…
MOV AX, 3

CALL Fred
ADD AX, BX
…
```

```
…
Fred:
 INC AX
 SUB BX, DX
 …
 RET
…
```

- Like a branch instruction
  - But control is returned to the point of call on completion
- You can think of a procedure as a new instruction
  - Only need to know *what* it does, not *how* it does it
- Very useful method for structuring programs
- See simpleProc.asm

The Nine String instructions are:

REP	Repeat
REPE (REPZ)	Repeat while equal (zero)
REPNE (REPNZ)	Repeat while not equal (NZ)
MOVS	Move byte or word string
MOVSB (MOVSW)	Move byte string (word string)
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string

1.Initializing the String pointers

- Before we can use any string we must set up the ESI and EDI registers.

```
MOV        ESI, OFFSET shopper
MOV        EDI, OFFSET shopping
```

Direction Flag

- There are two instructions used to set this flag.
- CLD (Clear direction flag) Clears the direction flag i.e. sets it to 0.
- SLD (Set direction flag) Sets the direction flag i.e. set it to 1.
- CLD selects auto-increment.
- SLD selects auto-decrement.