

# x86

Assembly and its concepts

Back to basics

1 0 1 1

Nibble

B

1 0 1 1 1 1 0 1

Byte

B

D

1 0 1 1 1 1 0 1 0 0 1 1 1 0 0 1

Word

B

D

3

9

# x86 Architecture

- 8086 introduced in 1978
- Then followed by 8088, 80186, 80286, 386, 486, Pentium, 686 ...
- CISC architecture
- 💬• 32-bit known as x86 or IA-32
- 💬• 64-bit known as x86-64, AMD64 and Intel 64

Intel 80386 or i386

- Introduced in 1986
- Has 32-bit word length
- Has 8 general purpose registers
- Supports paging and virtual memory
- Addresses up to 4GB of memory

# Byte Order

Endianness (Little and Big Endian)



- Little Endian



- Offset value: 78 56 34 12
- Actual value: 0x12345678

- Big Endian

- Offset value: 12 34 56 78
- Actual value: 0x12345678

# String Styles

- ASCIIZ – C style

H	e	l	l	o	
48	65	6C	6C	6F	00

- Null-terminated Unicode

H		e		l		o		
48	00	65	00	6C	00	6F	00	00

- Pascal

	H	e	l	l	o
05	48	65	6C	6C	6F

- Delphi

				H	e	l	l	o
05	00	00	00	48	65	6C	6C	6F

# Data Registers Layout

## General-Purpose Registers

31	16 15	8 7	0
	AH	AL	
	BH	BL	
	CH	CL	
	DH	DL	
	BP		
	SI		
	DI		
	SP		



16-bit

AX

BX

CX

DX



32-bit

EAX

EBX

ECX

EDX

EBP

ESI

EDI

ESP



EAX							
1	2	3	4	5	6	7	8
				AH		AL	
				AX			

Given, 12345678h

EAX = 12345678h

AX = 5678h

AH = 56h

AL = 78h

# Data Registers

Register	Description	Usage
AL / AH / AX/ EAX	Accumulator Register	Arithmetic operations
BL / BH / BX / EBX	Base register	General data storage, index
CL / CH / CX / ECX	Counter register	Loop constructs
DL / DH / DX / EDX	Data register	Arithmetic




# Example

## Accumulator

- `ADD EAX, EBX` ; add the value in EBX and EAX, and store the  
; result into EAX
- `SUB EAX, EBX` ; substitute the value in EBX from EAX, and store the  
; result into EAX
- `IMUL EAX, EBX` ; multiply EAX and EBX, and store the result into EAX



# Address Registers

Register	Description	Usage
IP / EIP 	Instruction Pointer	Program execution counter
SP / ESP 	Stack Pointer	ESP will hold an offset to top of stacks memory location
BP / EBP	Base Pointer	Stack frame
SI / ESI 	Source Index	String operation
DI / EDI	Destination Index	String operation

# Segment Registers

Register	Description	Usage
CS	Code Segment	Program code
DS	Data Segment	Program data
SS	Stack Segment	Stack
ES / FS /GS	Other Segments	Other uses

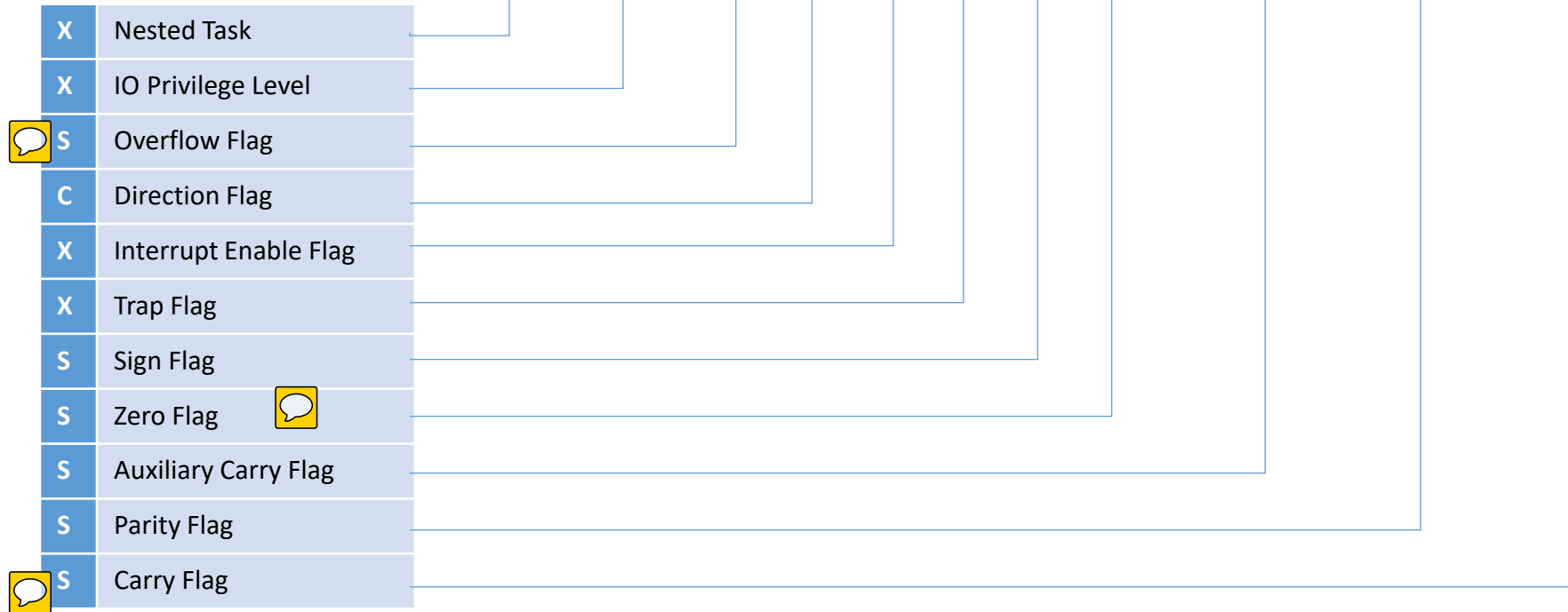
# Continue

- Modern Operating System such as Windows, MacOSX, \*nix are using protected mode flat model and segment registers are less relevant. CS, DS, SS, and ES are pointing to the same location, which is 0 offset.
- Windows NT is utilizing FS register to store Thread Information Block (TIB)
  - [http://en.wikipedia.org/wiki/Win32\\_Thread\\_Information\\_Block](http://en.wikipedia.org/wiki/Win32_Thread_Information_Block)

# EFLAGS Register (bit 0-15)

S	Status Flag
C	Control Flag
X	System Flag

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	NT	IOPL		OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF



# EFLAGS Register (bit 16-31)

S	Status Flag
C	Control Flag
X	System Flag

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0	0	0	0	0	0	0	0	0	0	ID	VIP	VIF	AC	VM	RF


X	ID Flag
X	Virtual Interrupt Pending
X	Virtual Interrupt Flag
X	Alignment Check
X	Virtual 8086 Mode
X	Resume Flag



# EFLAGS Register – Status Flags

- ZF - Zero flag - Set if the result is zero; cleared otherwise.
- SF - Sign flag - Set equal to the most-significant bit of the result, which is the sign bit of a signed integer. (0 indicates a positive value and 1 indicates a negative value.)
- CF - Carry flag - Set if an arithmetic operation generates a carry or a borrow out of the most significant bit of the result; cleared otherwise. This flag indicates an overflow condition for unsigned-integer arithmetic. It is also used in multiple-precision arithmetic.
- OF - Overflow flag - Set if the integer result is too large a positive number or too small a negative number (excluding the sign-bit) to fit in the destination operand; cleared otherwise. This flag indicates an overflow condition for signed-integer (two's complement) arithmetic.
- AF - Adjust flag - Set if an arithmetic operation generates a carry or a borrow out of bit 3 of the result; cleared otherwise. This flag is used in binary-coded decimal (BCD) arithmetic. (Rarely Used)
- PF - Parity flag - Set if the least-significant byte of the result contains an even number of 1 bits; cleared otherwise. (Rarely Used)

# EFLAGS Register – DF Flag

- The direction flag (DF, located in bit 10 of the EFLAGS register) controls string instructions (MOVS,  CMPS, SCAS, LODS, and STOS). Setting the DF flag causes the string instructions to auto-decrement (to process strings from high addresses to low addresses). Clearing the DF flag causes the string instructions to auto-increment (process strings from low addresses to high addresses).
- The STD and CLD instructions set and clear the DF flag, respectively.

# EFLAGS – Flag Changing Instructions



- Instruction that affecting flags
  - ADD/SUB – Modifies Flags: AF CF OF PF SF ZF
  - CMP- Modifies Flags: AF CF OF PF SF ZF
  - TEST - Modifies Flags: CF OF PF SF ZF (AF undefined)
  - CLD/STD – Modifies Flags: DF





# Signed Numbers

# Two's Complement

Binary Value	Hex Value	Two's Complement	Unsigned
00000000	0x00	0	0
00000001	0x01	1	1
...	...	...	..
01111110	0x7E	126	126
01111111	0x7F	127	127
10000000	0x80	-128	128
10000001	0x81	-127	129
10000010	0x82	-126	130
...	...	...	...
11111110	0xFE	-2	254
11111111	0xFF	-1	255

# How to convert?

- First method (calculation method)

Steps	Example 1	Example 2
1. Starting from the right, find the first '1'	010100 <b>1</b>	0101 <b>1</b> 00
2. Invert all of the bits to the left of that one	<b>1010111</b>	<b>1010</b> 100

- Second method (easy method)
  - Using x86 instruction for Two's Complement Negation, NEG
  - Example usage – NEG EAX

# Examples



## Example 1:

mov al, 0x5

add al, 0xFE ; 0xFE = -2 or 0xFE = 254

; al = 3



## Example 2:

mov al, 0x02

add al, 0xFA ; 0xFA = -6 or 0xFA = 250

; al = 0xFC