ASSEMBLY

ARM VS x86

- both low-level languages
- x86 is one of the most common assembly languages, with its architecture having both 32-bit and 64-bit versions
- Most Intel computers use x86 assembly

CISC	RICS
Emphasis on Hardware	Emphasis on Software
Includes multi-clock complex instructions	Single-clock, reduced instruction only
Memory-to-memory: "LOAD" and "Store" incorporated in instructions	Register-to-register: "Load" and "Store" are independent instruction
Small code sizes, high cycles per second	Low cycles per second, large code sizes
Transistors used for storing complex instructions	Spends more transistors on memory registers

- Big endian systems store the most significant byte (MSB) at the smallest memory address and the least significant byte (LSB) at the largest.
- Little endian systems store the LSB at the smallest memory and the MSB at the largest.
- Most processors are little endian.
- Most network protocols are big endian.

x86 Registers

-x86 Architecture has 8 General-Purpose Registers(GPR), 6 Segment Registers, 1 Flags Register and an Instruction Pointer for 32-bit x86

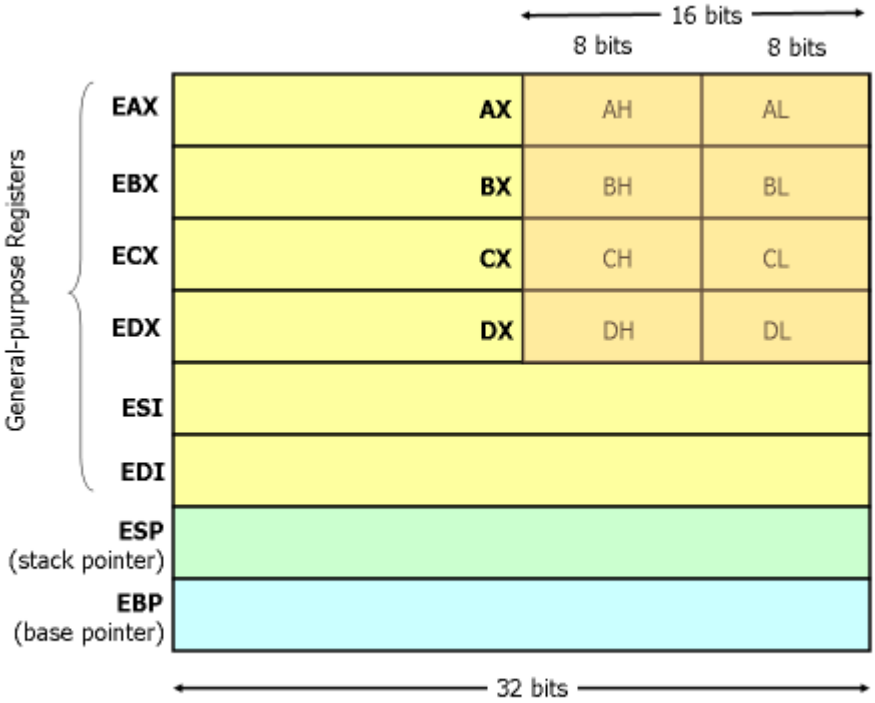

Figure 1. x86 Registers

> EAX -- Stores function return values
> EBX -- Base pointer to the data section
> ECX -- Counter for string and loop operations
> EDX -- I/O pointer
> ESI -- Source pointer for string operations
> EDI -- Destination pointer for string operations
> ESP -- Stack Pointer
> EBP -- Stack frame base pointer
> EIP -- Pointer to next instruction to execute, cannot be directly modified with mov but can inderectly be modified by referencing with operations

- GPR are used for basic arithmetic and typical operations. Pointer registers are used for pointing at memory for program control.

Flags and Segment Registers

> CS -- Pointer to Code segment in which your program runs
> DS -- Pointer to Data segment that your program accesses
> ES,FS,GS -- Extra segment registers available for far pointer addressing like video memory
> SS -- Pointer to Stack segment your program uses.
> OF -- Overflow flag, used if destination could not store the entire result

> SF -- Sign flag, used if last operation yielded a value with MSB set
> ZF -- Set if the result of an arithmetic operation is 0

Instructions

- mov/Move(Opcodes: 88, 89, 8A, 8B) -- The mov instruction copies the data item referred to by its second operand into the location referred to by its first operand. While register-to-register moves are possible, direct memory-to-memory moves are not.

  > mov eax, ebx — copy the value in ebx into eax
  > mov byte ptr [var], 5 — store the value 5 into the byte at location var

- push/Push stack(Opcodes: FF, 89, 8A, 8B, 8C) -- The push instruction places its operand onto the top of the hardware supported stack in memory. Specifically , push first decrements ESP by 4.

  > push eax — push eax on the stack
  > push [var] — push the 4 bytes at address *var* onto the stack

- lea/Load effective adress -- The lea instruction calculates indirect adress and stores the address in the destination.

  > lea edi, [ebx+4*esi] — the quantity EBX+4*ESI is placed in EDI.
  > lea eax, [var] — the value in *var* is placed in EAX

-jmp/Jump -- Transfers program control flow to the instructions at the memory location indicated by the operand.

> jmp `destination`

-add/Arithmetic Addition -- Adds the value specified to the value stored in the destination and replaces the destination with the result.

> add eax, 25

-sub/Arithmetic Subtraction -- Subtracts the value of its second operand from the value of its first operand. As with add.

> sub eax, 280

-inc, dec/Increment, Decrement -- Increments or decrements the contents of its operand by one.

> dec eax

-pop/Pop stack -- The pop instruction removes the 4-byte data element from the top of the hardware-supported stack into the specified operand. Useful for obtaining a pointer into a memory region.

> lea eax, [var] — the value in *var* is placed in EAX.
> lea eax, [val] — the value *val* is placed in EAX

- pusha -- Pushes the 16-bit register values AX, CX, DX, BX, SP, BP, SI, DI to the stack
- pushad -- pushes the 32-bit register values EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI to the stack
- popa -- inverse operation of pusha
- popad -- inverse operation of pushad

The Stack

- Each active function call has a frame that stores the values of all local variables, and the frames of all active functions are maintained on the Stack. It is used to store temporary data needed during the execution of a program, lik local variables and parameters, function return addresses.
- It is static memory, so It cannot be altered during runtime.
- Dynamic memory like the one allocated with malloc()/new() is stored on the heap.