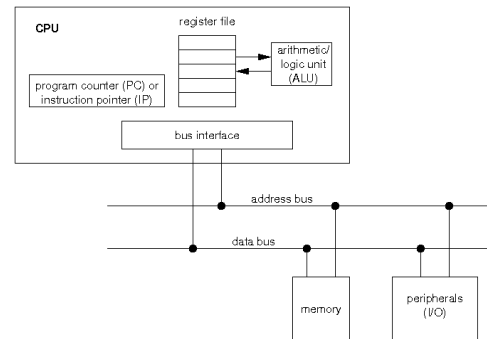


## Hardware and Instruction Set Architecture

CSE 361S

### Simple Computer System



### Fetch-Decode-Execute Cycle

- Fetch: grab (fetch) the instruction to be executed. It's address is in the instruction pointer (IP) or program counter (PC)
- Decode: figure out what instruction it is and what is to be done (e.g., this is an ADD inst. that needs two values from the register file)
- Execute: do the real work and store the result somewhere (as told by the instruction)

### Instruction Set Architecture (ISA)

- Programmer's view of the processor. It includes the following components:
  - Instruction set: the collection of instructions that are supported by the processor.
  - Register file: the programmer-visible storage within the processor.
  - Memory: the logical organization of the memory (again, programmer's view)
  - Operating modes: some processors have subsets of the instructions that are privileged based on being in a given "mode."

### x86 Instruction Set

- Arithmetic operations: (add, sub, mult, div, etc.)
- Boolean operations: (and, or, not, xor, etc.)
- Shift operations: (left shift, right shift)
- Comparison operations: (<, ≤, >, ≥, =, ≠)
- Memory operations: (load, store)

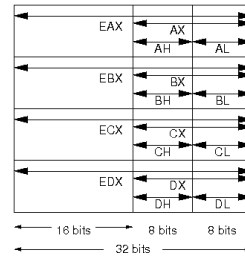
### x86 Instruction Set

- Control flow operations:
  - Unconditional branch: (jump)
  - Conditional branch: (beq, bne, etc.)
  - Procedure call/return: (call, ret)
- Peripheral access: (in, out)
- Conversion operations: (f2i, i2f)
- System operations: (nop, trap, halt, etc.)

### x86 Register File

- 4 classes of register in the x86 ISA:
  - Data/general registers
  - Address/index registers
  - Segment registers (obsolete, ignore)
  - Status registers
- Data/general registers named A (accumulator), B (base), C (counter), and D (data)
  - These names are old and irrelevant

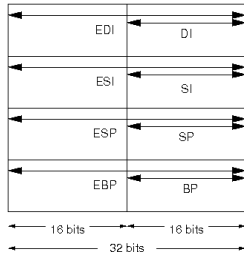
### Data/General Registers



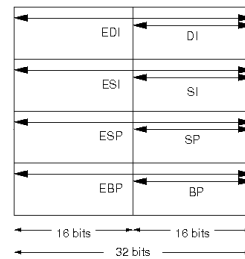
- Note: EAX, AX, and AH/AL occupy the same 32 bits of storage, so the programmer can only use one at a time
- We will use 32-bit versions: EAX, EBX, ECX, EDX

### Address Registers

- Names: DI (destination index), SI (source index), SP (stack pointer), and BP (base pointer). The names SP and BP are relevant!



### Address Registers (cont.)



- Note: again, EDI and DI occupy the same storage, so they cannot be used together
- We will stick with 32-bit versions: EDI, ESI, ESP, EBP

### Usage of Address Registers

- ESP and EBP are reserved for maintaining and accessing the system stack (stack frame)
- EBX can also be used for addressing (recall that EBX is a general register)
- EBX and EBP are classically base registers
- EDI and ESI are classically index registers (addressing relative to a base register)
- ESP and EBP are the ones that really matter!!

### Status Registers

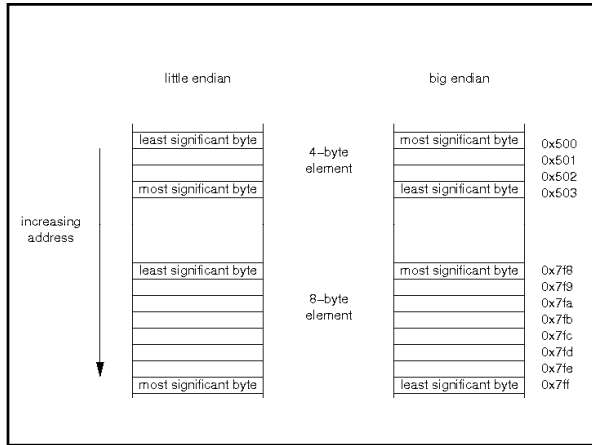
- IP is instruction pointer ⇒ holds address of next instruction to be executed
- FLAGS is flags register ⇒ status bits retaining results of previous operations:
  - ZERO result of previous arithmetic or logical op is 0
  - SIGN result of arithmetic or logic op is negative
  - CARRY result of unsigned add is too large
  - OVERFLOW result of signed op is out of range
  - INTERRUPT currently servicing an interrupt

### Memory Organization

- On an x86 machine, memory is byte addressable (i.e., each byte in the memory has a unique address)
- For multi-byte data elements, the address of the element is the lowest address the element occupies (e.g., for a 32-bit integer occupying 0x500, 0x501, 0x502, and 0x503, the address of the integer is 0x500)

### Little Endian vs. Big Endian

- The “endian”-ness of a processor defines the ordering it uses for multi-byte primitive data elements (e.g., 32-bit integers).
- Little endian  $\Rightarrow$  the least significant byte (LSB) goes in the lowest address, “littlest end first”
- Big endian  $\Rightarrow$  the most significant byte (MSB) goes in the lowest address, “biggest end first”
- x86 is a little endian machine



### Primitive Elements vs. Collections

- Note: endianness only applies to primitive elements (e.g., integers, floats, etc.), not collections of elements
- A “string” is an array of characters, and therefore is not impacted by the endianness of the machine. The first character is in the lowest address.

### Little Endian 32-bit Integer

```
int val = 0x12345678; /* &val is 0x5000 */
```

addr	byte
0x5000	0x78
0x5001	0x56
0x5002	0x34
0x5003	0x12