RUTGERS

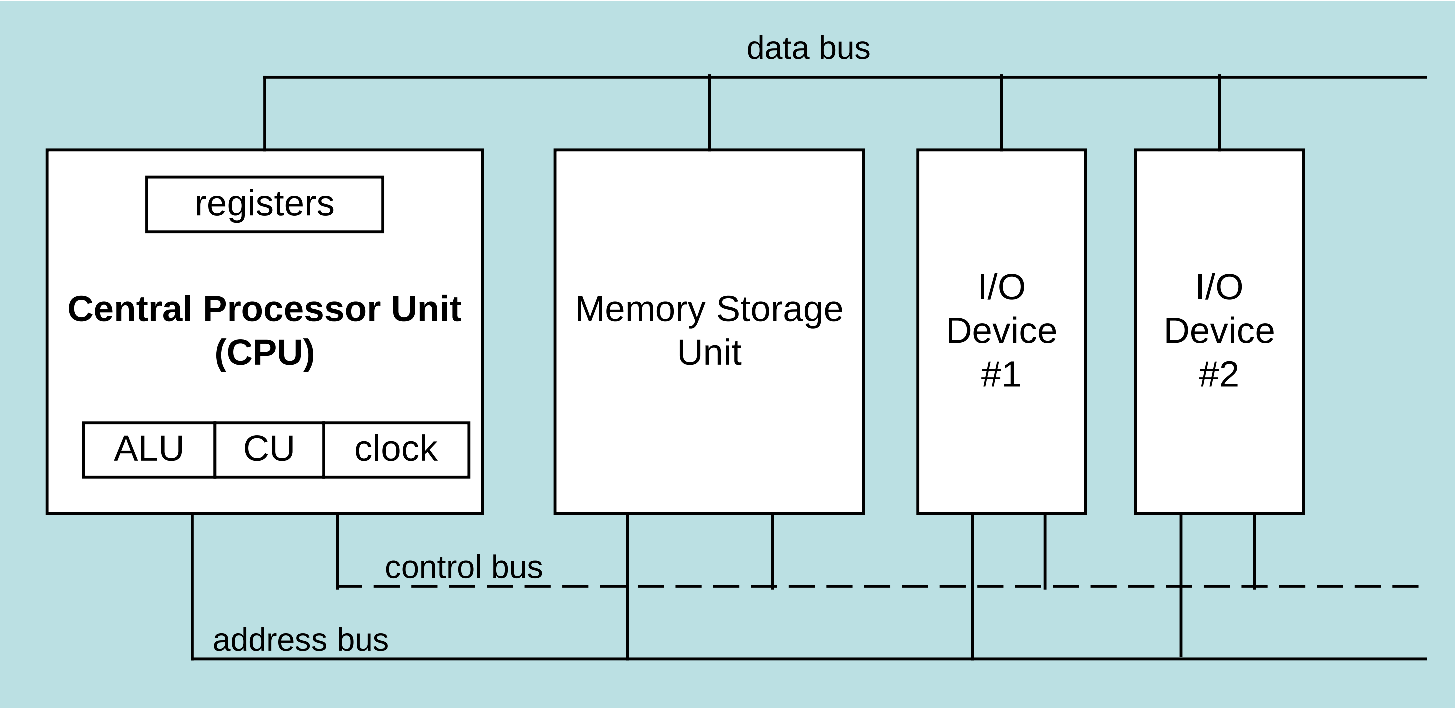
THE STATE UNIVERSITY OF NEW JERSEY

Recitation 6

Jae Woo Joo

Basic Hardware Organization

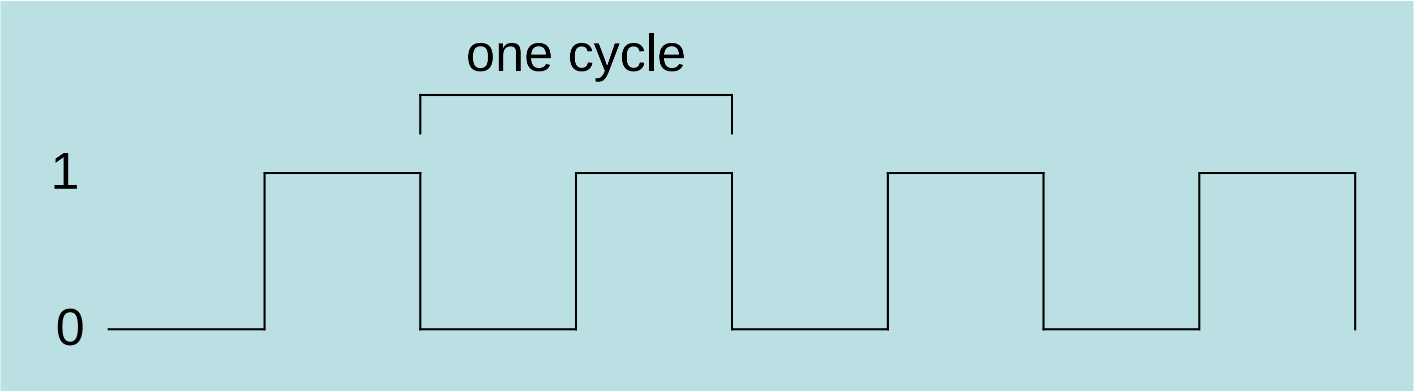
- Clock synchronizes CPU operations
- Control Unit coordinates sequence of execution steps
- ALU performs arithmetic and bitwise processing



jaewoo.joo@cs.rutgers.edu

Clock

- Clock synchronizes all CPU and BUS operations
- Clock cycle measures time of a single operation
- Clock is used to trigger events



Instruction Execution Cycle

- Basic operation cycle of a computer
  - **Fetch**: The next instruction is fetched from the memory that is currently stored in the program counter
  - **Decode**: The encoded instruction present in the IR is interpreted
  - **Execute**: The control unit passes the instruction to the ALU to perform mathematical or logic functions and writes the result to the register.

Instruction Execution Cycle

Loop

fetch next instruction

advance the program counter (PC)

decode the instruction

if memory operand needed read from memory

execute the instruction

if result is memory operand, write to memory

Continue loop

CISC and RISC

- CISC – Complex instruction set computer
  - Large instruction set
  - High-level operations
  - Requires microcode interpreter

- RISC – Reduced instruction set computer
  - Simple, atomic instructions
  - Small instruction set
  - Directly executed by hardware

What is Assembly Language

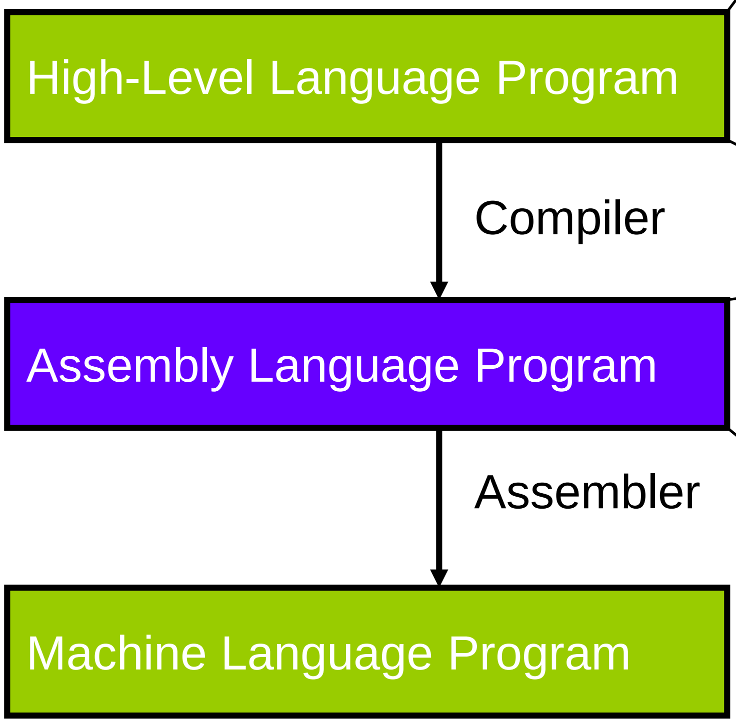
- It is used to write programs in terms of the basic operations of a processor
- A processor understands only machine language instructions
- Machine language is too obscure and complex
- So low-level assembly language is designed for the processors

Advantages of Assembly Language

- Requires less memory and execution time
- Allows hardware-specific complex jobs in an easier way
- Suitable for time-critical jobs

jaewoo.joo@cs.rutgers.edu

Brief structure




```
#include <stdio.h>
int main() {
    int x, y, temp;
    x=1; y=2;
    temp =x; x=y;  y=temp;
    printf("%d %d %d\n",x,y,temp);
}
```

```
movl    $1, -8(%ebp)
movl    $2, -12(%ebp)
movl    -8(%ebp), %eax
movl    %eax, -16(%ebp)
movl    -12(%ebp), %eax
movl    %eax, -8(%ebp)
movl    -16(%ebp), %eax
movl    %eax, -12(%ebp)
movl    -16(%ebp), %eax
movl    %eax, 12(%esp)
movl    -12(%ebp), %eax
movl    %eax, 8(%esp)
movl    -8(%ebp), %eax
movl    %eax, 4(%esp)
```

ISA

```
7f 45 4c 46 01 01 01
 00 00 00 00 00 00 00
 00 00 02 00 03 00 01
00 00 00 f0 82 04 08
34 00 00 00 c4 0c 00
00 00 00 00 00 34 00
```

Assembly Instructions

- Assembled into machine code by assembler
- Executed at runtime by the CPU
- Parts
  - Label (optional)
  - Opcode (also called as mnemonic)
  - Operand
  - Format
    - [label:]

      opcode operands
  - Example)

    movl %eax, %ebx

Labels

- Act as place markers
  - Marks the address of code and data

- Code label
  - Target of jump or loop instructions
  - Ex) L1:   (followed by colon)

jaewoo.joo@cs.rutgers.edu

Opcode

- Instruction opcode
  - MOV
  - ADD
  - SUB
  - MUL
  - JMP
  - …

Operands

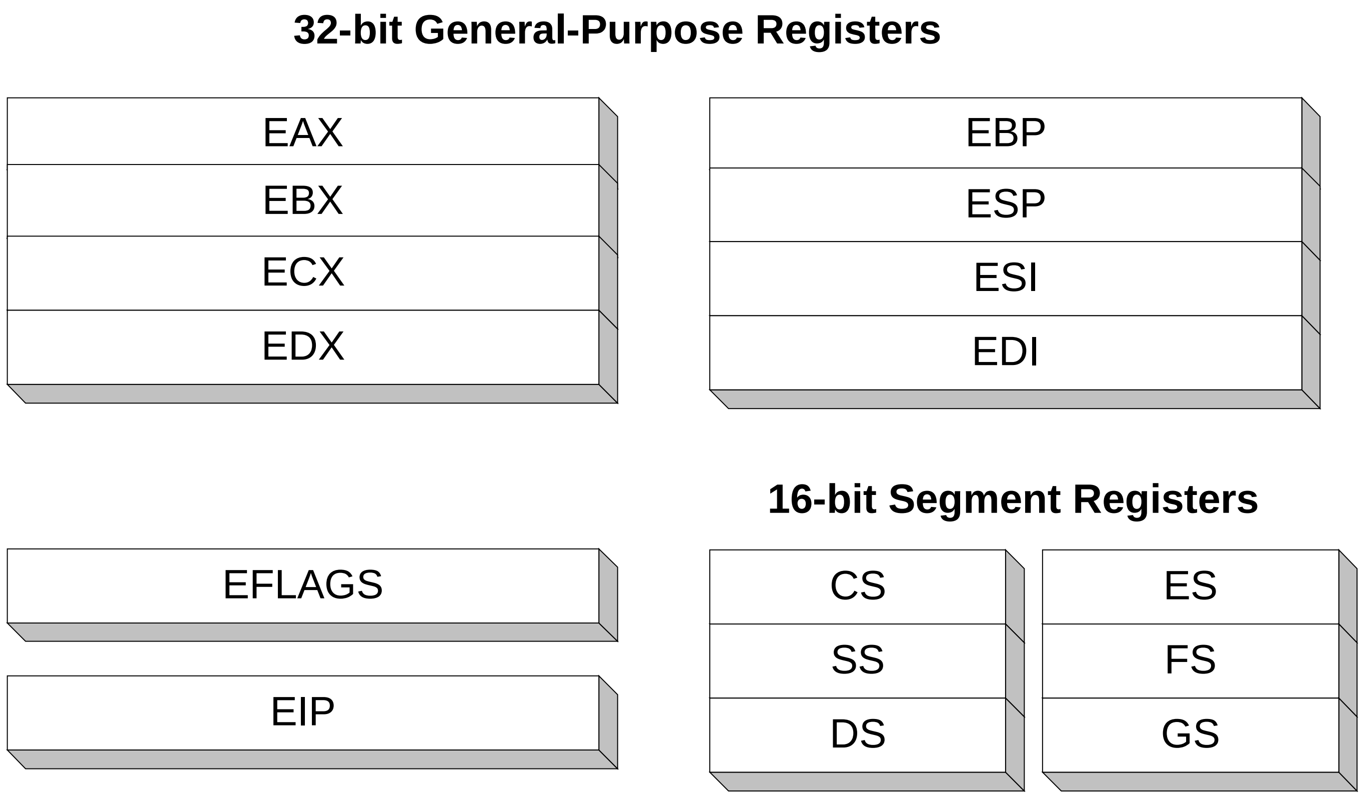
- Constant (immediate value)
  - Ex) 96
- Constant expression
  - Ex) 2 + 4
- Register
  - Ex) %eax

Registers

- Registers are CPU components that hold data and address
- Much faster to access than memory
- It is used to speed up CPU operations
- Categories
  - General registers
    - Data registers
    - Pointer registers
    - Index registers
  - Control registers
  - Segment registers

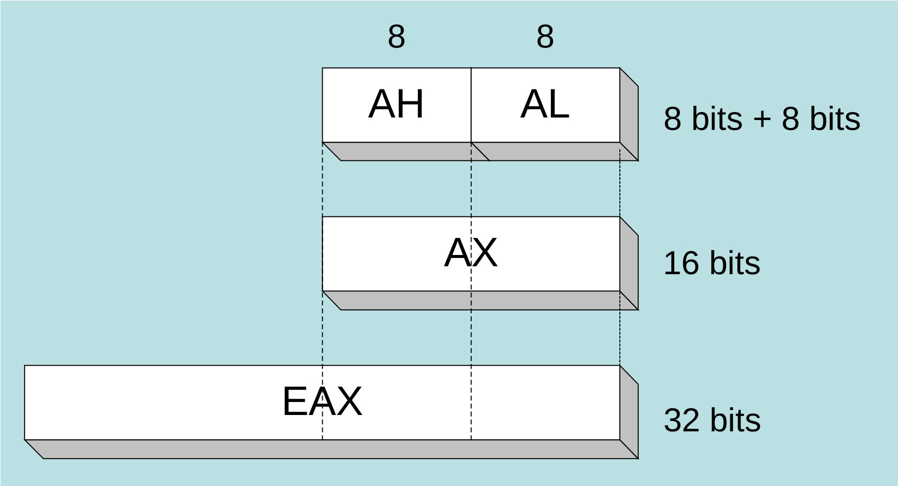
General-Purpose Registers

- Named storage locations inside the CPU, optimized for speed



General-Purpose Registers (Data)

- Can use 8-bit, 16-bit, or 32-bit name



32-bit	16-bit	8-bit (high)	8-bit (low)
EAX	AX	AH	AL
EBX	BX	BH	BL
ECX	CX	CH	CL
EDX	DX	DH	DL

General-Purpose Registers (Data)

- AX is the primary accumulator
  - Used in most arithmetic instruction, return value
- BX is the base register
  - Could be used in indexed addressing
- CX is the count register
  - Store the loop count in iterative operations
- DX is the data register
  - Used in input / output operations

General-Purpose Registers

- Some registers have only a 16-bit name for their lower half

32-bit	16-bit
ESI	SI
EDI	DI
EBP	BP
ESP	SP

General-Purpose Registers (Pointer)

- ESP is stack pointer
  - It refers to be current position of data or address within the program stack
  - Changed by push, pop instructions
- EBP is frame pointer
  - Referencing the parameter variables passed to a subroutine
- EIP is instruction pointer
  - It stores the offset address of the next instruction to be executed

General-Purpose Registers (Index)

- ESI and EDI are used for segmented addressing

- ESI is used as source index for string operations
- EDI is used as destination for string operations

Control Registers

- Many instructions involve comparisons and mathematical calculations and change the status of the flags

Control Registers

- Overflow flag (OF)
  - Indicates the overflow of a high-order bit
- Carry flag (CF)
  - Contains the carry of 0 or 1 from high-order bit after arithmetic operation
  - Stores the last bit of a shift or rotate operation
- Sign flag (SF)
  - Shows the sign of the result of an arithmetic operation
  - Positive -> 0, Negative -> 1
- Zero Flag (ZF)
  - Indicates the result of an arithmetic or comparison operation
  - Nonzero clears the ZF to 0
  - Zero results sets to 1

Data Formats

- Byte (1 byte = 8 bits)
  - E.g. Char b
- Word (2 bytes = 16 bits)
  - E.g. Short int w
- Double word (4 bytes = 32 bits)
  - E.g. Int, float l
- Quad word (8 bytes = 64 bits)
  - E.g. double q

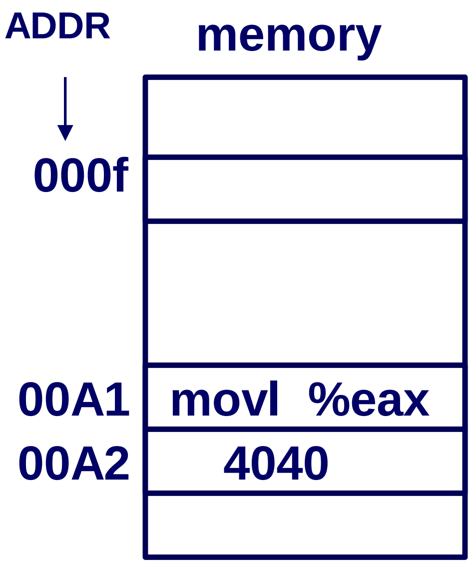
MOV instructions

- Instructions can operate on any data size

- MOVB: move byte from src to destination
- MOVW: move 2-byte word
- MOVL: move 4-byte double word
- MOVQ: move 8-byte quad word

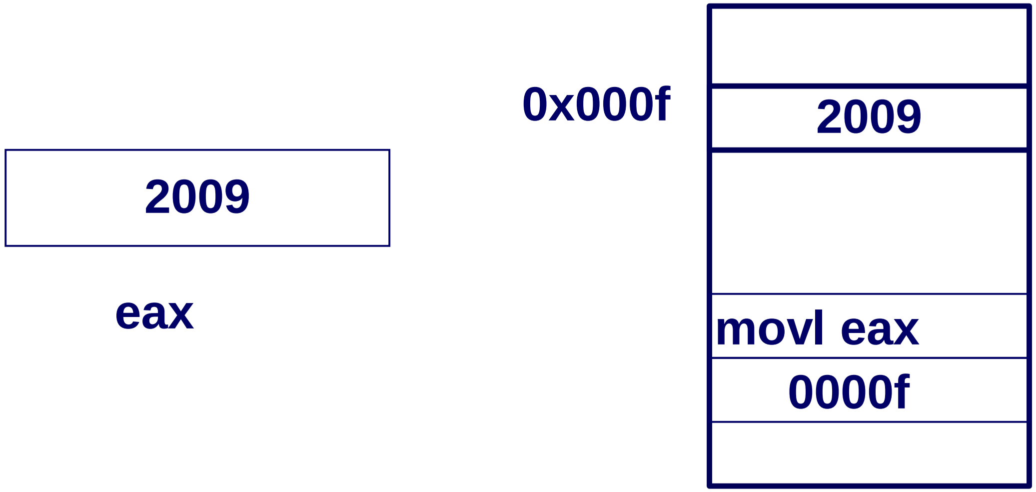
Immediate Addressing

- Operand is immediate
  - Operand value is found immediately following the instruction
  - $ in front of immediate operand
  - E.g. movl $0x4040, %eax



Direct Addressing

- Address of operand is found immediately after the instruction
  - Also known as direct addressing or absolute address
  - E.g. movl %eax, 0x0000f



Register Mode Addressing

- Use % to denote register
- Source operand: use value in specified register
- Destination operand: use register as destination for value
- Examples
  - movl %eax, %ebx
    - Copy content of %eax to %ebx
  - movl $0x4040, %eax -> immediate addressing
    - Copy 0x4040 to %eax
  - movl %eax, 0x000f -> direct addressing
    - Copy content of %eax to memory location 0x0000f

Indirect Mode Addressing

- Content of operand is an address
  - Designated as parenthesis around operand
- Offset can be specified as immediate mode
- Examples
  - movl (%ebp), %eax
    - Copy value from memory location whose address is in ebp into eax
  - movl -4(%ebp), %eax
    - Copy value from memory location whose address is -4 away from content of ebp into eax

Indexed Mode Addressing

- Add content of two registers to get address of operand
  - movl (%eab, %esi), %eax
    - Copy value at (address = eab + esi) into eax
  - movl 8(%eab, %esi), %eax
    - Copy value at (address = 8 + eab + esi) into eax

Address Computation Examples

Address	Value
0x100	$0xFF
0x104	$0xAB
0x108	$0x13
0x10C	$0x11

Register	Value
%eax	$0x100
%ebx	$0x104
%ecx	$0x001
%edx	$0x003

- movl (0x100), %eax — %eax?
- movl (%eax, %edx, 4), %ecx — %ecx?
- decl %ecx — %ecx?

Q & A

- Any questions?