RUTGERS

THE STATE UNIVERSITY OF NEW JERSEY

Computer Architecture (CS-211) Recitation-6

Song Wen

Topics

- Assembly Language

* Some materials are collected and compiled from previous year's CS 211 lectures and TAs

Programming Meets Hardware

High-Level Language Program

```
#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);
}
```

Compiler

Assembly Language Program

```
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

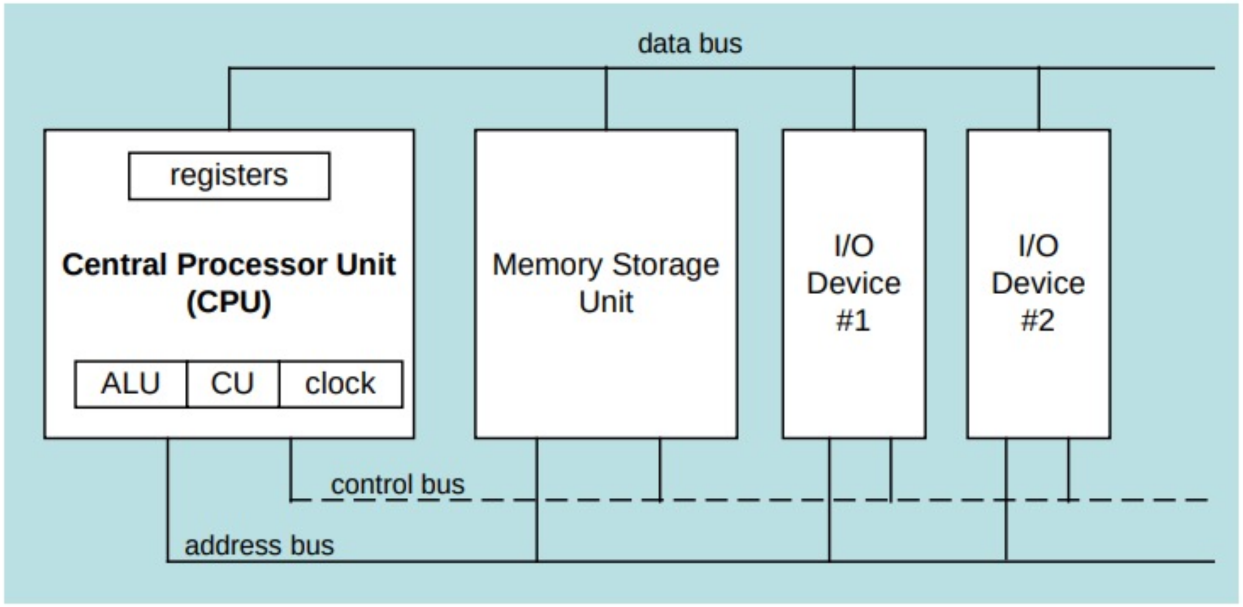
Assembler

Machine Language Program

```
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
```

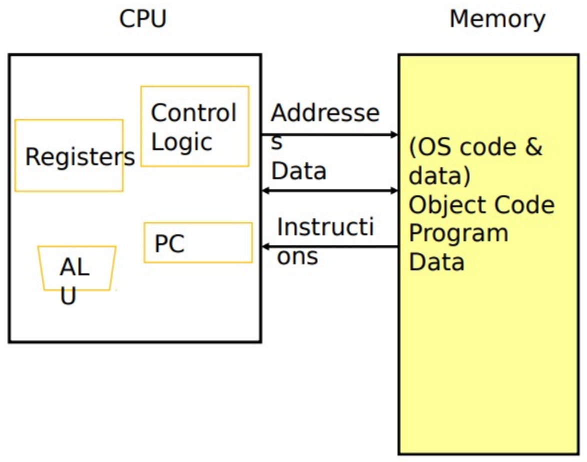
Basic Hardware Organization

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



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.



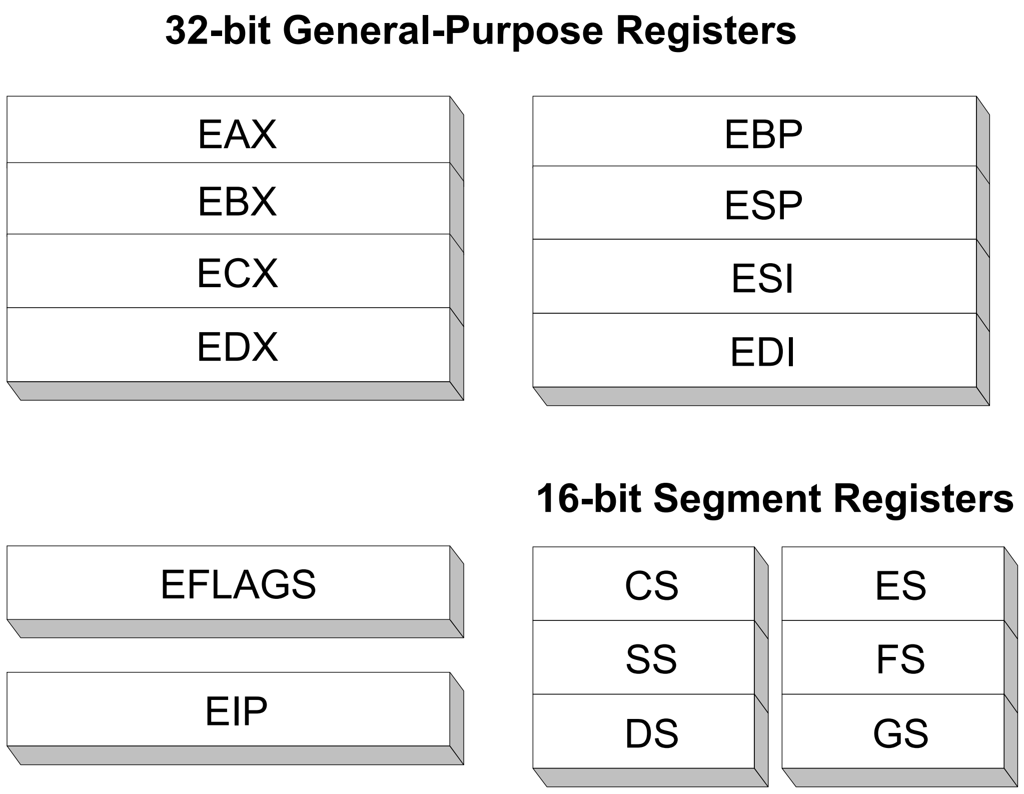
```
Loop
        fetch next instruction
        advance the program counter (PC)
        decode the instruction
        if memory operand needed read from memory
        execute the instruction
        if result is memory operand, write to memory
Continue loop
```

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 (Holds operands)
    - Pointer & index registers (Holds references to addresses as well as indices)
  - Control Register (e.g. CF,ZF)
  - Segment registers (Holds starting address of program segments)
    - CS, DS, SS, ES

Registers Overview

Named storage locations inside the CPU, optimized for speed



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, usually has no specific uses
- CX is the count register
  - Store the loop count in iterative operations
- DX is the data register
  - Used in input operations (function parameters)

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 a pointer to the address of the instruction that is currently executing

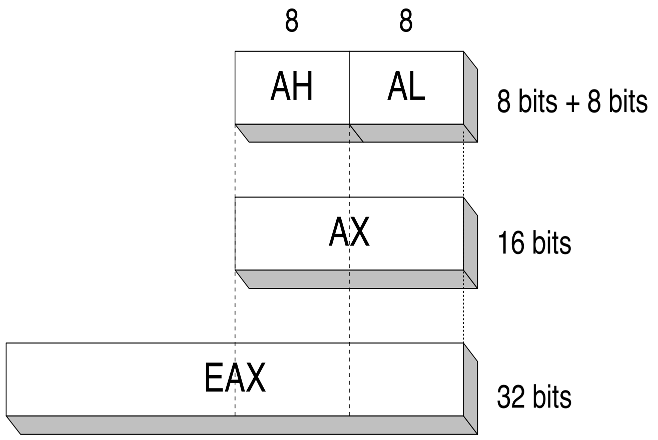
General-Purpose Registers (Index)

- ESI and EDI are used for segmented addressing (used as a pointer)

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

Data Registers

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

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)

Pointer Registers

- 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

Condition Codes

- Single Bit Registers (set after each instruction)
  - CF: Carry Flag, operation generated a carry out
  - SF: Sign Flag, operation yielded a negative value
  - ZF: Zero Flag, operation yielded zero
  - OF: Overflow Flag, operation caused 2’s complement overflow

Assembly Language

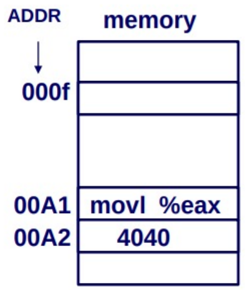
- Instruction format
  - opcode operands

- 3 types of operands:
  - Immediate
    - Constant values
  - Register
    - Contents of registers
  - Memory
    - Contents of memory

- Number of operands depends on the command

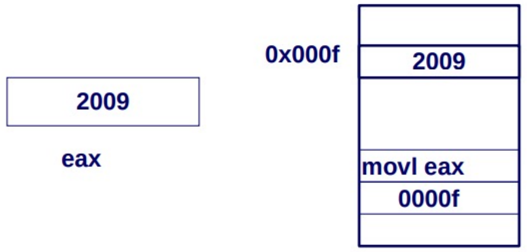
Immediate Addressing

- 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

Addressing mode

- Memory addressing

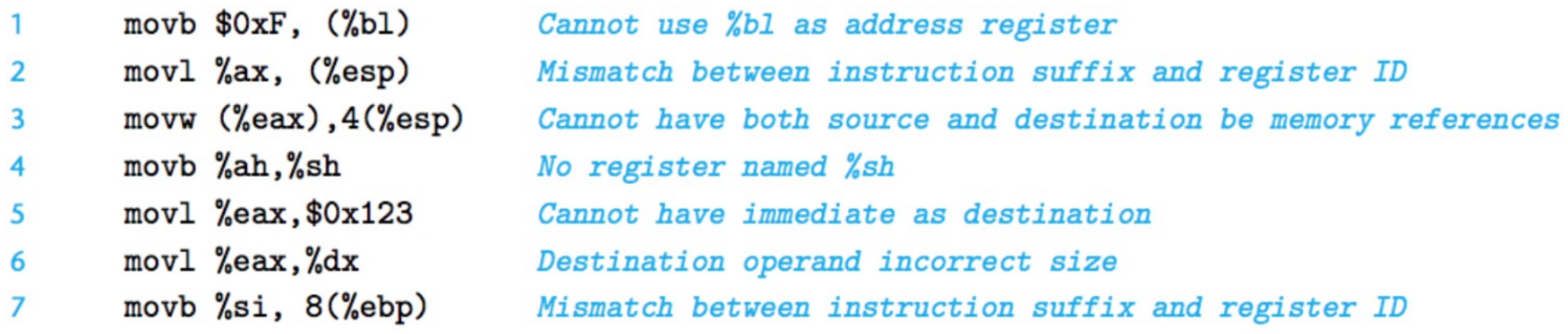
Type	Example	Comments
Direct	movl %eax, 0x0000f	copy value in eax to memory location 0x0000f
Indirect	movl (%ebp), %eax	copy value from memory location whose address is in ebp into eax
Indexed	movl (%ebp, %esi), %eax	copy value from memory at (address = ebp + esi) into eax
Scaled Indexed	movl 0x80(%ebx, %esi, 4), %eax	copy value from memory at (address = ebx + esi*4 + 0x80) into eax 0x80 is an address offset

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?	0xFF
• movl (%eax, %edx, 4), %ecx	%ecx?	0x11
• decl %ecx	%ecx?	0x00
• movl 0x004(%eax, %ecx, 8), %edx	%edx?	0x11



1	`movb $0xF, (%bl)`	*Cannot use %bl as address register*
2	`movl %ax, (%esp)`	*Mismatch between instruction suffix and register ID*
3	`movw (%eax),4(%esp)`	*Cannot have both source and destination be memory references*
4	`movb %ah,%sh`	*No register named %sh*
5	`movl %eax,$0x123`	*Cannot have immediate as destination*
6	`movl %eax,%dx`	*Destination operand incorrect size*
7	`movb %si, 8(%ebp)`	*Mismatch between instruction suffix and register ID*

Some arithmetic operation

Instruction		Computation
`addl`	Src, Dest	Dest = Dest + Src
`subl`	Src, Dest	Dest = Dest - Src
`imull`	Src, Dest	Dest = Dest * Src
`sall`	Src, Dest	Dest = Dest << Src (left shift)
`sarl`	Src, Dest	Dest = Dest >> Src (right shift)
`xorl`	Src, Dest	Dest = Dest ^ Src
`andl`	Src, Dest	Dest = Dest & Src
`orl`	Src, Dest	Dest = Dest | Src
`incl`	Dest	Dest = Dest + 1
`decl`	Dest	Dest = Dest - 1
`negl`	Dest	Dest = - Dest
`notl`	Dest	Dest = ~ Dest

Jump Operation

- Jump to different part of code depending on condition codes

jX	Condition	Description
`jmp`	`1`	Unconditional
`je`	`ZF`	Equal / Zero
`jne`	`~ZF`	Not Equal / Not Zero
`js`	`SF`	Negative
`jns`	`~SF`	Nonnegative
`jg`	`~(SF^OF)&~ZF`	Greater (Signed)
`jge`	`~(SF^OF)`	Greater or Equal (Signed)
`jl`	`(SF^OF)`	Less (Signed)
`jle`	`(SF^OF)|ZF`	Less or Equal (Signed)
`ja`	`~CF&~ZF`	Above (unsigned)
`jb`	`CF`	Below (unsigned)

Movement operation

- Five possible combinations of source and destination types
  - No memory to memory transfers with single instruction

```
movl $0x4050,%eax       Immediate--Register, 4 bytes
movw  %bp,%sp           Register--Register,  2 bytes
movb (%edi,%ecx),%ah    Memory--Register,    1 byte
movb $-17,(%esp)        Immediate--Memory,   1 byte
movl %eax,-12(%ebp)     Register--Memory,    4 bytes
```

Instruction		Effect
MOV	S, D	$D \leftarrow S$
movb		Move byte
movw		Move word
movl		Move double word

Q&A

Thanks!