w11-Lec

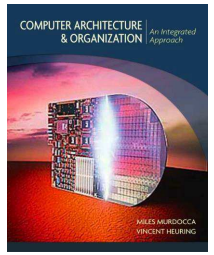
Machine Instruction Cycle

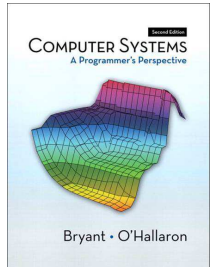
Assembled for 204111

by Ratsameetip Wita

Outline

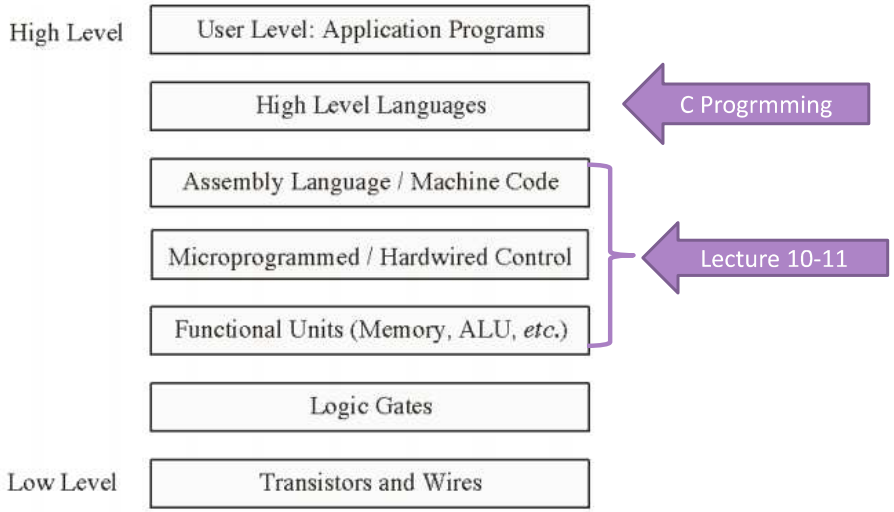
- **Running a program in Von Neumann Architecture**
- ISA -Instruction Set Architecture
  - CISC and RISC
  - Load-Store Architecture
- Addressing Mechanism
- Basic Instruction in AI32
- Assembly and Machine Code





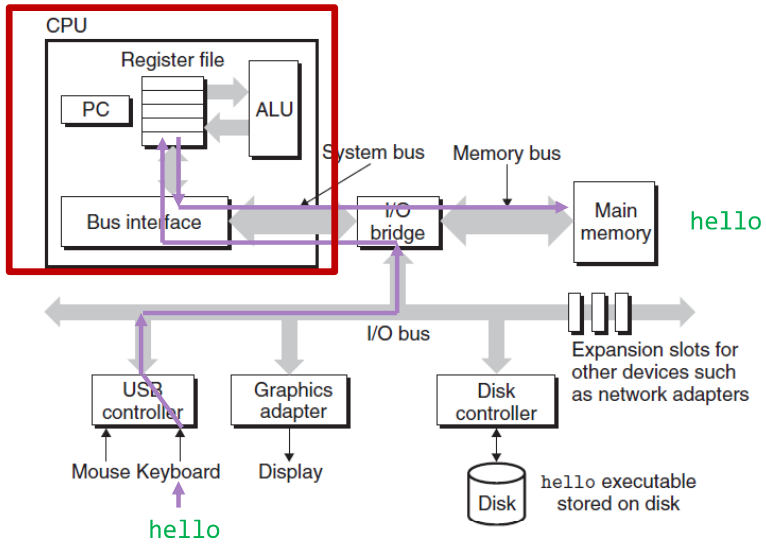
*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

Levels of Machines



*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

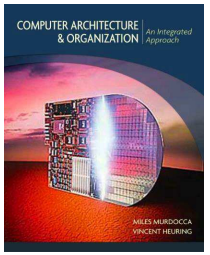
Running "hello" Program (Revisited)

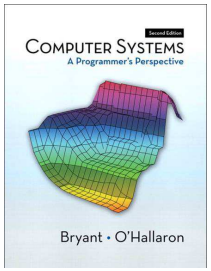


*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

Outline

- Running a program in Von Neumann Architecture
- **ISA -Instruction Set Architecture**
  - **CISC and RISC**
  - **Load-Store Architecture**
- Addressing Mechanism
- Basic Instruction in AI32
- Assembly and Machine Code





*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

Instruction Set Architecture -ISAs

- **ส่วนที่เชื่อมต่อระหว่าง Hardware และ Software**
- **เป็นขั้นตอนการทำงานที่ตกลงไว้อย่างดี (ทำอะไร และทำอย่างไร)**
  - **Functional definition of operations, modes, and storage locations supported by hardware รูปแบบของการทำงาน วิธีการ ที่อยู่ที่เก็บข้อมูลจริง**
  - **Precise description of how to invoke, and access them เรียกคำสั่งอย่างไร เก็บข้อมูลและอ่านข้อมูลอย่างไร**
  - **ISAs เป็นชุดคำสั่งเฉพาะสถาปัตยกรรมฮาร์ดแวร์ มีรูปแบบแตกต่างกันออกไป (ในที่นี่อ้างอิง ARC/SPARC* และ IA32**)**

*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

History of ISAs

- **CISC (Complex Instruction Set Computer) การสร้างสถาปัตยกรรมของคอมพิวเตอร์โดยใช้ชุดคำสั่งที่ซับซ้อน**
  - **เช่น โครงสร้างของสถาปัตยกรรม Intel ช่วงปี 90’ x86, VAX, Motorola 68000**
- **RISC (Reduce Instruction Set Computer) การสร้างสถาปัตยกรรมของคอมพิวเตอร์โดยการลดชุดของคำสั่ง**
  - **เช่น MIPS, PA-RISC, SPARC, PowerPC, Alpha, ARM**

*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

CISC Architecture

- **CISC (Complex Instruction Set Computer) การสร้างสถาปัตยกรรมของคอมพิวเตอร์โดยใช้ชุดคำสั่งที่ซับซ้อน**
- **มีขนาดของ instruction เปลี่ยนแปลงได้ เป็นจำนวนเท่าของ 8 bit (8, 16, 32, 64)**
- **มีการระบุคำสั่งแบบ Fixed code (1 ชุดคำสั่งต่อ 1 การทำงาน)**
- **เพิ่มการทำงาน = เพิ่มชุดคำสั่ง**
- **ในการเก็บชุดคำสั่งของ CISC นั้นจะเก็บเท่ากับจำนวนจริงของการใช้งาน จึงประหยัดเนื้อที่ในหน่วยความจำ**

*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

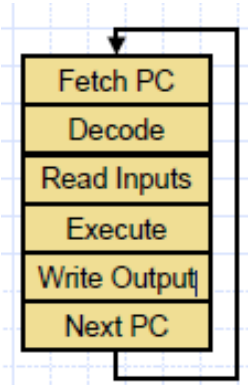
RISC Architecture

- **RISC (Reduce Instruction Set Computer)** การสร้างสถาปัตยกรรมของคอมพิวเตอร์โดยการลดชุดของคำสั่ง
- ออกแบบให้ซีพียู (CPU) ทำงานในวงรอบสัญญาณนาฬิกา (Cycle) ที่แน่นอน
- ลดจำนวนคำสั่งลงให้เหลือเป็นคำสั่งพื้นฐานมากที่สุด
- ใช้การทำงานของ CPU ร่วมกับ Registers
- เก็บชุดคำสั่งในรูปแบบ **Fix-length Encoding**
- มีขั้นตอนการทำงานระดับ Hardware แบบ **Load-Store (Fetch-Execute)**

*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

Load-Store (Fetch-Execute) Architecture

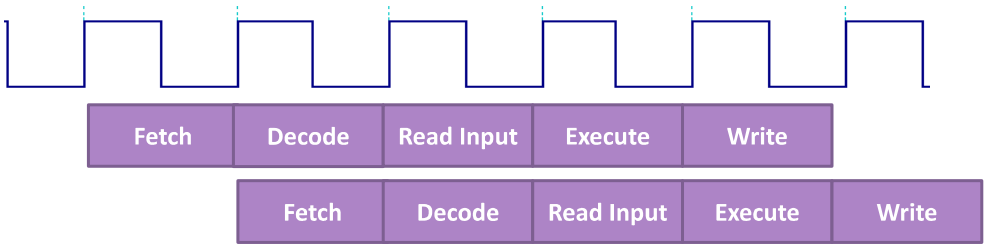
- มีขั้นตอนในการทำงานกำหนดโดยรอบสัญญาณนาฬิกา (Clock Cycle)
- ทำการ Load ข้อมูลเก็บไว้ใน Register โดยตรงและให้ Register ทำการประมวลผลจากนั้นค่อย Store เก็บไว้ใน Memory
  - **Fetch PC**– ทำการอ่านคำสั่ง ณ ตำแหน่ง Program counter
  - **Decode** - แปลความหมายของคำสั่งที่อ่านขึ้นมา
  - **Read Inputs** – อ่านค่าอินพุทของคำสั่งนั้น ๆ (ถ้ามี)
  - **Execute** - ทำการประมวลผลคำสั่ง
  - **Write Output** – เขียนผลลัพธ์ของคำสั่ง
  - **Next PC** – ขยับ Program Counter ไปตำแหน่งถัดไป



*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

Load-Store Architecture (2)

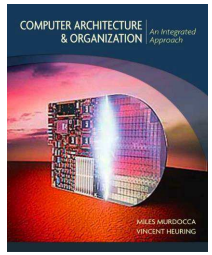
- อาศัยความสามารถของ **Compiler** ในการแปลคำสั่งให้อยู่ในรูปคำสั่งตาม **Architecture**
- ใช้ระยะเวลาการทำงานโดยเฉลี่ยเป็น 1 คำสั่งต่อ **Clock cycle** (โดยอาศัยการทำงานแบบ **Pipeline**)

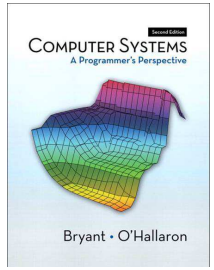


*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

Outline

- Running a program in Von Neumann Architecture
- ISA -Instruction Set Architecture
  - CISC and RISC
  - Load-Store Architecture
- **Addressing Mechanism**
- Basic Instruction in AI32
- Assembly and Machine Code





*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

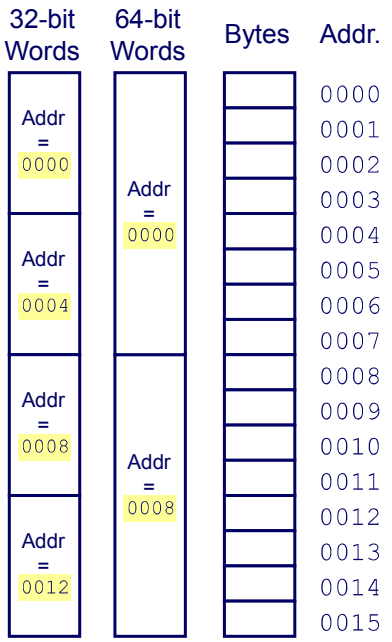
Addressing Mechanism

- ในระดับ Bus ข้อมูลจะมีการส่งในหน่วย word
- 1 word มีขนาดตามสถาปัตยกรรม (8 bit, 16 bit, 32 bit, 64 bit …)
- สถาปัตยกรรมที่มีขนาด word = w bit มีผลทำให้การอ้างตำแหน่งของ Memory มีข้อจำกัดอยู่ที่ 2^w ตำแหน่ง เช่น
- 32-bit word มีขนาด Memory ได้ $2^{32} = 4x10^9 = 4$Gigabytes

*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

Word-Oriented Memory Organization

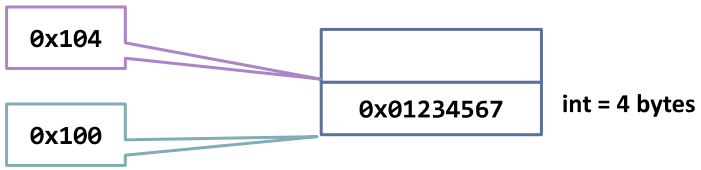
- การกำหนดตำแหน่งของ Address จะระบุตามตำแหน่งของ Byte เรียงตามลำดับ
  - เริ่มจากตำแหน่งแรกของ Byte แรก
  - Address ถัดไปจะขยับในหน่วย word เช่น 4 (32-bit) or 8 (64-bit)



Addressing Mechanism (2)

- ข้อมูล (Value) และที่อยู่ (Address) มีการกำหนดในรูปแบบฐาน 2 (สามารถแสดงในรูปฐาน 16 ได้)
- ในการกำหนดตัวแปร 1 ตัว จะมีเลขฐาน 16 ที่ต่างกัน เพื่อระบุ Value และ Address

```
int x;
x = 0x01234567;// (Value)
&x = 0x100; //(Address)
```



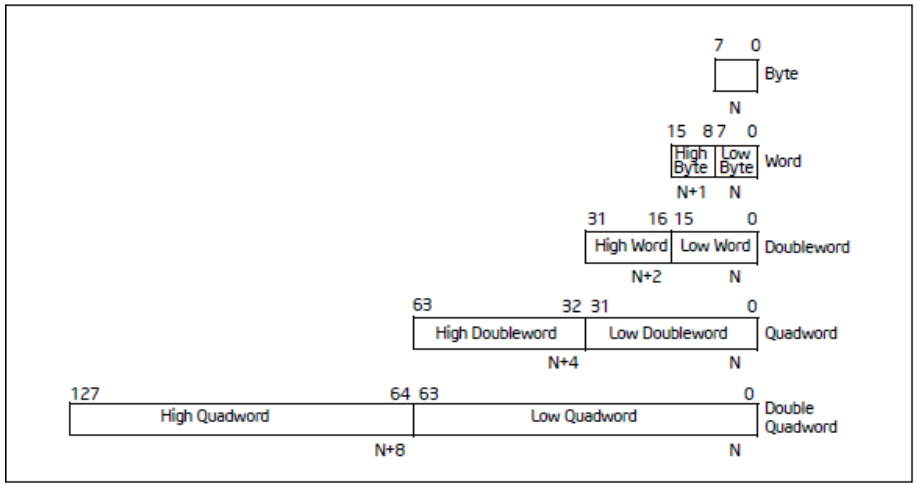
*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

Data Representations: IA32

C Data Type	Intel Data type	Assembly code suffix	Generic 32-bit	Intel IA32
char	Byte	b	1	1
short	Word	w	2	2
unsigned	Double word	l	4	4
int	Double word	l	4	4
long int	Double word	l	4	4
float	Single precision	s	4	4
double	Double precision	l	8	8
long double	Extended precision	t	8	10/12
char*	Double word	l	4	4

*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

Word and Data Types

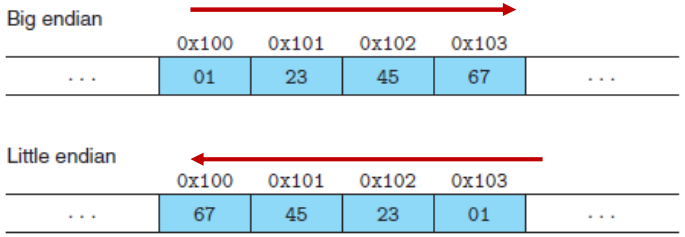


*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

Big Endian and Little Endian

- ในแต่ละระบบจะมีลำดับการอ้างถึงตำแหน่งในการเก็บข้อมูลที่แตกต่างกัน
  - **Machine code จึงมีความแตกต่างกันในแต่ละสถาปัตยกรรม**
  - **Big Endian: Sun, PPC Mac, Internet**
    - **Least significant byte has highest address**
  - **Little Endian: x86**
    - **Least significant byte has lowest address**

`x = 0x01234567`

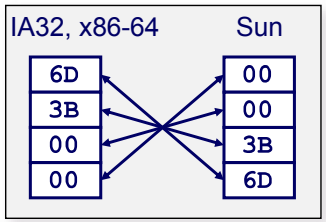


*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

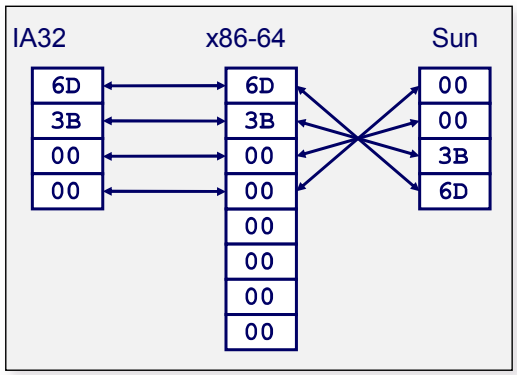
Representing Inte

Decimal:	15213
Binary:	0011 1011 0110 1101
Hex:	3 B 6 D

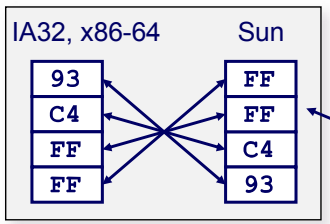
int A = 15213;



long int C = 15213;



int B = -15213;



Two's complement representation

Byte representations of different data values

```
1 void test_show_bytes(int val) {
2   int ival = val;
3   float fval = (float) ival;  //casting int to float
4   int *pval = &ival;  //ตัวแปร *pval  เก็บตำแหน่งของ ival
5   show_int(ival);
6   show_float(fval);
7   show_pointer(pval);
8 }
```

Var	Value	Type	Bytes(Hex)
ival	12,345	int	39 30 00 00
fval	12,345.0	float	00 e4 40 46
pval	&ival	int	b4 cc 22 00

*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

Byte representations of different data values

```
1 void test_show_bytes(int val) {
2   int ival = val;
3   float fval = (float) ival;
4   int *pval = &ival;
5   show_int(ival);
6   show_float(fval);
7   show_pointer(pval);
8 }
```

Machine	Value	Type	Bytes(Hex)
Linux32	12,345	int	39 30 00 00
Windows	12,345	int	39 30 00 00
Sun	12,345	int	00 00 30 39
Linux64	12,345	int	39 30 00 00
Linux32	12,345.0	float	00 e4 40 46
Windows	12,345.0	float	00 e4 40 46
Sun	12,345.0	float	46 40 e4 00
Linux64	12,345.0	float	00 e4 40 46
Linux32	&ival	int	e4 f9 ff bf
Windows	&ival	int	b4 cc 22 00
Sun	&ival	int	ef ff fa 0c
Linux64	&ival	int	b8 11 e5 ff ff 7f 00 00

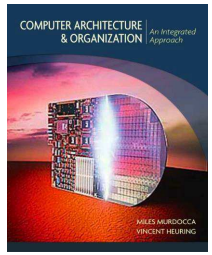
little endian

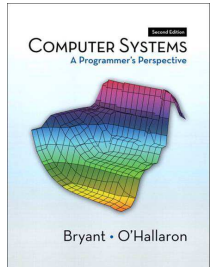
big endian

*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

Outline

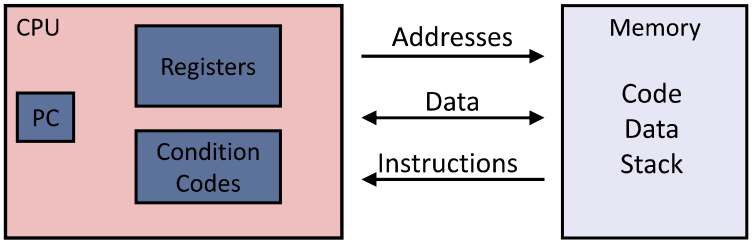
- Running a program in Von Neumann Architecture
- ISA -Instruction Set Architecture
  - CISC and RISC
  - Load-Store Architecture
- Addressing Mechanism
- **Basic Instruction in AI32**
- Assembly and Machine Code





*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

Programmer's View



- Programmer-Visible State
  - PC: Program counter
    - Address of next instruction
  - Register file
    - Heavily used program data
  - Condition codes
    - Store status information about most recent arithmetic operation
    - Used for conditional branching
- Memory
  - Byte addressable array
  - Code and user data
  - Stack to support procedures

*Compute st Edition, 2007 **Computer Systems: A Programmer's Perspective, 2nd Edition

Type of Instruction in AI32

- **Data transfer: move from source to destination**
- **Arithmetic: arithmetic on integer**

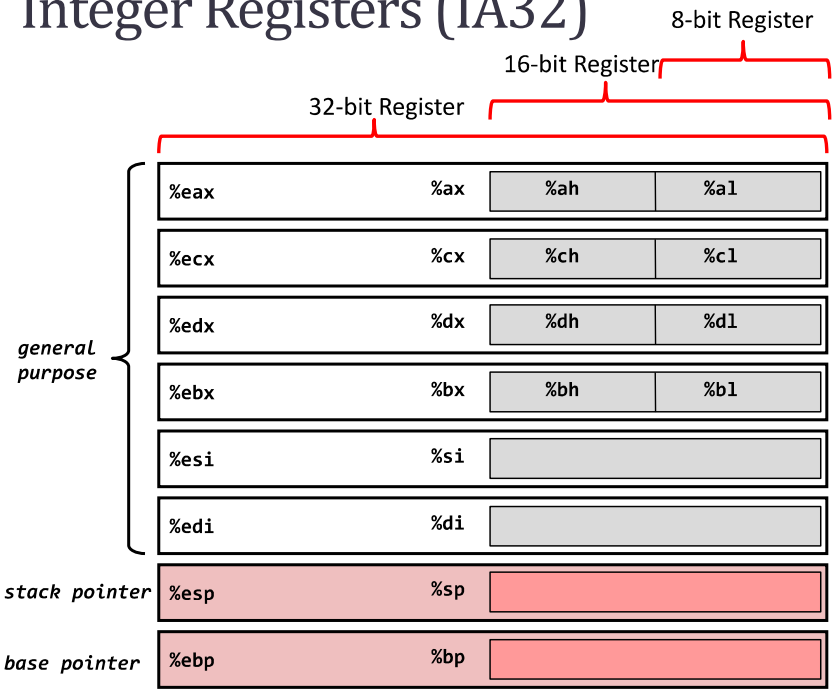
Overview in 204111

- **Floating point: x87 FPU move, arithmetic**
- **Logic: bitwise logic operations**
- **Control transfer: conditional and unconditional jumps, procedure calls**
- **String: move, compare, input and output**
- **Flag control: Control fields in EFLAGS**
- **Segment register: Load far pointers for segment registers**
- **SIMD**
  - **MMX: integer SIMD instructions**
  - **SSE: 32-bit and 64-bit floating point SIMD instructions**
  - **SSE2: 128-bit integer and float point SIMD instructions**
- **System**
  - **Load special registers and set control registers (including halt)**

Later in Comp Architecture Class

*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

Integer Registers (IA32)

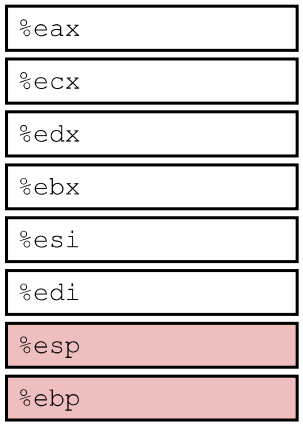


Basic Instruction in AI32

Instruction	Effect	Description
MOV S,D	D <- S	Move from S to D
movb	Move byte	
movw	Move word	
movl	Move double word	
pushl S	R[%esp]←R[%esp]– 4; M[R[%esp]]←*S*	Push double word
popl D	*D*←M[R[%esp]]; R[%esp]←R[%esp]+ 4	Pop double word

*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

Moving Data: IA32



- **Moving Data**
  `movl` ***Source, Dest*:**
- **Operand Types**
  - ***Immediate (Imm):*** ค่าคงที่ ระบุโดยกำหนด '$' ไว้
    - Example: **`$0x400`, `$-533`**
    - Encoded ในรูปแบบ 1, 2, หรือ 4 bytes
  - ***Register (Reg):*** **One of 8 integer registers**
    - Example: **`%eax`, `%edx`**
    - ยกเว้น **`%esp`** และ **`%ebp`** เก็บไว้เป็นส่วน Program counter
  - ***Memory (Mem):*** การอ้างถึงตำแหน่งใน memnory ที่เก็บไว้ใน register
    - Simplest example: **`(%eax)`**
    - Various other "address modes"

*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

`movl` Operand Combinations

	Source	Dest	Src,Dest	C Analog
movl	*Imm*	*Reg*	`movl $0x4,%eax`	`temp = 0x4;`
		Mem	`movl $-147,(%eax)`	`*p = -147;`
	Reg	*Reg*	`movl %eax,%edx`	`temp2 = temp1;`
		Mem	`movl %eax,(%edx)`	`*p = temp;`
	Mem	*Reg*	`movl (%eax),%edx`	`temp = *p;`

การ mov ระหว่าง memory-memory ไม่สามารถทำได้ใน single instruction

*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

Simple Memory Addressing Modes

- **Normal (R)** Mem[Reg[R]]
  - Register R เก็บค่าตำแหน่งใน Memory

```
movl (%ecx),%eax
```
//ย้ายตำแหน่งของ Memory ที่เก็บใน %ecx ไปเก็บไว้ใน %eax

- **Displacement D(R)** Mem[Reg[R]+D]
  - Register R เก็บตำแหน่งเริ่มต้นในช่วง Memory
  - D เป็นค่า offset หรือจำนวน Byte ที่จะขยับ

```
movl 8(%ebp),%edx
```
// เก็บตำแหน่งที่อยู่ใน %ebp+8 ไว้ใน %edx

*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

Operand Forms

Type	Form	Operand value	Name
Immediate	$Imm	Imm	Immediate
Register	E_a	$R[E_a]$	Register
Memory	Imm	$M[Imm]$	Absolute
Memory	(E_a)	$M[R[E_a]]$	Indirect
Memory	$Imm(E_b)$	$M[Imm + R[E_b]]$	Base + displacement
Memory	(E_b, E_i)	$M[R[E_b] + R[E_i]]$	Indexed
Memory	$Imm(E_b, E_i)$	$M[Imm + R[E_b] + R[E_i]]$	Indexed
Memory	$(, E_i, s)$	$M[R[E_i] \cdot s]$	Scaled indexed
Memory	$Imm(, E_i, s)$	$M[Imm + R[E_i] \cdot s]$	Scaled indexed
Memory	(E_b, E_i, s)	$M[R[E_b] + R[E_i] \cdot s]$	Scaled indexed
Memory	$Imm(E_b, E_i, s)$	$M[Imm + R[E_b] + R[E_i] \cdot s]$	Scaled indexed

Figure 3.3 **Operand forms.** Operands can denote immediate (constant) values, register values, or values from memory. The scaling factor s must be either 1, 2, 4, or 8.

*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

Basic mov instruction Sample

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

*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

Practice I: Addressing

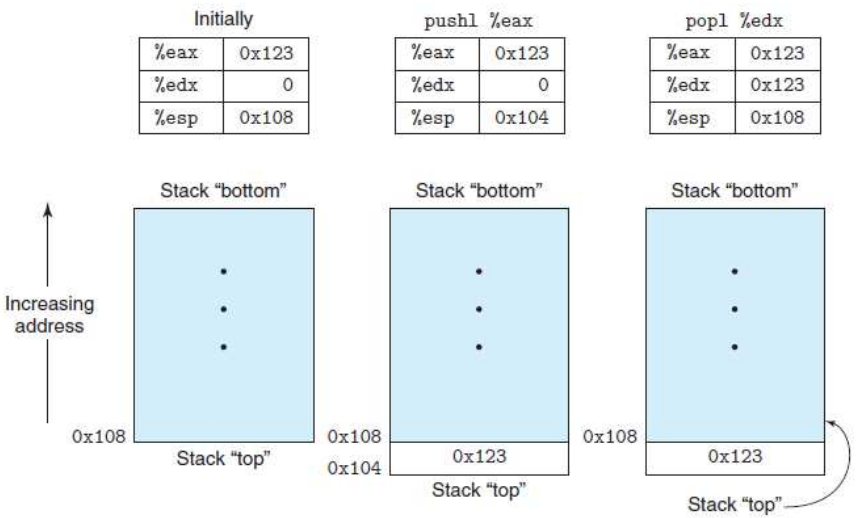
- ให้ค่า Address ใน Memory และ Register ดังนี้

Address	Value	Register	Value
0x100	0xFF	%eax	0x100
0x104	0XAB	%ecx	0x1
0x108	0x13	%edx	0x3
0x10c	0x11		

- ให้เติมค่าที่ถูกต้องเมื่อกำหนด operand ดังนี้

Operand	Value	Operand	Value
%eax	0x100	260(%ecx,%edx)	
0x104		0xFFC(,%ecx,4)	
(%eax)		(%eax,%edx,4)	
9(%eax,%edx)			

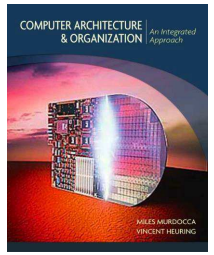
Basic Stack Instruction

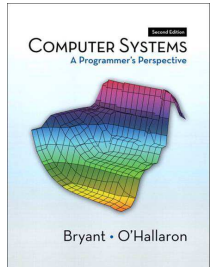


*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

Outline

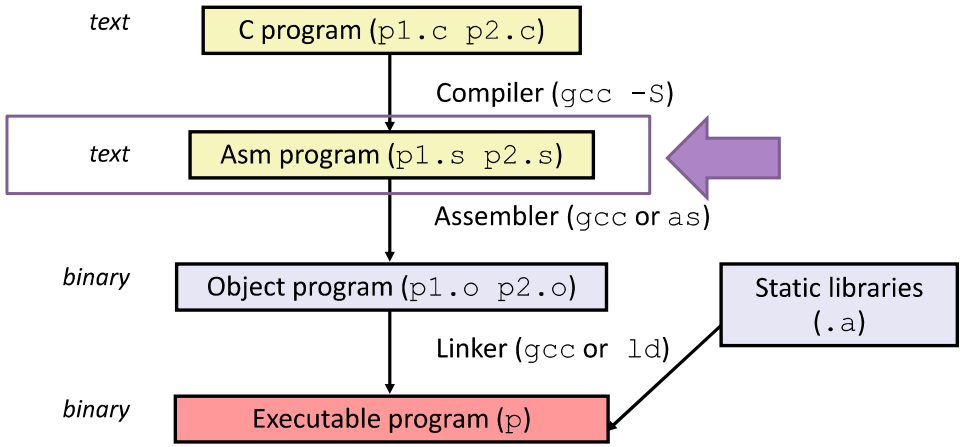
- Running a program in Von Neumann Architecture
- ISA -Instruction Set Architecture
  - CISC and RISC
  - Load-Store Architecture
- Addressing Mechanism
- Basic Instruction in AI32
- **Assembly and Machine Code**





*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

Compiling Into Assembly



*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

Compiling Into Assembly

```
int sum(int x, int y)
{
  int t = x+y;
  return t;
}
```

C Code

Generated IA32 Assembly

```
sum:
  pushl %ebp          //เริ่มต้นตำแหน่งของโปรแกรม
  movl %esp,%ebp      //เริ่มตั้งต่าตัวชี้ตำแหน่งโปรแกรม
  movl 12(%ebp),%eax  //ย้ายข้อมูลที่ตำแหน่ง %ebp+12ไว้ใน %edx
  addl 8(%ebp),%eax   //บวกข้อมูลที่ตำแหน่ง %ebp+8ไว้ใน %edx
  popl %ebp           //ส่งคืนค่าตำแหน่งปัจจุบันของโปรแกรม
  ret                 //รีเทิร์นค่า
```

*Compi

Machine Instruction Example

```
int t = x+y;
```

- C Code
  - Add two signed integers

```
addl 8(%ebp),%eax
```

```
Similar to expression:
x += y
More precisely:
int eax;
int *ebp;
eax += ebp[2]
```

```
0x80483ca:  03 45 08
```

- Assembly
  - Add 2 4-byte integers
    - "Long" words in GCC parlance
    - Same instruction whether signed or unsigned
  - Operands:
    - x: Register %eax
    - y: Memory M[%ebp+8]
    - t: Register %eax
    - Return function value in %eax
- Object Code
  - 3-byte instruction
  - Stored at address 0x80483ca

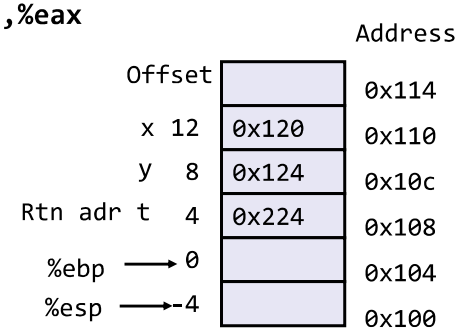
*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

Compiling Into Assembly

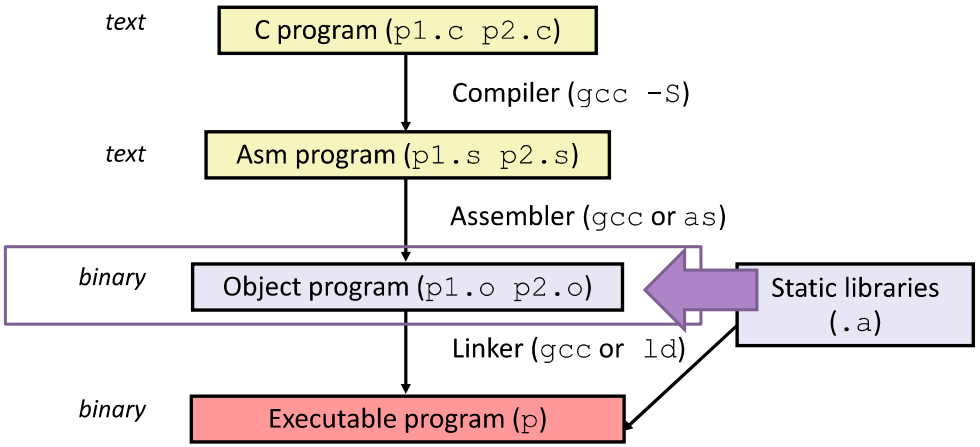
```
int sum(int x, int y)
{
  int t = x+y;
  return t;
}
```

```
sum:
 pushl %ebp
 movl %esp,%ebp
 movl 12(%ebp),%eax
 addl 8(%ebp),%eax
 popl %ebp
 ret
```

%eax	0x120
%edx	
%ecx	
%ebx	
%esi	
%edi	
%esp	
%ebp	0x104



Turning C into Object Code



*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

Object Code

- Assembler
  - แปลงไฟล์ .s เป็นไฟล์ .o
  - แปลงคำสั่งเป็นเลขฐานสอง
  - ใกล้เคียงกับ executable code
  - ขาดส่วนที่ลิ้งค์โค้ดโปรแกรมกับ Reference ต่าง ๆ
- Linker
  - เชื่อมโยงคำสั่งระหว่างโค้ดในไฟล์ต่าง ๆ
  - เชื่อมโยงคำสั่ง static run-time libraries
    - E.g., code for malloc, printf
  - Some libraries are *dynamically linked*
    - Linking occurs when program begins execution

Code for sum

```
0x401040 <sum>:
    0x55
    0x89
    0xe5
    0x8b
    0x45
    0x0c
    0x03
    0x45
    0x08
    0x5d
    0xc3
```

- Total of 11 bytes
- Each instruction 1, 2, or 3 bytes
- Starts at address 0x401040

*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

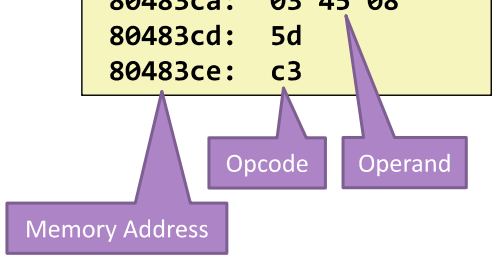
Disassembling Object Code

Object code

```
080483c4 <sum>:
 80483c4:  55
 80483c5:  89 e5
 80483c7:  8b 45 0c
 80483ca:  03 45 08
 80483cd:  5d
 80483ce:  c3
```

Assembly

```
pushl   %ebp
movl    %esp,%ebp
movl    0xc(%ebp),%eax
addl    0x8(%ebp),%eax
popl    %ebp
ret
```



*Computer Architecture and Organization: An Integrated Approach, 1st Edition, 2007
**Computer Systems: A Programmer's Perspective, 2nd Edition

Practice II: Opcode

- ในแต่ละคำสั่งด้านล่าง ให้เติม **instruction suffix**ให้เหมาะสมกับขนาดของ **Operand** ที่กำหนดให้

Opcode	Operands	Description
movl	%eax, (%esp)	register -> memory, 4 bytes
mov	(%eax), %dx	
mov	$0xFF, %bl	
mov	(%esp,%edx,4), %dh	
push	$0xFF	
mov	%dx, (%eax)	
pop	%ed	