# CS 261 Fall 2016

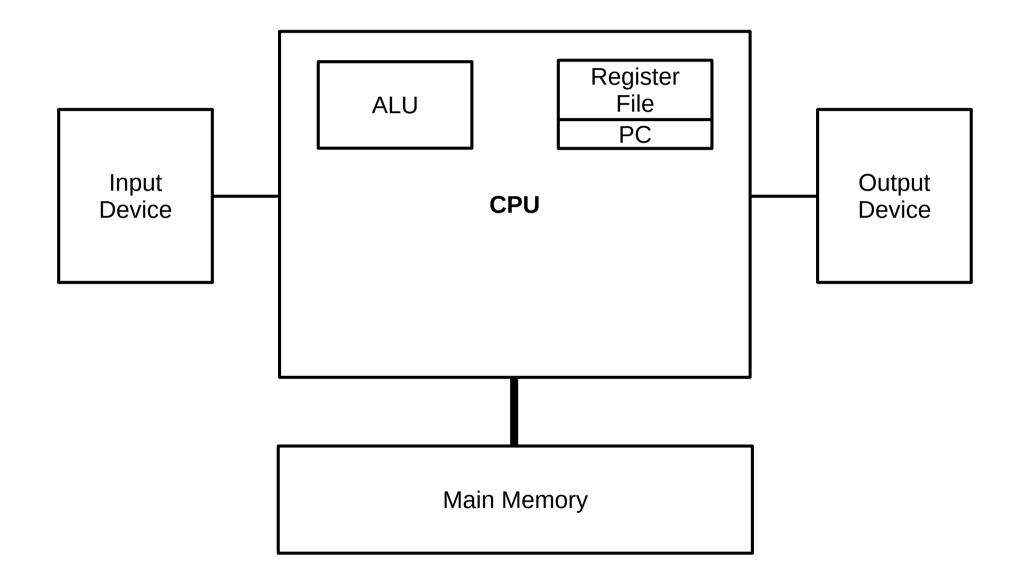
Mike Lam, Professor

### x86-64 Assembly

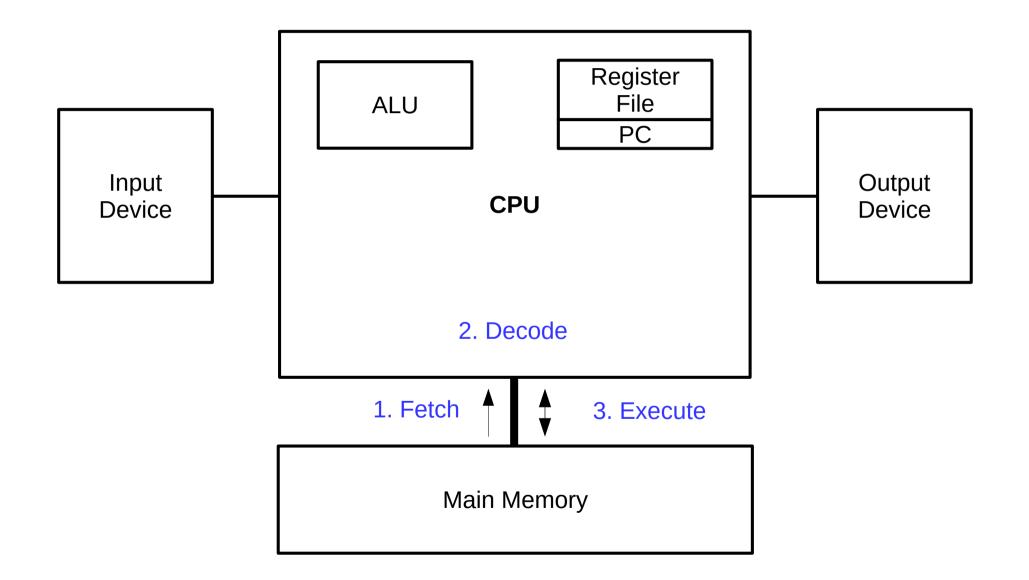
# Topics

- Architecture/assembly intro
- Data formats
- Data movement
- Arithmetic and logical operations

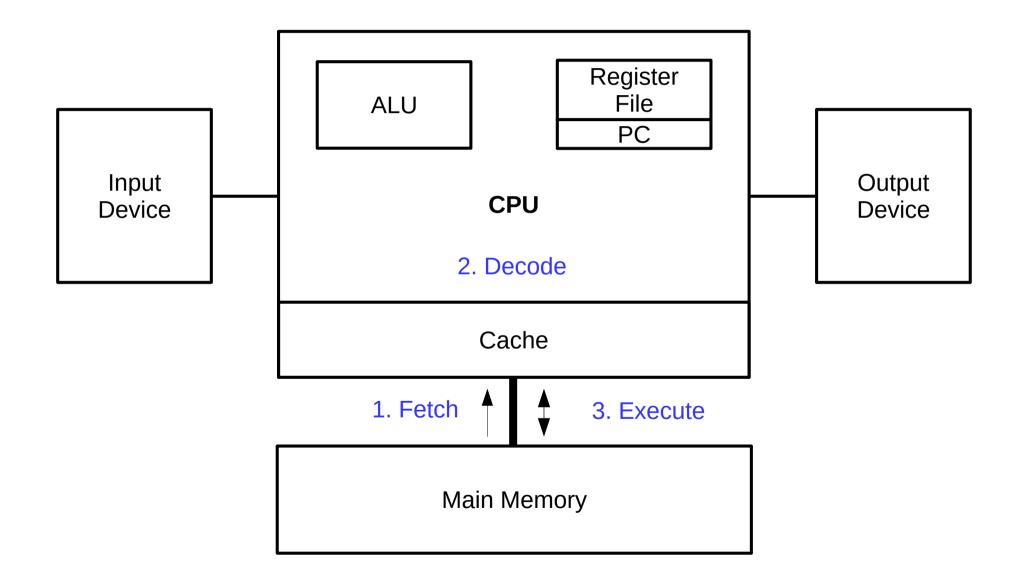
# von Neumann architecture



# von Neumann architecture



# von Neumann architecture



# Assembly programming

- Assembly: simple, CPU-specific programming language
  - However, x86-64 has become the industry standard
  - Based on fetch/decode/execute execution loop
  - Program is stored on disk along with data
  - Low-level access to machine (memory, I/O, etc.)
  - Each instruction = opcode and operands
  - Compilers often target assembly code instead of machine code for increased portability
  - Understanding assembly code can help you optimize and secure your programs

# Assembly code operand types

- Immediate
  - Operand embedded in instruction itself
  - Written in assembly using "\$" prefix (e.g., \$42 or \$0x1234)
- Register
  - Operand stored in register file
  - Accessed by register number
  - Written in assembly using name and "%" prefix (e.g., %eax or %rsp)
- Memory
  - Operand stored in main memory
  - Accessed by effective address
  - Written in assembly using a variety of addressing modes

# Registers

- General-purpose
  - AX: accumulator
  - BX: base
  - CX: counter
  - DX: address
  - SI: source index
  - DI: dest index
- Special
  - BP: base pointer
  - SP: stack pointer
  - IP: instruction pointer
  - FLAGS: status info

register encoding	zero-extended for 32-bit operands	1	-	low 8-bit	16-bit	32-bit	64-bi
0			AH*	AL	AX	EAX	RAX
3			BH*	BL	BX	EBX	RBX
1			CH*	CL	CX	ECX	RCX
2			DH*	DL	DX	EDX	RDX
6				SIL**	SI	ESI	RSI
7				DIL**	DI	EDI	RDI
5				BPL**	BP	EBP	RBP
4				SPL**	SP	ESP	RSP
8				R8B	R8W	R8D	R8
9				R9B	R9W	R9D	R9
10				R10B	R10W	R10D	R10
11				R11B	R11W	RIID	R11
12				R12B	R12W	R12D	R12
13				R13B	R13W	R13D	R13
14				R14B	<b>R14W</b>	R14D	R14
15				R15B	R15W	R15D	R15
63	32	2 31	16 15 8	7 0			
	0				RFLAGS	51	3-309.005
					RIP		
63	37	2 31		0		ddressable ( prefix is us	

# Memory addressing modes

- Absolute: mov \$1, x
  - Moves to M[x]
- Indirect: mov \$1, (r)
  - Moves to M[R[r]]
- Base + displacement: mov \$1, x(r)
  - Moves to M[x + R[r]]
- Indexed: mov  $1, x(r_b, r_i)$ 
  - Moves to  $M[x + R[r_b] + R[r_i]]$
- Scaled indexed: mov  $1, x(r_b, r_i, s)$ 
  - Moves to  $M[x + R[r_b] + R[r_i] \cdot s]$
  - Scale (s) must be 1, 2, 4, or 8

- Given the following machine status, what is the value for the following assembly operands?
  - \$42 - \$0x10 – %rax - 0x104 - (%rax)
  - 4(%rax)
  - 2(%rax, %rdx)
  - (%rax, %rdx, 4)

#### **Registers**

<u>Name</u>	<u>Value</u>
%rax	0x100
%rdx	0x2

#### Memory

<u>Address</u>	<u>Value</u>
0x100	0xFF
0x104	0xAB
0x108	0x13

- Given the following machine status, what is the value for the following assembly operands?
  - \$42 <mark>42</mark>
  - \$0x10 **16**

  - 2(%rax, %rdx) OxAB
  - (%rax, %rdx, 4) 0x13

#### Registers

<u>Name</u>	<u>Value</u>
%rax	0x100
%rdx	0x2

#### Memory

<u>Address</u>	<u>Value</u>
0x100	0xFF
0x104	0xAB
0x108	0x13

# Brief aside: data formats

- Historical artifact: "word" in x86 is 16-bit
  - 1 byte (8 bits) = "byte" (b)
  - 2 bytes (16 bits) = "word" (W)
  - 4 bytes (32 bits) = "double word" (1)
  - 8 bytes (64 bits) = "quad word" (q)

## Data movement

- Often, a "class" of instructions will perform similar jobs, but on different sizes of data
- Primary data movement instruction: "mov"
  - movb, movw, movl, movq, movabsq
- Zero-extension variant: "movz"
  - movzbw, movzbl, movzwl, movzbq, movzwq
- Sign-extension variant: "movs"
  - movsbw, movsbl, movswl, movsbq, movswq, movslq

# Stack management

- Push/pop instructions: pushq and popq
  - 8-byte (quadword) slots, growing "downward" from high addresses to low addresses
- Register %rsp stores address of top of stack
  - i.e., a pointer to the last value pushed
- pushq
  - Subtract 8 from stack pointer
  - Store value at new stack top location (%rsp)
- popq
  - Retrieve value at current stack top (%rsp)
  - Increment stack pointer by 8

- Given the following register state, what will the values of the registers be after the following instruction sequence?
  - pushq %rax
  - pushq %rcx
  - pushq %rbx
  - pushq %rdx
  - popq %rax
  - popq %rbx
  - popq %rcx
  - popq %rdx

#### **Registers**

<u>Name</u>	<u>Value</u>
%rax	0xAA
%rbx	0xBB
%rcx	0xCC
%rdx	0xDD

- Given the following register state, what will the values of the registers be after the following instruction sequence?
  - pushq %rax
  - pushq %rcx
  - pushq %rbx
  - pushq %rdx
  - popq %rax %rax =  $0 \times DD$
  - popq %rbx
  - popq %rcx
  - popq %rdx

%rbx = 0xbb %rbx = 0xBB %rcx = 0xCC %rdx = 0xAA

Registers			
<u>Name</u>	<u>Value</u>		
%rax	0xAA		
%rbx	0xBB		
%rcx	0xCC		
%rdx	0xDD		

# **Arithmetic operations**

Instruction		Effect	Description
leaq	S, D	$D \leftarrow \&S$	Load effective address
INC	D	$D \leftarrow D+1$	Increment
DEC	D	$D \leftarrow D-1$	Decrement
NEG	D	$D \leftarrow -D$	Negate
NOT	D	$D \leftarrow \neg D$	Complement
ADD	<i>S</i> , <i>D</i>	$D \leftarrow D + S$	Add
SUB	<i>S</i> , <i>D</i>	$D \leftarrow D - S$	Subtract
IMUL	<i>S</i> , <i>D</i>	$D \leftarrow D * S$	Multiply
XOR	<i>S</i> , <i>D</i>	$D \leftarrow D^{s}$	Exclusive-or
OR	<i>S</i> , <i>D</i>	$D \leftarrow D \mid S$	Or
AND	S, D	$D \leftarrow D \& S$	And
SAL	k, D	$D \leftarrow D << k$	Left shift
SHL	k, D	$D \leftarrow D << k$	Left shift (same as SAL)
SAR	k, D	$D \leftarrow D >>_A k$	Arithmetic right shift
SHR	k, D	$D \leftarrow D >>_{\mathrm{L}} k$	Logical right shift

**Figure 3.10 Integer arithmetic operations.** The load effective address (leaq) instruction is commonly used to perform simple arithmetic. The remaining ones are more standard unary or binary operations. We use the notation  $>>_A$  and  $>>_L$  to denote arithmetic and logical right shift, respectively. Note the nonintuitive ordering of the operands with ATT-format assembly code.

Instru	iction	Effect	Description	Registers
leaq	S, D	$D \leftarrow \&S$	Load effective address	Name Value
INC DEC NEG	D D D	$D \leftarrow D+1$ $D \leftarrow D-1$ $D \leftarrow -D$	Increment Decrement Negate	%rax 0x12 %rbx 0x56 %rcx 0x02
NOT	D	$D \leftarrow \neg D$	Complement	%rdx 0xF0
ADD	<i>S</i> , <i>D</i>	$D \leftarrow D + S$	Add	
SUB	S, D	$D \leftarrow D - S$	Subtract	What are the values of all registers
IMUL	S, D	$D \leftarrow D * S$	Multiply	after the following instructions?
XOR	S, D	$D \leftarrow D^{S}$	Exclusive-or	3
OR	S, D	$D \leftarrow D \mid S$	Or	addq %rax, %rax
AND	S, D	$D \leftarrow D \& S$	And	subq %rax, %rbx
SAL	k, D	$D \leftarrow D << k$	Left shift	imulq %rcx, %rax andq %rbx, %rdx
SHL	k, D	$D \leftarrow D << k$	Left shift (same as SAL)	andq %rbx, %rdx shrq \$4, %rdx
SAR	k, D	$D \leftarrow D >>_{A} k$	Arithmetic right shift	
SHR	k, D	$D \leftarrow D >>_{\rm L} k$	Logical right shift	

**Figure 3.10 Integer arithmetic operations.** The load effective address (leaq) instruction is commonly used to perform simple arithmetic. The remaining ones are more standard unary or binary operations. We use the notation  $>>_A$  and  $>>_L$  to denote arithmetic and logical right shift, respectively. Note the nonintuitive ordering of the operands with ATT-format assembly code.

1201				
Instru	ction	Effect	Description	Registers
leaq	S, D	$D \leftarrow \&S$	Load effective address	Name Value
INC DEC NEG NOT	D D D D	$D \leftarrow D+1$ $D \leftarrow D-1$ $D \leftarrow -D$ $D \leftarrow -D$	Increment Decrement Negate Complement	%rax 0x12 %rbx 0x56 %rcx 0x02 %rdx 0xF0
ADD SUB IMUL XOR	S, D S, D S, D S, D	$D \leftarrow D + S$ $D \leftarrow D - S$ $D \leftarrow D * S$ $D \leftarrow D^{*} S$	Add Subtract Multiply Exclusive-or	What are the values of all registers after the following instructions?
OR AND SAL SHL SAR	S, D S, D k, D k, D k, D	$D \leftarrow D \mid S$ $D \leftarrow D \& S$ $D \leftarrow D << k$ $D \leftarrow D << k$ $D \leftarrow D << k$ $D \leftarrow D >>_A k$	Or And Left shift Left shift (same as SAL) Arithmetic right shift	addq %rax, %rax %rax:0x24 subq %rax, %rbx %rbx:0x32 imulq %rcx, %rax %rax:0x48 andq %rbx, %rdx %rdx:0x30 shrq \$4, %rdx %rdx:0x03
SHR	k, D k, D	$D \leftarrow D >>_{A} k$ $D \leftarrow D >>_{L} k$	Logical right shift	

**Figure 3.10 Integer arithmetic operations.** The load effective address (leaq) instruction is commonly used to perform simple arithmetic. The remaining ones are more standard unary or binary operations. We use the notation  $>>_A$  and  $>>_L$  to denote arithmetic and logical right shift, respectively. Note the nonintuitive ordering of the operands with ATT-format assembly code.

%rax = 0x48 %rbx = 0x32 %rcx = 0x02 %rdx = 0x03

Instru	ction	Effect	Description	
leaq	S, D	$D \leftarrow \&S$	Load effective address	
INC	D	$D \leftarrow D+1$	Increment	
DEC	D	$D \leftarrow D-1$	Decrement	
NEG	D	$D \leftarrow -D$	Negate	
NOT	D	$D \leftarrow \neg D$	Complement	What door the following instruction
ADD	<i>S</i> , <i>D</i>	$D \leftarrow D + S$	Add	What does the following instruction do if $\%$ rax = $0x100?$
SUB	S, D	$D \leftarrow D - S$	Subtract	
IMUL	S, D	$D \leftarrow D * S$	Multiply	leaq (%rax, %rax, 2), %rax
XOR	S, D	$D \leftarrow D^{\circ}S$	Exclusive-or	
OR	S, D	$D \leftarrow D \mid S$	Or	
AND	S, D	$D \leftarrow D \& S$	And	
SAL	k, D	$D \leftarrow D << k$	Left shift	
SHL	k, D	$D \leftarrow D << k$	Left shift (same as SAL)	
SAR	k, D	$D \leftarrow D >>_{A} k$	Arithmetic right shift	
SHR	k, D	$D \leftarrow D >>_{\rm L} k$	Logical right shift	

**Figure 3.10 Integer arithmetic operations.** The load effective address (leaq) instruction is commonly used to perform simple arithmetic. The remaining ones are more standard unary or binary operations. We use the notation  $>>_A$  and  $>>_L$  to denote arithmetic and logical right shift, respectively. Note the nonintuitive ordering of the operands with ATT-format assembly code.

Instru	ction	Effect	Description	
leaq	S, D	$D \leftarrow \&S$	Load effective address	
INC	D	$D \leftarrow D+1$	Increment	
DEC	D	$D \leftarrow D-1$	Decrement	
NEG	D	$D \leftarrow -D$	Negate	
NOT	D	$D \leftarrow \neg D$	Complement	
ADD	S, D	$D \leftarrow D + S$	Add	What does the following instruction do if $\%$ rax = $0x100?$
SUB	<i>S</i> , <i>D</i>	$D \leftarrow D - S$	Subtract	
IMUL	S, D	$D \leftarrow D * S$	Multiply	<pre>leag (%rax, %rax, 2), %rax</pre>
XOR	<i>S</i> , <i>D</i>	$D \leftarrow D^{s}$	Exclusive-or	
OR	<i>S</i> , <i>D</i>	$D \leftarrow D \mid S$	Or	
AND	<i>S</i> , <i>D</i>	$D \leftarrow D \& S$	And	%rax = 0x300
SAL	k, D	$D \leftarrow D << k$	Left shift	(multiply by three)
SHL	k, D	$D \leftarrow D << k$	Left shift (same as SAL)	
SAR	k, D	$D \leftarrow D >>_A k$	Arithmetic right shift	
SHR	k, D	$D \leftarrow D >>_{\rm L} k$	Logical right shift	

**Figure 3.10 Integer arithmetic operations.** The load effective address (leaq) instruction is commonly used to perform simple arithmetic. The remaining ones are more standard unary or binary operations. We use the notation  $>>_A$  and  $>>_L$  to denote arithmetic and logical right shift, respectively. Note the nonintuitive ordering of the operands with ATT-format assembly code.