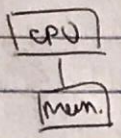
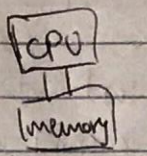
Von Nuemann architecture vs Harvard architecture



- can either retrieve instruction or transfer data



- Can do both at the same time

Modern Computer Components:

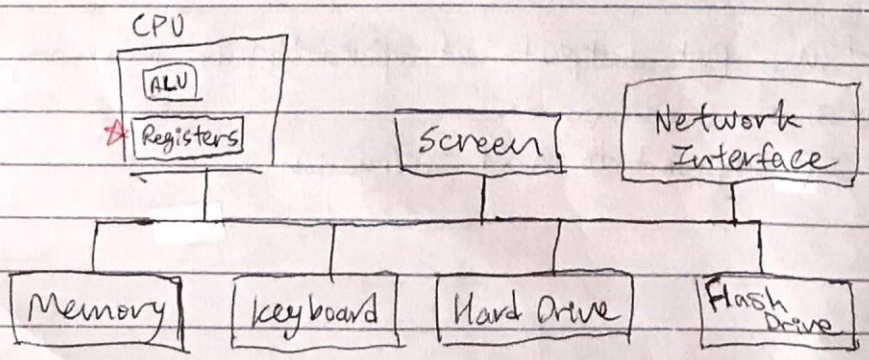
- CPU
  - Architecture logic unit (ALU)
  - Registers (has names)
  - Program Counter / Instruction Pointer (address of next instruction to execute)

- Memory (pattern of bits)
  - Random access, constant time
  - Location specified by numerical address
  - Volatile (when powered, it will lose memory)
  - Holds instructions, data

- BUS
  - Connects CPU, Memory, I/O Devices

- I/O Devices
  - mouse, touchpad
  - keyboard
  - Screen
  - Hard drive / flash drive
  - Network interface
  - Graphics hardware
  - Sound hardware

*Inside the Computer



*Bus limits read/write.

Modern Computer Components:

Arithmetic Logic Unit
- Fixed-point arithmetic operations: Add, Sub, Multiply, Divide
- Floating-point arithmetic operations
- Bitwise operations: And, or, not
- Comparison operations: fixed point, floating point, set condition code
- Control flow: conditional, unconditional jump (Go To, Fortran)
  - unconditional jump → change address to some other address
- Character operations: compare, move
- Interrupt handling

Fetch and Execute Cycle: How machine instructions are executed

- Starting the cycle
  - A hard-wired, architecture-specific values is put into the IP
  - when the computer is first powered up
    - 0xFFFFFFFF.... FFF0 on Intel

- Fetch
  - Get next instruction from memory at address in IP register (Instruction Pointer)

- Decode
  - Determine operation from instruction
  - Get input operands

- Execute
  - Do operation, put outputs of operation in memory or registers as appropriate
  - change IP to point to next instruction

→ used to quickly accept, store, and transfer data and instructions that are being used immediately by the CPU

Registers Line in CPU

Intel:

8 bit		16 bit		32 bit	64 bit
%AH	%AL	%AX	General Purpose Registers	%EAX	%RAX
%BH	%BL	%BX		%EBX	%RBX
%CH	%CL	%CX		%ECX	%RCX
%DH	%DL	%DX		%EDX	%RDX
		%BP	Base Pointers, frame pointer	%EBP	%RBP
		%SP	Stack Pointer	%ESP	%RSP
		%IP	Instruction Pointer	%EIP	%RIP

(General Purpose Registers: %EAX, %EBX, %ECX, %EDX; Dedicated Registers: %EBP, %ESP, %EIP)

Assembly Instruction Operands:

R	R
R	M
M	R
I	R
I	M

R = Register
M = Memory
I = immediate
→ push constant in! (a number)

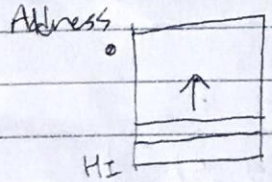
ex) I=3
3 ≠ I BAD!

Stack:

*Intel machines have a stack

- Push, pop values
- Can build a call stack
- Consisting of Stack frames or activation records A/R

*Intel stacks grow from high addresses → toward lower addresses



%RSP dedicated stack pointer register

Push, pop instructions implicitly use %RSP Register
(nothing in machine lang. that overtly change the register)

Intel Push Behavior:

1. Decrement %RSP (to lower address)
2. Copy operand value to where %RSP points

Intel Pop Behavior:

1. Copy value from memory (at address in %RSP) into operand
2. Increment %RSP (to higher address)

Assembly Instruction:

- Mnemonic operands
  - Mnemonic defines operation
  - There may be zero or more operands (depending on the instruction)
  - The assembler translates assembly code into machine instructions
  - Multiple operands are separated by commas

Push %RBX (copy it on the stack)
↳ register

Immediate → Constant value

, $ followed by integer, hex, octal, decimal

Push ex) $0x1F $35 $-35

↳ hardcoded constant → without $, it's an address

gcc -S myprog.c myprog.$ // convert c file to machine assembly file

gcc myprog.$ // compile the machine assembly file

Register:

%EAX %RAX

%EBP %RBP

Absolute address:

An unaddressed numbers, hex, octal, decimal

ex) push 35 (push what's at address 35)

Indirect

(%EAX) → change the value in this address // memory address contained in register

memory * Manipulate memory dynamically

ex) movQ %RAX, (%RBP) // move the value stored at RAX to memory location of RBP

Indirect with fixed displacement:

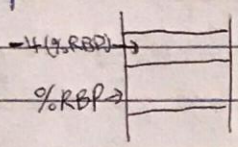
Imm (%RBP) // memory address is immediate value plus contents of register

↑ register

↑ fixed displacement

ex) -4(%RBP) Auto variable

// point to another memory location 4 bytes down



- bits don't have memory location
- ~~can only access individual bytes~~

Indirect with index register:

(%RAX, %RBX)

↑ Index Register

Memory address is the sum of the contents of the two registers.

Indirect with index register and fixed displacement:

Imm (%RBX, %RAX)

↑ Index Reg.

Memory address is sum of Imm contents of both registers

ex) -8 (%RBX, %RAX)

*	Intel
Byte	1 byte
Word	2 "
Long	4 "
Quad	8 "

Scaled Indirect

(, %RAX, 4)

↑ scale must be 1, 2, 4, or 8

memory address is contents of register times scale value

Scaled Indirect with fixed displacement:

Imm(, %RAX, 2)
↑ scale

Memory address is sum of Imm plus product of register and scale

Scaled Indirect with Index Register:

(%RAX, %RBX, 8)
↑ scale
other register ↑ ↑ Index register

Memory address is product of index register and scale added to other register

Scaled Indirect with Index register and fixed displacement:

Imm(%RBP, %RAX, 4)
↑ scale
other register ↑ ↑ Index register

Memory address is sum of Imm plus other register plus the product of index register and scale

07/12

Push and Pop Instruction:

W = word 16 bits

pushw	Reg16	%RSP -= 2;	%Reg16 → (%RSP)
pushw	Mem16	%RSP -= 2;	Mem16 → (%RSP)
pushW	Imm16	%RSP -= 2;	Imm16 → (%RSP)

L = Long 32 bits

pushL	Reg32	%RSP -= 4;	%Reg32 → (%RSP)
pushL	Mem32	%RSP -= 4;	Mem32 → (%RSP)
PushL	Imm32	%RSP -= 4;	Imm32 → (%RSP)

Q = Quadword 64 bits

PushQ	Reg64	%RSP -= 8;	%Reg64 → (%RSP)
PushQ	Mem64	%RSP -= 8;	Mem64 → (%RSP)
PushQ	Imm64	%RSP -= 8;	Imm64 → (%RSP)