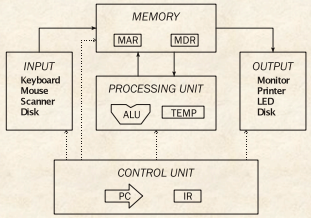
UNIVERSITY of WISCONSIN-MADISON
Computer Sciences Department

CS 202
Introduction to Computation

Professor Andrea Arpaci-Dusseau
Fall 2010

# Lecture 24: How does a computer… execute instructions?


## Finalizing Project 1

Sign up for 20-minute demo with one of the TAs

- Doodle poll available from course web site
- Will not be graded without a demo!

Winners of Demo Day: High Scores and/or Many

- Boomgarden, Cade
- Borkhus, Daniel
- Fox, Michael
- Francour, Adam
- Vang, Ser
- Brey, Lisa
- Borden, Francis
- Evans, Ben
- Kelly, Megan

Claim your prize after lecture!

## Today's Topic: Performing Computation

Given knowledge of topics so far:

Build a sequential circuit for every needed computation

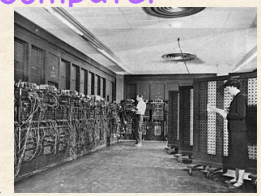
- Design new hardware for specific purpose
- Extremely impractical!
  - Don't have access to manufacturing processes
  - Long time to design and produce
  - Expensive

Want general purpose hardware that can execute any program

## History Lesson: Stored Program Computer

1943: ENIAC - Electronic Numerical Integrator and Computer

- US Army's Ballistic Lab
- Eckert and Mauchly: first general electronic computer
- Hard-wired program – weeks of settings of dials + switches



1944: Beginnings of EDVAC

- Program stored in memory

1945: John von Neumann

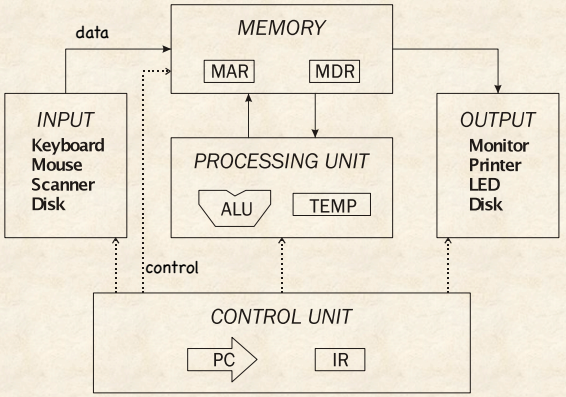
- Wrote report on stored program concept

Basic structure became known as "von Neumann machine" (or model)

## What do we need to execute general program?

1. Place to store instructions and data
   Memory
2. Perform arithmetic and logical operations
   Processing unit
3. Determine next instruction to execute
   Control unit
4. Get data into computer to manipulate
   Input devices
5. Display results to user
   Output devices

## Von Neumann Model

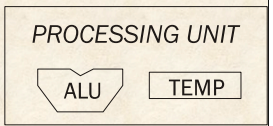



## Processing Unit

Purpose: Manipulate and modify data

1st Component: ALU

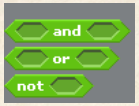
- ALU = Arithmetic and Logic Unit
- Many operations
  - ADD, SUB, AND and NOT
  - Could have many functional units (e.g., multiply, square root)
- Interface: Set one line high (ADD, AND, NOT)



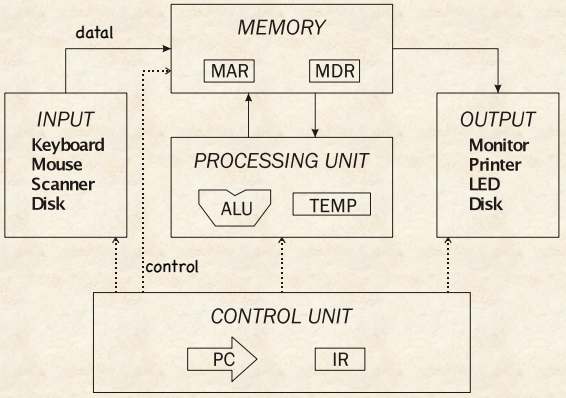


2nd Component: Registers

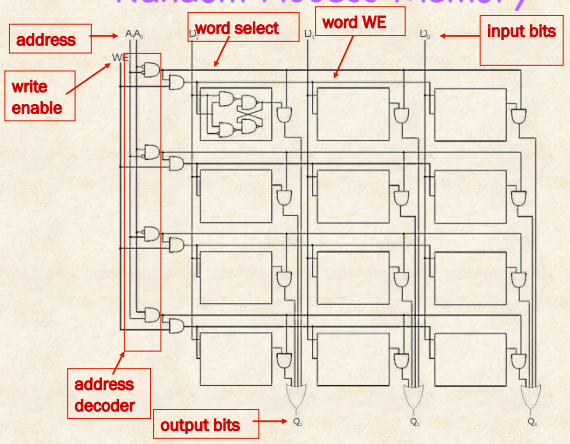
- Small, temporary storage in addition to memory
- Operands and results of functional units
- Interface: Specify register identified with number




## Von Neumann Model



# Low-Level View of Random Access Memory




# Higher-Level View of Random Access Memory

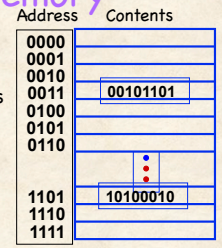
Purpose: Store data and program instructions

Address: Unique ($n$-bit) identifier of $2^n$ locations

Contents: $m$-bit value stored in location

Each variable stored at different address

Each instruction stored at different address

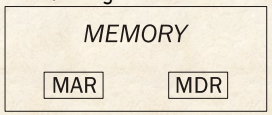



How to move data to/from memory?

Read = Load; Write = Store

Interface defined through two registers + R/W signal

- MAR: Memory Address Register
- MDR: Memory Data Register

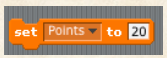



# Reading and Writing Memory


How to read Data (kept at location 0011) ?

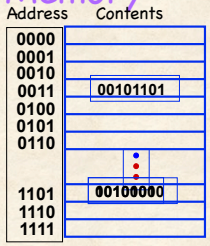
1. Write address 0011 into MAR
2. Send "read" signal to memory
3. Read data from MDR

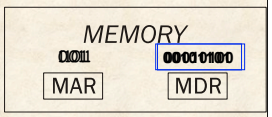
Data is 00101101 = 32+8+4+1 = 45




How to write value 20 to Points (loc 1101)?

1. Write address (1101) into MAR
2. Write data (00010100) to MDR
3. Send "write" signal to memory




# Using Random Access Memory


How to change a value in memory?

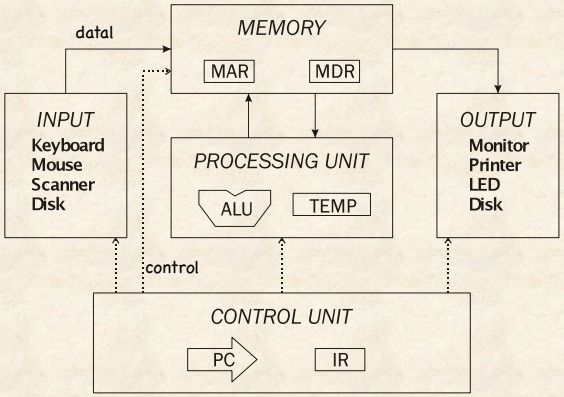
- Cannot change memory directly in many architectures
- Requires computation + memory

Work with processing unit

- Load value of Points from memory into register
- Add 1 to value of Points in register
- Write back new value of Points into memory

Higher-level instructions correspond to many machine instructions

## Von Neumann Model




## Input and Output

INPUT
Keyboard
Mouse
Scanner
Disk

OUTPUT
Monitor
Printer
LED
Disk

Purpose: Moves data in and out of memory to outside world

- Involves separate hardware device

Some devices provide both input and output

- Disk, network: More on these in later lectures!

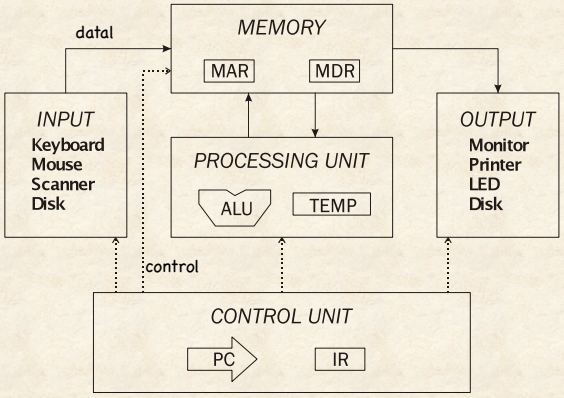
Each device has own interface (set of registers)

- Example: Keyboard: data (KBDR) and status (KBSR) registers

Device driver: (part of operating system)

- Low-level software that controls access to device
- Provides common interface to applications

## Von Neumann Model




## Control Unit

Purpose: Orchestrates execution of program

- Fetches program instructions from memory
- Tells processing unit what operations to perform

How does it know what instruction to fetch?

- Program Counter (PC) contains address of next instruction

How does it remember instruction to decode?

- Instruction Register (IR) contains current instruction

## What kinds of data must bits represent?

Logical: True, False
- Straight-forward: Two states
- True: 1, False: 0

Numbers
- Signed, unsigned, integers, floating point, complex, rational, irrational, …

Text
- Characters, words, strings, …

Images
- Pixels, colors, shapes, movies …

Sound

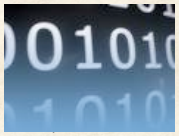
Machine Instructions – Today!!

## Machine Instructions

Definition: Fundamental unit of work
- High-level code is compiled into many low-level machine instructions

Instruction specifies two things
- opcode: operation to be performed
- operands: data/locations to be used for operation

Encoded as sequence of bits (just like data!)
- Usually fixed length
- Control unit interprets instruction: generates sequence of control signals to carry



Instruction Set Architecture (ISA)
- Computer's instructions and formats

## Example 16-bit ISA

Assume: 16 bit instructions (very small!)
- 16 bit words for data (registers and memory)

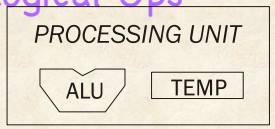
Assume: Each instruction has a four-bit opcode
- Bits [15:12]
  - specification of high-order bits; start with bit 0
- How many different operations can be performed?
  - 4 bits → $2^4$ combinations = 16 operations

Assume: 8 registers in architecture (R0-R7)
- How many bits needed to specify register?
  - 3 bits -> $2^3$ combinations = 8 registers

Explore 3 categories of instructions

## Instruction Type #1: Arithmetic and Logical Ops

Example: ADD instruction




What must ADD instruction specify?
- Data: Operand1, Operand2, Result

Where should data reside?
- In registers (too slow to operate on memory)

How to specify register?
- Each register is numbered; put register number in instruction

## Example: ADD Instruction

Assume: Eight *registers* (R0–R7)

- How many bits are needed to specify each register?
  - 3 bits to specify 8 registers

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADD | | | | Dst | | | Src1 | | | 0 | 0 | 0 | Src2 | | |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

*What is this instruction specifying?*
*"Add the contents of R2 to the contents of R6, and store the result back in R6."*


## Instruction Type #2: Memory

What operations for dealing with memory?
- Load (read) and Store (write)

What must Load instruction specify?
- Memory address and destination register

Don't put mem addr in instruction; why not?
- Not enough bits
  16 bits – 4 bits (opcode) – 3 bits (dest) = 9 bits

How much memory can 9 bits address?
- $2^9$ = 512 words is tiny!!

Put address in register: how much memory can 16-bit (toy example) register reach?
- $2^{16}$ = 64 K items
- Registers much larger in modern systems!

## Example: LDR Instruction

Load: Read data from memory into dest register

Base + offset mode:
- Memory address = Add offset to contents of base reg
- Load from memory address into dest reg

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| LDR | | | | Dst | | | Base | | | Offset | | | | | |

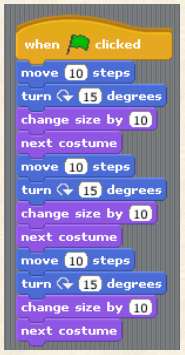
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |

*What is this instruction specifying?*
*"Add the value 6 to the contents of R3 to form a memory address. Load the contents stored in that address to R2."*
*Why base+offset useful?*
*Variables that are not lists?*



## Instruction Type #3

Where does control unit get next instruction from?
- Default: Fetch next sequential instruction




How does Control Unit fetch sequentially?
Set Program Counter: PC = PC + 1
Set MAR = PC
Read from memory
Set Instruction Reg: IR = MDR

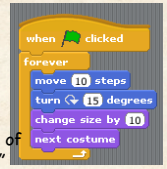
## Instruction Type #3: Control

Why does program sometimes want to execute different non-sequential instruction?

- Program specifies control structure
- Examples: forever loop, if-then, receive message

Type 3: Instructions that change contents of PC

- Jumps: unconditional (always change PC)
- Branches: conditional (change PC only if some condition is true)




Jmp to location of "move 10 steps"

If Points != 1 branch to "play sound"


## Example: JMP Instruction

Set PC to value contained in register

- Address of next instruction to fetch

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| JMP | | | | 0 | 0 | 0 | Base | | | 0 | 0 | 0 | 0 | 0 | 0 |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

*What is this instruction specifying?*
*"Load the contents of R3 into the PC."*


## Today's Summary

Stored program computers

- Instructions look just like data -- it's all interpretation
- Three basic kinds of instructions:
  - computational instructions (ADD, AND, …)
  - data movement instructions (LD, ST, …)
  - control instructions (JMP, Branch, …)

Reading

- Pages 188-232 (Chapter 5) of Invitation to CS

Announcements

- Sign up with Doodle poll to demo Project 1 with TA
- Homework 6 available: Due Friday, Nov 5th
  - Lists in Scratch