Announcements

Final Project : Card Game

- Due December 12 – In-class Demos
- Advice: focus on lists before interface; two-player version 1st

Intermediate Deadlines

- Wed (11/30): Find project partner – PAST
- Fri (12/2): Project proposal
  - At least 1 sentence email to cs202-tas@cs.wisc.edu (cc partner)
- Wed (12/7): Project draft to Learn@UW dropbox
  - Whatever you have completed

TA Lab Hours Today in 1370: 11:00 – 1:00

- Strongly recommend working in lab!

More Announcements

Extra Credit: Fill out survey for College Board

- Submit screenshot to Dropbox HW11-Extra Credit

Service-learning course (Bio375)

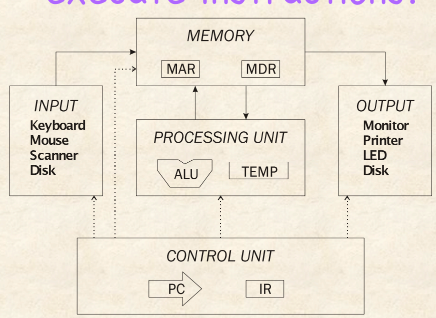
- Lead afterschool clubs for 4 – 8 graders about Scratch
- Full enrollment of 15 students, but contact me if interested

UNIVERSITY of WISCONSIN-MADISON
Computer Sciences Department

CS 202: Introduction to Computation

Professor Andrea Arpaci-Dusseau

How does a computer… execute instructions?



Today's Topic: Performing Computation

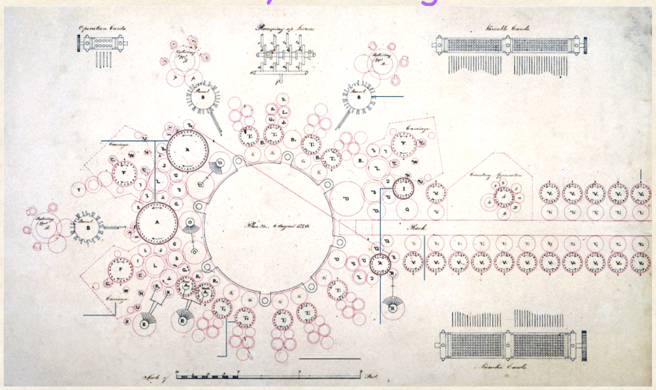
Given knowledge of topics so far:

Build a sequential circuit for every needed computation (e.g., playing tic-tac-toe)

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

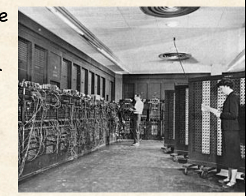
Want general purpose hardware that can execute any program written in software

History of Computation: Analytical Engine



Charles Babbage (1830s): Father of computing
Ada Lovelace: Saw wider potential: first programmer; machine can only do as instructed

Stored Program Computer

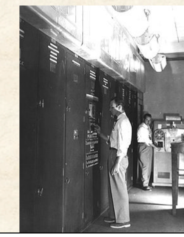


1930s: Alan Turing

- Universal (Turing) Machine: provably capable of computing anything that is computable

1943: ENIAC – Electronic Numerical Integrator and Computer

- US Army's Ballistic Lab
- Eckert/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

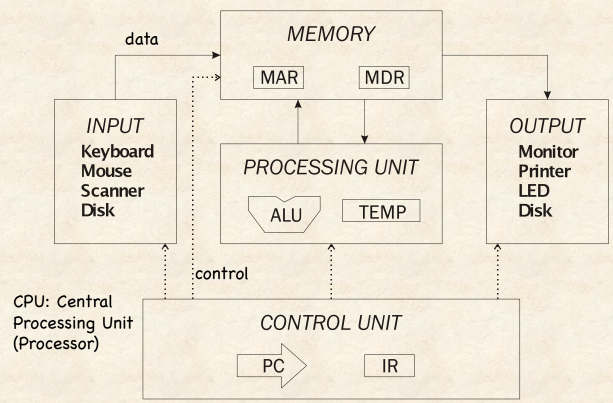
- Wrote report on stored program concept

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

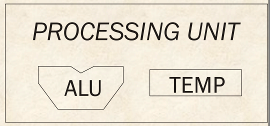
What do we need to Execute General Program?

1. Place to store instructions and data
   Memory
2. Do work – perform mathematical/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



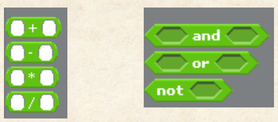
Processing Unit



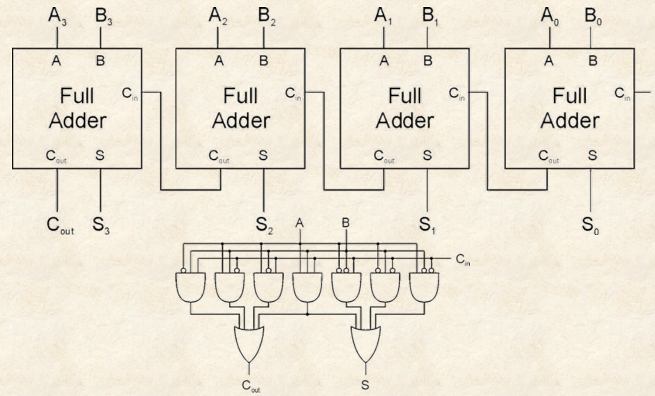
Purpose: Manipulate and modify data

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)

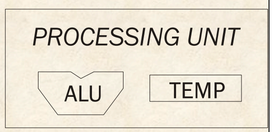


Review: Four-bit Adder



Where do A and B inputs come from? Where does output S go?

Processing Unit



Purpose: Manipulate and modify data

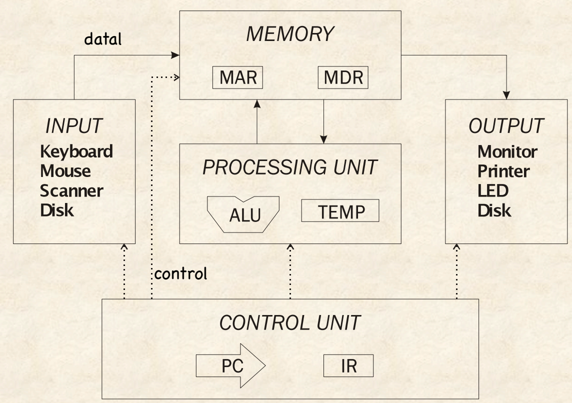
1st Component: ALU

- 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)

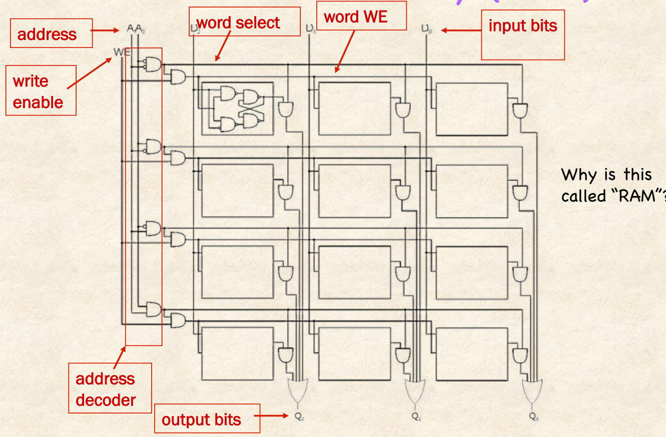
2nd Component: Registers

- Small, temporary, fast storage in addition to memory
- Holds operands and results of functional units
- Not exposed to high-level programmer
- Results will be moved to/from registers and memory

Von Neumann Model

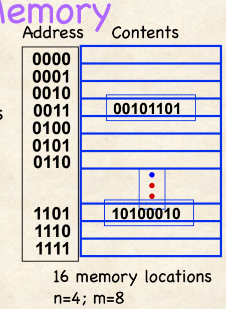


Low-Level View of Random Access Memory (RAM)



Why is this called "RAM"?

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

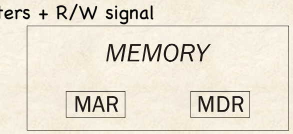
Each instruction stored at different address

How to move data to/from memory?

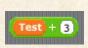
Read = Load; Write = Store

Interface defined through two registers + R/W signal

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

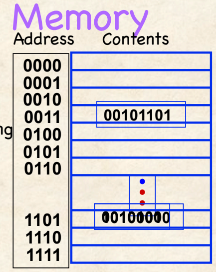


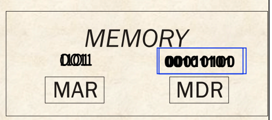
Reading and Writing Memory

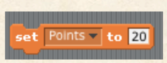


How to read "Test" (kept at location 0011) ?

1. Programming environment does work of mapping human variable names to memory addresses
2. Write address 0011 into MAR
3. Send "read" signal to memory
4. Read data from MDR
   Test is 00101101 = 32+8+4+1 = 45
5. Use ALU to add 45 and 3







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?

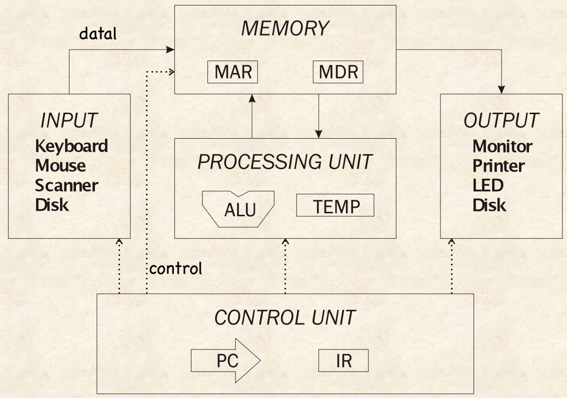
- Cannot change memory directly in many architectures
- Requires both computation + memory

Work with processing unit

- Load value of Points from memory into register
- Use ALU to 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

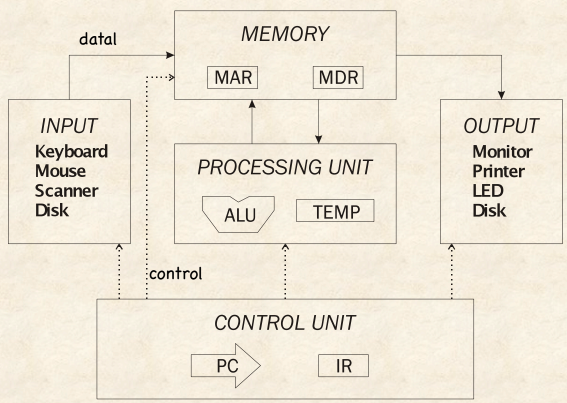
Purpose: Moves data in and out of memory to outside world
- Involves separate hardware device


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

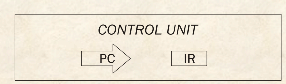


Control Unit

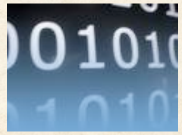
Purpose: Orchestrates execution of program
- Fetches program instructions from memory
- Tells processing unit what operations to perform

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

- True: 1, False: 0

Numbers

- Signed and unsigned integers, floating point

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 everything else!)

Instruction Set Architecture (ISA)

- Exact encoding of computer's instructions and formats

Example 16-bit ISA

Assume: 16 bit instructions (very small!)

Assume: Each instruction has a four-bit opcode

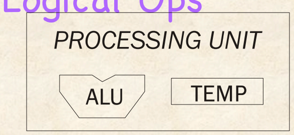
- Bits [15:12]
  - specification of high-order bits; start with bit 0
- How many different operations can be performed?
  - 4 bits → 2^4 combinations = 16 operations

Assume: 8 registers in architecture (R0-R7)

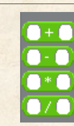
- How many bits needed to specify register?
  - 3 bits -> 2^3 combinations = 8 registers

Explore 2 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)

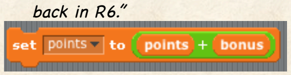
- How many bits are needed to specify each register?
  - 3 bits to specify 8 registers

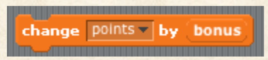
4 bit opcode

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."



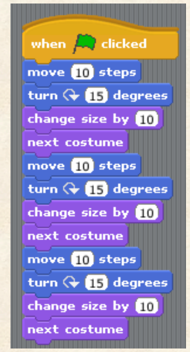


Programming environment manages this mapping between variables and registers

Instruction Type #2

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 #2: Control

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

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

Type 2: 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"

Change PC to different location

If Points != 1 branch to "play sound"

Change PC only in some cases

Example: JMP Instruction

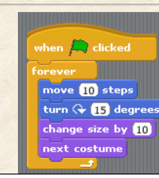
Set PC to value contained in register

- Address of next instruction to fetch

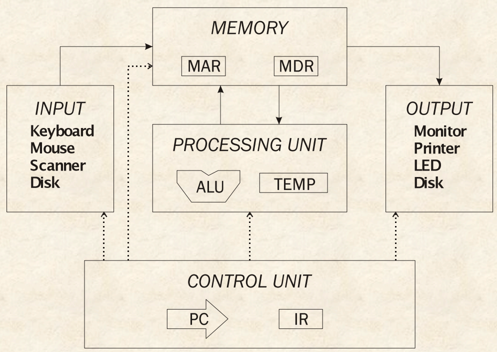
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JMP				0	0	0	Base			0	0	0	0	0	0

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."



Von Neumann Model



What is the flow of this add instruction?

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
Address: 0xaaaa	0	0	0	1	1	1	0	0	1	0	0	0	0	1	1	0

Announcements

Hardware – Stored Program Computer Architecture

- Instructions are just bits -- it's all interpretation

Next: Software – Operating Systems

Intermediate Deadlines

- Wed (11/30): Find project partner – PAST
- Fri (12/2): Project proposal
  - At least 1 sentence email to cs202-tas@cs.wisc.edu (cc partner)
- Wed (12/7): Project draft to Learn@UW dropbox

TA Lab Hours Today in 1370: 11:00 – 1:00

- Strongly recommend working in lab!

Extra Credit: Fill out survey for College Board

- Submit screenshot to Dropbox HW11-Extra Credit