The CPU & Computer Architecture

CSC 103
September 22, 2005

Overview for Today

- Paper topics (sheet)
  - No AI – class discussion
  - Outline and references in 2 weeks
- The CPU – central processing unit
  - Fetch-execute cycle
  - The Pippin simulator
- Hierarchy of Languages
- Program Control
  - Making decisions with ‘if-then’ statements

Chapter 3 Objectives

- To see how a mechanical/electrical device can automatically execute any list of instructions, written in binary
- To see how a simple machine that stores and executes programs (Pippin)
  - All computers do is execute instructions
  - Organized into programs: a precise list of simple instructions

Plexiglas Computer

How Do Computers Work?

- Thoughts so far?

- What are the main components of a computer?
- How do the pieces work together?
- How do the concepts (hardware and software) work together?
- Open questions?
- Great new ideas?

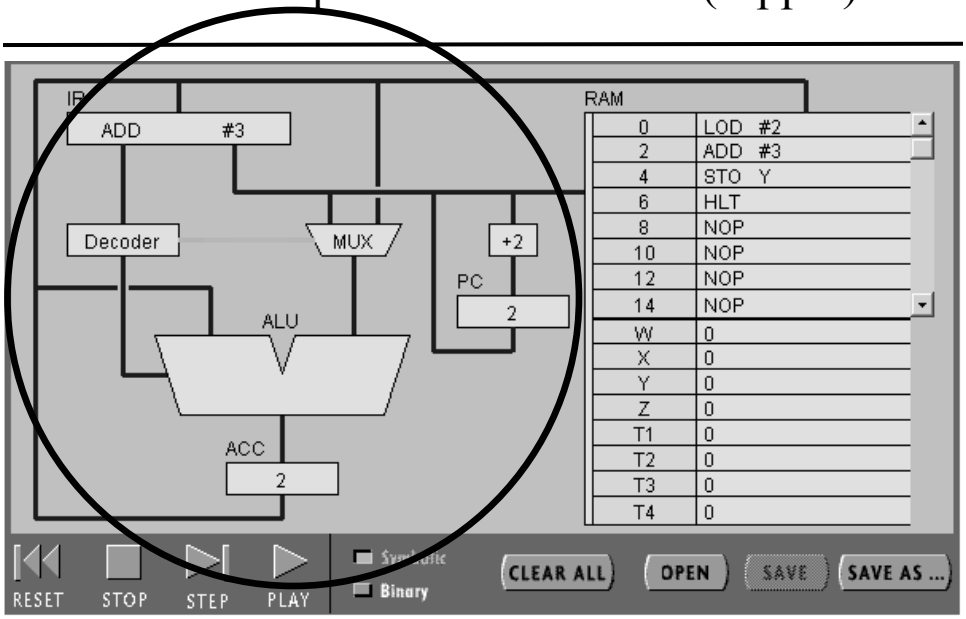
Class To Date: Hardware

- Logic circuits without feedback
  - Addition: Logic gates and Boolean algebra tools
  - Subtraction: A + (-B) ...
- Logic circuits with feedback
  - Memory circuits and RAM array
  - Data bus *and* control lines

Today: Moving from Hardware to Software

- The CPU & main memory
  - Hardware components ⇨ circuits
  - Designed and organized to execute (binary) instructions
    - Conceptually like a row of dominoes
- CPU is automatically given the first instruction when the computer 'boots up'
  - A clock circuit keeps everything moving on
  - Told where the first domino is

Components of the CPU (Pippin)

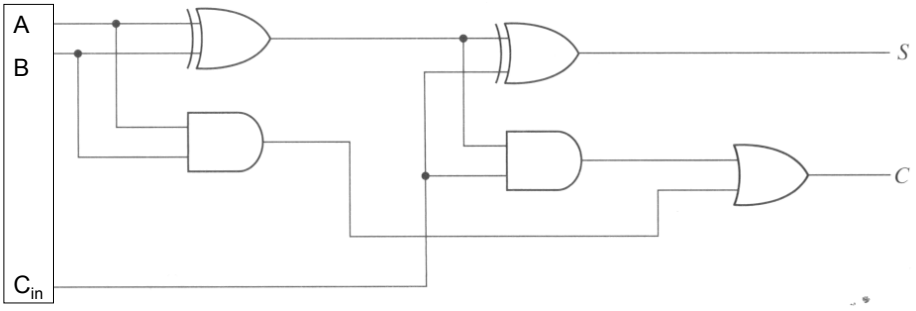


CPU Examples

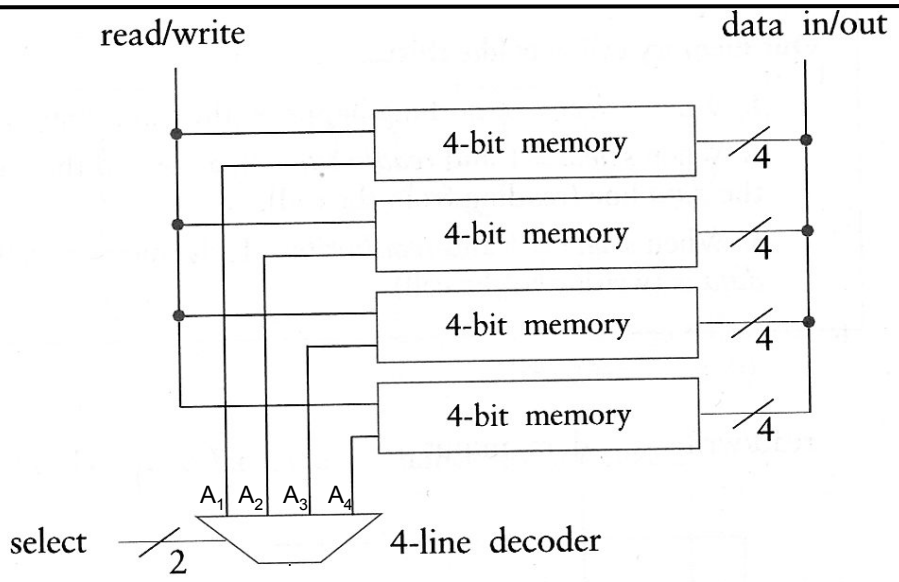
- Sample CPU in text is composed of
  - Clock
  - ALU
  - 8 registers – memory, temporary staging areas
  - Main memory
  - Control circuit
- PIPPIN has similar components / circuits
- *Read the text chapter! (chapter 3)*

The Arithmetic-Logic-Unit

- Inside the CPU is the Arithmetic-Logic-Unit (ALU), which performs addition, subtraction...



Memory: RAM & Registers

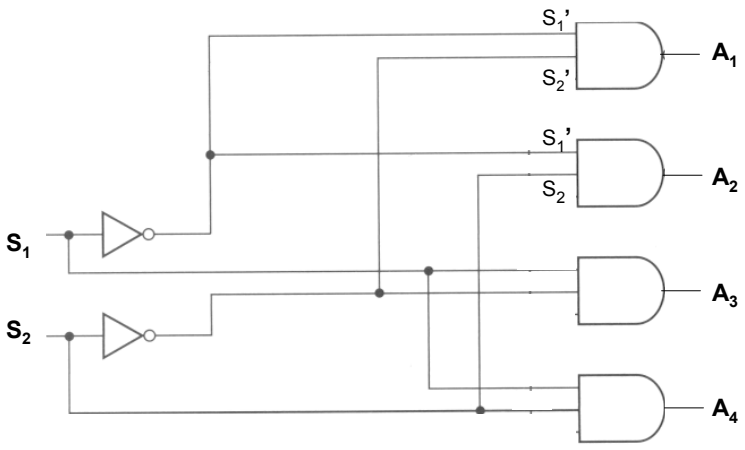


Control Circuit: Decoder

- RAM Address; Decode instruction
- Truth Table of a 2-to-4-Line Decoder

Inputs		Outputs			
S_1	S_2	A_0	A_1	A_2	A_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Control Circuit: Decoder



The CPU & 'Fetch-Execute'

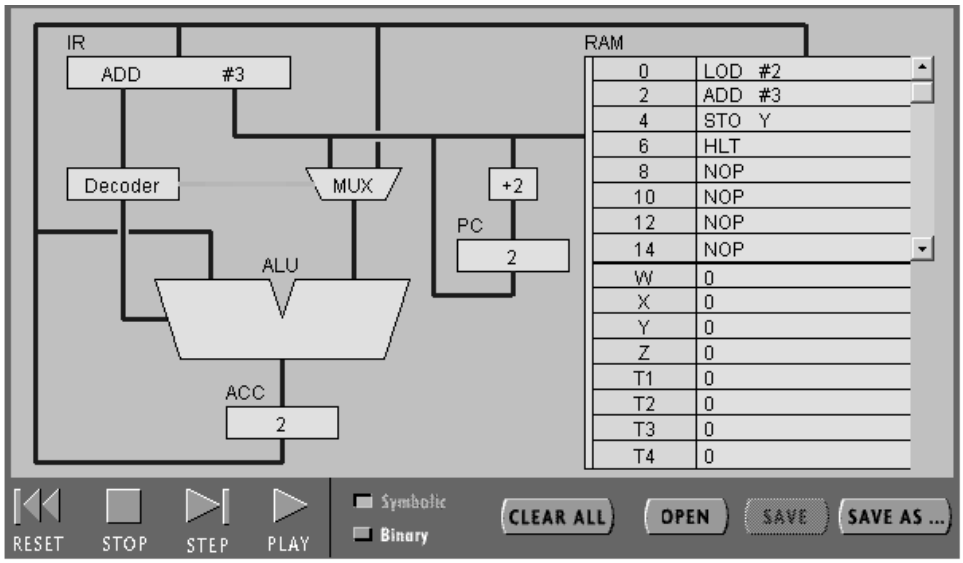
Addition Example

To perform 2 + 3 = 5 :

1. Load ‘2’ into the CPU
2. Add ‘3’ to ‘2’ and temporarily store the result, ‘5’
3. (=) Store the result to main memory

‘Fetch’ each instruction and data, and then do what it directs (‘execute’)

Components of Pippin



Components of Pippin

- Pippin handout
  - ALU
  - Registers
    - PC, the program counter
    - Instruction register
    - Accumulator (the computer's scratch pad)
  - Decoder
  - MUX
  - RAM

The Pippin Simulator

http://www.science.smith.edu/~jcardell/Courses/CSC103/CPUsim/cpusim.html

Addition Example Con't

- Keeping everything organized
  - The CPU must know where to find everything
    - Data and instructions
  - The CPU must know where to store the result
- Performing the example in *binary*
  - The operations (*e.g.,* 'ADD') need a binary code/number assigned to them
  - The operands (data) must be in binary

Assembly Language

- Our example must be written in simple steps
  - LOD #2 = 0001 0100 0000 0010
  - ADD #3 = 0001 0000 0000 0011
  - STO Y = 0000 0101 1000 0010
  - HLT = 0000 1111 0000 0000
- The 'operands'
  - Immediate mode, data follows: #2, #3
  - Direct addressing, data in RAM: Y

Binary Code Assignments for PIPPIN

- Load = LOD = 0001 0100 (data follows)
  = 0000 0100 (data in RAM)
- Add = ADD = 0001 0000 (data follows)
  = 0000 0000 (data in RAM)
- Store = STO = 0000 0101 (location follows)
- Halt = HLT = 0000 1111 (no data)

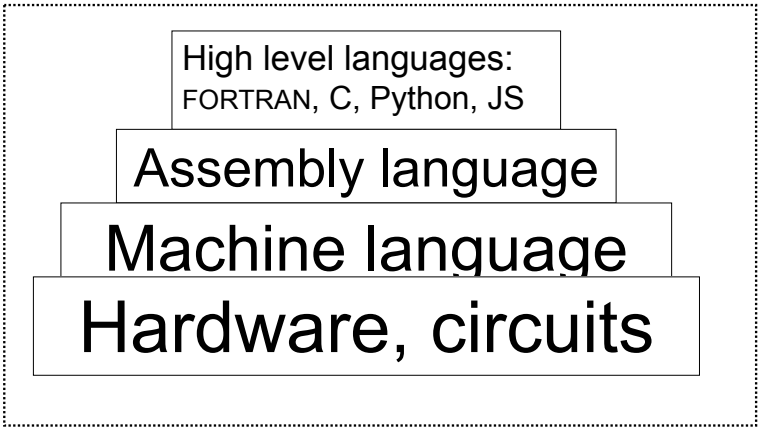
- Punch cards/Binary ⇨ Assembly ⇨ High level language
- See the handout and webpage link for full PIPPIN 'instruction set'

Pippin Lab

Software and the Hierarchy of Languages

Hierarchy of Languages

- The process for people (natural languages) to communicate with computers:



Computer Programs

- A computer computes: it manipulates symbols
  - The symbols are always binary data
- We must tell the computer everything
  - Where the data is
  - What the data is (instructions vs. operand)
  - What to do with the data
- Two tasks
  - Define the purpose of each instruction – ADD, etc.
  - Define a sequence of steps that will execute the instruction
- (We will then need a complex digital circuit to carry out the steps – the CPU – and a clock)

Computer Programs

- Sequential – start at the beginning and methodically execute instructions until done
  - Our examples so far are sequential

or

- Repeat sections; jump over parts... ⇨ ***Program control***

Program Control

- *Decisions*: If-else
- *Repeat*: Loops

- On-line shopping
  - Repeat: "Continue shopping?"
  - Decision: "Or proceed to checkout?"
- Setting preferences
  - Left- or right-handed mouse

Preview of JavaScript and Decisions

- Entering personal information to shop on-line
  - Name
  - Address
    - Zipcode ← Our next task
- Simple verification of information entered
  - Is the zipcode entered a number?
  - Is it 5 digits long?

Instruction Categories

- Three categories of instructions
  - Data flow – load and store
  - Arithmetic-logic – math, logic including compare
  - Control – jump, halt, nop
- New instructions
  - JMP n – go to instruction number ***n***
  - JMZ n – If Acc=0, goto instruction ***n***, else to go instruction immediately following
  - CPZ X – (compare zero) If X=0, set Acc to 1; else set Acc to 0
  - CPL X – (compare less) If X<0, set Acc to 1; else set Acc to 0

If-Else in Assembly

- Let 'X' represent the result from pulling the handle of a slot machine
  - X = 0 means you did not get 4-of-kind...
  - X ≠ 0 means you got a winning hand

```
if (X = 0)
        W = 0       // your winnings=0
else
        W = 100   // you win!!!
```

If-Else: If (X = 0)

```
0   LOD X     Useful to write out below:
2   CPZ X         if X=0, Acc=1, else Acc=0
4   JMZ ?         if Acc=0, goto 12
```

Note: Line numbers – we need to keep track of them to know what line to jump to with the 'JMZ' instruction

If-Else: Then W = 0

```
6   LOD #0
8   STO W
10  HLT
```

Don't forget to 'HLT' at the end of this branch (we do not want to execute both branches, only one)

If-Else: Else W = 100

```
12    LOD #100
14    STO W
16    HLT
```

Note: The ‘then’ *branch went to line* 10, *so we start the* ‘else’ *branch at line 12* ⇨ *This is the line we* jump (JMZ) *to*

Don’t forget to ‘HLT’

Complete If-Else Program

```
0   LOD X      Useful to write out below:
2   CPZ X          if X=0, Acc=1, else Acc=0
4   JMZ 12         if Acc=0, goto 12
6   LOD #0
8   STO W
10  HLT
12  LOD #100
14  STO W
16  HLT
```

The Pippin Simulator

http://www.science.smith.edu/~jcardell/Courses/CSC103/CPUsim/cpusim.html

Things to Remember

- The role of the accumulator
  - The result of the compare is stored in the accumulator
  - The jump occurs based on the accumulator
- Line numbers
  - Write out program FIRST to work out line numbers
- HLT
  - Do not forget the 'HLT' after the '*if*' branch and after the '*else'* branch

Summary

- The CPU
  - The fetch-execute cycle
  - The Pippin CPU simulator
- Programming Control
  - If-then statements
    - Using ‘compare’ and ‘jump’ for computer decision making