Hierarchy of Languages

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

- High level languages: FORTRAN, C, Python, JS
- Assembly language
- Machine language
- Hardware, circuits
Overview for Today

• The CPU – central processing unit
  – Elements of the CPU
  – Fetch-execute cycle

• Decoder circuit
  – Interpreting instruction codes (opcode)

• Assembly Language
  – Types of instructions
  – Instruction structure
    • Opcode and operand

Assembly Language

• To perform 2 + 3 = 5:
  – LOD #2 = load #2 into the CPU
  – ADD #3 = add #3 to whatever is there
  – STO Y = store the result to ‘Y’
  – HLT = stop!

• Fetch and then execute
Components of the CPU (Pippin)

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

- How are these instructions decoded by the CPU?
Understanding How Opcodes Work

Decoder Circuits

Memory: RAM & Registers

![Diagram of memory and decoder circuits]
Control Circuit: Decoder

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

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₁ S₂</td>
<td>A₁ A₂ A₃ A₄</td>
</tr>
<tr>
<td>0 0</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>1 0</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>1 1</td>
<td>0 0 0 1</td>
</tr>
</tbody>
</table>

![Diagram]
Control Circuit: Decoder

- To decode Pippin instructions

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Instruction Line High</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1, C_2, C_3, C_4$</td>
<td>$I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8, I_9...$</td>
</tr>
</tbody>
</table>
- `ADD`: 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
- `SUB`: 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0
- `MUL`: 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
- `DIV`: 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0
- `LOD`: 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
- `STO`: 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0

http://maven.smith.edu/~jcardell/courses/CSC103/PIPPINGuide.html

The CPU (Pippin)
What a Computer Does:

The CPU &

‘Fetch-Execute’
Assembly Language

• To perform $2 + 3 = 5$:
  (…fetch and then execute)
  – 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’
Program Control

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
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
  - No beeping sounds? Many sounds?
  - Left- or right-handed mouse
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...
  - \( X \neq 0 \) means you got a winning hand

\[
\text{if (} X = 0 \text{)} \\
\hspace{2cm} W = 0 \quad // \text{your winnings}=0
\]
\[
\text{else} \\
\hspace{2cm} W = 100 \quad // \text{you win}!!!
\]
If-Else: If \((X = 0)\)

<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>LOD X</td>
<td>Useful to write out below:</td>
</tr>
<tr>
<td>2</td>
<td>CPZ X</td>
<td>if (X=0), Acc=1, else Acc=0</td>
</tr>
<tr>
<td>4</td>
<td>JMZ ?</td>
<td>if Acc=0, goto 12</td>
</tr>
</tbody>
</table>

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

If-Else: Then \(W = 0\)

<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>LOD #0</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>STO W</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>HLT</td>
<td></td>
</tr>
</tbody>
</table>

*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/CSCI103/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