Designing a CPU

CPU is just a fancy state machine: step depends on
  • Memory
  • Accumulator
  • Instruction
Simple steps in CPU

Memory
• “File cabinet” to store and retrieve results
• Named by variables or numbers (0-15, A-P)

Accumulator
• “Scratch space” for carrying out operations
• acc for short

Basic Boolean operations

Recap: Instructions for CPU

Manipulate accumulator, memory using logic

Example:
   acc = A
   acc = acc or B
   E = acc

sets E to A or B.
Instructions, again

What equation describes the final result stored in variable H after this program runs:

acc = not A
H = acc
acc = not B
H = acc or H

A. not A
B. not B
C. A or B
D. not A or not B
E. true

Instructions, again

What equation describes the final result stored in variable K after this program runs:

acc = A
acc = acc and B
J = acc
acc = acc and C
K = acc
acc = not J
acc = acc and D
K = acc or K

A. D and not (A and B)
B. (C or D) and (A and B)
C. (C and A and B) or (D and not (A and B))
D. D and not (C and A and B)
E. false
### From last time

Our language has only 128 possible instructions

<table>
<thead>
<tr>
<th>acc=A</th>
<th>acc=B</th>
<th>acc=C</th>
</tr>
</thead>
<tbody>
<tr>
<td>acc=not A</td>
<td>acc=not B</td>
<td>acc=not C</td>
</tr>
<tr>
<td>acc=acc and A</td>
<td>acc=acc and B</td>
<td>acc=acc and C</td>
</tr>
<tr>
<td>acc=acc or A</td>
<td>acc=acc or B</td>
<td>acc=acc or C</td>
</tr>
<tr>
<td>A=acc</td>
<td>B=acc</td>
<td>C=acc</td>
</tr>
<tr>
<td>A=not acc</td>
<td>B=not acc</td>
<td>C=not acc</td>
</tr>
<tr>
<td>A=acc and A</td>
<td>B=acc and B</td>
<td>C=acc and C</td>
</tr>
<tr>
<td>A=acc or A</td>
<td>B=acc or B</td>
<td>C=acc or C</td>
</tr>
</tbody>
</table>

### We’ll give each instruction a number

Number = Pattern!

We’ll describe instruction in terms of pattern
- address (variable from memory)
- op (logical operation to perform)
- control (do we store result in variable or acc)
We’ll give each instruction a number

Number = Pattern!

Pattern controls
• circuit that retrieves bits from memory
• logic unit that carries out operations
• circuit that stores result where needed

Instructions as patterns

Write each step in 7 bits
What variable to use – four bits: 0000 to 1111
What operation to do – three bits
000: acc = acc or V
001: acc = acc and V
010: acc = V
011: acc = not V
100: V = acc or V
101: V = acc and V
110: V = acc
111: V = not acc
Carrying out instruction: overview

Logic of memory

Memory retrieval circuit has input
- contents of memory (e.g., 16 bits for variables) and address from instruction (value from 0-15)

Outputs bit stored at that address
Memory Retrieval

Summary of part of truth table:

<table>
<thead>
<tr>
<th>Address</th>
<th>Store</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>a------</td>
<td>a</td>
</tr>
<tr>
<td>0001</td>
<td>- b-----</td>
<td>b</td>
</tr>
<tr>
<td>0010</td>
<td>- c-----</td>
<td>c</td>
</tr>
<tr>
<td>0011</td>
<td>- - d----</td>
<td>d</td>
</tr>
<tr>
<td>0100</td>
<td>- - - e--</td>
<td>e</td>
</tr>
</tbody>
</table>

Accumulator

Special memory bit, always available.
Logical operations

Logic unit has three inputs:
• Bit M retrieved from memory
• Bit A retrieved from accumulator
• Three bits from instruction for a logical operation

Output:
• Bit computed by operation from M and A

Summary of part of truth table

<table>
<thead>
<tr>
<th>OP</th>
<th>M</th>
<th>A</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>m</td>
<td>a</td>
<td>m or a</td>
</tr>
<tr>
<td>001</td>
<td>m</td>
<td>a</td>
<td>m and a</td>
</tr>
<tr>
<td>010</td>
<td>m</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>011</td>
<td>m</td>
<td>a</td>
<td>not a</td>
</tr>
<tr>
<td>100</td>
<td>m</td>
<td>a</td>
<td>m or a</td>
</tr>
<tr>
<td>101</td>
<td>m</td>
<td>a</td>
<td>m and a</td>
</tr>
<tr>
<td>110</td>
<td>m</td>
<td>a</td>
<td>m</td>
</tr>
<tr>
<td>111</td>
<td>m</td>
<td>a</td>
<td>not m</td>
</tr>
</tbody>
</table>
Storing result

Memory storage circuit has input
- contents of memory (e.g., 16 bits for variables)
  address (value from 0-15), new value (bit), and
  control bit (from instruction)

Output is new memory
- with contents at address changed to new value,
  if control bit is true

If control bit is false, update accumulator

Carrying out instruction: overview
Keeping track of instructions

Store instructions in new memory
Keep a “program counter” of current instruction
In each step:
  • Retrieve current instruction
  • Execute current instruction
  • Increment program counter

Visualization

```
  PC --> Program
    |      |        |
    v      v        v
Add 1 --> Retrieve [Instruction]
```
**Summary: von Neumann Architecture**

A computer is just a big state machine
- Input: registers, memory, input devices
- Output: new values for registers, memory, output devices
- PC: address of next step to execute

**Power of CPU**

If we build one machine that takes
- truth values for variables
- sequence of simple steps
- and carries out those steps
This machine can compute any function!
We can *program* the machine to do what we want!