**Recap from last time**

- What is -24 as an 8-bit binary number?
  - A. 10011000
  - B. 10011001
  - C. 11100111
  - D. 11100110
  - E. 11101000

- Powers of two:
  - 128
  - 64
  - 32
  - 16
  - 8
  - 4
  - 2
  - 1
Recap from last time

24 in 8-bit binary:
00011000

Swap 0s and 1s:
11100111

Add one:
11101000

Recap from last time

What’s the product of these two binary numbers?

\[
\begin{array}{c}
1110 \\
\times \\
101
\end{array}
\]

A. 1 0 1 0 1 0
B. 1 1 0 1 1 0
C. 1 1 1 1 1 0
D. 1 0 0 0 1 1 0
E. 1 1 1 1 1 1 0
Recap from last time

What’s the product of these two binary numbers?

\[
\begin{array}{cccc}
1 & 1 & 1 & 0 \\
\times & 1 & 0 & 1 \\
\hline
1 & 1 & 1 & 0 \\
1 & 1 & 1 & 0 \\
\hline
1 & 0 & 0 & 0 & 1 & 1 & 0 \\
\end{array}
\]

Recap from last time

Counter as computer

- State: Number = Pattern
- Step: Number → Number or Pattern → Pattern
- Write step down as truth table
- Build a circuit
Universality

Can build any truth table with gates
  • but machine follows fixed step once and for all

Can describe truth table as program
  • same machine runs any step logic

Intuition

Convert truth table to “program”
  • A step-by-step procedure
  • Using simple operations
  • Using limited memory
  • That says whether formula is true
Making steps simple

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

Example steps

Load value into scratch space:
acc = B

Perform one Boolean operation:
acc = acc and C

Store result for later use:
E = acc
Recall counter logic

Three old bits: N

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Recall counter logic

F = not C
E = (B and not C) or (not B and C)
D = (A and not (B and C)) or (not A and (B and C))
"Programs"

F = not C
acc = not C
F = acc

E = (B and not C) or (not B and C)
acc = not C
acc = acc and B
G = acc
acc = not B
acc = acc and C
acc = acc or G
E = acc
“Programs”

\[ D = (A \text{ and } \neg (B \text{ and } C)) \text{ or } (\neg A \text{ and } (B \text{ and } C)) \]

\[ \text{acc} = B \]
\[ \text{acc} = \text{acc} \text{ and } C \]
\[ G = \text{acc} \]
\[ \text{acc} = \neg G \]
\[ \text{acc} = \text{acc} \text{ and } A \]
\[ H = \text{acc} \]
\[ \text{acc} = \neg A \]
\[ \text{acc} = \text{acc} \text{ and } G \]
\[ \text{acc} = \text{acc} \text{ or } H \]
\[ D = \text{acc} \]

Now map state and instruction to new state

Ingredients:
- Memory
- Accumulator
- Logical operations
- Program counter
Carrying out instruction: summary

How do we do it?

Same way as always
• Just say precisely what steps are
• How to follow them
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

Examples

acc = J
  0101001 (010: acc = V; 1001: J)
acc = acc and B
  0010001 (001: acc = acc and V; 0001: B)
acc = acc and L
  0011011 (001: acc = acc and V; 1011: L)
F = acc
  1100101 (110: V = acc; 0101: F)