20. Central Processing Unit
20. CPU

- Overview
- Bits, registers, and memory
- Program counter
- Components and connections
Let's build a computer!

CPU = Central Processing Unit

**Computer**
- Display
- Touchpad
- Battery
- Keyboard
- ...
- CPU (difference between a TV set and a computer)

**Previous lecture**
- Combinational circuits
- ALU (calculator)

**This lecture**
- Sequential circuits with *memory*
- CPU (computer)
A smaller computing machine: TOY-8

TOY instruction-set architecture.
• 256 16-bit words of memory.
• 16 16-bit registers.
• 1 8-bit program counter.
• 2 instruction types.
• 16 instructions.

TOY-8 instruction-set architecture.
• 16 8-bit words of memory.
• 1 8-bit register.
• 1 4-bit program counter.
• 1 instruction type.
• 8 instructions.

Purpose of TOY-8. Illustrate CPU circuit design for a "typical" computer.
### TOY-8 reference card

<table>
<thead>
<tr>
<th>opcode</th>
<th>operation</th>
<th>pseudo-code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>halt</td>
<td>halt</td>
</tr>
<tr>
<td>2</td>
<td>add</td>
<td>(R = R + M[addr])</td>
</tr>
<tr>
<td>4</td>
<td>bitwise and</td>
<td>(R = R &amp; M[addr])</td>
</tr>
<tr>
<td>6</td>
<td>bitwise xor</td>
<td>(R = R ^ M[addr])</td>
</tr>
<tr>
<td>8</td>
<td>load addr</td>
<td>(R = \text{addr})</td>
</tr>
<tr>
<td>A</td>
<td>load</td>
<td>(R = M[\text{addr}])</td>
</tr>
<tr>
<td>C</td>
<td>store</td>
<td>(M[\text{addr}] = R)</td>
</tr>
<tr>
<td>E</td>
<td>branch zero</td>
<td>if ((R == 0)) (PC = \text{addr})</td>
</tr>
</tbody>
</table>

**ZERO**  \(M[0]\) is always 0.

**STANDARD INPUT**  Load from \(M[F]\).

**STANDARD OUTPUT**  Store to \(M[F]\).

A TOY-8 program

<table>
<thead>
<tr>
<th>PC</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0 D</td>
</tr>
</tbody>
</table>

Add two numbers from memory and put sum in memory

\(R = M[5]\)

\(R = R + M[6]\)

\(M[7] = R\)

halt

Challenge for the bored:

Write Fibonacci seq, add numbers on stdin, ...
CPU circuit components for TOY-8

**TOY-8 CPU**
- ALU (adder, AND, XOR)
- Memory
- Register (R)
- PC
- IR
- Control
- Clock

**Goal.** Complete CPU circuit for TOY-8 (same design extends to TOY and to your computer).
Basic design of our circuits

- Organized as *components* (functional units of TOY: ALU, memory, register, PC, and IR).
- Connected by *busses* (groups of wires that propagate information between components).
- Controlled by *control lines* (single wires that control circuit behavior).

Conventions

- Bus inputs are at the top, input connections are at the left.
- Bus outputs are at the bottom, output connections are at the right.
- Control lines are blue.

These conventions *make circuits easy to understand.*

(Like style conventions in coding.)
**Q. Why TOY-8?**

A. TOY circuit width would be about 5 times TOY-8 circuit width.

**Sobering fact.** The circuit for your computer is *hundreds* to *thousands* of times wider.

**Reassuring fact.** Design of all three is based on the same fundamental ideas.
20. CPU

- Overview
- Bits, registers, and memory
- Program counter
- Connections
Sequential circuits

Q. What is a sequential circuit?
A. A digital circuit (all signals are 0 or 1) *with feedback* (loops).

Q. Why sequential circuits?
A. *Memory* (difference between a DFA and a Turing machine).

Basic abstractions
- On and off.
- Wire: Propagates an on/off value.
- Switch: Controls propagation of on/off values through wires.
- Flip-flop: *Remembers* a value (next).
**Simple circuits with feedback**

*Loops in circuits lead to time-varying behavior*
- *Sequence* of switch operation matters.
- Need tight control (see next slide).

**Example 1.** Two switches, each blocked by the other.
- State determined by whichever switches first.
- Stable (once set, state never changes).
- *Basic building block for memory circuits.*

**Example 2.** Three switches, blocked in a cycle.
- State determined by whichever switches first.
- *Not* stable (cycles through states).

a "buzzer"
A new ingredient: Circuits with memory

An *SR flip-flop* controls feedback.
- Add control lines to switches in simple feedback loop.
- R (reset) sets state to 0.
- S (set) sets state to 1.
- Q (state) is always available.

examples

- **R: set to 0**
  - stays 0
  - S: set to 1
  - stays 1
  - unused

Caveat. Timing of switch vs. propagation delay.
Flip-flop application: Memory bit

Add logic to an SR flip-flop for more precise control
- Provide data value on an input wire instead of using S and R controls.
- Use WRITE control wire to enable change in flip-flop value.
- Flip-flop value always available as output.

**set to 1**

WRITE on

1

WRITE off

stays 1

**set to 0**

WRITE on

0

WRITE off

stays 0
Memory bit application 1: fetch/execute clock

Assumptions
- Physical clock provides regular on/off pulses.
- Duration is just enough to trigger a flip-flop.
- Space between pulses is long enough to allow the longest chain of flip-flops in the circuit to stabilize.

Fetch/execute clock. Attach clock to a memory bit.
- RUN control wire starts clock when on.
- HALT control wire stops clock when on.
- Memory bit flips on each clock tick (flip value and feed back to input).
- Result: on/off sequence for FETCH and EXECUTE control wires that control the CPU (stay tuned).
Fetch/execute clock with write pulses

Generates on/off signals for 4 control wires
- \textit{FETCH}.
- \textit{FETCH WRITE} pulse.
- \textit{EXECUTE}.
- \textit{EXECUTE WRITE} pulse.

Implementation
- Add AND gates to fetch/execute clock.
- \textit{FETCH WRITE} = \textit{FETCH AND CLOCK}.
- \textit{EXECUTE WRITE} = \textit{EXECUTE AND CLOCK}.

Application
- Implements CPU fetch/execute cycles.
- Signals turn on control wires that change the state of PC, R, IR, and memory.

Interesting events occur at four distinct times in the cycle
Memory bit application II: Register

Register
• $w$ memory bits.
• $w$-bit input bus.
• values available on output bus.
• input loaded on WRITE pulse.

Implementation
• Connect memory bits to busses.
• Use WRITE pulse for all of them.

Applications for TOY-8
• PC holds 4-bit address.
• IR holds 8-bit instruction.
• R holds 8-bit data value.
Memory bit application III: Memory bank

Memory bank
- $2^n$ words, each $w$ bits.
- $n$-bit address input.
- $w$-bit input bus.
- value of selected word available on output bus.
- input loaded to selected word on WRITE pulse.
Memory bit application III: Memory bank

Memory bank
• $2^n$ words, each $w$ bits.
• $n$-bit address input.
• $w$-bit input bus.
• value of selected word available on output bus.
• input loaded to selected word on WRITE pulse.

Implementation
• Decoder/demux selects word.
• One-hot muxes take selected word to output bus.

Application for TOY-8
• Main memory.
**TOY-8 main memory bank**

**Interface**
- Input bus (for *store*)
- Output bus (for *load*)
- Address to select a word
- *Write* control signal

**Connections**
- Input bus from registers
- Output bus to IR and R
- Address bits from PC and IR

<table>
<thead>
<tr>
<th></th>
<th>words</th>
<th>bits/word</th>
<th>addr bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOY-8</td>
<td>16</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>TOY</td>
<td>256</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>your computer</td>
<td>1 billion</td>
<td>64</td>
<td>32</td>
</tr>
</tbody>
</table>
20. CPU

- Overview
- Bits, registers, and memory
- Program counter
- Components and connections
Steps to design a digital (sequential) circuit
- Design **interface**: input busses, output busses, control signals.
- Determine **components**.
- Determine **connections**.
- Establish **control sequence**.

**Warmup.** Design TOY-8 program counter (PC).

**First challenge.** Need an **incrementer** circuit.

**Second challenge.** Multiple bus connections.
Pop quiz on combinational circuit design

Q. Design a circuit to compute $x + 1$. 
Q. Design a circuit to compute $x + 1$.

A. Start with a bitwise adder
- Delete $y$ inputs, set carry in to 1.
- Compute carry with AND and sum with XOR.

<table>
<thead>
<tr>
<th>$X_i$</th>
<th>$C_i$</th>
<th>$C_{i+1}$</th>
<th>AND</th>
<th>$X_i$</th>
<th>$C_i$</th>
<th>$Z_i$</th>
<th>XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

4-bit incrementer

$$x + 1$$
Multiple bus connections

If component *outputs go to* multiple other components
- No problem, just use T connections.
- Values on both busses are the same.
- Example: Register connects to ALU and memory.

If component *inputs come from* multiple other components
- Problem.
- Values on the busses are *different*.
- Example: ALU, memory, and IR connect to register.
- Solution: Need a *selector switch* (bus mux).
A (virtual) bus selector switch

One-hot \( m \)-way bus mux
- \( m \) input busses.
- \( m \) control lines (selection).
- One output bus.
- At most one selection line is 1.
- Output bus lines have same value as selected input bus lines.

Example: 4-bit 3-way bus mux

Important to note. No direct connection from input to output.
One-hot bus mux

One-hot \( m \)-way bus mux
- \( m \) input busses.
- \( m \) control lines (selection).
- One output bus.
- At most one selection line is 1.
- Output bus lines have same value as selected input bus lines.

Implementation
- Bitwise one-hot muxes for output.

Application (next): Select among inputs to a component.
Program counter (PC)

The PC holds an address and supports 3 control wires:
- **INCREMENT**. Add 1 to value when WRITE becomes 1.
- **LOAD**. Set value from input bus when WRITE becomes 1.
- **WRITE**. Enable PC address to change as specified.

The current address is always available on the output bus.

**Components**
- PC register (4-bit).
- Incrementer (add 1).

**Connections**
- Input bus to PC register.
- Incrementer to PC register.
- PC register to incrementer.
- PC register to output bus.
Program counter (PC)

The PC holds an address and supports 3 control wires:
- **INCREMENT.** Add 1 to value when WRITE becomes 1.
- **LOAD.** Set value from input bus when WRITE becomes 1.
- **WRITE.** Enable PC address to change as specified.

The current address is always available on the output bus.

**Components**
- PC register (4-bit).
- Incrementer (add 1).
- 2-way bus mux.

**Connections**
- Input bus to bus mux.
- Incrementer to bus mux.
- Bus mux to PC register.
- PC register to incrementer.
- PC register to output bus.
Summary of TOY-8 PC circuit

The **PC** supports two control-signal sequences:
- **Load, then write.** Set address from input bus (example: branch instruction).
- **Increment, then write.** Add one to value.

Address is written to the PC register in both cases and always available on the output bus.

Important note: *write* pulse must be very short because of the cycle in this circuit.
20. CPU

- Overview
- Bits, registers, and memory
- Program counter
- Components, connections, and control
TOY-8: Interface

CPU is a circuit inside the machine

Interface to outside world
- Switches and lights
- ON/OFF
- RUN

Connections to outside (omitted)
- ADDR to PC
- DATA to memory bank input bus
- Buttons to control lines that activate memory load/store
Review: CPU circuit components for TOY-8

TOY-8 CPU
- ALU (adder, AND, XOR)
- Memory
- Register (R)
- PC
- IR
- Control
- Clock

Also needed (see next slide): Bus muxes for R and memory addr.
### Connections for TOY-8 CPU

#### Instructions

<table>
<thead>
<tr>
<th>Instructions</th>
<th>Bus Connections</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fetch (all)</strong></td>
<td>PC to memory addr</td>
</tr>
<tr>
<td></td>
<td>memory to IR</td>
</tr>
<tr>
<td><strong>halt</strong></td>
<td>none</td>
</tr>
<tr>
<td><strong>add, and, xor</strong></td>
<td>IR addr to memory addr</td>
</tr>
<tr>
<td></td>
<td>memory to ALU 1</td>
</tr>
<tr>
<td></td>
<td>R to ALU 0</td>
</tr>
<tr>
<td></td>
<td>ALU to R</td>
</tr>
<tr>
<td><strong>load address</strong></td>
<td>IR addr to R</td>
</tr>
<tr>
<td><strong>load</strong></td>
<td>IR addr to memory addr</td>
</tr>
<tr>
<td></td>
<td>memory to R</td>
</tr>
<tr>
<td><strong>store</strong></td>
<td>IR addr to memory addr</td>
</tr>
<tr>
<td></td>
<td>R to memory</td>
</tr>
<tr>
<td><strong>branch if zero</strong></td>
<td>IR addr to PC</td>
</tr>
</tbody>
</table>

#### Diagram

The diagram shows the connections for the TOY-8 CPU, including the ALU, memory, control unit, and registers (R, IR, addr mux, R mux). The connections are indicated by arrows, with some notes for bus muxes needing 2-way or 3-way bus muxes.
Control wires for TOY-8 CPU

<table>
<thead>
<tr>
<th>component</th>
<th>control wires</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOCK</td>
<td>RUN</td>
</tr>
<tr>
<td></td>
<td>HALT</td>
</tr>
<tr>
<td>CONTROL</td>
<td>FETCH</td>
</tr>
<tr>
<td></td>
<td>FETCH WRITE</td>
</tr>
<tr>
<td></td>
<td>EXECUTE</td>
</tr>
<tr>
<td></td>
<td>EXECUTE WRITE</td>
</tr>
<tr>
<td>ALU</td>
<td>ADD</td>
</tr>
<tr>
<td></td>
<td>XOR</td>
</tr>
<tr>
<td></td>
<td>AND</td>
</tr>
<tr>
<td>R mux</td>
<td>R MUX ALU</td>
</tr>
<tr>
<td></td>
<td>R MUX MEM</td>
</tr>
<tr>
<td></td>
<td>R MUX IR</td>
</tr>
<tr>
<td>R</td>
<td>R WRITE</td>
</tr>
<tr>
<td>IR</td>
<td>IR WRITE</td>
</tr>
<tr>
<td>memory</td>
<td>MEMORY WRITE</td>
</tr>
<tr>
<td>PC</td>
<td>PC INCREMENT</td>
</tr>
<tr>
<td></td>
<td>PC LOAD</td>
</tr>
<tr>
<td></td>
<td>PC WRITE</td>
</tr>
<tr>
<td>addr mux</td>
<td>ADDR MUX PC</td>
</tr>
<tr>
<td></td>
<td>ADDR MUX IR</td>
</tr>
</tbody>
</table>
One final combinational circuit: Control

Control. Circuit for control wire sequencing.

Inputs
- Four control wires from clock.
- opcode from IR.
- contents of R.

Outputs
- 15 control wires for CPU components.

Key feature. A simple combinational circuit.
Control wire sequences

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Execute</th>
<th>Execute Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>halt</td>
<td>HALT</td>
<td></td>
</tr>
<tr>
<td>add</td>
<td>ALU ADD</td>
<td>R WRITE</td>
</tr>
<tr>
<td>xor</td>
<td>ALU ADD</td>
<td>R WRITE</td>
</tr>
<tr>
<td>and</td>
<td>ALU ADD</td>
<td>R WRITE</td>
</tr>
<tr>
<td>load address</td>
<td>R MUX IR</td>
<td>R WRITE</td>
</tr>
<tr>
<td>load</td>
<td>R MUX MEMORY</td>
<td>R WRITE</td>
</tr>
<tr>
<td>store</td>
<td>ADDR MUX IR</td>
<td>MEMORY WRITE</td>
</tr>
</tbody>
</table>

* PC LOAD for branch if 0 if R is 0.
Sample program

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A5</td>
</tr>
<tr>
<td>2</td>
<td>26</td>
</tr>
<tr>
<td>3</td>
<td>C7</td>
</tr>
<tr>
<td>4</td>
<td>00</td>
</tr>
<tr>
<td>5</td>
<td>05</td>
</tr>
<tr>
<td>6</td>
<td>08</td>
</tr>
</tbody>
</table>

And THAT... is how your computer works!
Scanning electron microscope image of a real microprocessor

<table>
<thead>
<tr>
<th></th>
<th>Memory bits per square cm</th>
</tr>
</thead>
<tbody>
<tr>
<td>modern microprocessor</td>
<td>25 billion</td>
</tr>
<tr>
<td>TOY-8</td>
<td>1</td>
</tr>
</tbody>
</table>
A not-so-short answer, in case someone asks...

- A circuit known as the **CPU** is built from **switches** connected by **wires**.
- The CPU performs operations on information encoded in binary, including its own instructions.
- Circuits with feedback implement **memories**.
- Instructions move information among memories, specify the next operation, or implement mathematical functions based on **Boolean logic**.
- Clock pulses activate sequences of **control signals**, which cause state changes that implement machine instructions.
- Virtually everything else is implemented as **layers of software**, each layer adding additional power and scope.
What is this course about?

A broad introduction to computer science.

Goals

- Empower you to exploit available technology. ✔
- Build awareness of intellectual underpinnings. ✔
- Demystify computer systems. ✔
What is this course about?

A broad introduction to computer science.

Goals

• Empower you to exploit available technology. ✓
• Build awareness of intellectual underpinnings. ✓
• Demystify computer systems. ✓
Image sources
http://download.intel.com/pressroom/images/corefamily/Westmere4.jpg
20. Central Processing Unit

PART II: ALGORITHMS, MACHINES, and THEORY