Building Blocks for a CPU

- Instruction memory
- Program counter
- Adder
Designing a Processor

• The Five Classic Components of a Computer

• Processor = Datapath + Control
Middle third of course

• We’ll implement core MIPS processor three ways:
  
  Single cycle implementation
  
  Multi-cycle implementation (reduces hardware)
  
  Pipelined implementation (improves throughput)

• But first, we’ll review the building blocks.
Two types of logic components

- **Combinational Logic**
  - **Acyclic** - there are no loops in the circuit
  - Output depends only on the current input values (after enough time has elapsed for circuit to stabilize)

- **State elements**
  - The output can depend on previous history

\[
\begin{align*}
\text{if } a=1, \text{ then } x &= 0 \\
\text{if } a=0 \land b=1, \text{ then } x &= 1 \\
\text{if } a=b=0, \text{ then } x &= \text{stored value}
\end{align*}
\]
Some combinational logic blocks

- **Simple gates:**
  - and, or, not, nor, nand, xor

- **Multiplexor:**
  - control (c) chooses which input to pass through to output
  - lines may be multi-bit busses

- **Decoder:**
  - k-bit input selects which of $2^k$ outputs is set to “1”.  

```markdown
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```
More combinational logic blocks

• **Adder:**
  - Here, lines represent multi-bit busses

• **Arithmetic Logic Unit:**
  - control (c) chooses which operation will be used
  - op can be +, -, shift, xor, etc.
3 ways to make combinational circuit

Given a truth table of some function from N input bits to M output bits, you can implement it using:

1. **Random Logic**
   - Build function up from simple gates
   - May have long paths from input to output

2. **PLA (Programmable Logic Array)**
   - Implements function as sum-of-products
   - PLA is 2 logic levels deep (3 if you count inverters)

3. **ROM (Read-Only Memory)**
   - Use memory holding $2^N M$-bit values
   - Each memory cell holds output for one input combination
PLA’s

- Each vertical wire is an “and” of selected inputs (or their negations)
- Each output is an “or” of selected vertical wires

\[(\text{in1}\&\text{in3}) \mid (\neg\text{in2}\&\neg\text{in3})\]
Example: 3-bit adder

<table>
<thead>
<tr>
<th>Inputs</th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Carry</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Which is best depends on concerns

• Speed:
  - Random logic might be slow (signal can go many levels)
  - PLA can be the fastest (only 3 gates deep)

• Size:
  - ROM is usually the largest (it always needs $2^N$ cells)
  - PLA often similar to random logic; not always
    • Consider parity (mod-2 sum) of N inputs:
      - Random logic needs $N-1$ XOR gates (or $3N-3$ NAND’s)
      - PLA needs $2^{N-1}$ product terms (one for each “1” output)

• Ease of implementation:
  - ROM (esp. PROM = programmable ROM) is easy to change
  - PLA’s are convenient too
State Elements

D Latch: When latch is “open”, output = data

D flip-flop: Output only changes at clock edge
Storage Element: Register

- **Register**
  - Like a D Flip-Flop except
    - N-bit input and output
      - there are really N flip-flops
    - Write Enable input
  - **Write Enable:**
    - 0: Data in register will not change
    - 1: Data Out becomes Data In (on the clock edge)
Register File for MIPS

- **We need 32 reg’s and 3 ports:**
  - Two 32-bit output buses: (A & B)
  - One 32-bit input bus: (W)

- **Register selection:**
  - RA selects the register to put on busA
  - RB selects the register to put on busB
  - RW selects the register to be written via busW when Write Enable is 1

- **What happens if RW = RA and WriteEnable=1??**
Implementing read ports

read
address 1

read
address 2

read
data 1

read
data 2
Implementing the write port
Storage Element: Memory

• Memory
  - One input bus: Data In
  - One output bus: Data Out

• Memory word is selected by:
  - If Write Enable = 0, memory location selected by Address is put on Data Out bus
  - If Write Enable = 1, the memory location selected by the Address is overwritten by Data In

• Clock input (CLK)
  - The CLK input is used ONLY during write operation
  - For read, memory acts as combinational logic:
    • Address valid $\rightarrow$ Data Out valid after "access time."
Clocking Methodology

- All storage elements are clocked by same clock edge.
- Combinational logic between storage elements must settle to correct output values in time indicated by dark bar.
Computer building block of the day

CORE STORAGE

- Mercury delay lines (Univac I’s storage) were 100x cheaper than vacuum tubes.

- Replaced by CRT memory (similar, using light instead of sound).

- But memory was still expensive and unreliable.

- “Cores” (little donuts) of certain materials are interesting:
  - If you pass enough current through, it magnetizes “0” or “1”
  - If you pass less current through a magnetized core, it sends a pulse down a second wire but doesn’t change.

- Led to invention of “core storage”: 2-D arrays of cores.
  - “Read” by sending half-critical current through row, sensing column
  - “Write” selected core with half-critical current through row & column.
• Core storage used on Whirlwind computer developed at MIT in early 50’s
  - .14 inch in diameter
  - 2048 16-bit words of storage

• Cores improved steadily over next 20 years
  - .03” core with .019” hole.
  - 4 wires passed through each (X,Y, inhibit, sense)
  - Speeds around 1 microsecond

• And the inevitable patent problems
  - MIT got 2 cents per core. IBM made billion cores/year
  - In 1964, IBM paid one-time fee of $13M - biggest patent payment to date.