The Big Picture: Where are We Now?

- The Five Classic Components of a Computer
- Today’s Topic: Datapath Design, then Control Design

The Big Picture: The Performance Perspective

- Processor design (datapath and control) will determine:
  - Clock cycle time
  - Clock cycles per instruction
- Starting today:
  - Single cycle processor:
    - Advantage: One clock cycle per instruction
    - Disadvantage: long cycle time

The Processor: Datapath & Control

- We're ready to look at an implementation of the MIPS simplified to contain only:
  - memory-reference instructions: lw, sw
  - arithmetic-logical instructions: add, sub, and, or, slt
  - control flow instructions: beq
- Generic Implementation:
  - use the program counter (PC) to supply instruction address
  - get the instruction from memory
  - read registers
  - use the instruction to decide exactly what to do
- All instructions use the ALU after reading the registers
  - memory-reference? arithmetic? control flow?
The MIPS Subset

- **R-type**
  - `add rd, rs, rt`
  - `sub, and, or, slt`

- **LOAD and STORE**
  - `lw rt, rs, imm16`
  - `sw rt, rs, imm16`

- **BRANCH:**
  - `beq rs, rt, imm16`

---

Where We're Going - The High-level View

- **LOAD and STORE**
  - `lw rt, rs, imm16`
  - `sw rt, rs, imm16`

Clocking Methodology

- **Register**
  - Similar to the D Flip Flop except
    - N-bit input and output
    - Write Enable input
  - Write Enable:
    - 0: Data Out will not change
    - 1: Data Out will become Data In (on the clock edge)

- All storage elements are clocked by the same clock edge
**Storage Element: Register File**

- Register File consists of (32) registers:
  - Two 32-bit output buses:
  - One 32-bit input bus: busW
- Register is selected by:
  - RR1 selects the register to put on bus “Read Data 1”
  - RR2 selects the register to put on bus “Read Data 2”
  - WR selects the register to be written via WriteData when RegWrite is 1
- Clock input (CLK)

**Register Transfer Language (RTL)**

- is a mechanism for describing the movement and manipulation of data between storage elements:
  
  \[
  PC <- PC + 4 + R[5] \\
  R[rd] <- R[rs] + R[rt] \\
  R[rt] <- Mem[R[rs]] + immed
  \]

**Instruction Fetch and Program Counter Management**

- \[\text{Instruction} \rightarrow \text{PC} \rightarrow \text{Adder} \rightarrow \text{Sum} \]

- a. Instruction memory
- b. Program counter
- c. Adder
Overview of the Instruction Fetch Unit

- The common RTL operations
  - Fetch the Instruction: \( \text{inst} \leftarrow \text{mem}[\text{PC}] \)
  - Update the program counter:
    - Sequential Code: \( \text{PC} \leftarrow \text{PC} + 4 \)
    - Branch and Jump: \( \text{PC} \leftarrow \text{"something else"} \)

Datapath for Register-Register Operations

- \( \text{R[rd]} \leftarrow \text{R[rs] op R[rt]} \)
  - Example: \( \text{add rd, rs, rt} \)
  - \( \text{RR1}, \text{RR2}, \text{and WR} \) comes from instruction’s \( \text{rs}, \text{rt}, \text{and rd} \) fields
  - \( \text{ALU operation} \) and \( \text{RegWrite} \): control logic after decoding instruction

Datapath for Load Operations

\[ \text{R[rt]} \leftarrow \text{Mem[\text{R[rs]} + \text{SignExt[imm16]}]} \]

Example: \( \text{lw rt, rs, imm16} \)

Datapath for Store Operations

\[ \text{Mem[\text{R[rs]} + \text{SignExt[imm16]}]} \leftarrow \text{R[rt]} \]

Example: \( \text{sw rt, rs, imm16} \)
Datapath for Branch Operations

\[
Z \leftarrow (rs == rt); \text{if } Z, \text{ PC} = \text{PC}+4+\text{imm16}; \text{ else PC} = \text{PC}+4
\]

<table>
<thead>
<tr>
<th>op</th>
<th>rd</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>16 bits</td>
</tr>
</tbody>
</table>

**beq rs, rt, imm16**

- **PC** + 4 from instruction datapath
- **Sum**
- **Shift left 2**
- **Branch target**
- **ALU operation**

### Binary Arithmetic for the Next Address

- In theory, the PC is a 32-bit byte address into the instruction memory:
  - Sequential operation: \(\text{PC}_{31:0} = \text{PC}_{31:0} + 4\)
  - Branch operation: \(\text{PC}_{31:0} = \text{PC}_{31:0} + 4 + \text{SignExt[Imm16]} \times 4\)

- The magic number “4” always comes up because:
  - The 32-bit PC is a byte address
  - All our instructions are 4 bytes (32 bits) long
  - The 2 LSBs of the 32-bit PC are always zeros
  - There is no reason to have hardware to keep the 2 LSBs

- In practice, we can simplify the hardware by using a 30-bit \(\text{PC}_{31:2}\):
  - Sequential operation: \(\text{PC}_{31:2} = \text{PC}_{31:2} + 1\)
  - Branch operation: \(\text{PC}_{31:2} = \text{PC}_{31:2} + 1 + \text{SignExt[Imm16]}\)
  - In either case: Instruction Memory Address = \(\text{PC}_{31:2}\) concat “00”

### Putting it All Together: A Single Cycle Datapath

- We have everything except control signals

### The R-Format (e.g. **add**) Datapath

- ALU operation
- Write register
- Branch target
- Lower 32 bits of address
- Instruction memory
- Data memory
Key Points

- CPU is just a collection of state and combinational logic
- We just designed a very rich processor, at least in terms of functionality
- $ET = IC \times CPI \times Cycle\ Time$
  - where does the single-cycle machine fit in?