The Processor


- We are ready to look at an implementation of the MIPS CPU.
- Simplified to contain only:
  - Memory-reference instructions: \( lw, sw \).
  - Arithmetic-logical instructions: \( add, sub, and, or, slt \).
  - Control flow instructions: \( beq, j \).
- Generic implementation:
  - Use the Program Counter (PC) to supply the instruction address.
  - Get the instruction from memory.
  - Read values from registers.
  - Use the instruction to decide exactly what to do.
- All instructions (except \( j \)) use the ALU after reading the registers.
Implementation Overview:

- Abstract/Simplified View:

  - Two types of functional units:
    - Elements that operate on data values (*combinational* units).
      - Outputs depend only on the current inputs.
      - Example: ALU
    - Elements that contain state (*sequential* units).
      - Has some internal storage (*state*).
      - Example: instruction memory and data memory.
      - Example: registers.
Implementation Overview (continued):

- State Elements:
- Unclocked vs. Clocked: when do we read values, when do we write them?
- Clocks used in synchronous logic:
  - When should an element that contains state be updated?
  - Do not want to update and read at the same time, since state may be inconsistent.
Implementation Overview (continued):

- Clocking Methodology:
  - Will assume “edge-triggered”
    - Still completes in one clock cycle.
    - No feedback can occur in this design — chapter 4 designs fit this requirement.
  - Assume a “simple” implementation:
    - A single long clock cycle is used for every instruction.
    - Easier to understand (yea!).
  - Not practical:
    - Want different instruction classes to take different numbers of clock cycles
      - Improves overall performance.
      - Each clock cycle is then much shorter.
      - More realistic.
      - Requires more complex control.
**Building a datapath:**

- Used by all three classes of instructions.
  - J-format, R-format, I-format.
- Instruction memory:
  - State unit, holds the instructions that make up the program.
  - Given an address, produces an instruction (four bytes).
- Program Counter, PC:
  - State unit.
  - Holds address of the next instruction.
- Add ALU:
  - Combinational unit.
  - Hard-wired to perform only addition.
  - Always uses 4 as the second operand, which is the size (in bytes) of one instruction.

Building a Datapath (continued):

- **R-format Instructions:**
  - `add, sub, and, or, slt`.
  - Three register operands needed.
  - Read two data words from the register file.
  - Write one data word into the register file.

- **ALU** is the full-featured ALU (see Appendix B):
  - 3 control bits for ALU operations:
    - 2 to specify the **Operations** (`add, and, or, slt`).
    - 1 for **Bnegate** (used for `sub`, `slt`).

- **Registers** inputs and outputs:
  - Four inputs:
    - 3 register numbers (5 bits each).
    - 1 register value for writing the result (32 bits).
  - Two outputs:
    - 2 register values for ALU (32 bits each).
  - Control
    - 1 control value **asserted** to write a value to a register.
    - (Register file **always** outputs the contents of register numbers indicated on the Read register inputs.)
Building a Datapath (continued): R

- R-format Instructions:
  - add, sub, and, or, slt.

- What is missing?
  - Where/how do the control wires, RegWrite and ALU operation, get set?
Single-cycle Implementation (continued):

- Overview:
  - Don't Panic! We will look at this in smaller pieces.

What are the new major part(s)?

What part(s) are NOT used by an R-Format Instruction?
Single-cycle Implementation (continued):

- **R-Format instruction**: `add, sub, slt, and, or.`
- Fetch: Get instruction from memory and increment PC.

Blue lines/text indicate the current action.
Single-cycle Implementation (continued):

- R-Format instruction: Get the two source register values from Register file.
- Indicate register to write (will be written later).
- RegDst and RegWrite turned on.

- ALUOp set to “use funct code”.

CSc 252 — Computer Organization
Single-cycle Implementation (continued):

- R-Format instruction: Compute the answer.
  - MemtoReg is 0; we want the ALU result to go to the Registers.
Single-cycle Implementation (continued):

- R-Format instruction: Compute the answer.
  - Branch is 0; PC+4 written to PC.
  - ALU Result is written to Registers.

- Writes are done as clock cycle begins to rise.
**Building a datapath for **lw** and **sw**:**

- Used by all three classes of instructions.
  - J-format, R-format, I-format.

- Instruction memory:
  - State unit, holds the instructions that make up the program.
  - Given an address, produces an instruction (four bytes).

- Program Counter, PC:
  - State unit.
  - Holds address of the next instruction.

- Add ALU:
  - Combinational unit.
  - Hard-wired to perform only addition.
  - Always uses 4 as the second operand, which is the size (in bytes) of one instruction.

Building a Datapath (continued):

- Memory-reference instructions: \texttt{lw}, \texttt{sw}
  
  - Example: \texttt{lw} \texttt{ $t1, offset\_value($t2)}
  
  - Get contents of data memory at \texttt{$t2 + offset\_value} and place in register \texttt{$t1}.

- Sign extend unit: the \texttt{offset\_value} is a 16-bit signed value. Must convert (“extend”) to a 32-bit signed value.

- Data memory unit: Address gives location.
  
  - Two inputs: \texttt{Address} and \texttt{Write data} (both 32 bits).
  
  - One output: \texttt{Read data} (32 bits).
  
  - Two control wires: \texttt{MemWrite}, \texttt{MemRead}.

  
  - Want to \texttt{read} value for \texttt{lw}.
  
    - \texttt{MemRead} ??
    
    - \texttt{MemWrite} ??
Building a Datapath (continued):

- Memory-reference instructions: \texttt{lw, sw}
  - Example: \texttt{sw \ $t3, offset\_value($s7)}
  - Put contents of register $t3 into data memory at $s7 + offset\_value

- Sign extend unit: the \texttt{offset\_value} is a 16-bit signed value. Must convert ("extend") to a 32-bit signed value.

- Data memory unit: Address gives location. Can read value (output) or write value (input).
  - Two inputs: address and write data (both 32 bits).
  - One output: read Data (32 bits).
  - Two control wires: \texttt{MemWrite, MemRead}.

### Instruction Table

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>16 bit number</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
</tr>
</tbody>
</table>

- Want to write value for \texttt{sw}.
  - MemRead ??
  - MemWrite ??
Building a Datapath (continued):

- Memory-reference instructions: \texttt{lw}, \texttt{sw}
- Sign extend unit: the \texttt{offset\_value} is a 16-bit signed value. Must convert (“extend”) to a 32-bit signed value.
- Memory unit: Address gives location. Can read value (output) or write value (input).
  - Two inputs: address and write data (both 32 bits).
  - One output: read Data (32 bits).
  - Two control wires: \texttt{MemWrite}, \texttt{MemRead}.
- The ALU performs what operation for \texttt{lw}? for \texttt{sw}?
Single-cycle Implementation (continued):

- Overview:

  What part(s) are NOT used by \texttt{lw} or \texttt{sw} Instructions?
Single-cycle Implementation (continued):

- I-Format instruction: `lw, sw`.
- Fetch is exactly the same as for R-Format.
- Fetch: Get instruction from memory and increment PC.

This is the same figure as Slide 9.

```
lw $t0, -128($s5)
```
Single-cycle Implementation (continued):

- I-Format instruction: **Load word**.
- Second register specifies the register to write result.
- One register is read and sent to ALU.
- Offset is sign-extended and sent to ALU.

```plaintext
lw  $t0, -128($s5)
```
Single-cycle Implementation (continued):

- I-Format instruction: Load word.
  
  \[ \text{sw $t3, 64($s7$)} \]

- ALU completes operation.
  
  \[ \text{lw $t0, -128($s5$)} \]
Single-cycle Implementation (continued):

- I-Format instruction: Load word.
- Data memory reads contents of Address.
- Data sent to Mux.
Single-cycle Implementation (continued):

- I-Format instruction: Load word.
- Data sent from Mux to destination in Register File.
- PC+4 written to PC.

<table>
<thead>
<tr>
<th>I</th>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>16 bit number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
</tr>
</tbody>
</table>

lw  $t0, -128($s5)
sw  $t0, -128($s5)
Building a Datapath (continued):

- Implementing Control Flow Instructions:
  - **beq**: 3 operands; 2 registers to compare, and a 16-bit signed offset from the PC.
  - Base address is the address of the instruction following the current instructions.
  - Offset is shifted two places (all instruction addresses are word-aligned).
  - Two outputs: Result of comparison, and Branch target.

```
beq $t2, $s7, atTheEnd
```

### Diagram

- Instruction
- Registers
- ALU
- Sum
- Branch target
- RegWrite
- Sign extend
- PC + 4 fm instruction datapath
- Shift left 2
- ALU operation
- Zero
- ALU result
- To branch control logic
Single-cycle Implementation:

- Single Datapath from the four pieces shown earlier (and described in Section 4.2).
  - Memory-reference instructions: \texttt{lw}, \texttt{sw}.
  - Arithmetic-logical instructions: \texttt{add}, \texttt{sub}, \texttt{and}, \texttt{or}, \texttt{slt}.
  - Control flow instructions: \texttt{beq}, \texttt{j}.
- Simplest datapath: execute each instruction in a single clock cycle.
- Requires some duplication of elements (multiple ALU’s, for example).
- Requires separate instruction memory and data memory.
- Can share some elements from different datapaths.
  - Multiplexors are necessary several places.
  - Choose between two inputs depending on the instruction.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|c|c|}
\hline
J & op & 26 bit address & & & & \\
\hline
R & op & rs & rt & rd & shamt & funct \\
\hline
I & op & rs & rt & & 16 bit number & \\
\hline
6 bits & 5 bits & 5 bits & 5 bits & 5 bits & 6 bits & \\
\hline
\end{tabular}
\end{table}
Implementation Overview:

- Abstract/Simplified View:

  - Two types of functional units:
    - Elements that operate on data values (*combinational* units).
      - Outputs depend only on the current inputs.
      - Example: ALU
    - Elements that contain state (*sequential* units).
      - Has some internal storage (*state*).
      - Example: instruction memory and data memory.
      - Example: registers.
Single-cycle Implementation (continued):

- I-Format instruction: `beq`, Branch-if-Equal.
- Fetch: Get instruction from memory and increment PC.
- Fetch is exactly the same as for R-Format.
- This is the same figure as Slide 11.
Single-cycle Implementation (continued):

- I-Format instruction: `beq`, Branch-if-Equal.
- Two registers read and sent to ALU to be compared.
- Sign-extend offset and send to Shift left 2.

<table>
<thead>
<tr>
<th>I</th>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>16 bit number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
</tr>
</tbody>
</table>

- I-Format instruction: `beq`, Branch-if-Equal.
- Two registers read and sent to ALU to be compared.
- Sign-extend offset and send to Shift left 2.
Single-cycle Implementation (continued):

- I-Format instruction: `beq`, Branch-if-Equal.
- ALU performs comparison, sets Zero to result.
- Add ALU adds PC and offset, sends result to Mux.

```plaintext
beq $t2, $t7, downBelow
```
Single-cycle Implementation (continued):

- I-Format instruction: \texttt{beq}, Branch-if-Equal.
- AND Gate determines result of Branch AND Zero.
- Mux triggered to send correct result to PC.
- PC not updated until rising edge of clock cycle.

\begin{tabular}{|c|c|c|c|c|c|}
\hline
\textbf{I} & \textbf{op} & \textbf{rs} & \textbf{rt} & \textbf{16 bit number} \\
\hline
\end{tabular}

\begin{itemize}
\item I-Format instruction: \texttt{beq}, Branch-if-Equal.
\item AND Gate determines result of Branch AND Zero.
\item Mux triggered to send correct result to PC.
\item PC not updated until rising edge of clock cycle.
\end{itemize}
Single-cycle Implementation (continued):

- I-Format instruction: \texttt{bne}, Branch-if-Not-Equal.
- What is changed/added?
Single-cycle Implementation (continued):

- J-Format instruction: \( j \), jump.
- Extra items: Shift left 2, Mux.
- 26 bits shifted to give 28 bits + 4 bits from PC+4.

- Additional control wire.
- ALU not used.