CSE 141 – Computer Architecture
Fall 2003

Lecture 6
The Processor: Datapath and Control

Pramod V. Argade
## Schedule

<table>
<thead>
<tr>
<th>Lecture #</th>
<th>Date</th>
<th>Day</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sep. 25</td>
<td>Thursday</td>
<td>Introduction, Ch. 1</td>
</tr>
<tr>
<td>2</td>
<td>Sep. 30</td>
<td>Tuesday</td>
<td>Performance, Ch. 2</td>
</tr>
<tr>
<td>3</td>
<td>Oct. 2</td>
<td>Thursday</td>
<td>ISA, Ch. 3</td>
</tr>
<tr>
<td>4</td>
<td>Oct. 7</td>
<td>Tuesday</td>
<td>Arithmetic, Ch. 4</td>
</tr>
<tr>
<td>5</td>
<td>Oct. 9</td>
<td>Thursday</td>
<td>Arithmetic, Ch. 4, Continued</td>
</tr>
<tr>
<td>6</td>
<td>Oct. 14</td>
<td>Tuesday</td>
<td>Single cycle CPU, Ch. 5</td>
</tr>
<tr>
<td>7</td>
<td>Oct. 16</td>
<td>Thursday</td>
<td>Single-cycle CPU, Ch. 5</td>
</tr>
<tr>
<td>8</td>
<td>Oct. 21</td>
<td>Tuesday</td>
<td>Multi-cycle CPU, Ch. 5</td>
</tr>
<tr>
<td>9</td>
<td>Oct. 23</td>
<td>Thursday</td>
<td>Exceptions and Review for Midterm</td>
</tr>
<tr>
<td>10</td>
<td>Oct. 28</td>
<td>Tuesday</td>
<td>Mid-term Exam</td>
</tr>
<tr>
<td>11</td>
<td>Oct. 30</td>
<td>Thursday</td>
<td>Pipelining, Ch. 6</td>
</tr>
<tr>
<td>12</td>
<td>Nov. 4</td>
<td>Tuesday</td>
<td>Data and control hazards, Ch. 6</td>
</tr>
<tr>
<td>13</td>
<td>Nov. 6</td>
<td>Thursday</td>
<td>Data and control hazards, Ch. 6</td>
</tr>
<tr>
<td>No Class</td>
<td>Nov. 11</td>
<td>Tuesday</td>
<td>Veteran's Day Holiday</td>
</tr>
<tr>
<td>14</td>
<td>Nov. 13</td>
<td>Thursday</td>
<td>Data and control hazards, Ch. 6</td>
</tr>
<tr>
<td>15</td>
<td>Nov. 18</td>
<td>Tuesday</td>
<td>Advanced pipelining issues, Ch. 6</td>
</tr>
<tr>
<td>16</td>
<td>Nov. 20</td>
<td>Thursday</td>
<td>Memory &amp; cache design, Ch. 7</td>
</tr>
<tr>
<td>17</td>
<td>Nov. 25</td>
<td>Tuesday</td>
<td>Memory &amp; cache design, Ch. 7</td>
</tr>
<tr>
<td>No Class</td>
<td>Nov. 27</td>
<td>Thursday</td>
<td>Thanksgiving Holiday</td>
</tr>
<tr>
<td>18</td>
<td>Dec. 2</td>
<td>Tuesday</td>
<td>Memory &amp; cache design, Ch. 7</td>
</tr>
<tr>
<td>19</td>
<td>Dec. 4</td>
<td>Thursday</td>
<td>Review, Instruction ends</td>
</tr>
<tr>
<td></td>
<td>Dec. 11</td>
<td>Thursday</td>
<td>Final Exam</td>
</tr>
</tbody>
</table>

---

### Datapath and Control Design

- The Five Classic Components of a Computer

![Diagram](image)
Single Cycle Implementation ⇒ Datapath and Control

I. Fetch
II. Decode
Op. Fetch
Execute
Store
Next Instruction

Clock Cycle

Complete Execution of a Single Instruction

Datapath

• Abstract / Simplified View:
Two Types of Logic Components

- **Combinational**
  - elements that operate on data values
  - produces same output if given same inputs

- **State Elements**
  - contains internal storage
  - state elements can be read at any time
  - clock is used to determine when a state element should be written

Clock

- Clock is a free running signal
  - Fixed cycle time
  - Frequency = 1/(cycle time)
Edge-triggered Clocking

- Values stored in the machine are updated on a clock edge
  - The clock edge can be either rising or falling

- By default a state element is written every clock edge
  - An explicit write control signal is required otherwise.

- Edge triggered methodology allows, in the same clock cycle:
  - read the contents of a register
  - send the value through some combinational logic, and
  - write the contents

- Possible to have the same state element as input and output

Storage Elements

D Latch

- Two inputs:
  - the data value to be stored (D)
  - the clock signal (C) indicating when to read & store D

- Two outputs:
  - the value of the internal state (Q) and it's complement

Falling edge triggered D flip-flop

- Output changes only on the clock edge
CPU: Clocking

- All storage elements are clocked by the same clock edge

Components Required to implement the ISA

- Next PC generation
  - Add 4 or extended 16-bit immediate to PC
- Memory
  - Instruction read
  - Data read/write
- Registers (32 x 32-bit)
  - Read register rs
  - Read register rt
  - Write register rt or rd
- Sign extend immediate operand
- ALU to operate on the operands
Register: A Storage Element

- 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)

![Register Diagram]

Register File

- Register File consists of (32) registers:
  - Two 32-bit output busses: busA and busB
  - One 32-bit input bus: busW
- Register is selected by:
  - 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
- Clock input (CLK)

![Register File Diagram]
Memory

- Memory
  - One input bus: Data In
  - One output bus: Data Out
- Memory word is selected by:
  - Address selects the word to put on Data Out
  - Write Enable = 1: address selects the memory word to be written via the Data In bus
- Clock input (CLK)
  - The CLK input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
    - Address valid => Data Out valid after “access time.”

Basic 4 x 2 Static RAM

- 2-to-4 decoder
- Address
- Write enable

```
<table>
<thead>
<tr>
<th>Address</th>
<th>Din[1]</th>
<th>Din[0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
A Simple Implementation of MIPS CPU

- Simplified to contain only:
  - Memory-reference instructions: lw, sw
  - Arithmetic-logical instructions: add, sub, and, or, slt
  - Control flow instructions: beq
- Execution Time = Instructions * CPI * Cycle Time
- Processor design (datapath and control) will determine:
  - Clock cycle time
  - Clock cycles per instruction
- We will design a single cycle processor:
  - Advantage: One clock cycle per instruction
  - Disadvantage: long cycle time

Arithmetic Instructions (R-Type)

- ADD, SUB, AND, OR, SLT
- Example
  add rd, rs, rt

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>6 bits</td>
</tr>
</tbody>
</table>
Load/Store Instructions (I-Type)

- LW, SW
- Examples
  \[
  \text{lw rt, rs, imm16} \\
  \text{sw rt, rs, imm16}
  \]

```
  \begin{array}{cccc}
    & 31 & 26 & 21 & 16 & 0 \\
  \hline
  \text{op} & 6 \text{ bits} & \text{rs} & \text{rt} & \text{immediate} & 16 \text{ bits} \\
  \end{array}
```

Branch (I-Type)

- Beq
- Example
  \[
  \text{beq rs, rt, imm16}
  \]

```
  \begin{array}{cccc}
    & 31 & 26 & 21 & 16 & 0 \\
  \hline
  \text{op} & 6 \text{ bits} & \text{rs} & \text{rt} & \text{displacement} & 16 \text{ bits} \\
  \end{array}
```
Jump (J-Type)

- J
- Example
  J Label

\[ \text{op} \quad \text{target address} \]

6 bits 26 bits

CPU: Instruction Fetch

- RTL version of the instruction fetch step:
  - Fetch the Instruction: mem[PC]
    - Update the program counter:
      - Sequential Code: PC <- PC + 4
      - Branch and Jump: PC <- “something else”
**CPU: Register-Register Operations (Add, Subtract etc.)**

- \( R[rd] \leftarrow R[rs] \text{ op } R[rt] \)  
  Example: \( \text{addU } rd, rs, rt \)  
  - Ra, Rb, and Rw come from instruction’s rs, rt, and rd fields  
  - ALUctr and RegWr: control logic after decoding the instruction

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>6 bits</td>
<td></td>
</tr>
</tbody>
</table>

**CPU: Load Operations**

- \( R[rt] \leftarrow \text{Mem}[R[rs] + \text{SignExt}[\text{imm}16]] \)  
  Example: \( \text{lw } rt, rs, \text{imm}16 \)
### CPU: Store Operations

  Example: `sw rt, rs, imm16`

![CPU: Store Operations Diagram](slide6-25)

### CPU: Datapath for Branching

- `beq rs, rt, imm16`  
  Datapath generates condition (equal)

![CPU: Datapath for Branching Diagram](slide6-26)
CPU: Binary arithmetic for PC

- In theory, the PC is a 32-bit byte address into the instruction memory:
  - Sequential operation: $PC_{31:0} = PC_{31:0} + 4$
  - Branch operation: $PC_{31:0} = PC_{31:0} + 4 + \text{SignExt}[\text{Imm16}] \times 4$
- The magic number "4" always comes up because:
  - The 32-bit PC is a byte address
  - And all our instructions are 4 bytes (32 bits) long
- In other words:
  - 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 $PC_{31:2}$:
  - Sequential operation: $PC_{31:2} = PC_{31:2} + 1$
  - Branch operation: $PC_{31:2} = PC_{31:2} + 1 + \text{SignExt}[\text{Imm16}]$
  - In either case: Instruction Memory Address = $PC_{31:2}$ concat "00"

Single Cycle Implementation

- Putting it all together