Announcements

- **Reading Assignment**
  - Chapter 4. Assessing and Understanding Performance
    Sections 4.1 - 4.6
  - Chapter 5. The Processor: Datapath and Control
    Sections 5.1 - 5.3

- **Homework 3: Due Mon., October 17th in class**
  4.1, 4.2, 4.6, 4.7, 4.8, 4.11, 4.12, 4.19, 4.22, 4.45

  **Additional Problem for Chapter 2:** Write a program in C or C++ to declare and initialize an array of two “ints.” Print a word at a non-word aligned address starting in the first element of the array. Run the program on a PC and on a SPARC Station. **(Hint:** use a pointer to access the word.)

  A. What is the output on each computer?
  B. If the program behavior is different on the two computers give explanation.

- **Quiz**
  **When:** Mon., October 17th, First 10 minutes of the class
  **Topic:** Performance, Chapter 4  **Need:** Paper, pen
# Course Schedule

<table>
<thead>
<tr>
<th>Lecture #</th>
<th>Date</th>
<th>Day</th>
<th>Lecture Topic</th>
<th>Quiz Topic</th>
<th>Homework Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9/26</td>
<td>Monday</td>
<td>Introduction, Ch. 1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>9/28</td>
<td>Wednesday</td>
<td>ISA, Ch. 2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>10/3</td>
<td>Monday</td>
<td>Arithmetic Part 1, Ch. 4</td>
<td>ISA</td>
<td>#1</td>
</tr>
<tr>
<td>4</td>
<td>10/5</td>
<td>Wednesday</td>
<td>Arithmetic Part 2, Ch. 4</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>10/10</td>
<td>Monday</td>
<td>Performance, Ch. 3</td>
<td>Arithmetic</td>
<td>#2</td>
</tr>
<tr>
<td>6</td>
<td>10/12</td>
<td>Wednesday</td>
<td>Single cycle CPU, Ch. 5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>10/17</td>
<td>Monday</td>
<td>Single cycle CPU, Ch. 5</td>
<td>Performance</td>
<td>#3</td>
</tr>
<tr>
<td>8</td>
<td>10/19</td>
<td>Wednesday</td>
<td>Multi-cycle CPU, Ch. 5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>10/24</td>
<td>Monday</td>
<td>Multi-cycle CPU, Ch. 5</td>
<td>Single Cycle CPU</td>
<td>#4</td>
</tr>
<tr>
<td>10</td>
<td>10/26</td>
<td>Wednesday</td>
<td>Review for the Midterm</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>10/31</td>
<td>Monday</td>
<td>Mid-term Exam</td>
<td>Exceptions, Ch. 5 and Pipelining, Ch. 6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>11/2</td>
<td>Wednesday</td>
<td>Exceptions, Ch. 5 and Pipelining, Ch. 6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>11/7</td>
<td>Monday</td>
<td>Pipelining, Ch. 6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>13</td>
<td>11/9</td>
<td>Wednesday</td>
<td>Data and control hazards, Ch. 6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>14</td>
<td>11/14</td>
<td>Monday</td>
<td>Data and control hazards, Ch. 6</td>
<td>Pipeline Hazards</td>
<td>#5</td>
</tr>
<tr>
<td>15</td>
<td>11/16</td>
<td>Wednesday</td>
<td>Memory &amp; cache design, Ch. 7</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>16</td>
<td>11/21</td>
<td>Monday</td>
<td>Memory &amp; cache design, Ch. 7</td>
<td>Cache</td>
<td>#6</td>
</tr>
<tr>
<td>17</td>
<td>11/23</td>
<td>Wednesday</td>
<td>Virtual Memory &amp; cache design, Ch. 7</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>18</td>
<td>11/28</td>
<td>Monday</td>
<td>Course Review</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>12/8</td>
<td>Thursday</td>
<td>Final Exam 7 - 10 PM</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>
Datapath and Control Design

- The Five Classic Components of a Computer
Two Types of Logic Components

- **Combinational**
  - Elements that operate on data values
  - Produces same output if given same inputs

- **State Elements**
  - Contains internal storage
  - May produce different output if given same inputs
  - Clock is used to determine when a state element(s) should be written

Combinational Logic

\[ C = f(A, B) \]

State Element

\[ C = f(A, B, \text{state}) \]
Clock

- Clock is a free running signal
  - Fixed cycle time (period)
  - Frequency = 1/(cycle time)
  - Duty Cycle: (% high)/(%low), e.g. 50/50 Duty Cycle below
  - Jitter: Uncertainty/wobble in rising or falling edge
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 its complement

Falling edge triggered D flip-flop
- Output changes only on the clock edge
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 to:
  - read the contents of a register
  - send the value through some combinational logic, and
  - write the contents of the same or another register

- Possible to have the same state element as input and output
All storage elements are clocked by the same clock edge.
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 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)
Memory

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

- **Memory word**:
  - 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

Write enable

2-to-4 decoder

Address:

Din[1]  Din[0]

D  D  D  D
C latchQ  C latchQ  C latchQ  C latchQ
Enable   Enable   Enable   Enable

Dout[1]  Dout[0]
Register Transfer Language (RTL)

- Mechanism for describing the movement and manipulation of data between storage elements.

- Example:
  
  \[
  R[rd] \leftarrow R[rs] + R[rt] \\
  PC \leftarrow PC + 4 \\
  R[rt] \leftarrow Mem[R[rs] + \text{immediate}] 
  \]
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, j

- 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>

  e.g. `add $t3, $s0, $s5`
Load/Store Instructions (I-Type)

- **LW, SW**
- **Examples**
  
  lw rt, rs, imm16  
  sw rt, rs, imm16  

  
<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>26</td>
<td>21</td>
<td>16</td>
</tr>
</tbody>
</table>

  
  e.g. lw $s3, -4($s2)
Branch (I-Type)

- Beq
- Example
  \[
  \text{beq rs, rt, imm16}
  \]
e.g.
  \[
  0x4c \text{ beq } \$s1, \$t3, -12
  \]
Jump (J-Type)

- J
- Example
  J Label

<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>target address</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>6 bits</td>
<td>26 bits</td>
</tr>
</tbody>
</table>

e.g.

0x8000 0000  j  Label
...

0x8000 0000   j   Label
Single Cycle Implementation ⇒ Datapath and Control

- **Instruction Fetch**
- **Instruction Decode**
- **Operand Fetch**
- **Execute**
- **Result Store**
- **Next Instruction**

Clock Cycle

L. Fetch Decode Op. Fetch Execute Store Next PC

Complete Execution of a Single Instruction
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
Datapath

- Abstract / Simplified View:
CPU: Instruction Fetch

- Instruction fetch step:
  - Read Instruction Memory: Instruction Word $\leftarrow$ mem[PC]
  - Update the program counter:
    - Sequential Code: PC $\leftarrow$ PC + 4
    - Branch and Jump: PC $\leftarrow$ "something else"

![Diagram of CPU Instruction Fetch]
CPU: Register-Register Operations (Add, Subtract etc.)

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

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

![Diagram of CPU register operations]

32 32-bit Registers
CPU: Load Operations

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

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

\[
\begin{array}{cccc|c}
\text{op} & \text{rs} & \text{rt} & \text{immediate} \\
\hline
6 \text{ bits} & 5 \text{ bits} & 5 \text{ bits} & 16 \text{ bits} \\
\end{array}
\]
CPU: Store Operations

  Example: sw rt, rs, imm16

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>26</td>
<td>21</td>
<td>16</td>
</tr>
</tbody>
</table>

6 bits 5 bits 5 bits 16 bits
### CPU: Datapath for Branching

- **beq rs, rt, imm16**

  Datapath generates condition (equal)

<table>
<thead>
<tr>
<th>op</th>
<th>rs</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>

  **Instruction Address**

  - Sign extend to 32 bits and left shift by 2

  **“Cond” signal is used to compute “nPC_sel”**
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: $PC_{<31:0>} = PC_{<31:0>} + 4 + \text{SignExt}[\text{Imm16}] * 4$
- 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
  - Supports Arithmetic, Load/Store and Branch instructions