Last Time: CS 152 Course Introduction

Single-cycle CPU project
3 weeks

Pipelined CPU
4 weeks

Final Project
5 weeks
200 hr/student

Teams of 4-5 students
Today: Single Cycle Datapath Design

The book presentation of single cycle processors is sufficient to do Lab 2.

This lecture is not.

This lecture is a gentle introduction, to prepare you to read the book ...
Single cycle data paths: Assumptions

Processor uses synchronous logic design (a “clock”).

<table>
<thead>
<tr>
<th>f</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 MHz</td>
<td>1 μs</td>
</tr>
<tr>
<td>10 MHz</td>
<td>100 ns</td>
</tr>
<tr>
<td>100 MHz</td>
<td>10 ns</td>
</tr>
<tr>
<td>1 GHz</td>
<td>1 ns</td>
</tr>
</tbody>
</table>

All state elements act like positive edge-triggered flip flops.
Review: Edge-Triggered D Flip Flops

Value of D is sampled on **positive clock edge**. Q outputs sampled value for rest of cycle.
Module code has two bugs.

Where?

Value of D is sampled on positive clock edge. Q outputs sampled value for rest of cycle.

```verilog
module ff(D, Q, CLK);
    input D, CLK;
    output Q;

    always @ (CLK)
        Q <= D;

endmodule
```
Review: Edge-Triggered D Flip Flops

Value of D is sampled on positive clock edge. Q outputs sampled value for rest of cycle.

```
module ff(D, Q, CLK);
  input D, CLK;
  output Q;
  reg Q;
  always @(posedge CLK) Q <= D;
endmodule
```
Single cycle data paths: Definition

All instructions execute in a single cycle of the clock (positive edge to positive edge)

Advantage: a great way to learn CPUs.

Drawbacks: unrealistic hardware assumptions, slow clock period
Recall: MIPS R-format instructions

**Syntax:** ADD $8 $9 $10  
**Semantics:** $8 = $9 + $10

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

**Fetch next inst from memory:** 012A4020

<table>
<thead>
<tr>
<th>opcode</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
</table>

Decode fields to get: ADD $8 $9 $10

"Retrieve" register values: $9 $10

Add $9 to $10

Place this sum in $8

Prepare to fetch instruction that follows the ADD in the program.
Goal #1: An R-format single-cycle CPU

Syntax: ADD $8 $9 $10  Semantics: $8 = $9 + $10

| opcode | rs | rt | rd | shamt | funct |

Sample program:
ADD $8 $9 $10
SUB $4 $8 $3
AND $9 $8 $4
...

How registers get their initial values are not of concern to us right now.

No branches or jumps: machine only runs straight line code.

No loads or stores: machine has no use for data memory, only instruction memory.
Separate Read-Only Instruction Memory

Reads are **combinational**: Put a stable address on input, a short time later data appears on output.

Not concerned about how programs are loaded into this memory.

Related to separate instruction and data caches in "real" designs.
Task #1: Straight-line Instruction Fetch

Fetching straight-line MIPS instructions requires a machine that generates this timing diagram:

Why do we increment every clock cycle? Why +4 and not +1?

PC == Program Counter, points to next instruction.
New Component: Register (for PC)

In later examples, we will add an "enable" input: clock edge updates state only if enable is high.
New Component: A 32-bit adder (ALU)

Combinational: Put A and B values on inputs, a short time later A + B appears on output.

ALU: Combinational part that is able to execute many functions of A and B (add, sub, and, or, ...). The “op” value selects the function.

Sometimes, extra outputs for use by control logic ...
Design: Straight-line Instruction Fetch

CS 152: State machine design in the service of an ISA

+4 in hexadecimal

CLK

Addr

Data

Goal #1: An R-format single-cycle CPU

Syntax: ADD $8 $9 $10  
Semantics: $8 = $9 + $10

Fetch next inst from memory: 012A4020

Decode fields to get: ADD $8 $9 $10

"Retrieve" register values: $9 $10

Add $9 to $10

Place this sum in $8

Prepare to fetch instruction that follows the ADD in the program.
Register files: From the top down

Why is R0 special?

R0 - The constant 0

“two read ports”

Even more interesting from the bottom up...
Why do we need WE?

If we had a register file w/o WE, how could we work around it?
Goal #1: An R-format single-cycle CPU

Syntax: ADD $8 $9 $10  Semantics: $8 = $9 + $10

What do we do with these?

Fetch next inst from memory: 012A4020

Decode fields to get: ADD $8 $9 $10

"Retrieve" register values: $9 $10

Add $9 to $10

Place this sum in $8

Prepare to fetch instruction that follows the ADD in the program.
Computing engine of the R-format CPU

Decode fields to get: ADD $8 $9 $10

What do we do with WE?
Putting it all together ...

Is it safe to use same clock for PC and RegFile?

To rs1, rs2, ws, op decode logic ...
Recall: Edge-Triggered D Flip Flops

Value of $D$ is sampled on positive clock edge. $Q$ outputs sampled value for rest of cycle.
Reprise: Putting it all together ...

It IS safe to use same clock for PC and RegFile!
Next Steps:

- Design stand-alone machines for other major classes of instructions: branches, load/store, immediates.

- Learn how to efficiently “merge” single-function machines to make one general-purpose machine.

- Implementing control structures for the single-cycle datapath.
Memory Instructions: LW $1, 32($2)

For next class ...

Fetch the load inst from memory

```
<table>
<thead>
<tr>
<th>opcode</th>
<th>rs</th>
<th>rt</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;I-Format&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Decode fields to get: LW $1, 32($2)

“Retrieve” register value: $2

Compute memory address: 32 + $2

Load memory address contents into: $1

Prepare to fetch instr that follows the LW in the program. Depending on load semantics, new $1 is visible to that instr, or not until the following instr ("delayed loads").
Branch Instructions: \texttt{BEQ \$1,\$2,25}

Also for next class ...

Fetch branch inst from memory

```
| opcode | rs  | rt  | offset |
```

"I-Format"

Decode fields to get: \texttt{BEQ \$1,\$2, 25}

"Retrieve" register values: $1, \$2

Compute if we take branch: $1 == \$2$

ALWAYS prepare to fetch instr that follows the \texttt{BEQ} in the program ("delayed branch"). IF we take branch, the instr we fetch AFTER that instruction is PC + 4 + 100.

\texttt{PC} == "Program Counter"
Administrivia: Upcoming deadlines ...

**Friday:** “Teams meet the TAs”, 12-2, 125 Cory. For no-150 students, 150 Lab Lecture 1”, 2-3 PM, 125 Cory.

**Tuesday:** Lab 1 final report due, 11:59 PM, via the submit program.

**Thursday:** Lab 2 preliminary design document due to TAs via email, 11:59 PM.
Office Hours, Mid-terms ...

David: TBA
Udam: TBA
John: Mon 9:30-10:30 AM

Mid-term 1: Tuesday October 4th, 6:00 to 9:00 PM, 310 Soda.

Mid-term 2: Tuesday December 6th, 6:00 to 9:00 PM, 310 Soda.

Last call for schedule conflicts ...