2005-1-25
John Lazzaro
(www.cs.berkeley.edu/~lazzaro)

TAs: Ted Hong and David Marquardt

www-inst.eecs.berkeley.edu/~cs152/
Last Time: Goal #1, an R-format CPU

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

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

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.
Last Time: An R-format CPU design

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

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

Logic

RegFile

rs1
rs2
ws
wd

rd1
rd2
WE

ALU

op

32

32

32

32

32
Today’s Lecture: Single-Cycle Wrap-up

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

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

- Implementing control structures for the single-cycle datapath.
Goal #2: add I-format ALU instructions

Syntax: ORI $8 $9 64  Semantics: $8 = $9 | 64

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

In this example, $9 is rs and $8 is rt.

16-bit immediate extended to 32 bits.

Zero-extend: 0x8000 ⇔ 0x00008000

Sign-extend: 0x8000 ⇔ 0xFFFF8000

Some MIPS instructions zero-extend immediate field, other instructions sign-extend.
Computing engine of the I-format CPU

Decode fields to get: ORI $8 $9 64

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

In a Verilog implementation, what should we do with rs2?
Merging data paths ...

Add muxes

How many? Where?

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

R-format

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

I-format

Logic

op

Logic

Logic
The merged data path ...

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

\[ \text{RegDest} \]

\[ \text{RegFile} \]

- \( \text{RegDest} \)
- \( \text{RegFile} \) with \( \text{rs1}, \text{rs2}, \text{ws}, \text{wd}, \text{rd1}, \text{rd2} \)
- \( \text{WE} \)
- \( \text{Ext} \) with \( \text{ExtOp}, \text{ALUsrc} \)
- ALU with \( \text{op}, \text{ALUctr} \)

\[ \begin{array}{cc}
\text{op} & \text{rs} \\
\text{rt} & \text{rd} \\
\text{immediate} & \\
\end{array} \]
Administrivia: Upcoming deadlines ...

Thursday: Lab 2 preliminary design document due to TAs via email, 11:59 PM.

Friday: “Design Document Review”, 12-1, 119 Cory. For 61(c) students, 150 Lab Lecture 1”, 1-2 PM, 125 Cory.

Monday: Lab 2 final design document due to TAs via email, 11:59 PM.
Memory Instructions
Loads, Stores, and Data Memory ...

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

**Syntax:** \( \text{LW} \ $1, \ 30($2) \)  
**Action:** \( $1 = M[$2 + 30] \)

**Syntax:** \( \text{SW} \ $3, \ 10($4) \)  
**Action:** \( M[$4 + 10] = $3 \)

**Zero-extend or sign-extend immediate field?**

Reads are **combinational**: Put a stable address on \( \text{Addr} \), a short time later \( \text{Dout} \) is ready

Writes are **clocked**: If \( \text{WE} \) is high, memory \( \text{Addr} \) captures \( \text{Din} \) on positive edge of clock.

**Note:** Not a realistic main memory (DRAM) model ...

CS 152 L3: Single Cycle Wrap-up

UC Regents Spring 2005 © UCB
Adding data memory to the data path

Syntax: LW $1, 30($2)  
Action: $1 = M[$2 + 30]

Syntax: SW $3, 10($4)  
Action: M[$4 + 10] = $3
Branch Instructions
Conditional Branches in MIPS ...

Syntax: BEQ $1, $2, 12

Action: If ($1 != $2), PC = PC + 4

Action: If ($1 == $2), PC = PC + 4 + 48

Immediate field codes # words, not # bytes.
Why is this encoding a good idea?

Zero-extend or sign-extend immediate field?

??? (A bug fix for Lecture 1)
Adding branch testing to the data path

Syntax: BEQ $1, $2, 12
Action: If ($1 != $2), PC = PC + 4
Action: If ($1 == $2), PC = PC + 4 + 48
Recall: Straight-line Instruction Fetch

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

PC == Program Counter, points to next instruction.
Recall: Straight-line Instruction Fetch

**Syntax:** \texttt{BEQ} $1$, $2$, 12

**Action:** If ($1 \neq $2), \( PC = PC + 4 \)

**Action:** If ($1 = $2), \( PC = PC + 4 + 48 \)

### Diagram

- **PC**
- **Instr Mem**
- \( \text{Addr} \rightarrow \text{Data} \)
- **D**
- **Q**
- **Clk**
- **Addr**
- **Data**

- **IMem[PC]**
- **IMem[PC + 4]**
- **IMem[PC + 8]**

**How do we add this behavior?**
**Syntax:** BEQ $1, $2, 12

**Action:** If ($1 != $2), PC = PC + 4

**Action:** If ($1 == $2), PC = PC + 4 + 48
Single-Cycle Control
What is single cycle control?

Combinational Logic (Only Gates, No Flip Flops)
Just specify logic functions!

Instr Mem

RegFile

RegDest

RegWr

ExtOp

ALUsrc

ALUctr

ALU

MemToReg

Data Memory

MemWr

MemToReg

CS 152 L3: Single Cycle Wrap-up
Two goals when specifying control logic

**Bug-free:** One “0” that should be a “1” in the control logic function breaks contract with the programmer.

Should be easy for humans to read and understand: sensible signal names, symbolic constants ...

**Efficient:** Logic function specification should map to hardware with good performance properties: fast, small, low power, etc.
Where we are now, and what is next

We know how to map ISA syntax and semantics into single-cycle hardware

- **How to make sure your Lab 2 design implements the ISA correctly.**
- **Top-down view of how signals move through your processor in time.**
- **Software for teamwork, group dynamics, etc ...**