CS 152
Computer Architecture and Engineering

Lecture 3 – Single Cycle Wrap-Up

2005-9-6

John Lazzaro
(www.cs.berkeley.edu/~lazzaro)

TAs: David Marquardt and Udam Saini

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

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

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
Reminder: How data flows after posedge

0x4

RegFile
rs1
rs2
ws
wd
rd1
rd2
WE

Instr Mem
Addr
Data

PC
D
Q

Logic
op

ALU

Mem

Instr

Data

Addr

PC

+
Next posedge: Update state and repeat
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.

And also, Design Notebook for Lab 2 ...
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 \(\Rightarrow\) 0x00008000

Sign-extend: 0x8000 \(\Rightarrow\) 0xFFFF8000

Some MIPS instructions zero-extend immediate field, other instructions sign-extend.
In a Verilog implementation, what should we do with rs2?
Merging data paths ...

Add muxes

How many? Where?

R-format

I-format

op  rs  rt  immediate

op  rs  rt  funct

Logic

RegFile

32
rs1
rs2
rd1
rd2
ws
rd2
rs1
op

Logic

Ext

RegFile

32
rs1
rs2
rd1
rd2
ws
rd2

Logic

Ext

ALU
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>

RegFile

RegDest

ALUsrc

Ext

ExtOp

ALUsrc

op
rs
rt
immediate

Step 1a: The MIPS-lite Subset for today
° ADD and SUB
• addU ...
016212631
6 bits 16 bits 5 bits 5 bits
op rs rt immediate
016212631
6 bits 16 bits 5 bits 5 bits
Administrivia: Upcoming deadlines ...

Tonight: Lab 1 final report due, 11:59 PM, via the submit program. Or email to lazzaro@eecs

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

Friday: “Design Document Review” in section, 125 Cory. For non-150s, 150 Lab Lecture, 2-3 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:  \( \text{LW} \ $1, \ 32($2) \)
Action: \( $1 = M[\ $2 + 32] \)

Syntax:  \( \text{SW} \ $3, \ 12($4) \)
Action: \( M[\ $4 + 12] = $3 \)

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 ...
Adding data memory to the data path

Syntax:  
LW $1, 32($2)  
$1 = M[$2 + 32]

Syntax:  
SW $3, 12($4)  
M[$4 + 12] = $3
Branch Instructions
Conditional Branches in MIPS ...

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

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?
Adding branch testing to the data path

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

**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:** \( \text{BEQ } \$1, \$2, 12 \)

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

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

How do we add this behavior?
Design: Instruction Fetch with Branch

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

Action: If ($\$1 \neq \$2$), $\text{PC} = \text{PC} + 4$

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

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

RegFile
rs1  rd1  rs2  rd2
ws    wd
RegDest
Equal
RegWr
ExtOp
ALUsrc
ALUctr

Instr
Mem
Addr  Data

MemWr
MemToReg

RegDest

ALU
32
L
U
32
32
op

MemToReg

Data Memory
Addr  Dout
Din  WE

MemWr

PCSrc
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.
Advice: Carefully written Verilog will yield identical semantics in ModelSim and Synplicity. If you write your code in this way, many “works in Modelsim but not on Xilinx” issues disappear.

Always check log files, and inspect output tools produce!

Look for tell-tale Synplicity “warnings and errors” messages!

“latch generated”, “combinational loop detected”, etc

Automate with scripts if possible.
Labs: A small subset of MIPS ...

Implement the following instructions in your processor:

<table>
<thead>
<tr>
<th>Type</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>arithmetic</td>
<td>addu, subu, addiu</td>
</tr>
<tr>
<td>logical</td>
<td>and, andi, or, ori, xor, xori, lui</td>
</tr>
<tr>
<td>shift</td>
<td>sll, sra, srl</td>
</tr>
<tr>
<td>compare</td>
<td>slt, slti, sltu, sltui</td>
</tr>
<tr>
<td>control</td>
<td>beq, bne, bgez, bltz, j, jr, jal</td>
</tr>
<tr>
<td>data transfer</td>
<td>lw, sw</td>
</tr>
<tr>
<td>Other:</td>
<td>break</td>
</tr>
</tbody>
</table>

What if some other instruction appears in the instruction stream?

Note that unlike commercial implementations, your processor does not implement exception handling. So, if an instruction other than the ones listed above appears in the instruction stream, what your processor does is undefined by this spec (a practical option is to treat undefined instructions as no-ops).

For labs: undefined.  
Real world: exceptions.
Why not in labs? Doubles complexity!

Components in blue handle exceptions ...
Will cover this (pipelined CPU) example later in the term ...
Where we are now, and what is next

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

<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Th 9/8</td>
<td>Testing and Teamwork</td>
</tr>
<tr>
<td>F 9/9</td>
<td></td>
</tr>
<tr>
<td>Sa 9/10</td>
<td></td>
</tr>
<tr>
<td>Su 9/11</td>
<td></td>
</tr>
<tr>
<td>M 9/12</td>
<td>Timing</td>
</tr>
<tr>
<td>T 9/13</td>
<td></td>
</tr>
</tbody>
</table>

How to make sure your Lab 2 design implements the ISA correctly.

Software for teamwork, group dynamics, etc ...

Top-down view of how signals move through your processor in time.
In the news: Roll-out displays ...

Prototype from Philips: **Monochrome, 5 inch, 320 x 240 pixels, slow refresh ("e-paper").**
Design Notebook
Why should you keep a design notebook?

- Keep track of the design decisions and the reasons behind them
  - Otherwise, it will be hard to debug and/or refine the design
  - Write it down so that you can remember in long project: 2 weeks -> 2 yrs
  - Others can review notebook to see what happened

- Record insights you have on certain aspect of the design as they come up

- Record of the different design & debug experiments
  - Memory can fail when very tired

- Industry practice: learn from others mistakes
Why do we keep it on-line?

° You need to force yourself to take notes
  • Open a window and leave an editor running:
    1) Acts as reminder to take notes
    2) Makes it easy to take notes
  • 1) + 2) => will actually do it

° Take advantage of the window system’s “cut and paste” features

° It is much easier to read typing than writing

° Also, paper log books have problems
  • Limited capacity => end up with many books
  • May not have right book with you.
  • Can use computer to search files.
How to do it? See “Resources” web page

° Keep it simple
  • DON’T make it too elaborate (fonts, layout, ...)

° Separate the entries by dates
  • type “date” command in another window and cut&paste

° Start day with problems going to work on today

° Record output of simulation into log with cut&paste; add date
  • May help sort out which version of simulation did what

° Record key email with cut&paste

° Record of what works & doesn’t helps team decide what went wrong after you left

° Index: write a one-line summary of what you did at end of each day
* Index

Wed Sep  6 00:47:28 PDT 1995 - Created the 32-bit comparator component
Thu Sep  7 14:02:21 PDT 1995 - Tested the comparator
Mon Sep 11 12:01:45 PDT 1995 - Investigated bug found by Bart in comp32 and fixed it

Wed Sep  6 00:47:28 PDT 1995

Goal: Layout the schematic for a 32-bit comparator

I've layed out the schematics and made a symbol for the comparator. I named it comp32. The files are
~/wv/proj1/sch/comp32.sch
~/wv/proj1/sch/comp32.sym

Wed Sep  6 02:29:22 PDT 1995

• Add 1 line index at front of log file at end of each session: date+summary
• Start with date, time of day + goal
• Make comments during day, summary of work
• End with date, time of day (and add 1 line summary at front of file)
Goal: Test the comparator component

I've written a command file to test comp32. I've placed it in ~/wv/proj1/diagnostics/comp32.cmd.

I ran the command file in viewsim and it looks like the comparator is working fine. I saved the output into a log file called ~/wv/proj1/diagnostics/comp32.log

Notified the rest of the group that the comparator is done.
Goal: Investigate bug discovered in comp32 and hopefully fix it

Bart found a bug in my comparator component. He left the following e-mail.

---

From bart@simpsons.residence Sun Sep 10 01:47:02 1995
Received: by wayne.manor (NX5.67e/NX3.0S)
          id AA00334; Sun, 10 Sep 95 01:47:01 -0800
Date: Wed, 10 Sep 95 01:47:01 -0800
From: Bart Simpson <bart@simpsons.residence>
To: bruce@wayne.manor, old_man@gokuraku, hojo@sanctuary
Subject: [cs152] bug in comp32
Status: R

Hey Bruce,
I think there's a bug in your comparator.
The comparator seems to think that ffffffffff and fffffffff7 are equal.

Can you take a look at this?
Bart
---
I verified the bug. here's a viewsim of the bug as it appeared..
  (equal should be 0 instead of 1)

------------------
SIM>stepsize 10ns
SIM>v a_in A[31:0]
SIM>v b_in B[31:0]
SIM>w a_in b_in equal
SIM>a a_in ffffffff\h
SIM>a b_in ffffffff7\h
SIM>sim
  time =  10.0ns  A_IN=FFFFFFFF\H B_IN=FFFFFFFF7\H EQUAL=1
Simulation stopped at 10.0ns.
------------------

Ah. I've discovered the bug. I mislabeled the 4th net in
the comp32 schematic.

I corrected the mistake and re-checked all the other
labels, just in case.

I re-ran the old diagnostic test file and tested it against
the bug Bart found. It seems to be working fine. hopefully
there aren't any more bugs:)

On second inspection of the whole layout, I think I can remove one level of gates in the design and make it go faster. But who cares! the comparator is not in the critical path right now. the delay through the ALU is dominating the critical path. so unless the ALU gets a lot faster, we can live with a less than optimal comparator.

I e-mailed the group that the bug has been fixed

Mon Sep 11 14:03:41 PDT 1995

- ====================================================================

• Perhaps later critical path changes; what was idea to make comparator faster? Check log book!
Where we are now, and what is next

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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Th 9/8</td>
<td>Testing and Teamwork</td>
</tr>
<tr>
<td>F 9/9</td>
<td></td>
</tr>
<tr>
<td>Sa 9/10</td>
<td></td>
</tr>
<tr>
<td>Su 9/11</td>
<td></td>
</tr>
<tr>
<td>M 9/12</td>
<td></td>
</tr>
<tr>
<td>T 9/13</td>
<td>Timing</td>
</tr>
</tbody>
</table>

How to make sure your Lab 2 design implements the ISA correctly.

Software for teamwork, group dynamics, etc ...

Top-down view of how signals move through your processor in time.