Outline

1. Single clock cycle per instruction
   => Data path resource used at most once per instruction

2. Need to replicate some data path resources if need more than once: memory, ALUadders, ...

3. Timing, Testing of Single Cycle

4. Single Cycle Datapath
   - 5 generic steps (on next slide)
   - See how far we get this lecture; finish next time

5. Design Notebook

The MIPS Instruction Formats

- All MIPS instructions are 32 bits long. The three instruction formats:

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

- The different fields are:
  - op: operation of the instruction
  - rs, rt, rd: the source and destination register specifiers
  - shamt: shift amount
  - funct: selects the variant of the operation in the “op” field
  - address / immediate: address offset or immediate value
  - target address: target address of the jump instruction

Step 1a: The MIPS-lite Subset for today

- ADD and SUB
  - addu rd, rs, rt
  - subu rd, rs, rt

- OR Immediate:
  - ori rt, rs, imm16

- LOAD and STORE Work:
  - lw rt, rs, imm16
  - sw rt, rs, imm16

- BRANCH:
  - beq rs, rt, imm16

How to Design a Processor: step-by-step

1. Analyze instruction set => datapath requirements
   - the meaning of each instruction is given by the register transfers
   - datapath must include storage element for ISA registers
   - possibly more
   - datapath must support each register transfer

2. Select set of datapath components and establish clocking methodology

3. Assemble datapath meeting the requirements

4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.

5. Assemble the control logic

Review

- Timing key concept: critical path
- Xilinx: Physical, yet configurable
Lectures vs. Chapter 5 COD 3/e

- Difficult complaint for textbook author: Lecturing directly from the textbook!
- Good news: These lectures differ from book

Lectures
- MIPS-lite subset:
  - Add, Sub, LW, SW
  - BEQ, OR

Book
- MIPS-lite subset:
  - Add, Sub, LW, SW
  - BEQ, OR
  - AND, SLT, J

Using Hardware Description Lang.
- All start by fetching the instruction

<table>
<thead>
<tr>
<th>inst</th>
<th>HDL description</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>rs</td>
</tr>
<tr>
<td>ADDU</td>
<td>R[rd] &lt;= R[rs] + R[rt]; PC &lt;= PC + 4</td>
</tr>
<tr>
<td>SUBU</td>
<td>R[rd] &lt;= R[rs] – R[rt]; PC &lt;= PC + 4</td>
</tr>
<tr>
<td>ORI</td>
<td>R[rt] &lt;= R[rs]</td>
</tr>
<tr>
<td>LOAD</td>
<td>R[rt] &lt;= MEM[ R[rs] + sign_ext(imm16)]; PC &lt;= PC + 4</td>
</tr>
<tr>
<td>STORE</td>
<td>MEM[ R[rs] + sign_ext(imm16) ] &lt;= R[rt]; PC &lt;= PC + 4</td>
</tr>
<tr>
<td>BEQ</td>
<td>if ( R[rs] == R[rt] ) PC &lt;= PC + 4 + [sign_ext(imm16), 2's complement] else PC &lt;= PC + 4</td>
</tr>
</tbody>
</table>

Step 1: Requirements of the Instruction Set
- Memory
  - One for instructions, one for data
- Registers (32 x 32bit)
  - Read RS
  - Read RT
  - Write RT or RD
- PC
- Sign Extender (for immediate field)
- Add and Sub register or Extended Immediate
- Add 4 or Extended Immediate to PC

Step 2: Components of the Datapath
- Combinational Logic Elements
- Storage Elements (“State”)
  - Clocking methodology

Combinational Logic Elements (Basic Building Blocks)
- Adder
  - (to add values)
- MUX (multiplexer)
  - (to choose between values)
- ALU
  - (to do add, subtract, or)

Storage Element: Register (Basic Building Block)
- Register
  - Similar to the D Flip Flop except
    - N-bit input and output
    - Write Enable input
  - Write Enable:
    - negated (0): Data Out will not change
    - asserted (1): Data Out will become Data In
**Storage Element: 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 (number) selects the register to put on busA (data)
  - RB (number) selects the register to put on busB (data)
  - RW (number) selects the register to be written via busW (data) when Write Enable is 1
- Clock input (CLK)
  - The CLK input is a factor ONLY during write operation
  - During read operation, Register File behaves as a combinational logic block:
    - RA or RB valid => busA or busB valid after "access time."

**Storage Element: Idealized Memory**

- Memory (idealized)
  - 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."

---

**Administrivia**

- Last mini-lab Friday
- Lab #2 Design Document due Friday
  - Should already have groups
- Reading: Sections 5.1 to 5.4, 5.8, Appendix B of COD 3e

---

**Computers In The News**

- “Intel shifts focus to multicore chip performance” 9/7/04 S.F. Chronicle
  - Intel Corp., which has led the charge in the gigahertz race, is expected to change its tune... Excess heat in PC microprocessors has become an increasingly difficult problem to solve as chipmakers crank up the frequency of the tiny transistors to boost performance. Dual core will be part of Intel's big theme ... which shows Intel is changing direction from focusing on faster and faster gigahertz to other features,” he said. Chips with multiple cores are not new. Both IBM and Sun have such processors for larger server computers and AMD last week introduced its dual-core CPU.

---

**Review: Clocking Methodology**

- All storage elements are clocked by the same clock edge
- Cycle Time = CLK-to-Q + Longest Delay Path + Setup + Clock Skew
- (CLK-to-Q + Shortest Delay Path - Clock Skew) > Hold Time

---

**Step 3: Assemble Datapath meeting our requirements**

- HDL (Verilog) Requirements ⇒ Datapath Assembly
- Instruction Fetch
- Read Operands and Execute Operation
3a: Overview of the Instruction Fetch Unit

- The common operations:
  - Fetch the Instruction: mem[PC]
  - Update the program counter:
    - Sequential Code: PC <= PC + 4
    - Branch and Jump: PC <= “something else”

3b: Add & Subtract

- R[rd] <= R[rs] op R[rt]
  - Example: 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

3c: Logical Operations with Immediate

  - Example: lw rt, rs, imm16

3d: Load Operations

- Mem[R[rs] + SignExt[imm16]] <= R[rt]
  - Example: lw r1, r5, imm16

3e: Store Operations

- Mem[R[rs] + SignExt[imm16]] <= R[rt]
  - Example: sw r1, r5, imm16
### 3f. The Branch Instruction

- **beq** rs, rt, imm16
  - mem[PC] Fetch the instruction from memory
  - Equal <= (R[rs] == R[rt]) Calculate the branch condition
  - if (Equal) Calculate the next instruction’s address
    - PC <= PC + 4 + (SignExt(imm16), 2b00)
  - else
    - PC <= PC + 4

### Datapath for Branch Operations

- **beq** rs, rt, imm16 Datapath generates condition (equal)
- Why? Test if Reg file busA equals Reg file busB
- Why? Chose PC+4 or PC+4+(SignExtimm, 00)

### Lectures vs. Chapter 5 3/e

- **Lectures**
  - MIPS-lite subset:
    - AddU, SubU, LW, SW
    - BEQ, OR
  - Control lines names
    - MemtoReg, PCSrc, ALUSrc, RegDst
    - MemWr, RegWr
    - ExtOp (zero extend or sign extend)
    - ALUctr 3 bits (no NOR)
    - MemWr=0 => MemRead

- **Book**
  - MIPS-lite subset:
    - Add, Sub, LW, SW
    - BEQ, OR
    - AND, SLT, J
  - Control lines names
    - MemtoReg, PCSrc, ALUSrc, RegDst
    - MemWrite, RegWrite
    - No ExtOp since subset address just sign extend
    - ALUoperation 4 bits
    - MemRead & MemWrite

### Types of Bugs in Single Cycle Lab 2?

- Which Epoch most likely to uncover bugs for Single Cycle Design?
- Which Epoch would you expect for Pipelined Design?
- Which tests from Single Cycle Design can be reused for Pipelined Design?
- Suggestions for a good your plan?
Recap Test Plan: The testing timeline

Top-down testing
- Complete processor testing
- Processor testing with self-checks
- Multi-unit testing
- Unit testing

Bottom-up testing
- Processor assembly complete
- Correctly executes simple instructions
- Correctly executes short programs

Which testing types are good for each epoch?

Epoch 1
- Unit testing early
- Multi-unit testing

Epoch 2
- Processor testing with self-checks
- Multi-unit testing
- Unit testing diagnostics

Epoch 3
- Processor testing with self-checks
- Multi-unit testing
- Unit testing diagnostics

Epoch 4
- Complete processor testing
- Processor testing with self-checks
- Multi-unit testing
- Unit testing

Peer Evaluation Question Assumptions

- For the next few slides in this lecture assume that

1. Delay to Access Register File
2. Delay to do 32-bit add
3. Delay to do 32-bit ALU operation
4. Multiplexor delays 0 (ignore for now)

Peer Instruction: What is critical path for BEQ?

1. A, B, C, D, I, K
3. A, B, C, D, E, F, G, I, J, K

1. A, B, C, D, I, K
2. A, B, C, D, G, I, K
3. A, B, C, D, E, F, G, I, J, K

Peer Instruction: What is critical path for LW?

1. A, B, C, H, J, K
3. A, B, C, D, H, I, J, K

Peer Instruction: Which has longest critical path?

1. ADDU
2. BEQ
3. LW
4. ORI
5. SUBU
6. SW

A. PC’s Clk-to-Q
B. Instruction Memory’s Access Time
C. Register File’s Access Time
D. ALU to Perform a 32-bit Operation
E. PC Adder1 adds 4 to PC
F. PC Ext sign extends immediate
G. PC Adder2 adds imm to Adder1 sum
H. Data Memory Access Time
I. Setup Time for PC
J. Setup Time for Register File Write
K. Clock Skew

An Abstract View of the Critical Path

Ideal memory AND register file:
- The CLK input is a factor ONLY during write operation
- During read operation, behave as combinational logic:
  Address valid ⇔ Output valid after “access time.”

Critical Path (Load Operation):
A. PC’s Clk-to-Q
B. Instruction Memory’s Access Time +
C. Register File’s Access Time +
D. ALU to Perform a 32-bit Operation +
E. PC Adder1 adds 4 to PC +
F. PC Ext sign extends immediate +
G. PC Adder2 adds imm to Adder1 sum +
H. Data Memory Access Time +
I. Setup Time for PC +
J. Setup Time for Register File Write +
K. Clock Skew

An Abstract View of the Implementation

Control

Instruction Memory

Datapath

Why do we keep it on-line?

* You need to force yourself to take notes!
  - Open a window and leave an editor running while you work
    1) Acts as reminder to take notes
    2) Makes it easy to take notes
    3) + 2) will actually do it

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

* It is much easier to read your typing than your writing

* Also, paper log books have problems
  - Limited capacity => end up with many books
  - May not have right book with you at time vs. networked screens
  - Can use computer to search files/index files to find what looking for

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 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

An Abstract View of the Design

Why do we keep it on-line?

* You need to force yourself to take notes!
  - Open a window and leave an editor running while you work
    1) Acts as reminder to take notes
    2) Makes it easy to take notes
    3) + 2) will actually do it

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

* It is much easier to read your typing than your writing

* Also, paper log books have problems
  - Limited capacity => end up with many books
  - May not have right book with you at time vs. networked screens
  - Can use computer to search files/index files to find what looking for

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 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
How should you do it?

° Keep it simple
  • DON'T make it so elaborate that you won't use (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

On-line Notebook Example

° Refer to the handout "Example of On-Line Log Book" on CS 152 home page:
  ~cs152/handouts/online_notebook_example.html

1st page of On-line notebook (Index + Wed. 9/6/95)

° Index

Wed Sep 6 05:47:28 PDT 1995 - Created the 32-bit comparator component
Thu Sep 7 14:02:21 PDT 1995 - Tested the comparator
Mon Sep 11 01:45 PDT 1995 - Investigated bug found by Bart in comp32 and fixed it

° 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

2nd page of On-line notebook (Thursday 9/7/95)

° 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

3rd page of On-line notebook (Monday 9/11/95)

° 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 simpson@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> Subject: [cs152] bug in comp32
Status: R

Hey Bruce,

I think there's a bug in your comparator.
The comparator seems to think that ffffffff and fffffff7 are equal.
Can you take a look at this?
Bart

° Refer to the handout "Example of On-Line Log Book" on CS 152 home page:
  ~cs152/handouts/online_notebook_example.html

4th page of On-line notebook (9/11/95 contd)

I verified the bug. Here's a viewsim of the bug as it appeared. [equal should be 0 instead of 1]

From bart simpson@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> Subject: [cs152] bug in comp32
Status: R

Hey Bruce,

I think there's a bug in your comparator.

The comparator seems to think that ffffffff and fffffff7 are equal.

Can you take a look at this?
Bart

-------------------
From bart simpson@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> Subject: [cs152] bug in comp32
Status: R

Hey Bruce,

I think there's a bug in your comparator.

The comparator seems to think that ffffffff and fffffff7 are equal.

Can you take a look at this?
Bart

-------------------
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
SIM>a b_in fffffff7
SIM>sim
time =    10.0ns  A_IN=FFFFFFFF  B_IN=FFFFFFF7  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 on-line notebook!

Lecture Summary

5 steps to design a processor
1. Analyze instruction set => datapath requirements
2. Select set of datapath components & establish clock methodology
3. Assemble datapath meeting the requirements
4. Analyze implementation of each instruction to determine setting of control points that affects the register transfer.
5. Assemble the control logic (Next Lecture)

MIPS makes it easier
- Instructions same size; Source registers, immediates always in same place
- Operations always on registers/immediates

Single cycle datapath => CPI=1, CCT => long

On-line Design Notebook
- Open a window and keep an editor running while you work; cut & paste
- Former CS 152 students (and TAs) say they use on-line notebook for programming as well as hardware design; one of most valuable skills

Refer to the handout as an example