Lecture 5: Review CPU Design

Alvin R. Lebeck
CPS 220
Fall 1999

Administrivia

• Read Chapter 3
A "Typical" RISC

- 32-bit fixed format instruction (3 formats)
- 32 64-bit GPR (R0 contains zero)
- 3-address, reg-reg arithmetic instruction
- Single address mode for load/store: base + displacement
  - no indirection
- Simple branch conditions
- Delayed branch

see: SPARC, MIPS, MC88100, AMD29000, i960, i860
PARisc, POWERPC, DEC Alpha, Clipper,
CDC 6600, CDC 7600, Cray-1, Cray-2, Cray-3

The Big Picture

- The Five Classic Components of a Computer

- Today's Topic: Datapath and Control Design
The Big Picture: The Performance Perspective

- Performance of a machine was determined by:
  - Instruction count
  - Clock cycle time
  - Clock cycles per instruction

- Processor design (datapath and control) will determine:
  - Clock cycle time
  - Clock cycles per instruction

- In this lecture:
  - Single cycle processor:
    - Advantage: One clock cycle per instruction
    - Disadvantage: long cycle time
  - Multi cycle processor

The MIPS Instruction Formats

- All MIPS instructions are 32 bits long. The three instruction formats:
  - R-type
  - I-type
  - J-type

<table>
<thead>
<tr>
<th>Field</th>
<th>R-type</th>
<th>I-type</th>
<th>J-type</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>6 bits</td>
<td>6 bits</td>
<td>6 bits</td>
</tr>
<tr>
<td>rs, rt, rd</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
</tr>
<tr>
<td>shamt</td>
<td>5 bits</td>
<td></td>
<td></td>
</tr>
<tr>
<td>funct</td>
<td>5 bits</td>
<td></td>
<td></td>
</tr>
<tr>
<td>immediate</td>
<td></td>
<td>16 bits</td>
<td></td>
</tr>
<tr>
<td>target address</td>
<td></td>
<td></td>
<td>26 bits</td>
</tr>
</tbody>
</table>

- Fields:
  - op: operation of the instruction
  - rs, rt, rd: the source and destination registers specifier
  - 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
An Abstract View of the Implementation

The Steps of Designing a Processor

- Instruction Set Architecture => Register Transfer Language
- Register Transfer Language =>
  - Datapath components
  - Datapath interconnect
- Datapath components => Control signals
- Control signals => Control logic
RTL: The ADD Instruction

- **add rd, rs, rt**
  - mem[PC] Fetch the instruction from memory
  - PC <- PC + 4 Calculate the next instruction’s address

### Combinational Logic Elements (Building Blocks)

**adder**

![adder](image)

**mux**

![mux](image)

**alu**

![alu](image)
Storage Element: Register (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

![Register Diagram]

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 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)**
  - The CLK input is a factor ONLY during write operation
  - During read operation, 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.”

- Looks similar to register file. **Why have registers?**

---

Overview of the Instruction Fetch Unit

- **The common RTL operations**
  - Fetch the Instruction: mem[PC]
  - Update the program counter:
    - Sequential Code: PC <- PC + 4
    - Branch and Jump: PC <- “something else”
Datapath for Register-Register Operations

- \( R[rd] \leftarrow R[rs] \text{ op } R[rt] \)  
  - Example: add \( rd, rs, rt \) 
  - \( Ra, Rb, \) and \( Rw \) comes from instruction's \( rs, rt, \) and \( rd \) fields 
  - ALUctr and RegWr: control logic after decoding the instruction

<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>

A Single Cycle Datapath

- We have everything except control signals (underline)
Instruction Fetch Unit at the Beginning of Add / Subtract

- Fetch the instruction from Instruction memory: Instruction <- mem[PC]
  - This is the same for all instructions

The Single Cycle Datapath during Add and Subtract

- R[rd] <- R[rs] + / - R[rt]
Instruction Fetch Unit at the End of Add and Subtract

- **PC <- PC + 4**
  - This is the same for all instructions except: Branch and Jump

![Diagram of Instruction Fetch Unit]

The “Truth Table” for RegWrite

<table>
<thead>
<tr>
<th>op</th>
<th>00 0000</th>
<th>00 1101</th>
<th>10 0011</th>
<th>10 1011</th>
<th>00 0100</th>
<th>00 0010</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-type</td>
<td>ori</td>
<td>lw</td>
<td>sw</td>
<td>beq</td>
<td>jump</td>
<td></td>
</tr>
<tr>
<td>RegWrite</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **RegWrite = R-type + ori + lw**
  - (R-type)
  - (ori)
  - (lw)

![Diagram of Truth Table for RegWrite]
### PLA Implementation of the Main Control

![PLA Implementation Diagram]

### Putting it All Together: A Single Cycle Processor

![Putting it All Together Diagram]
**Drawback of this Single Cycle Processor**

- **Long cycle time:**
  - Cycle time must be long enough for the load instruction:
    - PC's Clock -to-Q +
    - Instruction Memory Access Time +
    - Register File Access Time +
    - ALU Delay (address calculation) +
    - Data Memory Access Time +
    - Register File Setup Time +
    - Clock Skew

- **Cycle time is much longer than needed for all other instructions**

---

**Overview of a Multiple Cycle Implementation**

- **The root of the single cycle processor’s problems:**
  - The cycle time has to be long enough for the slowest instruction

- **Solution:**
  - Break the instruction into smaller steps
  - Execute each step (instead of the entire instruction) in 1 clock cycle
    - Cycle time: time it takes to execute the longest step
    - Try to make all the steps have similar length
  - This is the essence of the multiple cycle processor

- **The advantages of the multiple cycle processor:**
  - Cycle time is much shorter
  - Different instructions take different number of cycles to complete
    - Load takes five cycles
    - Jump only takes three cycles
  - Allows a functional unit to be used more than once per instruction
The Five Steps of a Load Instruction

1. Instruction Fetch
2. Instr Decode / Reg Fetch
3. Address
4. Data Memory
5. Reg Wr

Clk-to-Q
Instruction Memory Access Time
Delay through Control Logic
Register File Access Time
Delay through Extender & Mux
ALU Delay
Data Memory Access Time

Multiple Cycle Datapath

© Alvin R. Lebeck 1999
Where to get more information?

• Chapter 5 of CPS 104 text book:
  – David Patterson and John Hennessy, “Computer Organization &
    Design: The Hardware / Software Interface,” Morgan Kaufman

• For a reference on the MIPS architecture:
  – Gerry Kane, “MIPS RISC Architecture,” Prentice Hall.

• Next Time: Pipelining