Lecture 26 –
CPU Design: Designing a Single-cycle CPU, pt 2

2006-10-30

Lecturer SOE Dan Garcia

www.cs.berkeley.edu/~ddgarcia

Halloween plans? ⇒ Try the Castro, SF!

Tomorrow 2005-10-31, runs until 11pm

...go at least once...

halloweeninthe Castro.com
Happy Halloween, everyone!
How to Design a Processor: step-by-step

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

2. Select set of datapath components and establish clocking methodology

3. Assemble datapath meeting requirements

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

5. Assemble the control logic
Step 3: Assemble DataPath meeting requirements

- Register Transfer **Requirements** ⇒ Datapath **Assembly**
- Instruction Fetch
- Read Operands and Execute Operation
3a: Overview of the Instruction Fetch Unit

• Common to all instructions:
  • Fetch the Instruction: mem[PC]
  • Update the program counter:
    ▪ Sequential Code: PC ← PC + 4
    ▪ Branch and Jump: PC ← “something else”
3b: Add & Subtract

- $R[rd] \leftarrow R[rs] \text{ op } R[rt]$  Ex.: $\text{addU rd,rs,rt}$

- $Ra$, $Rb$, and $Rw$ come from instruction’s $Rs$, $Rt$, and $Rd$ fields

- ALUctr and RegWr: control logic after decoding the instruction

---

Already defined the register file & ALU
Clocking Methodology

- Storage elements clocked by same edge
- Being physical devices, flip-flops (FF) and combinational logic have some delays
  - Gates: delay from input change to output change
  - Signals at FF D input must be stable before active clock edge to allow signal to travel within the FF (set-up time), and we have the usual clock-to-Q delay
- “Critical path” (longest path through logic) determines length of clock period
Register-Register Timing: One complete cycle

- **Clk**: Clock signal
- **PC**: Program Counter
- **Rs, Rt, Rd, Op, Func**: Register file and operation instructions
- **ALUctr**: ALU control
- **RegWr**: Register write
- **busA, B**: Bus signals
- **busW**: Bus write
- **RegFile**: Register file
- **ALU**: Arithmetic Logic Unit

### Timing Phases:
- **Instruction Memory Access Time**
- **Delay through Control Logic**
- **Register File Access Time**
- **ALU Delay**
- **Register Write Occurs Here**
3c: Logical Operations with Immediate

\[ R[rt] = R[rs] \text{ op ZeroExt}[imm16] \]

<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>21</th>
<th>16</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>op</td>
<td>rs</td>
<td>rt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>16 bits</td>
<td></td>
</tr>
</tbody>
</table>

Immediate value:
```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

16 bits

**Diagram:**

- **RegFile**:
  - Inputs: Rw, Ra, Rb
  - Outputs: RegWr, Rd, Rs, Rt
  - Clock (clk)

- **ALU**:
  - Inputs: busA, busB
  - Output: ALUctr

- **ALUctrl**:
  - Inputs: 32 bits

- **Bus System**:
  - Inputs: busW, busA, busB
  - Outputs: 32 bits
3c: Logical Operations with Immediate

- \( R[rt] = R[rs] \text{ op} \text{ ZeroExt}[imm16] \)

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>26</td>
<td>21</td>
<td>16</td>
</tr>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>16 bits</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What about \( Rt \) register read??

- Already defined 32-bit MUX; Zero Ext?
3d: Load Operations

- $R[rt] = \text{Mem}[R[rs] + \text{SignExt}[\text{imm16}]]$

Example: `lw rt, rs, imm16`

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>16 bits</td>
</tr>
</tbody>
</table>

```
RegDst  Rd  Rt
   1  0

RegWr  Rs  Rt
   5  5  5

RegFile

clk

imm16

ZeroExt

ALUSrc

ALU

busA  32

busB  0

ALUctrl

32

32

32

CS61C L26 CPU Design: Designing a Single-Cycle CPU II (10)

Garcia, Fall 2006 © UCB```
3d: Load Operations

- \( R[rt] = \text{Mem}[R[rs] + \text{SignExt}[\text{imm16}]] \)

Example: \( \text{lw} \ rt, rs, \text{imm16} \)

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>16 bits</td>
</tr>
</tbody>
</table>

RegDst \( \rightarrow \) Rd, Rt

RegWr \( \rightarrow \) Rs, Rt

busW \( \rightarrow \) Rw, Ra, Rb

RegFile

ALUctr

MemtoReg

MemWr

Data Memory

ALU

ExtOp

Imm16

Extender

Data In

0

1
3e: Store Operations

- **Mem[ R[rs] + SignExt[imm16] ] = R[rt]**
  
  **Ex.:** `sw rt, rs, imm16`

- **Example:**
  
  `[R[rs] + SignExt[imm16]] = R[rt]`

- **Operation Format:**
  
<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>16 bits</td>
</tr>
</tbody>
</table>

- **Register File:**
  
  - `RegDst`: 1 bit
  - `Rd`: 5 bits
  - `Rt`: 5 bits

- **ALU Control:**
  
  - `ALUctr`: 32 bits
  - `ExtOp`: 16 bits

- **Mem to Register:**
  
  - `MemtoReg`: 32 bits

- **Memory:**
  
  - `Data Memory`: 32 bits
  - `Adr`: 32 bits
  - `WrEn`: 1 bit

- **ALU:**
  
  - `ALU`: 32 bits
  - `Data In`: 32 bits
  - `WrEn`: 1 bit
  - `Adr`: 32 bits

- **Extender:**
  
  - `ExtOp`: 16 bits
  - `imm16`: 16 bits

- **Data Flow:**
  
  - `busA`, `busB`: 32 bits
  - `clk`: 1 bit
  - `Rw`, `Ra`, `Rb`: 32 bits
  - `Rw`, `Ra`, `Rb`: 32 bits
  - `RegFile`: 32 bits

- **Bus System:**
  
  - `busW`: 32 bits
  - `RegWr`: 5 bits
  - `Rs`: 5 bits
  - `Rt`: 5 bits

- **Clocking:**
  
  - `clk`: 1 bit
  - `ALUctr`: 32 bits
  - `ExtOp`: 16 bits
  - `Imm16`: 16 bits

- **Register File:**
  
  - `RegFile`: 32 bits
  - `Rw`, `Ra`, `Rb`: 32 bits
  - `RegWr`: 5 bits
  - `Rs`: 5 bits
  - `Rt`: 5 bits
3e: Store Operations

- Ex.: sw rt, rs, imm16

<table>
<thead>
<tr>
<th></th>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>16 bits</td>
</tr>
</tbody>
</table>

RegDst, Rd, Rt

RegWr, Rs, Rt

busW

RegFile

busA

ALU

busB

Extender

ALUSrc

Data Memory

MemWr

MemtoReg

Data In

WrEn, Adr

MemtoReg

clk
3f: The Branch Instruction

beq rs, rt, imm16

- mem[PC] Fetch the instruction from memory
- Equal = R[rs] == R[rt] Calculate branch condition
- if (Equal) Calculate the next instruction’s address
  - PC = PC + 4 + (SignExt(imm16) x 4)
- else
  - PC = PC + 4
Datapath for Branch Operations

- **beq rs, rt, imm16**

Datapath generates condition (equal)

![Datapath Diagram](image)

- Inst Address
- ALU
- RegFile
- Mux
- Adder
- PC Ext

**Already have mux, adder, need special sign extender for PC, need equal compare (sub?)**
Putting it All Together: A Single Cycle Datapath

Instruction<31:0> -> Inst Memory

Rs -> Rs
t -> Rt
Rd -> Rd
Imm16 -> Imm16

RegDst

Rd -> Rd
Rt -> Rt
RegWr

Rw -> Rw
Ra -> Ra
Rb -> Rb
RegFile

clk

ALUctr

Data In

Extender

ExtOp

ALUSrc

MemtoReg

MemWr

Mem to Reg

Adder

Adder

Mux

PC Ext

clk

imm16

nPC sel

4

PC Ext

imm16
An Abstract View of the Implementation

Control

Datapath

Ideal Instruction Memory

Instruction Address

Instruction

Rd Rs Rt

5 5 5

Register File

A 32
B

32

ALU

32

Ideal Data Memory

Data In clk

Data Out

Data Addr

clk
A. Our **ALU** is a synchronous device
B. **We should use the main ALU** to compute PC=PC+4
C. The **ALU is inactive** for memory reads or writes.
Summary: Single cycle datapath

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 effects the register transfer.
5. Assemble the control logic

Next time!