Review

• CPU design involves Datapath, Control
  • Datapath in MIPS involves 5 CPU stages
    1) Instruction Fetch
    2) Instruction Decode & Register Read
    3) ALU (Execute)
    4) Memory
    5) Register Write

Single Cycle CPU

For each instruction, how do we control the flow of information though the datapath?
• Single Cycle CPU: All stages of an instruction are completed within one **long** clock cycle.
  • The clock cycle is made sufficient long to allow each instruction to complete all stages without interruption and within one cycle.

1. Instruction Fetch
2. Decode/Register Read
3. Execute
4. Memory
5. Reg. Write

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.
   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

• The common RTL operations
  • Fetch the Instruction: mem(PC)
  • Update the program counter:
    - Sequential Code: PC ← PC + 4
    - Branch and Jump: PC ← “something else”

- Address Instruction Memory
  Instruction Word
  clk
  Next Address
  Logic
  Address
  32
3b: Add & Subtract
• \( R[rd] = R[rs] \text{ op } R[rt] \) Ex.: \text{addU } rd,rs,rt
• \( R_a, R_b, \text{ and } R_w \) come from instruction’s \( Rs, Rt, \) and \( Rd \) fields
• \( ALUctr \) and \( RegWr: \) control logic after decoding the instruction

Already defined the register file & ALU

Register-Register Timing: One complete cycle

3c: Logical Operations with Immediate
• \( R[rt] = R[rs] \text{ op ZeroExt}[imm16] \)

3d: Load Operations
• \( R[rt] = \text{Mem}[R[rs]] + \text{SignExt}[imm16] \)
  Example: \text{lw } rt,rs,imm16

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
3d: Load Operations
- \( R[rt] = \text{Mem}[R[rs] + \text{SignExt}[\text{imm16}]] \)
- Example: \( \text{lw} \ rt, rs, \text{imm16} \)

3e: Store Operations
- \( \text{Mem}[ R[rs] + \text{SignExt}[\text{imm16}] ] = R[rt] \)
- Ex: \( \text{sw} \ rt, rs, \text{imm16} \)

3f: The Branch Instruction
- \( \text{beq} \ rs, rt, \text{imm16} \)
  - \( \text{mem}[PC] \) 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
- \( \text{beq} \ rs, rt, \text{imm16} \)
  - Datapath generates condition (equal)

Putting it All Together: A Single Cycle Datapath
Peer Instruction

A. For the CPU designed so far, the Controller only needs to look at opcode/funct and Equal

B. Adding jal would only require changing the Instruction Fetch block

C. Making our single-cycle CPU multi-cycle will be easy

An Abstract View of the Implementation

Datapath

Control

Ideal Instruction Memory

Instruction Address

Instruction Memory

Control Signals

Conditions

Ideal Data Memory

Data Out

Data In

Ideal Register File

Register File

Rw

Ra

Rb

Rs

Rt

Rd

Reg Wr

Extender

Imm16

ALU

Mem to Reg

Mem Wr

zero

Instr fetch

Unit

nPC sel

Instruction<31:0>

<21:25>

<16:20>

<11:15>

<0:15>

Meaning of the Control Signals

• nPC sel:
  "+4" 0 ⇒ PC ← PC + 4
  "br" 1 ⇒ PC ← PC + 4 + (SignExt(Imm16), 00 )
  "n" = next
  Later in lecture: higher-level connection between mux and branch condition

Halfway there: A Single Cycle Datapath

• We have everything except control signals

Administrivia

• HW4 Due Friday 7/25
  • Cut out problem 8

• HW5 out tonight, due next Tuesday 7/29

• HW1,2 Grade Freeze
  • Submit appeals by Friday 7/25
**Meaning of the Control Signals**

- ExtOp: "zero", "sign"  
- ALUsrc: 0 ⇒ regB; MemtoReg: 0 ⇒ ALU; 1 ⇒ Mem  
- ALUctr: "ADD", "SUB", "OR"  
- MemWr: 1 ⇒ write memory

**RTL: The Add Instruction**

<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>21</th>
<th>16</th>
<th>11</th>
<th>6</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>up</td>
<td>rs</td>
<td>rt</td>
<td>rd</td>
<td>shamt</td>
<td>imm16</td>
<td>funct</td>
</tr>
<tr>
<td>8 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>6 bits</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- add rd, rs, rt
  - MEM[PC] Fetch the instruction from memory
  - PC = PC + 4 Calculate the next instruction's address

**Instruction Fetch Unit at the Beginning of Add**

- Fetch the instruction from Instruction memory: Instruction = MEM[PC]
- same for all instructions

**The Single Cycle Datapath during Add**

R[rd] = R[rs] + R[rt]

- Instruction<31:0> fetch unit
  - Instruction<31:0>:
    - nPC_sel=x  
    - Rs  
    - Rd  
    - Rs  
    - Rd  
    - Rs  
    - Rd  
  - ALUctr
    - Adder
    - Extender
    - Mux
    - ALU
    - RegFile
    - Data Memory

**Single Cycle Datapath during Or Immediate?**


- Instruction<31:0>:
  - nPC_sel=x  
  - Rs  
  - Rd  
  - Rs  
  - Rd  
  - Rs  
  - Rd  
- ALUctr
  - Adder
  - Extender
  - Mux
  - ALU
  - RegFile
  - Data Memory

**Instruction Fetch Unit at the End of Add**

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

- Instruction<31:0> fetch unit
  - Instruction<31:0>:
    - nPC_sel=x  
    - Rs  
    - Rd  
    - Rs  
    - Rd  
    - Rs  
    - Rd  
  - ALUctr
    - Adder
    - Extender
    - Mux
    - ALU
    - RegFile
    - Data Memory
The Single Cycle Datapath during Or Immediate?

- \( R[rt] = R[rs] \) OR ZeroExt[Imm16]

- Instruction<31:0> = Instruction<31:0>

The Single Cycle Datapath during Load?

- \( R[rt] = \) Data Memory (\( R[rs] \) + SignExt[Imm16])

The Single Cycle Datapath during Store?

- Data Memory (\( R[rs] \) + SignExt[Imm16]) = \( R[rt] \)

The Single Cycle Datapath during Store?

- Data Memory (\( R[rs] \) + SignExt[Imm16]) = \( R[rt] \)

- if \( (R[rs] - R[rt]) = 0 \) then Zero = 1; else Zero = 0
The Single Cycle Datapath during Branch

**Step 4: Given Datapath: RTL \( \rightarrow \) Control**

### A Summary of the Control Signals (1/2)

<table>
<thead>
<tr>
<th>Inst</th>
<th>Register Transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>( R[rd] \leftarrow R[rs] + R[rt]; ) PC ( \leftarrow PC + 4 )</td>
</tr>
<tr>
<td>sub</td>
<td>( R[rd] \leftarrow R[rs] - R[rt]; ) PC ( \leftarrow PC + 4 )</td>
</tr>
<tr>
<td>ori</td>
<td>( R[rt] \leftarrow R[rs] ) + zero_ext(Imm16); PC ( \leftarrow PC + 4 )</td>
</tr>
<tr>
<td>lw</td>
<td>( R[rt] \leftarrow MEM[R[rs]] ) + sign_ext(Imm16); PC ( \leftarrow PC + 4 )</td>
</tr>
<tr>
<td>beq</td>
<td>if ( R[rs] == R[rt] ) then PC ( \leftarrow PC + +signext(Imm16)) % else PC ( \leftarrow PC + 4 )</td>
</tr>
</tbody>
</table>

### Boolean Expressions for Controller

#### Where:

- \( \text{rtype} = \text{add}\), \( \text{add} \) + \( \text{sub} \) + \( \text{ori} \) + \( \text{lw} \) + \( \text{sw} \)
- \( \text{MembReg} = \text{lw} \)
- \( \text{RegWrite} = \text{add} \) + \( \text{add} \) + \( \text{ori} \) + \( \text{lw} \)
- \( \text{MemWrite} = \text{sw} \)
- \( nPCsel = \text{beq} \)
- \( \text{Jump} = \text{jump} \)
- \( \text{ExtOut} = \text{lw} + \text{sw} \)
- \( \text{ALUctrl[0]} = \text{sub} + \text{beq} \) (assume \( \text{ALUctrl} = 0\) \( \text{ADD} \), \( \text{SUB} \), \( \text{OR} \), \( \text{ADD} \))

\[
\text{ALUctrl[1]} = \text{or}
\]

We don’t care \( \text{add} + \text{sub} \)

### Instruction Fetch Unit at the End of Branch

- \( \text{R-type} = \text{beq} \) + \( \text{beq} \) + \( \text{beq} \) + \( \text{beq} \) + \( \text{beq} \)
- \( \text{L-type} = \text{lw} \) + \( \text{lw} \) + \( \text{lw} \) + \( \text{lw} \) + \( \text{lw} \)
- \( \text{J-type} = \text{jump} \) + \( \text{jump} \) + \( \text{jump} \) + \( \text{jump} \) + \( \text{jump} \)

### A Summary of the Control Signals (2/2)

- Jump = \text{jump} + \text{lw} + \text{sw}
- ALUctrl[0] = sub + beq (assume ALUctrl = 0 ADD, 0 SUB, 10 ORE)
- ALUctrl[1] = or

- Where:
  - \( \text{rtype} = \text{add}\), \( \text{add} \) + \( \text{add} \) + \( \text{add} \) + \( \text{add} \) + \( \text{add} \)
  - \( \text{lw} = \text{lw} \) + \( \text{lw} \) + \( \text{lw} \) + \( \text{lw} \) + \( \text{lw} \)
  - \( \text{beq} = \text{beq} \) + \( \text{beq} \) + \( \text{beq} \) + \( \text{beq} \) + \( \text{beq} \)
  - \( \text{jump} = \text{jump} \) + \( \text{jump} \) + \( \text{jump} \) + \( \text{jump} \) + \( \text{jump} \)

- How do we implement this in gates?
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
   - Formulate Logic Equations
   - Design Circuits

The Single Cycle Processor

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
   - Formulate Logic Equations
   - Design Circuits

Bonus slides

- These are extra slides that used to be included in lecture notes, but have been moved to this, the “bonus” area to serve as a supplement.
- The slides will appear in the order they would have in the normal presentation.

The Single Cycle Datapath during Jump

- New PC = { PC[31..28], target address, 00 }

  - New PC = { PC[31..28], target address, 00 }
Instruction Fetch Unit at the End of Jump

- New PC = { PC[31..28], target address, 00 }

Query
- Can Zero still get asserted?
- Does nPC_sel need to be 0?
- If not, what?