Review: Processor Design 5 steps
Step 1: Analyze instruction set to determine datapath requirements
  – Meaning of each instruction is given by register transfers
  – Datapath must include storage element for ISA registers
  – Datapath must support each register transfer
Step 2: Select set of datapath components & establish clock methodology
Step 3: Assemble datapath components that meet the requirements
Step 4: Analyze implementation of each instruction to determine setting of control points that realizes the register transfer
Step 5: Assemble the control logic

Processor Design: 5 steps
Step 1: Analyze instruction set to determine datapath requirements
  – Meaning of each instruction is given by register transfers
  – Datapath must include storage element for ISA registers
  – Datapath must support each register transfer
Step 2: Select set of datapath components & establish clock methodology
Step 3: Assemble datapath components that meet the requirements
Step 4: Analyze implementation of each instruction to determine setting of control points that realizes the register transfer
Step 5: Assemble the control logic

3c: Logical Op (or) with Immediate
\[ R_{rt} = R_{rs} \text{ op } \text{ZeroExt}[\text{imm16}] \]

Writing to Rt register (not Rd)!!

What about Rt Read?
3d: Load Operations

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

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

3e: Store Operations

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

Ex: \( sw \ rt, rs, \text{imm16} \)

3f: The Branch Instruction

\[ \text{beq } rs, rt, \text{imm16} \]

- \( \text{mem}[\text{PC}] \) Fetch the instruction from memory
- \( \text{Equal} = R[rs] == R[rt] \) Calculate branch condition
- if \( \text{Equal} \) Calculate the next instruction’s address
  • \( \text{PC} = \text{PC} + 4 + (\text{SignExt}[\text{imm16}] \times 4) \)
  
else
  • \( \text{PC} = \text{PC} + 4 \)

Datapath for Branch Operations \( \text{beq } rs, rt, \text{imm16} \)

Datapath generates condition (Equal)

Instruction Fetch Unit including Branch

- if (Zero == 1) then \( \text{PC} = \text{PC} + 4 + \text{SignExt}[\text{imm16}] \times 4 \); else \( \text{PC} = \text{PC} + 4 \)

Q: What logic gate?

- How to encode \( n\text{PC}_\text{self} \)?
  • Direct MUX select?
  • Branch inst. / not branch inst.
- Let’s pick 2nd option
Putting it All Together: A Single Cycle Datapath

Instruction Memory Addr

Given Datapath: RTL → Control

RTL: The Add Instruction

Single Cycle Datapath during Add

Instruction Fetch Unit at the Beginning of Add
- Fetch the instruction from Instruction memory: Instruction = MEM[PC]
  - same for all instructions

Datapath Control Signals
- ExtOp: “zero”, “sign”
- ALUsrc: 0 → regB; 1 → instr
- ExtOp: “ADD”, “SUB”, “CLR”
- ALUctr: 0 → “rt”; 1 → “rs”
- RegWr: 1 → write register

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

Single Cycle Datapath during Add

R[rd] = R[rs] + R[rt]
Instruction Fetch Unit at End of Add

- \( PC = PC + 4 \)
  - Same for all instructions except: Branch and Jump

Inst Memory

Inst Address

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[rs] \leftarrow \text{zero_ext}(\text{imm}16); PC \leftarrow PC + 4 )</td>
</tr>
<tr>
<td>lw</td>
<td>( R[rt] \leftarrow \text{MEM}[R[rs] + \text{sign_ext}(\text{imm}16)]; \text{PC} \leftarrow \text{PC} + 4 )</td>
</tr>
<tr>
<td>sw</td>
<td>( \text{MEM}[R[rs] + \text{sign_ext}(\text{imm}16)] \leftarrow R[rt]; PC \leftarrow PC + 4 )</td>
</tr>
<tr>
<td>beq</td>
<td>( \text{if (R[rs] == R[rt]) then PC }= PC + \text{sign_ext}(\text{imm}16) ) | ( \text{else } PC \leftarrow PC + 4 )</td>
</tr>
<tr>
<td>jump</td>
<td>( \text{jump} )</td>
</tr>
</tbody>
</table>

Summary of the Control Signals (2/2)

See Appendix A for details.

**Example 1:**

- \( \text{add} \quad \text{sub} \quad \text{ori} \quad \text{lw} \quad \text{sw} \quad \text{beq} \quad \text{jump} \)
- \( \text{RegDest} = 1, \text{MemWrite} = 1, \text{RegWrite} = 1, \text{MemtoReg} = 1, \text{RegDst} = \text{rt}, \text{Jump} = 1 \)
- \( \text{alusrc} = \text{Im}, \text{extop} = \text{ADD}, \text{nPCsel} = +4 \)
- \( \text{aluctr} = 1011 \)
- \( \text{opcode} = \text{add} \)

**Example 2:**

- \( \text{add} \quad \text{sub} \quad \text{ori} \quad \text{lw} \quad \text{sw} \quad \text{beq} \quad \text{jump} \)
- \( \text{RegDest} = 1, \text{MemWrite} = 1, \text{RegWrite} = 1, \text{MemtoReg} = 1, \text{RegDst} = \text{rt}, \text{Jump} = 1 \)
- \( \text{alusrc} = \text{Im}, \text{extop} = \text{ADD}, \text{nPCsel} = +4 \)
- \( \text{aluctr} = 1011 \)
- \( \text{opcode} = \text{add} \)

Boolean Expressions for Controller

- \( \text{RegDest} = \text{add} + \text{sub} \)
- \( \text{alusrc} = \text{ori} + \text{lw} + \text{sv} \)
- \( \text{memwrite} = \text{lw} \)
- \( \text{nPCsel} = \text{beq} \)
- \( \text{nPCsel} = \text{jump} \)
- \( \text{aluctr[2]} = \text{lw} + \text{sv} \)
- \( \text{aluctr[1]} = \text{or} \)
- \( \text{aluctr[0]} = \text{alu} \)

Controller Implementation

- \( \text{opcode} = \text{add} + \text{sub} + \text{ori} + \text{lw} + \text{sv} \)
- \( \text{alusrc} = \text{ori} + \text{lw} + \text{sv} \)
- \( \text{aluctr[2]} = \text{lw} + \text{sv} \)
- \( \text{aluctr[1]} = \text{or} \)
- \( \text{aluctr[0]} = \text{alu} \)

**AND** logic

**OR** logic
1) We should use the main ALU to compute PC=PC+4 in order to save some gates.

2) The ALU is inactive for memory reads (loads) or writes (stores).

Summary: Single-cycle Processor

- Five 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

- How to implement Jump

Single Cycle Datapath during Jump

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

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