Orange Coast College
Business Division
Computer Science Department

CS 116- Computer Architecture

Processor: Control
Main Control Unit

• Remember:
  – Opcode is always in bits 31-26
  – 2 registers to be read (rs & rt) are always in bits 25-21 and 20-16
    • Base register for load/store (rs) is always in bits 25-21
  – 16 bit offset for branch, load/store always in bits 15-0
  – Destination register is in one of 2 places:
    • For load (rt) in bits 20-16
    • For R-type instructions (rd) in bits 15-11
      – Need MUX to select the corresponding field
Main Control Unit

- Read address
- Instruction memory
- Instruction [31-0]

Flow:
1. PC to Add
2. 4 to Add
3. Read address
4. Instruction memory
5. Instruction [31-0]
6. Instruction [25-21]
7. Instruction [20-16]
8. Instruction [15-11]
9. Instruction [15-0]
10. RegDst
11. RegWrite
12. Register file
13. Read reg1
14. Read data1
15. Read reg2
16. Write reg
17. Read data2
18. Write data
19. Instruction [15-0]
20. 16
21. 32
22. Sign extend
23. ALUControl
24. ALUSrc
25. ALUOp
26. Zero
27. ALU result
28. Add
29. ALU result
30. Shift left 2
31. PCSrc
32. 1 MUXO
33. MemWrite
34. MemReg
35. MemRead
36. Address
37. Read data
38. Write data
39. Data memory
40. 1 MUXO
Main Control Unit

- 9-control signals:
  - 7 single-bit
  - 1 two-bit control line for ALUOp
    - 00 for load/store
    - 10 for subtraction
    - 10 for other operations

- The control unit sets all control signals based on the opcode field of the instruction

- Exception: PCSrc signal
  - To generate PCSrc signal, we use an AND gate with the “zero” signal from ALU
Main Control Unit

- For each opcode value, we need to define whether the control signal be 1, 0, or don’t care

<table>
<thead>
<tr>
<th>Signal name</th>
<th>If de-asserted (0)</th>
<th>If asserted (1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegDst</td>
<td>Destination register &lt;= rt field (20-16)</td>
<td>Destination register &lt;= rd field (15-11)</td>
</tr>
<tr>
<td>RegWrite</td>
<td>None</td>
<td>Write data input =&gt; Write register</td>
</tr>
<tr>
<td>ALUSrc</td>
<td>Reg Data2 =&gt; 2nd ALU operand</td>
<td>Sign-extnd 16-bits of instruction =&gt; 2nd ALU operand</td>
</tr>
<tr>
<td>PCSrc</td>
<td>PC &lt;= PC + 4 (from adder)</td>
<td>PC &lt;= Branch target (from adder)</td>
</tr>
<tr>
<td>MemRead</td>
<td>None</td>
<td>Mem[Address] =&gt; to Read data output</td>
</tr>
<tr>
<td>MemWrite</td>
<td>None</td>
<td>Mem[Address] &lt;= value on Write data input</td>
</tr>
<tr>
<td>MemtoReg</td>
<td>Value to Write data input comes from ALU</td>
<td>Value to Write data input comes from data memory</td>
</tr>
</tbody>
</table>
Datapath Combined with Control Unit
Main Control Unit

• Input:
  – 6-bit opcode field from the instruction

• Output:
  – 9 signals
    • 3 signals for multiplexors
      – RegDest, MemtoReg, ALUSrc
    • 3 signals to control reads & writes in register file & data memory
      – RegWrite, MemRead, MemWrite
    – 1 signal to control whether to Branch or not
      • Branch
    – 2 signals for ALU
      • ALUOp
Main Control Unit

- Mapping the Functions to Gates (App. C2, p. C7)

Input (6-bits Op code)

0 p5 0 p4 0 p3 0 p2 0 p1 0 p0

000000 100011 101011 000100

Output (Control signals)

RegD st ALUSrc MemtoReg
RegWrite MemRead MemWrite
Branch ALUOp1 ALUOp0
How do we select control lines?

• R-Format (add, sub, AND, OR, & slt)
  – Source registers: rs & rt
  – Destination register: rd
  – Writes a register (RegWrite = 1)
  • But not reads nor writes data memory
  – ALUOp for R-Type format = 10 indicating that ALU control should be generated from function field

<table>
<thead>
<tr>
<th>Instruct</th>
<th>RegDest</th>
<th>ALUSrc</th>
<th>MemtoReg</th>
<th>RegWrite</th>
<th>MemRead</th>
<th>MemWrite</th>
<th>Branch</th>
<th>ALUOp1</th>
<th>ALUOp0</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-Format</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>lw</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sw</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>beq</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
How do we select control lines?

- **When Branch:**
  - Control signal = 0, \( \text{PC} \leq \text{PC} + 4 \);
  - Otherwise it is replaced by Branch target

- **For lw**
  - MemRead = 1,

- **for sw**
  - MemWrite = 1

**Table:**

<table>
<thead>
<tr>
<th>Instruct</th>
<th>RegDest</th>
<th>ALUSrc</th>
<th>MemtoReg</th>
<th>RegWrite</th>
<th>MemRead</th>
<th>MemWrite</th>
<th>Branch</th>
<th>ALUOp1</th>
<th>ALUOp0</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-Format</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>lw</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sw</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>beq</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Datapath Operation for R-Format $(add)$

Add $t1$, $t2$, $t3$

- Step 1: Fetch instruction from memory
- Step 2: Read Registers
- Step 3: Perform ALU Op
- Step 4: Idle
- Step 5: Write result

Note:
- Step 1 is the same for all instructions
Datapath Operation for R-Format (add)

- Step 1: Fetch instruction & increment PC
Datapath Operation for R-Format \textit{(add)}

- Step 2: Read registers from register file
  - Main control unit uses the opcode to determine the control line setting
  - These units become active in addition to the units during Step 1
Datapath Operation for R-Format \((\text{add})\)

- **Step3 (Fig5.23)**: Perform ALU operation
  - Opcode fields are also used to set control lines

```
Step1
Step2
Step3
Step4
Control
```
Datapath Operation for R-Format \((add)\)

- Step4 (Fig5-24): idle
Datapath Operation for R-Format \textit{(add)}

- Step5 (Fig5-24): Write result from ALU into register
Datapath Operation for I-Format \( (slti) \)

\( slti \) \( \text{st1, st2, 33} \)

- Step 1: Fetch instruction from memory
- Step 2: Read register
- Step 3: Perform ALU Op
  - Using immediate value 33
- Step 4: Idle
- Step 5: Write result
Datapath Operation for I-Format \( (lw) \)

\[
lw \; \$t1, \; offset(\$t2)
\]

- Step 1: Fetch instruction from memory
- Step 2: Read register \$t2\ from register file
- Step 3: Do ALU op
- Step 4: Retrieve data from memory
- Step 5: Write contents into destination register
Datapath Operation for I-Format \((lw)\)

\(lw \ $t1, \ offset($t2)\)
Datapath Operation for I-Format \((sw)\)

\[sw \, \$t1, \, 33(\$t2)\]

- Step 1: Fetch instruction from memory
- Step 2: Read registers from register file
- Step 3: Do ALU op
- Step 4: Write data into data memory
- Step 5: Idle
Datapath Operation for I-Format \((sw)\)

\(sw \, $t1, \, offset($t2)\)
Datapath Operation for I-Format \((\text{beq})\)

\beq \texttt{$t1$, $t2$, offset}\)

- Step 1: Fetch instruction from memory
- Step 2: Read registers from register file
- Step 3: Do ALU op (both)
- Step 4: Idle
- Step 5: Use Zero signal to decide path
Datapath Operation for J-Format ($j$)

- Similar to branch, but computes PC differently
  - $j$ label
- Step 1: Fetch instruction from memory
- Step 2: Decode instruction result is $j$
- Step 3: Do ALU Op
  - Shift the label 26 bits left by 2 bits
  - Concatenate the upper 4 bits of PC +4 as the high-order bits (to get the complete memory address)
- Step 4: Idle
- Step 5: Store the calculated address into PC
Datapath Operation for J-Format

Lower-order 2 bits of a jump address are always 00 (Fig 5-29, p. 372)
Finalizing the Control

- Control function can be precisely defined
- We can create a separate truth table for each output, or we can build one large truth table for all the outputs
- Appendix C.2 has these details

<table>
<thead>
<tr>
<th>Instruction</th>
<th>RegDst</th>
<th>ALUSrc</th>
<th>Memto-Reg</th>
<th>RegWrit</th>
<th>MemRead</th>
<th>MemWrit</th>
<th>Branch</th>
<th>ALUOp1</th>
<th>ALUOp0</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-format</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>lw</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sw</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>beq</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Single Cycle Implementation

- Exercise: Calculate cycle time assuming negligible delays except:
  - memory (2ns)
  - ALU and adders (2ns)
  - Register file access (1ns)
Single Cycle Implementation

• **Exercise:**

  Assuming we have the following instruction mix:
  
  - Load 24%
  - Store 12%
  - ALU instructions 44%
  - Branch 18%
  - Jump 2%

  Calculate cycle time assuming negligible delays except for:
  
  - memory (2ns)
  - ALU and adders (2ns)
  - Register file access (1ns)
  - Note: For single-cycle instruction, CPI = 1

• **Answer:**

  - CPU execution time = Instruction count x CPI x Clock cycle time
    
    = Instruction count x 1 x Clock cycle time
    
    = Instruction count x Clock cycle time
Single Cycle Problems

• More complicated instructions like floating point
• Wasteful of area
• One Solution:
  – use a “smaller” cycle time
  – have different instructions take different numbers of cycles
  – a “multi-cycle” datapath
High-level of multicycle data path
Designing & Implementing Control

- Initial representation

- Sequencing Control

Logic Representation

- Implementation Technique

Finite State Diagram

Microprogram

Explicit next state

Function

Microprogram counter

+ dispatch ROMs

Logic equations

Truth tables

PLA

“hardwired control”

ROM

“microprogrammed control”
"Macroinstruction" Interpretation

User program plus Data
this can change!

one of these is mapped into one of these

AND micro-sequence
e.g., Fetch Calc Operand Addr
Fetch Operand(s)
Calculate Save Answer(s)
Multiprogramming & Microinstructions

• Can we design the control graphically?
  – Full MIPS instruction set contains > 100 instructions
  – An instruction can take (1-20) clock cycle
  – Control function is complex
  – Control unit might need > 1000’s states & 100’s different sequences
  – Specifying the control with graphical representation is cumbersome & difficult to understand

• Solution:
  – Use some ideas of programming to make it easier to understand
Terminology

• Microprogramming:
  – Designing the control as a program that implements machine instructions in terms of simpler microinstructions

• Micro-instruction:
  – A set of datapath signals that must be asserted in a given state
  – Usually stored in ROM or PLA (Appendixes B & C)
  – Has memory addresses
Terminology

• Executing a microinstruction:
  – Asserting the control signals specified by the microinstruction

• Inconsistent microinstruction
  – If the instruction requires that a given control signal be set to 2 different values

• Microinstructions are assumed to be executed sequentially, except for branching
Designing a Microinstruction Set

• Start with list of control signals
• Group like signals together that make sense
  – Creating “fields”
• Place fields in some logical order
  – ALU operation & ALU operands first
  – Microinstruction sequencing last
• Reuse fields
  – Signals that are never asserted simultaneously
Format of Microinstructions

• Should simplify the representation
  – Easy to write & understand

• Requirements:
  – A field to control the ALU
  – 3 fields determine ALU’s source/destination registers
  – Should prevent writing inconsistent microinstructions
    • Achieved by specifying non-overlapping set of control signals
    • Signals that that are never asserted simultaneously, can share the same field

• Micro-assemblers check consistency of microcode
Microprogram Control

- Three ways of deciding
  1. Sequential:
     - Increment address of current microinstruction
  2. Branch:
     - Begin execution of next MIPS instruction
     - Corresponds microinstruction state 0
  3. Dispatch:
     - Next microinstruction based on control unit input
     - Create a table (Dispatch table) containing the address of target microinstruction
     - Usually there is more than one dispatch table
## Microprogramming

<table>
<thead>
<tr>
<th>Label</th>
<th>ALU control</th>
<th>SRC1</th>
<th>SRC2</th>
<th>Register control</th>
<th>Memory</th>
<th>PCWrite control</th>
<th>Sequencing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch</td>
<td>Add</td>
<td>PC</td>
<td>4</td>
<td>Read PC</td>
<td>ALU</td>
<td>Seq</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Add</td>
<td>PC</td>
<td>Extshft</td>
<td>Read</td>
<td></td>
<td>Dispatch 1</td>
<td></td>
</tr>
<tr>
<td>Mem1</td>
<td>Add</td>
<td>A</td>
<td>Extend</td>
<td>Read ALU</td>
<td></td>
<td>Dispatch 2</td>
<td></td>
</tr>
<tr>
<td>LW2</td>
<td>Add</td>
<td>A</td>
<td>Extend</td>
<td>Read ALU</td>
<td></td>
<td>Seq</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Write MDR</td>
<td></td>
<td>Fetch</td>
<td></td>
</tr>
<tr>
<td>SW2</td>
<td>Func code</td>
<td>A</td>
<td>B</td>
<td>Write ALU</td>
<td></td>
<td>Fetch</td>
<td></td>
</tr>
<tr>
<td>Rformat1</td>
<td>Func code</td>
<td>A</td>
<td>B</td>
<td>Write ALU</td>
<td></td>
<td>Seq</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Fetch</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEQ1</td>
<td>Subt</td>
<td>A</td>
<td>B</td>
<td>ALUOut-cond</td>
<td></td>
<td>Fetch</td>
<td></td>
</tr>
<tr>
<td>JUMP1</td>
<td></td>
<td></td>
<td></td>
<td>Jump address</td>
<td></td>
<td>Fetch</td>
<td></td>
</tr>
</tbody>
</table>
Microprogramming Pros & Cons

- Pros:
  - Ease of design
  - Flexibility
    - Easy to adapt to changes in organization, timing
    - Can make changes late in design cycle
  - Can implement very powerful instruction sets
    - Just more control memory
  - Generality
    - Implement multiple instruction sets on one machine.
    - Can tailor instruction set to application.
  - Compatibility
    - Many organizations, same instruction set
Microprogramming Cons

• Cons:
  – Costly to implement
  – Slow