A Multicycle Data Path Implementation

Computer Science 104

Administrative

- Homework due Wednesday
- Midterm II Monday March 26
- Work on Projects
Outline of Today’s Lecture

- Review Finite State Machines
- Review Single Cycle Processor
- Introduction to the Concept of Multiple Cycle Processor
- Multiple Cycle Implementation of R-type Instructions
- Multiple Cycle Implementation of Or Immediate
- Multiple Cycle Implementation of Load and Store
- Putting it all Together
- Exceptions and Interrupts.

FSM State Diagram
Example: Traffic light Controller
Example: Traffic Light Controller

<table>
<thead>
<tr>
<th>I</th>
<th>S</th>
<th>NS</th>
<th>OT</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>01</td>
<td>01</td>
<td>012345</td>
</tr>
<tr>
<td>0-</td>
<td>00</td>
<td>00</td>
<td>100001</td>
</tr>
<tr>
<td>1-</td>
<td>00</td>
<td>01</td>
<td>100001</td>
</tr>
<tr>
<td>--</td>
<td>01</td>
<td>10</td>
<td>010001</td>
</tr>
<tr>
<td>--</td>
<td>10</td>
<td>10</td>
<td>001100</td>
</tr>
<tr>
<td>-1</td>
<td>10</td>
<td>11</td>
<td>001100</td>
</tr>
<tr>
<td>--</td>
<td>11</td>
<td>00</td>
<td>001010</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
NS1 &= S0'\cdot S1'\cdot I0 + S0\cdot S1\cdot I1 \\
     &= S1'\cdot (S0'\cdot I0 + S0\cdot I1) \\
NS0 &= S0'\cdot S1 + S0\cdot S1'\cdot I0' + S0\cdot S1\cdot I1 \\
     &= S0'\cdot S1 + S0\cdot S1' \\
OT0 &= S0'\cdot S1' \\
OT1 &= S0'\cdot S1 \\
OT2 &= S0\cdot S1' + S0\cdot S1 = S0 \\
OT3 &= S0\cdot S1' \\
OT4 &= S0\cdot S1 \\
OT5 &= S0'\cdot S1' + S0'\cdot S1 = S0'
\end{align*}
\]

Traffic Controller FSM implementation
A Single Cycle Processor

Instruction Fetch Unit
The Main Control

Drawbacks 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.
  Examples:
  • R-type instructions do not require data memory access
  • Jump does not require ALU operation nor data memory access
Multiple Cycle Processor

- 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

Multicycle Processor

- How should we design a multicycle processor?
- What are the smaller steps for each instruction?
- What if goal is to minimize HW resources?
The Five Steps of a Load Instruction

Instruction Fetch

Clk-to-Q

Instruction Memory Access Time

Delay through Control Logic

Register File Access Time

Data Memory Access Time

Reg Wr

The Cyclomatic Complexity of a Load Instruction

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clk</td>
<td>Clock signal</td>
</tr>
<tr>
<td>PC</td>
<td>Program Counter</td>
</tr>
<tr>
<td>Rs, Rt, Rd, Op, Func</td>
<td>Register File Access</td>
</tr>
<tr>
<td>ALUctr</td>
<td>ALU Control</td>
</tr>
<tr>
<td>ExtOp</td>
<td>Extension Operation</td>
</tr>
<tr>
<td>ALUSrc</td>
<td>ALU Source</td>
</tr>
<tr>
<td>RegWr</td>
<td>Register Write</td>
</tr>
<tr>
<td>busA</td>
<td>Bus A</td>
</tr>
<tr>
<td>busB</td>
<td>Bus B</td>
</tr>
<tr>
<td>Address</td>
<td>Address</td>
</tr>
<tr>
<td>busW</td>
<td>Bus W</td>
</tr>
</tbody>
</table>

Multiple Cycle Datapath

Multiple Cycle Datapath Diagram
Instruction Fetch Cycle: In the Beginning

- Every cycle begins right AFTER the clock tick:
  - \(\text{mem}[PC] \rightarrow \text{PC}<31:0> + 4\)

Q: What are control signal values?

Instruction Fetch Cycle: The End

- Every cycle ends AT the next clock tick (storage element updates):
  - \(\text{IR} \leftarrow \text{mem}[PC] \quad \text{PC}<31:0> \leftarrow \text{PC}<31:0> + 4\)
1: Instruction Fetch Cycle: Overall Picture

2: Register Fetch / Instruction Decode

Q: How to use ALU?
2: Register Fetch / Instruction Decode (Continue)

- \text{busA} \leftarrow \text{Reg}[rs] \text{; } \text{busB} \leftarrow \text{Reg}[rt] \text{;}
- \text{Target} \leftarrow \text{PC} + \text{SignExt(Imm16)}^4

Instruction Decode: We have \text{Beq}
3: Branch Completion

° if (busA == busB)
  • PC <- Target

2: Instruction Decode: We have a R-type!

° Next Cycle: R-type Execution
3: R-type Execution
° ALU Output ← busA op busB

4: R-type Completion
° R[rd] ← ALU Output
2: Instruction Decode: We have an Ori!

° Next Cycle: Ori Execution

ALUOutput <- busA or ZeroExt[Imm16]

3: Ori Execution
4: Ori Completion (keep control stable)

- Reg[rt] <- ALU output

   ... diagram showing control flow and data paths ...

2: Instruction Decode: We have a Memory Access!

- Next Cycle: Memory Address Calculation

   ... diagram showing control flow and data paths ...

3: Memory Address Calculation

° ALU output <- busA + SignExt[Imm16]

4: Memory Access for Store

° mem[ALU output] <- busB
4: Memory Access for Load

- Mem Dout <- mem[ALU output]

5: Write Back for Load

- Reg[rt] <- Mem Dout
Putting it all together: Multiple Cycle Datapath

Putting it all together: Control State Diagram
Undefined Instruction

- detected when no next state is defined from state Rfetch/Decode for the op value.
- We handle this exception by defining the next state value for all op values other than lw, sw, 0 (R-type), jump, beq, and ori as new state.
- Shown symbolically using “other” to indicate that the op field does not match any of the opcodes that label arcs out of state Rfetch/Decode.

Arithmetic overflow

- logic in the ALU to detect overflow
- Overflow is provided as an output from the ALU. This signal is used in the modified finite state machine to specify an additional possible next state for state Rfinish.
Extra States to Handle Exceptions

- Some problems could occur in the way the exceptions are handled.
- Example:
  - in the case of arithmetic overflow, the instruction causing the overflow completes writing its result, because the overflow branch is in the state when the write completes.
  - However, the architecture may define the instruction as having no effect if the instruction causes an exception; MIPS specifies this.
- When we get to virtual memory we will see that certain classes of exceptions must prevent the instruction from changing the machine state.
- This gets complex and potentially limits performance => this is why exceptions are hard
Control design

- First full scale stored program computer: EDSAC 1 in 1950, led by Prof. Maurice Wilkes at Cambridge
- After completed, decided that Memory design was regular, ALU design was regular, could purchase Input and Output devices (telegraph), but control was a mess.
- Why not have a matrix for control, and make specifying control more like programming memory?
- PLA is a good compromise between regularity, density and speed.

Initial Representation: Finite State Diagram
Sequencing Control: Explicit Next State Function

- Next state number is encoded just like datapath controls

Implementation Technique: Programmed Logic Arrays

- Each output line the logical OR of logical AND of input lines or their complement: AND minterms specified in top AND plane, OR sums specified in bottom OR plane

- Input values:
  - lw = 100011
  - sw = 101011
  - R = 000000
  - ori = 001011
  - beq = 000100
  - jmp = 000010
Summary

- Disadvantages of the Single Cycle Processor
  - Long cycle time
  - Cycle time is too long for all instructions except the Load

- Multiple Cycle Processor:
  - Divide the instructions into smaller steps
  - Execute each step (instead of the entire instruction) in one cycle
  - Multiple Cycle Processor executes each instruction in multiple clock cycles
  - Finite State Machine control