27 March

9 to go!

Instruction Execution
Five Execution Steps

Instruction Fetch

Instruction Decode and Register Fetch

Execution, Memory Address Computation, or Branch Completion

Memory Access or R-type instruction completion

Memory Read Completion

INSTRUCTIONS TAKE FROM 3 - 5 CYCLES!

A FSM looks at the op-code to determine how many...
Step 1: Instruction Fetch

Use PC to get instruction and put it in the Instruction Register. Increment the PC by 4 and put the result back in the PC. Can be described succinctly using RTL "Register-Transfer Language"

\[
\begin{align*}
IR & = \text{Memory}[PC]; \quad IR \text{ is "Instruction Register"} \\
PC & = PC + 4;
\end{align*}
\]

What is the advantage of updating the PC now?
Step 2: Instruction Decode and Register Fetch

Read registers rs and rt in case we need them
Compute the branch address in case the instruction is a branch

RTL:

\[
\begin{align*}
A &= \text{Reg}[\text{IR}[25-21]]; \\
B &= \text{Reg}[\text{IR}[20-16]]; \\
\text{ALUOut} &= \text{PC} + (\text{sign-extend}(\text{IR}[15-0]) \ll 2); \\
\end{align*}
\]

We aren't setting any control lines based on the instruction type (we are busy "decoding" it in our control logic)
Step 3 (instruction dependent)

ALU is performing one of three functions, based on instruction type

Memory Reference:
\[ ALUOut = A + \text{sign-extend}(IR[15-0]) ; \]

R-type:
\[ ALUOut = A \text{ op } B ; \]

Branch:
\[ \text{if } (A==B) \text{ } PC = ALUOut; \]
Step 4 (R-type or memory-access)

Loads and stores access memory

$$MDR = Memory[ALUOut]; \quad MDR \text{ is Memory Data Register}$$

or

$$Memory[ALUOut] = B;$$

R-type instructions finish

$$Reg[IR[15-11]] = ALUOut;$$
Step 5 Memory Read Completion

Reg[IR[20–16]] = MDR;
### Summary:

<table>
<thead>
<tr>
<th>Step name</th>
<th>Action for R-type instructions</th>
<th>Action for memory-reference instructions</th>
<th>Action for branches</th>
<th>Action for jumps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction fetch</td>
<td></td>
<td>IR = Memory[PC]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PC = PC + 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instruction decode/</td>
<td>A = Reg [IR[25-21]]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>register fetch</td>
<td>B = Reg [IR[20-16]]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ALUOut = PC + (sign-extend (IR[15-0]) &lt;&lt; 2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution, address</td>
<td>ALUOut = A op B</td>
<td>ALUOut = A + sign-extend (IR[15-0])</td>
<td>if (A == B) then</td>
<td>PC = PC [31-28] II</td>
</tr>
<tr>
<td>computation, branch/</td>
<td></td>
<td></td>
<td>PC = ALUOut</td>
<td></td>
</tr>
<tr>
<td>jump completion</td>
<td></td>
<td></td>
<td>(IR[25-0] &lt;&lt; 2)</td>
<td></td>
</tr>
<tr>
<td>Memory access or R-type</td>
<td>Reg [IR[15-11]] = ALUOut</td>
<td>Load: MDR = Memory[ALUOut] or</td>
<td></td>
<td></td>
</tr>
<tr>
<td>completion</td>
<td></td>
<td>Store: Memory [ALUOut] = B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory read completion</td>
<td></td>
<td>Load: Reg[IR[20-16]] = MDR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Data Path with Control (5.28)
FSM Control (5.31)

Start

Instruction fetch/decode and register fetch (Figure 5.32)

Memory access instructions (Figure 5.33)

R-type instructions (Figure 5.34)

Branch instruction (Figure 5.35)

Jump instruction (Figure 5.36)
IFetch and Decode (5.32)

- MemRead
- ALUSrcA = 0
- lorD = 0
- IRWrite
- ALUSrcB = 01
- ALUOp = 00
- PCWrite
- PCSource = 00

Instruction fetch:
- 0
- Start

Instruction decode/ Register fetch:
- 1
- ALUSrcA = 0
- ALUSrcB = 11
- ALUOp = 00

- (Op = ‘LW’) or (Op = ‘SW’)
- (Op = R-type)
- (Op = BEO)
- (Op = J)

Memory-reference FSM (Figure 5.33)
R-type FSM (Figure 5.34)
Branch FSM (Figure 5.35)
Jump FSM (Figure 5.36)
Memory Ref (5.33)

From state 1

(Op = 'LW') or (Op = 'SW')

Memory address computation

ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00

Memory access

MemRead
lorD = 1

Memory access

MemWrite
lorD = 1

Memory read completion step

RegWrite
MemtoReg = 1
RegDst = 0

To state 0
(Figure 5.32)
R-type (5.34)

From state 1

Op = R-Type

Execution

ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10

R-type completion

RegDst = 1
RegWrite
MemtoReg = 0

To state 0
(Figure 5.32)
Branch (5.35)

From state 1

(Op = 'BEQ')

Branch completion

8

ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01

To state 0
(Figure 5.32)
Jump (5.36)

From state 1
(Op = 'J')

Jump completion

9
PCWrite
PCSource = 10

To state 0
(Figure 5.32)
Full State Diagram

Instruction fetch

Instruction decode/register fetch

Start

MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00

ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00

MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00

ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00

(Op = 'LW') or (Op = 'SW')

Memory address computation

Execution

Branch completion

Jump completion

RegDst = 1
RegWrite
MemtoReg = 0

ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10

(Op = 'R-type')

Memory access

ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01

(Op = 'EQ')

R-type completion

MemWrite
IorD = 1

ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10

(Op = 'J')

RegDst = 1
RegWrite
MemtoReg = 0

ALUSrcA = 1
ALUSrcB = 00
ALUOp = 00

Memory read completion step

Comp 411 – Spring 2007