Homework for Wednesday April 5, 2000

1. Convert the RISCEE 1 Architecture into a pipeline Architecture (like Figure 6.30) (showing the number data and control bits).

2. Build the control line table (like Figure 6.28) for the RISCEE3 pipeline architecture for RISCEE1 instructions: (addi, subi, load, store, beq, jmp, jal).

3. Homework 6.23, p534
4. Homework 6.24, p534
Lecture

Computer Architecture and Engineering

Pipeline Control,

Data Hazards

and Branch Hazards
Models

**Single-cycle model** (non-overlapping)
- Each instruction executes in a single cycle
- Every instruction and clock-cycle must be stretched to the slowest instruction (p.438)

**Multi-cycle model** (non-overlapping)
- Each instruction executes in variable number of cycles
- The clock-cycle must be stretched to the slowest step
- Ability to share functional units within the execution of a single instruction

**Pipeline model** (overlapping)
- Each instruction executes in several cycles
- The clock-cycle must be stretched to the slowest step
- Gains efficiency by overlapping the execution of multiple instructions, increasing hardware utilization. (p. 377)
Overhead

**Single-cycle model**
- 8 ns Clock (125 MHz), (non-overlapping)
- 1 ALU + 2 adders
- 0 Muxes
- 0 Datapath Register bits (Flip-Flops)

**Multi-cycle model**
- 2 ns Clock (500 MHz), (non-overlapping)
- 1 ALU + Controller
- 5 Muxes
- 160 Datapath Register bits (Flip-Flops)

**Pipeline model**
- 2 ns Clock (500 MHz), (overlapping)
- 2 ALU + Controller
- 4 Muxes
- 373 Datapath Register bits (Flip-Flops)
Pipeline Designing

- **What makes it easy**
  - all instructions are the same length
  - just a few instruction formats
  - memory operands appear only in loads and stores

- **What makes it hard?**
  - structural hazards: suppose we had only one memory
  - control hazards: need to worry about branch instructions
  - data hazards: an instruction depends on a previous instruction
Pipeline Hazards

**Pipeline hazards**
- Solution #1 always works: stall, delay & procrastinate!

**Structural Hazards** (i.e. fetching same memory bank)
- Solution #2: partition architecture

**Control Hazards** (i.e. branching)
- Solution #1: stall! but decreases throughput
- Solution #2: guess and back-track
- Solution #3: delayed decision: delay branch & fill slot

**Data Hazards** (i.e. register dependencies)
- Worst case situation
- Solution #2: re-order instructions
- Solution #3: forwarding or bypassing: delayed load
Ideal Pipelining

Assume instructions are completely independent!

<table>
<thead>
<tr>
<th>IF</th>
<th>DCD</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>DCD</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>IF</td>
<td>DCD</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
<tr>
<td>IF</td>
<td>DCD</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
</tr>
</tbody>
</table>

Maximum Speedup $\leq$ Number of stages
Speedup $\leq$ Time for unpipedined operation
\[
\text{Speedup} \leq \frac{\text{Time for longest stage}}{\text{Time for unpipedined operation}}
\]
Recap: Graphically Representing Pipelines

Program execution order (in instructions)

lw $10, 20($1)
sub $11, $2, $3

Time (in clock cycles)

CC 1  CC 2  CC 3  CC 4  CC 5  CC 6

IM  Reg  ALU  DM  Reg

IM  Reg  ALU  DM  Reg
Figure 6.25
Pipeline Control: Controlpath Register bits

Figure 6.29
Pipeline Control: Controlpath Register bits

Figure 5.20, Single Cycle

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Reg Dst</th>
<th>ALU Src</th>
<th>Mem Reg</th>
<th>Reg Wrt</th>
<th>Mem Red</th>
<th>Mem Wrt</th>
<th>Branch</th>
<th>ALU op1</th>
<th>ALU op0</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>1</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>

Figure 6.28

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Reg Dst</th>
<th>ALU Op1</th>
<th>ALU Op0</th>
<th>ALU Src</th>
<th>Bra- nch</th>
<th>Mem Red</th>
<th>Mem Wrt</th>
<th>Reg Wrt</th>
<th>Mem Reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-format</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>lw</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>sw</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>beq</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>
Overhead

**Single-cycle model**
- 8 ns Clock (125 MHz), *(non-overlapping)*
- 1 ALU + 2 adders
- 0 Muxes
- 0 Datapath Register bits (Flip-Flops)

**Multi-cycle model**
- 2 ns Clock (500 MHz), *(non-overlapping)*
- 1 ALU + Controller
- 5 Muxes
- 160 Datapath Register bits (Flip-Flops)

**Pipeline model**
- 2 ns Clock (500 MHz), *(overlapping)*
- 2 ALU + Controller
- 4 Muxes
- 373 Datapath + 16 Controlpath Register bits (Flip-Flops)
Pipeline Datapath and Controlpath

Figure 6.30
Figure 6.31a

IF: lw $10, 20($1)
ID: before<1>
EX: before<2>
MEM: before<3>
WB: before<4>

Clock 1
Figure 6.32
Figure 6.32b

IF: or $13, $6, $7
ID: and $12, $2, $3
EX: sub $11, . . .
MEM: lw $10, . . .
WB: before<1>
Figure 6.33

IF: add $14, $8, $9
ID: or $13, $6, $7
EX: and $12, . . .
MEM: sub $11, . . .
WB: lw $10, . . .

Clock 5

IF: after<1>
ID: add $14, $8, $9
EX: or $13, . . .
MEM: and $12, . . .
WB: sub $11, . . .

Clock 6
Figure 6.34

Clock 7

IF: after<2>  ID: after<1>  EX: add $14, . . .  MEM: or $13, . . .  WB: and $12, . . .

Clock 8

Figure 6.35


Address memory
Instruction
Add
Control
RegWrite
Read register 1
Read register 2
Read data 1
Read data 2
Write register
Write data
Instruction
Instruction
Instruction
Instruction
Sign extend
0 Mux 1
Instruction 20–16
Instruction 15–11
Instruction 15–0
Zero ALU result
Shift left 2
Add result
ALUSrc
ALUOp
RegDst
MEMWB
WB
Mux
Write data
Data memory
Address
Read data
Write memory
MemWrite
MemRead
Branch
MemWB
Clock 9
# Pipeline Datapath and Controlpath

<table>
<thead>
<tr>
<th>Clock</th>
<th>IF/ID</th>
<th>ID/EX</th>
<th>EX/MEM</th>
<th>MEM/WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&lt;0,?&gt;</td>
<td>&lt;?,?,?,?,?&gt;</td>
<td>&lt;?,?,?,?&gt;</td>
<td>&lt;?,?,?&gt;</td>
</tr>
<tr>
<td>1</td>
<td>&lt;4,lw $10,20($1)&gt;</td>
<td>&lt;0,?,?,?,?&gt;</td>
<td>&lt;?,?,?,?&gt;</td>
<td>&lt;?,?,?&gt;</td>
</tr>
<tr>
<td>2</td>
<td>&lt;8,sub $11,$2,$3&gt;</td>
<td>&lt;4,C$1,C$10,20,$10,X&gt;</td>
<td>&lt;0,?,?,?,?&gt;</td>
<td>&lt;?,?,?&gt;</td>
</tr>
<tr>
<td>3</td>
<td>&lt;12,and $12,$4,$5&gt;</td>
<td>&lt;8,C$2,C$3,X,$3,$11&gt;</td>
<td>&lt;84,0,23,8,$10&gt;</td>
<td>&lt;?,?,?&gt;</td>
</tr>
<tr>
<td>4</td>
<td>&lt;16,or $13,$6,$7&gt;</td>
<td>&lt;12,C$4,C$5,X,$5,$12&gt;</td>
<td>&lt;X,1,0,4,$11&gt;</td>
<td>&lt;9,23,$10&gt;</td>
</tr>
<tr>
<td>5</td>
<td>&lt;20,add $14,$8,$9&gt;</td>
<td>&lt;16,C$6,C$7,X,$7,$13&gt;</td>
<td>&lt;X,0,1,7,$12&gt;</td>
<td>&lt;9,23,$10&gt;</td>
</tr>
</tbody>
</table>

Contents of Register 1 = C$1 = 3; C$2=4; C$3=4; C$4=6; C$5=7; C$10=8; ... Memory[23]=9;
Formats: add $rd,$rs=A,$rt=B; lw $rt=B,@($rs=A)