Review Problem 20

- Add the instruction “MUL Rd, Rn, Rm” to the add/sub datapath.
Store RTL

Store Instruction: STUR Rd, [Rn, DAddr9]

Instruction = MEM(Rn) 

Addr = Reg[Rn] + Sign Extend (DAddr9); 

Mem [Addr] = Reg[Rd]; 

PC = PC + 4;
Datapath + Store

\[
\text{Addr} = \text{Reg}[Rd] + \text{SEC (DAddr 9)} \\
\text{Mem}[\text{Add} J] = \text{Reg}[Rd]
\]
Branch RTL

Branch Instruction: B BrAddr26

\[ \text{Instruction} = \text{MEM}[\text{PC}] \]

\[ \text{PC}^e = \text{PC} + \text{SEC}(\text{BrAddr26}) \ll 2 \]
Datapath + Branch

Instruction = MEM[PC]
PC = PC + SE(Br.Addr 26) << 2
Conditional Branch RTL

Conditional Branch Instruction: CBZ Rd, CondAddr19

\[
\text{Instruction} = \text{MEM}[Rd]; \\
\text{Cond} = (\text{Reg}[Rd] == 0); \\
\text{if} (\text{Cond}) \\
\quad \text{PC} = \text{PC} + \text{SignExt}(\text{CondAddr19}) \ll 2; \\
\text{else} \\
\quad \text{PC} = \text{PC} + 4;
\]

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Opcode | CondAddr19 | Rd |
Datapath + Conditional Branch

\[
\text{Cond} = (\text{Reg}[RdJ] == 0)
\]

\[
\begin{align*}
\text{if}(\text{Cond}) \\
\quad \text{PC} &= \text{PC} + \text{SE} (	ext{Cond Addr} / 19) /2 \\
\text{else} \\
\quad \text{PC} &= \text{PC} + 4
\end{align*}
\]

Controlled by the zero flags
Control

Identify control points for pieces of datapath
  Instruction Fetch Unit
  ALU
  Memories
  Datapath muxes
  Etc.

Use RTL for determine per-instruction control assignments
Review Problem 22

- Develop a single-cycle CPU that can do LDUR and STUR (only). Make it as simple as possible.
### Control Signals

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ADD</td>
<td>SUB</td>
<td>LDUR</td>
<td>STUR</td>
<td>B</td>
<td>CBZ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reg2Loc</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALU Src</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mem To Reg</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RegWrite</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MemWrite</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Br Taken</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uncall Br</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALUOp</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>testB</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ADD Control

Instruction = Mem[PC];
Reg[Rd] = Reg[Rn] + Reg[Rm];
PC = PC + 4;
SUB Control

Instruction = Mem[PC];
Reg[Rd] = Reg[Rn] - Reg[Rm];
PC = PC + 4;
LDUR Control

Instruction = Mem[PC];
Addr = Reg[Rn] + SignExtend(DAddr9);
Reg[Rd] = Mem[Addr];
PC = PC + 4;
STUR Control

Instruction = Mem[PC];
Addr = Reg[Rn] + SignExtend(DAddr9);
Mem[Addr] = Reg[Rd];
PC = PC + 4;
B Control

Instruction = Mem[PC];
PC = PC + SignExtend(BrAddr26) <<2;

UncondBr
CondAddr19
BrAddr26
SE
<<2
PC
Adder
Adder
Adder
BrTaken

Address
Instruction
Memory

Rd Rd
Rm
Rn
0 1
Reg2Loc

Aw Ab Aa Da
Dw
RegFile Db

RegWrite
DAddr9
SE
ALUSrc
ALUOp

Zero
MemWrite
MemToReg

WrEn Addr
Din Dout
Data
Memory
CBZ Control

Instruction = Mem[PC];
Cond = (Reg[Rd] == 0);
if (Cond)
  PC = PC + SE(CondAddr19)<<2;
else
  PC = PC + 4;
Advanced: Exceptions

Exception = unusual event in processor
    Arithmetic overflow, divide by zero, ...
    Call an undefined instruction
    Hardware failure
    I/O device request (called an "interrupt")

Approaches
    Make software test for exceptional events when they may occur ("polling")
    Have hardware detect these events & react:
        Save state (Exception Program Counter, protect the GPRs, note cause)
        Call Operating System
            If (undef_instr) PC = C0000000
            If (overflow) PC = C0000020
            If (I/O) PC = C0000040
            ...

System Exception Handler

return from exception
Review Problem 24

- What mods are needed to support branch register:
  - PC = Reg[Rd]
Performance of Single-Cycle Machine

<table>
<thead>
<tr>
<th>CPI?</th>
<th>ADD, SUB</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>Instr. Memory</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LDUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CBZ</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
</tr>
</tbody>
</table>
Reducing Cycle Time

Cut combinational dependency graph and insert register / latch
Do same work in two fast cycles, rather than one slow one
Pipelined Processor Overview

Divide datapath into multiple stages

IF
Instruction Fetch

RF
Register Fetch

EX
Execute

MEM
Data Memory

WB
Writeback

Instr. Memory

Register File

Data Memory

Register File

PC

Cancel
Pipelining

Readings: 4.5-4.8

Example: Doing the laundry

Ann, Brian, Cathy, & Dave

each have one load of clothes to wash, dry, and fold

Washer takes 30 minutes

Dryer takes 40 minutes

“Folder” takes 20 minutes
Sequential laundry takes 6 hours for 4 loads

If they learned pipelining, how long would laundry take?
Pipelined Laundry: Start work ASAP

Time

6 PM  7  8  9  10  11  Midnight

30  40  40  40  40  20

Task Order

A  B  C  D

Pipelined laundry takes 3.5 hours for 4 loads

Resource Usage

Pipeline 5:11

Steady State

Pipeline Drain
Pipelining Lessons

6 PM 7 8 9

Time

30 40 40 40 40 20

Pipelining doesn’t help latency of single task, it helps throughput of entire workload

Pipeline rate limited by slowest pipeline stage

Multiple tasks operating simultaneously using different resources

Potential speedup = Number pipe stages

Unbalanced lengths of pipe stages reduces speedup

Time to “fill” pipeline and time to “drain” it reduces speedup

Stall for Dependences
Pipelined Execution

Now we just have to make it work
Single Cycle vs. Pipeline

Single Cycle Implementation:

Load

Store \(\rightarrow\) Waste

Cycle 1 \(\rightarrow\) Cycle 2 \(\rightarrow\) Cycle 3 \(\rightarrow\) Cycle 4 \(\rightarrow\) Cycle 5

Cycle 6 \(\rightarrow\) Cycle 7 \(\rightarrow\) Cycle 8 \(\rightarrow\) Cycle 9 \(\rightarrow\) Cycle 10

Pipeline Implementation:

Load

Ifetch | Reg | Exec | Mem | Wr

Store

Ifetch | Reg | Exec | Mem | Wr

R-type

Ifetch | Reg | Exec | Mem | Wr
Why Pipeline?

Suppose we execute 100 instructions

Single Cycle Machine
45 ns/cycle \times 1 \text{ CPI} \times 100 \text{ inst} = \frac{4,500}{\text{ns}}

Ideal pipelined machine
10 \text{ ns/cycle} \times (1 \text{ CPI} \times 100 \text{ inst} + 4 \text{ cycle drain}) = \frac{1,040}{\text{ns}}
CPI for Pipelined Processors

Ideal pipelined machine

10 ns/cycle x (1 CPI x 100 inst + 4 cycle drain) = ___ ns

CPI in pipelined processor is “issue rate”. Ignore fill/drain, ignore latency.

Example: A processor wastes 2 cycles after every branch, and 1 after every load, during which it cannot issue a new instruction. If a program has 10% branches and 30% loads, what is the CPI on this program?

<table>
<thead>
<tr>
<th>Cycles</th>
<th>% instr</th>
</tr>
</thead>
<tbody>
<tr>
<td>BR</td>
<td>1 + 2 = 3 x 10% = 0.3</td>
</tr>
<tr>
<td>LOAD</td>
<td>1 + 1 = 2 x 30% = 0.6</td>
</tr>
<tr>
<td>OTHER</td>
<td>1     x 60% = 0.6</td>
</tr>
</tbody>
</table>

1.5
Pipelined Datapath

Divide datapath into multiple pipeline stages
Pipelined Control

The Main Control generates the control signals during Reg/Dec
Control signals for Exec (ALUOp, ALUSrc, ...) are used 1 cycle later
Control signals for Mem (MemWE, Mem2Reg, ...) are used 2 cycles later
Control signals for Wr (RegWE, ...) are used 3 cycles later
Can pipelining get us into trouble?

Yes: Pipeline Hazards

**structural hazards**: attempt to use the same resource two different ways at the same time

E.g., combined washer/dryer would be a structural hazard or folder busy doing something else (watching TV)

**data hazards**: attempt to use item before it is ready

E.g., one sock of pair in dryer and one in washer; can’t fold until get sock from washer through dryer

instruction depends on result of prior instruction still in the pipeline

**control hazards**: attempt to make decision before condition evaluated

E.g., washing football uniforms and need to get proper detergent level; need to see after dryer before next load in branch instructions

Can always resolve hazards by **waiting**

pipeline control must detect the hazard
take action (or delay action) to resolve hazards