Review Problem 22

- Develop a single-cycle CPU that can do LDUR and STUR (only). Make it as simple as possible.
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
### Performance of Single-Cycle Machine

CPI? = 1.0

<table>
<thead>
<tr>
<th>Operation</th>
<th>Stages</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD, SUB</td>
<td>PC, Instr. Memory, Reg Read, ALU, Reg Setup</td>
</tr>
<tr>
<td>LDUR</td>
<td>PC, Instr. Memory, Reg Read, ALU, Data Memory, Reg Setup</td>
</tr>
<tr>
<td>STUR</td>
<td>PC, Instr. Memory, Reg Read, ALU, Data Memory</td>
</tr>
<tr>
<td>CBZ</td>
<td>PC, Instr. Memory, Reg Read, ALU, Reg Setup</td>
</tr>
<tr>
<td>B</td>
<td>PC, Instr. Memory, Adder, Reg Setup</td>
</tr>
</tbody>
</table>

Each period is a clock.
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

PC

Instr. Memory

Register File

Data Memory

Register File
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

Sequential laundry takes 6 hours for 4 loads
If they learned pipelining, how long would laundry take?
Pipelined Laundry: Start work ASAP

Pipelined laundry takes 3.5 hours for 4 loads
Pipelining Lessons

Pipe time: Fill time + Cycle time x jobs

6 PM 7 8 9

Time

Task

Order

30 40 40 40 20

30 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

40 min x # of jobs

Drain Time
Pipelined Execution

Time

Program Flow

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

Single Cycle Implementation:

- Load
- Store
- Waste

Cycle 1 | Cycle 2 | Cycle 3 | Cycle 4 | Cycle 5 | Cycle 6 | Cycle 7 | Cycle 8 | Cycle 9 | Cycle 10

Pipeline Implementation:

<table>
<thead>
<tr>
<th>Load</th>
<th>Ifetch</th>
<th>Reg</th>
<th>Exec</th>
<th>Mem</th>
<th>Wr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store</td>
<td>Ifetch</td>
<td>Reg</td>
<td>Exec</td>
<td>Mem</td>
<td>Wr</td>
</tr>
<tr>
<td></td>
<td>R-type</td>
<td>Ifetch</td>
<td>Reg</td>
<td>Exec</td>
<td>Mem</td>
</tr>
</tbody>
</table>
Why Pipeline?

Suppose we execute 100 instructions

Single Cycle Machine
45 ns/cycle \times 1 \text{ CPI} \times 100 \text{ inst} = \underline{4500} \text{ ns}

Ideal pipelined machine
10 \text{ ns/cycle} \times (1 \text{ CPI} \times 100 \text{ inst} + 4 \text{ cycle drain}) = \underline{1040} \text{ ns}
CPI for Pipelined Processors

Ideal pipelined machine

10 ns/cycle \times (1 \text{ CPI} \times 100 \text{ inst} + 4 \text{ cycle drain}) = \text{____ 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?