Homework Assignment 2

CS/ECE 6810: Computer Architecture
September 26,2018
Name: Jake Pitkin
UID: u0891770

ILP and Branch Prediction

Due Date: October 03, 2018.
120 points

1. **Multi-cycle Instructions.** A pipelined architecture comprises instruction fetch (IF), instruction decode (ID), register read (RR), execute (EX), and write-back (WB) stages.

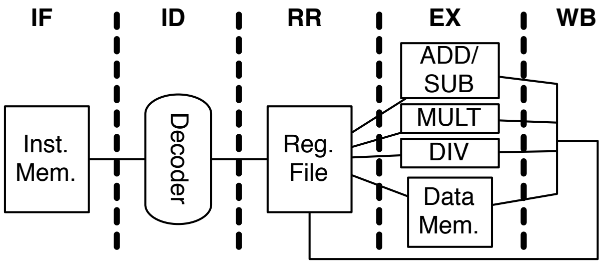

Figure 1: Pipelined core architecture.

Except EX, each stage requires one clock cycle to complete. The EX stage includes 4 functional units that perform memory and ALU operations—e.g., ADD, SUB, MULT, DIV, load, and store. The table below shows the latency of each operation in terms of clock cycles. The architecture implements forwarding paths from the EX/WB pipeline registers to the EX input. Moreover, the register file may be bypassed during the WB stage to send the register value to EX.

	ADD	SUB	MULT	DIV	Load	Store
Latency	1	1	3	7	2	1

i Identify all structural and data hazards in the following code. **(10 points)**

```
Load F6, 20(R5)
Load F2, 28(R5)
MUL F0, F2, F4
SUB F8, F6, F3
DIV F10, F0, F6
ADD F6, F8, F2
Store F8, 50(R5)
```

Below is a table of all data hazards present in the above list of instructions. *Instruction 1* is the instruction that comes first temporally which is followed by *Instruction 2*. *Hazard Type* indicates what type of hazard is detected and *Register* is the register causing the hazard.

Instruction 1	**Instruction 2**	**Hazard Type**	**Register**
Load F2, 28(R5)	MUL F0, F2, F4	RAW	F2
Load F6, 20(R5)	SUB F8, F6, F3	RAW	F6
MUL F0, F2, F4	DIV F10, F0, F6	RAW	F0
Load F6, 20(R5)	DIV F10, F0, F6	RAW	F6
SUB F8, F6, F3	ADD F6, F8, F2	RAW	F8
Load F2, 28(R5)	ADD F6, F8, F2	RAW	F2
SUB F8, F6, F3	Store F8, 50(R5)	RAW	F8
SUB F8, F6, F3	ADD F6, F8, F2	WAR	F6
DIV F10, F0, F6	ADD F6, F8, F2	WAR	F6
Load F6, 20(R5)	ADD F6, F8, F2	WAW	F6

My notes for the above data hazards: I listed all hazards including the hazards that didn't end up being an issue in part ii.

Structural hazards: The two successive load instructions will cause a structural hazard on the EX Data Mem as Load takes two cycles the second load will have to wait for the EX Data Mem resource.

ii Create a timing diagram for the code showing the execution of the code in time (clock cycles). **(20 points)**

Instruction	**Cycle 1**	**Cycle 2**	**Cycle 3**	**Cycle 4**	**Cycle 5**	**Cycle 6**
Load F6, 20(R5)	IF	ID	RR	EX Data	EX Data	WB
Load F2, 28(R5)	-	IF	ID	RR	STALL	EX Data
MUL F0, F2, F4	-	-	IF	ID	RR	STALL
SUB F8, F6, F3	-	-	-	IF	ID	RR
DIV F10, F0, F6	-	-	-	-	IF	ID
ADD F6, F8, F2	-	-	-	-	-	IF
Store F8, 50(R5)	-	-	-	-	-	-

Instruction	**Cycle 07**	**Cycle 08**	**Cycle 09**	**Cycle 10**	**Cycle 11**	**Cycle 12**
Load F6, 20(R5)						
Load F2, 28(R5)	EX Data	WB				
MUL F0, F2, F4	STALL	EX MULT	EX MULT	EX MULT	WB	
SUB F8, F6, F3	EX SUB	STALL	STALL	STALL	STALL	WB
DIV F10, F0, F6	RR	STALL	STALL	STALL	EX DIV	EX DIV
ADD F6, F8, F2	ID	RR	EX ADD	STALL	STALL	STALL
Store F8, 50(R5)	IF	ID	RR	EX Data	STALL	STALL

Instruction	**Cycle 13**	**Cycle 14**	**Cycle 15**	**Cycle 16**	**Cycle 17**	**Cycle 18**
Load F6, 20(R5)						
Load F2, 28(R5)						
MUL F0, F2, F4						
SUB F8, F6, F3						
DIV F10, F0, F6	EX DIV	EX DIV	EX DIV	EX DIV	EX DIV	WB
ADD F6, F8, F2	STALL	STALL	STALL	STALL	STALL	STALL
Store F8, 50(R5)	STALL	STALL	STALL	STALL	STALL	STALL

Instruction	**Cycle 19**	**Cycle 20**	**Cycle 21**	**Cycle 22**	**Cycle 23**	**Cycle 24**
Load F6, 20(R5)						
Load F2, 28(R5)						
MUL F0, F2, F4						
SUB F8, F6, F3						
DIV F10, F0, F6						
ADD F6, F8, F2	WB					
Store F8, 50(R5)	STALL	WB				

My Notes for the above timelne:

- Pipeline registers contain a FIFO queue allowing them to contain a queue of multiple waiting values.
- Stalls are used to maintain the WB order of the instructions even if there is no hazard.
- Instruction MUL F0, F2, F4 will have its register read of F2 overwritten in cycle 8 by a forward from the EX/WB pipeline.
- Instruction ADD F6, F8, F2 will have it's register read of F8 overwritten in cycle 9 by a forward from the EX/WB pipeline.
- Instruction Store F8, 50(R5) will have its register read of F8 overwritten in cycle 10 by a forward from the EX/WB pipeline.
- Instruction DIV F10, F0, F6 will have its register read of F0 overwritten in cycle 11 by a forward from the EX/WB pipeline.

2. **Points of Production and Consumption.** Consider an unpipelined processor where it takes 36 ns to go through the circuits and 0.5 ns for the latch overhead. Assume that the point of production and point of consumption in the unpipelined processor are separated by 12ns. Assume that half the instructions do not introduce a data hazard and half the instructions depend on their preceding instruction.

   i What is the maximum throughput of the unpipelined architecture in instructions per second (IPS). **(10 points)**

   The IPS of an unpipelined architecture is given by IPS = IPC * 1/CT. The architecture is unpipelined so IPC = 1 and the CT is given by 36 ns + 0.5 ns.

$$IPS = IPC * \frac{1}{CT}$$
$$IPS = 1 * \frac{1}{36.5 * 10^{-9}}$$
$$IPS = 2.7397 * 10^{7}$$

ii We build a 12-stage pipelined architecture for the processor. Please compute the percentages of increase/decrease in throughput for the pipelined architecture compared to unpipelined process. **(10 points)**

To compute the change in throughput we must first compute the new IPS. To calculate this we need the new IPC and the new CT.
We know the circuits take 36ns so with a 12-stage pipelined architecture each stage will take 3ns without the latch. This means the point of production and point of consumption is separated by 12ns/3ns or 4 pipeline stages.
If this hazard occurs half the time, we know that half our instructions will take one cycle while the other half take four. This means we will finish two instructions per five cycles so IPC = 2/5 = 0.4.
CT will simply be the length of a stage plus the latch time so CT = 3.5 ns.

$$IPS_{new} = IPC_{new} * \frac{1}{CT_{new}}$$
$$IPS_{new} = 0.4 * \frac{1}{3.5 * 10^{-9}}$$
$$IPS_{new} = 1.1429 * 10^{8}$$

Using this we can compute the change in throughput:

3. **Software Optimization.** Below are examples of a C and assembly codes for a **for**-loop. Notice that `i` is an 8-byte long integer. Register names indicate if floating point (F) or integer (R) operations are necessary for instructions.

Source code

```
for (i = 125; i > 0; i--) {
    x[i] = s * y[i] + z;
}
```

Assembly code

```
       ADDI R1, R0, #1000
       JMP Chck
Loop:  Load F1, 0(R1)
       MUL F3, F1, F2
       ADD F5, F3, F4
       Store F5, 20000(R1)
       ADDI R1, R1, #-8
Chck:  BNEQ R1, R0, Loop
```

Consider a five-stage scalar pipeline with multi-cycle functional units for floating-point and integer operations at the EX stage. Assume the following delays between dependent (producer-consumer) instructions:
(a) Load feeding any instruction: 1 stall cycle
(b) FP MULT/ADD feeding store: 4 stall cycles
(c) Integer ADD feeding a branch: 1 stall cycle
(d) A conditional branch has 1 delay slot (the next instruction after a conditional branch is fetched and executed to completion without knowing the outcome of the branch)

i First show all of the stall cycles necessary in the original assembly code. Then, find an optimized schedule for this loop through reordering instructions but without resorting to loop unrolling. **(10 points)**

Answer:

ii Optimize the schedule by loop unrolling 1x, 2x, and 3x. Notice that a 1x unroll includes the original loop body plus the 1 time unrolled instructions. **(10 points)**

Answer:

4. **Branch Prediction.** Assume a 32-bit five-stage scalar pipeline with the fetch, decode, execute, memory, and write back stages. All pipeline stages require 1 cycle except the load and store operations that need 3 cycles to access the data memory; branch instructions need 2 clock cycles to determine the outcome. Example C and assembly codes are given for a user application.

Source code	**Assembly code**
`n = 250;`	`      ADDI R3, R0, #1000`
`i = 0;`	`      ADDI R2, R0, #0`
`do {`	`Loop: Load R1, 0(R2)`
`   x[i] = x[i] + 1;`	`      ADDI R1, R1, #1`
`   i = i+1;`	`      Store R1, 0(R2)`
`} while(i < n);`	`      ADDI R2, R2, #4`
	`      SUB R4, R3, R2`
	`      BNEQ R4, R0, Loop`

i Without any branch prediction, how many stall cycles are necessary due to the branch instructions in the original code? **(10 points)**

Answer:

ii Assume a static branch predictor capable of predicting always-taken or always-not-taken. Compute the percentages of improvements in IPC compared to the previous case with no branch predictor? **(10 points)**

Answer:

iii Assume a single 3-bit saturating counter for dynamic branch prediction. The initial state of the counter is 000. States 000–011 predict not taken; while, 100–111 indicate taken. Compute the total number of mis-predictions. Compute the percentages of improvements in IPC compared to the case with no branch predictor? **(10 points)**

Answer:

Note: Assume an instruction does not enter the execution phase until all of its operands are ready.

5. **Bonus Question.** Consider running the following code on a pipelined machine. Every pipeline stage requires one cycle. Every branch instruction needs three cycles to produce an outcome. Assume that only branches may introduce stall cycles to the pipeline. We want to design a global branch predictor with 32 2-bit counters (n=2), where 5 bits from the PC are XORed with a 5-bit GHR (b=r=5) to produce an index to the shared counters.

i Without any branch prediction, compute the IPC of the code? **(5 points)**

ii Assume all the shared counters and the GHR are initialized to 0, compute the IPC and misprediction rate of the global branch predictor? **(15 points)**

Note: after executing a branch, GHR shifts 1 bit to the left. The least significant bit of GHR is set to the branch outcome (0: not-taken and 1: taken).

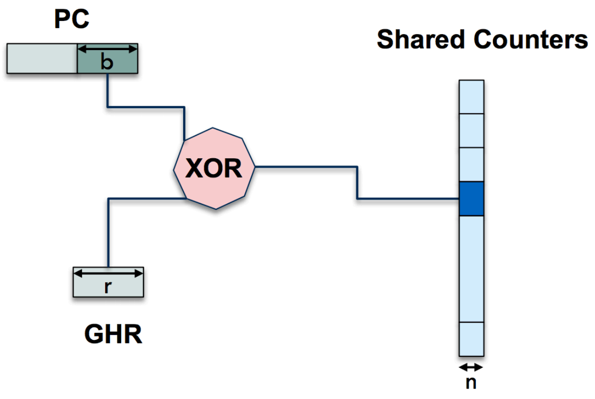

Figure 2: Global branch predictor.

PC	:	**Instruction**	
0000	:		`ADDI R1, R0, #1`
0004	:		`ADDI R2, R0, #9`
0008	:	`Loop:`	`ADDI R2, R2, #-1`
0012	:		`ANDI R3, R2, #3`
0016	:		`BNEQ R3, R1, Next`
0020	:		`ADD R4, R4, R3`
0024	:	`Next:`	`BNEQ R2, R0, Loop`