Example Solutions for Homework Assignment 2

CS/ECE 6810: Computer Architecture
September 26,2018

ILP and Branch Prediction

Due Date: October 03, 2018.
120 points

Important Notes:

- Solutions turned in must be your own. Please, mention references (if any) at the end of each question. Please refrain from cheating.
- All solutions must be accompanied by the equations used/logic/intermediate steps. Writing only the final answer will receive **zero** credit.
- All units must be mentioned wherever required.
- Late submissions **(after 11:59 pm on 10/03/2018)** will not be accepted
- All submissions must be in PDF only. Scanned copies of handwritten solutions will not be accepted as a valid submission.
- write down any extra assumptions which are not mentioned in the questions.

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

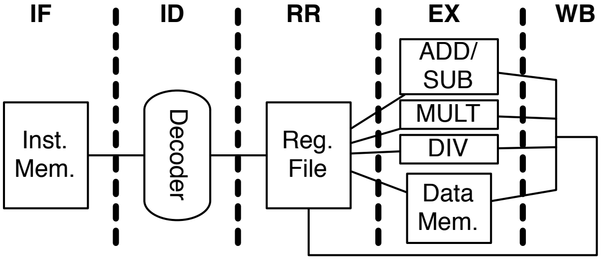

Figure 1: Pipelined core architecture.

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

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)

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

Answer As shown below, a total of 10 data hazards exist in the code. Two stall cycles due to a structural hazard is identified in the timing diagram (Table 1).

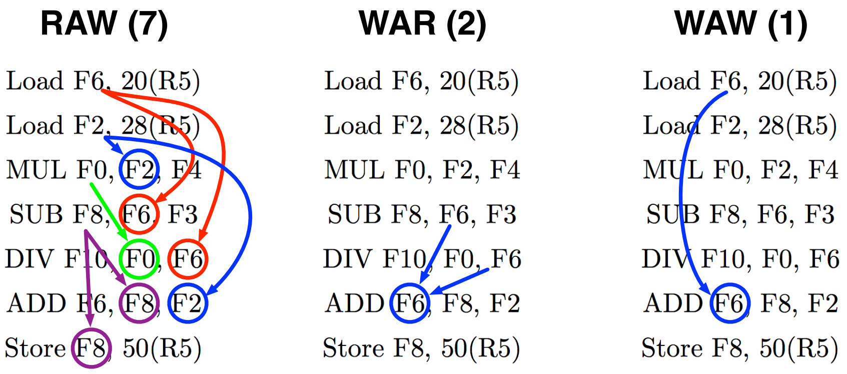

Figure 2: Data hazards.

Note: Identifying each data hazards has 0.8 score. Identifying each structural hazards has 1 score.

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

Answer The timing Diagram is shown in Table 1.

Table 1: Timing diagram.

time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Load	IF	ID	RR	EX	**EX**	WB(F6)														
Load		IF	ID	RR	Stall(SH)	**EX**	EX	WB(F2)												
MUL			IF	ID	Stall	RR	Stall	EX	EX	EX	WB(F0)									
SUB				IF	Stall	ID	Stall	RR	EX	Stall	Stall	**WB(F8)**								
DIV						IF	Stall	ID	RR	Stall	EX	EX	EX	EX	EX	EX	EX	WB(F10)		
ADD							IF	ID	Stall	RR	EX	Stall	Stall	Stall	Stall	Stall	Stall	WB(F6)		
Store								IF	Stall	ID	Stall(SH)	**RR(F8, R5)**	EX	Stall	Stall	Stall	Stall	Stall	WB(–)	

Note: Assumptions to solve the question:

- To avoid imprecise exception, keep the WB in order.

- Register file has 2 read and one write ports; a write and up to two read can happen at the same time for different locations but NOT for the same location.

Notes: 1- RR for the MUL does not have a valid value; however, it is not important since the RR can be bypassed and the value is prepared directly from the EX unit.

2- At cycle 5, the second load is stalled in the RR stage; therefore, the following instructions in the IF and ID stages are stalled.

3- At Cycle 7, MUL is stalled in the RR stage and hence the following instructions cannot proceed. 4- Store does not need a write back to the register file.

Correct pipelining of the first instruction has 2 points and others have 3 points.

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)**

   Answer The unpipelined architecture requires one cycle ($36ns + 0.5ns = 36.5ns$)

   per each instruction (5 points); therefore, the maximum attainable throughput is $IPS = \frac{1}{36.5\times10^{-9}} = 27397260.274$ (5 points).

   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)**

   Answer For the pipelined architecture, the circuit is first partitioned into 12 stages

   (each being $\frac{36ns}{12} = 3ns$ long); then, a latch overhead is added to each stage (cycle time=$3ns + 0.5ns = 3.5ns$) (3 points). The distance between the points of production and consumption is $12ns/3ns = 4$ cycles (3 points); therefore, 3 stall cycles are necessary between the dependent instructions. Every two instructions are executed in 5 cycles; therefore, $IPS = \frac{2}{5\times3.5\times10^{-9}} = 114285714.286$ (3 points). The percentages of increase in throughput will then be $\frac{IPS_{New}-IPS_{Old}}{IPS_{Old}} = 317\%$ (1 point).

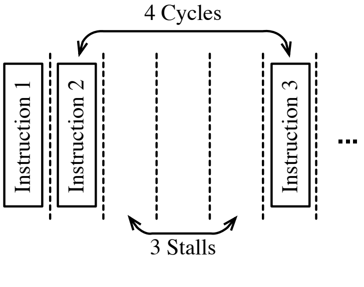

Figure 3: Points of production and consumption in the pipelined architecture.

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 Seven stall cycles are needed in the original code; the optimized code needs five cycles. Please notice based on announcement made in Canvas, considering one stall cycle between MUL and ADD is also acceptable.

Original Code

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

Optimized Code

```
ADDI R1, R0, #1000
JMP Chck
Loop:  Load F1, 0(R1)
Stall
MUL F3, F1, F2
ADD F5, F3, F4
Stall
Stall
Stall
ADDI R1, R1, #-8
Store F5, 20008(R1)
Chck:  BNEQ R1, R0, Loop
Stall
```

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 for simplicity, any prologue and epilogue are skipped and only the main loop is shown. (1x has 3 points; 2x and 3x have 4 points)

Optimized code after 1x unrolling

```
       ADDI R1, R0, #1000
       JMP Chck
Loop:  Load F1, 0(R1)
       Load F10, -8(R1)
       MUL F3, F1, F2
       ADD F5, F3, F4
       MUL F13, F10, F2
       ADD F15, F13, F4
       Stall
       Stall
       Store F5, 20000(R1)
       ADDI R1, R1, #-16
       Store F15, 20008(R1)
Chck:  BNEQ R1, R0, Loop
       Stall
```

Optimized code after 2x unrolling

```
       ADDI R1, R0, #1000
       JMP Chck
Loop:  Load F1, 0(R1)
       Load F10, -8(R1)
       MUL F3, F1, F2
       ADD F5, F3, F4
       MUL F13, F10, F2
       Load F20, -16(R1)
       ADD F15, F13, F4
       MUL F23, F20, F2
       ADD F25, F23, F4
       Store F5, 20000(R1)
       Stall
       Store F15, 19992(R1)
       ADDI R1, R1, #-24
       Store F25, 20008(R1)
Chck:  BNEQ R1, R0, Loop
       Stall
```

Optimized code after 3x unrolling

```
       ADDI R1, R0, #1000
       JMP Chck
Loop:  Load F1, 0(R1)
       Load F10, -8(R1)
       MUL F3, F1, F2
       ADD F5, F3, F4
       MUL F13, F10, F2
       Load F30, -24(R1)
       Load F20, -16(R1)
       MUL F31, F30, F2
       ADD F32, F31, F4
       ADD F15, F13, F4
       MUL F23, F20, F2
       ADD F25, F23, F4
       Store F5, 20000(R1)
       Store F32, 19976(R1)
       Store F15, 19992(R1)
       ADDI R1, R1, #-32
       Store F25, 20016(R1)
Chck:  BNEQ R1, R0, Loop
       Stall
```

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

```
n = 250;
i = 0;
do {
   x[i] = x[i] + 1;
   i = i+1;
} while(i < n);
```

Assembly code

```
       ADDI R3, R0, #1000
       ADDI R2, R0, #0
Loop:  Load R1, 0(R2)
       ADDI R1, R1, #1
       Store R1, 0(R2)
       ADDI R2, R2, #4
       SUB R4, R3, R2
       BNEQ R4, R0, Loop
```

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

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

Answer The branch instruction is executed for R2=4, 8, ..., 1000. Without any branch prediction, each execution of the branch requires two stall cycles, which results in a total of $250 \times 2 = 500$ stall cycles.

The original code with stall cycles

```
       ADDI R3, R0, #1000
       ADDI R2, R0, #0
Loop:  Load R1, 0(R2)
       Stall
       Stall
       Stall
       ADDI R1, R1, #1
       Store R1, 0(R2)
       Stall
       Stall
       Stall
       ADDI R2, R2, #4
       SUB R4, R3, R2
       BNEQ R4, R0, Loop
       Stall
       Stall
```

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 First, IPC for the previous part (the original code with all stalls) is computed by $IPC = \frac{N}{C}$; where N is the total number of all executed instructions in C cycles. Therefore, $IPC = \frac{2+250\times 6}{2+250\times 14} = 0.4288977727$. (4 points)

Second, assume an always-not-taken branch predictor.[1] Only in the last iteration, the prediction outcome is accurate; therefore, the total number of stalls (and consequently the execution time) will be decreased by two cycles. The new $IPC = \frac{2+250\times 6}{250\times 14} = 0.42914285714$ (5 points); therefore, IPC is improved by 0.05%. (1 point) Note: Always taken is also acceptable.

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 The branch is executed 250 times, of which only one is not-taken. Starting state 000, the 3-bit branch predictor requires 4 mis-predictions (not-taken due to states 000-011) until the first accurate prediction. The rest predictions are all accurate (taken) except during the last iteration of the loop. Therefore, the total number of mis-predictions by the 3-bit branch predictor is 5, which decreases the number of cycles by $245 \times 2 = 490$. (6 points) The new $IPC = \frac{2+250\times 6}{250\times 14-490} = 0.49900332225$ (3 points); therefore, IPC is improved by 16.34% (1 point).

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. 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).

[1] Other solutions using an always-taken predictor will be evaluated based on your clearly mentioned assumptions.

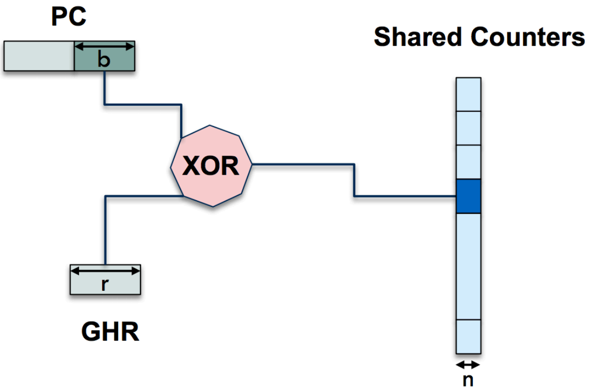

Figure 4: Global branch predictor.

```
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
```

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

Answer The branch instructions are executed for R2=8, 7, ..., 0. Without any branch prediction, each branch instruction requires 3 stall cycles (see the code below). Assume, $IPC = \frac{N}{C}$; where, N is the total number of all executed instructions in C cycles. Notice that `ADD R4, R4, R3` is executed only two times for R2=5 and R2=1. Therefore, $IPC = \frac{2+9\times 4+2}{2+9\times 10+2} = 0.42553191489$.

The original code with branch stalls

```
       ADDI R1, R0, #1
       ADDI R2, R0, #9
Loop:  ADDI R2, R2, #-1
       ANDI R3, R2, #3
       BNEQ R3, R1, Next
       Stall
       Stall
       Stall
       ADD R4, R4, R3
Next:  BNEQ R2, R0, Loop
       Stall
       Stall
       Stall
```

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)**

Answer The two branches result in a total of 18 predictions, out of which only 4 are predicted accurately. As a result, the mis-prediction rate is $14/18 = 0.77777777777$. To compute $IPC = \frac{N}{C}$, use $N = 2+9\times 4+2 = 40$ and $C = 2+9\times 10+2-4\times 3 = 82$; therefore, $IPC = \frac{40}{82} = 0.48780487804$.

PC	**PC$_b$**	**GHR**	**XOR$_{out}$**	**Counter**	**Prediction**	**Outcome**	**Result**
0016	10000	00000	10000	C_{16}=00	N	T	M
0024	11000	00001	11001	C_{25}=00	N	T	M
0016	10000	00011	10011	C_{19}=00	N	T	M
0024	11000	00111	11111	C_{31}=00	N	T	M
0016	10000	01111	11111	C_{31}=01	N	T	M
0024	11000	11111	00111	C_{7}=00	N	T	M
0016	10000	11111	01111	C_{15}=00	N	N	–
0024	11000	11110	00110	C_{6}=00	N	T	M
0016	10000	11101	01101	C_{13}=00	N	T	M
0024	11000	11011	00011	C_{3}=00	N	T	M
0016	10000	10111	00111	C_{7}=01	N	T	M
0024	11000	01111	10111	C_{23}=00	N	T	M
0016	10000	11111	01111	C_{15}=00	N	T	M
0024	11000	11111	00111	C_{7}=10	T	T	–
0016	10000	11111	01111	C_{15}=01	N	N	–
0024	11000	11110	00110	C_{6}=01	N	T	M
0016	10000	11101	01101	C_{13}=01	N	T	M
0024	11000	11011	00011	C_{3}=01	N	N	–