MIDTERM REVIEW

Mahdi Bojnordi

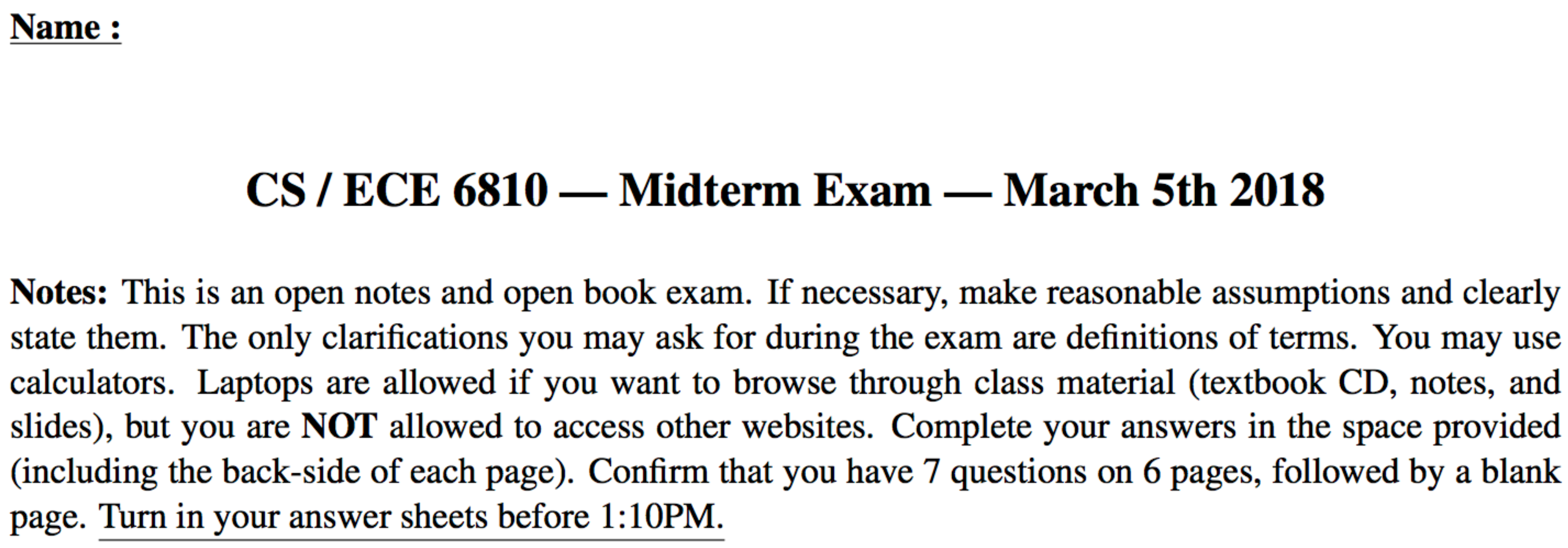
School of Computing

University of Utah

THE UNIVERSITY OF UTAH

CS/ECE 6810: Computer Architecture

Heading



Name :

CS / ECE 6810 — Midterm Exam — March 5th 2018

Notes: This is an open notes and open book exam. If necessary, make reasonable assumptions and clearly state them. The only clarifications you may ask for during the exam are definitions of terms. You may use calculators. Laptops are allowed if you want to browse through class material (textbook CD, notes, and slides), but you are **NOT** allowed to access other websites. Complete your answers in the space provided (including the back-side of each page). Confirm that you have 7 questions on 6 pages, followed by a blank page. Turn in your answer sheets before 1:10PM.

Q1. Processor Performance

1. **Processor Performance.** Two different software implementations (programs) are proposed for a particular scientific function that are called *prog-A* and *prog-B*. The programs are executed on *comp-1*, a scalar RISC processor, one at a time to collect the following statistics. The total numbers of executed instructions for *prog-A* and *prog-B* on *comp-1* are 1000 and 2000, respectively.

comp-1			
Instruction Type	Cycles	*prog-A*	*prog-B*
ADD	2	30%	60%
MULT	8	40%	25%
DIV	40	15%	5%
Branch	1	15%	10%

(a) Assuming that *comp-1* operates at 2GHz, find the execution times of the programs and identify which one runs faster. **(10 points)**

(b) A newer version of the processor is *comp-2* that provides a 6-cycle fused multiply and add (FMA) instruction in addition to the instructions supported by *comp-1* (with the same number of cycles per instruction). Assuming that *comp-2* operates at 1.8GHz, find the execution times of the programs and identify which one runs faster. **(10 points)**

i. Every ADD in *prog-A* is followed by a MULT, which can be replaced by an FMA.

ii. Every MULT in *prog-B* follows an ADD, which can be replaced by an FMA.

1. Processor Performance

- A) find execution times on comp-1 @ 2GHz
  - CPU Time = IC x CPI x CT

	Cycles	Prog-A	Prog-B
ADD	2	30%	60%
MULT	8	40%	25%
DIV	40	15%	5%
Branch	1	15%	10%

1. Processor Performance

- A) find execution times on comp-1 @ 2GHz
  - CPU Time = IC x CPI x CT

	Cycles	Prog-A	Prog-B
ADD	2	30%	60%
MULT	8	40%	25%
DIV	40	15%	5%
Branch	1	15%	10%

IC

Prog-A	Prog-B
300	1200
400	500
150	100
150	200

Total 1000 2000

1. Processor Performance

- A) find execution times on comp-1 @ 2GHz
  - CPU Time = IC x CPI x CT

	Cycles	Prog-A	Prog-B
ADD	2	30%	60%
MULT	8	40%	25%
DIV	40	15%	5%
Branch	1	15%	10%
			Total

IC

Prog-A	Prog-B
300	1200
400	500
150	100
150	200
1000	2000

IC x CPI

Prog-A	Prog-B
600	2400
3200	4000
6000	4000
150	200
9950	10600

1. Processor Performance

- A) find execution times on comp-1 @ 2GHz
  - CPU Time = IC x CPI x CT

	Cycles	Prog-A	Prog-B
ADD	2	30%	60%
MULT	8	40%	25%
DIV	40	15%	5%
Branch	1	15%	10%
			Total

IC

Prog-A	Prog-B
300	1200
400	500
150	100
150	200
1000	2000

IC x CPI

Prog-A	Prog-B
600	2400
3200	4000
6000	4000
150	200
9950	10600

CT = 1/(2e9) = 0.5e-9 ➔ CPU Time : 4.98e-6 5.3e-6

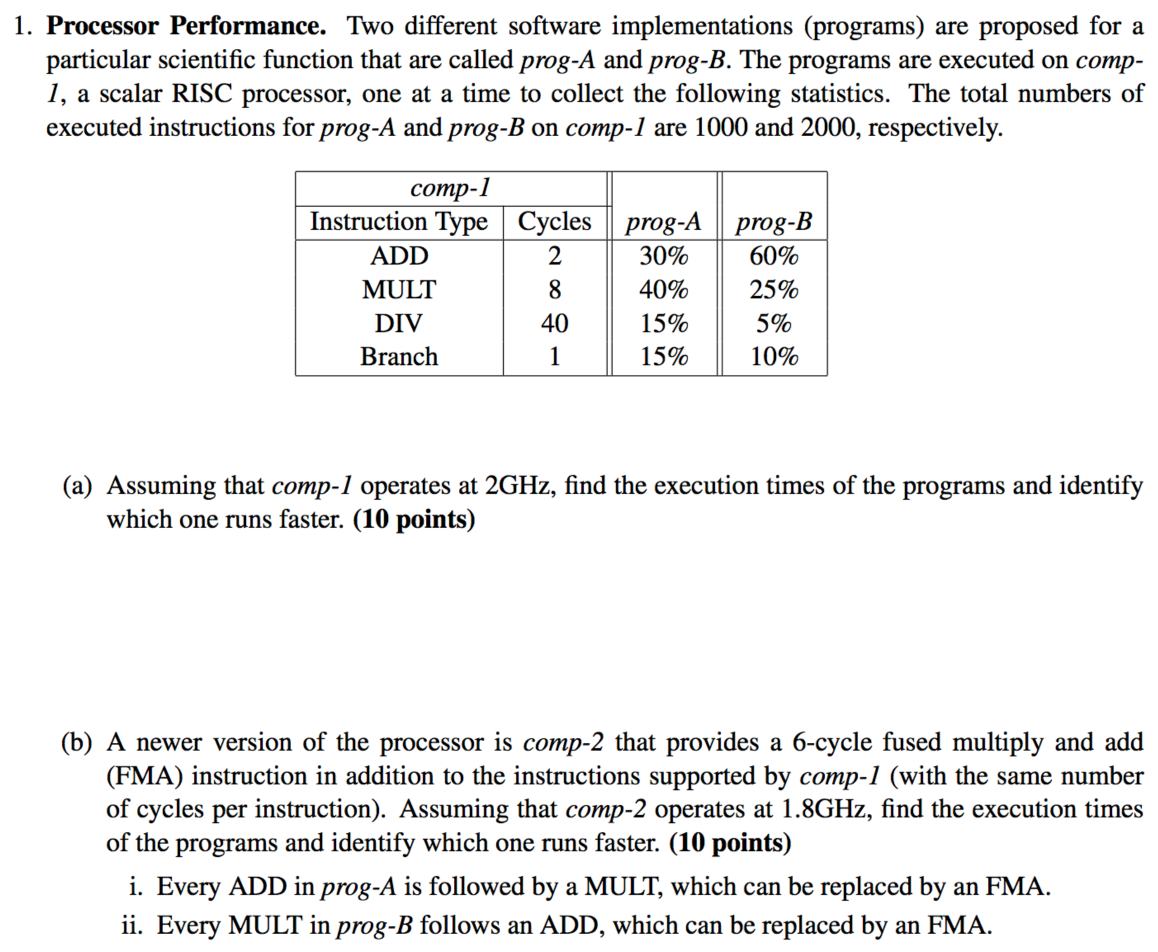
1. Processor Performance

- A) find execution times on comp-1 @ 2GHz
  - CPU Time = IC x CPI x CT

	Cycles	Prog-A	Prog-B	IC Prog-A	IC Prog-B	IC x CPI Prog-A	IC x CPI Prog-B
ADD	2	30%	60%	300	1200	600	2400
MULT	8	40%	25%	400	500	3200	4000
DIV	40	15%	5%	150	100	6000	4000
Branch	1	15%	10%	150	200	150	200
			Total	1000	2000	9950	10600

CT = 1/(2e9) = 0.5e-9 $\rightarrow$ CPU Time : $\boxed{4.98\text{e-}6}$ 5.3e-6

Q1. Processor Performance



1. **Processor Performance.** Two different software implementations (programs) are proposed for a particular scientific function that are called *prog-A* and *prog-B*. The programs are executed on *comp-1*, a scalar RISC processor, one at a time to collect the following statistics. The total numbers of executed instructions for *prog-A* and *prog-B* on *comp-1* are 1000 and 2000, respectively.

comp-1			
Instruction Type	Cycles	*prog-A*	*prog-B*
ADD	2	30%	60%
MULT	8	40%	25%
DIV	40	15%	5%
Branch	1	15%	10%

(a) Assuming that *comp-1* operates at 2GHz, find the execution times of the programs and identify which one runs faster. **(10 points)**

(b) A newer version of the processor is *comp-2* that provides a 6-cycle fused multiply and add (FMA) instruction in addition to the instructions supported by *comp-1* (with the same number of cycles per instruction). Assuming that *comp-2* operates at 1.8GHz, find the execution times of the programs and identify which one runs faster. **(10 points)**

i. Every ADD in *prog-A* is followed by a MULT, which can be replaced by an FMA.

ii. Every MULT in *prog-B* follows an ADD, which can be replaced by an FMA.

1. Processor Performance

- B) find execution times on comp-2 @ 1.8GHz
  - CPU Time = IC x CPI x CT

	Cycles	Prog-A	Prog-B
ADD	2		
MULT	8		
DIV	40		
Branch	1		
FMA	6		

IC

Prog-A	Prog-B
0	700
100	0
150	100
150	200
300	500

Total 700 1500

1. Processor Performance

- B) find execution times on comp-2 @ 1.8GHz
  - CPU Time = IC x CPI x CT

				IC		IC x CPI	
	Cycles	Prog-A	Prog-B	Prog-A	Prog-B	Prog-A	Prog-B
ADD	2			0	700	0	1400
MULT	8			100	0	800	0
DIV	40			150	100	6000	4000
Branch	1			150	200	150	200
FMA	6			300	500	1800	3000
			Total	700	1500	8750	8600

CT = 1/(1.8e9) ➔ CPU Time : 4.86e-6 4.78e-6

1. Processor Performance

- B) find execution times on comp-2 @ 1.8GHz
  - CPU Time = IC x CPI x CT

				IC		IC x CPI	
	Cycles	Prog-A	Prog-B	Prog-A	Prog-B	Prog-A	Prog-B
ADD	2			0	700	0	1400
MULT	8			100	0	800	0
DIV	40			150	100	6000	4000
Branch	1			150	200	150	200
FMA	6			300	500	1800	3000
			Total	700	1500	8750	8600

CT = 1/(1.8e9) ➔ CPU Time : 4.86e-6 $\boxed{4.78\text{e-}6}$

1. Processor Performance

- B) find execution times on comp-2 @ 1.8GHz
  - CPU Time = IC x CPI x CT

	Cycles	Prog-A	Prog-B
ADD	2	0%	~47%
MULT	8	~14%	0%
DIV	40	~21%	~7%
Branch	1	~21%	~13%
FMA	6	~42%	~33%

IC

Prog-A	Prog-B
0	700
100	0
150	100
150	200
300	500
700	1500

IC x CPI

Prog-A	Prog-B
0	1400
800	0
6000	4000
150	200
1800	3000
8750	8600

Total: 700 1500 8750 8600

CT = 1/(1.8e9) ➔ CPU Time : 4.86e-6 $\boxed{4.78e-6}$

Q2. Instruction Set Architecture

2. **Instruction Set Architecture.** The initial values for parts of the register file and main memory used in an 8-bit processor are shown in the table below.

Register File		Main Memory	
Address	*Value*	*Address*	*Value*
R1	100	200	250
R2	150	250	200
R3	200	251	250
R4	250	350	100
R5	300	400	300

Compute the effective address and final result for each of the following instructions. Register value changes are considered when moving from one instruction to another. All of the instructions are executed serially. **(20 points)**

```
ADD R5, R1, R2
LD R5, +(R5)
LD R4, 100(R5)
LD R1, @(R3)
ADD R4, R4, R1
LD R2, (R3+R1)
```

2. Instruction Set Architecture

- An 8-bit processor
  - Displacement (d) = 1

			R1	R2	R3	R4	R5	M[200]	M[250]	M[251]	M[350]	M[400]
Instructions	Effective Address		100	150	200	250	300	250	200	250	100	300
ADD R5, R1, R2	NA	NA	100	150	200	250	250	250	200	250	100	300
LD R5, +(R5)	251	NA	100	150	200	250	250	250	200	250	100	300
LD R4, 100(R5)	350	NA	100	150	200	100	250	250	200	250	100	300
LD R1, @(R3)	200	250	200	150	200	100	250	250	200	250	100	300
ADD R4, R4, R1	NA	NA	200	150	200	300	250	250	200	250	100	300
LD R2, (R3+R1)	400	NA	200	300	200	300	250	250	200	250	100	300

Q3. Pipelining

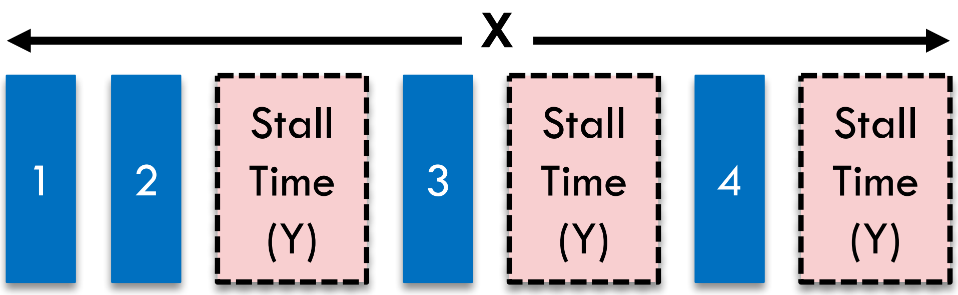
3. **Pipelining.** Consider an un-pipelined processor where it takes 20ns to go through the circuits and 1ns for the latch overhead. Assume that the point of production and point of consumption in the un-pipelined processor are separated by 10ns. Assume that one fourth of the instructions do not introduce a data hazard and the rest depend on their preceding instruction. What is the throughput—in million instructions per second (MIPS)—for this processor with (i) an un-pipelined architecture, and (ii) a 10-stage pipeline? **(10 points)**

3. Pipelining

- i) un-pipelined architecture
  - Cycle time = 20ns + 1ns = 21ns
  - P2P fits within one cycle → no stall cycles are needed
  - IPC=1 and CT=21e-9
    - IPS = 1/21e-9 → MIPS = 47.61

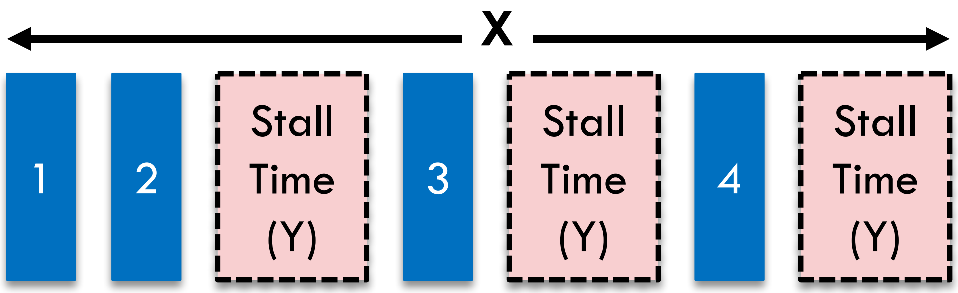
3. Pipelining

- ¼ instructions do not introduce data hazards



3. Pipelining

- ¼ instructions do not introduce data hazards



- ii) 10-stage pipelined architecture
  - Cycle time = 20ns/10 + 1ns = 3ns
  - P2P = $\lceil$10ns/2ns$\rceil$ = 5 ➔ Y=4
  - X = 16x3e-9 ➔ IPS = 4/(16x3e-9) ➔ MIPS = 83.33

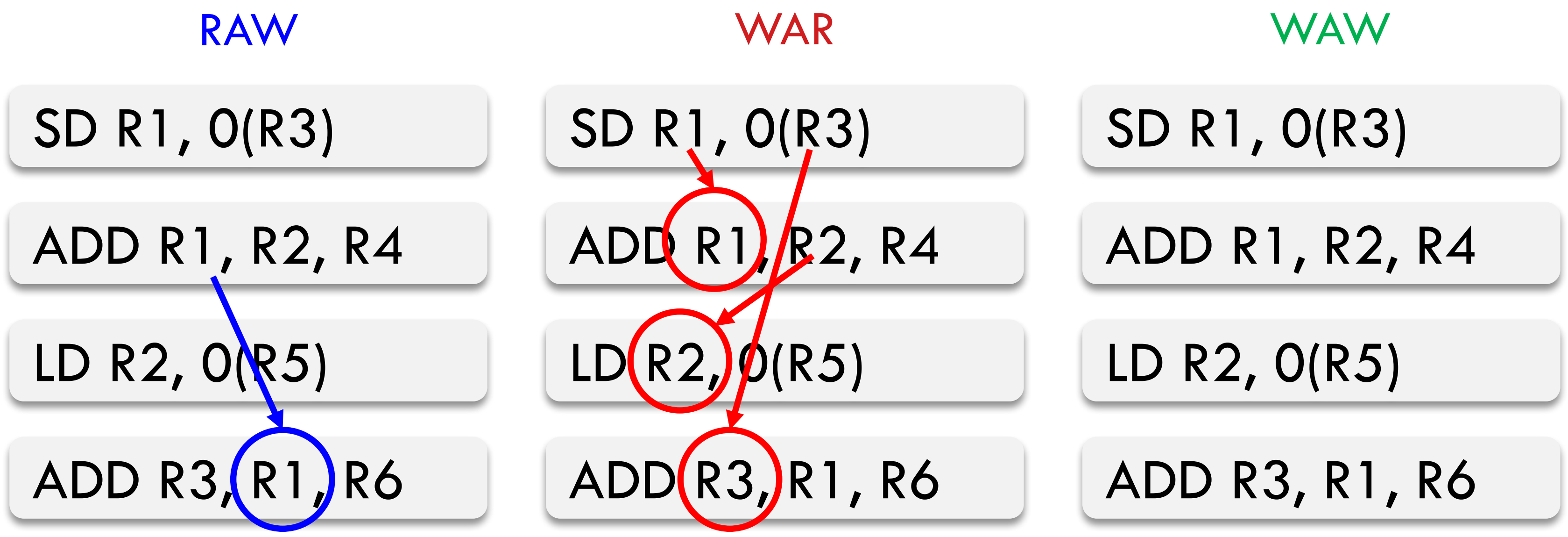
Q4. Data Hazards

4. **Data Hazards.** Identify all the data hazards in the following code. **(10 points)**

```
SD R1, 0(R3)
ADD R1, R2, R4
LD R2, 0(R5)
ADD R3, R1, R6
```

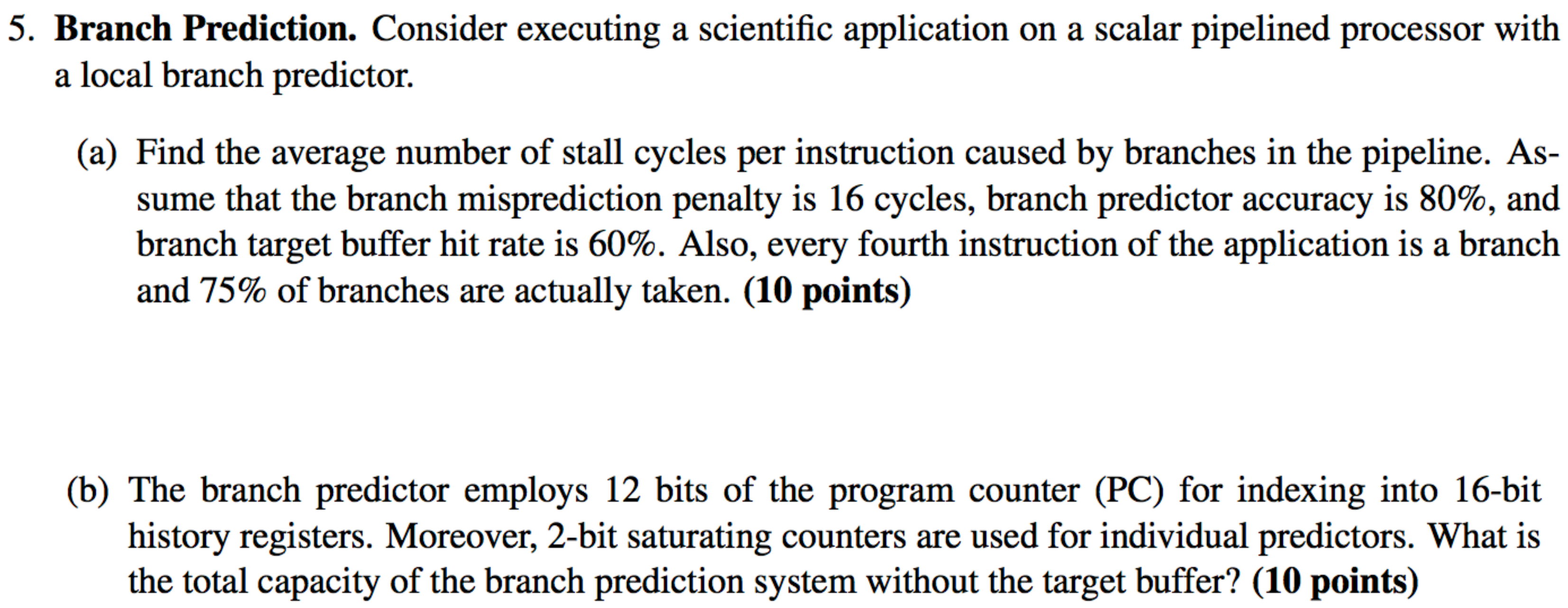
4. Data Hazards

- Identify data hazards



RAW	WAR	WAW
SD R1, 0(R3)	SD R1, 0(R3)	SD R1, 0(R3)
ADD R1, R2, R4	ADD R1, R2, R4	ADD R1, R2, R4
LD R2, 0(R5)	LD R2, 0(R5)	LD R2, 0(R5)
ADD R3, R1, R6	ADD R3, R1, R6	ADD R3, R1, R6

Q5. Branch Prediction



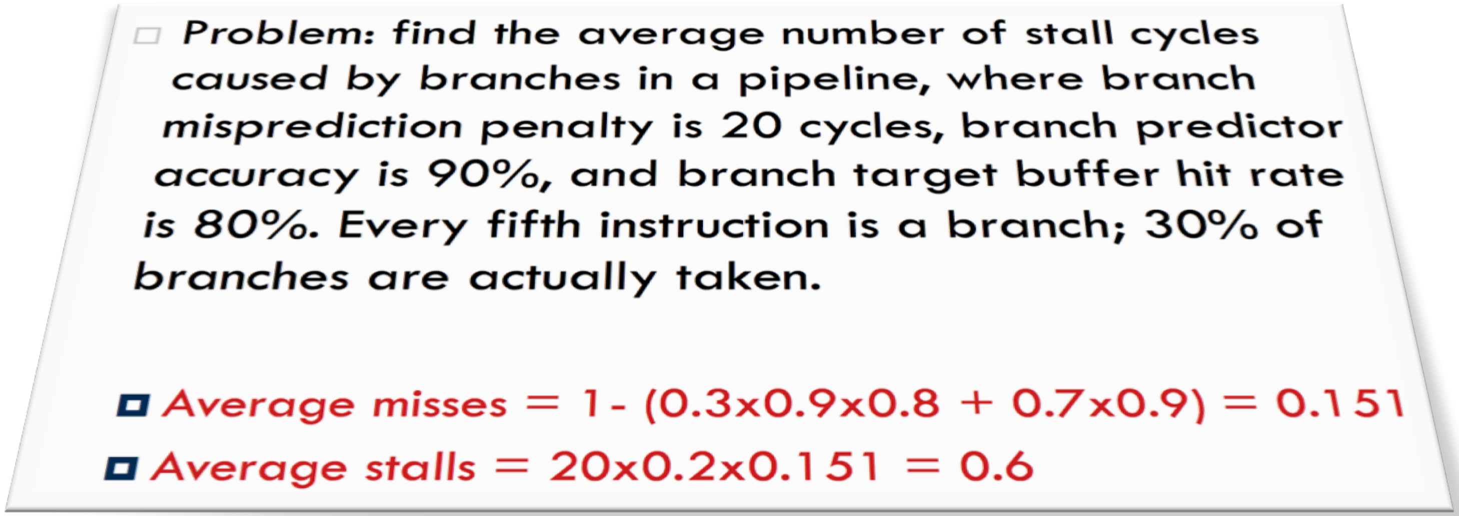
5. **Branch Prediction.** Consider executing a scientific application on a scalar pipelined processor with a local branch predictor.

   (a) Find the average number of stall cycles per instruction caused by branches in the pipeline. Assume that the branch misprediction penalty is 16 cycles, branch predictor accuracy is 80%, and branch target buffer hit rate is 60%. Also, every fourth instruction of the application is a branch and 75% of branches are actually taken. **(10 points)**

   (b) The branch predictor employs 12 bits of the program counter (PC) for indexing into 16-bit history registers. Moreover, 2-bit saturating counters are used for individual predictors. What is the total capacity of the branch prediction system without the target buffer? **(10 points)**

5. Branch Prediction

- Local branch predictor
- A) average number of stall cycles



- Problem: find the average number of stall cycles caused by branches in a pipeline, where branch misprediction penalty is 20 cycles, branch predictor accuracy is 90%, and branch target buffer hit rate is 80%. Every fifth instruction is a branch; 30% of branches are actually taken.

- Average misses = 1- (0.3x0.9x0.8 + 0.7x0.9) = 0.151
- Average stalls = 20x0.2x0.151 = 0.6

5. Branch Prediction

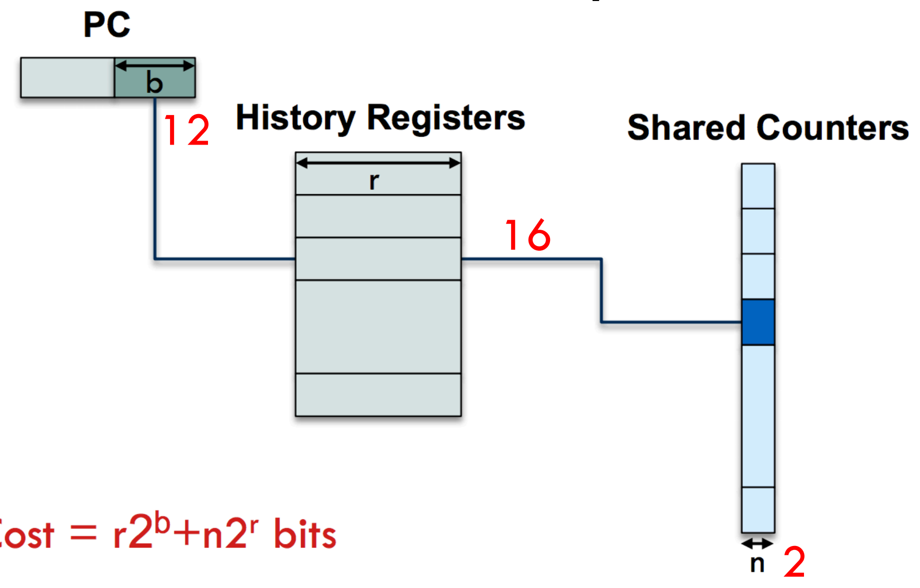
- Local branch predictor
- A) average number of stall cycles

Instructions	Outcome	Prediction	Target	Penalty	Avg. Stalls
0.25 (BR)	0.75 (T)	0.8 (T)	0.6 (H)	0	0
			0.4 (M)	16	0.96
		0.2 (N)	1 (N/A)	16	0.6
	0.25 (N)	0.8 (N)	1 (N/A)	0	0
		0.2 (T)	0.6 (H)	16	0.12
			0.4 (M)	16	0.08

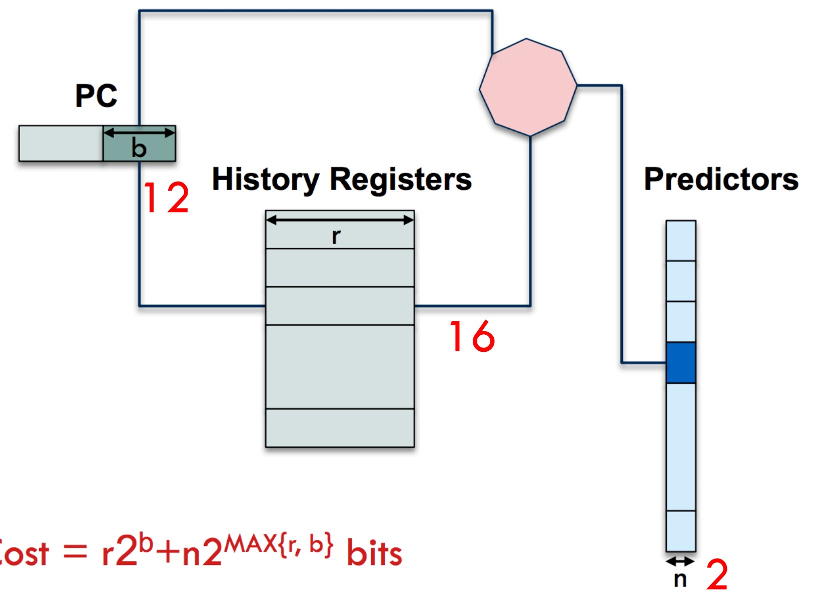
1.76

5. Branch Prediction

- Local branch predictor



Cost = $r2^b+n2^r$ bits



Cost = $r2^b+n2^{MAX\{r,\ b\}}$ bits

- B) cost = $r2^b+n2^{MAX\{r,\ b\}}$ bits

Cost = $16x2^{12} + 2x2^{16}$ = 192Kb = 24KB

Q6. Register Renaming

6. **Register Renaming.** Consider an out-of-order processor that has 8 logical registers (denoted by R) and 10 physical registers (denoted by P). On power up, assume that logical register R1 is mapped to physical register P1, R2 is mapped to P2, and so on; the following program starts executing:

```
SD R1, 0(R1)
ADD R4, R3, R1
SD R4, 8(R3)
LD R1, 16(R4)
SUB R3, R4, R1
ADD R1, R2, R3
SD R1, 8(R1)
```

Show the renamed version of this code. **(10 points)**

6. Register Renaming

- Show the renamed version

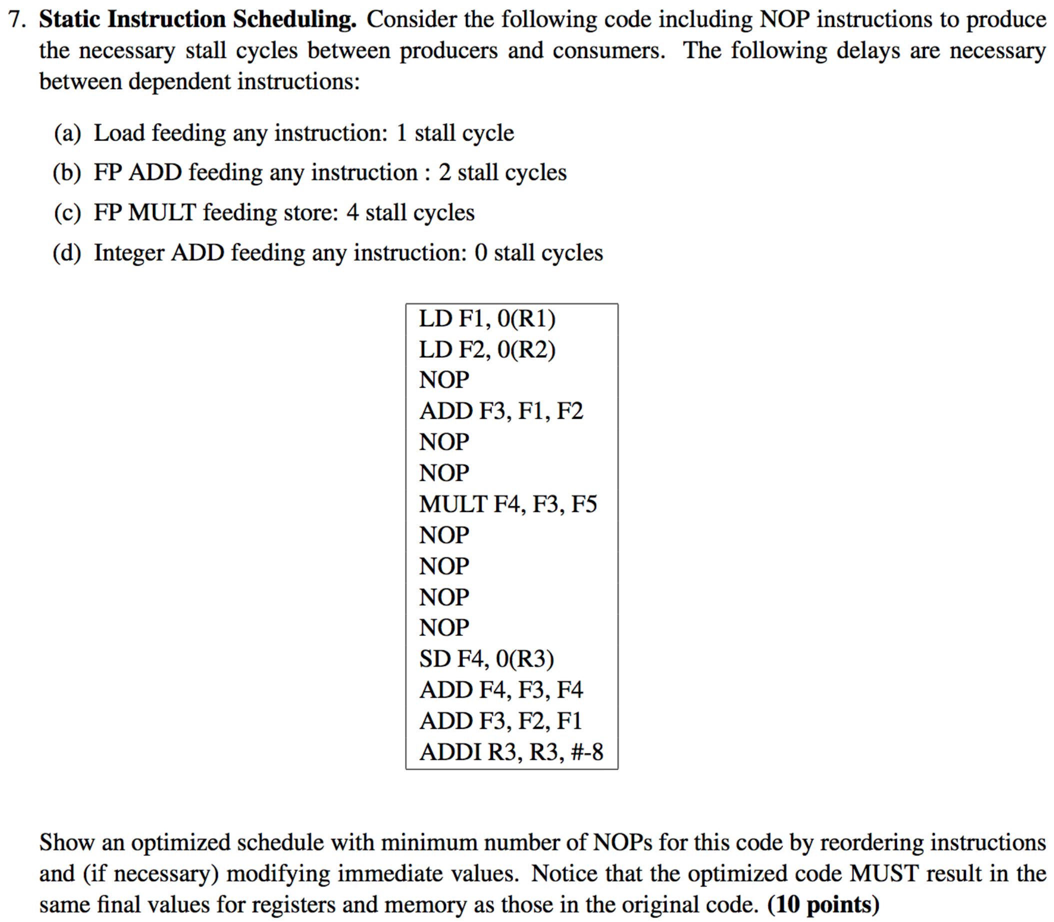
SD R1, 0(R1)
ADD R4, R1, R3
SD R4, 8(R3)
LD R1, 16(R4)
SUB R3, R1, R4
ADD R1, R2, R3
SD R1, 8(R1)

R1	R2	R3	R4	R5	R6	R7	R8
P1	P2	P3	P4	P5	P6	P7	P8
P1	P2	P3	P4	P5	P6	P7	P8
P1	P2	P3	P9	P5	P6	P7	P8
P1	P2	P3	P9	P5	P6	P7	P8
P10	P2	P3	P9	P5	P6	P7	P8
P10	P2	P4	P9	P5	P6	P7	P8
P1	P2	P4	P9	P5	P6	P7	P8
P1	P2	P4	P9	P5	P6	P7	P8

Free List	
P9	P10
P9	P10
P4	P10
P4	P10
P4	P1
P3	P1
P3	P10
P3	P10

SD P1, 0(P1)
ADD P9, P1, P3
SD P9, 8(P3)
LD P10, 16(P9)
SUB P4, P10, P9
ADD P1, P2, P4
SD P1, 8(P1)

Q7. Static Instruction Scheduling



7. **Static Instruction Scheduling.** Consider the following code including NOP instructions to produce the necessary stall cycles between producers and consumers. The following delays are necessary between dependent instructions:

   (a) Load feeding any instruction: 1 stall cycle

   (b) FP ADD feeding any instruction : 2 stall cycles

   (c) FP MULT feeding store: 4 stall cycles

   (d) Integer ADD feeding any instruction: 0 stall cycles

```
LD F1, 0(R1)
LD F2, 0(R2)
NOP
ADD F3, F1, F2
NOP
NOP
MULT F4, F3, F5
NOP
NOP
NOP
NOP
SD F4, 0(R3)
ADD F4, F3, F4
ADD F3, F2, F1
ADDI R3, R3, #-8
```

Show an optimized schedule with minimum number of NOPs for this code by reordering instructions and (if necessary) modifying immediate values. Notice that the optimized code MUST result in the same final values for registers and memory as those in the original code. **(10 points)**

7. Static Instruction Scheduling

- Reordering and modifying immediate value

```
LD F1, 0(R1)
LD F2, 0(R2)
NOP
ADD F3, F1, F2
NOP
NOP
MULT F4, F3, F5
NOP
NOP
NOP
NOP
SD F4, 0(R3)
ADD F4, F3, F4
ADD F3, F2, F1
ADDI R3, R3, #-8
```

7. Static Instruction Scheduling

- Reordering and modifying immediate value

```
LD F1, 0(R1)
LD F2, 0(R2)
NOP
ADD F3, F1, F2
NOP
NOP
MULT F4, F3, F5
NOP
NOP
NOP
NOP
SD F4, 0(R3)
ADD F4, F3, F4
ADD F3, F2, F1
▶ ADDI R3, R3, #-8
```

7. Static Instruction Scheduling

- Reordering and modifying immediate value

```
LD F1, 0(R1)
LD F2, 0(R2)
NOP
ADD F3, F1, F2
NOP
NOP
MULT F4, F3, F5
NOP
NOP
NOP
NOP
SD F4, 0(R3)
ADD F4, F3, F4
ADD F3, F2, F1
▶ ADDI R3, R3, #-8
```

7. Static Instruction Scheduling

- Reordering and modifying immediate value

```
  LD F1, 0(R1)
  LD F2, 0(R2)
  NOP
  ADD F3, F1, F2
  NOP
  NOP
  MULT F4, F3, F5
  NOP
  NOP
  NOP
  NOP
  SD F4, 0(R3)
  ADD F4, F3, F4
  ADD F3, F2, F1
▶ ADDI R3, R3, #-8
```

```
LD F1, 0(R1)
LD F2, 0(R2)
ADDI R3, R3, #-8
ADD F3, F1, F2
NOP
NOP
MULT F4, F3, F5
NOP
NOP
NOP
NOP
SD F4, +8(R3)
ADD F4, F3, F4
ADD F3, F2, F1
```

7. Static Instruction Scheduling

- Reordering and modifying immediate value

```
  LD F1, 0(R1)
  LD F2, 0(R2)
  NOP
  ADD F3, F1, F2
  NOP
  NOP
  MULT F4, F3, F5
  NOP
  NOP
  NOP
  NOP
  SD F4, 0(R3)
  ADD F4, F3, F4
  ADD F3, F2, F1
▶ ADDI R3, R3, #-8
```

```
  LD F1, 0(R1)
  LD F2, 0(R2)
  ADDI R3, R3, #-8
  ADD F3, F1, F2
  NOP
  NOP
  MULT F4, F3, F5
  NOP
  NOP
  NOP
  NOP
  SD F4, +8(R3)
  ADD F4, F3, F4
▶ ADD F3, F2, F1
```

7. Static Instruction Scheduling

- Reordering and modifying immediate value

```
  LD F1, 0(R1)
  LD F2, 0(R2)
  NOP
  ADD F3, F1, F2
  NOP
  NOP
  MULT F4, F3, F5
  NOP
  NOP
  NOP
  NOP
  SD F4, 0(R3)
  ADD F4, F3, F4
  ADD F3, F2, F1
▶ ADDI R3, R3, #-8
```

```
  LD F1, 0(R1)
  LD F2, 0(R2)
  ADDI R3, R3, #-8
  ADD F3, F1, F2
  NOP
  NOP
  MULT F4, F3, F5
  NOP
  NOP
  NOP
  NOP
  SD F4, +8(R3)
  ADD F4, F3, F4
▶ ADD F3, F2, F1
```

7. Static Instruction Scheduling

- Reordering and modifying immediate value

```
LD F1, 0(R1)
LD F2, 0(R2)
NOP
ADD F3, F1, F2
NOP
NOP
MULT F4, F3, F5
NOP
NOP
NOP
NOP
SD F4, 0(R3)
ADD F4, F3, F4
ADD F3, F2, F1
▶ ADDI R3, R3, #-8
```

```
LD F1, 0(R1)
LD F2, 0(R2)
ADDI R3, R3, #-8
ADD F3, F1, F2
NOP
NOP
MULT F4, F3, F5
NOP
NOP
NOP
NOP
SD F4, 8(R3)
ADD F4, F3, F4
▶ ADD F3, F2, F1
```

7. Static Instruction Scheduling

- Reordering and modifying immediate value

```
  LD F1, 0(R1)
  LD F2, 0(R2)
  NOP
  ADD F3, F1, F2
  NOP
  NOP
  MULT F4, F3, F5
  NOP
  NOP
  NOP
  NOP
  SD F4, 0(R3)
  ADD F4, F3, F4
  ADD F3, F2, F1
▶ ADDI R3, R3, #-8
```

```
  LD F1, 0(R1)
  LD F2, 0(R2)
  ADDI R3, R3, #-8
  ADD F3, F1, F2
  NOP
  NOP
  MULT F4, F3, F5
  NOP
  NOP
  NOP
  NOP
  SD F4, +8(R3)
  ADD F4, F3, F4
▶ ADD F3, F2, F1
```

```
LD F1, 0(R1)
LD F2, 0(R2)
ADDI R3, R3, #-8
ADD F3, F1, F2
ADD F3, F2, F1
NOP
MULT F4, F3, F5
NOP
NOP
NOP
NOP
SD F4, +8(R3)
ADD F4, F3, F4
```