CMU 18-447 Introduction to Computer Architecture, Spring 2013

Midterm Exam 1

Date: Wed., 3/6

Instructor: Onur Mutlu
TAs: Justin Meza, Yoongu Kim, Jason Lin

Name:

Legibility & Name (5 Points):	
Problem 1 (65 Points):	
Problem 2 (25 Points):	
Problem 3 (20 Points):	
Problem 4 (40 Points):	
Problem 5 (40 Points):	
Problem 6 (40 Points):	
Problem 7 (35 Points):	
Bonus (45 Points):	
Total (270 + 45 Points):	

Instructions:

1. This is a closed book exam. You are allowed to have one letter-sized cheat sheet.
2. No electronic devices may be used.
3. This exam lasts 1 hour and 50 minutes.
4. Clearly indicate your final answer for each problem.
5. Please show your work when needed.
6. Please write your initials at the top of every page.
7. Please make sure that your answers to all questions (and all supporting work that is required) are contained in the space required.

Tips:

- **Be cognizant of time.** Do not spend to much time on one question.
- **Be concise.** You will be penalized for verbosity.
- **Show work when needed.** You will receive partial credit at the instructors' discretion.
- **Write legibly.** Show your final answer.

1. Potpourri [65 points]

(a) **Full Pipeline [6 points]**

Keeping a processor pipeline full with useful instructions is critical for achieving high performance. What are the three fundamental reasons why a processor pipeline cannot always be kept full?

Reason 1.

Reason 2.

Reason 3.

(b) **Exceptions vs. Interrupts [9 points]**

In class, we distinguished exceptions from interrupts. Exceptions need to be handled when detected by the processor (and known to be non-speculative) whereas interrupts can be handled when convenient.

Why does an exception need to be handled when it is detected? In no more than 20 words, please.

What does it mean to handle an interrupt "when it is convenient"?

Why can many interrupts be handled "when it is convenient"?

(c) **Branch Target Buffer [5 points]**

What is the purpose of a branch target buffer (in no more than 10 words, please)?

What is the downside of a design that does not use a branch target buffer? Please be concrete (and use less than 20 words).

(d) **Return Address Prediction [4 points]**

In lecture, we discussed that a return address stack is used to predict the target address of a return instruction instead of the branch target buffer. We also discussed that empirically a reasonably-sized return address stack provides highly accurate predictions.

What key characteristic of programs does a return address stack exploit?

Assume you have a machine with a 4-entry return address stack, yet the code that is executing has six levels of nested function calls each of which end with an appropriate return instruction. What is the return address prediction accuracy of this code?

(e) **Restartable vs. Precise Interrupts [6 points]**

As we discussed in one of the lectures, an exception (or interrupt) is "restartable" if a (pipelined) machine is able to resume execution exactly from the state when the interrupt happened and after the exception or interrupt is handled. By now you also should know what it means for an interrupt to be precise versus imprecise.

Can a pipelined machine have restartable but imprecise exceptions or interrupts?

What is the disadvantage of such a machine over one that has restartable and precise exceptions or interrupts? Explain briefly.

(f) **Segmentation and Paging** **[4 points]**

In segmentation, translation information is cached as part of the _____.

In paging, translation information is cached in the _____.

(g) **Out-of-Order vs. Dataflow** **[8 points]**

When does the fetch of an instruction happen in a dataflow processor?

When does the fetch of an instruction happen in an out-of-order execution processor?

In class, we covered several dataflow machines that implemented dataflow execution at the ISA level. These machines included a structure/unit called the "matching store." What is the function of the matching store (in less than 10 words)?

What structure accomplishes a similar function in an out-of-order processor?

(h) **Tomasulo's Algorithm [5 points]**

Here is the state of the reservation stations in a processor during a particular cycle (× denotes an unknown value):

`ADD` **Reservation Station**

Tag	V	Tag	Data	V	Tag	Data
A	0	D	×	1	×	27
B	1	×	3	0	E	×
C	0	B	×	0	A	×
×	×	×	×	×	×	×

`MUL` **Reservation Station**

Tag	V	Tag	Data	V	Tag	Data
D	0	B	×	0	C	×
E	1	×	16	0	B	×

What is wrong with this picture?

(i) **Minimizing Stalls [10 points]**

In multiple lectures, we discussed how the compiler can reorder instructions to minimize stalls in a pipelined processor. The goal of the compiler in these optimizations is to find independent instructions to place in between two dependent instructions such that by the time the consumer instruction enters the pipeline the producer would have produced its result.

We discussed that control dependences get in the way of the compiler's ability to reorder instructions. Why so?

What can the compiler do to alleviate this problem? Describe two solutions we discussed in class.

Solution 1.

Solution 2.

What is the major disadvantage or limitation of each solution?

Solution 1.

Solution 2.

(j) **Tomasulo's Algorithm Strikes Back** **[8 points]**

You have a friend who is an architect at UltraFastProcessors, Inc. Your friend explains to you how their newest out-of-order execution processor that implements Tomasulo's algorithm and that uses full data forwarding works:

"After an instruction finishes execution in the functional unit, the result of the instruction is latched. In the next cycle, the tag and result value are broadcast to the reservation stations. Comparators in the reservation stations check if the source tags of waiting instructions match the broadcast tag and capture the broadcast result value if the broadcast tag is the same as a source tag."

Based on this description, is there an opportunity to improve the performance of your friend's design? Circle one:

YES NO

If YES, explain what type of code leads to inefficient (i.e., lower performance than it could be) execution and why. (Leave blank if you answered NO above.)

If YES, explain what you would recommend to your friend to eliminate the inefficiency. (Leave blank if you answered NO above.)

If NO, justify how the design is as efficient as Tomasulo's algorithm with full data forwarding can be. (Leave blank if you answered YES above.)

If NO, explain how the design can be simplified. (Leave blank if you answered YES above.)

2. Branch Prediction and Dual Path Execution [25 points]

Assume a machine with a 7-stage pipeline. Assume that branches are resolved in the sixth stage. Assume that 20% of instructions are branches.

(a) How many instructions of wasted work are there per branch misprediction on this machine?

[ ] instructions.

(b) Assume N instructions are on the correct path of a program and assume a branch predictor accuracy of A. Write the equation for the number of instructions that are fetched on this machine in terms of N and A. (Please show your work for full credit.)

(c) Let's say we modified the machine so that it used *dual path execution* like we discussed in class (where an equal number of instructions are fetched from each of the two branch paths). Assume branches are resolved before new branches are fetched. Write how many instructions would be fetched in this case, as a function of N. (Please show your work for full credit.)

(d) Now let's say that the machine combines branch prediction *and* dual path execution in the following way:

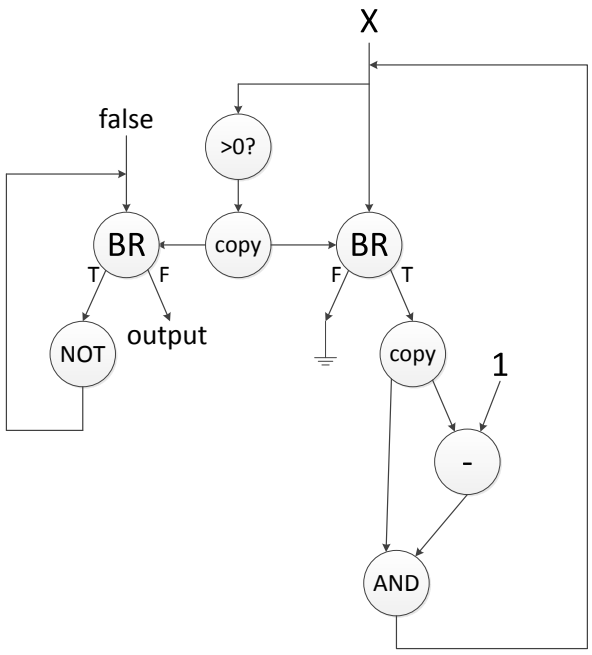
A branch confidence estimator, like we discussed in class, is used to gauge how confident the machine is of the prediction made for a branch. When confidence in a prediction is high, the branch predictor's prediction is used to fetch the next instruction; When confidence in a prediction is low, dual path execution is used instead.

Assume that the confidence estimator estimates a fraction C of the branch predictions have high confidence, and that the probability that the confidence estimator is wrong in its high confidence estimation is M.

Write how many instructions would be fetched in this case, as a function of N, A, C, and M. (Please show your work for full credit.)

3. Dataflow [20 points]

Here is a dataflow graph representing a dataflow program:



The following is a description of the nodes used in the dataflow graph:

-	subtracts right input from left input
AND	bit-wise AND of two inputs
NOT	the boolean negation of the input (input and output are both boolean)
BR	passes the input to the appropriate output corresponding to the boolean condition
copy	passes the value from the input to the two outputs
>0?	true if input greater than 0

Note that the input X is a non-negative integer.

What does the dataflow program do? Specify clearly in less than 15 words.

4. Mystery Instruction [40 points]

That pesky engineer implemented yet another mystery instruction on the LC-3b. It is your job to determine what the instruction does. The mystery instruction is encoded as:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1010				DR			SR1			0	0	0	0	0	0

The modifications we make to the LC-3b datapath and the microsequencer are highlighted in the attached figures (see the next two pages). We also provide the original LC-3b state diagram, in case you need it. (As a reminder, the selection logic for SR2MUX is determined internally based on the instruction.)

The additional control signals are

GateTEMP1/1: NO, YES

GateTEMP2/1: NO, YES

LD.TEMP1/1: NO, LOAD

LD.TEMP2/1: NO, LOAD

ALUK/3: OR1 (A|0x1), LSHF1 (A<<1), PASSA, PASS0 (Pass value 0), PASS16 (Pass value 16)

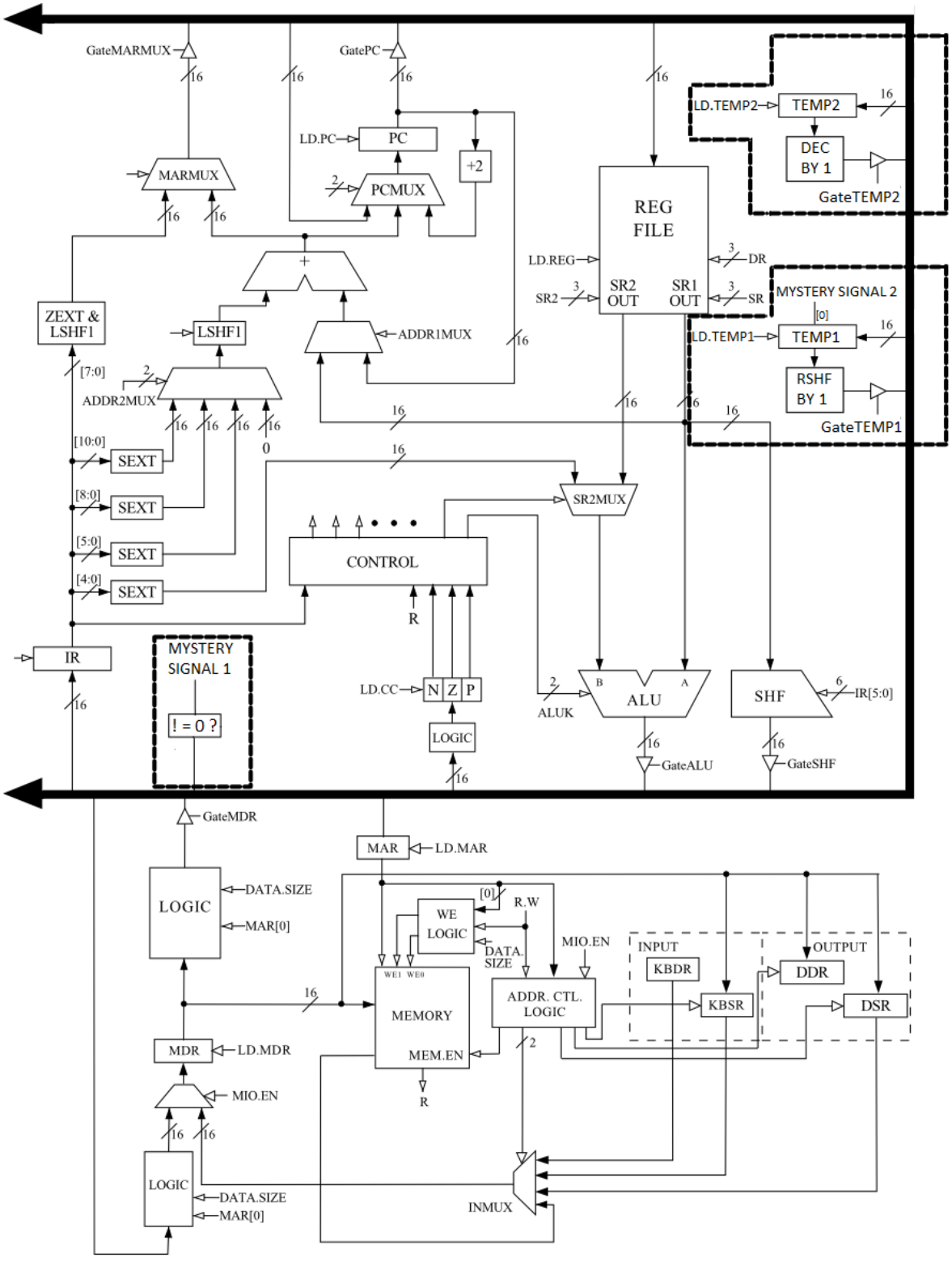
COND/4:
$COND_{0000}$;Unconditional
$COND_{0001}$;Memory Ready
$COND_{0010}$;Branch
$COND_{0011}$;Addressing mode
$COND_{0100}$;Mystery 1
$COND_{1000}$;Mystery 2

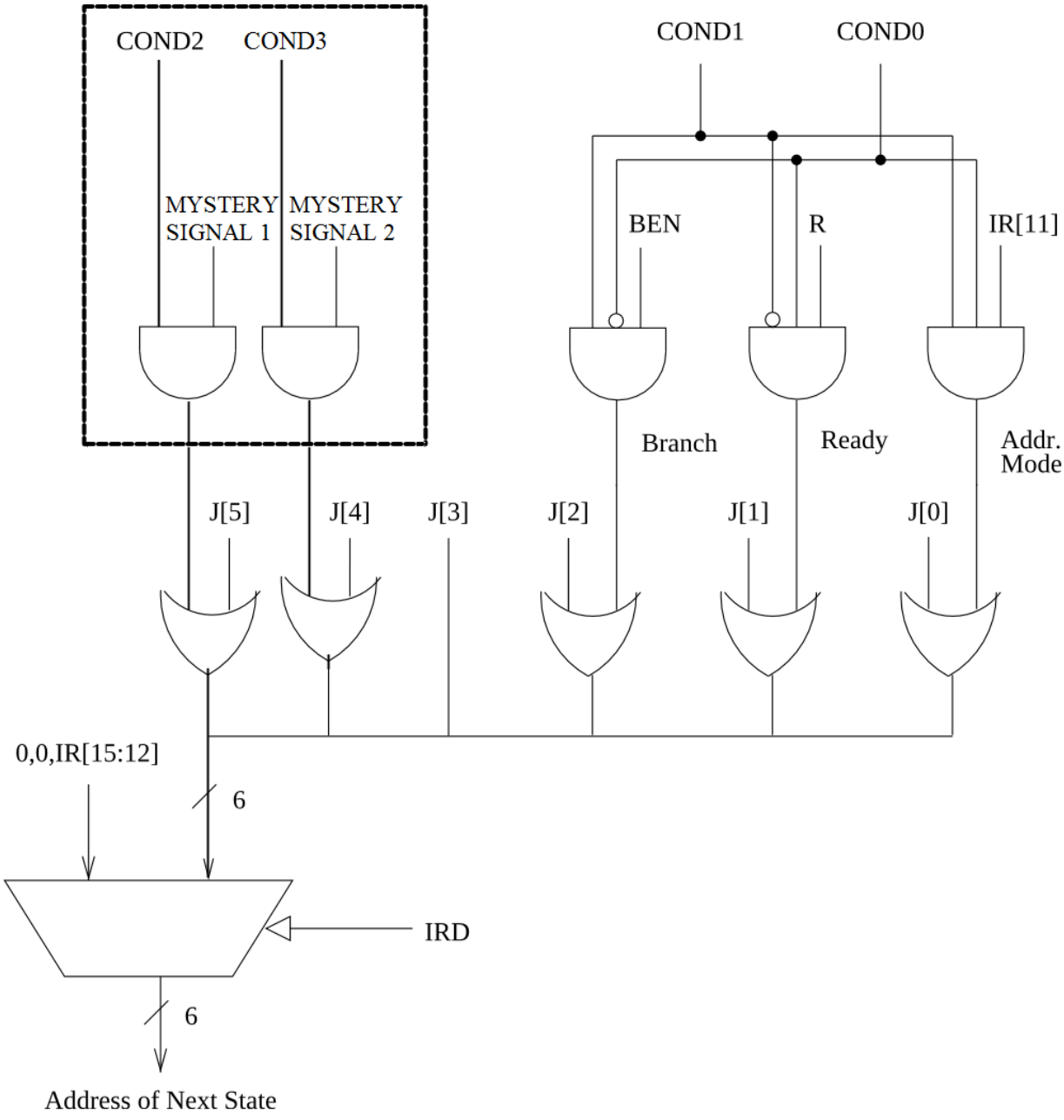
The microcode for the instruction is given in the table below.

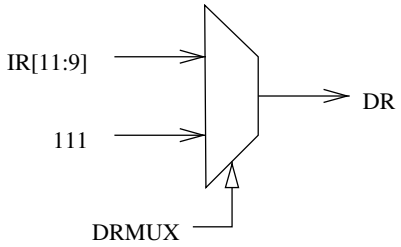
State	Cond	J	Asserted Signals
001010 (10)	$COND_{0000}$	001011	ALUK = PASS0, GateALU, LD.REG, DRMUX = DR (IR[11:9])
001011 (11)	$COND_{0000}$	101000	ALUK = PASSA, GateALU, LD.TEMP1, SR1MUX = SR1 (IR[8:6])
101000 (40)	$COND_{0000}$	110010	ALUK = PASS16, GateALU, LD.TEMP2
110010 (50)	$COND_{1000}$	101101	ALUK = LSHF1, GateALU, LD.REG, SR1MUX = DR, DRMUX = DR (IR[11:9])
111101 (61)	$COND_{0000}$	101101	ALUK = OR1, GateALU, LD.REG, SR1MUX = DR, DRMUX = DR (IR[11:9])
101101 (45)	$COND_{0000}$	111111	GateTEMP1, LD.TEMP1
111111 (63)	$COND_{0100}$	010010	GateTEMP2, LD.TEMP2

Describe what this instruction does.

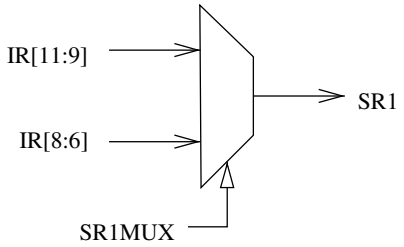
Initials:


GateMARMUX
GatePC
LD.PC
PC
+2
PCMUX
MARMUX
ZEXT & LSHF1
LSHF1
ADDR1MUX
ADDR2MUX
[7:0]
[10:0]
[8:0]
[5:0]
[4:0]
SEXT
SEXT
SEXT
SEXT
0
REG FILE
LD.REG
SR2 OUT
SR1 OUT
DR
SR
SR2
LD.TEMP2
TEMP2
DEC BY 1
GateTEMP2
MYSTERY SIGNAL 2
LD.TEMP1
TEMP1
RSHF BY 1
GateTEMP1
SR2MUX
CONTROL
R
IR
MYSTERY SIGNAL 1
! = 0 ?
LD.CC
N Z P
LOGIC
ALUK
ALU
SHF
IR[5:0]
GateALU
GateSHF
GateMDR
MAR
LD.MAR
LOGIC
DATA.SIZE
MAR[0]
WE LOGIC
[0]
R.W
DATA. SIZE
MIO.EN
INPUT
OUTPUT
KBDR
DDR
KBSR
DSR
WE1 WE0
MEMORY
MEM.EN
ADDR. CTL. LOGIC
MDR
LD.MDR
MIO.EN
LOGIC
DATA.SIZE
MAR[0]
INMUX

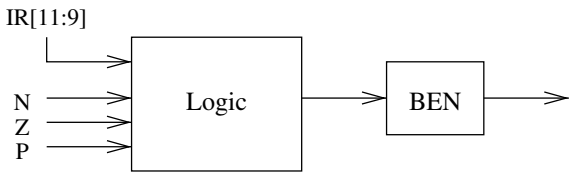

COND2
COND3
COND1
COND0
MYSTERY SIGNAL 1
MYSTERY SIGNAL 2
BEN
R
IR[11]
Branch
Ready
Addr. Mode
J[5]
J[4]
J[3]
J[2]
J[1]
J[0]
0,0,IR[15:12]
6
IRD
6
Address of Next State



(a)

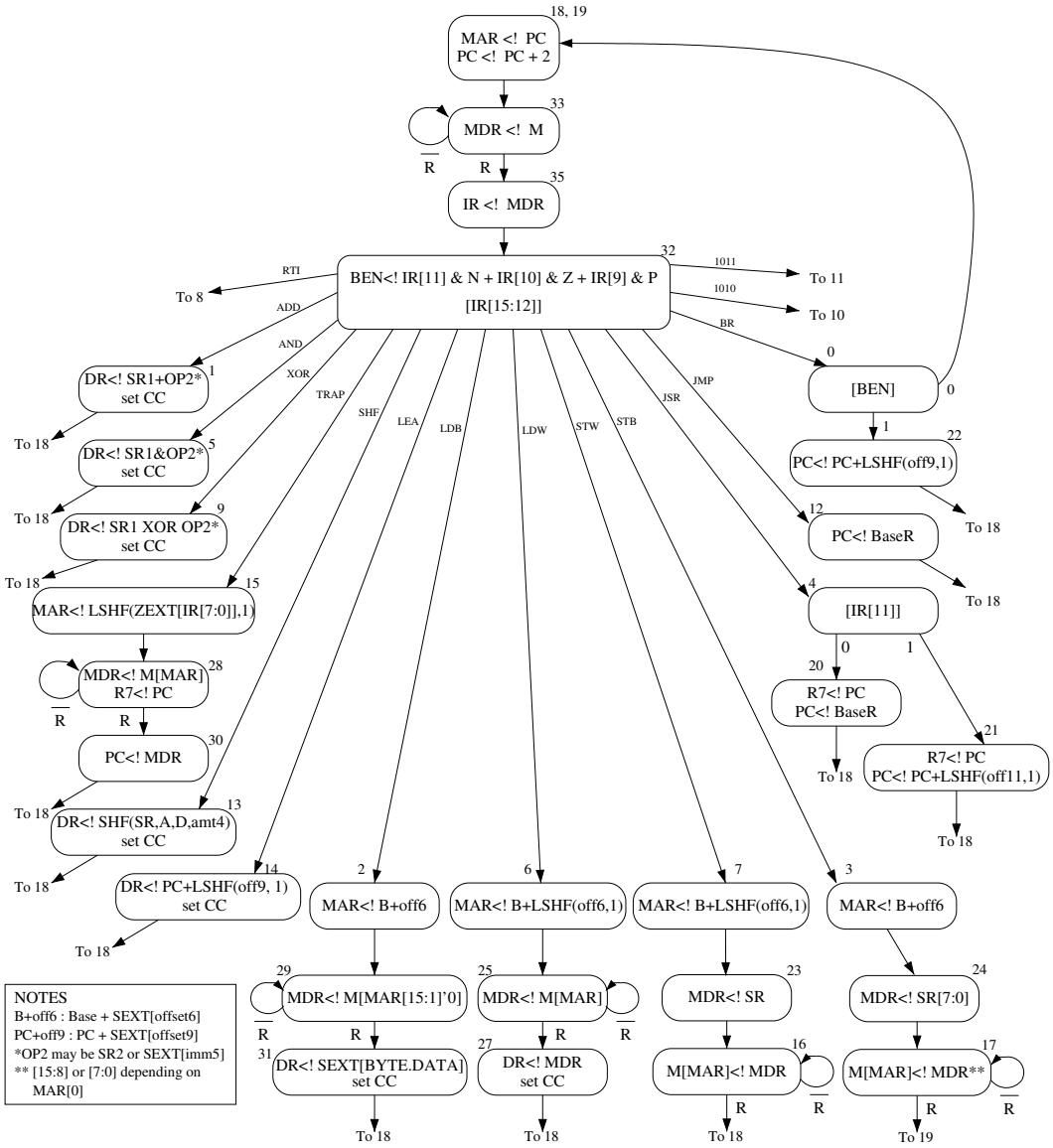


(b)



(c)

Initials:


18, 19
MAR <! PC
PC <! PC + 2
33
MDR <! M
R
R
35
IR <! MDR
32
BEN<! IR[11] & N + IR[10] & Z + IR[9] & P
[IR[15:12]]
RTI
To 8
1011
To 11
1010
To 10
BR
ADD
AND
XOR
TRAP
SHF
LEA
LDB
LDW
STW
STB
JSR
JMP
0
[BEN]
0
1
22
PC<! PC+LSHF(off9,1)
To 18
1
DR<! SR1+OP2*
set CC
To 18
5
DR<! SR1&OP2*
set CC
To 18
9
DR<! SR1 XOR OP2*
set CC
To 18
12
PC<! BaseR
To 18
15
MAR<! LSHF(ZEXT[IR[7:0]],1)
4
[IR[11]]
0
1
28
MDR<! M[MAR]
R7<! PC
R
R
20
R7<! PC
PC<! BaseR
To 18
21
R7<! PC
PC<! PC+LSHF(off11,1)
To 18
30
PC<! MDR
To 18
13
DR<! SHF(SR,A,D,amt4)
set CC
To 18
14
DR<! PC+LSHF(off9, 1)
set CC
To 18
2
MAR<! B+off6
6
MAR<! B+LSHF(off6,1)
7
MAR<! B+LSHF(off6,1)
3
MAR<! B+off6
29
MDR<! M[MAR[15:1]'0]
R
R
31
DR<! SEXT[BYTE.DATA]
set CC
To 18
25
MDR<! M[MAR]
R
R
27
DR<! MDR
set CC
To 18
23
MDR<! SR
16
M[MAR]<! MDR
R
R
To 18
24
MDR<! SR[7:0]
17
M[MAR]<! MDR**
R
R
To 19
NOTES
B+off6 : Base + SEXT[offset6]
PC+off9 : PC + SEXT[offset9]
*OP2 may be SR2 or SEXT[imm5]
** [15:8] or [7:0] depending on
MAR[0]

5. Virtual Memory [40 points]

Suppose a 32 K×8 K matrix A with 1-byte elements is stored in row major order in virtual memory. Assume only the program in question occupies space in physical memory. Show your work for full credit.

Program 1

```
for (i = 0; i < 32768; i++)
  for (j = 0; j < 8192; j++)
    A[i][j] = A[i][j] * A[i][j];
```

Program 2

```
for (j = 0; j < 8192; j++)
  for (i = 0; i < 32768; i++)
    A[i][j] = A[i][j] * A[i][j];
```

(a) If Program 1 yields 8 K page faults, what is the size of a page in this architecture?

Assume the page size you calculated for the rest of this question.

(b) Consider Program 2. How many pages should the physical memory be able to store to ensure that Program 2 experiences the same number of page faults as Program 1 does?

(c) Consider Program 2. How many page faults would Program 2 experience if the physical memory can store 1 page?

What about if the physical memory can store 4 K pages?

(d) Now suppose the same matrix is stored in column-major order. And, the physical memory size is 32 MB.

How many page faults would Program 1 experience?

How many page faults would Program 2 experience?

(e) Suppose still that the same matrix is stored in column-major order. However, this time the physical memory size is 8 MB.

How many page faults would Program 1 experience?

How many page faults would Program 2 experience?

6. Future File [40 points]

For this question, assume a machine with the following characteristics:

- Scalar, out-of-order dispatch with a 4-entry reorder buffer, future file, and full data forwarding.
- A 4-stage pipeline consisting of fetch, decode, execute, and writeback.
- Fetch and decode take 1 cycle each.
- Writeback takes 2 cycles and updates the future file and the reorder buffer.
- When the reorder buffer is filled up, fetch is halted.

A program that consists of three instructions: `ADD`, `DIV`, `LD` that have the following semantics:

- `ADD Rd ← Rs, Rt`: Adds the contents of `Rs` and `Rt` and stores the result in `Rd`.
- `DIV Rd ← Rs, Rt`: Divides the contents of `Rs` by the contents of `Rt` and stores the result in `Rd`. Raises an exception if `Rt` is zero.
- `LD Rd ← Rs, Rt`: Loads the contents of the base memory address `Rs` at the offset `Rt` and stores the result in `Rd`. Assume that calculated memory addresses are guaranteed to be 4-byte-aligned and the memory is bit-addressable.

An `ADD` instruction takes 1 cycle to execute, a `DIV` instruction takes 3 cycles to execute and a divide-by-zero exception, if present, is detected during the second cycle, and a `LD` instruction takes 5 cycles to execute.

Here is the state of the future file in the machine at the end of the cycle when a divide-by-zero exception is detected:

Future File

	V	Value
R1	0	21
R2	1	13
R3	1	0
R4	1	3
R5	1	25
R6	1	1
R7	1	17
R8	1	8
R9	1	9
R10	0	23
R11	1	7
R12	1	19

Using what you know about the reorder buffer and the future file, fill in the missing contents of the reorder buffer in the machine. Assume reorder buffer entries are allocated from top to bottom in the diagram.

Reorder Buffer

	V	Exception?	Opcode	Rd	Rs	Rt	Dest. Value	Dest. Value Ready
Oldest →	1			R1	R12			
	1						3	1
	1			R7				1
Youngest →	1			R10				

7. Branch Prediction [35 points]

Assume the following piece of code that iterates through a large array populated with **completely (i.e., truly) random** positive integers. The code has four branches (labeled `B1`, `B2`, `B3`, and `B4`). When we say that a branch is *taken*, we mean that the code *inside* the curly brackets is executed.

```
for (int i=0; i<N; i++) { /* B1 */
    val = array[i];       /* TAKEN PATH for B1 */
    if (val % 2 == 0) {   /* B2 */
        sum += val;       /* TAKEN PATH for B2 */
    }
    if (val % 3 == 0) {   /* B3 */
        sum += val;       /* TAKEN PATH for B3 */
    }
    if (val % 6 == 0) {   /* B4 */
        sum += val;       /* TAKEN PATH for B4 */
    }
}
```

(a) Of the four branches, list all those that exhibit *local correlation*, if any.

(b) Which of the four branches are *globally correlated*, if any? Explain in less than 20 words.

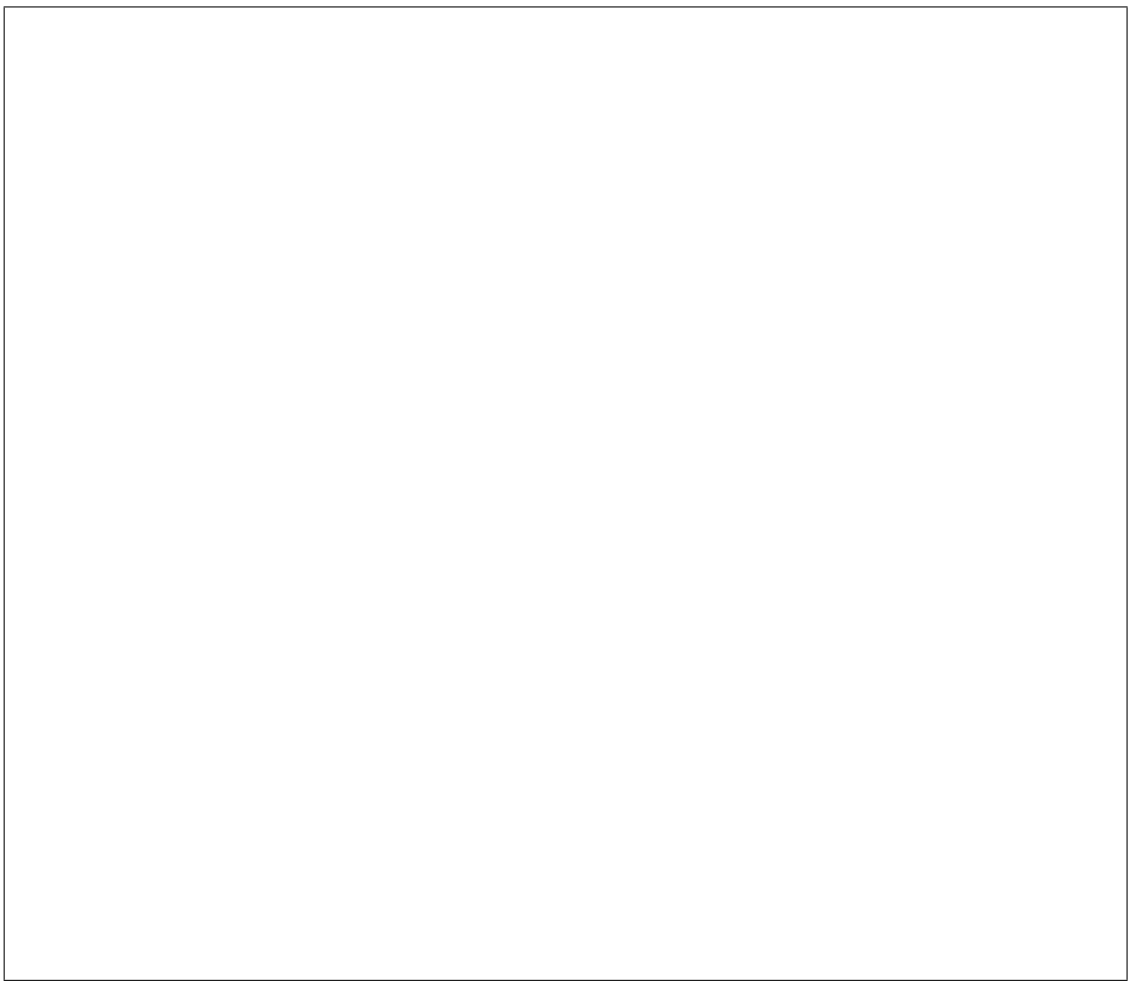
Now assume that the above piece of code is running on a processor that has a global branch predictor. The global branch predictor has the following characteristics.

- Global history register (GHR): 2 bits.
- Pattern history table (PHT): 4 entries.
- Pattern history table entry (PHTE): 11-bit signed saturating counter (possible values: -1024–1023)
- Before the code is run, all PHTEs are initially set to 0.
- As the code is being run, a PHTE is incremented (by one) whenever a branch that corresponds to that PHTE is taken, whereas a PHTE is decremented (by one) whenever a branch that corresponds to that PHTE is not taken.

(d) After 120 iterations of the loop, calculate the **expected** value for only the first PHTE and fill it in the shaded box below. (Please write it as a base-10 value, rounded to the nearest one's digit.)

Hint. For a given iteration of the loop, first consider, what is the probability that both B1 and B2 are taken? Given that they are, what is the probability that B3 will increment or decrement the PHTE? Then consider...

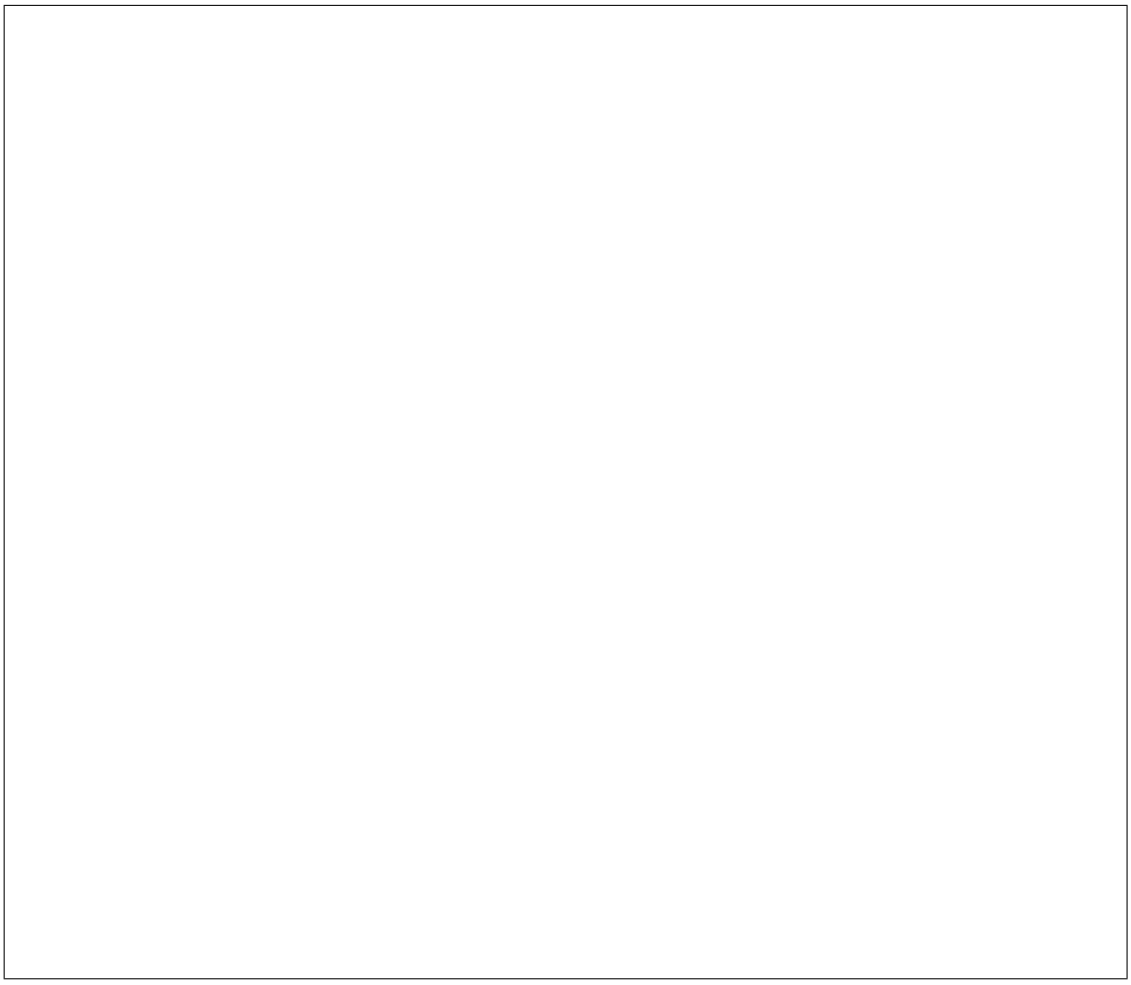
Show your work.

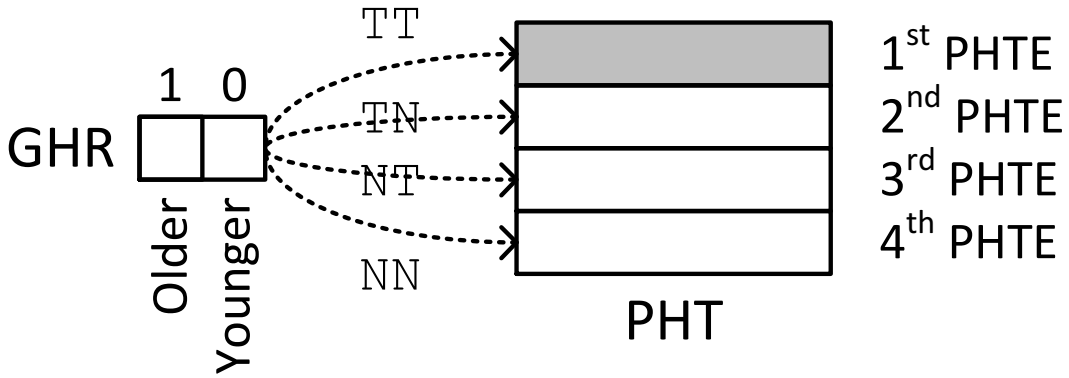


8. Bonus (Question 7 Continued) [45 points]

(a) Assume the same question in Part (d) of Question 7. Your job in this question is to fill in the rest of the PHTEs. In other words, after 120 iterations of the loop in Question 7, calculate the expected value for the rest of the PHTEs (i.e., PHTEs 2, 3, 4) and fill in the PHT below. (Please write them as base-10 values, rounded to the nearest one's digit.)

Show your work.





(b) After the first 120 iterations, let us assume that the loop continues to execute for another 1 billion iterations. What is the accuracy of this global branch predictor during the 1 billion iterations? (Please write it as a percentage, rounded to the nearest single-digit.)

Show your work.

(c) Without prior knowledge of the contents of the array, what is the highest accuracy that any type of branch predictor can achieve during the same 1 billion iterations as above? (Please write it as a percentage, rounded to the nearest single-digit.)

Show your work.

Stratchpad

Stratchpad

Stratchpad

Stratchpad

Stratchpad

Stratchpad

Stratchpad