Department of Electrical and Computer Engineering

The University of Texas at Austin
EE 460N, Spring 2017
Problem Set 2
Due: February 13, before class
Yale N. Patt, Instructor
Chirag Sakhuja, Sarbartha Banerjee, Jon Dahm, Arjun Teh, TAs

Name: Xinyuan (Allen) Pan
EID: xp572
Name: Qinyu Diao
EID: qd572

Instructions

You are encouraged to work on the problem set in groups and turn in one problem set for the entire group. The problem sets are to be submitted on Canvas. Only one student should submit the problem set on behalf of the group. The only acceptable file format is PDF. Include the name of all students in the group in the file.

Note: You still need to bring a hard copy of your student information sheet to the class on Monday.

You will need to refer to the assembly language handouts and the LC-3b ISA on the course website.

Questions

/ This question was moved here from Problem Set 1 */*

Problem 1

The following program computes the square (k*k) of a positive integer k, stored in location `0x4000` and stores the result in location `0x4002`. The result is to be treated as a 16-bit unsigned number.

Assumptions:

- A memory access takes 5 cycles
- The system call initiated by the `HALT` instruction takes 20 cycles to execute. This **does not** include the number of cycles it takes to execute the `HALT` instruction itself.

```
.ORIG X3000
AND R0, R0, #0  9
LEA R3, NUM     9
LDW R3, R3, #0  15
LDW R1, R3, #0  15
```

```
        ADD R2, R1, #0  9
LOOP    ADD R0, R0, R1  9
        ADD R2, R2, #-1 9
        BRP LOOP        9 or 10
        STW R0, R3, #1  15
        HALT            36
NUM     .FILL x4000
        .END
```

1. How many cycles does each instruction take to execute on the LC-3b microarchitecture described in Appendix C?
2. How many cycles does the entire program take to execute? (answer in terms of k)
3. What is the maximum value of k for which this program still works correctly? Note: Treat the input and output values as 16-bit unsigned values for part c. We will extend the problem to 2's complement values in part d.
4. How will you modify this program to support negative values of k? Explain in less than 30 words.
5. What is the new range of k?

1. labelled on each line

2. $57 + 28 \times (k-1) + 27 + 15 + 36 = 107 + 28k$

3. 255

4. Check the sign of R1, make R1 = -R1 if it's negative (therefore R2)

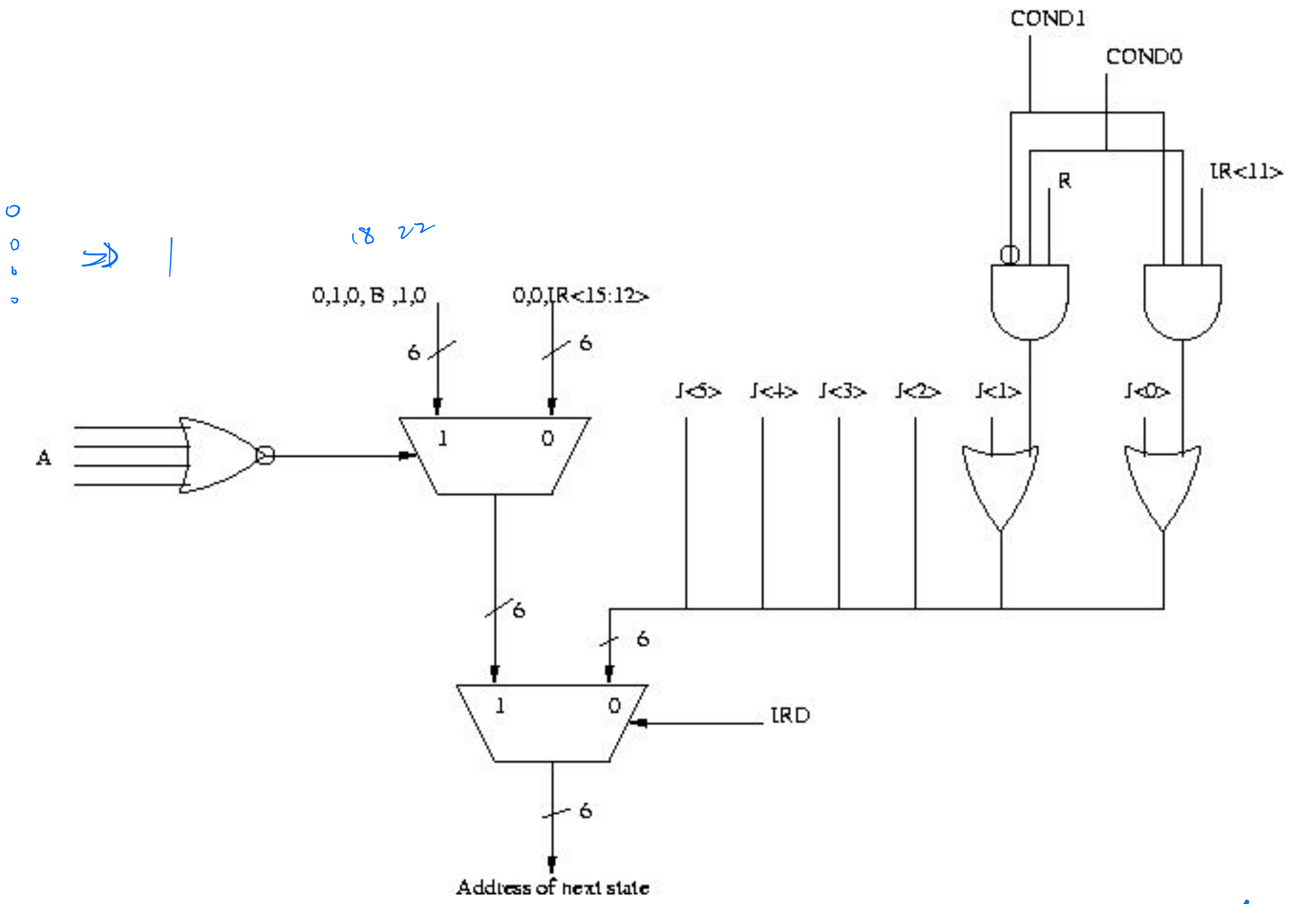
5. [-181, 181]

/ This question was moved here from Problem Set 1 */*

Problem 2

Please answer the following questions:

1. In which state(s) in the LC-3b state diagram should the LD.BEN signal be asserted? Is there a way for the LC-3b to work correctly without the LD.BEN signal? Explain.
2. Suppose we want to get rid of the BEN register altogether. Can this be done? If so, explain how. If not, why not? Is it a good idea? Explain.
3. Suppose we took this further and wanted to get rid of state 0. We can do this by modifying the microsequencer, as shown in the figure below. What is the 4-bit signal denoted as A in the figure? What is the 1-bit signal denoted as B?



1. state 32. Let BEN register be constantly loaded. BEN is only used in state 0 and N, Z, P would have already been set to the desired value.

2. Yes. At state 0, directly hook IR[11]&N|IR[10]&Z|IR[9]&P to the mux in the microsequencer (in place of BEN). This is not a good idea because moving this step to the execute stage will increase the time needed for that stage.

3. A is the opcode, IR[15:12], and B is IR[11]&N|IR[10]&Z|IR[9]&P

/ This question was moved here from Problem Set 1 */*

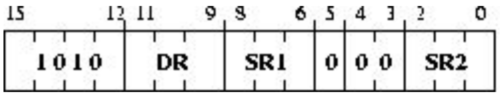
Problem 3

We wish to use the unused opcode “1010” to implement a new instruction ADDM, which (similar to an IA-32 instruction) adds the contents of a memory location to either the contents of a register or an immediate value and stores the result into a register. The specification of this instruction is as follows:

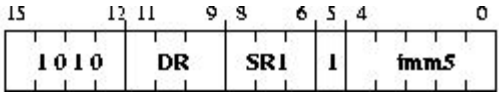
Assembler Formats

ADDM DR, SR1, SR2
ADDM DR, SR1, imm5

Encodings



15 12	11 9	8 6	5	4 3	2 0
1010	DR	SR1	0	0 0	SR2

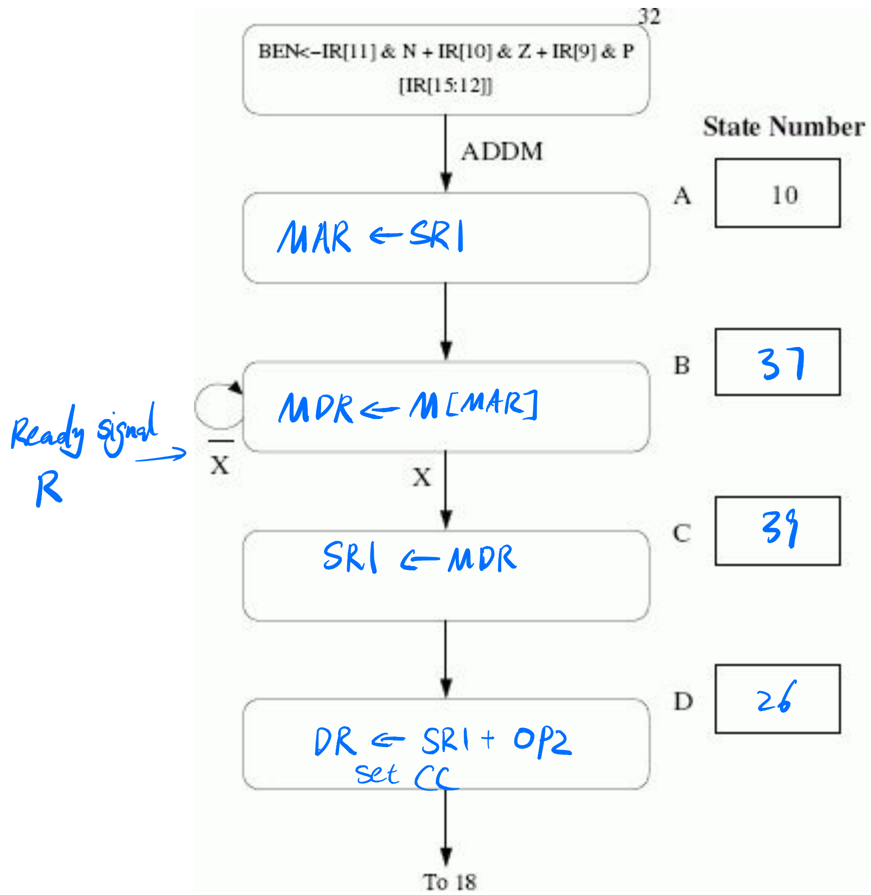


15 12	11 9	8 6	5	4 0
1010	DR	SR1	1	imm5

Operation

```
if (bit[5] == 0)
    DR = Memory[SR1] + SR2;
else
    DR = Memory[SR1] + SEXT(imm5);
setcc(DR);
```

1. We show below an addition to the state diagram necessary to implement ADDM. Using the notation of the LC-3b State Diagram, describe inside each “bubble” what happens in each state, and assign each state an appropriate state number (state A has been done for you). Also, what is the one-bit signal denoted as X in the figure? Note: Be sure your solution works when the same register is used for both sources and the destination (eg., ADDM R1, R1, R1).
   - *Hint: states 26, 34, and 36-63 in the control store are available*
   - *Hint: to make ADDM work when the same register is used for both sources and destination, you will need to change the datapath. Part b asks you to show the necessary changes to the datapath*



32	16	8	4	2	1
1	0	0	1	0	1

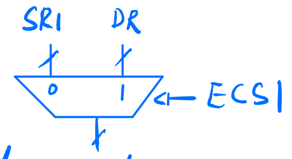
2. Add to the Data Path any additional structures and any additional control signals needed to implement ADDM. Label the additional control signals ECS 1 (for "extra control signal 1"), ECS 2, etc.
3. The processing in each state A,B,C,D is controlled by asserting or negating each control signal. Enter a 1 or a 0 as appropriate for the microinstructions corresponding to states A,B,C,D.
   - *Clarification: for ease of grading, only fill in the control values that are non-zero; entries you leave blank will be assumed to be 0 when we grade*
   - *Clarification: for the encoding of the control signals, see table C.1 of Appendix C. For each control signal, assume that the 1st signal value in the list is encoded as 0, the the 2nd value encoded as a 1, etc.*

2. add a control signal ECS1, taking value NO and YES.

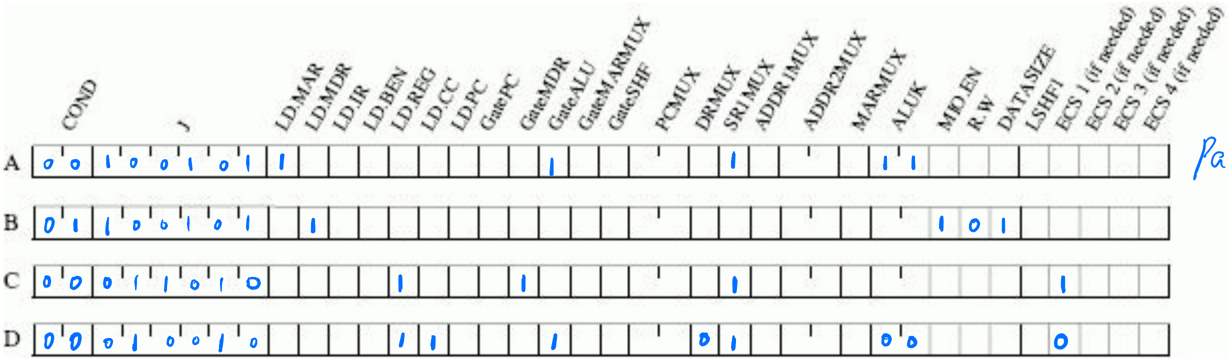
ECS1 and LD.REG together determines which register to load the data.

As we know when LD.REG = YES, register specified by DR will load the data.

Now add thus a mux:



the output is then used to load the data when LD.REG is asserted.



	COND	J	LD.MAR	LD.MDR	LD.IR	LD.BEN	LD.REG	LD.CC	LD.PC	GatePC	GateMDR	GateALU	GateMARMUX	GateSHF	PCMUX	DRMUX	SR1MUX	ADDR1MUX	ADDR2MUX	MARMUX	ALUK	MIO.EN	R.W	DATASIZE	LSHF1	ECS 1 (if needed)	ECS 2 (if needed)	ECS 3 (if needed)	ECS 4 (if needed)
A	00	100101	1									1					1				11								
B	01	100101		1																		1	0	1					
C	00	011010					1				1						1									1			
D	00	010010					1	1				1				0	1				00					0			

PassA

/ This question was moved here from Problem Set 1 */*

Problem 4

The Address Control Logic in the LC-3b datapath of Figure C.3 in Appendix C allows the LC-3b to support memory-mapped I/O. There are three inputs to this logic:

- 16-bit address in `MAR`. This signal can take the following values: `xFE00`, `xFE02`, `xFE04`, `xFE06`, and `OTHER` (any other address between `x0000` and `xFDFF`).
- 1-bit `control` signal `R.W`. The access is a read access if this signal is `R`, write access if it is `W`.
- 1-bit control signal `MIO.EN`. If this signal is 1, a memory or I/O access should be performed in this cycle.

The logic has five outputs:

- 1-bit `MEM.EN` signal. Memory is enabled if this signal is 1.
- 2-bit select signal for `INMUX`. This signal can take the following values: `KBDR`, `KBSR`, `DSR`, `MEMORY`.
- 1-bit `LD.KBSR` signal. `KBSR` will be load-enabled at the end of the current cycle if this signal is 1.
- 1-bit `LD.DDR` signal. `DDR` will be load-enabled at the end of the current cycle if this signal is 1.
- 1-bit `LD.DSR` signal. `DSR` will be load-enabled at the end of the current cycle if this signal is 1.

Your task is to draw the truth table for this Address Control Logic. Mark don't care values with "X" in your truth table. Use the conventions described above to denote the values of inputs and outputs. Please read Section C.6 in Appendix C on memory-mapped I/O before answering this question. Also, refer to table A.3 of Appendix A to find out the addresses of device registers.

MAR	MIO.EN	R.W	MEM.EN	INMUX	LD.KBSR	LD.DSR	LD.DDR
xFE00	0	R	0	X	0	0	0
	0	W	0	X	0	0	0
	1	R	0	KBSR	0	0	0
	1	W	0	X	1	0	0
xFE02	0	R	0	X	0	0	0
	0	W	0	X	0	0	0
	1	R	0	KBDR	0	0	0
	1	W	0	X	0	0	0
xFE04	0	R	0	X	0	0	0
	0	W	0	X	0	0	0
	1	R	0	DSR	0	0	0
	1	W	0	X	0	1	0
xFE06	0	R	0	X	0	0	0
	0	W	0	X	0	0	0
	1	R	0	X	0	0	0
	1	W	0	X	0	0	1
other	0	R	0	X	0	0	0
	0	W	0	X	0	0	0
	1	R	1	MEMORY	0	0	0
	1	W	1	X	0	0	0

/ This question was moved here from Problem Set 1 */*

Problem 5

The LC-3b state diagram handed out in class contained errors in states 4, 20, and 21. We have posted both versions of the handout: wrong and corrected. Briefly explain the problem we have corrected.

In the wrong version, PC is stored in R7 before increment. So when return back from subroutine, the instruction to execute will still be the JSR(R) instruction.

/ This question was moved here from Problem Set 1 */*

Problem 6

Answer the following short questions:

1. A memory's addressability is 64 bits. What does that tell you about the sizes of the MAR and the MDR?
2. We want to increase the number of registers that we can specify in the LC-3b ADD instruction to 32. Do you see any problem with that? Explain.

1. Doesn't tell us about MAR.
   MDR is 64-bit long.
2. The instruction is only 16 bit long.
   Having 32 registers will require 4+5×3+1 = 19 bits.

Problem 7

Given the following code:

```
MUL R3, R1, R2 9
ADD R5, R4, R3 7
ADD R6, R4, R1 7
MUL R7, R8, R9 9
ADD R4, R3, R7 7
MUL R10, R5, R6 9
```

Note: Each instruction is specified with the destination register first.

Calculate the number of cycles it takes to execute the given code on the following models:

1. A non-pipelined machine
2. A pipelined machine with scoreboarding and five adders and five multipliers.
3. A pipelined machine with scoreboarding and one adder and one multiplier.

Note: For all machine models, use the basic instruction cycle as follows:

- Fetch (one clock cycle)
- Decode (one clock cycle)
- 6 Execute (MUL takes 6, ADD takes 4 clock cycles). The multiplier and the adder are not pipelined.
- Write-back (one clock cycle)

Do not forget to list any assumptions you make about the pipeline structure (e.g., data forwarding between pipeline stages). In fact, we encourage you to solve the above mentioned questions with data forwarding as well, but, you are not required to do so.

1. 48 cycles

2. 24 cycles (with data forwarding)

3. 27 cycles (with data forwarding)

```
2. F D E E E E E E W
     F D S S S S S E E E E W
       F | | | | | D E E E E W
                   F D E E E E E E W
                     F D S S S S S E E E E W
                       F | | | | | D E E E E E E W
```

```
3. F D E E E E E E W
     F D S S S S S E E E E W
       F | | | | | D S S S E E E E W
                   F | | | D E E E E E E W
                         F D S S S S S E E E E W
                           F | | | | | D E E E E E E W
```

Problem 8

Consider the following snapshots of the register file for an in-order-execution pipelined machine:

Cycle 0		
	Valid	Value
R0	1	0
R1	1	5
R2	1	7

End of Cycle 8		
	Valid	Value
R0	1	10
R1	0	5
R2	0	7

After Cycle 11		
	Valid	Value
R0	1	10
R1	1	17
R2	1	20

The program consists of three ADD instructions and finishes execution after the 11th cycle.

- The ADD instruction takes **three** cycles to execute.
- Fetch, Decode and Store all take **one** cycle.   1 1 3 1 = 6
- The machine has **two** adders
- There is NO data forwarding

1. What was the program that was executed? *Only R0, R1, and R2 were used*.

ADD R0 , R1 , R1   10 7 3

ADD R1 , R0 , R2   17 10 7

ADD R2 , R0 , R0   20 10 10

2. Complete the following timeline of the program execution. The first instruction has been

filled already.

Question: how does instruction 2 get the new value of R0 if it decodes (and read register) at cycle 3?
(no data forward)

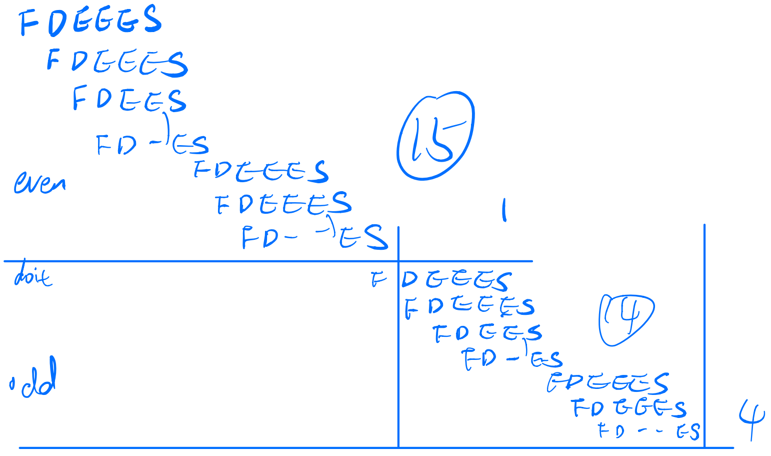
1	2	3	4	5	6	7	8	9	10	11
F	***D***	***E***	***E***	***E***	***S***					
	F	D	-	-	-	E	E	E	S	
		F	-	-	-	D	E	E	E	S

/* Updated on 2/10/2017 */

Problem 9

Suppose we have the following loop executing on a pipelined LC-3b machine.

```
DOIT    STW   R1, R6, #0
        ADD   R6, R6, #2
        AND   R3, R1, R2
        BRz   EVEN
odd     ADD   R1, R1, #3
        ADD   R5, R5, #-1
        BRp   DOIT
EVEN    ADD   R1, R1, #1
        ADD   R7, R7, #-1
        BRp   DOIT
```



Assume that before the loop starts, the registers have the following **decimal** values stored in them:

Register	Value
R0	0
R1	0
R2	1
R3	0
R4	0
R5	5
R6	4000

R7 5

14x7

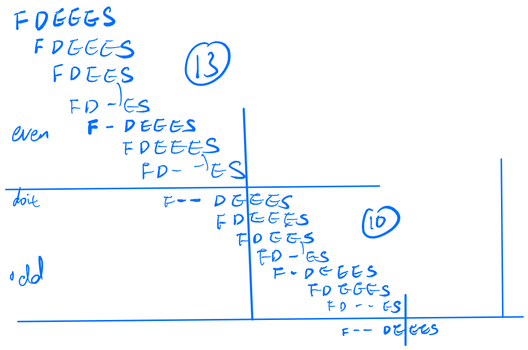
✓ 5
✓ 8
✓ 9
✓ 12
✓ 13
✓ 16
✓ 17

1 4 5 8 9

1 0 1 0

4 3 2 1

4 3 2 1 0



13+10x8 = 93

R7	5

The fetch stage takes **one** cycle, the decode stage also takes **one** cycle, the execute stage takes a variable number of cycles depending on the type of instruction (see below), and the store stage takes **one** cycle.

All execution units (including the load/store unit) are fully pipelined and the following instructions that use these units take the indicated number of cycles:

Instruction	**Number of Cycles**
STW	3
ADD	3
AND	2
BR	1

Data forwarding is used wherever possible. Instructions that are dependent on the previous instructions can make use of the results produced right after the previous instruction finishes the execute stage. Multiple Instructions can write the results to the register file concurrently in the same cycle.

The target instruction after a branch can be fetched when the BR instruction is in ST stage. For example, the execution of an ADD instruction followed by a BR would look like:

ADD	F	D	E1	E2	E3	ST		
BR		F	D	-	-	E1	ST	
TARGET							F	D

The pipeline implements "in-order execution." A scoreboarding scheme is used as discussed in class.

Answer the following questions:

1. How many cycles does the above loop take to execute if no branch prediction is used?
2. How many cycles does the above loop take to execute if all branches are predicted with 100% accuracy.
3. How many cycles does the above loop take to execute if a static BTFN (backward taken-forward not taken) branch prediction scheme is used to predict branch directions? What is the overall branch prediction accuracy? What is the prediction accuracy for each

branch?

1. 127 cycles
2. 93 cycles
3. 99 cycles

$\frac{12}{18}$ = 66.7% accuracy

Problem 10

Consider the following program:

```
  AND R0, R0, #0
  BRz A
  ADD R1, R0, #-1
C BRp B
A ADD R2, R0, #1
  BRp C
B RET
```

R0 = 0

R2 = 1

BRz EVEN 4 times right 5 times wrong $\frac{4}{9}$

BRp Doit (top) 4 times all right 100%

BRp Doit (bottom) 4 times right 1 time right 80%

A processor implements the GAg branch predictor as part of its microarchitecture. Assume the Branch History Register and Pattern History Table are as shown below before the progam is run. The direction of the most recent branch is the right-most bit of the BHR; i.e., 1=taken, 0=not taken.

Branch History Register

```
101
```

Branch History Table

0	10
1	00
2	10
3	01
4	10
5	11
6	00
7	01

1. How many times does the branch predictor predict correctly?
2. What does the Branch History Register look like after the program finishes? The Branch

History Table?

1. once

2. BHR $\boxed{111}$

0	10
1	00
2	10
3	10
4	10
5	11
6	00
7	10