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

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.

Update 02/09/17: The questions in red (11-13, inclusive) have been moved to PS3.

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
LEA R3, NUM
LDW R3, R3, #0
```

```
        LDW R1, R3, #0
        ADD R2, R1, #0
LOOP    ADD R0, R0, R1
        ADD R2, R2, #-1
        BRP LOOP
        STW R0, R3, #1
        HALT
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?

The comments indicate the number of cycles each instruction takes:

1.
```
    .ORIG X3000
        AND R0, R0, #0    ; 9 cycles
        LEA R3, NUM       ; 9 cycles
        LDW R3, R3, #0    ; 15 cycles
        LDW R1, R3, #0    ; 15 cycles
        ADD R2, R1, #0    ; 9 cycles
LOOP    ADD R0, R0, R1    ; 9 cycles
        ADD R2, R2, #-1   ; 9 cycles
        BRP LOOP          ; 10 cycles for Taken / 9 for Not
Taken
        STW R0, R3, #1    ; 15 cycles
        HALT              ; 35 cycles
NUM     .FILL x4000
        .END
```

To calculate the square of k, the inner loop gets executed k times. The branch is taken (k-1) times and not taken one time.

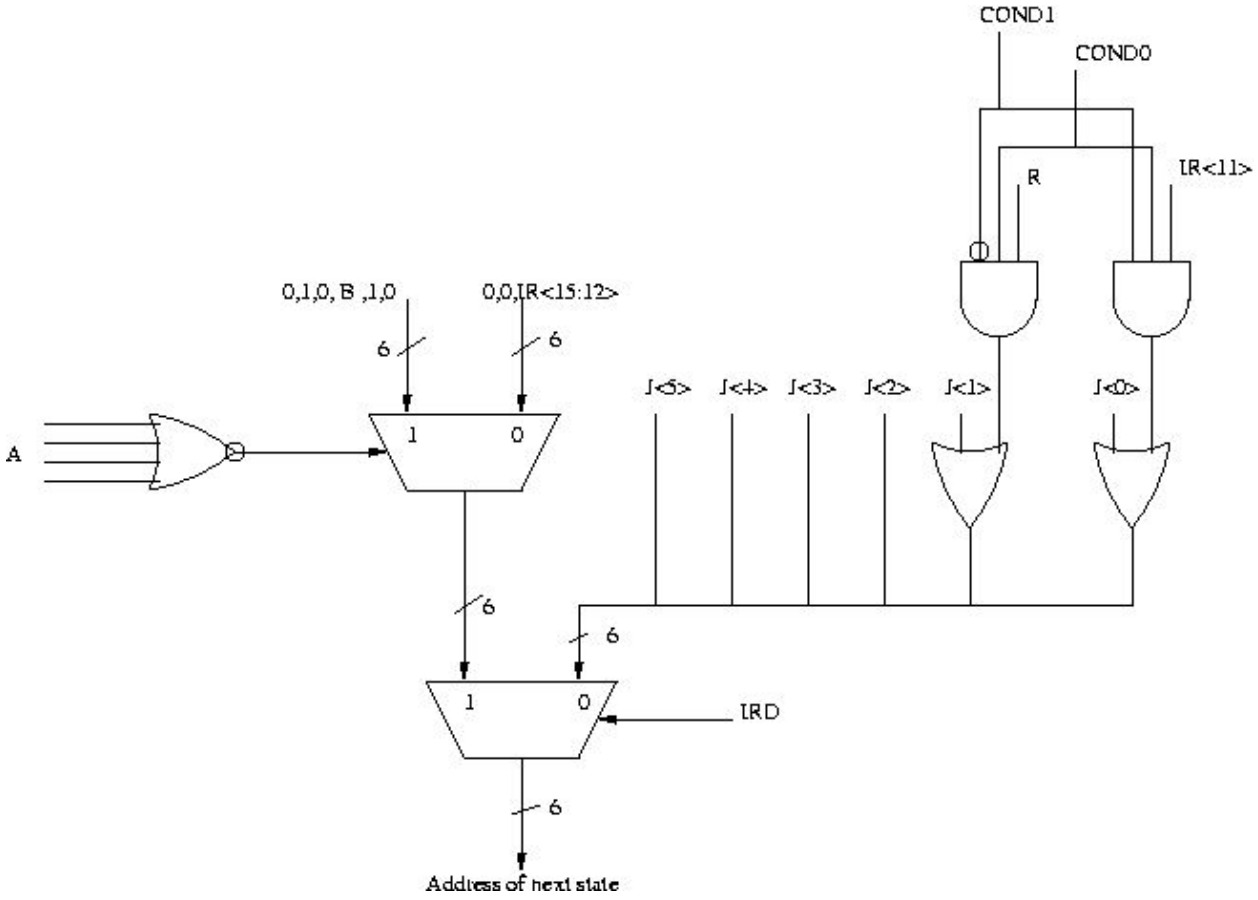
2. Number of cycles = 9 + 9 + 15 + 15 + 9 + (k-1)*(9 + 9 + 10) + 1*(9 + 9 + 9) + 15 + 35 = 28k + 106
3. k = 255
4. After we load the value of k, check if it is negative. If so, take the 2's complement before entering the loop.
5. k can range from -255 to +255

/ 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. We can get rid of the LD_BEN signal altogether and always load enable the BEN register.
2. The value that is loaded into BEN in state 32 could instead be calculated in state 0, but this would add delay for calculating the next state and would probably force the cycle time to be increased.
3. A = IR[15:12]
   B = IR[11]&N + IR[10]&Z + IR[9]&P (i.e., the old BEN signal)

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

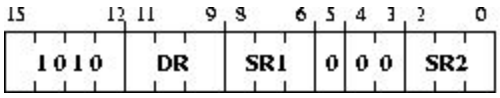
Problem 3

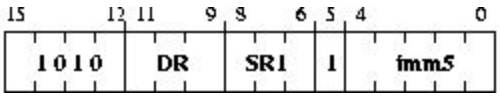
We wish to use the unused opcode “1010” to implement a new instruction `ADDM`, which (similar to an IA-32 instruction) adds the contents of a memory location to either the contents of a register or an immediate value and stores the result into a register. The specification of this instruction is as follows:

Assembler Formats

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

Encodings



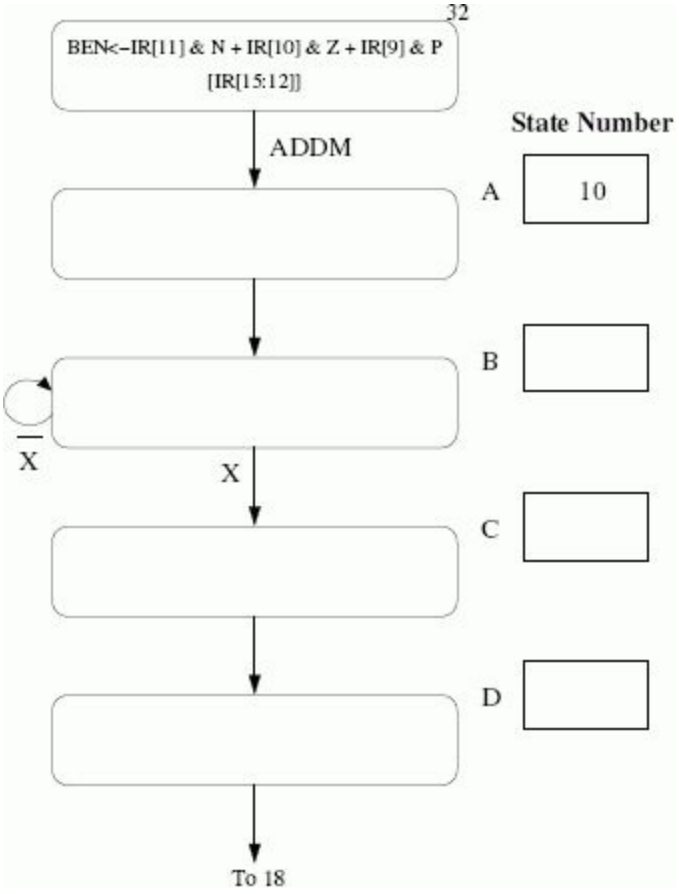


Operation

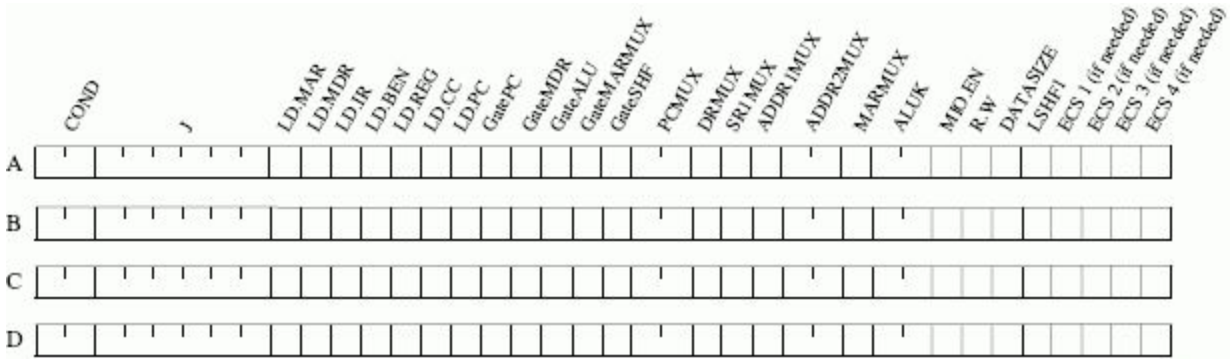
```
if (bit[5] == 0)
    DR = Memory[SR1] + SR2;
else
    DR = Memory[SR1] + SEXT(imm5);
setcc(DR);
```

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*

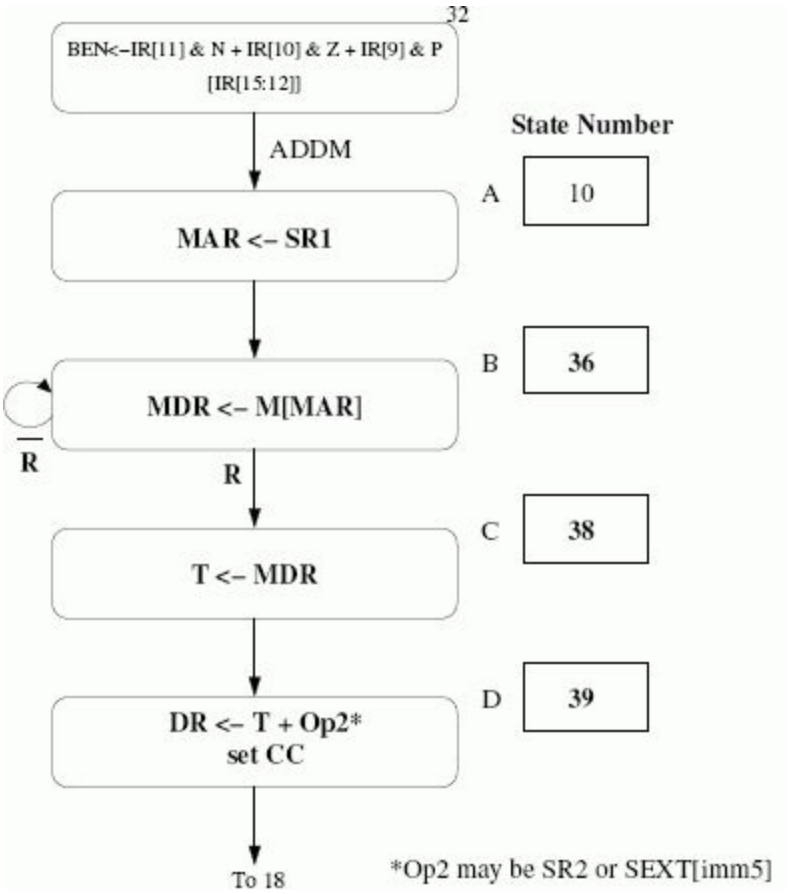
necessary changes to the datapath



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.*



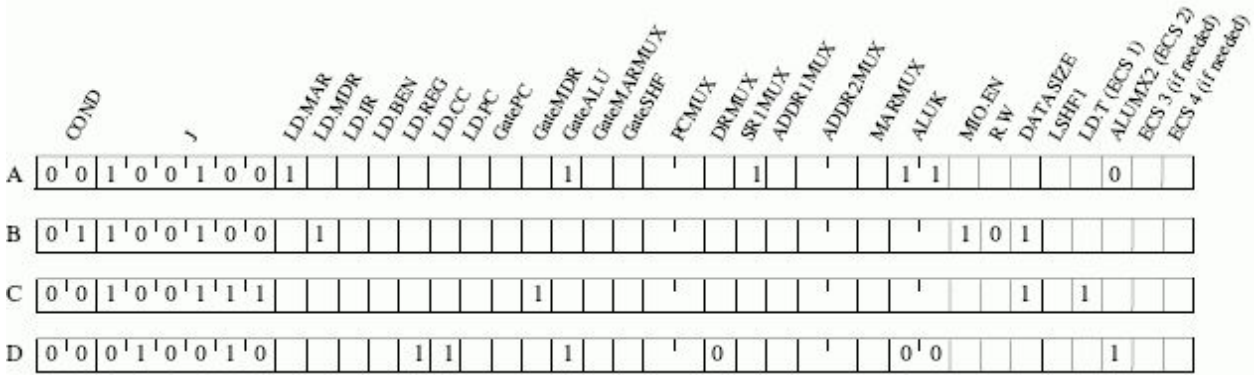
1. Filled in state sequence:



There are many possible state numberings, but state numbers must be chosen from 24, 34, and 36-63. The state number for C must differ from the state number for B only in the bit1 position, and bit1 must be 0 for state B. For example, if state B is 36 (100100), state C must be 38 (100110). The one-bit signal X is the Ready bit (R) from memory.

2. We need a 16-bit temporary register (T) which gets its inputs from the system bus. We need a signal LD.T (extra control signal 1) to control when to load this register. This

register holds the data that is fetched from memory. We also need a mux in front of the A input of the ALU. This mux should select between SR1 and the output of the temporary register. We need a control signal for the select line of this mux (ALUMX2 - extra control signal 2).
    a. ALUMX2 = 0 selects SR1
    b. ALUMX2 = 1 selects T
3. Filled in microinstructions:



All other signals are 0. The J bits will depend on the state numbering chosen in part (a). The J bits for states A and B must correspond to the state number for B, the J bits for state C must correspond to the state number for D, and the J bits for D must be 18 (010010). The bit encodings for control signals are the same as specified in Lab 3.

/ 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.

Truth table for the Address Control Logic:

MIO.EN	R.W	MAR	MEM.EN	INMUX	LD.KBSR	LD.DSR	LD.DDR
0	X	X	0	X	0	0	0
1	R	xFE00	0	KBSR	0	0	0
1	R	xFE02	0	KBDR	0	0	0
1	R	xFE04	0	DSR	0	0	0
1	R	xFE06	0	X	0	0	0
1	R	OTHER	1	MEMORY	0	0	0
1	W	xFE00	0	X	1	0	0
1	W	xFE02	0	X	0	0	0
1	W	xFE04	0	X	0	1	0
1	W	xFE06	0	X	0	0	1
1	W	OTHER	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, instructions JSR R7 and JSRR R7 would execute incorrectly. Their base register (R7) would be overwritten with the value of the PC in state 4. Thus, the addresses generated in states 20 and 21 would come out wrong.

/ 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. We cannot tell the size of the MAR, since it depends on the number of memory locations and does not depend on the addressability (the number of bits in each location). For the MDR, first consider the LC3b. The LC3b is Byte (8 bit) addressable with an MDR size of 16 bits. We cannot tell the size of the MDR based on addressability either.
2. Each register (DR, SR1, and SR2) would have to be specified with five bits. With the steering bit, the total number of bits used would be 16 – leaving no bits for the opcode.

Problem 7

Given the following code:

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

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)
- 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.

Problem 8

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

Cycle 0		
	Valid	Value

End of Cycle 8		
	Valid	Value

R0	1	0
R1	1	5
R2	1	7

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.
- 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

ADD R1 , R0 , R2

ADD R2 , R0 , R0

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

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

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
4. ? What is the overall branch prediction accuracy? What is the prediction accuracy for each branch?

Assumptions:

- There are enough ports to the register file.
- There are enough ports to the memory.
- There are separate execution units for ADD, AND, STW, and BR instructions (They can all be in the execute stage at the same time.)

1.

```
        1   2   3   4   5   6   7   8   9  10  11  12  13  14
------------------------------------------------------------------- Start of first iteration (R1 is even)
STW     F | D | E | E | E | S
ADD         F | D | E | E | E | S
AND             F | D | E | E | S
BRz                 F | D | D | E | S
ADD                             F | D | E | E | E | S
ADD                                 F | D | E | E | E | S
BRp                                     F | D | D | D | E | S
------------------------------------------------------------------- End of the first iteration (R1 is odd now)
STW                                                         F
```

The loop takes the same number of cycles to execute for even and odd values of R1. Each iteration takes 14 cycles in the steady state. There are 5 iterations for even values of R1 and 4 iterations for odd values of R1. The total number of cycles is:

(14 * 5) + (14 * 4) + 1 = 127

The extra 1 cycle comes from the last iteration (Store result stage of the BRp instruction).

2.

```
        1   2   3   4   5   6   7   8   9  10  11  12  13
------------------------------------------------------------------- Start of first iteration (R1 is even)
STW     F | D | E | E | E | S
ADD         F | D | E | E | E | S
AND             F | D | E | E | S
BRz                 F | D | D | E | S
ADD                     F | F | D | E | E | E | S
ADD                             F | D | E | E | E | S
BRp                                 F | D | D | D | E | S
------------------------------------------------------------------- End of the first iteration (R1 is odd now)
STW                                 F | F | F | D | E | E | E | S |
```

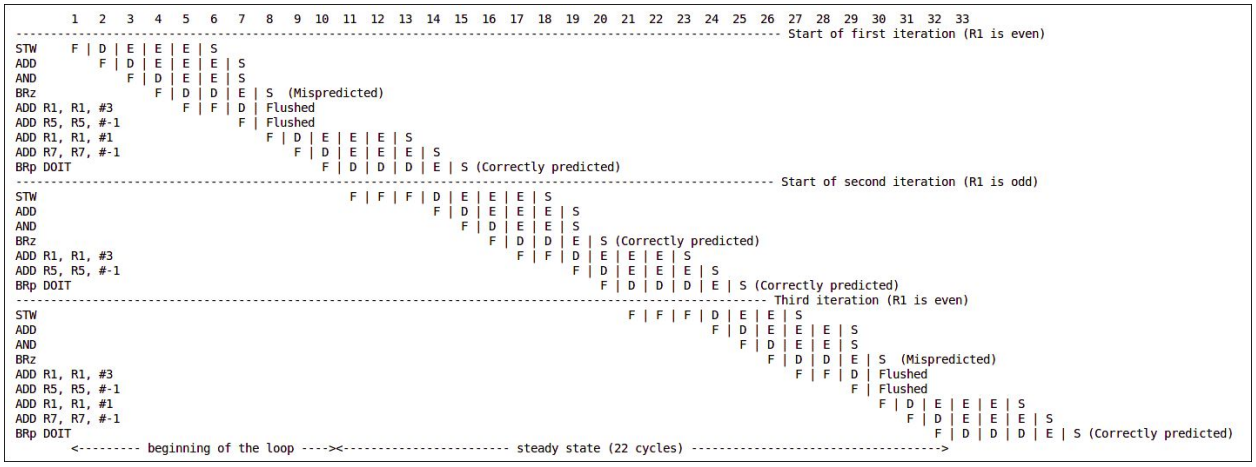
The loop takes the same number of cycles to execute for even and odd values of R1. Each iteration takes 13 cycles but 3 cycles can be overlapped with the next iteration. The total number of cycles is:

(10*9) + 3 = 93

3.

BRz instruction will always be predicted not taken. It is taken when R1 is even. So it will be mispredicted when R1 is even and correctly predicted when R1 is odd. The following diagram shows three consecutive iterations of the loop. In the first iteration, BRz is mispredicted, in the second iteration it is correctly predicted.

The first BRp instruction is always predicted taken. It is always predicted correctly. The second BRp instruction is also always predicted taken. It is mispredicted only once in the last iteration of the loop.



Loop steady state is shown above. It takes 22 cycles and it is repeated 4 times. The beginning of the loop (until the steady state) takes 10 cycles as shown above. The end of the loop (part of the last iteration which is not in steady state) takes 5 more cycles to execute. The total number of cycles is:

10 + (4 * 22) + 5 = 103

Prediction accuracies for each branch are:

- BRz = 4/9 (Correctly predicted when R1 is odd, R1 is odd for 4 iterations of the loop)
- first BRp = 4/4
- second BRp = 4/5 (Mispredicted only in the last iteration - Note that this misprediction does not affect the number of cycles it takes to execute the loop) v

Combined branch prediction accuracy = 12/18 = 67%

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
```

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?

The comments indicate whether the branch is predicted taken or not:

```
      AND R0, R0, #0
      BRz A           ; predicted taken, actually taken
      ADD R1, R0, #-1
C     BRp B           ; predicted not taken, actually taken
A     ADD R2, R0, #1
      BRp C           ; predicted not taken, actually taken
B     RET
```

Branch History Register

111

Branch History Table

0	10
1	00
2	10
3	**10**

4	10
5	**11**
6	00
7	**10**

Problem 11

Consider the following piece of code:

```
for(i = 0; i < 100; i++)
  A[i] = ((B[i] * C[i]) + D[i]) / 2;
```

1. Translate this code into assembly language using the following instructions in the ISA (note the number of cycles each instruction takes is shown with each instruction):

Opcode	Operands	Number of Cycles	Description
LEA	Ri, X	1	Ri ← address of X
LD	Ri, Rj, Rk	11	Ri ← MEM[Rj + Rk]
ST	Ri, Rj, Rk	11	MEM[Rj + Rk] ← Ri
MOVI	Ri, Imm	1	Ri ← Imm
MUL	Ri, Rj, Rk	6	Ri ← Rj × Rk
ADD	Ri, Rj, Rk	4	Ri ← Rj + Rk
ADD	Ri, Rj, Imm	4	Ri ← Rj + Imm
RSHFA	Ri, Rj, amount	1	Ri ← RSHFA (Rj, amount)
BRcc	X	1	Branch to X based on condition codes

   Assume it takes one memory location to store each element of the array.
   Also assume that there are 8 registers (R0-R7).
   How many cycles does it take to execute the program?

2. Now write Cray-like vector/assembly code to perform this operation in the shortest time possible. Assume that there are 8 vector registers and the length of each vector register is 64. Use the following instructions in the vector ISA:

Opcode	Operands	Number of Cycles	Description
LD	Vst, #n	1	Vst ← n
LD	Vln, #n	1	Vln ← n
VLD	Vi, X + offset	11, pipelined	
VST	Vi, X + offset	11, pipelined	
Vmul	Vi, Vj, Vk	6, pipelined	
Vadd	Vi, Vj, Vk	4, pipelined	
Vrshfa	Vi, Vj, amount	1	

How many cycles does it take to execute the program on the following processors? Assume that memory is 16-way interleaved.

i. Vector processor without chaining, 1 port to memory (1 load or store per cycle)
ii. Vector processor with chaining, 1 port to memory
iii. Vector processor with chaining, 2 read ports and 1 write port to memory

Problem 12

Consider the following example used to explain Tomasulo's Algorithm:

```
Format: Opcode Destination Source1 Source2
MUL R3,  R1, R2
ADD R5,  R3, R4
ADD R7,  R2, R6
ADD R10, R8, R9
MUL R11, R7, R10
ADD R5,  R5, R11
MUL R10, R4, R10
```

Construct the Data Flow Graph for this program.

Problem 13

A five instruction sequence executes according to Tomasulo's algorithm. Each instruction is of the form ADD DR,SR1,SR2 or MUL DR,SR1,SR2. ADDs are pipelined and take 9 cycles (F-D-E1-E2-E3-E4-E5-E6-WB). MULs are also pipelined and take 11 cycles (two extra execute stages). The microengine must wait until a result is in a register before it sources it (reads it as a source operand)

The register file before and after the sequence are shown below (tags for “After” are ignored).

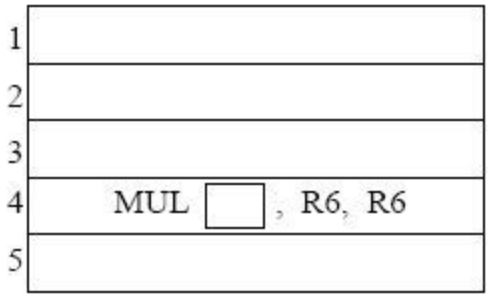
Before

	V	tag	value
R0	1	z	4
R1	1	z	5
R2	1	z	6
R3	1	z	7
R4	1	z	8
R5	1	z	9
R6	1	z	10
R7	1	z	11

After

	V	tag	value
R0	1		310
R1	1		5
R2	1		410
R3	1		31
R4	1		8
R5	1		9
R6	1		10
R7	1		21

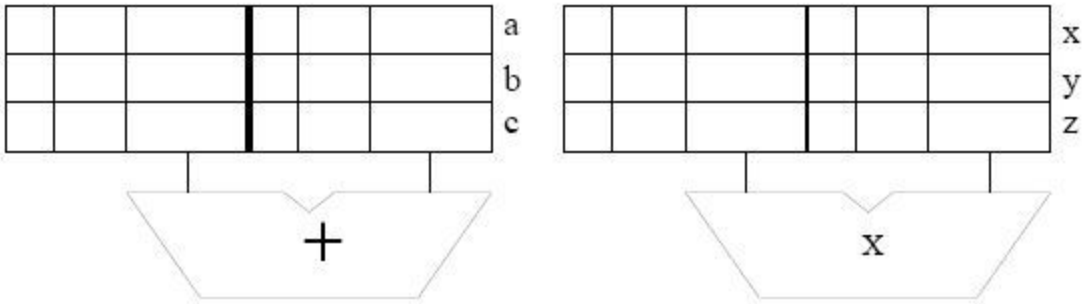
1. Complete the five instruction sequence in program order in the space below. Note that we have helped you by giving you the opcode and two source operand addresses for instruction 4. (The program sequence is unique.)



2. In cycle 1 instruction 1 is fetched. In cycle 2, instruction 1 is decoded and instruction 2 is fetched. In cycle 3, instruction 1 starts execution, instruction 2 is decoded, and instruction 3 is fetched.

   Assume the reservation stations are all initially empty. Put each instruction into the next available reservation station. For example, the first ADD goes into “a”. The first MUL goes into “x”. Instructions remain in the reservation stations until they are completed. Show the state of the reservation stations at the end of cycle 8.

   Note: to make it easier for the grader, when allocating source registers to reservation stations, please always have the higher numbered register be assigned to SR2.



3. Show the state of the Register Alias Table (V, tag, Value) at the end of cycle 8.

	V	tag	value
R0			
R1			
R2			
R3			
R4			
R5			
R7			