Department of Electrical and Computer Engineering
The University of Texas at Austin

EE 360N, Spring 2007
Yale Patt, Instructor
Chang Joo Lee, Rustam Miftakhutdinov, Poorna Samanta, TAs
Exam 2, April 18, 2007

Name:____

Problem 1 (25 points):_______

Problem 2 (20 points):_______

Problem 3 (15 points):_______

Problem 4 (20 points):_______

Problem 5 (20 points):_______

Total (100 points):______________

Note: Please be sure that your answers to all questions (and all supporting work that is required) are contained in the space provided.

Note: Please be sure your name is recorded on each sheet of the exam.

GOOD LUCK!

Name:____________________________________

Problem 1 (25 points)

Part a (4 points): Today's microprocessors run at more than 1 GHz. Some of them more than 3 GHz. This causes errors due to alpha particles. That is errors that are not due to the logic, and if the instruction is executed a second time, it will probably (with very high probability) execute correctly. We call them soft errors. Suppose we do not have any way to test for soft errors, and we just continue to execute as if the error had not occurred. For each of the items below, put a check mark if a soft error in that item could cause a program to execute incorrectly.

Branch History Register of the 2-level predictor:	______
LRU bit in a 2-way set associative cache:	______
Protection field in the PTE of a virtual page:	______
Z condition code:	______

Part b (5 points): A 10 by 4 matrix is stored in **column major** order, starting at location 2000. We wish to load the third row of the matrix into V0. To execute the vector load correctly, what values must be present in:

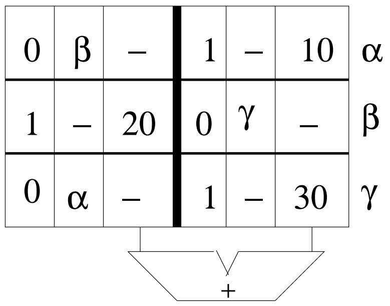
Vaddr:	
VLen:	
VStride:	

Name:____________________________________

Part c (4 points): A PAg 2-level branch predictor is implemented, which keeps 12 bits of history for each of the 1000 branches in the program. How many saturating 2-bit counters does it take to implement this predictor?

Answer:

Part d (5 points): The IBM 360/91, you recall is the computer for which Tomasulo's algorithm was designed. While the computer was executing, someone hit the halt button and found that the contents of the three reservation stations in front of the adder are as shown below.



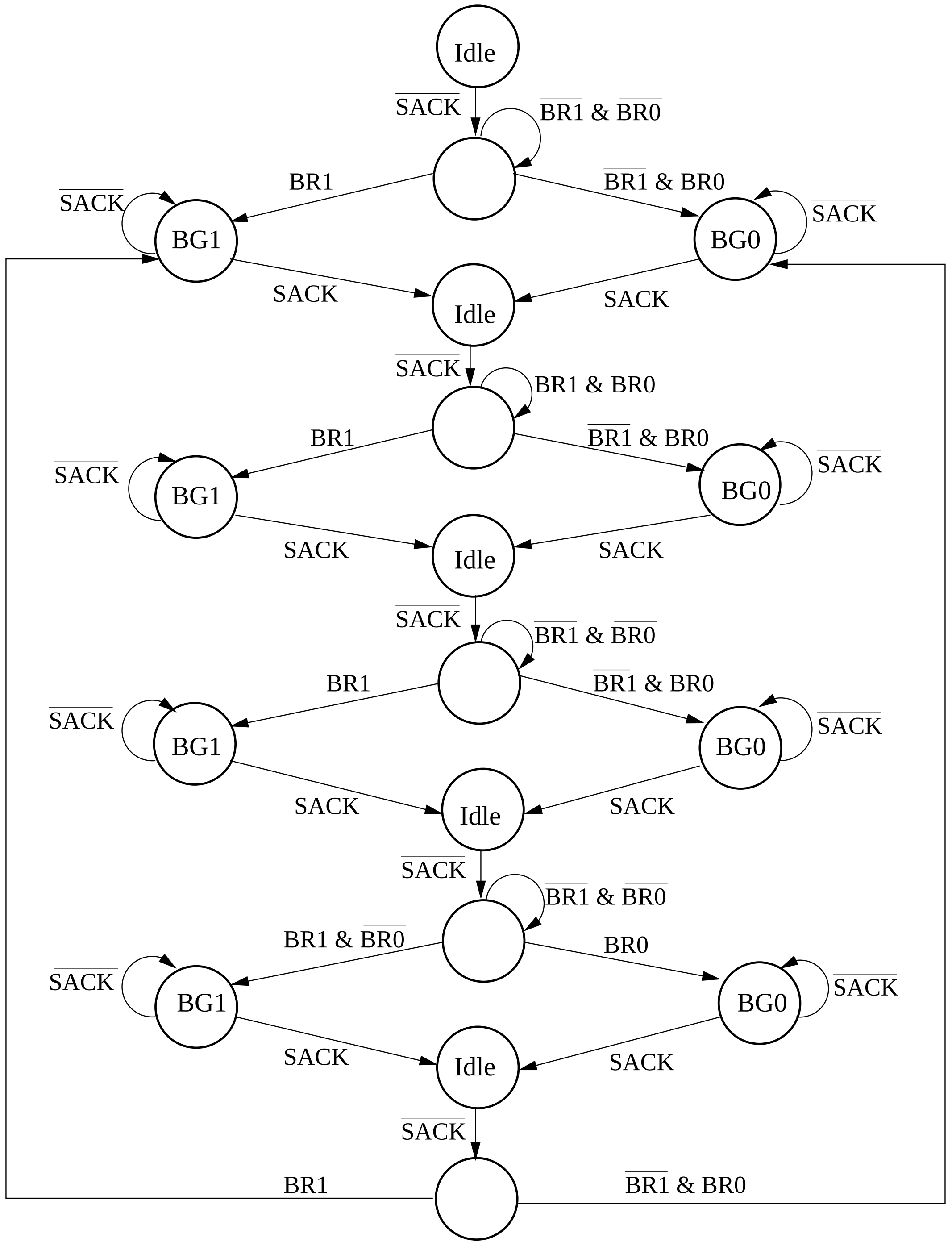
0	β	–	1	–	10	α
1	–	20	0	γ	–	β
0	α	–	1	–	30	γ

Assuming the adder is fully pipelined and takes four cycles to execute an ADD instruction, how many cycles will it take to complete the three ADD instructions after somebody hits the run button? Explain.

Answer:

Name:___________________________________

Part e (7 points): The Central Arbitration Unit (CAU) of a bus based system containing two potential bus masters has the following state machine:



What is the plus or minus of this CAU over the priority encoder schemes we have discussed in class?

Answer:

Name:__________________________

Problem 3 (20 points):

An array A consists of N one-byte elements, stored in N consecutive locations of memory, starting at address x4000.

A programmer wishes to add the integer P to each even numbered element of A, but does so by the very curious method of adding ”1” to each even numbered element, and then iterating that process P times. The C code segment looks like this:

```
for ( i = 0; i < P; i++ )
{
     for ( index = 0; index < N; index = index + 2 )
            A[index] = A[index] + 1;
}
```

Assume the compiler allocates registers for the loop variables i and index, and that the values P and N are also in registers before the code segment starts executing.

Your job is to compute the cache miss ratio during execution of this program segment if the processor has a 1 KB, 2-way set associative, physically addressed cache. Line size is 8 bytes. Replacement is LRU.
Show all work below. You may leave the answers as fractions.

Part a (5 points): N=512, P=10

Part b (5 points): N=512, P=50

Part c (5 points): N=2048, P=10

Part d (5 points): N=2048, P=50

Part a: Part b: Part c: Part d:

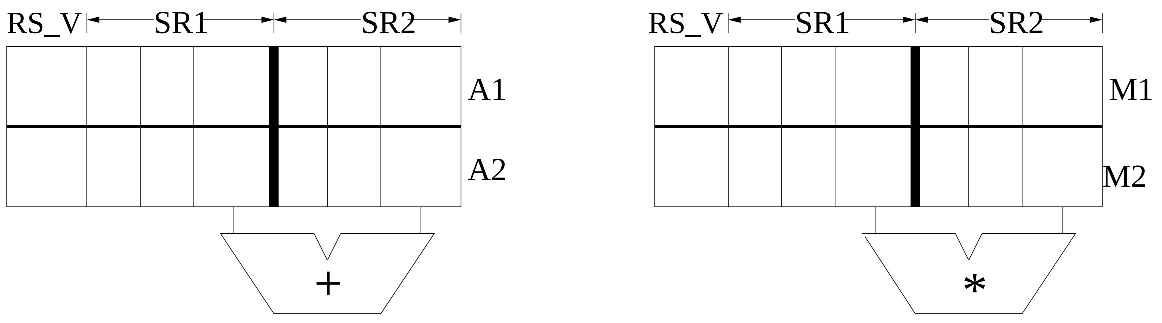
Name:___________________________________

Problem 3 (15 points)

The following sequence of instructions is to be executed on a machine similar to the IBM 360/91, i.e., out-of-order execution using the Tomasulo algorithm.

```
ADD R2,R2,R3
MUL R3,R1,R2
ADD R2,R1,R2
ADD R2,R2,R5
MUL R1,R4,R5
MUL R3,R5,R6
ADD R3,R1,R2
```

However, in this machine, there are only two reservation stations each in front of the adder and the multiplier, as shown below:



- Note the extra valid bit RS_V associated with each reservation station entry. It indicates whether a reservation station entry is "live." That is, when an instruction is written to a reservation station at the end of the decode stage, that bit is set to 1. When an instruction completes execution (in WB stage), that bit is set to 0. That is, the instruction remains live in the reservation station until it completes execution and writes back its result.
- ADD takes 4 cycles (F,D,E,WB). MUL takes 7 cycles (F,D,E1,E2,E3,E4,WB) and is fully pipelined.
- In Decode stage, instructions are decoded, renamed and stored in any available reservation station (one whose RS_V bit is set to 0).
- A source operand is ready for execution in the cycle immediately following the WB stage of the instruction that produced it.

Part a (9 points): Assume all reservation stations are initially empty (i.e., their RS_V bits are all 0s). The first instruction (ADD R2,R2,R3) is fetched in cycle 1. Show the state of the reservation stations in the above drawing at the end of cycle 8. The initial contents of the Register Alias Table are shown on the next page.

Name:____________________________________

Problem 3 continued:

Part b (6 points): Show the contents of the Register Alias Table at the end of cycle 8

Initial Contents of RAT

	V	TAG	VALUE
R1	1	——	1
R2	1	——	2
R3	1	——	3
R4	1	——	4
R5	1	——	5
R6	1	——	6

Contents of RAT at End of Cycle 8

	V	TAG	VALUE
R1			
R2			
R3			
R4			
R5			
R6			

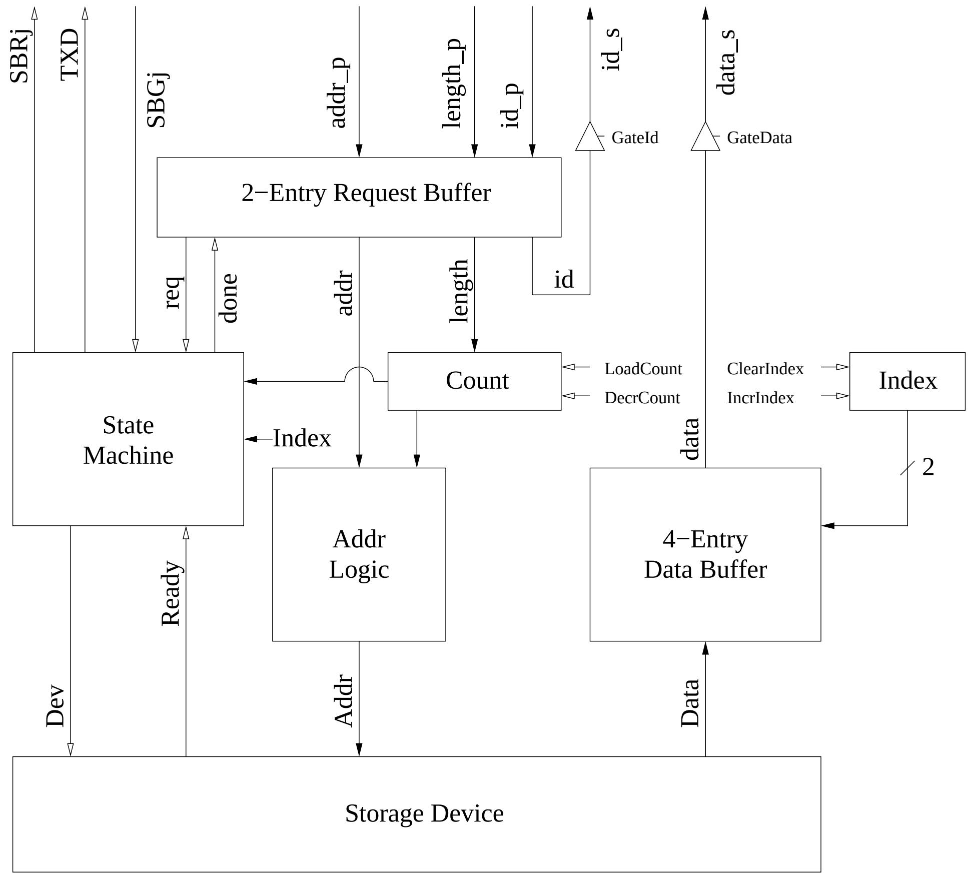
Name:________________________________

Problem 4 (20 points)

Recall the synchronous bus system from problem 1 of Problem Set 5 (reproduced on pages 15 and 16). To improve bus efficiency, we decided to allow the storage controllers to transfer data in bursts of up to four data elements. To this end, we added a new bus signal *TXD* (transaction done) and changed the bus protocol as follows:

After the storage controller is granted the data bus, it transfers a burst of up to four data elements in up to four bus cycles (one data element per cycle). The storage controller sets the *TXD* signal in the last cycle of the burst to let the storage arbitration unit know that the bus transaction is done.

To implement the new protocol, we added extra address logic and a 4-entry data buffer (indexed by a 2-bit register *Index*) to the storage controller, as shown in the figure:

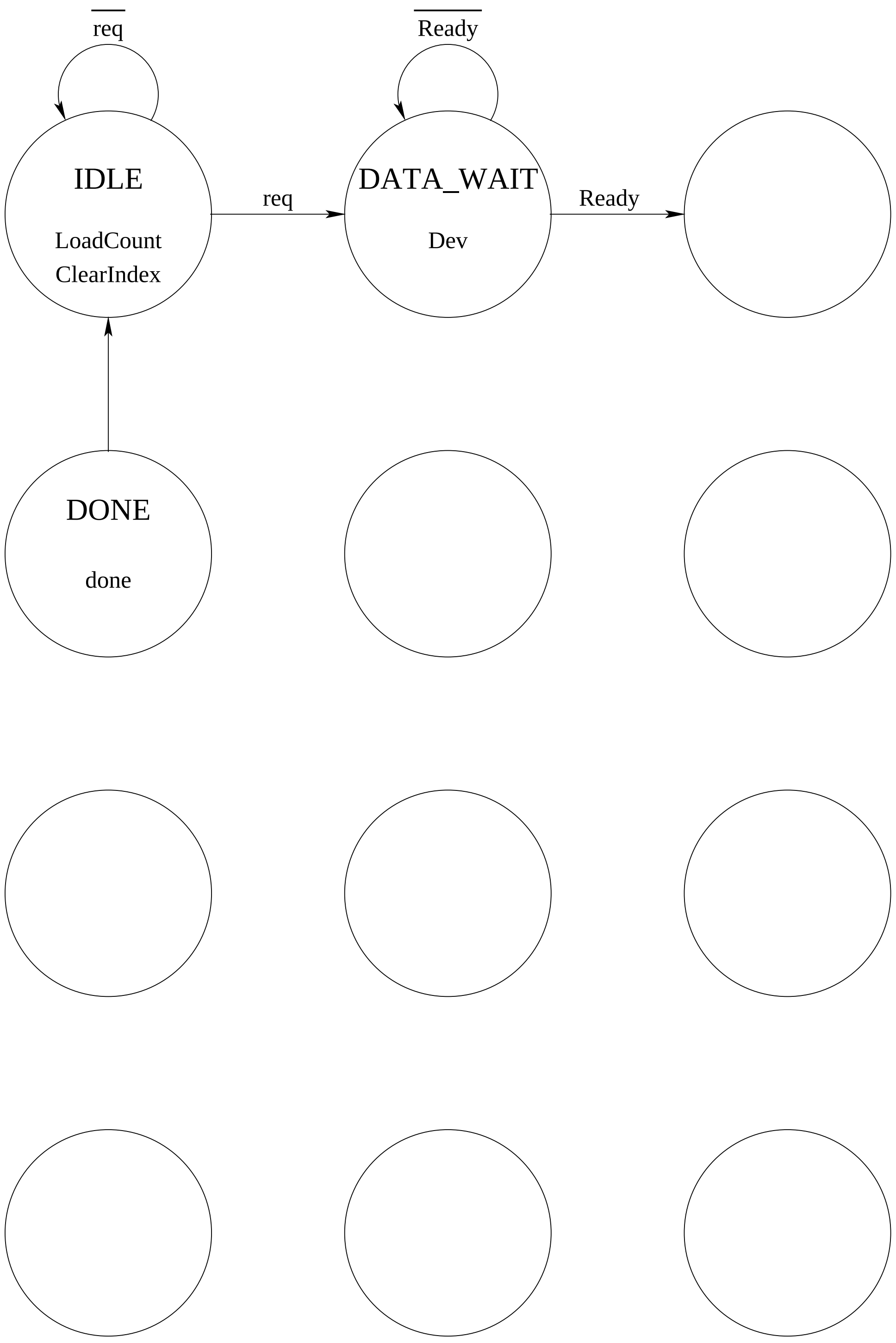


When a request comes from the 2-entry request buffer (*req* = 1), the storage controller asks the storage device (by setting the *Dev* signal) to fill the data buffer with four data elements (less if *Count* < 4). The address logic (independent of the state machine) controls which and how many elements need to be brought in. The state machine waits in state DATA_WAIT until the storage device loads the data into the data buffer and asserts *Ready*. The controller is now ready to request the data bus. Once the bus is granted, the controller puts consecutive elements from the buffer (starting with element 0) onto the bus by incrementing the *Index* each cycle.

Complete the state diagram for the new storage controller on the next page (Note: we have provided more than enough states for your use. Use as many as you need).

Name:__________________________________

Problem 4 continued



Name:________________________________

Problem 5 (20 points):

We wish to use the unused opcode 1010 to implement a new instruction MULB, which performs an unsigned 8 bit multiply on the low 8 bits in two source operands and writes back the 16 bit result to a destination register. The multiplicand value is always in a register and the multiplier value can be either the contents of a register or an immediate value. The specification of this instruction is as follows:

Assembler Formats

MULB DR, SR1, SR2
MULB DR, SR1, imm4

Encodings

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

	15 12	11 9	8 6	5		3 2 0
MULB	**1010**	**DR**	**SR1**	**1**	**0**	**imm4**

Operation

```
if ( bit[5] == 0 )
     DR = SR1 * SR2;
else
     DR = SR1 * ZEXT(imm4);
setcc();
```

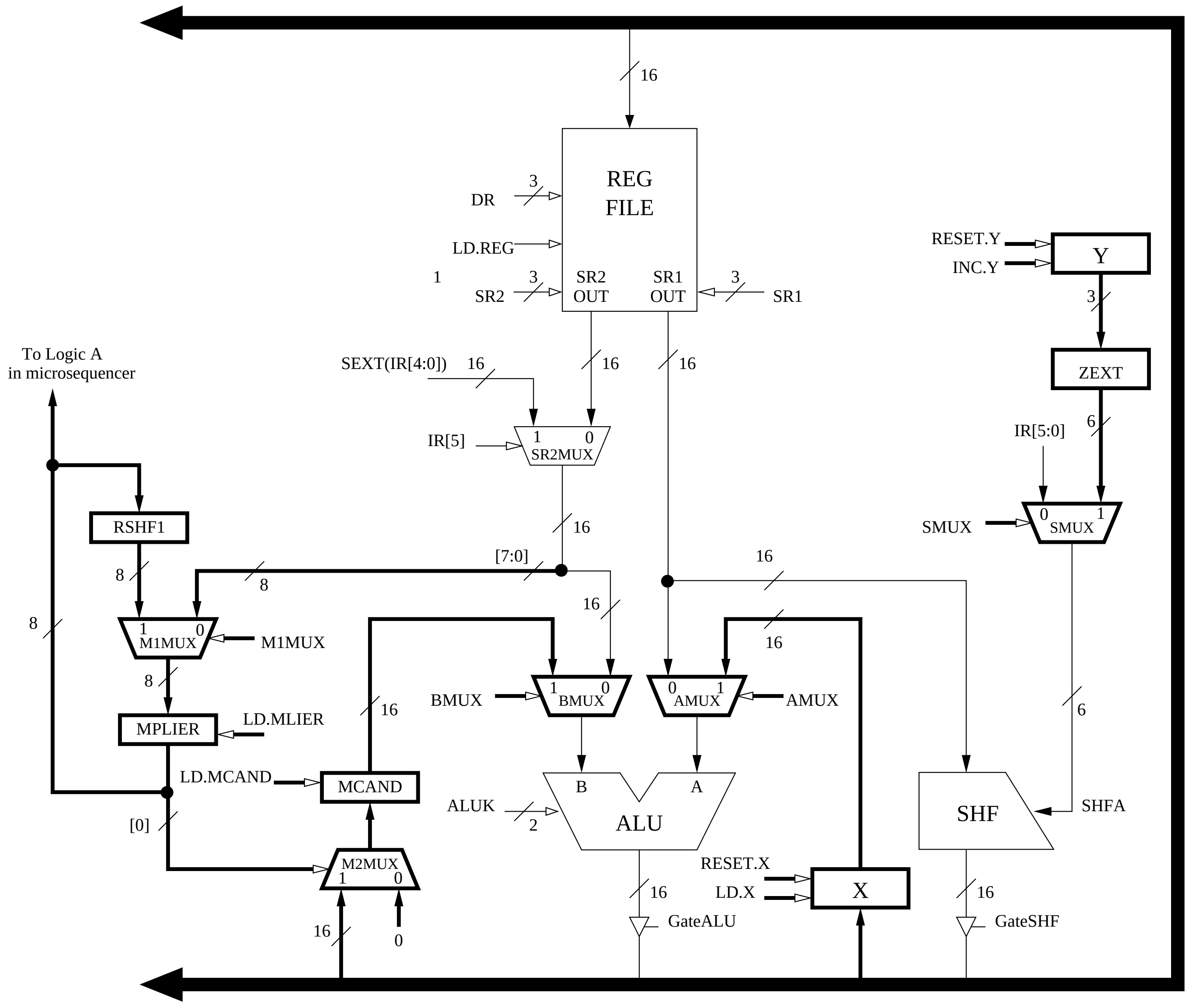
Note that it is the programmer's job to make sure SR1[15:8] = SR2[15:8] = 0.

To implement MULB, we have added the following structures to the LC-3b datapath as shown on the next page.

1. An 8-bit register MPLIER (Multiplier)
2. A 16-bit register MCAND (Multiplicand)
3. A logical 1-bit right shifter
4. A 16-bit register X (Note: RESET.X clears X)
5. A 3-bit register Y (Note: RESET.Y clears Y)
6. Five 2-input MUXes as needed

Name:___________________________________

Problem 5 continued:



Some useful control signals:

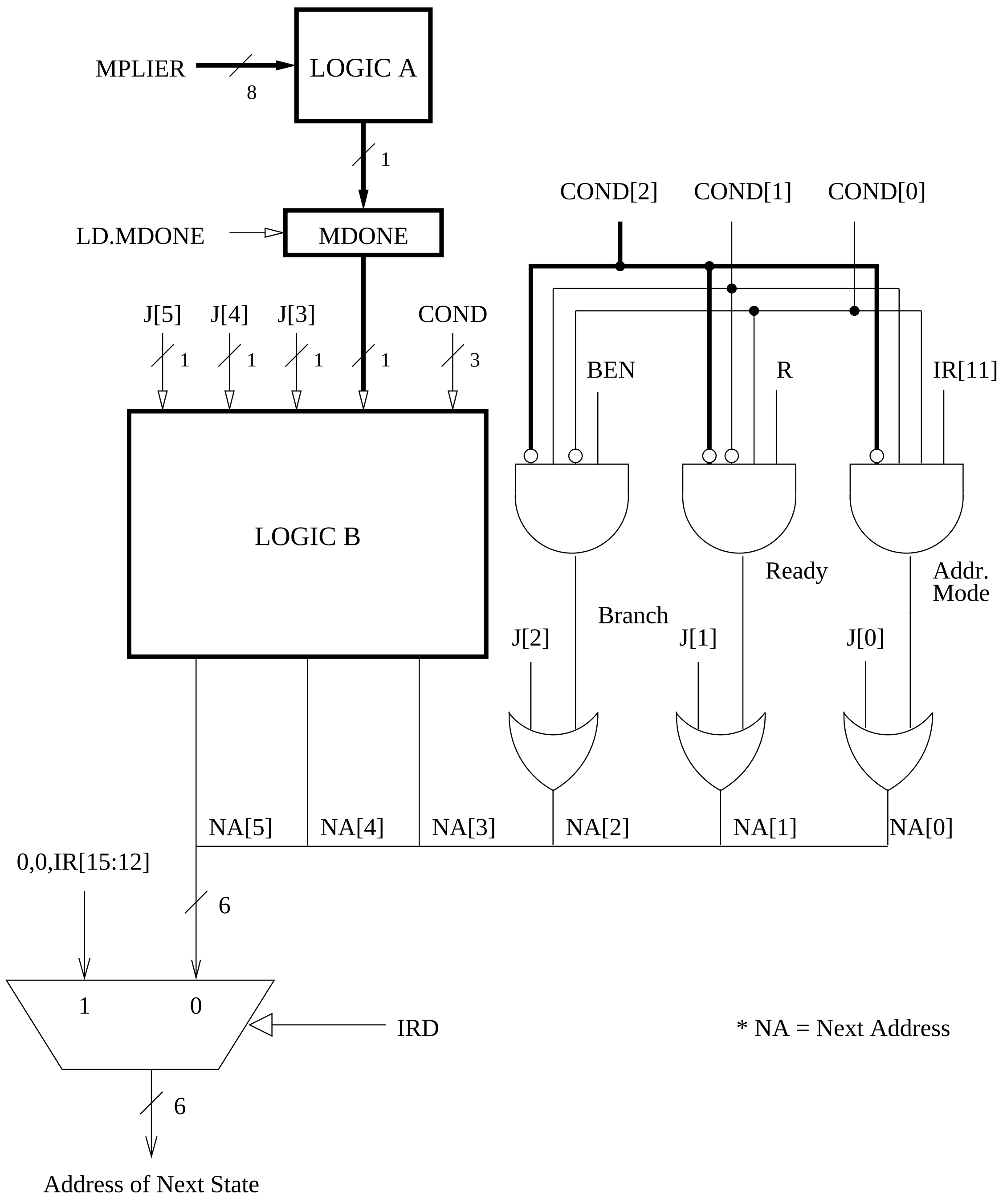
SHFA[5:4]:	00 = Left shift
	01 = Logical right shift
	10 = Undefined
	11 = Arithmetic right shift
SHFA[3:0]:	Shift amount
ALUK[1:0]:	00 = ADD
	01 = AND
	10 = XOR
	11 = PASS A

Name:____________________________________

Problem 5 continued:

We have also added the following to the microsequencer, as shown below.

1. A 1-bit register MDONE (Multiply done) and combinational logic A and B
2. A 3-bit COND field, instead of the 2-bit COND field



Name:__________________________________

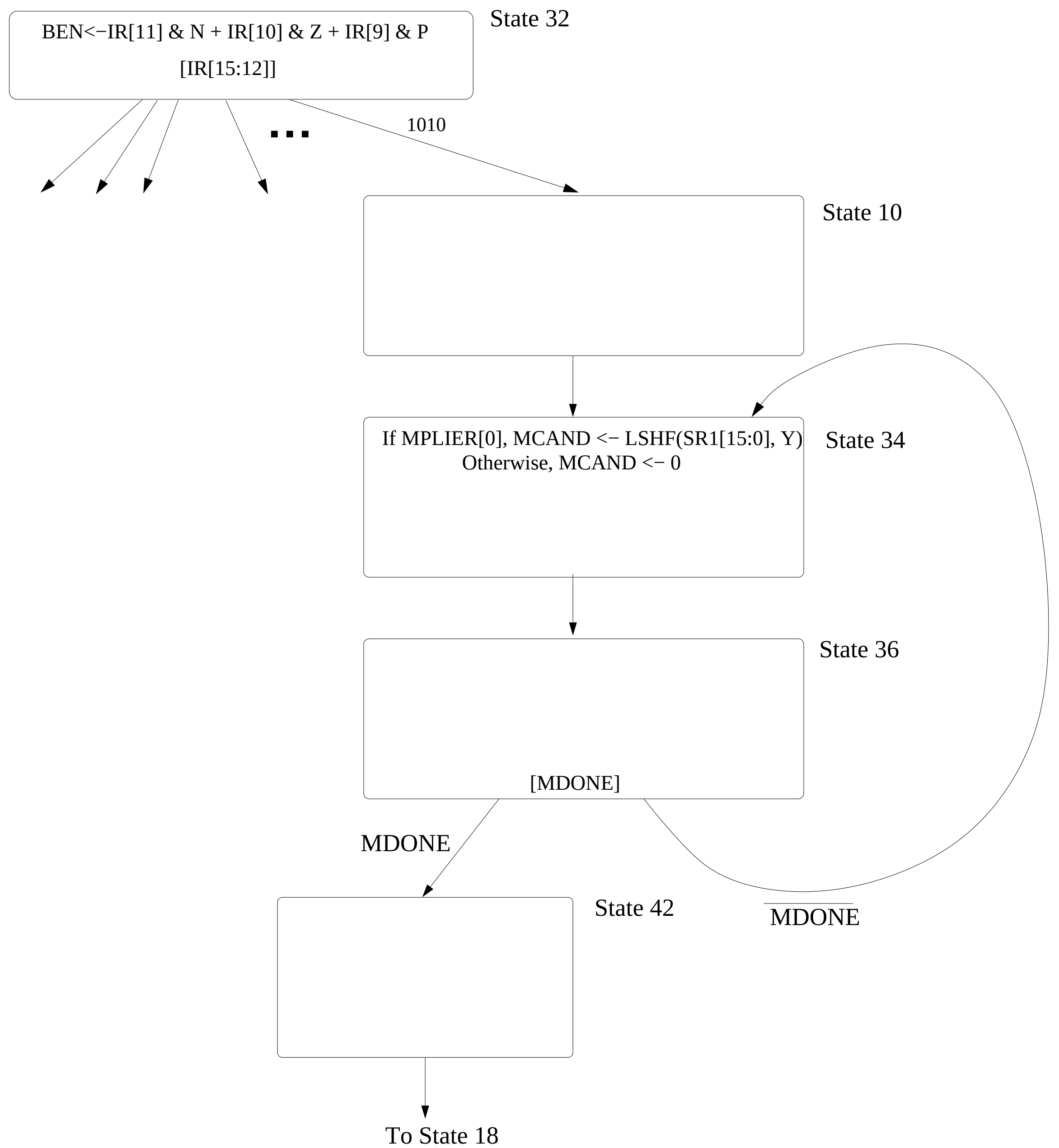
Problem 5 continued:

Part a (4 points): Describe what the X and Y registers do:

X:

Y:

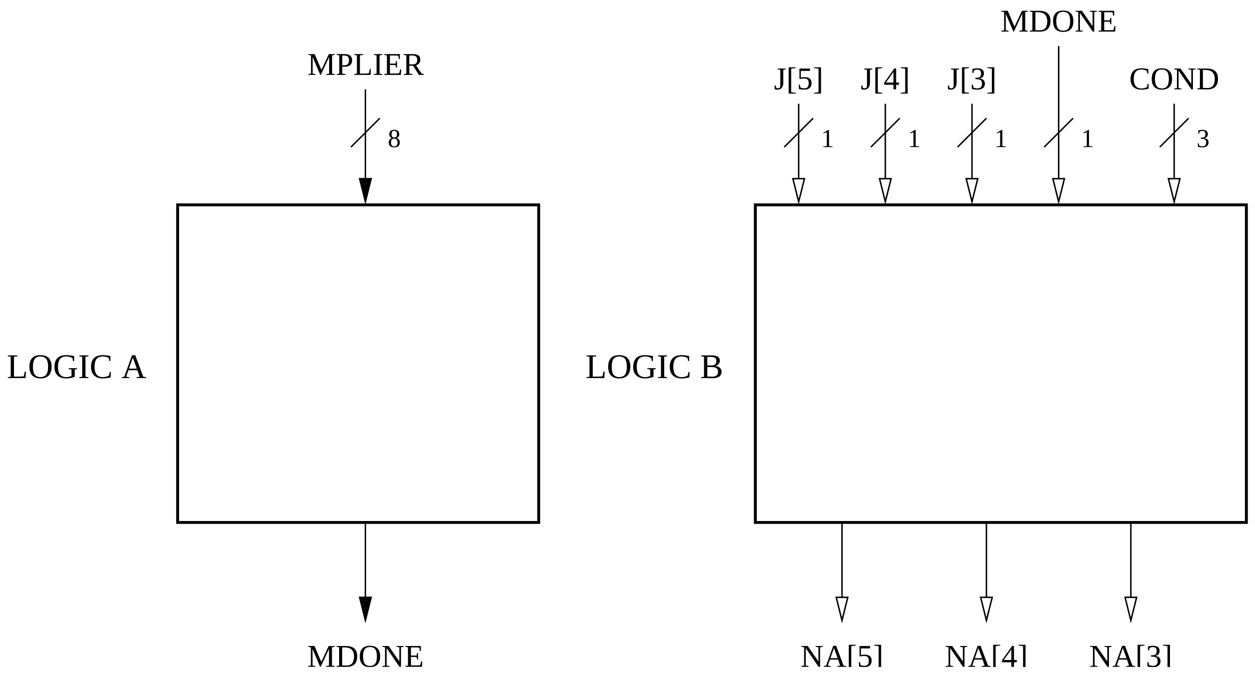
Part b (6 points): We show the beginning of the state diagram necessary to implement MULB. Using the notation of the LC-3b State Diagram, add the bubbles you need to implement the MULB instruction. Your job is to **describe** inside each bubble what happens in each state.


BEN<−IR[11] & N + IR[10] & Z + IR[9] & P
[IR[15:12]]
State 32
1010
State 10
If MPLIER[0], MCAND <− LSHF(SR1[15:0], Y)
Otherwise, MCAND <− 0
State 34
State 36
[MDONE]
MDONE
$\overline{\text{MDONE}}$
State 42
To State 18

Name:____________________________________

Problem 5 continued:

Part c (5 points): Show the logic circuits required to implement combinational logic A and logic B.
NOTE: If you need to assign any new encoding to the COND[2:0], use **COND[2:0] = 100**.



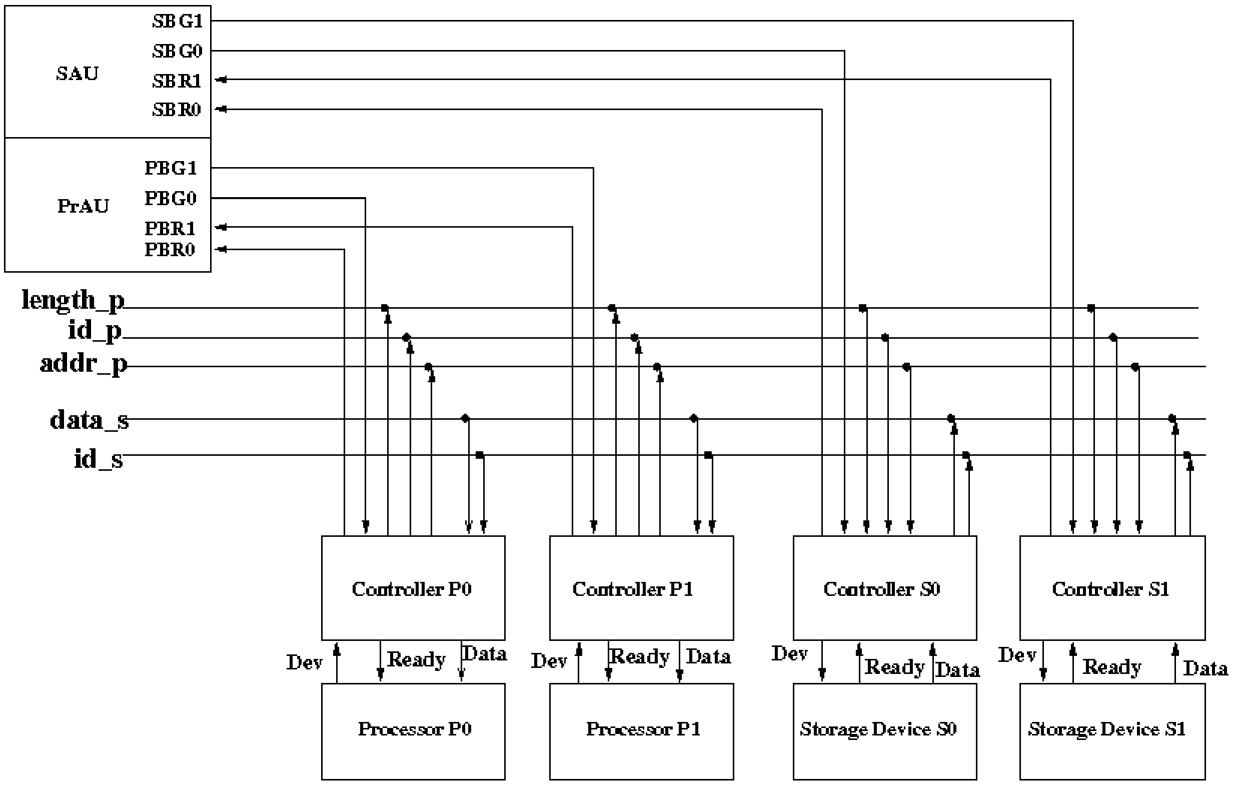
Part d (5 points): The processing in each state we added is controlled by asserting or negating each control signal. Enter a 1 or a 0 as appropriate for the microinstructions corresponding to the states we have added.

NOTE: Please use the encodings specified on the datapath and microsequencer figures for all the signals.

State	*IRD*	*COND[2:0]*			*J[5:0]*						*LD.REG*	*LD.CC*	*LD.MPLIER*	*LD.MCAND*	*LD.X*	*LD.MDONE*	*INC.Y*	*RESET.X*	*RESET.Y*	*M1MUX*	*AMUX*	*BMUX*	*SMUX*	*ALUK[1:0]*		*GateALU*	*GateSHF*
10																											
34																											
36																											
42																											

Problem Set 5 problem 1

We show below a synchronous split transaction bus system employing centralized arbitration. There are two processors (P0 and P1) and two storage devices (S0 and S1). For the sake of simplicity, we assume that processors will only **read** data from the the storage devices.



Arbitration

There are two arbitration units, Processor Arbitration Unit(PrAU) and Storage Device Arbitration Unit(SAU). PrAU arbitrates the *addr_p* bus across P0 and P1. SAU arbitrates the data bus across S0 and S1. This allows one processor to request a transfer while the other processor is receiving data. P0 has higher priority than P1 and S0 has higher priority than S1.

Data Transfer

The following is how a transfer between Processor *i* (P*i*) and Storage Device *j* (S*j*) proceeds:

1. When P*i* needs data from S*j*, it asserts the Bus Request signal (PBR*i*). Upon receiving the Bus Grant signal (PBG*i*) from the PrAU, the controller asserts the starting address on *addr_p*, the *Vaddr* (valid address signal, not shown in the figure), number of elements to be transfered on signal *length_p* (*Count*), and its Id (*i* in this case) on *id_p* signal. P*i* now waits for data from S*j*.
2. When S*j* sees an address that lies in its address range with *Vaddr* asserted, it latches the address, length, and Id and starts processing the request.

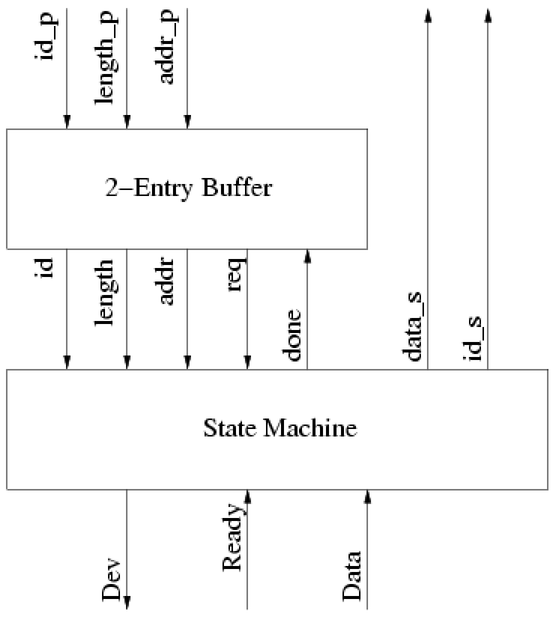
3. When S*j* is ready to return a data element, it asserts SBR*j* and upon receiving the Grant signal (SBG*j*), it asserts data on *data_s*, the Id of the processor that requested the data on *id_s* and the *Vid* (valid id signal, where *Vid* = SBG0 OR SBG1, not shown in the figure). It also decrements its *Count* of the remaining elements. The SBG*j* signal grants the bus to S*j* for a *single* cycle.
4. When P*i* sees its Id on *id_s* with *Vid* signal asserted, it latches the data element and decrements its *Count*.
5. P*i* and S*j* cycle through steps 3 and 4 until *Count* reaches 0.

The device and controller synchronize using the *Dev* and *Ready* signals. When a transfer needs to be carried out the *Dev* signal is asserted. The availability of valid data is indicated by asserting the *Ready* signal.

Note: for the sake of simplicity, assume each processor can have only one pending transfer. Each storage controller can buffer up to two requests since each processor could have one pending transfer from a given storage.

1. Construct the state machine for the processor controller P*i*. Show relevant inputs and outputs on all arcs. (Remember that the controller keeps track of the number of data transfers completed in a *Count* register.)

2. Assume that the controller consists of a state machine and a two-entry buffer:



Whenever the storage controller is requested on the bus, the buffer stores the *id_p*, *length_p*, and *addr_p* bus signals. The *req* output of the buffer indicates that at least one request is awaiting service in the buffer. The *id*, *length*, and *addr* signals are the buffered bus signals of the oldest buffered bus request. The *done* input tells the buffer to clear the oldest buffered request.

Draw the state diagram for the state machine in the figure above. Show relevant inputs and outputs on all arcs.