Problem Set 1

Due: September 17th @ 4:59 pm (Before class)

Department of Electrical and Computer Engineering
The University of Texas at Austin
EE 382N.1, Fall 2018
Instructor: Dam Sunwoo
TA: Pritesh Chhajed

Instructions

You are encouraged to work on the problem set in small groups (3 or 4 per group) and turn in one problem set for the entire group on Gradescope. Create a copy of this file to fill in your answers within the spaces provided. If you need more space than given box, make a comment there and use extra work space at end of file. There are 2 pages provided for the same.

Before submitting, convert it into a PDF file. Remember to put all your names and eid in the box below. The person submitting must choose everyone in the group.

You will need to refer to the assembly language handout and the LC-3b ISA, microarchitecture, and state diagram documents on the course website.

Student names and EID

Jiaqi Gu jg68999
Wencan Liu wl8784
Junhong Tong jt38826
Yuesen Lu   yl33489

Questions

Problem 1
Briefly explain the difference between the microarchitecture level and the ISA level in the transformation hierarchy. What information does the compiler *need* to know about the microarchitecture of the machine in order to compile the program correctly?

Microarchitecture refers to specific implementation of the ISA.
The ISA serves as the interface or contract between software and hardware.
Compiler need to know about the ISA but does not need to know about the microarchitecture, which is invisible to the compiler.

Problem 2
Classify the following attributes of LC-3b as either a property of its microarchitecture or ISA:

1. There is no subtract instruction in LC-3b.

   ISA

2. The ALU of LC-3b does not have a subtract unit.

   Microarchitecture

3. LC-3b has three condition code bits (n, z, and p).

   ISA

4. The n, z, and p bits are stored in three 1-bit registers.

   Microarchitecture

5. A 5-bit immediate can be specified in an `ADD` instruction

   ISA

6. It takes *n* cycles to execute an `ADD` instruction.

Microarchitecture

7. There are 8 general purpose registers used by operate, data movement and control instructions.

ISA

8. The registers MDR (Memory Data Register) and MAR (Memory Address Register) are used for Loads and Stores.

Microarchitecture

9. A 2-to-1 mux feeds one of the inputs to ALU.

Microarchitecture

10. The register file has one input and two output ports.

Microarchitecture

Problem 3

Both of the following programs cause the value `x0004` to be stored in location `x3000`, but they do so at different times. Explain the difference.

1. First program:

```
    .ORIG x3000
    .FILL x0004
    .END
```

2. Second program:

```
    .ORIG x4000
    AND R0, R0, #0
    ADD R0, R0, #4
    LEA R1, A
    LDW R1, R1, #0
    STW R0, R1, #0
    HALT
A   .FILL x3000
    .END
```

The first program stores x0004 in memory at location of x3000, right after the whole program being loaded into memory starting from x3000;

The second program spends time on AND, ADD, Register write, memory load and memory write, which will take more time than the first program. This program will store x0004 in memory after executing “STW R0, R1, #0”.

Problem 4

At location `x3E00`, we would like to put an instruction that does nothing. Many ISAs actually have an opcode devoted to doing nothing. It is usually called NOP, for NO OPERATION. The instruction is fetched, decoded, and executed. The execution phase is to do **nothing**! Which of the following three instructions could be used for NOP and have the program still work correctly?

1. `0001 001 001 1 00000`
2. `0000 111 000000000`
3. `0000 000 000000000`

For each of the three that cannot be used for NOP, explain why.

1. 0001 001 001 1 00000 : ADD R1 R1 #0
2. 0000 111 000000000 :   BRnzp x0
3. 0000 000 000000000:

The third one can be used as NOP.

The first one will add 0 to R1, which will set condition codes, so it cannot be used as NOP.
The second one will unconditionally load 0 to PC counter which will change the next instruction that is about to be processed and cause severe influence to the program.
The third one will never branch because NZP=000, and it will not change any current state. So the third one can be used as NOP.

Problem 5

A small section of byte-addressable memory is given below:

Address	Data
`x0FFE`	`xA2`

`x0FFF`	`x25`
`x1000`	`x0E`
`x1001`	`x1A`
`x1002`	`x11`
`x1003`	`x0C`
`x1004`	`x0B`
`x1005`	`x0A`

Add the 16-bit two's complement numbers specified by addresses `x1000` and `x1002` if

1. the ISA specifies a little-endian format
2. the ISA specifies a big-endian format

1. MEM[x1000] + MEM[x1002] = x1A0E + x0C11 = x261F
2. MEM[x1000] + MEM[x1002] = x0E1A + x110C = x1F26

Problem 6

Say we have 32 megabytes of storage, calculate the number of bits required to address a location if

1. the ISA is bit-addressable
2. the ISA is byte-addressable
3. the ISA is 128-bit addressable

1. Log2(32 * 2^20 * 8) = 28 bits
2. Log2(32 * 2^20) = 25 bits
3. Log2(32 * 2^20 / 16) = 21 bits

Problem 7

A zero-address machine is a stack-based machine where all operations are done using values stored on the operand stack. For this problem, you may assume that its ISA allows the following operations:

- `PUSH M` - pushes the value stored at memory location M onto the operand stack.
- `POP M` - pops the operand stack and stores the value into memory location M.

- `OP` - Pops two values off the operand stack, performs the binary operation OP on the two values, and pushes the result back onto the operand stack.

Note: To compute A - B with a stack machine, the following sequence of operations are necessary: `PUSH A`, `PUSH B`, `SUB`. After execution of `SUB`, A and B would no longer be on the stack, but the value A-B would be at the top of the stack.

A one-address machine uses an accumulator in order to perform computations. For this problem, you may assume that its ISA allows the following operations:

- `LOAD M` - Loads the value stored at memory location M into the accumulator.
- `STORE M` - Stores the value in the accumulator into Memory Location M.
- `OP M` - Performs the binary operation OP on the value stored at memory location M and the value present in the accumulator. The result is stored into the accumulator (ACCUM = ACCUM OP M).

A two-address machine takes two sources, performs an operation on these sources and stores the result back into one of the sources. For this problem, you may assume that its ISA allows the following operation:

- `OP M1, M2` - Performs a binary operation OP on the values stored at memory locations M1 and M2 and stores the result back into memory location M1 (M1 = M1 OP M2).

Note 1: `OP` can be `ADD`, `SUB`, or `MUL` for the purposes of this problem.

Note 2: A, B, C, D, E and X refer to memory locations and can be also used to store temporary results.

1. Write the assembly language code for calculating the expression (do not simplify the expression):

   X = (A + (B × C)) × (D - (E + (D × C)))

   a. In a zero-address machine
   b. In a one-address machine
   c. In a two-address machine
   d. In a three-address machine like the LC-3b, but which can do memory to memory operations and also has a `MUL` instruction.

Write code here

1.a

```
PUSH B
PUSH C
MUL
PUSH A
ADD
PUSH D
PUSH D
PUSH C
MUL
PUSH E
ADD
SUB
MUL
POP X
```

1.b

```
LOAD B
MUL C
ADD A
STORE A
LOAD D
MUL C
ADD E
STORE B
LOAD D
SUB B
MUL A
STORE X
```

1.c

```
SUB X X
MUL B C
ADD A B
MUL C D
ADD C E
SUB D C
MUL A D
ADD X A
```

1.d

```
MUL B B C
ADD A A B
MUL C C D
```

```
ADD E E C
SUB D D E
MUL X A D
```

2. Give an advantage and a disadvantage of a one-address machine versus a zero-address machine.

Advantage: For the same program, the number of instructions of a one-address machine is fewer than that of a zero-address machine.

Disadvantage: A one-address machine has longer instructions, which makes it harder to decode and harder to implement compared to a zero-address machine.

Problem 8

Consider the following LC-3b assembly language program:

```
      .ORIG x3000
      AND R5, R5, #0
      AND R3, R3, #0
      ADD R3, R3, #8
      LEA R0, B
      LDW R1, R0, #1
      LDW R1, R1, #0
      ADD R2, R1, #0
AGAIN ADD R2, R2, R2
      ADD R3, R3, #-1
      BRp AGAIN
      LDW R4, R0, #0
      AND R1, R1, R4
      NOT R1, R1
      ADD R1, R1, #1
      ADD R2, R2, R1
      BRnp NO
      ADD R5, R5, #1
NO    HALT
B     .FILL XFF00
A     .FILL X4000
      .END
```

1. The assembler creates a symbol table after the first pass. Show the contents of this symbol table.

Symbol	Address
AGAIN	x300E
NO	x3022
B	x3024
A	x3026

2. What does this program do? (in less than 25 words)

Output 1 if the high byte and low byte of a word are the same, else 0.

3. When the programmer wrote this program, he/she did not take full advantage of the instructions provided by the LC-3b ISA. Therefore the program executes too many unnecessary instructions. Show what the programmer should have done to reduce the number of instructions executed by this program.

```
        .ORIG x3000
        AND R5, R5, #0
        LEA R0, B
        LDW R1, R0, #1
        LDW R1, R1, #0
        LSHF R3, R1, #8
        LDW R4, R0, #0
        AND R1, R1, R4
        XOR R1, R1, R3
        BRNP NO
         ADD R5, R5, #1
NO       HALT
B        .FILL Xff00
A        .FILL x4000
        .END
```

Use LSHF rather than loop;
Use XOR to check equality.

Problem 9
Consider the following two LC-3b assembly language programs.

```
      .ORIG x4000                      .ORIG x5000
MAIN1 LEA R3, L1                 MAIN2 LEA R3, L2
A1    JSRR R3                    A2    JMP R3
      HALT                             HALT

L1    ADD R2, R1, R0             L2    ADD R2, R1, R0
      RET                              RET
```

Is there a difference in the result of executing these two programs? If so, what/why is there a difference? Could a change be made (other than to the instructions at Labels A1/A2) to either of these programs to ensure the result is the same?

There is a difference.
JSRR instruction will store the incremented PC into R7, so the RET instruction of the subroutine L1 will reload R7 into the PC, which points to HALT.

JMP will directly load the address of L2 into the PC, without saving return address, so that the RET instruction in L2 may never come back to the next instruction of A2, because R7 may store any unknown value, which may make the program work incorrectly.

Problem 10
Use one of the unused opcodes in the LC-3b ISA to implement a conditionally executed ADD instruction. Show the format of the 16 bit instruction and discuss your reasoning assuming that:

1. The instruction doesn't require a steering bit. (The ADD is a register-register operation).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	DR			SR1			N	Z	P	SR2		

Since 1010 is an unused opcode, we can use that as conditionally executed ADD. We need 9 bit to put DR, SR1, and SR2. We also need 3 bits to put N,Z, and P. To keep consistency with other instructions, we put DR in bits[11:9], SR1 in bits[8:6], NZP in bits[5:3], and SR2 in bits[2:0].

2. The instruction requires a steering bit. (The ADD has both register-register and register-immediate forms).

Register -register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	DR/SR1			N	Z	P	0	0	0	SR2		

register-immediate

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	DR/SR1			N	Z	P	1	IMM5				

Since we don't have enough bits for 4-bit opcode, 3 registers, 1 steering bit and N,Z,P, we make it a restricted version, that is DR and SR1 should always be the same register, thus sharing bits[11:9]. We let bit[5] be the steering bit. When the steering bit is 0, it will use SR2 as the second operand, otherwise, it will take bits[4:0], a five-bit immediate value, as the second operand.

Problem 11

Discuss the tradeoffs between a variable instruction length ISA and a fixed instruction length ISA. How do variable length instructions affect the hardware? What about the software?

Below are advantages and disadvantages of variable length instructions and fixed length instructions.

For variable length instruction, it can take advantage of every bits of its instruction, which leads to a higher instruction density; Also we are able to design more complicated instructions with variable length instruction, which leads to a richer instruction set. Also, for the same program, using variable length instruction will lead to shorter codes. However, variable length instructions can be very hard to decode, and also harder to implement.
For fixed length instruction, it is easier to decode and implement. However, it has limited number of instructions, which decreases its richness. Also, for simple instructions with a few arguments, fixed length instruction means lower utilization of bits compared to variable length instructions.

The variable length instruction will influence hardware in several aspects. First, given its higher complexity, it will be harder for the hardware to decode and implement it. Specifically, it requires more hardware resources and more complicated hardware design.

For software, since variable length instruction offers more flexibility for programmers to design their programs, software programming will benefit a lot, especially for high-level languages, which also means the compiler will find it easier to compile those programs with higher-level languages.

Problem 12
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?

```
      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 if taking branch; 9 cycles if not taking branch
      STW R0, R3, #1 ;      15 cycles
      HALT ;                35 cycles
NUM   .FILL x4000
      .END
```

2. How many cycles does the entire program take to execute? (answer in terms of k)

9 + 9 + 15 + 15 + 9 + k * (9 + 9) + (k - 1)*10 + 9 + 15 + 35 = 28 k + 106

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.

Max(k) = floor(sqrt(2^16-1)) = floor(sqrt(65535)) = 255

4. How will you modify this program to support negative values of k? Explain in less than 30 words.

Since k*k output the same result for both positive and negative value of k, k should be transformed to –k if k < 0 before getting into the loop.

5. What is the new range of k?

k belongs to [-255, 255]

Problem 13

Please answer the following questions:

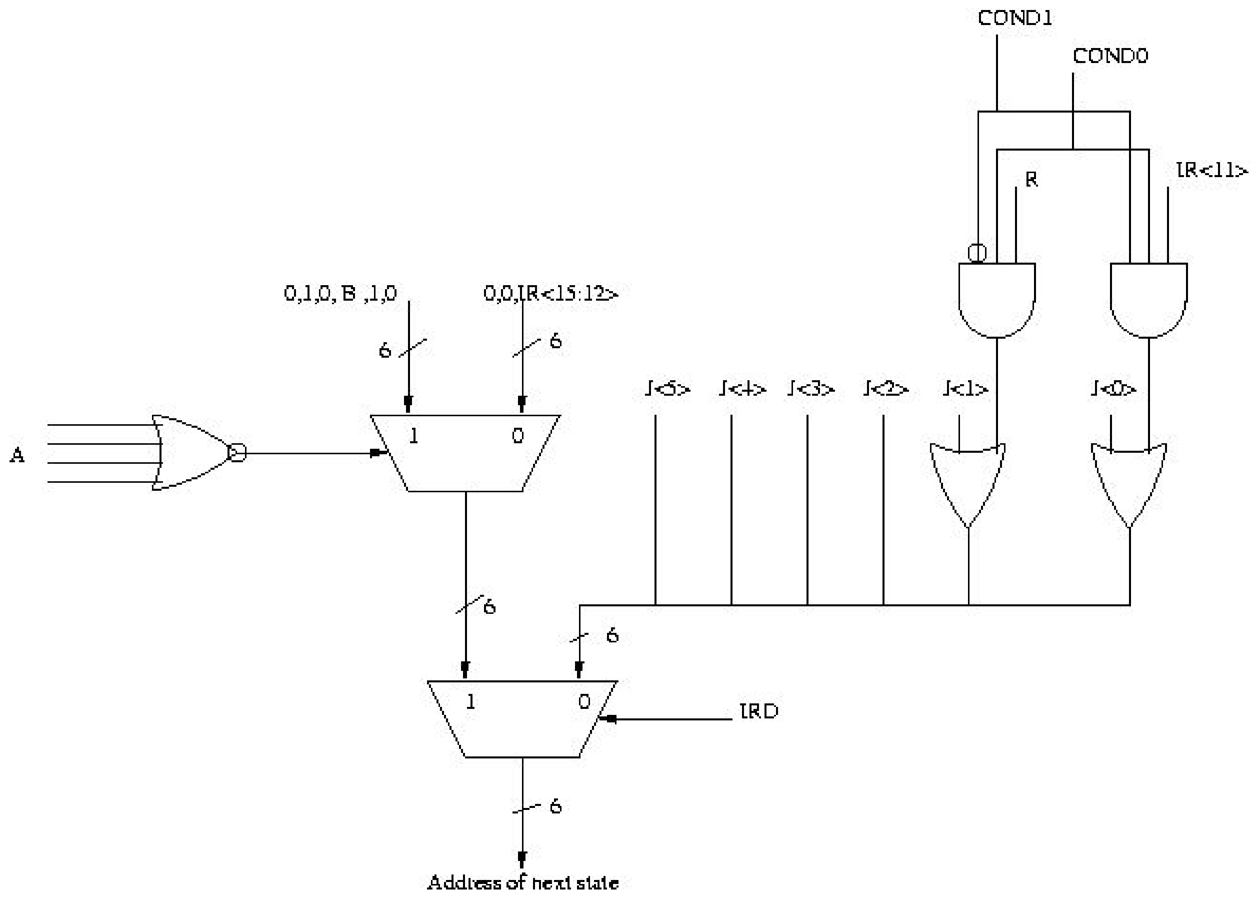
1. In which state(s) in the LC-3b state diagram should the `LD.BEN` signal be asserted? Is there a way for the LC-3b to work correctly without the `LD.BEN` signal? Explain.

State 32. We can always set BEN register to be 1, then we can remove LD.BEN register.

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.

We can get rid of BEN register by moving the calculation of it to state 0, where the BR need this BEN signal to decide whether to take branch.
It may not be a good idea. If we move BEN calculation to state 0, BEN signal cannot be calculated before getting into state 0, it may cause delay to calculate the address of the next state. Given the clock period must be long enough to allow the address calculation to be done, the aforementioned delay may cause increase in clock period.

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`?



A = IR[15:12]
B = IR[11]&N + IR[10]&Z + IR[9]&P

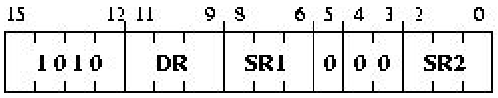
Problem 14

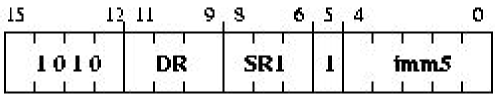
We wish to use the unused opcode “1010” to implement a new instruction ADDM, which (similar to an IA-32 instruction) adds the contents of a memory location to either the contents of a register or an immediate value and stores the result into a register. The specification of this instruction is as follows:

Assembler Formats

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

Encodings

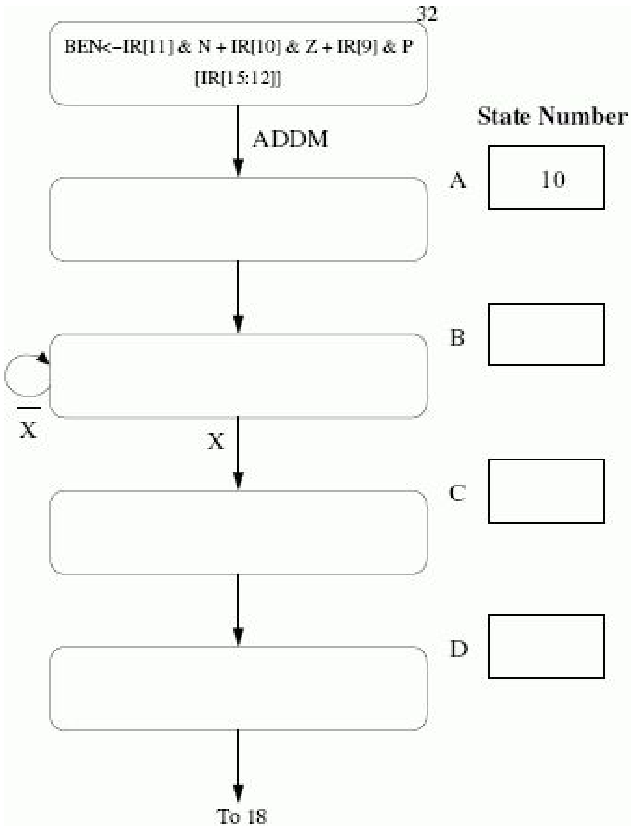




Operation

```
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 2 asks you to show the necessary changes to the datapath



Use the table to fill in your answers

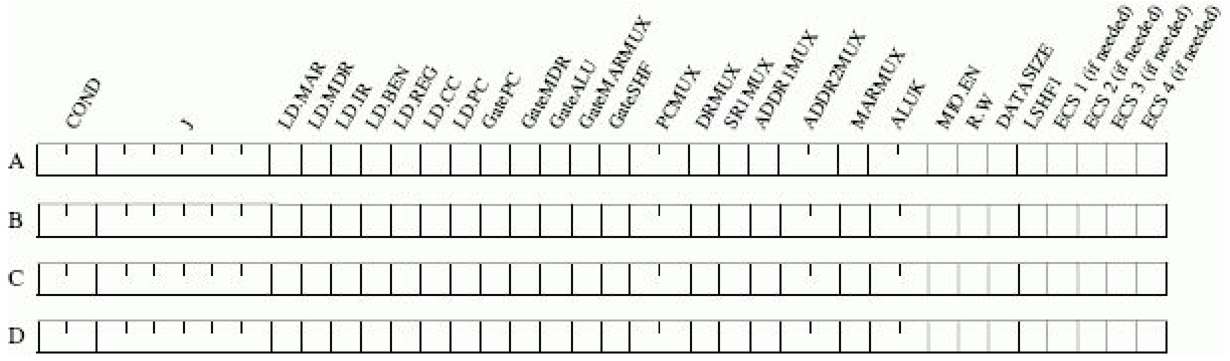
State	Number	Action
A	10	MAR <- SR1
B	36	MDR <- M[MAR]
C	38	SRT* <- MDR *SRT is an extra register to hold data from memory
D	39	DR <- SRT + OP2** ** OP2 may be SEXT[imm5] or SR2

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.

In state 38, we need SRT to take the data on the data bus from MDR, so we have to add an extra control signal ECS 1 to control when SRT should take data from the bus. If ECS 1 is asserted, data from MDR will be loaded to SRT, otherwise, SRT will not load data from the bus.

For the original ADD instruction, one of the operand of ALU is connected to SR1, but for our ADDM instruction, it could be from SRT. So, we have to add a MUX2 to choose the register the ALU should take. For this MUX, we have to add the second control signal ECS 2. If ECS 2 is asserted, data in SRT will be loaded into the operand of ALU, otherwise, SR1 will be loaded into the operand of ALU.

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.



	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																													
B																													
C																													
D																													

State	Cond	J	Other Control signals that are 1
A	00	100100	LD.MAR ; GateALU ; SR1MUX; ALUK[1:0]
B	01	100100	LD.MDR; MIO.EN; DATA.SIZE;
C	00	100111	GateMDR; DATA.SIZE; ECS 1
D	00	010010	LD.REG; LD.CC; GateALU; ECS 2;

Extra work

Extra work