Problem Set 1 - Solutions

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

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?

ISA is the contract between software and hardware, and microarchitecture refers to a specific implementation of the ISA. Compiler does not need to know any of the microarchitecture or the implementation specific details for compilation process.

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 causes x0004 to be stored in location x3000 when the assembled code is loaded into the memory. The second program causes x0004 to be stored in x3000 during the execution of the program.

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.

NOP doesn't affect the architecture state ie Registers, Memory, Condition codes and PC.
1. ADD R1, R1, #0 – Can't be used as a NOP as the conditional registers might change which a NOP is not supposed to do
2. BRnzp #0 (Branch unconditionally with '0' offset) – Can be used as a NOP
3. Never branch, nzp check set to 0 – Can be used as a 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. Little endian : MEM[x1000] + MEM[x1002] = x1A0E + x0C11 = x261F
2. Big endian : 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. The memory consists of 2^{28} addressable locations. 28 bits are required to uniquely address each location.
2. The memory consists of $2^{28}/2^{3}$ addressable locations. 25 bits are required to uniquely address each location.
3. The memory consists of $2^{28}/2^{7}$ addressable locations. 21 bits are required to uniquely address each location.

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):

   $X = (A + (B \times C)) \times (D - (E + (D \times C)))$

   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.

a. In a zero-address machine

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

Advantages: Operate instructions only require an opcode so they can be encoded very densely.
Disadvantages: Not flexible in terms of manipulating operands, more instructions required to write programs.

b. In a one-address machine

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

Advantages: Only a single register required, fewer instructions compared to the stack machine.
Disadvantage: Not as flexible as 2 or 3-address machines. We need instructions to move data to and from the accumulator (We don't need these in a 3-address machine that supports memory-to-memory operations).

c. In a two-address machine

```
MUL B, C
ADD A, B
MUL C, D
ADD C, E
SUB D, C
MUL A, D
SUB X, X    ; these two instructions emulate a MOV X, A
ADD X, A
```

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

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

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

Answered in part 1 of question

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)

If the high and low byte of the word stored at location x4000 are equal, R5 is set to 1, else to 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.

There are several possible answers to this code optimization question

a. The programmer used a loop to left shift a value in R2 by 8 bits. He/she could have done this using a single LC-3b instruction, LSHF. He/she should have replaced the loop

```
AGAIN   ADD R2, R2, R2
        ADD R3, R3, #-1
        BRp AGAIN
```

with

```
        LSHF R2, R2, #8
```

b. Instead of using "subtraction" to compare the high and low byte, the programmer could have used a single XOR instruction to check the equality of the two bytes. He/she could have replaced the following instructions

```
        NOT R1, R1
        ADD R1, R1, #1
       ADD R2, R2, R1
```
with
```
       XOR R2, R1, R2
```
c. The program is comparing the high byte and low byte of the word in memory location x4000. The programmer could have utilized the LDB instruction to load the high byte and low byte into two separate registers and compare them. This way there would be no need for shifting and masking. The optimized program could look like this:

```
      .ORIG  x3000
   AND    R5, R5, #0
   LEA    R0, A
   LDW    R0, R0, #0
   LDB    R1, R0, #0
   LDB    R2, R0, #1
   XOR    R2, R1, R2
   BRnp   NO
   ADD    R5, R5, #1
NO HALT
A    .FILL   x4000
     .END
```

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?

Yes, there is a difference. The program at x5000 does not save the return address from the subroutine at L2 because it uses a JMP instruction. Thus, that program will not work correctly.

A possible change that could be made:

```
      .ORIG x5000
      LEA   R7, B
MAIN2 LEA   R3, L2
A2    JMP   R3
B     HALT
      ;
L2    ADD   R2, R1, R0
      RET
```

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).

4 opcode bits, 3 NZP bits, 3 DR bits, 3 SR1 bits, 3 SR2 bits

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

A 2 address operation can be used instead of a 3 address. Thus, we have 4 opcode bits, 3 NZP bits, 3 DR/SR1 bits, 1 steering bit. The remaining 5 bits will either be used as 5 immediate bits, or 2 unused bits + 3 SR2 bits.

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?

Variable instruction length ISAs have more complex decode logic. Variable instruction length ISA programs can be encoded more densely. Variable instruction length ISAs also generally imply a richer instruction set than that of a fixed length ISA. Since a richer instruction set has more instructions that directly correspond to higher level language programming constructs, the compilation process can be easier.

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?

ADD – 9 cycles
AND – 9 cycles
XOR – 9 cycles
TRAP – 15 cycles when TRAP instruction is not HALT, 35 when TRAP is HALT
SHF – 9 cycles
LEA – 9 cycles
LDB – 15 cycles
LDW – 15 cycles
STW – 15 cycles
STB – 15 cycles
JSR – 10 cycles
JMP – 9 cycles
BR – 9 when branch not taken, 10 when taken

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
```

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

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.

1. Number of cycles = 9 + 9 + 15 + 15 + 9 + (k-1)*(9 + 9 + 10) + 1*(9 + 9 + 9) + 15 + 35 = 28k + 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.

   k = 255

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

   After we load the value of k, check if it is negative. If so, take the 2's complement before entering the loop.

5. What is the new range of k?

   The new range of k is -255 to 255 (since the squares of negative numbers is always positive) assuming that the result is 16-bit unsigned

Problem 13

Please answer the following questions:

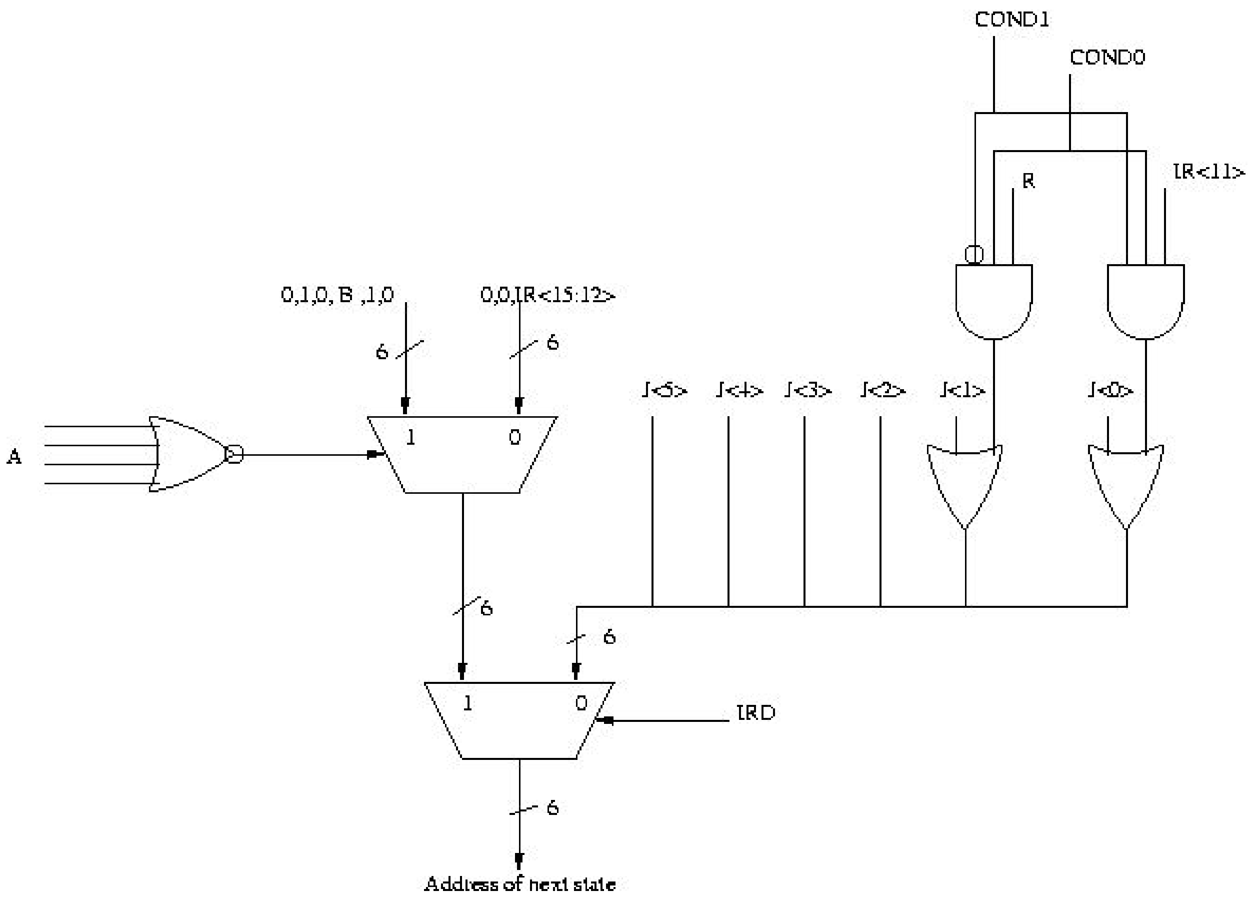
1. In which state(s) in the LC-3b state diagram should the `LD.BEN` signal be asserted? Is there a way for the LC-3b to work correctly without the `LD.BEN` signal? Explain.

   State 32. We can get rid of the LD_BEN signal altogether and always load enable the 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.

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 might cause the cycle time to be increased.

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 (i.e., the old BEN signal)

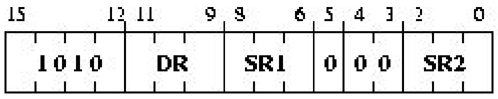
Problem 14

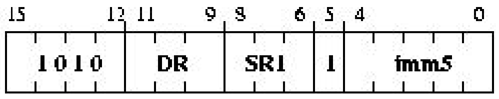
We wish to use the unused opcode “1010” to implement a new instruction ADDM, which (similar to an IA-32 instruction) adds the contents of a memory location to either the contents of a register or an immediate value and stores the result into a register. The specification of this instruction is as follows:

Assembler Formats

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

Encodings





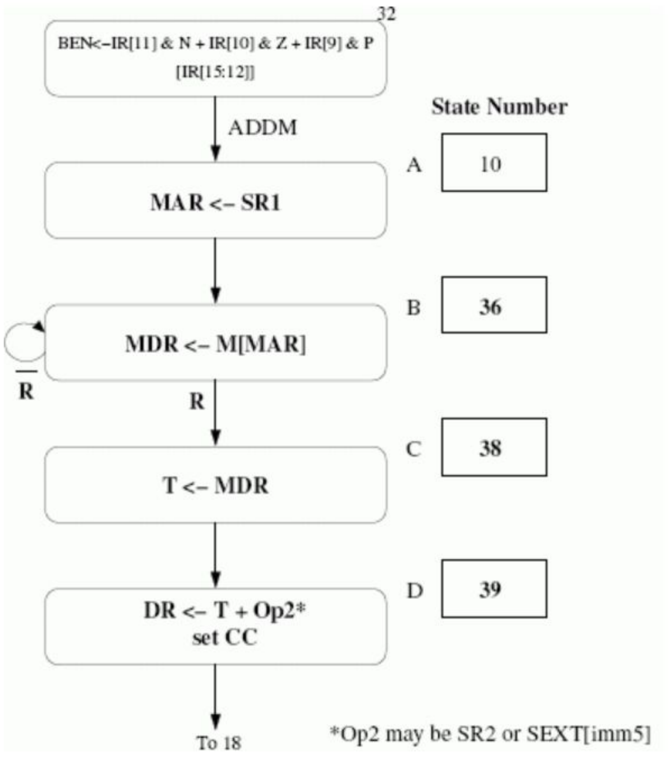
Operation

```
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

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. 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.

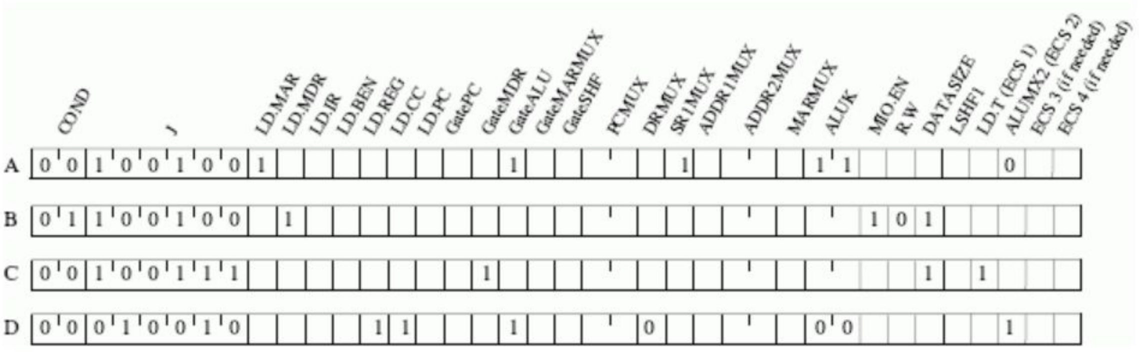
We need a 16-bit temporary register (T) which gets its inputs from the system bus. We need a signal LD.T (extra control signal 1) to control when to load this register. This register holds the data that is fetched from memory. We also need a mux in front of the A input of the ALU. This mux should select 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).

ALUMX2 = 0 selects SR1

ALUMX2 = 1 selects T

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.

Filled in microinstructions:



	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	LD.T (ECS 1)	ALUMX2 (ECS 2)	ECS 3 (if needed)	ECS 4 (if needed)
A	0 0	1 0 0 1 0 0	1									1					1				1 1						0		
B	0 1	1 0 0 1 0 0		1																		1	0	1					
C	0 0	1 0 0 1 1 1									1													1		1			
D	0 0	0 1 0 0 1 0					1	1				1				0					0 0						1		

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.