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*](http://users.ece.utexas.edu/~patt/18f.460N/handouts/ch07.pdf) *and the* [*LC-3b ISA*](http://users.ece.utexas.edu/~patt/18f.460N/handouts/appA.pdf)*,* [*microarchitecture*](http://users.ece.utexas.edu/~patt/18f.460N/handouts/appC.pdf)*, and* [*state diagram*](http://users.ece.utexas.edu/~patt/18f.460N/handouts/state_machine.pdf) *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?

|  |
| --- |
| Microarchitecture level is a specific implementation of ISA and Not visible to the ISA. You can’t tell difference between two microarchitectures without measuring time or resources. ISA serves is the contract between software and hardware. Software that has been written for an ISA can run on different implementations of the same ISA.  Nothing. |

### **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 |

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

|  |
| --- |
| Microarchitecture |

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

|  |
| --- |
| ISA |

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

|  |
| --- |
| ISA |

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

|  |
| --- |
| ISA |

1. It takes *n* cycles to execute an ADD instruction.

|  |
| --- |
| ISA |

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

|  |
| --- |
| ISA |

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

|  |
| --- |
| ISA |

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

|  |
| --- |
| Microarchitecture |

1. 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 directly uses ‘.FILL’ to tell the assembler to set aside the next location (which is x3000) in the program and initialize it with the value of x0004.  The second program uses ‘.FILL’ to tell the assembler to set aside the next location in the program and initialize it with the value of x3000 with a label ‘A’. When the program runs, it starts at address x4000(.ORIG). Then it clears R0 and adds 4 to it. Now R0 has value of x0004. Then it loads the address of A to R1. After that it loads the value in the address of R1 to R1. Now R1 has value of x3000. Finally, the contents of R0 is stored in the address of R1, which is to say that x0004 is stored in the address of x3000. |

### 

### **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.

|  |
| --- |
| All of the three instructions can be used for 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. x261F       2. 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

|  |
| --- |
| 32x1024x1024x8 bits totally.  1. 28bits  2. 25bits  3. 21bits |

### **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)))**
   1. In a zero-address machine
   2. In a one-address machine
   3. In a two-address machine
   4. In a three-address machine like the LC-3b, but which can do memory to memory operations and also has a MUL instruction.

|  |
| --- |
| 1.  PUSH B;  PUSH C;  MUL;  PUSH A;  ADD;  POP A;  PUSH D;  PUSH C;  MUL;  PUSH E;  ADD;  POP E;  PUSH D;  PUSH E;  SUB;  PUSH A;  MUL;  2.  LOAD B;  MUL C;  ADD A;  STORE A;  LOAD D;  MUL C;  ADD E;  STORE E;  LOAD D;  SUB E;  MUL A;  3.  MUL B C;  ADD A B;  MUL C D;  ADD E C;  SUB D E;  MUL A D; |

|  |
| --- |
| 4.  MUL B B C;  ADD A A B;  MUL C C D;  ADD E E C;  SUB D D E;  MUL A A D; |

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

|  |
| --- |
| Advantage: simpler to write codes  Disadvantage: usage of an extra register. |

### 

### **Problem 8**

Consider the following LC-3b assembly language program:

.ORIG x3000  
 AND R5, R5, #0 0  
 AND R3, R3, #0 2  
 ADD R3, R3, #8 4  
 LEA R0, B 6  
 LDW R1, R0, #1 8  
 LDW R1, R1, #0 A  
 ADD R2, R1, #0 C  
AGAIN ADD R2, R2, R2 E  
 ADD R3, R3, #-1 10  
 BRp AGAIN 12  
 LDW R4, R0, #0 14  
 AND R1, R1, R4 16  
 NOT R1, R1 18  
 ADD R1, R1, #1 1A  
 ADD R2, R2, R1 1C  
 BRnp NO 1E  
 ADD R5, R5, #1 20  
NO HALT 22  
B .FILL XFF00 24  
A .FILL X4000 26  
 .END 28

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

|  |
| --- |
| AGAIN X300E  NO X3022  B X3024  A X3026 |

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

|  |
| --- |
|  |

1. 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 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  NO HALT  B .FILL XFF00  A .FILL X4000   .END |

### **Problem 9**

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

|  |  |
| --- | --- |
| .ORIG x4000  MAIN1 LEA R3, L1  A1 JSRR R3  HALT  L1 ADD R2, R1, R0  RET | .ORIG x5000  MAIN2 LEA R3, L2  A2 JMP R3  HALT  L2 ADD R2, R1, R0  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?

|  |
| --- |
| No difference |

### **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).

|  |
| --- |
| 1010 DR SR1 npz SR2 |

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

|  |
| --- |
| 1010 DR SR1 0np SR2  1010 DR SR1 1np imm3 |

### **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?

|  |
| --- |
| For a variable length instruction ISA, it is hard to distinguish where is the end of an instruction. We have to wait for more instructions to determine the boundary of two instructions. It is also more complicated to design. However, due to the length of a variable length instruction is often smaller, it has greater code density.  For a fixed instruction length ISA, it is easy to find the next instruction because the instruction length is fixed no matter what instruction type it is. It makes fetching and parsing more than one instruction per cycle relatively simple and thus increasing the execution speed.  The hardware has to pay more attention to find the start of instructions. |

### **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?

|  |
| --- |
|  |

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

|  |
| --- |
| 55+10k |

1. 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.

|  |
| --- |
|  |

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

|  |
| --- |
|  |

1. What is the new range of k?

|  |
| --- |
|  |

### **Problem 13**

Please answer the following questions:

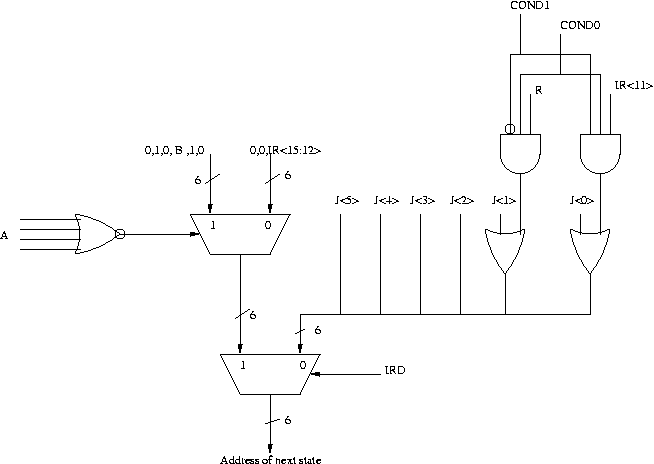
1. In which state(s) in the LC-3b state diagram should the LD.BEN signal be asserted? Is there a way for the LC-3b to work correctly without the LD.BEN signal? Explain.

|  |
| --- |
|  |

1. 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.

|  |
| --- |
|  |

1. 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?



|  |
| --- |
|  |

### 

### 

### 

### **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

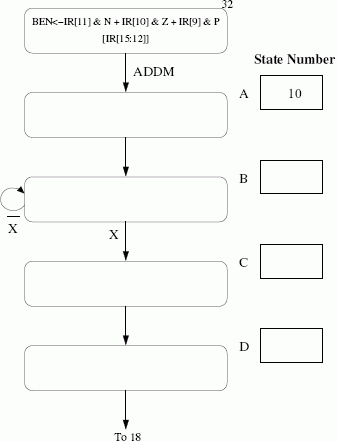
### **Encodings**

**ADDM instruction encoding**

### **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

****

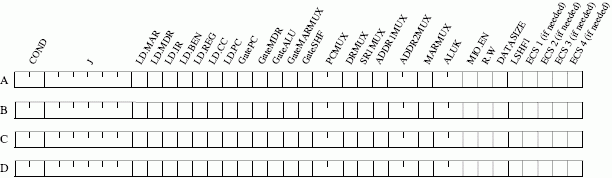
**Use the table to fill in your answers**

|  |  |  |
| --- | --- | --- |
| **State** | **Number** | **Action** |
| **A** | **10** |  |
| **B** |  |  |
| **C** |  |  |
| **D** |  |  |

1. 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.

|  |
| --- |
|  |

1. 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](http://users.ece.utexas.edu/~patt/16f.460N/handouts/360n.appC.pdf). 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.



|  |  |  |  |
| --- | --- | --- | --- |
| State | Cond | J | Other Control signals that are 1 |
| A |  |  |  |
| B |  |  |  |
| C |  |  |  |
| D |  |  |  |

|  |
| --- |
|  |

|  |
| --- |
| Extra work |

|  |
| --- |
| Extra work |