Department of Electrical and Computer Engineering

The University of Texas at Austin

EE 460N, Spring 2023
Problem Set 1
Due: Feb 20th, before class

Yale N. Patt, Instructor
Michael Chen, Kayvan Mansoorshahi, 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 Gradescope. Only one student should submit the problem set on behalf of the group, but everyone should create a gradescope account and be tagged on the homework. The entry code is **9RPGX3**.

Note: You are to turn in the student information sheet along with your picture on Sept. 16th as well.

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

Questions

Problem 1
Briefly explain the advantage of microarchitecture/ISA separation.

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.
2. The ALU of LC-3b does not have a subtract unit.
3. LC-3b has three condition code bits (n, z, and p).
4. The n, z, and p bits are stored in three 1-bit registers.
5. A 5-bit immediate can be specified in an `ADD` instruction
6. It takes *n* cycles to execute an `ADD` instruction.
7. There are 8 general purpose registers used by operate, data movement and control instructions.
8. The registers MDR (Memory Data Register) and MAR (Memory Address Register) are used for Loads and Stores.
9. A 2-to-1 mux feeds one of the inputs to ALU.
10. The register file has one input and two output ports.

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
```

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.

Problem 5

Consider the following possibilities for saving the return address of a subroutine:

1. In a processor register.
2. In a memory location associated with the subroutine; i.e., a different memory location is used for each different subroutine.
3. On a stack.

Which of these possibilities supports subroutine nesting, and which supports subroutine recursion (that is, a subroutine that calls itself)?

Problem 6

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

Problem 7

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

Problem 8

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

Problem 9

The following table gives the format of the instructions for the LC-1b computer that has 8 opcodes.

Opcode	7	6	5	4	3	2	1	0
ADD	0	0	0	DR		A	SR	
AND	0	0	1	DR		A	SR	
BR(R)	0	1	0	N	Z	P	TR	
LDImm	0	1	1	signed immediate				
LEA	1	0	0	signed offset				
LD	1	0	1	DR		0	TR	
ST	1	1	0	SR		0	TR	
NOT	1	1	1	DR		0	0	0

Notes:

- Interpretation of all instructions is similar to that of the LC-3b, unless specifically stated otherwise.
- The destination register for the instructions `LDImm` and `LEA` is always register R0. (e.g. `LDImm #12` loads decimal 12 to register R0.)
- TR stands for Target Register. In the case of the conditional branch instruction `BR`, it contains the target address of the branch. In the case of `LD`, it contains the address of the source of the load. In the case of `ST`, it contains the address of the destination of the store.
- `ADD` and `AND` provide immediate addressing by means of a steering bit, bit[2], labeled A. If A is 0, the second source operand is obtained from SR. If A is 1, the second source operand is obtained by sign-extending bits[1:0] of the instruction. A bit is called a “steering” bit if its value “steers” the interpretation of other bits (instruction bits 1:0 in this case).
- Bits labeled 0 must be zero in the encoding of the instruction.

1. What kind of machine (n-address) does the above ISA specification represent?
2. How many general purpose registers does the machine have?
3. Using the above instructions, write the assembly code to implement a register to register mov operation.
4. How can we make a PC-relative branch? (HINT: You will need more than one LC-1b instruction; also, note that the LEA instruction does NOT set condition codes)

Problem 10

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

Problem 11

Consider the following LC-3b assembly language program.

```
      .ORIG x4000
MAIN  LEA R2,L0
      JSRR R2
```

```
        JSR L1
        HALT

L0      ADD  R0,R0,#5
        RET

L1      ADD  R1,R1,#5
        RET
```

1. Assemble the above program. Show the LC-3b machine code for each instruction in the program as a hexadecimal number.
2. This program shows two ways to call a subroutine. One requires two instructions: `LEA` and `JSRR`. The second requires only one instruction: `JSR`. Both ways work correctly in this example. Is it ever necessary to use `JSRR`? If so, in what situation?

Problem 12

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?

Problem 13

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

Problem 14

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?

Problem 15

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
```

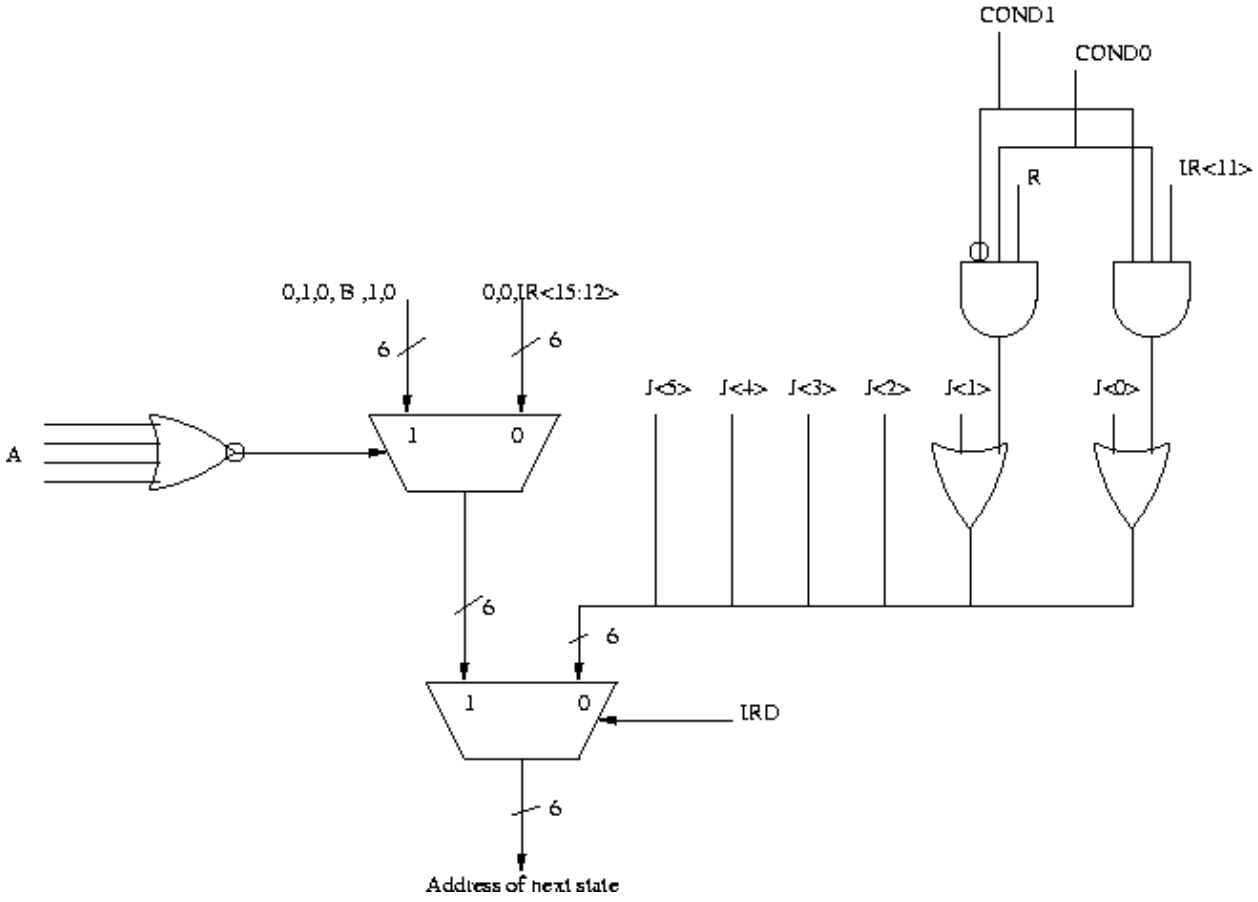
1. How many cycles does each instruction take to execute on the LC-3b microarchitecture described in Appendix C?
2. How many cycles does the entire program take to execute? (answer in terms of k)
3. What is the maximum value of k for which this program still works correctly? Note: Treat the input and output values as 16-bit unsigned values for part 3. We will extend the problem to 2's complement values in part 4.
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?

Problem 16

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`?



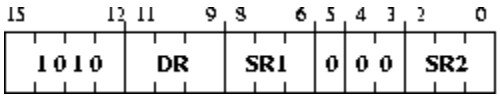
Problem 17

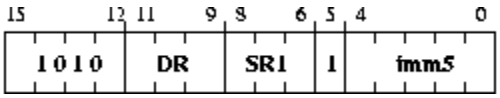
We wish to use the unused opcode “1010” to implement a new instruction ADDM, which (similar to an IA-32 instruction) adds the contents of a memory location to either the contents of a register or an immediate value and stores the result into a register. The specification of this instruction is as follows:

Assembler Formats

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

Encodings

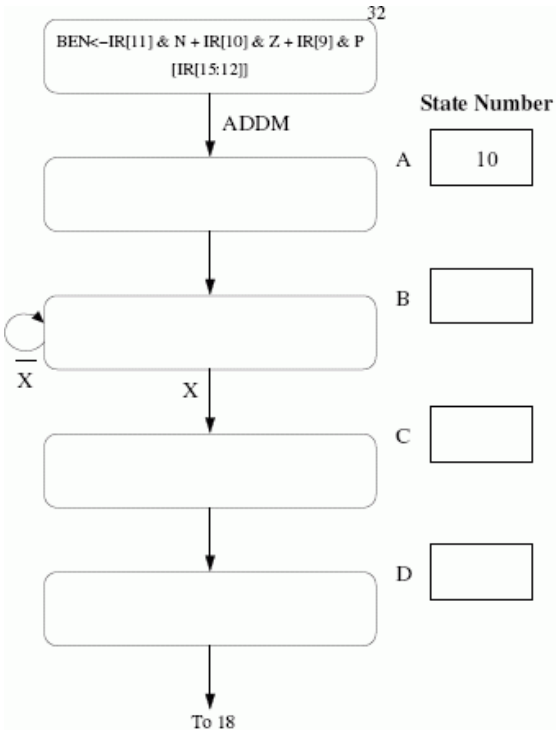




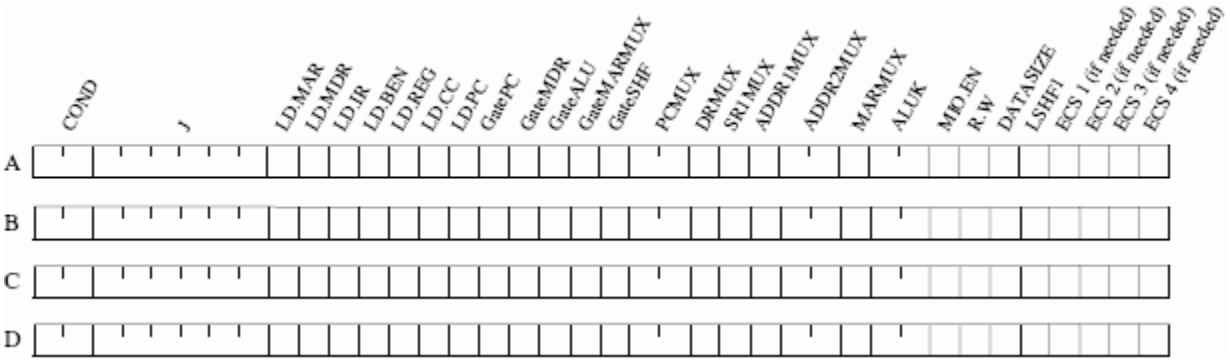
Operation

```
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



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.



Problem 18

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.

Problem 19

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.

Problem 20

Please go to the handouts section of the course web site, print and fill out the student information sheet, and turn it in with a recognizable recent photo of yourself on September 16th (the same day this problem set is due). Please submit an electronic copy.