CS 2110 Homework 05
LC-3 Introduction

Madison Grams, Jarrett Schultz, Joshua Viszlai, and Maddie Brickell

Fall 2018

Contents

1 Overview

So far in CS 2110 you've learned the basics of digital logic and how to build simple circuits, but how does all of this become relevant to computer science? This assignment will help pull it altogether as you will work step-by-step and build a functional computer! This computer will be able to run simple programs and be a stepping stone into the rest of the material in this class.

It is very important that you read ALL of this PDF before beginning this assignment, as it goes into much detail about how to do this homework.

2 Instructions

IMPORTANT: Please make sure you are running **CircuitSim 1.8.0** or else this assignment will not work!

Your friend was building his own computer in CircuitSim, but then he accidentally deleted some portions of it. Now, your friend needs your help to fill in the missing parts! The computer you will be building will be a modification of the LC-3 known as the LC-M. The LC-M supports the following instructions:

- `ADD`
- `AND`
- `BR`
- `LDR`
- `LEA`
- `NOT`
- `STR`
- `HALT`

We have provided `LCM.sim`, which has the following blank subcircuits where you will be creating various components:

- `ALU`
- `CC-Logic`
- `PC`
- `ADDR`
- `SRC2MUX`

You **should not edit anything in the main circuit or other subcircuits**, however, feel free to take a look at them to get a better understanding of the rest of a computer!

Before starting, we recommend that you have a basic understanding of the LC-M's cousin, the LC-3. Chapters 4 and 5 of the Patt & Patel textbook are a good resource, as well as Appendix A, which goes into more detail about each instruction. The only new instruction is HALT. In the LC-3, there is an instruction, TRAP, which allows for calling a variety of other pre-created instructions in memory. You will not have to worry about TRAPs for this assignment and merely interpret the TRAP opcode as HALT, which indicates the end of a program.

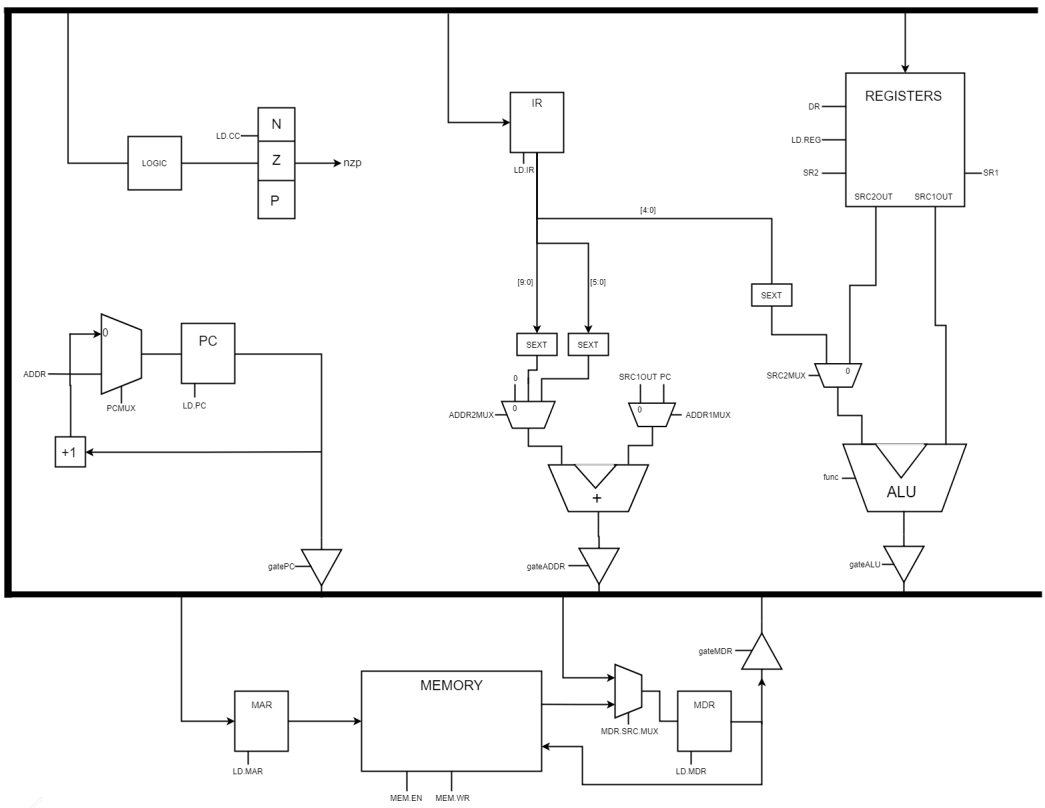


Figure 1: Diagram of the LC-M

2.1 Program Counter (PC)

The PC is a 16 bit register that holds the address of the next instruction to be executed. During the FETCH stage, the contents of the PC are loaded into the memory address register (MAR), and the PC is updated with the address of the next instruction. There are two scenarios for updating the PC:

1. The contents of the PC are incremented by 1.
2. The result of the ADDR is the address of the next instruction. The output from the ADDR should be stored in the PC. This occurs if we use the branching instruction, (`BR`).

2.2 Instruction Register (IR)

The IR is a 16 bit register that holds the machine code of the current instruction. During the FETCH stage, the LC-M interrogates memory for the instruction, which gets loaded into the memory data register (MDR). Then, the contents of the MDR are loaded into the IR. The contents of the IR are then decoded and executed. For more reference on how instructions are stored, check out the diagram in section 2.7.2 of the PDF.

2.2.1 Sign Extension

There are two LC-M instructions that use constants: `ADD` and `AND`. For example, `ADD R0, R1, 2` adds the constant 2 to the contents of register 1 and stores the result in register 0. This constant is called imm5

(immediate 5) and is stored in the 5 least significant bits of the instruction. These 5 bits are sign extended to 16 bits so that imm5 can be used in other components of the LC-M.

2.3 ADDR

The ADDR is used to calculate addresses during the following instructions:

- `BR | 0000 | nzp | PCoffset9 |`
  Here, the PCoffset9 (the 9 least signifcant bits of the instruction) is sign extended to 16 bits and added to the contents of the PC.
- `LDR | 0110 | DR | baseR | offset6 |`
  Offset6 (the 6 least significant bits of the instruction) is sign extended to 16 bits and added to the contents of the base register, which can be accessed by SRC1OUT.
- `LEA | 1110 | DR | PCoffset9 |`
  PCoffset9 (the 9 least signifcant bits of the instruction) is sign extended to 16 bits and added to the contents of the PC.
- `STR | 0111 | SR | baseR | offset6 |`
  Offset6 (the 6 least significant bits of the instruction) is sign extended to 16 bits and added to the contents of the base register, which can be accessed by SRC1OUT.

Notice that in each of these instructions we are adding either SR1OUT or PC to either sign-extended offset6 or sign-extended offset9 from IR. Hint: you will need two multiplexers for this part.

2.4 ALU

The LC-M uses an ALU to execute various instructions, such as `ADD` and `NOT`. The ALU takes in two 16 bit inputs and has four operations:

0. A + B
1. A AND B
2. NOT A
3. PASS A (A is unchanged)

2.5 SRC2MUX

The LC-M's ALU has two inputs: the A input is from SRC1OUT and the B input is from SRC2MUX. The SRC2MUX is a one-bit multiplexer that selects the second input to the ALU. It has two choices depending on the value of bit 5 in the IR:

1. SRC2OUT. This refers to the contents of the second source register in the instruction. For example, for the instruction `AND R0, R1, R2`, the register file outputs the contents of R2 as SRC2OUT and SRC2MUX selects this as the B input to thw ALU.
2. imm5. imm5 is the 5-bit constant in the instruction word. For example, the instruction `AND R0, R1, #4`, SRC2MUX selects the sign-extended 5-bit imm5 from the bits 0-4 of the instruction.

2.6 Condition Codes (CC)

The LC-M has three condition codes: N (negative), Z (zero), and P (positive). These codes are set after the LC-M executes instructions that include loading a result into a register, such as `ADD` and `LDR`. For example, if `ADD R2, R0, R1` results in a positive number, then NZP = 001. Opcodes that are followed by "+" in 2.7.2 are operations that set the condition codes. The CC subcircuit should set N to 1 if the input is negative, set Z to 1 if the input is zero, and set P to 1 if the input is positive. Only 1 bit in NZP should be set at any given time. Zero is not considered a positive number.

Hint: you are allowed to use a comparator, which can be found under Arithmetic!

2.7 Running a Program

Now that we've finished building the LC-M, we want to run a program with it! We have provided a simple assembly program which you will need to translate to machine code so that you can execute it on the LC-M. The program is equivalent to:

```
int i = 5;
while (i > 0) {
    i = i - 2;
}
```

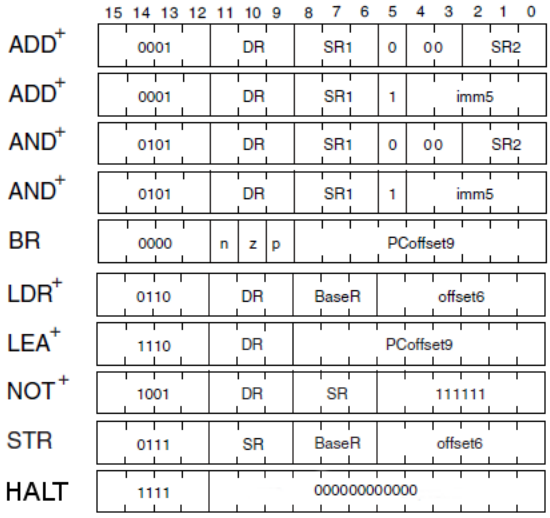
with a couple extra instructions special to assembly to help test your circuit.

2.7.1 Machine Code

So far in most Computer Science courses, you've likely mostly only been exposed to high-level languages such as Python or Java and while these are readable somewhat easily by a human, they cannot be directly interepreted by a computer. The language of computers is binary. The process to get to binary for a language like C (which you will learn later in this course!) is compilation to assembly and then assembling that to machine code which is written in binary. You will function as the assembler for this program.

2.7.2 Instruction Format

The LC-M is set up to interpret instructions in a certain format. The chart below shows how instructions are stored to help you understand how to convert the assembly instructions to binary.



For any instruction, the 4 most significant bits are the opcode which indicates which instruction is being performed. For example, if an instruction said `ADD R3, R3, R4`, which means take the contents of register 3 and register 4, add them and store the value back into register 3, the machine code would be as follows: 0001 011 011 000 100.

Some hints: Values indicated as offset or imm simply need to sign extended to the correct number of bits (such as 3 for imm5 would be written as 00011). Values called PCoffset are determined based on the value of the PC at the time. The excel sheet you're given indicates the PC at each instruction to assist you with these calculations.

2.7.3 Loading the instructions to the LC-M

After you have converted the instructions to machine code, the excel document will have converted them into hexadecimal. Copy the hexadecimal into a blank text file called `machinecode.txt`. In the main circuit, click the RST button and then right click on the RAM component and choose "Edit Contents". A new window should pop up and in the upper right hand corner you should see a button that says "Load from file", which you can click on and then navigate to where you saved `machinecode.txt` and load it. The LC-M is now set up to run a program!

2.7.4 Running the Program

You can run the program manually by clicking the CLK on and off, but you can also use the simulator tool in Circuit Sim to run it faster. It's important to pay attention to the way the control signals change in different states. You also can double click subcircuits while the program is running to see more specifics as the program runs (such as the values changing in the registers).

When the program ends, the isHalt LED in the main circuit will be red. When running it on the highest frequency, the program should not take more than a couple seconds to run.

2.7.5 Information about State Machines

An important part of running a program is the state machine. The state machine keeps track of what state the computer is in (such as FETCH), and outputs the proper control signals for each state (such as the value of a selector for a multiplexer). The state machine has already been built for you so that you will be able to easily run a program, but try and pay attention to see what control signals (indicated by LEDs in the

main circuit) light up during different times of a program. An important note to this is that while many instructions (such as ADD and NOT) can be implemented in one clock cycle, some (such as LDR) can take even more!

Hint: The state machine always follows this format:

$$\text{FETCH} \rightarrow \text{DECODE} \rightarrow \text{INSTR}$$

Fetch is 3 cycles that involves incrementing the PC and loading the instruction from memory into the instruction register. The decode state is 1 cycle and uses the opcode from the instruction to determine which instruction is being executed. The instruction state takes a variable number of cycles dependent on the instruction and once completed, the computer returns to the fetch state.

3 Testing Your Work

You can run the autograder using Gradescope which will test individual circuits as well as your machine code.

You can also run your machine code on your completed computer! One way to indicate that the program ran correctly is to look at the Register File (DPRF) after the program ran. A correctly run program will have these values in the registers:

```
R0:  0000
R1:  0000
R2:  0003
R3:  ffff
R4:  fffe
R5:  000f
R6:  0000
R7:  0000
```

This isn't a guarantee that the circuit is fully correct, but a good way to measure a correctly built circuit.

4 Rubric

This homework will be demoed! Look out for demo times to be released after the homework is due.

5 Deliverables

Please upload the following files to Gradescope:

1. `LCM.sim`
2. `machinecode.txt`

Be sure to check your score to see if you submitted the right files, as well as your email frequently until the due date of the assignment for any potential updates.

6 Rules and Regulations

6.1 General Rules

1. Starting with the assembly homeworks, any code you write must be meaningfully commented. You should comment your code in terms of the algorithm you are implementing; we all know what each line of code does.

2. Although you may ask TAs for clarification, you are ultimately responsible for what you submit. This means that (in the case of demos) you should come prepared to explain to the TA how any piece of code you submitted works, even if you copied it from the book or read about it on the internet.

3. Please read the assignment in its entirety before asking questions.

4. Please start assignments early, and ask for help early. Do not email us the night the assignment is due with questions.

5. If you find any problems with the assignment it would be greatly appreciated if you reported them to the author (which can be found at the top of the assignment). Announcements will be posted if the assignment changes.

6.2 Submission Conventions

1. All files you submit for assignments in this course should have your name at the top of the file as a comment for any source code file, and somewhere in the file, near the top, for other files unless otherwise noted.

2. When preparing your submission you may either submit the files individually to Canvas/Gradescope or you may submit an archive (zip or tar.gz only please) of the files. You can create an archive by right clicking on files and selecting the appropriate compress option on your system. Both ways (uploading raw files or an archive) are exactly equivalent, so choose whichever is most convenient for you.

3. Do not submit compiled files, that is .class files for Java code and .o files for C code. Only submit the files we ask for in the assignment.

4. Do not submit links to files. The autograder does not understand it, and we will not manually grade assignments submitted this way as it is easy to change the files after the submission period ends.

6.3 Submission Guidelines

1. You are responsible for turning in assignments on time. This includes allowing for unforeseen circumstances. If you have an emergency let us know **IN ADVANCE** of the due time supplying documentation (i.e. note from the dean, doctor's note, etc). Extensions will only be granted to those who contact us in advance of the deadline and no extensions will be made after the due date.

2. You are also responsible for ensuring that what you turned in is what you meant to turn in. After submitting you should be sure to download your submission into a brand new folder and test if it works. No excuses if you submit the wrong files, what you turn in is what we grade. In addition, your assignment must be turned in via Canvas/Gradescope. Under no circumstances whatsoever we will accept any email submission of an assignment. Note: if you were granted an extension you will still turn in the assignment over Canvas/Gradescope.

3. There is a 6-hour grace period added to all assignments. You may submit your assignment without penalty up until 11:55PM, or with 25% penalty up until 5:55AM. So what you should take from this is not to start assignments on the last day and plan to submit right at 11:54AM. You alone are responsible for submitting your homework before the grace period begins or ends; neither Canvas/Gradescope, nor

your flaky internet are to blame if you are unable to submit because you banked on your computer working up until 11:54PM. The penalty for submitting during the grace period (25%) or after (no credit) is non-negotiable.

6.4 Syllabus Excerpt on Academic Misconduct

Academic misconduct is taken very seriously in this class. Quizzes, timed labs and the final examination are individual work.

Homework assignments are collaborative, In addition many if not all homework assignments will be evaluated via demo or code review. During this evaluation, you will be expected to be able to explain every aspect of your submission. Homework assignments will also be examined using computer programs to find evidence of unauthorized collaboration.

What is unauthorized collaboration? Each individual programming assignment should be coded by you. You may work with others, but each student should be turning in their own version of the assignment. Submissions that are essentially identical will receive a zero and will be sent to the Dean of Students' Office of Academic Integrity. Submissions that are copies that have been superficially modified to conceal that they are copies are also considered unauthorized collaboration.

You are expressly forbidden to supply a copy of your homework to another student via electronic means. This includes simply e-mailing it to them so they can look at it. If you supply an electronic copy of your homework to another student and they are charged with copying, you will also be charged. This includes storing your code on any site which would allow other parties to obtain your code such as but not limited to public repositories (Github), pastebin, etc. If you would like to use version control, use github.gatech.edu

6.5 Is collaboration allowed?

Collaboration is allowed on a high level, meaning that you may discuss design points and concepts relevant to the homework with your peers, share algorithms and pseudo-code, as well as help each other debug code. What you shouldn't be doing, however, is pair programming where you collaborate with each other on a single instance of the code. Furthermore, sending an electronic copy of your homework to another student for them to look at and figure out what is wrong with their code is not an acceptable way to help them, because it is frequently the case that the recipient will simply modify the code and submit it as their own. Consider instead using a screen-sharing collaboration app, such as `http://webex.gatech.edu/`, to help someone with debugging if you're not in the same room.

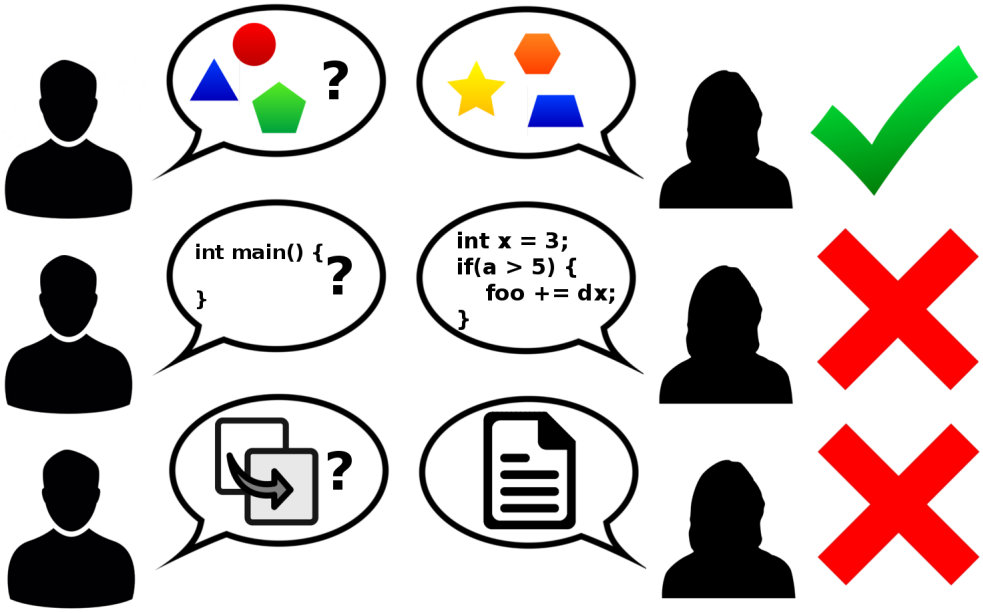


Figure 2: Collaboration rules, explained colorfully