

Lab 1. Construct an 8-Bit Instruction Set Architecture CSE 141L

1 Graphics Processor

For this first lab you will design the instruction set for a processor. You will design the hardware for the processor in subsequent labs. This will be an 8-bit processor which you will optimize to run ONE particular graphics program described below. You will design the instruction set and instruction formats, and code this program to run on your instruction set. Given the limited instruction bits, the needs of the target program should be considered carefully. The best design will probably come from an iterative process of designing the ISA, then coding the program, then redesigning the ISA. Your goal is, of course, to design an ISA that can be implemented so as to run the program as fast as possible.

You can work in groups of 1 to 3 people, but no larger. Once you form a group, you cannot change groups, so be sure you can work with your lab members. Only one lab write up is required for each group, and all group members will receive the same grade for the lab, no matter how much work each person in the group does.

Your instruction set architecture should feature fixed-length instructions 8 bits wide. Your instruction-set specification should describe in detail when you turn it in:

- what operations it supports and what their opcodes are
- how many instruction formats it supports and what they are
- number of registers, and how many; general-purpose or specialized
- the size of main memory
- addressing modes supported
- give the bit representation of each instruction type

You need to make sure that all of your opcode encodings are unique. There needs to be no ambiguity by the processor when it tries to decode it as to what operation is to be performed. From the opcode, the processor will get the format of the instruction.

In order to fit this all in 8 bits, the memory demands of this program will have to be small. For example, you will have to be clever to have a conventional main memory even as big as 256 bytes. For the program you run, you may not need all of that. You should consider how much data and instruction space you will need before you finalize your instruction format. You can assume that instructions are stored in different memory than data memory, so that your data addresses need only be big enough to hold data. You should also notice that this program probably won't need procedure calls and stack pointers, but the design is up to you. This will be an 8-bit machine for every aspect. Memory is byte-addressable, and registers and important data types are also 8 bits.

Again, your goal is to minimize the final running time of this program. You may wish to simplify the ISA design process by trying to optimize the following 2 things:

1. Minimize dynamic instruction count (i.e., the number of instructions needed to execute each program).
2. Simplify your processor hardware design.

These two goals may conflict with each other. That is, you may want to design a very specialized instruction to minimize instruction count, but you need to consider its complexity of implementation – which would negatively impact final execution time by impacting cycle time.

2 Additional Design Restrictions

Your processor must be a single cycle CPU. This means the execution of each instruction needs to complete in a single cycle. No exceptions. You must use a load instruction to read any value from memory.

In each cycle you can perform *at most one memory operation*, and the *data width of that memory operation is limited to 8-bits*. So you can perform one store or one load of 8-bits per cycle, but not both at the same time. The memory operations must be explicit load and store instructions in your ISA.

Your program must terminate by executing a halt instruction.

3 What to Turn In

The turning for Lab one will occur in 2 parts. Part A is due in one week. Part B is due in 3 weeks. Part A is intended to get you thinking about your ISA design.

For Part A you need to turn in a mini-lab report giving a complete description of your ISA, in general, and outlining each instruction's descriptions specifically. I want you to start thinking about the program that you will be writing in this ISA and come up with the instructions that you think you will need to accomplish it. Specifically, you want to consider making some specialized instructions that will ease the specific graphics manipulations that you will perform.

For Part B you will turn in (again) a complete and **FINAL** ISA description (you can change the one you produced for Part A if you have found problems with it). Additionally, you will be building an assembler to compile your program into a binary format. You will also be building a simulator/emulator to simulate the execution of this binary format. More details are on the class website. The assembler and emulator (besides being part of your grade for the 1st project) will allow you to test out your ISA to make sure it works before building your design.

1. The complete description of your ISA, in general, and each instruction's descriptions specifically.
2. An assembler program that consumes your assembly program (which you design by hand) and outputs the binary format of the program to be executed.
3. An emulator that reads in the binary format of your program and program input, and emulates the execution of your program outputting the correct output.
4. The assembly language program from your parser to implement the text formatting algorithm.
5. Answers to all the questions asked in the Lab 1 handout.

Instructions on how to turn the above in will be posted on the class website.

4 Questions to Answer

For Part B, you will turn in a lab report, and answer the following questions. In describing your architecture, keep in mind that the person grading it has much less experience with your ISA than you do. It is your responsibility to make everything clear.

1. What instruction formats are supported and what do they look like? Give an example of each instruction type in assembly language instruction, then translate it into machine code.
2. What instructions are supported and what are their opcodes?
3. How many registers are supported? Anything special about the registers?
4. What addressing modes are supported? How are addresses calculated? Give an example.
5. How large is the main memory?
6. In what way did you optimize for dynamic instruction count?
7. How did you optimize for ease of design?
8. If you optimized for anything else, what and how? (It's OK if you didn't).
9. What do you think will be the bottleneck in your design (i.e., what resource will you run out of the most quickly for bigger, more complex programs)?
10. What would you have done differently if you had 4 more bits per instruction?
11. Can you classify your machine in any of the classical ways (e.g., stack machine, accumulator, register, load-store)? If so, which? If not, give a name for your class of machine.

5 The Graphics Algorithm

For this course you will design a single cycle processor that does some basic analysis and modification of a 2-D graphics array. The algorithm will have 2 parts. In the first part you will read in a 2 dimensional array from memory, store a copy of it elsewhere in memory, and determine the number of *s in the array. Store the answer in memory location 0. In the second part, a row number will be read in from a file, and all the elements of that row in the copied array will be changed to *s.

The input file will contain the number of rows, the number of columns, then all of the data for the array (one byte for each array entry), and finally a row number (for the second part).

- The maximum number of rows or columns is 6

- In each cycle you can perform one memory operation, and the data width of that memory operation is limited to 8-bits. So you can perform one store or one load of 8-bits per cycle, but not both at the same time. The memory operations must be explicit load and store instructions in your ISA.
- No other operation (compares, etc) can occur in the same cycle as a load or a store.
- The number of rows will be stored at memory location 10, the number of columns at memory location 11
- The input array will be stored starting at location 12