CSE 141L Lab 1. 9-Bit Instruction Set Architecture

Due 11:59pm Mon., 24 Apr.

In this lab, you will design the instruction set for your own 9-bit processor, which you will optimize for three simple programs (described below). This is the crucial first of several steps in building your own custom processor, which is the objective of the course.

For this lab, you will:

design the instruction set and instruction formats for your processor;
code three assembly programs to run on your processor.

Given the tight limit on instruction bits, you need to consider the target programs and their needs carefully. The best design will come from an iterative process of designing an ISA, then coding the programs, modifying the ISA, etc.

Your instruction set architecture should feature **fixed-length instructions 9 bits wide**. Your instruction-set specification should describe:

1. What operations it supports, including a "**halt**" instruction, and what their respective opcodes are.
2. How many instruction formats it supports and what they are (in detail -- how many bits for each field, and where they're found in the instruction). Your instruction format description should be detailed enough that someone could write an assembler (a program that creates machine code from assembly code) for it.
3. The number of registers, and how many general-purpose or specialized. The internal data storage will be **8 bits wide**.
4. The size of main memory.
5. Addressing modes supported (this applies to both memory instructions and branch instructions). That is, how are addresses constructed or calculated?

Given the 9-bit limit, the memory demands of these programs will have to be small. For example, you will have to be clever to have a conventional main memory even as big as 512 bytes. You should consider how much data space you will need before you finalize your instruction format. You may assume that instructions are stored in a separate instruction

memory, so that your data addresses need be only big enough to hold data needed for all three programs. You should also assume that memory is byte addressable, and that loads and stores read and write exactly **8 bits** (one byte).

You will write three programs for your ISA and run them.

You may assume that the first starts at address 0, and the other two are found in instruction memory after the end of the preceding program (at some nonoverlapping address of your choosing). You may also elect to set up each program independently, with all three starting at address 0 -- many students find this approach easier. The specification of your branch instructions may depend somewhat on where your programs reside in memory, so you should make sure they still work if the starting address changes (e.g., if you have to rewrite one of the programs and it causes the others to also shift). This approach will allow you to put all three programs in the same instruction memory later in the quarter.

The following constraints on your design will make it a bit simpler.

1) Assume a data memory with a single address port for both reads and writes. It may, and presumably will, have separate 8-bit-wide data input and data output ports. This is where your design will receive data operands from the test bench and return results to it.

2a) You may incorporate a register file (or whatever internal cache storage you support) that can write to only one register per instruction. The sole exception to this rule is that you may have a separate single 1-bit condition register (e.g., carry out, or shift out, sign result, zero bit, etc.) that can be written at the same time as an 8-bit register, if you want.

2b) You may read from more than one register per cycle.

2c) Please restrict register file size to no more than **16 registers** of one byte each.

3) Manual loop unrolling of your code is **not** allowed.

The following *suggestions* will either improve your performance or greatly simplify your design effort.

The design of the pipelined machine is simpler if, for every instruction, you always either:

1) compute (including address arithmetic) then access memory; or
2) access memory, then compute.

Thus, every instruction in your ISA should follow the same pattern, which you choose. In

optimizing for performance, distinguish between what *must* be done in *series* vs. what *can* be done in *parallel*. An instruction that does an add followed by a subtract could require a deep pipeline or long cycle time and impact performance. An instruction that does an add and a subtract (in which neither depends on the output of the other) takes no longer than a simple add instruction. Similarly, a branch instruction whose branch condition or target depends on a memory operation will make things more difficult later on.

You will be optimizing for the following two goals:

1) Minimize execution time.
2) Simplify your processor hardware design.

You will be rewarded, in particular, for doing well on **goal 1**. Execution time is cycle count divided by clock frequency. Cycle count, for now, will depend very heavily on dynamic instruction count. That should probably be your focus at this point – in fact, we shall reward the team with the lowest dynamic instruction count at the end of class, and it will factor into the grade for lab 1. The other factors will become more clear later.
The simpler your processor design, the easier your job will be for the following labs.

Generic ISAs will let you pass the course, but will otherwise be frowned upon. We really want you to **optimize for these programs only**.

Please submit a lab report as described below. Be sure to include all relevant program listings and source data files, so that a TA or tutor can *easily* verify your work.

The report will 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 -- one objective of this course is to help you improve your technical writing and reporting skills, which will benefit you richly in your career.

For each lab, there will be a set of requirements and questions that direct the format of the writeup and make it easier to grade, but strive to create a report you can be proud of. Sometimes that may require a little repetition, e.g. describing something where you think it belongs in the report, and then again in the “question” part, so the graders won’t miss it.

Components of lab 1:

1. (10 pts) Introduction.

This should include the name of your architecture, overall philosophy, specific goals strived for and achieved.

2. (5 pts) Instruction formats.

Give all formats and an example of each.

3. (10 pts) Operations.

Give all instructions supported and their opcodes/formats.

4. (5 pts) Internal operands.

How many registers are supported? Is there anything special about the registers?

5. (5 pts) Control flow (branches).

What types of branches are supported? How are the target addresses calculated? What is the maximum branch distance supported?

6. (5 pts) Addressing modes.

What memory addressing modes are supported? How are addresses calculated? For example, if you need to access memory location 128, how would you do that?

Additionally, please explicitly answer the following questions.

7. (5 pts) How large is the main memory?

8. (10 pts) In what ways did you optimize for dynamic instruction count? Please explain.

9. (10 pts) In what ways did you optimize for short cycle time? Please explain.

10. (5 pts) Your chief competitor just announced a load-store ISA with two explicit operands (one source register same as destination), four registers (i.e., a 2-bit register specifier), and 32 instructions (5 opcode bits). Tell us why your ISA is better, and give examples to support your claim.

11. (5 pts) What would you have done differently if you had two more bits for instructions? Please explain.

12. (5 pts) Can you classify your machine in any of the classical ways (e.g., stack machine, accumulator, register, load-store)? If so, which? If not, come up with a name for your class of machine.

13. (5 pts) Give an example of an “assembly language” instruction in your machine, then translate it into machine code.

For 14-16, give assembly instructions. Make sure your assembly format is well described, and that the code is well commented. State any assumptions you make. If you also want to include machine code, the effort will not be wasted, since you will need it later. We shall not correct/grade the machine code yet.

14. **CORDIC** (40 pts) --Write a program that converts 12-bit Cartesian (x, y) coordinates into their polar (radius, angle) equivalents, also of 12-bit precision. The operands are found in memory locations 1 (most significant word of X), 2 (least significant word of X), 3 and 4 (most and least significant words of Y). The result shall be written into locations 5 (MSW of R) through 8 (LSW of Θ).

    Format for x and y = 16'b0XXX_XXXX_XXXX_0000, i.e., a 12-bit two's complement number guaranteed to be nonzero and left-justified.

    Format for R = 16'bXXXX_XXXX_XXXX_????, where the top 12 bits are unsigned magnitude binary and the bottom four are don't cares.

    Format for Theta = 16'bXXXX_XXXX_XXXX_????, where the top 12 bits may be regarded as either two's comp. or unsigned, because on a unit circle, -pi/2 = 3pi/2, etc.

(b). What is the dynamic instruction count of this program if the value of A = 1023 (location 1 has MSB's of A, location 2 has LSB's of A) and B = 32 (location 3 has MSB's of B, location 4 has LSB's of B)? Thus:

Loc. 1 = 0x3F; Loc. 2 = 0xF0; Loc. 3 = 0x02; Loc. 4 = 0x00

15. **String match** (40 pts) -- Write a program that finds, in a long string of unsigned bytes, the number of times a given 4-bit sequence occurs. For example, if the 4-bit string is 0101, then it would be found three times in the sequence 00001010101000. The string starts at position 32, continues into position 33, etc., and is 64 bytes long. The sequence you search for will be in the lower 4 bits of memory address 9. The results shall be written in location 10. Your count should *saturate* at 255, to fit an 8-bit field. (What is the maximum possible?)

(b). What is the dynamic instruction count of this program? If it varies according to the values, assume the first 10 bytes have value 00011001, the next 20 have the value 10111101, the next 9 have value 11100111, and the rest have 00000000. Assume the string is 0011.

16. **Signed Integer Division** (40 pts) -- The 16-bit two's complement dividend will occupy data memory locations 128 (MSW) and 129 (LSW). The 8-bit two's complement divisor will occupy location 130. Write the 16-bit quotient into locations 126 and 127 (LSW).

(b). What is the dynamic instruction count of this program if dividend = 16'h0180 and divisor = 8'h40?