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

*Due 11:59pm Mon. .*

In this lab you will design the instruction set for a special-purpose reduced instruction set (RISC) processor. You will design the hardware for your processor core in subsequent labs.

This processor will have 9-bit instructions (machine code) and will be optimized for three simple programs, described below. For this lab, you will design the instruction set and instruction formats and code three programs to run on your instruction set. 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, redesigning the ISA, etc.

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

- what operations it supports and what their respective opcodes are.
  For ideas, see the MIPS, ARM, and/or SPARC instruction lists
- 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.
- number of registers, and how many general-purpose or specialized. The internal data storage will be 8 bits wide.
- addressing modes supported (this applies to both memory instructions and branch instructions). That is, how are addresses constructed or calculated?

For this to fit in a 9-bit field, the memory demands of these programs will have to be small. For example, you will have to be clever to support a conventional main memory of 256 bytes (8-bit address pointer). You should consider how much data space you will need before you finalize your instruction format. Your instructions are stored in a different memory, so that your data addresses need be only big enough to hold data. Your memory is byte addressable, i.e., loads and stores read and write exactly 8 bits (one byte).

You will write and run three programs. You may assume that the first starts at address 0, and the other two are found in memory after the end of the first program (at some nonoverlapping address of your choosing). 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 a little (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 on in the quarter.

We shall impose the following constraints on your design, which will make the design a bit simpler. You should assume single-ported data memory (a maximum of one read or one write per instruction, not both). You should also assume a register file (or whatever internal storage you support) that can write to only one register per instruction. The sole exception to this rule is that you may have a 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. You may read more than one register per cycle. Please restrict register file size to no more than 16 registers. Also, manual loop unrolling of your code is not allowed.

In addition to these constraints, the following *suggestions* will either improve your performance or greatly simplify your design effort. 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 and a subtract (but neither depends on the output of the other) takes no longer than a simple add instruction. Similarly, a branch instruction where the branch condition or target depends on a memory operation will make things more difficult later on.

You will be optimizing for the following goals:

1. Minimize clock cycle count.
2. Simplify your processor hardware design.

You will be rewarded, in particular, for doing a good job with goal 2.

Generic, general-purpose ISAs (that is, those that will execute other programs just as efficiently as those shown here) will be seriously frowned upon. We really want you to optimize for these programs only.

You will turn in a lab report no more than eight pages long (excluding the program listings). The report will answer the questions posed below. 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. Introduction.
This should include the name of the architecture, overall philosophy, specific goals strived for and achieved.

2. Instruction formats.
List all formats and an example of each.

3. Operations.
List all instructions supported and their opcodes/formats.

4. Internal operands.
How many registers are supported? Is there anything special about any of the registers?

5. Control flow (branches).
What types of branches are supported? How are the target addresses calculated? What is the maximum branch distance supported?

6. Addressing modes.
What memory addressing modes are supported? How are addresses calculated? Give examples.

Additionally, please explicitly answer the following questions.

7. 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 me a name for your class of machine.

8. Give an example of an "assembly language" instruction in your machine, then translate it into machine code.

For 9-11, give assembly instructions. Make sure your assembly format is either very obvious or well described, and that the code is well commented. If you also want to include machine code, the effort will not be wasted, since you will need it later. State any assumptions you make. If you need initial nonzero values in registers or memory, you need to put them there. (Exception -- the test bench will load the incoming operands for you.)

Hint: You can save yourself a lot of debugging time in Lab 4 if you take the time now to write an interpreter for your ISA as part of Lab 1. We shall provide a template.

9. **Product** --Write a program that finds the product of three unsigned numbers, ie, A * B * C. The operands are found in memory locations 1 (A), 2 (B), and 3 (C). The result will be written into locations 4 (high bits) and 5 (low bits). There may be overflow, in which case you only store the low 16 bits. Your ISA will not have a "multiply" operation.

10. **String match** -- Write a program that finds the number of entries in an array which contain a 4-bit string. For example, if the 4-bit string is 0101, then 11010110 and 01010101 would both count (the latter would count once, even though the string occurs 3 times), while 00111011 would not. The array starts at position 32 and is 64 bytes long. The string you search for will be in the lower 4 bits of memory address 6. The result shall be written in location 7.

11. **Closest pair** -- Write a program to find the least distance among all pairs of values in an array of 20 bytes. Assume all values are signed 8-bit integers. The array of integers starts at location 128. Write the minimum distance in location 127.