

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

Due Friday, January 23, 2009

In this lab, you will design the instruction set for a processor. You will design the hardware for that processor core in subsequent labs. This will be an 8-bit processor which you will optimize for a few simple programs (described on the next page). For this lab, you will design the instruction set and instruction formats, and code three programs to run on your instruction set. Given the extreme limit on instruction bits, the target programs and their needs should be considered carefully. The best design will come from an iterative process of designing an ISA, then coding the programs, redesigning the ISA, and so on.

Your instruction set architecture should feature fixed-length instructions 8 bits wide. Your instruction-set specification should describe:

- what operations it supports and what their opcodes are. You should have a “halt” instruction in the ISA. You should also have one instruction in your ISA called “to be defined.”
- 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. You, in fact, may want to do so yourself.
- number of registers, and how many; general-purpose or specialized. The internal storage will be 16 bits wide.
- the size of main memory.
- addressing modes supported (this applies to both memory instructions and branch instructions). That is, how are addresses constructed/calculated?

In order to fit this all in 8 bits, 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 256 half-words. You should consider how much data space you will need before you finalize your instruction format. You can assume that instructions are stored in a different memory, so that your data addresses need only be big enough to hold data. You should also assume that memory is half-word addressable, and that loads and stores read and write exactly 16 bits (a half word).

You will write and run three programs. You can assume each starts at location 0, or you can assume that the first starts at 0, and the other two are found in memory after the end of the first program. However, the specification of your branch instructions may depend on which approach you take (where in instruction memory particular programs reside).

When implemented, this will be a pipelined machine, and will be evaluated based on actual performance. We will impose the following constraints on your design, which will make the design a bit easier. You will assume single-ported data memory (a maximum of one read or one write per instruction, not both). You will also assume a register file (or whatever internal storage you support) that can only write one register per instruction. The only exception to this rule is that you can have a single 1-bit condition register (e.g., carry out, or shift out, sign result) that can be written at the same time as a 16-bit register, if you want. You can, of course, read more than one register per cycle. Please restrict register file size to no more than 16 registers. Also, manual loop unrolling is not allowed.

In addition to these constraints, the following *suggestions* will either improve your performance or greatly simplify your design effort. The design of the pipelined machine is greatly simplified if you always either compute (including address arithmetic) then access memory or always access memory, then compute. By this, I mean every instruction in your ISA should follow the same pattern, whichever 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 variable shift, followed by a subtract could severely hurt your cycle time. An instruction that does an add, a variable shift, and a subtract (but none depends on the output of the others) takes no longer than a simple add instruction. In addition, a branch instruction where the branch condition or target depends on a memory operation will make things very 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 a good job with goal 1. Execution time is a combination of cycle time and cycle count. Cycle count, for now, will depend very heavily on dynamic instruction count. That should probably be your focus at this point – in fact, we will 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.

Generic ISAs (that is, ISAs 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 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.

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 try and create a report you can be proud of. Sometimes that may require a little repetition (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. Give all formats and an example of each.
3. Operations. Give all instructions supported and their opcodes/formats.
4. Internal operands. How many registers are supported? Is there anything special about 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. How large is the main memory?
8. In what ways did you optimize for dynamic instruction count?
9. In what ways did you optimize for ease of design?
10. In what ways did you optimize for short cycle time?
11. If you optimized for anything else, what and how? (It’s OK if you didn’t)
12. 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 16 instructions (4 opcode bits). Tell me why your ISA is better.
13. What do you think will be the bottleneck in your design? That is, what don’t you have that you will miss the most if you were to have to write other, longer, programs.
- 14(a). What would you have done differently if you had 2 more bits for instructions?
14(b). 2 fewer bits?
15. 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.
16. Give an example of an “assembly language” instruction in your machine, then translate it into machine code.

for 17-19, 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. We will not correct/grade the machine code. State any assumptions you make. You cannot assume anything about the values in registers or data memory, other than those specifically given, when the program starts. That means, for example, that if you need zeroes in registers or memory, you need to put them there.

17. Write a program that computes the product of two signed integers, stored in address 0 and 1, respectively. You may not use a multiply instruction. The program should run, but need not produce the correct value, if it does not fit in a halfword. You may assume both inputs are between -255 and 255 for the purpose of optimization. The result will be written to address 2.
17(b). What is the dynamic instruction count of this program if the input values are 200 and -62?
18. Write a program that finds the average number of leading zeroes in each halfword in an array. For example, if the first three values were 0000010111010110, 1111011100100110, and 000000001111011, the number of leading zeroes would be 5, 0, 8, and the average value would be 4 (rounded down to next integer). The array is 64 half-words long, and starts at address 16. As in the example, just round down to the nearest (lower) integer. Place the result in address 15.
18(b). What is the dynamic instruction count of this program? If it depends on the values in the array, assume half have 4 leading zeroes and half have 7.
19. Assume array V[], starting at address 96, consists of 65 entries, all valued between 0 and 31 (inclusive). Find the median value, and store it in location 5.
19(b). What is the dynamic instruction count of this program? If it depends on the values, assume all valid integers appear twice, except 0 which appears three times.