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

Due Friday, October 5.

In this lab, you will design the instruction set for a processor. You will design the hardware for that processor in subsequent labs. This will be an 8-bit processor which you will optimize for a few simple programs (described on the next page) – you will never have to execute any other programs on it. 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, then redesigning the ISA.

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.
• 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 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 bits required for addressing memory will have to be few – but that doesn’t mean that memory addresses are also small. 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 notice that these programs probably don’t need procedure calls and stack pointers. You should also assume that memory is byte addressable, and that loads and stores read and write exactly 8 bits.

When implemented, this will be a single-cycle machine, so realize that there is a limit to what can be done in a single cycle. For example, assume you cannot read two data memory locations (or read one and write one) in a single cycle. Nor will you be able to access instruction memory twice, so no multiple-byte instructions allowed. However, you will be able to design a register file that reads two and writes one register in a cycle (or more).

To simplify the ISA design process, you need only optimize for the following two goals:

1. Minimize dynamic instruction count (i.e., the number of instructions executed during the running of a particular program).
2. Simplify your processor hardware design.

You are welcome to also optimize for other things (e.g., cycle time, ease of pipelining), but if you do so, we will expect you to discuss that optimization intelligently, and these two goals should still take highest priority.

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 all the labs, your report will have two parts: a lab report (in this case, your ISA description) and the answers to all questions. Since, particularly in this case, a good report will already have the answers to many of the questions, it is okay to put something like “2. See section 3 of ISA description” as an answer. But for short answers, it is probably wiser to just repeat the answer.
Questions for lab 1:

1. What instruction formats are supported and what do they look like? Give an example of each.

2. What instructions are supported and what are their opcodes?

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

4. How are branches supported? What is the maximum branch distance supported?

5. What addressing modes are supported? How are addresses calculated (including instruction addresses for branches)? Give an example.

6. How large is the main memory?

7. In what ways did you optimize for dynamic instruction count?

8. In what ways did you optimize for ease of design?

9. If you optimized for anything else, what and how? (It’s OK if you didn’t)

10. Your chief competitor just announced a load-store ISA with three operands, two registers (i.e., a 1-bit register specifier), and 32 instructions (5 opcode bits). Tell me why your ISA is better.

11. 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.

12(a). What would you have done differently if you had 2 more bits for instructions?
12(b). 2 fewer bits?

13. 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.

14. Give an example of an “assembly language” instruction in your machine, then translate it into machine code.

for 15-17, give assembly instructions. Make sure your assembly format is either very obvious or well described, and 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.

15. Write a program to compute the remainder of A / B (that is, A modulo B). A is in location 0, B is in location 1.
   Write the result in location 56. Assume unsigned integers.
15(b). What is the dynamic instruction count of this program if A = 177 and B = 5?

16. Write a program that counts the number of neighboring ones in an array of 64 bytes. Neighboring ones are two bits next to each other which both have value one. For example, 00110110 has 2 neighboring ones, and 11110110 has four. Ignore neighbors across different bytes. Assume the array starts in location 24. Put the result in location 2.
16(b). What is the dynamic instruction count of this program? If it varies according to the number of neighbors, assume each byte contains exactly three.

17. Assume two arrays – an index array(I[]), starting at 0, of 16 values, and a value array (V[]) of up to 256 values. Compute the average of the 16 values of V[] indexed by I[]. That is, the average of A[I[0]], A[I[1]], … Assume unsigned integers. You may assume that the sum of the 16 values fits in 8 bits with no overflow. Round appropriately. Put the result in memory location 0.
17(b). What is the dynamic instruction count of this program?