COMP 4300 Computer Architecture

Project 1: Instruction Set Architectures

Frequently Asked Questions

1. I was having trouble installing XSpim on my Mac. On further inspection of the site that hosts Spim/XSpim I realized that they have created a newer version called QtSpim. Is this ok to use?

Answer: Yes, you are encouraged to use the new version. The new version has the same source code and everything as the previous one, but the new version is easier to use and has a better GUI. It was extremely simple to install. It's available for OSX, PC, and Linux. The new version is called.

2. Is there a method you want us to use when assigning addresses to our virtual machine's "memory" or are the addresses arbitrary? If there is a method, what method would you like us to use?

Answer: Addresses can be arbitrary. However, you will have to implement a way to translate (or map) addresses of a simulated memory space into addresses of the memory space in your simulator. You can learn more by reading the data structures defined in memory.c of SPIM

Read the “read_mem_inst” function in mem.c (line 314) for details on how to translate (or map) memory addresses.

3. Is it safe to assume that the content held in our "memory" will be of the int type? If not, what data type should we expect our content to be stored as?

Answer: It is NOT recommended to use int as a data type. Memory may be partitioned into stack, code, and data segments. You need to use the “Instruction” type to define the code segment. Please read inst.h in SPIM and learn how to create a good data structure for the “Instruction” type.

When it comes to the stack and data segments, you can use the “mem_word” type which is the int32 type in C. Please read mem.h to find how to define the “mem_word” type in C.

4. Do the source files which our simulator reads need to be written in ASCII?, as in:

push X

push Y
mul

or in the binary encoding (or Hex representation) that we create for the written portion?, as in:
11 X
11 Y
10

--If the former is the case, then can our definition of an instruction type (for our memory model), be a string (because of the fact that you do not ask us to make an assembler, I should assume this is the case?)

Answer: Your simulators read assembly source files in ASCII. In part 3 of project 1, you will have to invent separate binary encodings for the stack-based and accumulator-based machines. Note that there is no programming in part 3 of the project. Each simulator just needs to support five instructions described in part 2 of the project.

5. Concerning the instruction to use "separate binary encodings", does this mean, for example "stack ADD corresponds to 01, and accumulator ADD to 10" or "stack ADD corresponds to 01 in this encoding, and accumulator ADD corresponds to 01 in this other encoding"?

Answer: stack ADD corresponds to 01 in this encoding; accumulator ADD corresponds to 01 in another encoding.

6. Do the binary versions of the source code files need to be a special file type, or can they be the same as the ASCII versions?

Answer: You can either use binary code for the binary version or you may use ASCII to represent the binary code. Either way is fine for the simulators.

7. I understand that we will need to write a simulator to execute instructions in binary format, but I am a little confused about the file input and output for this situation. Should the binary simulator read in an ASCII file from the command line (like in part 2), translate it into our custom binary format, and then execute it on the simulator? Or should the binary simulator read in a flat binary (text) file from the command line and execute it? Sorry for waiting so long to ask this question, but I wasn't aware there was a deliverable associated with part four of the lab.

Answer: Your simulator reads an ASCII file from the command line (like in part 2). Next, your simulator translates the ASCII code into our custom binary format. Finally, simulated machines execute the binary instructions.

8. You mentioned in class that we should split the memory space into every 4 cells correct? So an instruction: push A, would take up 4 array cells. In this case, should push go in array[0] and the address of A go into array[1] and array[2] and [3] go unused? Also, should the binary representation of the word "push" get stored or the actual string "push"?

Answer: In the first phase of your project, you may have ASCII-encoded instructions in memory. That means you do not need to split the memory space into four cells. Once you have completed this phase, you must address the encoding issue - the problem of encoding instructions and data as densely as possible - so that means we need dense, binary encodings for the instruction sets of both simulated machines. See Section 4 in the project description for details on Binary Instruction Encoding.

9. Are we supposed to turn in two versions of the simulator? One that works with ASCII and another that works with binary? If we only need to turn in one, could i just turn in the one that uses binary?

Answer: You can turn in the version that works with binary encoding. Please also submit your binary code used as an input of your simulator.

10. Are we supposed to write a quadratic_eval program in C and have it imported into our machines and then have each line of that program stored in an array?

Answer: You need to change the quadratic_eval code in a way (i.e., using ISA of the stack-based and accumulator-based machines) that your simulators can execute the modified code.

11. What kind of output do you expect to see when running our simulators?

Answer: Please print out the result of application (i.e., input code of your simulators).

12. Should the assembly code written for these 2 simulators contain a .data section in order to have the variables X, A, B, and C initialized globally (like in quadratic_eval.s), or is it sufficient to to pass them in as immediate values (i.e LOADI imm or PUSHI imm) and forego a .data section entirely for this assignment?

Answer: The assembly code of each simulator must contain a *data* section.

13. What data structure do I use to simulate a stack? I'm thinking about a linked list where push is adding a node, and pop is removing a node, but is this correct?

Answer: The simplest data structure to simulate a stack is an `array` of operands.

14. Could you give a hint on how to simulate open source code file; load source code and data into memory (this is just like to initialize the array, right?) and to initialize PC?

Answer: Loading code and data into memory can be implemented through simple file I/Os. The sample C code can be found here: http://www.programiz.com/c-programming/c-file-input-output

15. Do the stack and accumulator simulators need to be able to read and translate system calls, or can we just do the quadratic equation and print a message and the result in the simulator's main function?

Answer: You stack and accumulator simulators need to read system calls (e.g., print). If the system call is print, then your simulator should print a message or result on a terminal.

16. How I am supposed to store a variable (say X) into my data_seg array. The only way I can think of so far, is to turn the variable name into a number with a function (hashing?), and use that number as the index in the data_seg array. That way, when I run into the instruction "STO X" I can call the function on the operand X and retrieve the value from the data_seg.

Answer: There are two approaches. In approach 1, you build a table that allows you to translate a variable name into a memory address. In approach 2, you hard code all the variables' memory addresses, meaning that you don't maintain variable names in your assembly code; rather, all the variables become memory addresses in the assembly code. Approach 1 is an advanced solution compared to approach 2.

17. Is the accumulator stored in memory? Or is it separate?

Answer: The accumulator doesn't belong to memory, because the accumulator is a component of the simulated processor. If you create a data structure of processor, then accumulator is an item of the processor structure.

18. Since we're supposed to implement system calls, I assume we must implement a way of storing .asciiz strings in the data section. Currently my data section is an array of mem_words, where mem_word = int32. I'm not sure how I would store a string into my array. Would I just store each character separately as its ASCII value, and have two addresses so I know the start and ending points of each string? I suppose I

could also make all the strings null-terminated so I don't need the ending address. Am I going in the right direction here?

Answer: What you proposed is an advanced solution to implement the PRINT system call. An easy implementation is to create a separate array of messages (i.e., strings); the index of each string in the array is the message ID passed as a parameter to the system PRINT call. Your simulators retrieve a string from the message array and have the string printed out.