Lab 5 - ARM Emulator Part 2

Due Oct 31 by 11:59pm **Points** 20 **Submitting** a file upload

Objective

1. Understand how the ARM stack functions.
2. Understand function calls in ARM.

Submitting Your Work

Save your C program file as ARM2.c and submit it and your report via the **Lab Exercise 5 Submission** item on the course web site. Remember to do thorough testing and devote sufficient time to your lab report.

Introduction

Your task for this lab is to write a program that extends the simulation of the ARM processor and includes stack operations.

The program will have an array that represents the general ARM registers, and a second multi-dimensional array that represents memory. The program must also have allocated stack space within the memory array.

The program will read content into the memory array from a file, execute the instructions it finds in the array one instruction at a time, and print the contents of the registers (including a program counter) and stack after each instruction. Note, when printing the stack do not print anything below the stack pointer, or above your stack base.

1. The program will implement the following instructions: ADD, ADDI, SUBI, LDUR, STUR, B, BL, BR. And at least on Conditional Branch instruction.
2. Instructions in memory will use the assembly mnemonics rather than binary op codes.
3. Instruction arguments will be register aliases (ex. X0, X1, X2, …) memory locations in the memory array (100, 104, …), or immediate values (#-3, #5, #0, …).
4. The program should use the specific address - 200, as the default address for the start instructions in memory.
5. Load and store instructions will only use indirect addressing (ie. [Xn, #nn]).
6. Branch instructions will use immediate or register addressing.

7. The program can limit the number of registers available for use in the program. At least 8 must be available, including X0, X1, X9, X10, SP (X28), LR (X30). Additional registers may be implemented.
8. Data will be signed integers.
9. The Stack:
    1. The stack must be at least 100 words deep. Make sure your stack will not grow over your program/data area.
    2. The program must have a check and a “Ran out of Stack Space/Stack Overflow” error message.
    3. Choose a high memory area for the stack.
    4. The stack should grow down.
    5. Stack memory locations should be on double word boundaries (increment/decrement by 8).
    6. Remember to initiate the stack pointer (SP) to the top of the stack.
10. Use BR XZR to halt the program.

Test Files:

Students will create an input file to test their emulator. The program must be able to read the name of the file as an argument. Students may include a default filename for use if the argument is not used.

The instructions in the file to be executed must include:

- At least one example of each instruction implemented.
- At least 12 instructions.
- Instructions to complete at least one loop.
- At least one push and pop on the stack.
- At least one function call.

The file format will be the same as the previous program.

A different file will be used to test the programs. The TA’s have a program that calculates the sum of a series of numbers from 1 to N using recursion. Your test program need not do that same function, but should test the functionality used.

Output

The output of the program will be a listing showing:

- The current instruction
- The values of any registers used in that instruction. Including the Program Counter.
- The portion of the stack which is in use – (note: this may be limited to the current stack frame).

- Additional information, such as the value of all registers, or other useful items may be shown as well.

SAMPLE OUTPUT

```
Instruction executed:   LDUR X0, XZR, #100

Registers:  X0 = 512, SP=1000, PC=204*

Stack:  1000: 98 ***

Hit enter for next instruction: >

Instruction executed:  SUBI SP, SP, #8

Registers:  SP=992, PC=208

Stack:    1000: 98

             992:         **

Hit enter for next instruction:>

Instruction executed:  STUR  X0, [SP, #0]

Registers:  X0 = 512, SP=992, PC=212

Stack:    1000: 4A

             992: 512

Hit enter for next instruction: >
```

*note: only registers affected need be displayed, or alternately, only a subset consisting of the registers used in the program – or available for use in the program need to be displayed.

**note: this is blank as nothing has been put there yet.

***note: The used portion of the stack needs to be displayed, but not unused portions.

HINTS

1.   First, make sure your previous program is functioning properly. Fix any errors and resolve any other

problems. Make sure you understand what it is doing and why.

2.   Stacks grow down.

3. Make sure your memory is large enough to hold the stack and all instructions.