ECE331 Project1

Armv8 Assembly Language Coding and DS-5

Objectives

- Understand high-level language statements and their relationship with low-level assembly code.
- Understand procedure call mechanisms and resizing of the stack during procedure calls.
- Gain familiarity with the structure of a stack frame.
- Gain familiarity with converting a C language to assembly code.
- Gain experience with the DS-5 Arm simulator.

The Problem: Split an array of numbers into two separate arrays of primes and composites.

In this lab assignment, you will perform operations involving one array with 10 integers stored in the computer memory. You will write a program in Arm assembly to split this array into two arrays. One stores prime numbers, and the other stores composites. In earlier courses, you learned how to define and use groups of integers stored in an array. For example, in the C language an integer array is defined to store values as follows.

```
int intarray[]={1, 2, 3, 4, 5, 6};
```

In this assignment you will learn how to represent this array in memory using assembler representation and use a series of MIPs instructions to perform the following tasks:

- Split the arrayA (array size = 10). Make two arrays (arrayPrime and arrayComposite) in prime and composite numbers from arrayA. arrayA has 5 prime elements and 5 composite elements. You should note that the values can be in any order. The program should work prime if the prime/composite values in the original array don't alternate. For example, consider arrayA as :

  ```
  int arrayA = {7, 16, 23, 40, 11, 39, 37, 10, 2,18 }
  ```

  The final split arrays, arrayPrime and arrayComposite are :

  ```
  int arrayPrime = {7, 2, 23, 11, 37}
  ```

  ```
  int arrayComposite = {40, 16, 39, 10, 18}
  ```

- To Debug and simulate your program using DS-5

- The starter code for this assignment is already imported into DS-5 and can be found under the 'Project Explorer' tab. Double click on the DS-5 icon on the Ubuntu desktop to launch DS-5. The root username of the Ubuntu install is 'ece331' and the root password is 'ARM-LEGv8'. (in case you need them)

- On the left pane of the browser, you will see a list of files. The one relevant for this exercise is main.c. Double click the file to see the source code.

- At the top of the .c file, you will see an ‘extern’ function prototype -- this points to your assembly code. More on this later.

- The next two functions implement the Prime algorithm. The first iterates through the input array and splits prime and composite numbers into two output arrays. Since the function arguments are arrays, these are actually pointers to the starting memory address of the arrays.

- In main(), the first few lines initialize the arrays mentioned above. The numbers are different than the example, and the output arrays are initialized to zero.

- After initialization, you will see a function call to the primeIterator() function, which in turn calls the isPrime() function. After this function returns, print statements display the contents of the arrays. isPrimeAssembly() is commented out. This is intentional, as you will uncomment this line to run your assembly instead of the C code.

- To build the code, first select project -> clean... , then project -> build project. You should see some compiler outputs indicating it was built successfully.

- The next step is to debug your code. Right click on the project name on the left pane, and select debug as -> debug configurations…

- In the next screen that pops up click on ‘prime’ under DS-5 debugger and click debug. This should bring you into the debug perspective. In the upper left corner, you can click the green triangle to debug the code. In the lower right corner under Target Console you will be able to see the output print functions. To return to the code editor, select Window -> open perspective -> C/C++

Implementation Using Arm Assembly

- In the left pane of the C/C++ perspective, there is a file called ‘isPrimeAssembly.s’. This is where you will be writing your ArmV8 instructions to implement an equivalent of the C function. In the main.c code, comment the line that calls ‘PrimeIterator()’ and uncomment the line ‘isPrimeAssembly()’.

- Navigate the editor to isPrimeAssembly.s. Here you will see stubs for the functions/procedures you had implemented in C. The first procedure is called ‘isPrimeAssembly’ and will perform the function of the primeIterator C function. You will see a second procedure label called ‘isPrime:’ which will implement the equivalent C function to see if an individual number is prime.

- There is a nop statement (no operation or dummy instruction) with a breakpoint for debugging purposes. The base addresses of a, prime, composite, and the array length are passed to the procedure using the conventions we discussed in class. This means that you can find them in X0, X1, X2, and X3 to operate on the constants stored in memory by the C array initialization. In the upper right of the debug perspective, you will see a tab called ‘registers’. Click this, then expand AArch64->Core, and you will see the core ISA registers we have been discussing in class. If you inspect the contents of these registers, you will find the memory locations of each array, and X3 holds the constant 16 (i.e. 0x10).

- Debug your assembly code using DS-5. You can set breakpoints in DS-5! Single stepping is also highly recommended.

Important note:

The text uses an Arm assembly language called LEGv8, while DS-5 uses Armv8. There are many similarities between these two assembly languages. One difference is that you cannot use the alias 'LR' for the link register, but instead you need to use the register number, X30.

There are several other critical differences and caveats, at the bottom of this page:
https://bitbucket.org/HarryBroeders/legv8/wiki/Home

Our Ubuntu VM and assignment is based on the infrastructure first established by Harry Broeders.