Discussion of Assignment 9

CSE 2312
Computer Organization and Assembly Language Programming
Vassilis Athitsos
University of Texas at Arlington

Returning Versus Printing

- Regarding the programming tasks in assignment 9, many students have asked: "do we print/return decimal, ASCII, or hexadecimal numbers"?
- If you have this question, it means that you are making a **fundamental** mistake: you are confusing "returning a value" with "printing a value".
- There are multiple ways to answer/clarify this question.
- The simplest one, is to answer with another question: what do the given C functions do?
  - You are asked to implement specific C functions, so you have questions about what your code should be doing, just try to do exactly what the C functions do.

Returning Versus Printing

- Regarding the programming tasks in assignment 9, many students have asked: "do we print/return decimal, ASCII, or hexadecimal numbers"?
- What do the given C functions do?
  - The given C functions do not do any printing.
  - Thus, your functions should not do any printing.
- It is understandable (even recommended) that you may put some code to print stuff for debugging purposes.
  - I would actually recommend that you copy and paste the print_digit and print_number functions to each of your programs, so that you can call print_number for debugging.
- However, once your code is done, you should clean it up and remove, before you submit, any code that does printing.

Returning Versus Printing

- In general, you are asked to implement functions that compute and return something.
- Once you have computed this something, you should store it on register r0.
  - This is the convention we follow for "returning a value".
- ASCII codes are only used for printing.
- When you return a number, the ASCII code of that number is irrelevant.
- It is worth repeating, returning a value has nothing to do with printing a value.

Reading Assembly Code

- Assembly code is painful to read and understand.
- However, you are expected to read any assembly code that you are given.
- How to read assembly code?
  - Start at the beginning.
  - Start mentally executing instructions, one by one.
  - On a piece of paper, write the values of registers and memory addresses that you're using.
  - For each instruction that you "execute" in your mind, update those values on your piece of paper.

Reading Assembly Code

- If you ask me a question of the sort "I do not understand how this piece of code works", I will always ask you to show me how you manually execute this code line by line.
- Not understanding the code means that there is one specific line such that:
  - You do not understand that line.
  - You understand everything before that.
- If you ask me questions where you identify that line, I will be happy to tell you what that line does.
- If you ask me questions of the sort "what does this code do?" I will simply ask you to show me how you manually execute the code.
  - Most of the times, by the time you are done with this exercise, you have answered your own questions.

Existing Assembly Examples

- Look at assembly examples that are available on the slides and the course website.
- A lot of questions can be answered by just looking at those examples.
- For example, consider the factorial function.
  - How does it handle "returning" a value?
  - How does it handle recursive calls?
- Identifying available code that does things similar to what you need to do can save you a lot of time.

The GDB Debugger

- Using the debugger is also a great tool for:
- Understanding code.
- Debugging your own code.
- There are instructions on the course website on how to use the debugger.

Infinite Loops

- Some people have reported that their program gets into an infinite loop where it keeps rerunning from the beginning.
- I have verified that even correct code can get into this behavior.
- This phenomenon is somewhat complicated to explain, but it is easy to fix.

Infinite Loops

- First of all, how to fix:

Short version:

- At the very end of your program (every program that you write), put these lines:

```
the_end:
    b the_end
```

- These lines make sure that your program, when it reaches the end, stays there forever.

Infinite Loops

Longer version:

- At the very end of your program (every program that you write), put the code on the right.
- This way, when you get to the end of the program, you see the word END printed, so you know that you reached the end of your program (as opposed to getting stuck in some infinite loop somewhere else).

```
        ldr r4,=0x101f1000
        mov r1, #13
        str r1, [r4]
        mov r1, #10
        str r1, [r4]
        mov r1, #'E'
        str r1, [r4]
        mov r1, #'N'
        str r1, [r4]
        mov r1, #'D'
        str r1, [r4]

the_end:
        b the_end
```

Infinite Loops

- The assembly programs that we write run in a very primitive environment.
- How does a program know when to stop?
- What does the CPU execute when the program stops?

Infinite Loops

- The assembly programs that we write run in a very primitive environment.
- How does a program know when to stop?
- What does the CPU execute when the program stops?
- These are issues that are typically handled by an operating system.
- In our case, the the program runs on a simulated machine with no operating system.
- When the program finishes, what is the CPU supposed to do?

Infinite Loops

- When the program finishes, what is the CPU supposed to do?
- The CPU just fetches the next instruction from memory.
- What is the next instruction?
- It is just whatever happened to reside in memory at that time.
- Thus, while you think that your program has finished executing, the program still executes meaningless instructions.
- However, at some point, the program may reach memory that you have used on your stack.
- Some of the data you have stored on the stack, when interpreted as instructions, executes a branch to the beginning of the program.

Infinite Loops

- In summary: correct code getting into an infinite loop is a problem that you may or may have not run across.
- If you have not run across it,do not worry about it.
- If you have, it may take hours trying to find the mistake where there isn't one.
- Using the suggested fixes (especially the one that prints END at the end) resolves this issue.
- Plus, using the suggested fixes ensures that if you do observe an infinite loop, the problem is with your code.
- If you mess up your stack (by adding or subtracting the wrong values, or restoring the value of lr from the wrong place) you may get all sorts of weird execution behavior.