Computer Organization
Getting Started Pi Assembly – Hello World

Prof. Charles W. Kann

Overview of Class

- **Working with shell**
- **Editing a file**
- **Writing Hello World**
  - Introduction to memory basics
  - ldr and mov instructions
- **Compiling, Linking, and Running Assembly**
- **Makefile and touch**

Bring up a Shell

- From Putty you should already have a shell
- From lxde, you will need to go to the Panel and start an lxterm from system tools
- If you are using lxde, you can also open a file manage, though we will do everything from shell.

Make a directory for this class

- **Make a directory to store programs for this class using mkdir**
  - mkdir Assembly
  - Note that capital letters sort before lower case letters. The convention is generally to make directories start with an upper case letter so they sort to the top of a list of files.
- **Change directory (cd) to the class directory**
  - cd Assembly (note tab completion of file names)

Edit the file HelloWorld.s

- Open you editor (vim, Geary, nanoetc)
- Enter the following test into the file (Note: cut and paste does not work as the cut is on the Pi and paste on your Mac or PC).

```
.text
.global main

# Print Message
main:
   SUB sp, sp, #4
        STR lr, [sp, #0]

   LDR r0, =message
   BL printf

        LDR lr, [sp, #0]
        ADD sp, sp, #4
        MOV pc, lr
.data
message:.asciz "hello world\n"
```

Assemble, Link, and run the program

- **Assemble the program – creates ".o" object file**
  - gcc helloWorld.s -o helloWorld
- **Run the HelloWorld program**
  - ./helloWorld
  - Note you must include the "./" . For security reasons the "." (current directory) is not included in the system Path (where executable files can be found). So you must tell the system the file is in the current current directory.

Registers and Memory

- **To understand assembly you must understand registers and memory.**
- **Anything that stores a value can be thought of as memory.**
- **Registers**
  - Registers are memory that is built into the CPU. They are always called registers, never memory. The term memory is reserved for memory not on the CPU.
  - There is a limited number of registers, but they are very fast.
  - An ARM programmer has access to 16 registers, but some are used for specific purposes, so practically there are less.
  - To do almost anything in ARM assembly, you have to store values in registers.
  - There is only one set of registers. Calling a subprogram or function does not give you another set of registers.

Memory

- **Memory is any component in the system that stores values anywhere except in the CPU itself.**
  - Memory can be *on the processor die* (on the same chip) as the CPU, but that memory is called cache, not a register.
  - Registers and L1 (and possibly L2) cache are SRAM (static RAM). Do not confuse this with static (data) memory. The two are unrelated.
  - Other memory is generally implemented as DRAM, Flash Memory or SD cards. Note DRAM is Dynamic RAM, do not confuse this with dynamic (heap) memory. The two are unrelated.

Addressing Memory

- Memory can be thought of as a big array where data is collected into groups of 8 bits (called a byte).
- There is an index to this *array* of memory. This *index* can reference any specific byte. This index runs from 0x00000000 to 0xffffffff (or about 0 to 4 billion). This index is called a address.
- Since the address is to a group of 8 bits (1 byte), the address scheme is call *byte addressable*.
- When you see the word *address*, you can substitute the concept of a 32-bit index to this very huge array.

Understanding the HelloWorld Program
.data section

- Data to be used in the program should be included in a section of the code labeled ".data". This is call data (or static) memory.
- message: is a label (or alias) for an address at which to store the string "hello word\n".
- .asciz is a assembler directive saying store a null terminated array of characters at this address
- Len is a label (or alias) for an address at which to store the message length. More on this on a later slide.

```
.data
Message: .asciz "hello world\n"
```

Understanding the HelloWorld Program
program entry point

- .text opens a segment of the program which contains program text
- .global is a compiler directive saying a label in this file is to be made available to the linker. The linker builds the executable file for this program.
- In this case, the linker knows to make the main symbol the place to start executing the program.

```
.text
.global main
main:
```

Accessing Registers and Memory

- The first two lines and last two lines can be thought of as the stack push and pop when entering a function
- The ldr (load data register) loads a value into a register. This value can be an address that again (recursively) points to a value of or address.

```
LDR r0, =message    @ store the of address of message in r0.
BL printf           @ call the C function printf
```

Executing system calls

- mov r7, #4 tells the system to run System Call 4, which is write to file
- System Call 4 knows the address of a string will be in r1, and the length of the string will be in r2
- swi is a supervisor call. svc is the same thing, but the more modern name.
- mov r7, #1 tells the system to exit the program

```
ldr r2, =len
ldr r1, =message
mov r7, #4
swi 0

mov r7, #1
svc 0
```

Automating building the program

- Make will be used to build programs
- Type the file EXACTLY as given into a file named “Makefile” in the Assembly directory. Note tabs matter! Where tabs are indicated, use tabs! The line starting “gcc” starts with TAB! (NOT BLANKS or SPACES)
- From the command line, type “make”. This should *make* the programs

```
helloWorld: helloWorld.s
	gcc $helloWorld.s -o $helloWorld
```

sftp program

- You will likely need to move the files to your PC/MAC for backups, setting up for submission, etc.
- To do this you will need a sftp program.
- psftp is the putty sftp program. There are myrids of others with better interfaces. I have several that I use for different classes as they all have strengths and weaknesses. Choose one for yourself.
- A free graphical option is CyberDuck

Congratulations!

- You have completed your first ARM assembly program!