CS 2110 Quiz 3 Study Guide

Krish Katariya

Last updated: **March 02, 2024**

Contents

1. Pseudo-Ops (Assembler Directives)

Note 1.1

Pseudo-ops (or assembler directives) do not refer to operations performed by assembly, but rather it is a message from the assembly language to the assembler to help the assembler during its process.

1.1. .ORIG

Definition 1.1.1

.ORIG tells the assembler where in memory to place the LC-3 program. So .ORIG x3050 says place the first LC-3 ISA instruction at location x3050

1.2. .FILL

Definition 1.2.1

.FILL tells the assembler to set aside the next location in the program and initialize it with the value of an operand. The value can either be a number or label.

1.3. .BLKW

Definition 1.3.1

.BLKW tells the assembler to set aside some number of sequential memory locations (BLocK of Words) in the program.

A common use of BLKW is to set aside a piece of memory and then have another section of code produce a number and store it at that memory.

1.4. .STRINGZ

Definition 1.4.1

.STRINGZ tells the assembler to initalize a sequence of n+1 memory locations. It takes in a string as an argument, enclosed in double quotes.

```
.STRINGZ "Hello, World!"
```

1.5. .END

Definition 1.5.1

.END tells the assembler it has reached the end of the program and need never look at everything after

2. Two Pass Process

In order to work properly, assembly has to go through the code in two separate passes, otherwise it will encounter errors with not understanding symbolic names or labels, such as `PTR`.

Theorem 2.1

The objective of the **first pass** is to identify the actual binary addresses corresponding to symbolic labels. These set of correspondences is stored in the *symbol table*. In the first pass, we construct the symbol table.

In pass two, we translate the individual assembly language instructions into their corresponding language instructions.

For example:

```
GETCHAR ADD R3, R3, #-1
        LDR R1, R1, #0
        BRnzp TEST
```

In this case, if we only had a 1 pass technique, assembly wouldn't be able to recognize what `GETCHAR` and `TEST` refer to, so it would fail. Instead on the first pass we add `GETCHAR` & `TEST` to the symbol table with their respective address.

Now on the second pass, we can go through each line and substitute a symbol for its value in the symbol table.

3. Subroutines

Definition 3.1

Subroutines are program fragments that are reusable, similar to functions in other languages.

The **call/return mechanism** allows us to execute instruction sequences only once by requiring us to include it as a subroutine in our program only ones.

THere are two instructions that use this call/return mechanism:

3.1. JSR(R)

Definition 3.1.1

JSR(R) calls the subroutine.

- It loads the PC with the starting address of the subroutine and it loads R7 with the address immediately after the address of the JSR(R) instruction.

R7 now holds the address we want to come back to after our subroutine is done, also known as the *return linkage*.

Theorem 3.1.1

JSR(R) uses two addressing modes for computing the starting address of the subroutine, PC-Relative addressing or Base Register addressing.

You can use JSR/JSRR depending on which mode you are using.

JSR uses PC offset, while JSRR uses baseR offset.

Note 3.1.1

Because subroutines destroy/overwrite the values in registers, we must save the values in the registers we are using, and then restore them after.

4. Stack

Definition 4.1

Stacks are *LIFO*, last in first out

The stack consists of a **stack pointer**, which keeps track of the top of the stack. We use R6 as the stack pointer.

4.1. Stack Buildup

Stack buildup is the first half of the full calling convention.

Caller Prepares the Subroutine

1. Push arguments to the stack in reverse order (from last arg to first)
2. JSR/JSRR to subroutine

Callee preserves values and allocates space

1. Allocate space for return value
2. Store the return address
3. Store old frame pointer(R5)
4. Set frame pointer to be the space above the old frame pointer (which holds the first local variable)
5. Allocate space for local variable
6. Store R0-R4

4.2. Stack Teardown

Callee prepares for return

1. Store return value in previously allocated space
2. Restore R0-R4 to previous value and pop from stack

do laterr