CS 264 Assembly-C Lab

Fall 2020

1 Intro

In this activity, we are going to write a function in assembly language and call it from C. To recap, when one function calls another, it places the parameters in registers `r0`, `r1`, `r2`, and `r3` and then executes the `bl func` instruction, where `func` is the name of the function to call.

2 Simple Tutorial

In this tutorial, you will write a simple assembly language function that just adds two numbers and call it from C. First, create a new directory (`mkdir tutorial`) and `cd tutorial`. Inside your new directory, you will create two files: one assembly file and one C file (we can't mix assembly and C code in the same file). Start by opening up your assembly file (call it `tutorial.s`) in nano/vim/whatever, and type in the program:

```
.text
.global add2
.type   add2, %function /* TELL THE COMPILER THAT add2 IS A FUNCTION*/
/*
 * Adds two integers, which are passed in R0 and R1. Return value passed back
 * in R0.
*/
add2:
    push {lr} /* Prologue */
    add r0,r1 /* R0 <- R0 + R1 Return value in R0 */
    pop {lr}  /* Epilogue */
    bx lr     /* Return to caller */
```

Then save `tutorial.s` and create a new file (call it `tutorialmain.c`) and put the follwing code in:

```
#include <stdio.h>

int add2(int,int); // Function prototype for add2() so C knows how to call it

int main(int argc, char **argv) {
    int a = 5, b = 20;  // Create some dummy variables with numbers
    int c = add2(a, b); // Call the add2 function

    printf("add2() calculated %d + %d = %d\n", a, b, c);
    return 0;
}
```

Once you've typed in both files, you can compile with:

```
root@zeropi:~/tutorial# gcc -g -o easypart tutorialmain.c tutorial.s
root@zeropi:~/tutorial# ./easypart
add2() calculated 5 + 20 = 25
```

3 A Little (not much) Harder One

Now you're going to call the function from your linked list homework that adds an element to a list. For this part, I would suggest creating a new directory, new `main.c`, and a new assembly files. Copy the `list_add` function from your homework into your assembly file. Make sure you have the right compiler directives above your `list_add` function (`.text`, `.global list_add`, and `.type list_add, %function`).

Now you're going to create a C file and call your `list_add` function from C. The first thing you need to do is define the layout of a data structure that will be used to hold each element. Then you can instantiate the data structure and link it into a list.

```
#include <stdio.h>

// Data structure for a linked list element, just like from our assembly homework
struct linked_list_element{
    struct linked_list_element *next; // next pointer (4 bytes)
    int payload;                      // payload (4 bytes)
}; // Don't forget to put a semicolon at the end of a struct def

struct linked_list_element *head = NULL; // Head pointer of the list

// 1. Create a function prototype for your list_add() function.

int main(int argc, char **argv) {

    // 2. Instantiate a linked_list_element struct.

    // 3. Call list_add() to add it to the list pointed to by head.

    return 0;
}
```