Machine Project Assignment

mp2: i386 C-callable Assembly Language Functions

Assigned: 10 October 2017 **Due: Part 1:** 17 October 2017
Parts 2+3: 26 October 2017

I. OBJECTIVES:

The purpose of this assignment is to gain some familiarity with writing C callable functions that do things that can't be done in C such as accessing I/O ports. Copy all files from /courses/cs341/f17/cheungr/mp2 to your mp2 subdirectory of cs341.Use the provided makefile for all builds except where instructed otherwise. As in mp1, use ulab to build all your executables and use VMs to run and debug your code.

II. DESCRIPTION:

Part 1: program to count the number of characters

1. Write a C callable assembler function that counts the number of characters in a string. The function should work for any string and character. The address of the string and character to be counted are passed as arguments according to the C function prototype:

```
int count (char *string, char c);
```

Since there are arguments, your assembly function should use a stack frame. It should return the count to the calling C program in %eax. Use 32-bit quantities for all data. chars have only 8 significant bits but they are stored in memory (and on the stack) as 32 bits in "little endian" format. After moving a 32 bit char value from memory to a register, the char value is available in the 8 lsb's, e.g. %al.

You are given a C calling program that calls count to count the number of a user entered character in a user entered string and prints the result. The C code "driver" is in countc.c. Put your assembly code in count.s. Then build it using the same makefile by invoking "make A=count".

In count.script, provide a script showing a run with a Tutor breakpoint set where the count is incremented, showing the count (in a register) each time the breakpoint is hit, just before the increment is made. In the script, show how you determined where to set the breakpoint, i.e., how you determined where the increment instruction is located in memory.

Part 2: Program to print binary characters

2. You are given a C calling program (printbinc.c). Call your file printbin.s. The C caller for this program provides a char to the assembly function printbin. It asks the user for a hex number between 0 and 0xff, converts the input from an ASCII string to a char value, passes it to printbin as an argument, and displays the returned string which should be the ASCII characters for the binary value, e.g. for entry of the hex value 0x3d, you will get the printout: "The binary format

for character = is 0011 1101" You can see the function prototype for the printbin function in the calling C code.

The function printbin should be C callable using a stack frame and call an assembly language subprogram "donibble" that is not required to be C callable. Avoiding the use of stack frames is one way assembly code can be more efficient than C compiler generated code. The function printbin needs to declare space in the .data section for storage of a string to be returned and return the address of that location in the %eax. While processing the bits of the input argument, keep a pointer to the string in an available register. printbin and donibble can store an ascii character 0x20, 0x30, or 0x31 in the string indirectly via that register and then increment the pointer in that register until the entire return string has been filled in.

The donibble function handles one half of the char value producing the four ASCII character (0/1) for the bits in one hex digit. Printbin should call donibble twice once with each nibble to be processed in half of an available register, e.g. the %al. Donibble should scan the 4 bits of the register and move an appropriate ascii code into the string for each bit. Printbin adds the space between the two nibbles. In file printbin.script, show a run of printbin on the SAPC to display 0xab, and then a run with a breakpoint set at the helper function, to prove that it is called exactly twice. You can use Tutor or remote gdb here as you wish.

Part 3: Program to copy strings

3. Write a C-callable assembler version of the library strcpy function called mystrcpy (to avoid conflicts with the library version) that copies the contents of one string to a user provided array and returns a pointer to the provided array. The function prototype is:

```
char *mystrcpy(char *s, char *ct);
```

Write your code in a source file named strcpy.s. The provided C driver (strcpyc.c) takes user entered input for the source string and checks both the pointer returned and the effect of the copy. Choose test cases that exercise different possibilities in the logic of your code, e.g. a null string. What would happen if you choose a string longer than the destination array in the C driver?

III. TURN-IN FOR GRADING:

On ulab, make one typescript file with the script command for each of the four parts of this project capturing the execution of following commands: pwd, ls -lg, cat xxx.s, and the make build for the .lnx file. The name of these files should be xxx_ulab.script. On the tutor-vserver, make one typescript file for each of the four parts of this project capturing the following commands: scp transfer of the .lnx file to the VM and mtip to run the .lnx file on tutor. The name of these files should be xxx_vm.script. When you have completed each part, use scp to transfer the script files back to your mp2 directory. Turn in a hard copy of all script files.

Leave working versions of the source files and the script files in your mp2 project directory. The grader or I may rebuild them and/or run them to test them. In the event that you are unable to

correctly complete this assignment by the due date, do not remove the work you were able to accomplish - partial credit is always better than none.