CMPSCI 201 – Fall 2006

Professor William T. Verts

Assignment #3

Initial Set Up

1. Download and unpack the ARMulator from the class web site onto the `C:` drive of your computer. Keep the folder structure the same as in the `.ZIP` file (an `ARM200` folder on the root of the `C:` drive, with folders `BIN`, `INCLUDE`, `LIB`, and `WORK` enclosed inside `ARM200`, as well as several shortcut files).

2. Download and unpack the `Lab3.s` from `LAB3.ZIP` on the class web site into your computer. It will be referenced as `C:\ARM200\WORK\Lab3.s` when placed into the correct folder. A copy of the file is shown at the end of this document.

3. Using your favorite text editor to add code to the file as described later in this document. Be careful using Notepad: the file can be saved as `Lab3.s.txt`, forcing you to remove the `.txt` extension (save the file as `"Lab3.s"` with quotes to force Notepad to save it correctly).

4. Run ARMasm and enter the file name `Lab3` (without an extension) to assemble the program. If the program was written correctly it will create an `.ali` file and an `.o` file in `C:\ARM200\WORK`). Load the `.ali` file into Notepad to see if it is OK. If not, edit the program file with Notepad to fix the errors, save the file, and reassemble it. Repeat until the `.ali` file contains no errors.

5. Run ARMlink and enter `Lab3` (no extension and no path information) to link the program. This will create a `.map` file and a file with no extension in the same folder as the `.o` file. The file with no extension contains the binary version of your program.

6. Run the emulator (WINDBG). Maximize the debugger window. Click File-Load Image... and load the assembled program into the emulator (use the Browse button to find the file). Make certain that the console window is visible. Click on View-Registers-User Mode to bring up a view of the integer registers. Click on View-Low Level Symbols. Click on View-Memory and set the Memory Address to 0x8080.

7. Click on Execute-Go to run the program. Arrange the windows to fit on the screen. Make sure you can see your entire program, the console window, symbols, memory, and the integer registers all at once with no overlaps. If the program doesn't work correctly, go back to step 3 and fix the mistakes. Don't proceed until the program works.

8. Using any appropriate program, capture and print an image of the screen. There is a program called Printkey2000 in the home distribution package which can be used for this. Otherwise, you can hit the Print Screen button to capture the screen to the clipboard, paste it into Windows Paint, and print it from there (in landscape mode, please).

9. Remove the "box" or "female" symbol from the top of the final `.ali` file and print it. Staple and turn in the two printouts (the screen shot on top followed by the `.ali` listing).

The Assignment

The exact code framework I want you to use for your program is at the end of this document. ***Do not*** add any new code to the main driver program, except for putting your name in the comment fields. Your job is to fill in the code for the subroutines and get everything working. The main program does nothing more than loop through a series of values in `R0`, and for each one it will call `Print_Hex`, `Print_Blank`, `Print_Binary`, `Print_Blank`, `Print_Unsigned`, `Print_Blank`, `Print_Signed`, and `Print_LF`,. The `Print_Hex` subroutine will call another subroutine called `Print_Nybble`. After all numbers have been printed the program will shut down. Your subroutines must be ***completely transparent***; if a routine changes the contents of any register it must restore the value of that register to the value it had when the subroutine was called. ***No locations in main memory are allowed***; all program actions happen in the registers. Registers are saved to and restored from the stack only. Here are the definitions of the subroutine bodies that you must write:

1. The `Print_Nybble` subroutine prints out the hexadecimal character (`'0'` through `'F'`) equivalent to the lowest four bits of the number in `R0`. For example, if the lower four bits of `R0`=4 then `'4'` will be printed; and if the lower four bits of `R0`=13 then `'D'` will be printed. On the ARMulator, software interrupt 0 (`SWI &0`) is defined to print the ASCII character in register `R0` onto the console window. The `Print_Nybble` routine must ignore the upper 28 bits of `R0`; it cannot assume that they contain zero bits.

2. The `Print_Hex` subroutine calls `Print_Nybble` eight times in order to print out the entire 32-bit word in `R0` in hexadecimal. This requires that `R0` be shifted by the correct amount each time to align the desired four bits into the bottom of the word for `Print_Nybble` to work properly, and print out the eight hex characters from left to right.

3. The `Print_Binary` subroutine prints out the entire contents of register `R0` in binary, as 32 `0` and `1` characters printed out from left to right.

4. The `Print_Blank` and `Print_LF` routines print out a blank (space) and a line-feed (`#10`) character, respectively. Write the `Print_Blank` subroutine to move a literal space character into `R0` instead of its equivalent ASCII number (i.e., `MOV R0,#' '` instead of `MOV R0,#32`).

5. The `Print_Unsigned` and `Print_Signed` subroutines are to print integers passed in through register `R0` on the console as decimal (base 10) values. `Print_Signed` must handle both positive and (two's-complement) negative numbers properly. Both must be able to handle an arbitrary number of digits, with complete suppression of *unnecessary* leading zeroes. Negative numbers must be printed with a leading minus sign, while positive numbers are to be printed with a leading plus sign. Zero has no leading character (i.e., a plus sign is not required for zero). To support these subroutines I provide an existing subroutine called `UDiv10`, which divides the unsigned integer value in `R0` by 10 and puts the quotient in `R0` and the remainder in `R1`. As with many "provided" routines in real life, this routine is not completely transparent; it trashes the contents of `R2` (and you aren't allowed to fix it). Thus, your routines must take pains to accommodate the deficiencies of `UDiv10`.

Grading

Part of the grade for this assignment will be based on how neatly your code is written, commented, and whether you have made each subroutine properly transparent, as well as whether or not the program runs correctly. You are to turn in a printout for your `.ALI` listing file, as well as a screen shot in **landscape mode** of your program execution (showing the top of your code with your name and the execution bar on the `SWI &11` instruction, the user registers, and the console window containing the result of running the program). Staple the assignments together with the screen shot on top.

The main program runs through arguments that step by 8s from -120 up through +120. Each number is printed on its own line in the console window. Registers `R1` through `R11` are "preloaded" with special values so we can see if your routines preserve register values correctly.

When working, print out the `.ALI` assembly listing (in landscape mode) and a screenshot (also in landscape mode) containing the source code after execution with **your name visible**, the user registers, and the console window showing the resulting printed numbers. Staple the screenshot on top of the listing. Here are the point penalties for this 20-point assignment (no assignment will score less than zero):

1. -2 (each) for cosmetic errors:
   * Printouts not stapled together,
   * Screenshot not on top,
   * Screenshot not in landscape mode,
   * Listing not in landscape mode.
2. -20 for name not visible on `.ALI` listing (did you write the code?)
3. -10 for not printing values -120 to 120 by 8s (does it work at all?)
4. -5 for non-transparent subroutines (are registers saved and restored?)
5. -5 for not showing the user registers (can we can check for transparency?)
6. -5 for modifying any existing code (did you change the assignment?)
7. -5 for using explicit memory locations (did you use the stack?)
8. -5 for incorrect digit order or not suppressing leading zeroes (number formatting)
9. -5 for neatness and code readability (did you indent and comment?)

The Lab3.s Code

```
;-----------------------------------------------------
; {Insert your name and ID here}
;-----------------------------------------------------

            AREA  PROGRAM3, CODE
            ENTRY

Start       MOV   R1,#129
            MOV   R2,#130
            MOV   R3,#131
            MOV   R4,#132
            MOV   R5,#133
            MOV   R6,#134
            MOV   R7,#135
            MOV   R8,#136
            MOV   R9,#137
            MOV   R10,#138
            MOV   R11,#139

            MOV   R0,#-120
Main_Loop   BL    Print_Hex
            BL    Print_Blank
            BL    Print_Binary
            BL    Print_Blank
            BL    Print_Unsigned
            BL    Print_Blank
            BL    Print_Signed
            BL    Print_LF
            ADD   R0,R0,#8
            CMP   R0,#120
            BLE   Main_Loop
            SWI   &11

;-----------------------------------------------------
; {Insert your name and ID here}
;-----------------------------------------------------

Print_Nybble      …

;-----------------------------------------------------

Print_Hex         …

;-----------------------------------------------------

Print_Binary      …

;-----------------------------------------------------

Print_Signed      …

;-----------------------------------------------------

Print_Unsigned    …

;-----------------------------------------------------
```

```
Print_Blank       …

;-----------------------------------------------------

Print_LF          …

;-----------------------------------------------------
;           R1 := R0 mod 10, R0 := R0 div 10
;           R2 trashed
;-----------------------------------------------------

UDiv10      SUB   R1,R0,#10
            SUB   R0,R0,R0,LSR #2
            ADD   R0,R0,R0,LSR #4
            ADD   R0,R0,R0,LSR #8
            ADD   R0,R0,R0,LSR #16
            MOV   R0,R0,LSR #3
            ADD   R2,R0,R0,LSL #2
            SUBS  R1,R1,R2,LSL #1
            ADDPL R0,R0,#1
            ADDMI R1,R1,#10
            MOV   PC,LR

;-----------------------------------------------------

            END
```