Lab 4: Arithmetic and Logic Instructions

EE 5385

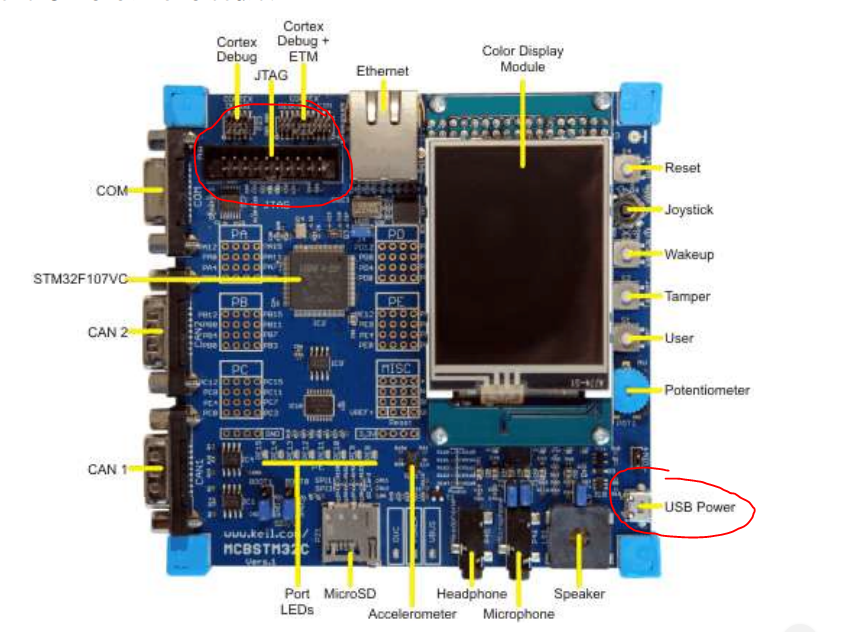
Lab Report

3/9/18

Victor Yuan

Turn on and setup (same as previous two labs)

First, I setup the board like the first lab by connecting an USB cable with an ULINK-ME adapter. Then I connect this adaptor to the JTAG pins in the figure below:



Then I connected the USB to the USB power port on the bottom right of the figure. Connect both ends of the USB cables to the computer one two power the board and one to program the board.

Then I used the program Keil uVision, first I created a new project from scratch. I selected the microcontroller vendor and device in the device database but did not use the system’s default startup code since we were provided with a custom startup file. A created a file and named it logic.s and added this file to the project along with the startup file given to us through this link “http://goo.gl/BB3mnx”.

This lab we are tasked with writing a program that must satisfy a few conditions. First it must search through memory starting at location of “TEST\_DATA” looking for ASCII characters. Each ASCII character is represented by a 8-bit Hex value. The specific letters we are looking for are H E L P and they correspond with the following hex numbers respectively 0x48 0x45 0x4c 0x50. However, if the search runs through the entirety of the TEST\_DATA which is terminated with a 0 then whenever the 0 is found the program will stop the search for any additional letters.

Code:

AREA asm\_code, CODE, ALIGN=2

ENTRY

mov r0, #0 ; register used specifically for the offset to first H

mov r8, #0 ; set starting value to be 0

ldr r1, =TEST\_DATA ; loads location in memory where test values are

mov r4, #0x48

mov r5, #0x45

mov r6, #0x4c

mov r7, #0x50 ; hex values for the ascii characters HELP

findh

ldrb r2, [r1], #1 ; loads one byte of information and post index

; this will load exactly 8 bits of data, which is exactly one

; ASCII letter

add r8, r8, #1 ; counter for the offset

cmp r2, 0 ; first determine if the string has ended

; 0 means that the string has terminated

beq no\_H\_found0 ; If equal zero has been found and program skip to next

; letter since H was not found

; If 0 has not been found then the following code for finding hex value matching H is ran

cmp r2, r4 ; compares the value just loaded with 0x48 to the hex value

; of the letter H

moveq r4, r8 ; if r2 and r4 are equal then H has been found therefore

; I no longer need the ASCII code and I can overwrite

beq foundh ; of H has been found the beq will execute leading to the

; wrap up instructions

bne findh ; if not then the whole process its repeated until H or 0 has

; been found

no\_H\_found0

mov r4, #0 ; used later on in the program

sub r1, r1, r8 ; program realizes there is no H in memoy when 0 is found

mov r8, #0 ; resets the counter and the memory address

b finde ; continues to look for the next letter omitting H

foundh

mov r0, r4 ; moves the offset to r0 as required by the instructions

; if 0 was found the whole section of foundh

; would not be seen

finde ; loop to find e

ldrb r2, [r1], #1 ; assumption is it continues from where H was found

add r8, r8, #1 ; increases the offset from the start of the string to the

; location of E

cmp r2, #0 ; looks for terminal 0 to decide if it should skip to next

; letter or not

beq no\_E\_found0 ; program skips to next letter if terminal 0 is found

cmp r2, r5 ; compares the loaded letter to 0x45

moveq r5, r8 ; loads the total offset from start of the string to r5, since

; r8 isn’t reset unless no H is found

beq founde ; branches like before if a match is found

bne finde

no\_E\_found0 ; resets same as before

mov r5, #0 ; used later on for order tests zero represents no E found

sub r1, r1, r8

mov r0, #0

b findl

founde

; no additional instruction needed here

findl ; loop to find l

ldrb r2, [r1], #1

add r8, r8, #1

cmp r2, #0

beq no\_L ; same format as before

cmp r2, r6 ; same logic for finding L

moveq r6, r8 ; once found move to r6

bne findl ; loops to look for L

beq foundlcheck1

no\_L

mov r6, #0 ; 0 here represents no L was found used later on

sub r1, r1, r8

mov r8, #0

b findp

foundlcheck1 ; this check is needed in cases such as LHaaaLp

; H is present but E is not so it would cause the

; program to restart from the beginning and to look for L

; but it would see the first instance of L

cmp r6, r4 ; comparing the offset for H determines if placement of L is

blt findl ; valid or not

cmp r6, r5 ; after checking H it checks E as well the offset of L must

blt findl ; greater then the offset of both H and E

foundh ; if no L is found after the first instance no\_L will activate  
 ; otherwise the code will just fall through cleanly

findp ; loop to find p

ldrb r2, [r1], #1

add r8, r8, #1

cmp r2, #0

beq no\_H ; same algorithm as before

cmp r2, r7 ; checks for P

moveq r7, r8

bne findp

beq foundpcheck1

no\_P

b STOP ; no more letters to find so B stop works

foundpcheck1

cmp r7, r4

blt findp

cmp r7, r5

blt findp

cmp r7, r6 ; same checks as for the letter L except it has an additional check

; for the letter L

blt findp

STOP b STOP ;infinite loop processing complete

;AREA asm\_data, DATA

TEST\_DATA

;DCB "psdfnOLHHNOHDP[[HLkhjxvcphelHELLpppHELP",0

DCB "EFLP",0

END

The most noticeable issue I had turn this lab was testing the program for omission of letters that’s why I added the checks. If I tested the string LHLP After E was skipped by the program it would restart the search from the start and see the first L and think that was after H, by implementing an system of checks to make sure the offset of L was not less then the offset of the letters before it I was able to mitigate this issue. Other minor’s issues included understanding the instruction set of the lab, at first, I didn’t quite understand what the instructions wanted me to do with my program.

From this lab I learned that DCB assigns each index a byte of data and I only need to increase the index by one to access the next set of data.

Additional questions:

What is a DCB? - DCB allocates an area within memory where byte accessible data components could be stored. Meaning each location is aligned by 8 bits, 1 byte and increasing the index by 1 will access the next 8 bits, 1 byte.

What is the difference between DCD and DCB?

DCD stores one or more word(s) while DCB stores 1-2 byte(s).

What is the purpose of the 0 at the end of the string?

The 0 represents the end of the string DCD and DCB are both limited memory allocated instructions. The 0 null terminates the string and indicates where the string ends.

