Assignment # 4: Problem Set 2, Problems 2 & 3

Brian Arnberg

2/1/2013

Assignment

For the following problems, write an ARM assembly language program, and in the Keil MDK-ARM IDE, create a project, enter the program, and then execute and debug it in the Keil MDK-ARM debugger. You may run the program either in the simulator or in RAM on the STM32F4-DISCOVERY board. All program variables are to be 32-bit integers. You may choose your own test data values.

Problem 2

Implement the following C code, to exercise program control statements. Place mm, nn, jj, and cc in the code area, with initial values defined by DCD directives, and place kk and xx in the data area. Circle the values of kk and xx in the final debug window. Execute the program twice, once for a TRUE condition, and once for a FALSE condition.

```
if ((mm-nn) <15) {
        kk = jj - 5;
        xx = 0;
} else {
        kk = cc +18;
        xx = 1;
}
```

Program 2 is named `PS2-2.s` and is listed at the end of this document.

Problem 3

Implement the following C code, to exercise memory addressing modes to handle arrays. Place arrays aa and bb in the code area, with initial values. Place variable i and array zz in the data area. Circle the final values of i and zz in the debug window.

```
for(I=0; i<15; i++)
        zz[i] = aa[i] - bb[i] +5;
```

Program 3 is named `PS2-3.s` and is listed at the end of this document.

Debugging

The ARM assembly language programs were written inside a Linux environment, but were debugged on school computers with the Keil MDK-ARM debugger tool. The target addresses for ROM and RAM were set according to the set-up guide (0x20000000 and 0x20003000, respectively). Figures 1 through 3 show the "Memory" section of the debugger, with final values circled. Figure 1 shows the memory after `PS2-2.s` was run for the TRUE condition, and Figure 2 shows the memory after it was run for the FALSE condition. The memory values circled in both Figures indicate that the program runs as expected. Figure 3 shows the memory after `PS2-3.s` is executed. The memory here also indicates that the program runs as expected. Therefore, by looking at the results, or memory output, of both programs, one can see that they both function properly.

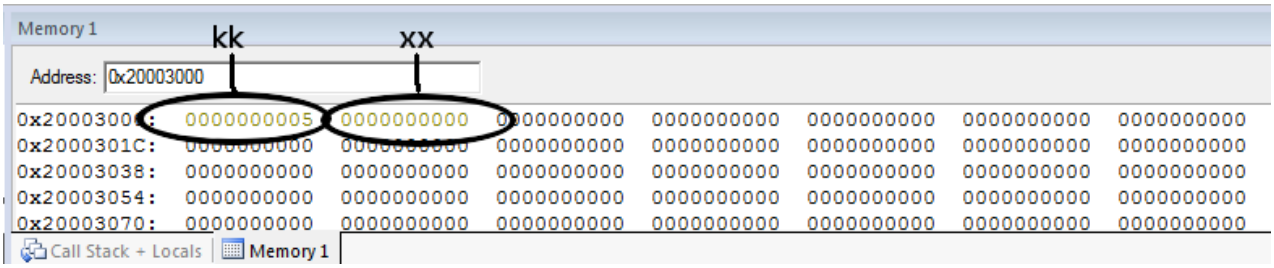

Figure 1: The program results for `PS2-2.s` after the program is run for a TRUE condition. In this case, mm is 30, nn is 20, jj = 10, and cc = 2. Under these conditions, kk is expected to be 5, and xx is expected to be 0. The memory map here indicates that these are, in fact, the resulting values. Therefore, `PS2-2.s` runs correctly for a TRUE condition.

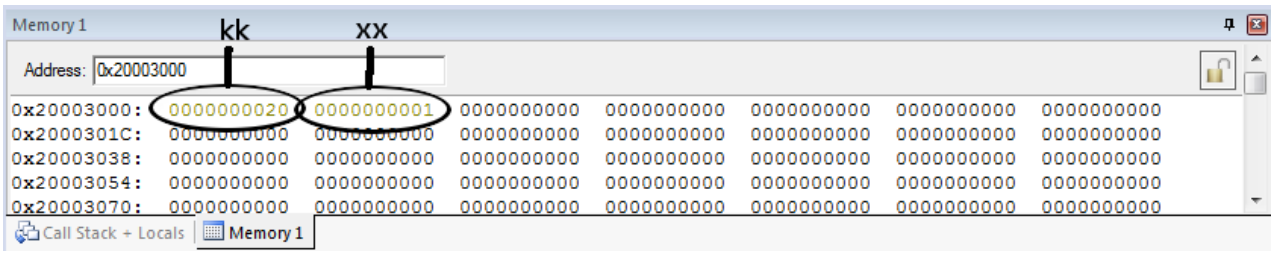

Figure 2: The program results for `PS2-2.s` after the program is run for a FALSE condition. In this case, mm is 30, nn is 10, jj = 10, and cc = 2. Under these conditions, kk is expected to be 20, and xx is expected to be 1. The memory map here indicates that these are, in fact, the resulting values. Therefore, `PS2-2.s` runs correctly for a FALSE condition.

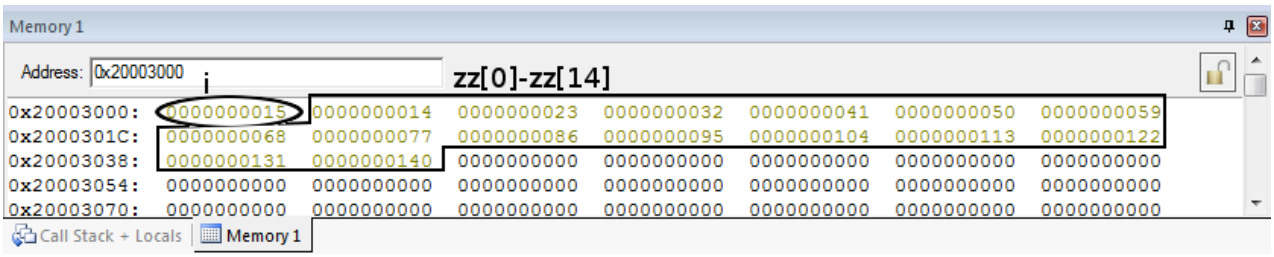

Figure 3: The program results for `PS2-3.s` after it is executed. The final expected value of i is 15, which is circled. The values for the array, zz, are also circled, and match the expected values, given the arrays used for aa and bb. Because these are the expected values, and because the expected values occur in their expected locations, it can be concluded that `PS2-3.s` functions properly.

Source Programs

PS2–2.s

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;      Brian Arnberg - ELEC6260            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;      Problem Set 2, problem 2            ;;
;; Compute the following:                   ;;
;;       if ((mm - nn) < 15) {              ;;
;;               kk = jj - 5;               ;;
;;               xx = 0;                    ;;
;;       } else {                           ;;
;;               kk = cc + 18;              ;;
;;               xx = 1;                    ;;
;;       }                                  ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Test both cases (TRUE vs FALSE)          ;;
;;      cc = 2, jj = 10                     ;;
;;  TRUE                                    ;;
;;     mm = 30, nn = 20                     ;;
;;      kk = 5, xx = 0                      ;;
;;  FALSE                                   ;;
;;     mm = 30, nn = 10                     ;;
;;      kk = 20, xx = 1                     ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        ; Define the CODE block
        AREA    RESET, CODE
        ; Begin
        ENTRY               ; Define ENTRY location
        ; Test the condition
        LDR r4, =mm0        ; Get Address for mm0
        LDR r0, [r4]        ; Store value of mm0 to r0
        LDR r4, =nn0        ; Get Address for nn0
        LDR r1, [r4]        ; Store value of nn0 to r1
        SUB r0, r0, r1      ; Compute mm-nn, store to r0
        MOV r1, #15         ; Put #15 in r1
        CMP r0, r1          ; Compare (mm-nn) < 15
        BLT true            ; if true, branch to true
false   ; False Block
        LDR r4, =cc0        ; Get Address for cc0
        LDR r0, [r4]        ; Store to r0
        MOV r1, #18         ; Set r1 to #18
        ADD r0, r0, r1      ; r0 = cc + 18
        LDR r4, =kk0        ; Get address of kk
        STR r0, [r4]        ; Store r0 to kk0
        MOV r1, #1          ; Put #1 in r1
        LDR r4, =xx0        ; Get address of xx0
        STR r1, [r4]        ; Store r1 to xx0
        B m                 ; Branch to m (skip over true block)
true    ; True Block
        LDR r4, =jj0        ; Get address for jj0
        LDR r0, [r4]        ; Store value to r0
        MOV r1, #5          ; Put #5 in r1
        SUB r0, r0, r1      ; r0 = jj - #5
        LDR r4, =kk0        ; Get address of kk
        STR r0, [r4]        ; Store r0 to kk0
        MOV r1, #0          ; Put #0 in r1
        LDR r4, =xx0        ; Get address of xx0
        STR r1, [r4]        ; Store r1 to xx0
m       B m                 ; Branch to m, never end
        ; Define the value of the 4 input variables
mm0     DCD 0x1E            ; mm = 30
nn0     DCD 0x14            ; nn = 10
jj0     DCD 0x0A            ; jj = 10
cc0     DCD 0x02            ; cc = 2

        ; Define the DATA block
        AREA    data1, DATA
        ; Set space for variables
kk0     SPACE 4             ; Set 4 bytes aside for kk
xx0     SPACE 4             ; Set 4 bytes aside for xx

        END                 ; End the program
```

PS2–3.s

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;      Brian Arnberg - ELEC6260            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;      Problem Set 2, problem 2            ;;
;; Implement the following:                 ;;
;;    for(i=0; i<15; i++)                   ;;
;;        zz[i] = aa[i] - bb[i] + 5;        ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;    aa = 10*i                             ;;
;;    bb = 1*i                              ;;
;;                                          ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        ; Define the CODE block
        AREA    RESET, CODE
        ; Begin
        ENTRY               ; Define ENTRY location
        ; Initialize loop
        MOV r0, #0          ; r0 = I; initialize to 0
        MOV r1, #15         ; r1 = 15 (use for end condition)
        MOV r7, #5          ; move #5 into r7, will use each loop
        LDR r2, =zz0        ; r2 will point to zz[i]
        LDR r3, =aa0        ; r3 points to aa[i]
        LDR r5, =bb0        ; r5 points to bb[i]
        LDR r9, =ii0        ; r9 points to ii
        ; Do the loop
loop
        LDR r4, [r3], #4; load aa[i] into r4, then point to next value
        LDR r6, [r5], #4; load bb[i] into r6, then point to next value
        SUB r4, r4, r6      ; r4 = aa[i] - bb[i]
        ADD r4, r4, r7      ; r4 = r4 + #5
        STR r4, [r2], #4; store the result to zz[i], then point to next value
        ADD r0, r0, #1      ; increment i
        STR r0, [r9]        ; store current value of ii, the point to next value
        CMP r0, r1          ; Compare (r0 - 15)
        BLT loop            ; if r0 < 15, branch to loop
halt    B halt              ; branch to self
        ; Define the value of the 2 input variables
aa0     DCD  10,20,30,40,50,60,70,80,90,100,110,120,130,140,150
bb0     DCD  1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
        ; Define the DATA block
        AREA    data1, DATA
        ; Set space for variables
ii0     SPACE 4             ; Set 4 bytes aside for the i counter
zz0     SPACE 60            ; Set 60 bytes aside for zz

        END                 ; End the program
```