Assignment # 5: Problem Set 3

Brian Arnberg

2/6/2013

Problem 1

Practice writing and using subroutines by writing a subroutine that calculates
Z = (a0 * x0) - (a1 * x1).

The values of the four variables a0, a1, x0, x1 are to be passed to the subroutine in registers. The address of variable Z is to be passed to the subroutine in a register.

The "main" program is to call the subroutine two times, once for each of the following sets of data. You may not use "immediate" data all values are to be read from memory and results are to be stored in memory.

x0 and x1 are to be 32-bit integers. a0, a1, and z are to be stored using Q24.8 format.

First call: x0=200, x1=100, a0=5.25, a1=6.75. Store the answer at variable z1.

Second call: x0=300, x1=200, a0=3.5, a1=4.125. Store the answer at variable z2.

For convenience, define the arguments to be passed to the subroutine at the end of your code section, and the variables to be written in a data section. There should be eight "variables" in the code section two sets of four arguments. There should be two variables in the data section.

You can test this with the simulator, but version to be submitted must be executed in RAM on the Discovery board. Submit a printout of your source program, and a screen image of the debug window, with the results highlighted (circled) in a memory window.

Debugging Problem 1

The Assembly Language program for Problem 1 is named `PS3-1.s`, and can be found in the Source Program Section at the end of this document. For the first run, the expected value of z1 is 375_{10}. In Q24.8 format, this is 0x00017700. The value was expected to be stored at Memory Address 0x20003014. For the second run, the expected value of z2 is 225_{10}. In Q24.8 format, this is 0x0000E100. The value was expected to be stored at Memory Address 0x20003018.

After the program was written and simulated, it was executed in RAM on the Discovery board. Figures 1 and 2 show the debugging window after the program was executed in RAM on the board, with the values circled. Both figures indicate that the program for Problem 1 executed correctly because the results are correct.

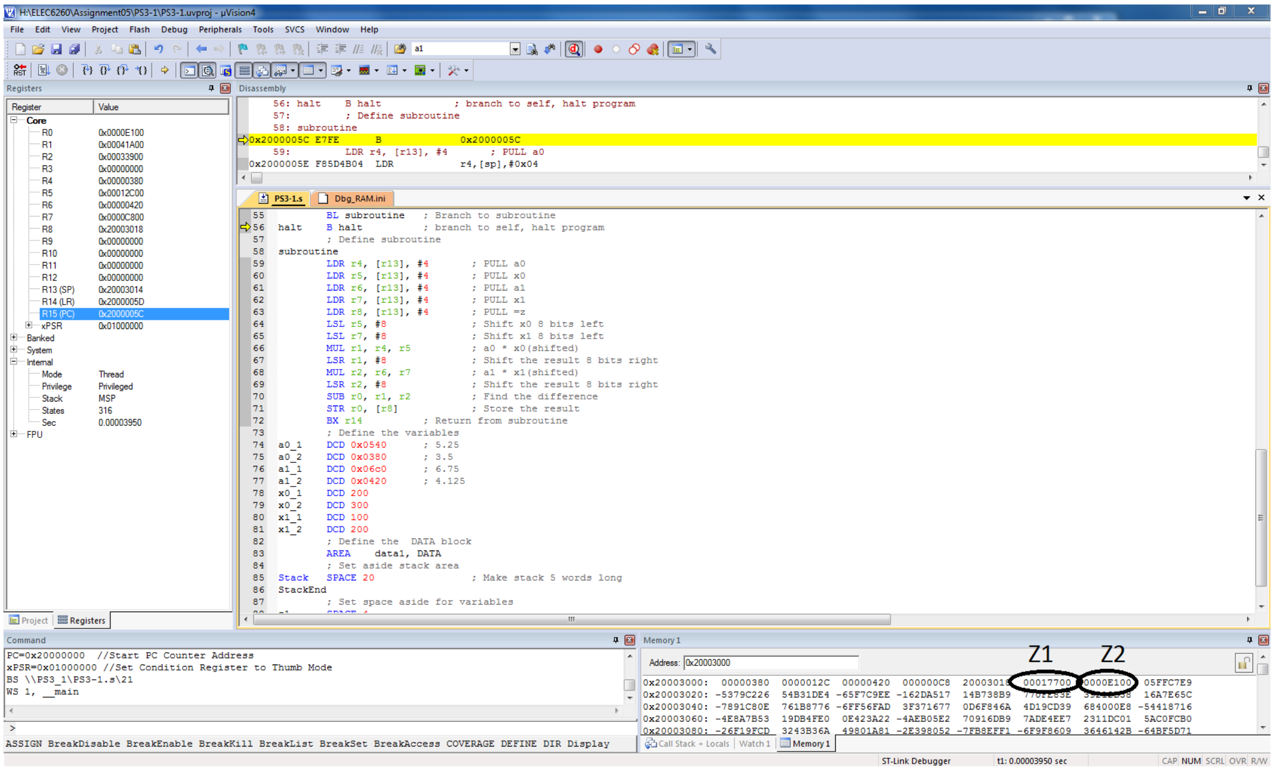

Figure 1: A full view of the debugging window for Problem 1. The values for z1 and z2 are circled. z1 is 0x00017700 and z2 is 0x0000E100. These values are correct, so one can see that the program executed correctly.

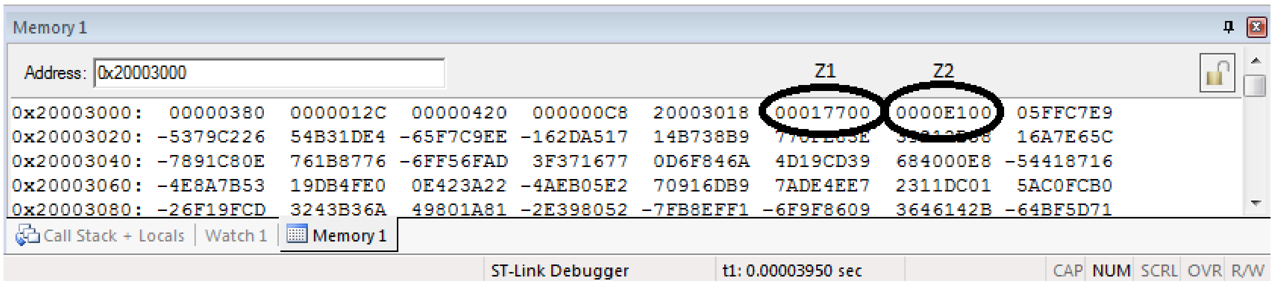

Figure 2: A view of the memory window for Problem 1 after the program was executed. As stated in Figure 1, the values for z1 and z2 are 0x00017700 and 0x0000E100, respectively. These are the expected values, and the occur at the expected locations, so one can conclude that the program written for Problem 1 executed correctly.

Problem 2

Given below is a C program that includes several functions calls.

(a) Write the equivalent routines in ARM assembly language and test the program in the Keil debugger. Submit two screen captures of the debug window, showing the final result returned for each of the two functions called by the main program. The main program and the three "functions" should be implemented as separate routines/subroutines as listed (i.e. do not "merge" them to optimize the program.)

(b) Enter and compile the C program, as given, and compare the "listing file" to your hand-written assembly language program. Alternatively, you may view the disassembled code in the debugger window to see the C and equivalent assembly language instructions. Turn off any optimizations that would normally be performed by the C compiler.

```c
int k;        //global variable

int f1(int x1, int x2) {
    return x1 + x2;
}

int f2(int x1) {
    return x1 + 1;
}

void f3(int r) {
    int j;
    for (j = 0; j < 2; j++)
            k = k + f1(r + j, 5);
}

void main () {
    int a;
    k = 0;
    f3(3);
    a = f2(2);
}
```

Problem 2, part (a)

The ARM assembly language program that emulates the above C program was written and named `PS3-2.s`. The program can be found in the Source Programs section at the end of this document. The program was tested in the Keil debugger. Figure 3 displays the memory on the board after `f3` was called, and Figure 4 displays the memory on the board after `f2` was called. Both figures display the expected values for the final result returned for each of the two functions called by the main program, so it can be observed that the hand-written program works correctly.

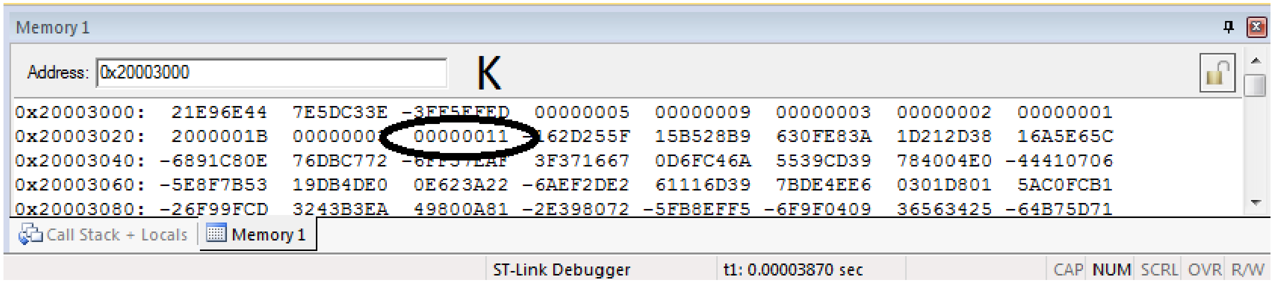

Figure 3: A view of the memory window after the main function in Problem 2 calls `f3`. The circled value is the value for k, 0x00000011, which corresponds to the expected result, 17_{10}. This means that the program executed `f3` and `f1` correctly.

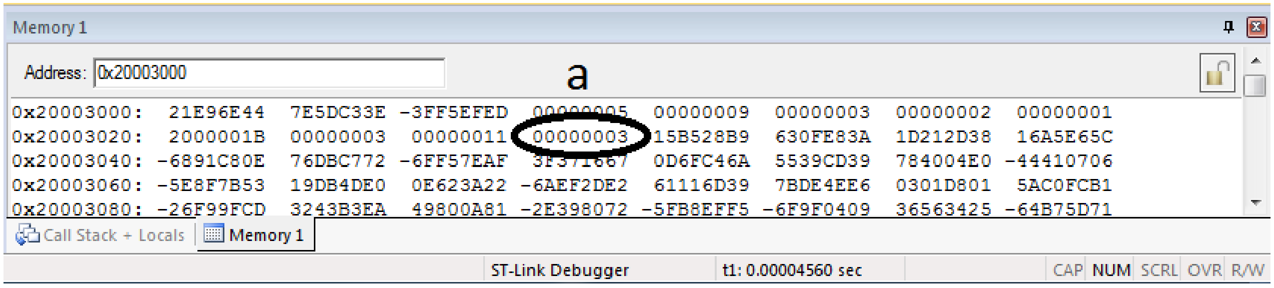

Figure 4: A view of the memory window after the main function in Problem 2 calls `f2`. The circle value is the value for a, 0x00000003, which corresponds to the expected result, 3. This means that the program executed `f2` correctly.

Problem 2, part (b)

The C program listed above was entered and compiled, as given. All compiler optimizations were turned off. Then, it was debugged on the Discovery board so that the disassembled code could be used to compare the compiler output to my hand-written assembly language program. A section of the disassembled code was copied into `PS3-2b.txt`, which can be found in the Source Programs section at the end of this document.

The hand-written program was greater than 60 lines of assembly, while the disassembled code was just over 30 lines of assembly. This means that the C program, once disassembled, was approximately twice as fast as the hand-written program. The line length difference can be attributed, at least in part, to the way the C program treated certain register values as constants. Additionally, the hand-written code interacted with the stack value by value (`STR r1, [r13, #-4]!`), while the C program disassembled to use `PUSH` and `POP`. Also, the disassembled code was simply more elegant, even without any optimizations.

Conclusions

Both ARM assembly language programs written were functionally correct. Each program was executed on the Discovery board, and each program returned expected values. The C program, when disassembled, was more efficient than the hand-written program.

Source Programs

PS3–1.s

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;      Brian Arnberg - ELEC6260             ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;      Problem Set 3, Problem 1             ;;
;; Compute the following:                    ;;
;;  Z = (a0*x0) - (a1*x1)                    ;;
;;    Using a subroutine.                    ;;
;; The 'main' will call the subroutine       ;;
;;  twice, each time passing all 5           ;;
;;  variables with registers.                ;;
;; Z will be passed as an ADDRESS            ;;
;; x0,x1 are 32bit integers                  ;;
;; a0,a1,Z using Q24.8 format                ;;
;;                                           ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        ; Define the CODE block
        AREA    RESET, CODE, READONLY
        ; Begin
        ENTRY
__main  LDR r13, =StackEnd
        LDR r0, =a0_1           ; get a0_1
        LDR r4, [r0]            ;  r4 = a0_1
        LDR r0, =x0_1           ; get x0_1
        LDR r5, [r0]            ;  r5 = x0_1
        LDR r0, =a1_1           ; get a1_1
        LDR r6, [r0]            ;  r6 = a1_1
        LDR r0, =x1_1           ; get x1_1
        LDR r7, [r0]            ;  r7 = x1_1
        LDR r8, =z1             ; get address for z1
        ; Load variables into stack
        STR r8, [r13, #-4]!     ; PUSH =z1 to stack
        STR r7, [r13, #-4]!     ; PUSH x1 to stack
        STR r6, [r13, #-4]!     ; PUSH a1 to stack
        STR r5, [r13, #-4]!     ; PUSH x0 to stack
        STR r4, [r13, #-4]!     ; PUSH a0 to stack
        ; Branch to sub-routine (for first run)
        BL subroutine    ; Branch to subroutine
        ; Return from subroutine, get new values
        LDR r0, =a0_2           ; get a0_2
        LDR r4, [r0]            ;  r4 = a0_2
        LDR r0, =x0_2           ; get x0_2
        LDR r5, [r0]            ;  r5 = x0_2
        LDR r0, =a1_2           ; get a1_2
        LDR r6, [r0]            ;  r6 = a1_2
        LDR r0, =x1_2           ; get x0_2
        LDR r7, [r0]            ;  r7 = x1_2
        LDR r8, =z2             ; get address for z1
        ; Load variables into stack
        STR r8, [r13, #-4]!     ; PUSH =z1 to stack
        STR r7, [r13, #-4]!     ; PUSH x1 to stack
        STR r6, [r13, #-4]!     ; PUSH a1 to stack
        STR r5, [r13, #-4]!     ; PUSH x0 to stack
        STR r4, [r13, #-4]!     ; PUSH a0 to stack
        BL subroutine    ; Branch to subroutine
halt    B halt           ; branch to self, halt program
        ; Define subroutine
subroutine
        LDR r4, [r13], #4       ; PULL a0
        LDR r5, [r13], #4       ; PULL x0
        LDR r6, [r13], #4       ; PULL a1
        LDR r7, [r13], #4       ; PULL x1
        LDR r8, [r13], #4       ; PULL =z
        LSL r5, #8              ; Shift x0 8 bits left
        LSL r7, #8              ; Shift x1 8 bits left
        MUL r1, r4, r5          ; a0 * x0(shifted)
        LSR r1, #8              ; Shift the result 8 bits right
        MUL r2, r6, r7          ; a1 * x1(shifted)
        LSR r2, #8              ; Shift the result 8 bits right
        SUB r0, r1, r2          ; Find the difference
        STR r0, [r8]            ; Store the result
```

```
        BX r14          ; Return from subroutine
        ; Define the variables
a0_1    DCD 0x0540      ; 5.25
a0_2    DCD 0x0380      ; 3.5
a1_1    DCD 0x06c0      ; 6.75
a1_2    DCD 0x0420      ; 4.125
x0_1    DCD 200
x0_2    DCD 300
x1_1    DCD 100
x1_2    DCD 200
        ; Define the  DATA block
        AREA    data1, DATA
        ; Set aside stack area
Stack   SPACE 20                ; Make stack 5 words long
StackEnd
        ; Set space aside for variables
z1      SPACE 4
z2      SPACE 4
        END
```

PS3–2.s

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;      Brian Arnberg - ELEC6260            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;      Problem Set 3, Problem 2            ;;
;;        Execute the C program             ;;
;;                                          ;;
;;                                          ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        ; Define the CODE block
        AREA    RESET, CODE
        ; Begin
        ENTRY
__main  LDR r13, =StackEnd
        ; int a
        LDR r5, =aa                 ; Address of aa to r0
        ; k = 0;
        LDR r0, =kk                 ; Get Address of kk
        MOV r1, #0                  ; Set r1 to 0
        STR r1, [r0]                ; Store r1 (#0) to r0 ([kk])
        ; f3(3)
        MOV r5, #3                  ; Put argument (#3) in r5
        STR r5, [r13, #-4]!         ; Push argument (#3) to stack
        BL func3
        ADD r13, #4                 ; POP argument off of stack
        ; a = f2(2)
        MOV r1, #2
        STR r1, [r13, #-4]!         ; Push argument (#2) to stack
        BL func2
        LDR r0, =aa                 ; Get address for aa
        LDR r1, [r13], #4           ; Get returned value from f2
        STR r1, [r0]                ; Store returned value to aa
halt    B halt          ; branch to self, halt program

;; int f1(int x1, int x2) {
;;      return x1 + x2;
;; }
func1
        LDR r0, [r13], #4           ; Take 1st argument from stack, then change sp
        LDR r1, [r13], #4           ; Take 2nd argument from stack, then change sp
        ADD r0, r0, r1              ; Add arg1 and arg2, then put in r0
        STR r0, [r13, #-4]!         ; Store result to stack
        BX r14 ; Return from function
;; int f2(int x1) {
;;      return x1 + 1;
;; }
func2
        LDR r0, [r13]               ; Load argument from stack
        ADD r0, #1                  ; Add 1 to the argument
        STR r0, [r13]               ; Store result to stack, save over previous value
        BX r14  ; Return from function
;; void f3(int r) {
;;      int j;
;;      for (j=0; j<2; j++)
;;              k = k + f1(r+j, 5);
;; }
func3
        LDR r7, [r13]               ; load argument from stack
        STR r14, [r13, #-4]!        ; Put current return address in stack
        EOR r5, r5, r5              ; Use r5 for j, Clear all bits
        MOV r6, #2                  ; Use for N
loopf3  CMP r5, r6      ; Compare j to 2
        BGE endf3                   ; if j >= 2, branch to endf3
        ADD r0, r7, r5              ; Compute r + j
        MOV r1, #5                  ; Set r1 to 5
        STR r5, [r13, #-4]!         ; PUSH j to stack
        STR r6, [r13, #-4]!         ; PUSH N to stack
        STR r7, [r13, #-4]!         ; PUSH Argument to stack
        STR r0, [r13, #-4]!         ; PUSH first argument to f1 to stack
        STR r1, [r13, #-4]!         ; PUSH second argument to f1 to stack
        BL func1        ; Call f1(r + j, 5)
        LDR r2, [r13], #4           ; Get returned value
        LDR r7, [r13], #4           ; Get Argument from stack
        LDR r6, [r13], #4           ; Get N
```

```
        LDR r5, [r13], #4       ; Get j
        LDR r10, =kk            ; Get address of kk
        LDR r3, [r10]           ; Store current kk to r3
        ADD r2, r2, r3          ; SUM k and result of the function call
        STR r2, [r10]           ; Store k
        ADD r5, r5, #1          ; Increment j
        B loopf3
endf3
        LDR r14, [r13], #4      ; Get return address from stack
        BX r14  ; Return from function

        ; Define the DATA block
        AREA    data1, DATA
        ; Set aside stack area
Stack   SPACE 40                ; Make stack 10 words long
StackEnd
        ; Set space aside for variables
kk      SPACE 4
aa      SPACE 4
        END
```

PS3–2b.txt

```
    14: int f1(int x1, int x2) {
0x20000228 4602      MOV           r2,r0
    15:         return x1 + x2;
0x2000022A 1850      ADDS          r0,r2,r1
    16: }
    17:
0x2000022C 4770      BX            lr
    18: int f2(int x1) {
0x2000022E 4601      MOV           r1,r0
    19:         return x1 + 1;
0x20000230 1C48      ADDS          r0,r1,#1
    20: }
    21:
0x20000232 4770      BX            lr
    22: void f3(int r) {
    23:         int j;
0x20000234 B510      PUSH          {r4,lr}
0x20000236 4604      MOV           r4,r0
    24:         for (j = 0; j < 2; j++)
0x20000238 2300      MOVS          r3,#0x00
0x2000023A E009      B             0x20000250
    25:                 k = k + f1(r + j, 5);
0x2000023C 18E0      ADDS          r0,r4,r3
0x2000023E 2105      MOVS          r1,#0x05
0x20000240 F7FFFFF2  BL.W          f1 (0x20000228)
0x20000244 490A      LDR           r1,[pc,#40]  ; @0x20000270
0x20000246 6809      LDR           r1,[r1,#0x00]
0x20000248 4408      ADD           r0,r0,r1
0x2000024A 4909      LDR           r1,[pc,#36]  ; @0x20000270
0x2000024C 6008      STR           r0,[r1,#0x00]
0x2000024E 1C5B      ADDS          r3,r3,#1
0x20000250 2B02      CMP           r3,#0x02
0x20000252 DBF3      BLT           0x2000023C
    26: }
    27:
0x20000254 BD10      POP           {r4,pc}
    28: void main() {
    29:         int a;
0x20000256 B500      PUSH          {lr}
    30:         k = 0;
0x20000258 2000      MOVS          r0,#0x00
0x2000025A 4905      LDR           r1,[pc,#20]  ; @0x20000270
0x2000025C 6008      STR           r0,[r1,#0x00]
    31:         f3(3);
0x2000025E 2003      MOVS          r0,#0x03
0x20000260 F7FFFFE8  BL.W          f3 (0x20000234)
    32:         a = f2(2);
0x20000264 2002      MOVS          r0,#0x02
0x20000266 F7FFFFE2  BL.W          f2 (0x2000022E)
0x2000026A EE000A10  VMOV          s0,r0
    33: }
```