Oregon State University

CS 472 - Computer Architecture

Spring 2014

Lab 3

Arm Assembly Book Exercises

Author:
Drake Bridgewater
Ryan Phillips

Professor:
Kevin McGrath

April 27, 2014

For this lab, you will be implementing some exercises from the book in ARM assembly.

All requested exercises should be tested from a main body of the code, and should comprise a single, testable program in ARM assembly.

3.57 We need to swap the following registers. Do this using block moves.

Before	After
r1	r3
r2	r4
r3	r5
r4	r6
r5	r7
r6	r1
r7	r2

block_move/block_move.asm

```
  AREA block_move, CODE, READWRITE
    ENTRY

  ; note - memmap:
  ; rw for: 0xffff0000, 0xffffffff
  ; for stack pointer?

  ; adding in values for testing
  MOV r1, #1;
  MOV r2, #2;
  MOV r3, #3;
  MOV r4, #4;
  MOV r5, #5;
  MOV r6, #6;
  MOV r7, #7;

  STMDB   sp!,{r1-r7}
  LDMIA   sp!, {r3-r7}
  LDMIA   sp!, {r1,r2}

l   b     l ; infinite loop

  END
```

3.59 Write a function (subroutine) that inputs a data value in register r0 and returns value in r0. The function returns $y = a + bx + cx^2$, where a, b, and c are parameters built into the function (i.e., they are not passed to it). The subroutine also performs clipping. If the output is greater than the value d, it is constrained to d (clipped). The input in r0 is a positive binary value in hte range 0 to 0xFF. Apart from r0, no other registers may be modified by this subroutine.

quad_func/quad_func.asm

```
    AREA quad_func, CODE, READWRITE
    ENTRY

    ; mul requires more than one register
    ; we must assume then that we can modify registers?
    ; let's just save r1-r4 and then restore at the end
    ;
    ; note: i had to memmap 0xfffffff0, 0xffffffff to write in order to
       use sp
    STMDB   sp!,{r1-r4}

    MOV   r0, #5 ; user input

    MUL   r1, r0, r0 ; x^2 -> r1
    LDR   r2, Cv ;
    MUL   r4, r1, r2 ; cx^2 -> r4

    LDR   r2, Bv ;
    MUL   r3, r0, r2 ; bx -> r3

    LDR   r2, Av ; a -> r2

    ADD   r1, r2, r3 ; a + bx -> r1
    ADD   r1, r1, r4 ; (a + bx) + cx^2 -> r1

    LDR   r2, Dv ;
    CMP   r1, r2 ; compare result to Constraint
    BLE   Finish ; it's less than... good!
    MOV   r1, r2 ; else: replace with constaint

Finish    MOV   r0, r1 ; r0 contains final result
    ; restore registers
    LDMIA   SP!, {r1-r4}
Done  B   Done

    ; These are the customs variables for A,B,C, & D
Av    DCD   2
Bv    DCD   3
Cv    DCD   4
Dv    DCD   200 ; our constraint

    END
```

3.60 A computer has three eight-element vectors in memory, Va, Vb, and Vc. Each element of a vector is a 32-bit word. Write the code to calculate all elements of Vc if the ith element is given by

$Vc_i = \frac{1}{2}(Va_i + Vb_i)$

avg_vector/avg_vector.asm

```
    AREA avg_vector, CODE, READONLY

    ; once again: map -> rwe: 0, 0x0000ffff

    MOV   r0, #0 ; our counter

    ADR   r1, Va
    ADR   r2, Vb
    ADR   r3, Vc

loop  LDR   r4, [r1], #4
    LDR   r5, [r2], #4
    CMP   r0, #8 ; if at 7, we have performed all 8 calculations
    BEQ   done
    ADD   r0, #1 ; increment our loop counter
    ADD   r7, r4, r5
    ASR   r7, r7, #1 ; shift of one bit divides by 2

    STR   r7, [r3], #4 ; save value to Vc

    b   loop

done  b     done

Va    DCD 5,5,5,5,7,7,7,8
Vb    DCD 1,2,3,4,5,6,7,8
Vc    DCD 0,0,0,0,0,0,0,0 ; this is just a place holder for Vc

    END
```

Endianness Test/Flip A function which tests the endianness of the system, and flips it as requested. This will be expanded upon in a later lab to examine the concept of endian neutral programming.

endianness_testing/endianness2.asm

```
  AREA endianness_testing, CODE, READWRITE
    ENTRY
;A function which tests the endianness of the system, and flips it as
   requested. This will be expanded upon in a later lab to examine the
   concept of endian neutral programming.

  MOV   R0, #1 ; if 1, swap

  ADR   R1, x
  LDRB  R2, [R1], #1
  LDRB  R2, [R1], #1
  LDRB  R2, [R1], #1

  ADR   R3, y
  LDRB  R4, [R3], #1
```

```
14
    CMP      R2, R4
16  BEQ      big ;
    b    small
18
big
20  CMP    R0, #1
    BNE    l
22  b    swap

24 small
    CMP    R0, #1
26  BNE    l
    b    swap
28
swap
30
l    b      l ; infinite loop
32
x           DCD    4294945450 ; ffffaaaa
34        space    10
y           DCD    2863311530 ; aaaaaaaa
36
    END
```