Drake Bridgewater & Ryan Phillips
April 28, 2014
CS 472 HW 4

(3.8)

What are the relative advantages and disadvantages of general-purpose registers compared to separate address and data registers?

(3.9)

What is a misaligned operand? Why are misaligned operands such a problem in programming?

A misaligned operand is just what it sounds like an operand that is not aligned causing the operand to span two cache lines either slowing the system down substantially or causing a bus error. From some articles online typically x86 and x64 will happily work with unaligned data but will tend to be slower where as on other processors the system will just error. From the ARM website ”Non word-aligned load and store multiple, double, semaphore, synchronization, and coprocessor accesses always signal Data Abort with an Alignment fault status code when the U bit is set.” [1]

(3.24)

What is the meaning of each of the P,U,B,W, and L bits in the encoding of an ARM memory reference instruction?

P	Do you want to adjust pointer before using?
U	Do you want to decrement the pointer instead of incrementing it?
B	Is this word access?
W	Do you want to update the pointer after use?
L	Do you want to store data in memory?

(3.26)

What is the effect of LDR r0, [r5, r6, LSL r2] ?

$[r0] \leftarrow [r5 + [r6 * r2]]$ or in english store what is in R5 with the addition of the offset determined by logically shifting r6 left by r2 in r0

(3.30)

What is the meaning of sign-extension in the context of copying data from one location to another?

Sign extension is maintaning the binary sign when changing the length of represented value. For instance if you are moving a 16-bit value into a 32-bit register you have to ensure that you sign extend the bits this means that the most significant bit will need to be repeated until the end in the new register.

(3.33)

Most RISC processors do not include a block move instruction. What are the advantages and disadvantages of the ARM's LDM and STM instructions?

Using a block load or store instruction can be excuted in few cycles for more then a single register. For instance if you want to block move 16 register it will take 8 cycles if

(3.34)

What is the effect of executing STMIB r13!,{r0-r2,r4}? Draw a picture of the state of the stack pointed at by r13 before and after this operation.

The instruction STMIB is the Store Multiple Registers instruction and it increments the address before each transfer and with the addition of the exclimation mark on Rd the final address is written back to Rd, R13

Assuming this is a Full Ascending, decrement after stack

	Before	After
$\& \, n+0$	$\rightarrow$	$R0$
$\& \, n+4$		$R1$
$\& \, n+8$		$R2$
$\& \, n+C$		$\rightarrow R4$
$\& \, n+F$		Free

(3.36)

Without using the ARM's multiplication instruction, write one or more instructions (using ADD, SUB, and shifting) to multiply by the following integers: 33, 1025, 4095

In binary:

33	0010 0001
1025	0100 0000 0001
4095	1111 1111 1111
$33 \times 1025 = 33825$	1000 0100 0010 0001

```
AREA test, CODE, READWRITE
    ENTRY

    ; multiply input by 33 -> original += 4 byte shift
    MOV        R0, #4095              ;
    MOV        R1, R0, LSL #5         ; r1 = r0 * 2^5
    ADD        R2, R1, R0             ; r2 = r1 + r0
                                      ; r2 contains r0 * 33
    space     10
    ; multiply input by 33 -> original += 4 byte shift
```

```
        MOV        R0, R2               ;
        MOV        R1, R0, LSL #10      ; r1 = r0 * 2^5
        ADD        R2, R1, R0           ; r2 = r1 + r0

        space     10

        END
```

(3.44)

What does the following code do?

```
TEQ     r0, #0
RSBMI   r0, r0, #0
```

TEQ is the instruction test equivalence it uses a bitwise exclusive OR operation when it is complete it sets the flags and the result is discarded unlike the EORS instruction
RSBMI is the instruction Reverse Subtract without carry but the addition of MI means that it will only execute when N status register is set, it is negative.

TEQ will set the N, negative flag, for any negative value. If the value is a negative then RSBMI instruction will execute; yielding 0 - R0. Overall these two instructions used together will take the absolute value of R0.

(3.48)

What, in the context of assembly language, is a psuedo-operation?

These are keywords which do not directly translate to a machine instruction

(3.54)

Explain what this fragment of code does instruction by instruction and what purpose it achieves (assuming that register r0 is the register of interest). Note that the data in r0 must not be 0 on entry.

```
        MOV        r1,#0
loop    MOVS       r0,r0,LSL #1
        ADDCC      r1,r1,#1
        BCC        loop
```

Moves 0 into R1
Moves R0 left shifted by 1 into R0 and updated the status register
Adds $R1 = R1 + 1$ if carry then flag is set
Loops back to loop if carry then flag is set

This will multiply the value in R0 by 2 each until the most significant bit is a 1 and the value accrued in R1 is the value that is was multiplied by.

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

../Lab3/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
```

(3.61)

Register r15 is the program counter. You can use it with certain instructions such as a MOV (e.g., MOV pc, r14). However, r15 cannot be used in conjunction with most data processing instructions. Why?

Since the program counter is not connected to the ALU instuctions can't be directly executed using the value in the program counter. If you want to use the program counter you must load it into a register then manipulate that register

References

[1] ARM Ltd. The architecture for the digital world, 2014.