Drake Bridgewater & Ryan Phillips
April 21, 2014
CS 472 HW 3

(3.1)

Why is the program counter a pointer and not a counter?

> It saves a step. If it were a counter, an address would still need to be stored somewhere, and arithmatic on this address would need to be performed using the counter. But since the program instructions are stored sequentially in memory, the address itself can simply be incremented to get the next instruction.

(3.2)

Explain the function of the following registers in a CPU.

a) PC

> The Program Counter. It is incremented after an instruction is fetched. It holds the memory address of the next instruction to be executed.

b) MAR

> The Memory Address Register. It works in tandom with the MDR. It either contains the location of the where the MDR data should be written to, or the location of data which will be read into the MDR.

c) MDR

> The Memory Data Register. It contains the data referred to in the MAR explanation above.

d) IR

> Instruction Register. This contains the instruction which is currently being executed. It is fetched from the location pointed to by the PC.

(3.3)

For each of the following 6-bit operations, calculate the values of the C, Z, V, and N flags.

a.

$$\begin{array}{r} 001011 \\ +001101 \\ \hline 011000 \end{array}$$

C = 0, Z = 0, V = 0, N = 0

b.

$$\begin{array}{r} 111111 \\ +000001 \\ \hline 1000000 \end{array}$$

C = 1, Z = 1, V = 0, N = 0

c.

$$\begin{array}{r} 000000 \\ -111111 \\ \hline 111111 \end{array}$$

C = 0, Z = 0, V = 1, N = 1

d

$$\begin{array}{r} 101101 \\ +011011 \\ \hline 1001000 \end{array}$$

C = 1, Z = 0, V = 0, N = 1

e

$$\begin{array}{r} 000000 \\ -000001 \\ \hline 111110 \end{array} \quad \text{C = 0, Z = 0, V = 0, N = 1}$$

f.

$$\begin{array}{r} 111110 \\ +111111 \\ \hline 1111101 \end{array} \quad \text{C = 1, Z = 0, V = 1, N = 1}$$

(3.10)

Why does the ARM provide a reverse subtract instruction RSB r0,r1,r2 that implements [r0] = [r2] - [r1] when the normal subtraction instruction SUB r0,r2,r1 will do exactly the same job?

These two operand only differ by the order of operation, and can be seen as unnecessary as switching the two operands for SUB should suffice. The problem arises when only the second operand can be constant or other special forms. All operations could be done using the SUB operations but additional registers would be necessary as seen in this example: $SUB\ R2,\ R4,\ \#8$ if you wanted to do the exact opposite you would need an an additional register to hold the constant #8, but using RSB it would just be $RSB\ R2,\ R4,\ \#8$

(3.17)

ARM instructions have a 12-bit literal. Instead of permitting a word in the range 0 to 2^12-1, the ARM uses an 8-bit format for the integer and a 4-bit alignment field that allows the integer to be shifted in steps of two. What are the advantages and disadvantages of this mechanism in comparison to a straight 12-bit integer?

These 8-bit literal that can be scaled by a power of 2. you could even say that arm provides a type of floating point literal.

The four most significant bits of the literal field specify the literal's alignment within a 32-bit word. if the 8 bit immediate value is N and the 4-bit alignment is n in the range 0-12 then the value of the literal is given by $N \times 2^{2n}$. Thus, arm provides an 8-bit literal that is scaled by a power of 2.

(3.18)

Write one or more ARM instructions that will clear bits 20 to 25 inclusive in register r0. All the other bits of r0 should remain unchanged.

$LDR \quad R3, = 0xFC0F\ FFFF$
$AND \quad R0, R3, R0$

```
1     AREA clearbits, CODE, READONLY
      ENTRY
3 start
      AND r0,r0,#2_11111111111111111110000001111111
5     END
```

(3.19)

This is a classic problem of assembly language programming. Write a sequence of ARM instructions that swap the contents of registers r0 and r2 without using any additional registers or memory storage (that is, you can't move r1 to a temporary location).

Our original thought similar the the integer implementation where you add them and together and subtract the other like this

```
x=15;
y=21;

x=x+y;
y=x-y;
x=x-y;
```

But we had some concern with the S bits. This lead us to some research online where we found an article titled **The Magic of XOR** [1] which introduced us to an extremely fascinating method of swapping two values. This process is basically XORing the three times as the pseudocode below depicts.

```
x=1101;
y=0110;

x=x EOR y;        //1101 EOR 0110 = 1011
y=x EOR y;        //1011 EOR 0110 = 1101
x=x EOR y;        //1011 EOR 1101 = 0110
```

(3.25)

What is the binary encoding of the following instructions?

	Condition	0 0		Op-code	S	$r_{source1}$	$r_{destination}$	Operand 2
A								

	Condition	0 1	#	P	U	b	W	L	r_{base}	$r_{transfer}$	Operand 2
B											
C											
D											

a) STRB r1, [r2]

E5C21000 → 11100101110000100001000000000000

b) LDR r3, [r4, r5]!

E7B43005 → 11100111101101000011000000000101

c) LDR r3, [r4], r5

E6943005 → 11100110100101000011000000000101

d) LDR r3, [r4,#-6]!

E5343006 → 11100101001101000011000000000110

(3.39)

Write an ARM program that scans a string terminated by the null byte 0x00 and copies the string from a source location pointed by r0 to a destination pointed at by r1.

NullTerminatingString/NullTerminatingString.asm

```
1           AREA NullTerminatingString, CODE, READWRITE
            ENTRY
3           ;;modification of code from section 3.9.11
            ;;copies from r0 to r1
5
  copystr   ADR R0, src ; r0 points at string
7           LDR R1, =0x0000005f   ; a location of safe mem?

9 loop      LDRB  R2, [R0], #1 ; grab first character, increment pointer
            CMP   R2, #0x00 ; are we at the end?
11          STRB  R2, [r1], #1 ; if not, copy to address location (from r1)
            BNE   loop ; if not at the end... we move onto the next
               character
13          MOV   PC, R14

15 src =    "aaaaaaaazzzzzz", &0A, &0D, 0
            END
```

(3.51)

Write an ARM assembly language program to determine whether a string of characters with an odd length is a palindrome (for example, mom) under the following constraints.

a. The string of ASIC-encoded characters is stored in memory.

b. At the start of the program, register r1 contains the address of the first character in the string, and r2 contains the address of the last character. On exit from the program, register r0 contains a 0 if the string is a not a palindrome, and 1 if it is.

../lab2.new/lab2_part2/palindrome.asm

```
            AREA palindrome, CODE, READWRITE
            ENTRY

            MOV     R0, #1 ; palindrome flag set at true
            ADR   R1, src ; r0 points at string
            ADR   R2, src ; r1 points at string END

            ;get pointer to end of the string and store in r2
            ;R2 is temp pointer
getstrend   LDRB    R3, [R2], #1        ;load r0 into r2 and increment

            CMP R3, #0x00
            BNE   getstrend

            SUB     R2, #2
loop        CMP     R1, R2
            BEQ     pass
            LDRB  R3, [R1], #1 ; grab first character, increment pointer
            LDRB  R4, [R2], #-1 ; grab last character, decrement pointer
            CMP   R3, R4 ; do they match?
            BNE     fail
            CMP     R1, R2 ; if these addresses are the same... we are
               done
            BEQ   pass

            B   loop ; if not at the end... we move onto the next
               character

fail        MOV     R0, #0 ; is not a palindrome
            B   fail ; infinite loop!

pass        MOV     R0, #1 ; if a palindrome
            B   pass ; infinite loop!


src =       "amanaplanacanalpanama"
            END
```

References

[1] Charles Lin. The magic of xor, 2003,2004.