Study Questions - #1

(Arithmetic +/- and Pseudo Instructions)

1. **Question 3.2 at page 224: Explain the function of the following registers in a CPU**

   a. PC

   A program counter is a CPU register that is resided in the processor whose main function is to hold the next instruction address which has to be executed.

   b. MAR

   A Memory Address Register is a CPU register which holds the instruction address that is currently reading from the memory or writing into the memory.

   c. MBR

   A Memory Buffer Register is a register that holds the data which is reading from the memory or writing into the memory. The MBR holds the copy of the memory location which is specified by the MAR.

   d. IR

   The Instruction Register is to hold the instruction currently being read. It stores the instruction most recently read from main memory.

2. **Question 3.3 at page 224: For each of the following 6-bit operations calculate the values of the C, Z, V, and N flags.**

   a. 001011 + 001101

   I believe C = Z = N = V = 0

a. 001011 + 001101

$$\begin{array}{r} 001011 \\ +001101 \\ \hline 0011000 \end{array}$$

In the above arithmetic operation:

- No Carry occurs while addition, hence carry flag is not set that is C=0.
- The result is not all zeros, hence zero flag is not set that is Z=0.
- The MSB is zero, hence it is not negative that is N=0.
- Overflow is, carry into the MSB (0) exor carry out of MSB (0), hence it is zero. That is V=0.

b. 111111 + 000001 - C=1, Z=1, all others zero

b. 111111 + 000001

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

In the above arithmetic operation:

- Carry occurs while addition, hence carry flag is set that is C=1.
- The result is all zeros, hence zero flag is set that is Z=1.
- The MSB is one, hence it is not negative that is N=0.
- Overflow is, carry into the MSB (1) exor carry out of MSB (1), hence it is zero. That is V=0.

c. 000000 - 111111 - all flags are zero

c. 000000 - 111111

$$\begin{array}{r} 000000 \\ -111111 \\ \hline 000001 \end{array}$$

In the above arithmetic operation:

- No borrow occurs while subtraction, hence carry flag is not set that is C=0.
- The result is not all zeros, hence zero flag is not set that is Z=0.
- The MSB is zero, hence it is not negative that is N=0.
- Overflow in case of subtraction operation is, borrow into the MSB (0) exor borrow out of MSB (0), hence it is zero. That is V=0.

d. 101101 + 011011 - C=1, not sure about N, all others are zero

d. 101101 + 011011

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

In the above arithmetic operation:

- Carry occurs while addition, hence carry flag is set that is C=1.
- The result is not all zeros, hence zero flag is not set that is Z=0.
- The MSB is zero, hence it is not negative that is N=0.
- Overflow is, carry into the MSB (1) exor carry out of MSB (1), hence it is zero. That is V=0.

e. 000000 – 000001 = N=1 all others are zero

e. 000000 – 000001

$$\begin{array}{r} 000000 \\ -000001 \\ \hline 111111 \end{array}$$

In the above arithmetic operation:

- No borrow occurs while subtraction, hence carry flag is not set that is C=0.
- The result is not all zeros, hence zero flag is not set that is Z=0.
- The MSB is one, hence it is negative that is N=1.
- Overflow in case of subtraction operation is, borrow into the MSB (1) exor borrow out of MSB (1), hence it is zero. That is V=0.

f. 111110 + 111111 N= 1, C=1 , all others are zero

f. 111110 + 111111

$$\begin{array}{r} 111110 \\ +111111 \\ \hline 1111101 \end{array}$$

In the above arithmetic operation:

- Carry occurs while addition, hence carry flag is set that is C=1.
- The result is not all zeros, hence zero flag is not set that is Z=0.
- The MSB is one, hence it is not negative that is N=0.
- Overflow is, carry into the MSB (1) exor carry out of MSB (1), hence it is zero. That is V=0.

4. **Question 3.30 at page 225: What is the meaning of *sign-extension* in the context of copying data from one location to another?**

Sign-Extension is needed when values of registers are copied or moved to registers with higher memory capacity. If the value is an 8-bit register is 1010 1010, then when we move it to a 16-bit or 32-bit register would mean that the left bits are all wasted.

5. **Question 3.31 at page 225: Why is sign-extension an important issue when we use a LDR instruction to load a register from memory, but of no importance when we use an STR to store a register in memory?**

   To keep the sign-bit unchanged? She so horny and love you long long time. Try to store less in STR. Want to use up as much in LDR?

   Pretty sure it's because a STR only stores values with an offset further down memory - therefore it is an unsigned value and does not need to be sign extended?

6. **Question 3.48 at page 226: What, in the context of assembly language, is a *pseudo-operation*?**

   A pseudo-operation defines that a program sending its code to the hardware device for compilation and later hardware device will translate these instructions into equivalent machine language

7. **Question 3.49 at page 226: Explain the meaning of the following two ARM pseudo-operations. What do they do and why have they been implemented?**

   a. ADR r0,table
   The ADR instruction loads the destination register using an address. It creates "suitable code" to load the 32-bit value to register. The effective address label is "table" which is loaded into the register "r0" and is used to set up the pointer

   b. LDR r0,=0x1234FEDC
   The LDR instruction loads the register using a stated value. Here, it creates "suitable code" to load the register with the stated 32-bit value in hex.

   Not exactly, this is correct except the value is actually created into memory, and then is loaded into the register.

8. **Question 3.50 at page 226: Suppose you execute LDR,r0=0x12345678 on an ARM machine followed by STR r0,[r1] where r1 =0x1000. Now, suppose you do a byte read to the same address with LDRB r2,[r1]. What would the value be in r2 if (a) the ARM was big-endian configured and (b) it was little-endian configured?**

Correct answer: In a big-endian world, the byte read would be 12, and in a little-endian world 78 would be read.

(a) The ARM was big - endian configured:

According to the big-endian mechanism, the most significant value in the sequence is stored at the lowest storage address.

Therefore, the representation is as follows:

Address	Value
00	0001
01	0000
10	0000
11	0000

(b) It was little - endian configured:

According to the little-endian system, the least significant value in the sequence is stored first.
Therefore, the representation is as follows:

Address	Value
00	0000
01	0000
10	0000
11	0001

9. **Question 3.56 at page 226: The following is a loop expressed in ARM code. The code is wrong. Why?**

```
MOV r0,#10 ;Loop counter – ten times
```

```
MOV r1,#0 ;initialize the total to zero

Next    ADD r1,r1,r0 ;add loop counter to total

SUBS r0,r0,#1 ;decrement loop counter

BNE Next ;continue until all done
```

10. Question 3.61 at page 227: 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?

The PC is the program counter and holds the address of the next instruction that has to be executed. When the same register r15 is used in conjunction with the data processing instructions which are used to modify the values may alter the instructions execution flow and crash the system. This is why most microprocessors like ARM do not allow access to PC

11. What do the following terms stand for?

a. MAR: memory address register

b. MBR: memory buffer register

c. IR: instruction register

d. PC: program counter

e. SP: stack pointer

f. LR: link register

g. ISA: instruction set architecture

h. RTL: register transfer language

i. RISC: reduced instruction set computer

j. CISC: complex instruction set computer

k. ARM: advanced RISC machines

l. ALU: arithmetic logic unit

m. CCR: condition code register

N. CPSR: current program status register

12. Describe, using RTL notation, how RISC processors perform the fetch phase of the processing cycle.

[MAR] ← [PC]

[PC] ← [PC] + 4

[MBR] ← [[MAR]]

[IR] ← [MBR]

13. What does the term *word aligned address* mean?

A word aligned address is 32-bits long , meanwhile a half-word is 16-bits.

Correctly aligned on 32-bit boundaries i.e to start at multiple of 4 address location

14. What does the term *half-word aligned address* mean?

If the address ends in 0 you're good (divisible by 2). Correctly aligned on 16 bit boundaries.

15. Is the address 0x123468AA a *word aligned* address?

- Converted to decimal: 305424554
- Divided by 4 (because word): 76356138.5
  - § No, the address is not a word aligned address
  - § A = 10; 10 != 0 mod 4

A represents 10, not divisible by 4; not word-aligned (word aligned is 32 bits)

16. Is the address 0x123468AA a *half-word aligned* address?

- Converted to decimal: 305,424,554

- Divided by 2 (because halfword): 152,712,277
  - § Yes, the address is a half-word aligned address
  - § A = 10; 10 = 0 mod 2

A represents 10, divisible by 2 (half word is 16 bits)

17. Is the address 0x123468AB a *word aligned* address?

- Converted to decimal: 305424555
- Divided by 4 (because word): 76356138.75
  - § No, the address is not a word aligned address
  - § B = 11; 11 != 0 mod 4

18. Is the address 0x123468AB a *half-word aligned* address?

- Converted to decimal: 305424555
- Divided by 2 (because word): 152712277.5
  - § No, the address is not a half-word aligned address
  - § B = 11; 11 != 0 mod 2

19. Is the address 0x123468AC a *word aligned* address?

- Converted to decimal: 305424556
- Divided by 4 (because word): 75,356,139
  - § Yes, the address is a word aligned address
  - § C = 12; 12 = 0 mod 4

20. Is the address 0x123468AC a *half-word aligned* address?

- Converted to decimal: 305424556

- Divided by 2 (because halfword): 152712278
  - § Yes, the address is a halfword aligned address
  - § C = 12; 12 = 0 mod 2

21. What is the main function of *program counter* and how it is differ than *location counter*?

Location Counter is a variable inside assembler to keep track of memory locations during assembling a program, whereas a Program Counter is a register to keep track of the next instruction to be executed in a program at run time

23. Write only ONE ARM instruction to copy into r0 the value 0x12

Mov r0,#0x12

24. Write only ONE ARM instruction to copy into r0 the value -0x12

MOV r0, #-0x12

25. Write only ONE ARM instruction to copy into r0 the value 0x12345678

LDR r0, =0x12345678 (when it's over a threshold of bits)

26. Write only ONE ARM instruction to copy into r0 the value -0x12345678

LDR r0, =-0x12345678

27. Write only ONE ARM instruction to copy into r0 the value 0xFFFFFFEE

LDR r0, =0xFFFFFFEE

28. Write only ONE ARM instruction to copy into r0 the value -0xFFFFFFEE

LDR r0, =-0xFFFFFFEE

29. Write only THREE ARM instructions to copy 0xFF into r1, copy the value in r1 into r2, add the values in r1 and r2, and finally store the result in r0. How do you enforce updating the condition flags after the addition operation?

MOV r1, #0xFF
MOV r2, r1
ADDS r0, r1, r2

You enforce updating the condition flags by using the ADDS command instead of simply ADD.

30. Write only THREE ARM instructions to copy 0xFF into r1, copy the negative of the value in r1 into r2, add the values in r1 and r2, and finally store the result in r0.How do you enforce updating the condition flags after the addition operation?

LDR r1, = 0xFF

NEG r2, r1

ADDS r0, r1, r2

31. Write only THREE ARM instructions to copy 0xFF into r1, copy the complement of each bit of the value in r1 into r2, add the values in r1 and r2, and finally store the result in r0. How do you enforce updating the condition flags after the addition operation?

MOV r1, #0xFF

MVN r2, r1

ADDS r0, r1, r2

32. Write only THREE ARM instructions to copy 0xFF into r1, copy 0xEE into r2, add the values in r1 and r2, and finally store the result in r0. How do you enforce updating the condition flags after the addition operation?

APPEND S to ADD so ADDS

33. Write only THREE ARM instructions to copy 0xFF into r1, copy 0xFEDCBA98 into r2, add the values inr1 and r2, and finally store the result in r0. How do you enforce updating the condition flags after the addition operation?

MOV r1, #0xFF

```
LDR r2, =0xFEDCBA98

ADDS r0, r1, r2
```

34. Write only TWO ARM instructions to copy 0xFF into r1, add 0xEE to the values in r1, and finally store the result in r0. How do you enforce updating the condition flags after the addition operation?

```
MOV r1, #0xFF

ADDS r0, r1, #0xEE
```

35. Write only THREE ARM instructions to copy 0xFF into r1, copy 0xFEDCBA98 into r2, subtract the value in r1 from the value in r2, and finally store the result in r0. How do you enforce updating the condition flags after the subtraction operation?

```
MOV r1, #0xFF

MOV r2, #0xFEDCBA98

SUBS r0, r2, r1
```

36. Write only TWO ARM instructions to copy 0xFF into r1, subtract 0xEE from the value in r1, and finally store the result in r0. How do you enforce updating the condition flags after the subtraction operation?

```
MOV r1, #0xFF

SUBS r0,r1,#0xEE
```

37. Write only TWO ARM instructions to copy 0xFF into r1, subtract the value in r1 from 0xEE, and finally store the result in r0. How do you enforce updating the condition flags after the subtraction operation?

```
MOV r1,#0xFF

RSBS r0, r1, #0xEE
```

38. How to extend ARM arithmetic capabilities to make it able to add two extended precisions integer values (64-bits each). Give an example. Write down all assumptions you will make.

Take advantage of the carry bit and the alternative ADD operations ADDS & ADC

First, assign the first 64-bit number to the register pair r0 & r1, and the second to r2 & r3.

ADDS r4, r0, r2 ; Using ADDS to force the carry bit to be updated

ADC r5, r1, r3 ; USING ADC to use the carry bit from before

39. How to extend ARM arithmetic capabilities to make it able to add two extended precisions integer values (96-bits each). Give an example. Write down all assumptions you will make.

Same as above but with 3 registers per number

40. What are the main differences between ADD, ADDS, ADC, and ADCS? Provide numeric examples to demonstrate the differences.

ADD, ex:

ADDS, add and update status flags

ADC, Add with carry

ADCS, add with carry and update status flags

45. After executing the following ARM instructions

LDR **r0**,=0x87654321

LDR **r1**,=0x87654321

ADDS **r2**,r1,r0

ADC **r3**,r1,r0

What will be the values of the NZCV flags, as well as the values of r0, r1, r2, and r3?

N=0 , V=1, C=1, Z=0 , r0 = 0x87654321 , r1 = 0x87654321 , r2 = 0x0ECA8642, r3 = 0x0ECA8643

46. After executing the following ARM instructions

LDR **r0**,=0xFEDCBA98

LDR **r1**,=0x87654321

ADDS **r2**,r1,r0

ADC **r3**,r1,r0

What will be the values of the NZCV flags, as well as the values of r0, r1, r2, and r3?

N=0 , V=1, C=1, Z=0 , r0 = 0xFEDCBA98 , r1 = 0x87654321 , r2 = 0x8641FDB9, r3 = 0x19652FDB9

N will also be set to 1 because 31st bit is 1. r3 = 0x8641FDBA

47. After executing the following ARM instructions

LDR **r0**,=0xFEDCBA98

LDR **r1**,=0xFEDCBA98

ADDS **r2**,r1,r0

ADC **r3**,r1,r0

What will be the values of the NZCV flags, as well as the values of r0, r1, r2, and r3?

48. After executing the following ARM instructions

LDR **r0**,=0x87654321

LDR **r1**,=0x87654321

SUBS **r2**,r1,r0

ADC **r3**,r1,r0

What will be the values of the NZCV flags, as well as the values of r0, r1 r2, and r3?

49. After executing the following ARM instructions

LDR **r0**,=0xFEDCBA98

LDR **r1**,=0x87654321

SUBS **r2**,r1,r0

ADC **r3**,r1,r0

What will be the values of the NZCV flags, as well as the values of r0, r1, r2, and r3?

50. After executing the following ARM instructions

LDR **r0**,=0xFEDCBA98

LDR **r1**,=0xFEDCBA98

SUBS **r2**,r1,r0

ADC **r3**,r1,r0

What will be the values of the NZCV flags, as well as the values of r0, r1, r2, and r3?

51. After executing the following ARM instructions

LDR **r0**,=0x87654321

LDR **r1**,=0x87654321

RSBS **r2**,r1,r0

ADC **r3**,r1,r0

What will be the values of the NZCV flags, as well as the values of r0, r1, r2, and r3?

52. After executing the following ARM instructions

LDR **r0**,=0xFEDCBA98

LDR **r1**,=0x87654321

RSBS **r2**,r1,r0

ADC **r3**,r1,r0

What will be the values of the NZCV flags, as well as the values of r0, r1, r2, and r3?

53. After executing the following ARM instructions

LDR **r0**,=0xFEDCBA98

LDR **r1**,=0xFEDCBA98

RSBS **r2**,r1,r0

ADC **r3**,r1,r0

What will be the values of the NZCV flags, as well as the values of r0, r1, r2, and r3?

c=1, v=0, n=?, z=0

54. After executing the following ARM instructions

LDR **r0**,=0x1234567

MOV **r1**,r0

What will be the values of the NZCV flags, as well as the values of r0 and r1?

55. After executing the following ARM instructions

LDR **r0**,=0x1234567

MOVS **r1**,r0

What will be the values of the NZCV flags, as well as the values of r0 and r1?

56. After executing the following ARM instructions

LDR **r0**,=0xFEDCBA98

MOV **r1**,r0

What will be the values of the NZCV flags, as well as the values of r0 and r1?

57. After executing the following ARM instructions

LDR **r0**,=0xFEDCBA98

MOVS **r1**,r0

What will be the values of the NZCV flags, as well as the values of r0 and r1?

58. After executing the following ARM instructions

LDR **r0**,= 0x1234567

NEG **r1**,r0

What will be the values of the NZCV flags, as well as the values of r0 and r1?

59. After executing the following ARM instructions

LDR **r0**,=0x1234567

NEGS **r1**,r0

What will be the values of the NZCV flags, as well as the values of r0 and r1?

Study Questions - #2

(Bitwise Logical Operations, multiplication, shifts)

1. **Question 3.11 at page 224: If r1 = 11110000111000101010000011111101, and r2 = 00000000111111110000111100001111, what is the value of r3 after executing BIC r3,r1,r2?**

The given instruction is, BIC r_3, r_1, r_2

r_1=11110000111000101010000011111101

r_2=00000000111111110000111100001111

Complement of r2 is:

$\overline{r_2}$=11111111000000001111000011110000

Perform ADD operation between r_1 and r_2 .

Rules for performing AND operation is as follows:

- Start from the Least Significant bit (LSB) of both the registers r_1 and r_2.
- When a bit of r_1 and the same position bit of r_2 are 1 then the result in that position is 1.
- Else zero (0).

r_1=11110000111000101010000011111101

$(+)r_2$=11111111000000001111000011110000

$\overline{r_3}$=11110000000000001010000011110000

Therefore, after performing BIC between r_1 and r_2 the result of r_3 is

$\boxed{11110000000000001010000011110000}$

4. Question 3.14 at page 224: What is the effect of the instruction MOV r0,r0, ASR #31?

If you perform an arithmetic shift right on a two's complement number, the sign bit is propagated. If you do this operation 31 times, you will be left only with 32 copies of the sign bit. That is, a negative number would yield 1111…1 and a positive number would yield 0000…0. This is a way of creating a 'sign mask'

6. Question 3.18 at page 224: 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.

1111 1100 0000 1111 1111 1111 1111 1111 1111
r1 = 0x0xFC0FFFFF
AND r0,r0,r1

7. Question 3.19 at page 224: This is a classic problem of assembly language programming. Write a sequence of ARM instructions that swap the contents of registers r0 and r1 without using any additional registers or memory storage; that is, you can't move r1 to a temporary location.

Answer:
ADD r0, r0, r1
SUB r1, r0, r1
SUB r1, r0, r1
EOR r0,r0,r1
EOR r1,r0,r1
EOR r0,r0,r1

11. Question 3.44 at page 225: What does the following code do?

TEQ r0,#0
RSBMI **r0**,r0,#0

12. Question 3.45 at page 226: What is the meaning of the following mnemonics (and what do they do)?

a. LDRB
b. RSBLES
c. CMPS

13. Question 3.47 at page 226: What is wrong with the following instruction?

MLA **r0**,r0,r1,r2

94. Write a suitable ARM assembly segment of code to implement the following code, where Z is an integer array (4 bytes per element) which is located at address 0x120.

```
for(r0 = 0; r0 <=
   20; r0++) Z[r0]
   = r0;
```

95. Write a suitable ARM assembly segment of code to implement the following code, where Z is an integer array (4 bytes per element) which is located at address 0x120.

```
for(r0 = 0; r0 <=
   20; r0++) Z[r0]
   += 0x10;
```

96. Write a suitable ARM assembly segment of code to implement the following code.

```
if ((r0 == r1) && (r2 == r3)) r4 += 16 else r5 += 32;
r6 += 64;
```

97. Write a suitable ARM assembly segment of code to implement the following code.

```
if ((r0 == r1) && (r2 == r3)) r4 /= 16 else r5 *= 32;
r6 -= 64;
```

98. Write a suitable ARM assembly segment of code to implement the following code.

```
{ r1 += r1*65;
```

```
    r1 = r1 + r0 << 10;
    if(r1 is odd)
    THEN  r0 = r0 - 1;
    ELSE  r0 = r0 – 2;
}
```

99. Write a suitable ARM assembly segment of code to implement the following code.

```
r0 = 10; r1 = 1; while(r0 >0)
r0 = 10;
r1 = 1;
{ r1 += r1*65;
    r1 = r1 + r0 << 10;
    if(r1 is even)
    THEN
     r0 = r0
    - 1;
    ELSE
     r0 = r0
    – 2;
}
until(r0 < 0)
```

100. Write an ARM code to implement without using any multiplication instruction.

```
if(r0 > r1)
{ r2 = 65535 * r3;
}
else
{ if(r0 == r1)
    { r2 = 65536 * r3;
    }
    else
    { r2 = 65537 * r3;
    }
}
```

101. Write a suitable ARM assembly segment of code to implement the following code.

```
if((r0 == r1) && (r2 == r3))
      r4 /= 4096;
else
      r5 *= 4096;
r6 -= 4096;
```

Study Questions - #3

(Addressing mode, branching, data processing, encoding/decoding)

7. **Question 3.32 at page 225: Assume that r2 contains the initial value 00001000_{16}. Explain the effect of each of the following six instructions, and give the value in r2 after each instruction executes**

   a. STR r1,**[r2]**
   - *R1 = 0x1000*
   - *R2 = 0x1000*

   b. STR r1,**[r2, #8]**
   - *R1 = 0x1008*
   - *R2 = 0x1000*

   c. STR r1,**[r2, #8]!**
   - *R1 = 0x1008*
   - *R2 = 0x1008*

   d. STR r1,**[r2], #8**
   - *R1 = 0x1008*
   - *R2 = 0x1000*

   e. STR r1,**[r2, r0, LSL #8]**
   - *R1 = 0x1000 + (R0)(2^8)*
   - *R2 = 0x1000*

10. **Register r10 contains 0x10000000. Beginning at that address there are four integers in a row (4 bytes each). Write ONE ARM instruction that loads the last integer into register r7.**
    - *LDR r7, [r10, #16]*

11. **Register r10 contains the address 0x10000100. Write ONE ARM instruction that stores the content of r7 into the four bytes that precede this address.**
    - *STR r10, [r7, #4]*

12. **Write only one ARM instruction that adds the value 0x3EC0000 to the value in register r6.**
    - *ADD r6, 0x3EC00000*

13. **Beginning at address 0x100, there are four integer numbers in a row (4 bytes each), i.e., these numbers are at addresses 0x100, 0x104, 0x108, and 0x10C. Assume that register**

r0 contains 0x100. Write one ARM instruction that loads the value of the last integer (i.e., the 4th integer at location 0x10C) into register r1.

- *LDR r1, [r0, #12]*

14. Write a suitable ARM code to implement the following code.

```
int total;
int i;

total = 0;
for (i = 10; i > 0; i--)
{
    total += i;
}
```

```
        MOV r0, #0
        MOV r1, #10
Loop    ADDS r0,r1
        SUBS r1, #1
        BNE Loop
```

15. Explain what this fragment of code does. What are the values of registers at the end of the execution?

```
        MOV  R0, #0       ; R0 accumulates total
        MOV  R1, #10      ; R1 counts from 10 down to 1
again   ADD  R0, R0, R1
        SUBS R1, R1, #1
        BNE  again
halt  B    halt           ;infinite loop to stop computation
```

16. Write a suitable ARM code to implement the following code.

```
    a = 40;
    b = 25;
    while (a != b) {
        if (a > b) a -= b;
        else    b -= a;
    }
```

17. Explain what this fragment of code does. What are the values of registers at the end of the execution?

```
        MOV R0, #40  ; R0 is a
        MOV R1, #25  ; R1 is b
again   CMP R0, R1
        SUBGT R0, R0, R1
        SUBLT R1, R1, R0
```

```
        BNE
again halt  B
halt
```

18. Write a suitable ARM code to implement the following code.

iters ← 0
while *n* ≠ 1**:**
    iters ← *iters* + 1
  if *n* is odd**:**
      n ← 3 x *n* + 1
  else:
    n ← *n* / 2

19. Explain what this fragment of code does. What are the values of registers at the end of the execution?

```
        MOV  R0, #5       ; R0 is current number
        MOV  R1, #0       ; R1 is count of number of
iterations again   ADD  R1, R1, #1  ; increment
number of iterations
        TST  R0, #1         ; test whether R0 is odd
        BEQ  even
        ADD  R0, R0, R0, LSL #1 ; if odd, set R0 = R0 + (R0
        << 1) + 1
        ADD  R0, R0, #1   ; and repeat
        (guaranteed R0 > 1) B  again
even  MOV  R0, R0, ASR #1 ; if even, set R0 = R0 >> 1
        SUBS R7, R0, #1   ; and repeat if R0 != 1
        BNE
         again
halt  B    halt           ; infinite loop to stop
computation
```

20. What are the values of `r0`, `r1`, `r2`, and `r3`*(in hexadecimal)* after executing the following program?

```
      AREA prog, CODE, READWRITE
      ENTRY
      LDR  r0,=AAA1
      LDR  r1,=0x224488
      LDRB r2,[r0,#1]
      LDRB r3,[r0,#2]
Loop  B     Loop
AAA1  DCB  0x10, 0x20, 0x30, 0x40
      END
```

- *R1 = 0x1C*
- *R2 = 0x224488*
- *R2 = 0x20*
- *R3 = 0x30*

21. What are the values of `r0`, `r1`, `r2`, and `r3` *(in hexadecimal)* after executing the following program?

```
        AREA prog, CODE, READWRITE
        ENTRY
        LDR  r0,=AAA1
        LDR  r1,=0x224488
        STR  r1,[r0]
        LDRB r2,[r0,#1]
        LDRB r3,[r0,#2]
Loop    B     Loop
AAA1    DCB  0x10, 0x20, 0x30, 0x40
        END
```

- *R1 = 0x18* ***???why???***
- *R2 = 0x224488*
- *R2 = 0x20*
- *R3 = 0x30*

22. What are the values of `r0`, `r2`, and `r3` *(in hexadecimal)* after executing the following program?

```
        AREA prog, CODE, READWRITE
        ENTRY
        LDR  r0,=AAA1
        LDRB r2,[r0,#4]
        LDRB r3,[r0,#8]
Loop    B     Loop
AAA0    DCB  0x10, 0x20, 0x30, 0x40
AAA1    DCB  0x50, 0x60, 0x70, 0x80
AAA2    DCB  0x90, 0xA0, 0xB0, 0xC0, 0xD0, 0xE0, 0xF0, 0xFF
        END
```

- *R1 = 0x14*
- *R2 = 0x0*
- *R2 = 0x90*
- *R3 = 0xD0*

23. What are the values of `r0`, `r2`, and `r3` *(in hexadecimal)* after executing the following program?

```
        AREA prog,
        CODE,
        READWRITE
        ENTRY
       LDR r0,=AAA1
        LDR r2,[r0,#4]
        LDR r3,[r0,#8]
Loop    B     Loop
AAA0    DCB  0x10, 0x20, 0x30, 0x40
```

```
AAA1  DCB  0x50, 0x60, 0x70, 0x80
AAA2  DCB  0x90, 0xA0, 0xB0, 0xC0, 0xD0, 0xE0, 0xF0, 0xFF
      END
```

- *R1 = 0x14*
- *R2 = 0x0*
- *R2 = 0x90A0B0C0*
- *R3 = 0xD0E0F0FF*

24. What are the values of `r0`, `r1`, `r2`, `r3`, `r4`, `r5`, and `r6`*(in hexadecimal)* after executing the following program?

```
      AREA prog,
      CODE,
      READONLY
      ENTRY
      MOV  r0, #0xC MOV  r1, #0x1
      MOV  r2, #0x8
      MOV  r3, #0x30
      LDRB r4,[r3, -r1, LSL #3]
      LDRB r5,[r3],-r2, ASR #2
      LDRB r6,[r2,  r1, ROR #27]!
      ADD  r1, r0,  r1, LSL r0
Loop B Loop
      DCB  0x11, 0x22, 0x33, 0x44, 0x55
      DCB  0x66, 0x77, 0x88, 0x99, 0xAA
      DCB  0xBB, 0xCC, 0xDD, 0xEE, 0xFF
      END
```

25. What are the values of `r0`, `r1`, `r2`, `r3`, `r4`, `r5`, and `r6`*(in hexadecimal)* after executing the following program?

```
      AREA prog, CODE, READONLY
      ENTRY
      MOV  r0, #0xC MOV  r1, #0x1
      MOV  r2, #0x8
      MOV  r3, #0x30
      LDR r4,[r3, -r1, LSL #3]
      LDR r5,[r3],-r2, ASR #2
      LDR r6,[r2,  r1, ROR #27]!
      ADD  r1, r0,  r1, LSL r0
Loop B Loop
      DCD  0x11, 0x22, 0x33, 0x44, 0x55
      DCD  0x66, 0x77, 0x88, 0x99, 0xAA
      DCD  0xBB, 0xCC, 0xDD, 0xEE, 0xFF
      END
```

Study Questions - #4

(ARM LDR and STR instructions encoding/decoding)

1. Question 3.24 at page 225: 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 = Pointer Adjust (Pre/Increment) = like #+1
U = Pointer Direction (Up/Down) = shift
B = Operand Size (Byte/Word) = small
W = Pointer Update (Write-back) = save
L = Data Direction (Load/Store) = l or s.

2. Write only ONE ARM instruction that writes the value 0x12 in register r6. Encode this ARM assembly instruction to ARM machine language code.

ARM instruction : MOV R6, #0X12

3. **Write only ONE ARM instruction that writes the value 0x124 in register r6. Encode this ARM assembly instruction to ARM machine language code.**

ARM instruction: MOV R6, #0X124

4. Write only ONE ARM instruction that writes the value 0x1248 in register r6. Encode this ARM assembly instruction to ARM machine language code.

ARM instruction: MOV R6, #0X1248

5. Encode the following ARM assembly instruction to ARM machine language code.

```
STREQ r1,[r2]
```

0 0 0 0 0 1 1 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Condition | 0 1| # |P | U | B |W| L | r base | r destin | operand 2 |

Gives:

0000 0110 0100 0010 0001 0000 0000 0010
0x 0 6 4 2 1 0 0 2

6. **Decode the following ARM machine language code to ARM assembly instruction.**
0x05821000

0x 0 5 8 2 1 0 0 0
0000 0101 1000 0010 0001 0000 0000 0000

0 0 0 0 0 1 0 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Condition | 0 1| # |P | U | B |W| L | r base | r destin | operand 2 |

Gives: STREQ, r1, [0]

Study Questions - #5

(Programs and Subroutines)

1. **Question 3.15 at page 224: Write an ARM assembly language routine to count the number of 1s in a 32-bit word in r0 and return the result in r1.**

```
        LDR r0, =2_00000110001000000010010111011101   ;Value with 12 1s
         MOV r2, #32          ;Counter to loop through all 32 bits
        MOV r1, #0    ;Counter of # of 1s
shift   CMP r2, #0            ;Check if all bits have been checked
         SUBNE r2, #1         ;Subtract loop counter
         LSLNES r0, #1         ;Shift bit to Carry flag
         ADDCS r1, #1         ;If carry bit is 1: increment 1-bit counter
        BNE shift             ;Read next
loop    B loop
```

Also:

```
AREA Load, CODE, READONLY
        ENTRY
        ldr r0, =2_01100101101111100101101001101011
        mov r2, #32
```

```
        b loop


loop    TST r0, #1
            addne r1, #1
            lsr r0, #1
            sub r2, #1
            cmp r2, #0
            bne loop

lop b loop
            END
```

3. **Question 3.16 at page 224: A word consists of the bytes b4, b3, b2, b1. Write an ARM assembly language function to re-order (transpose) these bytes in the order b1, b3, b2, b4.**

```
AREA Load, CODE, READONLY
ENTRY
ldr r2, =0xABCDEF98
and r3, r2, #0xFF
and r4, r2, #0xFF00
and r5, r2, #0xFF0000
and r6, r2, #0xFF000000
lsr r6, #24
lsr r5, #8
lsl r4, #8
lsl r3, #24
orr r3, r3, r4
orr r3, r3, r6
orr r1, r3, r5
lop b loop
    END
```

4. **Question 3.22 at page 224: Write ARM code to implement the following C operation.**

```
int s = 0;
for ( i = 0; i < 10; i++) {
s = s + i*i;)
```

Answer:

```
        AREA prog1, CODE, READONLY
        ENTRY

        MOV r0, #0          ; this is the counter
        MOV r1, #0          ; this is s

main    NOP
```

```
        CMP r0, #10
        BNE floop

loop    B loop

        AREA prog1, CODE, READONLY

floop   NOP

        MLA r1, r0, r0, r1
        ADD r0, r0, #1


        B main
        END
Or
  AREA Load, CODE, READONLY
  ENTRY
  mov r0, #0
  mov r1, #0
  b floop
floop   cmp r1, #10
        beq end
        mov r2, r1
        mul r3, r2, r1
        add r0, r0, r3
        add r1, #1
        b floop

end     b end
        END
```

4. **Question 3.38 at page 225: Write suitable ARM code to implement if x = y call PQR else call ZXY**

```
        AREA   s5q3, CODE, READONLY
        ENTRY

        MOV    r0,#0
        MOV            r1,#0

        CMP    r0,r1
        BLEQ   PQR
        BLNE   ZXY
Loop    B              Loop

PQR     MOV            r2,#1    ; dummy op
```

```
        MOV     r15,r14
ZXY     MOV     r3,#1     ; dummy op
        MOV     r15,r14
```

5. **Question 3.39 at page 225: Write an ARM assembly language program that scans a string terminated by the null byte 0x00 and copies the string from a source location pointed at by r0 to a destination pointed at by r1.**

```
        AREA    s5q3, CODE, READWRITE
        ENTRY

        LDR             r0,=String1
        LDR             r1,=Dest
Scan    LDRB    r2,[r0],#1
        CMP             r2,#0
        STRB    r2,[r1],#1
        BNE             Scan
Loop    B               Loop

String1         DCB     "this is a string",0x00
Dest    SPACE   20
```