4.7.3.5 Assembly Language

1) **Figure 1** shows an assembly language program together with the contents of a section of the main memory of the computer that the program will be executed on. Each main memory location and register can store a 16-bit value.

Figure 1

Program
LDR R1, 100
LSL R2,R1,#2
ADD R1,R1,R2
LDR R3, 101
CMP R3,R1
BEQ labela
MOV R4, #0
B labelb
labela:
MOV R4, #1
labelb:
STR R4, 102
HALT

Memory Address (in decimal)	Main Memory Contents (in decimal)
100	10
101	50
102	80

a) Complete the trace table below, **in decimal**, to show how the values stored in the registers and main memory change as the program in **Figure 1** is executed. You may not need to use all the rows.

[4 marks]

Register Contents				Main Memory Location Contents		
R1	R2	R3	R4	100	101	102
10	40					
50		50	1			1

b) Explain what the assembly language program in **Figure 1** does.

[1 mark]

- **Checks if the value stored in memory location 101 is five times larger than the value stored in memory location 100**

2) The pseudo-code in **Figure 2** shows one method for carrying out encryption of a single character using the Caesar Cipher.

If the character to be encrypted is a capital letter, then the encrypted character will be shifted along the alphabet by the number of positions specified by the key. If the character is not a capital letter, then the encrypted character is set to be equal to the original character.

The pseudo-code assumes that the letter to encrypt is stored using the Unicode UTF-8 encoding method, for which the values of capital letters (in decimal) are shown in **Table 2**.

Figure 2

```
IF characterCode >= 65 AND characterCode <= 90 THEN
      encryptedCode ← characterCode + keyValue
      IF encryptedCode > 90 THEN
            encryptedCode ← encryptedCode – 26
      ENDIF
ELSE
      encryptedCode ← characterCode
ENDIF
```

Table 2

A	65
B	66
C	67
D	68
E	69
F	70
G	71
H	72
I	73
J	74
K	75
L	76
M	77

N	78
O	79
P	80
Q	81
R	82
S	83
T	84
U	85
V	86
W	87
X	88
Y	89
Z	90

Figure 3

```
    CMP R1, #65
    BLT doNotEncrypt
    CMP R1, #90
    BGT doNotEncrypt
    ADD R3, R1, R2
    CMP R3, #91
        BLT finished
        SUB R3, R3, #26
        B finished
doNotEncrypt:
    MOV R3, R1
finished:
    HALT
```

a) By analysing the incomplete assembly language program in **Figure 3**, explain the purpose for which R1, R2 and R3 have been used.

[2 marks]

- **R1: The plaintext letter // the characterCode**
- **R2: The key // the keyValue**
- **R3: The ciphertext letter // the encryptedCode**

b) Complete the assembly language program in **Figure 3** by filling the missing code from positions ❶ and ❷ so that it implements the same task as the pseudo-code in **Figure 2**.

[4 marks]

Example above. Note there are multiple possible solutions

3) The greatest common divisor of two positive integers A and B is the largest positive integer that divides both numbers without leaving a remainder.

For example, the greatest common divisor of 4 and 6 is 2.

Below is pseudocode from find the greatest common divisor. When the procedure terminates the value in A (and also B) is the greatest common divisor of A and B.

```
WHILE A ≠ B
      IF A > B THEN
            A = A – B
      ELSE
            B = B – A
      ENDIF
ENDWHILE
```

Write a program in assembly code to calculate the greatest common divisor of two possible integers.

- At the start, the positive integer A will be stored in memory location 102 and the positive integer B in memory location 103. Your program should use these values to find their greatest common divisor.
- When your program terminates it should store the greatest common divisor of these two numbers in memory location 104.

[8 marks]

```
      LDR R1, 102
      LDR R2, 103
loop:
      CMP R1, R2
      BEQ finish
      BGT aGreater
      SUB R2, R2, R1
      B loop
aGreater:
      SUB R1, R1, R2
      B loop
finish:
      STR R1, 104
```

4) A Vernam cipher encrypts a plaintext character by performing a logical operation between a character in the plaintext and part of the key.

Write an assembly language program to encrypt a plaintext character using the method.

- The character code of the plaintext character to be encrypted is stored in memory location 101
- The part of the key to use to encrypt the character is stored in memory location 102

The encrypted ciphertext character should be stored in memory location 103.

[3 marks]

```
LDR R1, 101
    LDR R2, 102
    EOR R3, R1, R2
    STR R3, 103
```

5) **Figure 4** shows an assembly language.

Figure 4

```
      CMP R2, #0
      BEQ exit
      MOV R0, #0
      MOV R3, #1
moveleft:
      LSL R2, R2, #1
      LSL R3, R3 #1
      CMP R2, R1
      BLT moveleft
      BEQ mainloop
      LSR R2, R2, #1
      LSR R3, R3, #1
mainloop:
      CMP R1, R2
      BLT skip
      ADD R0, R0, R3
      SUB R1, R1, R2
skip:
      AND R4, R3, #1
      CMP R4, #1
      BEQ skipshiftR2
      LSR R2, R2, #1
skipshiftR2:
      LSR R3, R3, #1
      CMP R3, #0
      BNE mainloop
exit:
      HALT
```

The program takes its input values from registers R1 and R2 and stores its output in registers R0 and R1.

a) Complete the trace table below to show the results of executing the program in **Figure 4** when the initial values in registers R1 and R2 are 34 and 6.

Each register can hold a 16-bit value.

You may not need to use all rows in the table.

[6 marks]

R0	R1	R2	R3	R4
	100010 (34)	110 (6)		
0(0)			**1 (1)**	
		1100 (12)	**10 (2)**	
		11000 (24)	**100 (4)**	
		110000 (48)	**1000 (8)**	
		11000 (24)	**100 (4)**	
100 (4)				**0(0)**
		1100 (12)	**10 (2)**	**0 (0)**
		110 (6)	**1 (1)**	
101 (5)	**100 (4)**			**1 (1)**
			0 (0)	

b) By considering the inputs and outputs in the trace table, describe the purpose of the program.

[2 marks]

- **It performs integer division**
- **Outputs the quotient in R0 and the remainder in R1**

6) **Figure 5** shows an assembly language.

Figure 5

```
      LDR R0, 120
      LDR R1, 121
      MOV R3, #0
loop:
      CMP R1, #0
      BEQ exit
      AND R2, R1, #1
      CMP R2, ,#0
      BEQ skip
      ADD R3, R3, R0
skip:
      LSL R0, R0, #1
      LSR R1, R1, #1
      B loop
exit:
      STR R3, 122
      HALT
```

a) State the name of the addressing mode used in the instruction ADD R3, R3, R0.

[1 mark]

- **Direct addressing**

b) Memory location 120 contains the value 23 and memory location 121 contains the value 5.

Complete the trace table to show how the contexts of the memory locations and registers change when the program in **Figure 5** is executed.

[5 marks]

Memory locations			Registers			
120	**121**	**122**	**R0**	**R1**	**R2**	**R3**
23	5		**23**	**5**		**0**
					1	**23**
			46	**2**	**0**	
			92	**1**	**1**	**115**
			184	**0**		
		155				

c) State the purpose of the program in **Figure 5**.

[1 mark]

- **To multiply the two numbers in memory locations 120 and 121 together and store the result in memory location 122**

d) The program has been written using assembly language.

State **two** reasons why the programmer may have chosen to write this program in assembly language rather than in a high-level programming language.

[2 marks]

- **So it will execute more quickly**
- **So it will use less memory when translated**
- **A translator for a high-level language might not have been available**
- **The programmer would have complete control over the final machine code that is output by the assembler.**

e) The program will be translated into machine code.

Explain the relationship between an assembly language instruction and a machine code instruction.

[1 mark]

- **The is a one-to-one mapping between an assembly language instruction and a machine code instruction**

7) Assembly language instructions can be used to perform masking, which allows the values of individual bits or groups of bits within a number to be isolated or set independently of the values of the other bits in the number.

For example, to isolate the values of the rightmost four bits of an 8-bit number, the number could be ANDed with the binary value 00001111.

The assembly language instruction AND R0, R1, #15 performs a bitwise logical AND operation between the value in register R1 and the number 15 (equivalent to 00001111 in binary), storing the result in register R0.

a) In binary, show the result of applying the instruction AND R0, R1, #15 when register R1 contains the decimal value 70 which is 46 in hexadecimal.

R1	0	1	0	0	0	1	1	0
15	0	0	0	0	1	1	1	1
R0	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**

[1 mark]

b) In binary, show the result of applying the instruction ORR R0, R1, #48 when register R1 contains the decimal value 6 which is 6 in hexadecimal.

R1	0	0	0	0	0	1	1	0
48	0	0	1	1	0	0	0	0
R0	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**0**

[1 mark]

8) A computer program is required to display the value of the contents of a memory location that stores an 8-bit value. The value should be displayed on the screen of the computer in hexadecimal.

Part of the process required to do this is to convert the value stored in the memory location into the correct.

For example, if the memory location contained:

1	0	0	1	1	1	1	0

which is 9E in hexadecimal, then the ASCII codes of the characters that need to be displayed are:

0	0	1	1	1	0	0	1

0	1	0	0	0	1	0	1

The first of these is the ASCII code of the character 9, the second is the ASCII code of the character E.

Write an assembly language program using the AQA assembly language instruction set that will load a value from memory location 100 and store the ASCII code of the first (lefthand) digit of the hexadecimal representation of this value in memory location 101 and the ASCII code of the second (righthand) digit of the hexadecimal representation of this value in memory location 102.

Your program should use masking and/or shifting to complete this task.

The ASCII codes of the hexadecimal digits are shown in **Table 1** below.

Table 1

	ASCII Code	
Digit	**Decimal**	**Binary**
0	48	0110000
1	49	0110001
2	50	0110010
3	51	0110011
4	52	0110100
5	53	0110101
6	54	0110110
7	55	0110111

	ASCII Code	
Digit	**Decimal**	**Binary**
8	56	0111000
9	57	0111001
A	65	1000001
B	66	1000010
C	67	1000011
D	68	1000100
E	69	1000101
F	70	1000110

[10 marks]

```
        LDR R0, 100
        AND R2, R0, #15
        CMP R2, #10
        BLT isnumber
        ADD R2, R2, #55
        B doleftdigit
isnumber:
        ADD R2, R2, #48
doleftdigit:
        AND R1, R0, #240
        LSR R1, R1, #4
        CMP R1, #10
        BLT isnumber2
        ADD R1, R1, #55
        B storetomemory
isnumber2:
        ADD R1, R1, #48
storetomemory:
        STR R1, 101
        STR R2, 102
```

9) **Figure 6** shows an assembly language.

Figure 6

```
      LDR R1, 130
      MOV R2, #0
      MOV R4, #0
repeat: ADD R2, R2, #1
      AND R3, R1, #1
      CMP R3, #0
      BEQ skip
      ADD R4, R4, #1
skip: LSR R1, R1, #1
      CMP R2, #7
      BNE repeat
      LDR R1, 130
      AND R4, R4, #1
      CMP R4, #0
      BNE else
      ORR R1, R1, #128
      B end
else:
      AND R1, R1, #127
end:
      STR R1, 130
      HALT
```

The program performs a task on a value stored in memory location 130. The value in this memory location is a 7-bit ASCII code.

For example, if memory location 130 was used to store the ASCII character ‘S’ then it would contain the value 83, which in binary is 01010011

a) Complete the trace table below to show the results of executing the program in **Figure 6** when the initial value in memory location 130 is 83

Each register can hold an 8-bit value.

You may find it easier to understand the operation of the program if you write the contents of memory location 130 and register R1 out in both binary and decimal.

You may not need to use all the rows in the table.

[6 marks]

Memory Location 130	R1	R2	R3	R4
83(01010011)	**83(01010011)**	**0**		**0**
		1	**1**	**1**
	41(00101001)	**2**	**1**	**2**
	20(00010100)	**3**	**0**	
	10(00001010)	**4**	**0**	
	5(00000101)	**5**	**1**	**3**
	2(00000010)	**6**	**0**	
	1(00000001)	**7**	**1**	**4**
	0(00000000)			
	83(01010011)			**0**
	211(11010011)			
211(11010011)				

b) The value in memory location 130 before the program is executed is the program's input and the value stored in memory location 130 when the program finishes executing is its output.

Describe the purpose of the program

[2 marks]

- **Sets the parity bit (for the ASCII character); using odd parity;**
- **Counts the number of 1s (in the ASCII code);**

c) Describe **two** advantages of writing programs in assembly language over writing programs using a high-level language

[2 marks]

- **Machine code produced by assembler / assembling may use up less memory // contain fewer instructions than machine code produced by compiler / converting HLL code would**
- **Machine code produced by assembler / assembling may execute more quickly than machine code produced by compiler / converting HLL code would**
- **Assembly language may allow (better) access to hardware / registers / (low-level) operating system routines**

10) **Figure 7** shows a block of code, written in a high-level language, that is used to find out if the number stored in the variable A is even. If the value is even then the number 69_{10} is stored in the variable B, otherwise the number 79_{10} is stored in the variable B.

Figure 7

```
IF IsEven(A)
      THEN B ← 69
      ELSE B ← 79
ENDIF
```

a) Write a sequence of assembly language instructions that would perform the same operations as the high-level language code in **Figure 7**.

Assume that register R1 currently stores the value associated with A, that register R2 stores the value currently associated with B and that register R3 is available for general use, if necessary.

[6 marks]

```
      AND R3, R1, #1
      CMP R3, #1
      BEQ odd
      MOV R2, #69
      B end
odd:
      MOV R2, #79
end:
      HALT
```

b) What addressing mode is being used with the second operand in the assembly language instruction MOV R2, #0?

[1 mark]

- **Immediate**