1.2.4 Assembly Languages & Addressing Modes

1. Below is part of a program written using the Little Man Computer instruction set. This section of code will exit to the code labelled pass or fail depending on the values in the accumulator.

```
test SUB ten
     BRZ pass
     BRP test
     BRA fail
ten  DAT 10
```

a)

i. Explain what the line ten DAT 10 does.

[3 marks]

- **Stores the value 10**
- **… in a memory location**
- **… given by the label ten**

ii. Complete the table below determining whether the program branches to pass or fail given the following values in the Accumulator when it is run.

Starting value in Accumulator	pass or fail
29	**fail**
30	**pass**
31	**fail**

[3 marks]

b) The complete program is shown below:

```
      INP
main  STA  entry
      BRA  test
fail  LDA  entry
      ADD  one
      BRA  main

test  SUB  ten
      BRZ  pass
      BRP  test
      BRA  fail

pass  LDA  entry
      OUT
      HLT

entry DAT
ten   DAT  10
one   DAT  1
```

i. Give one instruction in the program that when executed, changes the value in the Accumulator.

[1 mark]

- **LDA / SUB / ADD / INP**

ii. Give one instruction in the program that when executed, changes the value in the program counter.

[1 mark]

- **BRA / BRZ / BRP**

iii. State the value the code outputs for the input 18.

[1 mark]

- **20**

iv. State the value the code outputs for the input 37.

[1 mark]

- **40**

v. Describe the purpose of the program.

[2 marks]

- **Rounds up the number input**
- **… to the nearest multiple of ten and outputs it**

2. Write a program using the little man instruction set that will allow a user to input two numbers and then output the larger of the two numbers. The program should loop continuously.

[6 marks]

```
start INP
      STA x
      INP
      STA y
      BRP first
      LDA x
      OUT
      BRA start
first LDA y
      OUT
      BRA start
x     DAT
y     DAT
```

3. The following is a program written using the Little Man Computer instruction set.

```
start LDA one
      OUT
      LDA zero
      OUT
      LDA count
      SUB one
      STA count
      BRP start
      HLT
one   DAT 1
zero  DAT 0
count DAT 3
```

a) Describe the difference between the STA and LDA instructions.

[2 marks]

- **STA store the value in the accumulator into a given memory location.**
- **LDA loads the value in a memory location into the accumulator.**

b) Describe what the instruction BRP does.

[2 marks]

- **The program flow jumps to a label/another point in the program**
- **… if the value in the accumulator is positive or zero**

c) Identify the type of memory addressing the program uses.

[1 mark]

- **Direct Addressing**

d) State the output this program generates.

[3 marks]

- **1 0 1 0 1 0 1 0**

e) State what type of translator program would be needed to convert the code above into machine code.

[3 marks]

- **Assembler**

4. A Little Man Computer (LMC) assembly language program is stored in memory as shown below.

```
0   LDA &7
1   ADD #4
2   OUT
3   HLT
4   6
5   2
6   10
7   15
8   16
9   17
```

In this variant of LMC the symbols & and # are used to denote different modes of addressing.

a) Given that the output is 17, state the addressing mode represented by each symbol.

i. &

[1 mark]

- **Immediate addressing**

ii. #

[1 mark]

- **Indirect addressing**

b) Explain how pipelining would help a CPU execute the code in Fig. 3.1 more quickly.

[3 marks]

- **Pipelining would allow one instruction to be fetched as the previous one is being decoded and the one before that is being executed.**
- **For example, OUT could be fetched, while ADD #4 is decoded and LDA &7 is executed.**
- **As there are no jump/branch instructions it pipelines well as there is no need to flush the pipeline.**

5. In assembly language, different modes of addressing memory can be used.

Discuss the different modes used. You should include:

- How the operand value is determined
- What an operand of 31 would refer to in that mode
- The reasons for requiring multiple modes of addressing

[12 marks]

Knowledge

- **Immediate addressing is where the operand holds the actual data to be used; no address referenced**
- **Direct addressing is where the operand holds the address that holds the data to be used.**
- **Indirect addressing is where the operand holds an address which is where the data to be used is stored**
- **Indexed addressing is where the operand holds an address which is offset using the Index Register to find the true address of the data to be used**

Application

- **Immediate addressing; operand of 27 would refer to the value 27**
- **Direct addressing; operand of 27 would tell the processor to fetch the data held at address 27.**
- **Indirect addressing; operand of 27 would tell the processor to fetch the data held at address 27, which itself would then be used an address to fetch data from.**
- **Indexed addressing; operand of 27 would be added to the value of the Index register and this would then be used as the address to fetch data from.**

Evaluation

- **Immediate addressing allows simple access to data with no fetch required but limited by the data size of the operand.**
- **Direct addressing allows data to be fetched from memory. Data can potentially be larger in size that with immediate addressing, but address range limited by size of operand.**
- **Indirect addressing allows a larger range of addresses to be accessed as address fetched. However, multiple fetches required to access data.**
- **Indexed addressing allows the Index register to be manipulated to access data stored sequentially e.g. in an array.**