THE ANSWERS

A: Analysis, B: Bookwork, D: Design

1. a) Convert the 32 bit IEEE-754 format number 0x3E300000 to decimal, showing your method.

[4]

B. 0.171875
s = 0 (1 mark)
exp= 124, exponent = -3 (1 mark)
frac = $0.011_{(2)} = 0.375_{(10)}$ (1 mark)
$1 \times 1.375 \times 2^{-3} = 0.171875$ (1 mark)

b) A 32 bit ARM CPU has a direct mapped cache of size 1024 lines each of 32 words.

i) What is the total data size of the cache not including tag storage. Give your answer in bytes.

[1]

ii) Which ARM address bit fields constitute the tag, select and index fields of this cache?

[3]

iii) State, giving your working, what is the total number of bits required to store tag values in the cache?

[2]

iv) The CPU performs a sequence of contiguous word READ operations such that the *n*th operation has address 4**n*. What is the ratio of misses to hits if this sequence continues for a long time? Is your answer affected by the number of lines in the cache?

[4]

1024 lines each of 32 words means 32K words or 128K (131072) bytes.
1024 lines ⇒ 10 bits index. 32 words (128 bytes) ⇒ 7 bits select. Therefore tags are 32-10-7=15 bits long.
A(31:17) tag A(16:7) index A(6:0) select
The cache has 1024 lines so total of 15*1024=1536 bits used for tag storage.
There is one miss per line and 31 hits, hence the ratio is 1/31. The number of lines has no affect.

c) Write a fragment of ARM code which sets R0 equal to the least significant 32 bits of 16*R1 + 16385*R2. All values are unsigned. R1 and R2 must not be changed. Credit will be given for efficient implementations that use the minimum number of registers. You may find it useful to note that 16384 is a power of 2.

[4]

```
ADD   R0, R2, R2, lsl #14
ADD   R0,R0, R1, lsl #4
```

d) State, giving reasons, whether your code in part c) would change given the same requirement on *two's complement signed* values of R0, R1 and R2 given that R0 must still equal the least significant 32 bits of the answer.

[2]

No change, because addition is identical for unsigned and signed and the truncation means that overflows - which would be different - do not matter.

Question 1. has a total of 20 marks.

2. a) A fragment of assembly code containing loops is shown in Figure 2.1.

i) For each branch in the code state whether it is escape, looping, or skipping.

[4]

ii) State the number of times that LOOP1 executes.

[1]

iii) For a single execution of LOOP1, state the number of times that LOOP2 executes and LOOP3 executes.

[2]

iv) When this code fragement is executed from START through to FINISH the *execution count* of each instruction is defined to be the number of times it is condition true executed. State the largest execution count of any instruction in this code fragment, and the set of instructions that have this maximum execution count.

[3]

B, E, H are looping branches. G is escape branch. There are no skipping branches. LOOP1 executes 5 times.
LOOP2 executes 100 times.
LOOP3 executes 101 times.
Therefore the maximum executed instructions are inside LOOP3. The branch is condition true executed once less than the others, so the maximum set is {LOOP3,D} and is executed 505 times.

b) This part concerns an ARM block copy subroutine which has inputs R0, R1, R2. The subroutine copies R2 words of memory:

$$\text{mem}_{32}[\text{R0}+i] \rightarrow \text{mem}_{32}[\text{R1}+i] \quad \text{where } 0 \leq i \leq \text{R2}-1$$

You may assume that the source and destination blocks of memory do not overlap and that both R0 and R1 are word-aligned pointers. You may assume that R2 is non-zero and positive.

i) Write an ARM code fragment that performs this block move, with inputs R0,R1,R2 as specified. Credit will be given for code which has a small number of instructions. Load/Store Multiple (LDM/STM) instructions may not be used.

[4]

ii) Suppose the source and destination blocks of memory overlap as shown in Figure 2.2. Explain in words how a block copy loop would need to be written to work in this case.

[2]

iii) Write the subroutine entry and exit code such that no register value is changed by the subroutine, assuming a stack which moves upwards on PUSH and has stack pointer R13 pointing to an empty location.

[4]

```
LOOP   LDR    R3, [R0,R2,LSL #2]
       STR    R3, [R1,R2,LSL #2]
       SUBS   R2, R2, #1
       BPL    LOOP
```

The loop would need to copy words moving downwards through addresses.

```
                ;entry
LDMEA           R13!, R2
                ; exit
STMEA           R13!, R2
MOV PC, R14
```

Question 2. has a total of 20 marks.

```
START   MOV   R2, #4
LOOP1   MOV   R1, #100
A       MOV   R0, R1
LOOP2   SUBS  R0, R0, #1
B       BNE   LOOP2
C       MOV   R0, R1
LOOP3   CMP   R0, #0
D       SUB   R0, R0, #1
E       BNE   LOOP3
F       SUBS  R2, #1
G       BMI   FINISH
H       B     LOOP1
FINISH
```

Figure 2.1: Assembler Loop.

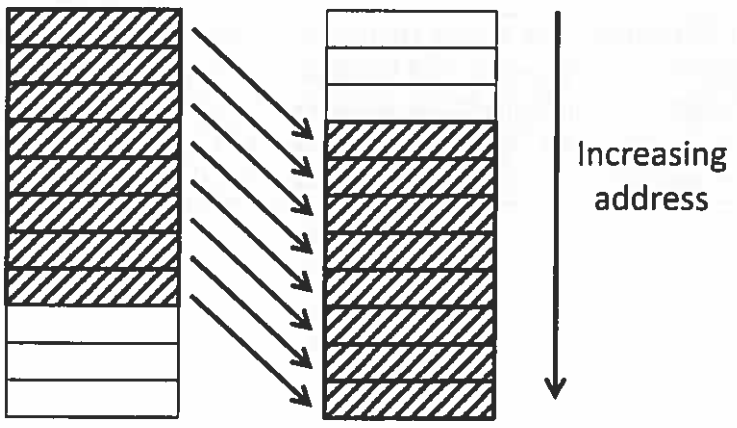

Figure 2.2: Overlapping source and destination blocks.

This question requires the analysis and detailed understanding of a new example of code, together with the writing of modified code (last part)

3. The code shown in Figure 3.1 has input R1 and output R2. You may assume that R1 has precisely 1 non-zero bit and lies in the range $1 \leq R1 \leq 2^7$. The label TABLE points to a table of constant data as shown in Figure 3.1, where each number in the DCB statement determines the value of one 8 bit byte. The code outputs in R2 the number of the bit set to 1 in the binary reprepresentation of R1. Throughout this question the least significant bit is numbered 0.

   a) Showing all your working, state how many cycles are taken by this code. Explain why it is advantageous to use LDRNEB rather than LDRB for the instruction with label S1.

   [4]

   ADR instruction takes one cycle (PC relative addressing).
   LDRNEB is 3 cycles shorter than LDR in the case that it is not executed. Exactly one of the two ANDS instructions will return NE condition, since only 1 bit is non-zero. So one of the LDRs takes 4 cycles. Note that the ADD instruction takes one cycle regardless of condition.
   (2 marks for description)
   Therefore total time is: 1 + 1 + 4 + 1 + 1 + 1 = 9 cycles. (2 marks for time)

   b) If the value in R1 is written as $16a + b$, where $0 \leq a, b \leq 15$ and as before R1 has 1 non-zero bit, determine the memory addresses used in the instructions with labels S1 and S2 as functions of a, b and TABLE. Show your reasoning.

   [4]

   R10 contains TABLE
   first AND sets R0 = b
   S1: effective address is: TABLE + b
   second AND sets R0 = $a \times 16$
   S2: lsr 4 → division of offset by 16 so the effective address is TABLE + $(a \times 16)/16$ = TABLE + a

   c) Explain how the algorithm works and state which of the TABLE data locations, if any, have values on which the algorithm correctness does *not* depend.

   [4]

   Either S1 or S2 will be condition true executed but not both. Each LDR looks up in the same 16 word table to determine the number of the bit set in the 4 bit fields a,b. For a (S2) the ADD correctly adds 4 to the the value returned because a is multipled by 2^4 to obtain a bit in R1. If the bit actually set is in b the two instructions processing a do not execute (NE from second ANDS) and the result is that got from b. Byte offsets 2^n in TABLE must be set to n ($0 \leq n \leq 3$) respectively. all other words are don't care.

d) Write a new version of the code in Figure 3.1 using a different table of length 16 bytes and modified code which returns in R2 a number equal to the number of bits set to 1 in the binary representation of R1, where R1 is known to satisfy $0 \leq R1 \leq 255$. Explain clearly your method.

[8]

In the answer 50% credit is given for correct method, and 50% for correct code:
Table: 1+1
Masks: 1+1
Addition: 1+1
Correct initialisation and data flow: 1+1

```
        ADR     R10, TABLE
        MOV     R2, #0
        ANDS    R0, R1, #0xF
S1      LDRB    R2, [R10,R0]
        ANDS    R0, R1, #0xF0
S2      LDRB    R3, [R10,R0,LSL #4]
        ADD     R2, R2, R3#4
TABLE   DCB     0,1,1,2,1,2,2,3,1,2,2,3,2,3,3,4
```

The TABLE data (length 16 bytes) must have the byte with offset n equal to the number of bits set to 1 in the binary representation of n.

Question 3. has a total of 20 marks.

```
        ADR     R10, TABLE
        ANDS    R0, R1, #0xF
S1      LDRNEB  R2, [R10, R0]
        ANDS    R0, R1, #0xF0
S2      LDRNEB  R2, [R10, R0, LSR #4]
        ADDNE   R2, #4

TABLE   DCB     0, 0, 1, 0, 2, 0, 0, 0, 3
```

Figure 3.1: Code.