Paper Number(s): **E2.7B**

(= E1.9 sec A)

IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE
UNIVERSITY OF LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2002

EEE PART II: B.Eng., M.Eng. and ACGI

PRINCIPLES OF COMPUTERS AND SOFTWARE ENGINEERING

Monday, 10 June 2:00 pm

There are THREE questions on this paper.

Answer TWO questions.

Corrected Copy

Q3

This exam is OPEN BOOK.

Time allowed: 1:30 hours.

Examiners responsible:

First Marker(s): Cheung,P.Y.K.
Second Marker(s): Demiris,Y.K.

Information for Invigilators:

Students may bring any written or printed aids into the examination.

Information for Candidates:

None.

1. Consider the following code fragment in ARM assembly language.

```
        MOV     r1, #0
        MOV     r0, #10
LOOP1   STR     r0, [r1], #4
        SUBS    r0, r0, #1
        BNE     LOOP1
        MOV     r1, #0
        MOV     r0, #5
LOOP2   LDR     r2, [r1, #20]
        LDR     r3, [r1]
        ADD     r2, r2, r3
        STR     r2, [r1], #4
        SUBS    r0, r0, #1
        BNE     LOOP2
```

a) Write down an order list of memory locations, which are accessed by this code fragment, showing the memory address and data, and whether it is a read or a write access.

[8 marks]

b) Assuming that the microprocessor takes 100ns per clock cycle, all instructions with and without data memory access take 2 and 1 clock cycles respectively, state how long this code fragment will take to execute.

[2 marks]

c) Assume that the microprocessor uses 32 bytes of direct-mapped cache for data only, and each cache line is 4 bytes. Further assume that the entire data cache is dirty at the start of the code fragment. How many memory accesses result in cache 'hit' and cache 'miss' respectively when this code fragment is executed?

[7 marks]

d) As a result of using cache in the microprocessor, each clock cycle is shortened to 10ns. The cache miss penalty is 120ns. How long will this code fragment take to execute as a result of using cache?

[3 marks]

2. Run-length coding is a method of compression where repeated data values are represented by a repeat count (i.e. the length of the run) followed by the data value itself. For example a sequence of byte values (in hexadecimal)

```
4A 4A 4A 4A 4A 4A 09 09 09 00 A7 A7 A7 A7 69 01
```

is compressed to:

06 4A **03** 09 **01** 00 **04** A7 **01** 69 . . . .

The repeat count value has a maximum value of 255 and the data value are from 0 to 255.

a) Write a subroutine `RunLength` in ARM assembly language for the following specification:

```
; Subroutine RunLength - run-length compress a block of data stored as bytes
;
;   Input parameters:        r1 - starting address of data to be compressed
;                            r2 - starting address of output buffer where
;                                 compressed data is to be stored
;                            r3 - no of bytes to be compressed
;   Return parameters:       None
;
; The output format should be:
;     <repeat_count> <byte_value> <repeat_count> <byte_value> . . . . .
```

[10 marks]

b) An alternative run-length encoding rule is given below:

i) If (datavalue = 0) or (run-length > 3), encode it as

<00> <repeat_count> <byte_value>

ii) For all other situations, the data are left as they are (i.e. no encoding is applied).

Therefore, the above byte sequence will be encoded as:

00 06 4A 09 09 09 **00 01** 00 **00 04** A7 69 . . . .

Modify the subroutine in a) to implement this encoding rule.

[10 marks]

3. The following ARM code fragment processes the characters in a NULL-terminated string. In order to use the code, r0 should point to the start of the string.

```
Loop        LDRB        r1, [r0], #1
            CMP         r1, #0
            BEQ         finished
            CMP         r1, #'A'
            BLT         loop
            CMP         r1, #'Z'
            BGT         loop
            SUB         r2, r1, #'A'-'a'
            STRB        r2, [r0, #-1]
            B           loop
finished
```

a) What is the effect of executing the above code on a string?

[3 marks]

b) Re-write the above code to make it into a subroutine called "TL" that could be called from the program below as shown. Use an "empty decreasing" stack.

```
            AREA        prog, CODE, READONLY
SWI_Exit    EQU         &11
            ENTRY
            MOV         r1, #0
            MOV         r2, #5
            ...
L1          ADR         r0, string
            BL          TL
            SWI         SWI_Exit
string      = "Hello World!", 0x0a, 0x0d, 0
            END
```

[6 marks]

c) In the program shown above, the value of label L1 is 0x8080 and the stack pointer has value 0x1000 before entry into the subroutine. State and justify the value of the link register during execution of subroutine TL.

[3 marks]

d) Draw a diagram showing the numerical addresses and numerical contents of the stack immediately after pushing the necessary data onto the stack. (Assume that no intervening code marked "..." alters either register r1 or register r2).

[4 marks]

e) You are provided with a subroutine "printc" which prints the character in register r2 to a connected peripheral device. An example use is shown below.

```
            MOV         r2, #'A'
            BL          printc
```

Re-write your subroutine so that it also calls printc for each character of the modified string

[4 marks]

E2.76

2002

Answer to Question 1

a)

Address (hex)	Data (hex)	R/W	hit/miss (for part c.)
0000	0000 000A	W	Miss
0004	0000 0009	W	Miss
0008	0000 0008	W	Miss
000C	0000 0007	W	Miss
0010	0000 0006	W	Miss
0014	0000 0005	W	Miss
0018	0000 0004	W	Miss
001C	0000 0003	W	Miss
0020	0000 0002	W	Miss
0024	0000 0001	W	Miss
0014	0000 0005	R	Hit
0000	0000 000A	R	Miss
0000	0000 000F	W	Hit
0018	0000 0004	R	Hit
0004	0000 0009	R	Hit
0004	0000 000D	W	Hit
001C	0000 0003	R	Hit
0008	0000 0008	R	Hit
0008	0000 000B	W	Hit
0020	0000 0002	R	Hit
000C	0000 0007	R	Miss
000C	0000 0009	W	Hit
0024	0000 0001	R	Hit
0010	0000 0006	R	Miss
0010	0000 0007	W	Hit

[8 marks]

b) 89 cycles @ 100ns = 8.9 microseconds.

[2 marks]

c) 14 'miss', 11 'hit' (see table above).

[7 marks]

d) 89 x 10ns + 14 x 110 ns = 2.43 microseconds.

[3 marks]

Answer to Question 2

a)

```
RunLength   STMED     r13!, {r0-r6, r14}  ; preserve context
            ADD       r6, r1, r3       ; r6 has last address of buffer + 1
Start_loop  MOV       r4, #1           ; r4 counts the run-length
            LDB       r5, [r1], #1     ; fetch a byte
loop2       CMP       r1,r6            ; if reached terminating address
            BCS       finished         ;     finished,
            CMP       r4, #$ff         ;   else if run-length is maximum
            BEQ       end_run          ;     output current data
            LDB       r0, [r1], #1     ;  else get the next byte
            CMP       r0, r5           ; if not the same,
            BNE       end_run          ;     terminate run and output
            ADD       r4, r4, #1       ; else  increment run-length count
            B         loop2            ; loop back for another test
end_run     MOV       r4, [r2], #1     ; output run-length
            MOV       r5, [r2], #1     ; output data value
            B         start_loop       ; loop back for more
finished    LDMED     r13!, {r0-r6, pc}
            END
```

[10 marks]

b)

```
RunLength2  STMED  r13!, {r0-r6, r14} ; preserve context
            ADD    r6, r1, r3         ; r6 has last address of buffer + 1
start_loop  MOV    r4, #1             ; r4 counts the run-length
            LDB    r5, [r1], #1       ; fetch a byte
loop2       CMP    r1,r6              ; if reached terminating address
            BCS    finished           ;     finished,
            CMP    r4, #$ff           ;   else if run-length is maximum
            BEQ    end_run            ;     output current data
            LDB    r0, [r1], #1       ;  else get the next byte
            CMP    r0, r5             ; if not the same,
            BNE    end_run            ;     terminate run and output
            ADD    r4, r4, #1         ; else  increment run-length count
            B      Loop2              ; loop back for another test
;
;  so far same as before
;
end_run     CMP    r5, #0             ; if data is zero, run-length encode
            BEQ    run_encode
            CMP    r4, #03            ; else if run-length > 3
            BHI    run_encode         ;   encode it,
no_encode   MOV    r5, [r2], #1       ; else just output data
            SUB    r4, r4, #1         ;   … the required no of times
            BNE    no_encode
            B      start_loop         ; loop back for more
;
;  if gets here, run-length encode
run_encode  MOV    r0, #0             ; 0 is special code
            MOV    r4, [r2], #1       ; output run-length
            MOV    r5, [r2], #1       ; output data value
            B      start_loop         ; loop back for more
finished    LDMED  r13!, {r0-r6, pc}
            END
```

[10 marks]

Answers to Question 3

This question tests the students understanding of stacks and subroutine calls in assembly language.

a) This code converts any upper-case characters in the string to their equivalent lower-case characters. Any other characters remain unchanged. The modified string overwrites the original string.

[3 marks]

b) One possible solution is shown below.

```
TL        STMED r13!, {r0, r1, r2}
loop      LDRB      r1, [r0], #1
          CMP       r1, #0
          BEQ       ret
          CMP       r1, #'A'
          BLT       loop
          CMP       r1, #'Z'
          BGT       loop
          SUB       r2, r1, #'A'-'a'
          STRB      r2, [r0, #-1]
          B         loop
ret       LDMED r13!, {r0, r1,r2}
          MOV       pc, r14
```

Two marks for PUSHing r0, r1 and r2, two marks for POPing r0, r1 and r2 back in the correct order. One mark for using the correct pair (STMED, LDMED) of stack instructions. Whether r14 is pushed or whether lr is moved into pc doesn't matter – award one mark for each of these solutions. Deduct one mark per unnecessary register PUSHed or POPed.

[6 marks]

c) ADR instruction has address 0x8080, BL instruction has address 0x8084, SWI instruction has address 0x8088. The link register (r14) will therefore hold the value 0x8088 during execution of subroutine TL.

[3 marks]

d) Answers will vary depending on solution to (b), but for the solution given above:

Address	Data
0x1000	0x0005
0x0FFC	0x0000
0x0FF8	0x808C

One mark for correctly recognizing an EMPTY stack, one mark for correctly recognizing a DECREASING stack. One mark for recognizing that addresses differ by 4 bytes. One mark for ordering the data in the correct way.

[4 marks]

e) This question tests nested subroutines. The key modification necessary is to store the link register. One possible solution is shown below

```
TL      STMED r13!, {r0, r1, r2, r14}
loop    LDRB    r1, [r0], #1
        CMP     r1, #0
        BEQ     ret
        CMP     r1, #'A'
        BLT     print
        CMP     r1, #'Z'
        BGT     print
        SUB     r2, r1, #'A'-'a'
        STRB    r2, [r0, #-1]
print   BL      printc
        B       loop
ret     LDMED r13!, {r0, r1, r2, r14}
        MOV     pc, r14
```

One mark for inserting the BL instruction, one mark for recognizing the need to save and one mark for recognizing the need to restore the link register. One mark for printing ALL characters of the modified string (not just the modified characters)

[4 marks]