(ISE 1)

Paper Number(s): **E1.9**

IMPERIAL COLLEGE OF SCIENCE, TECHNOLOBY AND MEDICINE
UNIVERSITY OF LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2002

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

COMPUTER SYSTEMS

Monday, 10 June 2:00 pm

There are FIVE questions on this paper

Answer TWO questions from Section A and ONE question from Section B

Section A is open-book
Section B is closed-book

Use a separate answer book for each section

Corrected Copy

See Q3

Time allowed:	Section A	1:30 hours
	Section B	1:00 hours

Examiners responsible:

First marker(s):	Cheung, P.Y.K, Demiris, Y.K.
Second marker(s):	Demiris, Y.K., Shanahan, M.S.

Section A
(Please use a separate answer book for each Section.)

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
            BE          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]

E1 9 **Corrected Copy**

Section B
Use a separate answer book for each section.

Answer ONLY ONE of the following two questions

QUESTION 1:

(i) Describe the round-robin process scheduling algorithm and list its advantages and disadvantages [4]

(ii) For the following set of processes with their corresponding duration, arrival times and priority levels [higher number indicates a higher priority]:

a. Show the order of execution (including timing information) of the processes if the scheduler implements the following scheduling algorithms:
   i. Shortest remaining job first (SRJF) [2]
   ii. Priority scheduling without pre-emption [2]
   iii. Priority scheduling with pre-emption [2]

b. For each of the algorithms calculate the average waiting time, and the average turnaround time. [2]

Process	Arrival time (ms)	Duration (ms)	Priority level
A	0	2	4
B	2	5	2
C	4	1	5
D	7	5	6

(iii) Describe the Optimal, First-In-First-Out (FIFO), and Least-Recently-Used (LRU) page replacement algorithms, and list their advantages and disadvantages. [7]

(iv) In the context of memory management, describe the condition known as "thrashing" [1]

QUESTION 2:

(i) When is a set of processes deadlocked? [2]

(ii) In the context of deadlock avoidance, describe what it means for a system state to be "*safe*". Describe the difference between an *unsafe* state and a *deadlock* state. [2]

(iii) A system has 14 instances of a resource type and there are currently four processes running; their maximum needs and their current allocation are shown in the table below.

a. Determine whether the current state is a safe state, and show why. [2]

b. Assume that the system is using banker's algorithm for dynamic deadlock avoidance. Given the current state below, determine the algorithm's response [i.e. grant or refuse request] for the following allocation requests. Explain your answer.

   i. Process C requests 1 instance [2]

ii. Process B requests 4 instances [2]
iii. Process D requests 4 instances [2]

	Max	Current
Process A	5	2
Process B	14	2
Process C	4	2
Process D	12	3
Free: 5		

(iv) In the "producer-consumer" problem, two processes are communicating through a buffer that can hold *0* to ***n*** items. The producer process continuously produces items and places them in the buffer, while the consumer continuously fetches items from the buffer and consumes them. The following conditions are in place:

a. The producer process can only place items in the buffer if there is space available in the buffer – it blocks otherwise.
b. The consumer process can fetch items from the buffer only if there are items available – it blocks otherwise.
c. Mutual exclusion is required: the producer and the consumer processes cannot access the buffer at the same time.

Using semaphores to ensure that the conditions above hold, provide Pascal procedures for the producer and consumer processes. Declare and properly initialise all semaphores you use. The data type *Semaphore*, and the standard semaphore primitives *init(Sem, number), wait(Sem),* and *signal(Sem)* are available. You may assume that the following procedures are also available: produce_item, write_item, get_item, consume_item, and that the constant *"**n**"* has been declared. [8]

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]

E1.9

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]

E1.9

Y. Demiris
29.8.02

E1.9 – section B: Operating Systems
Model answers to exam questions 2002

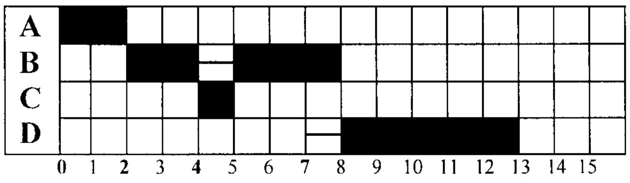
Question 1:

(i) The Round robin scheduling algorithm allocates the CPU to a process for a *time quantum* (or *time slice*). If the process is still running at the end of the quantum, it is pre-empted and CPU is given to the next process in the ready queue. The preempted process is put at the end of the queue. New processes are also added at the end of the queue.

Advantages: Simple to implement; fair.

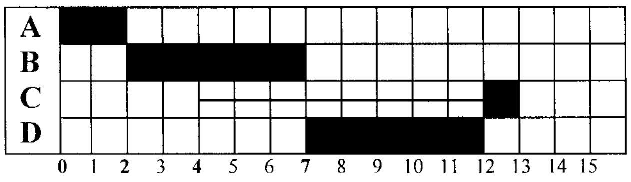
Disadvantages: Difficult to determine appropriate time quantum
- too small: good response time, but large overheads
- too large: bad response time.

(ii) SRJF



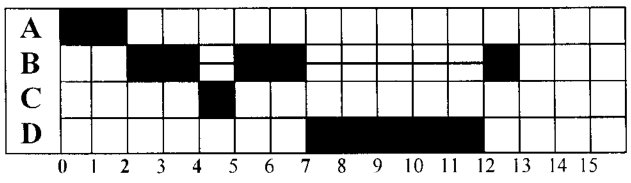
Average waiting time: (0+1+0+1) / 4 = 0.5 ms
Average turnaround time: (2+6+1+6) / 4 = 15 / 4 = 3.75 ms

Priority Scheduling (without pre-emption)



Average waiting time: (0+0+8+0) / 4 = 2 ms
Average turnaround time: (2+5+9+5) / 4 = 21 / 4 = 5.25 ms

Priority Scheduling (with pre-emption)



Average waiting time: (0+6+0+0) / 4 = 1.5 ms
Average turnaround time: (2+11+1+5) / 4 = 19 / 4 = 4.75 ms

(iii) (a) Optimal page replacement algorithm: replaces the page that will not be used for the longest period of time.
Advantages: Lowest page-fault rate of all algorithms
Disadvantages: Difficult to impossible to implement – we need to know in

P. 5 of 7

advance the stream of page requests.

(b) FIFO: replaces the page that has been in memory for the longest time.
Advantages: easy to understand and implement (FIFO queue)
Disadvantages: sub-optimal performance – does not account for usage of pages.

(c) LRU: replaces the page that has not been used for the longest time.
Advantages: good performance
Disadvantages: not easy to implement

(iv) Thrashing: the processor spends more time swapping memory pieces than executing instructions.

Question 2:

(i) A set of processes is deadlocked if each of the processes in the set is waiting for an event (e.g. a resource to become available) that only another process in the set can cause.

(ii) A state is safe if the system can allocate resources to each of the processes (up to the maximum declared by that process) in some order, and avoid a deadlock. An unsafe state is NOT a deadlock state; it *may* lead to it.

(iii) (a) Current state is safe – resources can be allocated in a specific order (e.g. A->C->D->B, note that there are other sequences too) so system can avoid a deadlock.
(b) -I: It will grant the request; the resulting state is safe (A->C->D->B)
-II: It will refuse the request since it leads to an unsafe state
-III: It will refuse the request since it leads to an unsafe state

(iv) Producer-consumer problem

```
var item, space, mutex: Semaphore;
init(item,0);  init(space,n);  init(mutex,1);
```

Producer process:

```
Procedure producer()
Begin
  While(TRUE) do
    Begin
      Produce_item;
      Wait(space);
      Wait(mutex);
      Write_item;
      Signal(mutex);
      Signal(item);
    End;
End;
```

Consumer process:

```
Procedure producer()
Begin
  While(TRUE) do
    Begin
      Wait(item);
      Wait(mutex);
      get_item;
      Signal(mutex);
      Signal(space);
      Consume_item;
     End;
End;
```

[illegible]