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

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2004

PRINCIPLES OF COMPUTERS AND SOFTWARE ENGINEERING: SECTION A

Wednesday 9th June 2004 2:00pm

There are THREE questions on this paper.

Answer TWO questions.

This exam is OPEN BOOK.

Corrected Copy

Q.2

Time allowed: 1:30 hours.

Any special instructions for invigilators and information for candidates are on page 1.

Examiners responsible:

First Marker(s): Constantinides, G.A
Second Marker(s): Demiris,Y.K.

E19/E27 B (a)

SECTION A

Special Information for Invigilators:
This section of the examination is open book. Candidates may bring any *written* or *printed* material to the examination.

Information for Candidates
Throughout this section of the paper, the notation "0x" before a number means that the number is expressed using hexadecimal representation.

The Questions

1.

A subroutine `scramble` is shown below.

```
scramble
        STMED       r13!, {r0-r3}
        MOV         r3, #0
loop
        LDRB        r2, [r0], #1
        ADD         r2, r2, r3
        CMP         r2, #'Z'
        SUBGT       r2, r2, #('Z'-'A')
        ADD         r3, r3, #1
        CMP         r3, #('Z'-'A')
        MOVGT       r3, #0
        STRB        r2, [r0,#-1]
        SUBS        r1, r1, #1
        BNE         loop
        LDMED       r13!, {r0-r3}
        MOV         pc, lr
```

a) Consider the following instructions. For each one, state which registers, memory locations, and flags may be modified as a result of execution.

(i) `LDRB r2, [r0], #1`

(ii) `STRB r2, [r0,#-1]`

(iii) `SUBGT r2, r2, #('Z'-'A')`

(iv) `SUBS r1, r1, #1`

(v) `STMED r13!, {r0-r3}`

[7]

b) Describe the purpose of the link register.

[2]

c) Assuming that on entry `r0` points to a message consisting of upper-case characters, what is the function of subroutine `scramble`?

[2]

Prior to subroutine entry, `r0` has the value 0x8100 and `r1` has the value 0x3. A partial content listing of the memory is shown in Table 1, below.

Table 1

Address	Data
0x8100	ASCII encoding of 'A'
0x8101	ASCII encoding of 'B'
0x8102	ASCII encoding of 'C'

d) Provide a partial content listing of the memory after subroutine execution, listing all locations where that data has changed.

[3]

e) Modify the code so that any spaces in the original message are left unscrambled.

[6]

2.

Given a block of N memory words, a length-2 moving sum filter finds the arithmetic sum of the values of 2 consecutive memory words, as illustrated in Figure 1. This process is repeated a total of N-1 times, with the starting location of the window of 2 locations changing by one memory word per iteration.

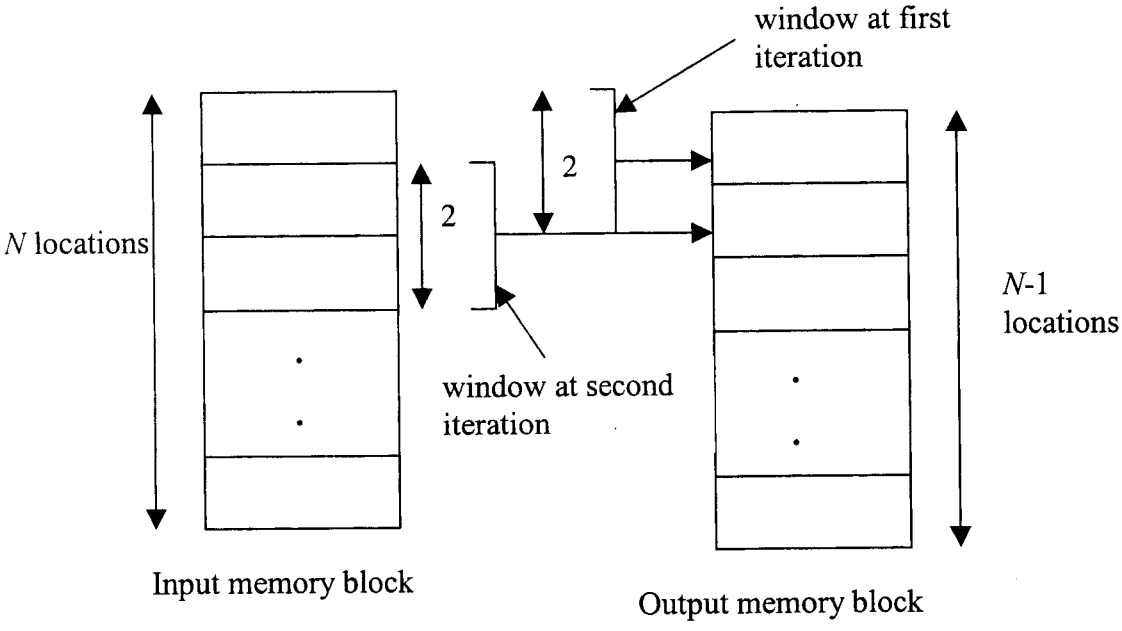


Figure 1

a) Write a subroutine called `movingavg` to implement this moving average filter. The subroutine should take three arguments: `r0` should contain the starting address of the input block of memory, `r1` should contain the starting address of the output block, and `r2` should contain N.

[10]

b) In general, a length-K moving sum filter finds the arithmetic sum of the values of K consecutive memory words. This process is repeated a total of N-K+1 times, with the starting location of the window of K locations still changing by one memory word per iteration.

Extend your subroutine to this general case. The subroutine should now have four arguments: `r0` should contain the starting address of the input block of memory, `r1` should contain the starting address of the output block, `r2` should contain N, and `r3` should contain K.

[10]

3.

a) Assemble the following sequence of ARM instructions into (binary or hex) machine code.

```
loop    LDR     r2, [r0], #4
        ADD     r2, r2, #1
        CMP     r2, #0
        STRGT   r2, [r0,#-4]
        BGT     loop
        SWI     0x11
```

[10]

The address of the first instruction is 0x8000, and `r0`=0x1000 immediately before entering this code fragment. A partial content listing of the memory is shown in Table 2, below.

Table 2

Address	Data
0x1000	0x00000100
0x1004	0x00000200
0x1008	0x00000000

b) Write a time-ordered list of instruction fetch accesses for this code. For each memory access, state the address of the word accessed, whether the access is a read or write, and the data read or written (in hex).

[NB: You may assume that the processor is not pipelined]

[2]

c) Write a time-ordered list of memory data accesses performed by this code. For each memory access, state the address of the word accessed, whether the access is a read or write, and the data read or written (in hex).

[2]

d) It is proposed to use both an instruction cache and a separate data cache to speed up the execution of this code fragment. There are to be 4 lines in each cache, each of one word.

Assuming the caches are initially empty, draw a diagram illustrating the cache contents after the access sequence above has completed. For each cache line, include the tag, the valid bit, and the data.

[4]

e) For each cache, state the number of hits and misses caused by this execution.

[2]

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

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2004

PRINCIPLES OF COMPUTERS AND SOFTWARE ENGINEERING: SECTION B

Wednesday 9th June 2004 2:00pm

CORRECTED COPY

There are TWO questions on this paper.

Answer ONE question.

Time allowed: 1:00 hour.

Any special instructions for invigilators and information for candidates are on page 1.

Examiners responsible:

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

E1.9 – SECTION B: Operating Systems [1 hour]

Answer ONLY ONE of the following two questions

QUESTION 1:

(a) Describe the pre-emptive version of the priority-based scheduling algorithm and list its advantages and disadvantages. [3]

(b) Consider the following set of processes, with their corresponding duration, arrival times, and priority levels [*higher numbers indicate higher priority*]:

Process	Arrival time (ms)	Duration (ms)	Priority level
A	0	2	3
B	2	4	2
C	3	2	1
D	4	3	4
E	8	2	5

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) [3]
(ii) Priority-scheduling without preemption [3]
(iii) Round-Robin with a time quantum of 5ms. [3]

For each of the algorithms calculate the average waiting time, and the average turnaround time.

(c) Two processes A & B have critical regions Critical_A and Critical_B. Provide code for implementing Peterson's software solution to the mutual exclusion problem (i.e. not allowing Critical_A & Critical_B to be executed at the same time) for the two processes A and B. [6]

(d) In the context of interprocess synchronization, describe the concept of race conditions. [2]

QUESTION 2:

(a) Describe the four conditions that must be present for a deadlock to occur. [4]

(b) In the context of a memory paging system, consider the following scenario:

- You have four available frames
- The reference string is 3-1-2-2-5-6-7-3-5-2

Starting with empty frame contents, show the sequence of frame contents after each request, and count the number of page faults for each of the following page replacement algorithms:

(i) Optimal page replacement [3]
(ii) FIFO (first-in, first out) page replacement [3]
(iii) LRU (Least Recently Used) page replacement [3]

(c) Describe how a least-recently-used page-replacement algorithm could be implemented using

i. Counters [2]
ii. A stack [2]

(d) Using pseudo-code, describe the function of the three semaphore primitives init(S, number), wait(S) and signal(S), given the following definition for the semaphore data type:

```
Type semaphore: record
     Counter: int;
     Queue: list of processes
End
```
[3]

E1.9 – Principles of Computers & Software Engineering Sections A + B. Model answers – 2004

E1.9(a)
E2.7 B

PRINCIPLES OF COMPUTERS – MODEL ANSWERS

1. a)

INSTR	REGS	MEM	FLAGS
(i)	r0, r2	[r0]	NONE
(ii)	r2	[r0 - 1]	NONE
(iii)	r2	NONE	NONE
(iv)	r1	NONE	ALL
(v)	r13	[r13] & [r13 - 15]	NONE

b) TO STORE THE RETURN ADDRESS FROM A SUBROUTINE CALL (BL INSTRUCTION)

c) IT ENCODES THE MESSAGE. IF P_i DENOTES PLAINTEXT CHARACTER i, C_i DENOTES CIPHERTEXT CHARACTER i, AND COUNTING OF CHARACTERS STARTS FROM ZERO, THEN

$$C_i = i + P_i$$

WITH WRAP-AROUND FROM Z → A.

d)

ADDRESS	DATA
0x8101	ASCII ENCODING OF 'C'
0x8102	ASCII ENCODING OF 'E'

e)

```
scramble
        STMED   r13!, {r0-r3}
        MOV     r3, #0
loop
        LDRB    r2, [r0], #1
        CMP     r2, #'A'
        BEQ     skip
        ADD     r2, r2, r3
        CMP     r2, #'Z'
        SUBGT   r2, r2, #('Z' - 'A')
        ADD     r3, r3, #1
        CMP     r3, #('Z'- 'A')
        MOVGT   r3, #0
        STRB    r2, [r0, #-1]
skip    SUBS    r1, r1, #1
        BNE     loop
        LDMED   r13!, {r0-r3}
        MOV     pc, lr
```

2. a)

```
movingavg   STMED   r13!, {r0-r4}
            SUB     r2, r2, #1      ; N-1 iterations
Loop        LDR     r3, [r0], #4
            LDR     r4, [r0]
            ADD     r3, r3, r4      ; form sum
            STR     r3, [r1], #4
            SUBS    r2, r2, #1
            BNE     Loop
            LDMED   r13!, {r0-r4}
            MOV     pc, lr
```

b)

```
movingavg   STMED   r13!, {r0-r6}
            SUB     r3, r3, #1  ; more convenient to keep K-1
            SUB     r2, r2, r3  ; r2 holds N-K+1 now
Loop1       MOV     r6, #0      ; r6 holds running total
            MOV     r5, r3, LSL #2 ; offset for sum term (in bytes)
Loop2       LDR     r4, [r0, r5]
            ADD     r6, r6, r4  ; running total
            SUBS    r5, r5, #4  ; next term
            BPL     Loop2       ; positive or zero offset => more terms
            STR     r6, [r1], #4 ; final result
            ADD     r0, r0, #4  ; start of window
            SUBS    r2, r2, #1  ; sample update
            BNE     Loop1
            LDMED   r13!, {r0-r6}
            MOV     pc, lr
```

3. a)

LDR	r2, [r0], #4	→	0x E4902001	← (1)
ADD	r2, r2, #1	→	0x E2822001	← (2)
CMP	r2, #0	→	0x E3520000	← (3)
STRGT	r2, [r0, #-4]	→	0x C5002001	← (4)
BGT	Loop	→	0x CAFFFFFA	← (5)
SWI	0x11	→	0x EF000011	← (6)

b)

ADDRESS	READ/WRITE	DATA	MISS/HIT [FOR PART e.]
0x8000	R	(1) ABOVE	M
0x8004	R	(2) "	M
0x8008	R	(3) "	M
0x800C	R	(4) "	M
0x8010	R	(5) "	M
0x8000	R	(1) "	M
0x8004	R	(2) "	H
0x8008	R	(3) "	H
0x800C	R	(4) "	H
0x8010	R	(5) "	M
0x8000	R	(1) "	M
0x8004	R	(2) "	H
0x8008	R	(3) "	H
0x800C	R	(4) "	H
0x8010	R	(5) "	M
0x8014	R	(6) "	M

c)

ADDRESS	READ/WRITE	DATA	MISS/HIT [FOR PART e]
0x1000	R	0x100	M
0x1000	W	0x101	H
0x1004	R	0x200	M
0x1004	W	0x201	H
0x1008	R	0x000	M

d) INSTRUCTION CACHE

CACHE LINE #	TAG	VALID	DATA
0	0x801	✓ (Y)	0x CAFFFFFA
1	0x801	✓ (Y)	0x EF000011
2	0x800	✓ (Y)	0x E3520000
3	0x800	✓ (Y)	0x C5002001

DATA CACHE

CACHE LINE #	TAG	VALID	DATA
0	0x100	✓ (Y)	0x101
1	0x100	✓ (Y)	0x201
2	0x100	✓ (Y)	0x000
3	?	X (N)	?

3. e) INSTRUCTION CACHE: 10 MISSES / 6 HITS

DATA CACHE: 3 MISSES / 2 HITS

E1.9 – (section B): Operating Systems
Model answers to exam questions 2004

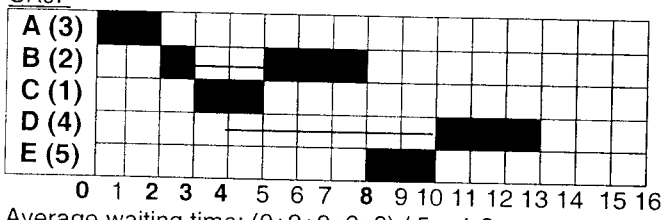
Question 1

(i) [bookwork] A priority is associated with each process; CPU is allocated to the process with the highest priority. FCFS is used to resolve situations involving processes with equal priority.

Advantages: Takes into account external factors regarding the importance of the various processes
Disadvantages: Might result to the "starvation" of low priority processes

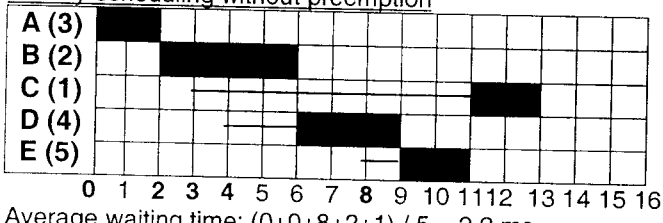
(ii) [new computed example]

SRJF



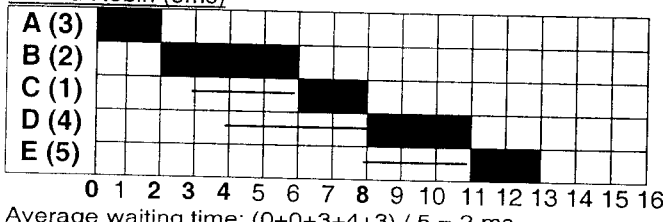
Average waiting time: (0+2+0+6+0) / 5 = 1.6 ms
Average turnaround time: (2+6+2+9+2) / 5 = 21 / 5 = 4.2 ms

Priority-scheduling without preemption



Average waiting time: (0+0+8+2+1) / 5 = 2.2 ms
Average turnaround time: (2+4+10+5+3) / 5 = 24 / 5 = 4.8 ms

Round Robin (5ms)



Average waiting time: (0+0+3+4+3) / 5 = 2 ms
Average turnaround time: (2+3+5+7+5) / 5 = 4.4 ms

(iii) [Bookwork]

Turn is a character variable; Interested_A and Interested_B are boolean variables initially set to FALSE;

```
Interested_A = TRUE;
Turn = 'B';
While (interested_B = TRUE
        AND Turn = 'B')
                Do_nothing;
Critical_A;
Interested_A = FALSE;
```

```
interested_B = TRUE;
Turn = 'A';
 while (interested_A = TRUE)
        AND Turn = 'A')
                Do_nothing;
Critical_B;
interested_B = FALSE:
```

(iv) [Bookwork]: Race conditions are situations in IPC that two or more processes are reading or writing some shared data, and the final result depends on who runs precisely when.

Question 2:

(a) [bookwork] Four conditions must be present for a deadlock to occur:
- *(1) Mutual exclusion:* only one process may use a resource at a time.
- (2) *Hold and Wait:* A process may hold allocated resources while awaiting assignments of others
- (3) No pre-emption: No resource can be forcibly removed from a process holding it.
- (4) Circular wait: A closed chain of processes exist, such that each process holds at least one resource needed by the next process in the chain.

(b) [new computed example]

Optimal page replacement algorithm (6 page faults)

	3	**1**	**2**	**2**	**5**	**6**	**7**	**3**	**5**	**2**
Frame1	**3**	3	3		3	3	3			
Frame2	**-**	1	1		1	6	7			
Frame3	**-**	-	2		2	2	2			
Frame4	**-**	-	-		5	5	5			

FIFO page replacement algorithm (8 page faults)

	3	**1**	**2**	**2**	**5**	**6**	**7**	**3**	**5**	**2**
Frame1	3	3	3		3	6	6	6		6
Frame2	**-**	1	1		1	1	7	7		7
Frame3	**-**	-	2		2	2	2	3		3
Frame4	**-**	-	-		5	5	5	5		2

LRU (Least recently used) page replacement algorithm (8 page faults)

	3	**1**	**2**	**2**	**5**	**6**	**7**	**3**	**5**	**2**
Frame1	**3**	3	3		3	6	6	6		2
Frame2	**-**	1	1		1	1	7	7		7
Frame3	**-**	-	2		2	2	2	3		3
Frame4	**-**	-	-		5	5	5	5		5

(c) [bookwork]
- (1) [Counters] A global counter gets updated with every memory reference; each page has a counter associated with it. When a reference to a page is made, the contents of the global counter are copied to the associated counter. LRU algorithm searches through the pages and picks the one with the lowest counter value.
- (2) [Stack] A stack of page numbers is kept; whenever a page is references, it is removed from the stack and placed on the top. Therefore, the top of the stack is the most recently used page, bottom of the stack is the LRU page.

(d)

```
Procedure Init(var S: semaphore)
  s.count = 1;   s.queue = EmptyList;
end

Procedure wait(var s: semaphore)
s.count = s.count – 1;
if s.count < 0 then
        add process in s.queue
block process;
end;

Procedure signal(var s:semaphore)
s.count = s.count+1;
if s.count <= 0 then
  remove next process from s.queue
  place process in the ready queue
end;
```