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

IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE
UNIVERSITY OF LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2000

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

**PRINCIPLES OF COMPUTERS AND SOFTWARE ENGINEERING**

Friday, 16 June 2000, 2:00 pm

There are FIVE questions on this paper.

There are two sections. Answer THREE questions including at least ONE question from each section.

Use a separate answer book for each section.

This is an open book examination.

Time allowed: 2:00 hours

**Corrected Copy**

Q4

Examiners: Dr M.P. Shanahan, Mr P.Y.K. Cheung, Dr J.V. Pitt

## SECTION A

*Use a separate answer book for each section*

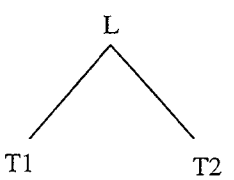
1. a) Write Pascal type definitions for a binary tree of linked lists of integers.

**[4 marks]**

b) Using the type definitions from Part (a), write a function called Largest that takes a tree and returns the largest integer it contains. You may assume a function Max(X,Y) that returns the greater of the two integers X and Y, and that all the integers in the tree are positive.

**[8 marks]**

c) Modify the type definition from Part (a) so that the length of each list in the tree is recorded along with the list. Using this modified data type, write a procedure MakeTree that takes two sub-trees T1 and T2 and a list of integers L and returns the following tree.




Make sure that the new length field is maintained.

**[8 marks]**

2. Here is the Pascal type declaration for a dynamic linked list of strings.

```
type
        TList = ^TLink;
        TLink =
        record
                First : string;
                Rest : TList;
        end;
```

a) The function `member` is supposed to take a string `S` and a list `L` and return `true` if `S` occurs in `L` but `false` otherwise. What is wrong with the following attempt to write this function? How will the function behave if called with `S = "Fred"` and `L = ["Mary", "John"]`? How will it behave given the input `S = "Mary"` and `L = ["Mary", "John"]`?

```
function member(S : string; L : TList): boolean;
var Found : boolean;
begin
        Found := false;
        while (L^.First <> S) and (L <> nil) do
        begin
                if L^.First = S
                then Found := true
                else L := L^.Rest;
        end;
        member := Found;
end;
```

**[7 marks]**

b) Here's another attempt at the same function. What's wrong with this one? How will it behave given the same inputs as for part (a)?

```
function member(S : string; L : TList): boolean;
begin
        if L <> nil
        then begin
                repeat
                        L := L^.Rest;
                until (L = nil) or (L^.First = S);
        end;
        member := L <> nil;
end;
```

**[7 marks]**

c) Write two working versions of the member function - one using recursion, and one without recursion.

**[6 marks]**

## SECTION B

*Use a separate answer book for each section*

**All data values and addresses are given in hexadecimal and answers are expected in this format.**

3. The following program, in ARM assembly language, is loaded into memory starting at the address 0x00008080.

```
1.    ;
2.                  AREA  prog1, CODE, READONLY
3.    SWI_Exit      EQU   &11                    ; system call to exit
4.                  ENTRY
5.                  ADR   r1, stk_ptr
6.                  LDR   r13, [r1]
7.                  LDR   r0, [r1, #4]
8.                  BL    SUB1
9.                  SWI   SWI_Exit               ; end of execution
10.   ;
11.   stk_ptr       DCD   0x80f0
12.   const         DCD   0x335577aa
13.   ;
14.   ; Subroutine SUB1
15.   ;
16.   SUB1          STMED r13!,{r1,r14}
17.                 EOR   r1, r0, r0, ROR #16
18.                 BIC   r1, r1, #0xff0000
19.                 MOV   r0, r0, ROR #8
20.                 EOR   r0, r0, r1, LSR #8
21.                 LDMED r13!, {r1, pc}
22.                 END
```

a) List the values of all the labels: `SWI_Exit, stk_ptr, const, SUB1`.

**[3 marks]**

b) State and justify the contents of registers `r0, r1, r13` and `r14` immediately after the `BL` instruction (line 8) is completed.

**[5 marks]**

c) Draw a diagram showing the contents of the stack immediately after the `STMED` instruction is completed.

**[3 marks]**

d) List the values of registers `r0, r1, r13` and `r15` after the processor steps through each instruction in the subroutine `SUB1`.

**[7 marks]**

e) What function does `SUB1` perform?

**[2 marks]**

4. The subroutine `div10` takes a 32-bit unsigned integer `v` in `r0` and divides it by 10. It returns the quotient `v/10` in `r0` and the remainder in `r1`. This subroutine is already written and is available for use.

a) Using the subroutine `div10` or otherwise, write a subroutine `utoa` in ARM assembly language to convert a 32-bit unsigned integer into its equivalent decimal representation as a string of ASCII characters. For example, the number `0x0000ffff` is converted to the ASCII string "`0000065536`".

5

15:15 pm.

On entry to the subroutine `utoa`, r0 contains the 32-bit unsigned integer to be converted, `r1` points to a 10-byte buffer in memory where the resulting ASCII string is to be stored. The least significant digit is stored in the lowest memory address.

**[7 marks]**

b) Write a program to test the subroutine `utoa` by converting the number `0x0000ffff`, and writing the ASCII string to the console window. You may assume that the system call `SWI_WriteC` is available.

**[7 marks]**

c) Modify your subroutine in a) so that leading '0' characters are stored as `space` characters.

**[6 marks]**

5. a) Table 1 compares the cache architecture of two competing microprocessors:

*Table 1: Comparison of cache architecture of two processors*

| Features | Processor K (500MHz) | Processor P (500MHz) |
|---|---|---|
| L1 cache | • on-chip, full-speed<br>• 32kb direct-mapped instruction cache<br>• 32kb write-back, dual-port direct-mapped data cache | • on-chip, full-speed<br>• 16kb direct-mapped instruction cache<br>• 32kb write-through, 2-way set associative data cache |
| L2 cache | • on-chip, half speed<br>• 256kb unified, 4-way set-associative cache | • on-chip, full-speed<br>• 128kb unified direct-mapped cache |
| L3 cache | • off-chip, at 100MHz front-side bus speed, unified direct-mapped | • none |

Compare and contrast these two microprocessors based on the information given in Table 1.

**[10 marks]**

b) Explain briefly any THREE of the following features of a bus bridge and memory controller circuit for interfacing between a microprocessor and the PCI bus:

i) Supports up to five PCI masters

ii) Zero wait state PCI master and slave burst transfer

iii) PCI to system memory data streaming up to 132Mbyte/sec

iv) Symmetric arbitration between host and PCI bus

v) Complete programmable interrupt priorities of PCI modules

**[6 marks]**

c) Explain the purpose of Translation Look-aside Buffers (TLBs) in a Memory Management Unit. What is the implication on system performance if TLB is not present?

**[4 marks]**

July 00

E2.7 Principles of Computers and Software Engineering

2000

1/7

## Solution to Question 1

a)

**[4 marks]**

```
type
   TList = ^TLink;
   TLink =
   record
      First : integer;
      Rest : TList;
   end;

   TTree = ^TNode;
   TNode =
   record
      Node : TList;
      Left : TTree;
      Right : TTree;
   end;
```

b)

**[8 marks]**

```
function Largest(T : TTree): integer;
var X, Y, Z : integer;
begin
   if T = nil
   then Largest := 0
   else begin
      X := Largest(T^.Left);
      Y := Largest(T^.Right);
      Z := MaxInList(T^.Node);
      Largest := Max(X,Max(Y,Z));
   end;
end;
function MaxInList(L : TList): integer;
begin
   if L = nil
   then MaxInList := 0
   else MaxInList := Max(L^.First,
      MaxInList(L^.Rest));
end;
```

c)

**[8 marks]**

TTree becomes

```
type
   TTree = ^TNode;
   TNode =
   record
      Node : TList;
      Lngth : integer;
      Left : TTree;
      Right : TTree;
   end;
function MakeTree(L : TList; T1,T2 : TTree): TTree;
var T : TTree;
begin
   N := Length(L);
   new(T);
   T^.Left := T1;
   T^.Right := T2;
   T^.Node := L;
   T^.Lngth := Length(L);
end;
```

```
function Length(L : TList): integer;
begin
   if L = nil
   then Length := 0
   else Length := Length(L^.Rest) + 1;
end;
```

**Solution to Question 2**

a)

**[7 marks]**

(**not** Found) **and** (L <> Empty) should be (L <> Empty) **and** (**not** Found). As it stands, the function will crash if S doesn't occur in L, because it will call First(L) with L = nil.
So with S = "Fred" and L = ["Mary", "John"], function will crash. With S = "Mary" and L = ["Mary", "John"], function works fine and returns true.

b)

**[7 marks]**

Function misses out first element of list, because repeat loop doesn't check condition until after loop executed.
So with S = "Fred" and L = ["Mary", "John"], function works fine and returns false. With S = "Mary" and L = ["Mary", "John"], function also returns false, incorrectly.

c)

**[6 marks]**

Non-recursive version:

```
function member(S : string; L : TList): boolean;
var Found : boolean;
begin
        Found := false;
        while (L <> Empty) and (not Found) do
        begin
                if First(L) = S
                then Found := true
                else L := Rest(L);
        end;
        member := Found;
end;
```

Recursive version:

```
function member(S : string; L : TList): boolean;
begin
        if L = Empty
        then member := false
        else if First(L) = S
        then member := true
        else member := member(S,Rest(L));
end;
```

**Solution to Question 3**

This question tests students' ability to walk-through a simple assembly language program with good understanding of: addressing modes, stack operations, assembly language syntax, arithmetic operations, function of the barrel-shifter, subroutine calls etc..

a)

**[3 marks]**

| | |
|---|---|
| SWI_exit | = 0x11 |
| stk_ptr | = 0x00008094 |
| const | = 0x00008098 |
| SUB1 | = 0x0000809c |

b)

**[5 marks]**

| | | |
|---|---|---|
| r0 | = 0x335577aa | contains the input value to this subroutine |
| r1 | = 0x00008094 | initialized in the main program to this value |
| r13 | = 0x000080f0 | stack pointer which is intialized to this value |
| r14 | = 0x00008090 | contains the return address |

c)

**[3 marks]**

The stack looks like this:

| | **Addr** | **Value** |
|---|---|---|
| | 0x000080f0 | 0x00008090 |
| | 0x000080ec | 0x00008094 |
| **r13** ----------→ | 0x000080e8 | (previous value – unchanged) |

d)

**[7 marks]**

This shows contents of registers AFTER each instructions:

| **Instr** | **r0** | **r1** | **r13** | **r15** |
|---|---|---|---|---|
| STMED | 335577aa | 00008094 | 000080e8 | 000080a0 |
| EOR | - | 44ff44ff | - | 000080a4 |
| BIC | - | 4400eeff | - | 000080a8 |
| MOV | aa335577 | - | - | 000080ac |
| EOR | aa775533 | - | - | 000080b0 |
| LDMED | - | 00008094 | 000080f0 | 00008090 |

e)

**[2 marks]**

This subroutine reverses the byte order of the 32-bit number passed through r0.

5/7

## Solution to Question 4

### (a)

**[7 marks]**

```
;
; Subroutine utoa
;
utoa    STMED   r13!, {r1-r3, r14}     ; save all used registers
        MOV     r2, r1          ; r2 = buffer address
        MOV     r3, #10         ; do 10 times
_loop
        BL      div10           ; r0 = r0/10, remainder in r1
        ADD     r1, r1, #'0'    ; convert remainder into ascii
        STRB    r1, [r2], #1    ; store in buffer, update pointer
        SUBS    r3, r3, #1
        BNE     _loop           ; end_do
        LDMED   r13!, {r1-r3, pc}      ; pop and return
```

### (b)

**[7 marks]**

```
        AREA    prog2, CODE, READONLY
SWI_WriteC      EQU     0
SWI_Exit        EQU     &11             ; finish program
        ENTRY
        ADR     r1, const
        LDR     r0, [r1], #4            ; fetch value and r1 -> str_buf
        BL      utoa
        ADD     r1, r1, #9              ; r1 points to MSD
        MOV     r2, #10
loop    LDRB    r0, [r1], #-1           ; output character
        SWI     SWI_WriteC
        SUBS    r2, r2, #1
        BNE     loop
        SWI     SWI_Exit                ; end of execution
;
const   DCD     0x0000ffff
str_buf %       10                      ; reserve 10 bytes of storage
```

### (c)

**[6 marks]**

```
;
; Subroutine utoa
;
utoa    STMED   r13!, {r1-r4, r14}     ; save all used registers
        MOV     r2, r1          ; r2 = buffer address
        MOV     r3, #10         ; do 10 times
        MOV     r4, #1          ; r4 = 0 means done!
_loop
        BL      div10           ; r0 = r0/10, remainder in r1
        CMP     r4, #0          ; if r4=1,
        ADDNE   r1, r1, #'0'    ;   convert remainder into ascii
        MOVEQ   r1, #' '        ; else store space
        STRB    r1, [r2], #1    ; store in buffer, update pointer
        CMP     r0, #0          ; if quotient = 0,
        MOVEQ   r4, #0          ;    ... r4=0
        SUBS    r3, r3, #1
        BNE     _loop           ; end_do
                LDMED           r13!, {r1-r4, pc}              ; pop and return
```

**Solution to Question 5**

This question is mostly book work, but students are expected to understand the concept and apply them to features found in commercially available processors and chip-sets.

(a)

**[10 marks]**

**L1 cache:**
Both on-chip, full-speed means that both processors will operating very efficiently if data is in L1 cache.

Proc-K has more instruction cache, therefore will enjoy better hit rate than Proc-P.

The 32kb dual-port direct-mapped data cache of Proc-K may not be as efficient as the 32kb 2-way set-associative data cache.

Write-back cache is important and much better than the write-through cache of Proc-P.

**L2 cache:**
Proc-K has larger L2 cache, but slower speed. Therefore it has higher miss penalty if L1 cache missed.

The full-speed L2 cache of Proc-P means that the smaller L1 cache is less significant. Unfortunately, Proc-P used a direct-mapped cache which is less effective than the 4-way set-associative cache of Proc-K.

**L3 cache:**
Proc-K has L3 cache at a reasonable speed. Therefore L2 cache miss does not incur the full cost of reading from DRAM.

Proc-P has no L3 cache, therefore L2 miss will cost. Therefore although L2 cache is fast, it is likely to have lower performance than Proc-K.

(b)

**[6 marks]**

i) Five separate PCI slots, each can have an I/O module that arbitrate for the control of the bus.

ii) Transfer between PCI master and anther PCI module acting as slave can proceed with effectively one data word per clock cycle more or less continuously.

iii) PCI bus can transfer data directly to main memory at full-speed without the interference of the processor (via DMA).

iv) Neither a module on the PCI bus acting as bus master, nor the microprocessor (called the host) can monopolise any of the buses at the exclusion of others.

v) All the PCI modules can be programmed to have different interrupt priorities. It implies that a central interrupt priority arbitration is used instead of daisy chaining.

7/7

(c)

**[4 marks]**

TLBs are caches for the address page tables which is used by the MMU to translate logical addresses into physical addresses. If TLB is not present, each memory reference involve at least two memory access: the first to read the translation address (which is then used to compute the physical address), and the second perform the actual memory read/write. This is obviously very inefficient. Therefore without TLB, not virtual memory or memory management can be used.