# UNIVERSITY OF LONDON
## IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

## EXAMINATIONS 1996

MSc Degree in Computing Science
for Internal Students of the Imperial College of Science, Technology and Medicine

*This paper is also taken for the relevant examinations for the Diploma of Membership of Imperial College*

PAPER COMP II

COMPUTER SYSTEMS AND PROGRAMMING
Friday, May 3rd 1996, 10.00 - 12.00

*Answer THREE questions*
*Answer at least ONE question from Section A*
*Answer at least ONE question from Section B*

For admin. only: paper contains
5 questions
5 pages (excluding cover page)

**Section A** *(Use a separate answer book for this Section)*

1 This question relates to a computer system based around a Motorola M68000 micro-processor and your answers should use this system as an exemplar. The relevant system connections of the micro processor and memory components of this system are

a 16 bit data bus

a 24 bit address bus

2M Bytes of random access memory built from 2 x 1M Byte RAM chips

2M Bytes of read only memory, also built from 2 x 1M Byte ROM chips

a i) What is the maximum real address space of this processor?

ii) Outline, by way of an annotated diagram, how the 4 memory chips might be connected to the address and data buses of the system.

iii) Draw an appropriate memory map for your answer to part (ii)

b The next instruction to be fetched and executed by the micro processor is the following

```
ADD #1234, D3
```

i) How would this instruction be represented in memory? (You are NOT expected to reproduce the exact bit pattern, but should clearly show which group of bits represent which part of the instruction).

ii) In note form, describe the movement of information both internally and externally to the micro processor for the complete fetch execute cycle for this instruction.

*The two parts carry, respectively, 40% and 60% of the marks.*

2 The first two parts of this question are related specifically to the M68000 microprocessor. For these parts of this question you should use the M68000 as your example. However, you are NOT expected to remember exact mnemonics of syntax as long as it is clear what you intend.

a i) Describe the operation of the LINK and UNLK instructions.

ii) Describe the operation of the JSR and RTS instructions.

b Give a sequence of M68000 code that would implement the following Pascal procedure...

```
procedure swap(var This, That: integer);

var

  Temp: integer;

begin

Temp := This;

This := That;

That := Temp

end;
```

c Floating point numbers can be represented by the following three components

$m \times r^e$

Where **m** is the Mantissa, **r** is the Radix and **e** is the Exponent.

Clearly explain how these three parts are represented by the IEE standard for floating point representations. In your description you should include justification for the representations of the various components.

*The three parts carry, respectively, 30%, 35% and 35% of the marks.*

*Turn over ...*

3a By means of an example, explain what is meant by a *pointer variable* in Pascal. When are such variables generally used? Use your example to explain the effect of the standard procedures `new` and `dispose`.

b For the purposes of this question, an *expression tree* is a tree in which each node has exactly zero or two subtrees. This special case of tree is one which often occurs in many computing applications. For example a Pascal expression of the form:

```
A := b * c - d
```

can be represented by the expression tree shown in **figure 1**.

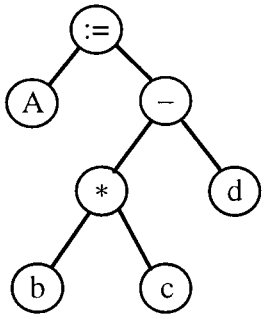



**figure 1**

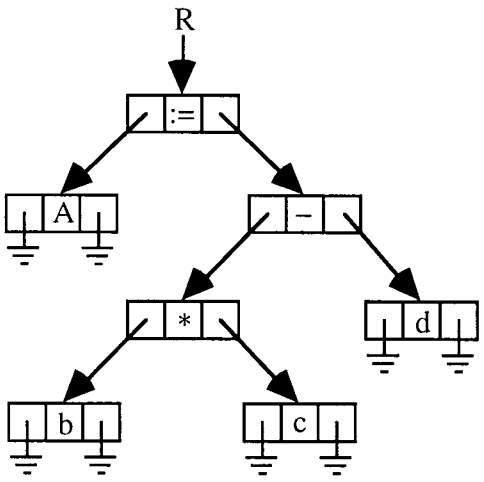



**figure 2**

Give a type declaration `ExprTree` in Pascal for an expression tree using pointers and records to represent structures such as the one shown in **figure 2.** Include a `Data` field of type `Token` to store the data at each node (i.e. `A`, `:=`, etc.)

c Write a recursive procedure :

```
PrintExprTree(Tree : ExprTree)
```

that traverses an expression tree and prints the values of the data for each node. The recursive algorithm should be *symmetric order traversal*, (i.e. first traverse the left subtree, then visit the root and finally traverse the right subtree). Also, to print the data for each node, assume the existence of a procedure:

```
Print(NodeData : Token).
```

d Using the same data structure for expression trees as in (c), we can write an iterative algorithm (non-recursive) for symmetric traversal. However, in this version we need a stack to hold pointers. A pointer is pushed on the stack when the node it points to is first encountered in the traversal; the pointer is popped off when we return to that node from a lower level of the tree. Along these lines, implement an iterative version of the procedure `PrintExprTree`. You may find it useful to implement the stack as an array of `ExprTree` pointers.

*The four parts carry , respectively , 20%, 20%, 30% and 30% of the marks.*

**Section B** *(Use a separate answer book for this Section)*

4 This question concerns the simple kernel that was presented in lectures.

a i) Describe briefly the functions of the Time and Delay system calls.

ii) Suppose the system clock runs at 50 ticks (clock interrupts) per second. The following procedure is intended to be the code for a process that displays the elapsed real time in seconds, updating it every second.

```
procedure p1;
var n:integer;
begin
    n := 0;
    while true do begin
        display(n);     {a procedure call to display
                            n on the screen}
        n := n+1;
        Delay(50)
    end
end p1;
```

Explain why this will fail to behave as intended.

iii) Rewrite the procedure so that it displays as nearly as possible at one second intervals, with no cumulative error.

b i) Explain carefully why the idle process is always either running or ready, and how this knowledge is used in the scheduling procedure (Dispatch).

ii) Part i) can break down if interrupt handlers make certain system calls. Explain how, and which system calls can cause this problem.

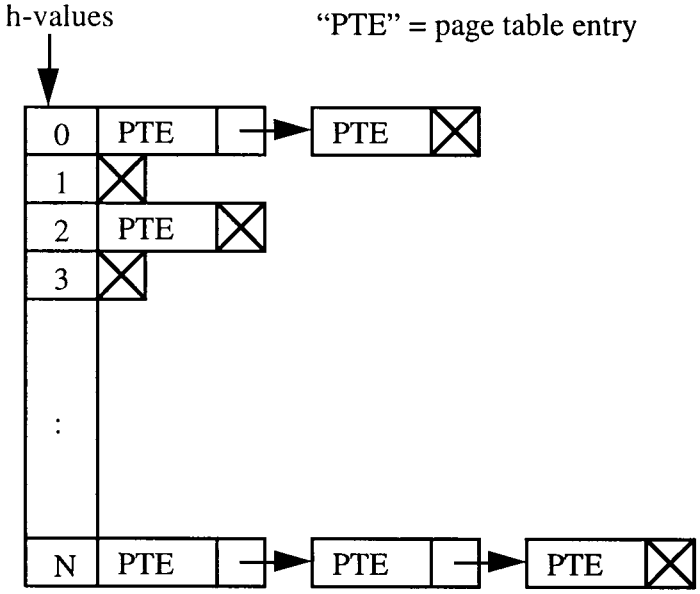
*Turn over ...*

5a i) Explain the difference between *absolute* addressing and addressing *relative* to a base register.

ii) Give the main reasons why absolute memory addresses in a program cannot be known at compile time.

iii) Explain how relative addressing and paged virtual memory both allow programs to run at a range of different positions in memory.

b An "inverted page table" has entries only for the pages currently in main memory. It works using a function h, which to each page number p assigns an integer h(p) between 0 and some maximum value N. For each possible h-value there is maintained a linked list of page table entries for the resident pages with that value. To look up a particular page number p, you just search the list corresponding to h(p). For instance, in the example illustrated, for any p with h(p) = 0 we must check two page table entries.




Suppose that a system has $2^{20}$ pages of virtual memory address space, but at most $2^{10}$ page frames in memory.

i) What would be the advantages and disadvantages of using an inverted page table?

ii) How many h-values are needed if the performance is to be comparable with a conventional page table?

iii) What kind of function h is needed to make best use of this scheme?

*End of paper*