Two hours

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

M.Sc. in Computational Science & Engineering

Fundamentals of High Performance Execution

Wednesday 16th January 2008

Time: 09:45 – 11:45

Please answer TWO Questions from the FOUR questions provided

This is an OPEN book examination

The use of electronic calculators is permitted provided they are not programmable and do not store text
1. a) Describe the general format and operation of the main classes of instruction that are encountered in a typical computer with a register-based load/store instruction set architecture (ISA). Include details of any memory accesses made during execution of each class of instruction. (5 marks)

b) The following C program is to be compiled for a register-based load/store ISA and then executed serially on a single CPU.

```c
/* define a constant */
#define size 5000

/* declare tileTable as a globally accessible array */
double tileTable[size][size];

/* signature of function tileTableLookup must */
/* be declared before main program is compiled */
double tileTableLookup(int x, int y);

/* definition of main program */
int main( ) {
    int i,j;
    double result[size][size];

    /* assume arrays are filled with values before here */

    /* following is the main loop (see question) */
    for (j = 0; j < size; j++) {
        for (i = 0; i < size; i++) {
            result[i][j] += tileTableLookup(i,j);
        }
    }

    /* assume result[][] is read after this point */
}

/* now give full definition of function tileTableLookup */
double tileTableLookup(int x, int y) {
    return tileTable[x][y];
}
```

Question 1 continues on the following page.
Question 1 continues from the previous page.

i) Without attempting any optimisation sketch out the machine-level instructions that might be generated by a C compiler for the main loop (marked) of this program. (You may use here either an assembler language with which you are familiar, or one which you have invented, or an assembler-level pseudo-code. In the latter cases, please make clear what each instruction is intended to do.) (6 marks)

ii) Identify two sections of the code in your answer to part i) which may lead to poor performance, explaining in each case the factors causing performance to be poor. (4 marks)

iii) Suggest instruction-level optimisations that could be applied to the code from part i) in order to remedy the performance problems identified in part ii). Making reasonable assumptions, quantify the potential benefits of applying your suggested optimisations. (5 marks)

(Note: remember that arrays in C are stored in memory in row-major order)
The following C program computes the factorial of 5 using a recursive algorithm.

```c
#define size 5

int fac(int n);

int main() {
    int answer;
    answer = fac(size);
    /*assume answer printed here*/
}

int fac(int n){
    int result;
    if (n<2)
        result = 1;
    else
        result = n*fac(n-1);
    return result
}
```

i) Define suitable activation records for the two functions `main` and `fac`. Explain what each field in each activation record is used for. (5 marks)

ii) Show how copies of each activation record are placed on the stack and the values in them are changed as execution proceeds from start to finish. (6 marks)

iii) How many times is the statement `return result` executed? What are the values returned at each time? (2 marks)

b) Describe the features in a typical register-based instruction set architecture that assist with entry to, and exit from, a function such as that defined above. In both cases, explain the sequence of actions that are necessary and show how the features that you describe help each action. (7 marks)
3. In a typical modern microprocessor system the main memory cycle time is nearly three orders of magnitude larger than the CPU clock cycle time. Computer system designers have thus invented several mechanisms that minimise the need to access main memory.

a) CPU registers typically deliver data within one CPU clock cycle. Why do instruction set designers not provide a very large number of CPU registers in order to minimise memory accesses? (2 marks)

b) Explain how a cache memory is able to provide the same interface as the main memory and yet provides faster access to data in appropriate circumstances. What characteristics are important in a program in order to take advantage of a cache memory? Why is the notion of a cache line important? (5 marks)

c) What policies are available for:

i) placement of a cache line when it is first brought into the cache memory;

ii) writing a changed cache line back to the main memory; and

iii) displacement of a cache line when a new cache line is needed but there is no vacant place to put it in the cache memory?

What effects do the above policies have on cache memory performance? (7 marks)

d) A memory system designer is faced with a choice between the two two-level cache memory schemes defined in Table 1. Which of the two schemes would you select, and why? (6 marks)

<table>
<thead>
<tr>
<th>Scheme A</th>
<th>Scheme B</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Level-1 cache:</strong></td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>8 kb</td>
</tr>
<tr>
<td>Cycle time</td>
<td>2ns</td>
</tr>
<tr>
<td>Expected read hit rate</td>
<td>90%</td>
</tr>
<tr>
<td><strong>Level-2 cache:</strong></td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>256 kb</td>
</tr>
<tr>
<td>Cycle time</td>
<td>20ns</td>
</tr>
<tr>
<td>Expected read hit rate</td>
<td>80%</td>
</tr>
<tr>
<td>Main memory access time</td>
<td>100ns</td>
</tr>
</tbody>
</table>

Table 1: (1 ns = 10^{-9} seconds; the Level-2 hit rate is the local rate, that is, the percentage of accesses referred to the Level-2 cache memory by a miss in the Level-1 cache memory that result in a Level-2 hit.)
4. Consider the following grammar that describes arithmetic expressions involving the operators $+,-,\times,\div,$ and brackets, and the associated semantic rules for computing the postfix notation of the expression.

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{expr} \rightarrow \text{expr} + \text{term}$</td>
<td>$\text{expr}.t := \text{expr}.t</td>
</tr>
<tr>
<td>$\text{expr} \rightarrow \text{expr} - \text{term}$</td>
<td>$\text{expr}.t := \text{expr}.t</td>
</tr>
<tr>
<td>$\text{expr} \rightarrow \text{term}$</td>
<td>$\text{expr}.t := \text{term}.t$</td>
</tr>
<tr>
<td>$\text{term} \rightarrow \text{term} \times \text{factor}$</td>
<td>$\text{term}.t := \text{term}.t</td>
</tr>
<tr>
<td>$\text{term} \rightarrow \text{term} \div \text{factor}$</td>
<td>$\text{term}.t := \text{term}.t</td>
</tr>
<tr>
<td>$\text{term} \rightarrow \text{factor}$</td>
<td>$\text{term}.t := \text{factor}.t$</td>
</tr>
<tr>
<td>$\text{factor} \rightarrow (\text{expr})$</td>
<td>$\text{factor}.t := \text{expr}.t$</td>
</tr>
<tr>
<td>$\text{factor} \rightarrow \text{id}$</td>
<td>$\text{factor}.t := \text{id}'$</td>
</tr>
<tr>
<td>$\text{factor} \rightarrow \text{num}$</td>
<td>$\text{factor}.t := \text{num}'$</td>
</tr>
</tbody>
</table>

Here $\text{id}$ is an identifier and $\text{num}$ is a integer constant.

a) By constructing the annotated syntax trees, show that

i) $(a + 2) \times (b + c) \div d$

ii) $a + 2 \times (b + c) \div d$

and

iii) $a + 2 \times b + c \div d$

where $a, b, c, d$ are identifiers, and $2$ is an integer constant, are valid arithmetic expressions and obtain the postfix notations for the three expressions.

(14 marks)
b) Given an abstract stack machine with the following instructions:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>push v</td>
<td>push integer constant v onto the stack</td>
</tr>
<tr>
<td>rvalue l</td>
<td>push contents of data location l onto stack</td>
</tr>
<tr>
<td>lvalue l</td>
<td>push address of data location l onto stack</td>
</tr>
<tr>
<td>+</td>
<td>pop and add two topmost values on stack and push result onto stack.</td>
</tr>
<tr>
<td>–</td>
<td>pop the two topmost values from stack, subtract the topmost value from the second topmost value and push result onto stack.</td>
</tr>
<tr>
<td>×</td>
<td>pop and multiply two topmost values on stack and push result onto stack.</td>
</tr>
<tr>
<td>÷</td>
<td>pop the two topmost values from stack, divide the second topmost value by the topmost value and push result onto stack.</td>
</tr>
<tr>
<td>:=</td>
<td>the r-value on top of the stack is placed in the l-value below and both are popped.</td>
</tr>
</tbody>
</table>

write machine instructions to perform the assignment

\[ e = (a + 2) \times (b + c) \div d \]

where \( e \) is an identifier. Trace the contents of the stack throughout the set of machine instructions. (6 marks)