One and a half hours

"ARM Instruction Set Summary" is attached

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

Fundamentals of Computer Architecture

Date: Wednesday 25th January 2012
Time: 14:00 - 15:30

Please answer Question ONE and ONE other question

This is a CLOSED book examination

The use of electronic calculators is NOT permitted
**1. Compulsory**

a) Convert the decimal number 73 to binary, then the binary to octal and to hexadecimal, briefly explaining how you do it.  
(3 marks)

b) Explain why only certain literal values can be used in ARM instructions.  
Explain two ways in which the ARM assembler helps people writing assembly-code to cope with this limitation.  
(3 marks)

c) What is the difference between “direct” and “indirect” addressing? Which of these two addressing forms do ARM load and store instructions use, and why?  
(2 marks)

d) What is the effect of the following ARM instructions, in terms of equivalent multiplications:
   
   ADD R1, R1, R1, LSL #2  
   RSB R2, R2, R2, LSL #3

(2 marks)

e) An Operating System consist of basically two main components: the kernel and its libraries. The kernel manages hardware resources and it has a set of managers to manage these basic resources. Name the three managers and briefly describe each.  
(4 marks)

f) The TST operation is one of ARMs instructions. Give a brief description of the TST operation. Assuming that R0 contains #0xAC, explain what happens when the instruction:
   
   TST R0, #0x80

   is performed. To gain full marks you must show full working.  
(4 marks)

g) A running Java program has three basic areas of storage in the JVM memory, as shown in the diagram on the next page:
   
   Area 1 (containing class variables and method code),  
   Area 2 (containing parameters and local variables), and  
   Area 3 (containing objects, instance variables and method tables).  
   Name these three areas.  
(2 marks)
2. a) Describe in detail what happens when the following ARM program is obeyed. At each step, clearly describe the movement of information (both numbers and instructions) between the memory (RAM) and the CPU, and how the values in the memory and in the registers R0, R1, and R15 (PC) change. Assume that the program starts at memory location 0.

LDR R0, x  
STR R0, y  
SUB R1, R0, R0  
LDR R1, z  
SWI 2  
x DEFW 123  
y DEFW 456  
z DEFW 789

b) A Java method (below) is passed three integer parameters via the stack, and returns the result in R0. All registers used by the method, except for R0, must be saved and restored within the method.

```java
int volume (int width, int depth, int length) {
    return width * depth * length;
}
```

Give the ARM code for the “volume” method and draw a diagram of its stack frame. If this method called another method, how would you need to change the start and end of the “volume” method, and the offsets it uses to access its parameters?

(6 marks)

c) Translate the following Java statements, which are just part of a much larger program, into an equivalent sequence of ARM instructions. You should assume that the integer variables x, y and z are in memory and can be accessed just by using their name in a load or store instruction. Try to make your code as efficient as possible.

```java
while (x > 2 && y != z) {
    if (x < y || x < z) {
        x = x + y * z;
    } else {
        y = y - x;
    }
}
```

// lots more code preceding  
// lots more code following
3. a) Multiple interrupt sources (peripherals etc.) require multiple handlers. There are 4 steps required to handle multiple interrupts; List the four steps. (4 marks)

b) Some ARM code uses a “table” of information about peripherals to see which caused an interrupt. The “table”, starting at memory location 0x088, is given below, for 4 peripherals (0-3). Each peripheral has a memory-mapped status register at a different address, and a “ready” bit in some position within the status register:

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>088</td>
<td>DEFW 0x40000008</td>
<td>peripheral 0 - address of status register</td>
</tr>
<tr>
<td>08C</td>
<td>DEFW 0x00000080</td>
<td>peripheral 0 - “ready” bit-pattern</td>
</tr>
<tr>
<td>090</td>
<td>DEFW 0x40000018</td>
<td>peripheral 1 - address of status register</td>
</tr>
<tr>
<td>094</td>
<td>DEFW 0x00000100</td>
<td>peripheral 1 - “ready” bit-pattern</td>
</tr>
<tr>
<td>098</td>
<td>DEFW 0x4000006C</td>
<td>peripheral 2 - address of status register</td>
</tr>
<tr>
<td>09C</td>
<td>DEFW 0x00000200</td>
<td>peripheral 2 - “ready” bit-pattern</td>
</tr>
<tr>
<td>0A0</td>
<td>DEFW 0x40000040</td>
<td>peripheral 3 - address of status register</td>
</tr>
<tr>
<td>0A4</td>
<td>DEFW 0x00000400</td>
<td>peripheral 3 - “ready” bit-pattern</td>
</tr>
</tbody>
</table>

The “loop” of ARM code that uses the “table” above is depicted below.

For your answer, draw up a chart similar to the one below, which has six columns for: address, instruction, registers (R0, R1, R2) and comments. Use it to describe in detail exactly what happens at each step when the ARM program is obeyed: use the register columns to show how the values in R0, R1 and R2 change; use the comments column to clearly describe the movement of information (both numbers and instructions) between the peripheral and the CPU.

Assume that the program starts at memory location 0 and the simulated status register [at address 0x40000008] contains 0x80; and that the loop is executed only once [from address 0 to 20].

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ADR R0, table</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>loop LDR R1, [R0], #4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>LDR R1, [R1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>LDR R2, [R0], #4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>TST R1, R2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>BEQ loop</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(10 marks)

c) In general, an array access involves a computed index which is added to the base of the array. Draw up a diagram that depicts:
“a” the base of an array with ten 32-bit words, and
“i” the index, with a value of 5. (3 marks)
d) The following array access code extract has some errors. Rewrite it correcting all the errors. There are six errors to be found in the code snippet below: (3 marks)

LD R0, i ; i contains index
LDR R, a ; a contains base address
MOV R2, #2 ; because array of int
MUL R3, R0 R2 ; i * 4 bytes
LDR R,[R1,R3 ; R4 = a[i];

A copy of an “ARM Instruction Set Summary” is attached