One and a half hours

Closed Book Examination

(A copy of an "ARM Instruction Set Summary" is attached)

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

Fundamentals of Computer Architecture

Thursday 22\textsuperscript{nd} January 2014?

Time: 14:00 – 15:30?

Please answer Question One
and
ONE other Question from the remaining TWO questions provided

Use a SEPARATE answer book for each SECTION

For full marks your answers should be CONCISE as well as ACCURATE. Marks will be awarded for reasoning and method as well as being correct. Excessively verbose answers will be penalised.

The use of electronic calculators is NOT permitted.
Section A

THREE short questions from RN

1. **Computer Architecture**

   a) Explain briefly the following instruction styles. Excessively verbose answers will be penalised.  
      (3 marks)

      i) 3-address
      ii) 1-address
      iii) load-store

      Bookwork - 1 mark each.

      **3-address**: The processor operates over up to 2 variables and stores the result into a 3rd variable. All the variables are stored in memory.

      **1-address**: The processor operates over up to 2 variables and stores the result into a 3rd variable. At most one of the variables is stored in memory, the rest are in registers.

      **load-store**: The processor can only use register to do operations and has special instructions to move things from/to memory.

   b) What is a stack? How does it work? Why is it needed in modern computing systems? Excessively verbose answers will be penalised.  
      (3 marks)

      Bookwork - 1 mark each.

      The stack is an area of the memory that is used to store information dynamically and safely.

      It works like a pile of heavy books. You can only put (push) or remove (pop) things on/from the top, but not from any arbitrary positions of the pile.

      It is needed to store local method-related information that could be overwritten by nested methods, such as the registers, local variables, …
c) Perform the following base translations using any method you like. Do not use binary as an intermediate representation. (3 marks)

i) 0b1011011011010 to Hexadecimal
ii) @3121 to Binary
iii) 411 to Binary
iv) 0x357 to Dec
v) 0b1100110010 to Dec
vi) 349 to Oct

Practical - 0.5 marks each. Any method is acceptable as long as binary is not used as an intermediate representation.

i) 0b1011011011010 to Hexadecimal
   0xB59

ii) @3121 to Binary
    0b011001010001

iii) 411 to Binary
    0b110011011

iv) 0x357 to Dec
    855

v) 0b1100110010 to Dec
   818

vi) 349 to Oct
    @535

d) How many bits would we need to represent: (1 mark)

i) The 256 colours supported by the VGA Mode 13h standard.
   \[ 256 \leq 2^8 \] \[ \rightarrow \] 8 bits

ii) The 3,120,000 processing cores of the largest supercomputer in the Top500 list of supercomputers in June 2013.
   \[ 3,120,000 \leq 2^{22 \times (4 \times 1024 \times 1024)} \] \[ \rightarrow \] 22 bits

Practical - 0.5 marks each. Logarithms can be used as well.

i) \[ 256 \leq 2^8 \] \[ \rightarrow \] 8 bits

ii) \[ 3,120,000 \leq 2^{22 \times (4 \times 1024 \times 1024)} \] \[ \rightarrow \] 22 bits
e) During your lectures on interrupt vectors a statement was made: 
“*In order for the processor to execute interrupts, a **mapping** must exist between interrupt and handler.”*

i) What mechanism is used to achieve this **mapping**? (1 mark)

ii) Briefly explain this mechanism? (1 mark)

---

1.e.

**Bookwork (2 marks):**

The following points should be covered to some degree in the answer:

i) What mechanism is used to achieve this **mapping**? (1 mark)

The mechanism is an **interrupt vector table**.

ii) Briefly explain this mechanism? (1 mark)

The **vector table** is [composed of] an array of **pointers** to **handlers**; and the processor uses the **interrupt number** as its index into this **array**.

Alternatively:

The **Interrupt vector table** consist of [or is composed of] designated addresses in external memory that hold information necessary to handle an interrupt.

**2 marks** for an answer that depicts all the salient facts in a sensible way; all three managers correctly delineated and briefly described;

**2 ½ marks** for correct answer but not detailed [enough];

**1 ½ marks** for a right-lines approach;

**1 mark** for some basic understanding (or attempt).

Reference Learning Resources, Background Reading, and Lecture itself for detailed information; Lecture(s) No(s).: Lecture 14: Input/Output (2).

**TOTAL marks (2 marks) [2]**
f) The table, in figure 1.f, shows a code snippet from a very simple interrupt vector handler. Copy the table into your answer book and then describe in full what happens when the ARM program is obeyed; using the table in figure 1.f. In the table clearly describe the movement of information (both numbers and instructions) between memory, registers and the CPU in the comments column, and how the values in the registers R0, R1, and memory change, at each step.

(4 marks)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADR</td>
<td>R1, Acknowledge</td>
<td></td>
</tr>
<tr>
<td>MOV</td>
<td>R0, #1</td>
<td>:Load R0 with #1 to acknowledge read.</td>
</tr>
<tr>
<td>STR</td>
<td>R0, [R1]</td>
<td>:Store the acknowledge where R1 points.</td>
</tr>
</tbody>
</table>

Application (Critique) (4 marks):

The following points should be covered to some degree in the answer:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADR</td>
<td>R1, Acknowledge</td>
<td>:Load R1 with the Acknowledge address.</td>
</tr>
<tr>
<td>MOV</td>
<td>R0, #1</td>
<td>:Load R0 with #1 to acknowledge read.</td>
</tr>
<tr>
<td>STR</td>
<td>R0, [R1]</td>
<td>:Store the acknowledge where R1 points.</td>
</tr>
</tbody>
</table>

4 marks for an answer that depicts all the salient facts in a sensible way; good briefly described and expression is evaluated totally correct; 3 marks for correct answer but not detailed [enough]; 2 mark for a right-lines approach; 1 mark for some basic understanding (or attempt). Reference Learning Resources, Background Reading, and Lecture itself for detailed information; Lecture(s) No(s): Lecture 13: Input/Output (1). TOTAL marks (4 marks) [6]
g) Explain base plus index addressing as implemented in the ARM processor.  
(4 marks)

1.g. 
Bookwork & explanation (4 marks):

- Explain base plus index addressing as implemented in the ARM processor.  
(4 marks)

The following points should be covered explicitly in any exam; to gain full marks for your answer:

**Explanation:**

1. Base plus index addressing is a form of *indirect* addressing where the address in memory is formed by *adding* or *subtracting* the contents of two registers.
2. One is normally assumed to point to the *base* of a *data structure* while the other holds an *offset* [index] within that structure.

**4 marks** for an answer that depicts all the salient facts in a sensible way; good briefly described and expression is evaluated totally correct;

**3 marks** for correct answer but not detailed [enough];

**2 mark** for a right-lines approach;

**1 mark** for some basic understanding (or attempt).

Reference Learning Resources, Background Reading, and Lecture itself for detailed information; Lecture(s) No(s): Lecture 19: Arrays (1): Array Implementation.

TOTAL marks (4 marks) [10]
## Section B

2.

a) Name the four steps and briefly describe each step of an assembler. (4 marks)

<table>
<thead>
<tr>
<th>Bookwork (4 marks):</th>
</tr>
</thead>
<tbody>
<tr>
<td>The following points should be covered to some degree in the answer:</td>
</tr>
</tbody>
</table>

Step 1) **Lexical** (word) analysis; the process of converting a sequence of characters into a sequence of tokens;

Step 2) **Syntactic** (structure) analysis; checking instructions are legal;

Step 3) **Semantic** (meaning) analysis, check user-defined names - declared exactly once; and

Step 4) **Code generation**; translate to binary machine code.

4 marks for majority of above; e.g. for an answer that mentions the salient facts in a sensible way (2 marks for a detailed and concise description.),

3 marks for correct answer but not detailed [enough],

2 mark for brief explanation & for a right-lines approach,

1 marks for some basic understanding (or attempt).

Reference Learning Resources, Background Reading, and Lecture itself for detailed information; Lecture(s) No(s.): Lecture 16: Assemblers and Compilers.

TOTAL marks (4 marks) [4]
b) Given the ARM code snippet:

```
start LDR R0, a
```

A set of keywords aligned to the assembly steps are: values; discard spaces; instructions; character; list; space; names; word; sequence; tokens; and legal.

i) Align the appropriate keywords above to the correct assembly step (you named in 3.a) in a table drawn up in your answer book similar to that depicted in figure 3.b.; whilst also naming the assembly steps – in the first column. (4 mark)

<table>
<thead>
<tr>
<th>Assembly step</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1:</td>
<td></td>
</tr>
<tr>
<td>Step 2:</td>
<td></td>
</tr>
<tr>
<td>Step 3:</td>
<td></td>
</tr>
<tr>
<td>Step 4:</td>
<td></td>
</tr>
</tbody>
</table>

Question figure 3.b. A table; showing four steps in the assembly process.

ii) A compiler has five steps; name and briefly describe the compilers last step; also give examples of how it archives the fifth steps goal. (3 mark)

iii) State the three [hierarchical] abstraction levels of programming languages; also state which is human and which machine understandable. (3 mark)
3.b.i.
Application (example) (Critique) (4 marks):

i) Align the appropriate keywords above to the correct assembly step (you named in 3.a);

(4 mark)

The following points should be covered to some degree in the answer:

<table>
<thead>
<tr>
<th>Assembly step</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1: Lexical (word) analysis</td>
<td>character; word; space; discard spaces;</td>
</tr>
<tr>
<td></td>
<td>sequence; tokens</td>
</tr>
<tr>
<td>Step 2: Syntactic (structure) analysis</td>
<td>instructions; legal</td>
</tr>
<tr>
<td>Step 3: Semantic (meaning) analysis</td>
<td>list; names; values;</td>
</tr>
<tr>
<td>Step 4: Code generation</td>
<td>-</td>
</tr>
</tbody>
</table>

**4 marks** for an answer that depicts all the salient facts in a sensible way; all keywords correctly delineated [in context] and aligned to appropriate step.

**3 marks** for correct answer but not all keywords aligned correctly,

**2 mark** for a right-lines approach; but more alignment errors,

**1 mark** for some basic understanding (or attempt).

Reference Learning Resources, Background Reading, and Lecture itself for detailed information; Lecture(s) No(s.): Lectures 16: Assemblers and Compilers.

TOTAL marks (4 marks) [8]
3.b.ii.
Application (example) (3 marks):

ii) A compiler has five steps; name and briefly describe the compiler's last step; also give examples of how it archives the fifth steps goal.

The following points should be covered to some degree in the answer:

The compiler's fifth and final step is: Code optimisation:
This is performed in order to optimise the translations for: speed, memory [usage] – examples of how it achieves its goal are:

i. keep variables in registers;
ii. move code out of loop bodies; and
iii use “clever” instructions.

3 marks for an answer that depicts all the salient facts in a sensible way; correctly delineated [in context],
2 mark for a right-lines approach; but not as detailed,
1 mark for some basic understanding (or attempt).

Reference Learning Resources, Background Reading, and Lecture itself for detailed information; Lecture(s) No(s.).: Lectures 16: Assemblers and Compilers.

TOTAL marks (3 marks) [11]
3.b.iii. Application (example) (4 marks):

iii) State the three [hierarchical] abstraction levels of programming languages; also state which is human and which machine understandable. (3 marks)

The following points should be covered to some degree in the answer:

Three “abstraction levels” of Programming Languages are:

1. Binary Machine Code; is Machine understandable;
2. Assembly Language; is Low-Level Language;
3. High-Level Language; is Human understandable.

3 marks for an answer that depicts all the salient facts in a sensible way; and correctly delineated [in context].

2 mark for a right-lines approach;

1 mark for some basic understanding (or attempt).

Reference Learning Resources, Background Reading, and Lecture itself for detailed information; Lecture(s) No(s.): Lectures 16: Assemblers and Compilers.

TOTAL marks (3 marks) [14]
c) What is bytecode in the context of Java and how does it get executed by a real processor? (3 marks)

3.c. Bookwork (Critique) (3 marks):

c) What is bytecode in the context of Java and how does it get executed by a real processor? (3 marks)

The following points should be covered to some degree in the answer:

1) The Java compiler does not produce real machine instructions.
2) It produces instead, bytecode, which is a stack based (zero address) instruction set.
3) This has the advantages of portability and compactness.
4) The simplest way to execute bytecode is with a software interpreter.
5) However, performance considerations have led to the development of dynamic compilation virtual machines.
6) Here bytecode is initially interpreted but, if monitoring shows that the code is being heavily executed, the bytecode is translated to native machine code.

3 marks for all six points – covered to some degree,
2 marks for three to four points – covered to some degree,
1 marks for 1 a few, and some errors.

Reference Learning Resources, Background Reading, and Lecture itself for detailed information; Lecture 17: Java bytecode.

TOTAL marks (3 marks) [17]
d) Name the three Operating Systems kernel managers and briefly describe each.

(3 marks)

3.d.
Bookwork (Critique) (3 marks):

d) Name the three Operating Systems kernel managers and briefly describe each.

(3 marks)

The three-managers and descriptions are:

1. **Process** Manager (CPU): creates the illusion that several things can be going on at once.

2. **Memory** Manager (main memory): each program has all the memory it needs, but can’t access anything else.

3. **Peripheral** Managers: separate manager (device driver) for each (kind of) peripheral.

**3 marks** for an answer that depicts all the salient facts in a sensible way; all three managers correctly delineated and briefly described;

**2 marks** for correct answer but not detailed enough;

**1 mark** for a right-lines approach;

Reference Learning Resources, Background Reading, and Lecture itself for detailed information; Lecture(s) No(s.): Lecture 15: Interrupts.

TOTAL marks (3 marks) [20]
Section C

3.

a) Explain briefly the difference between Instructions, Pseudo-instructions and Directives and give an example of each. (3 marks)

<table>
<thead>
<tr>
<th>Bookwork – 1 mark each</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instructions</strong> are directly understandable by the target architecture.</td>
</tr>
<tr>
<td>ADD, SUB, RSB, ...</td>
</tr>
<tr>
<td><strong>Pseudo-instructions</strong> are understood by the compiler and translated into one or several regular instructions.</td>
</tr>
<tr>
<td>ADR, ADRL, some LDR and STR, ...</td>
</tr>
<tr>
<td><strong>Directives</strong> are commands for the compiler that are not translated into instructions. They are typically used to define state.</td>
</tr>
<tr>
<td>DEFW, ORIGIN, ...</td>
</tr>
</tbody>
</table>
b) The following code selects an output text (its starting address) depending on the value of three positive integer variables (a, b and c). The different output texts are stored in a table with fixed-size elements of 64 bytes. Transform the code into ARM assembly simplifying and optimising it as much as possible. Explain your optimisations. Assume the inputs are always valid, i.e. a, b and c are lower than the number of elements in the table.  

```
if (a > b && a>c)
    result=table[a]; // Address of the first character of element a
if (b > a && b>c)
    result=table[b]; // Address of the first character of element b
else
    result=table[c]; // Address of the first character of element c
```

**Problem solving –**

- 2 marks for correct code
- + 1 mark for implementing ‘lazy &&’ or 0.5 marks for using conditional instructions
- + 1 mark for using a single STR
- + 1 mark for using shifts to perform the multiplication.

```asm
init
    ADRL R3, table        ; address of table[0]
    LDR R0, a
    LDR R1, b
    LDR R2, c
if
    CMP R0, R1
    BLE else1            ; Lazy &&: only do the second comparison
    CMP R0, R2
    BLE else1
    ADD R4, R3, R0, LSL #6 ; address of table + (a*64) => table[a]
else1
    CMP R1, R0
    CMPGT R1, R2          ; Using conditional instructions - not lazy
    ADDGT R4, R3, R1, LSL #6 ; address of table + (b*64) => table[b]
else2
    ADDLE R4, R3, R2, LSL #6 ; address of table + (c*64) => table[c]
end
    STR R4, result
```
c) The Fibonacci sequence (0, 1, 1, 2, 3, 5, 8, ...) has a number of applications in diverse knowledge areas (mathematics, computer science and biology, to cite just a few). The following code presents an iterative algorithm to compute the $n^{th}$ Fibonacci number ($n \geq 0$) into variable $f$. Transform it into ARM assembly simplifying and optimising it as much as possible. Explain your optimisations.

(6 marks)

```c
if (n==0)
    f=0;
else if (n==1)
    f=1;
else {
    f1=1; // Fibonacci (i-1)
    f2=0; // Fibonacci (i-2)
    for (i=2; i<=n; i++) {
        f=f1+f2; // Fibonacci (i)
        f2=f1;
        f1=f;
    }
}
```

Problem solving – 3 mark for correct code + 0.5 mark for optimising into a decreasing loop + 0.5 for compacting 2 ifs into a single one + 1 mark for optimising loop and if(s) with conditional instructions + 0.5 marks for keeping $f_1$ and $f_2$ in registers + 0.5 marks for using a single STR at the end.

```asm
init LDR R0, n
if   CMP R0, #1       ; condense 2 ifs into a single LS condition
    MOVLS R1, R0     ; put result in R1, will be stored later
    BLS end
fib  MOV R2, #1       ; keep $f_1$ in a register, not use memory
    MOV R3, #0       ; keep $f_2$ in a register, not use memory
    SUBS R0, R0, #2  ; use a decreasing loop, n-2..0 instead of 2..n.
    for  ADD R1, R2, R3   ; put result in R1, will be stored later
        MOV R3, R2
        MOV R2, R1
        SUBS R0, R0, #1 ; 1 CMP instruction saved per iteration!
        BGE for
    end  STR R1, f
```
d) The following function computes the $n^{th}$ Fibonacci number recursively. Transform it into ARM assembly simplifying and optimising it as much as possible. Explain your optimisations and how your code passes parameters and results. (6 marks)

```c
int fib(int n){
    if (n==0)
        return 0; Fibonacci(0)=0
    if (n==1)
        return 1; Fibonacci(1)=1
    else
        return fib(n-1)+fib(n-2);
}
```

Problem solving – 4 marks for correct use of stack (using BL and SP, not overwriting any registers and not leaving spurious elements in the stack) + 1 mark for using only 2 stack positions per call + 0.5 for compacting 2 ifs into a single one + 0.5 marks for using 2 registers (reuse R1 for parameter and then one of the results)

Passing parameter n in R1 and returning the result in R0:

```assembly
fib   CMP R1, #1
      MOVLS R0, R1    ; Put the result (R1=0 or R1=1) in R0
      MOVLS PC, LR    ; and return.
else  PUSH {R1, LR}   ; Save LR and R1 to avoid overwriting.
      SUB R1, R1, #1  ; Put n-1 for the next call into R1
      BL fib          ; and call fib to get fib(n-1).
      POP {R1}        ; Retake the original parameter (R1) from the stack.
      PUSH {R0}       ; Save fib(n-1) in the stack
      SUB R1, R1, #2  ; Put n-2 for the next call into R1
      BL fib          ; and call fib to get fib(n-2).
      POP {R1}        ; Retake fib(n-1) from the stack. Reuse R1
      ADD R0, R1, R0  ; Compute fib(n). R1:fib(n-1), R0:fib(n-2).
      POP {PC}        ; Return and remove LR from the stack.
```

END OF EXAMINATION