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

? January

Time: ? – ? + 1.30

Marker’s feedback version
Marking Scheme Included

Please answer Question ONE and ONE other question.

The use of electronic calculators is not permitted.
Model answer and marking scheme
Context: this course-unit is mainly about translating Java to ARM-code (supposing that a Java compiler did that - students learn about JVM towards the end of the course-unit). It is intended to reinforce their understanding of Java programming, and give them some idea of the separation between high-level and low-level languages. It doesn’t involve consideration of a broader context such as compiling other programming languages or running programs within a genuine operating system environment - this is dealt with in course-units in later years.

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)

      Model answer and marking scheme
      Similar to paper+pencil exercises + previous exams.
      e.g. repeatedly divide by 2, saving remainders (reversed):
      73/2=36r1, 36/2=18r0, 18/2=9r0, 9/2=4r1, 4/2=2r0, 2/2=1r0, 1/2=0r1
      OR subtract (largest first) powers of 2 e.g. 73-64=9, 9-8=1
      so $73_{10} = 1001001_{2}$
      = 1 001 001$_2$ = 111$_8$
      = 0100 1001$_2$ = 49$_{16}$
      Distribution of Marks:
      answer+brief explanation for each:
      binary=1, octal=1, hexadecimal=1
      If binary wrong but then correctly converted to e.g. hex, still get that mark
      Octal/hex: .5 if partition bits correctly but then convert to wrong digit
      Correct values + very brief explanation gets full marks;
      if the values are wrong a very good explanation can still get full marks

      Marker’s feedback
      Some important words here are “explain” and “briefly” – some explanation is essential, but you don’t need to write pages of it.

   b) Explain why only certain literal values can be used [as operands] in ARM instructions.

Page 2 of 21
Explain two ways in which the ARM assembler helps people writing assembly-code to cope with this limitation.

(3 marks)

Model answer and marking scheme
from lectures:
1 mark: only have about 12 bits (or any similar smallish number, or 32 bits – registers – operation – flags) for literals in data-processing instructions
1 mark each for up to 2 of e.g.:
- ADD –ve literal converted to SUB +ve literal & vice-versa
- similarly MOV/MVN and CMP/CMN (and maybe bigger values than ADD & SUB)
- ADRL & LDR= pseudo-instructions, equivalent to multiple real instructions
If they need it, they can get both marks for a very detailed description of just one of these.
only .5 mark for indirect addressing, or DEFW

Marker’s feedback
This part had some of the worst answers for question 1.
I did not ask “Explain two ways in which the ARM assembler helps people writing assembly-code” so a general discussion of how an assembler works or how it converts names to numbers got no marks.
I wanted more than e.g. “the assembler gives an error message if the literal is too big”
Don’t bother restating the question as if that was enough of an answer.

c) (in the context of low-level computer instructions) What is the difference between “direct” and “indirect” addressing [of memory locations]? Which of these two addressing forms do ARM load and store instructions use, and why? (2 marks)

Model answer and marking scheme
From lectures:
1 mark: direct (address is part of the instruction) v. indirect (instruction says where to find the address e.g. in a register)
1 mark: ARM uses indirect addressing, because a full 32-bit address can’t fit inside a 32-bit instruction.
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)

Model answer and marking scheme

from lectures:

1 mark each: R1 * 5 and R2 * 7
get marks for working if there is an arithmetic mistake e.g. \(2^3\) is 9, or for \(R2 \times -7\), or for explaining that e.g. LSL #3 means \(2^3\)
only 0.5 mark for +/– then shift (instead of shift then +/–)
0 marks for \(R1^2\) or \(R2^3\) or \(-1+2*3\) is 5 or \(-1+3\) is 2

Marker’s feedback

“equivalent” does not mean that the two instructions do the same thing.
I wasn’t expecting the equivalent MUL instructions (because they are hard to get right) but I accepted them as a valid answer.

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)
**Model answer and marking scheme**

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

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.

**Distribution of Marks:**

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

3 marks for correct answer but not detailed enough;

2 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 15: Interrupts.
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:

\[
\text{TST R0, #0x80}
\]

is performed. To gain full marks you must show full working. (4 marks)
Model answer and marking scheme

Bookwork & Application (example):
The following points should be covered to some degree in the answer:

Brief description of TST instruction:
- Similar to CMP.
- does not alter its operands, R0 and data.
- sets [conditions] flags for use in a subsequent conditional instruction.


Bit-wise AND calculation: First convert to binary, then perform bit-wise AND:

0...0 1010 1100 = R0 0xAC
0...0 1000 0000 = 0x80
0...0 1000 0000 = result of AND i.e. 0x80

Result is condition codes set to indicate result > 0 etc.

Distribution of Marks: 4 marks for an answer that depicts all the salient facts in a sensible way; good briefly described and bit-wise AND 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: Peripherals, Polling & Interrupts.
Marker’s feedback
Pedagogic assessment [criterion]:
The question assesses lecture 13 learning objective [LOs]; Explain how ARM polling utilises status registers and data registers; explicitly [in this question] Give a brief description of the TST operation & perform the TST calculation and state the result.
The answer should clearly evidence knowledge of required salient facts. In the answer [some of] the following terminology (keywords and naming conventions) should be utilised in context; for example: compare (CMP), operands, not altered, sets, conditions, flags, bit-wise, 32-bit, AND.
Well done, most of you explained the TST in a very technically [correct] manner; also [they] stated it was equivalent to a (logic[al]) AND; as well as giving examples [of the TST in operation - the three line mathematics of it]; you would have received full marks - if every thing was absolutely correct.
However, some students did not:
1/ Give a technically correct description of the TST instruction;
2/ Give an example of a logical AND of two binary numbers (one no. being 0x80) and the correct resultant answer. Again, it must state explicitly the answer (logic output) of the TST function; given the two numbers in (your) example. Mostly, students forgot to give an example [at all]; and this lost [them] marks - as they did not explicitly show [state] that is was equivalent to an AND function, and give an explicit example of two binary numbers ANDed together - so marks were not awarded.
3/ A few students had no real idea what TST was and were marked accordingly.
Sadly a few students did not state [or realise] it is an [or it performs] AND operation; they though it was an OR function [or operation]; this is not the case.
Sometimes students forgot to give an example; and this lost marks - as they did not explicitly show [or depict in an example calculation] the ANDing together of the R0 and #0x80; this should have been undertaken in base or radix two representation - e.g. binary - i.e. two binary numbers ANDed together - if this was not done marks were not awarded.

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)
**Model answer and marking scheme**

Bookwork: The following points should be covered to some degree in the answer: The areas are:
1. Class area.
2. Stack.
3. Heap.

**Distribution of Marks:**
- 2 marks for an answer that depicts all the salient facts in a sensible way; all three names correct;
- 1.5 marks for two areas;
- 1 mark for some basic understanding (or attempt).

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

---

**Marker’s feedback**

Pedagogic assessment [criterion]:
The question assesses lecture 18 learning objective [LOs]; Identify the 3-basic java areas of storage.
The answer should clearly evidence knowledge of required salient facts. In the answer [some of] the following terminology (keywords and naming conventions) should be utilised in context; for example: Class, stack, heap.

Again, most of you (correctly) ascertained the correct names of the three areas; and were awarded full marks.

Those that either wrote down 2 or 1 out of the 3 correctly were awarded appropriate marks. This theory of the three areas was covered in the lecture series in three different audio visual medias: 1) the actual live lecture; 2) the audio recording of the live lecture; and 3) the real time video of the lecture.
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

LDR R0, x, i.e. copy word containing 123 from memory address 20, put into R0 (R0 = 123)
Repeat fetch, PC += 4, execute as above; instructions do:
- copy 123 from R0 into memory address 24 (y) overwriting previous 456
- subtract two copies of 123 (from R0) from each other to put 0 into R1
- copy 789 from memory address 28 to R1, overwriting previous 0
- software interrupt stops program running
Final values: R0 = 123, R1 = 789, y = 0

Model answer and marking scheme
Similar to lecture example + previous exams.
PC = 0: fetch instruction from word 0; PC += 4;
execute LDR R0, x, i.e. copy word containing 123 from memory address 20, put into R0 (R0 = 123)
Repeat fetch, PC += 4, execute as above; instructions do:
- copy 123 from R0 into memory address 24 (y) overwriting previous 456
- subtract two copies of 123 (from R0) from each other to put 0 into R1
- copy 789 from memory address 28 to R1, overwriting previous 0
- software interrupt stops program running
Final values: R0 = 123, R1 = 789, y = 0
Distribution of Marks:
explicit instruction fetch = 1
(explicit) operand fetch/store = 1
accurate description of operations = 1
explicitly state changed register + memory contents = 1
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 [i.e. “callee saved”].

```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 [clearly showing the offsets from SP].
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)
Model answer and marking scheme
Similar to lecture examples and exercises.

STMFD SP!, {R1} or STR R1, [SP,#-4]!
LDR R0, [SP,#12]
LDR R1, [SP,#8]
MUL R0, R0, R1
LDR R1, [SP,#4]
MUL R0, R0, R1
LDR R1, [SP],#16 ; or even explicit ADD
MOV PC, LR

Stack frame:

<table>
<thead>
<tr>
<th>R1</th>
<th>← SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter 3</td>
<td>SP+4</td>
</tr>
<tr>
<td>parameter 2</td>
<td>SP+8</td>
</tr>
<tr>
<td>parameter 1</td>
<td>SP+12</td>
</tr>
</tbody>
</table>

Non-leaf method needs to stack LR:

STMFD SP!, {R1,LR}

... 

LDMFD SP!, {R1} or LDR R1, [SP],#4
LDR PC, [SP],#12

(or a very clear description)
and so all parameter offsets increase by 4

Distribution of Marks:
The important points are the use of the stack, rather than every minor
detail of the instructions. Code does not need to be the most efficient, or
even perfectly correct. e.g. no marks lost for using extra registers
correctly, or even for accessing parameters in the wrong order.
push and pop of R1 etc. = 1
parameter accesses = 1
method exit (including popping parameters) = 1
stack-frame = 1
non-leaf code and offsets = 2
Marker’s feedback
This part was answered the worst within question 2. It amazed me that some students could answer part (c) well but didn’t even use (the same) legal ARM instructions for part (b).
I wanted to see the specific stack frame for this example, not a general stack frame.
I didn’t want code for the method call.
A lot of students missed the last part: “If the method calls another method...”

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. (10 marks)

```java
// lots more code preceding
while (x > 2 && y != z) {
    if (x < y || x < z) {
        x = x + y * z;
    } else {
        y = y - x;
    }
}
// lots more code following
```
Model answer and marking scheme
Similar to paper+pencil/laboratory exercises + previous exams.

LDR R0, x
LDR R1, y
LDR R2, z
while
CMP R0, #2
BLE done
CMP R1, R2
BEQ done
CMP R0, R1 ; start of if

; lose 1 mark for not using conditional instructions:
BLT then
CMP R0, R2
BGE else
then
MLA R0, R1, R2, R0 ;and accept MLA R0,R0,R1,R2
B while
else
SUB R1, R1, R0
B while

; better translation is e.g. either of these:
CMPGE R0, R2
MLALT R0, R1, R2, R0
BLT while
SUB R1, R1, R0
B while

done
STR R0, x
STR R1, y

(any consistent permutation of registers is ok)

Distribution of Marks:
10 marks –1 per error or sub-optimal code
(ignores repetitions of same mistake)
e.g. –1 if no conditional instructions at all (except branches)
–2 for (correct) & instead of && etc.
another –1 if code for & is also inefficient
1 or 2 if incomplete and full of errors, but some valid ARM code
Marker's feedback

Several students wrote code like:

\[
\text{CMP} \ldots \\
\text{Bcondition-true next} \\
\text{next}
\]

instead of:

\[
\text{CMP} \ldots \\
\text{Bcondition-false skip} \\
\text{skip}
\]

Many students got the code for && correct but then also treated the || as if it was &&.

Don’t duplicate code for the “then” part (but I didn’t take marks off for this).

R0=R0+R1*R2 is not the same as “ADD R0,R0,R1,LSL R2” nor R0=R0+R1 and then R0=R0*R2; it would need an extra register e.g. R3=R1*R2 then R0=R0+R3

Remember to STR x & y (but not z - it doesn’t change) at the end of the loop.

You are asked to “Try to make your code as efficient as possible”, so:

Only load variables into registers at the start of the loop.

Only store registers into variables at the end of the loop.

Think about using conditional instructions

Think about using unusual instructions, like MLA in this example
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)

Model answer and marking scheme
Bookwork: The following points should be covered to some degree in the answer: The 4 steps required to handle multiple interrupts are:
1. Set a program running as normal;
2. Allow anything to interrupt;
3. When something does interrupt, pause the program;
4. Decide which interrupt handler to start.

Distribution of Marks: 4 marks for an answer that depicts all the salient facts in a sensible way; all four steps correctly delineated,
3 marks for correct answer but not detailed [enough]; three steps,
2 mark for a right-lines approach; two steps,
1 mark for some basic understanding (or attempt); one step.
Reference Learning Resources, Background Reading, and Lecture itself for detailed information; Lecture(s) No(s.): Lecture 14: Interrupts.

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>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>

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].
Model answer and marking scheme

Application (example)
The following points should be covered to some degree in the answer:
Note change of value of “table” pointer R0 when post-indexed, auto-indexing is used.

<table>
<thead>
<tr>
<th>Addr</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>0x88</td>
<td></td>
<td></td>
<td>Load address of “table” into R0</td>
</tr>
<tr>
<td>4</td>
<td>loop LDR R1, [R0], #4</td>
<td>0x8C</td>
<td>0x40000008</td>
<td></td>
<td>Load status register addr [from table/R0] to R1 and increment R0</td>
</tr>
<tr>
<td>8</td>
<td>LDR R1, [R1]</td>
<td>0x8C</td>
<td>0x80</td>
<td></td>
<td>Load actual status [from peripheral/R1] into R1</td>
</tr>
<tr>
<td>12</td>
<td>LDR R2, [R0], #4</td>
<td>0x90</td>
<td>0x80</td>
<td>0x80</td>
<td>Load status test [from table/R0] into R2 and increment R0</td>
</tr>
<tr>
<td>16</td>
<td>TST R1, R2</td>
<td>0x90</td>
<td>0x80</td>
<td>0x80</td>
<td>Test ready bit [bit-wise (32-bit) AND]</td>
</tr>
<tr>
<td>20</td>
<td>BEQ loop</td>
<td>0x90</td>
<td>0x80</td>
<td>0x80</td>
<td>Branch back to “loop” if result of TST is 0</td>
</tr>
</tbody>
</table>

Distribution of Marks:
10 marks for all four [extra] columns [totally] correct, 5 marks for two of the columns correct, 2 marks for 1 one column, or some errors. Reference Learning Resources, Background Reading, and Lecture itself for detailed information; Lectures 14: Input/Output (2).
**Marker’s feedback**

Pedagogic assessment [criterion]:
The question assesses lecture 14 learning objective [LOs]; (associated to Distinguish between ARM’s interrupts [interrupt vector table], i.e. draw up a table, similar to the one below, which has the six columns for: addresses, mnemonics, registers and the comments in your answer book when you answer the question.
The answer should clearly evidence knowledge of required salient facts.
In the answer [some of] the following terminology (keywords and naming conventions) should be utilised in context; for example: Load, address, table, R0, Load, status, register, address, R1, peripheral, table, Test, ready, bit, bit-wise, 32-bit, AND, Branch, back, loop, Z=1.
Well done most of you correctly delineated the four-steps; utilising the correct technical terminology - in the correct sequence - those whom did this achieved full marks.
Some [students] gave fuzzy [either grammatically or technically] answers, and did not utilise the correct naming conventions for particular (or all) steps; so marks were not awarded.
Some made [or wrote] related [domain knowledge] in their answer - that did not utilise the correct naming conventions and steps - again, it was hard to award marks in this case.
Again, well done those of you whom were verbose, explicit, concise, and utilised the correct naming conventions; also you will have correctly given the correct numeric (addresses) in the columns R0, R1, & R2; and you also mentioned any types of incrementing...
Those whom did not:
1/ Give verbose, explicit and concise descriptions in the comment column;
2/ Put the correct numbers (addresses) in the columns (R0, R1, R2, or addresses);
3/ Use the correct technical naming convention(s) for what is being loaded, or where it was loaded from;
4/ Those that did not mention auto indexing or which register was incremented.
were marked accordingly.

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)
Model answer and marking scheme
Application (example):
Application of their knowledge via diagrammatic depiction. The following points should be covered to some degree in the answer:

Distribution of Marks:
3 marks for diagram that is [totally] correct (includes “a” the base, “i” the index, and i points at correct element),
1.5 marks for average diagram.
Reference Learning Resources, Background Reading, and Lecture itself for detailed information; Lectures 19: Input/Output (2).

Marker’s feedback
Pedagogic assessment [criterion]:
The question assesses lecture 19 learning objective [LOs]; Differentiate between: reference pointer, array base, array index; i.e. Question: Draw up a diagram that depicts ‘a’ the base, ‘i’ the index. The array has ten elements. Depict the case where ‘i=5’ in your diagram.
The answer should clearly evidence knowledge of required salient facts.
In the answer [some of] the following terminology (keywords and naming conventions) should be utilised in context; for example: base, array, index, address, a, i , 0, a[0], a[i].

Well done, most of you were happy drawing up a diagram of:
The following: a (base address), 2) I (index), 3) a + i (base + index address), 4) an array a[] e.g. a[0] to a[n] (n = maximum index, plus some arrows to indicate the different locations of 1) to 4); this would have enabled you to receive full marks.
However, those whom did not draw up a comprehensive explicit diagram with all the required details (i.e. that included verbose naming conventions) depiction the items listed in 1) to 4) were awarded reduced 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];
```

**Model answer and marking scheme**

Application (example): The following points should be covered to some degree in the answer:
1: LD should be LDR
2: R should be R1
3: #2 should be #4
4: needs a “,” between R0 and R2
5: R should be R4; needs a ”]” after R3
or replace lines 3 to 5 by: LDR R4, [R1, R0, LSL #2]

**Distribution of Marks:**
3 marks for all six errors correctly found and corrected,
.5 marks for each error corrected,
Reference Learning Resources, Background Reading, and Lecture itself for detailed information; Lectures 19: Input/Output (2).

**Marker’s feedback**
?

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