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) Without using a calculator, convert the decimal number 97 to binary, then the binary to octal and to hexadecimal, briefly explaining how you do it. (3 marks)

**Model Answer:** Similar to paper+pencil exercises + previous exams. e.g. repeatedly divide by 2, saving remainders (reversed):

97/2=48 r 1, 48/2=24 r 0, 24/2=12 r 0, 12/2=6 r 0, 6/2=3 r 0, 3/2=1 r 0, 1/2=0 r 1

OR subtract (largest first) powers of 2 e.g. 97-64=33, 33-32=1, 1-1=0

so 97\(_{10}\) = 1100001\(_{2}\)

= 001 100 001\(_{2}\) = 141\(_{8}\)

= 0110 0001\(_{2}\) = 61\(_{16}\) (and briefly explain e.g. 0110=6)

**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
b) What is meant by a “stored-program computer”? Explain the purpose of the PC register in an ARM computer. (2 marks)

**Model Answer:** from lectures:
The program instructions are stored in memory, along with the variables.
PC holds the address of the next instruction to be obeyed.

**Distribution of Marks:** stored program=1, PC=1

**Marker’s feedback**
Don’t just reword the question e.g. “a computer that has programs stored in memory”
The question asks about “stored program computer”, not just “stored program”.
Your answer needs to make clear that you understand that this is the usual sort of computer, that uses RAM etc. to store programs and data, not e.g. an embedded system that has programs pre-loaded in ROM
It is not sufficient to say that data is stored in memory - you have to mention program instructions as well.
PC is short for “Program Counter”, but:
– you have to tell me more than this
– it doesn’t count the number of instructions obeyed.
I don’t want to be told about PC-relative addressing here.

c) Briefly describe two differences between a 3-address instruction set and a Load-Store instruction set. Illustrate your answer by writing code for “a = b + c;”
(Java, using int variables) for both instruction sets. (3 marks)

**Model Answer:** from lectures + examples class:
e.g. 3-address doesn’t use registers/takes operands directly from memory
3-address doesn’t use explicit load and store instructions/only needs operations like add
load-store can be more efficient e.g. for more complex expressions
\( a \leq b + c \)
\( v. \)
LDR R2, b
LDR R3, c
ADD R1, R2, R3
STR R1, a

**Distribution of Marks:** 1 per difference (max 2), 1 for example code
simple explanation of both order codes (+ examples) = 2 out of 3
load-store/ARM only = 1 out of 3

**Marker’s feedback**
The question asks about “instruction set”, not just “instruction”.
3-address is **not** the same as 3-registers (e.g. ADD R1, R2, R3)
An “operand” (as used by ARM instructions) is more general than an address e.g. it can allow registers or literals as well – don’t confuse the two.
I asked for 2 differences – if you only gave one you lost a mark.

d) Explain what is meant by “pre-indexed” and “post-indexed” addressing, giving an example of each. (2 marks)
**Model Answer:** from lectures:
pre-indexed: modify register containing indirect address, then access memory e.g. `LDRB R0, [R1,#1]`!
post-indexed: access memory, then modify register e.g. `LDRB R0, [R1],#1`
must explain different meanings, not just give different notations

**Distribution of Marks:** 1=pre explanation+example (or just very good explanation)
1=post (ditto)
half marks if don’t mention address/memory access e.g. “copy R1 to R0 then add 1 to R1” instead of “copy contents of memory location pointed at/given by value of R1 ...”

**Marker’s feedback**
You must distinguish between e.g. “MOV R0, R1” i.e. copy the contents of R1 to R0, and “LDR R0, [R1]” i.e. copy the memory word pointed to/addressed by R1 to R0.
Tell me what the instructions do, not just how to write them down.
On ARM, these operands are only used by LDR and STR instructions, not e.g. ADD.
A lot of you forgot the “!” for pre-indexed – you should have looked at the “ARM Instruction Set Summary” that you were given.

e) Two naming conventions are associated with garbage collection: “live” and “fragmentation”. Explain what each term means in the context of garbage collection.
Model Answer: Bookwork:
The following points should be covered to some degree in the answer:

Live: To discover if an object is garbage (or stale); all references in the class variable and stack areas are followed to objects. If an object is referenced (in this context); the object is marked as “live”. (e.g. Objects in use, that is...) Any references in those objects are followed to other objects – also marked “live”. Etc. until no more unexplored references exist. If the object is not live it is a candidate for the garbage collector; to remove...

Fragmentation: Every object on the heap is scanned; by the garbage collector. Any object not marked as garbage (marked as “live”) will be removed by the garbage collector. Their store is “returned” to the memory manager. But inevitably “holes” form in memory where these objects were. After some time holes form across the entire memory - this effect is usually known as “fragmentation”.

Reference Learning Resources, Background Reading, and Lecture itself for detailed information; Lecture 18: Java Memory Usage.

Distribution of Marks: 4 marks for an answer that depicts all the salient facts in a sensible way,
3 marks for correct answer but not detailed (enough),
2 mark for a right-lines approach,
1 marks for some basic understanding (or attempt).

(4 marks)

Marker’s feedback
The question assesses lecture learning objective 8; define what is meant by (the term) Garbage collector; as the terms “live” and “fragmentation” are directly related to garbage collection.
The question’s answer should clearly evidence knowledge of required salient facts relating to the terms “Live” and “Fragmentation”.
In the answer (some of) the following terminology (keywords and naming conventions) should be utilised in context; for example: With respect to “Live”: discovery of objects, stale objects, reference to an object, object is garbage, ... With respect to Fragmentation: holes, gaps, varying sizes, not marked live, objects removed...
Main differentiation that must be clearly evidenced in your answer is that: Live is a term directly related to garbage collection (objects still referenced (in use) are marked “live”). Whereas fragmentation is a “side effect” of garbage collection of the stale objects (where the stale object is removed from the memory) ...
f) An assembler can be viewed as having four assembly steps. Name the four steps and briefly describe each step.

**Model Answer:** Bookwork:
The following points should be covered to some degree in the answer:
1) Lexical (word) analysis; the process of converting a sequence of characters into a sequence of tokens.
2) Syntactic (structure) analysis; checking instructions are legal.
3) Semantic (meaning) analysis; check user-defined names: declared exactly once.
4) Code generation; translate to binary machine code.

Reference Learning Resources, Background Reading, and Lecture itself for detailed information; Lecture 16: Assemblers and Compilers

**Distribution of Marks:** 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).

(4 marks)

**Marker's feedback**
The question assesses lecture learning objective 3 and 4; List the four Assembly Steps, and Explain in detail what each Assembly Step does; as the these steps relate directly to a deeper understanding of the theory of Assemblers.

The question's answer should clearly evidence knowledge of required salient facts relating to the terms “Lexical (word) analysis”, “Syntactic (structure) analysis”, “Semantic (meaning) analysis”, and “Code generation”.

In the answer (some of) the following terminology (keywords and naming conventions) should be utilised in context; for example: (evaluate) characters, (build) words, (discard) spaces, instructions (from words), legal (instructions) (or allowable (Opcodes)), (use-defined) names (values), (create) list, translation (instructions to machine code)...

Main differentiation that must be clearly evidenced in your answer is that: each of the four steps has a clear operation (or process) that it is carrying out; and each of these must be carried out in the correct sequence...

g) Briefly explain how “array bounds checking” would be performed; illustrate your
answer using ARM code.

**Model Answer:** Bookwork & Application:
The following points should be covered to some degree in the answer:
The array index would be compared to array size “N” prior to loading (or storing) data from the array at a specified index. The code to check the index used is in bounds would be something like:

```arm
CMP R0, #N
BHI ErrorCode
LDR R3, [R1,R0]
```
where “R0” contains the index; “N” is array size; “ErrorCode” is the start of an error handler – in the case where the index is “out of bounds”.

(Note: R0 is a member of a set; with minimum value 0; and maximum value N-1; i.e. \( R0 \in \{0, ..., N - 1\} \); while “N” is the size of the array, **not** its index, which is “R0”.)

Reference Learning Resources, Background Reading, and Lecture itself for detailed information; Lecture 20: Arrays (2).

**Distribution of Marks:** 2 marks for an answer that mentions the salient facts in a sensible way (1 marks for a sensible explanation (text), 1 mark for correct ARM codes),
1.5 marks for correct answer but not detailed [enough],
1 mark for a right-lines approach,
.5 marks for some basic understanding (or attempt).

(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, R2 and R15 (PC) change. Assume that the program starts at memory location 0

LDR R0, x
ADD R1, R0, R0
STR R1, y
LDR R2, z
SWI 2
x DEFW 123
y DEFW 456
z DEFW 789
Model Answer: 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 x i.e. memory address 20, put into R0 (R0=123)
Repeat fetch,PC+=4,execute as above; instructions do:
– add two copies of 123 from R0 to put 246 into R1
– copy 246 from R1 into memory address 24, overwriting previous 456
– copy 789 from memory address 28 to R2
– software interrupt stops program running

**Distribution of Marks:**
- instruction fetch = 1
- (role of) PC and increment by 4 = 1
- explicit operand fetch/store = 1
- explicitly state register + memory contents = 1
- accurate description of operations = 1

Marker’s feedback
Lots of you forgot about instruction fetch, and some of you about the PC register, even though they are mentioned in the question.
As part of telling me what the instructions do, say what the results are e.g. “R0 becomes 123” and “memory location y changes from 456 to 246”, so that I know for sure that you understand.
Despite my reminders, again many of you told me that the “STR” instruction changed the value of R1 i.e. behaved like an “LDR”.
Lots of you told me what the “DEFW” assembler directives do, even though they are not instructions and are never obeyed. Some of you even told me that they were obeyed. Even worse, some of you told me that the “LDR” instruction has to search memory for its variable.
All this happens during assembly or loading, well before run-time, and should not be part of the answer to this question – you are just wasting your time and mine writing about it, and making me think that you don’t really understand.

b) When implementing a method call on the ARM, its parameters, return address (link), and variables can be stored on the stack and accessed via the SP register. Draw an example “stack frame” to show how it is laid out in memory. Give an example of using a parameter, and an example of using a method variable. Give example ARM code to call a method, and to return from it. (5 marks)
**Model Answer:** Similar to lecture examples.

Stack frame - see below

e.g. parameter: `LDR R0, [SP, #20]`
e.g. variable: `LDR R0, [SP, #4]`
e.g. call:
  `LDR R0, first parameter in call`
  `STR R0, [SP, #-4]`
and push the rest of the parameters
BL method
(and now use the result, in R0)
e.g. return:
  `ADD SP, SP, #bytes of variables`
  `LDR PC, [SP],#bytes of parameters+4 or just MOV PC, LR`

**Distribution of Marks:**
- stack-frame = 1
- using parameter = .5
- using variable = .5
- call = 2
- return = 1

---

**Model answer and marking scheme**

Stack frame looks like e.g.:

```
<table>
<thead>
<tr>
<th>method variables ← SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>LR</td>
</tr>
<tr>
<td>parameters</td>
</tr>
</tbody>
</table>
```

or e.g.

```
| variable 2 ← SP |
| variable 1 SP+4 |
| LR SP+8        |
| parameter 3 SP+12 |
| parameter 2 SP+16 |
| parameter 1 SP+20 |
```
Marker’s feedback

I think that this was the worst-answered part of any of my questions. I expected to see a stack frame for a method, containing parameters, return link and variables, and pointed at by the SP register, as asked for. Lots of you just drew (and explained) a stack, which is not the same thing at all! (Also, many of you drew a stack frame with top and bottom reversed compared to the lectures, which was unhelpful. Similarly, lots of you used a different stack convention from the lectures e.g. adding to push instead of subtracting, or even mixed conventions, which I had to mark as wrong.)

The question clearly asked for several different things (stack frame, using a parameter, using a variable, calling a method, returning from a method). You should include all of these in your answer, and also clearly indicate the different parts of your answer accordingly. Some of you just wrote explanations instead of giving ARM code.

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 w, 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,
// including declarations for w, x, y, z as integer variables
while (x > 2) {
    y = (x - z) * y;
    switch (y) {
    case 0: w = w + 1; break;
    case 1: y = y - 1; break;
    default: z = 1 - z;
    }
}
// lots more code following
```
Model Answer: Similar to paper+pencil/laboratory exercises + previous exams.

```
.    LDR   R0, w
.    LDR   R1, x
.    LDR   R2, y
.    LDR   R3, z
.    ADR   R4, table
.loop  CMP   R1, #2 ; x :: 2
.    BLE   end
.    SUB   R4, R1, R3 ; x - z
.    MUL   R2, R2, R4 ; * y
.    CMP   R2, #1 ; y :: 1
.    BHI   default
.    LDR   PC, [R4, R2,LSL #2]
.table  DEFW  case0
.   DEFW  case1
.case0  ADD   R0, R0, #1 ; w + 1
.    B     loop
.case1  SUB   R2, R2, #1 ; y - 1
.    B     loop
.default RSB  R3, R3, #1 ; 1 - z
.    B     loop
.end    STR   R0, w
.    STR   R1, x
.    STR   R2, y
.    STR   R3, z
```

Distribution of Marks: 10 marks
-1 per significant error (but ignoring repetitions of same error)  
(-4 if no implementation of switch at all)
-1 per suboptimal code e.g. not pre-loading registers
-1 forgetting to save registers at end  
(any consistent permutation of registers is ok)
1 or 2 if incomplete and full of errors, but some valid ARM code
Model Answer: continued...
Alternative implementations of switch:

MULS R2, R2, R4
BEQ case0
CMP R2, #1
BEQ case1
default ...

CMP R2, #1
BHI default
BEQ case 1
case0 ...

CMP R2, #1
BHI default
ADDNE R0, R0, #1
SUBEQ R2, R2, #1
B loop
default ...

MULS R2, R2, R4
ADDEQ R0, R0, #1
BEQ loop
CMP R2, #1
SUBEQ R2, R2, #1
BEQ loop
RSB R3, R3, #1
B loop

MULS R2, R2, R4
ADDEQ R0, R0, #1
BEQ loop
CMP R2, #1
SUBEQ R2, R2, #1
RSBNE R3, R3, #1 ; or RSBHI
B loop

Distribution of Marks: I accepted those using conditional instructions without penalty; the rest lost a mark
Marker’s feedback
This was a little harder than things I have asked in previous exams, so I was a bit more generous than normal.
Also, by mistake, the loop doesn’t change “x” so isn’t very sensible (e.g. I should replace “w” by “x” in “case 0”). I gave an extra mark for anyone who pointed this out, or used it to explain why their code was odd.
Very few of you actually used a table, instead using a series of compares and conditional branches. I tended to knock a mark off for this, unless you said why (e.g. “there are only two tests”) or gave code that was nearly as efficient (e.g. using conditional instructions like “ADDEQ”).
Some common problems:
– not moving all the “LDR”s and “STR”s outside the loop
– not having any “STR”s at all
– finishing with a “SWI” (I didn’t knock a mark off, but it shows you didn’t read the question carefully)
– following the “MUL R2,...” with “CMP R2, #0” instead of using “MULS”
– forgetting to use “RSB”
– having branches to branches (e.g. from the switch to the while)
– using e.g. “MOV R5, #2” and “CMP R1, R5” instead of “CMP R1, #2”
3. a) In the context of data exchange between CPU and peripherals:
Differentiate between the two main data exchange protocols; polling and interrupts.

(4 marks)

**Model Answer:** Bookwork, Critique
The following points should be covered to some degree in the answer:
The two main data exchange protocols are:
Polling and interrupts; polling is initiated by the CPU, and interrupts are initiated by the peripherals.
1. Polling basically has two phases:
   Polling phase 1: CPU polls the status register to check if a key has been pressed.
   Polling phase 2: Then it reads the data register (in the peripheral) in order to transfer the data from the peripheral to the CPU. (Note this is in the context of a keyboard connected to a peripheral.)
2. Interrupts nominally involve three steps:
   Interrupts step 1: When the CPU starts the interrupt handler it first checks the status register. If the status (specified by a set bit) is incorrect the handler initiates an error handler.
   Interrupts step 2: As long as the status facilitates a data transfer; the data register is then read; and the data is saved (in the appropriate location) (or fetched and written).
   Interrupts step 3: Finally an acknowledgement is written back to the peripheral.

Reference Learning Resources, Background Reading, and Lecture itself for detailed information; Lecture 14: Input/Output (1).

**Distribution of Marks:** Bookwork (2 marks), Critique (2 marks)
4 marks for all points (issues) covered correctly (Must cover 1 & 2 for full marks);
2 marks for half of the points (issues) covered correctly
Marker’s feedback

The question assesses lecture 13 learning objective 2 to 4; Indicate what is meant by polling; Explain how ARM polling utilises status registers and data registers; Illustrate why interrupts are used instead of polling; as the terms “polling” and “interrupts” are directly related to explaining how a processor interacts with an I/O device - in cases where the processor may poll the I/O device or the I/O device interrupts the processor.

In the answer (some of) the following terminology (keywords and naming conventions) should be utilised in context, for example with respect to polling: registers, status (register), data (register), check. In the case of interrupts: (external I/O) interrupts (processor), (processor checks) status (register), (if no) error, (processor reads) data register, (if error) error handler [called].

Main differentiation that must be clearly evidenced in your answer is that: the two methods are that:

i) transfer of data initialled (and under total control of) the processor. The processor interrogates the “status register”. If the status register signals that a data item (character) is ready to be transferred. Then processor reads data register of I/O device in order to read data item to be transferred; from I/O device.

ii) The second method is not initiated by the processor. It is the external I/O device that interrupts the processor; to request the processor to transfer data. Implicit in this is that transfer of data is required - which may appear to imply that there is no need to read the status register. But after being interrupted and selecting the correct interrupt handler - it is the process that reads the status register - to check is an error has occurred - this would be the case if the status register bit pattern signalled that no transfer was required - hence an error condition has occurred. Once the status has been checked and verified the processor reads the data register; in order to transfer the data item.

The question’s answer should clearly evidence knowledge of required salient facts relating to the two issues; this was not done explicitly in some of the answers given. If the answers did not detail the differences plainly as stated in the above (and in the Example answer); e.g. answer gives evidence of knowledge of required salient facts, full marks were not awarded.

b) In the context of data exchange between CPU and peripherals:

Describe in detail exactly what happens when the following ARM program is obeyed. Clearly describe the movement of information between peripheral and the CPU, and how the values in the registers R0, R1 and R15 (PC) change, at each step. Assume that the program starts at memory location 0 and the simulated Status_Reg contains 0x80 while the simulated Data_Reg contains 0x72 or ASCII character ‘H’.
Model Answer: Application
The following points should be covered to some degree in the answer:
Detailed description of what happens when the above ARM program is obeyed:
There are a number of ways the students could answer this (see below)

Reference Learning Resources, Background Reading, and Lecture itself for detailed information; Lecture 14: Input/Output (1)

Distribution of Marks: Application (5 marks + 5 marks)
10 marks for all points (issues) covered correctly (Must cover 1 & 3 or 2 & 3 for full marks, either as in the tables below, or as a textual paragraph covering the same points.);
5 marks for half of the points (issues) covered correctly.

Model answer and marking scheme
1. Visually sequencing through the program in a table:

<table>
<thead>
<tr>
<th>R15(PC)</th>
<th>Label+Mnemonic</th>
<th>RO</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>loop</td>
<td>?</td>
<td>0x1c</td>
</tr>
<tr>
<td>0x4</td>
<td>ADR R1, Status_Reg</td>
<td>0x80</td>
<td>0x1c</td>
</tr>
<tr>
<td>0x8</td>
<td>TST R0, #0x80</td>
<td>0x80</td>
<td>0x1c</td>
</tr>
<tr>
<td>0xc</td>
<td>BEQ loop</td>
<td>0x80</td>
<td>0x1c</td>
</tr>
<tr>
<td>0x10</td>
<td>ADR R1, Data_Reg</td>
<td>0x80</td>
<td>0x20</td>
</tr>
<tr>
<td>0x14</td>
<td>LDRB R0, [R1]</td>
<td>0x72</td>
<td>0x20</td>
</tr>
<tr>
<td>0x18</td>
<td>SWI 2</td>
<td>0x72</td>
<td>0x20</td>
</tr>
</tbody>
</table>
Model answer and marking scheme
2. Visually sequencing through the program’s PC in a list:

PC (R15) starts at:
0x0, instruction ADR R1, Status_Reg executed; then R0=? & R1=0x1c;
(Note ADR pseudo instruction composed of an ADD instruction)
0x4, instruction LDRB R0, [R1] executed, R0=0x80 & R1=0x1c;
0x8, instruction TST R0, #0x80 executed, R0=0x80 & R1=0x1c;
0xC, instruction BEQ loop executed, R0=0x80 & R1=0x1c;
0x10 instruction ADR R1, Data_Reg executed, R0=0x80 & R1=0x20;
0x14 instruction LDRB R0, [R1] executed, R0=0x72 & R1=0x20;
0x18 instruction SWI 2 executed, R0=0x72 & R1=0x20.

Model answer and marking scheme
3. Visually sequencing through the program in a listing (table):

ADR R1, Status_Reg ; R1 points to status reg.
; R1 is loaded with the Status_Reg add.
LDRB R0, [R1] ; Read status, Byte load of reg. R0 pointed to by R1;
; loads 8-bits into R0; top 24 bits are "0"s
; Status_Reg data pointed to by R1.
TST R0, #0x80 ; Test ready bit (bit 7), bit-wise (32-bit) AND of R0 and
; 0x80 to test if bit 7 is set in Status_Reg [in R0].
BEQ loop ; If not ready, try again; if Z conditions flag set
; (Equal) Z=1 branch [back] to "loop" label.
ADR R1, Data_Reg ; R1 points to data reg.,
; R1 is loaded with the Data_Reg add.
LDRB R0, [R1] ; Ready, so read data,
; load R0 with Data_Reg, add. Pointed to by R1,
; loads 8-bits into R0; top 24 bits are "0"s.
SWI 2 ; halt program
Marker’s feedback

The question assesses lecture 13 learning objective 3: Explain how ARM polling utilises status registers and data registers; as the terms “status” and “data” registers are directly related to explaining how a processor interacts with an I/O device - in cases where the processor initiates the read or write process.

In the answer (some of) the following terminology (keywords and naming conventions) should be utilised in context, for example with respect to polling (reading the 8 status and data registers): points (ADR), load (R1), address (of status register); byte (load), (read data into) R0, (R1) points (to address of status register), loads bottom 8-bits; (bit) test (TST), (bit-wise) AND, (AND with) #0x80 (test pattern - validates correct bit set), branch (BEQ) (if bit not set - as this signal no data to be transferred), points (ADR), load (R1), address (of data register); byte (load), (read data into) R0, (R1) points (to address of data register), loads bottom 8-bits.

Main differentiation that must be clearly evidenced in your answer is that: the sequence of 7 instructions is each explicitly and concisely explained in detail - the process flow - the use of registers - what each register is explicitly used for - clearly differentiating between loading data into a register and utilising the register as a pointer “[R1]” - clear differentiation of when a DCD (the DCD directive is used to label and initialize the data operands) addresses of the status and data register are loaded (e.g. using pseudo code ADR) - finally when the I/O (simulation code) is halted (SWI2).

The question’s answer should clearly evidence knowledge of required salient facts relating to the two issues; this was not done explicitly in some of the answers given. If the answers did not detail the differences plainly as stated in the above (and in the Example answer); e.g. answer gives evidence of knowledge of required salient facts, full marks were not awarded.

c) State and briefly describe each of the five main functions a kernel performs for an operating system. (6 marks)
Model Answer: Application
The following points should be covered to some degree in the answer:
1. File Manager – manages files on disk and the file structure;
2. Device Drivers – control I/O devices;
3. Memory Manager – allocates memory to programs;
4. Scheduler and Dispatcher – decides which program to run and ensures that it has the correct resources; and
5. Network manager – controls networking (connections to other computers)...

Reference Learning Resources, Background Reading, and Lecture itself for detailed information; Lecture 15: System Software.

Distribution of Marks: 6 marks for correctly naming all five and giving concise brief descriptions. Three named and described; 3 marks

Marker’s feedback

The question assesses lecture 15 learning objective 2; List five of the main functions a Kernel performs for an (OS - operating system); as the term “main functions” directly related to explaining the important functions a OS kernel performs - as there are a large number, we only look at (five) some of the more important functions.

In the answer (some of) the following terminology (keywords and naming conventions) should be utilised in context, for example with respect to the five main functions a Kernel performs for an (OS - operating system): file, device, memory, scheduler, 9 dispatcher, network, manager. Pointing out keywords such as: files, disk, structure, I/O, control, allocates, decides (which to run), controls, connects, (other) computers.

Main differentiation that must be clearly evidenced in your answer is that: the five issues are that: each performs different functions. Some - like the file manager and memory manager - may indeed rely on each other.

The question’s answer should clearly evidence knowledge of required salient facts relating to the two issues; this was not done explicitly in some of the answers given. If the answers did not detail the differences plainly as stated in the above (and in the Example answer); e.g. answer gives evidence of knowledge of required salient facts, full marks were not awarded.

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