COMP 15111

One and a half hours

"ARM Instruction Set Summary" is attached

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

Fundamentals of Computer Architecture

Date: Wednesday 16th January 2019

Time: 09:45 - 11:15

Please answer all FIVE Questions.

Use a SEPARATE answerbook for each QUESTION.

This is a CLOSED book examination

The use of electronic calculators is NOT permitted

[PTO]

1. **Representation of Information**

a) Convert the following decimal numbers into unsigned 8-bit binary form and then into hexadecimal by any method, showing your working.

(i) 43_{10} (4 marks)
(ii) 253_{10} (4 marks)

b) The following numbers are in 8-bit signed two's complement form. Convert each of them to decimal format.

(i) 01011001 (3 marks)
(ii) 11111110 (3 marks)

c) In a system where colours are represented as RGB values, different shades of grey have identical values of R, G and B. If each colour is represented by 8 bits, state how many different shades of grey can be represented (including black and white), showing your reasoning. (4 marks)

d) In the data instructions of the ARM processor, only 12 bits are available to encode literals (constants). How many different literal values can be expressed? You are not required to describe how such literals are encoded. (2 marks)

2. **Computational Models**

a) (i) List the sequence of actions that the processor must perform during the execution of an instruction. Hence explain why a Von Neumann architecture computer needs to have a program counter (PC). (5 marks)

(ii) In an ARM processor how is the PC used in the execution of a Branch instruction? (3 marks)

b) (i) Why is a Link Register necessary for the implementation of methods? (4 marks)

(ii) If a method calls a second method, what has to be done to enable return to the correct place once execution of the first method is complete? (2 marks)

c) What are the advantages of a stack compared to other methods of allocating memory? (6 marks)

3. **Code and Data Structures in Assembly Language Programs**

a) State the function of the ARM assembly language program below and briefly explain how it works.

```
      LDR R1, a
      LDR R2, b
      MOV R0, #0
loop  ADD R0, R0, R1
      SUB R2, R2, #1
      CMP R2, #0
      BNE loop
      STR R0, c
```

(6 marks)

b) State two cases where the program could produce an erroneous result, and show how each case could be detected by modifications to the program. In each case if an error is detected the program should simply branch to a specific error handling routine. You do not need to write the error handling routines. (8 marks)

c) Give two different ways that the program in part a) could be modified to execute more quickly. (6 marks)

4. **Assemblers and Assembly**

a) List the sequence of operations performed by an assembler, briefly stating what the assembler does in each operation. (8 marks)

b) Explain the difference between an instruction and a pseudo-instruction, giving an example of an ARM assembler pseudo-instruction and the action that an assembler might take to assemble it. (6 marks)

c) Show how a constant and a variable could be created and used in ARM assembly language, giving an example of each. (6 marks)

[PTO]

5. **Virtual machines**

a) What is the advantage of a virtual machine? Give an example of a situation in which Java is used where the use of the Java Virtual Machine (JVM) is useful. (6 marks)

b) The Java Virtual Machine is a zero-address or stack-based architecture. Explain the operation of the Java Virtual Machine by reference to the execution of some simple instructions such as addition and conditional branch. (10 marks)

c) By reference to the usage of the Java Virtual Machine explain why it is designed as a stack-based architecture. (4 marks)

END OF EXAMINATION

COMP15111 ARM instruction set summary

Data Processing

ADD	Rd, Rn, Op	Rd= Rn + Op
SUB	Rd, Rn, Op	Rd= Rn – Op
RSB	Rd, Rn, Op	Rd= Op – Rn ("reverse subtract")
AND	Rd, Rn, Op	Rd= Rn AND Op
ORR	Rd, Rn, Op	Logical OR
EOR	Rd, Rn, Op	Exclusive OR
BIC	Rd, Rn, Op	Bit Clear: Rd= Rn & !Op
MOV	Rd, Op	Rd= Op
MVN	Rd, Op	Rd= !Op
CMP	Rn, Op	set status on Rn – Op
CMN	Rn, Op	set status on Rn + Op
TST	Rn, Op	set status on Rn AND Op
TEQ	Rn, Op	set status on Rn EOR Op
MUL	Rd, Rn, Rs	Rd= Rn * Rs
MLA	Rd, Rn, Rs, Rp	Rd= (Rn * Rs) + Rp

An "Op" in the table above can be a literal (e.g. `#10`) or a register (e.g. `R1`) or a shifted register.

If a shift (by a literal e.g. `#3` or a register e.g. `R6`) is required it is specified as the fourth parameter e.g.
`ADD R1, R4, R5, LSL #3` ; R1 = R4 + R5*8
`ADD R1, R4, R5, LSL R6` ; R1 = R4 + R5*2^{R6}

LSL Shift	logical shift left by $0 \leq Shift \leq 31$ places, putting 0s into least significant end
LSR Shift	logical shift right by $0 \leq Shift \leq 32$ places, putting 0s in most significant end
ASR Shift	arithmetic shift right by $0 \leq Shift \leq 32$ places, copying the sign bit into the most significant end
ROR Shift	circular rotate right by $0 \leq Shift \leq 32$ places, moving bits lost from one end into other end
RRX	one place right shift. Carry from status reg. shifts into most significant bit. If status reg. set by instruction, carry bit = least significant bit. This gives a 1-bit circular rotate through the carry bit

Loads and Stores

LDR	Rd, Address	loads a 32-bit word into Rd from memory location
LDRB	Rd, Address	loads 8-bits into Rd; top 24 bits are '0's
STR	Rd, Address	stores Rd at memory location
STRB	Rd, Address	stores bottom byte of Rd at memory location
LDMFD	Rd, register list	multiple reg load,load from addr Rd
STMFD	Rd!, register list	multiple reg store, Rd is updated after each store operation

An "Address" in the table above can be a numerical address (e.g. `100`) or a named memory location (e.g. `fred`) or calculated from the contents of registers and literals e.g.:

operand form	address	final value of R0
$[R0]$	R0	(unchanged)
$[R0, R1]$	R0 + R1	(unchanged)
$[R0, \#1]$	R0 + 1	(unchanged)
$[R0], R1$	R0	R0 + R1
$[R0], \#1$	R0	R0 + 1
$[R0, R1]!$	R0 + R1	R0 + R1
$[R0, \#1]!$	R0 + 1	R0 + 1

In the examples above the value copied from `R1` can be modified by a shift (but `R1` itself is unchanged) e.g.
`LDR R2, [R0, R1, LSL #3]` ; address = R0 + 8*R1
The possible shifts are LSL, LSR, ASL and ROR as described for Data Processing instructions above.

Please Turn Over

COMP15111 ARM instruction set summary continued

Control Transfer

B	Label	branch to label: R15= label
BL	Method	branch and link: R14= R15, R15= method
SWI	Number	software interrupt

Condition Codes

Any instruction can be made conditional e.g. `ADD` can become `ADDEQ` etc.

Code	Meaning	Flag condition
EQ	Equal	Z
NE	Not Equal	!Z
GE	Greater than or Equal (signed)	N = V
GT	Greater Than (signed)	(N = V) . !Z
LT	Less Than (signed)	N != V
LE	Less than or Equal (signed)	(N != V) + Z
HI	Higher (unsigned)	C . !Z
LS	Lower or Same (unsigned)	!C + Z
CS/HS	Carry Set/Higher or Same (unsigned)	C
CC/LO	Carry Clear/Lower (unsigned)	!C
MI	Minus (negative)	N
PL	Plus (positive)	!N
VS	Overflow Set	V
VC	Overflow Clear	!V
AL	Always	TRUE

Also, any Data Processing instruction can be followed by an "S" e.g. `ADD` can become `ADDS` meaning that the result of the instruction is to be compared with zero.

Assembler Supplied 'Pseudo Operations'

NOP		no operation
MOV	Rd, #–nn	replaced by MVN
CMP	Rn, #–nn	replaced by CMN
ADR	Rd, label	Rd=label address. replaced by ADD Rd, R15, #nn
ADRL	Rd, label	Rd=label address. Used when #nn in ADR is out of range
LDR	Rd, =nnnnnnnn	Load constant

Directives

ORIGIN	&address	Set address of code following
ALIGN		to next 4-byte boundary
DEFW	&12345678	Define the value of the next word(s)
DEFB	0, 2, "bytes"	Define the value of the next byte(s)
DEFS	&20	Reserve the next 20 bytes

DEFB, DEFW and DEFS allow data to be planted amongst the instructions in a program.