COMP 15111

One and a half hours

"ARM Instruction Set Summary" is attached

Questions A1 and B1 are COMPULSORY

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

Fundamentals of Computer Architecture

Date: Wednesday 18th January 2017

Time: 09:45 - 11:15

Please answer Questions A1 and B1
and
ONE other Question: either A2 or B2

Use a SEPARATE answerbook for each SECTION

This is a CLOSED book examination

The use of electronic calculators is NOT permitted

[PTO]

Section A

A1. **COMPULSORY**

Computer Architecture

a) Perform the following base translations. Not showing the process you use to do the conversion will be penalized. (3 marks)

i) 189_{10} to Hexadecimal
ii) 101010_2 to Decimal
iii) 27_8 to Binary

b) Explain the difference between the following pairs of ARM instructions and how we can use them to make our code more efficient: (4 marks)

i) LDR and LDMFD
ii) SUB and SUBS

c) Explain why modern processors include registers within their architecture and how registers are different from RAM. (3 marks)

A2.

a) Explain why a stack is important for modern computing systems and how it operates. (3 marks)

b) Explain briefly the differences between the following addressing modes and when to use one over the other. (4 marks)

i) Direct addressing and Indirect addressing
ii) Base+Offset addressing and Post-index addressing

c) We want to implement a tic-tac-toe style game with a 4×4 grid. The current board status is stored in memory in a data structure like the one below in which the status of each square is saved as a character (either '_', 'o' or 'x').

board →

'_'	'_'	'_'	'_'
'_'	'_'	'_'	'_'
'_'	'_'	'_'	'_'
'_'	'_'	'_'	'_'

Write the ARM code to update the contents of a single square assuming that R0 contains the row [0…3], R1 contains the column [0…3] and R2 contains the character to write in that location {'_', 'o', 'x'}. Optimize your code as much as possible and explain what optimizations you have performed.
As an example, if the input is R0=1, R1=3, R2='o' then the (empty) board above will be updated to:

board →

'_'	'_'	'_'	'_'
'_'	'_'	'_'	'o'
'_'	'_'	'_'	'_'
'_'	'_'	'_'	'_'

(5 marks)

d) A Java method (below) is passed three integer parameters via the stack, and returns its result in R0. All registers used by the method must be saved and restored within the method, i.e. "callee saved". The method `check_overflow()` is callee saved as well and returns nothing.

```
int sum(int a, int b, int c) {
  int res = a + b + c;
  check_overflow();
  return res;
}
```

Give the ARM code for this method and draw a diagram of its stack frame clearly showing the offsets from SP. (8 marks)

Section B

B1. **COMPULSORY**

a) Explain the term ‘Garbage Collection’ and explain why it is important to the implementation of Java. (4 marks)

b) Two naming conventions are associated with garbage collection: ‘live’ and ‘fragmentation.’ Explain what each term means in the context of garbage collection. (2 marks)

c) Explain the principle of zero address instructions with the aid of an example of simple arithmetic expression evaluation. (4 marks)

B2.

a) In the context of data exchange between CPU and peripherals. Differentiate between the two main data exchange protocols; polling and interrupts. (4 marks)

b) The following array access ARM code extract has some errors. Rewrite it correcting all the errors. There are six errors to be found in the code snippet below, figure B2.b. (3 marks)

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

Question figure B2.b. Code contains six errors.

c) What is the purpose of the Java Virtual Machine? (4 marks)

d) A ‘stack’ program is depicted in figure B2.d.1.

Instruction		
Address	Label	Mnemonic
000	`StackTop2`	`DEFW 0`
004	`StackStart2`	`DEFW 0`
008	`StackBottom2`	`DEFW 0`
00C	`StackProg2`	`MOV R0, #1`
010		`ADR SP, StackStart2`
014		`STR R0, [SP]`
018		`MOV R0, #2`
01C		`STR R0, [SP, #4]`
020		`SVC 2 ;(or SWI 2)`

Question figure B2.d.1. A ‘stack’ program; showing the address, Labels, & Mnemonics columns.

Describe in detail exactly what happens when the above ARM program is obeyed; using the table below, figure B2.d.2. In the table clearly describe the movement of information (both numbers and instructions) between memory, stack and the CPU in the comments column, and how the values in the registers R0, and R13 [SP] change, at each step.

Address/ label	Mnemonic	R0	R13 [SP]	**COMMENTS**
00C	`MOV R0, #1`	-	-	
010	`ADR SP, StackStart2`			
014	`STR R0, [SP]`	-	-	
018	`MOV R0, #2`	-	-	
01C	`STR R0, [SP, #4]`	-	-	
020	`SVC 2; (or SWI 2)`	-	-	

Question figure B2.d.2. A ‘stack’ program; showing the address, Mnemonics, R0, R13, & comments columns.

Assume that the program starts at memory location 0x00C; for this exam question.

In your answer please draw up a table, similar to the one above, which has the five columns for: addresses, mnemonics, registers and the comments in your answer book when you answer the question.

(9 marks)

END OF EXAMINATION

COMP15111 ARM Instruction Set Summary

This is not a complete list and you may need to consult other documentation for more detail.

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.