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 17th January 2018

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 decimal number 237_{10} into binary by any method, showing your working (4 marks)

b) Convert the binary number 101011110001_2 into hexadecimal and into octal. (4 marks)

c) The ASCII character set includes the upper-case and lower-case characters of the Latin (English) alphabet, the decimal digits and a limited set of punctuation marks and arithmetic symbols.

(i) Estimate the number of characters that need to be included in the ASCII character set (4 marks)

(ii) What is the minimum number of bits required to represent one character in ASCII? (4 marks)

(iii) When are character coding schemes with greater than 8 bits per character required? (4 marks)

2. **Computational Models**

The ARM processor has a Load-Store architecture. In a Load-Store architecture the instruction set separates operations that transfer information between memory and the processor from operations that process information.

a) Why must a processor with a Load-Store architecture have registers? (6 marks)

b) For each of part (i) and part (ii) state an ARM instruction that transfers one word of information and an ARM instruction that transfers one byte of information:

(i) From memory to a register in the processor. (2 marks)

(ii) From a register in the processor to memory. (2 marks)

c) In the 32-bit ARM processor each instruction is encoded into one 32 bit word and the processor has 16 registers, R0 – R15.

(i) In an instruction of the form ADD Rd, Rn, Rm what is the minimum number of bits required to specify the registers to be used? (5 marks)

(ii) Why does the 32-bit instruction length limit the number of registers that could be included in the processor? (5 marks)

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

a) Give examples showing how the following could be achieved in an ARM assembly language program:
(i) Creating a constant
(ii) Creating and initialising a variable in memory
In each case state why your example is an appropriate method. (4 marks)

b) Write a short section of ARM assembly language code that will read and print out each character of a string in turn.
The last character of the string should be followed by a byte with the numerical value zero, i.e. the string is null-terminated.
You should assume that the SuperVisor Call SVC 0 prints the character in R0.
(8 marks)

c) An array of 32-bit integers is stored in the memory of an ARM system.
(i) State an addressing mode that is suitable for loading each element of the array in turn. (2 marks)
(ii) Show how this mode works and explain why it is particularly suitable for addressing arrays, giving an example of an instruction or instructions that use the mode. (6 marks)

4. **Methods and the stack**

a) With the help of a diagram show how a stack is arranged in memory and explain the operation of ARM instructions that store and retrieve 32-bit words to and from a stack. (8 marks)

b) (i) Write short sections of ARM code showing how a method can be called and how execution is returned to the correct instruction at the end of the method. Assume that the method does not call another method. (4 marks)
(ii) If the method did call another method, what problem would arise? Explain how the stack could be used to overcome the problem. (4 marks)

c) Methods can be implemented in ARM code such that the calling program passes parameters to the method either in registers or on the stack. State one advantage and one disadvantage of each approach (4 marks)

[PTO]

5. **The Operating System and Accessing Peripherals**

a) A keyboard is used to input characters to an ARM-based computer. Give two techniques that could be used to transfer a character from the keyboard to the computer, giving an advantage and a disadvantage of each. (8 marks)

b) A simple ARM-based system has only one input device connected. List the actions that an interrupt service routine must perform when an input event occurs. (6 marks)

c) Explain the difference between a hardware interrupt and a software interrupt (a software interrupt is caused by the SWI or SVC instruction). (6 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.

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.

Please Turn Over