✓CA

+ Exam Notes

Paper Number(s): **EE2-19**

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2012

EEE Part II: MEng, BEng and ACGI

INTRODUCTION TO COMPUTER ARCHITECTURE

Friday 15 June, 2.00 pm

Corrected Copy

Time allowed: 1:30 hours

There are THREE questions on this paper.

Answer ALL questions.

All questions carry equal marks.

Any special instructions for invigilators and information for candidates are on page 1.

Examiners responsible:

First Marker(s): Clarke, T.
Second Marker(s): Demiris, Y.

Special instructions for invigilators

The sheet Exam Notes should be distributed with the Examination Paper.

Special instructions for students

The prefix 0x, or suffix $_{(16)}$, introduces a hexadecimal number, e.g: 0x1C0, $1C0_{(16)}$.

Unless otherwise specified negative numbers are represented in two's complement.

Unless otherwise specified machine addressing is little-endian.

The sheet Exam Notes, as published on the course web pages, is provided and contains reference material.

Answer ALL the questions.

The Questions

1.

a) Calculate the following values encoded as stated in a 32 bit ARM word. Write your answers in hexadecimal.

(i) -51.25 encoded as an IEEE 754 floating point number

[2]

(ii) -100 encoded as a two's complement signed number

[1]

(ii) 6.375 encoded as a fixed point binary number with 8 bits to the right of the binary point

[1]

b) For each of the following 32 bit integer operations, ignoring overflow, state whether the hardware required is different when applied to signed and unsigned numbers, or identical in these two cases. If different give an example which illustrates this.

(i) **binary addition (no carry in)**

(ii) **binary "greater than" comparison**

(iii) **multiplication with 64 bit result**

[6]

c) Write three separate fragments of ARM assembly code which respectively set R1, R2, R3 to the values specified in Figure 1.1 without using multiply or divide instructions. Credit will be given for code which is compact.

[6]

d) The registers R2:R1 are used to store a two's complement signed 64 bit number, with R2 holding bits 63:32 and R1 bits 31:0. Write a fragment of code which will calculate the absolute value of this number, overwriting the original number with the result.

[4]

R1	R0 modulo 8	The remainder when R0 (interpreted as unsigned integer) is divided by 8
R2	sign(R0)	-1 if R0 < 0 otherwise 1. R0 is interpreted as two's complement signed.
R3	R0*33	Bits of the result above bit 31 are truncated

Figure 1.1

2. Each code fragment (a) - (c) below executes with all condition codes and registers initially 0, and memory locations as in Figure 2.1. State the values in R0-R3, the condition codes, and any changed memory locations, after execution of the code fragment. Write your answers using as a template a copy of the table in Figure 2.3, *deleting the example row labelled (x)*. Each answer must be written with register values in signed decimal (55, -31), memory location values in hexadecimal, and condition codes in binary. The example row (x) illustrates this. In Figure 2.3 n/a indicates a value which is not required.

a) Code as in Figure 2.2a. [6]

b) Code as in Figure 2.2b. [6]

c) Code as in Figure 2.2c. [8]

Location (word)	Value
0x100	0x04030201
0x104	0x08070605
≥ 0x108	0x0

Figure 2.1. Memory locations

```
MOV R0, #-1
ADDCSS R1, R0,R0
ADCCS R2, R2, #0
RSB R3, R0, #1
MVNS R0, #-1
```
(a)

```
MOV   R3, #0x100
LDMIA R3, {R0,R1}
STMED R3!, {R0}
LDRB  R1, [R6,#103]
MOV R7, #40
MOV R0,#1
STRB  R1, [R0,R7, lsl #2]
LDRB  R2, [R0,#0x100]!
```
(b)

```
MOV R0, #61
MOV R1, #100
SUBS R3, R0, R1, lsl #26
ADD R4, R0, R0, asr #1
ADD R5, R0, R0, lsr #1
EOR R2, R1, R0
ORRS R1, R1, R1, ror #1
```
(c)

Figure 2.2. Code fragments

	R0	R1	R2	R3	NZCV	Changed memory locations
(x)	0	222	-33	4	0110	Mem32[0x200]=0x23451000
(a)						n/a
(b)					n/a	
(c)						n/a

Figure 2.3. Template for answers.

3. An ARM write-through cache has line (block) size 16 bytes, and contains 8 lines. You may assume that ARM addresses are 32 bit long, and that as normal consecutive 32 bit words have ARM addresses offset by 4.

a) Indicate the precise address bits comprising the cache select, index and tag fields.

[3]

b) Draw a block diagram of the cache hardware, indicating the width of all busses. Calculate the total storage in bits required to implement the cache, both tags and data, showing your working

[5]

c) Initially all cache lines are invalid (V=0), and cache data is undefined. The main memory byte with address *n* is set equal to *n* modulo 256 (the least significant binary 8 bits of n). The sequence of CPU memory operations A, B, C, D in Figure 3.1 is executed consecutively from the CPU. After each CPU memory operation state:

(i) The (possibly null) set of byte memory locations transferred to and/or from the cache during execution of the operation.

[4]

(ii) Which cache line or lines (labelled by index) are used by the operation

[4]

(iii) Cache line data that was used at the end of the operation.

[4]

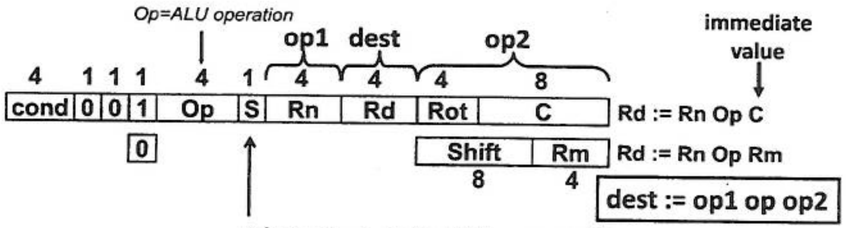
A	READ byte 0
B	WRITE byte 1
C	WRITE byte 16
D	READ byte 129

Figure 3.1

Introduction to Computer Architecture

EXAM NOTES

Data processing (ADD,SUB,AND,CMP,MOV, etc)



S bit = 1 => status bits are written
S bit = 0 => status bits unchanged

The second operand, Op2, is either a constant C or register Rm

Assume Shift=0, Rot=0, for unshifted Rm or immediate C

C	1 => carry
V	1 => signed overflow
N	1 => negative
Z	1 => zero

Op-codes	
AND	
ANDEQ	EQ,NE,... => Condition
ANDS	S => set status on result
ANDEQS	note position of S

Op	Assembly	Operation	Pseudocode
0000	AND Rd,Rn,op2	Bitwise logical AND	Rd := Rn AND op2
0001	EOR Rd,Rn,op2	Bitwise logical XOR	Rd := Rn XOR op2
0010	SUB Rd, Rn, op2	Subtract	Rd := Rn – op2
0011	RSB Rd, Rn, op2	Reverse subtract	Rd := op2 – Rn
0100	ADD Rd,Rn,op2	Add	Rd := Rn + op2
0101	ADC Rd,Rn,op2	Add with carry	Rd := Rn + op2 + C
0110	SBC Rd, Rn, op2	Subtract with carry	Rd := Rn op2 + C – 1
0111	RSC Rd, Rn, op2	Reverse sub with C	Rd := op2 – Rn + C – 1
1000	TST Rn, op2	set NZ on AND	Rn AND op2
1001	TEQ Rn, op2	set NZ on EOR	Rn EOR op2
1010	CMP Rn, op2	set NZCV on -	Rn - op2
1011	CMN Rn, op2	set NZCV on +	Rn + op2
1100	ORR Rd,Rn,op2	Bitwise logical OR	Rd := Rn OR op2
1101	MOV Rd, op2	Move	Rd := op2
1110	BIC Rd,Rn,op2	Bitwise clear	Rd := Rn AND NOT op2
1111	MVN Rd,op2	Bitwise move invert	Rd := NOT op2

Data Processing Op2

ADD r0, r1, op2
MOV r0, op2

Examples

ADD r0, r1, r2
MOV r0, #1
CMP r0, #-1
EOR r0, r1, r2, lsr #10
RSB r0, r1, r2, asr r3

Op2	Conditions	Notes
Rm		r15=pc, r14=lr,r13=sp
#imm	imm = s * 2^n (0 ≤ s ≤ 255, if n is even) (0 ≤ s ≤ 127, if n is odd)	Assembler will translate negative values changing op-code as necessary Assembler will work out s,n from imm if they exist, or give error.
Rm, shift #s Rm, rrx #1	(1 ≤ s ≤ 31) shift => lsr,lsl,asr,asl,ror	rrx always writes carry ror writes carry if S=1 shifts do not write carry
Rm, shift Rs	shift => lsr,lsl,asr,asl,ror	shift Rm by no of bits equal to register Rs value (takes 2 cycles)

Multiply in detail

- MUL,MLA were the original (32 bit LSW result) instructions
  - Why does it not matter whether they are signed or unsigned?
- Later architectures added 64 bit results

Register operands only
No constants, no shifts

NB d & m must be different for MUL, MLA

ARM3 and above

MUL	rd, rm, rs	multiply (32 bit)	Rd := (Rm*Rs)[31:0]
MLA	rd,rm,rs,rn	multiply-acc (32 bit)	Rd:= (Rm*Rs)[31:0] + Rn
UMULL	rl, rh, rm, rs	unsigned multiply	(Rh:Rl) := Rm*Rs
UMLAL	rl, rh, rm, rs	unsigned multiply-acc	(Rh:Rl) := (Rh:Rl)+Rm*Rs
SMULL	rl,rh,rm,rs	signed multiply	(Rh:Rl) := Rm*Rs
SMLAL	rl,rh,rm,rs	signed multiply-acc	(Rh:Rl) :=(Rh:Rl)+Rm*Rs

ARM7DM core and above (64 bit multiply result)

Data transfer (to or from memory LDR,STR)

4									4	4	12	
cond	0	1	0	P	U	B	W	L	Rn	Rd	S	Rd ↔ mem[Rn+S]
			1								Shift | Rm	Rd ↔ mem[Rn+Rm*]

Rm* = Rm shifted left by constant (scaled)

Bit in word	0	1
U	subtract offset [Rn-S]	add offset [Rn+S]
B	Word	Byte
L	Store	Load

P bit	W bit	address	Rn
0	0	[Rn]	Rn := Rn+S
0	1	not allowed	NB S = 0 gives Rn no change
1	0	[Rn+Rm] or [Rn+S]	no change
1	1	[Rn+Rm] or [Rn+S]	Rn := Rn+Rm or Rn := Rn+S

Data Transfer Instructions

```
LDR      load word
STR      store word
LDRB     load byte
STRB     store byte
LDREQB  ; NB B is after EQ condition
STREQB  ;
```

```
LDR   r0, [r1]                ; register-indirect addressing
LDR   r0, [r1, #offset]       ; pre-indexed addressing (base + offset)
LDR   r0, [r1, #offset]!      ; pre-indexed, auto-indexing (base + offset + writeback)
LDR   r0, [r1], #offset       ; post-indexed, auto-indexing (change Rn after)
LDR   r0, [r1, r2]            ; register-indexed addressing (base + reg)
LDR   r0, [r1, r2, lsl #shift] ;scaled register-indexed addressing (base + reg * 2^shift)
LDR   r0, address_label; PC relative addressing (pc+8 is read, offset calculated)
ADR   r0, address_label; load PC relative address (pc+8 is read, offset calculated)
```

```
LDMED   r13!, {r0-r4,r6,r6} ;  ! => write-back to address register
STMFA   r13, {r2}              ; no write-back
STMEQIB r2!, {r5-r12}          ; note position of EQ or other condition
```

<u>higher reg nos go to/from higher mem addresses</u>

[E|F][A|D] Empty|Full, Ascending|Descending

[I|D][A|B] Increment|Decrement, After|Before

Name	Stack	Other
pre-increment load	LDMED	LDMIB
post-increment load	LDMFD	LDMIA
pre-decrement load	LDMEA	LDMDB
post-decrement load	LDMFA	LDMDA
pre-increment store	STMFA	STMIB
post-increment store	STMEA	STMIA
pre-decrement store	STMFD	STMDB
post-decrement store	STMED	STMDA

Instruction Timing

Exact instruction timing is very complex and depends in general on memory cycle times which are system dependent. The table below gives an approximate guide.

Instruction	Typical execution time (cycles)
Any instruction, with condition false	1
data processing (except register-valued shifts)	1 (+3 if Rd = R15)
data processing (register-valued shifts): MOV R1, R2, lsl R3	2 (+3 if |Rd = R15)
LDR,LDRB, STR, STRB	4 (+3 more if Rd = R15)
LDM (n registers)	n+3 (+3 more if Rd = R15)
STM (n registers)	n+3
B, BL	4
Multiply	7-14

IEEE 754

s	e	f
31	30 – 23	22 – 0

$$x = (-1)^s \, 2^{(e-127)} 1.f$$

s, e, f are all **unsigned** binary fields

f defines number in range 1-2:
e defines binary multiplier or divisor (e-127)

1.	f

Shadow registers

- **FIQ** mode: R8 - R14 shadowed
- **IRQ**, **SVC**, **abort**, **UND** modes: R13 - R14 shadowed
- Return from interrupt: set status bits to restore CPSR, so use SUBS to set PC equal to stored return address with offset & restore CPSR.
- **R13** is SP

BL, SWI	R14 = Return address
IRQ,FIQ,UND	R14 = Return address + 4
Abort	R14 = Return address + 8

Exception	Mode	Vector address
Reset	SVC	0x00000000
Undefined instruction	UND	0x00000004
Software interrupt (SWI)	SVC	0x00000008
Prefetch abort (instruction fetch memory fault)	Abort	0x0000000C
Data abort (data access memory fault)	Abort	0x00000010
IRQ (normal interrupt)	IRQ	0x00000018
FIQ (fast interrupt)	FIQ	0x0000001C

Cond		Condition	Status Bits
0000	EQ	Equal	Z set
0001	NE	Not equal	Z clear
0010	CS/HS	Unsigned ≥ (High or Same)	C set
0011	CC/LO	Unsigned < (Low)	C clear
0100	MI	Minus (negative)	N set
0101	PL	Plus (positive or 0)	N clear
0110	VS	Signed overflow	V set
0111	VC	No signed overflow	V clear
1000	HI	Unsigned > (High)	C set and Z clear
1001	LS	Unsigned ≤ (Low or Same)	C clear OR Z set
1010	GE	Signed ≥	N equals V
1011	LT	Signed <	N is not equal to V
1100	GT	Signed >	Z clear and N equals V
1101	LE	Signed ≤	Z set and N not equal to V
1110	AL	Always	any
1111	NV	Never (do not use)	none

Machine Instruction Overview (1)

Data processing (ADD,SUB,CMP,MOV)

cond	0	0	0	Op	Rn	Rd	Shift	Rm	Rd := Rn Op Rm*
		1		ALU operation			S		Rd := Rn Op S

Rm* = Rm with optional shift

multiply instructions are special case

Data transfer (to or from memory LDR,STR)

cond	0	1	0	Trans	Rn	Rd	S		Rd ↔ mem[Rn+/-S]
		1		Byte/word, load/store, etc			Shift	Rm	Rd ↔ mem[Rn+/-Rm*]

Multiple register transfer

cond	1	0	0	Type	Rn	Register list	Transfer registers to/from stack

Overview (2)

Branch B, BL, BNE, BMI...

cond	1	0	1	L	S

$\boxed{0}$ L = 0 => Branch, B ...

$\boxed{1}$ L = 1 => Branch and link (R14 := PC+4), BL ...

PC := PC+8+4*S
S is sign extended
NB +8 because of pipelining.

---

cond	1	1	0	0	
cond	1	1	0	1	
cond	1	1	1	0	

coprocessor interface

Software Interrupt (SWI)

cond	1	1	1	1	S

Simulate hardware interrupt: S is passed to handler in swi mode

ARM (Keil) Assembler Reference

```
<label> <directive>   <operands>   ; <comment>   all fields are optional
<label> <op-code>     <operands>   ; <comment>   all fields are optional

        DCD     c1, c2, c3, ...    ; load memory with consecutive 32 bit words
        DCB     b1, b2, b3, ...    ; load memory with consecutive bytes
    SYM EQU     TABLE + 4          ; set label SYM equal to constant expression
        ENTRY                      ; marks first instruction executed
        ORG     0x1234             ; load program/data at given address
```

constant	notes
1234	Decimal
0x10fc	Hex
2_101110	binary
's'	Single character
"cat"	Sequence of characters: LSB first
LABEL	Symbols are numeric constants, usually addresses

NB - course notes and questions use & as hex prefix as well as 0x, Keil assembler only allows 0x

Symbols are case sensitive: Loop and LOOP are different

Mnemonics, **directives**, and shift, register names are case insensitive but must be all upper or lower case

ASCII code

b_7					0	0	0	0	1	1	1	1
b_6					0	0	1	1	0	0	1	1
b_5					0	1	0	1	0	1	0	1
Bits b_4	b_3	b_2	b_1	Row ↓ / Column →	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FC	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

EE2-19

Introduction to Computer Architecture - Answers 2012

All questions are compulsory, questions are weighted equally.

Answer to Question 1

Q1 is an easy question testing basic knowledge & understanding.

1.

a)

(i) C24D0000 [2]

Common mistakes: not realising hex was required. Getting bits displaced by 1 thinking that exp field end at second hex digit (it overflows third hex digit by one.

(ii) FFFFFFBC [1]

Common mistakes: not giving all initial hex digits

(iii) 110.011 [1]

Common mistakes: not giving binary (!). Not specifying point.

b)

(i) identical [2]

(ii) different 1 > -1 (signed) but 1~00000001 < -1~FFFFFFFF (unsigned) [2]

(iii) different because -1*-1 = 1 (signed), FFFFFFFF*FFFFFFFF=(2^32-1)^2

=2^64-2*2^32+1 ~ FFFF0000 (unsigned, note 2^64 is lost) [2]

c)

AND R1, R0, 0x7 [2]

CMP R0, #0
MOVGE R2, #1
MOVLT R2, #-1
Common mistakes: using branches [2]

ADD R3, R0, R0, lsl #5 [2]
Common mistakes: ADD + MOV, not using shift.

d)

CMP R2, #0
RSBLTS R1, #0 ; conditional negate R1 storing carry
RSCLT R2, #0 ; conditional negate R2 with carry
Common mistakes: lots of people did not know how to use the carry with S instruction followed by RSC/SBC addition.

[4]

Answer to Question 2

This question tests ability to understand and analyse operation of ARM assembly code in detail. It requires accuracy and comprehensive understanding of the instructions, but is straightforward.

(a) *tests understanding of two's complement arithmetic, and condition codes*

(b) *test understanding of memory addressing modes*

(c) *tests understanding of logical operations*

Mistakes too various to identify!

Location (word)	Value
0x100	0x04030201
0x104	0x08070605
≥ 0x108	0x0

Figure 2.1. Memory locations

a)

	R0	R1	R2	R3	R4	R5	NZ CV	comments
MOV R0,#-1	-1						00 00	
ADDCSS R1,R0,R0								not executed
ADCCS R2,R2,#0								not executed
RSB R3,R0,#1				2				1 - R0 = 1 - (-1)=2
MVNS R0, #-1	0	0	0	2	n/a	n/a	00 00	MVN is invert not negate

b)

	R0	R1	R2	R3	comments
MOV R3,#0x100				0x100	
LDMIA R3,{R0,R1}	0x04030201	0x08070605		0x108	
STMED R3!,{R0}				0x10C	push, mem[0x108] := 0x40302010
LDRB R1,[R6,#103]		0x07			little-endian, addr=103
MOV R7,#40					R7:=40
MOV R0,#1	1				
STRB R1,[R0,R7,lsl #2]					addr=R0+R7*4=1+160 mem8[161]=07 or mem32[160] = 0x700 NB 160= 0xA0, 161=0xA1
LDRB R2, [R0,#0x100]!			2		

c)

	R0	R1	R2	R3	NZ CV	comments
MOV R0,#61	61					
MOV R1,#100		100				
SUBS R3,R0, R1, lsl #26				-671088579	0010	
ADD R4,R0,R0 asr #1						R0+R0/2
ADD R5,R0,R0 lsr #1						R0+R0*2
EOR R2,R1,R0			0x59=89			
ORRS R1,R1,R1 ror #1		0x76=118			0010	NZ are overwritten but don't change

Answer to Question 3

a)

Select A(3:0)

Index A(6:4)

Tag A(31:7)

Common mistakes: ignoring byte select (2 bit displacement) or not specifying all fields.

[3]

b)

must include tag registers, storage for each line. select & index field, tag field going to comparator

Total storage: 8 * (16*8 + (31-7+1)) = 8 * (128 + 25) = 153 bits

[5]

c)

A - read[0-15], line 0, 0F0E0D0C0B0A09080706050403020100
B write[1], line 0, as above with byte 1 replaced by written data
C read[16-31], line 1, write[16], as above with MS nibble 0 replaced by 1 and byte 16 replaced by written data
D read[128-141], line 0, as above with MS nibble 0 replaced by 80.

[4+4+4]