Master

i

Paper Number(s): **E2.19**

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2010

EEE Part II: MEng, BEng and ACGI

Corrected Copy

INTRODUCTION TO COMPUTER ARCHITECTURE

Tuesday, 1 June 2.00 pm

Time allowed: 1:30 hours

There are FOUR questions on this paper.

Question 1 is compulsory and carries 40% of the marks.

Answer Question 1 and two others from Questions 2-4 which carry equal marks (30% each).

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 booklet Exam Notes 2010 should be distributed with the Examination Paper.

Special instructions for students

The prefix &, or suffix $_{(16)}$, introduces a hexadecimal number, e.g: &1C0, $1C0_{(16)}$.

The booklet Exam Notes 2010, as published on the course web pages, is provided and contains reference material.

Question 1 is compulsory and carries 40% of marks. Answer only TWO of the Questions 2-4, which carry equal marks.

The Questions

1. **[Compulsory]**

a)

(i) Calculate the 32 bit IEEE-754 representation of -0.75, writing your answer as 8 hexadecimal digits.

(ii) The 32 bit words:

a = &70000000

b = &70000001

represent real numbers a', b' in 32 bit IEEE-754 floating point format. Calculate $b' - a'$.

[8]

b) A write-through direct mapped cache has line (block) length n words. What types of locality are exploited in the two cases $n = 16$ and $n = 1$? In each case state the words read or written to memory during a cache word write miss of address 0.

[8]

c) Write a fragment of ARM assembly code in as few instructions as possible, without using multiply instructions, which implements:

R0 := 9*R1 + 15*R3

.

[8]

d) Compute the values of R4-R11 at the end of executing the ARM assembly code in *Figure 1.1*, giving your answers in decimal or hexadecimal.

[16]

```
MOV   R0, #0
MOV   R1, #1
MOV   R2, #2
MOV   R3, #3
EORS  R4, R2, R3
SUBS  R5, R2, R3
ADC   R6, R0, R1
SBC   R7, R0, R1
ORR   R8, R3, R3, lsl #4
ADD   R9, R3, R3, ror #2
STRB  R2, [R0], #1
STRB  R1, [R0], #1
STRB  R3, [R0], #1
STRB  R2, [R0,#1]
MOV   R10, R0
MOV   R0, #0
LDR   R11, [R0]
```

Figure 1.1. ARM assembly code

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 value of 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* omitting the row labelled (x) which indicates the required format of your answer. Each answer must be written in hexadecimal except the condition codes, which must be in binary, as indicated in row (x).

a) Code as in *Figure 2.2a*.

[10]

b) Code as in *Figure 2.2b*.

[10]

c) Code as in *Figure 2.2c*.

[10]

Location	Value
&100	&11121314
&104	&10203040
&108	&01020304
&10C	&80706050
> &10C	&0

Figure 2.1. Memory locations

```
MOV   R10,#&110000
MOVS  R11,#&888
ORRPL R0,R10,R10,ror #2
EORMI R1,R10,R11
RSB   R2,R11,R10
BIC   R3,R11,R10,ror #13
```
(a)

```
MOV   R10, &100
MOV   R11, #1
LDRB  R0, [R10]
LDRB  R1, [R10,R11,lsl #1]
LDRB  R2, [R10, #3]!
STRB  R11, [R10],#4
MOV   R3, R10
```
(b)

```
MOV   R0, #1
CMP   R0, #-1
ADCS  R0, R0, R0, rol #8
SBCS  R1, R1, #0
ADC   R2, R2, R2
MOVGE R3, #4
```
(c)

Figure 2.2. Code fragments

	R0	R1	R2	R3	NZCV	Memory
(x)	0	&1020	&FFFFFFFF	&C	0110	mem_8[&120] = &10 mem_{32}[&300]=&FFFF0000
(a)						
(b)						
(c)						

Figure 2.3. Template for answers

3.

a) Explain the distinction between code that is condition false executed, and code not executed, in the ARM pipeline.

[6]

b) The code in *Figure 3.1* is executed from START to FINISH. Calculate the number of times each instruction is condition false executed, each instruction is condition true executed, and the total execution time in cycles of the code for each of the two sets of initial register values listed in *Figure 3.2*. Explain with reference to the ARM pipeline and memory system why the execution time in cycles is greater than the total number of instructions executed condition true or false.

[8]

c) Rewrite the code in *Figure 3.1* without using any branch instructions and give the new execution time.

[8]

d) Generalise your execution time results from part (b) & (c) to the case where the instruction at label A is replaced by n ADD instructions and that at B by m ADD instructions. Derive conditions on n, m for which the rewrite without branch instructions is faster than the original code for all initial register values.

[8]

```
START  CMP R0, R1
       BCS A
B      ADD R3, R4, R5
       B FINISH
A      ADD R6, R7, R8
FINISH
```

Figure 3.1. ARM code

R0	R1
0	0
0	1

Figure 3.2. Initial register values

4.

a) Assuming that the stack grows upwards, and the stack pointer is R13 and points to a full memory word, write as short as possible a sequence of ARM instructions which implement the stack operations detailed in *Figure 4.1*. Explain why your code would not be correct if modified to use a downwards growing stack.

[8]

b) R8-R14 are shadow registers in ARM FIQ mode. Explain how shadow registers operate, and why IRQ, FIQ, and user mode stacks are distinct.

[4]

c) *Figure 4.2* shows an IRQ exception handler located in memory immediately after the FIQ vector **FIQVEC**. Draw a diagram which details where data is stored on the IRQ mode stack when executing the instruction with label **X** of the handler routine. Where is the exception return address stored?

[8]

d) Rewrite this code to perform the same function from an FIQ interrupt exception as efficiently as possible, stating any assumptions you need to make. You may alter register use inside the handler, relocate handler code, and assume shadow registers have defined initial values, as necessary to improve performance.

[10]

```
PUSH R2
PUSH R3
PUSH R5
POP R2
POP R1
POP R0
```

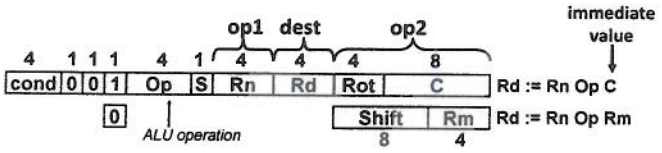
Figure 4.1. Stack operations

```
                ORG &00000018          ; define address for code output
IRQVEC          B IRQHANDLER           ; IRQ exception vector
FIQVEC          DC 0                   ; FIQ exception vector
IRQHANDLER      STMED R13!, {R3,R8-R9}
X               ADR R3, DEVICE1
                ADR R8, DEVICE2
                LDR R9, [R3]
                STR R9, [R8]
                LDMED R13!, {R3,R8-R9}
                SUBS PC, R14, #4
```

Figure 4.2. Exception handler code

Exam & Test Notes 2009/2010
ARM Instruction Set

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



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

dest := op1 op op2

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

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

Data processing

Op-codes
AND — S => set flags on result
ANDEQ
ANDS — EQ,NE,... => Condition
ANDEQS

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 rotate 2r (0 ≤ s ≤ 255, 0 ≤ r ≤ 15)	Assembler will translate negative values changing op-code as necessary Assembler will work out rotate if it exists
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 by register 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

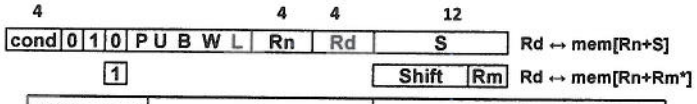
NB d & m must be different for MUL, MLA

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)



Bit in word	0	1
P	use base register addressing [Rn]	use indexed or offset address [Rn+Rm], [Rn+S]
U	subtract offset [Rn-S]	add offset [Rn+S]
B	Word	Byte
W	leave Rn unchanged if P=1	write indexed or offset address back into Rn if P=1
L	Store	Load

NB - if P=0, W=0

If P=0, always write offset address back into Rn

Data Transfer Instructions

LDR	load word
STR	store word
LDRB	load byte
STRB	store byte
LDREQB	; NB B at end
STREQB	

LDMED r13!,{r0-r4,r6,r6} ; ! => write-back to register
STMFA r13, {r2} ; no write-back
STMEQIB r2!,{r5-r12} ; note position of EQ
; higher reg nos go to/from higher mem addresses
;[E|F][A|D] empty|full, ascending|descending
;[I|D][A|B] incr|decr,after|before

```
LDR   r0, [r1]              ; register-indirect addressing
LDR   r0, [r1, #offset]     ; pre-indexed addressing
LDR   r0, [r1, #offset]!    ; pre-indexed, auto-indexing
LDR   r0, [r1], #offset     ; post-indexed, auto-indexing
LDR   r0, [r1, r2]          ; register-indexed addressing
LDR   r0, [r1, r2, lsl #shift] ; scaled register-indexed addressing
LDR   r0, address_label     ; PC relative addressing
ADR   r0, address_label     ; load PC relative address
```

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

Comparison Operations & Status Bits

- Here are ARM's register test operations:

```
CMP   r1, r2    ; set NZCV on (r1 - r2)
CMN   r1, r2    ; set NZCV on (r1 + r2)
TST   r1, r2    ; set NZ on (r1 and r2)
TEQ   r1, r2    ; set NZ on (r1 xor r2)
```

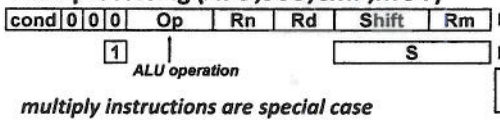
- Results of the subtract, add, and, xor are NOT stored in any registers, so destination register Rd is not used
- Status flags in the CPSR are set or cleared by these instructions as well as any data processing instruction with S at the end.

- Take **CMP r1,r2 instruction**:
  - N = 1 if MSB of (r1 - r2) is '1' (BMI,BPL)
  - Z = 1 if (r1 - r2) = 0 (BEQ,BNE)
  - C = 1 if carry-out of addition is 1 (BCS,BCC)
  - V = 1 if there is a two's complement overflow. (BVS,BVC)

Op		Operation	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

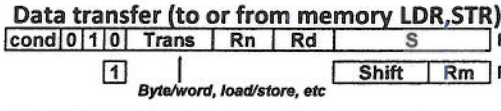
Machine Instruction Overview (1)

Data processing (ADD,SUB,CMP,MOV)

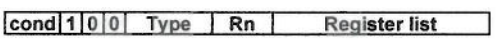


multiply instructions are special case

Data transfer (to or from memory LDR,STR)

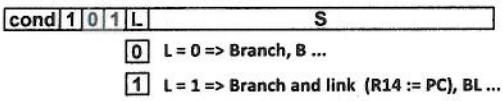


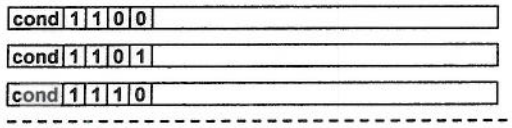
Multiple register transfer



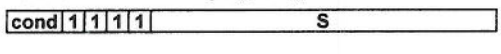
Overview (2)

Branch B, BL, BNE, BMI...





Software Interrupt (SWI)



Answer to Question 1

This is an easy (compulsory) question testing basic knowledge.

1.

a)

(i) &BF400000

(ii) $2^{74} = 1.889*10^{22}$

[8]

b) $n = 16$ - spatial & temporal - read address 0-15, write address 0-15

and $n = 1$ - temporal - write address 0

[8]

c)

```
ADD R0, R1, R1, lsl #3
ADD R0, R3, lsl #4
SUB R0, R0, R3
```

[8]

d)

```
MOV  R0, #0
MOV  R1, #1
MOV  R2, #2
MOV  R3, #3
EORS R4, R2, R3 ;R4 = 1
SUBS R5, R2, R3 ;R5 = -1 = &FFFFFFFF
ADC  R6, R0, R1 ;R6 = 1
SBC  R7, R0, R1 ;R7 = -2 = &FFFFFFFE
ORR  R8, R3, R3, lsl #4; R8 = 51 = &33 = 51
ADD  R9, R3, R3, ror #2; R9 = &C0000003 = 3,221,225,475
STRB R2, [R0], #1
STRB R1, [R0], #1
STRB R3, [R0], #1
STRB R2, [R0,#1]
MOV  R10, R0 ; R10 = 3
MOV  R0, #0
LDR  R11, [R0] ; R11 = &030102  (top byte undefined)
```

[16]

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.

a) Column (a) of Figure 2.3.

[10]

b) Column (b) of Figure 2.3

[10]

c) Column (c) of Figure 2.3

[10]

Location	Value
&100	&11121314
&104	&10203040
&108	&01020304
&10C	&80706050
> &10C	&0

Figure 2.1 - memory locations

```
MOV   R10,#&110000
MOVS  R11,#&888
ORRPL R0,R10,R10,ror #2
EORMI R1,R10,R11
RSB   R2,R11,R10
BIC   R3,R11,R10,ror #13
```
(a)

```
MOV   R10, &100
MOV   R11, #1
LDRB  R0, [R10]
LDRB  R1, [R10,R11,asl #1]
LDRB  R2, [R10, #3]!
STRB  R11, [R10],#4
MOV   R3,  R10
```
(b)

```
MOV   R0, #1
CMP   R0, #-1
ADCS  R0, R0, R0, rol #8
SBCS  R1, R1, #0
ADC   R2, R2, R2
MOVGE R3, #4
```
(c)

Figure 2.2 - code fragments

	R0	R1	R2	R3	NZCV	Memory
(x)	0	&1020	&FFFFFFFF	&C	0110	mem_8[&120] = &10 mem_{32}[&300]=&FFFF0000
(a)	&154000	0	&10F778	&800	0000	
(b)	&14	&12	&11	&107	0000	mem8[&103]=&01
(c)	&102	&FFFFFFFF	0	0	1000	

Figure 2.3 - template for answers

Answer to Question 3

This question tests knowledge of the ARM pipeline and instruction execution conditions. Part (c) is implementation, Part (a) is bookwork, the other parts are analysis.

3.

a) In the ARM pipeline instructions pass through FETCH,DECODE & EXECUTE stages. An instruction not executed may be fetched or decoded, but will be aborted before EXECUTE. An instruction executed condition false will go through execute stage but has a branch condition which evaluates false at the execute stage, and therefore the execution is inhibited.

[6]

b) T = condition true, F = condition false, 0 = not executed. Total time is larger than sum T & F because the ARM memory system requires one wait cycle on pipeline & two pipeline fill cycles on flush which happens every true executed branch.

So total time is 6 or 7 cycles.

R0	R1	instr	status	
0	0	CMP	T	6 cycles
		BCS A	T	
		B ADD R3,R4,R5	0	
		B FINISH	0	
		A ADD R6,R7,R8	T	
			0+3	
0	1	CMP	T	7 cycles
		BCS A	F	
		B ADD R3,R4,R5	T	
		B FINISH	T	
		A ADD R6,R7,R8	0	
			1+3	

[8]

c) Rewrite the code in Figure 3.1 without using any branch instructions.

CMP R0, R1
ADDCS R6,R7, R8 (or ADDHS)
ADDCC R3, R4, R5 (or ADDLO)

[8]

d) Generalise your execution time results from part b) & c) to the case where the instruction at label A is replaced by *n* instructions and that at B by *m* instructions. Derive a condition on *n*, *m* for the rewrite without branch instructions to be faster than the original code for all initial register values.

F = no false executed, T= no true executed, X = total execution time

R0,R1: F, T, X
0, 0: 0, 2+n, 5+n
0, 1: 1, 2+m, 6+m

Total execution time for rewrite is: n+m+1

=> n+m+1 < 6+m and n+m+1 < 5+n

Simplifying: n < 5, m < 4

[8]

Answer to Question 4

This questions tests understanding of ARM exception handling architecture, and how multiple register transfer instructions implement stacks. Part (b) is bookwork, (a), (d) are implementation, and (c) is analysis.

4.

a) For a descending stack the registers would be pushed & popped on the wrong order, because higher register numbers always correspond to higher addresses.

```
STMFA R13!, {R2,R3,R5}
LDMFA R13!, {R0-R2}
```

[8]

b) **R8-R14 are shadow registers in ARM FIQ mode. Explain how shadow registers operate, and why IRQ, FIQ and user mode stacks are distinct.**

Shadow registers replace normal registers in specific operating modes, storing data separate from the contents of the normal register which is "visible" in user mode. Stacks normally use R13 as SP, since this register has a separate shadow register for IRQ & FIQ modes, the three stacks are separte.

[4]

c) Figure 4.2 shows an IRQ exception handler located in memory immediately after the FIQ vector FIQVEC. Draw a diagram which details the contents of the top of the IRQ mode stack when executing instruction X of the handler routine. Where is the exception return address stored?

exception return address is R14 IRQ mode shadow register.

Top of stack:

```
SP+12:  R9 (user)
 SP+8:  R8 (user)
 SP+4:  R3 (user)
        <---IRQ SP
```

[8]

d)

Assumptions: R10, R8 in FIQ Mode are initially set to the addresses of DEVICE1 & DEVICE2.

```
FIQVEC    LDR R9, [R10]
          STR R9, [R8]
          SUBS PC, R14, #4
```

[10]