

Master copy

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

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2007

EEE Part II: MEng, BEng and ACGI

INTRODUCTION TO COMPUTER ARCHITECTURE

Wednesday, 23 May 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): Constantinides, G

© University of London 2007

Special information for invigilators:

The booklet Exam Notes 2007 should be distributed with the Examination Paper.

Information for candidates:

The booklet Exam Notes 2007, 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) Perform the following numeric conversions:

- (i) $1111_{(16)}$ into decimal
- (ii) $1111_{(10)}$ into hexadecimal
- (iii) $E2_{(16)}$ two's complement 8 bit hexadecimal into signed decimal
- (iv) $-3_{(10)}$ into 8 bit two's complement octal.

[4]

b) Derive in hexadecimal the IEEE-754 representations for:

- (i) 20.5
- (ii) -2.0625

What is the maximum error when converting a real number close to 20.5 into the nearest possible IEEE-754 number?

[6]

c) The ARM assembler in Figure 1.1 is executed from START with r0 – r14 initially 0. For each instruction executed until label FINISH, determine any changes in value of registers R0-R14, memory locations, and condition flags.

[5]

d) A given RISC CPU architecture has identical timing for all non-branch instructions. A throughput of 10^7 instructions per second and instruction latency of 400ns is observed during execution of a section of code without branches. Explain how this throughput is possible. Calculate the CPU throughput when executing code with one branch every three instructions, and 20% of branches correctly predicted.

[5]

```
START SUBS    R0, R0, #12
      ADD     R1, R0, R0, lsl #3
      RSB     R3, R1, #0
      STRB    R0, [R3, #4]!
      STRBPL  R0, [R3], #10
FINISH
```

Figure 1.1

2.

- a) In the ARM architecture state, giving reasons, what are the conditions on PSR flags after an arithmetic instruction that indicate:

- (i) unsigned overflow
- (ii) two's complement signed overflow

[2]

- b) Memory locations $2000_{(16)} - 20FF_{(16)}$ and $2100_{(16)} - 21FF_{(16)}$ contain two 64 word (2048 bit) two's complement numbers stored as a sequence of words least significant word first. Write efficient and compact ARM code using the minimum number of registers which will add these two numbers together, putting the result, stored similarly, in locations $2200_{(16)} - 22FF_{(16)}$.

[4]

- c) What is the speed of your code in cycles per byte written, assuming the timing information in Exam Notes 2007?

[3]

- d) How would your answer to b) change if the numbers were stored in memory as a sequence of bytes LSB first and ordered:

- (i) little-endian
- (ii) big-endian

You need not write the new code.

[2]

- e) Using LDM/STM instructions, with no limitation on registers used, write a faster solution to the problem in b).

[4]

3.

- a) Explain, with reference to ARM CPU datapath, why the ARM instructions in Figure 3.1 take the specified number of cycles to execute. You may assume that fetch and decode stages of execution are pipelined and therefore do not normally contribute to the total time.

[3]

- b) State the difference between ARM rotate right, arithmetic shift right, logical shift right operations. Why are rotate left and arithmetic shift left operations unnecessary given logical shift left?

[3]

- c) Write efficient ARM code to set R1 equal to R0 multiplied by 17, explaining how your code works.

[3]

- d) Write a single efficient fragment of ARM code which sets all the bits of R1 as specified in Figure 3.2. Note that in this Figure (*a:b*) denotes the bit field from bit *a* to bit *b* and **NOT** performs a bitwise inversion.

[6]

Operation	Cycles
ADD Ra, Rb, Rc, lsl n	1
ADD Ra, Rb, Rc, lsl Rn	2

Figure 3.1

R1(31:30) := R0(1:0)
R1(29:27) := R0(31:29)
R1(26:20) := R0(26:20)
R1(15:0) := NOT R0(15:0)
R1(19:16) := 1111 ₍₂₎

Figure 3.2

4. This question relates to the ARM assembler code fragment TEST in Figure 4.1.
- a) Write simplified pseudo-code equivalent to TEST. [4]
 - b) What are the maximum and the minimum number of cycles taken by TEST, assuming the instruction timing given in the 2007 Exam notes? [4]
 - c) Rewrite TEST using ARM conditional execution to minimise the total code size. [4]
 - d) You are told that an ARM CPU has pipeline length of 5, and correct branch prediction for all unconditional branches taken, and conditional branches not taken. All non-branch instructions have the same timing as ARM7. State, giving reasons, your assumptions about branch timing on the new CPU. Calculate the maximum and minimum code execution time for TEST on the new CPU. [3]

```

TEST
    CMP R0, R1
    BCS L1
    ADD R2, R3, R4
    ADDS R2, R2, R5
    B L2
L1    RSB R2, R3, R3, lsl #2
    ADDS R2, R2, R5, lsl #1
L2    BEQ L3
    RSB R2, R2, #0
    B L4
L3    ADD R2, R2, #100
L4

```

Figure 4.1

EXAM NOTES 2007

Introduction to Computer Architecture Principles of Computing

Memory Reference & Transfer Instructions

LDR load word
STR store word
LDRB load byte
STRB store byte
LDREQB ; note position
; of EQ
STREQB

LDMEQ r13,{r0-r4,r6,r6}; ! => write-back to register
STMFA r13,{r2}
STMEQB r2,{r5-r12}; note position of EQ
; higher reg nos go to/from higher mem addresses always
[E]F[A]D] empty/full, ascending/descending
[I]D][A]B] :incr/decr,after/before

LDR r0, [r1]
LDR r0, [r1, #offset]
LDR r0, [r1, #offset]!
LDR r0, [r1], #offset
LDR r0, [r1, r2]
LDR r0, [r1, r2, lsl, #shift]
LDR r0, address_label
ADR r0, address_label

; register-indirect addressing
; pre-indexed addressing
; pre-indexed, auto-indexing
; post-indexed, auto-indexing
; register-indexed addressing
; scaled register-indexed addressing
; PC relative addressing
; load PC relative address

R2.1

ARM Data Processing Instructions Binary Encoding

Opcode [24:21]	Mnemonic	Meaning	Effect
0000	AND	Logical bit-wise AND	Rd := Rn AND Op2
0001	EOR	Logical bit-wise exclusive OR	Rd := Rn EOR Op2
0010	SUB	Subtract	Rd := Rn - Op2
0011	RSB	Reverse subtract	Rd := Op2 - Rn
0100	ADD	Add	Rd := Rn + Op2
0101	ADC	Add with carry	Rd := Rn + Op2 + C
0110	SBC	Subtract with carry	Rd := Rn - Op2 + C - 1
0111	RSC	Reverse subtract with carry	Rd := Op2 - Rn + C - 1
1000	TST	Test	Sec on Rn AND Op2
1001	TEQ	Test equivalence	Sec on Rn EOR Op2
1010	CMN	Compare	Sec on Rn - Op2
1011	CMN	Compare negated	Sec on Rn + Op2
1100	ORR	Logical bit-wise OR	Rd := Rn OR Op2
1101	MOV	Move	Rd := Op2
1110	BIC	Bit clear	Rd := Rn AND NOT Op2
1111	MVN	Move negated	Rd := NOT Op2

R2.3

Conditions Binary Encoding

Opcode [31:28]	Mnemonic extension	Interpretation	Status flag state for execution
0000	EQ	Equal / equals zero	Z set
0001	NE	Not equal	Z clear
0010	CS/HS	Carry set / unsigned higher or same	C set
0011	CC/LO	Carry clear / unsigned lower	C clear
0100	MI	Minus / negative	N set
0101	PL	Plus / positive or zero	N clear
0110	VS	Overflow	V set
0111	VC	No overflow	V clear
1000	HI	Unsigned higher	C set and Z clear
1001	LS	Unsigned lower or same	C clear or Z set
1010	GE	Signed greater than or equal	N equals V
1011	LT	Signed less than	N is not equal to V
1100	GT	Signed greater than	Z clear and N equals V
1101	LE	Signed less than or equal	Z set or N is not equal to V
1110	AL	Always	any
1111	NV	Never (do not use!)	none

R2.2

Data Processing Operand 2

Examples

ADD r0, r1, op2
MOV r0, op2
CMP r0, #-1
EOR r0, r1, r2, lsl, #10
RSB r0, r1, r2, asr, r3

Op2	Conditions	Notes
Rm		
#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	(1 ≤ s ≤ 31)	rrx always sets carry
Rm, rrx #1	shift => lsl, lsl, asr, asl, ror	ror sets carry if S=1
Rm, shift Rs	shift => lsl, lsl, asr, asl, ror	shifts do not set carry
		shift by register value (takes 2 cycles)

R2.4

Multiply Instructions



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

ARM3 and above

MUL rd, rm, rs multiply (32 bit) Rd := (Rm*Rs)[31:0]
 MULA rd, rm, rs, rn multiply-acc (32 bit) Rd := (Rm*Rs)[31:0] + Rn
 UMLAL r1, r1, r1, r1 unsigned multiply (Rh:Rl) := Rm*Rs
 UMLAL r1, r1, r1, r1 unsigned multiply-acc (Rh:Rl) := (Rh:Rl)+Rm*Rs
 SMLAL r1, r1, r1, r1 signed multiply (Rh:Rl) := Rm*Rs
 SMLAL r1, r1, r1, r1 signed multiply-acc (Rh:Rl) := (Rh:Rl)+Rm*Rs

ARM7DM core and above

l5ve - 2-Apr-07 ISE/EE2 Introduction to Computer Architecture 2.5

NB: r1, r1, r1, r1 must be different for MUL, MULA

Exceptions & Interrupts

Exception	Return	Shadow registers
SWI or undefined instruction	MOVS pc, R14	R13, R14, SPSR
IRQ, FIQ, prefetch abort	SUBS pc, r14, #4	as above + R8-R12
Data abort (needs to rerun failed instruction)	SUBS pc, R14, #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

R2.7

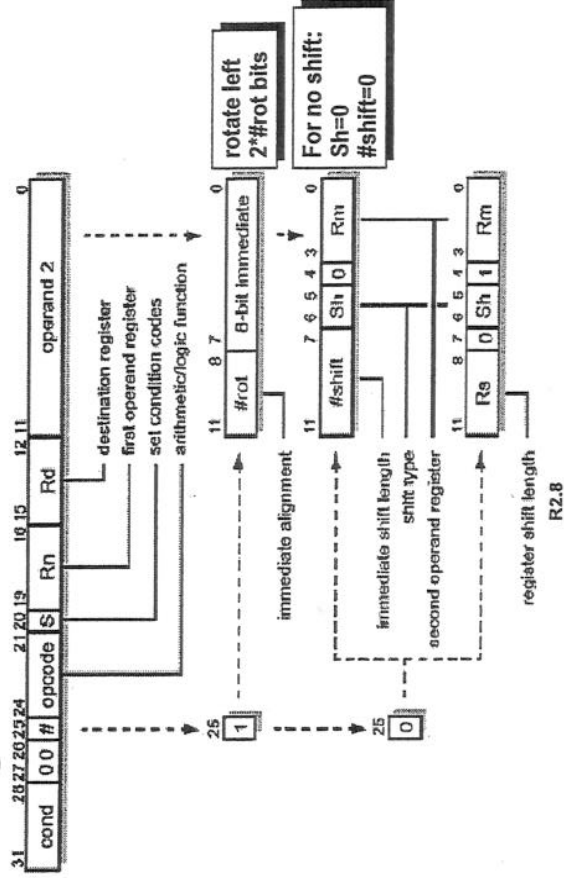
Assembly Directives

SIZE EQU 100 ; defines a numeric constant
 BUFFER % 200 ; defines bytes of zero initialised storage
 ALIGN ; forces next item to be word-aligned
 MYWORD DCW &80000000 ; defines word of storage
 MYDATA DCD 0, 1, &ffff0000, &12345 ; defines one or more words of storage
 TEXT = "string", &0d, &0a, 0 ; defines one or more bytes of storage. Each
 ; operand can be string or number in range 0-255
 ; assembles to instructions that set r0 to immediate
 ; value numb - numb may be too large for a MOV operand
 LDR r0, =numb ;

NB:
 & prefixes hex constant: &3FFF
 Case does not matter anywhere (except inside strings)

R2.6

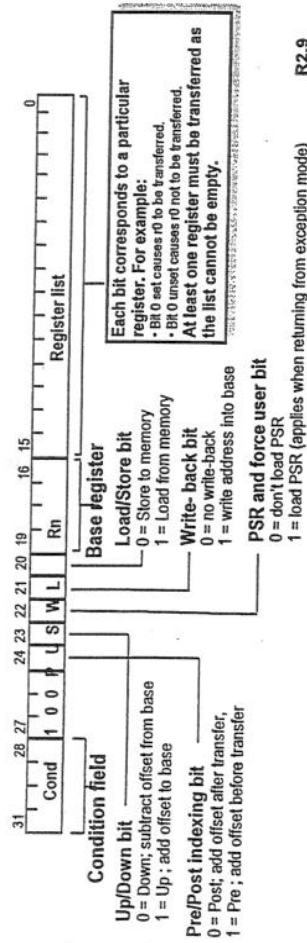
Data Processing Instruction Binary Encoding



R2.8

Multiple Register Transfer Binary Encoding

- ❖ The Load and Store Multiple instructions (LDM / STM) allow between 1 and 16 registers to be transferred to or from memory.

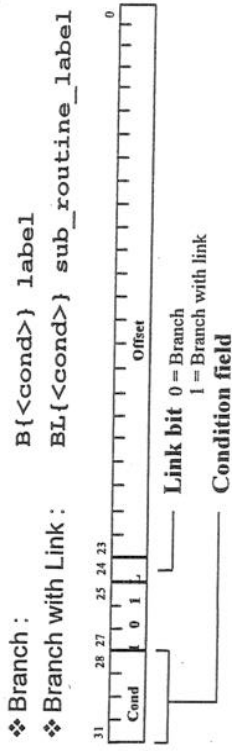


R2.9

Instruction Set Overview

31	28	27	26	25	24	23	22	21	20	19	16	15	12	11	8	7	3	4	3	0					
Cond	0	0	1	OpCode				S	Rn				Rd	Operand 2				Data Processing PGR Transfer							
Cond	0	0	0	0	0	0	0	A		S	Rd	Rn	Hs	1		0	0	1	Rm		Multiply				
Cond	0	0	0	1	0	B		0	0	Rn		Rd	0	0	0	0	1	0	0	1	Rm	Single Data Swap			
Cond	0	1	1	P	U	B	W	L	Rn				Rd	offset				Single Data Transfer							
Cond	0	1	1	XXXXXXXXXXXXXXXXXXXX				1				XXXX				Undefined									
Cond	1	0	0	P	U	S	W	L	Rn				Register List				Block Data Transfer								
Cond	1	0	1	1	offset																Branch				
Cond	1	1	0	P	U	N	W	L	Rn				CRd	CP#	offset				Coproc Data Transfer						
Cond	1	1	1	0	CP OpC				CRn				CRd	CP#	0	CRm				Coproc Data Operation					
Cond	1	1	1	0	CP OpC				L	CRn				Rd	CP#	1	CRm				Coproc Register Transfer				
Cond	1	1	1	1	Ignored by processor														Software Interrupt						

Branch Instruction Binary Encoding



- ❖ The offset for branch instructions is calculated by the assembler:
 - + By taking the difference between the branch instruction and the target address minus 8 (to allow for the pipeline).
 - + This gives a 26 bit offset which is right shifted 2 bits (as the bottom two bits are always zero as instructions are word – aligned) and stored into the instruction encoding.
 - + This gives a range of ± 32 Mbytes.

R2.10

ARM 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) (if instruction condition is TRUE – otherwise 1 cycle)
Any instruction, with condition false	1
data processing (all except register-valued shifts)	1
data processing (register-valued shifts)	2
LDR, LDRB	4
STR, STRB	4
LDM (n registers)	n+3 (+3 if PC is loaded)
STM (n registers)	n+3
B, BL	4
Multiply	7-14 (varies with architecture & operand values)

Solutions for E1.9/E2.19

A=analysis, D=design, C=calculated solution using taught theory, B=bookwork
 NB - marking will be 1 mark for every 2% indicated on question paper (max 50 marks)
 Solution to Question 1

36 minutes for the question => 9 minutes each part

a)

i) 4369

ii) 457

iii) -30

iv) $-3_{(8)} [8 \text{ bits}] = 375$

[1 mark each]

[4C]

b)

see eeee emmm mmmm mmmm mmmm mmmm mmmm

$20.5 = 1010.1 = 1.0101 \times 16 \rightarrow \text{sign}=0, \text{exp}=83, \text{mant}=280000 \rightarrow 41A40000$ [2 marks]

$-2.0625 = -10.0001 = -1.00001 \times 2 \rightarrow \text{sign}=1, \text{exp}=80, \text{mant}=040000 \rightarrow C0040000$ [2 marks]

+1 mantissa = $2^{-23} \rightarrow \text{quantisation} = 2^{-19} \rightarrow \text{error} = 2^{-20}$

[1 mark 2^{-19} , 2 marks 2^{-20}]

[6C/A]

c)

START SUBS	R0, R0, #11	r0=-11, C=0, N=1, Z=0, V=0
ADD	R1, R0, R0, lsl 3	r1=-99
RSB	R3, R1, #0	r3=99
STRB	R0, [R3, #2]!	Mem8[101] := -11, R3 := 101
STRBPL	R0, [R3], #10	not executed
FINISH		

[5B]

d)

10^7 IPS, 400ns latency => pipeline of 4

0.333×0.8 = frequency of pipeline stalls

Overall throughput = $10^7 / (1 + 4 \times 0.333 \times 0.8) = 4.84$ MIPS

[5C]

Solution to Question 2

27 minutes for the question

a)

(i) unsigned oflow => C=1. Unsigned ranges 0-255, > 255=> carry is set

(ii) signed overflow => V=1.

V= carry into sign bit xor C. C=> error $+2^{32}$, carry in to sign bit = error $2*(-2^{31})=-2^{32}$.

Hence no overflow if carry in = 0, C=0, or carry in=1, C=1

[2B]

b)

```

MOV R0, #2000
ADDS R0, R0, #0 ; clear carry bit
MOV R3, #64
LOOP
    LDR R1, [R0]
    LDR R2, [R0]
    ADDCS R1, R1, R2 ; carry in and out in C
    STR R1, [R0], #4 ; postincrement pointer for next iteration
    SUB R3, R3, #1 ; subtract without bsetting C
    ORRS R3, #0 ; test for 0 without setting C
    BNE LOOP
    
```

[4D]

c)

(4+4+1+4+1+1+4) cycles for 4 bytes written => 19/4=4.75 cycles/byte

Small correction downwards due to BNE not being executed on last loop allowed but not required.

[3C]

d)

(i) no change.

(ii) would have to read/write individual bytes, reversing normal order within each word, or else swap bytes in registers

[2A]

Solutions for E1.9/E2.19

e)

```
MOV    R0, #&2000
ADDS   R1, R0, #&1000 ; clear carry bit
ADD    R2, R1, #&1000;
MOV    R11, #16
; R0, R1, R2 address the three numbers

LOOP
LDMIA  {R3,R4,R5,R6}, R0!
LDMIA  {R7,R8,R9,R10}, R1!
ADDCS  R3, R3, R7
ADDCS  R4, R4, R8
ADDCS  R5, R5, R9
ADDCS  R6, R6, R10
STMIA  {R3,R4,R5,R6}, R2!
ADD    R0, R0, #4
SUB    R11, #4
ORRS   R11, #0
BNE    LOOP
```

[4D]

Solution to Question 3

27 minutes for the question

a)

In the execute stage 2 read ports + 1 write port are available to register file (+ one special read/write port for pc INCREMENT). Thus the first instruction can be implemented in one cycle. Note that ALU & shifter are both contained in datapath so can be used in parallel. The second instruction requires read access to three registers so must require two cycles.

[3B]

b)

The leftmost bit equals b0, b31, 0 for ror, asr, lsr respectively [1 mark]. Ror n = ror 32-n [1 mark]. asl n = lsl n [1 mark].

[3B]

c)

ADD R1, R0, R0 lsl 4

$17 = 1 + 16 = 1 + 2^4$. Multiplication by 2^4 can be implemented with shift of 4 bits left.

[3B/A]

d)

ependently, without combining. I gave some credit for this if correct]

```
MOV R1, R0 ror 2
AND R1, R1, #&F8000000
BIC R2, R0, #&F8000000
EOR R2, R2, #&FFFF
ORR R2, R2, #&000F0000
ORR R1, R1, R2
```

[6D]

Solution to Question 4

27 minutes for the question

a)

[1 mark for each condition correct, 1 mark for then+else part of each IF correct]

If $R0 \geq R1$ (unsigned comparison)

Then (L1)

$R2 := R3 * 3 + R5 * 2$

Else

$R2 := R3 + R4 + R5$

End

If $R2 = 0$

Then

$R2 := R2 + 100$

Else

$R2 := -R2$

End

[4A]

b)

Branches taken = 4 cycles (B). First IF = $3 + B/4 + B$, second IF = $1 + B/2 + B$

Hence $\min = 4 + 2B = 12$, $\max = 6 + 2B = 14$

[4A]

c)

To minimise code size make all code conditionally executed to reduce branches:

TEST

```

CMP R0, R1
BCS L1
ADDCC R2, R3, R4
ADDSCC R2, R2, R5
RSBCS R2, R3, R3, lsl #2
ADDSCS R2, R2, R5, lsl #1
RSBNE R2, R2, #0
ADDEQ R2, R2, #100
    
```

[4 marks for correct code] This results in 8 cycles.

[4D]

d)

Assume all instructions (including B) are one cycle except BCS/BEQ when taken. Assume these are 6 cycles. (will accept 5).

Min time is when both are not taken: 8 cycles.

Max time is when both are taken: $4 + 12 = 16$ cycles. (14 cycles if branch taken time = 5)

[3A]