

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)

- (i) What is the decimal value represented by 0x4B500001 in IEEE-754?
- (ii) What is the decimal value represented by 0x4B500002 in IEEE-754?
- (iii) A real number x has nearest IEEE-754 representation 0x4BD00001. What in decimal are the highest and lowest possible values of x ?

[6]

b) 32 bit little-endian ARM memory is as shown in Figure 1.1. Give the value of each of the expressions below *in hexadecimal*, or if the expression is not valid state what action an ARM CPU would take on a load instruction referencing the expression.

- (i) **mem₃₂[0x104]**
- (ii) **mem₈[0x101]**
- (iii) **mem₃₂[0x102]**

[3]

c) Write efficient separate ARM code fragments which will perform the specified operations, ignoring overflow. You may assume that $C = 0$ on entry to the fragment:

- (i) $R1 := 65 * R2$
- (ii) $R3 := 2 * R4 - R3 - 1$

[6]

d) A 100MHz ARM7 CPU executes branch instructions 30% of the time, and 75% of branch instructions are condition false executed. Assuming no instruction causes pipeline stalls other than branches what is the CPU throughput? Give one specific reason why you would expect typical CPU throughput with these branch instruction execution statistics to be less than this.

[5]

Word address in <i>hexadecimal</i>	Value in <i>unsigned decimal</i>
0x100	1003
0x104	216357

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 as required the values in R0-R4 and the condition codes after execution of the code fragment. State the execution time in cycles of the instructions. Write your answers using as a template a copy of the table in Figure 2.3, *deleting the example row labelled (x)* which indicates the required format of your answer. Each answer must be written with register values in signed *decimal* and condition codes in *binary*. The example row (x) shows this. Where an answer is labelled n/a you need not give it.

a) Code as in Figure 2.2a.

[7]

b) Code as in Figure 2.2b.

[6]

c) Code as in Figure 2.2c.

[7]

Location (word)	Value
0x100	0x04030201
0x104	0x08070605
0x108	0x0F0C0D0A
>0x108	0

Figure 2.1. Memory locations

	MOV R1, #10 MOV R2, #3 MOV R3, #0 MOVS R2, R2, ror #1 ADDMI R3, R3, R1 MOVS R2, R2, ror #1 ADDMI R3, R3, R1, lsl #1 MOVS R2, R2, ror #1 ADDMI R3, R3, R1, lsl #2	
MOV R4, #1 MOV R5, #-2 ADDS R0, R4, R5 SBCS R1, R5, R4 RSB R2, R4, #80000000 MVNS R3, #0		MOV R4, #0x104 ANDS R3, R4, #7 LDRB R0, [R4, #3] LDRB R1, [R4, #-1] LDR R2, [R4, R3, lsl #1]
(a)	(b)	(c)

Figure 2.2. Code fragments

	R0	R1	R2	R3	R4	NZCV	Time
(x)	0	222	-33	4		0110	3
(a)							n/a
(b)	n/a				n/a		
(c)							

Figure 2.3. Template for answers

3.

- a) Figure 3.1 shows a listing of a subroutine with inputs R0, R1 which changes memory locations. Suppose that R0, R1, R13 have values a , b , c when the subroutine is called. State precisely what changes to memory locations, other than stack, are made by the subroutine, and what is the total execution time of the subroutine including the branch and link instruction that calls it.

[6]

- b) If $R13 = c$ on entry to the subroutine, state what changes to memory locations and registers are made by instructions 1, 8 and 9 of Figure 3.1, and explain why these are necessary.

[4]

- c) Show how instructions 2-7 of Figure 3.1 can be implemented more efficiently using LDM/STM instructions, and state the number of execution cycles saved by this optimisation.

[5]

- d) Using your optimised code from part c, rewrite the subroutine, with appropriate changes to instructions 1 and 8. What, overall, is the saving in execution time of the optimised subroutine compared with the code in Figure 3.1?

[5]

1	MOVE	STMEA	R13!, {R2}
2		LDR	R2, [R1]
3		STR	R2, [R0]
4		LDR	R2, [R1, #4]
5		STR	R2, [R0, #4]
6		LDR	R2, [R1, #8]
7		STR	R2, [R0, #8]
8		LDMEA	R13!, {R2}
9		MOV	PC, R14

Figure 3.1.

Introduction to Computer Architecture - Answers 2013

All questions are compulsory, questions are weighted equally.

Answer to Question 1

Qa,b,d are easy questions testing basic knowledge & understanding. C tests ability to optimise code (i) is bookwork.

1.

a)

- | | | | |
|-------|---|---|---|
| (i) | 13631489 | $(2^{23} * 0b1.101\ 000\ 000\ 000\ 0001)$ | 2 |
| (ii) | 13631490 | $(2^{23} * 0b1.101\ 000\ 000\ 000\ 0010)$ | 2 |
| (iii) | As (i) but exp is one larger => X2. | | |
| | Note that the exp=24, so difference between successive IEEE-754 numbers in this range is 2. Half of this (1) is the max deviation from the nominal value: | | |
| | $27262977 - 27262979\ (13631489 * 2 \pm 1)$ | | 2 |

Common mistakes were to get exponent wrong, or to give an approximate answer (calculator will not have enough precision unless used carefully)

[6]

b)

- | | | |
|-------|---|---|
| (i) | 0x 34D25 | 1 |
| (ii) | 0x03 | 1 |
| (iii) | invalid, will cause memory abort exception | 1 |

Many did not identify that (iii) was invalid, and that it would result in an exception.

[3]

c)

- | | | |
|------|---|---|
| (i) | ADD R1, R2, R2, lsl #6 | 3 |
| (ii) | RSC R3, R3, R4, lsl #1 ; works because C=0 | 3 |
- Many took 2 or mor instructions to implement (ii)

[6]

d)

- | | |
|--|---|
| $T = 100\text{MHz} / (1 + 3 * 0.3 * 0.25) = 81.6\text{MHz}$ (or 0.816 instructions/cycle) | 3 |
| Typical code will contain many LDR/STR instructions which also cause pipeline stalls, so decreasing throughput. | 2 |

[5]

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.

(b) test understanding of shift and rotate instructions.

(c) tests understanding of LDR/LDRB instructions.

Marks are awarded 1 per answer not flagged n/a. Note that condition codes CV and NZ are counted as separate answers.

Mistakes here were too varied to categorise. Worth noting that many lost marks with incorrect condition codes.

Location (word)	Value
0x100	0x04030201
0x104	0x08070605
0x108	0x0F0C0D0A
>0x108	0

Figure 2.1. Memory locations

	MOV R1, #10	
	MOV R2, #3	
	MOV R3, #0	
MOV R4, #1	MOVS R2, R2, ror #1	
MOV R5, #-2	ADDMI R3, R3, R1	MOV R4, #0x104
ADDS R0, R4, R5	MOVS R2, R2, ror #1	ANDS R3, R4, #7
SBCS R1, R5, R4	ADDMI R3, R3, R1, lsl #1	LDRB R0, [R4, #3]
RSB R2, R4, #80000000	MOVS R2, R2, ror #1	LDRB R1, [R4, #-1]
MVNS R3, #0	ADDMI, R3, R3, R1, lsl #2	LDR R2, [R4, R3, lsl #1]

(a)

(b)

(c)

Figure 2.2. Code fragments

	R0	R1	R2	R3	R4	NZCV	Time	Marks
(a)	-1	-4	0x4C4B3FF7999999	0xFFFFFFF-1	1	1010	n/a	7

(b)	n/a	10	0x60000001610612736	30	n/a	00	9	6
(c)	8	4	0	4	0x104	00	14	7

Figure 2.3. Template for answers

Answer to Question 3

This question tests understanding of subroutines, and operation of LDM/STM instructions, as well as how to optimise sequential access to memory.

a)

```

mem32[a] := mem32[b]
mem32[a+4] :=
mem32[b+4]
mem32[a+8] :=
mem32[b+8]
execution time = 4 (STM)
+ 6*4 (LDR/STR) +
4(LDM) + 4 (MOV) + 4
(BL) = 40
[allow 1/2 for time if
working shows STM,
LDR/STR, and LDM time
is all correct]

```

This question was well answered by those who worked out what the memory transfer was. Some gave incorrect answers in involving registers.

b)

```

1:
mem32[c] := R2
R13 := R13+4 (push R2)
8/9:
R2 := mem32[c]
R13 := R13 - 4 (pop R2)
PC := R14
R2 is pushed onto an
empty ascending stack
with SP R13, and then
popped back off it at end.
Finally a branch is made
to the address stored in
R14 (which will be the
return address written by
a BL instruction).
R2 must be saved on
stack because it is used as
working register by
subroutine.

```

Many missed the reasons

for the operations

c)

```

LDMIA R1, {R2,R3,R4}
STMIA R0, {R2,R3, R4}
24 - 12 = 12 cycles

```

Some got the direction of mem transfers wrong here. 2

Common mistake was to increment the base registers R1,R0, that must not happen 1

d)

```

1      MOVE      2
      STMEA      R13!,
      {R2,R3,R4}
2
      LDMIA      R1,
      {R2,R3,R4}
3
      STMIA      R0,
      {R2,R3,R4}
4      [6]
      LDMEA      R13!,
      {R2,R3,R4}
5      MOV      PC, R14

```

**Total time saved = 12 - 4
(longer STMED/LDMED) = 8
cycles**

Most found this question quite easy. 2

1

1
[4]