master - July 08

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

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2008

EEE Part II: MEng, BEng and ACGI

INTRODUCTION TO COMPUTER ARCHITECTURE

Tuesday, 27 May 10:00 am

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

Special information for invigilators:

The booklet Exam Notes 2008 should be distributed with the Examination Paper.

Information for candidates:

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

The booklet Exam Notes 2008, 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) $432.C_{(16)}$ fixed point hexadecimal into decimal

(ii) $9999_{(10)}$ into hexadecimal

(iii) $70_{(16)}$ 8 bit two's complement into signed decimal

(iv) $3FF_{(16)}$ 10 bit two's complement into signed decimal

(v) $3FF_{(16)}$ 11 bit two's complement into signed decimal

[8]

b) Give the decimal equivalent of the following machine words interpreted as IEEE-754 floating point numbers:

(i) $4F000000_{(16)}$

(ii) $BE900000_{(16)}$

Assume that hardware is available in all 32 bit CPUs which performs an unsigned comparison of the bottom 31 bits of the machine word, as part of the logic necessary to perform signed and unsigned integer comparisons. Explain, with appropriate examples, why therefore IEEE-754 numbers do not use two's complement representation in the exponent field.

[12]

c) The ARM assembly code in Figure 1.1 is executed from location 0 with R0 – R14 initially 0. State the values of all registers after the execution of this code fragment. If the FETCH stage of the instruction at location 0 executes in machine cycle 0, determine the machine cycle in which each of the first four instructions starts and completes execution on the ARM-7 architecture.

[8]

d)

(i) Suppose the number in R0 is initially *x*. After execution of the ARM instructions in Figure 1.2, assuming no overflow, what is the value of R1?

(ii) Write an ARM assembly code fragment which executes in 2 cycles and sets R1 := 105*R0, ignoring overflow, and using the minimum number of registers. Note that 105 = 7*15.

(iii) Suppose that R0 is unsigned and R1 signed. What is the maximum value in R0 for which the multiplication in (ii) does not overflow?

[12]

```
&00     MOV  R0, #2
&04     ADD  R1, R0, R0
&08     SUB  R2, R0, R0, lsl R1
&0C     ORR  R3, R1, R0
&10     AND  R4, R1, R0
&14
```

Figure 1.1

```
ADD  R1, R0, R0, lsl #2
ADD  R2, R0, R0, lsl #5
SUB  R1, R1, R2
```

Figure 1.2

2. For each part a - c below record the value of R0-R4, the condition codes, and any *changed* memory locations, after execution of the specified code. You must assume in each case that initially all registers and flags have value 0, and the memory contains values as shown in Figure 2.1. Write your answers using as a template a copy of the table in Figure 2.3. Each answer may be written in either hexadecimal, decimal, or as powers of 2, this is illustrated in the row labelled 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	&00000001
&104	&FFFFFFFF
&108	&01020304
&10C	&00000108
> &10C	&0

Figure 2.1 - memory locations

```
MOVS   R0,#-2
MOV    R1, #3
MOV    R2, #&100
ADDMI  R3, R1, R0
EOR    R4, R0, R1
STR    R1, [R2, #4]
```

(a)

```
MOV    R0, #&108
MOV    R1, #&1
LDR    R2, [R0]
LDRB   R3, [R0,#3]!
STRB   R3, [R1]
```

(b)

```
MOV    R0, #&100
ADCS   R1, R0, R0, lsl 23
ADCS   R2, R1, R1
ADD    R3, R2, R2
MOV    R4, R0, ror 16
STMED  R0!, {R3}
```

(c)

Figure 2.2 - code fragments

	r0	r1	r2	r3	r4	NZCV	Memory
x)	0	&1020	$2^{30}+2^{8}$	10	-3	0110	$mem_{32}[\&120] = 10$
a)							
b)							
c)							

Figure 2.3 - template for answers

3. A 32 bit ARM CPU makes a sequence of word cache operations 1, 2, 3, ... as in Figure 3.1. Assume that all cache lines are initially invalid.

a) Suppose the CPU has a direct mapped cache with total size of 4 words (16 bytes) which contains two lines:

(i) For the first five operations, $i = 1, 2, 3, 4, 5$, state the tag, index and word select, and whether the operation is a hit or miss.

(ii) For the operations $i = 6$ - 16 state whether the operation is a hit or miss.

[10]

b) Suppose the CPU has a direct mapped cache with four lines each of one word (four bytes):

(i) For the first five operations, $i = 1, 2, 3, 4, 5$, state the tag, index and word select, and whether the operation is a hit or miss.

(ii) For the operations $i = 6$ - 16 state whether the operation is a hit or miss.

[10]

c) The sequence of word cache operations is now as in Figure 3.2 with all cache lines initially invalid. Suppose the cache is write-back with two lines each of two words (8 bytes). State the memory operations required to implement each cache operation using the notation illustrated in Figure 3.3 which indicates reads of locations &4, &8, followed by writes to &C,&10.

[10]

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Read/Write	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
Memory address	&0	&4	&8	&C	&10	&14	&18	&14	&10	&C	&8	&4	&0	&4	&8	&C

Figure 3.1

i	1	2	3	4	5	6	7	8	9	10
Read/Write	R	W	R	R	W	W	R	W	W	R
Memory address	&0	&4	&8	&C	&10	&14	&C	&10	&0	&4

Figure 3.2

R4, R8, WC, W10

Figure 3.3

4. This question relates to the ARM assembler code fragment TEST in Figure 4.1.

a) Write simplified pseudo-code equivalent to TEST. Explain why the total number of cycles executing TEST is always more than the number of instructions executed.

[8]

b) Rewrite the code in Figure 4.1 so that it always executes in 4 machine cycles on an ARM-7 architecture.

[8]

c) Suppose that initially all registers are zero. Draw a diagram of the execution of TEST under these conditions which shows each stage of the ARM-7 pipeline for each instruction.

[8]

d) Suppose one in four instructions executed is a branch, and that an ARM-X processor with pipeline length 5 running at 100MHz correctly predicts 60% of all branches. If all non-branch instructions execute in a single cycle calculate the average instruction rate of ARM-X, stating any assumptions you make to obtain your answer.

[6]

```
TEST
        CMP     R1, R0
        CMPEQ   R3, R2
        BEQ     T1
        RSB     R5, R4, #0
        B T2
T1
        MOV     R5, R4
T2
```

Figure 4.1

EXAM NOTES 2008

Introduction to Computer Architecture

Memory Reference & Transfer Instructions

```
LDR     load word
STR     store word
LDRB    load byte
STRB    store byte

LDREQB ; note position
; of EQ
STREQB
```

```
LDMED r13!,{r0-r4,r6,r6} ; ! => write-back to register
STMFA r13, {r2}
STMEQIB 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]                ; 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
```

R2.1

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

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	Set on Rn AND Op2
1001	TEQ	Test equivalence	Set on Rn EOR Op2
1010	CMP	Compare	Set on Rn - Op2
1011	CMN	Compare negated	Set 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

Op-codes

AND
ANDEQ
ANDS
ANDEQS

S=> set flags

R2.3

Data Processing Operand 2

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		
#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 sets carry ror sets carry if S=1 shifts do not set carry
Rm, shift Rs	shift => lsr,lsl,asr,asl,ror	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
- Note that some multiply instructions have 4 register operands!
  - + Multiply instructions must have **register** operands, **no immediate constant**
  - + Multiplication by small constants can often be implemented more efficiently with data processing instructions – see Lecture 10.

NB d & m must be different for MUL, MULA

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
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

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
        LDR r0, =numb               ; assembles to instructions that set r0 to immediate
                                    ; value numb -- numb may be too large for a MOV operand
                                    ;
```

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

R2.6

Exceptions & Interrupts

Exception	Return
SWI or undefined instruction	MOVS pc, R14
IRQ, FIQ, prefetch abort	SUBS pc, r14, #4
Data abort (needs to rerun failed instruction)	SUBS pc, R14, #8

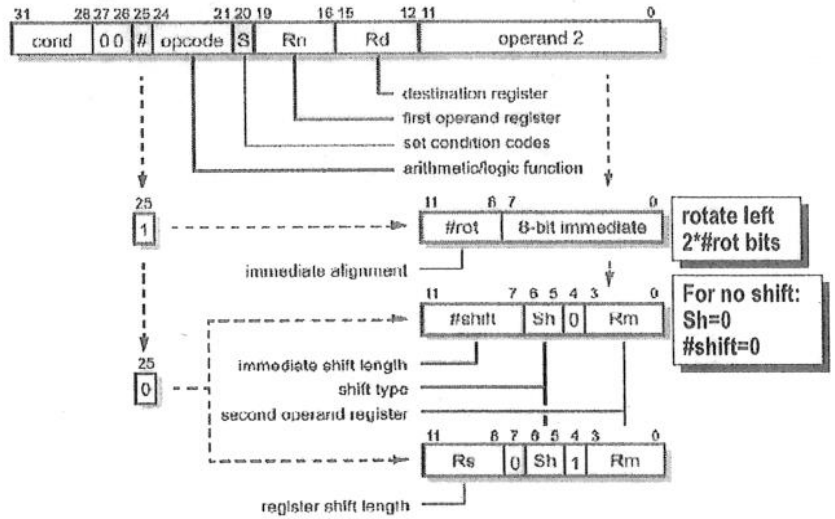
Exception Mode	Shadow registers
SVC,UND,IRQ,Abort	R13, R14, SPSR
FIQ	as above + R8-R12

(0x introduces a hex constant)

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

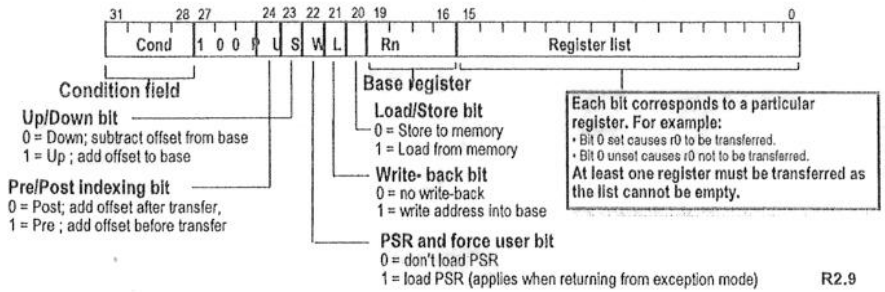
Data Processing Instruction Binary Encoding



R2.8

Multiple Register Transfer Binary Encoding

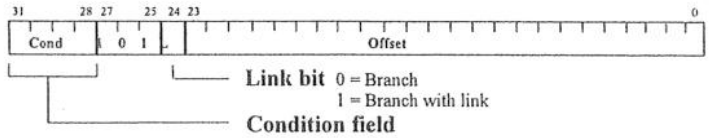
- The Load and Store Multiple instructions (LDM / STM) allow betweeen 1 and 16 registers to be transferred to or from memory.



R2.9

Branch Instruction Binary Encoding

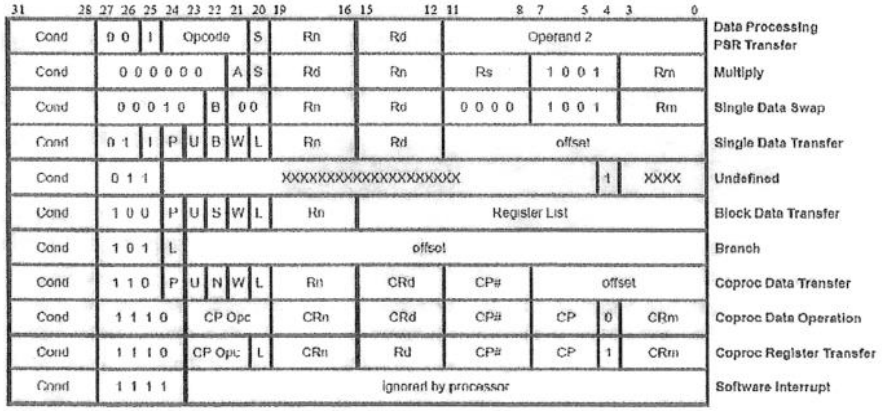
- Branch : `B{<cond>} label`
- Branch with Link : `BL{<cond>} sub_routine_label`



- 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

Instruction Set Overview



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)