Master Copy
June 05

Paper Number(s): **E2.19** (EZ.7 π)

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2005

EEE Part II: MEng, BEng and ACGI

INTRODUCTION TO COMPUTER ARCHITECTURE

Wednesday 8th June 2005 2:00pm

There are THREE questions on this paper.

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

Answer Question 1 and EITHER Question 2 (carrying 60%) or Question 3 (carrying 60%).

This exam is **open book**

Time allowed: 1:30 hours.

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.K.

© University of London 2005

Students may bring any written aids into this examination.

Question 1 is compulsory, and carries 40% of the total mark, this equals a time of 36 minutes.

Answer EITHER Question 2 OR Question 3, each of which carries 60% of the total mark, corresponding to a time of 54 minutes.

Question 1 is compulsory

1.

a) Convert into a decimal integer or rational number each of the following hexadecimal words, interpreted as specified.

(i) 0xc4e2, 16 bit unsigned
(ii) 0xfde2, 16 bit signed
(iii) 0x90600000, IEEE 754 floating point.

[6]

b) Write in assembler a length optimal ARM assembler fragment which implements the following pseudo-code, assuming the comparison to be signed:

```
if r0 < r9 + 5 then r1 := r2-r3 else r1 := r4+r5
```

[8]

c) Noting that 2159 = (128-1)*(16+1) , determine a sequence of ARM data processing instructions that will set `r1` equal to 2159*`r0` in two machine cycles leaving all other registers unchanged.

[6]

d) Determine the changes to registers and/or memory locations resulting from each instruction in the ARM interrupt service routine (ISR) shown in Figure 1.1. Explain the use of r13 and hence the overall function of the code. Write an optimised version of this code for use with the ARM Fast Interrupt (FIQ).

[10]

e) Trace through execution of code fragment COPYLOOP in Figure 1.2 when the initial value of `r2` is 9, giving the sequence of conditionally executing instructions and stating in each case whether the instruction condition is true. You may assume that, when the instruction condition is true, branch instructions take 2 machine cycles, multiple register load and store instructions with n registers in their masks take $n + 1$ machine cycles, and all other instructions (including any with instruction condition false) take 1 machine cycle. Determine the number of machine cycles taken by the COPYLOOP loop when it iterates n times ($n > 1$). Hence or otherwise determine the asymptotic limit for large n of the ratio of number of memory bytes written to the number of machine cycles used by this code fragment.

[10]

```
        STR   r0, [r13], #1
        LDR   r0, TIME
        ADD   r0, r0, #1
        STR   r0, TIME
        LDR   r0, [r13, #-1]!
        SUBS  pc, r14, #4
TIME    %     4
```

Figure 1.1

```
COPYLOOP  SUBS      r2, r2, #8
A         LDMPLIA   r0!, {r3-r10}
B         STMPLIA   r1!, {r3-r10}
C         BPL       COPYLOOP
```

Figure 1.2

2 The subroutine PARITY is shown in Figure 2.1.

a) Which registers are changed as the result of execution of PARITY? Indicate how, by the addition of appropriate instructions, the registers `r0 - r13` can be preserved across the subroutine call. Specify precisely where in the PARITY code the additional instructions must be inserted, and what these are.

[15]

b) The code between PA and PB calculates a result `r4` $= f($`r3`$)$. Describe the function f.

[15]

c) State precisely in what way PARITY modifies the contents of memory, using where necessary the function f from part b.

[15]

d) Write an alternative to the code between PA and PB using a lookup from a 256 byte constant table in memory to speed up execution. You need not provide code to initialise the table, or a definition of the table in assembler, but must specify precisely the table's contents.

[15]

```
PARITY    ADR     r0, BUFFER
          MOV     r1, #0
          MOV     r2, #0
          MOV     r7, #0
L1        LDRB    r3, [r0,r1]
          EOR     r7, r7, r3
PA
          MOV     r4,#0
          MOV     r5, r3, lsl #24
L2        MOVS    r5, r5, lsl #1
          ADDMI   r4, r4, #1
          BNE     L2
          AND     r4, r4, #1
PB
          AND     r3, r3, #&7f
          ORR     r3, r3, r4, lsl #7
          EOR     r7, r7, r3
          STRB    r3, [r0,r1]
          ADD     r1, r1, #1
          CMP     r1, #128
          BNE     L1
          STRB    r7, [r0,r1]
          MOV     pc, r14

BUFFER    %       129       ;129 byte memory buffer
```

Figure 2.1

3.

a) Write paragraphs of no more than 50 words which answer each of the following questions:

(i) What is an Instruction Set Architecture?

[3]

(i) How do instructions in the ARM Instruction Set Architecture support the implementation of stacks.

[4]

(ii) How does the mechanism of shadow registers in the ARM Instruction Set Architecture enable transparent handling of IRQ mode interrupts.

[3]

b) Draw a diagram illustrating how the bits of r0, r1 and the Carry status bit change after the execution of the ARM code fragment in Figure 3.1. Write in ARM assembler a subroutine REVERSE, using an ascending stack in which the register `r13` points to the highest memory word used by the stack. On exit register `r0` must be set equal to its value on subroutine entry but with bits reversed, so that bit *n* becomes bit 31- *n* and vice versa. All other registers are unchanged.

[20]

c) A CPU with 8 bit data bus uses a write-back direct access data cache to speed up its access to main memory. The cache contains only 1 line of length 4 bytes. The data memory usage of a program consists of 5 byte pushes, followed by 5 byte pops, on an ascending stack, where the first stack location written has address hexadecimal 0x100. You may assume that instructions are fetched from a separate instruction memory, and that this process does not affect the data cache.

(i) What is the total size, in bytes, of the data cache? Assuming that the CPU address bus is 20 bits in length, determine the (possibly empty) sets of bits in the CPU address corresponding to the cache tag, index, and select fields.

[8]

(ii) Determine the sequence of data read and write addresses issued by the CPU to the data cache during the program execution.

[4]

(iii) Assume initially that valid bits for all data cache lines are false. Trace through sequence of CPU data operations specified in part (ii) indicating the sequence of main memory reads and writes that are required, and the state of data cache valid and dirty bits after each CPU data operation.

[18]

```
MOV r0, r0, rrx #1
ADCS r1, r1, r1
```

Figure 3.1

Master Copy

SOLUTIONS - E2.19/E1.9 Sec A

June 2005

Question 1

a) *Determine the numbers represented by the hexadecimal bit-pattern 0x90003001 when interpreted as specified.*

(i) *0xc4e2, 16 bit unsigned*
(ii) *0xfde2, 16 bit signed*
(iii) *0x90600000, IEEE 754 floating point.*

(i) *50402*
(ii) *-542*
(iii) $-2^{32-127} * 1.75 = -4.42 * 10^{-29}$

b)

```
ADD r10, r9, #5
SUB r0, r10
SUBMI r0, r2,r3
ADDPL r0, r4, r5
```

c)

```
ADD r1, r0, r0, lsl 4
RSB r1, r1, r1, lsl 7
```

d) The first instruction stores `r0` on the stack – notice this is the IRQ stack since the interrupt will have swapped to a shadow r13. The next three instructons load `r0` from a word in memory, increment it, and save to memory. The penultimate instruction restores the old value of r0. The final instruction returns from the interrupt swapping back to user registers.

Note that the save & load are needed because r0 is not shadowed. In an FIQ there are extra shadow registers available. One of these (r8-r12) could be used instead of `r0` in which case the `r0` save/restore would not be needed. In fact the value of TIME could be kept permanently in the FIQ register so reducing the FIQ to just 2 instructions.

e)

Execution if r2=9. Condition true unless otherwise specified.

COPYLOOP
A
B
C
COPYLOOP
A (condition false)
B (condition false)
C (condition false)

Time taken is 1+9+9+2 cycles for all iterations other than the last, or 1+1+1+1 cycles for last iteration only. So t = 21*n-17. 8 words => 32 bytes are transferred each iteration, so asymptotically 32/21 bytes are transferred each cycle.

Question 2

a)

r0,r1,r2,r3,r4,r5,r7 changed.

STMED r13!,{r0,r1,r2,r3,r4,r5,r7} ; insert before first instruction

LDMED r13!,{r0,r1,r2,r3,r4,r5,r7} ; insert before MOV pc, r14,
; or add r15 to register list & replace the MOV pc

b)

f calculates the *xor* of the bottom 7 bits of `r3`, and sets the bottom 7 bits of `r4` = `r3`, and the top bit equal to this *xor* (the 7 bit parity).

c)

Bytes BUFFER – BUFFER+127 are modified as *mem*[x] ← *f*(*mem*[x]). Byte BUFFER+128 is set to the 8 bit bitwise xor of all the bytes from BUFFER to BUFFER+127.

d)

ADR r1, TABLE
LDR r4, [r1,r3]

Byte i of TABLE initialised to contain *f*(*i*)

Question 3

a) An ISA is a precise definition of the operation of a CPU at the level of machine instructions which defines the CPU registers, the function, but not timing, of each instruction and also the function (but not timing) of interrupts and exceptions.

The ARM ISA provides multiple register load & store instructions, which contain a register mask allowing any subset of the 16 ARM registers to be loaded or stored from successive locations in memory as determined by another of the registers. The 4 types of stack: ascending/descending and SP full or empty, each have corresponding versions of these instructions: LDM/STM suffix EA,ED, FA, FD SP points to empty/full location, stack grows Ascending, descending. Furthermore, each LDM/STM instruction can either update the register used as stack pointer, or leave the SP unchanged as when accessing data on the stack.

IRQ mode shadows r14 & r13 & also has a register which saves the user CPSR. The IRQ mode r13 points to a separate IRQ stack allowing interrupt code to execute safely independent of user code. Other registers can be saved & restored using this stack as necessary, and the user instruction stream restored by restoring the CPSR from the IRQ SPSR, and the user pc from the IRQ r14.

b)

```
REVERSE     STMFA    r13!, {r1,r2}
            MOV      r2, #32
REV1        MOV      r0, r0, lsr #1
            MOV      r1, r1, lsl #1
            SUBS     r2, r2, #1
            BPL      REV1
            LDMFA    r13!, {r1,r2,r15}
```

c)
(i) size 4 bytes, select A1:0, tag A19:2, index has no bits.

(ii) (all in hex) W100, W101, W102, W103, W104, R104, R103, R102, R101, R100.
(R=Read, W=Write)

(iii) There is only one cache line, it is always (after the first op) valid.

Data op	Valid	Dirty	Memory operation
W100	1	1	R100-103
W101	1	1	
W102	1	1	
W103	1	1	
W104	1	1	W:100-103 R:104-107
R104	1	1	
R103	1	0	W104-107 R:100-103
R102	1	0	
R101	1	0	
R100	1	0	