

**UNIVERSITY OF VICTORIA**  
**EXAMINATIONS SUMMER 2008**

***C SC 230 - Introduction to Computer Architecture and  
Assembly Language***

**STUDENT NUMBER:** \_\_\_\_\_

**TIME:** 3 hours

**INSTRUCTOR:** M. Serra

**TOTAL MARKS:** 68

**TO BE ANSWERED ON THE PAPER**

Question No.	Value	Mark	Question No.	Value	Mark
1	4		8	6	
2	4		9	6	
3	5		10	5	
4	4		11	5	
5	6		12	4	
6	7		13	6	
7	6		<b>TOTAL</b>	<b>68</b>	

**INSTRUCTIONS:**

1. STUDENTS MUST COUNT THE NUMBER OF PAGES IN THIS EXAMINATION PAPER BEFORE BEGINNING TO WRITE, AND REPORT ANY DISCREPANCY IMMEDIATELY TO THE INVIGILATOR
2. This examination paper consists of 12 pages including this cover page plus 2 pages of Appendix.
3. No aids are permitted. However, an Appendix describing the ARM instruction set is provided for your use.
4. The marks assigned to each question are shown within square brackets. Partial marks are available for all questions.
5. Please be precise but brief, and use point form where appropriate.
6. It is strongly recommended that you read the entire exam through from beginning to end before beginning to answer the questions.

**Question 1. [4]** The data portion of an ARM program contains:

```
NumList: .skip    100
Holder:  .skip     4
K:       .skip     4
```

All the variables declared above already contain some values. An integer is a word = 4 bytes. R1 contains the current value of variable K; R2 contains the address of NumList, which is an array of integers; R3 contains the current value of variable Holder. State the *single* ARM instruction equivalent to the *single* C language instruction (do not write a loop!):

```
NumList[K] = Holder;
```

**Question 2. [4]** Fill in the most appropriate ARM assembly language instructions to accomplish the tasks stated. Some tasks may require more than one instruction, but only 1 register at most.

**R1 = AL1** @ AL1 is a variable in the "data" section

**R2 = address of AL2** @ AL2 is a variable in the "data" section

**R3 = R1 - R2** @ The condition codes in the CPSR must also be set

**R1 = 5116**

**Question 3. [5]** The Motorola 68000 microprocessor has byte addressable memory, with a 24-bit address bus and a 16-bit data bus, where one word contains 2 bytes. (Reminder:  $1K=1,024=2^{10}$  and  $1M=1,024K=2^{20}$ )

- (a) [1] What is the total number of addressable locations? \_\_\_\_\_
- (b) [1] How big is each of location? **8 bits** **1 byte** **2 bytes** **1 word** (circle ALL that apply)
- (c) [1] Your manager wants to build a system based on the 68000. It will have 32 peripherals each of which is byte addressable and requires an address space of 1Mb. A word addressable memory unit of 16Mb has also been bought and the manager asks you to check that it is all feasible. Is it?

YES NO (circle one)



- d [2] Give a complete explanation to back your answer above, by explaining your logical reasoning, as you might do in a meeting to your team at work.

**Question 4.** [4] A computer system has a RAM containing 64K bytes, each of which needs its own distinct address. Moreover it has 4 peripherals and they each require  $2^4$  distinct addresses in order to interface properly.

- (a) [1] How many distinct addresses in total are necessary in this system? Write the number as a simple expression or a decimal number in the centre of the line below.

$$2^{\boxed{\phantom{00}}} < \underline{\hspace{2cm}} < 2^{\boxed{\phantom{00}}}$$

total number of addresses here

- (b) [2] Place the number of addresses just computed between the appropriate powers of two in the expression above, by writing the correct exponent in the boxes (for example, if the result in the centre had been 18, you would have  $2^4 < 18 < 2^5$  by writing the exponents 4 and 5 in the boxes).
- (c) [1] How many lines does an address bus for this system require, given that it must be able to carry all the needed values for the addresses? \_\_\_\_\_

**Question 5.** [6] Program execution time,  $T$ , is defined in section 1.6.2 of the textbook as:

$$T = \frac{N \times S}{R} \quad \text{where}$$

- $T$  is the total elapsed time from start to finish,
- $N$  is the number of machine language instructions executed (not necessarily the number of machine instructions in the object code),
- $S$  is the number of basic steps needed to execute one machine instruction (where each basic step is assumed to take 1 clock cycle),
- $R$  is the clock rate.

You are asked to examine execution time for a certain high-level language program. The program can be run on a RISC or a CISC computer. The parameters for the former are  $T_r$ ,  $N_r$ ,  $S_r$  and  $R_r$ ; the parameters for the latter are  $T_c$ ,  $N_c$ ,  $S_c$  and  $R_c$ . Both computers use pipelined instruction execution and have the same clock rate so that  $R_r = R_c$ . Pipelining in the RISC machine is more effective than in the CISC machine, such that  $S_r = 1.2$  while  $S_c = 1.2$ . On the other hand, it usually takes more instructions to accomplish a task on RISC computer than on a CISC computers, so that  $N_r > N_c$  will normally hold.

**a[1]** What formula can you write for elapsed time on the RISC architecture?

$T_r$

**b[1]** What formula can you write for elapsed time on the CISC architecture?

$T_c$

**c[4]** What ratio  $N_c/N_r$  will give the same elapsed time for both architectures? (Show your work)

**Question 6.** [7] It is always good to learn something new, even in a test situation, to show that one has really grasped the material by being able to expand on it. On the topic of parameter passing to a subroutine, you have learned to have them placed in registers only. Another technique is to place them on the system stack. This requires careful management in order to avoid mixing the parameters with the storage of locally used registers and with the allocation of local variables. The protocol is described in the following and you are to show your understanding of it by practising on an example.

- The caller routine places the parameters on the stack using either STR or STMFD instructions. In this example a mixture is used.

These three ARM instructions are used to call the function Foo with three input parameters, whose current values are in R2, R3 and R4, passed on the stack.

```
CallFoo:
    STR    R2, [SP, #-4]!
    STMFD  SP!, {R3-R4}
    BL     Foo
Next: . . .
```

- (a) [1]** Show the contents of the stack and any relevant pointers after the execution of these instructions and immediately after the execution of "BL Foo". Fill in Figure 1 on the right.

**Figure 1.** The stack after the instruction "BL Foo"

- At the entry point in the routine called here "Foo", the usual STMFD instruction to save registers used locally is issued.
- This is followed by the setting a value for the "Frame Pointer", labelled "FP", which is mapped to register R12 in ARM. The Frame Pointer provides a local address within the stack for the frame (also called scope) of the routine currently being executed. This is needed, especially



during recursion, since the global stack pointer may be moved by subsequent calls, while FP can be saved and restored locally. In fact FP is treated in the same way as the Link Register, 'LR', and a copy of it is saved together on the stack together with LR and the other registers. For this example "Foo", the code at the entry point is shown below, given that R5, R6 and R7 are used locally and that space for 2 local variables needs to be allocated on the stack.

The following three instructions are found at the entry of function Foo.

```

Foo: STMFD    SP!, {R5-R7, FP, LR}
      ADD     FP, SP, #12 @ set FP
      ADD     SP, SP, #-8 @ space for 2
                           @ local variables

```

b) [3] Show the updated contents of the stack, the position of SP and FP using Figure 2.

**Figure 2.** The stack after the 3 initial instructions in Foo.

- When exiting a routine, the stack must be restored and similarly the various pointers SP, FP and LR. First of all the space allocated for local variables must be released and then the stack cleared. This is done with two instructions shown below for this example Foo.

At the exit from function Foo the following two instructions are executed.:

```

exitFoo: ADD    SP, SP, #8 @ free space
              @for 2 locals
          LDMFD  SP!, {R5-R7, FP, PC}

```

Control next returns to the calling routine, at "Next" above.

(c) [2] Show the updated contents of the stack and of the SP pointer only after execution of each of these two instructions using Figures 3(i) and 3(ii) respectively.

**Figure 3. (i)** Stack after ADD . . .

**Figure 3. (ii)** Stack after LDMFD . . .

- (d) [1] Finally, back to the calling routine at label Next. It was the calling routine which pushed the parameters on the stack to start with and thus it is its responsibility to clear the stack. Give the one instruction needed at this point to restore the stack to the condition it was before any preparation for the call to Foo was done, that is, before execution at label "CallFoo".

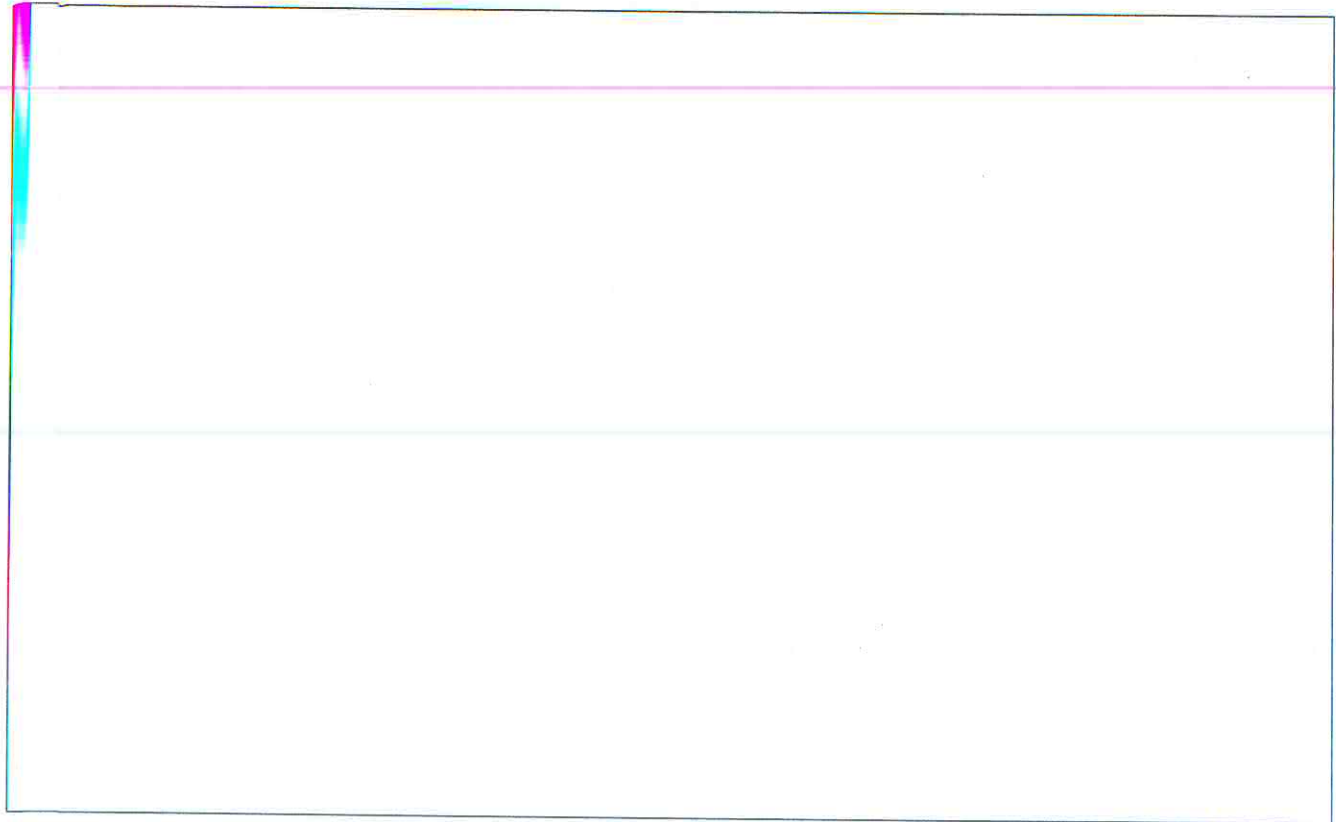
**Question 7. [6]** Pipelines are an extremely powerful architectural feature to improve performance. At a co-op interview you are asked to describe concisely and precisely two items, such as you would give to a second year student who has not taken CSC 230 yet and is curious about what you have learned. Assuming a basic knowledge of what a pipeline is in a processor architecture, you are aware that there are some possible impediments to the smooth and super efficient operations of a pipeline. State what such issues might be and then select one of them and give a more detailed explanation, again keeping in mind who your audience is as above.

**a[3]** What are some possible impediments?

**b[3]** Describe one of these impediments:

**Question 8. [6]** In the same interview situation, one of the interviewers is actually a current co-op student at the company, albeit a junior one who has only completed the first year in the CSC program and has only used Java as a programming language. You are asked to explain, briefly and concisely, to this person two items. Be brief and clear – being hired for the job of your dreams depends on it!

**(a) [4]** What does a compiler do? Start from a text file containing source code for a C program and follow through the steps until one obtains a fully executable module in memory with the PC having been assigned the address of the first instruction (thus everything is ready to execute). Feel free to use an annotated diagram for clarity.



- (b) [2] What is the difference between the process just described for a C source file and what would happen for a Java source file? Again feel free to use an annotated diagram for clarity.



**Question 9.** [6] Describe the steps which occur when the ARM instruction:

ADD R1, R2, R3

is fetched, decoded and executed in the context of the example structure of the internal datapath of a processor of Figure 4. Give precise details of everything that happens, which registers, buses, control lines, etc. are used, how each element is addressed and describe the purpose of each micro-operation. Be brief yet precise

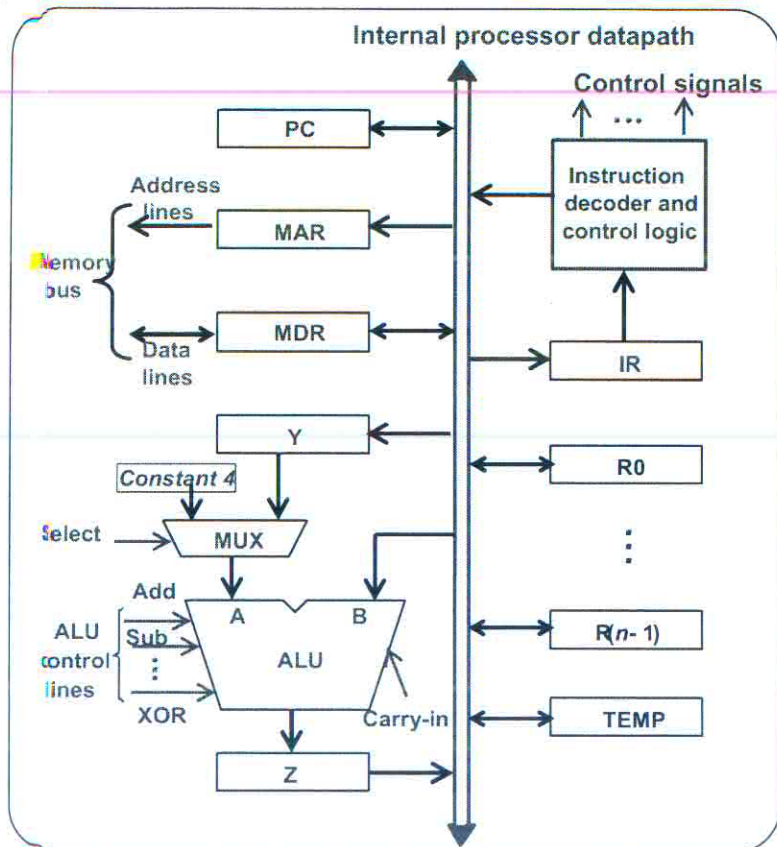


Figure 4. An example of datapath

## [2] FETCH CYCLE

## [1] DECODE CYCLE

## [3] EXECUTE CYCLE



**Question 10.** [5] Assume you have a system which contains both Virtual Memory and a Cache. You must explain what happens when the processor needs to read data and produces a virtual address is to be translated into a physical address. Answer the following questions:

(a) [1] Where is the Page Table stored?

(b) [1] Where is the TLB stored?

(c) [3] In Table 2, there are various sequences of actions involving various components. If a component is named, it means that the component performs an action or is accessed or is updated. Some sequences are redundant or even implausible. However three of them are correct and describe the three real possible cases labelled [1], [2] and [3] in Table 1. Match these three scenarios to three correct sequences from the list A to J and place the letter label in the right column of Table 1.

**Table 1: Three scenarios**

		Sequence match letter
[1]	Best scenario: there is a cache hit	
[2]	Worst scenario: data is on disk	
[3]	Data is in memory, but not in cache	

**Table 2: Possible Sequences of Components/Actions Accessed/Updated**

Match 3 of these sequences to the 3 scenarios in Table 1	
MMU → TLB → Cache	A
MMU → TLB → Page Table → Page Fault → Disk → Memory → Cache → Page Table → TLB	B
MMU → TLB → Page Table → Memory → Cache → Page Table → TLB	C
TLB → Cache	D
MMU → Cache	E
MMU → TLB → Page Table → Cache	F
TLB → Page Table → Cache	G
TLB → Page Table → Memory → Disk → Page Fault → Cache	H
MMU → TLB → Page Table → Memory → Disk → Page Fault → Cache → Page Table → TLB	I
TLB → Page Table → Memory → Page Table	J

**Question 11.** [5] Suppose that an address bus error occurs during the execution of an application. Describe what typically happens on a computer system (such as Windows or Linux) as a result of the address bus interrupt signal being generated. Be precise, discuss the levels of interrupts and what is the final outcome for the system. This means that you must state whether the application resumes or crashes, or whether the system goes back to the OS ready for another application or simply continues with other currently multitasking processes for that application program, or whether everything freezes (e.g. the 'blue screen of death').

**Question 12. [4]**

(a) [1] What does “multi-level” cache refer to?

(b) [1] What does “split” cache refer to?

(c) [1] State one possible advantage of a split cache structure.

d[1] A system has a 4-way set associative cache, while a second system has an 8-way set associative cache. What is the difference?

Question 13. [6] Examine carefully the mystery ARM code below, step through it manually, state what it does and show the contents of register R0 at the end of execution.

a) [4] What is in R0 at the end? R0 = \_\_\_\_\_

(b)[2] Explain what the code is doing for this example and in general.



```

        NUL = 0
_start:
        ldr        r1,=ST1
        ldr        r2,=ST2
        bl         SS
        swi        0x11      @ what is in R0 here? Write above

SS:     STMFD      sp!,{lr}   @function SS
        mov        r4,#0
        mov        r0,#-1
Mloop:
        ldrb       r3,[r1]
        cmp        r3,#NUL
        beq        DS
        mov        R5,r1
        mov        R6,r2
        bl         SW
        cmp        r0,#0
        beq        DY
        add        r1,r1,#1
        add        r4,r4,#1
        bal        Mloop
DY:     mov        r0,r4
DS:     LDMFD      sp!,{pc}   @ end of SS

SW:     STMFD      sp!,{lr}   @function SW
        mov        r0,#0
Tloop:
        ldrb       R7,[R5],#1
        ldrb       R8,[R6],#1
        cmp        R8,#NUL
        beq        DSW
        cmp        R7,R8
        beq        Tloop
NSW:    mov        r0,#-1
DSW:    LDMFD      sp!,{pc}   @ end of SW
@ *****

        .data
ST1:    .asciz     "avvjda"
ST2:    .asciz     "vj"
        .end

```

\*\*\*\*\* THE END \*\*\*\*\*