UNIVERSITY OF VICTORIA

EXAMINATIONS SUMMER 2010

C SC 230 - Introduction to Computer Architecture and Assembly Language - CRN# 30198

STUDENT NUMBER: _____

TIME: 3 hours

INSTRUCTOR: M. Serra

TOTAL MARKS: 117

TO BE ANSWERED ON THE PAPER

Question No.	Value	Mark	Question No.	Value	Mark
1	11		10	3	
2	12		11	2	
3	4		12	6	
4	4		13	20	
5	8		14	6	
6	10		15	6	
7	11		16 (bonus)	4	
8	3		**TOTAL**	**115**	
9	9				

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 15 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. [11] In the "Fetch/Decode/Execute" phases of an instruction, the "Fetch" phase is the same for every instruction. (The Fetch phase includes the operation of incrementing the PC by 4; assume all instructions are 4 bytes in size.)

The sequence of steps for the Fetch phase is shown in Table 1 and it follows the type of CPU organization seen in the lecture notes. Imitate the style and level of detail when answering parts the question below. Please note that the instructions you are analyzing below are not necessarily part of the ARM ISA.

TABLE 1. Steps Performed during Fetch Phase of an Instruction

Step	Action
1.	PC → MAR
2.	MAR → address bus
3.	Read signal → control lines
4.	PC → ALU input
5.	#4 → ALU input

Step	Action
6.	Add signal → ALU control lines
7.	ALU output → PC
8.	Wait for memory to finish reading
9.	Data bus → MDR
10.	MDR → IR

(a) [] Give the detailed steps performed during the phases which follow Fetch for the instruction:

```
LDOI R1, [R2,(R3)]
```

where `LDOI` is a Load-Offset-Indirect instruction. In this example, register `R2` holds an address, while register R3 holds the address of a word of memory which in turn contains a value to be used as offset. That is, the content of the word whose address is in R3 must be added to the address in R2 to compute the effective address of the word in memory to be loaded into register R1. The semantics can be expressed as:

```
R1 = Memory[R2 +Memory[R3]]
```

(b)[3] Give the detailed steps performed during the phases which follow Fetch for the instruction:

```
BLX R1
```

where `BLX` is the Branch-with-Link-and-Exchange instruction. It causes the address of the instruction following the `BLX` to be copied into register `R14` and an immediate transfer of control to the address held in register `R1`.

(c) [2] `BLX` is an ARM instruction, however `LDOI` is not. What ARM instruction(s) would you have to write to achieve the same effect as `LDOI R1, [R2, (R3)]`? (You only need 2 instructions).

Question 2. [12] Fill in the table below with the appropriate information about the instructions.

Instruction	Addressing Modes (all operands)	What it does (use appropriate notation)
LDR R0,=X		
TST R1,#0		
STR R1,[SP,R2,LSL #2]!		

Question 3. [4] Consider the sequence of ARM instructions listed below. Where in this sequence might pipelining hazard(s) occur? Give the line number of the instruction, and state which kind of hazard it could be. Assume that the branch in line 4 is taken.

```
1]              ...
2]              MUL       r1,r2,r3
3]              SUBS      r1,r1,#1
4]              BEQ       L1              @ taken!!!
5]              STR       r1,[r4]
6]  L1:         LDR       r5,[r5]
7]              ...
```

Question 4. [4] Figure 1 shows the possible organization of a 4-stage pipeline.

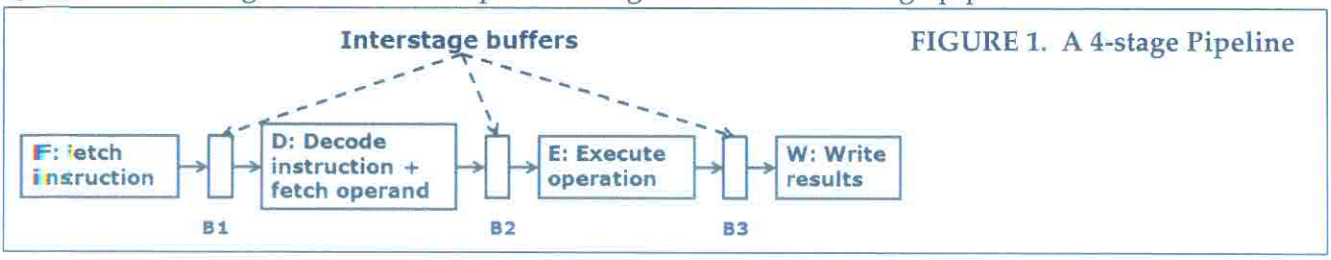


FIGURE 1. A 4-stage Pipeline

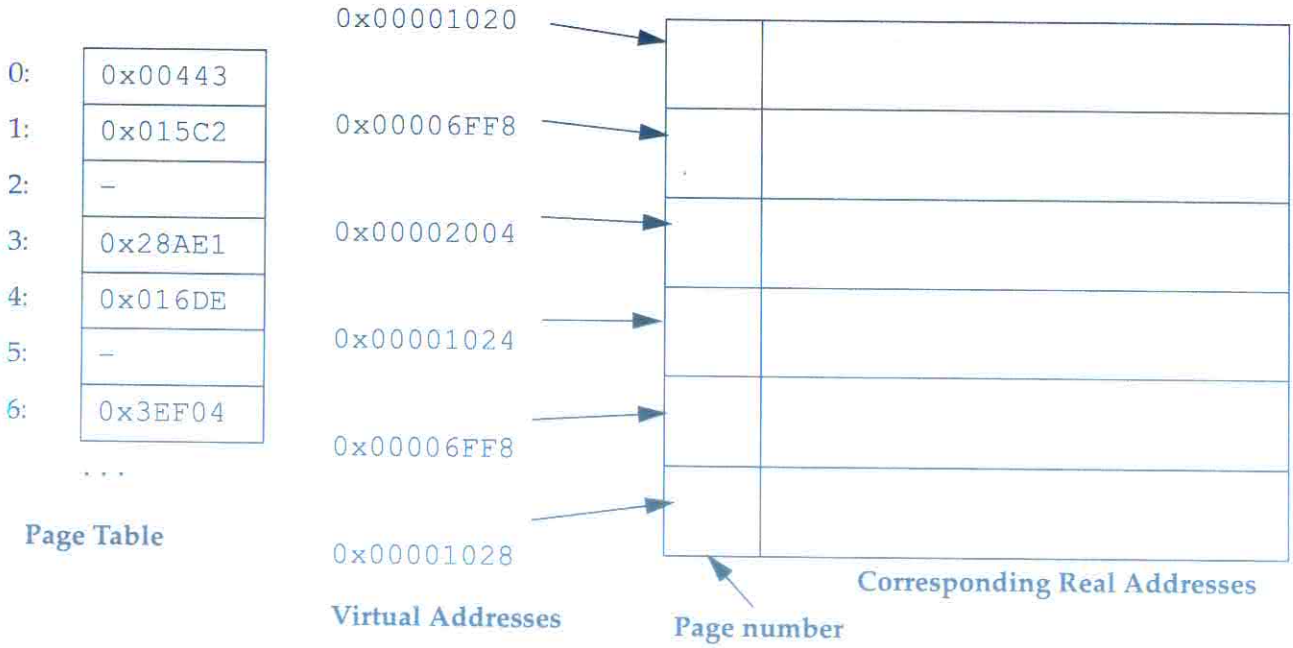
Define the speedup which could possibly be obtained, in the best case, by using a pipeline with m stages executing N instructions. Show a good explanation of your answer (do not just state a formula).

Question 5. [8] Consider a system which includes virtual memory. It has a 32-bit address space with everything being byte addressable; this implies a virtual address space which is 2^{32} bytes or 4 GB in size ($1G = 2^{30}$). The system, using VM, is organized into pages, each of which is 4 KB in size ($1K = 2^{10}$). The page table, obviously, must have an entry for each of these pages. The main memory (RAM memory) is 1 GB in size.

(a) **[1]** How many entries does the page table have?

(b) **[1]** How many bits are necessary to represent the page table numbers (given the number of pages stated above)? Also state the number of hexadecimal digits required.

(c) **[6]** Suppose that the CPU generates the sequence of memory addresses shown below, written in hexadecimal in the centre column. Also suppose that the page table at this point contains the entries shown on the left. Invalid (unmapped) entries in the page table are shown as empty (e.g. the entries at 2 and 5). Show the corresponding sequence of addresses in main memory which the virtual addresses are mapped to. If any address cannot be mapped, show your answer as the words 'page fault'. State explicitly the page number first in the leftmost column.



Question 6. [10] You are involved in the design of a system which needs to have a 16-bit address bus and a 16-bit data bus. The system is expected to be byte-addressable for everything and a word is defined to be 16 bits (2 bytes). There are both peripherals and memory units to be connected within it and it is expected that the whole address space will be used. The diagram below may help you visualize the required answers to the questions. State the answers in the appropriate boxes. Leave the values in the answers as powers of 2 or simple approximations to them or simple factors - there is no need to give the actual number. Hint: follow the order given for the answers, it will help.

(a) [1] What is the total number of addressable locations for this system?

(a)

(b) [1] 1/4 of the addressable space is to be used for the peripherals requirements. What is the total number of available addresses for peripherals?

(b)

(c) [1] 3/4 of the addressable space is to be used for the memory requirements of RAM and ROM. What is the total number of addresses available for memory?

(c)

The addressable space for peripherals is further subdivided as shown below.

(d) [1] The ROM has already been bought and its size is 16KB. How many addresses does it need?

(d)

(e) [1] How many addresses are left for RAM?

(e)

(f) [1] The cheapest RAM chips contain 8K words. How many should you buy to fit into this system?

(f)

(g) [1] 1/8 of the addresses are to be used for input components. How many?

(g)

(h) [1] 1/8 of the addresses are to be used for output components. How many?

(h)

(i) [1] 6/8 of the addresses are to be used for disk addressing. How many?

(i)

(j) [1] The chosen disk drives each require 4KB of address space. How many can one fit into the system?

(j)

Question 7. [11+5] Consider a small direct-mapped cache which can contain eight 16-bit words, as shown in Table 2. The cache blocks numbers are shown on the left, numbered 0 to 7. Each word has an associated tag. When a miss occurs during a read operation, the requested word is read from the main memory and sent to the processor. At the same time, it is copied into the cache, and its block number is stored in the associated tag. The computation of the tag is ignored in this exercise.

Table 2: A small Direct Mapped Cache with 8 blocks

	Tag	16-bit content
line 0		
line 1		
line 2		
line 3		
line 4		
line 5		
line 6		
line 7		

Consider now the code segment below containing a loop, where all instructions and operands are 16 bits long (2 bytes) (similar to ARM but the instructions are shorter). The code segment starts at address `02E0` and the addresses of each instruction are shown on the right hand side.

Instruction	Address in Memory	Mapped to cache block number
`LOOP:LDR R2,[R1],#2`	`02E0`	
`ADD R0,R0,R2`	`02E2`	
`SUBS R3,R3,#1`	`02E4`	
`BNE LOOP`	`02F6`	

Assume that, before this loop is entered, registers R0, R1 and R3 contain: `R0 = 0x0000, R1 = 0xC544, R3 = 0x0003`. Also assume that main memory contains the following data, shown with its address, only for 3 locations.

DATA in memory	Address in Memory	Mapped to cache block number
`A03C`	`C544`	
`05D9`	`C546`	
`10D7`	`C548`	

When an instruction or a piece of data is loaded into the cache, the address of the cache block is given by the rightmost 3 bits of the memory address, since the mapping function is: (address MOD 8). *(Please note that this is very similar to an example you should have studied from the lectures.)*

(a) [2] Fill in the right hand columns above with the cache block number where the instructions and the data will be placed when loaded.

(b) [9] Simulate the execution of the code, where the loop iterates 3 times. For each iteration, state the content of the cache showing any collision. Moreover it is given that the access time of the main memory is *10 t* and that of cache is *1 t*. At each iteration, examine the execution time, ignoring the time taken by the processor between memory cycles, and adding the number of memory and cache accesses. Thus you must show, for each iteration:

- the content of the cache;
- the number of memory accesses and the number of cache accesses;
- the total execution time (for theses accesses only).

To help you in this, 3 copies of the cache are shown below (use one for each iteration), with the requested answers to be written below.

	Tag	Content
line 0		
line 1		
line 2		
line 3		
line 4		
line 5		
line 6		
line 7		

Iteration 1

memory accesses:
cache accesses:
total time:

	Tag	Content
line 0		
line 1		
line 2		
line 3		
line 4		
line 5		
line 6		
line 7		

Iteration 2

memory accesses:
cache accesses:
total time:

	Tag	Content
line 0		
line 1		
line 2		
line 3		
line 4		
line 5		
line 6		
line 7		

Iteration 3

memory accesses:
cache accesses:
total time:

(c) [1] **BONUS:** Repeat the same work assuming however that the cache is used *only* to store instructions. Data operands are fetched directly from the main memory and not copied into the cache.

	Tag	Content
line 0		
line 1		
line 2		
line 3		
line 4		
line 5		
line 6		
line 7		

Iteration 1

memory accesses:
cache accesses:
total time:

	Tag	Content
line 0		
line 1		
line 2		
line 3		
line 4		
line 5		
line 6		
line 7		

Iteration 2

memory accesses:
cache accesses:
total time:

	Tag	Content
line 0		
line 1		
line 2		
line 3		
line 4		
line 5		
line 6		
line 7		

Iteration 3

memory accesses:
cache accesses:
total time:

Question 8. [3] Assume that dynamic random access memory (DRAM) must be refreshed every 64ms. Suppose that we have a DRAM chip which has 8192 rows of 8 bytes each, and that each row can be refreshed in 4 clock cycles. The clock speed is 1GHz (=1,000 MHz, in cycles per seconds).

(a) [1] How many clock cycles are needed to refresh the entire DRAM chip?

(b) [1] Given the number of clock cycles from above, how much time, in ms, are they equivalent to? (Feel free to leave the answer as an expression if you cannot face the arithmetic).

(c) [1] Let your answer from above be N ms. What fraction of time is spent refreshing the DRAM chip? (Give you answer as a simple expression using N – you do not need to evaluate it.)

Question 9. [9] Some first year students ask you questions and you need to answer them clearly and precisely, with the main goal being of complete understanding on their part, without writing a long essay. Show what a great *Teaching Assistant* you could be!

(a) [3] Can you state at least 3 main characteristics which differentiate between static and dynamic RAM? Explain how they apply to each.

(b) [2] Where is static RAM used mostly and where is dynamic RAM used mostly?

(c) [1] What is volatile and non-volatile memory?

(d) [1] What is an example (or application) of non-volatile memory?

(e) [1] Access to data on disk is much slower than to data in main memory. What are the factors in computing disk access time? Just state, do not need to describe fully.

Question 10. [3] Suppose that execution time for a program is directly proportional only to instruction access time. Access time to an instruction is 2 *ns* from the cache and 20 *ns* from memory. The probability of a cache hit is 96%. In the case of a cache miss, the instruction is fetched from main memory and copied into the cache, and then a second access must take place to copy the instruction from the cache (this time it will be a hit).

(a) [1] Compute the execution time of a program with 100 instructions without the cache (an expression is fine).

(b) [1] Compute the execution time of a program with 100 instructions with the cache (an expression is fine).

(c) [1] If the cache size is doubled, the probability of not finding an instruction is cut in half. Compute the execution time of a program with 100 instructions with the larger cache (an expression is fine).

Question 11. [2] What are the main differences between a subroutine and an interrupt-service routine?

Question 12. **[6]** *Amdhal's law* was discussed, stated as:.

$$\text{Speedup} = \frac{1}{f + \frac{1-f}{p}}$$

(a)2] Define f and p.

(b)4] What is the meaning of this law, or what is its impact? What does it say about increasing the number of processing elements and the effect on performance?

Question 13. **[20]** Write an ARM function to print the diagonal and the trace of square matrices. Given the address of a square matrix of integers and its size (size = number of rows or columns), the program implements the following actions using subroutines with parameters in registers:

- print the matrix in row major format (that is, row by row) ==> *this code is given to you;*
- print the diagonal of the matrix (from top-left to bottom-right);
- print the trace of the matrix (that is, the sum of the elements of the diagonal).

The matrix is assumed to be stored in row major format (i.e. row by row). In order to print the elements, you are given a subroutine "`PrintMat(R0:&Mat;R1:SizeMat)`" which uses the now well known SWI instructions in ARMSim#. Here are some examples:.

Matrix = $\begin{bmatrix} 0 & 1 & 2 \\ 10 & 11 & 12 \\ 20 & 21 & 22 \end{bmatrix}$,

stored in memory as the array MAT = [0, 1, 2, 10, 11, 12, 20, 21, 22], the output should look similar to:

```
Matrix (row order):
0    1    2
10   11   12
20   21   22
Diagonal is: 0 11 22
Trace is: 33
```

Matrix = $\begin{bmatrix} 42 \end{bmatrix}$,

stored in memory as the array MAT = [42], the output should look similar to:

```
Matrix (row order):
42
Diagonal is: 42
Trace is: 42
```

Matrix = $\begin{bmatrix} 0 & -1 & -2 & -3 \\ 1 & 2 & 3 & 4 \\ 2 & -3 & -4 & -1 \\ 3 & 4 & 1 & 2 \end{bmatrix}$,

stored in memory as the array MAT = [0,-1,-2,-3,1,2,3,4,2,-3,-4,-1,3,4,1,2], the output should look similar to:

```
Matrix (row order):
0    -1   -2   -3
1    2    3    4
2    -3   -4   -1
3    4    1    2
Diagonal is: 0  2  -4  2
Trace is: 0
```

```
        .data
MAT:    .skip   100
MatMsg: .asciz  "Matrix (row order):\n"
TraceMsg: .asciz "Trace is:   "
DiagMsg: .asciz  "Diagonal is:   "
Nline:  .asciz  "\n"
BL:     .asciz  "  "
```

```
        .equ SWI_PrInt,     0x6b
        .equ SWI_PrStr,     0x69
        .equ SWI_Exit,      0x11
        .text
_start:
        LDR  R0,=MAT    @Read matrix and size
        BL   READMAT    @R0:Size = READMAT(R0:&MAT):assume it is given
        MOV  R1,R0      @R1 = size of MAT
        LDR  R0,=MAT
        BL   PRINTMAT   @PRINTMAT(R0:&MAT,R1:size) ==> code given here
        BL   DIAG       R0:trace = DIAG(R0:&MAT,R1:size)==> write this
        MOV  R2,R0
        MOV  R0,#1      @to Stdout
        LDR  R1,=TraceMsg
        SWI  SWI_PrStr
        MOV  R1,R2      @print trace
        SWI  SWI_PrintInt
        SWI  SWI_Exit
PRINTMAT:       @(R0:&MAT,R1:size) - prints a 2D matrix
        STMFD   SP!,{R0,R1,R6-R9,LR}
        MOV     r7,r1           @r7 = size
        MOV     r9,r1           @r9 = size
        MOV     r8,r0           @r8 = &matrix
        ldr     r1,=MatMsg
        mov     r0,#1           @print to Stdout
        swi     SWI_PrStr
RowLoop:
        MOV     r6,r9           @r6 = col counter
ColLoop:
        ldr     r1,[r8],#4      @get matrix element
        swi     SWI_PrInt       @and print it
        ldr     r1,=BL
        swi     SWI_PrStr       @print blanks
        SUBS    r6,r6,#1        @decrease column counter
        BNE     ColLoop
        ldr     r1,=Nline
        swi     SWI_PrStr       @newline
        SUBS    r7,r7,#1        @decrease row counter
        BNE     RowLoop
        ldr     r1,=Nline
        swi     SWI_PrStr       @newline
        LDMFD   SP!,{R0,R1,R6-R9,PC}
```

```
IAG:    @R0:trace = DIAG(R0:&MAT,R1:size)
        @prints the diagonal of a matrix and returns trace in R0
```

Question 14. [6] You have seen the diagram in Figure 2 before, showing some possible interconnections between peripheral devices and a processor, through the interfaces.

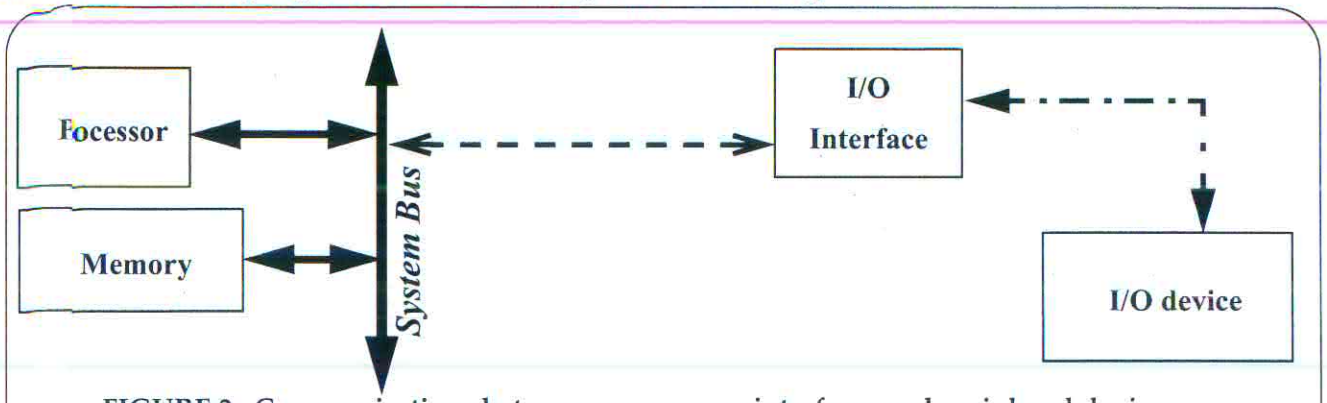


FIGURE 2. Communications between a processor, interfaces and peripheral devices.

(a) [1] Name/state at least two possible protocols for the communication between the processor and the interfaces (no descriptions).

(b) [2] Name/state at least two possible protocols for the communication between the interfaces and the devices (no descriptions).

(c) [1] State at least one reason why interfaces are needed.

(d) [1] Is an interface usually a software or a hardware component?

Question 15. [6] Consider a system that has it all: Cache, MMU (Memory Management Unit), TLB (Translation Lookaside Buffer), Virtual Memory, Page Table, DMA (Direct Memory Access), Pipelining and so on. Describe what happens at a CPU request for a piece of data in two of the following scenarios, from best to worst. In each, state the steps in order where the various components are involved and how. SELECT ONLY TWO SCENARIOS.

(a) The data requested in is cache.

(b) The data requested is in memory, not in cache, and it has been recently loaded into it.

(c) The data requested is in memory, not in cache, and it was loaded into it a while ago.

- The data requested is on a disk.

Question 16. [4] **BONUS** Explain what happens when an application uses dynamic linking.

THE END