UNIVERSITY OF VICTORIA

EXAMINATIONS SUMMER 2013

C SC 230 - Introduction to Computer Architecture and Assembly Language - A01 CRN# 30141

STUDENT NUMBER: ______________________________

TIME: 3 hours

INSTRUCTOR: M. Serra

TOTAL MARKS: 100

TO BE ANSWERED ON THE PAPER

Question No.	Value	Mark	Question No.	Value	Mark
1	8		10	5	
2	5		11	4	
3	5		12	4	
4	2		13	6	
5	8		14	7	
6	12		15	2	
7	6		16	8	
8	9		17	4	
9	5		18		
			TOTAL	**100**	

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 16 pages including this cover page.
3. No aids are permitted. However, a handout 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. [8] Consider a computer system whose main memory (RAM) is byte addressable and 32KB in size. This computer's address bus is 16 bits wide, and uses virtual memory with 4KB pages.

(a) [2] If the processor generates a virtual address of `0x3A72`, what are the page number and the offset components of that address? (Give both answers in hexadecimal.)

Page #: ____________ **Offset:** ____________

(b) [5] Suppose that a program has been running for a while. A snapshot at an instant in time shows the following pages in main memory (as shown on the right-hand side). Complete the entries in the page table (the table on the left) which would be used by the MMU to map the virtual addresses to real addresses.

Page Number	Valid	Frame Number
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		

page 11	*frame 0*
---	*frame 1*
page 4	*frame 2*
page 0	*frame 3*
---	*frame 4*
page 1	*frame 5*
page 8	*frame 6*
page 5	*frame 7*

Write your answer in the boxes in this table. If the entry for Valid is zero, leave the Frame Number field blank.

(c) [1] A TLB (*Translation Lookaside Buffer*) is normally used to accelerate the translations from virtual addresses to real addresses. Assuming now 16 bit addresses and pages only 2KB in size, suppose the TLB contains the entries shown below. To what real address is the virtual address translated?

Page Number	Valid	Frame Number
7	1	3
3	1	1
6	1	7

Virtual Address = 313C

Real Address = ______________

Question 2. [5] (a) Amdahl's law is stated below. Give a definition for f and p.

$\text{SpeedUp} = \dfrac{1}{f + \dfrac{1-f}{p}}$	f is	p is

In the lecture an example was given for the application of Amdahl's law, with p = 20 and f = 0.2. The speedup was calculated to be 4.1. Then the example was changed to p = 40 and the new speedup became 4.54. For both cases the efficiency was also calculated and comments were made on their relative values.

(b) Calculate the efficiency for both cases.

Efficiency for p = 20	Efficiency for p = 40

(c) What comments can you make on the values above? What do they mean or appear to imply?

Question 3. [5] Fill in the right column of the table with your answers.

Given 16 binary bits, the largest positive integer that can be represented using a 2's complement representation is:	
The hexadecimal value FFD corresponds to the binary:	
The hexadecimal value FFD viewed as an integer in a12-bit 2's complement context, corresponds to the decimal:	
Convert the decimal integer -27 to 8-bit-binary in 2's complement	
Convert the decimal integer -27 to 2 hexadecimal digits in 2's complement	

Question 4. **[2]** In the table below, we see that the relative performance of the IBM 360 Model 75 is 50 times that of the 360 Model 30 (see row 4), yet the instruction cycle time is only 5 times as fast (see row 3). How would you account in general for this discrepancy? (No computation is involved here).

Characteristic	Model 30	Model 40	Model 50	Model 65	Model 75
Maximum memory size (bytes)	64K	256K	256K	512K	512K
Data rate from memory (Mbytes/s)	0.5	0.8	2.0	8.0	16.0
Processor cycle time (μs)	**1.0**	0.625	0.5	0.25	**0.2**
Relative performance	**1**	3.5	10	21	**50**
Maximum number of data channels	3	3	4	6	6
Maximum data rate on one channel (Kbytes/s)	250	400	800	1250	1250

Question 5. **[8]** 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 function, you have learned to have them placed in registers only. Another technique is to place them on the system stack. You have also been told how a mixture of these two protocols is used as a standard convention by the C compiler. 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 on the stack* (new to you). The protocol is described in the following specifications and you are to show your understanding of it by practising on an example.

- The caller routine in this case places all the parameters on the stack using either `STR` or `STMFD` instructions. In this example a mixture is used. [1]

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

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

(a) **[2]** Show the contents of the stack and the stack pointer after the execution of the instructions above (i.e. immediately after the execution of "`BL Foo`").

1. Reminder: STMFD copies registers on the stack from high to low numbers.

- At the entry point in the function "`Foo`", the usual `STMFD` instruction to save registers used locally is issued. This is followed by setting a value for the "`Frame Pointer`", labelled "`FP`", which is register `R12` in ARM. The `Frame Pointer` provides a local pointer within the stack for the *frame* (also called *scope*) of the function 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. If local variables need to be allocated (as is often the case), their space is found on the stack (*not* in the `.DATA`). Why? Because local variables need to be de-allocated after the exit from a function - they disappear. 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.

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

(b) [3] Show the updated contents of the stack, with SP and FP clearly marked, after the execution of the 3 instructions above.

- When exiting a function, 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`.

```
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 the two diagrams (one diagram after each instruction).

(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 6. [12] Consider a small direct-mapped cache which can contain eight 16-bit words, as shown in Table 1. The cache block 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. Con-

Table 1: 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		

sider now the code segment below containing a loop, where all instructions and operands are 2 bytes long (this is basically ARM code but each instruction is 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`	`02E6`	

FIGURE 1. Instructions, their address in memory and their cache block.

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`	

FIGURE 2. Data, address in memory and their cache block.

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) [3] Fill in the right hand columns in Figures 1 and 2 with the cache block number where the instructions and the data will be placed when loaded.

(b) [6] 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 have been drawn. Use one for each iteration, with the requested answers to be written below. The code segment is also replicated for your convenience.

```
LOOP:LDR  R2,[R1],#2
     ADD  R0,R0,R2
     SUBS R3,R3,#1
     BNE  LOOP
```

Iteration 1

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

Iteration 2

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

Iteration 3

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

Iteration 1	Iteration 2	Iteration 3
# memory accesses:	# memory accesses:	# memory accesses:
# cache accesses:	# cache accesses:	# cache accesses:
total time:	total time:	total time:

(c) [3] 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.

Iteration 1

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

Iteration 2

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

Iteration 3

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

Iteration 1	Iteration 2	Iteration 3
# memory accesses:	# memory accesses:	# memory accesses:
# cache accesses:	# cache accesses:	# cache accesses:
total time:	total time:	total time:

Question 7. [6] A benchmark program is run on a 40 MHz[1] processor. The executed program consists of 100,000 instruction executions, with the following instruction mix and clock cycle count:

Instruction type	Instruction count	Cycles per instruction
Integer arithmetic	45,000	1
Data transfer	32,000	2
Floating point	15,000	2
Control transfer	8,000	2

(a) [2] Determine the effective CPI (show your calculations).

(b) [2] Compute the execution time (show your calculations).

(c) [2] A common measure of performance for a processor is the rate at which instructions are executed, expressed as millions of instructions per second (MIPS), referred to as the MIPS rate. We can express the MIPS rate in terms of the clock rate and CPI as follows:

$$\text{MIPS rate} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

State the MIPS rate for the benchmark program above (at least show the complete equation even if you cannot handle the computation).

1. 1 second = 10^9 ns and Hz is cycles/sec. MHz = cycles/sec x 10^6

Question 8. [9] Consider the following sequence of ARM instructions

```
[1]  ADD  r1,r0,#20
[2]  MOV  r3,#99
[3]  ADD  r4,r1,r3
[4]  SUB  r5,r0,r2
```

and suppose that the ARM processor has a four stage pipeline: fetch / decode / execute / write. Further suppose that in the execute phase, any number of registers can be read simultaneously. However, in the write phase, only one register can be written and, furthermore, no registers can be read (preventing the execute step from accessing registers).

(a) [6] Complete the following diagram showing which stage of each instruction is performed in each clock cycle. Use the notation F1, D1, E1 and W1 for the four stages of instruction 1, and similarly for the other instructions. (Note: even though instruction 2 has very little to do in its execute stage, there is still an E2 step to insert into the diagram.) If no action can be performed in some clock cycle for some instruction, write an X in the box. When there is a choice of actions to be performed in a clock cycle, always give priority to the instruction which was fetched earlier.

Time →

Clock Cycle	1	2	3	4	5	6	7	8	9	10
Instruction										
I_1	*F1*	*D1*	*E1*	*W1*						
I_2										
I_3										
I_4										

(b) [1] Your answer to part (a) should have shown some empty boxes where pipeline stalls occurred. Which of these stalls were *structural hazards*? (Give a list of instruction steps which stalled for this reason; e.g. W3 might be an item in this list.)

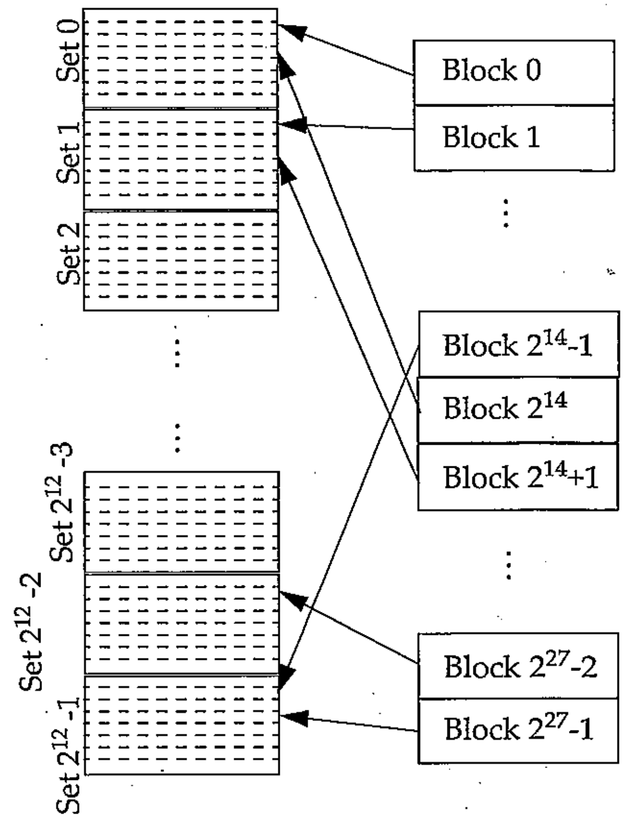
(c) [1] Which of these stalls were *data hazards*?

(d) [1] What is a *control hazard* (or *instruction hazard*)? Give a one sentence definition below.

Question 9. [5] The diagram below shows a memory and an 8-way set associative mapped cache organization. It also shows the mapping of a few memory blocks to their set/cache lines (memory is comprised of 2^{27} blocks).

(a) [2] Redraw the diagram to show what would change if the cache used a direct mapping strategy. Relabel the number of blocks precisely.

Redraw the direct mapped cache here



(b) [3] The memory blocks listed above already showed the corresponding mapped cache block. Now show below the new cache block numbers in the direct mapping organization.

Memory Block Number	Cache Block Number in a Direct Mapped Cache
1	
$2^{14}-1$	
2^{14}	
$2^{14}+1$	
$2^{27}-2$	
$2^{27}-1$	

Question 10. [5] A computer has a cache, main memory, and a disk used for virtual memory. If a referenced word is in the cache, 20ns are required to access it. If it is in main memory but not in the cache, 60 ns are needed to load it into the cache, and the the reference is started again. If the word is not in main memory, 12 ms are required to fetch the word from disk, followed by 60 ns to copy it to the cache, and then the reference is started again. The cache hit ratio is 0.9 and the main memory hit ratio is 0.6. What is the average time *in ns* required to access a referenced word on this system (show your work in all details but the final calculated number is not necessary)?[1]

Question 11. [4] (a) [2] The following is *not* an ARM instruction, but it could be an assembly language instruction in some other processor.

```
LOAD        R3,[[R4],#8]          @ R3 = *((*R4)+8)
```

Its semantics are shown using a C-like expression, as it appears in the comment, assuming that R4 is a pointer containing a valid address in memory and R3 is an integer variable. Show how you would implement this LOAD task using only 2 ARM instructions and only R3 and R4. Make sure to analyze which register changes and which does not.

(b) [2] Consider the following C statement and the declaration of variables in:

```
int  R3;
int  **R4;
R3 = (*(*R4))+4;
```

Show how the same functionality can be implemented using only 2 ARM instructions and only R3 and R4. Make sure to analyze which register changes and which does not.

1. 1 s = 1,000 ms = 1,000,000,000 ns. Thus 1 ms = 10^6 ns

Question 12. **[4]** For a system with two levels of cache, define the following:

T_{C1} = first-level cache access time;

T_{C2} = second-level cache access time;

T_M = memory access time;

H_1 = first-level cache hit ratio;

H_2 = combined first/second level cache hit ratio.

Provide an equation for T, the total cache access time for a read operation.

Question 13. **[6]** A system has Virtual Memory, L1 and L2 caches where the L1 cache is divided into Instruction and Data Cache, MMU, DMA, Page Table and TLB. Explain briefly the sequence of steps occurring for the event of a page fault, in precise words (point form is suggested) or using a flow-chart, .

Question 14. [7] A `For-Loop` which analyses the content of two arrays, M and N, and places some result in a 3rd array, P, is shown below. The initialization and the control of the loop have already been done. The actual body of the loop, containing the `If-Else` statement dealing with the 3 arrays remains to be coded by you and it is given in C-like pseudo code, in two versions (to make sure it is super clear to you). Comments are worth 1 mark. Make sure you implement *only the bold pointer version.*

Pseudo code using array notation

```
for (i=0;i<size;i++) {
   if (M[i] > N[i])then P[i]=M[i];
     else P[i]=N[i];
}
```

Pseudo code using pointer notation

```
for (i=0;i<size;i++)
   if (*M > *N) then *P = *M;
     else *P = *N;
     M++; N++; P++;}
```

```
      @arrays N and M are initialized here with some code
      . . . . . .
      @re-initialize pointers before loop starts
      LDR  R1,=M       @ R1 = address of array M (non empty array)
      LDR  R2,=N       @ R2 = address of array N (non empty array)
      LDR  R3,=P       @ R3 = address of array P
      LDR  R4,=size
      LDR  R4,[R4]     @ R4 = size
      MOV  R5,#0       @loop counter i=0
 FORLOOP:
      CMP  R5,R4       @i<size?
      BGE  DONEFORLOOP
      @write YOUR CODE for the body of the loop here
      @ if (*M > *N) *P = *M; else *P = *N;








      @loop increments
 INCR:ADD  R5,R5,#1    @i++
      ADD  R1,R1,#4    @move pointer for M
      ADD  R2,R2,#4    @move pointer for N
      ADD  R3,R3,#4    @move pointer for P
      BAL  FORLOOP
 DONEFORLOOP:. . . . .
      .data
 M:        .skip     100
 N:        .skip     100
 P:        .skip     100
 size:     .skip     4
```

Question 15. [2] Consider the following code:

```
for (i=0; i<20; i++)
    for (j=0; j<10; j++)
        a[i] = a[i] * j;
```

(a) [2] Give one example of the spatial locality in the code.

Question 16. [8] (a) [7] Number the following steps from 1 to 7 in the order they are performed in processing a general interrupt sequence using the interrupt jump table technique:

	recognize the interrupt event and set the event flag
	load the PC with the address from the interrupt vector table
	execute the first instruction of the general interrupt handling routine
	load the PC with the address of the specific interrupt routine
	push the processor registers onto the stack
	execute the specific interrupt routine
	determine the interrupt vector number

(b) [1] What is the one main difference between a subroutine and an interrupt-service routine?

SELECT ONE OF THE FOLLOWING TWO QUESTIONS. **[4]** No bonus points if you do both.

Question 17. Describe briefly the difference between static and dynamic linking. Drawing a figure with explanations around it might help.

Question 18. State briefly the phases in the compilation process with a short definition of their main function.

THE END!