UNIVERSITY OF VICTORIA

EXAMINATIONS SUMMER 2012

C SC 230 - Introduction to Computer Architecture and Assembly Language - CRN# 30161

STUDENT NUMBER: ______________________________

TIME: 3 hours

INSTRUCTOR: M. Serra

TOTAL MARKS: 116

TO BE ANSWERED ON THE PAPER

Question No.	Value	Mark	Question No.	Value	Mark
1	6		11	4	
2	4		12	2	
3	4		13	5	
4	8		14	4	
5	11		15	3	
6	2		16	11	
7	8		17	9	
8	8		18	3	
9	3		19	15	
10	6		TOTAL	116	

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.
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.

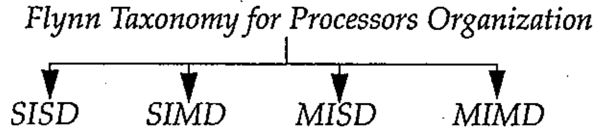
Question 1. [6] Fill in the table below with the appropriate information about the instructions. The column *"Bus used?"* refers ONLY to the execution phase, not the fetch phase.

Instruction	Addressing Modes of All Operands	Bus used?	What it does (be precise)
`CMP r1,#1`			
`STR r1,[r2,#4]!`			
`MOV r1,r2,LSL #4`			

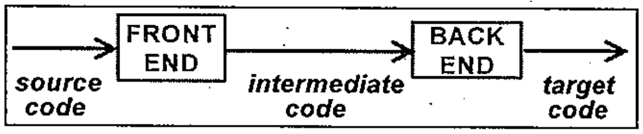
Question 2. [4] **(a)** [3] State at least 3 main characteristics to differentiate between static and dynamic RAM. Show how they apply to each.

(b) [1] State at least 1 main characteristic to differentiate between volatile and non-volatile memory.

Question 3. [4] **(a)** [2] Choose one portion of the Flynn taxonomy and describe it *briefly*.



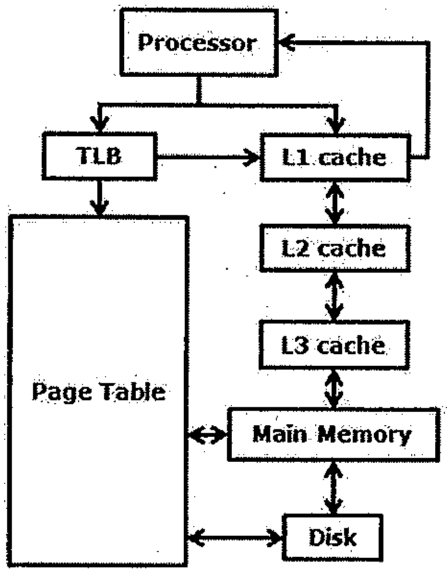
(b) [2] Draw a diagram showing the difference organization of what is called a "Multi-core" design versus a "Multi-processor" design.



Question 4. [8] (a) [5] The figure shows the structure of a compiler decomposed into a "Front End" and a "Back End" with Intermediate Code generated in between. Explain briefly in point form what the units do, how, what the features and dependencies are; in summary, anything which gives a terse yet complete explanation. In an interview situation, you should not take more than 2 minutes; be similarly concise and precise.

(b) [3] Explain briefly what is dynamic linking. Be precise and concise.

Question 5. [11] The diagram below is a reproduction from your textbook, discussed in class (we talked also about other possibilities for arrows). Consider such a system with Virtual Memory, using an MMU, a Page Table and a TLB, plus 3 levels of cache, L1, L2 and L3.



(a) [1] Where is the TLB located?

(b) [1] Where is the Page Table located?

(c) [1] Where is the MMU located?

(d) [8] Suppose the CPU needs access to some data. There are 4 cases:

1. The data is in L1
2. The data is in L2 or in L3
3. The data is in memory
4. The data is on disk

State algorithmically the steps to be followed in the search and retrieval of this data for each of the 4 cases, clearly naming which component is involved and how.

Case 1: data is in L1 cache

Case 2: data is in L2 or L3 cache

Case 3: data is in memory

Case 4: data is on disk

Question 6. [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).

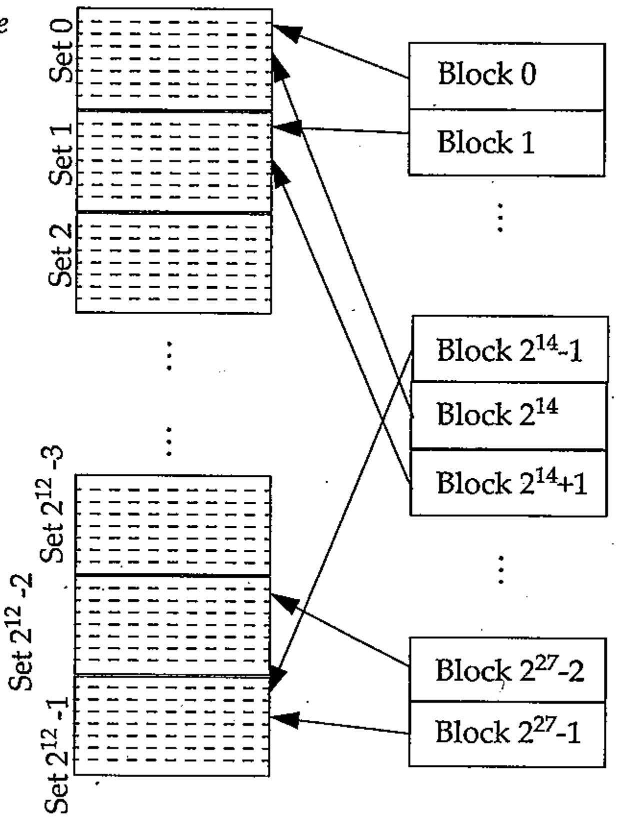
Characteristic	Model 30	Model 40	Model 50	Model 65	Model 75
Maximum memory size (bytes)	64K	256K	256K	512K	512K
Data rate from memory (Mbytes/s)	0.5	0.8	2.0	8.0	16.0
Processor cycle time (μs)	1.0	0.625	0.5	0.25	0.2
Relative speed	1	3.5	10	21	50
Maximum number of data channels	3	3	4	6	6
Maximum data rate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

Question 7. [8] Write the ARM code segment to implement the loop structure given below in C. Make sure to add comments. Assume that `X` has been declared in the DATA section as '`X:  .skip 4`'

```
X=15;
for (k=100;k>0;k--){
     X = X+3;
}
```

Question 8. **[8]** 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). Redraw the diagram to show what would change if the cache used a 4-way set associative mapped cache organization. and state which cache blocks would be used by the memory blocks in the order presented in the table below. Relabel the number of blocks precisely.

Redraw the 4-way set associative mapped cache here



Memory Block Number	State where it is mapped in a 4-way set associative Cache
1	
2^{14} -1	
2^{14}	
2^{14} +1	
2^{27} -2	
2^{27} -1	

Question 9. [3] Given the 4-bit hexadecimal number below, state the *decimal* equivalent according to the representation shown in each heading:

Hexadecimal	Unsigned Integer	2's Complement	Signed Magnitude
$D2_{16}$			

Question 10. [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. Hz is cycles/sec. MHz = cycles/sec x 10^6

Question 11. [4] Consider a hypothetical 32-bit microprocessor having 32-bit instructions composed of two fields: the first byte contains the opcode and the remainder the immediate operand or an operand address.

(a) What is the maximum directly addressable space capacity (in bytes)? ____________________

(b) What is the maximum number of instructions which can be supported? ____________________

(c) What is the smallest size needed in bits for the program counter and the instruction register?
PC: ____________________ IR: ____________________

Question 12. [2] Consider the following code:

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

(a) [1] Give one example of the spatial locality in the code.

(b) [1] Give one example of the temporal locality in the code.

Question 13. [5] A processor accesses main memory with an average access time of *T2*. A smaller cache memory is interposed between the processor and main memory. The cache has a significantly faster access time of *T1* < *T2*. The cache holds, at any time, copies of some main memory words and is designed so that the words more likely to be accessed in the near future are in the cache. Assume that the probability that the next word accessed by the processor is in the cache is *H* (known as the Hit ratio).

(a) [1] For any single memory access, what is the theoretical speedup of accessing the word in the cache rather than in main memory?

(b) [2] Let *T* be the average access time. Express *T* as a function of *T1*, *T2* and *H*.

(c) [2] In practice, a system may be designed so that the processor must first access the cache to determine if the word is in the cache and, if it is not, then access main memory, so that on a miss (opposite of a hit), memory access time is *T1*+*T2*. Express *T* as a function of *T1*, *T2* and *H*, with the new assumption.

Question 14. [4] Consider an L1 cache with an access time of 1 ns and a hit ratio of H=0.95. The memory access, in the case of a cache miss, is 10 ns. Suppose that we can change the cache design (size of cache, cache organization) such that we increase H to 0.97, but increase cache access time to 1.5 ns (memory access time remains the same). In general, average access time from CPU can be stated as:

$$\text{T average} = H \times \text{cache access time} + (1 - H) \times \text{memory access time}$$

(a) [1] State T1 as the average CPU access time before the cache redesign.

(b) [1] State T2 as the average CPU access time after the cache redesign.

(c) [2] Let a general memory access time be M (instead of the 10 ns above). If we want the cache redesign at T2 to result in improved performance from T1, what condition on M must be met by comparing T2 and T1?

Question 15. **[3]** In the context of disk drives, define the terms *seek time, rotational delay, access time.*

Question 16. [11] (a) [2] When a device interrupt occurs, how does a processor determine which device issued the interrupt?

(b) [5] Number the following steps from 1 to 5 in the order they are performed in processing a general interrupt sequence using the interrupt jump table technique:

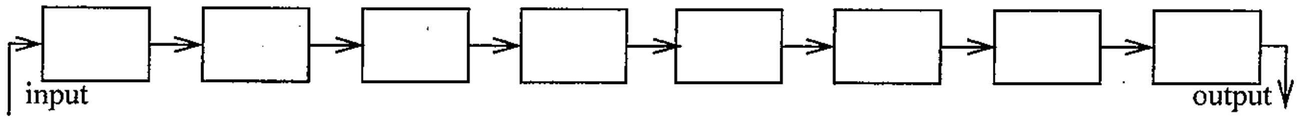
	recognize the interrupt event and set the event flag
	load the PC with the address from the interrupt vector table
	execute the first instruction of the interrupt handling routine
	push the processor registers onto the stack
	determine the interrupt vector number

(c) [2] Explain briefly the necessity for "levels" of interrupts.

(d) [2] After an exception or interrupt has been processed, it may be the case that normal processing can resume within the application. Give two examples in which this is not the case.

Question 17. [9] (a) [4] When describing the functionality of a pipelined execution processor, three possibilities for hazards, which can reduce the performance increase, were described. State what hazards are possible and give a complete definition for *one* of them (your choice).

(b) [2] Suppose that you need to execute 152 processes and that each process requires 8 clock cycles. In total, the 152 processes would take 152 × 8 = 1,216 clock cycles. Now suppose that you are given a pipeline with 8 stages to perform the processes, as drawn below. How many clock cycles



will it take if there are no hazards? Show the answer as a number and also show how you calculated it.

(c) [2] When evaluating the performance of a pipeline, we calculate the possible speedup as the ratio:

$$Speedup = \frac{T_{serial}}{T_{pipeline}}$$

What is the speedup that can be obtained using the pipeline of part (b)?

(d) [1] In general, the speedup can be calculated as:

$$Speedup = \frac{T_{serial}}{T_{pipeline}} = \frac{m \times N}{m + N - 1}$$

where m denotes the number of stages in a pipeline and N denotes the number of items to be processed. If one assumes that $N \gg m$, what can you say about the asymptotic speedup which could be obtained in theory?

Question 18. [3] Circle the correct value returned in R0 after executing the following code segment:

```
          .equ P,6
          LDR  R9,#1
          LDR  R8,#P
          CMP  R8,#0
          BEQ  Done
Loop:     MOV  R9,R9,ASL #1
          SUBS R8,R8,#1
          BNE  Loop
Done:     MOV  R0,R9
```

(a) R0 = 10^6	**(b) R0 = 2^6**	**(c) R0 = 10^{-6}**	**(d) R0 = 2^{-6}**	**(e) R0 = 2*6**

Question 19. [15] The subroutine *Identity* is required to initialize the elements of a $N \times N$ matrix to be all zeroes, except for the elements along the diagonal which are initialized to one. For example, a 3×3 matrix should be initialized to the values shown on the right. The maximum size for any matrix was given as:
`#define MAXSIZE.`

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(a) [5] Give the C code for the *Identity* subroutine. It is called with two parameters: the address of a $N \times N$ matrix of integers and the value of N.

(b) [10] Give ARM code for the *Identity* subroutine. The address of the matrix is passed in `R1` and N is passed in `R2`. No result is returned in any register. The matrix elements are arranged consecutively in memory so that an element $A[i, j]$ has the address $A + (i \times N + j) \times 4$, where $0 \le i < N$, $0 \le j < N$.

```
@ Identity(MatrixAddress, N)
@    initializes a N by N matrix at address MatrixAddress to be
@    an identity matrix. On entry, R1 = MatrixAddress, R2 = N.
Identity:
     stmfd sp!,{r0-r10,lr}  ; save all registers




     ldmfd sp!,{r0-r10,pc}  ; restore registers and return
```

END

APPENDIX CSC 230

Operation	Assembler	Action
Move	MOV{S} Rd, <Oprnd2>	Rd := Oprnd2 {CPSR}
	MVN{S} Rd, <Oprnd2>	Rd := NOT Oprnd2 {CPSR}
Arithmetic	ADD{S} Rd, Rn, <Oprnd2>	Rd := Rn + Oprnd2 {CPSR}
	ADC{S} Rd, Rn, <Oprnd2>	Rd := Rn + Oprnd2 + Carry {CPSR}
	SUB{S} Rd, Rn, <Oprnd2>	Rd := Rn - Oprnd2 {CPSR}
	SBC{S} Rd, Rn, <Oprnd2>	Rd := Rn + Oprnd2 + Carry {CPSR}
	RSB{S} Rd, Rn, <Oprnd2>	Rd := Oprnd2 - Rn {CPSR}
	RSC{S} Rd, Rn, <Oprnd2>	Rd := Oprnd2 - Rn - NOTCarry {CPSR}
	MUL{S} Rd, Rm, Rs	Rd := Rm * Rs {CPSR}
	MLA{S} Rd, Rm, Rs, Rn	Rd := Rm * Rs + Rn {CPSR}
	CLZ Rd, Rm	Rd := # leading zero in Rm
Logical	AND{S} Rd, Rn, <Oprnd2>	Rd := Rn AND Oprnd2 {CPSR}
	EOR{S} Rd, Rn, <Oprnd2>	Rd := Rn EXOR Oprnd2 {CPSR}
	ORR{S} Rd, Rn, <Oprnd2>	Rd := Rn OR Oprnd2 {CPSR}
	TST Rn, <Oprnd2>	Update CPSR on Rn AND Oprnd2
	TEQ Rn, <Oprnd2>	Update CPSR on Rn EOR Oprnd2
	BIC{S} Rd, Rn, <Oprnd2>	Rd := Rn AND NOT Oprnd2 {CPSR}
	NOP	R0 := R0
Compare	CMP Rd, <Oprnd2>	Update CPSR on Rn - Oprnd2
Branch	B{cond} label	R15 := label
	BL{cond} label	R14 := R15-4; R15 := label
Swap	SWP Rd, Rm	temp := Rn; Rn := Rm; Rd := temp
Load	LDR Rd, <a_mode2>	Rd := address
	LDM <a_mode4L> Rd{!}, <reglist>	Load list of registers from [Rd]
Store	STR Rd, <a_mode2>	[address]:= Rd
	STM <a_mode4S> Rd{!}, <reglist>	Store list of registers to [Rd]
SWI	SWI <immed_24>	Software Interrupt

Addressing Mode 2 - Data Transfer		
Pre-indexed	Immediate offset	[Rn, #+/-<immed_12>]{!}
	Zero offset	[Rn]
	Register offset	[Rn, +/-Rm]{!}
	Scaled register offset	[Rn, +/-Rm, LSL #<immed_5>]{!}
		[Rn, +/-Rm, LSR#<immed_5>]{!}
		[Rn, +/-Rm, ASR #<immed_5>]{!}
		[Rn, +/-Rm, ROR #<immed_5>]{!}
		[Rn, +/-Rm, RRX]{!}
Post-indexed	Immediate offset	[Rn], #+/-<immed_12>
	Register offset	[Rn], +/-Rm
	Zero offset	[Rn]
	Scaled register offset	[Rn], +/-Rm, LSL #<immed_5>
		[Rn], +/-Rm, LSR #<immed_5>
		[Rn], +/-Rm, ASR #<immed_5>
		[Rn], +/-Rm, ROR #<immed_5>
		[Rn], +/-Rm, RRX

Key to tables	
{cond}	See Condition Field
<Oprnd2>	See Operand 2
{S}	Updates CPSR if present
<immed>	Constant
<a_mode2>	See Addressing Mode 2
<a_mode4>	See Addressing Mode 4
<reglist>	List of registers with commas
{!}	Updates base register if present

OVER

CSC 250 Table 2

Operand 2	
Immediate value	#<immed_8>
Logical shift left immediate	Rm, LSL #<immed_5>
Logical shift right immediate	Rm, LSR #<immed_5>
Arithmetic shift right immediate	Rm, ASR #<immed_5>
Rotate right immediate	Rm, ROR #<immed_5>
Register	Rm
Rotate right extended	Rm, RRX
Logical shift left register	Rm, LSL Rs
Logical shift right register	Rm, LSR Rs
Arithmetic shift right register	Rm, ASR Rs
Rotate right register	Rm, ROR Rs

Condition Field	
EQ	Equal
NE	Not equal
CS	Carry Set
CC	Carry clear
MI	Negative
PL	Positive or zero
VS	Overflow
VC	No overflow
HI	Unsigned higher
LS	Unsigned lower or same
GE	Signed greater or equal
LT	Signed less than
GT	Signed greater than
LE	Signed less than or equal
AL	Always

Dec	Bin	Hex
0	00000000	00
1	00000001	01
2	00000010	02
3	00000011	03
4	00000100	04
5	00000101	05
6	00000110	06
7	00000111	07
8	00001000	08
9	00001001	09
10	00001010	0A
11	00001011	0B
12	00001100	0C
13	00001101	0D
14	00001110	0E
15	00001111	0F

Addressing Mode 4 - Multiple Data Transfer			
Block load		**Stack pop**	
IA	Increment After	FD	Full Descending
IB	Increment Before	ED	Empty Descending
DA	Decrement After	FA	Full Ascending
DB	Decrement Before	EA	Empty Ascending
Block store		**Stack push**	
IA	Increment After	EA	Empty Ascending
IB	Increment Before	FA	Full Ascending
DA	Decrement After	ED	Empty Descending
DB	Decrement Before	FD	Full Descending

D	BIN	H	D	BIN	H	D	BIN	H	D	BIN	H
0	00000000	00	4	00000100	04	8	00001000	08	12	00001100	0C
1	00000001	01	5	00000101	05	9	00001001	09	13	00001101	0D
2	00000010	02	6	00000110	06	10	00001010	0A	14	00001110	0E
3	00000011	03	7	00000111	07	11	00001011	0B	15	00001111	0F

OVER

CSC 230 PAGE 2

Operand 2	
Immediate value	#<immed_8>
Logical shift left immediate	Rm, LSL #<immed_5>
Logical shift right immediate	Rm, LSR #<immed_5>
Arithmetic shift right immediate	Rm, ASR #<immed_5>
Rotate right immediate	Rm, ROR #<immed_5>
Register	Rm
Rotate right extended	Rm, RRX
Logical shift left register	Rm, LSL Rs
Logical shift right register	Rm, LSR Rs
Arithmetic shift right register	Rm, ASR Rs
Rotate right register	Rm, ROR Rs

Condition Field	
EQ	Equal
NE	Not equal
CS	Carry Set
CC	Carry clear
MI	Negative
PL	Positive or zero
VS	Overflow
VC	No overflow
HI	Unsigned higher
LS	Unsigned lower or same
GE	Signed greater or equal
LT	Signed less than
GT	Signed greater than
LE	Signed less than or equal
AL	Always

Dec	Bin	Hex
0	00000000	00
1	00000001	01
2	00000010	02
3	00000011	03
4	00000100	04
5	00000101	05
6	00000110	06
7	00000111	07
8	00001000	08
9	00001001	09
10	00001010	0A
11	00001011	0B
12	00001100	0C
13	00001101	0D
14	00001110	0E
15	00001111	0F

Addressing Mode 4 - Multiple Data Transfer			
Block load		**Stack pop**	
IA	Increment After	FD	Full Descending
IB	Increment Before	ED	Empty Descending
DA	Decrement After	FA	Full Ascending
DB	Decrement Before	EA	Empty Ascending
Block store		**Stack push**	
IA	Increment After	EA	Empty Ascending
IB	Increment Before	FA	Full Ascending
DA	Decrement After	ED	Empty Descending
DB	Decrement Before	FD	Full Descending

D	BIN	H	D	BIN	H	D	BIN	H	D	BIN	H
0	00000000	00	4	00000100	04	8	00001000	08	12	00001100	0C
1	00000001	01	5	00000101	05	9	00001001	09	13	00001101	0D
2	00000010	02	6	00000110	06	10	00001010	0A	14	00001110	0E
3	00000011	03	7	00000111	07	11	00001011	0B	15	00001111	0F

OVER