UNIVERSITY OF VICTORIA
FINAL EXAM DECEMBER 2010

Course Name & No.:	C SC 230 - Introduction to Computer Architecture and Assembly Language
Section(s):	A01 and A02
CRN:	10853
Instructor:	M. Serra
Duration:	3 hours
STUDENT NUMBER:	

Question No.	Value	Mark	Question No.	Value	Mark
1	*10*		10	*9*	
2	*8*		11	*8*	
3	*2*		12	*6*	
4	*3*		13	*9*	
5	*8*		14	*15*	
6	*12*		15	*12*	
7	*8*		16	*4*	
8	*4*		17	14	
9	10		TOTAL	142	

INSTRUCTIONS:

1. Students must count the number of pages and report any discrepancy immediately to the Invigilator.
2. This examination paper has a total of 14 pages including this cover page and 2 pages of handout.
3. No aids are permitted. However, an Appendix describing the ARM instruction set is provided.
4. The marks assigned to each question are shown within square brackets. Partial marks are available for all questions.
5. This exam is to be answered on the paper.
6. It is strongly recommended that you read the entire exam through from beginning to end before beginning to answer the questions.

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

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

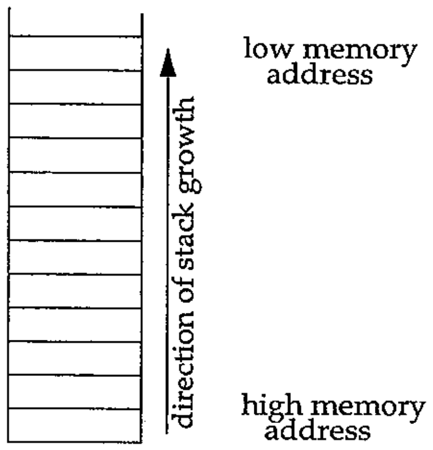
Dec	Bin	Hex
0	00000000	00
1	00000001	01
2	00000010	02
3	00000011	03
4	00000100	04
5	00000101	05
6	00000110	06
7	00000111	07
8	00001000	08
9	00001001	09
10	00001010	0A
11	00001011	0B
12	00001100	0C
13	00001101	0D
14	00001110	0E
15	00001111	0F

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

Question 1. [10] The main part of a program is calling a function named FOO passing to it 6 parameters. The first 4 parameters have already been placed in registers R0,R1,R2 and R3, as the standard conventions expect (their type is irrelevant at this point). The next two parameters, the 5th and 6th ones, are currently in registers R9 and R8 and need to be pushed on the stack, again following the C convention, *from right to left* - this implies that the 6th parameter is pushed on the stack first! After executing, FOO returns an integer which will be placed in R0 (as the standard conventions expect). Thus FOO would be something of the form: direction of stack growth

```
Result = FOO (A,B,C,D,E,F);
```

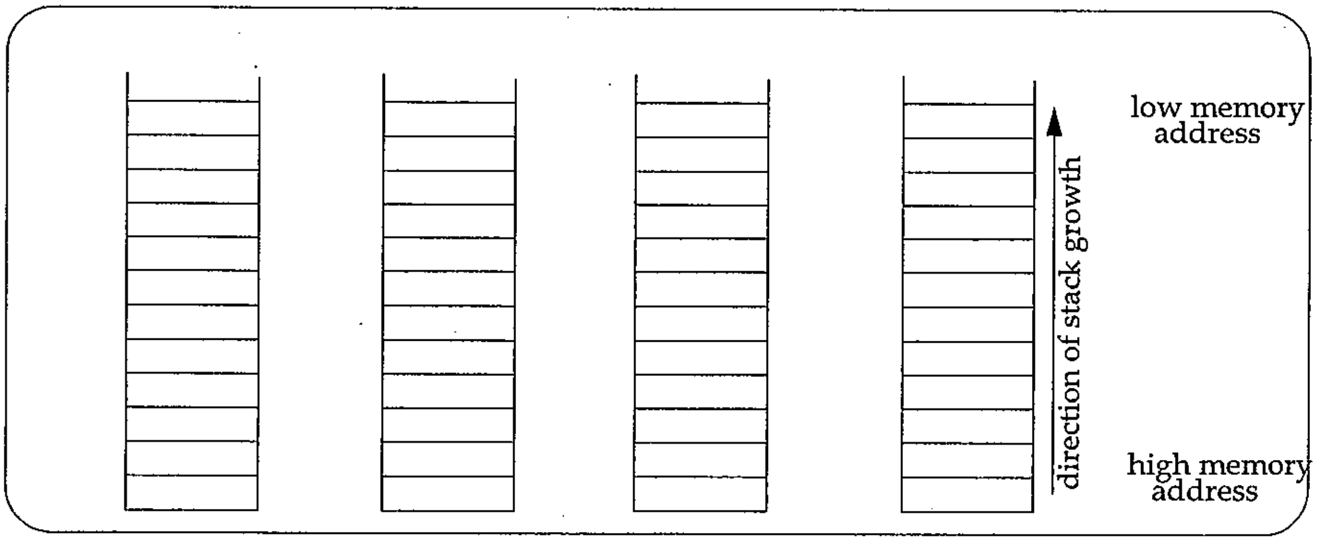
(a) [2] State the ARM instruction(s) needed in the main part to place the 5th and 6th parameter on the stack as expected by FOO.	**(b)** [2] Draw the stack after execution of your instruction(s). Show the Stack pointer.  low memory address direction of stack growth high memory address

(c) [2] Inside FOO, the registers containing the parameters are used for local computing as well, yet they are expected to be returned unchanged. Moreover FOO needs to use registers R5,R7 and R10 for local computation. State what the first and last ARM instructions in FOO should be to enforce this protocol.

First: ____________________

Last: ____________________

(d) [4] The system stack may have been changed by the instructions you gave above. Draw the content of the stack, with appropriate pointers, before and after each of the 2 instructions you stated.



Question 2. **[4] [8]** The C language provides two different ways to process the elements of an array, as illustrated by the following two versions of a function for summing the elements of an array.

```
int Plus1( int *L, int N ) {
     int i, p;
     i = p = 0;
     while( i<N ) {
          p += L[i]; i++;
     }
     return p;
}
```

```
int Plus2( int *L, int N ) {
     int *LEnd, p;
     p = 0;
     LEnd = L+N;
     while( L < LEnd ) {
          p += *L; L++;
     }
     return p;
}
```

(a) [4] Assuming that `L`, `N`, `i` and `p` are held in registers R1, R2, R3 and R4 respectively, give the ARM assembly language code which corresponds to the two statements `p+=L[i]; i++;` in *Plus1*. Please note carefully that you must give the code ***only*** for the two statements, *not* for the whole loop/function.

(b) [4] Assuming that `L`, `N`, `LEnd` and `p` are held in registers R1, R2, R3 and R4 respectively, give the ARM assembly language code which corresponds to the two statements `p+=*L; L++;` in *Plus2*.

Question 3. [2] In your program for the elevator controller (assignment 3) you used the function `SetButtonsArray` which was given to you. The function contained the following code segment which you were expected to understand fully. Show now your understanding by stating what the value in `R0` is after the execution of the last line, given that initially `R0 = 0x00004000`.

```
clz  r1,r0          @count leading zeros
rsb  r0,r1,#32      @position of leading 1
sub  r0,r0,#1       @index=button-1
```

Value in R0 = ______________________

Question 4. [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
D_{16}			

Question 5. [8] **(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 in a system?

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

(d) [2] 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, you do not need to describe fully.

Question 6. **[12]** Consider the ARM program below.

(a) [4] Trace it and give the result for R0 where indicated next to the code itself.

(b) [2] What does the function Mystery do? Give a precise explanation in English of what it does - its purpose as you would explain it in the top few lines of header documentation. It is not appropriate to give a line by line summary of the execution.

```
          .equ         NUL,0           ;Null character
          .equ         SWI_EXIT,0x11
          .text
          .global _start
_start:                ldr   r1,=M1
          ldr          r2,=M2
          bl           Mystery     @ int:r0 Mystery
                                   @(&M1:r1;&M2:r2)
@ here the result is in r0 and it is printed
@      ==> fill in for R0 here
          ldr          r2,=M3
          bl           Mystery     @ int:r0 Mystery
                                   @(*M1:r1;*M3:r2)
@ here the result is in r0 and it is printed
@      ==> fill in for R0 here
          SWI          SWI_EXIT    @ end of program
Mystery:
          STMFD        sp!,{r1-r4,lr}
          mov          r0,#0
AGAIN:
          ldrb         r3,[r1],#1
          ldrb         r4,[r2],#1
          cmp          r4,#NUL
          beq          Return
          cmp          r3,r4
          beq          AGAIN
WHY:  mov              r0,#-1
Return:                LDMFD       sp!,{r1-r4,pc};
          .data
M1:             .asciz      "abcdabcca"
M2:             .asciz      "abcd"
M3:             .asciz      "bcc"
          .end
```

R0 = ______________________ **[2]**

R0 = ______________________ **[2]**

(c) [6] Give the correct (and elegant) C code corresponding to the ARM function "Mystery" above. Give the code *only* for the function, not the whole program. (Hint: use the ARM code as your guideline pseudo-code, do not re-invemt an algorithm)

```
int Mystery (char *M1, char *M2) {











}
```

Question 7. **[8]** Explain briefly what happens when dynamic linking is used. Pretend that this is one of the technical questions in a co-op job interview, that is, be clear and precise, but also concise (you would only be given a couple of minutes to state your answer).

Question 8. [4] Program execution time, *T*, can be defined as:

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

- *T* is the total elapsed time,
- *N* is the number of machine language instructions used during the execution (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 *T* for a certain high-level language program. The program can be run on a RISC or a CISC computer. Both computers use pipelined instruction execution and have the same clock rate *R*. However pipelining in the RISC machine is more effective than in the CISC machine, such that the effective value of *Sr* for the RISC machine is 1.2, but it is *Sc* = 1.5 for the CISC machine. The machine code for RISC and CISC produced from the respective compilers contains a different number of executable instructions, labelled *Nr* and *Nc*, respectively, with *Nc* having 80% of the number of instructions of *Nr* (i.e. the CISC program has fewer executable instructions). What can you conclude about the performance of this program on the two systems? Show your reasoning.

(a) [1] State the equation for the elapsed time in RISC:	Tr =
(b) [1] State the equation for the elapsed time in CISC:	Tc =
(c) [2] Substitute values for *Sc, Sr, Nc* and *Nr* in the equations above and state your conclusions about the relative performances *Tr* and *Tc*	

Question 9. [10] (a) [4] There is at least one READ command issued by the CPU for each instruction. It occurs during the "Fetch" cycle, namely to copy the instruction from memory into the IR. After decoding, does the ARM instruction "`ADDS R2,R2,#3`" need another bus cycle to access memory for either a READ or a WRITE? Answer YES or NO and then state what the instruction is supposed to do.

What does the instruction do? **CIRCLE one: YES or NO**

(b) [4] Repeat the question above for the instruction `STR "R2,[R3,R5, LSL #4]"`.

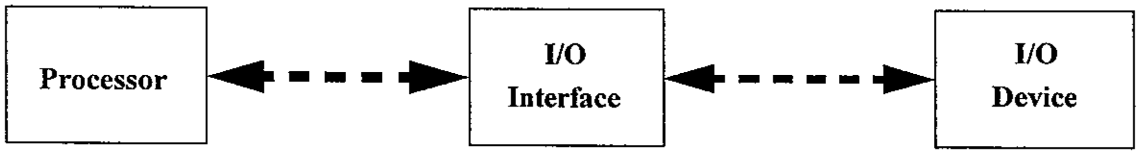
What does the instruction do? **CIRCLE one: YES or NO**

(c) [2] When "`STR R2,[R3,R5, LSL #4]`" is translated by the assembler into binary object code, what is stored in the instruction itself? You are absolutely NOT expected to be able to give the exact content – state only what "kind" of information or values are found in the various portions.

Question 10. [9] For each feature listed below, indicate with an "X" in the appropriate box(es) whether it *usually* appears in a RISC-based system, in a CISC-based system, or in neither. It is OK to tick both the RISC and CISC boxes if the feature applies to both. (Note the emphasis on the word '*usually*' – no common computers can be described as being pure-RISC or pure-CISC in nature.)

Feature	RISC	CISC	Neither
Has a large number of general-purpose registers			
Operands must be in registers for arithmetic instructions			
Has a simple instruction set			
Has a single instruction size (all instructions occupy the same number of bytes)			
Has a single data operand size (all memory operands occupy the same number of bytes)			
An instruction is fetched in each clock cycle			
Every instruction is 4 bytes in size			
The typical instruction allows many addressing modes for its operands			
Supports interrupts			

Question 11. [8] Name the two methods of synchronization between a Processor and the Interface chips for the peripherals. Describe each method, very briefly, as presented in class.



Question 12. [6] Suppose you have the following:

i) a compiler front-end which understands the programming language C

ii) a compiler front-end which understands the programming language Ada

iii) a compiler back-end which understands the architecture of the Intel x86 processor

iv) a compiler back-end which understands the architecture of the PowerPC processor

The intermediate format produced by the front-ends is identical for all four translation and is the input used by the back-ends. Indicate which of the following language translations you can do using the above components:

Translation	YES/NO
C code → Ada code	
C code → PowerPC code	
Ada code → ARM code	
PowerPC code → Intel x86 code	
FORTRAN code → Intel x86 code	
Ada code → PowerPC code	

Question 13. [9] You have seen the calculation for hit and miss times for system with and without a cache. Now consider a system with a two-level cache, L1 and L2. The L1 hit time is 5 ns, the L2 hit time is 20 ns, and the L2 miss time is 100 ns (that is, this is the time to access memory). Assume that there are 10,000 references from the CPU. Of these, 9,900 are found in L1 (L1 hits). Of the remaining 100 accesses, 90 are L1 misses but are L2 hits, and 10 are L2 misses.

(a) [3] Compute the hit ratio for L1, that is, the ratio of the number of times the result is in the L1 cache to the total number of accesses. State the calculation (equation), you do not need to calculate an actual number result.

(b) [3] Repeat the process for the L2 cache. State the calculation (equation), you do not need to calculate an actual number result.

(c) [3] Now compute the total effective access time over the whole 10,000 accesses, using the hit and miss times in ns shown above. State the calculation (equation), you do not need to calculate an actual number result.

Question 14. [15] Consider a system with Virtual Memory, using an MMU, a Page Table and a TLB, plus 2 levels of cache, L1 and L2.

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

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

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

(d) [12] Suppose the CPU needs access to some data. There are 4 cases:

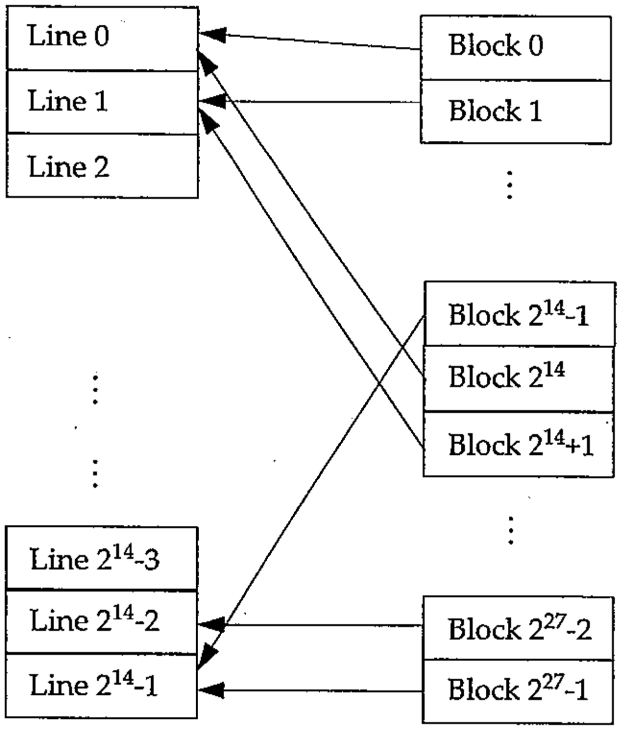
1. The data is in L1
2. The data is in L2
3. The data is in memory
4. The data is on disk

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 in involved and how.

Case 1: data in L1 cache
Case 2: data in L2 cache
Case 3: data is in memory
Case 4: data is on disk

Question 15. **[12]** The diagram below shows a memory and a direct mapped cache organization. It also shows the mapping of a few memory blocks to their cache lines. Redraw the diagram to show what would change if the cache used a 4-way associative mapping strategy and state which cache blocks would be used by the memory blocks in the order presented in the table below.

Redraw the cache here



Memory Block Number	State where it is mapped to a 4-way Associative Cache
0	
1	
2^{14} -1	
2^{14}	
2^{14} +1	
2^{27} -2	
2^{27} -1	

Question 16. [4] State Amdhal's Law, define each element and describe briefly its main significance regarding the speedup which can be obtained with parallelism.

Question 17. [14] Read the following passage and then answer some questions about it.

The graduate advisor of the department is sitting in her office reading e-mail. A student comes to the open office door and asks if she can come in. The student and the graduate advisor start discussing the course requirements for a Master degree. A high pitched noise startles them both. After a few seconds, when the noise repeats itself for the 3rd time, the advisor realizes that her personal cell phone is ringing. Since it could only be a personal call, arriving during work time, she chooses to ignore it, she clicks the phone off and the discussion resumes. However, a short time later a very similar shrill noise interrupts them again and after a couple of seconds it becomes clear that it is the office phone ringing. The advisor excuses herself and answers the phone. While talking, another student appears at the door trying to get her attention, but she waves that student away, gesturing to come back later. After finishing the brief phone call, the advisor hangs up and returns to the student in the office. Soon afterwards, another loud noise intrudes again – this time it is a colleague tapping on the door frame, loudly reminding the advisor of an important meeting which is just about to start. The discussion with the student is immediately terminated and the advisor leaves the office with the colleague. While they are walking to the meeting room, loud bells resonate through the hallway – the fire alarm has been activated. The advisor and her colleague promptly leave the building, and work for that morning is suspended indefinitely.

(a) We can consider the graduate advisor to be analogous to a CPU which has a number of different tasks to perform, where these tasks are initiated by external events (corresponding to interrupts). List each interrupt and the associated possible task in the table on the next page. (Note: there may be more than one occurrence of a particular interrupt/task combination – in which case it is to be listed only once, and the table may have more rows than are needed to answer the question.)

(b) It is clear that some tasks can interrupt other tasks. For each interrupt listed in the table, show a possible priority number according to how they have been handled in the story above. Note that some events are allowed to share the same priority level. Use 1 for the highest priority task, use 2 for the second highest priority task, etc.

(c) Each interrupt has to be handled in its own way. For each kind of interrupt which you listed in the table, briefly describe what has to be done to handle the interrupt in the context of the story. For example, "*accept or not*", "*suspend and resume later after ...*", "*quit task*", etc.

	Form of Interrupt and the Associated Task	Action	Priority
A	• *INTERRUPT:* • *TASK:*		
B	• *INTERRUPT:* • *TASK:*		
C	• *INTERRUPT:* • *TASK:*		
D	• *INTERRUPT:* • *TASK:*		
E	• *INTERRUPT:* • *TASK:*		
F	• *INTERRUPT:* • *TASK:*		
G	• *INTERRUPT:* • *TASK:*		

- - - - THE END - - - -