UNIVERSITY OF VICTORIA
EXAMINATIONS SPRING 2014
C SC 230 - Introduction to Computer Architecture and Assembly Language - A01 CRN# 20684

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	2		10	7	
2	3		11	2	
3	2		12	7	
4	9		13	6	
5	2		14	9	
6	6		15	5	
7	5		16	8	
8	3		17	8	
9	4		18	12	
			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 13 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. [2] Figure 1 shows the bus organization when there is a single bus for memory and peripherals. This is called a "Memory-Mapped" topology.

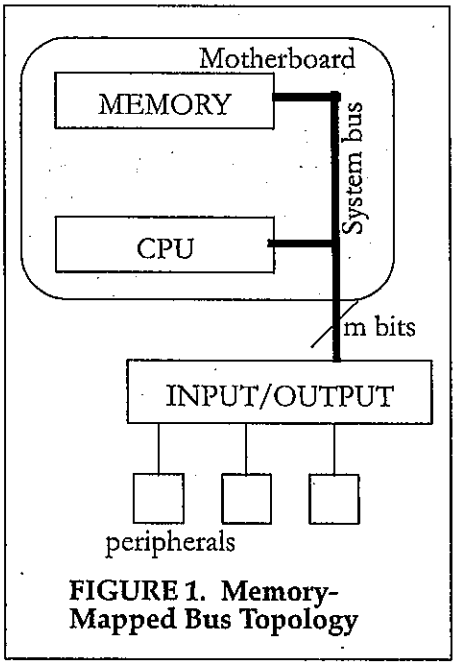
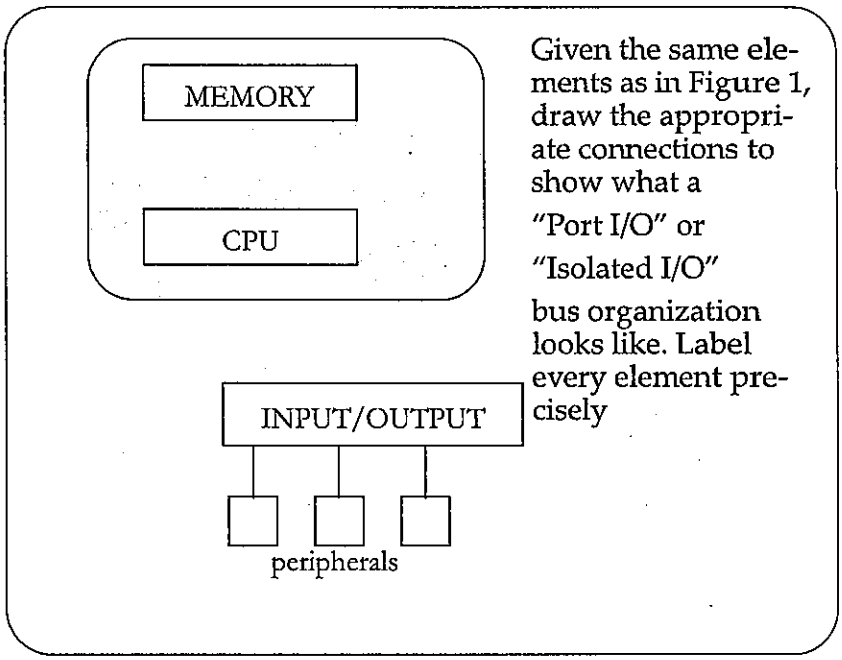


FIGURE 1. Memory-Mapped Bus Topology



Question 2. [3] 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 called "M", whose size is K elements.

```
int Sum1( int *M, int K ) {
      int i, ss;
      i = ss = 0;
      while( i<K ) {
            ss += M[i]; i++;
      }
      return ss;
}
```

```
int Sum2( int *M, int K ) {
      int *MEnd, ss;
      ss = 0;
      MEnd = M+K;
      while( M < MEnd ) {
            ss += *M; M++;
      }
      return ss;
}
```

(a) [2] What is the C datatype of M in *Sum1* and *Sum2*? Tick the most appropriate answer below:

☐ **int** ☐ **array of int** ☐ **pointer to int** ☐ **pointer to array of int**

(b) [1] Suppose that *Sum2* is called with the values `0xA000` for `M` and `7` for `K`. What value, in hexadecimal, is stored in variable `MEnd` by the statement which initializes it? Tick the correct answer.

☐ `0xA01C` ☐ `0xA028` ☐ `0xA018` ☐ `0xA007`

Question 3. [2] A program compiled for a SPARC ISA writes the 32-bit unsigned integer 0xABCDEF01 to a file, and reads it back correctly. The same program compiled for a Pentium ISA also works correctly. However, when the file is transferred between machines, the program incor-

rectly reads the integer from the file as `0x 01EFCDAB`. What is going wrong? Can you identify a possible reason for the behaviour (without going into deep explanations)?

Question 4. [9]: Given the following code segment, give the content of registers or variables, as indicated at the various points in the computation. The instructions are not independent, they follow each other sequentially even if the overall code is probably nonsensical. The data is allocated starting at address `0x0000 0C10`.

This is the data portion of the program:

```
.data
0000 0C10 K:    .word     4
0000 0C14 R:    .word     0
0000 0C18 L:    .word     5,8,7,6,5,4,3,2,5,0
0000 0C40 S:    .word     3
```

This is part of the text portion of the program:

```
LDR  R1,=L      @ what is the content of R1? _____
LDR  R2,=S      @ what is the content of R2? _____
LDR  R2,[R2]    @ what is the content of R2? _____
LDR  R3,=R
LDR  R3,[R3]
LDR  R4,=K      @ what is the content of R4? _____
LDR  R4,[R4]    @ what is the content of R4? _____
LDR  R6,[R1]    @ what is the content of R6? _____
LDR  R5,[R1,R3]       what is the content of R5? _____
LDR  R5,[R1,R2,LSL #2]   @ what is the content of R5? _____
MOV  R5,#10
STR  R5,[R1,R6,LSL #2]
@ list now the (new?) content of memory starting at the address given by
@ the label L: __ __ __ __ __ __ __ __ __ __
```

Question 5. [2] You are given that the data portion of an ARM program contains:

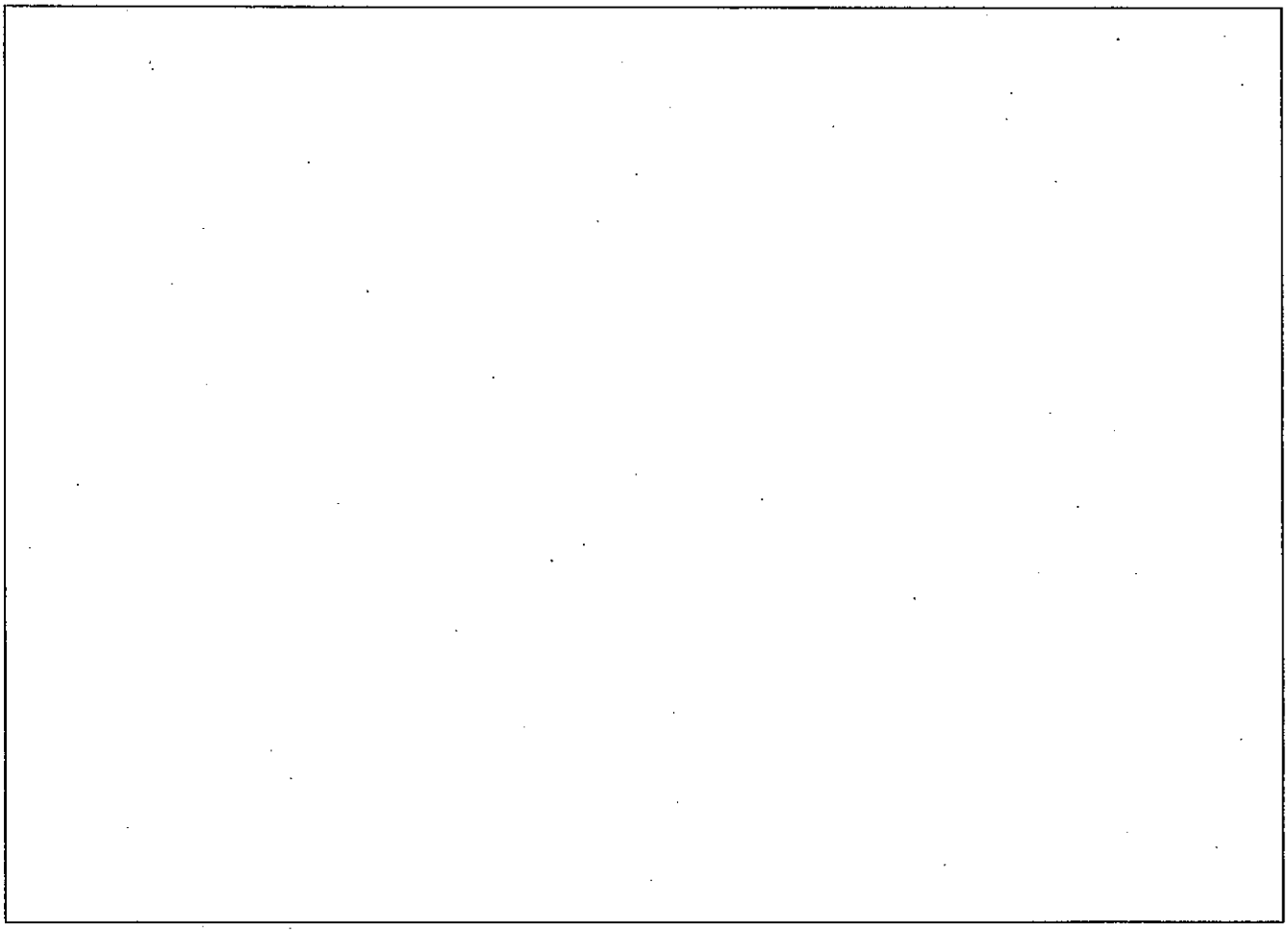
```
MyList:    .skip     100
Temp:      .skip     4
K:         .skip     4
```

- All the variables declared above have been used already and they contain some values.
- `R1` contains the current value of variable "`K`";
- `R2` contains the address of `MyList` which is an array of integers.
- `R3` contains the current value of "`Temp`".

State the correct *one instruction* in ARM equivalent to:

```
MyList[K] = Temp;
```

Question 6. [6] A system has Virtual Memory, L1 cache divided into Instruction and Data Cache, MMU, Page Table and TLB. Explain briefly the sequence of steps occurring for the event when the required data is in RAM but has not been used recently, in precise words (point form is suggested), or using a flowchart, or some sort of diagram.



Question 7. [5] It is always good to learn something new, even during a test. You have been programming in ARM assembly language calling function and passing input parameters in registers. If you remember, it was briefly discussed in the lectures that a different strategy is to pass parameters on the stack (jn fact, it is automatically done by a compiler when the number of parameters exceeds a fixed number). The question below uses parameter passing through the stack, placed there by the caller and retrieved by the called function. Moreover you will also learn that local variables are allocated in memory on the stack, such that thay can be easily deallocated at the exit of the function. Follow the logic below with the explanation of some new concepts and answer the questions.

[2] The following ARM code is used to call the function `Foo` with three input parameters passed on the stack by the caller.

```
     STR       R2,[SP,#-4]!   @1st parameter on stack
     STMFD     SP!,{R3-R4}    @2nd and 3rd parameters
     BL        Foo            @Call int Foo(R2,R4,R3)
Next: .  .  .  .
```

The following instructions are found at the entry of function Foo. Show the contents of the stack and the positions of the pointers SP and FP after the execution of the 2nd ADD instruction. Use figure 2 on the right side. The label "FP" refers to the "frame pointer" which keeps track of the stack frame for this particular call only (FP is register R12 in ARM).

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

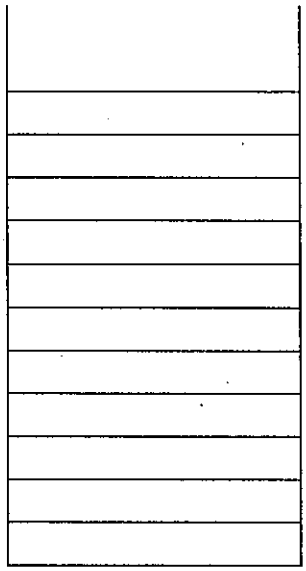


FIGURE 2. Stack to be filled in

[3] Give the three instructions required to copy the 3 parameters from the stack and place them in R5,R6,R7. Note: you are simply "reading" (loading) from the stack, that is, your base register is SP instead of any other register containing an address.

Question 8. [3] For a system with two levels of cache, define the following:

T_{C1} = first-level cache access time; H_1 = first-level cache hit ratio;

T_{C2} = second-level cache access time; H_2 = second level cache hit ratio (i.e. miss on C1, hit on C2).

T_M = memory access time;

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

Question 9. [4] The example below shows a sequence of 5 instructions in a 4 stage pipeline.

(a) [1] Which time slots are the most efficient/effective in terms of CPU usage?

Time ⟹

	clock cycles					
Instructions	1	2	3	4	5	6
I1	F1	D1	E1	W1		
I2		F2	D2	E2	W2	
I3			F3	D3	E3	W3
I4				F4	D4	E4
I5					F5	D5

(b) [1] What sort of pipeline hazard is in the following code? Name it only, no explanation.

```
[1]   MUL   r1,r2,r3
[2]   LDR    r1,[r1]
```

(c) [1] What sort of pipeline hazard is in the following code? Name it only, no explanation.

```
      CMP   r1,#5
      BEQ   B1    @branch taken!
      STR   r1,[r5]
B1:   STR   r1,[r4]
```

(d) [1] What is the penalty incurred from a branch instruction in a system using a pipeline with m stages executing N instructions?

Question 10. [7] **(a)** [1] In the instruction "`MOV R1,#12`", register `R1` gets the number 12 through the datapath. From which unit of the processor does the number 12 come?

[ ] → datapath → R1

(b) [2] Given the instruction: `LDR R2,[R5]`, fill in the missing parts from the four steps in the execute phase below.

[i] R5 → datapath → [ ] → [ ]

[ii] Read signal → Control line

[iii] Wait for memory to respond

[iv] [ ] → [ ] → datapath → R2

(c) [2] What are the addressing modes of each of the operands in the instruction below?

```
LDR  R1,[R2],R3
```

Addressing mode of operand 1:	
Addressing mode of operand 2:	

(d) [2] What does the following instruction do?

```
STR  R1,[R2,R3,LSL #2]!
```

Question 11. [2]

(a) [1] In a system which has a 33-bit address space with byte addressable memory and pages that are 512KB in size, how many entries does the page table have?

(b) [1] Given the page table below and a virtual address where the first two nibbles[1] are the page number, what is the physical address for the virtual address 011234?

0	08
1	77
2	-
3	14

Question 12. [7] **(a)** [1] State the different types of mapping functions available for a cache organization (state only, no definition or explanation of them needed).

(b) [1] State (no explanation) the different types of Write Policies available for cache organization.

1. nibble = 1 Hex digit = 4 bits

(c) [1] Where is a level 1 cache usually located?

(d) [3] What is the MMU? That is, first explain what the acronym means; secondly describe the function of the MMU; and thirdly state where the MMU is located.

(e) [1] What does "split cache" imply?

Question 13. [6] You are given the following specifications for a cache:

- four-way set associative;
- line size of two 16-bit words;
- able to accommodate a total of 4K 16-bit words from main memory;
- used with a 16-bit processor which issues 24-bit addresses.

(a) [1] What is the total size of the cache in bytes?	
(b) [1] How many lines/slots does the cache have?	
(c) [1] How many sets does the cache have?	
(d) [1] How many lines/slots are in each set?	
(e) [2] Given a memory address, how would the cache line position be computed? State the function steps, using the numbers from above and the example address 0xCDE8F8, but you do not need to compute the final answer - just show how to get it.	

Question 14. [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 15. [5] Each of the following five situation is described by two statements. It is the case that either one of them is true and the other false, or both are true. State what is the case for each by ticking with "X" only one box per statement.

	Statements	True	False	Both True
1	Demand paging is when each page of a process is brought into memory from disk only when it is needed.			
	Demand paging is the process that causes a page fault.			
2	When the OS brings one page into memory, it must throw another page out (page replacement). If it throws out a page just before it is about to be used, then it will just have to go and get that page again almost immediately. Too much of this is a condition known as *thrashing*.			
	When the OS brings one page into memory, it must throw another page out (page replacement). If it throws out a page just before it is about to be used, then it will just have to go and get that page again almost immediately. Too much of this is a condition known as *page replacement hazard*.			
3	A TLB is the cache for the Page Table.			
	A TLB is a subset of the Page Table.			
4	The TLB is normally in the MMU while the Page Table is normally in RAM.			
	The TLB is normally in cache while the Page Table is normally in RAM.			
5	There is a separate Page Table for each program, but only one common TLB.			
	There is a separate Page Table for each program, and a separate TLB for each program.			

Question 16. [8]

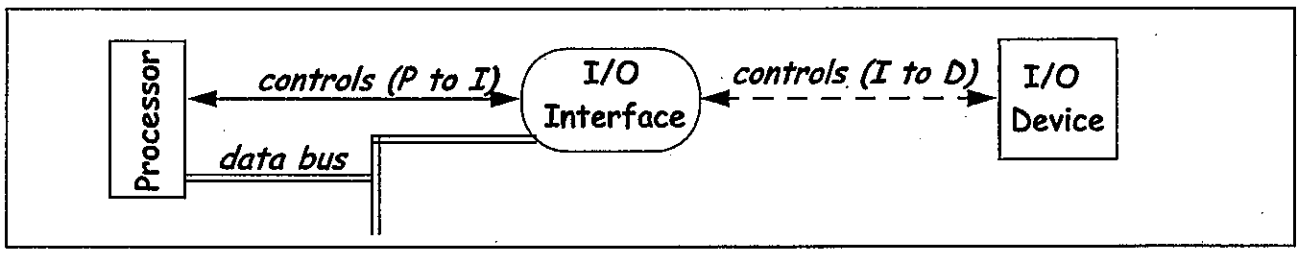
(a) [5] Number the following steps from 1 to 5 in the order they are performed in processing a general interrupt sequence using the interrupt jump table technique	
	Recognize the interrupt event and set the event flag
	Load the PC with the address from the interrupt vector table
	Execute the first instruction of the interrupt handling routine
	Push the processor registers onto the stack
	Determine the interrupt vector number

(b) [2] Give an example of *internal* and *external* interrupt/exception.

Internal (exception)	
External (interrupt)	

(c) [1] After an exception or interrupt has been processed, it may be the case that normal processing can resume within the application. Give one example in which this is not the case.	

Question 17. [8] The figure below shows some of the connections between a processor and one I/O device, using the bus and the control lines. Two sets of control connections are shown: (a) the "controls (P to I)" which represent the connections between the processor and the interface; (b) the "controls (I to D)" which represent the connections between the interface and the device itself. List/state the possible protocols which can exists for each set of connections: state the choices and what they are called. No definitions are required.



(A) Controls (P to I)

(B) Controls (I to D)

Question 18. You are to implement the following computation both in C and ARM:

$$R = \sum_{i=0}^{n} \prod_{j=1}^{k} (2i + 3j)$$

where n and k have been already assigned elsewhere and are available as variables. You must include excellent documentation. You need not state all headers - *just the code segment*. You are free to declare extra variables you might need. In case you are unfamiliar with the notation, Π means the product of the expressions generated when j is assigned in turn the values from *1* to k, while Σ means the sum of the resulting expressions when i is assigned in turn the values from *0* to n. For example, for n=1 and k=2, we have:

$$R = \sum_{i=0}^{1} \prod_{j=1}^{2} (2i + 3j)$$

$$= (\{[(2 \times 0) + (3 \times 1)] \times [(2 \times 0) + (3 \times 2)]\} + \{[(2 \times 1) + (3 \times 1)] \times [(2 \times 1) + (3 \times 2)]\})$$

$j = 1$ $i = 0$ | $j = 2$ $i = 0$ | $j = 1$ $i = 1$ | $j = 2$ $i = 1$

$i = 0$ | $i = 1$

(a) Write the code using C, with appropriate documentation.

```
int FIN14(int n, int k){
   int i, j;









}
```

(b) Now provide the correct ARM code with documentation on the next page.

```
@ Register usage and assumptions:
@     R0 for return parameter R
@     R1 for input parameter n
@     R2 for input parameter k
FIN14:     STMFD     SP!,{R1-R8,LR}       @save registers (you may not
                                          @ need all of them)




ENDFIN14: LDMFD      SP!,{R1-R8,PC}       @return to caller
```

----------THE 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, [Rn]	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