UNIVERSITY OF VICTORIA

EXAMINATIONS FALL 2012

C SC 230 - Introduction to Computer Architecture and Asembly Language - A01 CRN# 12749 and A02 CRN# 12750

STUDENT NUMBER: _____

TIME: 3 hours

INSTRUCTOR: M. Serra

TOTAL MARKS: 102

TO BE ANSWERED ON THE PAPER

Question No.	Value	Mark	Question No.	Value	Mark
1	6		11	2	
2	6		12	4	
3	5		13	4	
4	4		14	4	
5	4		15	4	
6	4		16	6	
7	6		17	8	
8	4		18	8	
9	4		19	13	
10	6		TOTAL	102	

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. [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 Mode of Source Operand	Bus used ?	What it does (be precise)
TT r1,#1	register direct	none	test if r1 is 0 or negative
LR r1,[r2,r3,lsl #2]!	register direct data direct	control data address	loads register
MV r1,r2,asr #4	register direct	control data address	copies r2 into r1

Question 2. [6] You have a system with Virtual memory, a DMA, a Page Table, a TLB, RAM memory, disks, and two levels of cache L1 and L2, with L1 being further subdivided into an L1 Data cache and an L1 Instruction cache.

(a) [3] The CPU wants to access some data and the data is in L2, but not in L1. Describe the series of events precisely and concisely.

- request is sent to L1 and TLB in parallel
- data not in L1 so L2 is checked
- L2 returns the data

b [3] Repeat, assuming the data is in RAM memory and not in the L1 or L2 caches.

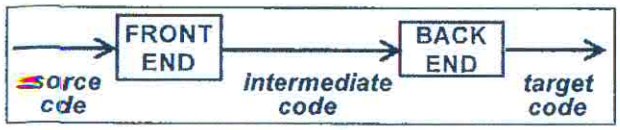
- request sent to L1 and TLB.
- data not in L1 so L2 checked
- at same time Page table checked
- data not in L2
- data found in main memory or disk using Page table
- data returned

Question 3. [5] (a) [4] The top portion of the Flynn taxonomy is shown below. State the *definitions only* for the four categories.

Flynn Taxonomy for Processors Organization

SISD *SIMD* *MISD* *MIMD*

(b)1] Describe the differences between SIMD and MIMD *briefly*.



Question 4. [4] (a) [3] 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 phases are in the front end, that is, when the high level source code is translated into intermediate code. In an interview situation, you should not take more than 2 minutes; be similarly concise and precise.

(b) [1] In the explanation above, somewhere, the notion of a *Symbol Table* should have surfaced. Describe in more depth what the function of the Symbol Table is within the process.

Question 5. [4] Dynamic linking is an important concept and the differences with static linking have been discussed. Draw *two* diagrams to summarize the elements of the processes of dynamic and static linking. Diagrams only with the elements involved are required here, not a complete explanation of how dynamic linking works.

Question 6. [4] (a) [3] An interrupt signal is sent to the processor from a device interface. Summarize the possible series of events which follow by drawing a flowchart or a process chart or some sort of clear and precise diagram. For each step you should not write a long detailed explanation. However you should attach a short label/definition to each step that makes it clear that you know and understand what is supposed to happen in the interrupt process sequence of steps.

(b) [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 7. [6] The following initialization loop is included in the C code that solves a vector processing algorithm:

```
for (i=0; i++; i<50)
     V[i] = -85 *i;
```

You are in the process of converting this algorithm to ARM assembly language. In the data section, 50 words have been reserved for the vector V and, in the code section, the address of V has been loaded into R0 (as shown below). This code is in the middle of a complex routine that uses all registers except R7 and R9. Using these two available registers and no more than six ARM instructions, write the initialization loop. (Six instructions is optimal, however answers with more instructions are also accepted with minimal penalty).

```
    @ === Other code comes before this line
    ldr R0, =V          @R0 = Address of vector V
    ldr R7, 0
loop: multi R9, R7, -85
    st R0, R9
    Pinc R0
    inc R7
    cpi R7, 50
    brlt loop

                                  @ for (i=0; i++; i<50)
                                  @     V[i] = -85 *i;
                                  @ WRITE YOUR ANSWER HERE

    @ === Other code comes after this line
    .data
V:  .skip 200
    .end
```

Question 8. [4] Fill in the right column of the table with your answers.

The range of decimal values that can be represented as *unsigned* 16 bit integers is (answer in decimal):	(2^{15} , $2^{16}-1$)
Given 8 binary bits, the largest positive integer that can be represented using a 2's complement representation is (answer in decimal):	$2^8-1 = 255$
Given 16 binary bits, the largest negative integer that can be represented using a signed magnitude representation is:	-2^{15}
The hexadecimal value FFE corresponds to the 12-bit binary:	1111 1111 1110
The hexadecimal value FFE viewed as a 12-bit integer in a 2's complement context, corresponds to the decimal:	-2
Convert the decimal integer -23 to 8-bit binary in 2's complement	11101001
Convert the decimal integer -23 to 3-digit hexadecimal in 2's complement	FE9
The 2-digit hexadecimal quantity AA is even and negative (TRUE/FALSE/NEITHER)	True both even and negative

5. 1111 1111 1110 0000 0000 0001 +1 0000 0000 0010 = -2

6. 23 0001 0111 1110 1000 +1 1110 1001

7. 1111 1110 1001 F E 9

8. 1010 1010 match 0101 0101 +1 0101 0110

$64kB = 2^6 \cdot 2^{10} = 2^{16}$ bytes

total $2^{16} + 2^4 + 2^4 + 2^4 + 2^4 = 65600$

Question 9. [4] A computer system has a RAM containing 64K bytes, each of which needs its own distinct address. Moreover it has 4 peripherals and they each require 2^4 distinct addresses in order to interface properly.

(a) How many distinct addresses in total are necessary in this system? Write the total number in the centre of the expression below on the left. (Feel free to leave it as sum of powers).

$2^{\boxed{16}} < \underline{65600} < 2^{\boxed{17}}$

total number of addresses here

(b) Place the number of addresses just computed between the appropriate powers of two in the expression on the left, by writing the correct exponent in the boxes (for example, if the result were 18, you would have $2^4 < 18 < 2^5$ by writing the exponents 4 and 5.

(c) How many lines does an address bus for this system require, given that it must be able to carry all the needed values for the addresses? 17

Question 10. [6] Consider two different machines, with two different instruction sets, and with the same clock rate of 200MHz.[1] The following measurements are recorded on the two machines running a given set of benchmark programs:

Instruction type	Instruction count (millions)	Cycles per instruction
Machine A		
Arithmetic and logic	8	1
Load and store	4	3
Branch	4	2
Others	4	3
Machine B		
Arithmetic and logic	10	1
Load and store	8	2
Branch	2	4
Others	4	3

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

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

b [2] Compute the execution time in ns of the benchmarks for each machine (show your work).

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 each machine above (at least show the complete equation for the calculation, even if you cannot handle the computation).

Question 11. [2] While browsing at a computer store, you overhear a customer asking a salesperson what is the fastest computer in the store that can be purchased. The employee replies: "You are standing and looking at the moment at our Macintoshes. The fastest Mac here runs at a clock speed of 1.2 GHz. If you really want the fastest machine, you should buy our 2.4 GHz Intel Pentium IV instead." Is the salesperson correct? What would you say to help the customer?

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.

$T = H_1 \times T_{C1} + (H_2 - H_1) \times T_{C2} + (1 - H_2) \times T_M$

Question 13. [4] When evaluating the performance of a pipeline, we calculate the possible speedup as the ratio:

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

An old processor (very similar to the Intel 8088) consists of a bus interface unit (BIU) and an execution unit (EU), which form a 2-stage pipeline. The BIU fetches instructions into a 4-byte instruction queue. The BIU also participates in address calculation, fetches operands, and writes results in memory as requested by the EU. If no such requests are outstanding and the bus is free, the BIU fills any vacancies in the instruction queue. When the EU completes execution of an instruction, it passes any results to the BIU (destined for memory or I/O) and proceeds to the next instruction.

(a) [2] Suppose the tasks performed by the BIU and the EU take about equal time. By what factor does pipelining improve the performance of this processor? Ignore any consideration of branch instructions. Show your work.

It fetches the next instruction while the EU processes the current instruction

speed up = 2

(b) [2] Repeat the evaluation assuming that the EU takes twice as long as the BIU. Show your work.

Question 14. [4] 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 then 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 0.95 and the main memory hit ratio is 0.9. What is the average time in ns required to access a referenced word on this system (show your work in detail)?[1]

$T_{avg} = 0.95 \times 20ns + 0.9 \times [\underbrace{60ns}_{\text{copy to cache}} + \underbrace{20ns}_{\text{reference}}] + (1 - 0.9 \times 0.95)[\underbrace{12ms}_{\text{fetch from disk}} + \underbrace{60ns}_{\text{copy to cache}} + \underbrace{20ns}_{\text{re-reference}}]$

Question 15. [4] **(a)** [1] Describe briefly the main function of a DMA.

(b) [1] State what peripheral interfaces are.

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

(c) [2] Explain "polling" as a synchronization method for communication between a processor and peripherals through their interfaces.

Question 16. [6] Make comparisons between caches and virtual memory with respect to: purpose, unit of data transfer, and method of implementation:

	Cache	Virtual Memory
purpose	speed	expand memory
unit of data transfer	line / block	page
method of implementation: hardware or software or both	hardware	both

Question 17. [8] Consider a memory of 64K words (= 2^{16} words), addressable by 16-bit addresses, to be viewed as 4K blocks (= 2^{12} blocks) of 16 words each (2^{12} blocks x 2^4 words = 2^{16} words). Consider also a cache consisting of 128 blocks of 16 words each, for a total of 2048 (2K) words. Assume that a direct mapping configuration is used for the cache organization, as shown in one part of figure 1.

(a) [1] Describe precisely how and where an arbitrary 16 word block *K* from memory is loaded into the cache by summarizing the general strategy.

block k is loaded into address k modulo # of blocks in cache
address k = k % 128

(b) [3] Show that you understand the strategy by giving the cache block numbers assigned to the following *sequence* of blocks fetched from memory – the blocks are fetched in the order as written from left to right (there might be collisions).

Memory Block #	0	1	65	200	202	128	129
Cache Block #	0	1	65	72	74	0	1

(c) [1] Assume now that a set associative mapping organization is used for the same cache, seen as 32 sets of 4 blocks each. Describe precisely how the mapping is configured differently from direct mapping, again looking also at the left part of figure 1.

cache set # = k % # of sets (32 in this case)
cache block # = which ever one is open
with set associative strategy the k is for sure in one set but can be in any block in that set

(d) [3] Repeat part (b) from above assuming set associative mapping for the same cache, seen as 32 sets of 4 blocks each. Give the cache set and block numbers assigned to the following *sequence* of blocks fetched from memory – the blocks are fetched in the order as written from left to right (there might be collisions).

Memory Block #	0	1	65	200	202	128	129
Cache Set #	0	1	65	72	74	0	1
Cache Block #	0	0	0	0	0	1	1

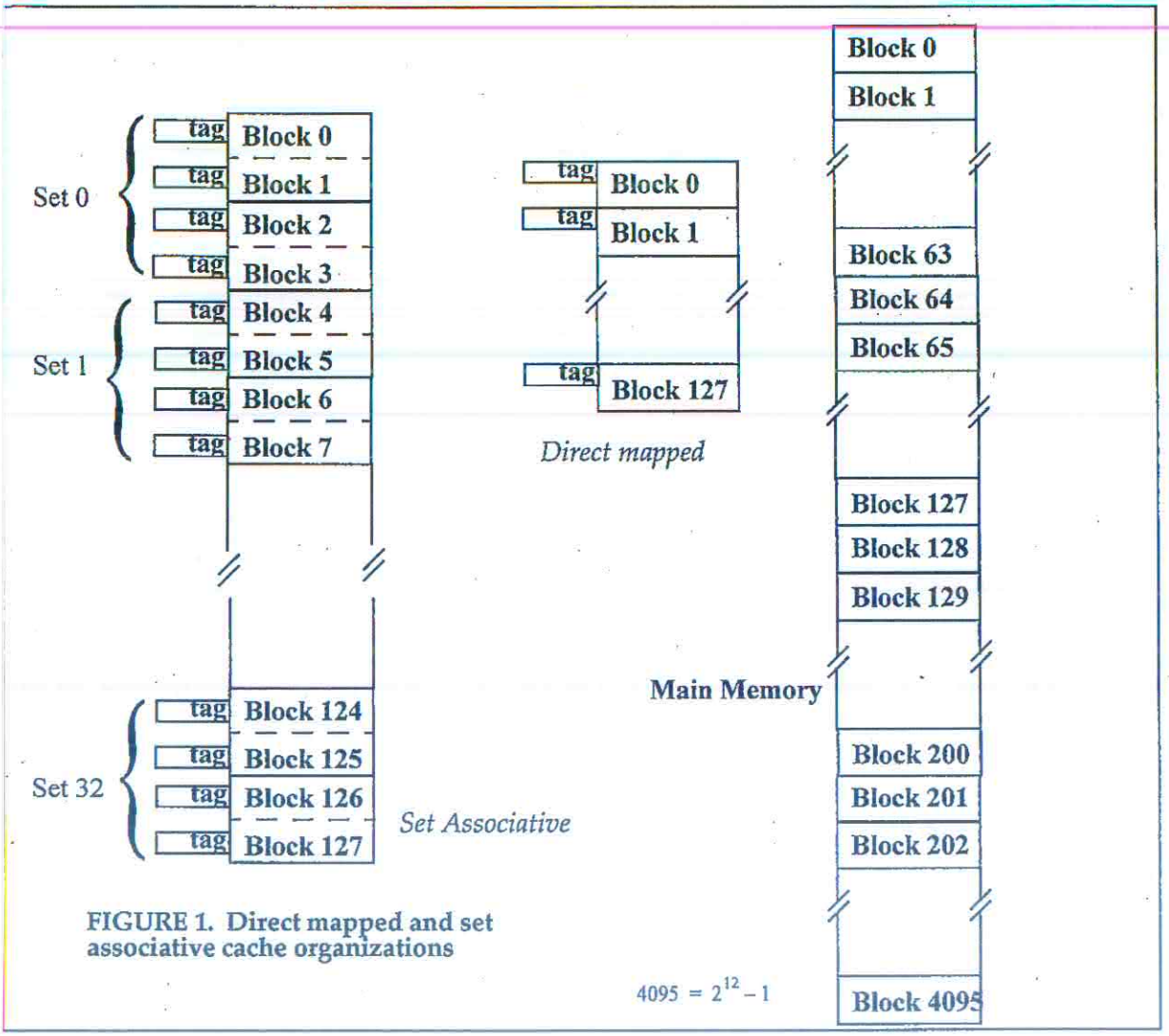
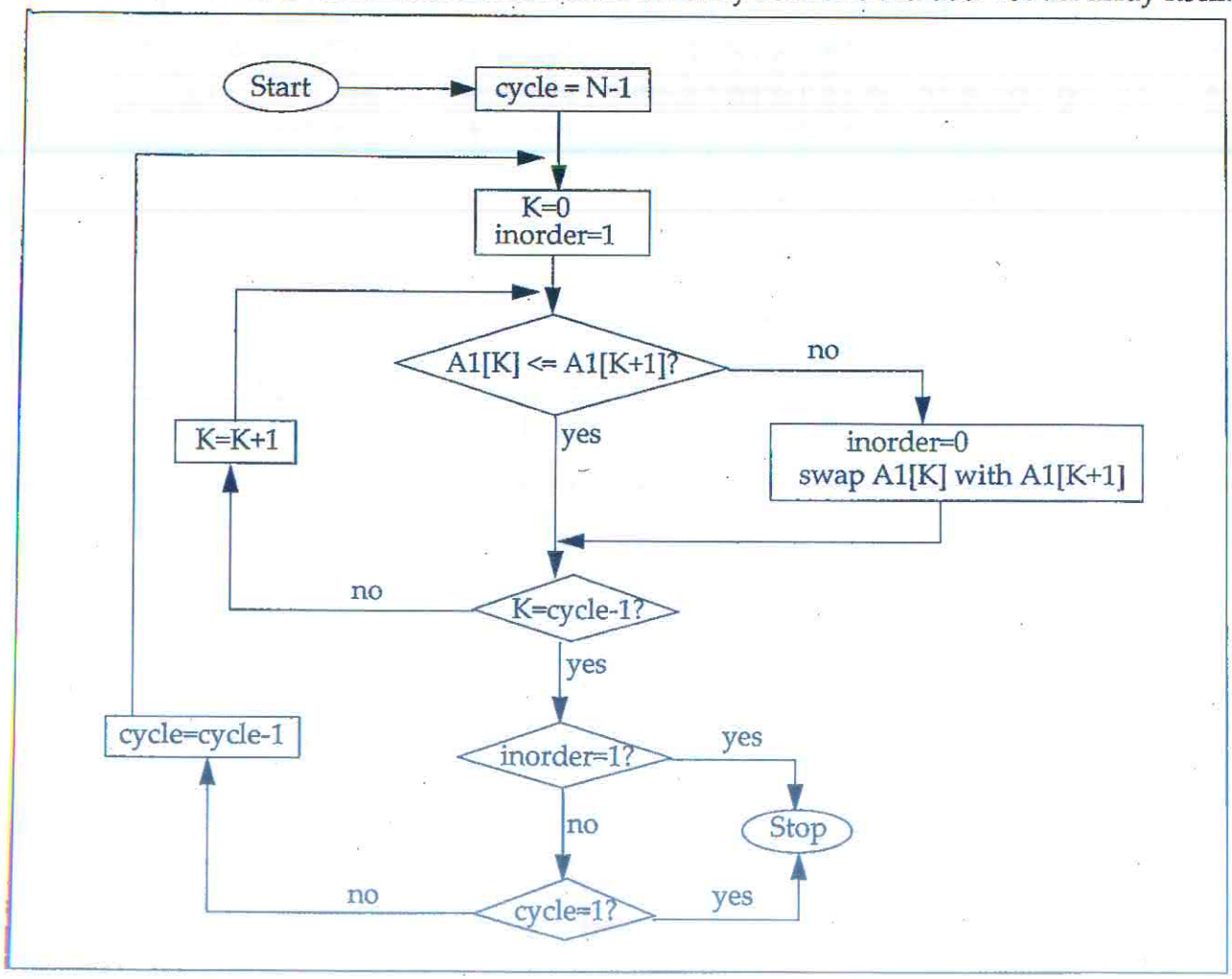


FIGURE 1. Direct mapped and set associative cache organizations

Question 18. [8] Give brief definition for the following concepts related to virtual memory:

Concept/Term	Definition
Page	a continuous block of virtual memory (several lines of memory together)
Page fault	when a page is not found in the page table
TLB (Translation Lookaside Buffer)	checks cache for recent address translations
MMU (Memory Management Unit)	manages the reading and writing from the main memory to disk

Question 19. [13] Write a procedure to sort an array of 32-bit integers using the bubble sort method. The array is called `ARR1` (declared by main in `.DATA`) and its size is stored in `ARCNT`. Assume that the main routine calls an external function `INIT` which, given the address of the array, returns it filled with data and with its current size in `R0`. The main program passes the address of `ARR` and its size from `ARCNT` as parameters to the procedure `BubbleSort` in `R1` and `R0` respectively. You are supposed to write the whole program. The flowchart for the algorithm of `BubbleSort` is given below. Test it manually first on a small example to understand well what you are supposed to be coding. In the flowchart, `N` denotes the number of elements in the array and `A1` is the label for the array itself.[1]



1. Should you think there is any problem with the algorithm and/or the flowchart, disregard for the moment and just proceed to code as shown.

CSC230

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

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