TUTORIAL #2 Addressing Modes & Assembly Programming (CE/CZ1006)

2.1 Addressing Modes and Their Characteristics

(To be done over 2 week)

Table 2.1 shows the mnemonics of several consecutive VIP instructions.

(1) For each mnemonic, give the addressing modes for the source and destination operands.

(2) Complete Table 2.1 by giving the instruction lengths (in number of 12-bit words) and the number of cycles required to execute each of the instruction.

Hint: Use the VIPAS (Basic) simulator (available on the Edventure site) and the *Tutorial 2_1.vasprj* project file to help you figure out the instruction length and the cycles required to execute each instruction.

No.	Mnemonics	Instruction length (in words)	Cycle Count
1.	`MOV R0,R1`		
2.	`MOV R0,[0x100]`		
3.	`MOV R0,#0x100`		
4.	`MOV R1,#0x100`		
5.	`MOV R0,[R1]`		
6.	`MOV R0,[R2+7]`		

Table 2.1 – Instruction length and number of execution cycles of several VIP instructions

(3) The initial 12-bit hexadecimal contents in registers **R1**, **R2**, **SR** and data memory location **0x100** are all **0x000**. Answer the following questions with reference to Table 2.1:

(a) Given that the initial content in register **R0** is **0xFFF**, what register(s) in the VIP processor will be modified by the execution of **MOV R0,R1**.

(b) (i) Will the execution of instructions **MOV R0,[0x100]** and **MOV R0,#0x100** load different contents into the register **R0**?

(ii) What are the differences in source addressing mode between these two instructions?

(iii) Can you tell what will be the result in **R0** by just examining the machine code of each of these instructions?

(c) Consider the situation where instructions 1 to 5 are executed one after another starting with instruction 1.

(i) Would the execution of the instruction **MOV R0,[0x100]** and **MOV R0,[R1]** produce the same results in **R0**?

(ii) If the content of memory location **0x100** has to be repeatedly loaded into register **R0** during the execution of your program, which would be the preferred instruction to use? Give a reason for your choice.

(iii) Suggest an alternative single instruction to replace instruction 4 (i.e. **MOV R1,#0x100**). In what way is this instruction better than the original one?

(d) Assume instructions 1 to 6 are consecutive instructions in memory and the first instruction is found in code memory location 0x000.

(i) What do you think is the content in the register **R0** when the instruction **MOV R0,[R2+7]** is executed?

Note: You may assume that the VIP processor has a Von Neumann architecture.

2.2 Addressing Modes and Data Movement

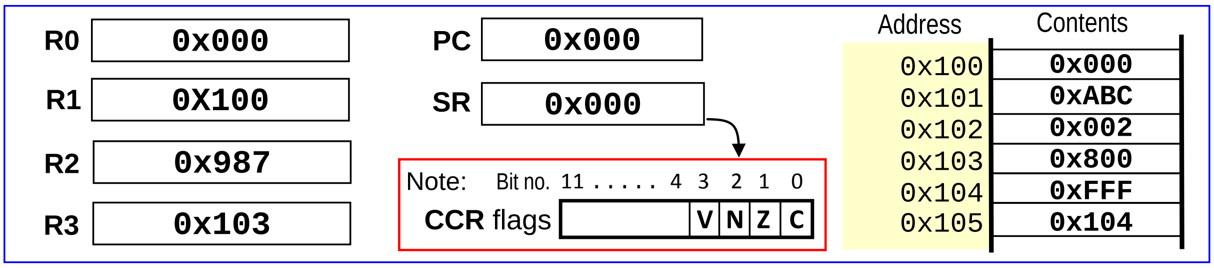

Figure 2.2 – Initial state of some VIP processor registers and memory contents

Give the 12-bit hexadecimal contents of **any registers** (i.e. general registers, **PC** and **SR**) and **memory** locations that are altered by the execution of each of the VIP instructions given below.

Note:

- Assume the initial processor state just before the execution of **each of the non-consecutive instruction** is shown in Figure 2.2.
- Highlight if any of the VIP mnemonics below are illegal.

(1) **`MOV R0,R2`** (2) **`MOV R1,[R1]`**

(3) **`MOV #0x000,R1`** (4) **`MOV [0x102],[0x103]`**

(5) **`MOV R0,[R3+0xFFE]`** (6) **`MOV R1,0x100`**

(7) **`MOV SR,#0x001`**

2.3 Arithmetic and Logical Instructions

(1) Give **five** different ways to **clear** register **`R0`** to **`0x000`** using only a single VIP mnemonic. Discuss the possible advantages and disadvantages between these different alternatives.

(2) Give **three** different ways to **add the value of 1** to the register **`R1`** using VIP mnemonics. Discuss the possible advantages and disadvantages between these different alternatives. Note: You may use more than one instruction to perform this operation.

(3) You are required to complement bits 0, 4 and 8 in register **`R2`** while leaving all other bits unchanged. For example if **`R2`**=**`0xFFF`**, after execution, **`R2`**=**`0xEEE`**. Give a single VIP mnemonic that will allow you to perform this operation.

(4) With the aid of one of the Rotate instructions, write a VIP assembly code segment that will multiply the signed content in register **`R0`** by the constant value 3.

   (i) Give the largest positive signed 12-bit value in **`R0`** that will not result in an overflow after this operation.

   (ii) Suggest how this idea could be extended to multiply the content in register **`R0`** with the constant value 7.

2.4 Assembly Program Analysis – Comparison and Counting

Fig. 2.4 shows a VIP assembly language program that scans through a sample of average daily temperature readings stored as unsigned 12-bit numbers in data memory starting at address **`0x102`** onwards. Two memory variables **`HotDays`** and **`DaySum`** are located at addresses **`0x100`** and **`0x101`** respectively. The end of valid temperature readings in the array is detected when the decimal value of -128 is encountered.

Note: You may want to use the template *Tutorial 2_4.vasprj* to help you analyse the operation of the given assembly language program. All memory values as shown in Figure 2.4 has been pre-loaded into their respective memory locations in this VIPAS project file.

```
; For simplicity of analysis, the Jump
; instructions have used address labels.
; If you want to simulate this code on
; VIPAS, use the relative offset values
; given in the associated comments.

Start:      MOV   R1,#0x102    ; (a)
            MOV   R0,#0
            MOV   [0x100],R0
Loop:       MOV   R3,[R1]
            CMP   R3,#0xF80
            JEQ   Done
            ADD   R0,#1
            INC   R1
            CMP   R3,#36
            JHS   HotFound
            JMP   Loop
HotFound:   MOV   R2,[0x100]
            ADD   R2,#1
            MOV   [0x100],R2
            JMP   Loop
Done:       MOV   [0x101],R0   ; (b)
```

Address	Contents
0x100	**0x000**
0x101	**0x000**
0x102	**0x024**
0x103	**0x032**
0x104	**0x017**
0x105	**0x023**
0x106	**0x028**
0x107	**0x022**
0x108	**0x013**
0x109	**0xF80**
0x10A	**0x020**
0x10B	**0x017**
0x10C	**0xF80**
0x10D	**0x025**
0x10E	**0x014**
0x10F	**0x02D**

Memory map of data area

Figure 2.4 – A VIP assembly language program and some contents in memory

(1) Based on the hexadecimal memory contents shown in Figure 2.4, compute the hexadecimal contents in registers **R0**, **R1**, **R2**, **R3** and word-sized memory variables **`HotDays`** and **`DaySum`** at the end of program execution. Assume execution begins at instruction (a) and ends after instruction (b).

(2) Describe briefly what the assembly language program in Figure 2.4 does?

(3) Suggest modifications to the program in Figure 2.4 that can help speed up its execution.
Note: Should you want to use the VIPAS simulator to evaluate your modifications, be careful when you make changes to the instruction in the program. The jump offsets indicated in the comments column will no longer be valid due to the change in size of the modified instructions.

(4) It was observed that if temperature samples had negative values (e.g. -1 or **`0xFFF`**), the results obtained were incorrect. Could you suggest a reason for this error and how it can be rectified in the program shown in Figure 2.4?

Not necessary to be covered during tutorial classes.

The following questions are for your self-practice and may not be covered during tutorial classes. Suggested solutions to these problems will be uploaded to the Edventure site about a week after the tutorial.

2.5 Program Counter related Addressing Modes

Figure 2.5 shows a VIP assembly program and the starting address of various incomplete instructions in code memory. Complete the mnemonics M1 to M4 based on their respective comments.

Note: Except for instruction M1, where absolute addressing is used, you should give solutions for M2 to M4 that support **position-independent code**.

Address	Mnemonics or Hex Data	Comments
:		; Other parts of the program
0x020	MOV PC, ?	; M1 - Absolute Jump to instruction at address 0x030
:		; Other parts of the program
0x030	ADD R0, ?	; M2 - Add constant C1 to R0
:		; Other parts of the program
0x05D	MOV R0, ?	; M3 – Move start address of next instruction into R0.
0x05E	MOV ? , ?	; M4 - Create infinite loop to prevent execution beyond code space
0x060	0x005	; C1 - Data constant stored in code memory
:		

Figure 2.5 – Various VIP mnemonics and their respective start addresses in code memory

2.6 Assembly Programming Exercise – Computing Average

Write a VIP assembly language program to compute the average value of **eight** numbers in an array.

(1) Your program should be written based on the following specifications:

i) These numbers are word-sized unsigned integers (i.e. **12-bit unsigned** numbers).

ii) The eight numbers are stored in consecutive memory locations starting at address **`0x100`**.

iii) The resulting average (quotient) and remainder should be stored in register **`R0`** and **`R1`**, respectively.