**what is a register?**

Think a register as a variable which is implemented with a specific digital circuit inside the microprocessor. Each microprocessor has its own registers, with specific names and capacity. For developing an Assembly program and for exchanging data with the microprocessor, the available registers are used.

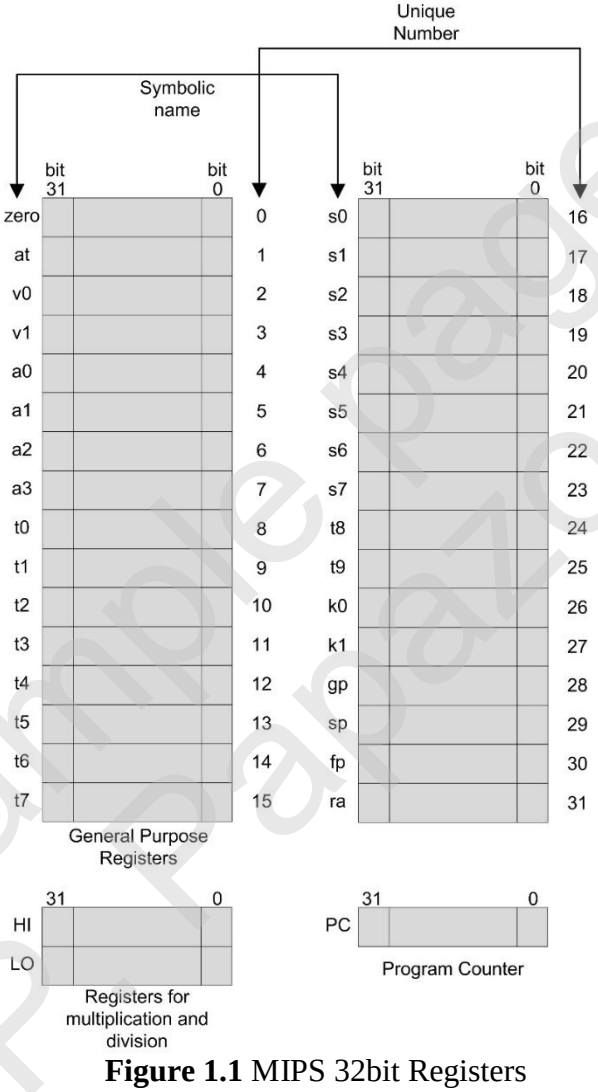



**Figure 1.1** MIPS 32bit Registers

For using registers 0 to 31, the corresponding unique number or symbolic name can be used. Practically, all the programmers prefer the symbolic names. Moreover, inside programs the prefix '$' is used on the left of a register number or symbolic name. As an example, the register v0 can be used by writing $v0 or $2.

Example 1

Assume that a program checks the content of $t1. If $t1=10, then the value 4 will be assigned to $t2, otherwise the value 3.

```
beq
$t1,10,timi4
li $t2,3
j L2
timi4:
li $t2,4
L2:
```

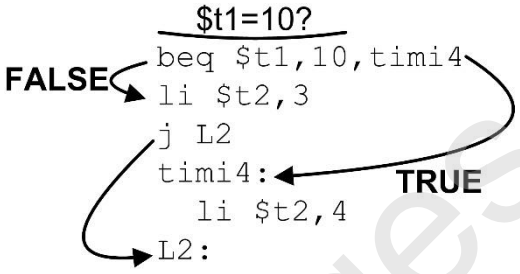



**Figure 1.9** Checking a register content and branching

As shown in figure 1.9, a comparison of `$t1` with 10 is performed. If the content of $t1 is equal to 10 (TRUE case), then a branch to label `timi4` is performed and the execution flow is continued from that point (the number 4 is assigned to `$t2`). Otherwise, (FALSE case), no branch is performed to label `timi4` and the execution flow is continued from the next instruction which assigns the number 3 to `$t2`. The instruction `j L2` drives directly the control flow to label `L2` in order to avoid the code section which starts from the label `timi4` and belongs to the TRUE case. Figure 1.10 shows the flow chart for the above code.

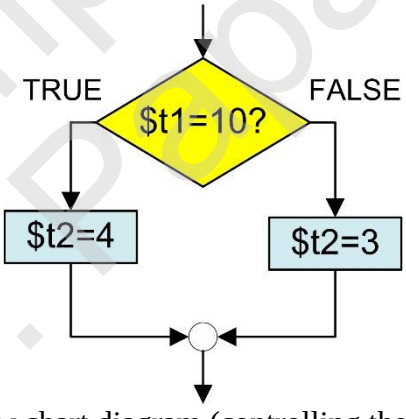



**Figure 1.10** Flow chart diagram (controlling the execution flow)

Example 2

A program checks if `$t1`=0 and `$t2`>=5. If this is TRUE, then the assignment `$t0`=1 is performed, otherwise nothing (execution flow continues from the

For A=1 to N (where N is the number of lines that have be chosen by the user)
    For B=1 to A (where B is the numbers of asterisks per line)

The following flow chart (figure 2.10) shows the implementation logic of the iteration loops as well as the logic of the asterisks display.

**Code development**

```
.text 0x00400000
```

❶

```
li $v0,4
la $a0,dose
syscall

li $v0,5
syscall
move $t1,$v0
```

Display the message "N=" and read a number (number of lines).

❷

```
li $t3,1
```

Initialize the line counter

```
again:
```

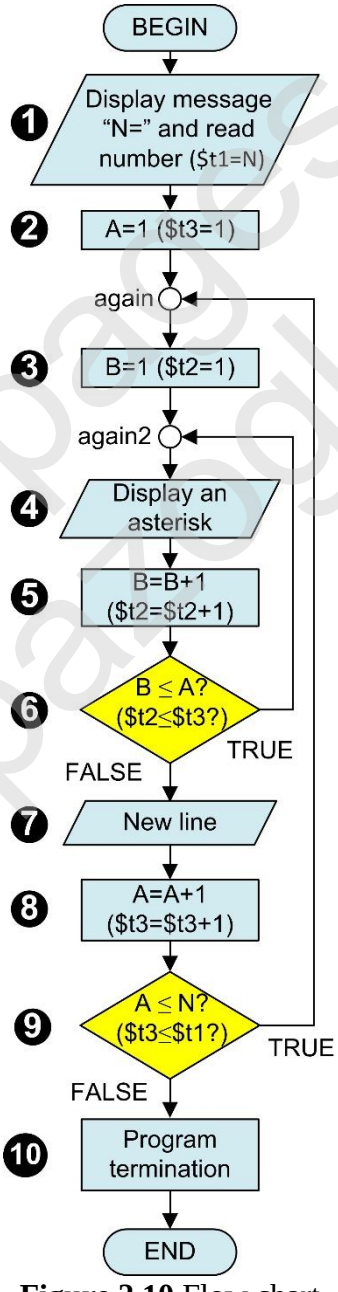



**Figure 2.10** Flow chart

```
$t1 = 02 88 FA 22
$t2 = 00 29 AD 66
```

The registers content within the memory will has the following form (figure 3.3):

| Address | Content | Deviation from the address arrayA | Register |
|---|---|---|---|
| arrayA | 02 | +0 | |
| | 88 | +1 | $t1 |
| | FA | +2 | |
| | 22 | +3 | |
| | 00 | +4 | |
| | 29 | +5 | $t2 |
| | AD | +6 | |
| | 66 | +7 | |
| | ... | +8 | |
| | ... | +9 | |

**Figure 3.3** Registers storage in memory

From figure 3.3 it is obvious that every storage of a new register starts always with a deviation of four memory locations. At Assembly level, the above storage (for two registers) is achieved with the following instructions:

```
sw $t1,arrayA(0)
sw $t2,arrayA(4)
```

The instruction `sw $t1,arrayA(0)` stores a word (word = 32bit) which is extracted from the register `$t1` starting from the address that is resulted from the addition `arrayA+0`. The label `arrayA` corresponds in a real address which represents the beginning of the memory area (array). Moreover, the instruction `sw $t2,arrayA(4)` stores the content of the register `$t2` starting from the address `arrayA+4`. The next register storage starts from the address `arrayA+8` and so on.

### Step 13

Modify the program of step 10 for displaying a help message during the filling of the array. This message will be displayed for every number that is entered by the user. For example, the help message for the two first numbers will be:

```
A[0]=
A[1]=
```

# Exercise 9

***Array management (A)***

### Step 1

Write the needed code for calculating the square of a positive number. Assume that the number is stored in $t1.

### Step 2

Develop a program for calculating the square of positive numbers of an array. The resulted square will be store in the same array location. The above operation can be described as follows:

```
A=array[i]
B=A*A
array[i]=B
```

### Step 3

Write a program for calculating and displaying the summation of the numbers that are stored in an array.

### Step 4

Write the code for checking a number if it is odd or even.

# 5 Selected MIPS Assembly Instructions

**Content-Goals**
In this chapter, the basic MIPS Assembly instructions will be presented as well as instruction lists with the corresponding description.

**Instructions and functions of the MIPS Assembly**

# add

**Operation**

Add the content of two registers or add a register with an integer value

**Expression**

```
add r1,r2,ri
```

`r1`,`r2` = register (register name)
`ri` = register (register name) or integer value

**Description**

r1 = r2 + ri

**Example**

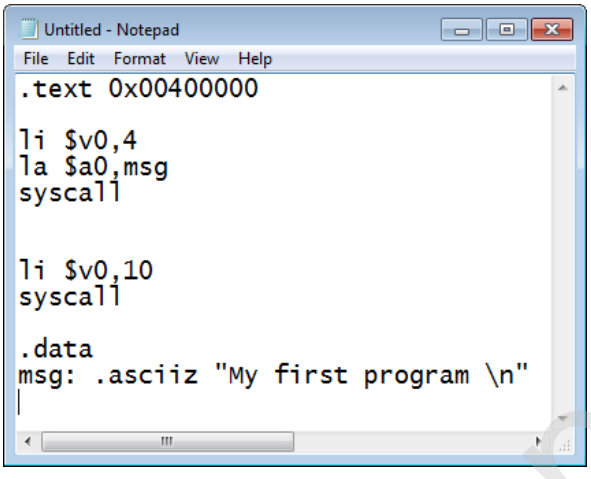
**#Adding registers**
```
add $t0,$t1,$t2 #$t0=$t1+$t2
```

**#Adding a register and an integer value**
```
add $t0,$t1,4 #$t0=$t1+4
```

**#Implementing the addition $t0=$t1+$t2+$t3**
```
add $t0,$t1,$t2
add $t0,$t0,$t3
```

# **Step 2** Enter the source code

Enter the source code within the text editor




# **Step 3** Create source file

Save the source code in a file with the extension .s.