**2.12 Assume that registers $s0 and $s1 hold the values 0x80000000 and 0xD0000000, respectively.**

**2.12.1 What is the value of the $t0 for the following assembly code?**

**Add $t0, $s0, $s1**

**Ans.** To perform the addition convert the hexadecimal values in $s0 and $s1 into binary.

$s0 = 0x80000000 = 1000 0000 0000 0000 0000 0000 0000 0000

$s1 = 0xD0000000 = 1101 0000 0000 0000 0000 0000 0000 0000

$t0 = $s0 + $s1

= 1000 0000 0000 0000 0000 0000 0000 0000

+ 1101 0000 0000 0000 0000 0000 0000 0000

$t0 = **1**0101 0000 0000 0000 0000 0000 0000 0000

Now converting the binary result stored $t0 to hexadecimal we get,

**$t0 = 0x50000000**. Since the register is of 32 bits, the 1 stored on the 33rd will be ignored because there has been an overflow.

**2.12.2 Is the result in $t0 the desired result or has been overflow?**

Ans. After adding the registers $s0 and $s1. The result obtained is 0x150000000 which is more than 32bits. As we have the 32 bits registers, we will not be able to store the result value of the 33rd bit. Hence **the result obtained after the addition is not the desired result and there has been overflow.**

**2.12.3 For the contents of the register $s0 and $s1 as specified above, what is the value of $t0 for the following assembly code?**

**Sub $t0, $s0, $s1**

**ANS.** To perform the addition convert the hexadecimal values in $s0 and $s1 into binary.

$s0 = 0x80000000 = 1000 0000 0000 0000 0000 0000 0000 0000

$s1 = 0xD0000000 = 1101 0000 0000 0000 0000 0000 0000 0000

Taking 2’s compliment of $s1 we get,

$s1 = 0011 0000 0000 0000 0000 0000 0000 0000

$t0 = $s0 + $s1

= 1000 0000 0000 0000 0000 0000 0000 0000

+ 0011 0000 0000 0000 0000 0000 0000 0000

1011 0000 0000 0000 0000 0000 0000 0000

Now converting the binary result stored $t0 to hexadecimal we get,

**St0 = 0xB0000000.**

**2.12.4 Is the result in $t0 the desired result or there has been overflow?**

**Ans.** The result stored in the register $t0 is the desired result. And hence there is no overflow.

**2.12.5 For the contents of the registers $s0 and $s1 as specified above, what is the value of $t0 for the following assembly code?**

**Add $t0, $s0, $s1**

**Add $t0, $t0, $s0**

**Ans.** To perform the addition, convert the hexadecimal values in $s0 and $s1 into binary.

$s0 = 0x80000000 = 1000 0000 0000 0000 0000 0000 0000 0000

$s1 = 0xD0000000 = 1101 0000 0000 0000 0000 0000 0000 0000

$t0 = $s0 + $s1

= 1000 0000 0000 0000 0000 0000 0000 0000

+ 1101 0000 0000 0000 0000 0000 0000 0000

$t0 = **1** 0101 0000 0000 0000 0000 0000 0000 0000

Now performing the second operation we get,

$t0 = $t0 + $s0,

= **1** 0101 0000 0000 0000 0000 0000 0000 0000

+ 1000 0000 0000 0000 0000 0000 0000 0000

$t0 = **1** 1101 0000 0000 0000 0000 0000 0000 0000

Now converting the binary result stored $t0 to hexadecimal we get,

**$t0 = 0xD0000000**. Since the register is of 32 bits, the 1 stored on the 33rd will be ignored because there has been an overflow.

**2.12.6 Is the result in $t0 the desired result or has been overflow?**

Ans. After adding the registers $s0 and $s1. The result obtained is 0x1D0000000 which is more than 32bits. As we have the 32 bits registers, we will not be able to store the result value of the 33rd bit. Hence **the result obtained after the addition is not the desired result and there has been overflow.**

**2.14 Provide the type and assembly language instruction for the following binary value: 0000 0010 0001 0000 1000 0000 0010 0000two**

**Ans.**

MIPS instructions consist of the following fields-

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Op | rs | rt | rd | shamt | funct |

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

**Therefore, grouping the bits according to the MIPS instruction we get,**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 000000 | 10000 | 10000 | 10000 | 00000 | 100000 |

**Hence the decimal representation of the above is,**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 16 | 16 | 16 | 0 | 32 |

The first and the last fields in combination tell the MIPS computer that this instruction performs addition.

The second field gives us the register for first source operand of the addition that is **rs is 16 = $s0**

The third field gives us the register for second source operand of the addition that is **rt is 16 = $s0**

The forth field gives us the register which receives the sum of the first and second operand that is rd is 16 = $s0.

The fifth field is unused in this instruction, so it is set 0.

So the MIPS instruction is **R-type** instruction :-

**add $s0, $s0, $s0**

**2.15 Provide the type and hexadecimal representation of following instruction: sw $t1, 32($t2)**

To keep all the instructions of the same length, there by requiring different kinds of instruction formats for different kinds of instructions. One of the instruction format is called I-format or I type and it is used for the immediate type and data transfer instructions.

The instruction sw $t1, 32($t2) is a data transfer instruction and hence its an I-Type instruction.

The fields of the I-format are as follows:-

|  |  |  |  |
| --- | --- | --- | --- |
| Op | rs(source) | rt(destination) | constant/address |

6 bits 5 bits 5 bits 16 bits

Let us consider the instruction- sw $t1, 32($t2)

Here 10 (for $t2) is placed in the rs field, 9 (for $t1) is placed in the rt field, and 32 is placed in the address field. And 43ten that is opcode for store word (sw) is placed in the op field.

Therefore decimal representation above instruction is,

|  |  |  |  |
| --- | --- | --- | --- |
| 43 | 10 | 9 | 32 |

6 bits 5 bits 5 bits 16 bits

The binary representation of the above is

|  |  |  |  |
| --- | --- | --- | --- |
| 101011 | 01010 | 01001 | 0000000000100000 |

Grouping the binary bits in order to obtain the hexadecimal,

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1010 | 1101 | 0100 | 1001 | 0000 | 0000 | 0010 | 0000 |

Thus the hexadecimal representation of the instruction is: - **0xAD490020**

**2.24 Suppose the program counter (PC) is set to 0x20000000. Is it possible to use the jump (j) MIPS assembly instruction to set the PC to the address as 0x40000000? Is it possible to use the branch-on-equal (beq) MIPS assembly**

**instruction to set the PC to this same address?**

Answer:

A jump instruction is an unconditional branch instruction it can take us to any address in the 26 bits of the target. So the range of jump instructions is up to 226-1 which is a smaller value than going to 0x40000000 from 0x20000000 So **NO** it is not possible to use the jump (j) instruction to set the PC to the address as 0x40000000.

The BEQ instruction takes you to an offset from the current program counter, but the offset is only 16 bits. So a branch can only take us to any address 216-1 from the value in the PC. Thus, **NO** we cannot reach 0x40000000 from 0x20000000 by using beq.

**2.39 Write the MIPS assembly code that creates the 32-bit constant**

**0010 0000 0000 0001 0100 1001 0010 0100two and stores that value to register $t1.**

**Ans.**

First we would load the upper 16 bits, which is 8193 in decimal using lui to set the upper 16 bits of a constant in the register:

lui $t1, 8193 #8193 decimal = 0010 0000 0000 0001 binary

The value of the register $t1 afterward is

0010 0000 0000 0001 0000 0000 0000 0000

The next step is to insert the lower 16 bits, whose decimal value is 18724:

ori $t1, $t1, 18724 #18724 decimal = 0100 1001 0010 0100 binary

**The final value in the register $t1 is the desired value,**

**0010 0000 0000 0001 0100 1001 0010 0100**

**2.40 If the current value of the PC is 0x00000000, can you use a single jump instruction to get to the PC address as shown in Exercise 2.39?**

**Ans.**

The MIPS jump instruction format, called the J-type, consist of 6 bits for the operation field and the rest of the bits for the address field.

|  |  |
| --- | --- |
| op | Address |

6bits 26 bits

As this instruction has 26 bits for its target which is 226-1 that is 0x3FFFFFF. And the address of the instruction in the exercise 2.39 is 0x20014924 which is more than 26 bits. **Hence we cannot use the single jump instruction to get to the PC address as shown in the exercise 2.39**