**Number Systems**

**Conversions:**

**Decimal to Binary** → Divide by 2 until you can no longer divide the number anymore. From bottom to the right is your binary.

**Binary to Hex** → Divide into 4s → Count how many of each “place” you have via 2n → add them up, convert.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bin** | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1001 | 1100 | 1101 | 1110 | 1111 |
| **Hex** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |

**If bit 0 is 0, even number, if bit 0 is 1, odd number**

**Sign Magnitude →** 0 = positive, 1 = negative 33 = 00100001 -33 = 10100001

**1’s Complement** → Flip everything from unsigned -33 = 11011110

**2’s Complement** → Add 1 to flipped 1’s complement -33 = 11011111

Bit (binary digit) → nibble (4 binary digits) → byte (8 binary digits) → word (16/32)

Smallest val = 010 Largest val = 25510 (256 range)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1024 | 512 | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

**MIPS Assembly Code**

**op:** operation (opcode) **rs:** first register operand **rt:** second register operand **rd:** destination

**shamt:** shift amount **funct:** function code/ ID what ops needs

Why do we only need 5 bis to shift? Only need 5 to tell you how to shift

0 / 20 hex → opcode / funct → take first 6 bits bc funct only takes 6 bits in R-type

**imm# can be constant or bit**

PSEUDO-ops:

li (load immediate 16/32 bit) $t0, 0x00012345 → lui $at, 0x001 then ori $t0, $at, 0x2345

la $a0, L1 → lui $at (upper half L1) then ori $a0, $at (lower half L2)

Left most = MSB, right most = LSB activation frame does not store instructions

Big Endian → MSB is smallest address → 12, 34, 56, 78 no relative addressing in j-type jump

Little Endian → LSB is smallest address → 78, 56, 34, 12 MIPS is little endian

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| The answer is no. As discussed in class, the instruction addi uses I-type which has 16-bit immediate value. The immediate value is in two's complement format. The value 0x89ab is a 16-bit value with the most significant bit 1 (10001001101010112). Thus, it is considered a negative number. If the instuction addi is used as shown in the question, the actual value stored in the register $s0 will be 0x12340000 + 0xffff88ab.  Consider the code below:  addi $t0, $zero, 5 addi $t1, $zero, 9 and  $t2, $t0, $t1  What is the value stored in $t2 (in decimal without subscript) after the above code has been executed? **1**   * For bit-wise and: * 0 & 0 = 0 * 0 & 1 = 0 * 1 & 0 = 0 * 1 & 1 = 1  |  | | --- | | Consider the following code where numbers on the left column (in decimal) is the address of instructions:  36|         add $t0, $t1, $t2 40|         j aLabel 44|         sub $t0, $t1, $t2 : : 76|         and $t0, $t1, $t2 80| aLabel: or  $t0, $t1, $t2 84|         nor $t0, $t1, $t2  What is the 26-bit address field (in decimal) of the instruction j aLabel? | | |  |  | | --- | --- | | Response Feedback: | Insturction jump (j) uses word addressing but MIPS uses byte addressing. Thus, we need to shift the 26-bit address field by 2 (multiplied by 4) to get the 28-bit byte addressing. We patch the top 4 bits of the 28-bit address by the top 4 bit of the program counter at the jump instructoin to get the 32-bit address.  The jump instruction is located at the address 40 (in decimal). Thus the top 4 bits of the program counter is 0000. The destination at aLabel is at the address 80 (in decimal). Thus, the top 4 bits of the destination address is also 0000. We do this just to check in case it is out-of-range of jump. In this case, they are the same. So, the destination is in range of jump.  The 28-bit address of the destination is 80 in decimal. This is the byte addressing. To get the word addressing, we need to shift right by 2 (or divided by 4). Thus, the 26-bit address field of the instruction jump is 80/4 or 20. | | | |  | | --- | | Consider the code fragment below:  sll  $t0, $t0, 1 sra  $t0, $t0, 1  The above code fragement supposes to clear the MSB of $t0 to 0. Is there anything wrong with the above code fragment? | | |  |  | | --- | --- | |  | Correctb.  If bit 30 is of $t0 is 1, after sll, sra will shift in 1 which makes the MSB of $t0 to be 1. | | |
|  |  |