**Instruction 1: 0x0295001A**

Each MIPS instruction is of 32 bit wide, now the first task is to convert the above Hexadecimal instruction into it’s binary equivalent.

Binary Format of the above instruction is

|  |
| --- |
| 0000 0010 1001 0101 0000 0000 0001 1010 |

Let us rearrange them into instruction tokens

Split into instruction format

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 6 bit (opcode) | 5 bit | 5 bit | 5 bit | 5 bit | 6 bit |
| 000000 | 10100 | 10101 | 00000 | 00000 | 011010 |
| 0 | 20 | 21 | 0 | 0 | 26 |

Since the opcode is 0 (i.e. 000000), it is a **R-Type** of instruction (explained below)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **6 bit (opcode)** | 5 bit | 5 bit | 5 bit | 5 bit | 6 bit |
| 000000 | 10100 | 10101 | 00000 | 00000 | 011010 |
| 0 | 20 | 21 | 0 | 0 | 26 |

To find the instruction, let us examine first 6 bits (i.e. bit number 0 to 2 and 3 to 5) (Explained in the table below)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 6 bit (opcode) | 5 bit | 5 bit | 5 bit | 5 bit | **6 bit (DIV)** | |
| 000000 | 10100 | 10101 | 00000 | 00000 | 011 | 010 |
| 0 | 20 | 21 | 0 | 0 | 3 | 2 |

The instruction is “**div**”

Next we have to find the registers (rs, rt) that are explained below-

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 6 bit (opcode) | **5 bit (rs)** | **5 bit (rt)** | 5 bit | 5 bit | 6 bit |
| 000000 | 10100 | 10101 | 00000 | 00000 | 011010 |
| 0 | 20 | 21 | 0 | 0 | 26 |

MIPS has 32 registers (number from 0 to 31) here our register values are 20 and 21 respectively. While 20 represents **$s4**, 21 is the value for register **$s5**.

Now out final MIPS instruction is

**div $s4, $s5**

**Instruction 2: 0x00007010**

Each MIPS instruction is of 32 bit wide, now the first task is to convert the above Hexadecimal instruction into it’s binary equivalent.

Binary Format of the above instruction is

|  |
| --- |
| 0000 0000 0000 0000 0111 0000 0001 0000 |

Let us rearrange them into instruction tokens

Split into instruction format

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 6 bit (special bits) | 10 bits | 5 bit | 5 bit | 6 bit |
| 000000 | 0000000000 | 01110 | 00000 | 0100000 |
|  |  | 14 | 0 |  |

The opcode for this can be determined from bit 0 to 5 (6 bits) as the last 6 bits hold 0 followed by 10 bits (a 0 again)

The instruction stands for **mfhi** that moves from hi registers

And then the rd target register can be obtained from bit numbr10 to 15 which is 14 that stands for **$t6**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 6 bit (special bits) | 10 bits | 5 bit | 5 bit | 6 bit |
| 000000 | 0000000000 | 01110 | 00000 | 0100000 |
|  |  | 14 | 0 |  |

So final instruction is

**mfhi $t6**

**Instruction 3: 0x29CCFFFF**

Each MIPS instruction is of 32 bit wide, now the first task is to convert the above Hexadecimal instruction into it’s binary equivalent.

Binary Format of the above instruction is

|  |
| --- |
| **0010 1001 1100 1100 1111 1111 1111 1111** |

Let us rearrange them into instruction tokens

Split into instruction format

|  |  |  |  |
| --- | --- | --- | --- |
| 6 bit (opcode) | 5 bit | 5 bit | 16 bit |
| 001010 | 01110 | 01100 | 111111111111111 |
|  | 14 | 12 | FFFFF |

Since the opcode is 0 (i.e. 001010), it is a **Immediate-Register-Type** of instruction (explained below)

|  |  |  |  |
| --- | --- | --- | --- |
| 6 bit (opcode) | 5 bit | 5 bit | 16 bit |
| 001010 | 01110 | 01100 | 111111111111111 |
| **slti** | 14 | 12 | FFFF |

The instruction is “**slti**”

Next we have to find the registers (rs, rt) and immediate that are explained below-

|  |  |  |  |
| --- | --- | --- | --- |
| 6 bit (opcode) | 5 bit (rs) | 5 bit (rt) | 16 bit (immediate) |
| 001010 | 01110 | 01100 | 111111111111111 |
| 0 | 14 | 12 | FFFF |

MIPS has 32 registers (number from 0 to 31) here our register values are 14 and 12 respectively. While 14 represents **$t6**, 12 is the value for register **$t4**.

Now out final MIPS instruction is

**slti $t4, $t6, 0xFFFF**

**Instruction 4: 0XAE8CFFF0**

Each MIPS instruction is of 32 bit wide, now the first task is to convert the above Hexadecimal instruction into it’s binary equivalent.

Binary Format of the above instruction is

|  |
| --- |
| 1010 1110 1000 1100 1111 1111 1111 0000 |

Let us rearrange them into instruction tokens

Split into instruction format

|  |  |  |  |
| --- | --- | --- | --- |
| 6 bit (opcode) | 5 bit | 5 bit | 16 bit (Offset) |
| **101011** | 10100 | 01100 | 1111 1111 1111 0000 |
| sw | 20 | 12 | FFFF0 |

Op code is 101011 which is “**sw**” instruction

Next we have to find the registers (rs, rt) offset that are explained below-

|  |  |  |  |
| --- | --- | --- | --- |
| 6 bit (opcode) | 5 bit | 5 bit | 16 bit (Offset) |
| **101011** | 10100 | 01100 | 1111 1111 1111 0000 |
| sw | 20 | 12 | FFFF0 |

MIPS has 32 registers (number from 0 to 31) here our register values are 20 and 12 respectively. While 20 represents **$s4**, 12 is the value for register **$t4**.

Now out final MIPS instruction is

**sw $t4, 0xFFF0($s4)**