1. Write the following instruction in Hex: ADDU $t0, $s4, $t3

0000 0010 1000 1011 0100 0000 0010 0001

0x028b4021

2. Decode the following instruction: 0x36890010

0011 0110 1000 1001 0000 0000 0001 0000

001101 10100 01001 0000000000010000

ori $t1, $s4 16

ori $t1, $s4 0x10

3. Write the following instruction in Hex: ADDU $s2, $s3, $t4

000000 10011 01100 10010 00000 100001

0000 0010 0110 1100 1001 0000 0010 0001

0x026c9021

4. Decode the following instruction: 0x25b40005

0010 0101 1011 0100 0000 0000 0000 0101

addiu $s4, $t5 5

addiu $s4, $t5 0x5

**ADD -- *Add***

|  |  |
| --- | --- |
| Description: | Adds two registers and stores the result in a register |
| Operation: | $d = $s + $t; advance\_pc (4); |
| Syntax: | add $d, $s, $t |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 0000 |

**ADDI -- *Add immediate (with overflow)***

|  |  |
| --- | --- |
| Description: | Adds a register and a sign-extended immediate value and stores the result in a register |
| Operation: | $t = $s + imm; advance\_pc (4); |
| Syntax: | addi $t, $s, imm |
| Encoding: | 0010 00ss ssst tttt iiii iiii iiii iiii |

**ADDIU -- *Add immediate unsigned (no overflow)***

|  |  |
| --- | --- |
| Description: | Adds a register and a sign-extended immediate value and stores the result in a register |
| Operation: | $t = $s + imm; advance\_pc (4); |
| Syntax: | addiu $t, $s, imm |
| Encoding: | 0010 01ss ssst tttt iiii iiii iiii iiii |

**ADDU -- *Add unsigned (no overflow)***

|  |  |
| --- | --- |
| Description: | Adds two registers and stores the result in a register |
| Operation: | $d = $s + $t; advance\_pc (4); |
| Syntax: | addu $d, $s, $t |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 0001 |

**AND -- *Bitwise and***

|  |  |
| --- | --- |
| Description: | Bitwise ands two registers and stores the result in a register |
| Operation: | $d = $s & $t; advance\_pc (4); |
| Syntax: | and $d, $s, $t |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 0100 |

**ANDI -- *Bitwise and immediate***

|  |  |
| --- | --- |
| Description: | Bitwise ands a register and an immediate value and stores the result in a register |
| Operation: | $t = $s & imm; advance\_pc (4); |
| Syntax: | andi $t, $s, imm |
| Encoding: | 0011 00ss ssst tttt iiii iiii iiii iiii |

**OR -- *Bitwise or***

|  |  |
| --- | --- |
| Description: | Bitwise logical ors two registers and stores the result in a register |
| Operation: | $d = $s | $t; advance\_pc (4); |
| Syntax: | or $d, $s, $t |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 0101 |

**ORI -- *Bitwise or immediate***

|  |  |
| --- | --- |
| Description: | Bitwise ors a register and an immediate value and stores the result in a register |
| Operation: | $t = $s | imm; advance\_pc (4); |
| Syntax: | ori $t, $s, imm |
| Encoding: | 0011 01ss ssst tttt iiii iiii iiii iiii |

**SB -- *Store byte***

|  |  |
| --- | --- |
| Description: | The least significant byte of $t is stored at the specified address. |
| Operation: | MEM[$s + offset] = (0xff & $t); advance\_pc (4); |
| Syntax: | sb $t, offset($s) |
| Encoding: | 1010 00ss ssst tttt iiii iiii iiii iiii |

**SLL -- *Shift left logical***

|  |  |
| --- | --- |
| Description: | Shifts a register value left by the shift amount listed in the instruction and places the result in a third register. Zeroes are shifted in. |
| Operation: | $d = $t << h; advance\_pc (4); |
| Syntax: | sll $d, $t, h |
| Encoding: | 0000 00ss ssst tttt dddd dhhh hh00 0000 |

**SLT -- *Set on less than (signed)***

|  |  |
| --- | --- |
| Description: | If $s is less than $t, $d is set to one. It gets zero otherwise. |
| Operation: | if $s < $t $d = 1; advance\_pc (4); else $d = 0; advance\_pc (4); |
| Syntax: | slt $d, $s, $t |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 1010 |

**SLTI -- *Set on less than immediate (signed)***

|  |  |
| --- | --- |
| Description: | If $s is less than immediate, $t is set to one. It gets zero otherwise. |
| Operation: | if $s < imm $t = 1; advance\_pc (4); else $t = 0; advance\_pc (4); |
| Syntax: | slti $t, $s, imm |
| Encoding: | 0010 10ss ssst tttt iiii iiii iiii iiii |

**SLTU -- *Set on less than unsigned***

|  |  |
| --- | --- |
| Description: | If $s is less than $t, $d is set to one. It gets zero otherwise. |
| Operation: | if $s < $t $d = 1; advance\_pc (4); else $d = 0; advance\_pc (4); |
| Syntax: | sltu $d, $s, $t |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 1011 |

**SRL -- *Shift right logical***

|  |  |
| --- | --- |
| Description: | Shifts a register value right by the shift amount (shamt) and places the value in the destination register. Zeroes are shifted in. |
| Operation: | $d = $t >> h; advance\_pc (4); |
| Syntax: | srl $d, $t, h |
| Encoding: | 0000 00-- ---t tttt dddd dhhh hh00 0010 |

**SUB -- *Subtract***

|  |  |
| --- | --- |
| Description: | Subtracts two registers and stores the result in a register |
| Operation: | $d = $s - $t; advance\_pc (4); |
| Syntax: | sub $d, $s, $t |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 0010 |

**SW -- *Store word***

|  |  |
| --- | --- |
| Description: | The contents of $t is stored at the specified address. |
| Operation: | MEM[$s + offset] = $t; advance\_pc (4); |
| Syntax: | sw $t, offset($s) |
| Encoding: | 1010 11ss ssst tttt iiii iiii iiii iiii |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Register Number** | **Usage** | **Preserved by callee?** |
| $zero | 0 | hardwired 0 | N/A |
| $v0-$v1 | 2-3 | return value and expression evaluation | no |
| $a0-$a3 | 4-7 | arguments | no |
| $t0-$t7 | 8-15 | temporary values | no |
| $s0-$s7 | 16-23 | saved values | YES |
| $t8-$t9 | 24-25 | more temporary values | no |
| $gp | 28 | global pointer | YES |
| $sp | 29 | stack pointer | YES |
| $fp | 30 | frame pointer | YES |
| $ra | 31 | return address | YES |