|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 211 | 210 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | . | 2-1 | 2-2 | 2-3 | 2-4 | 2-5 | 2-6 |
| 2048 | 1024 | 512 | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 | . | .5 | .25 | .125 | .0625 | .03125 | .015625 |

**Convert decimal to hex**

Decimal Number: -127.25

Step 1: Convert to binary -1111111.112

Step 2: Scientific notation **-**1.111111112 X 26

|  |  |  |
| --- | --- | --- |
| S | EXP(133) | Fraction(11111111) |
| 1 | 10000101 | 11111111000000000000000 |

Step 3: Separate

Notes

1. If negative S = 1 else S = 0
2. Exp = 127 + Exp (always 127)
3. Fraction = .xxxxxxxx…x

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1100 | 0010 | 1111 | 1111 | 10000 | 0000 | 0000 | 0000 |
| C | 2 | F | F | 8 | 0 | 0 | 0 |

Step 4: Reorganize and convert

0xC2FF8000

**Convert hex to decimal**

Hex Number: 0XC0A80000

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | 0 | A | 8 | 0 | 0 | 0 | 0 |
| 1100 | 0000 | 1010 | 1000 | 0000 | 0000 | 0000 | 0000 |

Step 1: Convert to binary

|  |  |  |
| --- | --- | --- |
| S | Exp | Fraction |
| 1 | 10000001 | 0101000000000000000 |

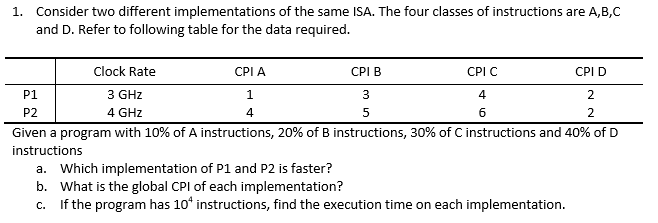
Step 2: Rearrange

Step 3: Decipher -1.01012 X 22

-101.012

-5.25

**Performance**



1. P1 is faster because it has a lower clock rate?
2. P1: 1(.1) + 3(.2) + 4(.3) + 2(.4) = 2.7

P2: 4(.1) + 5(.2) + 6(.3) + 2(.4) = 4.0

1. 104 = 10,000 total instructions

A (10%): 10000 \* .1 = 1000

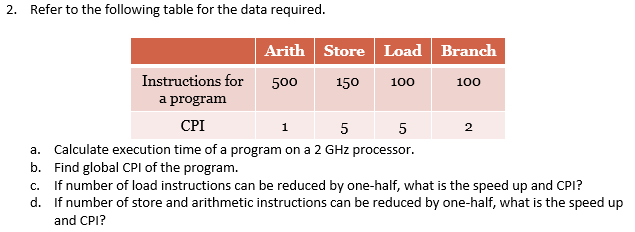
B (20%): 10000 \* .2 = 2000

C (30%): 10000 \* .3 = 3000

D (40%): 10000 \* .4 = 4000

P1:

P2:



1(500) + 5(150) + 5(100) + 2(100) = 1950 cycles

500 + 150 + 100 + 100 = 850 instructions



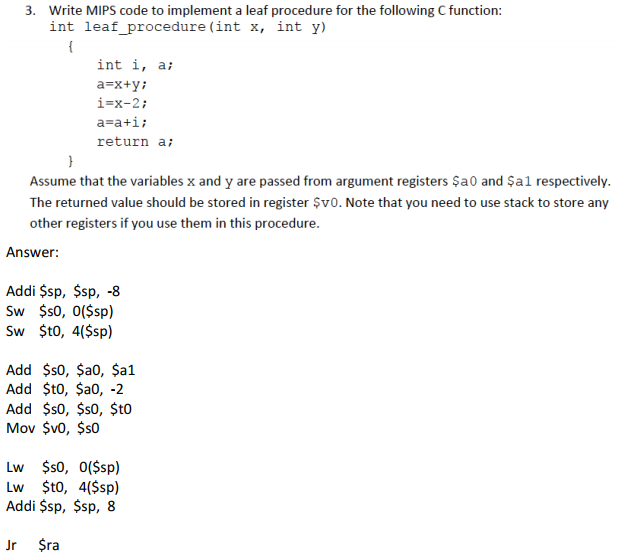
Speed Up =

CPI = Total instructions = 1950 – 5(50) cycles = 1700 | 850 instructions – 50 = 800

Speed Up =

CPI = Total instructions = 1950 – 250 -5(75) cycles = 1325 | 850 instructions – 250 - 75 = 525

**MIPS code**



**Overflow**

|  |  |  |  |
| --- | --- | --- | --- |
| **Operation** | **Operand A** | **Operand B** | **Result** |
| A + B | **+** | **+** | **-** |
| A + B | **-** | **-** | **+** |
| A - B | **+** | **-** | **-** |
| A - B | **-** | **+** | **+** |

Add $t1, $t2, $t3

$t2 = 0xf0000000 $t3 = 0xf0000000

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 1 | 1 | 1 |  |  |  |  |  |  |  |  |
|  |  | 1 | 1 | 1 | 1 | 0 | 0 | … | 0 | 0 | 0 | 0 |
| + |  | 1 | 1 | 1 | 1 | 0 | 0 | … | 0 | 0 | 0 | 0 |
|  | 1 | 1 | 1 | 1 | 1 | 0 | 0 | … | 0 | 0 | 0 | 0 |

There is no overflow because

Sub $t1, $t2, $t3 >> $t2 - $t3

$t2 = 0x0000000f $t3 = 0xf800000f

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  | 1 | 1 | 1 | 1 |  |
|  | 0 | 0 | 0 | 0 | 0 | 0 | … | 0 | 1 | 1 | 1 | 1 |
| + | 1 | 0 | 0 | 0 | 0 | 0 | … | 0 | 1 | 1 | 1 | 1 |
|  | 1 | 0 | 0 | 0 | 0 | 0 | … | 1 | 1 | 1 | 1 | 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Instruction** | **Format** | **Syntax** | **Operation** | **Opcode** | |
| **Dec** | **Hex** |
| Add | R | add $d, $s, $t | $d = $s + $t | 100000 | 20 |
| Add immediate | I | addi $t, $s, i | $t = $s + i | 001000 | 8 |
| And | R | and $d, $s, $t | $d = $s & $t | 100100 | 24 |
| And immediate | I | andi $t, $s, i | $t = $s & i | 001100 | c |
| Or | R | or $d, $s, $t | $d = $s | $t | 100101 | 25 |
| Ori | I | ori $t, $s, i | $t = $s | i | 001101 | d |
| Shift left logical | R | sll $d, $t, a | $d = $t \* 2a | 000000 | 00 |
| Shift right logical | R | Srl $d, $t, a | $d = $t / 2a | 000010 | 2 |
| Subtract | R | Sub $d, $s, $t | $d = $s - $t | 100010 | 22 |
| Branch on equal | I | Beq $s, $t, label | If ($s == $t) {label} | 000100 | 4 |
| Branch on not equal | I | Bne $s, $t, label | If ($s != $t) {label} | 000101 | 5 |
| Branch on greater than | I | Bgt $s, $t, label | If ($s > $t) {label} |  | |
| Branch on greater or equal | I | Bge $s, $t, label | If ($s >= $t) {label} |  | |
| Branch on less than | I | Blt $s, $t, label | If ($s < $t) {label} |  | |
| Branch on less or equal | I | Ble $s, $t, label | If ($s <= $t) {label} |  | |
| Jump | J | J label | Go to label | 000010 | 2 |
| Load word | I | lw $t, i($s) | $t = MEM [$s + i]  \*i is a mult of 4 | 100011 | 23 |
| Store word | I | sw $t, i($s) | MEM [$s + i] = $t  \*i is a mult of 4 | 101011 | 2b |

**Formats**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R | Opcode | | Rs | | Rt | | Rd | | Shamt | | funct | |
| 31 | 26 | 25 | 21 | 20 | 16 | 15 | 11 | 10 | 6 | 5 | 0 |
| I | Opcode | | Rs | | Rt | | immediate | | | | | |
| 31 | 26 | 25 | 21 | 20 | 16 | 15 | | | 0 | | |
| J | Opcode | | address | | | | | | | | | |
| 31 | 26 | 25 | | | | | 0 | | | | |

**Bitwise**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **And** | | | | | | | | **Or** | | | | | | | |  |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |  | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |  | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |  | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |

**Register name and number use**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Register name** | **Dec number** | **Bin number** |  | **Register name** | **Dec number** | **Bin number** |
| $zero | 0 | 00000 |  | $t5 | 13 | 01101 |
| $at | 1 | 00001 |  | $t6 | 14 | 01110 |
| $v0 | 2 | 00010 |  | $t7 | 15 | 01111 |
| $v1 | 3 | 00011 |  | $s0 | 16 | 10000 |
| $a0 | 4 | 00100 |  | $s1 | 17 | 10001 |
| $a1 | 5 | 00101 |  | $s2 | 18 | 10010 |
| $a2 | 6 | 00110 |  | $s3 | 19 | 10011 |
| $a3 | 7 | 00111 |  | $s4 | 20 | 10100 |
| $t0 | 8 | 01000 |  | $s5 | 21 | 10101 |
| $t1 | 9 | 01001 |  | $s6 | 22 | 10110 |
| $t2 | 10 | 01010 |  | $s7 | 23 | 10111 |
| $t3 | 11 | 01011 |  | $t8 | 24 | 11000 |
| $t4 | 12 | 01100 |  | $t9 | 25 | 11001 |
| $t5 | 13 | 01101 |  |  |  |  |

**Conversions**

|  |  |
| --- | --- |
| **Binary (101110)** | |
| **Hexadecimal** | **Decimal** |
| |  |  |  | | --- | --- | --- | | Binary | 0010 | 1110 | | Decimal | 2 | 14 | | Hex | 2E | | | (25 \* 1) + ( 24 \* 0) + ( 23 \* 1) + ( 22 \* 1) + ( 21 \* 1) + ( 20 \* 0) = 46 |
| **Hexadecimal (2E)** | |
| **Binary** | **Decimal** |
| |  |  |  | | --- | --- | --- | | Hex | 2 | E | | Decimal | 2 | 14 | | Bin | 0010 | 1110 | | (161 \* 2) + ( 160 \* 14) = 46 |
| **Decimal (46)** | |
| **Hexadecimal** | **Binary** |
| |  |  |  | | --- | --- | --- | | Binary | 0010 | 1110 | | Decimal | 2 | 14 | | Hex | 2E | | | |  |  | | --- | --- | | 46 | 0 | | 23 | 1 | | 11 | 1 | | 5 | 1 | | 2 | 0 | | 1 | 1 |   101110 |

**2’s complement**

Given: FFFFFFAB = 1111 1111 1111 1111 1111 1111 1010 1011

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1111** | **1111** | **1111** | **1111** | **1111** | **1111** | **1010** | **1011** | Given |
|  | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0101 | 0100 | Switch 1 -> 0 and 0 -> 1 |
| + |  |  |  |  |  |  |  | 1 | Add 1 |
|  | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0101 | 0101 | Final answer in binary |
|  |  |  |  |  |  |  |  | = 85 | Convert |

**Examples MIPs <=> high level language <=> machine code**

|  |
| --- |
| for (i = 0; i < 100; ++i)  {  sum += A[i];  } |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | &A[0]: $s0 sum: $v0 $t0 = 1 | | | | | | |  | and | $t0, | $t0, | $zero | # let i = 0 | |  | addi | $t1, | $zero, | 100 | # intermediate 100 | | Loop: | lw | $t3, | 0($s0) |  | # temp1 = A[i] | |  | add | $v0, | $v0, | $t3 | # sum += temp1 | |  | addi | $s0, | $s0, | 4 | # address of A[i + 1] | |  | addi | $t0 | $t0 | 1 | #i = i + 1 | |  | bne | $t1, | $t0, | Loop | # if i < 100 | | Exit: |  |  |  |  |  | |

|  |
| --- |
| for (i = 0; i < 100; ++i)  {  A[i] = i;  } |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | and | $t0, | $zero, | $zero | |  | addi | $t1, | $zero, | 100 | | Loop: | sw | $t0, | 0($s2) |  | |  | addi | $s2, | $s2, | 4 | |  | addi | $t0 | $t0 | 1 | |  | beq | $t0, | $t1, | Loop | | Exit: |  |  |  |  | |

**FP Multiplication**

|  |  |
| --- | --- |
| **Decimal** | **Binary** |
| Consider a 4-digit decimal example  (1.110 x 1010) \* (9.200 x 10-5)   1. Add exponents    * 10 + (-5) = 5 2. Multiply significands  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 1. | 1 | 1 | 0 |  | | x | 9. | 2 | 0 | 0 |  | | 1 | 0. | 2 | 1 | 2 | X 105 |  1. Normalize check for over/underflow    * 1.0212 x 106 2. Round and renormalize if necessary    * 1.021 x 106 3. Determine sign (from original numbers)    * + \* + = +    * + \* - = -    * - \* - = +   1.021 x 106 | Consider a 4-digit binary example  (1.0002 x 2-1) \* (-1.1102 x 2-2)   1. Add exponents    * Unbiased: -1 + -2 = -3    * Biased: (–1 + 127) + (–2 + 127) = –3 + 254 – 127 = –3 + 127 2. Multiply significands  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | |  | 1. | 0 | 0 | 0 |  | | x | 1. | 1 | 1 | 0 |  | | 1. | 1 | 1 | 0 | 2 | X 10-3 |  1. Normalize check for over/underflow    * 1.1102 x 2-3 2. Round and renormalize if necessary    * No change 3. Determine sign (from original numbers)    * + \* + = +    * + \* - = -    * - \* - = +   -1.1102 x 10 -3 |

**Not optimized way to multiply Optimized way**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **M1** | **M2** | **Product** |  | **itr** |  | **Multiplicand** | **Product** |  | **itr** |
| 0011 | 0000 0010 | 0000 0000 | Initial | 0 | 1001 | 0000 0101 | LSB = 1 | 1 |
| 0011 | 0000 0010 | **0000 0010** | Prod += M2 | 1 | 1001 | 1001 0101 | Add |
| 0011 | **0000 0100** | 0000 0010 | SLL M2 | 1001 | 0100 1010 | Shift |
| **0001** | 0000 0100 | 0000 0010 | SRL M1 | 1001 | 0100 1010 | LSB = 0 | 2 |
| 0001 | 0000 0100 | **0000 0110** | Prod += M2 | 2 | 1001 | 0010 0101 | Shift |
| 0001 | **0000 1000** | 0000 0110 | SLL M2 | 1001 | 0010 0101 | LSB = 1 | 3 |
| **0000** | 0000 1000 | 0000 0110 | SRL M1 | 1001 | 1011 0101 | Add |
| 0000 | 0000 1000 | **0000 0110** | No more M1 | 3 | 1001 | 0101 1010 | Shift |
| 0000 | **0001 0000** | 0000 0110 | SLL M2 | 1001 | 0101 1010 | LSB = 0 | 4 |
| **0000** | 0001 0000 | 0000 0110 | SRL M1 | 1001 | 0010 1101 | Shift |
| 0000 | 0001 0000 | **0000 0110** | No more M1 | 4 |  | **0010 1101** |  | done |
| 0000 | **0010 0000** | 0000 0110 | SLL M2 |  |  |  |  |
| **0000** | 0010 0000 | **0000 0110** | SRL M1 |  |  |  |  |

**Not optimized Division**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7/2 = 0111/0010 = 0000 0111/0010 0000 | | | | |
| **Quotient** | **Divisor** | **Remainder** |  | **itr** |
| 0000 | 0010 0000 | 0000 0111 | Rem -= Div -> < 0 | 0 |
| 0000 | 0001 0000 | 0000 0111 | SRL Div | 1 |
| 0000 | 0001 0000 | 0000 0111 | Rem -= Div -> < 0 |
| 0000 | 0000 1000 | 0000 0111 | SRL Div | 2 |
| 0000 | 0000 0100 | 0000 0111 | Rem -= Div -> <= 0 | 3 |
| 0000 | 0000 0100 | 0000 0111 | Rem =- Divisor | 4 |
| 0001 | 0000 0100 | 0000 0011 | +1 Quotient |
| 0001 | 0000 0010 | 0000 0011 | SRL Div |
| 0011 | 0000 0001 | 0000 0001 | Rem -= Div -> <= 0 | 5 |
| Quotient = 3, Remainder = 1 | | | | |

**Optimized Division**

|  |  |  |  |
| --- | --- | --- | --- |
| 11/4 = 1011/0100 = 0000 1011/0100 | | | |
| **Divisor** | **Remainder** |  | **Itr** |
| 0100 | 0000 1011 | Initial | 0 |
| 0100 | 0001 0110 | SLL rem | 1 |
| 0100 | 0010 1100 | SLL rem | 2 |
| 0100 | 0101 1000 | SLL rem | 3 |
| 0100 | 0001 1000 | Subtract |
| 0100 | 0011 0001 | SLL rem & append 1 | 4 |
| 0100 | 0110 0010 | SLL rem |
| 0100 | **0011 0010** | SLL **first 4 bits leave last 4 bits** | done |
| **Quotient = 0010 = 2**, **Remainder = 0011 = 3** | | | |

**Special Operations**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operation** | **Result** |  | **Operation** | **Result** |
| n ÷ ±Infinity | 0 |  | 0 / 0 | NaN |
| ±Infinity × ±Infinity | ±Infinity |  | Infinity – Infinity | NaN |
| ±nonzero ÷ 0 | ±Infinity |  | ±Infinity / ±Infinity | NaN |
| Infinity + Infinity | Infinity |  | ±Infinity \* 0 | NaN |