**Q1.** (20 pts) A program, being executed on a processor, has the following instructions mix:

|  |  |  |
| --- | --- | --- |
| **Operation** | **Frequency** | **Clock cycles per instruction** |
| ALU | 40% | 2 |
| Load | 20% | 10 |
| Store | 15% | 4 |
| Branches | 25% | 3 |

1. (3 pts) Compute the average clock cycles per instruction.

**Average CPIa = 0.4\*2 + 0.2\*10 + 0.15\*4 + 0.25\*3 = 4.15**

1. (6 pts) Compute the percent of execution time spent by each class of instructions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operation** | **Frequency** | **CPI** | **CPI \* Frequency** | **% Execution Time** |
| ALU | 40% | 2 | 0.8 | 0.8 / 4.15 = 19.3% |
| Load | 20% | 10 | 2.0 | 2.0 / 4.15 = 48.2% |
| Store | 15% | 4 | 0.6 | 0.6 / 4.15 = 14.4% |
| Branches | 25% | 3 | 0.75 | 0.75 / 4.15 = 18.1 % |

1. (6 pts) A designer wants to improve the performance. He designs a new execution unit that makes 80% of ALU operations take only **1** cycle to execute. The other 20% of ALU operations will still take **2** cycles to execute. The designer also wants to improve the execution of the memory access instructions. He does it in a way that **95%** of the **load** instructions take only **2** cycles to execute, while the remaining **5%** of the **load** instructions take **10** cycles to execute per **load**. He also improves the store instructions in such a way that each **store** instruction takes **2** cycles to execute. Compute the new average cycles per instruction.

**Compute the new average cycles per instruction**

**Average CPIc = 0.8\*0.4\*1 + 0.2\*0.4\*2 + 0.2\*0.95\*2 + 0.2\*0.05\*10 + 0.15\*2 + 0.25\*3 = 2.01**

1. (2 pts) What is the speedup factor by which the performance has improved in part **c**?

**Speedup = 4.15 / 2.01 = 2.06 (I-count & clock are the same)**

1. (3 pts) The designer decides to improve the clock speed in such a way to **triple** the overall performance of the original CPU specified in part **a**. By what factor should the clock rate be improved if the designer uses the design specified in part **c**?

**Speedup = (CPIa / CPIc) \* (Clock Ratec/Clock Ratea)**

**Speedup = 3 = (4.15/2.01) \* (Clock Ratec/Clock Ratea)**

**Clock should be faster by 3/2.06 = 1.45 (45% faster)**

**Q2 (20 pts) Fill in Blanks or Tables**

1. (3 pts) Assume that the instruction **j NEXT** is at address **0x00401030**, and the label **NEXT** is at address **0x00400A18**. Then, the **26-bit immediate** stored in the jump instruction for the label **NEXT** is **0x00400A18 >> 2 = 0x100286** .
2. (3 pts) Assume that the instruction **beq $t0, $t1, NEXT** is at address **0x00401030**, and the label **NEXT** is at address **0x00402A18**. Then, the **16-bit immediate** stored in the branch instruction is **(0x00402A18-0x00401034)>>2 = 0x0679** **.**
3. (14 pts)Consider the following data definitions:

**.data**

**var1: .byte 3, -2, 'A'**

**var2: .half 1, 256, 0xffff**

**var3: .word 0x3de1c74, 0xff**

**.align 3**

**str1: .asciiz "ICS233"**

1. (9 pts) Show the content of each byte of the allocated memory, **in hexadecimal** for the above data definitions. The **Little Endian** byte ordering is used to order the bytes within words and half words. The ASCII code of character 'A' is 0x41, and '0' is 0x30. Indicate which bytes are skipped or unused in the data segment.

**Data Segment Symbol Table**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Address** | **Byte 3** | **Byte 2** | **Byte 1** | **Byte 0** |  | **Label** | **Address** |
| 0x10010000 | -- | 0x41 | 0xFE | 0x03 |  | var1 | 0x10010000 |
| 0x10010004 | 0x01 | 0x00 | 0x00 | 0x01 |  | var2 | 0x10010004 |
| 0x10010008 | -- | -- | 0xFF | 0xFF |  | var3 | 0x1001000C |
| 0x1001000C | 0x03 | 0xDE | 0x1C | 0x74 |  | str1 | 0x10010018 |
| 0x10010010 | 0x00 | 0x00 | 0x00 | 0xFF |  |  |  |
| 0x10010014 | -- | -- | -- | -- |  |  |  |
| 0x10010018 | 0x32 | 0x53 | 0x43 | 0x49 |  |  |  |
| 0x1001001C |  | 0x00 | 0x33 | 0x33 |  |  |  |
| 0x10010020 |  |  |  |  |  |  |  |
| 0x10010024 |  |  |  |  |  |  |  |
| 0x10010028 |  |  |  |  |  |  |  |
| 0x1001002C |  |  |  |  |  |  |  |

1. (4 pts) Construct a symbol table showing the symbols and their corresponding addresses in hexadecimal.
2. (1 pts) How many bytes are allocated in the data segment including the skipped bytes?

**31 Bytes including the skipped ones** .

**Q3. (10 pts) Floating-Point Number Representation**

1. (5 pts) Given that *x* is a single-precision IEEE 754 floating-point number:

*x* = **1 10000101 100 1010 0001 1010 0000 00112**

What is the decimal value of *x*?

**Sign bit = 1 (negative)**

**Biased Exponent = 10000101 = 133**

**Exponent Value = 133 – 127 = +6**

**Value = - (1.10010100001101000000011)2 × 26**

**= - (1100101.00001101000000011)2**

**Decimal Value = - 101.0508**

1. (5 pts) Convert **-6.25** from decimal to the IEEE 754 single-precision floating point format. Show all your work for each step in the solution.

**0.25 × 2 = 0.5**

**0.50 × 2 = 1.0**

**6.25 (decimal) = 110.01 (binary)**

**Normalize:**

**110.01 (binary) = 1.1001 × 22**

**Biased Exponent = 2 + 127 = 129 = 10000001 (binary)**

**IEEE 754 Single-Precision Representation:**

**1 10000001 100 1000 0000 0000 0000 0000**

**Q4. (15 pts) Tracing the Execution of Assembly Language Code**

1. (7 pts) Given that **Array** is defined as shown below, determine the content of register **$v0 and $v1** after executing the following code. Show your steps.

**Array: .word 15, -19, 17, 20, -10, 12, 100, -5**

**la $a0, Array # $a0 = 0x10010000**

**addi $a1, $a0, 28**

**move $v0, $a0**

**lw $v1, 0($v0)**

**move $t0, $a0**

**loop: addi $t0, $t0, 4**

**lw $t1, 0($t0)**

**bge $t1, $v1, skip**

**move $v0, $t0**

**move $v1, $t1**

**skip: bne $t0, $a1, loop**

**$v0 = 0x10010004 (address of minimum element)**

**$v1 = -19 (minimum value)**

1. (8 pts) Given that Array is defined as shown below, determine the content of Array after executing the following code. Show your steps.

**Array: .half 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12**

**la $a0, Array**

**li $a1, 6**

**move $t0, $a0**

**addi $t1, $a0, 12**

**loop: lh $t3, ($t0)**

**lh $t4, ($t1)**

**sh $t3, ($t1)**

**sh $t4, ($t0)**

**addi $t0, $t0, 2**

**addi $t1, $t1, 2**

**addi $a1, $a1, -1**

**bne $a1, $zero, loop**

**New Array Content:**

**7, 8, 9, 10, 11, 12, 1, 2, 3, 4, 5, 6**

**(swapping the first six elements with the last six)**

**Q5. (25 pts) Writing Assembly Language Functions**

1. (12 pts) Write a MIPS function named **count1s** to count the number of 1's in register **$a0** and put the result in register **$v0**. For example, if **$a0 = 0xffff0000** then the number of 1's will be **$v0 = 16**.

**count1s:**

**li $v0, 0 # initialize $v0 = 0**

**loop: andi $t0, $a0, 1 # $t1 = bit 0 of $a0**

**add $v0, $v0, $t0 # $v0 = count bit in $t0**

**srl $a0, $a0, 1**

**bne $a0, $zero, loop # loop until ($a0 == 0)**

**jr $ra # return to caller**

1. (13 pts) Write a function **gcd** to compute the greatest common divisor of two unsigned integers as follows:

**gcd(a,0) = a**

**gcd(a,b) = gcd(b,a%b) // a%b is the remainder of division**

For example: **gcd(8,12)=gcd(12,8)=gcd(8,4)=gcd(4,0)=4**.

The arguments are passed in registers **$a0** and **$a1** and the result is returned in **$v0**.

**gcd:**

**bne $a1, $0, else # branch if (b != 0) else**

**move $v0, $a0 # $v0 = a**

**jr $ra # return to caller**

**else: divu $a0, $a1 # divide a by b**

**move $a0, $a1 # $a0 = b**

**mfhi $a1 # $a1 = remainder a%b**

**j gcd # jump to gcd**