CDA 4205 Computer Architecture Exam I Practice

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

|  |  |  |
| --- | --- | --- |
| **Operation** | **Frequency** | **Clock cycles per instruction** |
| ALU | 35% | 3 |
| Load | 25% | 8 |
| Store | 20% | 5 |
| Branches | 20% | 4 |

1. Compute the average clock cycles per instruction.

**Average CPIa = 0.35\*3 + 0.25\*8 + 0.20\*5 + 0.20\*4 = 4.85**

1. Compute the percent of execution time spent by each class of instructions.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operation** | **Frequency** | **CPI** | **CPI \* Frequency %** | **Execution Time** |
| ALU | 35% | 3 | 1.05 | 1.05/4.85 = 21.65% |
| Load | 25% | 8 | 2.00 | 2.00/4.85 = 41.24% |
| Store | 20% | 5 | 1.00 | 1.00/4.85 = 20.62% |
| Branches | 20% | 4 | 0.80 | 0.80/4.85 = 16.49% |

1. A new execution unit has been designed and the new designed processor makes 70% of ALU operations take only **2** cycle to execute. The other 30% of ALU operations will still take **3** cycles to execute. Also, **85%** of the **load** instructions take only **3** cycles to execute, while the remaining 1**5%** of the **load** instructions take **8** cycles to execute per **load**. Each **store** instruction in the new designed processor takes **4** cycles to execute. Compute the new average cycles per instruction.

**Average CPIc = 0.70 \* 0.35 \* 2 + 0.30 \* 0.35 \* 3**

**+ 0.25 \* 0.85 \* 3 + 0.25 \* 0.15 \* 8**

**+ 0.20 \* 4 + 0.20 \* 4 = 3.3425**

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

**Speedup = 4.85 / 3.3425 = 1.45 (I-count & clock are the same)**

1. The designer decides to improve the clock speed in such a way to **double** 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 = 2 = (4.85 / 3.3425) \* (Clock Ratec/Clock Ratea)**

**Clock should be faster by 2/1.45 = 1.38 (38% faster)**

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

1. (3 pts) Assume that the instruction **j NEXT** is at address **0x00DAE05C**, and the label **NEXT** is at address **0x00DAFA28.** Then, the **26-bit immediate** stored in the jump instruction for the label **NEXT** is (**0x00DAFA28) >>2 = 0X36BE8A** .
2. (3 pts) Assume that the instruction **beq $t0, $t1, NEXT** is at address **0x04DAE05C**, and the label **NEXT** is at address **0x04DAFA28**. Then, the **16-bit immediate** stored in the branch instruction is **(0x04DAFA28 - 0x04DAE05C) >> 2 = 0x0673 .**
3. Consider the following data definitions:

**.data**

**var1: .byte 'Z', 1, 2, 5, 'B'**

**var2: .half -5, 0xDfCf**

**var3: .word 0x12345678, 0xff**

**.align 3**

**str1: .asciiz "My String\n"**

1. 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. Indicate which bytes are skipped or unused in the data segment.

**Data Segment Symbol Table**

**Unused.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Address** | **Byte 3** | **Byte 2** | **Byte 1** | **Byte 0** |  | **Label** | **Address** |
| 0x10010000 | **05** | **02** | **01** | **5A** |  | **var1** | **0x10010000** |
| 0x10010004 | **FF** | **FB** | **--** | **42** |  | **var2** | **0x10010006** |
| 0x10010008 | **--** | **--** | **DF** | **CF** |  | **var3** | **0x1001000C** |
| 0x1001000C | **12** | **34** | **56** | **78** |  | **str1** | **0x10010018** |
| 0x10010010 | **00** | **00** | **00** | **FF** |  |  |  |
| 0x10010014 | **--** | **--** | **--** | **--** |  |  |  |
| 0x10010018 | **53** | **20** | **79** | **4D** |  |  |  |
| 0x1001001C | **6E** | **69** | **72** | **74** |  |  |  |
| 0x10010020 | **--** | **00** | **0A** | **67** |  |  |  |
| 0x10010024 | -- | -- | -- | -- |  |  |  |
| 0x10010028 | -- | -- | -- | -- |  |  |  |
| 0x1001002C | -- | -- | -- | -- |  |  |  |

1. Construct a symbol table showing the symbols and their corresponding addresses in hexadecimal.
2. How many bytes are allocated in the data segment including the skipped bytes? **35** **bytes including the skipped bytes**.

**Q3. Floating-Point Number Representation**

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

*x* = **1 10000100 101 1011 0000 0000 1000 01112**

What is the decimal value of *x (accurate to 4 digits after decimal point)*?

**Solution:**

**Sign bit = 1 (negative)**

**Biased Exponent = 1000 0100 = 132**

**Exponent Value = 132 – 127 = +5**

**Value = -**

**= - = - 54.75**

1. Convert **-10.75** from decimal to the IEEE 754 single-precision floating point format. Show all your work for each step in the solution.

**Solution:**

**=**

**Normalize:**

**1010.11 (binary) =**

**Biased Exponent = 3 + 127 = 130 = 1000 0010 (binary)**

**IEEE 754 Single-Precision Representation:**

**1 10000010 010 1100 0000 0000 0000 0000**

**Q4. Tracing the Execution of Assembly Language Code**

The following code fragment processes two arrays and produces an important result in register **$v0**. Assume that each array consists of **N** words, the base addresses of the arrays **A** and **B** are stored in **$a0** and **$a1** respectively, and their sizes are stored in **$a2** and **$a3**, respectively. Describe what the above code does and what will be returned in register **$v0**.

**sll $a2, $a2, 2**

**sll $a3, $a3, 2**

**addu $v0, $zero, $zero**

**addu $t0, $zero, $zero**

**outer: addu $t4, $a0, $t0**

**lw $t4, 0($t4)**

**addu $t1, $zero, $zero**

**inner: addu $t3, $a1, $t1**

**lw $t3, 0($t3)**

**bne $t3, $t4, skip**

**addiu $v0, $v0, 1**

**skip: addiu $t1, $t1, 4**

**bne $t1, $a3, inner**

**addiu $t0, $t0, 4**

**bne $t0, $a2, outer**

**Solution:**

**This code compares every element in the first array against all elements of the second array. It counts the number of matching elements between the two arrays. $v0 will contain the count of the number of matching elements between the two arrays.**

**Q5. Writing Assembly Language Functions**

* + - * 1. Translate the following **if-else** statement into assembly language:

**if (($t0 >= '0') && ($t0 <= '9'))**

**{$t1 = $t0 – '0';}**

**else if (($t0 >= 'A') && ($t0 <= 'F'))**

**{$t1 = $t0+10-'A';}**

**else if (($t0 >= 'a') && ($t0 <= 'f'))**

**{$t1 = $t0+10-'a';}**

**Solution:**

**blt $t0, '0', else1**

**bgt $t0, '9', else1**

**addiu $t1, $t0, -48 # '0' = 48**

**j next**

**else1:**

**blt $t0, 'A', else2**

**bgt $t0, 'F', else2**

**addiu $t1, $t0, -55 # 10-'A' = 10-65=-55**

**j next**

**else2:**

**blt $t0, 'a', next**

**bgt $t0, 'f', next**

**addiu $t1, $t0, -87 # 10-'a' = 10-97=-87**

**next:**

* + - * 1. Translate the following loop into assembly language where a and b are integer arrays whose base addresses are in $a0 and $a1 respectively. The value of n is in $a2.

**for (i=0; i<n; i++) {**

**if (i > 2) {**

**a[i] = a[i-2] + a[i-1] + b[i];**

**}**

**else {**

**a[i] = b[i]**

**}**

**}**

**Solution:**

**li $t0, 0 # $t0 = i = 0**

**beq $a2, $0, skip # skip loop if n is zero**

**loop: lw $t1, 0($a1) # $t1 = b[i]**

**bgt $t0, 2, else # if (i>2) goto else**

**lw $t2, -8($a0) # $t2 = a[i-2]**

**lw $t3, -4($a0) # $t3 = a[i-1]**

**addu $t2, $t2, $t3 # $t2 = a[i-2]+a[i-1]**

**addu $t1, $t2, $t1 # $t1 = a[i-2]+a[i-1]+b[i]**

**else: sw $t1, 0($a0) # a[i] = $t1**

**addiu $a0, $a0, 4 # advance array a pointer**

**addiu $a1, $a1, 4 # advance array b pointer**

**addiu $t0, $t0, 1 # i++**

**bne $t0, $a2, loop**

**skip:**