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.
2. Compute the percent of execution time spent by each class of instructions.
3. 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.
4. What is the speedup factor by which the performance has improved in part **c**?
5. 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**?

**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 .
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 **.**
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. 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 |  |  |  |  |  |  |  |
| 0x10010004 |  |  |  |  |  |  |  |
| 0x10010008 |  |  |  |  |  |  |  |
| 0x1001000C |  |  |  |  |  |  |  |
| 0x10010010 |  |  |  |  |  |  |  |
| 0x10010014 |  |  |  |  |  |  |  |
| 0x10010018 |  |  |  |  |  |  |  |
| 0x1001001C |  |  |  |  |  |  |  |
| 0x10010020 |  |  |  |  |  |  |  |
| 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? .

**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*?

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.

**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**

**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';}**

* + - * 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];**

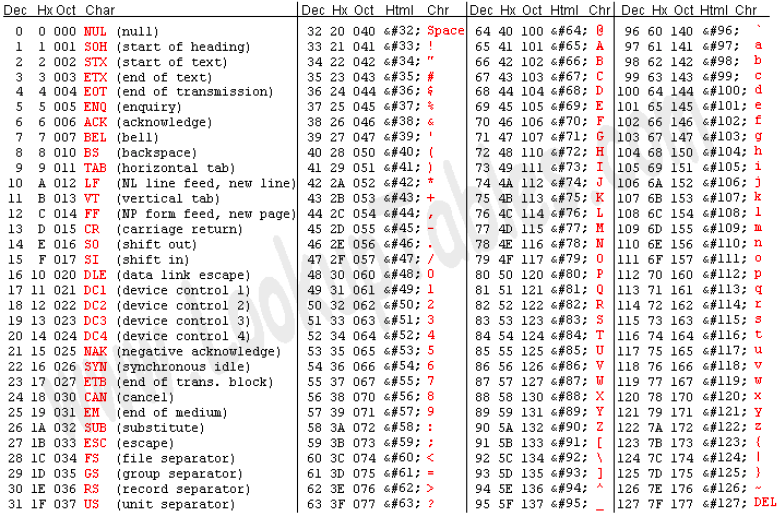
**}**

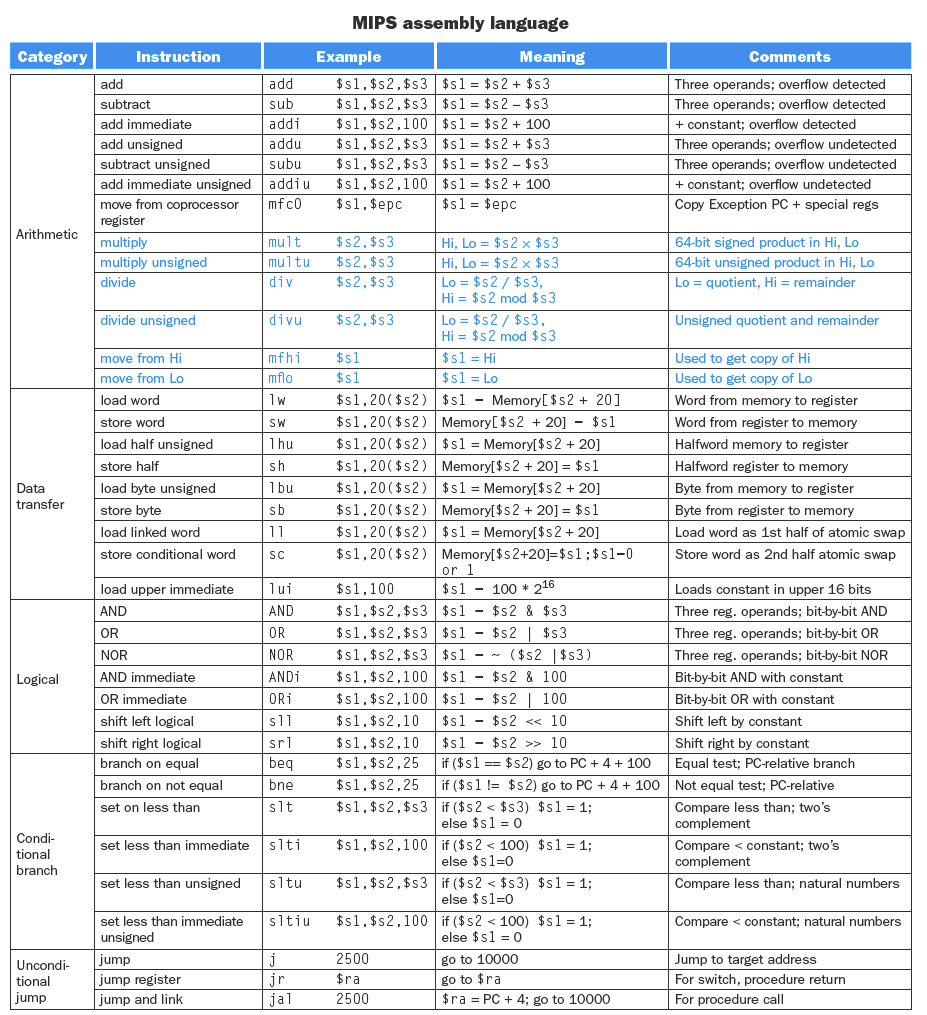
**else {**

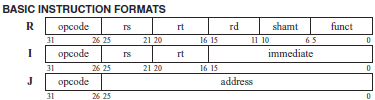
**a[i] = b[i]**

**}**

**}**

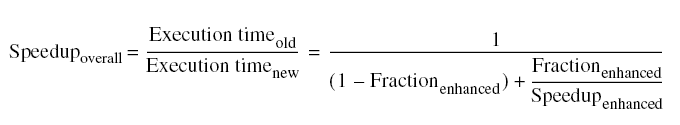


****

****

**CPU Time = Instruction Count × CPI × cycle time**

**Amdahl’s Law:**

****