1.5 [4] <§1.6> Consider three different processors P1, P2, and P3 executing the same instruction set. P1 has a 3 GHz clock rate and a CPI of 1.5. P2 has a 2.5 GHz clock rate and a CPI of 1.0. P3 has a 4.0 GHz clock rate and has a CPI of 2.2.

a. Which processor has the highest performance expressed in instructions per second?

b. If the processors each execute a program in 10 seconds, find the number of cycles and the number of instructions.

c. We are trying to reduce the execution time by 30% but this leads to an increase of 20% in the CPI. What clock rate should we have to get this time reduction?

Step 1 of 7

Consider three different processors P1, P2 and P3 which are executing on the same instruction set with the following clock rates and CPI's:

Processor	Clock Rate	CPI
P1	3 GHz	1.5
P2	2.5 GHz	1.0
P3	4 GHz	2.2

Table 1: Processors with clock rate and CPI.

Here, CPI denotes the clock cycles per instruction.

Step 2 of 7

a.

The performance of each processor is calculated by using the following formula:

$$Performance(P) = \frac{Clock\ Rate}{CPI} instructions\ per\ \text{sec}ond$$

For processor P1:

$$\begin{aligned} Performance(P_1) &= \frac{Clock\ Rate}{CPI} instructions\ per\ \text{sec}ond \\ &= \frac{3\times10^9}{1.5} instructions\ per\ \text{sec}ond \\ &= 2\times10^9\ instructions\ per\ \text{sec}ond \end{aligned}$$

Thus, the performance of processor P_1 is $\boxed{2\times10^9}$ $instructions\ per\ \text{sec}ond$.

For processor P2:

$$\begin{aligned} Performance(P_2) &= \frac{Clock\ Rate}{CPI} instructions\ per\ \text{sec}ond \\ &= \frac{2.5\times10^9}{1.0} instructions\ per\ \text{sec}ond \\ &= 2.5\times10^9\ instructions\ per\ \text{sec}ond \end{aligned}$$

Thus, the performance of processor P_2 is $\boxed{2.5\times10^9}$ $instructions\ per\ \text{sec}ond$.

For processor P3:

$$\begin{aligned} Performance(P_3) &= \frac{Clock\ Rate}{CPI} instructions\ per\ \text{sec}ond \\ &= \frac{4\times10^9}{2.2} instructions\ per\ \text{sec}ond \\ &= 1.81\times10^9\ instructions\ per\ \text{sec}ond \end{aligned}$$

Thus, the performance of processor P_3 is $\boxed{1.81\times10^9}$ $instructions\ per\ \text{sec}ond$.

As the performance is inversely proportional to the time, the processor with less time performs better. Thus, among the 3 processors, the least time is taken by the processor P_3 resulting in highest performance.

Thus, the processor P3 results in the highest performance expressed in instructions per second.

Step 3 of 7

b.

Consider the CPU time for executing each program is 10 seconds.

The number of cycles and number of instructions for each processor is calculated by using the following formulae:

$$Number\ of\ cycles(P) = Time \times Clock\ Rate$$

$$Number\ of\ instructions(P) = \frac{Number\ of\ cycles}{CPI}\ instructions$$

For processor P1:

$$\begin{aligned} Number\ of\ cycles(P_1) &= Time \times Clock\ Rate \\ &= 10 \times 3 \times 10^9 \\ &= 30 \times 10^9 \end{aligned}$$

Thus, the number of cycles for processor P_1 is $\boxed{30 \times 10^9}$.

$$\begin{aligned} Number\ of\ instructions(P_1) &= \frac{Number\ of\ cycles}{CPI}\ instructions \\ &= \frac{30 \times 10^9}{1.5} \\ &= 20 \times 10^9 \end{aligned}$$

Thus, the number of cycles for processor P_1 is $\boxed{20 \times 10^9}$.

For processor P2:

$$\begin{aligned} Number\ of\ cycles(P_2) &= Time \times Clock\ Rate \\ &= 10 \times 2.5 \times 10^9 \\ &= 25 \times 10^9 \end{aligned}$$

Step 4 of 7

Thus, the number of cycles for processor P_2 is $\boxed{25 \times 10^9}$.

$$\begin{aligned} Number\ of\ instructions(P_2) &= \frac{Number\ of\ cycles}{CPI}\ instructions \\ &= \frac{25 \times 10^9}{1.0} \\ &= 25 \times 10^9 \end{aligned}$$

Thus, the number of instructions for processor P_2 is $\boxed{25 \times 10^9}$.

For processor P3:

$$\begin{aligned} Number\ of\ cycles(P_3) &= Time \times Clock\ Rate \\ &= 10 \times 4 \times 10^9 \\ &= 40 \times 10^9 \end{aligned}$$

Thus, the number of cycles for processor P_3 is $\boxed{40 \times 10^9}$.

$$\begin{aligned} Number\ of\ instructions(P_3) &= \frac{Number\ of\ cycles}{CPI}\ instructions \\ &= \frac{40 \times 10^9}{2.2} \\ &= 18.18 \times 10^9 \end{aligned}$$

Thus, the number of instructions for processor P_3 is $\boxed{18.18 \times 10^9}$.

Step 5 of 7

c.

Consider the old CPU time is 10 seconds.

Now, calculate the new CPU time as follows:

$$CPU\ Time = \frac{(I \times CPI)}{clock\ rate}$$

The time is decreased by 30%.

$$t_1 = \frac{70 \times t}{100}$$
$$= 0.7t$$

So, the CPU time is 7s.

CPI is increased by 20%.

$$CPI = \frac{(120 \times CPI)}{100}$$
$$= 1.2 \times CPI$$

So, $CPI = 1.2 \times CPI$.

Calculate the clock rate to get the time reduction by using the following formula:

$$Clock\ rate = \frac{(Number\ of\ instruction \times CPI)}{Time}$$

Calculate number of cycles and number Of instructions of each processor by using the following formulas:

$$Number\ of\ cycles(P) = Time \times Clock\ Rate$$

$$Number\ of\ instructions(P) = \frac{Number\ of\ cycles}{CPI}\ instructions$$

For processor P1:

$$CPI = 1.2 \times CPI$$
$$= 1.2 \times 1.5$$
$$= 1.8$$

$$CPI = \boxed{1.8}$$

$$Number\ of\ cycles(P_1) = Time \times Clock\ Rate$$
$$= 10 \times 3 \times 10^9$$
$$= 30 \times 10^9$$

Thus, the number of cycles for processor P_1 is $\boxed{30 \times 10^9}$.

$$Number\ of\ instructions(P_1) = \frac{Number\ of\ cycles}{CPI}\ instructions$$
$$= \frac{30 \times 10^9}{1.5}$$
$$= 20 \times 10^9$$

Thus, the number of instructions for processor P_1 is $\boxed{20 \times 10^9}$.

$$Clock\ rate(P1) = \frac{(Number\ of\ instructions \times CPI)}{Time}$$
$$= \frac{(20 \times 10^9 \times 1.8)}{7}$$
$$= \frac{36000000000}{7}$$
$$= 5.14\,GHz$$

Thus, the Clock rate for processor P1 is $\boxed{5.14\,\text{GHz}}$.

Step 6 of 7

For processor P2:

$$\begin{aligned} CPI &= 1.2 \times CPI \\ &= 1.2 \times 1.0 \\ &= 1.2 \end{aligned}$$

$CPI = \boxed{1.2}$

$$\begin{aligned} Number\ of\ cycles(P_2) &= Time \times Clock\ Rate \\ &= 10 \times 2.5 \times 10^9 \\ &= 25 \times 10^9 \end{aligned}$$

Thus, the number of cycles for processor P_2 is $\boxed{25 \times 10^9}$.

$$\begin{aligned} Number\ of\ instructions(P_2) &= \frac{Number\ of\ cycles}{CPI}\ instructions \\ &= \frac{25 \times 10^9}{1.0} \\ &= 25 \times 10^9 \end{aligned}$$

Thus, the number of instructions for processor P_2 is $\boxed{25 \times 10^9}$.

$$\begin{aligned} Clock\ rate &= \frac{(Number\ of\ instruction \times CPI)}{Time} \\ &= \frac{(25 \times 10^9 \times 1.2)}{7} \\ &= \frac{300000000000}{7} \\ &= 4.28 GHz \end{aligned}$$

Thus, the Clock rate for processor P2 is $\boxed{4.28 \text{GHz}}$.

Step 7 of 7

For processor P3:

$$\begin{aligned} CPI &= 1.2 \times CPI \\ &= 1.2 \times 2.2 \\ &= 2.64 \end{aligned}$$

$CPI = \boxed{2.64}$

$$\begin{aligned} Number\ of\ cycles(P_3) &= Time \times Clock\ Rate \\ &= 10 \times 4 \times 10^9 \\ &= 40 \times 10^9 \end{aligned}$$

Thus, the number of cycles for processor P_3 is $\boxed{40 \times 10^9}$.

$$\begin{aligned} Number\ of\ instructions(P_3) &= \frac{Number\ of\ cycles}{CPI}\ instructions \\ &= \frac{40 \times 10^9}{2.2} \\ &= 18.18 \times 10^9 \end{aligned}$$

Thus, the number of instructions for processor P_3 is $\boxed{18.18 \times 10^9}$.

$$\begin{aligned} Clock\ rate &= \frac{(Number\ of\ instruction \times CPI)}{Time} \\ &= \frac{(18.18 \times 10^9 \times 2.64)}{7} \\ &= \frac{47995200000}{7} \\ &= 6.85 GHz \end{aligned}$$

Thus, the clock rate for processor P_3 is $\boxed{6.85 \text{GHz}}$.

1.7 [15] <§1.6> Compilers can have a profound impact on the performance of an application. Assume that for a program, compiler A results in a dynamic instruction count of 1.0E9 and has an execution time of 1.1 s, while compiler B results in a dynamic instruction count of 1.2E9 and an execution time of 1.5 s.

a. Find the average CPI for each program given that the processor has a clock cycle time of 1 ns.

b. Assume the compiled programs run on two different processors. If the execution times on the two processors are the same, how much faster is the clock of the processor running compiler A's code versus the clock of the processor running compiler B's code?

c. A new compiler is developed that uses only 6.0E8 instructions and has an average CPI of 1.1. What is the speedup of using this new compiler versus using compiler A or B on the original processor?

Step 1 of 5

Consider the two compilers A and B. The values of some factors such as execution time and instruction count for the two compilers are as follows:

Compiler	Execution Time	Instruction Count
A	1.1s	1×10^{9}
B	1.5s	1.2×10^{9}

The clock cycle time of the processor is $1\ ns\left(10^{-9}\ \text{sec}\right)$.

Step 2 of 5

a.

To calculate the CPI for each compiler, need to use the following formula:

$$CPI = \frac{CPU\ execution\ time}{number\ of\ instructions \times Clock\ cycle\ time}$$

For compiler A:

CPI for compiler A when the Cycle Time is 1 ns is as follows:

$$\begin{aligned} CPI &= \frac{CPU\ execution\ time}{number\ of\ instructions \times Clock\ cycle\ time} \\ &= \frac{1.1}{1\times10^{9}\times10^{-9}} \\ &= 1.1 \end{aligned}$$

Thus, the average CPI for compiler A is 1.1 .

For compiler B:

CPI for compiler B when the Cycle Time is 1 ns is as follows:

$$\begin{aligned} CPI &= \frac{CPU\ execution\ time}{number\ of\ instructions \times Clock\ cycle\ time} \\ &= \frac{1.5}{1.2\times10^{9}\times10^{-9}} \\ &= 1.25 \end{aligned}$$

Thus, the average CPI for compiler B is 1.25 .

Step 3 of 5

b.

Consider that the execution time on the two different processors is same.

The formula to calculate the execution time of a processor is as follows:

$$\begin{aligned} Execution\ time\ of\ A &= Instructions\ for\ a\ program \times Average\ CPI \times Clock\ cycle\ time\ of\ A \\ &= 1\times10^{9}\times1.1 \times Clock\ cycle\ time\ of\ A \\ Execution\ time\ of\ B &= Instructions\ for\ a\ program \times Average\ CPI \times Clock\ cycle\ time\ of\ B \\ &= 1.2\times10^{9}\times1.25 \times Clock\ cycle\ time\ of\ B \end{aligned}$$

If the execution times on the two processors are same, then

$$1\times10^{9}\times1.1 \times \text{Clock cycle time of A} = 1.2\times10^{9}\times1.25 \times \text{Clock cycle time of B}$$

$$\Rightarrow \text{Clock cycle time of A} = \frac{1.2\times10^{9}\times1.25}{1\times10^{9}\times1.1} \times \text{Clock cycle time of B}$$

$$\Rightarrow \text{Clock cycle time of A} = \frac{1.2\times1.25}{1\times1.1} \times \text{Clock cycle time of B}$$

$$\Rightarrow \text{Clock cycle time of A} = \frac{1.5}{1.1} \times \text{Clock cycle time of B}$$

$$\Rightarrow \text{Clock cycle time of A} = 1.364\ (\text{Clock cycle time of B})$$

Thus, the clock of the processor running compiler A code is 1.364 times faster than the clock of the processor running compiler B's code.

Step 4 of 5

c.

Consider a new compiler.

The number of instructions for new compiler= 6×10^8

= 0.6×10^9

The average CPI for new compiler= 1.1

$$\begin{aligned} CPU\ execution\ time &= Instructions\ for\ a\ program \times Average\ CPI \times Clock\ cycle\ time \\ &= 0.6 \times 10^9 \times 1.1 \times 10^{-9}\ seconds \\ &= 0.6 \times 1.1\ seconds \\ &= 0.66\ seconds \end{aligned}$$

For compiler A:

Execution time for compiler A = 1.1 seconds.

$$\begin{aligned} Speed\ up\ of\ new\ compiler\ using\ compiler\ A &= \frac{Execution\ time\ of\ compiler\ A}{Execution\ time\ of\ new\ compiler} \\ &= \frac{1.1}{0.66} \\ &= 1.67 \end{aligned}$$

Step 5 of 5

Thus, speed up of using new compiler with compiler A is 1.67 .

For compiler B:

Execution time for compiler B = 1.5 seconds.

$$\begin{aligned} Speed\ up\ of\ new\ compiler\ using\ compiler\ B &= \frac{Execution\ time\ of\ compiler\ B}{Execution\ time\ of\ new\ compiler} \\ &= \frac{1.5}{0.66} \\ &= 2.27 \end{aligned}$$

Thus, speed up of using new compiler with compiler B is 2.27 .

2.4 [5] <§§2.2, 2.3> For the MIPS assembly instructions below, what is the corresponding C statement? Assume that the variables `f`, `g`, `h`, `i`, and `j` are assigned to registers `$s0`, `$s1`, `$s2`, `$s3`, and `$s4`, respectively. Assume that the base address of the arrays A and B are in registers `$s6` and `$s7`, respectively.

```
sll  $t0, $s0, 2     # $t0 = f * 4
add  $t0, $s6, $t0   # $t0 = &A[f]
sll  $t1, $s1, 2     # $t1 = g * 4
add  $t1, $s7, $t1   # $t1 = &B[g]
lw   $s0, 0($t0)     # f = A[f]
addi $t2, $t0, 4
lw   $t0, 0($t2)
add  $t0, $t0, $s0
sw   $t0, 0($t1)
```

Step 1 of 1

To convert the given MIPS assembly code to the respective C code, It is said to suppose that *$s0, $s1, $s2, $s3, $s4* points are assigned to *f, g, h, i, j* respectively. It is also said that base address of the arrays *A, B* are stored in the registers *$s6, $s7*. There are the following steps that show the step by step process to translate the given MIPS assemble code:

1. ***sll* $t0, $s0, 2**

sll means shift logically left. Moving logically left by 1 bit means it multiply by 2; the given code shifting logically by four, it means multiplication by 4.

According to the instruction multiply the contents of *$s0* with 4 and *$s0* points assigned to *f*. This means, it should be f **4* and the result has to be stored in register *$t0*.

Thus the first instruction in C language can be written as: *$t0=f*4*.

2. **add $t0, $s6, $t0**

It means to add the content of register $t0, $s6 and store the value into register *$t0*. It was already given that the *$s6* contains the base address of array *A*.

So what this basically does is adding the value of *$t0* to the base address of array *A* which we have assigned in step 1.

So the second instruction in C language can be written as: *$t0=&A[f]*

3. **sll $t1, $s1, 2**

Process is same as step 1 but this time the result will be stored in register *$t1* and value in the register *$s1* will be multiplied by four. It was already given that *$s1* contains variable *g*.

So this instruction in C language can be written as: *$t1=g*4*

4. **add $t1, $s1, $t1**

This instruction is written for adding the value of register *$s1* and value of $t1. Since the value in register *$s1* is the base address of array *B*.

So this instruction in C language can be written as: *$t1=&B[g]*

5. **lw $s0, 0($t0)**

The data transfer instruction that copies data from memory to a register is traditionally called *load*. The format of the load instruction is the name of the operation followed by the register to be loaded, then a constant and register used to access memory.

The sum of the constant portion of the instruction and the contents of the second register forms the memory address. The actual MIPS name for this instruction is lw, standing for *load word*.

So this instruction in C language can be written as: *f = A[f]* so register *$s0=A[f]*.

6. **addi $t2, $t0, 4**

Here *addi* is an MIPS instruction which means add immediate. This instruction is represents to add 4 to the contents of the register *$t0* and store the result in the *$t2* register. From the step 2 it is shown that *$t0=&A[f]*.

So this instruction in C language can be written as: *$t2 = &A[f+1]*. Here it is considered that the word to be of length *4*.

7. **lw $t0, 0($t2)**

From step 6 it is shown that *$t2 = &A[f+1]* and this MIPS instruction means to load the contents of memory address pointed by *$2* to the register *$t0*.

So this instruction in C language can be written as: *$t0 =A[f+1]*

8. **add $t0, $t0, $s0**

From step 5 and 7 it is shown that *$s0=A[f]* and *$t0 =A[f+1]* and this instruction is telling to add them both.

So this instruction in C language can be written as: *$t0=A[f] + A[f+1]*.

9. **sw $t0, 0($t1)**

The sw used to Store Word, hence it is telling that the value in the register *$t0* to the location pointed by content of register *$t1*, it is already given that *$t1* is the address of *B[g]* from step 4.

So this instruction in C language can be written as *B[g] = A[f] +A[f=1]*

2.5 [5] <§§2.2, 2.3> For the MIPS assembly instructions in Exercise 2.4, rewrite the assembly code to minimize the number if MIPS instructions (if possible) needed to carry out the same function.

Step 1 of 2

Consider the following MIPS instructions:

```
sll $t0, $s0, 2 # $t0 = f * 4
add $t0, $s6, $t0 # $t0 = &A[f]
sll $t1, $s1, 2 # $t1 = g * 4
add $t1, $s7, $t1 # $t1 = &B[g]
lw $s0, 0($t0) # f = A[f]
addi $t2, $t0, 4 # $t2 = &A[f+1]
lw $t0, 0($t2) # $t0 = A[f+1]
add $t0, $t0, $s0 # $t0 = A[f+1]+A[f]
sw $t0, 0($t1) # B[g] = $t0
```

Step 2 of 2

Minimizing the number of instructions:

The number of MIPS instructions can be reduced by 1. So, the total number of MIPS instructions after minimization is 8.

```
sll $t0, $s0, 2 # $t0 = f * 4
add $t0, $s6, $t0 # $t0 = &A[f]
sll $t1, $s1, 2 # $t1 = g * 4
add $t1, $s7, $t1 # $t1 = &B[g]
lw $s0, 0($t0) # f = A[f]
lw $t0, 4($t0) # $t0 = A[f+1]
add $t0, $t0, $s0 # $t0 = A[f+1] + A[f]
sw $t0, 0($t1) # B[g] = $t0
```

Explanation:

- The instruction addi $t2, $t0, 4 can be eliminated from the considered sequence of instructions. This is because, it adds the value 4 to the address of $t0 and stores the result in $t2. The value of $t2 is fetched and then loaded in the register $t0.
- During the minimization of instructions, the value of $t0 at the memory location 4 is directly fetched and loaded in the register $t0.

2.7 [5] <§2.3> Show how the value `0xabcdef12` would be arranged in memory of a little-endian and a big-endian machine. Assume the data is stored starting at address 0.

Step 1 of 2

Consider the following hexadecimal number:

0xabcdef12

Assume the data is stored starting at address 0.

Little-endian: Little-endian describes the order in which sequences of bytes are stored in computer memory. It starts with Least Significant Bit (LSB) is stored first.

The given hexadecimal number arranged in memory of a little-endian as shown below:

Address	Data
12	ab
8	cd
4	ef
0	12

Step 2 of 2

Big-endian: Big-endian describes the order in which sequences of bytes are stored in computer memory. It starts with Most Significant Bit (MSB) is stored first.

The given hexadecimal number arranged in memory of a big-endian as shown below:

Address	Data
12	12
8	ef
4	cd
0	ab

2.9 [5] <§§2.2, 2.3> Translate the following C code to MIPS. Assume that the variables `f`, `g`, `h`, `i`, and `j` are assigned to registers `$s0`, `$s1`, `$s2`, `$s3`, and `$s4`, respectively. Assume that the base address of the arrays `A` and `B` are in registers `$s6` and `$s7`, respectively. Assume that the elements of the arrays `A` and `B` are 4-byte words:

```
B[8] = A[i] + A[j];
```

Step 1 of 4

Consider the following C statement:

B[8] = A[i] + A[j];

Step 2 of 4

Translation into MIPS:

- Assume the variables i and j are assigned to the registers $s3 and $s4 respectively.
- $s6 and $s7 stores the base address of the arrays A and B respectively.
- Assume that the elements of the arrays are integers, represented as a 4-byte word.

```
sll $t0, $s3, 2 # $t0=$s3<<2; Multiply index i by 4
add $t0, $t0, $s6 # $t0=$t0+$s6; Place A[i] into $t0
lw $t0, 0($t0) # $t0=A[i]; load value in A[i] to $t0
sll $t1, $s4, 2 # $t1=$s4<<2; Multiply index j by 4
add $t1, $t1, $s6 # $t1=$t1+$s6; Place A[j] into $t1
lw $t1, 0($t1) # $t1=A[j]; load value in A[j] to $t1
add $t2, $t0, $t1 # $t2=$t0+$t1; add A[i],A[j].
# Store in $t2
sw $t2, 32($s7) # B[8]=A[i]+A[j]; Store value in $t2
# to the 8th(8*4=32)location in $s7.
```

2.12 Assume that registers `$s0` and `$s1` hold the values `0x80000000` and `0xD0000000`, respectively.

2.12.1 [5] <§2.4> What is the value of `$t0` for the following assembly code?

```
add $t0, $s0, $s1
```

2.12.2 [5] <§2.4> Is the result in `$t0` the desired result, or has there been overflow?

Step 1 of 1

Consider the MIPS architecture registers contain the following values.

Register $s0 = $0x80000000_{hex}$

Register $s1 = 0xD0000000_{hex}$

Convert register's hexadecimal values into binary values by using below table.

The hexadecimal-binary conversion table:

Hexadecimal	Binary	Hexadecimal	Binary
0_{hex}	0000_{two}	8_{hex}	1000_{two}
1_{hex}	0001_{two}	9_{hex}	1001_{two}
2_{hex}	0010_{two}	A_{hex}	1010_{two}
3_{hex}	0011_{two}	B_{hex}	1011_{two}
4_{hex}	0100_{two}	C_{hex}	1100_{two}
5_{hex}	0101_{two}	D_{hex}	1101_{two}
6_{hex}	0110_{two}	E_{hex}	1110_{two}
7_{hex}	0111_{two}	F_{hex}	1111_{two}

```
$s0 = 0x80000000 = 1000 0000 0000 0000 0000 0000 0000 0000
$s1 = 0xD0000000 = 1101 0000 0000 0000 0000 0000 0000 0000
```

The sign extension of both the operands is negative.

The assembly code: add $t0, $s0, $s1

Addition of both of the binary numbers that is the value which is stored in the register $t0:

Addition of $s0, $s1 :

```
 1000 0000 0000 0000 0000 0000 0000 0000
 1101 0000 0000 0000 0000 0000 0000 0000
-----------------------------------------
10101 0000 0000 0000 0000 0000 0000 0000
```

Convert this binary value into hexa-decimal by using above hexadecimal-binary conversion table.

$$\underbrace{10101}_{1\ 5}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0}\ \underbrace{0000}_{0} = 150000000_{hex}$$

An overflow occurs because the result is greater than 32 bits.

Thus, the value of $t0 is 50000000 by truncating the overflow.

2.14 [5] <§§2.2, 2.5> Provide the type and assembly language instruction for the following binary value: 0000 0010 0001 0000 1000 0000 0010 0000_{two}

Step 1 of 1

Consider the following binary value:

$(00000010000100001000000000100000)_{two}$

The type and assembly language instruction for the above binary value are shown below:

• This could be R-Type Instruction, if it is divided in the following bit patterns:

000000	10000	10000	10000	00000	100000
Opcode(6 bits)	RS(5 bits)	RD(5 bits)	RT(5 bits)	Shamt(5 bits)	Funct(6 bits)

• Above instruction set describes add (the combination of opcode 0 and Funct 32 is reserved for add) instruction of assembly language.

• The instruction is shown below as per the values (16 on each) of different registers (RS, RD and RT):

add $\$s0,\$s0,\$s0$

The value 16 is reserved for the register $s0.

2.15 [5] <§§2.2, 2.5> Provide the type and hexadecimal representation of following instruction: `sw $t1, 32($t2)`

Step 1 of 1

Consider the MIPS instruction:

sw $t1, 32($t2)

Based on the MIPS instruction encoding (Refer FIGURE 2.5 in text book):

sw(store word) Opcode =43= 101011

Immediate address (16-bit address) =32=0000000000100000

The MIPS instruction is an I-type instruction.

Based on the MIPS register conventions (Refer FIGURE 2.14 in text book):

The value of destination register $t1 = 9 =01001

The value of source register $t2= 10=01010

I-type instruction format:

	Opcode	**rs**	**rt**	Immediate **address**
Binary	101011	01010	01001	0000000000100000

So, the binary representation:

1010 1101 0100 1001 0000 0000 0010 0000 $_{two}$

Convert this binary value to hexadecimal representation by using below table:

Binary	Hexadecimal	Binary	Hexadecimal
0000_{two}	0_{hex}	1000_{two}	8_{hex}
0001_{two}	1_{hex}	1001_{two}	9_{hex}
0010_{two}	2_{hex}	1010_{two}	A_{hex}
0011_{two}	3_{hex}	1011_{two}	B_{hex}
0100_{two}	4_{hex}	1100_{two}	C_{hex}
0101_{two}	5_{hex}	1101_{two}	D_{hex}
0110_{two}	6_{hex}	1110_{two}	E_{hex}
0111_{two}	7_{hex}	1111_{two}	F_{hex}

Then,

1010 1101 0100 1001 0000 0000 0010 0000

| | | | | | | | |

A D 4 9 0 0 2 0

So, the hexadecimal representation of the instruction (sw St1,32(St2)**) = 0xAD490020**

2.17 [5] <§2.5> Provide the type, assembly language instruction, and binary representation of instruction described by the following MIPS fields:

```
op=0x23, rs=1, rt=2, const=0x4
```

Step 1 of 2

The **type** of the instruction presented in the problem 2.17 is of **I-Type.**

Assembly language instruction for the above values is shown below:

lw Sv0, 4(Sat)

Explanation: Value 2 represents the return value register Sv0 and the value 1 represents the register Sat (reserved for the assembler).

• The opcode 0X23 is in hexadecimal form. It's binary for is $(100011)_2$ and decimal form is 35.

• It is the opcode of the operation lw. Its mean load the value 4 multiplied by the value in assembler register Sat into the register Sv0.

Step 2 of 2

Binary Representation of the instruction is shown below:

100011	00001	00010	0000000000000100
Opcode	RS	RT	Constant

Here constant represents 16bits because it is an immediate field. Therefore constant is 0000 0000 0000 0100

2.16 [5] <§2.5> Provide the type, assembly language instruction, and binary representation of instruction described by the following MIPS fields:

```
op=0, rs=3, rt=2, rd=3, shamt=0, funct=34
```

Step 1 of 3

Consider the various instructions of MIPS fields:

op=0, rs=3, rt=2, rd=3, shamt=0, funct=34

Based on the MIPS instruction encoding (Refer FIGURE 2.5 in text book):

• The opcode (op) value is "0"

• The "funct" field is used to decide the variant of the operation (32-addition or 34-subtract). Here the "funct" field is 34. So, the instruction is sub (subtract) and the type of instruction format is R-type.

So, the MIPS fields contain R-type instruction format.

Step 2 of 3

Based on the MIPS register conventions table (Refer FIGURE 2.14 in text book):

• rs=3 contains register $v1

• rt=2 contains register $v0

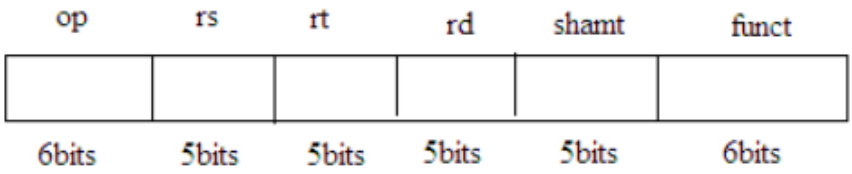
• rd=3 contains register $v1

• The "funct" field is 34. So, should use the instruction is "sub(subtract)"

So, the assembly language instruction is "sub $v1, $v1, $v0".

Step 3 of 3

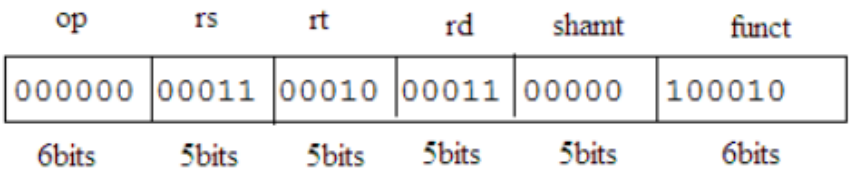
The fields of R-type Instruction format:



Convert decimal values of MIPS fields into binary values:

- opcode (op) = 0 = 000000
- rs =3 =00011
- rt =2 =00010
- rd =3 =00011
- shamt =0 =000000
- funct =34 =100010

After filling the values in R-type instruction format:



Therefore, the binary representaion of Instruction:

000000 00011 00010 00011 00000 100010

3.1 [5] <§3.2> What is 5ED4 − 07A4 when these values represent unsigned 16-bit hexadecimal numbers? The result should be written in hexadecimal. Show your work.

Step 1 of 1

Unsigned 16 bit hexadecimal number subtraction:

Hexadecimal number system is defined as the base 16 number system digits 0-9 and alphabets A to F are used. Alphabets A to F represent 10 to 15 respectively.

The numbers 5ED4 and 07A4 are unsigned hexadecimal numbers hence they will b subtracted simply. No carry is involved here.

Starting from the right side,

- 4 will be subtracted from 4 which will leave 0 at 1's place.
- A will be subtracted from D which will leave 3 at 16's place.
- 7 will be subtracted from E which will leave 7 at 256's place
- 0 is subtracted from 5 which will leave 5 at 4096's place.

$$\begin{array}{r} 5ED4 \\ 07A4 \\ \hline 5730 \end{array}$$

Hence, $5ED4 - 07A4 = 5730$

3.2 [5] <§3.2> What is 5ED4 − 07A4 when these values represent signed 16-bit hexadecimal numbers stored in sign-magnitude format? The result should be written in hexadecimal. Show your work.