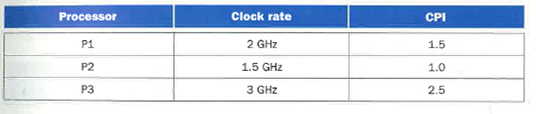
**Question1.3.1**

Consider three different processors P1, P2, and P3 executing the same instruction set with the clock rates and CPIs given in the following table.



[5] <1.4> Which processor has the highest performance?

**Answer**

**Step 1**

|  |  |  |
| --- | --- | --- |
| Processor | Clock Rate | CPI |
| P1 | 2 GHz | 1.5 |
| P2 | 1.5GHz | 1.0 |
| P3 | 3 GHz | 2.5 |

Let the number of instructions be I

CPU Time = (Instructions Count) \* (CPI) / (Clock Rate)

**Step 2**

For P1, CPU Time = (I \* 1.5) / (2 \* 109)

= 0.75 \* I \* 10-9

For P2, CPU Time = (I \* 1.0) / (1.5 \* 109)

= 0.667 \* I \* 10-9

For P3, CPU Time = (I \* 2.5) / (3 \* 109)

= 0.833 \* I \* 10-9

**Step 3**

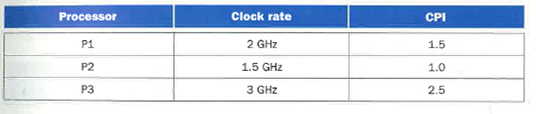
Now performance is inversely proportional to CPU Time

Therefore the processor taking least time will perform better.

As P2 is taking the least time, it will perform better.

**Question1.3.2**

Consider three different processors P1, P2, and P3 executing the same instruction set with the clock rates and CPIs given in the following table.



[10] <1.4> If the processors each execute a program in 10 seconds, find the number of cycles and the number of instructions.

**Answer**

**Step 1**

|  |  |  |
| --- | --- | --- |
| Processor | Clock Rate | CPI |
| P1 | 2 GHz | 1.5 |
| P2 | 1.5GHz | 1.0 |
| P3 | 3 GHz | 2.5 |

Let the number of instructions be I

CPU Time = (Instructions Count) \* (CPI) / (Clock Rate)

**Step 2**

For P1, CPU Time = (I \* 1.5) / (2 \* 109)

Given CPU time = 10 seconds

Therefore 10 = (I \* 1.5) / (2 \* 109)

Hence I = (20 \* 109)/ 1.5

Number of instructions I = 13.333 \* 109

Number of cycles = CPI \* I

= 1.5 \* 13.333 \* 109

Number of cycles = 20 \* 109 seconds

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

For P2, CPU Time = (I \* 1.0) / (1.5 \* 109)

Given CPU time = 10 seconds

Therefore 10 = (I \* 1.0) / (1.5 \* 109)

Hence I = 15 \* 109

Number of instructions I = 15 \* 109

Number of cycles = CPI \* I

= 1.0 \* 15 \* 109

Number of cycles = 15 \* 109 seconds

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

For P3, CPU Time = (I \* 2.5) / (3 \* 109)

Given CPU time = 10 seconds

Therefore 10 = (I \* 2.5) / (3 \* 109)

Hence I = (30 \* 109)/ 2.5

Number of instructions I = 12 \* 109

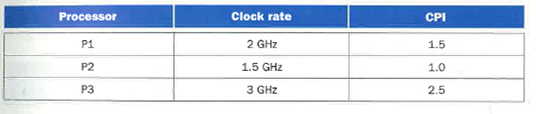
Number of cycles = CPI \* I

= 2.5 \* 12 \* 109

Number of cycles = 30 \* 109 seconds

**Question1.3.3**

Consider three different processors P1, P2, and P3 executing the same instruction set with the clock rates and CPIs given in the following table.



[10] <1.4> We are trying to reduce the 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?

**Answer**

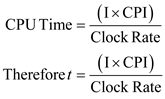
**Step 1**

**Calculating Clock Rate**

Consider tree different processors P1, P2, and P3 executing the same instruction set with the clock rate and CPI’s given in the following table:

|  |  |  |
| --- | --- | --- |
| **Processor** | **Clock Rate** | **CPI** |
| P1 | 2 GHz | 1.5 |
| P2 | 1.5 GHz | 1.0 |
| P3 | 3 GHz | 2.5 |

Now, calculate the CPU time:

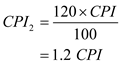


**Step 2**

Now, time (*t*) percent always 100%,*t* has decreased by 30%. So, time percent is 70%.



And also CPI percent always 100%, CPI is increased by 20%.So, CPI percent is 120%.

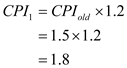


**Step 3**

Now, Calculate the **New CPI** for each processor.



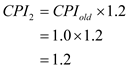
**New CPI for processor P1:**





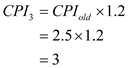
**Step 4**

**New CPI for processor P2:**





**New CPI for processor P3:**



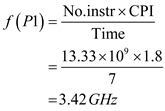


**Step 5**

Finally, calculate the clock rate by using the following formula:

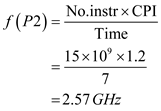


**Clock rate for processor P1:**



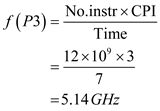


**Clock rate for processor P2:**





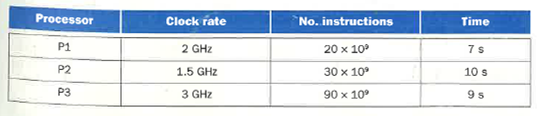
**Clock rate for processor P3:**





**Question1.3.4**

For problems below, use the information in the following table.



[10] <1.4> Find the IPC (instructions per cycle) for each processor.

**Answer**

**Step 1**

|  |  |  |  |
| --- | --- | --- | --- |
| Processor | Clock Rate | No. instructions | Time |
| P1 | 2 GHz | 20 \* 109 | 7s |
| P2 | 1.5GHz | 30 \* 109 | 10 s |
| P3 | 3 GHz | 90 \* 109 | 9 s |

CPU Time = (Instructions Count) \* (CPI) / (Clock Rate)

**Step 2**

For P1, CPU Time = (20 \* 109 \* CPI) /(2 \* 109)

7 = 10\* CPI

CPI = 0.7

Therefore Instructions per cycle IPC = (1/CPI)

= 1/0.7

=1.429

**Step 3**

For P2, CPU Time = (30 \* 109 \* CPI) /(1.5 \* 109)

10 = 20\* CPI

CPI = 0.5

Therefore Instructions per cycle IPC = (1/CPI)

= 1/0.5

= 2

**Step 4**

For P3, CPU Time = (90 \* 109 \* CPI) / (3 \* 109)

9 = 30\* CPI

CPI = 0.3

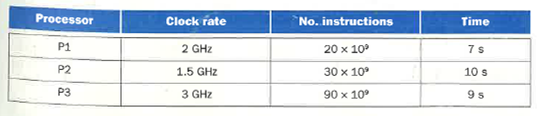
Therefore Instructions per cycle IPC = (1/CPI)

= 1/0.3

= 3.333

**Question1.3.5**

For problems below, use the information in the following table.



[5] <1.4> Find the clock rate for P2 that reduces its execution time to that of P1.

**Answer**

**Step 1**

728-1.3-5E SA Code: 6456

SR Code: 4460

|  |  |  |  |
| --- | --- | --- | --- |
| Processor | Clock Rate | No. instructions | Time |
| P1 | 2 GHz | 20 \* 109 | 7s |
| P2 | 1.5GHz | 30 \* 109 | 10 s |
| P3 | 3 GHz | 90 \* 109 | 9 s |

CPU Time = (Instructions Count) \* (CPI) / (Clock Rate)

**Step 2**

For P2, CPU Time = (30 \* 109 \* CPI) /(1.5 \* 109)

10 = 20 \* CPI

CPI =0.5

Now if the Execution time of P2 is reduced to that of P1,

Then 7s = (30 \* 109 \* CPI) / (clock rate)

Therefore clock rate = (30 \* 109 \* 0.5)/7

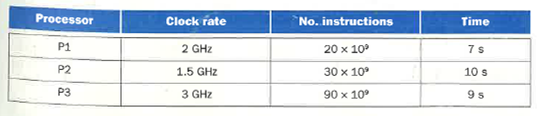
= (15/7)\* 109

= 2.142 \* 109

New clock rate = 2.142 GHz

**Question1.3.6**

For problems below, use the information in the following table.



[5] <1.4> Find the number of instructions for P2 that reduces its execution time to that of P3.

**Answer**

**Step 1**

728-1.3-6E SA Code: 6456

SR Code: 4460

|  |  |  |  |
| --- | --- | --- | --- |
| Processor | Clock Rate | No. instructions | Time |
| P1 | 2 GHz | 20 \* 109 | 7s |
| P2 | 1.5GHz | 30 \* 109 | 10 s |
| P3 | 3 GHz | 90 \* 109 | 9 s |

CPU Time = (Instructions Count) \* (CPI) / (Clock Rate)

**Step 2**

For P2, CPU Time = (30 \* 109 \* CPI) /(1.5 \* 109)

10 = 20 \* CPI

CPI =0.5

Now if the Execution time of P2 is reduced to that of P3,

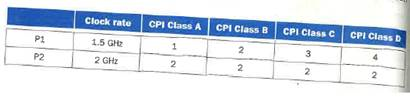
9 = (I \* CPI) / (1.5 \* 109)

9 = (I \* 0.5) / (1.5 \* 109)

Therefore the number of instructions should be I = 27 \* 109

**Question1.4.1**

Consider two different implementations of the same instruction set architecture. There are four classes of instructions, A, B, C, and D. The clock rate and CPI of each implementation are given in the following table.



[10] <1.4> Given a program with 106 instructions divided into classes as follows: 10% class A, 20% class B, 50% class C, and 20% class D, which implementation is faster?

**Answer**

**Step 1**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Clock Rate | CPI class A | CPI class B | CPI class C | CPI class D |
| P1 | 1.5GHz | 1 | 2 | 3 | 4 |
| P2 | 2 GHz | 2 | 2 | 2 | 2 |

**Step 2**

For P1

CPU Time = ∑ (I \* CPI)/ Clock rate

= 106 \* [(0.1 \* 1) + (0.2 \* 2) + (0.5 \* 3) + (0.2 \* 4)] / (1.5 \* 109)

= [0.1 + 0.4 + 1.5 + 0.8] \* 10-3 /1.5

= (2.8/1.5) milliseconds [since 1 millisecond = 10-3 seconds]

= 1.8667 milliseconds

**Step 3**

For P2

CPU Time = ∑ (I \* CPI)/ Clock rate

= 106 \* [(0.1 \* 2) + (0.2 \* 2) + (0.5 \* 2) + (0.2 \* 2)] / (2 \* 109)

= [0.2 + 0.4 + 1.0 + 0.4] \* 10-3 /2

= (2.0/2.0) milliseconds [since 1 millisecond = 10-3 seconds]

= 1 millisecond

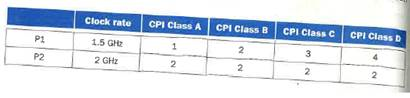
**Step 4**

As performance is inversely proportional to CPU Time, The Processor taking least CPU time performs better

Therefore the Implementation P2 is faster.

**Question1.4.2**

Consider two different implementations of the same instruction set architecture. There are four classes of instructions, A, B, C, and D. The clock rate and CPI of each implementation are given in the following table.



[5] <1.4> What is the global CPI for each implementation?

**Answer**

**Step 1**

**Calculating global CPI (Clock cycles per instructions)**

Consider four classes of instructions A, B, C, and D.The clock rate and CPI of each implementation are given in the below table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Clock Rate** | **CPI class A** | **CPI class B** | **CPI class C** | **CPI class D** |
| P1 | 1.5 GHz | 1 | 2 | 3 | 4 |
| P2 | 2 GHz | 2 | 2 | 2 | 2 |

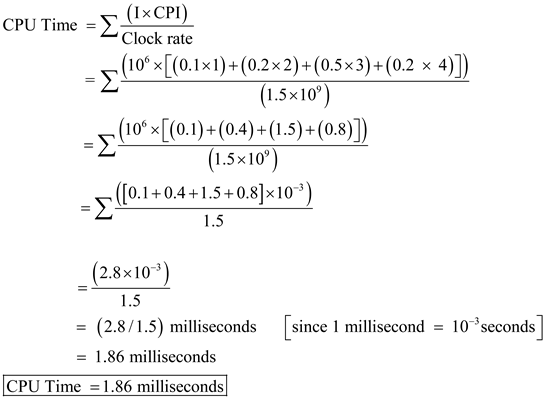
**Step 2**

And also consider  instructions divided into classes as follows: 10% for class A, 20% for class B, 50% for class C and 20% for class D.

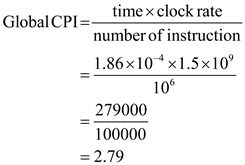
**Step 3**

**For processor P1**:

Calculate the **CPU Time** for P1 processor:



Now, calculate the **global CPI** for P1 processor:

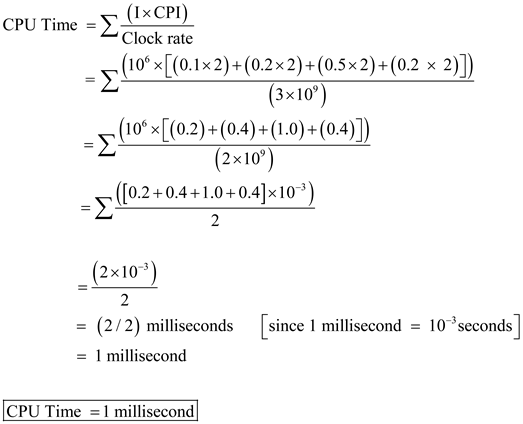
****

****

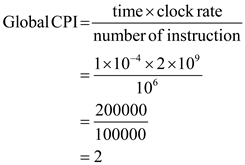
**Step 4**

**For processor P2**:

Calculate the **CPU Time** for P2 processor:



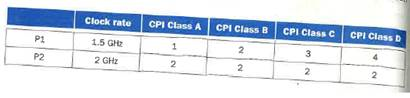
Now, calculate the **global CPI** for P2 processor:

****

****

**Question1.4.3**

Consider two different implementations of the same instruction set architecture. There are four classes of instructions, A, B, C, and D. The clock rate and CPI of each implementation are given in the following table.



[5] <1.4> Find the clock cycles required in both cases.

**Answer**

**Step 1**

728-1.4-3E SA Code: 6456

SR Code: 4460

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Clock Rate | CPI class A | CPI class B | CPI class C | CPI class D |
| P1 | 1.5GHz | 1 | 2 | 3 | 4 |
| P2 | 2 GHz | 2 | 2 | 2 | 2 |

**Step 2**

For P1

CPU Time = ∑ (I \* CPI)/ Clock rate

= 106 \* [(0.1 \* 1) + (0.2 \* 2) + (0.5 \* 3) + (0.2 \* 4)] / (1.5 \* 109)

= [0.1 + 0.4 + 1.5 + 0.8] \* 10-3 /1.5

= (2.8/1.5) milliseconds [since 1 millisecond = 10-3 seconds]

Now global CPI = (CPU Time \* clock rate) / I

= ((2.8/1.5) \* 10-3 \*(1.5 \* 109)) / 106

= 2.8 \* 106/ 106

= 2.8

Global CPI = 2.8

Number of clock cycles = Global CPI \* Instructions

= 2.8 \* 106

**Step 3**

For P2

CPU Time = ∑ (I \* CPI)/ Clock rate

= 106 \* [(0.1 \* 2) + (0.2 \* 2) + (0.5 \* 2) + (0.2 \* 2)] / (2 \* 109)

= [0.2 + 0.4 + 1.0 + 0.4] \* 10-3 /2

= (2.0/2.0) milliseconds [since 1 millisecond = 10-3 seconds]

= 1 millisecond

Now global CPI = (CPU Time \* clock rate) / I

= (1.0 \* 10-3 \*(2 \* 109)) / 106

= 2.0 \* 106/ 106

= 2.0

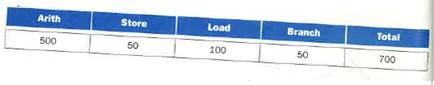
Global CPI = 2.0

Number of clock cycles = Global CPI \* Instructions

= 2.0 \* 106

**Question1.4.4**

The following table shows the number of instructions for a program.



[5] <1.4> Assuming that arith instructions take 1 cycle, load and store 5 cycles, and branches 2 cycles, what is the execution time of the program in a 2 GHz processor?

**Answer**

**Step 1**

728-1.4-4E SA Code: 6456

SR Code: 4460

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Arith | Store | Load | Branch | Total |
| 500 | 50 | 100 | 50 | 700 |

**Step 2**

Execution Time = ∑ (clock cycles needed)/ clock rate)

= [(1\* 500) + (5 \* 50) + (5 \* 100) + (2 \* 50) ] / (2 \* 109)

= 1350 / (2 \* 109)

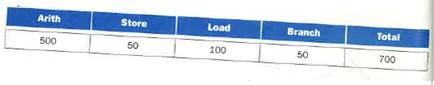
= 675 \* 10-9

= 675 nanoseconds

Therefore Execution Time= 0.675 microseconds

**Question1.4.5**

The following table shows the number of instructions for a program.



[5] <1.4> Find the CPI for the program.

**Answer**

**Step 1**

728-1.4-5E SA Code: 6456

SR Code: 4460

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Arith | Store | Load | Branch | Total |
| 500 | 50 | 100 | 50 | 700 |

**Step 2**

Execution Time = ∑ (clock cycles needed)/ clock rate)

= [(1\* 500) + (5 \* 50) + (5 \* 100) + (2 \* 50) ] / (2 \* 109)

= 1350 / (2 \* 109)

= 675 \* 10-9

= 675 nanoseconds

Therefore Execution Time= 0.675 microseconds

**Step 3**

Execution Time = (total instructions \* total CPI) / clock rate

675 \* 10-9 = (700 \* CPI) / (2 \* 109)

Therefore CPI = (675 \* 2) / 700

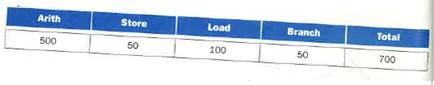
= 1350/ 700

= 1.929

Total CPI for the program = 1.929

**Question1.4.6**

The following table shows the number of instructions for a program.



[10] <1.4> If the number of load instructions can be reduced by one half, what is the speedup and the CPI?

**Answer**

**Step 1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Arith | Store | Load | Branch | Total |
| 500 | 50 | 100 | 50 | 700 |

**Step 2**

Execution Time = ∑ (clock cycles needed)/ clock rate)

= [(1\* 500) + (5 \* 50) + (5 \* 100) + (2 \* 50) ] / (2 \* 109)

= 1350 / (2 \* 109)

= 675 \* 10-9

= 675 nanoseconds

Therefore Execution Time= 0.675 microseconds

**Step 3**

If the load instructions are reduced by half, then number of load instructions is 50

Execution Time = ∑ (clock cycles needed)/ clock rate

= [(1\* 500) + (5 \* 50) + (5 \* 50) + (2 \* 50)] / (2 \* 109)

= 1100 / (2 \* 109)

= 550 \* 10-9

= 550 nanoseconds

Now the speed-up acquired = (Execution Time when Load instructions are 100)/ (Execution Time when Load instructions are 50)

Therefore speedup = (675/ 550)

= 1.227

**Step 4**

Execution Time = (total instructions \* total CPI) / clock rate

550 \* 10-9 = (650 \* CPI) / (2 \* 109)

Therefore CPI = (550 \* 2) / 650

= 1100/ 650

= 1.692

New CPI for the program = 1.692

# Question2.4.1

The following problems deal with translating from C 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.



[10] <2.2, 2.3> For the C statements above, what is the corresponding MIPS assembly code?

# Answer

## Step 1

**Translating C to MIPS Statements:**

**Case a:**

MIPS stand for Million Instructions per Second. Assign the variables f, g and h to registers $s0, $s1 and $s2 respectively. Initialize the address of the arrays A and B are in registers $s6 and $s7 respectively. Consider the following C statement:

f = g + h + B[4]

Now, translating the given statements to MIPS statements as follows:

**MIPS assembly code**:

**lw $t0, 16($s7) # load the B[4] into $t0**

**add $s0, $t0, $s2 # f = B[4]+ h**

**add $s0, $s0, $s1 # f = f + h = g+ h+ B[4]**

**Explanation:**

• First, loads B[4] value to the temporary register $t0.

• Perform addition operation for **$t0** and**$s2** registers and store the result in **$s0 (f=** h + B[4]**).**

• Perform addition operation for **$s0** and**$s1** registers and store the result in **$s0 (f=** g+ h + B[4]**).**

## Step 2

**Case b:**

Assign the variables f, g and h to registers $s0, $s1 and $s2 respectively.

Initialize the address of the arrays A and B are in registers $s6 and $s7 respectively.

Consider the following C statement:

f = g – A[ B[4] ]

Now, translating the given statements to MIPS statements as follows:

**MIPS assembly code**:

**lw $t0, 16($s7) # temporary register t0 gets B[4]**

**lw $s0, 0($t0) # A[B[4]]**

**sub $s0, $s1, $s0 # f = g - A[B[4]]**

**Explanation:**

• First loads B[4] value to temporary register $t0.

• First, loads A[B[4]] value to $s0.

• At last do the subtract operation g and A[B[4]].

**Question2.4.2**

The following problems deal with translating from C 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.



[5] <2.2, 2.3> For the C statements above, how many MIPS assembly instructions are needed to perform the C statement?

**Answer**

**Step 1**

|  |  |
| --- | --- |
| a. | f = g + h + B[4] |
| b. | f = g – A[ B[4]] |

**Step 2**

a)

There is a single operation in the C statement, one of the operand is in memory, and we must first transfer B[4] to a register. The array of this array element is the sum of the base of the array B, found in register $s7, plus the number to select element 4. (We’ll use simplified version for now, make slight adjustment further problems)

add $s0, $s1, $s2 # f= g + h

lw $t0, 4($s7) # temporary register t0 gets B[4]

add $s0,$s0, $t0 # f = g + h + B[4]

Therefore 3 MIPS assembly instructions are needed to execute the above C statement

**Step 3**

b)

lw $t0, 4($s7) # temporary register t0 gets B[4]

add $t0, $t0, $s6 # initialize A array

lw $t1, 0($t0) # A[B[4]]

sub $s0, $s1, $t1 # f = g- A[B[4]]

Therefore 4 MIPS assembly instructions are needed to execute the above C statement

# Question2.4.3

The following problems deal with translating from C 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.



[5] <2.2, 2.3> For the C statements above, how many different registers are needed to carry out the C statement?

# Answer

## Step 1

**Case a:**

Assume that the variables f, g and h are assigned to registers $s0, $s1 and $s2.

Assume that the base address of the array B is in register $s7.

Consider the following C statement:

f = g + h + B[4]

Now, translating the given statements to MIPS statements as follows:

**MIPS assembly code**:

There is a single operation in the C statement, one of the operand is in memory, and we must first transfer B[4] to a register. The corresponding MIPS code is given below:

lw $s0, 16($s7) # s0 gets B[4]

add $s0, $s0, $s1 # f = f + g

add $s0,$s0, $s2 # f = f + h

Hence,**4 registers ($s0, $s1, $s2 and $7)** are needed to execute the above C statement.

## Step 2

**Case b:**

Assume that the variables f and g are assigned to registers $s0 and $s1.

Assume that the base address of the arrays A and B are in registers $s6 and $s7.

Consider the following C statement:

f = g – A[ B[4]]

Now, translating the given statements to MIPS statements as follows:

**MIPS assembly code**:

There is a single operation in the C statement, one of the operand is in memory, and we must first transfer B[4] to a register. The corresponding MIPS code is given below:

lw $t0, 16($s7) # temporary register t0 gets B[4]

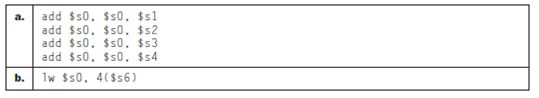
lw $s0, 0($t0) # A[B[4]]

sub $s0, $s1, $s0 # f = g - A[B[4]]

Hence, **4 registers ($s0, $s1, $t0 and $s7)** are needed to execute the above C statement.

**Question2.4.4**

The following problems deal with translating from MIPS to C. 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.



[10] <2.2, 2.3> For the MIPS assembly instructions above, what is the corresponding C statement?

**Answer**

**Step 1**

**Translate MIPS (Million Instructions per Second) Instructions into C Statements**

**Case a:**

Consider the following MIPS assembly instructions:

|  |
| --- |
| add $s0, $s0, $s1  add $s0, $s0, $s2  add $s0, $s0, $s3  add $s0, $s0, $s4 |

It is assumed that the variables f, g, h, i, and j are assigned to registers $s0, $s1, $s2, $s3, and $s4, respectively. The base address of the arrays A and B are in registers $s6 and $s7 respectively.

**Interpretation of MIPS instructions:**

add $s0, $s0, $s1 # f = f + g ;

add $s0, $s0, $s2 # f = f + g + h

add $s0, $s0, $s3 # f = f + g + h + i

add $s0, $s0, $s4 # f = f + g + h + i + j

Therefore, the corresponding C statement is f = f + g + h + i + j.

**Step 2**

**Case b:**

Consider the following MIPS assembly instruction:

|  |
| --- |
| lw $s0, 4($s6) |

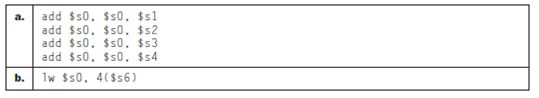
It is assumed that the variables f is assigned to registers $s0. The base address of the array A is in register $s6.

Now, translate the MIPS statement into c statement. The index has been accounting for the 4 byte lengths of each word in the array. So 4($s6) is actually A [1].In the same way, 0($s6) =A [0] and 8($s6) =A [2].Multiply the index by 4 to get the appropriate value.

Therefore, the corresponding C statement is f = A[1].

# Question2.4.5

The following problems deal with translating from MIPS to C. 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.



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

# Answer

## Step 1

**MIPS Instruction Minimization**

**Case a:**

Consider the following MIPS (Million Instructions per Second) instructions:

add $s0, $s0, $s1

add $s0, $s0, $s2

add $s0, $s0, $s3

add $s0, $s0, $s4

The above MIPS instructions performs adding contents of registers $s1, $s2, $s3 and $s4.Now we have to rewrite the assembly code to minimize the number of MIPS instructions to carry out the same function using temporary registers.

Minimized MIPS instructions:

**add $t0, $s1, $s2**

**add $t1, $s3, $s4**

**add $s0, $t0, $t1**

## Step 2

**Case b:**

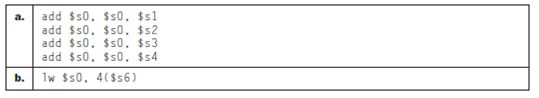
Consider the following MIPS instruction:

lw $s0, 4($s6)

The above MIPS instruction A[1]is loaded into the register at $s0.Here only one instruction is needed to load A[4]which is in memory to register $s0.Therefore we cannot minimize the MIPS instruction.

**Question2.4.6**

The following problems deal with translating from MIPS to C. 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.



[5] <2.2, 2.3> How many registers are needed to carry out the MIPS assembly as written above? If you could rewrite the code above, what is the minimal number of registers needed?

**Answer**

**Step 1**

|  |  |
| --- | --- |
| a. | add $s0, $s0, $s1  add $s0, $s0, $s2  add $s0, $s0, $s3  add $s0, $s0, $s4 |
| b. | lw $s0, 4($s6) |

**Step 2**

a)

We are using the registers $s0, $s1, $s2, $s3, $s4

Therefore number of registers used is 5.

We need at least 4 MIPS instructions to perform the above C statement which are for calculating (f + g), (h + i), (h + i) + j, (f + g) + ((h + i) + j).

Therefore minimum number of registers used in any other approach also takes 5 registers

**Step 3**

b)

We are using the register $s0 to load A[4] from memory.

Therefore number of registers used is 1.

We cannot minimize the MIPS instructions since one instruction is needed to load A[4] which is in memory to register $s0 which is variable f.

Therefore minimum number of registers used in any other approach is also 1.

**Question2.11.1**

In the following problems, the data table contains bits that represent the opcode of an instruction. You will be asked to translate the entries into assembly code and determine what format of MIPS instruction the bits represent.



[5] <2.4, 2.5> What binary number does the above hexadecimal number represent?

**Answer**

**Step 1**

Given data:

|  |  |
| --- | --- |
| a. | 0xAE0BFFFC |
| b. | 0x8D08FFC0 |

**Step 2**

Case a:

0xAE0BFFFC represents the binary number 1010 1110 0000 1011 1111 1111 1111 1100

**Step 3**

Case b:

0x8D08FFC0 represents the binary number 1000 1101 0000 1000 1111 1111 1100 0000

**Question2.11.2**

In the following problems, the data table contains bits that represent the opcode of an instruction. You will be asked to translate the entries into assembly code and determine what format of MIPS instruction the bits represent.



[5] <2.4, 2.5> What decimal number does the above hexadecimal number represent?

**Answer**

**Step 1**

Given data:

|  |  |
| --- | --- |
| a. | 0xAE0BFFFC |
| b. | 0x8D08FFC0 |

**Step 2**

Case a:

0xAE0BFFFC represents the decimal number

= 10  167 +14 166 + 11  164 + 15163 + 15162 + 15161 + 12

= 12 + 240 + 3840 + 61440 + 720896 + 234881024 + 2684354560

= 2920022012

**Step 3**

Case b:

0x8D08FFC0 represents the decimal number

= 8 167 +13 166 + 8  164 + 15163 + 15162 + 12161

= 2147483648 + 218103808 + 524288 + 61440 + 3840 + 192

= 2366177216

**Question2.11.3**

In the following problems, the data table contains bits that represent the opcode of an instruction. You will be asked to translate the entries into assembly code and determine what format of MIPS instruction the bits represent.



[5] <2.5> What instruction does the above hexadecimal number represent?

**Answer**

**Step 1**

|  |  |
| --- | --- |
| a. | 0xAE0BFFFC |
| b. | 0x8D08FFC0 |

**Step 2**

Case a:

0xAE0BFFFC represents the binary number 1010 1110 0000 1011 1111 1111 1111 1100

It can be divided into 101011 10000 01011 1111111111111100

The opcode is 43,

rs is16,

rt is 11,

The 16-bit address means a load word instruction can load any word within a region of or 32,768 bytes.

Here the address field is more than 32,768. So, we have to do 2’s complement.

1111111111111100

1’s complement 0000000000000011

Add 1, and get result as 0000000000000100

Address/immediate field is -4

Therefore the instruction is sw $t3, -4($s0)

**Step 3**

Case b:

0x8D08FFC0 represents the binary number 1000 1101 0000 1000 1111 1111 1100 0000

It can be divided into 100011 01000 01000 1111111111000000

The opcode is 35,

rs is 8,

rt is 8,

The 16-bit address means a load word instruction can load any word within a region of or 32,768 bytes.

Here the address field is more than 32,768. So, we have to do 2’s complement.

1111111111000000

1’s complement 0000000000111111

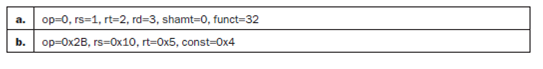
Add 1, and get result as 0000000001000000

Address/immediate field is -64

Therefore the instruction is lw $t0, -64($t0)

**Question2.11.4**

In the following problems, the data table contains the values of various fields of MIPS instructions. You will be asked to determine what the instruction is, and find the MIPS format for the instruction.



[5] <2.5> What type (I-type, R-type) instruction do the instructions above represent?

**Answer**

**Step 1**

Given data:

|  |  |
| --- | --- |
| a. | Op=0, rs=1, rt=2, rd=3, shamt=0, funct=32 |
| b. | Op=0x2B, rs=0x10, rt=0x5, const= 0x4 |

**Step 2**

Case a:

The opcode of the above instruction is zero and funct is 32

Therefore the instruction is add which is of R-type

**Step 3**

Case b:

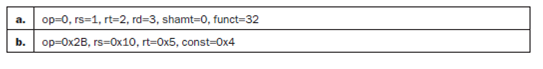
The instruction has only 4 fields which is typical for I-type instructions.

The opcode of the above instruction is 43 and also a const is given

Therefore the instruction is sw which is of I-type

**Question2.11.5**

In the following problems, the data table contains the values of various fields of MIPS instructions. You will be asked to determine what the instruction is, and find the MIPS format for the instruction.



[5] <2.5> What is the MIPS assembly instruction described above?

**Answer**

**Step 1**

Given data:

|  |  |
| --- | --- |
| a. | Op=0, rs=1, rt=2, rd=3, shamt=0, funct=32 |
| b. | Op=0x2B, rs=0x10, rt=0x5, const= 0x4 |

**Step 2**

Case a:

The opcode of the above instruction is zero and funct is 32

rs = 1, therefore rs= $at

rt = 2, therefore rt= $v0

rd = 3, therefore rd= $v1

Therefore the instruction is add $v1, $at, $v0

**Step 3**

Case b:

The instruction has only 4 fields which is typical for I-type instructions.

The opcode of the above instruction is 43 and also a const is given 4

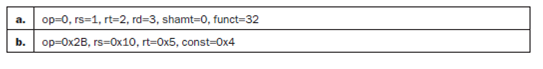
rs = 16, therefore rs= $s0

rt = 5, therefore rt= $a1

Therefore the instruction is sw $a1, 4($s0)

**Question2.11.6**

In the following problems, the data table contains the values of various fields of MIPS instructions. You will be asked to determine what the instruction is, and find the MIPS format for the instruction.



[5] <2.4, 2.5> What is the binary representation of the instructions above?

**Answer**

**Step 1**

Given data:

|  |  |
| --- | --- |
| a. | Op=0, rs=1, rt=2, rd=3, shamt=0, funct=32 |
| b. | Op=0x2B, rs=0x10, rt=0x5, const= 0x4 |

**Step 2**

Case a:

The binary representation of the instruction above is

000000 00001 00010 00011 00000 100000

**Step 3**

Case b:

The binary representation of the instruction above is

101011 10000 00101 0000000000000100

**Question2.13.1**

In the following problems, the data table contains the values for registers $t0 and $t1. You will be asked to perform several MIPS logical operations on these registers.



[5] <2.6> For the lines above, what is the value of $t2 for the following sequence of instructions:



**Answer**

**Step 1**

Given data:

|  |  |
| --- | --- |
| a. | $t0 = 0x55555555, $t1 = 0x12345678 |
| b. | $t0 = 0xBEADFEED, $t1 = 0xDEADFADE |

Instructions:

sll $t2, $t0, 4

or $t2, $t2, $t1

**Step 2**

Case a:

After the first instruction $t2 contains 0x55555550 [the value in $t0 is shifted four times]

After second instruction $t2 contains (0x55555550) or (0x12345678)

=(0101 0101 0101 0101 0101 0101 0101 0000) or (0001 0010 0011 0100 0101 0110 0111 1000)

=0101 0111 0111 0101 0101 0111 0111 1000

= 0x57755778

**Step 3**

Case b:

After the first instruction $t2 contains 0xEADFEED0 [the value in $t0 is shifted four times]

After second instruction $t2 contains (0x EADFEED0) or (0xDEADFADE)

=(1110 1010 1101 1111 1110 1110 1101 0000) or (1101 1110 1010 1101 1111 1010 1101 1110)

=1111 1110 1111 1111 1111 1110 1101 1110

= 0xFEFFFEDE

**Question2.13.2**

In the following problems, the data table contains the values for registers $t0 and $t1. You will be asked to perform several MIPS logical operations on these registers.



[5] <2.6> For the values in the table above, what is the value of $t2 for the following sequence of instructions:



**Answer**

**Step 1**

Given data:

|  |  |
| --- | --- |
| a. | $t0 = 0x55555555, $t1 = 0x12345678 |
| b. | $t0 = 0xBEADFEED, $t1 = 0xDEADFADE |

Instructions:

sll $t2, $t0, 4

and $t2, $t2, -1

**Step 2**

Case a:

After the first instruction $t2 contains 0x55555550 [the value in $t0 is shifted left four times]

After second instruction $t2 contains (0x55555550) and (0xFFFFFFFF)

= (0101 0101 0101 0101 0101 0101 0101 0000) and (1111 1111 1111 1111 1111 1111 1111 1111)

= 0101 0101 0101 0101 0101 0101 0101 0000

= 0x55555550

**Step 3**

Case b:

After the first instruction $t2 contains 0xEADFEED0 [the value in $t0 is shifted four times]

After second instruction $t2 contains (0x EADFEED0) and (0xFFFFFFFF)

=(1110 1010 1101 1111 1110 1110 1101 0000) and (1111 1111 1111 1111 1111 1111 1111 1111)

= 1110 1010 1101 1111 1110 1110 1101 0000

= 0xEADFEED0

**Question2.13.3**

In the following problems, the data table contains the values for registers $t0 and $t1. You will be asked to perform several MIPS logical operations on these registers.



[5] <2.6> For the lines above, what is the value of $t2 for the following sequence of instructions:



**Answer**

**Step 1**

Given data:

|  |  |
| --- | --- |
| a. | $t0 = 0x55555555, $t1 = 0x12345678 |
| b. | $t0 = 0xBEADFEED, $t1 = 0xDEADFADE |

Instructions:

srl $t2, $t0, 3

and $t2, $t2, 0xFFEF

**Step 2**

Case a:

In the first instructions 0x55555555 is right shifted 3 times

0x55555555 in binary is 0101 0101 0101 0101 0101 0101 0101 0101

Shifting 3 times gives 0000 1010 1010 1010 1010 1010 1010 1010

Therefore $t2 contains 0x0AAAAAAA

**Step 3**

After second instruction $t2 contains (0x0AAAAAAA) and (0xFFEF)

= (0000 1010 1010 1010 1010 1010 1010 1010) and (0000 0000 0000 0000 1111 1111 1110 1111)

= 0000 0000 0000 0000 1010 1010 1010 1010

= 0x0000AAAA

Therefore $t2 contains 0x0000AAAA

**Step 4**

Case b:

In the first instructions 0xBEADFEED is right shifted 3 times

0xBEADFEED in binary is 1011 1110 1010 1101 1111 1110 1110 1101

Shifting 3 times gives 0001 0111 1101 0101 1011 1111 1101 1101

Therefore $t2 contains 0x17D5BFDD

**Step 5**

After second instruction $t2 contains (0x17D5BFDD) and (0xFFEF)

= (0001 0111 1101 0101 1011 1111 1101 1101) and (0000 0000 0000 0000 1111 1111 1110 1111)

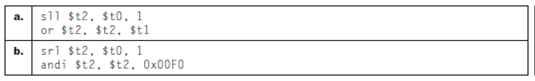
= 0000 0000 0000 0000 1011 1111 1100 1101

= 0x0000BFCD

Therefore $t2 contains 0x0000BFCD

**Question2.13.4**

In the following exercise, the data table contains various MIPS logical operations. You will be asked to find the result of these operations given values for registers $t0 and $t1.



[5] <2.6> Assume that $t0 = 0x0000A5A5 and $t1 = 00005A5A. What is the value of $t2 after the two instructions in the table?

**Answer**

**Step 1**

Given data:

|  |  |
| --- | --- |
| a. | Sll $t2, $t0, 1  Or $t2, $t2, $t1 |
| b. | Sll $t2, $t0, 1  andi $t2, $t2, 0x00F0 |

$t0 = 0x0000A5A5

$t1 = 0x00005A5A

**Step 2**

Case a :

In the first instructions 0x0000A5A5 is left shifted 1 time

0x0000A5A5 in binary is 0000 0000 0000 0000 1010 0101 1010 0101

Shifting left 1 time gives 0000 0000 0000 0001 0100 1011 0100 1010

Therefore $t2 contains 0x00014B4A

**Step 3**

After second instruction $t2 contains (0x00014B4A) or (0x00005A5A)

= (0000 0000 0000 0001 0100 1011 0100 1010) or (0000 0000 0000 0000 0101 1010 0101 1010)

= 0000 0000 0000 0001 0101 1011 0101 1010

= 0x00015B5A

Therefore $t2 contains 0x00015B5A

**Step 4**

Case b :

In the first instructions 0x0000A5A5 is left shifted 1 time

0x0000A5A5 in binary is 0000 0000 0000 0000 1010 0101 1010 0101

Shifting left 1 time gives 0000 0000 0000 0001 0100 1011 0100 1010

Therefore $t2 contains 0x00014B4A

**Step 5**

After second instruction $t2 contains (0x00014B4A) and (0x000000F0)

= (0000 0000 0000 0001 0100 1011 0100 1010) and (0000 0000 0000 0000 0000 0000 1111 0000)

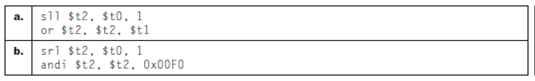
= 0000 0000 0000 0000 0000 0000 0100 0000

= 0x00000040

Therefore $t2 contains 0x00000040

**Question2.13.5**

In the following exercise, the data table contains various MIPS logical operations. You will be asked to find the result of these operations given values for registers $t0 and $t1.



[5] <2.6> Assume that $t0 = 0xA5A50000 and $t1 = A5A50000. What is the value of $t2 after the two instructions in the table?

**Answer**

**Step 1**

Given data:

|  |  |
| --- | --- |
| a. | Sll $t2, $t0, 1  Or $t2, $t2, $t1 |
| b. | Sll $t2, $t0, 1  andi $t2, $t2, 0x00F0 |

$t0 = 0xA5A50000

$t1 = 0xA5A50000

**Step 2**

Case a:

In the first instructions 0xA5A50000 is left shifted 1 time

0xA5A50000 in binary is 1010 0101 1010 0101 0000 0000 0000 0000

Shifting left 1 time gives 0100 1011 0100 1010 0000 0000 0000 0000

Therefore $t2 contains 0x4B4A0000

**Step 3**

After second instruction $t2 contains (0x4B4A0000) or (0xA5A50000)

= (0100 1011 0100 1010 0000 0000 0000 0000) or (1010 0101 1010 0101 0000 0000 0000 0000)

= 1110 1111 1110 1111 0000 0000 0000 0000

= 0xEFEF0000

Therefore $t2 contains 0xEFEF0000

**Step 4**

Case b:

In the first instructions 0xA5A50000 is left shifted 1 time

0xA5A50000 in binary is 1010 0101 1010 0101 0000 0000 0000 0000

Shifting left 1 time gives 0100 1011 0100 1010 0000 0000 0000 0000

Therefore $t2 contains 0x4B4A0000

**Step 5**

After second instruction $t2 contains (0x4B4A0000) and (0x00F0)

= (0100 1011 0100 1010 0000 0000 0000 0000) and (0000 0000 0000 0000 0000 0000 1111 0000)

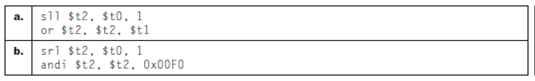
= 0000 0000 0000 0000 0000 0000 0000 0000

= 0x00000000

Therefore $t2 contains 0x00000000

# Question2.13.6

In the following exercise, the data table contains various MIPS logical operations. You will be asked to find the result of these operations given values for registers $t0 and $t1.



[5] <2.6> Assume that $t0 = 0xA5A5FFFF and $t1 = A5A5FFFF. What is the value of $t2 after the two instructions in the table?

# Answer

## Step 1

Consider the following data:

$t0 = 0xA5A5FFFF

$t1 = A5A5FFFF

## Step 2

**a.**

Consider the following instructions:

sll $t2, $t0, 1

or $t2, $t2, $t1

**Explanation:**

• The first instruction is shift logical left instruction. The value in the register $t0 is shifted left by 1 position and stored in $t2.

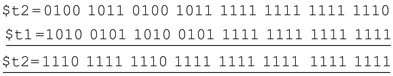
• The value of $t0 is 0xA5A5FFFF. Its binary representation is 1010 0101 1010 0101 1111 1111 1111 1111.

• Shifting the bit positions of $t0 to left by 1 results in 0100 1011 0100 1011 1111 1111 1111 1110. So, the value of $t2 is 0x4B4BFFFE.

• The second instruction performs and immediate operation to the value in $t2 and -1.

• The binary value of $t1 is 1010 0101 1010 0101 1111 1111 1111 1111.

• The or operation results 0 when the both inputs are 1 otherwise 1. The or operation is as follows:



**Thus, the value of $t2** **is 1110 1111 1110 1111 1111 1111 1111 1111** **and its representation in hexadecimal is 0xEFEFFFFF** **.**

## Step 3

**b.**

Consider the following instructions:

srl $t2, $t0, 1

andi $t2, $t2, 0x00F0

**Explanation:**

• The first instruction is shift logical right instruction. The value in the register $t0 is shifted left by 1 position and stored in $t2.

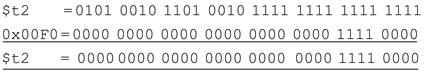
• The value of $t0 is 0xA5A5FFFF. Its binary representation is 1010 0101 1010 0101 1111 1111 1111 1111.

• Shifting the bit positions of $t0 to right by 1 results in 0101 0010 1101 0010 1111 1111 1111 1111. So, the value of $t2 is 52D2FFFF.

• The second instruction performs and immediate operation to the value in $t2 and 0x00F0.

• The value of 0x00F0 is 0000 0000 0000 0000 0000 0000 1111 0000.

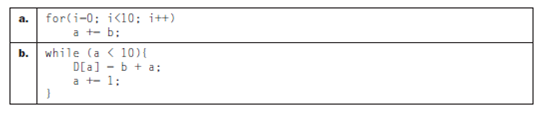
• The and immediate operation results 1 when the both inputs are 1 otherwise 0. The and operation is as follows:



**Thus, the value of $t2** **is 0000 0000 0000 0000 0000 0000 1111 0000** **and its representation in hexadecimal is 000000F0.**

**Question2.18.1**

For these problems, the table holds some C code. You will be asked to evaluate these C code statements in MIPS assembly code.



[5] <2.7> For the table above, draw a control-flow graph of the C code.

**Answer**

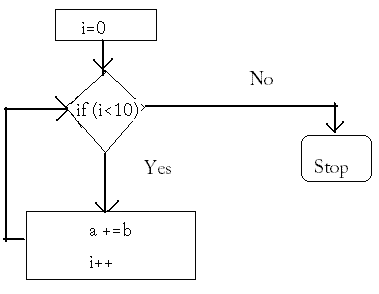
**Step 1**

Given data:

|  |  |
| --- | --- |
| a. | for(i=0; i<10; i++)  a += b; |
| b. | While (a <10)  D[i] = b + a;  A +=1; |

**Step 2**

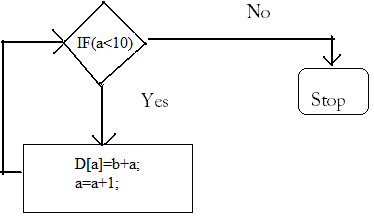
Case a:



Control-flow graph

**Step 3**

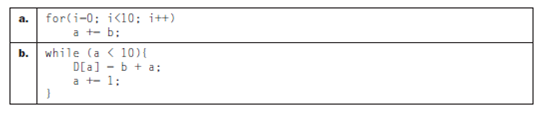
Case b:



Control-flow graph

**Question2.18.2**

For these problems, the table holds some C code. You will be asked to evaluate these C code statements in MIPS assembly code.



[5] <2.7> For the table above, translate the C code to MIPS assembly code. Use a minimum number of instructions. Assume that the value a, b, i, j are in registers $s0, $s1, $t0, $t1, respectively. Also, assume that register $s2 holds the base address of the array D.

**Answer**

**Step 1**

**C Code to MIPS Assembly Code Translation**

**Case a:**

Consider the following C code:

for(i=0; i<10; i++)

a += b;

Values of a, b, i, and j are assumed to be in registers $s0, $s1, $t0, and $t1.

And $s2 holds the base address of array D, Translate the C code to MIPS code as follows:

**MIPS Code:**

**add $t0, $tzer0, $zer0 #sets i=0**

**LOOP: slti $t1, $t0, 10 #checks whether i<10**

**beq $t1, $zero, DONE #if i is not less than 10,**

**#jump to DONE**

**add $s0, $s0, $s1 #it is for statement a = a+ b**

**addi $t0, $t0, 1**

**j LOOP**

**DONE:**

**Step 2**

**Case b:**

Consider the following C code:

While (a <10)

D[a] = b + a;

A +=1;

Values of a, b, i, and j are assumed to be in registers $s0, $s1, $t0, and $t1.

And $s2 holds the base address of array D. Translate the C code to MIPS code as follows:

**MIPS Code:**

**LOOP: slti $t1, $s0, 10 # checks whether a<10**

**beq $t1, $zero, DONE # if i is not less than 10, # jump to DONE**

**add $t0, $s1, $s0 # it is for calculating a+ b**

**add $t2, $s2, $s0 # $t2 contains the address of D[a]**

**sw $t0, 0($t2) # storing b+a in D[a]**

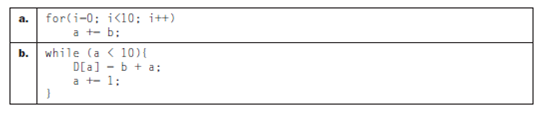
**addi $s0, $s0, 1 # adds 1 to a (now a =a+1)**

**j LOOP**

**DONE:**

**Question2.18.3**

For these problems, the table holds some C code. You will be asked to evaluate these C code statements in MIPS assembly code.



[5] <2.7> How many MIPS instructions does it take to implement the C code? If the variables a and b are initialized to 10 and 1 and all elements of D are initially 0, what is the total number of MIPS instructions that is executed to complete the loop?

**Answer**

**Step 1**

Given data:

|  |  |
| --- | --- |
| a. | for(i=0; i<10; i++)  a += b; |
| b. | While (a <10)  D[i] = b + a;  A +=1; |

Given a=10 and b=1 and all elements of D are initialized to zero

**Step 2**

Case a:

The first instruction is always executed.

The LOOP is executed 10 times because I is equal to zero initially and it becomes 10 after 10 iterations of LOOP

There are 5 instructions in LOOP. Therefore each instruction in the LOOP is executed 10 times.

Therefore total number of instructions executed = (10  instructions in LOOP) + first instruction

= 10  5 + 1

= 50 + 1

= 51

**Step 3**

Case b:

Given a=10 initially.

The first instruction is executed and $t1 is set to zero since a (which is 10) is not less than 10.

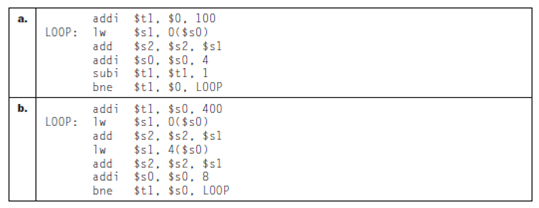
Now in the second instruction as $t1 is equal to zero it will jump to DONE.

Therefore only two instructions are executed.

Total number of instructions executed = 2

# Question2.18.4

For these problems, the table holds MIPS assembly code fragments. You will be asked to evaluate each of the code fragments, familiarizing you with the different MIPS branch instructions.



[5] <2.7> What is the total number of MIPS instructions executed?

# Answer

## Step 1

Case a:

The first instruction is executed once and it enters the LOOP.

After the first instruction $t1 contains the value 100.

Now the LOOP is iterated 100 times since $t1 becomes zero only after 100 iterations.

Therefore the instructions in the loop are executed 100 times

There are 5 instructions in LOOP and each instruction is executed 100 times.

Therefore total instructions executed are instructions in LOOP + first instruction

= 5100 + 1

= 501

## Step 2

Case b:

The first instruction is executed once and it enters the LOOP.

After the first instruction $t1 contains the value $s0 + 400.

In every iteration of the LOOP $s0 is incremented by 8 and the LOOP breaks when $t1 equals $s0

## Step 3

After 50 iterations $s0 equals $t1.

Therefore LOOP iterates 50 times.

There are 6 instructions in the LOOP and each instruction will be executed 50 times (since there are 50 iterations)

Total number of instructions executed is instructions in LOOP + first instruction

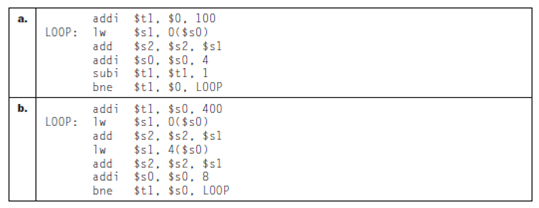
= 650 + 1

= 300 +1

= 301

# Question2.18.5

For these problems, the table holds MIPS assembly code fragments. You will be asked to evaluate each of the code fragments, familiarizing you with the different MIPS branch instructions.



[5] <2.7> Translate the loops above into C. Assume that the C-level integer i is held in register $t1, $s2 holds the C-level integer called result, and $s0 holds the base address of the integer MemArray.

# Answer

## Step 1

Case a:

The equivalent C code is:

i = 0 + 100;

int temp=0;

While (i !=0)

{

result += MemArray[temp];

temp++;

i--;

}

## Step 2

Case b:

The equivalent C code is:

i = &MemArray[0] + 400; //here &MemArray[0] gives the address of MemArray i.e. $s0

int\* temp= &MemArray[0];

While (i! = temp))

{

result += \*temp; //this adds the element at address temp to result

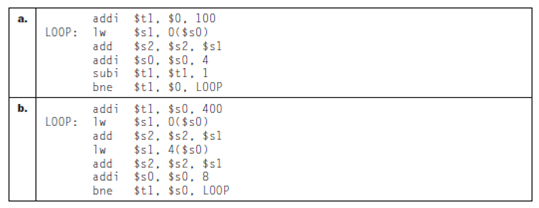
result += \* (temp +4); //adds the next element in the array to result

temp +=8;

}

# Question2.18.6

For these problems, the table holds MIPS assembly code fragments. You will be asked to evaluate each of the code fragments, familiarizing you with the different MIPS branch instructions.



[5] <2.7> Rewrite the loop in MIPS assembly to reduce the number of MIPS instructions executed.

# Answer

## Step 1

Case a:

The first instruction is executed once and it enters the LOOP.

After the first instruction $t1 contains the value 100.

Now the LOOP is iterated 100 times since $t1 becomes zero only after 100 iterations.

Therefore the instructions in the loop are executed 100 times

There are 5 instructions in LOOP and each instruction is executed 100 times.

Therefore total instructions executed are instructions in LOOP + first instruction

= 5100 + 1

= 501

## Step 2

Now in order to reduce the number of MIPS instructions executed, we can modify the loop such that every time two continuous elements of the array are added to the result. Therefore the LOOP has to execute only 50 times.

## Step 3

The MIPS assembly Code is:

Addi $t1, $0, 100

LOOP: lw $s1, 0($s0)

lw $s2, 4($s0)

add $s2, $s2, $s1

addi $s0, $s0, 8

subi $t1, $t1, 2

bne $t1, $0, LOOP

Therefore total instruction executed = 50  6 + 1 = 301

## Step 4

Case b :

The first instruction is executed once and it enters the LOOP.

After the first instruction $t1 contains the value $s0 + 400.

In every iteration of the LOOP $s0 is incremented by 8 and the LOOP breaks when $t1 equals $s0

After 50 iterations $s0 equals $t1.

Therefore LOOP iterates 50 times.

## Step 5

There are 6 instructions in the LOOP and each instruction will be executed 50 times (since there are 50 iterations)

Total number of instructions executed is instructions in LOOP + first instruction

= 650 + 1

= 300 +1

= 301

We can modify the loop such that every time two continuous elements of the array are added to the result. We use only one add $s2, $s2, $s1 instruction instead of two.

## Step 6

The MIPS assembly Code is:

addi $t1, $s0, 400

LOOP: lw $s1, 0($s0)

lw $s2, 4($s0)

add $s2, $s2, $s1

addi $s0, $s0, 8

bne $t1, $0, LOOP

Therefore total instruction executed = 50 \* 5 + 1 = 251

# Question2.27.1

In the following problems, you will be using exploring different addressing modes in the MIPS instruction set architecture. These different addressing modes are listed in the table below.



[5] <2.10> In the table above are different addressing modes of the MIPS instruction set. Give an example MIPS instructions that shows the MIPS addressing mode.

# Answer

## Step 1

728-2.27-1E SA Code: 9420

SR Code: 2603

a) Register Addressing**:**

Register addressing is the simplest addressing mode. Instructions using registers execute quickly because they avoid the delays associated with memory access. Register addressing is a form of direct addressing, because we are interested in the number in the register, rather than using that number as a memory address.

Example of register addressing mode in MIPS instruction:

add $t0,$t1,$t2 # $t0=$t1+$t2

## Step 2

b) PC-relative Addressing**:**

PC-relative addressing, where the address is the sum of the program counter and a constant in the instruction, is used for conditional branches.

Example of PC-relative addressing in MIPS instructions:

beqz $t0,loop # Repeat loop until value at register t0 is not equal to zero.

# Question2.27.2

In the following problems, you will be using exploring different addressing modes in the MIPS instruction set architecture. These different addressing modes are listed in the table below.



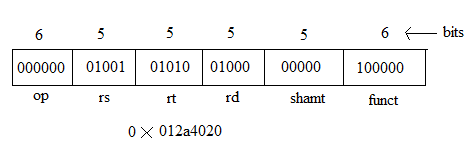
[5] <2.10> For the instructions in 2.27.1, what is the instruction format type used for the given instruction?

# Answer

## Step 1

a) Example of register addressing mode in MIPS instruction:

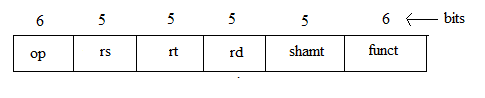
add $t0,$t1,$t2 # $t0=$t1+$t2



## Step 2

R-type instruction format:

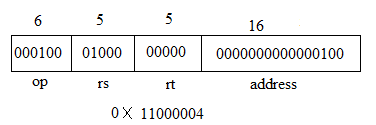
General R-type instruction format is:



## Step 3

b) Example of register addressing mode in MIPS instruction:

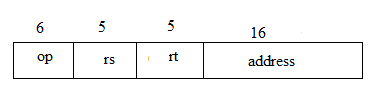
beqz $t0,loop # Repeat loop until value at register t0 is not equal to zero



## Step 4

I-type instruction format **:**

General I-type instruction format is:



# Question2.27.3

In the following problems, you will be using exploring different addressing modes in the MIPS instruction set architecture. These different addressing modes are listed in the table below.



[5] <2.10> List benefits and drawbacks of a particular MIPS addressing mode. Write MIPS code that shows these benefits and drawbacks.

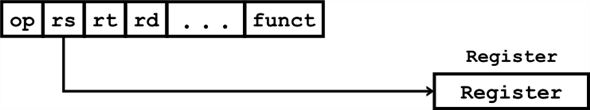
# Answer

## Step 1

**Benefits and Drawbacks of MIPS addressing modes**

**a) Register addressing mode:**

The following diagram shows the register addressing mode.



**Register addressing mode**

The different fields in the register addressing mode are as follows:

• op – used to store the opcode.

• rs – used to store the address of the source register.

• rt – used to store the address of the target register.

• rd – used to store the address of the destination register.

• funct – used to store the address of the function pointer.

In register addressing mode, the operand is a register.

## Step 2

**Benefit:**Using register addressing mode, it is possible to jump to any 32 bit address.

For example consider the following MIPS (Million Instructions per Second) code:

add $s0, $a1, $t7

j next

In the above code after performing the addition operation, it jumps to the address 0x08000002 (say) if the address specified by next is 0x08000002. Hear, it jumped to a 32 bit address directly.

**Drawback:**There is a draw back in using register addressing mode. Since it can jump to a 32 bit address, to load the address it requires multiple cycles.

For example consider the following MIPS (Million Instructions per Second) code:

add $s0, $a1, $t7

j next

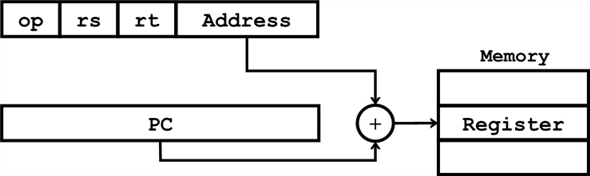
In the above code after performing the addition operation, it jumps to the address 0x08000002 (say) if the address specified by next is 0x08000002.

To load the 32 bit address it requires 4 cycles (at 8 bits per cycle).

## Step 3

**b) PC-relative addressing mode:**

The following diagram shows the PC (program counter) -relative addressing mode.



**PC – relative addressing mode**

The different fields in the register addressing mode are as follows:

• op – used to store the opcode.

• rs – used to store the address of the source register.

• rt – used to store the address of the target register.

• address – used to store the address part of the memory location.

## Step 4

**Benefit:**

Using PC – relative addressing mode it is possible to do following operations.

• Setting PC to the current PC + 4 (incrementing counter).

• Quick forward and backward of branches is possible. It is supported by the BranchAddr.

For example consider the following MIPS (Million Instructions per Second) code:

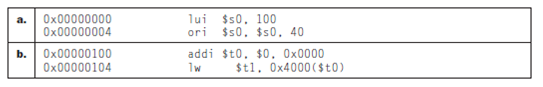
bne $t0, $t1,

In the above code, it execute the branch to PC + = 4 + label if $t0 != $t1. Based in the contents of the two registers the branch to PC will be executed.

**Drawback:**The range of address than can be represented by PC – relative addressing mode is smaller than large programs.

**Question2.27.4**

In the following problems, you will be using the MIPS assembly code as listed below to explore the tradeoffs of the immediate field in the MIPS I-type instructions.



[15] <2.10> For the MIPS statements above, show the bit-level instruction representation of each of the instructions in hexadecimal.

**Answer**

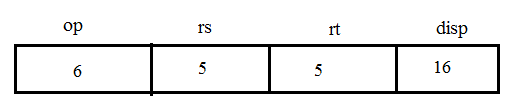
**Step 1**

a)

Consider the data:

|  |  |
| --- | --- |
| a. | 0x00000000 lui $s0,100  0x00000004 ori $s0,$s0,40 |
| b. | 0x00000100 addi $t0,$0,0x0000  0x00000104 lw $t1,0x4000($t0) |

beq is an I-format instruction, so an I-format instruction can be represented in the below format



Generally, 4-bytes are equal to 32bits.

Calculate the displacement.

The displacement can be obtained by subtracting the address just after the beg instruction from the address of the label FAR i.e.



Converting the displacement in to binary and shifting the displacement right by two bits then we get

0001 0010 0000 0000 0000 1100 0011 1111

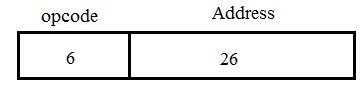
and this can be grouping by fours and converting to hexadecimal then we get

The hexadecimal number is 0x12000C3F

**Step 2**

b)

j is a J-format instruction, so J-format instruction can be represented in the below format



The address can be obtained is 0x04000000.

Converting address in to binary and shifting the address right by 2 bits then we get

0000 1000 0001 0000 0000 0000 0000 0000

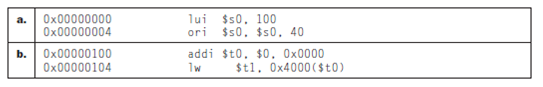
Shift by 2 bits

0000 1000 0001 0000 0000 0000 0000 0000

And this can be grouping by fours and converting to hexadecimal, then we get the hexadecimal number is 0x81000000.

# Question2.27.5

In the following problems, you will be using the MIPS assembly code as listed below to explore the tradeoffs of the immediate field in the MIPS I-type instructions.



[10] <2.10> By reducing the size of the immediate fields of the I-type and J-type instructions, we can save on the number of bits needed to represent instructions. If the immediate field of I-type instructions were 8 bits and the immediate field of J-type instructions were 18 bits, rewrite the MIPS code above to reflect this change. Avoid using the lui instruction.

# Answer

## Step 1

a) MIPS Code:

lui instruction loads upper 16 bits, which is decimal, using lui:

lui $s0, 100 # 100 decimal = 0000 0000 0110 0100

The value of register $s0 afterward is **32 bit**

0000 0000 0110 0100 0000 0000 0000 0000

But addi adds the value **8 bit** 0x80 (1000 0000) to Zero. It saves the size.

addi $s0,$zero, 0x80

sll $s0,$s0,17

ori $s0,$s0,40

## Step 2

b)MIPS Code :

addi $t0, $0, 0x0000 # it needs 16 bit data

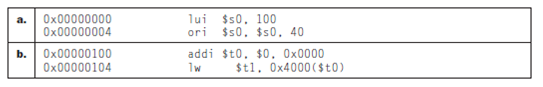
addi $t0,$0, 0x40 # It needs 8 bit to store and save the size

sll $t0,$t0,8

lw $t1,0($t0)

# Question2.27.6

In the following problems, you will be using the MIPS assembly code as listed below to explore the tradeoffs of the immediate field in the MIPS I-type instructions.



[5] <2.10> How many extra instructions are needed to execute your code in 2.27.5 MIPS statements in the table versus the code shown in the table above?

# Answer

## Step 1

a) The MIPS Code is given problem,

addi $s0,$zero, 080

sll $s0,$s0,17

ori $s0,$s0,40

Only one instruction is needed to execute the above code. There is “lui”.

Then MIPS code:

lui $s0,100

addi $s0,$zero, 080

sll $s0,$s0,17

ori $s0,$s0,40

## Step 2

b)The MIPS Code is given problem,

addi $t0,$0, 00040

sll $t0,$t0,8

lw $t1,0($t0)

Only one instruction is needed to execute the above code. There is “lui”.

Then MIPS code:

lui $s0,100

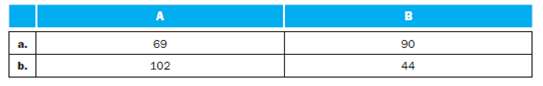
addi $t0,$0, 00040

sll $t0,$t0,8

lw $t1,0($t0)

**Question3.3.1**

Overflow occurs when a result is too large to be represented accurately given a finite word size. Underflow occurs when a number is too small to be represented correctly—a negative result when doing unsigned arithmetic, for example. (The case when a positive result is generated by the addition of two negative integers is also referred to as underflow by many, but in this textbook, that is considered an overflow). The following table shows pairs of decimal numbers.



[5] <3.2> Assume A and B are unsigned 8-bit decimal integers. Calculate A − B. Is there overflow, underflow, or neither?

**Answer**

**Step 1**

Given data:

|  |  |  |
| --- | --- | --- |
|  | A | B |
| a. | 69 | 90 |
| b. | 102 | 44 |

Given A, B are unsigned decimal numbers

**Step 2**

Case a:

A – B = 69 – 90

= -21

There will be underflow since we got a negative result while doing unsigned arithmetic.

**Step 3**

Case b:

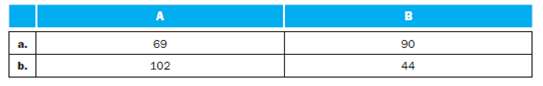
A + B = 102 -44

= 58

There won’t be any underflow, overflow since 58 can be represented using 8-bits

**Question3.3.2**

Overflow occurs when a result is too large to be represented accurately given a finite word size. Underflow occurs when a number is too small to be represented correctly—a negative result when doing unsigned arithmetic, for example. (The case when a positive result is generated by the addition of two negative integers is also referred to as underflow by many, but in this textbook, that is considered an overflow). The following table shows pairs of decimal numbers.



[5] <3.2> Assume A and B are signed 8-bit decimal integers stored in sign-magnitude format. Calculate A + B. Is there overflow, underflow, or neither?

**Answer**

**Step 1**

Given Data:

|  |  |  |
| --- | --- | --- |
|  | A | B |
| a. | 69 | 90 |
| b. | 102 | 44 |

Given A, B are 8-bit signed decimal numbers

**Step 2**

Case a:

A + B = 69 + 90

= 01000101 + 01011010

= 10011111

= -31 (sign magnitude)

There will be overflow since 69+90 = 159 cannot fit into 8bits Sign magnitude format.

**Step 3**

Case b:

A + B = 102 + 44

= 01100110 + 00101100

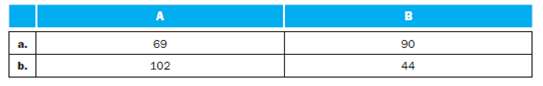
= 10010010

= -18 (sign magnitude)

There will be overflow since 102 +44 = 146 cannot fit into 8bits Sign magnitude format.

**Question3.3.3**

Overflow occurs when a result is too large to be represented accurately given a finite word size. Underflow occurs when a number is too small to be represented correctly—a negative result when doing unsigned arithmetic, for example. (The case when a positive result is generated by the addition of two negative integers is also referred to as underflow by many, but in this textbook, that is considered an overflow). The following table shows pairs of decimal numbers.



[5] <3.2> Assume A and B are signed 8-bit decimal integers stored in sign-magnitude format. Calculate A − B. Is there overflow, underflow, or neither?

**Answer**

**Step 1**

Given Data:

|  |  |  |
| --- | --- | --- |
|  | A | B |
| a. | 69 | 90 |
| b. | 102 | 44 |

Given A, B are 8-bit signed decimal numbers

**Step 2**

Case a :

A - B = 69 – 90

= -21

= 11101011 in binary

There wont be any underflow or overflow since A-B= -21 can be represented in 8-bits

**Step 3**

Case b:

A - B = 102 – 44

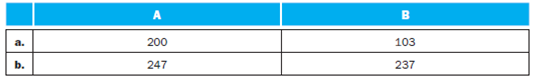
=58

= 00111010 in binary

There wont be any underflow or overflow since A-B= 58 can be represented in 8-bits

**Question3.3.4**

The following table also shows pairs of decimal numbers.



[10] <3.2> Assume A and B are signed 8-bit decimal integers stored in two’s-complement format. Calculate A + B using saturating arithmetic. The result should be written in decimal. Show your work.

**Answer**

**Step 1**

**Saturating Arithmetic Calculation**

**Case a:**

Pair of decimal numbers A and B are shown in the following table.

|  |  |
| --- | --- |
| **A** | **B** |
| 200 | 103 |

Binary value of A, 200 is 11001000. The most significant bit 1 represents the sign as negative.

It is in two’s complement form. So the actual value is obtained by the two’s complement of 200.

Two’s complement of 11001000:



Decimal value of 00111000 is -56. So the actual value in A is -56.

Binary value of B, 103 is 01100111. The most significant bit 0 represents the sign as Positive.



**Therefore, the result is 47.**

**Case b:**

Pair of decimal numbers A and B are shown in the following table.

|  |  |
| --- | --- |
| **A** | **B** |
| 247 | 237 |

Binary value of A, 247 is 11110111. The most significant bit 1 represents the sign as negative.

It is in two’s complement form. So the actual value is obtained by the two’s complement of 247.

Two’s complement of 11110111:



Decimal value of 00001001 is 9 with negative sign. So the actual value in A is -9.

Binary value of B, 237 is 11101101. The most significant bit 1 represents the sign as negative.

It is in two’s complement form. So the actual value is obtained by the two’s complement of 237.

Two’s complement of 11101101:



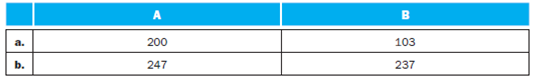
Decimal value of 00010011 is 19 with a negative sign. So the actual value in A is -19.



**Therefore, the result is -28.**

**Question3.3.5**

The following table also shows pairs of decimal numbers.



[10] <3.2> Assume A and B are signed 8-bit decimal integers stored in two’s-complement format. Calculate A − B using saturating arithmetic. The result should be written in decimal. Show your work.

**Answer**

**Step 1**

Given Data:

|  |  |  |
| --- | --- | --- |
|  | A | B |
| a. | 200 | 103 |
| b. | 247 | 237 |

Given A, B are 8-bit signed decimal numbers.

Every 8-bit signed integer must be in the range of -128 <= int <= 127

Therefore the maximum value can be 127 and the minimum value can be -128

**Step 2**

Case a:

A - B = 200 - 103

= 97 (it is in the range of -128<= int <=127, so by saturating arithmetic the result will be 97)

Therefore A- B = 97

**Step 3**

Case b:

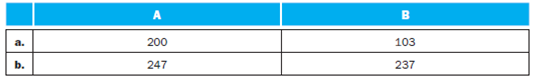
A - B = 247 - 237

= 10 it is in the range of -128<= int <=127, so by saturating arithmetic the result will be 97)

Therefore A- B = 10

**Question3.3.6**

The following table also shows pairs of decimal numbers.



[10] <3.2> Assume A and B are unsigned 8-bit integers. Calculate A + B using saturating arithmetic. The result should be written in decimal. Show your work.

**Answer**

**Step 1**

Given Data :

|  |  |  |
| --- | --- | --- |
|  | A | B |
| a. | 200 | 103 |
| b. | 247 | 237 |

Given A, B are 8-bit unsigned decimal numbers.

Every 8-bit signed integer must be in the range of 0<= int <= 255

Therefore the maximum value can be 255 and the minimum value is 0

**Step 2**

Case a:

A + B = 200 + 103

= 303 (it exceeds the maximum value (255), so by saturating arithmetic the result will be 255)

Therefore A+ B = 255

**Step 3**

Case b:

A + B = 247 + 237

= 484 (it exceeds the maximum value (255), so by saturating arithmetic the result will be 255)

Therefore A+ B = 255.

**Question3.10.1**

In a Von Neumann architecture, groups of bits have no intrinsic meanings by themselves. What a bit pattern represents depends entirely on how it is used. The following table shows bit patterns expressed in hexademical notation.



[5] <3.5> What decimal number does the bit pattern represent if it is a two’s-complement integer? An unsigned integer?

**Answer**

**Step 1**

**Convert two’s Complement Integer and Unsigned Integer into Decimal Number.**

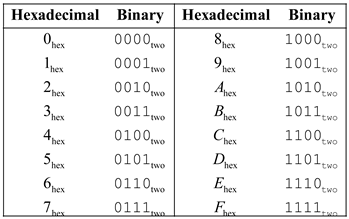
**Case a:**

Consider the following hexadecimal number:

024A60004

Hexadecimal notation can be converted into binary representation by using the following table.

The hexadecimal-binary conversion table:

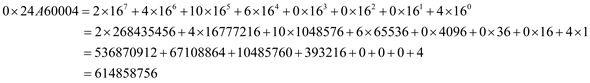


The binary representation of hexadecimal number 024A60004:

024A60004 = 0010 0100 1010 0110 0000 0000 0000 0100two.

**Step 2**

Now, the most significant bit is 0 (32nd bit is 0). So, both 2’s complement and unsigned integer represent the same decimal number. So, if it’s an unsigned integer, then write as follows:



Therefore, 2’s complement and the unsigned integer are represented in the decimal

number as 614858756.

**Step 3**

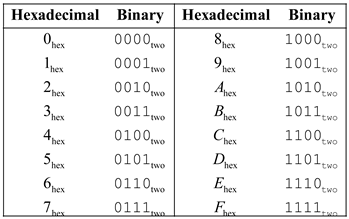
**Case b:**

Consider the following hexadecimal number:

0AFBF0000

Hexadecimal notation can be converted into binary representation by using the following table.

The hexadecimal-binary conversion table:

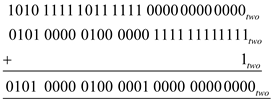


The binary representation of the hexadecimal number is 024A60004:

0AFBF0000 =1010 1111 1011 1111 0000 0000 0000two.

**Step 4**

Now, the most significant bit is 1 (32nd bit is 1). So, It’s a 2’s complement integer, then simply invert the process every 0 to 1 and every 1 to 0 and then add one to the resultant binary number.



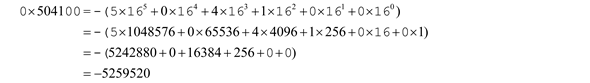
So, the 2’s complement binary representation is as follows:



Now, convert this binary representation into hexadecimal notation by using the above hexadecimal-binary conversion table. Then, get the following:



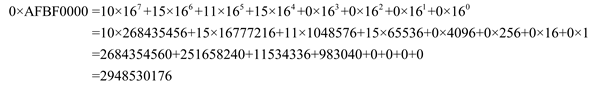
Now, convert this notation into decimal number as follows:



Therefore, if specified hexadecimal notation is a two’s complement integer, then the decimal number is **-5259520**.

**Step 5**

If the specified hexadecimal notation is an unsigned integer, then get the following:



**Therefore, if the specified hexadecimal notation is an unsigned integer, then the decimal number is 2948530176**.

**Question3.10.2**

In a Von Neumann architecture, groups of bits have no intrinsic meanings by themselves. What a bit pattern represents depends entirely on how it is used. The following table shows bit patterns expressed in hexademical notation.



[10] <3.5> If this bit pattern is placed into the Instruction Register, what MIPS instruction will be executed?

**Answer**

**Step 1**

|  |  |
| --- | --- |
| a. | 0x24A60004 |
| b. | 0xAFBF0000 |

**Step 2**

Case a:

0x24A60004 = 00100100101001100000000000000100 in binary

It can be divided into

001001 00101 00110 0000000000000100

i.e. 9 5 6 4

This represents the instruction addiu $a2, $a1, 4

**Step 3**

Case b:

0xAFBF0000 = 10101111101111110000000000000000 in binary

It can be divided into

101011 11101 11111 0000000000000000

i.e. 43 29 31 0

This represents the instruction sw $ra, 0($sp)

**Question3.10.3**

In a Von Neumann architecture, groups of bits have no intrinsic meanings by themselves. What a bit pattern represents depends entirely on how it is used. The following table shows bit patterns expressed in hexademical notation.



[10] <3.5> What decimal number does the bit pattern represent if it is a floating-point number? Use the IEEE 754 standard.

**Answer**

**Step 1**

|  |  |
| --- | --- |
| a. | 0x24A60004 |
| b. | 0xAFBF0000 |

**Step 2**

Case a:

0x24A60004 = 00100100101001100000000000000100 in binary

The 32nd bit is zero, therefore the sign is positive.

The exponent is 01001001 which is equal to 73

Therefore exponent- bias = 73- 127

= -54

**Step 3**

And the fraction is

1.01001100000000000000100 = 1+ 0.25 + 0.03125 + 0.015625+ 0.0000019073

= 1.2968769073

Therefore the floating point number is + 1.2968769073  2-54

**Step 4**

Case b:

0xAFBF0000 = 10101111101111110000000000000000 in binary

The 32nd bit is one, therefore the sign is negative

The exponent is 01011111 which is equal to 95

Therefore exponent- bias = 95- 127

= -32

**Step 5**

And the fraction is

1.01111110000000000000000 = 1+ 0.25 + 0.125 + 0.0625 + 0.03125 + 0.015625 + 0.0078125

= 1.4921875

Therefore the floating point number is - 1.4921875  2-32

**Question3.10.4**

The following table shows decimal numbers.



[10] <3.5> Write down the binary representation of the decimal number, assuming the IEEE 754 single precision format.

**Answer**

**Step 1**

|  |  |
| --- | --- |
| a. | -1609.5 |
| b. | -938.8125 |

**Step 2**

Case a:

The sign bit will be one since the given number is negative.

1609 in binary is 11001001001 and 0.5 in binary is 0.1

1609.5 can be written as 11001001001.1 in binary

This can be written as 1.10010010011  210

Now the fraction is 10010010011

**Step 3**

Therefore the exponent is 10 + 127 = 137

= 10001001 in binary

Now the binary representation is

|  |  |  |
| --- | --- | --- |
| 1 | 10001001 | 1001001001100000000000 |

**Step 4**

Case b:

The sign bit will be one since the given number is negative.

938 in binary is 1110101010 and 0.8125 in binary is 0.1101

938.8125 can be written as 1110101010.1101 in binary

This can be written as 1.1101010101101  29

Now the fraction is 1101010101101

**Step 5**

Therefore the exponent is 9 + 127 = 136

= 10001000 in binary

Now the binary representation is

|  |  |  |
| --- | --- | --- |
| 1 | 10001000 | 11010101011010000000000 |

**Question3.10.5**

The following table shows decimal numbers.



[10] <3.5> Write down the binary representation of the decimal number, assuming the IEEE 754 double precision format.

**Answer**

**Step 1**

Given data:

|  |  |
| --- | --- |
| a. | -1609.5 |
| b. | -938.8125 |

**Step 2**

Case a:

The sign bit will be one since the given number is negative.

1609 in binary is 11001001001 and 0.5 in binary is 0.1

1609.5 can be written as 11001001001.1 in binary

This can be written as 1.10010010011  210

Now the fraction is 10010010011

Therefore the exponent is 10 + 1023 = 1033

= 10000001001 in binary

**Step 3**

Now the binary representation in double precision format is

|  |  |  |
| --- | --- | --- |
| 1 | 10000001001 | 100100100110000000000000000000000000000000000000000 |

**Step 4**

Case b:

The sign bit will be one since the given number is negative.

938 in binary is 1110101010 and 0.8125 in binary is 0.1101

938.8125 can be written as 1110101010.1101 in binary

This can be written as 1.1101010101101  29

Now the fraction is 1101010101101

Therefore the exponent is 9 + 1023 = 1032

= 10000001000 in binary

**Step 5**

Now the binary representation in double precision format is

|  |  |  |
| --- | --- | --- |
| 1 | 10000001000 | 1101010101101000000000000000000000000000000000000000 |

**Question3.10.6**

The following table shows decimal numbers.



[10] <3.5> Write down the binary representation of the decimal number assuming it was stored using the single precision IBM format (base 16, instead of base 2, with 7 bits of exponent).

**Answer**

**Step 1**

Given data:

|  |  |
| --- | --- |
| a. | -1609.5 |
| b. | -938.8125 |

**Step 2**

Case a:

The sign bit will be one since the given number is negative.

1609 in binary is 11001001001 and 0.5 in binary is 0.1

1609.5 can be written as 11001001001.1 in binary.

In IBM format we have to move 4bits at a time and multiply by 16

**Step 3**

Therefore 11001001001.1 can be written as 0.0110010010011  163

Now the fraction is 01100100010011 (which has t be represented in 24 bits)

Therefore the exponent is 3 + 64 = 67

= 1000011 (which ahs to be represented in 7 bits)

**Step 4**

Now the binary representation in IBM format is

|  |  |  |
| --- | --- | --- |
| 1 | 1000011 | 011001001001100000000000 |

**Step 5**

Case b:

The sign bit will be one since the given number is negative.

938 in binary is 1110101010 and 0.8125 in binary is 0.1101

938.8125 can be written as 1110101010.1101 in binary

In IBM format we have to move 4bits at a time and multiply by 16

**Step 6**

Therefore 1110101010.1101 can be written as 0.0011101010101101  163

Now the fraction is 0011101010101101 (which has t be represented in 24 bits)

Therefore the exponent is 3 +64 = 67

= 1000011 (which has to be represented in 7 bits)

**Step 7**

Now the binary representation in IBM format is

|  |  |  |
| --- | --- | --- |
| 1 | 1000011 | 001110101010110100000000 |

**Question3.11.1**

In the IEEE 754 floating point standard the exponent is stored in “bias” (also known as “Excess-N”) format. This approach was selected because we want an all-zero pattern to be as close to zero as possible. Because of the use of a hidden 1, if we were to represent the exponent in two’s-complement format an all-zero pattern would actually be the number 1! (Remember, anything raised to the zeroth power is 1, so 1.00 = 1.) There are many other aspects of the IEEE 754 standard that exist in order to help hardware floating-point units work more quickly. However, in many older machines floating-point calculations were handled in software, and therefore other formats were used. The following table shows decimal numbers.



[20] <3.5> Write down the binary bit pattern assuming a format similar to that employed by the DEC PDP-8 (leftmost 12 bits are the exponent stored as a two’s-complement number, and the rightmost 24 bits are the mantissa stored as a two’s-complement number.) No hidden 1 is used. Comment on how the range and accuracy of this 36-bit pattern compares to the single and double precision IEEE 754 standards.

**Answer**

**Step 1**

a)

Given Decimal number =

Convert decimal number into binary form

 = 



**The following is the format of DEC PDP-8:**

Picture 1

The range of exponent in DEC PDP-8 is  to.

So, the exponent is 2-18 which is equal to. The exponent value is 2030 and its 2’s complement is 2066 and its value is 1000 0001 0010.

The mantissa is 2’s complement of. After flipping the bits the number is 0.101. Adding 1 the two’s complement is 0.110.

The DEC PDP-8 of number is as shown:

|  |  |
| --- | --- |
| **1000 0001 0010** | **0000 0101 1011 1111 1111 1110** |

**Step 2**

**The following is the format of IEEE 754 single precision:**

32-bit single precision representation of same number is



Picture 3

**Step 3**

**The following is the format of IEEE 754 double precision:**

32-bit double precision representation of same number is



Picture 5

**Step 4**

b)

Given Decimal number = -2.691650390625 x 10-2

Convert decimal number into binary form -0.02691650390625 x 100

= (-) 0.00000110111001 x 20

= 1.10111001 x 2-6

**Step 5**

**The following is the format of DEC PDP-8:**

Picture 6

The range of exponent in DEC PDP-8 is  to.

So, the exponent is 2-6 which is equal to. The exponent value is 2042 and its 2’s complement is 2054 and its value is 1000 0000 0110.

The mantissa is 2’s complement of. After flipping the bits the number is 0.01000110. Adding 1 the two’s complement is 0.01000111.

The DEC PDP-8 of number is as shown:

|  |  |
| --- | --- |
| **1000 0000 0110** | **0100 0111 0000 0000 0000 0000** |

**Step 6**

**The following is the format of IEEE 754 single precision:**

32-bit single precision representation of same number is



Picture 7

**Step 7**

**The following is the format of IEEE 754 double precision:**

32-bit double precision representation of same number is



Picture 9

# Question3.11.2

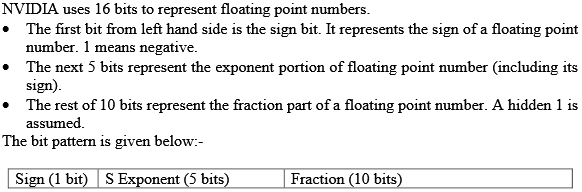
In the IEEE 754 floating point standard the exponent is stored in “bias” (also known as “Excess-N”) format. This approach was selected because we want an all-zero pattern to be as close to zero as possible. Because of the use of a hidden 1, if we were to represent the exponent in two’s-complement format an all-zero pattern would actually be the number 1! (Remember, anything raised to the zeroth power is 1, so 1.00 = 1.) There are many other aspects of the IEEE 754 standard that exist in order to help hardware floating-point units work more quickly. However, in many older machines floating-point calculations were handled in software, and therefore other formats were used. The following table shows decimal numbers.



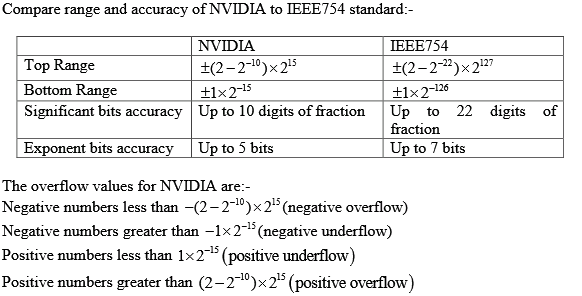
[20] <3.5> NVIDIA has a “half” format, which is similar to IEEE 754 except that it is only 16 bits wide. The leftmost bit is still the sign bit, the exponent is 5 bits wide and stored in excess-16 format, and the mantissa is 10 bits long. A hidden 1 is assumed. Write down the bit pattern assuming this format. Comment on how the range and accuracy of this 16-bit pattern compares to the single precision IEEE 754 standard.

# Answer

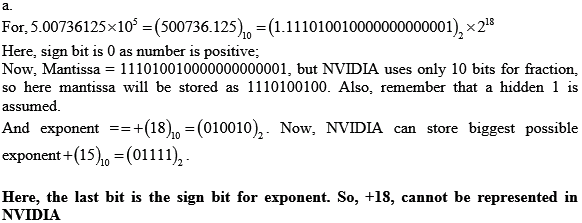
## Step 1



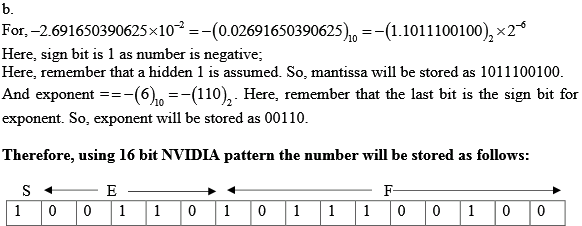
## Step 2



## Step 3



## Step 4



# Question3.11.3

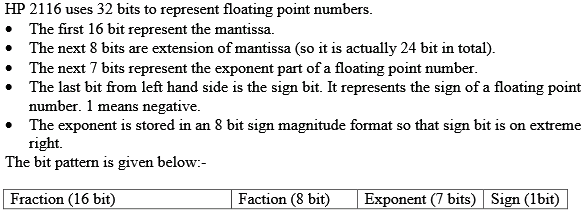
In the IEEE 754 floating point standard the exponent is stored in “bias” (also known as “Excess-N”) format. This approach was selected because we want an all-zero pattern to be as close to zero as possible. Because of the use of a hidden 1, if we were to represent the exponent in two’s-complement format an all-zero pattern would actually be the number 1! (Remember, anything raised to the zeroth power is 1, so 1.00 = 1.) There are many other aspects of the IEEE 754 standard that exist in order to help hardware floating-point units work more quickly. However, in many older machines floating-point calculations were handled in software, and therefore other formats were used. The following table shows decimal numbers.



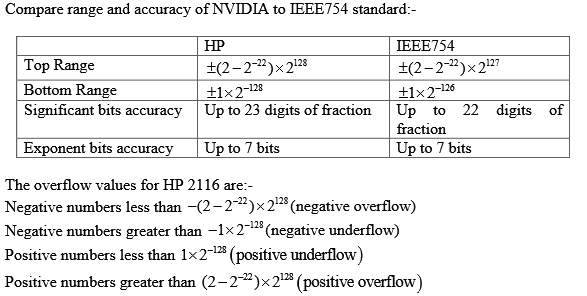
[20] <3.5> The Hewlett-Packard 2114, 2115, and 2116 used a format with the leftmost 16 bits being the mantissa stored in two’s-complement format, followed by another 16-bit field which had the leftmost 8 bits an extension of the mantissa (making the mantissa 24 bits long), and the rightmost 8 bits representing the exponent. However, in an interesting twist, the exponent was stored in sign-magnitude format with the sign bit on the far right! Write down the bit pattern assuming this format. No hidden 1 is used. Comment on how the range and accuracy of this 32-bit pattern compares to the single precision IEEE 754 standard.

# Answer

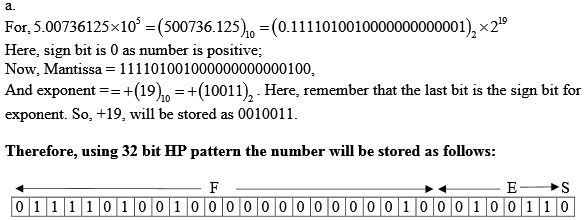
## Step 1



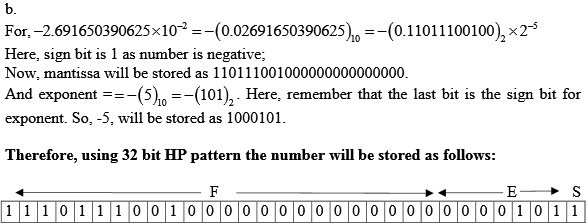
## Step 2



## Step 3



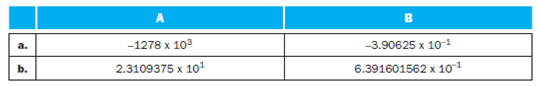
## Step 4



## Step 5

**Question3.11.4**

The following table shows pairs of decimal numbers.



[20] <3.5> Calculate the sum of A and B by hand, assuming A and B are stored in the 16-bit NVIDIA format described in Exercise 3.11.2 (and also described in the text). Assume one guard, one round bit and one sticky bit, and round to the nearest even. Show all the steps.

**Answer**

**Step 1**

NVIDIA format stores floating point numbers in following 16 bit format:-

|  |  |  |
| --- | --- | --- |
| Sign -1 bit | Exponent – 5 bits | Fraction 10 bits |

To calculate sum of A and B where they are stored in NVIDIA binary format and one bit each is reserved for guard, round and sticky bit.

The steps to add two numbers involve:-

1. Compare the exponents of two numbers. Match the two exponents by shifting the smaller number to right.

2. Add the fraction bits.

3. Normalize the above sum. This can be done by either

i. Shifting the fraction right and incrementing the exponent or

ii. Shifting the fraction left and decrementing the exponent.

4. Check for overflow or underflow. In case of either the two cannot be conclusively added.

5. Round off the fraction to the appropriate number of bits.

**Step 2**

Here,  ,

First convert the numbers to binary form and use only 10 significant bits:-



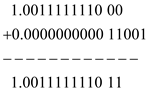
Following the step for addition;

1. Here, the smaller exponent is of B, which is -2, so shifting the fraction to right and incrementing the exponent. Now, exponent of both fractions are same:-



2. Add the fraction bits to get

GR

Remember G, R and S are Guard, Round and Sticky bit. . In the sum, Guard-1, Round -1.

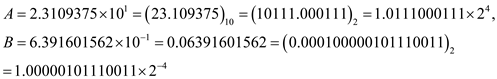
3. The sum after normalization and rounding off is



**Step 3**

Here, 

First convert the numbers to binary form and use only 10 significant bits:-



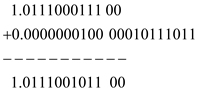
Following the step for addition;

1. Here, the smaller exponent is of B, which is -4, so shifting the fraction to right and incrementing the exponent. Now, exponent of both fractions are same:-



2. Add the fraction bits to get

GR



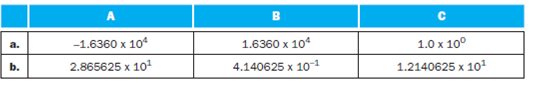
Remember G, R and S are Guard, Round and Sticky bit. In the sum, Guard-0 and Round-0.

3. The sum after normalization and rounding off is



# Question3.13.1

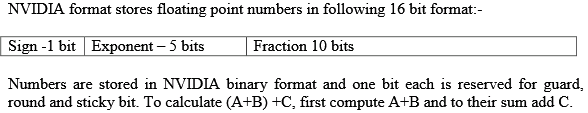
Operations performed on fixed-point integers behave the way one expects—the commutative, associative, and distributive laws all hold. This is not always the case when working with floating-point numbers, however. Let’s first look at the associative law. The following table shows sets of decimal numbers.



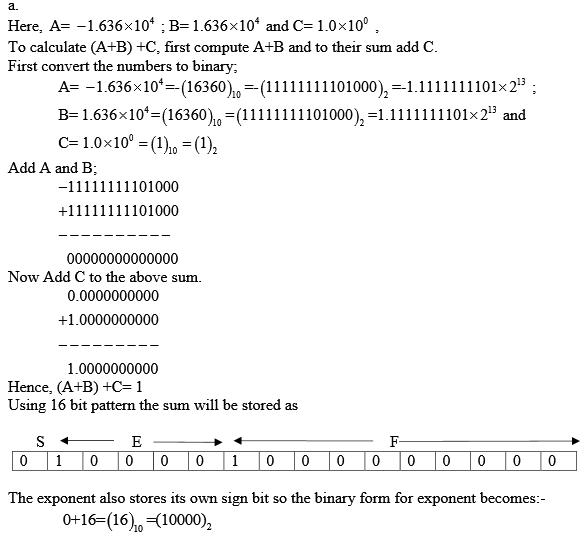
[20] <3.2, 3.5, 3.6> Calculate (A + B) + C by hand, assuming A, B, and C are stored in the 16-bit NVIDIA format described in Exercise 3.11.2 (and also described in the text). Assume one guard, one round bit and one sticky bit, and round to the nearest even. Show all the steps, and write your answer both in 16-bit floating-point format and in decimal.

# Answer

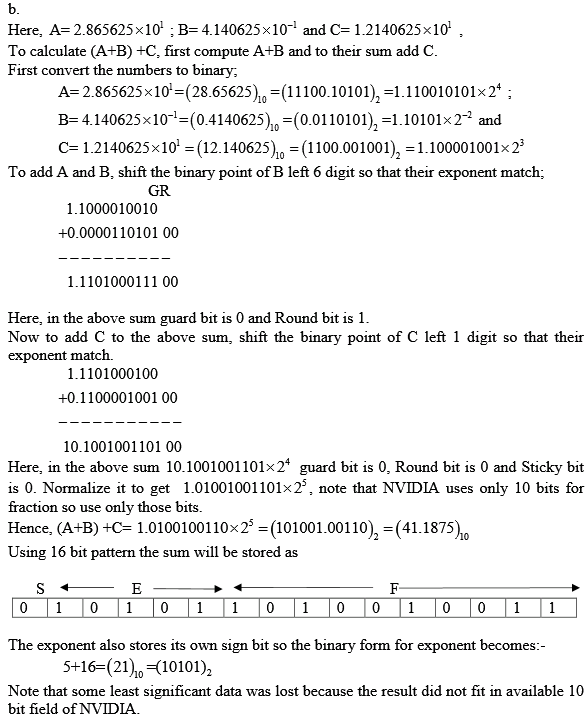
## Step 1



## Step 2

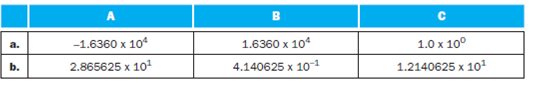


## Step 3



# Question3.13.2

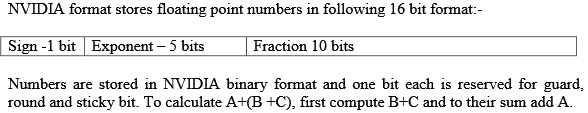
Operations performed on fixed-point integers behave the way one expects—the commutative, associative, and distributive laws all hold. This is not always the case when working with floating-point numbers, however. Let’s first look at the associative law. The following table shows sets of decimal numbers.



[20] <3.2, 3.5, 3.6> Calculate A + (B + C) by hand, assuming A, B, and C are stored in the 16-bit NVIDIA format described in Exercise 3.11.2 (and also described in the text). Assume one guard, one round bit and one sticky bit, and round to the nearest even. Show all the steps, and write your answer both in 16-bit floating-point format and in decimal.

# Answer

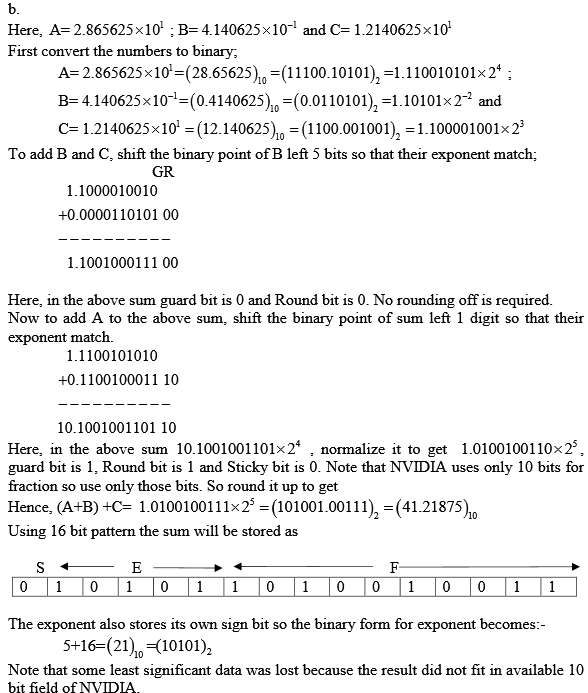
## Step 1



## Step 2

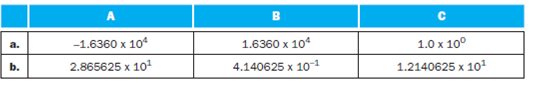


## Step 3



# Question3.13.3

Operations performed on fixed-point integers behave the way one expects—the commutative, associative, and distributive laws all hold. This is not always the case when working with floating-point numbers, however. Let’s first look at the associative law. The following table shows sets of decimal numbers.



[10] <3.2, 3.5, 3.6> Based on your answers to Exercise 3.13.1 and Exercise 3.13.2, does (A + B) + C = A + (B + C)?

# Answer

## Step 1

Here, 

The answer to (A+B)+C from solution 3.13.1E is 1, for the same values of A,B and C the answer to A+(B+C) from solution 3.13.2E is 0.

While the exact calculated value of A+B+C= 1.

No, the two calculations are not equal. Hence, the associative law does not hold true.



## Step 2

Here, 

The answer to (A+B)+C from solution 3.13.1E is 41.1875;

For the same values of A,B and C the answer to A+(B+C) from solution 3.13.2E is 41.21875.

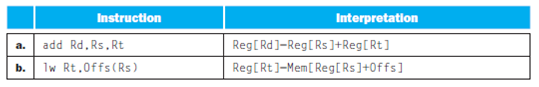
While the exact calculated value of A+B+C= 41.2109375.

No, the two calculations are not equal. Hence, the associative law does not hold true.



**Question4.1.1**

Different instructions utilize different hardware blocks in the basic single-cycle implementation. The next three problems in this exercise refer to the following instruction:



[5] <4.1> What are the values of control signals generated by the control in Figure 4.2 for this instruction?

**Answer**

**Step 1**

**Values of Control Signals**

a) The values of control signals generated by the control for the following instruction:

add Rd, Rs, Rt

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Reg-Write | Mem-Read | Mem-write | ALU-Mux | ALU-OP | Reg-Mux | Branch |
| 1 | 0 | 0 | 0(reg) | add | 0(ALU) | 0 |

In this instruction, the source register fields are Rs and Rt and the destination field is Rd.

ALUMux is the control signal that controls the Mux at the ALU input, 0 (Reg) selects the output of the register file and 1 (Immediate) selects the immediate from the instruction word as the second input to the ALU. RegMux is the control signal that controls the Mux at the Data input to the register file, 0 (ALU) selects the output of the ALU and 1 (Mem) selects the output of memory.

**Step 2**

b) The values of control signals generated by the control for the following instruction:

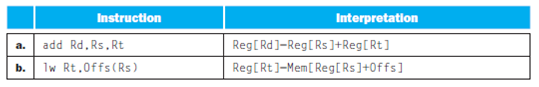
lw Rt, Offs (Rs)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Reg-Write | Mem-Read | Mem-write | ALU-Mux | ALU-OP | Reg-Mux | Branch |
| 1 | 1 | 0 | 1(Imm) | Add | 0(Mem) | 0 |

ALUMux is the control signal that controls the Mux at the ALU input, 0 (Reg) selects the output of the register file and 1 (Immediate) selects the immediate from the instruction word as the second input to the ALU. RegMux is the control signal that controls the Mux at the Data input to the register file, 0 (ALU) selects the output of the ALU and 1 (Mem) selects the output of memory. A value of X is a “don’t care” (does not matter if signal is 0 or 1).

# Question4.1.2

Different instructions utilize different hardware blocks in the basic single-cycle implementation. The next three problems in this exercise refer to the following instruction:



[5] <4.1> Which resources (blocks) perform a useful function for this instruction?

# Answer

## Step 1

728-4.1-2E SA Code: 6376

SR Code: 4578

(a)

Instruction memory, Registers, multiflexer and ALU are the resources (blocks) that perform a useful function for the instruction

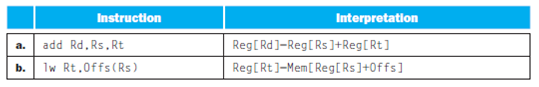
## Step 2

(b)

Instruction memory, Registers, ALU, and Data memory are the resources (blocks) that perform a useful function for the instruction

# Question4.1.3

Different instructions utilize different hardware blocks in the basic single-cycle implementation. The next three problems in this exercise refer to the following instruction:



[10] <4.1> Which resources (blocks) produce outputs, but their outputs are not used for this instruction? Which resources produce no outputs for this instruction?

# Answer

## Step 1

(a)

These blocks that are produce output but not used for instruction is: Branch Add

Resources produce no output is: Data memory.

## Step 2

(b)

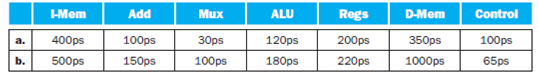
The blocks that produce output but not used for instruction are:

Branch Add, second read port of Registers.

The resources performing useful function “lw Rt, Offs(Rs)”: All blocks are producing outputs.

**Question4.1.4**

Different execution units and blocks of digital logic have different latencies (time needed to do their work). In Figure 4.2 there are seven kinds of major blocks. Latencies of blocks along the critical (longest-latency) path for an instruction determine the minimum latency of that instruction. For the remaining three problems in this exercise, assume the following resource latencies:



[5] <4.1> What is the critical path for a MIPS AND instruction?

**Answer**

**Step 1**

**Finding Critical Path for an MIPS AND Instruction**

**Case a:**

Latencies (time needed to do their work) of various blocks are as shown.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **I-Mem** | **Add** | **Mux** | **ALU** | **Regs** | **D-Mem** | **Control** |
| 400 ps | 100 ps | 30 ps | 120ps | 200 ps | 350 ps | 100 ps |

Critical path of an instruction is the longest-latency path. There are several relevant paths for the AND instruction. They are:

1) PC Add Mux PC

This path goes through the adder and a mux.

2) PC I-Mem Regs Mux ALUMux

This path reads an instruction, reads registers, goes through MUX, performs ALU operations, and writes to registers through MUX.

**Step 2**

3) PC I-Mem Control This path reads the instruction and then goes through the control block by preventing memory write.

**Step 3**

PC is ignored as it is basic and used for every instruction. Also the latency for PC is not given. Following are the latencies of the above paths:

1) PC Add Mux PC = 100 + 30 = 130 ps

2) PC I-Mem Regs Mux ALU Mux = 400+200+30+120+30

= 780 ps

3) PC I-Mem Control = 400+100 = 500 ps

Since the path, **PC I-Mem Regs Mux ALU Mux** has the longest latency of 780, it is the critical path.

**Step 4**

**Case b:**

Latencies (time needed to do their work) of various blocks are as shown.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **I-Mem** | **Add** | **Mux** | **ALU** | **Regs** | **D-Mem** | **Control** |
| 500 ps | 150 ps | 100 ps | 180 ps | 220 ps | 1000 ps | 65 ps |

Critical path of an instruction is the longest-latency path. There are several relevant paths for the AND instruction. They are:

1) PC Add Mux PC

This path goes through the adder and a mux.

2) PC I-Mem Regs Mux ALU Mux

This path reads an instruction, reads registers, goes through MUX, performs ALU operations, and writes to registers through MUX.

**Step 5**

3) PC I-Mem Control This path reads the instruction and then goes through the control block by preventing memory write.

**Step 6**

PC is ignored and the latency for PC is also not given. The latencies of the above paths:

4) PC Add Mux PC = 150 + 100 = 250ps

5) PC I-Mem Regs Mux ALU Mux = 500+220+100+180+100

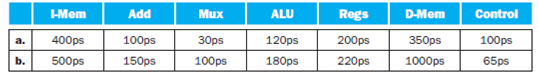
= 1100 ps

6) PC I-Mem Control = 500+65 = 565 ps

Since the path, **PC I-Mem Regs Mux ALU Mux** has the longest latency of 1100ps, it is the critical path.

# Question4.1.5

Different execution units and blocks of digital logic have different latencies (time needed to do their work). In Figure 4.2 there are seven kinds of major blocks. Latencies of blocks along the critical (longest-latency) path for an instruction determine the minimum latency of that instruction. For the remaining three problems in this exercise, assume the following resource latencies:



[5] <4.1> What is the critical path for a MIPS load (LD) instruction?

# Answer

## Step 1

728-4.1-5E SA Code: 6376

SR Code:4578

(a)

Critical path of an instruction is the longest-latency path. The long path is to read instruction, read registers, use the Mux to select the immediate as the second ALU input, use ALU (compute address), access D-Mem, and use the Mux to select that as register data input, so we have I-Mem, Regs, Mux, ALU, D-Mem, Mux. The other long path is similar, but goes through Control instead of Regs (to generate the control signal for the ALU MUX).

There are several relevant paths for the load instruction. They are:

1) PC Add Mux PC

2) PC I-Mem Regs Mux ALU D-Mem Mux

3) PC I-Mem Control

## Step 2

PC is ignored and the latency for PC is also not given. The latencies of the paths are:

1) PC Add Mux PC = 100 + 30 = 130ps

2) PC I-Mem Regs Mux ALU D-Mem Mux

= 400+200+30+120+350+30

= 1130ps

3) PC I-Mem Control = 400+100 = 500ps

## Step 3

Since the path, PC I-Mem Regs Mux ALU D-Mem Mux has the longest latency of 1130, it is the critical path.

## Step 4

(b)

Critical path of an instruction is the longest-latency path. There are several relevant paths for the load instruction. They are:

1) PC Add Mux PC

2) PC I-Mem Regs Mux ALU D-Mem Mux

3) PC I-Mem Control

## Step 5

PC is ignored and the latency for PC is also not given. The latencies of the paths are:

1) PC Add Mux PC = 150 + 100 = 250ps

2) PC I-Mem Regs Mux ALU D-Mem Mux

= 500+220+100+180+1000+100

= 2100ps

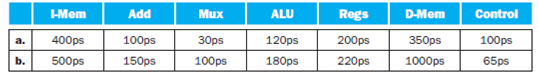
3) PC I-Mem Control = 500+65 = 565ps

## Step 6

Since the path, PC I-Mem Regs Mux ALU D-Mem Mux has the longest latency of 2100ps, it is the critical path.

# Question

Different execution units and blocks of digital logic have different latencies (time needed to do their work). In Figure 4.2 there are seven kinds of major blocks. Latencies of blocks along the critical (longest-latency) path for an instruction determine the minimum latency of that instruction. For the remaining three problems in this exercise, assume the following resource latencies:



[10] <4.1> What is the critical path for a MIPS BEQ instruction?

# Answer

## Step 1

728-4.1-6E SA Code: 6376

SR Code:4578

(a)

Critical path of an instruction is the longest-latency path. This instruction has two kinds of long paths, those that determine the branch condition and those that compute the new PC. To determine the branch condition, we read the instruction, read registers or use the Control unit, then use the ALU Mux and then the ALU to compare the two values, then use the Zero output of the ALU to control the Mux that selects the new PC.

There are several relevant paths for the BEQ instruction. They are:

1) PC Add Add Mux PC

2) PC I-Mem Regs Mux ALU

3) PC I-Mem Control

## Step 2

PC is ignored and the latency for PC is also not given. The latencies of the paths are:

1) PC Add Add Mux PC = 100 + 100 + 30 = 230ps

2) PC I-Mem Regs Mux ALU = 400+200+30+120 = 750ps

3) PC I-Mem Control = 400+100 = 500ps

## Step 3

Since the path, PC I-Mem Regs Mux ALU has the longest latency of 750ps, it is the critical path.

## Step 4

(b)

Critical path of an instruction is the longest-latency path. There are several relevant paths for the BEQ instruction. They are:

1) PC Add Add Mux PC

2) PC I-Mem Regs Mux ALU

3) PC I-Mem Control

## Step 5

PC is ignored and the latency for PC is also not given. The latencies of the paths are:

1) PC Add Add Mux PC = 150 + 150 +100 = 400ps

2) PC I-Mem Regs Mux ALU = 500+220+100+180 = 1000ps

3) PC I-Mem Control = 500+65 = 565ps

## Step 6

Since the path, PC I-Mem Regs Mux ALU has the longest latency of 1000ps, it is the critical path.

# Question4.7.1

In this exercise we examine how latencies of individual components of the datapath affect the clock cycle time of the entire datapath, and how these components are utilized by instructions. For problems in this exercise, assume the following latencies for logic blocks in the datapath:



[10] <4.3> What is the clock cycle time if the only type of instructions we need to support are ALU instructions (add, and, etc.)?

# Answer

## Step 1

728-4.7-1E SA Code: 6376

SR Code: 4578

(a)

The longest-latency path for ALU operations is through Instruction-Memory, Regs, Mux (to select ALU operand), ALU, and Mux (to select value for register write). Other path is the PC-increment path through Add (PC + 4) and Mux, which is much shorter. So, the path is I-Mem, Regs, Mux, ALU, and Mux.

The clock cycle time of ALU instruction path = I-Mem Regs Mux ALUMux

=400+200+30+120+30

= 780ps

The Cycle time of the path = Add Mux

= 100 + 30

= 130ps

Since, the path through I-Mem Regs Mux ALUMux takes longer cycle time,

Therefore, the Clock cycle time for ALU instructions = 780ps

## Step 2

(b)

The clock cycle time of ALU instruction path = I-Mem Regs Mux ALUMux

=500+220+100+180+100

= 1100ps

The clock Cycle time of path=Add Mux

= 150 + 100

= 250

Since, the path through I-Mem Regs Mux ALUMux takes longer cycle time, Therefore, the Clock cycle time for ALU instructions = 1100ps

**Question4.7.2**

In this exercise we examine how latencies of individual components of the datapath affect the clock cycle time of the entire datapath, and how these components are utilized by instructions. For problems in this exercise, assume the following latencies for logic blocks in the datapath:



[10] <4.3> What is the clock cycle time if we only had to support lw instructions?

**Answer**

**Step 1**

**Clock Cycle Time**

**Case a:**

Consider the follwoing latencies for logic blocks in the datapath:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **I-Mem** | **Add** | **Mux** | **ALU** | **Regs** | **D-Mem** | **Sign-extend** | **Shift-left-2** |
| 400 ps | 100 ps | 30 ps | 120 ps | 200 ps | 350 ps | 20 ps | 0 ps |

The longest-latency path for lw (load) is through I-Mem, Regs, Mux (to select ALU input), ALU, D-Dem, and Mux (to select what is written to register). Other paths are the PC-increment path (which is much shorter) and the path through Sign-extend unit in address computation instead of through Registers.

**Step 2**

However, Regs has a longer latency than Sign-extend, so the path is I-Mem, Regs, Mux, ALU, D-Mem, and Mux path. There are several relevant paths for the load instruction with different cycle times. They are:

1) Add Mux = 100 + 30 = 130

2) I-Mem Regs Mux ALU D-Mem Mux = 400+200+30+120+350+30

= 1130

3) I-Mem Sign-extend Mux ALU D-Mem Mux

= 400+20+30+120+350+30 = 950

Since, the path through **I-Mem Regs Mux ALUD-MemMux Regs** takes longer cycle time, it is considered as the clock cycle time for the lw instructions.

Therefore, the clock cycle time is 1130 ps.

**Step 3**

**Case b:**

Consider the follwoing latencies for logic blocks in the datapath:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **I-Mem** | **Add** | **Mux** | **ALU** | **Regs** | **D-Mem** | **Sign-extend** | **Shift-left-2** |
| 500 ps | 150 ps | 100 ps | 180 ps | 220 ps | 1000 ps | 90 ps | 20 ps |

The longest-latency path for lw (load) is through I-Mem, Regs, Mux (to select ALU input), ALU, D-Dem, and Mux (to select what is written to register). Other paths are the PC-increment path (which is much shorter) and the path through Sign-extend unit in address computation instead of through Registers.

**Step 4**

However, Regs has a longer latency than Sign-extend, so the path is I-Mem, Regs, Mux, ALU, D-Mem, and Mux path. There are several relevant paths for the load instruction with different cycle times. They are:

1) Add Mux 150 + 100 = 250ps

2) I-Mem Regs Mux ALU D-Mem Mux

=500+220+100+180+1000+100

= 2100ps

3) I-Mem Sign-extend Mux ALU D-Mem Mux

=500+90+100+180+1000+100

= 1970ps

Since, the path through **I-Mem Regs Mux ALUD-MemMux** takes longer cycle time, it is considered as the clock cycle time for the lw instructions.

Therefore, the clock cycle time is 2100 ps.

**Question4.7.3**

In this exercise we examine how latencies of individual components of the datapath affect the clock cycle time of the entire datapath, and how these components are utilized by instructions. For problems in this exercise, assume the following latencies for logic blocks in the datapath:



[20] <4.3> What is the clock cycle time if we must support add, beq, lw, and sw instructions?

**Answer**

**Step 1**

**Finding Clock Cycle Time**

**Case a:**

Latencies (time needed to do their work) of various blocks are tabulated as follows:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **I-Mem** | **Add** | **Mux** | **ALU** | **Regs** | **D-Mem** | **Sign-extend** | **Shift-left-2** |
| 400 ps | 100 ps | 30 ps | 120 ps | 200 ps | 350 ps | 20 ps | 0 ps |

The clock cycle time is the longest latency path. Then, consider the longest paths followed by each of the instructions are:

Add: I-Mem Regs Mux ALUMux

The clock cycle time for Add = 400+200+30+120+30 = 780 ps

Beq: I-Mem Regs Mux ALU

The clock cycle time for Beq= 400+200+30+120 = 750 ps

**Step 2**

LW: I-Mem Regs Mux ALU D-Mem Mux

The clock cycle time for LW = 400+200+30+120+350+30 = 1130 ps

Sw: I-Mem Regs Mux ALU D-Mem

The clock cycle time for SW = 400+200+30+120+350 = 1100 ps

The LW instruction has the longest critical path. The longest path for sw is shorter by one Mux latency (no write to register), and the longest path for add or beq is shorter by one D-Mem latency.

So, the clock cycle time to support add, beq, lw and sw instruction is 1130 ps.

**Step 3**

**Case b:**

Latencies (time needed to do their work) of various blocks are tabulated as follows:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **I-Mem** | **Add** | **Mux** | **ALU** | **Regs** | **D-Mem** | **Sign-extend** | **Shift-left-2** |
| 500 ps | 150 ps | 100 ps | 180 ps | 220 ps | 1000 ps | 90 ps | 20 ps |

The clock cycle time is the longest latency path. Then, consider the longest paths followed by each of the instructions are:

Add: I-Mem Regs Mux ALUMux

The clock cycle time for Add = 500+220+100+180+100 = 1100 ps

Beq: I-Mem Regs Mux ALU

The clock cycle time for Beq=500+220+100+180 = 1000 ps

**Step 4**

LW: I-Mem Regs Mux ALU D-Mem Mux

The clock cycle time for LW=500+220+100+180+1000+100 = 2100 ps

Sw: I-Mem Regs Mux ALU D-Mem

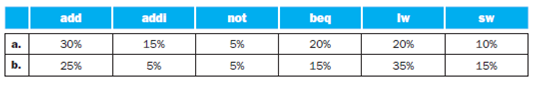
The clock cycle time for SW =500+220+100+180+1000 = 2000 ps

The LW instruction has the longest critical path. The longest path for sw is shorter by one Mux latency (no write to register), and the longest path for add or beq is shorter by one D-Mem latency.

So, the clock cycle time to support add, beq, lw and sw instruction is 2100 ps.

# Question4.7.4

For the remaining problems in this exercise, assume that there are no pipeline stalls and that the breakdown of executed instructions is as follows:



[10] <4.3> In what fraction of all cycles is the data memory used?

# Answer

## Step 1

728-4.7-4E SA Code: 6376

SR Code:4578

(a)

Of all instruction cycles, Data memory is used only for lw and sw instructions.

Fraction = 20% (lw) + 10% (sw)

= 30%

## Step 2

(b)

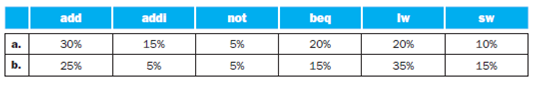
Of all instruction cycles, Data memory is used only for lw and sw instructions.

Fraction = 35% (lw) + 15% (sw)

= 50%

# Question4.7.5

For the remaining problems in this exercise, assume that there are no pipeline stalls and that the breakdown of executed instructions is as follows:



[10] <4.3> In what fraction of all cycles is the input of the sign-extend circuit needed? What is this circuit doing in cycles in which its input is not needed?

# Answer

## Step 1

728-4.7-5E SA Code: 6376

SR Code: 4578

(a)

The sign-extend circuit is actually computing a result in every cycle, but its output is ignored for add and not instructions. The input of the Sign-extend circuit is needed by the instructions addi, beq, lw and store. Arithmetic immediate operands are sign extended to 32 bits, beq uses sign extend unit to compute the branch target address, lw and sw instructions compute a memory address by adding the base register to the 16-bit signed offset field.

The Fraction of all cycles is the input of the sign-extend circuit

= 15% + 20% + 20% + 10%

= 65%

## Step 2

(b)

The sign-extend circuit is actually computing a result in every cycle, but its output is ignored for add and not instructions. The input of the Sign-extend circuit is needed by the instructions addi, beq, lw and store. Arithmetic immediate operands are sign extended to 32 bits, beq uses sign-extend unit to compute the branch target address and lw and sw instructions compute a memory address by adding the base register to the 16-bit signed offset field.

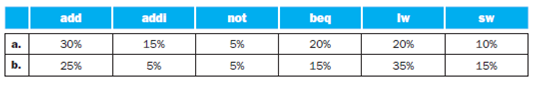
The Fraction of all cycles is the input of the sign-extend circuit

= 5% + 15% + 35% + 15%

= 70%

# Question4.7.6

For the remaining problems in this exercise, assume that there are no pipeline stalls and that the breakdown of executed instructions is as follows:



[10] <4.3> If we can improve the latency of one of the given datapath components by 10%, which component should it be? What is the speed-up from this improvement?

# Answer

## Step 1

728-4.7-6E SA Code: 6376

SR Code: 4578

(a)

The clock cycle time is determined by the critical path for the instruction that has the longest critical path. This is the LW instruction, and its critical path goes through I-Mem, Regs, Mux, ALU, D-Mem, and Mux.Since the clock cycle time depends on the latency of the critical path instruction, improving the latency of such block will improve the performance.

Of all the instructions, lw takes the longest path that goes through I-Mem, Regs, Mux, ALU, D-Mem, and Mux.I-Mem has the longest latency among all the blocks. So we improve its latency by 10%

Given latency of I-Mem = 400ps

10% of latency of I-Mem = 40

Improved latency = 400 – 40 = 360ps

Clock cycle time before improvement = 400 + 200 + 30 + 120 + 350 + 30

= 1130ps

Clock cycle time before improvement = 360 + 200 + 30 + 120 + 350 + 30

= 1090ps

Speed-up achieved = 1130 / 1090

= 1.03

## Step 2

(b)

The clock cycle time is determined by the critical path for the instruction that has the longest critical path. This is the LW instruction, and its critical path goes through I-Mem, Regs, Mux, ALU, D-Mem, and Mux.Since the clock cycle time depends on the latency of the critical path instruction, improving the latency of such block will improve the performance.

Of all the instructions, LW takes the longest path that goes through I-Mem, Regs, Mux, ALU, D-Mem, and Mux.D-Mem has the longest latency among all the blocks. So we improve its latency by 10%

Given latency of D-Mem = 1000ps

10% of latency of D-Mem = 100

Improved latency = 1000 – 100 = 900ps

Clock cycle time before improvement = 500 + 220 + 100 + 180 + 1000 + 100

= 2100ps

Clock cycle time before improvement = 500 + 220 + 100 + 180 + 900 + 100

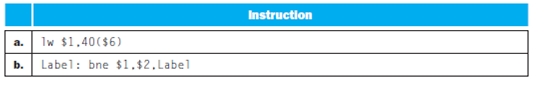
= 2000ps

Speed-up achieved = 2100 / 2000

= 1.05

# Question4.9.1

In this exercise we examine the operation of the single-cycle datapath for a particular instruction. Problems in this exercise refer to the following MIPS instruction:



[10] <4.4> What is the value of the instruction word?

# Answer

## Step 1

**Value of Instruction Code**

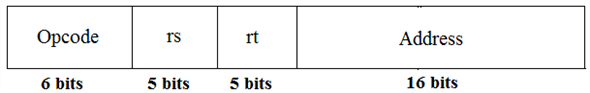
**Case a:**

Consider the following instruction:

lw $1, 40($6)

Main processor instructions that do not require a target address, immediate value, or branch displacement use an I-type coding format. This format has fields for specifying of up to three registers and a shift amount. For instructions that do not use all of these fields, the unused fields are coded with all 0 bits. All I-type instructions use a 000000 opcode. The operation is specified by the function field. The I-type instruction format for the lw instruction is:

**I-type instruction format:**



## Step 2

The instruction is:

lw $1, 40($6)

• Here, the opcode of lw is 35.

• The source register value is 6.

• The destination register value is 1.

• The shift amount is 40.

Decimal representation is as shown:



The binary representation of the fields is:



So, the value of the instruction word is 100011 00110 00001 0000000000101000

Therefore, the Hexadecimal representation of the value is 8CC10028.

## Step 3

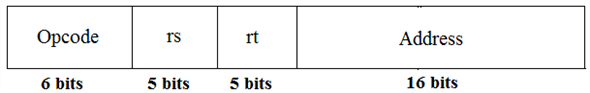
**Case b:**

Consider the following instruction:

Label: bne $1, $2, Label

The I-type instruction format for the instruction Label: bne $1, $2, Label is

**I-type instruction format:**



The instruction is:

Label: bne $1, $2, Label

• Here, the opcode of bne is 5.

• The source register value is 1.

• The destination register value is 2.

Decimal representation is as shown:



The binary representation of the fields is:

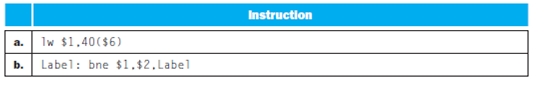


So, the value of the instruction word is 000101 00001 00010 1111111111111111

Therefore, the Hexadecimal representation of the value is 1422FFFF.

# Question4.9.2

In this exercise we examine the operation of the single-cycle datapath for a particular instruction. Problems in this exercise refer to the following MIPS instruction:



[10] <4.4> What is the register number supplied to the register file’s “Read register 1” input? Is this register actually read? How about “Read register 2”?

# Answer

## Step 1

728-4.9-2E SA Code: 6376

SR Code: 4578

(a)

Consider the instruction,

lw $1, 40 ($6)

Here, read register 1 is $6. So the register number supplied is 6(00110). This register is actually read and used. The second register is $1.the register number supplied is 1(00001). This register is actually read, but not used.

## Step 2

(b)

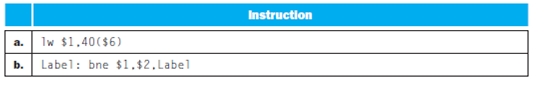
Let us consider the instruction,

Label: bne $1, $2, Label

Here, read register 1 is $1. So the register number supplied is 1(00001). This register is actually read. The second register is $2.the register number supplied is 2(00010). This register is actually read and used.

# Question4.9.3

In this exercise we examine the operation of the single-cycle datapath for a particular instruction. Problems in this exercise refer to the following MIPS instruction:



[10] <4.4> What is the register number supplied to the register file’s “Write register” input? Is this register actually written?

# Answer

## Step 1

728-4.9-3E SA Code: 6376

SR Code: 4578

(a)

Consider the instruction,

lw $1, 40 ($6)

Here, write register is $1. So, the register number supplied is 1(00001). This register is actually written and used.

## Step 2

(b)

Consider the instruction,

Label: bne $1, $2, Label

Here, write register is $1. So, the register number supplied is 1, but for this instruction, it may also be written to RegDst because the value of this register for bne is X (don’t care; it may be 1 or 0). So the register number supplied can be even 31(Op 5 of control unit)

No. This register is not actually written. Since, the two values in the registers will be compared and depending on the result the branch occurs.

# Question4.9.4

Different instructions require different control signals to be asserted in the datapath. The remaining problems in this exercise refer to the following two control signals from Figure 4.24:



[20] <4.4> What is the value of these two signals for this instruction?

# Answer

## Step 1

728-4.9-4E SA Code: 6376

SR Code: 4578

RID: 146

(a)

The values of control signals for the instructions are:

RegDst = 0

MemRead =1

## Step 2

(b)

The values of control signals for the instructions are:

RegWrite = 0;

MemRead = 0

**Question4.9.6**

Different instructions require different control signals to be asserted in the datapath. The remaining problems in this exercise refer to the following two control signals from Figure 4.24:



[20] <4.4> Repeat Exercise 4.9.5, but now implement both of these signals.

**Answer**

**Step 1**

728-4.9-6E SA Code: 6376

SR Code: 4578

(a)

The truth table for the instructions are add, lw, sw, beq, j (jump) with their respective input and output values.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Input or Output | Signal Name | add | lw | sw | beq | Jump |
| Inputs | Op5 | 0 | 1 | 1 | 0 | 0 |
|  | Op4 | 0 | 0 | 0 | 0 | 0 |
|  | Op3 | 0 | 0 | 0 | 0 | 0 |
|  | Op2 | 0 | 0 | 1 | 1 | 0 |
|  | Op1 | 0 | 1 | 1 | 0 | 1 |
|  | Op0 | 0 | 1 | 1 | 0 | 0 |
| Outputs | RegDst | 1 | 0 | X | X | X |
|  | MemRead | 0 | 1 | 0 | 0 | 0 |

**Step 2**

The inputs are taken with respect to the opcodes of the instructions. Their opcodes are :

Add – 0 (000000),

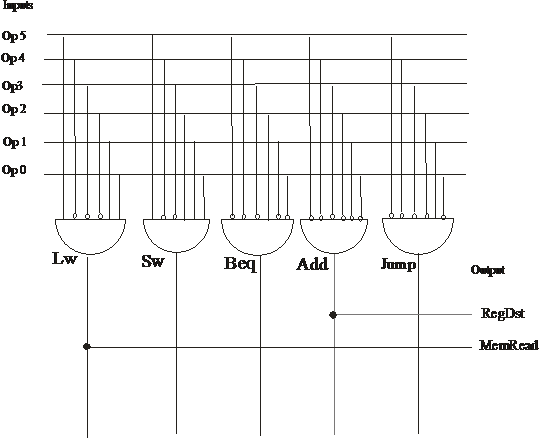
lw – 35 (100011),

sw – 43 (100111),

beq – 4 (000100),

j – 2 (000010)

Using the output values of the instructions in the truth table, the circuit implementation of the control unit for the given signals RegDst and MemRead.



**Step 3**

(b)

The truth table for the instructions adds, lw, sw, beq, j (jump) with their respective input and output values.

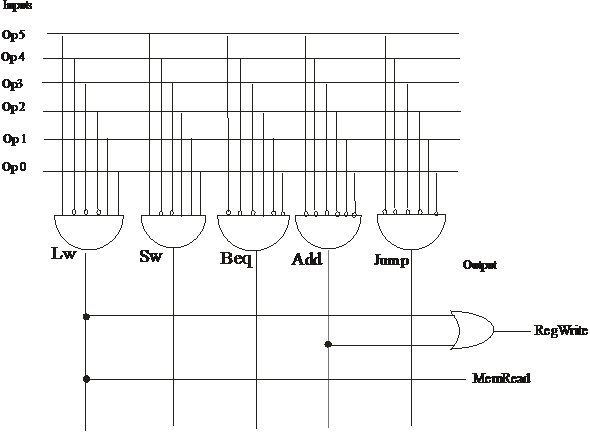
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Input or Output | Signal Name | add | lw | sw | beq | Jump |
| Inputs | Op5 | 0 | 1 | 1 | 0 | 0 |
|  | Op4 | 0 | 0 | 0 | 0 | 0 |
|  | Op3 | 0 | 0 | 0 | 0 | 0 |
|  | Op2 | 0 | 0 | 1 | 1 | 0 |
|  | Op1 | 0 | 1 | 1 | 0 | 1 |
|  | Op0 | 0 | 1 | 1 | 0 | 0 |
| Outputs | RegWrite | 1 | 1 | 0 | 0 | 0 |
|  | MemRead | 0 | 1 | 0 | 0 | 0 |

**Step 4**

The inputs are taken with respect to the opcodes of the instructions. Their opcodes are :

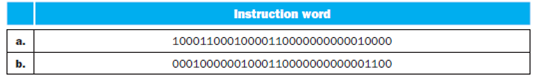
Add – 0 (000000), lw – 35 (100011), sw – 43 (100111), beq – 4 (000100), j – 2 (000010)

Using the output values of the instructions in the truth table, the circuit implementation of the control unit for the given signal RegWrite and MemRead.



# Question4.11.1

In this exercise we examine in detail how an instruction is executed in a single-cycle datapath. Problems in this exercise refer to a clock cycle in which the processor fetches the following instruction word:



[5] <4.4> What are the outputs of the sign-extend and the jump “Shift left 2” unit (in the upper left of Figure 4.24) for this instruction word?

# Answer

## Step 1

728-4.11-1E SA Code: 4475

SR Code: 4578

a)

The given instruction word is 32 bits long. In instruction word, 0-15 bits are 0000000000010000.The output of the sign-extend unit extends the sequence to32(16+16) bits, which is 00000000000000000 000000000010000.Bits 0-25 in instruction word is 00010000110000000000010000.The jump “Shift left 2” shifts the sequence left by 2 bits. The output of the jump “Shift left 2” is 0001000011000000000001000000 i.e. 28 bits

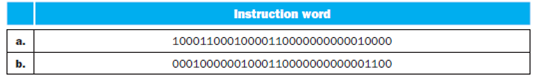
## Step 2

b)

  The given instruction word is 32 bits long. In instruction word, 0-15 bits are 0000000000001100.The output of the sign-extend unit extends the sequence to32(16+16) bits, which is 00000000000000000 0000000000001100 .Bits 0-25 in instruction word is 0001000110000000000001100.The jump “Shift left 2” shifts the sequence left by 2 bits. The output of the jump “Shift left 2” is 0000100011000000000000110000 i.e. 28 bits

# Question4.11.2

In this exercise we examine in detail how an instruction is executed in a single-cycle datapath. Problems in this exercise refer to a clock cycle in which the processor fetches the following instruction word:



[10] <4.4> What are the values of ALU control unit’s inputs for this instruction?

# Answer

## Step 1

a)

The given instruction word is 10001100010000110000000000010000

Here, the opcode = 100011.This can be representing lw instruction. The main control unit generates the ALUOp bits, which are used as input to the ALU control that generates the actual signals to control the ALU unit.

Here ALU control inputs are based on the 2-bit ALUOp control and the 6-bit function field. The ALUOp is depends on opcode and function field.

The values of 2-bit ALUOp are 00 because the opcode represent the lw instruction and the 6-bit function field is 010000 from instruction word.

## Step 2

b)

The given instruction word is 00010000001000110000000000001100

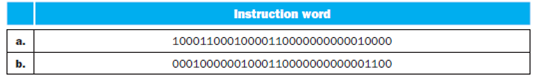
Here, the opcode = 000100.This can be representing bne instruction. The main control unit generates the ALUOp bits, which are used as input to the ALU control that generates the actual signals to control the ALU unit.

Here ALU control inputs are based on the 2-bit ALUOp control and the 6-bit function field. The ALUOp is depends on opcode and function field.

The values of 2-bit ALUOp are 01 because the opcode represent the bne instruction and the 6-bit function field is 001100 from instruction word.

# Question4.11.3

In this exercise we examine in detail how an instruction is executed in a single-cycle datapath. Problems in this exercise refer to a clock cycle in which the processor fetches the following instruction word:



[10] <4.4> What is the new PC address after this instruction is executed? Highlight the path through which this value is determined.

# Answer

## Step 1

For a 32 bit instruction word. The format is such that

• The first 6 bits are reserved for opcode. Is bits 31:26, which are referred as Op [5:0].

• The 10 bits are for register, called rt and rs fields at position 25:21 and 20:16. The base register for load and store instructions is always rs.

• Branch equal, load and store instructions use the 16–bit offset from position 15:0.

• For an R- type instruction

• For load instructions, the destination register is in bit position 20:16 (rt). For an R-type instruction it is in bit position 15:11 (rd).

## Step 2

Here, the instruction word fetched by the processor in a single clock-cycle data path is:- 10001100010000110000000000010000

• The first 6 bits for opcode are 100011 i.e. decimal 35. So, the given instruction is a load instruction.

• The common operations for load instruction in a clock data path consist of

• Fetch the Instruction from memory.

• Update the program counter (PC) in sequential code manner. This means PC => PC + 4. At this stage it is not known what instruction type is it until it is interpreted. So, all instructions initially increment the PC.

• Read the value of register from register file. ALU computes the sum of register and the offset.

• Address for the data memory is this calculated sum from ALU.

• The data from the memory unit is written in the register destination given by bits (20:16) of the instruction.

• So, the new PC address after execution of the given instruction is PC + 4.

• Also, the Path through which the above value is determined consist of PC to Add (PC+4) to branch Mux or to jump Mux to PC.

## Step 3

Here, the instruction word fetched by the processor in a single clock-cycle data path is:- 00010000001000110000000000001100

• The first 6 bits for opcode are 000100 i.e. decimal 4. So, the given instruction is a conditional branch.

• The common operations for branch instruction in a clock data path consist of

o Fetch the Instruction from memory and increment the PC.

o Read two registers $t1 and $t2 from register file.

o ALU subtract the data values read above. Add the value of PC+4 to the offset, shifted left by two to get the branch target address.

o The result of subtraction in ALU decides which adder result is to be stored in PC.

• In a branch instruction, the PC might not take the value PC + 4. To decide whether the branch condition has been met, the given instruction makes use of the ALU, to perform a subtraction on the two register arguments. If the result is not zero, the program takes the branch, which means PC => PC + 4 + o◊set. If the two register values are equal, then the program continues to the next instruction i.e. PC => PC + 4.

• Instructions are word aligned in memory, therefore, MIPS considers the two lowest order bits will have the value 0. Thus, in the 16 bit address ◊eld of a conditional branch, MIPS doesn’t waste these two bits by always setting them to 00. Instead, the 16 bit ◊eld represents an o◊set of 4 times the o◊set representation in binary.

• In this case, if $1 and $3 are not equal, then PC=> PC+4. But, if $1 and $3 are equal, than 

• Path through which the above value is determined consist of PC to Add (PC+4) to branch Mux, or PC to Add (add offset) to branch Mux (depending on the condition). After that go through jump Mux and then into the PC.

**Question4.11.4**

The remaining problems in this exercise assume that data memory is all-zeros and that the processor’s registers have the following values at the beginning of the cycle in which the above instruction word is fetched:



[10] <4.4> For each Mux, show the values of its data output during the execution of this instruction and these register values.

**Answer**

**Step 1**

**Case a:**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | $0 | $1 | $2 | $3 | $4 | $5 | $6 | $8 | $12 | $31 |
| a. | 0 | 1 | 2 | 3 | -4 | 5 | 6 | 8 | 1 | -32 |
| b. | 0 | -16 | -2 | -3 | 4 | -10 | -6 | -1 | 8 | -4 |

The different types of Muxes are

1. WriteRegister Mux

2. ALU Mux

3. Memory Mux

4. Branch Mux

5. Jump Mux

The values of its data output during the execution of this instructions and these register values are

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| WrReg Mux | ALU Mux | Mem Mux | Branch Mux | Jump Mux |
| 3 | 16 | 0 | PC+4 | PC+4 |

In the above table, the WrRegMux contains the register on the write register input is written with the value on the write data input that is 3 from the above table.

The ALU Mux is the sign extended lower 16 bits of the instruction.

The Mem Mux is the value fed to the register write data input comes from the data memory that is 0 from the above table.

The Branch Mux is the PC is replaced by the output of the adder that computes the value of PC+4.

Similar the jump Mux is the PC jump by the output of the adder that computes the value of PC+4.

**Step 2**

**Case b:**

The values of its data output during the execution of this instructions and these register values are

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| WrReg Mux | ALU Mux | Mem Mux | Branch Mux | Jump Mux |
| 0 or 3 | -3 | X | PC+4 | PC+4 |

In the above table, the WrRegMux contains the register on the write register input is written with the value on the write data input that is 3 from the above table.

The ALU Mux is the sign extended lower 16 bits of the instruction.

The Mem Mux is the value fed to the register write data input comes from the data memory that is 0 from the above table.

The Branch Mux is the PC is replaced by the output of the adder that computes the value of PC+4.

Similar the jump Mux is the PC jump by the output of the adder that computes the value of PC+4.

**Question4.11.5**

The remaining problems in this exercise assume that data memory is all-zeros and that the processor’s registers have the following values at the beginning of the cycle in which the above instruction word is fetched:



[10] <4.4> For the ALU and the two add units, what are their data input values?

**Answer**

**Step 1**

**Case a:**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | $0 | $1 | $2 | $3 | $4 | $5 | $6 | $8 | $12 | $31 |
| a. | 0 | 1 | 2 | 3 | -4 | 5 | 6 | 8 | 1 | -32 |
| b. | 0 | -16 | -2 | -3 | 4 | -10 | -6 | -1 | 8 | -4 |

The different types of ALU units are

1. ALU

2. Add (PC+4)

3. Add (Branch)

The values of its data output during the execution of this instructions and these register values are

|  |  |  |
| --- | --- | --- |
| ALU | Add(PC + 4) | Add (branch) |
| 2 and 16 | PC and 4 | PC+4 and 16 4 |

The ALU Mux is the sign extended lower 16 bits of the instruction.

Similar the jump Mux is the PC jump by the output of the adder that computes the value of PC+4.

The Branch Mux is the PC replaced by the output of the adder that computes the value of PC+4.

**Step 2**

**Case b:**

The values of its data output during the execution of this instructions and these register values are

|  |  |  |
| --- | --- | --- |
| ALU | Add(PC + 4) | Add (branch) |
| -16 and -3 | PC and 4 | PC+4 and 124 |

The ALU Mux is the sign extended lower 16 bits of the instruction.

The Branch Mux is the PC is replaced by the output of the adder that computes the value of PC+4.

Similar the jump Mux is the PC jump by the output of the adder that computes the value of PC+4.

**Question4.11.6**

The remaining problems in this exercise assume that data memory is all-zeros and that the processor’s registers have the following values at the beginning of the cycle in which the above instruction word is fetched:



[10] <4.4> What are the values of all inputs for the “Registers” unit?

**Answer**

**Step 1**

**Case a:**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | $0 | $1 | $2 | $3 | $4 | $5 | $6 | $8 | $12 | $31 |
| a. | 0 | 1 | 2 | 3 | -4 | 5 | 6 | 8 | 1 | -32 |
| b. | 0 | -16 | -2 | -3 | 4 | -10 | -6 | -1 | 8 | -4 |

The different types registers are

1. Read Register1

2. Read Register2

3. Write Register

4. Write data

5. RegWrite

All input values of the “Register” units are

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Read Register1 | Read Register2 | Write Register | Write data | RegWrite |
| 2 | 3 | 3 | 0 | 1 |

The ALU Mux is the sign extended lower 16 bits of the instruction.

The Branch Mux is the PC is replaced by the output of the adder that computes the value of PC+4.

Similar the jump Mux is the PC jump by the output of the adder that computes the value of PC+4.

**Step 2**

**Case b:**

All input values of the “Register” units are

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Read Register1 | Read Register2 | Write Register | Write data | RegWrite |
| 1 | 3 | X(3 or 0) | X | 0 |

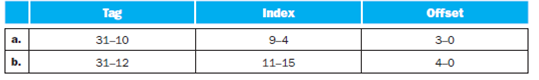
The ALU Mux is the sign extended lower 16 bits of the instruction.

The Branch Mux is the PC is replaced by the output of the adder that computes the value of PC+4.

Similar the jump Mux is the PC jump by the output of the adder that computes the value of PC+4.

# Question5.4.1

For a direct-mapped cache design with 32-bit address, the following bits of the address are used to access the cache.



[5] <5.2> What is the cache line size (in words)?

# Answer

## Step 1

a) The cache line size = 

= 

= 16 bytes

Cache line size = 4 words.

## Step 2

b) The cache line size = 

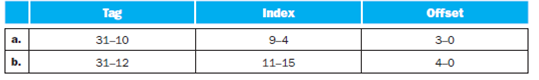
= 

= 32 bytes

Cache line size = 8 words.

# Question5.4.2

For a direct-mapped cache design with 32-bit address, the following bits of the address are used to access the cache.



[5] <5.2> How many entries does the cache have?

# Answer

## Step 1

A memory address of the word in the cache block is uniquely specified by the index and tag fields of the cache block.

• Index is used to select specific block of the cache.

• Tag is used to compare the value of the tag filed of cache.

An index field is used as address to the cache. As *n*-bit field has 2nentries, the cache has 2indexnumber of entries.

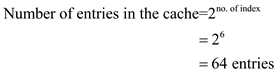
Therefore, the formula used to determine the number of entries in the cache is 2index bits.

## Step 2

**a.**

consider the given information, the index bits are from 4-9.

Therefore, the number of index bits are 6. Substitute the index bits on the power of 2 and determine the number of entries of the cache as shown below:



**The total number of entries in the cache are 64.**

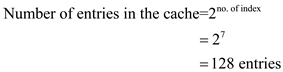
## Step 3

**b.**

**Note: The number of index bits are from 11-5. But it is misprinted as 11-15 in the textbook. Please consider it as 11-5 and solve the answer.**

consider the given information, the index bits are from 11-5.

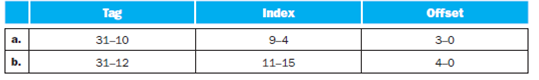
Therefore, the number of index bits are 7. Substitute the index bits on the power of 2 and determine the number of entries of the cache as shown below:



**The total number of entries in the cache are 128** **.**

# Question5.4.3

For a direct-mapped cache design with 32-bit address, the following bits of the address are used to access the cache.



[5] <5.2> What is the ratio between total bits required for such a cache implementation over the data storage bits?

# Answer

## Step 1

The total number of bits in a direct-mapped cache = 

Storage bits = 

Where n = index,

Number of words in the block = 

## Step 2

a)  n = 6 (index  9- 4)

= 

= 4 words (as m is no of words in the block)

=   (4 words)

Therefore m = 2

## Step 3

The total number of bits in a direct-mapped cache = 

= 

= 

= 

= 9664 bits

## Step 4

Storage bits = 

= (offset 3-0, offset should be taken in 2 powers)

= 

= 

= 8192 bits

## Step 5

Ratio = total bits / storage bits

= 9664/8192

= 1.179

## Step 6

b)  n = 7 (index 11- 5)



=   (8 words)

Therefore m = 3

## Step 7

The total number of bits in a direct-mapped cache = 

= 

= 

= bits

= 35456 bits

## Step 8

Storage bits = 

= 

= 

= 

= 32768 bits

## Step 9

Ratio = total bits / storage bits

= 35456/32768

= 1.08

**Question5.4.4**

Starting from power on, the following byte-addressed cache references are recorded.



[10] <5.2> What is the hit ratio?

**Answer**

**Step 1**

Consider a direct-mapped cache consisting of 32-bit address.

After the power is on, the recorded byte-addresses cache is as follows:

|  |
| --- |
| **Address** |
| 0 |
| 4 |
| 16 |
| 132 |
| 232 |
| 160 |
| 1024 |
| 30 |
| 140 |
| 3100 |
| 180 |
| 2180 |

**Step 2**

**a.**

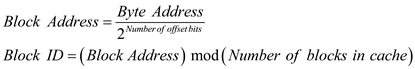
Consider the bit pattern to access the cache is as follows:

|  |  |  |
| --- | --- | --- |
| **Tag** | **Index** | **Offset** |
| 31-10 | 9-5 | 4-0 |

Number of blocks = 

Number of offset bits = 5

Using the following formulae, calculate the block address and block ID as follows:



The 32-bit binary address for the byte-address cache is as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Byte Address** | **Block address** | **Block ID** | **Hit/Miss** |
| 0 | 0 | 0 | Miss |
| 4 | 0 | 0 | Hit |
| 16 | 0 | 0 | Hit |
| 132 | 4 | 4 | Miss |
| 232 | 7 | 7 | Miss |
| 160 | 5 | 5 | Miss |
| 1024 | 32 | 0 | Miss |
| 30 | 0 | 0 | Hit |
| 140 | 4 | 4 | Hit |
| 3100 | 96 | 0 | Miss |
| 180 | 5 | 5 | Hit |
| 2180 | 68 | 4 | Miss |

**Hit ratio:**

Hit ratio is the ratio of number of hits to the total number of access. Thus, the hit ratio can be calculated as:

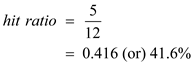


From the above table,

The number of hits=5

Total number of accesses=12

The hit ratio is calculated as,



**Thus, the hit ratio is 41.6%** **.**

**Step 3**

**b.**

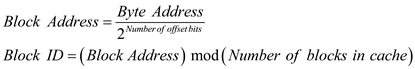
Consider the bit pattern to access the cache is as follows:

|  |  |  |
| --- | --- | --- |
| **Tag** | **Index** | **Offset** |
| 31-12 | 11-6 | 5-0 |

Number of blocks = 

Number of offset bits = 6

Using the following formulae, calculate the block address and block ID as follows:



The 32-bit binary address for the byte-address cache is as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Step 4**  **Byte Address** | **Block address** | **Block ID** | **Hit/Miss** |
| 0 | 0 | 0 | Miss |
| 4 | 0 | 0 | Hit |
| 16 | 0 | 0 | Hit |
| 132 | 2 | 2 | Miss |
| 232 | 3 | 3 | Miss |
| 160 | 2 | 2 | Hit |
| 1024 | 16 | 16 | Miss |
| 30 | 0 | 0 | Hit |
| 140 | 2 | 2 | Hit |
| 3100 | 48 | 48 | Miss |
| 180 | 2 | 2 | Hit |
| 2180 | 34 | 34 | Miss |

**Hit ratio:**

Hit ratio is the ratio of number of hits to the total number of access. Thus, the hit ratio can be calculated as:

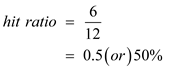


From the above table,

The number of hits=6

Total number of accesses=12

The hit ratio is calculated as,



**Thus, the hit ratio is 50%** **.**

**Question**

Starting from power on, the following byte-addressed cache references are recorded.



[20] <5.2> List the final state of the cache, with each valid entry represented as a record of .

**Answer**

**Step 1**

Consider a direct-mapped cache consisting of 32-bit address.

After the power is on, the recorded byte-addresses cache is as follows:

|  |
| --- |
| **Address** |
| 0 |
| 4 |
| 16 |
| 132 |
| 232 |
| 160 |
| 1024 |
| 30 |
| 140 |
| 3100 |
| 180 |
| 2180 |

**Step 2**

**a.**

Consider the bit pattern to access the cache is as follows:

|  |  |  |
| --- | --- | --- |
| **Tag** | **Index** | **Offset** |
| 31-10 | 9-5 | 4-0 |

Number of blocks = 

Number of offset bits = 5

The 32-bit binary address for the byte-address cache is as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Byte Address** | **Hit/Miss** | **Valid/Invalid** | **Cache block binary address** |
| 0 | Miss | 1 | 00000 |
| 4 | Miss | 1 | 00100 |
| 16 | Miss | 1 | 10000 |
| 132 | Hit | 0 | 00100 |
| 232 | Miss | 1 | 01000 |
| 160 | Hit | 0 | 00000 |
| 1024 | Hit | 0 | 00000 |
| 30 | Miss | 1 | 11110 |
| 140 | Miss | 1 | 01100 |
| 3100 | Miss | 1 | 11100 |
| 180 | Miss | 1 | 10100 |
| 2180 | Hit | 0 | 00100 |

A valid record is determined using valid bit. A block is valid if its valid bit is set to ‘1’. Otherwise, the block is invalid ‘0’.

The final state of cache is as follows:

|  |  |  |
| --- | --- | --- |
| **Index** | **Tag** | **Data** |
| 00000 | 0000 0000 0000 0000 0000 00 | 0 |
| 00000 | 0000 0000 0000 0000 0000 00 | 4 |
| 00000 | 0000 0000 0000 0000 0000 00 | 16 |
| 00111 | 0000 0000 0000 0000 0000 00 | 232 |
| 00000 | 0000 0000 0000 0000 0000 00 | 30 |
| 00100 | 0000 0000 0000 0000 0000 00 | 140 |
| 00000 | 0000 0000 0000 0000 0000 11 | 3100 |
| 00101 | 0000 0000 0000 0000 0000 00 | 180 |

**Step 3**

**b.**

Consider the bit pattern to access the cache is as follows:

|  |  |  |
| --- | --- | --- |
| **Tag** | **Index** | **Offset** |
| 31-12 | 11-6 | 5-0 |

Number of blocks = 

Number of offset bits = 6

The 32-bit binary address for the byte-address cache is as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Byte Address** | **Hit/Miss** | **Valid/Invalid** | **Cache block binary address** |
| 0 | Miss | 1 | 000000 |
| 4 | Miss | 1 | 000100 |
| 16 | Miss | 1 | 010000 |
| 132 | Hit | 0 | 000100 |
| 232 | Miss | 1 | 101000 |
| 160 | Miss | 1 | 100000 |
| 1024 | Hit | 0 | 000000 |
| 30 | Miss | 1 | 011110 |
| 140 | Miss | 1 | 001100 |
| 3100 | Miss | 1 | 011100 |
| 180 | Miss | 1 | 110100 |
| 2180 | Hit | 0 | 000100 |

A valid record is determined using valid bit. A block is valid if its valid bit is set to ‘1’. Otherwise, the block is invalid ‘0’.

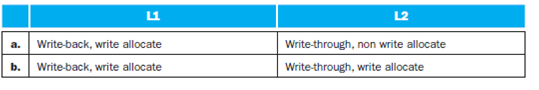
**Step 4**

The final state of cache is as follows:

|  |  |  |
| --- | --- | --- |
| **Index** | **Tag** | **Data** |
| 000000 | 0000 0000 0000 0000 0000 | 0 |
| 000000 | 0000 0000 0000 0000 0000 | 4 |
| 000000 | 0000 0000 0000 0000 0000 | 16 |
| 000011 | 0000 0000 0000 0000 0000 | 232 |
| 000010 | 0000 0000 0000 0000 0000 | 160 |
| 000000 | 0000 0000 0000 0000 0000 | 30 |
| 000010 | 0000 0000 0000 0000 0000 | 140 |
| 110000 | 0000 0000 0000 0000 0000 | 3100 |
| 000010 | 0000 0000 0000 0000 0000 | 180 |

# Question5.5.1

Recall that we have two write policies and write allocation policies, their combinations can be implemented at either in L1 or L2 cache.



[5] <5.2, 5.5> Buffers are employed between different levels of memory hierarchy to reduce access latency. For this given configuration, list the possible buffers needed between L1 and L2 caches, as well as L2 cache and memory.

# Answer

## Step 1

a) Possible buffer in L1:

L1: Write-back buffer.

Possible buffer in L2:

L2: Write buffer.

## Step 2

b) Possible buffer in L1:

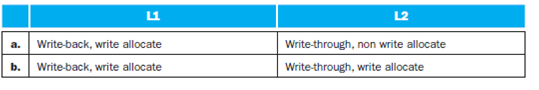
L1: Write-back buffer.

Possible buffer in L2:

L2: Write buffer.

# Question5.5.2

Recall that we have two write policies and write allocation policies, their combinations can be implemented at either in L1 or L2 cache.



[20] <5.2, 5.5> Describe the procedure of handling an L1 write miss, considering the component involved and the possibility of replacing a dirty block.

# Answer

## Step 1

a) Procedure:

• Allocate cache block for the missing data, select a replacement victim.

• If victim dirty, put it into the write-back buffer, which will be further forwarded into L2 write buffer.

• Issue write miss request to the L2 cache.

• If hit in L2, source data into L1 cache; if miss, send write request to memory.

• Data arrives and is installed in L1 cache.

• Processor resumes execution and hits in L1 cache, set the dirty bit.

## Step 2

b) Procedure:

• If L1 miss, allocate cache block for the missing data, select a replacement victim.

• If victim dirty, put it into the write-back buffer, which will be further forwarded into L2 write buffer.

• Issue write miss request to the L2 cache.

• If hit in L2, source data into L1 cache, go to (8).

• If miss, send write request to memory.

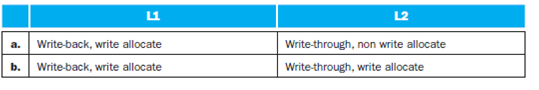
• Data arrives and is installed in L2 cache.

• Data arrives and is installed in L1 cache.

• Processor resumes execution and hits in L1 cache, set the dirty bit.

# Question5.5.3

Recall that we have two write policies and write allocation policies, their combinations can be implemented at either in L1 or L2 cache.



[20] <5.2, 5.5> For a multilevel exclusive cache (a block can only reside in one of the L1 and L2 caches) configuration, describe the procedure of handling an L1 write miss, considering the component involved and the possibility of replacing a dirty block.

# Answer

## Step 1

a) Procedure :

1. Allocate cache block for the missing data, select a replacement victim.

2. If victim clean, put it into a victim buffer between both L1 and L2 caches.

3. If victim dirty, put it into the write-back buffer, which will be further forwarded into L2 write buffer.

4. Issue write miss request to the L2 cache.

5. If hit in L2, source data into L2 cache, invalidate the L2 COPY

6. Data arrives and is installed in L1 cache.

7. Processor resumes execution and hits in L1 cache, set the dirty bit.

## Step 2

b) Procedure:

1. If L1 miss, allocate cache block for the missing data, select a replacement victim.

2. If victim clean put it into a victim buffer between on both L1 and L2 caches.

If victim dirty, put it into the write-back buffer, which will be further forwarded into L2 write buffer.

3. Issue write miss request to the L2 cache.

4. If hit in L2, source data into L1 cache, invalidate copy in L2.Go to (8)

5. If miss, send write request to memory.

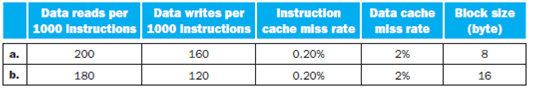
6. Data arrives and is installed in L2 cache.

7. Data arrives and is installed in L1 cache.

8. Processor resumes execution and hits in L1 cache, set the dirty bit.

# Question5.5.4

Consider the following program and cache behaviors.



[5] <5.2, 5.5> For a write-through, write-allocate cache, what’s the minimum read and write bandwidths (measured by byte-per-cycle) needed to achieve a CPI of 2?

# Answer

## Step 1

**Minimum read and write band widths**

a)

Consider the behaviour of the cache as follows:

Data reads per 1000 instructions = 200

Data writes per 1000 instructions = 160

Instruction cache miss rate = 0.20%

Data cache miss rate = 2%

Block size = 8 bytes

Data reads per instruction 

Data writes per instruction

Data cache miss rate = 0.02

Instruction miss rate per block = 0.2% of 8 = 0.016

The number of cycles in which data read or write occurs is 0.5 cycles.

The effective data reads

= Data reads per instruction – Data cache miss rate

– Instruction miss rate per block.

=0.2 – 0.02 – 0.016

=0.164 reads

The minim band width required for read to achieve a CPI of 2



The minim band width required for write to achieve a CPI of 2



## Step 2

b)

Consider the behaviour of the cache as follows:

Data reads per 1000 instructions = 180

Data writes per 1000 instructions = 120

Instruction cache miss rate = 0.20%

Data cache miss rate = 2%

Block size = 16 bytes

Data reads per instruction 

Data writes per instruction

Data cache miss rate = 0.02

Instruction miss rate per block = 0.2% of 16 = 0.032

The number of cycles in which data read or write occurs is 0.5 cycles.

The effective data reads

= Data reads per instruction – Data cache miss rate

– Instruction miss rate per block.

=0.18 – 0.02 – 0.032

=0.128 reads

The minim band width required for read to achieve a CPI of 2

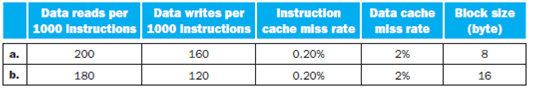


The minim band width required for write to achieve a CPI of 2



# Question5.5.5

Consider the following program and cache behaviors.



[5] <5.2, 5.5> For a write-back, write-allocate cache, assuming 30% of replaced data cache blocks are dirty, what’s the minimal read and write bandwidths needed for a CPI of 2?

# Answer

## Step 1

**Minimum read and write band widths**

a)

Consider the behaviour of the cache as follows:

Data reads per 1000 instructions = 200

Data writes per 1000 instructions = 160

Instruction cache miss rate = 0.20%

Data cache miss rate = 2%

Block size = 8 bytes

30% of replaced data cache blocks are dirty.

Data reads per instruction 

Considering dirty blocks, data reads per instruction



Data writes per instruction

Considering dirty blocks, data reads per instruction



Data cache miss rate = 0.02

Instruction miss rate per block = 2% of 8 = 0.016

The number of cycles in which data read or write occurs is 0.5 cycles.

The effective data reads

= Data reads per instruction – Data cache miss rate

– Instruction miss rate per block.

=0.142 – 0.02 – 0.016

=0.106 reads

The minim band width required for read to achieve a CPI of 2



The minim band width required for write to achieve a CPI of 2



## Step 2

b)

Consider the behaviour of the cache as follows:

Data reads per 1000 instructions = 180

Data writes per 1000 instructions = 120

Instruction cache miss rate = 0.20%

Data cache miss rate = 2%

Block size = 16 bytes

Data reads per instruction 

Considering dirty blocks, data reads per instruction



Data writes per instruction

Considering dirty blocks, data reads per instruction



Data cache miss rate = 0.02

Instruction miss rate per block = 2% of 16 = 0.032

The number of cycles in which data read or write occurs is 0.5 cycles.

The effective data reads

= Data reads per instruction – Data cache miss rate

– Instruction miss rate per block.

=0.126 – 0.02 – 0.032

=0.074 reads

The minim band width required for read to achieve a CPI of 2

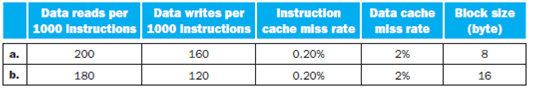


The minim band width required for write to achieve a CPI of 2



# Question5.5.6

Consider the following program and cache behaviors.



[5] <5.2, 5.5> What are the minimal bandwidths needed to achieve the performance of CPI = 1.5?

# Answer

## Step 1

**Minimum read and write band widths**

a)

Consider the behaviour of the cache as follows:

Data reads per 1000 instructions = 200

Data writes per 1000 instructions = 160

Instruction cache miss rate = 0.20%

Data cache miss rate = 2%

Block size = 8 bytes

Data reads per instruction 

Data writes per instruction

Data cache miss rate = 0.02

Instruction miss rate per block = 2% of 8 = 0.016

The number of cycles in which data read or write occurs is 0.5 cycles.

The effective data reads

= Data reads per instruction – Data cache miss rate

– Instruction miss rate per block.

=0.2 – 0.02 – 0.016

=0.164 reads

The minim band width required for read to achieve a CPI of 1.5



The minim band width required for write to achieve a CPI of 1.5



## Step 2

b)

Consider the behaviour of the cache as follows:

Data reads per 1000 instructions = 180

Data writes per 1000 instructions = 120

Instruction cache miss rate = 0.20%

Data cache miss rate = 2%

Block size = 16 bytes

Data reads per instruction 

Data writes per instruction

Data cache miss rate = 0.02

Instruction miss rate per block = 2% of 16 = 0.032

The number of cycles in which data read or write occurs is 0.5 cycles.

The effective data reads

= Data reads per instruction – Data cache miss rate

– Instruction miss rate per block.

=0.18 – 0.02 – 0.032

=0.128 reads

The minim band width required for read to achieve a CPI of 1.5

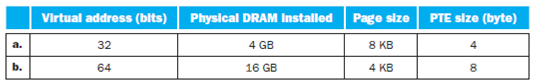


The minim band width required for write to achieve a CPI of 1.5



# Question5.11.1

In this exercise, we will examine space/time optimizations for page tables. The following table shows parameters of a virtual memory system.



[10] <5.4> For a single-level page table, how many page table entries (PTE) are needed? How much physical memory is needed for storing the page table?

# Answer

## Step 1

a) Given data:

Virtual address = 32 bits

Page size = 8KB

PTE size = 4 bytes

Page table entries (PTE) = Virtual address – page size

= 32 – 13(8KB)

= 19 bits or 512 entries

Physical memory = 

= 

= 2048 or 2MB

## Step 2

b) Given Data:

Virtual address = 64 bits

Page size = 4KB

PTE size = 8 bytes

Page table entries (PTE) = Virtual address – page size

= 64 – 12(4KB)

= 52 bits or entries

## Step 3

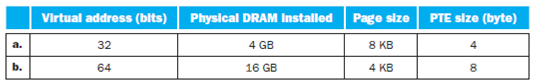
Physical memory = 

= 

= bytes

# Question5.11.2

In this exercise, we will examine space/time optimizations for page tables. The following table shows parameters of a virtual memory system.



[10] <5.4> Using a multilevel page table can reduce the physical memory consumption of page tables by only keeping active PTEs in physical memory. How many levels of page tables will be needed in this case? And how many memory references are needed for address translation if missing in TLB?

# Answer

## Step 1

a) Given data:

Virtual address = 32 bits

Page size = 8KB

PTE size = 4 bytes

Page table entries (PTE) = Virutal address – page size

= 32 – 13(8KB)

= 19 bits or 512 entries

## Step 2

8 KB page/4 byte PTE =  pages indexed per page,

Hence with  PTE’s will need 2-level page table setup.

Each address translation will require at least 2 physical memory accesses.

## Step 3

b) Given Data:

Virtual address = 64 bits

Page size = 4KB

PTE size = 8 bytes

Page table entries (PTE) = Virutal address – page size

= 64 – 12(4KB)

= 52 bits or  entries

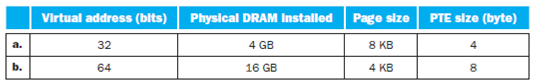
4KB page/8 byte PTE =  pages indexed per page,

Hence with  PTEs’will need will need 6-level page table setup.

Hence address translation will require at least 6 physical memory accesses.

# Question5.11.3

In this exercise, we will examine space/time optimizations for page tables. The following table shows parameters of a virtual memory system.



[15] <5.4> An inverted page table can be used to further optimize space and time. How many PTEs are needed to store the page table? Assuming a hash table implementation, what are the common-case and worse-case numbers of memory references needed for servicing a TLB miss?

# Answer

## Step 1

a) Given Data:

Virtual address = 32 bits.

Physical DRAM = 4GB

Page size = 8KB and

PTE size = 4 byte.

Number of PTE = 32-24

= 8bits.

Therefore 512K PTE’s are needed to store the page table.

In common case: no hash conflict, so one memory reference per address translation.

In worst case: 512 memory references are needed if hash table degrade into a link list.

## Step 2

b) Given Data:

Virtual address = 64 bits.

Physical DRAM = 16 GB

Page size = 4KB, and

PTE size = 8 byte.

Number of PTE = 64-12

= 52 bits.

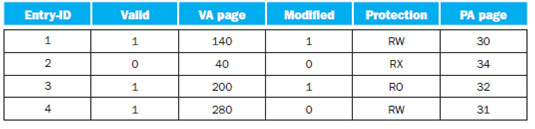
Therefore  PTE’s are needed to store the page table.

In common case: no hash conflict. So one memory reference per address translation.

In worst case:  memory references are needed if hash table degrade into link list.

**Question5.11.5**

The following table shows the contents of a four-entry TLB.



[5] <5.4> What happens when an instruction writes to VA page 30? When would a software-managed TLB be faster than a hardware-managed TLB?

**Answer**

**Step 1**

The technique of implementing the secondary storage (magnetic disks) to act as cache to the primary (main) memory is called as **virtual memory**.

A block of virtual memory is called as **page**. A **page fault** occurs when there is a miss in virtual memory.

The processor produces a **virtual address (VA)** using virtual memory.

Virtual addresses are translated into physical address by the combination of software and hardware addresses.

The process of virtual address to physical address is called address translation or **address mapping**.

The page tables are stored in the main memory. The program requires at least two operations to access every memory. To get physical address one operation is used and other to get the actual data.

**Step 2**

To reduce the memory access operations, dependency on the locality of page table references is developed. A special cache is used to keep track if most recently used translations. This cache is called as **TLB (Translation Look aside Buffer)**.

The TLB table tag entry consists of the following details:

• Entry identification number

• Valid status

• Virtual address page number

• Modification status

• Protection status

• Page address (PA) page number

**Four-Entry TLB table along with contents is follows:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Entry-ID** | **Valid** | **VA page** | **Modified** | **Protection** | **PA Page** |
| 1 | 1 | 140 | 1 | RW | 30 |
| 2 | 0 | 40 | 0 | RX | 34 |
| 3 | 1 | 200 | 1 | RO | 32 |
| 4 | 1 | 280 | 0 | RW | 31 |

**Step 3**

If the instructions want to write to VA page 30, it tries to find the VA page 30 in the TLB table. The pages available in the TLB table are 140, 40 , 200, and 280. The VA page 30 is not available in the TLB entries. So, a TLB miss occurs.

**Therefore, a TLB miss occurs when an instruction writes to VA page 30** **.**

TLBs are managed by both software and hardware implementations. Sometimes software managed TLB are faster than hardware managed TLB.

The following steps take place when TLB is managed by software:

• From memory, it brings in the page table entry.

• Then the instruction which caused the TLB miss will be re–executed.

• The instruction will get a TLB hit after re–execution.

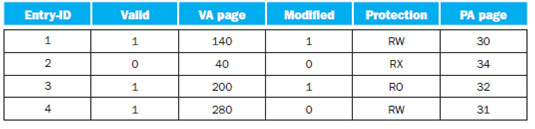
• A page fault exception will occur if the page is not in the memory indicated by the page table entry.

If the above process of software managed TLB is observed; the instruction causing TLB miss is re–executed by bringing the page table entry from memory. If the TLB missed entry is in the processor cache, then the software managed TLB results in faster work than the hardware managed TLB.

Therefore, if the processor cache caches the most of the missed TLB entries, then the software managed TLB is faster when compared to hardware managed TLB.

# Question5.11.6

The following table shows the contents of a four-entry TLB.



[5] <5.4> What happens when an instruction writes to VA page xxx?

# Answer

## Step 1

When an instruction writes to VA page then write protection exception occur because actual page table entry contains the address of the page, validity and protection information. If this information says that either the page is not present in memory or the protections are not correct, the translation stops and a write protection exception are occur.