Computer Architecture HW2

2013210111 남세현

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

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

Answer

```
sll $s0, $s3, 2
add $s0, $s0, $s6
lw $s0, 0($s0)
sll $s1, $s4, 2
add $s1, $s1, $s6
lw $s1, 0($s1)
add $s2, $s0, $s1
sw $s2, 32($s7)
```

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

Answer

op(6bits) : 0

rs(5bits) : 16

rt(5bits) : 16

rd(5bits) : 16

shamt(5bits) : 0

funct(6bits) : 32

type : add

instruction : add $s0, $s0, $s0

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

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

Answer

type : sub

instruction : sub $v1, $v1, $v0

binary : 0000 0000 0110 0010 0001 1000 0010 0010

2.24 [5] <§2.7> Suppose the program counter (PC) is set to 0x2000 0000. Is it possible to use the jump (j) MIPS assembly instruction to set the PC to the address as 0x4000 0000? Is it possible to use the branch-on-equal (beq) MIPS assembly instruction to set the PC to this same address?

Answer

No, jump(j) mips instruction can jump with upper 4 bit of PC, and 28 bit from instruction, and two 0 lower bits. So, jump can move 2^28(0x800 0000) relatively.

Also beq use 16 bit offset, can move between -2^15 and 2^15-1 so beq mips can't set PC to same address.

2.25 The following instruction is not included in the MIPS instruction set:

rpt $t2, loop # if(R[rs]>0) R[rs]=R[rs]−1, PC=PC+4+BranchAddr

2.25.1 [5] <§2.7> If this instruction were to be implemented in the MIPS instruction set, what is the most appropriate instruction format?

Answer

I Struction format. Because that need register $t2 and BranchAddr, so I struction format is most appropriate.

2.25.2 [5] <§2.7> What is the shortest sequence of MIPS instructions that performs the same operation?

Answer

```
slt     $zero, R[rs], $t1
sub     R[rs], $t1, R[rs]
bne     $t1, $zero, BranchAdder
```

2.39 [5] <§2.10> Write the MIPS assembly code that creates the 32-bit constant 0010 0000 0000 0001 0100 1001 0010 0100(two) and stores that value to register $t1.

Answer

```
lui     $t1, 0b0010 0000 0000 0001
ori     $t1, $t1, 0b010 01001 0010 0100
```

2.40 [5] <§§2.6, 2.10> If the current value of the PC is 0x00000000, can you use a single jump instruction to get to the PC address as shown in Exercise 2.39?

Answer

Using jump register instrument, can jump to that address, like below

```
jr      $t1
```

2.41 [5] <§§2.6, 2.10> If the current value of the PC is 0x00000600, can you use a single branch instruction to get to the PC address as shown in Exercise 2.39?

Answer

Can't. Because branch instruction can jump from PC – 2^15 to PC + 2^15 – 1.

3.14 [10] <§3.3> Calculate the time necessary to perform a multiply using the approach given in Figures 3.3 and 3.4 if an integer is 8 bits wide and each step of the operation takes 4 time units. Assume that in step 1a an addition is always performed—either the multiplicand will be added, or a zero will be. Also assume that the registers have already been initialized (you are just counting how long it takes to do the multiplication loop itself). If this is being done in hardware, the shifts of the multiplicand and multiplier can be done simultaneously. If this is being done in soft ware, they will have to be done one after the other. Solve for each case.

Answer

In first case, it cost 4 * 2 time units calculating 1bits, so the total cost time is 4 * 2 * 8 = 64.

In second case, it cost 4 * 3 time units calculating 1 bits, so the total cost time is 4 * 3 * 8 = 96.

3.16 [20] <§3.3> Calculate the time necessary to perform a multiply using the approach given in Figure 3.7 if an integer is 8 bits wide and an adder takes 4 time units.

Answer

It cost (8 + 4 + 2 + 1) * 4 = 60.

3.22 [10] <§3.5> What decimal number does the bit pattern 0×0C000000 represent if it is a floating point number? Use the IEEE 754 standard.

Answer

0xC = 0b1100.

The sign bit is 0, it mean positive.

The exponent bits is 0b0001 1000 = 24.

The faraction bit is 0, so the value of decimal is (-1)^0 * (1 + 0) * 2 ^ (24 - 127)

= 2^-103

3.23 [10] <§3.5> Write down the binary representation of the decimal number 63.25 assuming the IEEE 754 single precision format.

Answer

63.25 = 253 / 4 = 1111 1101(two) / 2^2 = 1.1111 101(two) * 2^-9

The sign bit is 0 becuase it is positive.

The exponent is 9 + bias = 9 + 127, and the bit is 0b1000 1000

The faction bit is 0b1111 1010 0000....00

So, the answer is 0100 0100 0111 1101 0000 000 0000 0000

3.27 [20] <§3.5> IEEE 754-2008 contains a half precision that is only 16 bits wide. The left most bit is still the sign bit, the exponent is 5 bits wide and has a bias of 15, and the mantissa is 10 bits long. A hidden 1 is assumed. Write down the bit pattern to represent -1.5625 * 10^-1 assuming a version of this format, which uses an excess-16 format to store the exponent. Comment on how the range and accuracy of this 16-bit floating point format compares to the single precision IEEE 754 standard.

Answer

-1.5625 * 10^-1 = -0.15625 = -1.25 * 2^-3 = -1.01(two) * 2^-3

The sign bit is 1, and the exeponent is 12 (because 12 – 15 = -3)

The faction is 0.25, So in binary, 0b1011 0001 0000 0000.

IEEE 754 standard's range : -2^-127 ~ 2^128, accuracy is.. 23 bit under point.

IEEE 754-2008 16bits' range : -2^15 ~ 2^16, accuracy is 10 bit under point.

3.29 [20] <§3.5> Calculate the sum of 2.6125 * 10^1 and 4.150390625 * 10 ^ -1 by hand, assuming A and B are stored in the 16-bit half precision described in Exercise 3.27. Assume 1 guard, 1 round bit, and 1 sticky bit, and round to the nearest even. Show all the steps.

Answer

2.6125 * 10 ^ 1 = 2.6125 = 2 * 1.30625 = 2 * (1 + 0.25 + 0.03125 + 0.015625 + 0.0078125 + 0.0009765625 + ...)

4.150390625 * 10^-1 = 0.4150390625 = 2^2 * (1.66015625) = 2^2 * (1 + 0.5 + 0.125 + 0.03125 + 0.00390625)

	S	E	F	
A	0	10000	1.0100 1110 01(two)	000 (guard, round, sticky)
B	0	10011	1.1010 1001 00	000

A should shift 3 times to align exponent.

Then,

	S	E	F	
A	0	10011	0.0010 1001 11	001
B	0	10011	1.1010 1001 00	000

Sum!

SUM	0	10011	1.1101 0010 11	001

Becuase stiky bit is 1, round up.

SUM	0	10011	1.1101 0011 00	

In decimal, 7.296875.