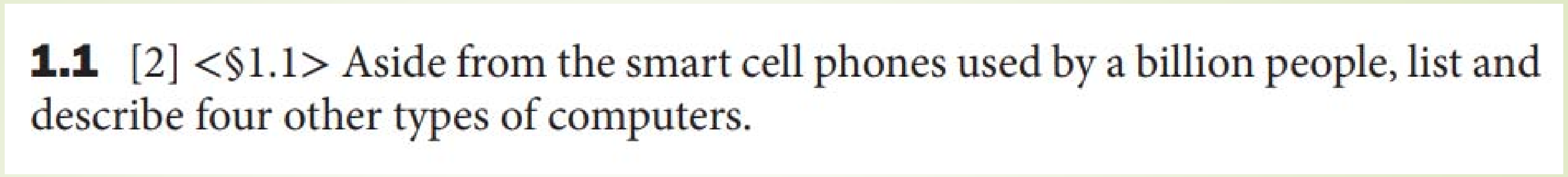
Homework and Solutions

Haifeng Liu

- 1-1, 1-2, 1-4, 1-6, 1-7, 1-14
- 2.3，2.4，2.5，2.11，2.12，2.13，2.23，2.24，2.25，2.26，2.31，2.35, 2.36, 2.40
- 3.1， 3.5, 3.8, 3.9, 3.13, 3.19， 3.20-24, 3.27, 3.29
- 4.1，4.4，4.6，4.7，4.8，4.13


1.1 [2] <§1.1> Aside from the smart cell phones used by a billion people, list and describe four other types of computers.

- Embedded computer
- PC
- Server
- Supercomputer

1.2 [5] <§1.2> The eight great ideas in computer architecture are similar to ideas from other fields. Match the eight ideas from computer architecture, "Design for Moore's Law," "Use Abstraction to Simplify Design," "Make the Common Case Fast," "Performance via Parallelism," "Performance via Pipelining," "Performance via Prediction," "Hierarchy of Memories," and "Dependability via Redundancy" to the following ideas from other fields:

a. Assembly lines in automobile manufacturing

b. Suspension bridge cables

c. Aircraft and marine navigation systems that incorporate wind information

d. Express elevators in buildings

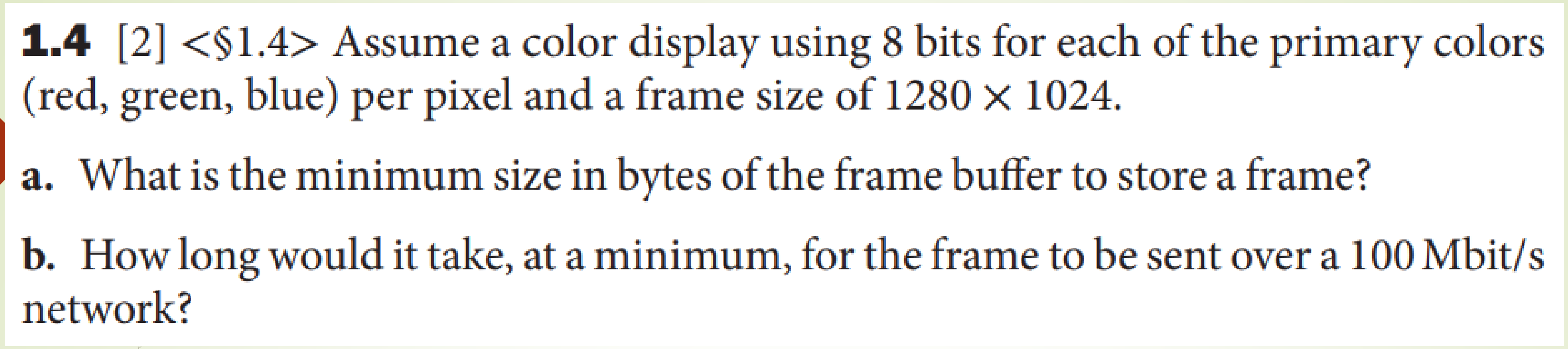
e. Library reserve desk

f. Increasing the gate area on a CMOS transistor to decrease its switching time

g. Adding electromagnetic aircraft catapults (which are electrically powered as opposed to current steam-powered models), allowed by the increased power generation offered by the new reactor technology

h. Building self-driving cars whose control systems partially rely on existing sensor systems already installed into the base vehicle, such as lane departure systems and smart cruise control systems

a. Performance via Pipelining
b. Dependability via Redundancy
c. Performance via Prediction
d. Make the Common Case Fast
e. Hierarchy of Memories
f. Performance via Parallelism
g. Design for Moore's Law
h. Use Abstraction to Simplify Design

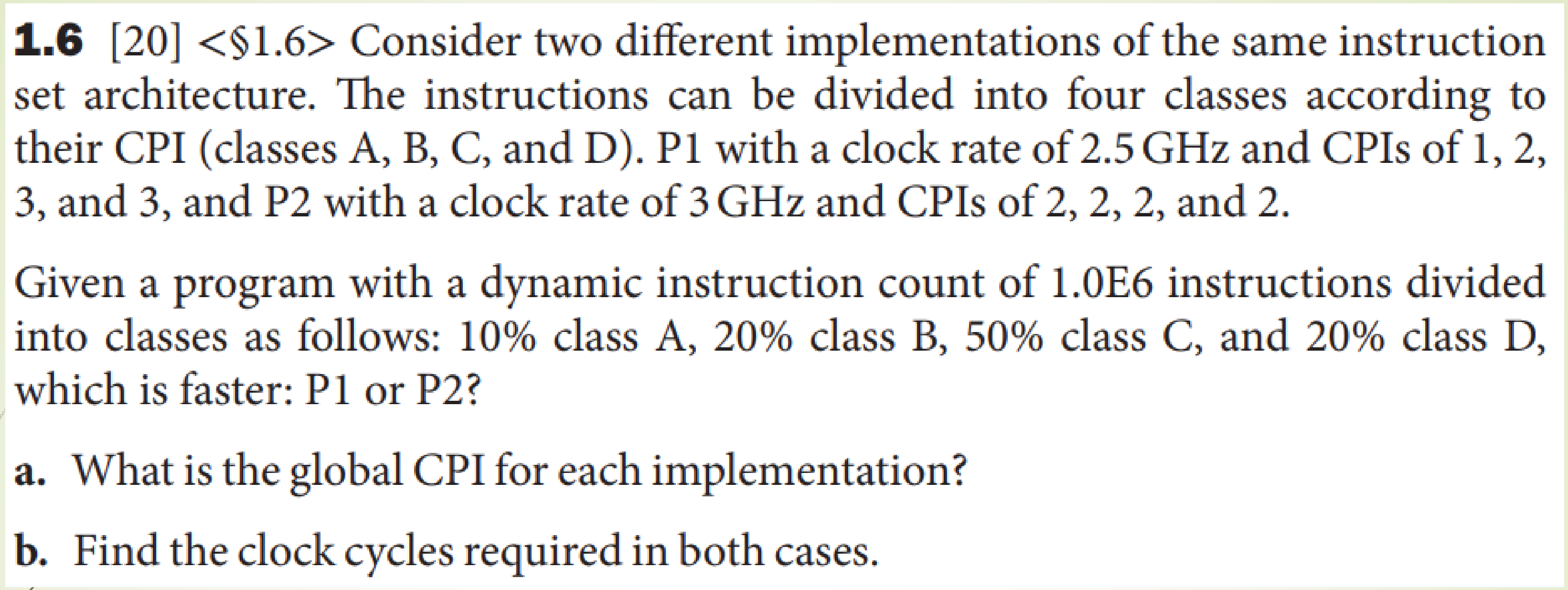


1.4 [2] <§1.4> Assume a color display using 8 bits for each of the primary colors (red, green, blue) per pixel and a frame size of 1280 × 1024.

a. What is the minimum size in bytes of the frame buffer to store a frame?

b. How long would it take, at a minimum, for the frame to be sent over a 100 Mbit/s network?

- 1280 x 1024 x 3 = 3,932,160 bytes/frame
- 3,932,160 x 8 / 100M = 0.3146 seconds



1.6 [20] <§1.6> Consider two different implementations of the same instruction set architecture. The instructions can be divided into four classes according to their CPI (classes A, B, C, and D). P1 with a clock rate of 2.5 GHz and CPIs of 1, 2, 3, and 3, and P2 with a clock rate of 3 GHz and CPIs of 2, 2, 2, and 2.

Given a program with a dynamic instruction count of 1.0E6 instructions divided into classes as follows: 10% class A, 20% class B, 50% class C, and 20% class D, which is faster: P1 or P2?

a. What is the global CPI for each implementation?

b. Find the clock cycles required in both cases.

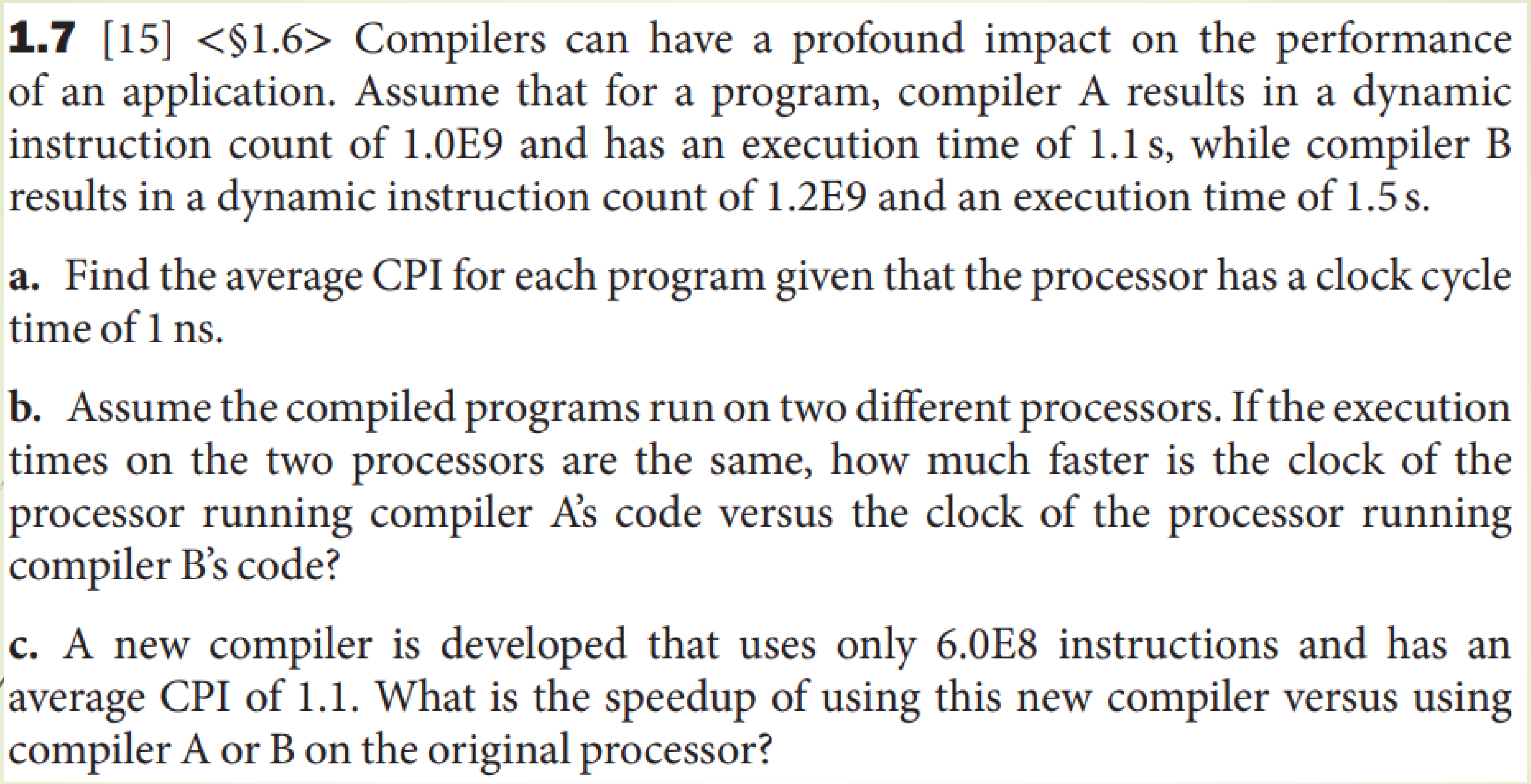
- Class A: 10^5 instr. Class B: 2×10^5 instr. Class C: 5×10^5 instr. Class D: 2×10^5 instr.
- b. Total cycles of P1 = $10^5 + 2 \times 10^5 \times 2 + 5 \times 10^5 \times 3 + 2 \times 10^5 \times 3 = 26 \times 10^5$

  Total cycles of P2 = $10^5 \times 2 + 2 \times 10^5 \times 2 + 5 \times 10^5 \times 2 + 2 \times 10^5 \times 2 = 20 \times 10^5$
- a. CPI(P1) = $26 \times 10^5 / 10^6 = 2.6$

  CPI(P2) = $20 \times 10^5 / 10^6 = 2.0$

- Total time (P1) = 26×10^5 / (2.5 GHz) = 1.04 ms

  Total time (P2) = $20 \times 10^5 / (3 \times 10^9) = 0.666$ ms



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

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

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

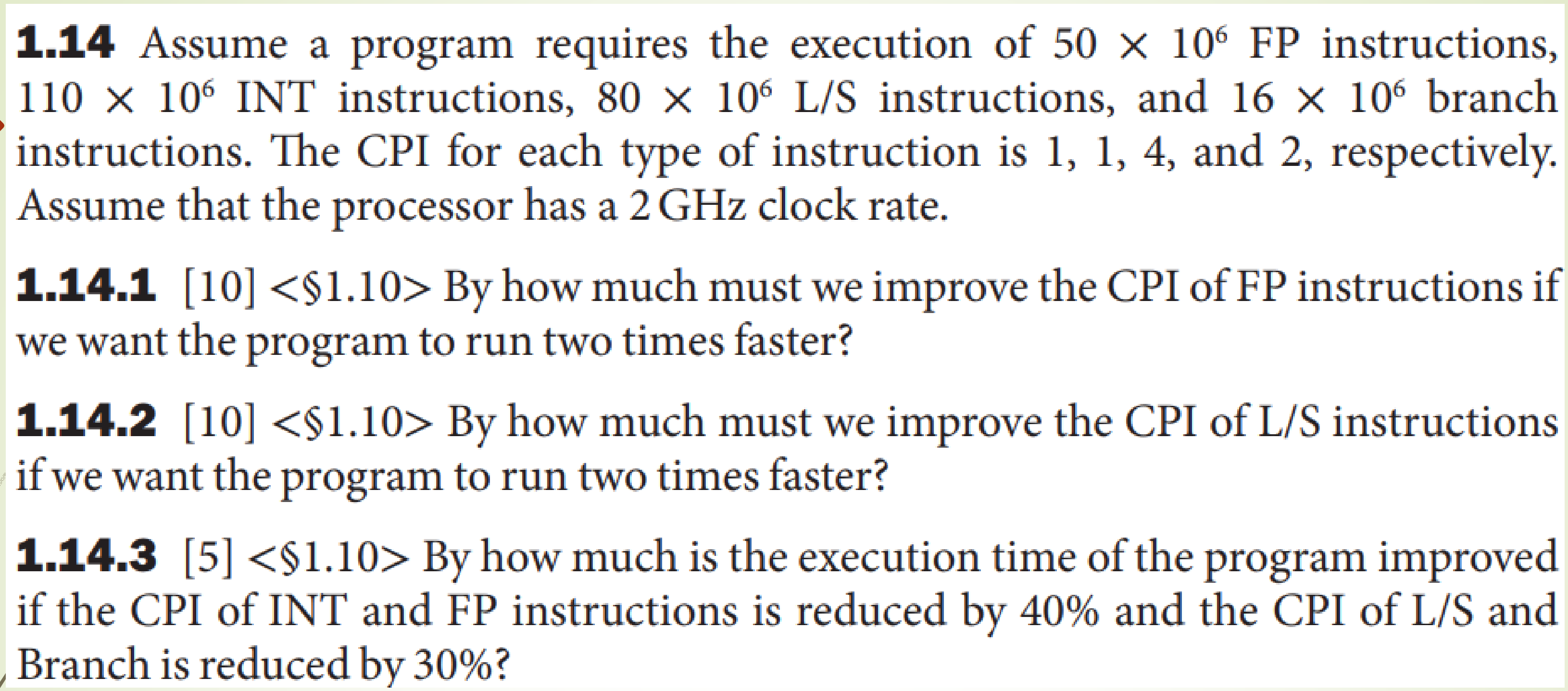
c. A new compiler is developed that uses only 6.0E8 instructions and has an average CPI of 1.1. What is the speedup of using this new compiler versus using compiler A or B on the original processor?

- a. CPI = $T_{exec} \times$ f/No. instr.

  Compiler A CPI = 1.1

  Compiler B CPI = 1.25
- b. f_B / f_A = (No. instr.(B) $\times$ CPI(B))/(No. instr.(A) $\times$ CPI(A)) = 1.37
- c. T_A / T_{new} = 1.67

  T_B / T_{new} = 2.27

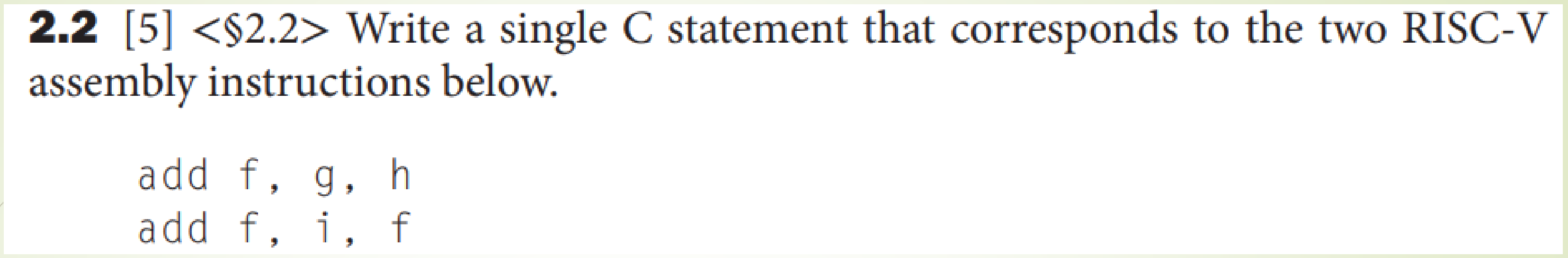

1.14 Assume a program requires the execution of 50×10^6 FP instructions, 110×10^6 INT instructions, 80×10^6 L/S instructions, and 16×10^6 branch instructions. The CPI for each type of instruction is 1, 1, 4, and 2, respectively. Assume that the processor has a 2 GHz clock rate.

1.14.1 [10] <§1.10> By how much must we improve the CPI of FP instructions if we want the program to run two times faster?

1.14.2 [10] <§1.10> By how much must we improve the CPI of L/S instructions if we want the program to run two times faster?

1.14.3 [5] <§1.10> By how much is the execution time of the program improved if the CPI of INT and FP instructions is reduced by 40% and the CPI of L/S and Branch is reduced by 30%?

- Solution:
- 1.14.1 (1*50 + 1*110+4*80+2*16) /2 – (1*110+4*80+2*16) = 256-462 < 0 so impossible
- 1.14.2 CPI $_{\text{improve}}$d = ((1*50 + 1*110+4*80+2*16) /2 – (1*50+1*110+2*16))/80 = 64 /80 = 0.8
- 1.14.3 ((50+110)*0.6 + (4*80+ 2* 16)* 0.7) / 512 = 66.875%

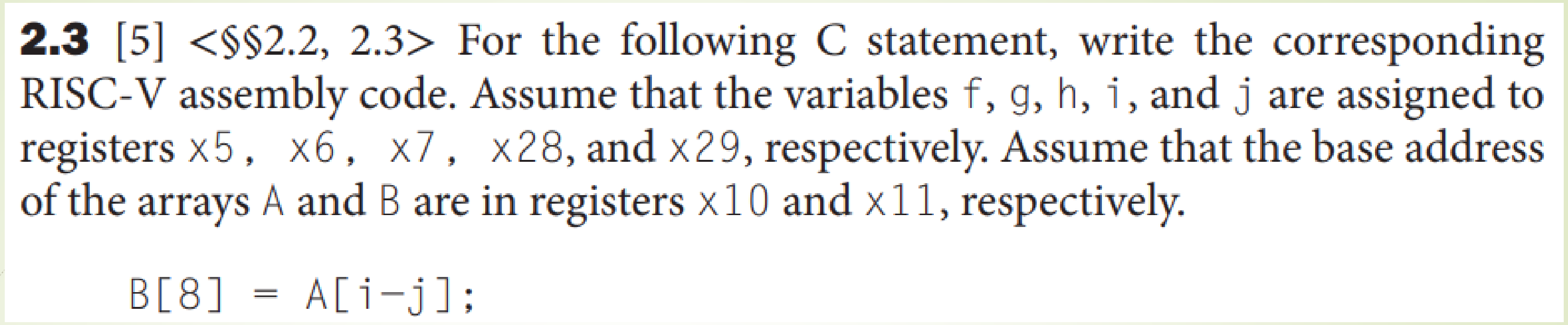


2.2 [5] <§2.2> Write a single C statement that corresponds to the two RISC-V assembly instructions below.

```
add f, g, h
add f, i, f
```

- Solution:

  f = g+h+i

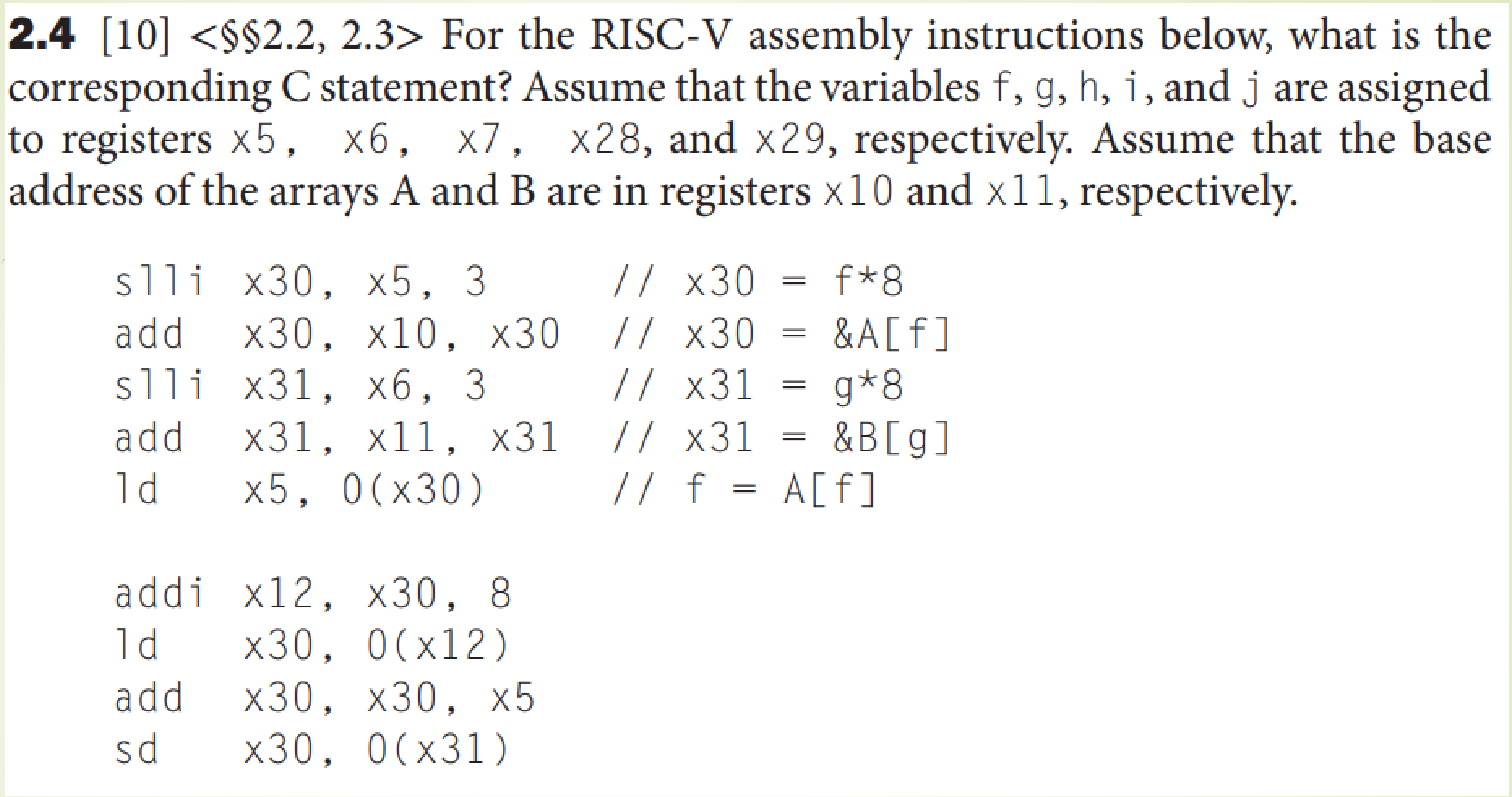


2.3 [5] <§§2.2, 2.3> For the following C statement, write the corresponding RISC-V assembly code. Assume that the variables f, g, h, i, and j are assigned to registers x5, x6, x7, x28, and x29, respectively. Assume that the base address of the arrays A and B are in registers x10 and x11, respectively.

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

- Solution:

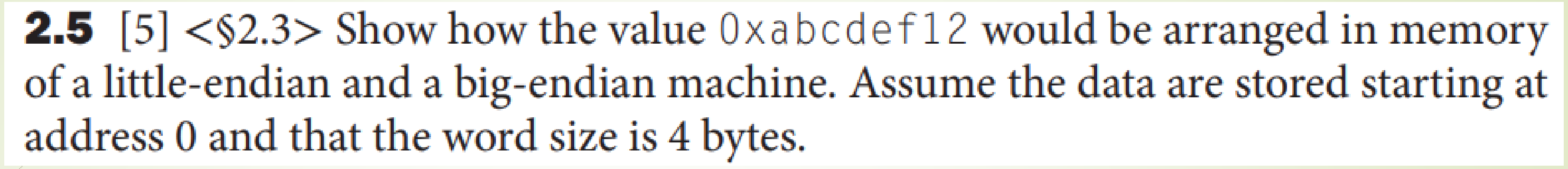
```
sub x30, x28, x29  // compute i-j
slli x30, x30, 3      // multiply by 8 to convert the word offset to a byte offset
add x3, x30, x10      // get the address of A[i-j]
ld x30, 0(x3) // load A[i-j]
sd x30, 64(x11) // store in B[8]
```



2.4 [10] <§§2.2, 2.3> For the RISC-V assembly instructions below, what is the corresponding C statement? Assume that the variables f, g, h, i, and j are assigned to registers x5, x6, x7, x28, and x29, respectively. Assume that the base address of the arrays A and B are in registers x10 and x11, respectively.

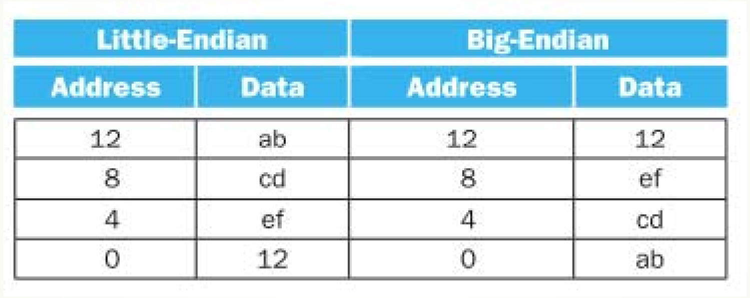
```
slli  x30, x5, 3      // x30 = f*8
add   x30, x10, x30   // x30 = &A[f]
slli  x31, x6, 3      // x31 = g*8
add   x31, x11, x31   // x31 = &B[g]
ld    x5, 0(x30)      // f = A[f]

addi  x12, x30, 8
ld    x30, 0(x12)
add   x30, x30, x5
sd    x30, 0(x31)
```

- Solution: B[g] = A[f] + A[f+1]

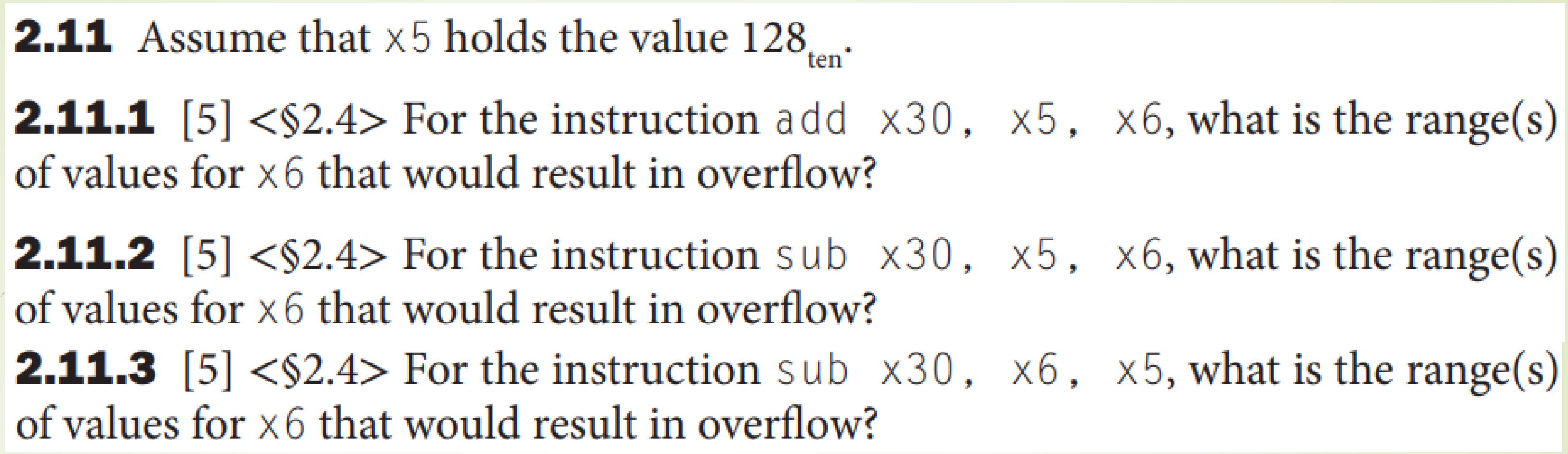


2.5 [5] <§2.3> Show how the value `0xabcdef12` would be arranged in memory of a little-endian and a big-endian machine. Assume the data are stored starting at address 0 and that the word size is 4 bytes.

- Solution:



Little-Endian		Big-Endian	
Address	**Data**	**Address**	**Data**
12	ab	12	12
8	cd	8	ef
4	ef	4	cd
0	12	0	ab



2.11 Assume that `x5` holds the value 128_{ten}.

2.11.1 [5] <§2.4> For the instruction `add x30, x5, x6`, what is the range(s) of values for `x6` that would result in overflow?

2.11.2 [5] <§2.4> For the instruction `sub x30, x5, x6`, what is the range(s) of values for `x6` that would result in overflow?

2.11.3 [5] <§2.4> For the instruction `sub x30, x6, x5`, what is the range(s) of values for `x6` that would result in overflow?

- Solution:
- 2.11.1 There is an overflow if:

  $128 + x6 > 2^{63} - 1$, that is $x6 > 2^{63} - 129$.

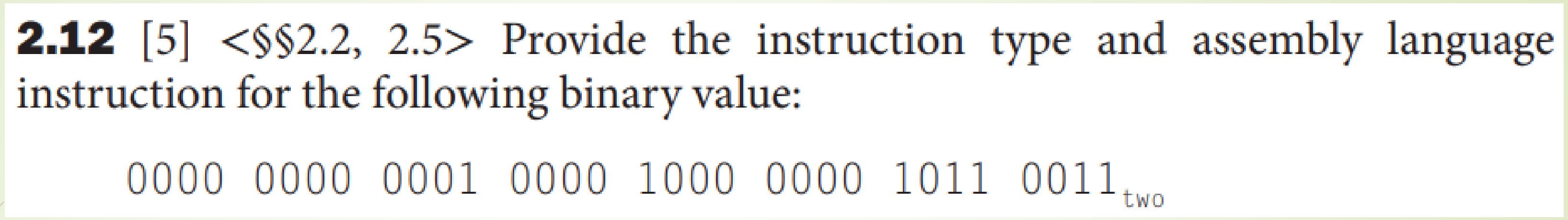
  Or $128 + x6 < -2^{63}$, that is $x6 < -2^{63} - 128$ (which is impossible given the range of x6).
- 2.11.2 There is an overflow if:

  $128 - x6 > 2^{63} - 1$, that is $x6 < 2^{63} + 129$

  or $128 - x6 < -2^{63}$, that is $x6 > 2^{63} + 128$ (which is impossible given the range of x6).
- 2.11.3 There is an overflow if:

  $x6 - 128 > 2^{63} - 1$, that is $x6 > 2^{63} + 127$ (which is impossible given the range of x6).

  or $x6 - 128 < -2^{63}$, that is $x6 < -2^{63} + 128$.

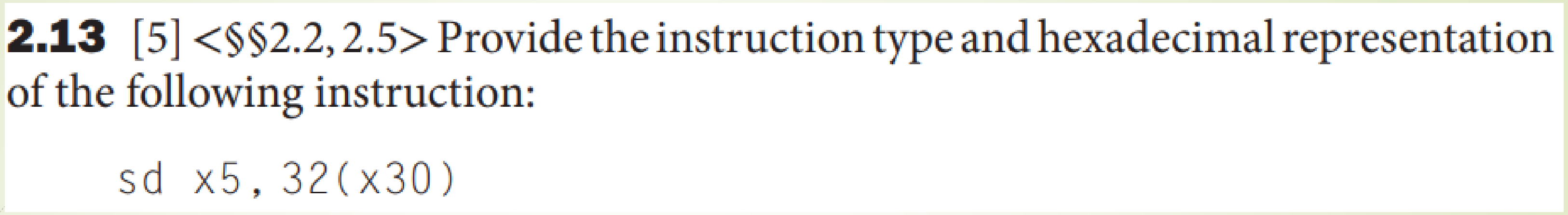


2.12 [5] <§§2.2, 2.5> Provide the instruction type and assembly language instruction for the following binary value:

$0000\ 0000\ 0001\ 0000\ 1000\ 0000\ 1011\ 0011_{\text{two}}$

- Solution:

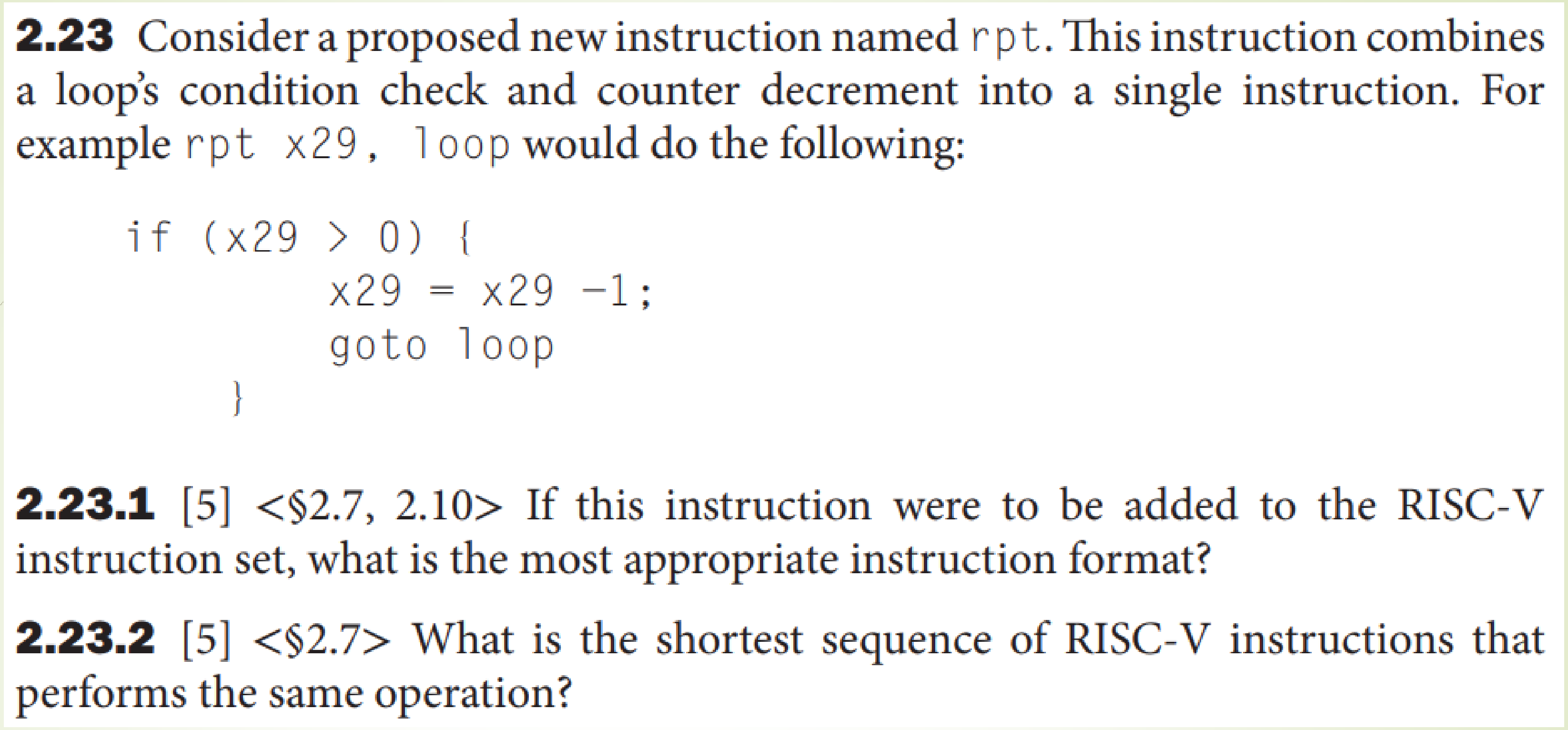
R-type: add x1, x1, x1


2.13 [5] <§§2.2, 2.5> Provide the instruction type and hexadecimal representation of the following instruction:

```
    sd x5, 32(x30)
```

- Solution:

  S-type: 0x25F3023 (0000001 00101 11110 011 00000 0100011)



2.23 Consider a proposed new instruction named `rpt`. This instruction combines a loop's condition check and counter decrement into a single instruction. For example `rpt x29, loop` would do the following:

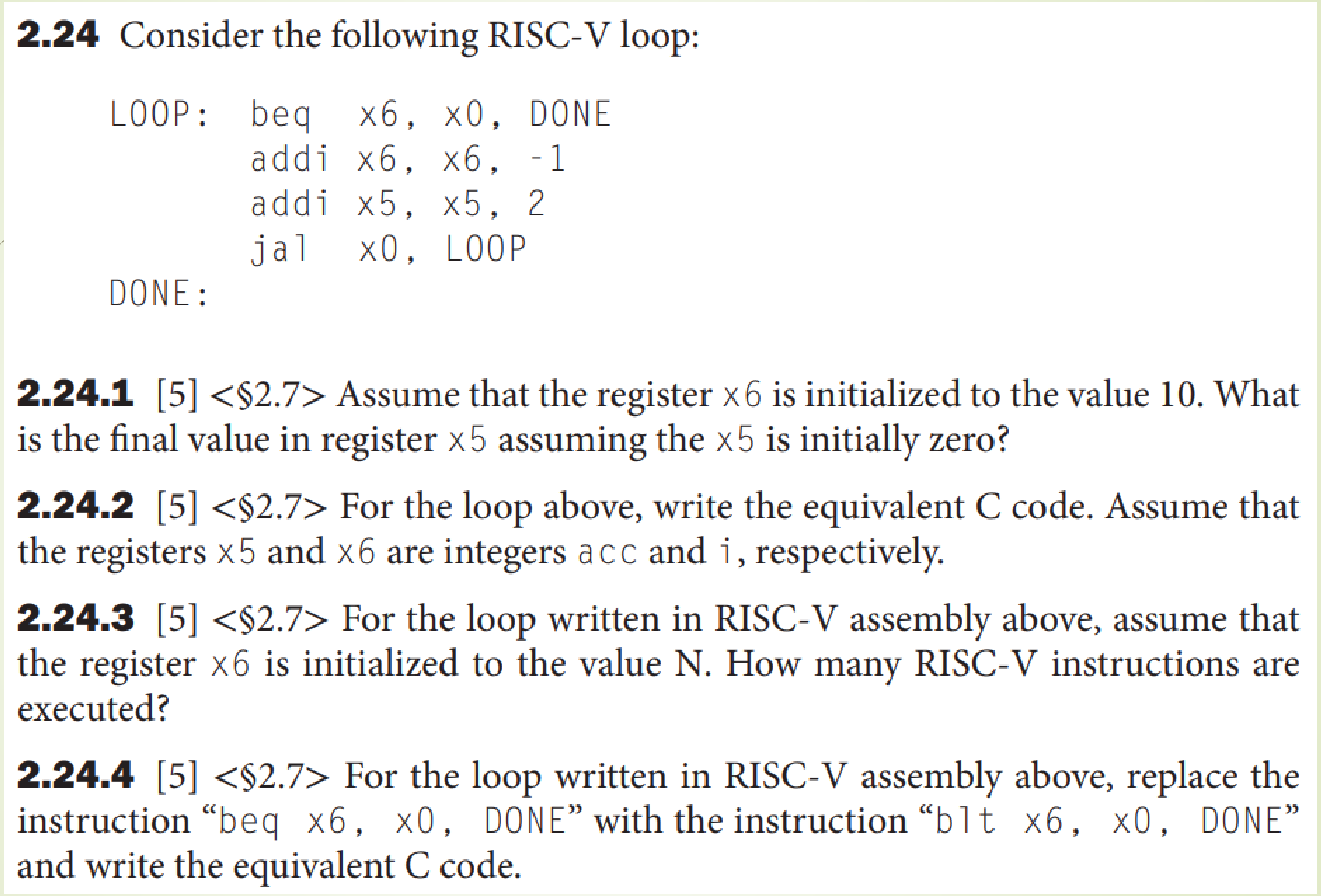
```
if (x29 > 0) {
        x29 = x29 -1;
        goto loop
    }
```

2.23.1 [5] <§2.7, 2.10> If this instruction were to be added to the RISC-V instruction set, what is the most appropriate instruction format?

2.23.2 [5] <§2.7> What is the shortest sequence of RISC-V instructions that performs the same operation?

- Solution:
  - 2.23.1 The UJ instruction format would be most appropriate because it would allow the maximum number of bits possible for the " loop " parameter, thereby maximizing the utility of the instruction.
  - 2.23.2 It can be done in three instructions:

```
loop:  addi  x29, x29, -1    //Subtract 1 from x29
       blt  x0, x29, loop    // Continue if x29 not negative
       addi  x29, x29, 1     // Add back 1 that shouldn't have been subtracted.
```



2.24 Consider the following RISC-V loop:

```
LOOP:  beq  x6, x0, DONE
       addi x6, x6, -1
       addi x5, x5, 2
       jal  x0, LOOP
DONE:
```

2.24.1 [5] <§2.7> Assume that the register `x6` is initialized to the value 10. What is the final value in register `x5` assuming the `x5` is initially zero?

2.24.2 [5] <§2.7> For the loop above, write the equivalent C code. Assume that the registers `x5` and `x6` are integers `acc` and `i`, respectively.

2.24.3 [5] <§2.7> For the loop written in RISC-V assembly above, assume that the register `x6` is initialized to the value N. How many RISC-V instructions are executed?

2.24.4 [5] <§2.7> For the loop written in RISC-V assembly above, replace the instruction "`beq x6, x0, DONE`" with the instruction "`blt x6, x0, DONE`" and write the equivalent C code.

- Solution:
  - 2.24.1 The final value of xs is 20 .
  - 2.24.2

```
acc = 0;
i = 10;
while (i ! = 0) {
        acc += 2;
        i--;
}
```

  - 2.24.3 4*N + 1 instructions.
  - 2.24.4 (Note: change condition ! = to > = in the while loop)

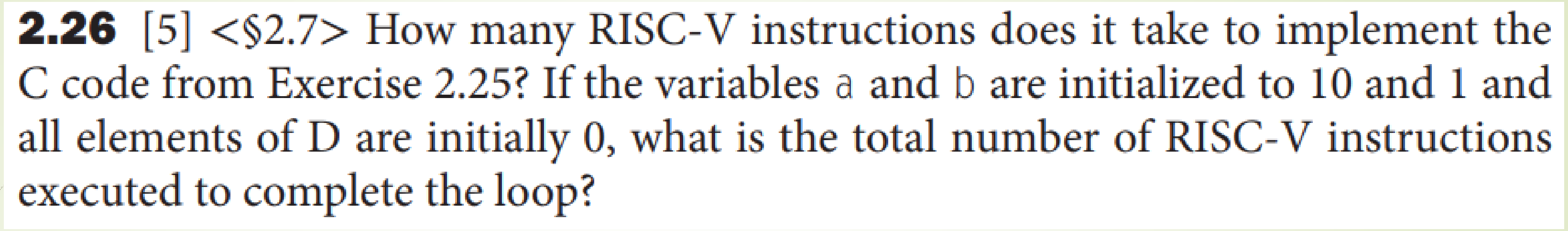
```
acc = 0;
i = 10;
while (i >= 0) {
        acc += 2;
        i--;
}
```

2.25 [10] <§2.7> Translate the following C code to RISC-V assembly code. Use a minimum number of instructions. Assume that the values of a, b, i, and j are in registers x5, x6, x7, and x29, respectively. Also, assume that register x10 holds the base address of the array D.

```
for(i=0; i<a; i++)
    for(j=0; j<b; j++)
        D[4*j] = i + j;
```

- Solution:

```
LOOPI:
        addi  x7,   x0, 0      // Init i = 0
        bge   x7,   x5, ENDI   // While i < a
        addi  x30,  x10, 0     // x30 = &D
        addi  x29,  x0, 0      // Init j = 0
LOOPJ:
        bge   x29,  x6, ENDJ   // While j < b
        add   x31,  x7, x29    // x31 = i+j
        sd    x31,  0(x30)     // D[4*j] = x31
        addi  x30,  x30, 32    // x30 = &D[4*(j+1)]
        addi  x29,  x29, 1     // j++
        jal   x0,   LOOPJ
ENDJ:
        addi  x7,   x7, 1      // i++;
        jal   x0,   LOOPI
ENDI:
```

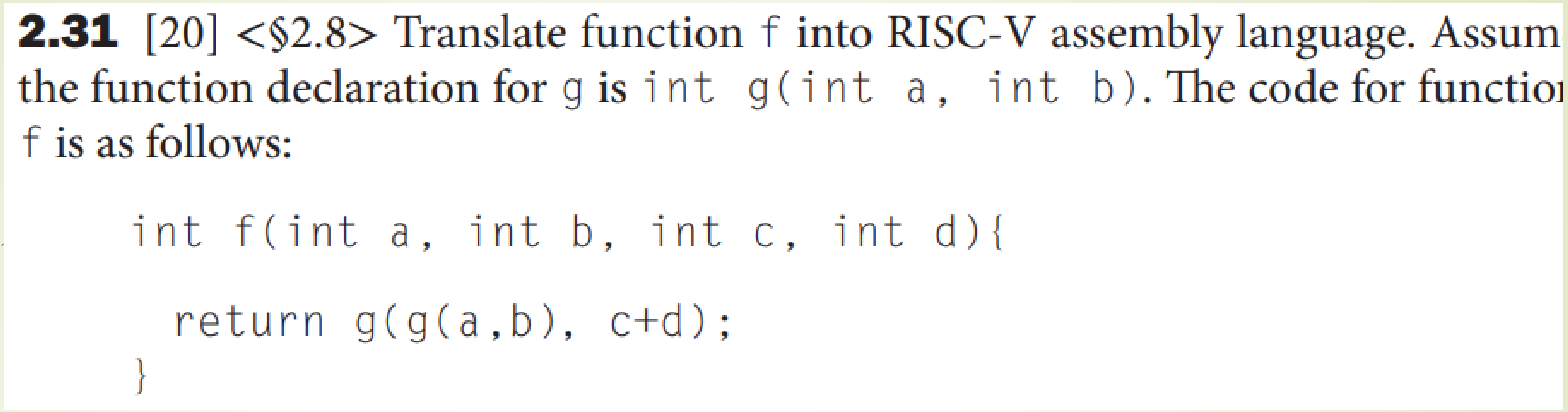


2.26 [5] <§2.7> How many RISC-V instructions does it take to implement the C code from Exercise 2.25? If the variables a and b are initialized to 10 and 1 and all elements of D are initially 0, what is the total number of RISC-V instructions executed to complete the loop?

- Solution:

  The code requires 13 RISC-V instructions.

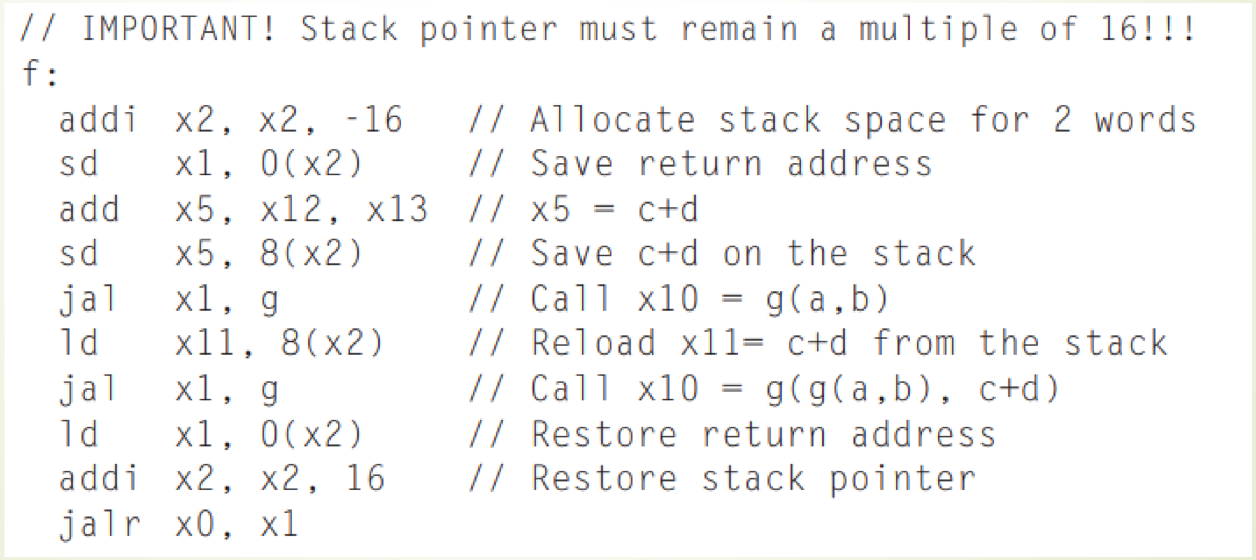
  When a = 10 and b = 1, this results in 123 instructions being executed.



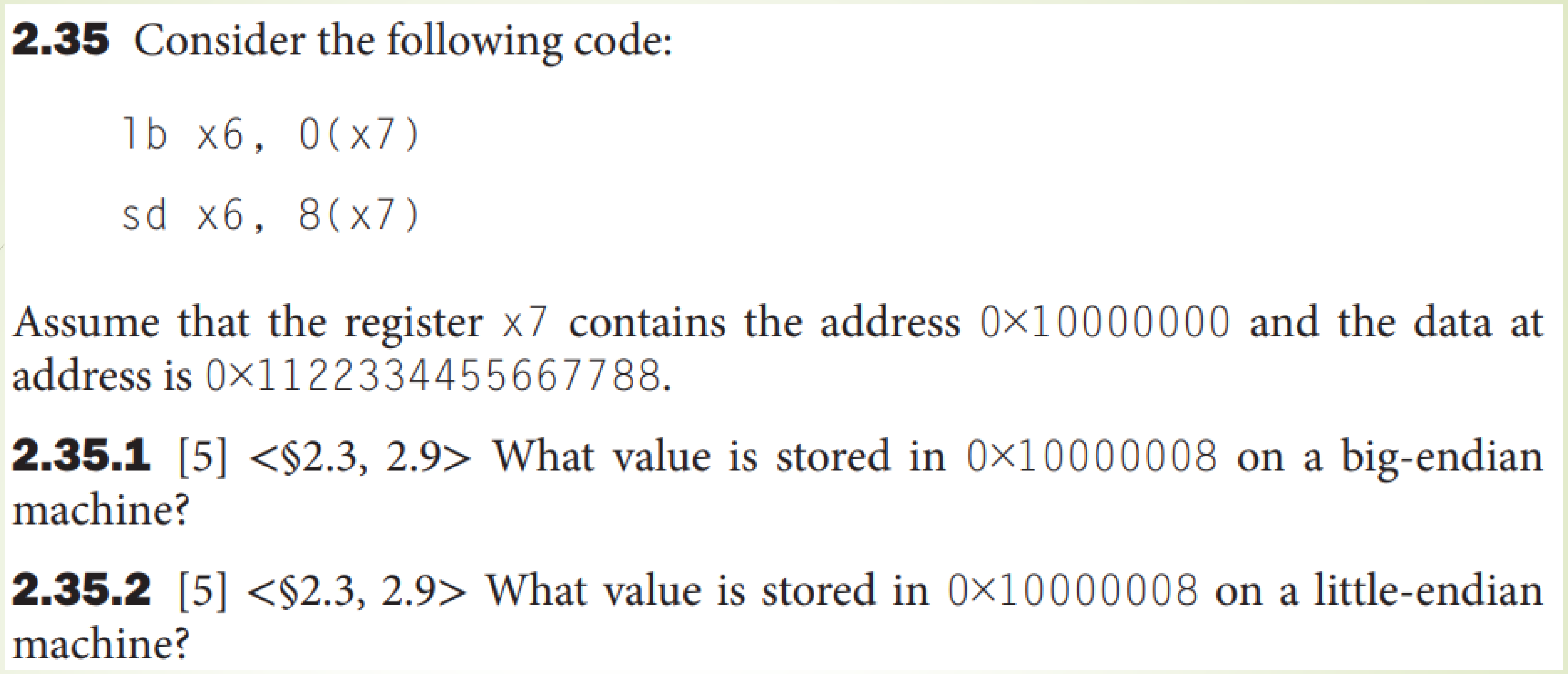
2.31 [20] <§2.8> Translate function `f` into RISC-V assembly language. Assum
the function declaration for `g` is `int g(int a, int b)`. The code for functio
`f` is as follows:

```
int f(int a, int b, int c, int d){
  return g(g(a,b), c+d);
}
```

- Solution:



```
// IMPORTANT! Stack pointer must remain a multiple of 16!!!
f:
  addi  x2, x2, -16   // Allocate stack space for 2 words
  sd    x1, 0(x2)     // Save return address
  add   x5, x12, x13  // x5 = c+d
  sd    x5, 8(x2)     // Save c+d on the stack
  jal   x1, g         // Call x10 = g(a,b)
  ld    x11, 8(x2)    // Reload x11= c+d from the stack
  jal   x1, g         // Call x10 = g(g(a,b), c+d)
  ld    x1, 0(x2)     // Restore return address
  addi  x2, x2, 16    // Restore stack pointer
  jalr  x0, x1
```



2.35 Consider the following code:

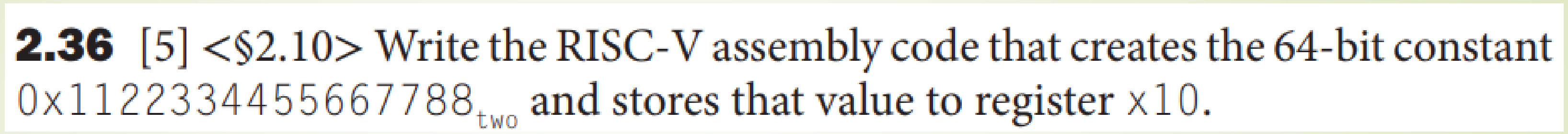
```
lb x6, 0(x7)
sd x6, 8(x7)
```

Assume that the register `x7` contains the address `0×10000000` and the data at address is `0×1122334455667788`.

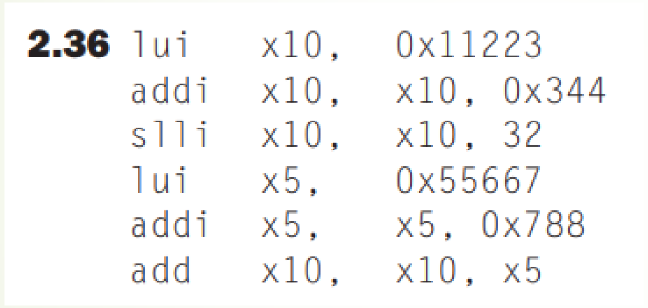
2.35.1 [5] <§2.3, 2.9> What value is stored in `0×10000008` on a big-endian machine?

2.35.2 [5] <§2.3, 2.9> What value is stored in `0×10000008` on a little-endian machine?

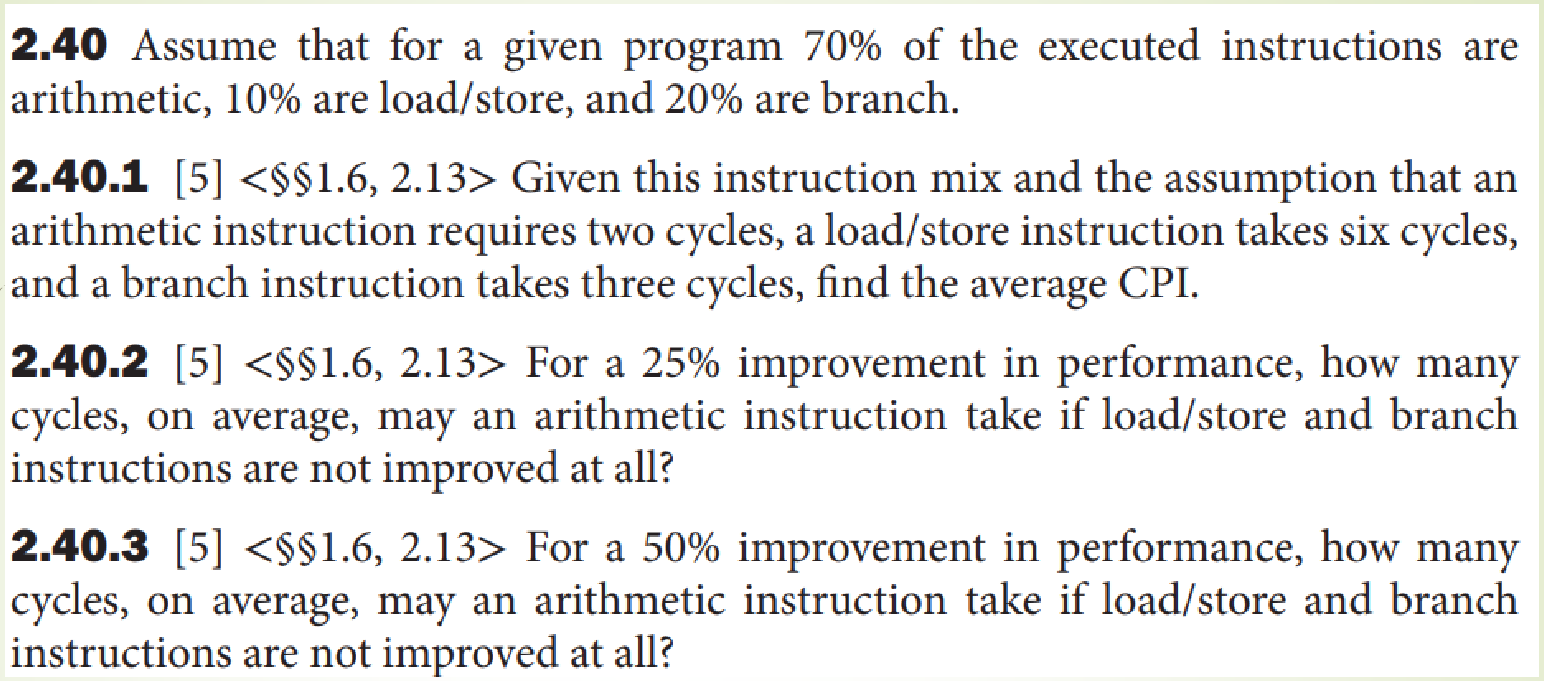
- Solution:
  - 2.35.1 0x11
  - 2.35.2 0x88


2.36 [5] <§2.10> Write the RISC-V assembly code that creates the 64-bit constant `0x1122334455667788`$_{two}$ and stores that value to register `x10`.

- Solution:


2.36

```
lui     x10,    0x11223
addi    x10,    x10, 0x344
slli    x10,    x10, 32
lui     x5,     0x55667
addi    x5,     x5, 0x788
add     x10,    x10, x5
```



2.40 Assume that for a given program 70% of the executed instructions are arithmetic, 10% are load/store, and 20% are branch.

2.40.1 [5] <§§1.6, 2.13> Given this instruction mix and the assumption that an arithmetic instruction requires two cycles, a load/store instruction takes six cycles, and a branch instruction takes three cycles, find the average CPI.

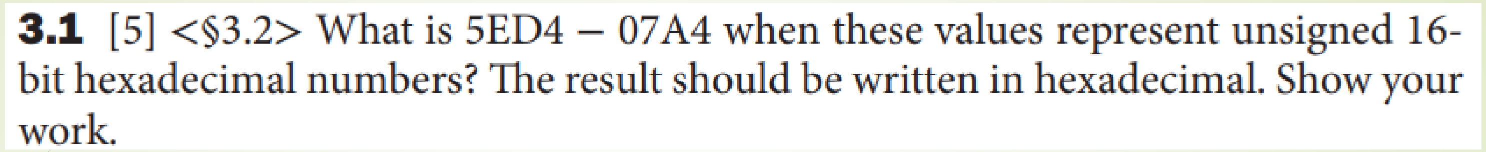
2.40.2 [5] <§§1.6, 2.13> For a 25% improvement in performance, how many cycles, on average, may an arithmetic instruction take if load/store and branch instructions are not improved at all?

2.40.3 [5] <§§1.6, 2.13> For a 50% improvement in performance, how many cycles, on average, may an arithmetic instruction take if load/store and branch instructions are not improved at all?

- Solution:
  - 2.40.1 Take the weighted average: $0.7*2 + 0.1*6 + 0.2*3 = 2.6$
  - 2.40.2 For a 25% improvement, we must reduce the CPU to $2.6*.75 = 1.95$.

    Thus, we want $0.7*x + 0.1*6 + 0.2*3 <= 1.95$. Solving for x shows that the arithmetic instructions must have a CPI of at most 1.07.
  - 2.40.3 For a 50% improvement, we must reduce the CPU to $2.6*.5 = 1.3$.

    Thus, we want $0.7*x + 0.1*6 + 0.2*3 <= 1.3$. Solving for x shows that the arithmetic instructions must have a CPI of at most 0.14



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

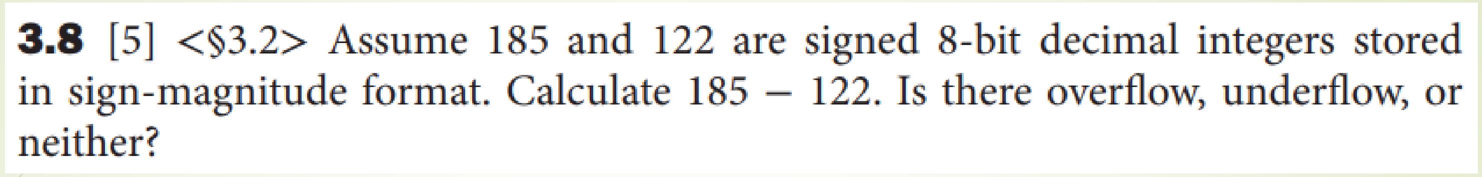
- Solution: 5730

3.5 [5] <§3.2> What is 4365 − 3412 when these values represent signed 12-bit octal numbers stored in sign-magnitude format? The result should be written in octal. Show your work.

- Solution: 7777 (-3777)

4365: 100 011 110 101 = $(-365)_8$

3412: 011 100 001 010 = 3412



3.8 [5] <§3.2> Assume 185 and 122 are signed 8-bit decimal integers stored in sign-magnitude format. Calculate 185 − 122. Is there overflow, underflow, or neither?

- Solution:

  Overflow (result = −179, which does not fi t into an SM 8-bit format)

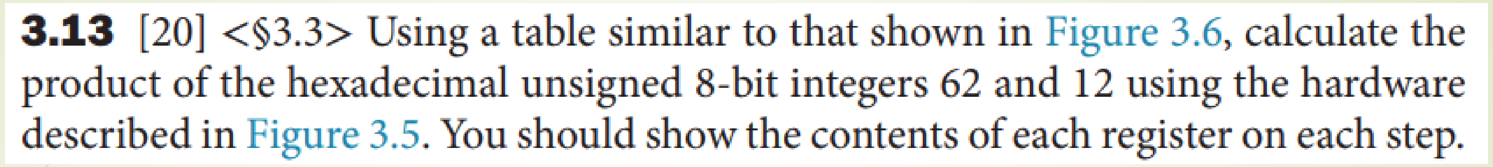
3.9 [10] <§3.2> Assume 151 and 214 are signed 8-bit decimal integers stored in two's complement format. Calculate 151 + 214 using saturating arithmetic. The result should be written in decimal. Show your work.

- Solution:

  151: 10010111 = - 1101001 = -105

  214: 11010110 = - 0101010 = - 42

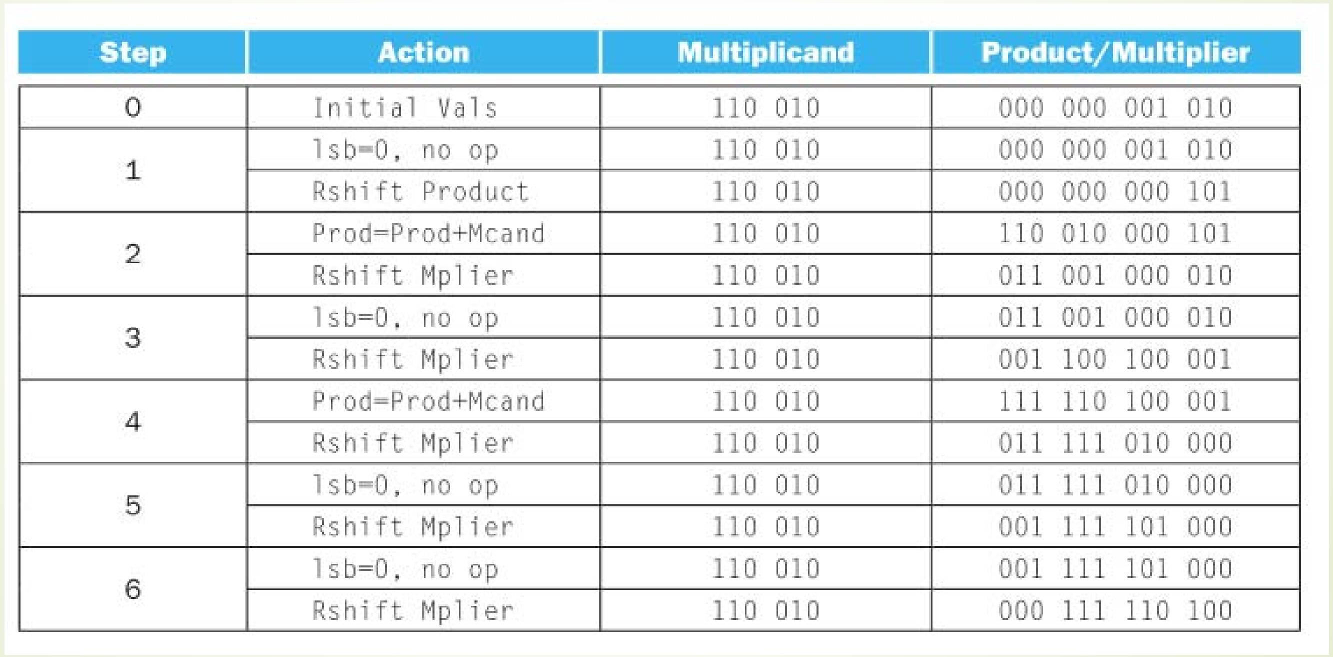
  −105 − 42 = −128 (− 147)



3.13 [20] <§3.3> Using a table similar to that shown in Figure 3.6, calculate the product of the hexadecimal unsigned 8-bit integers 62 and 12 using the hardware described in Figure 3.5. You should show the contents of each register on each step.

好像有问题

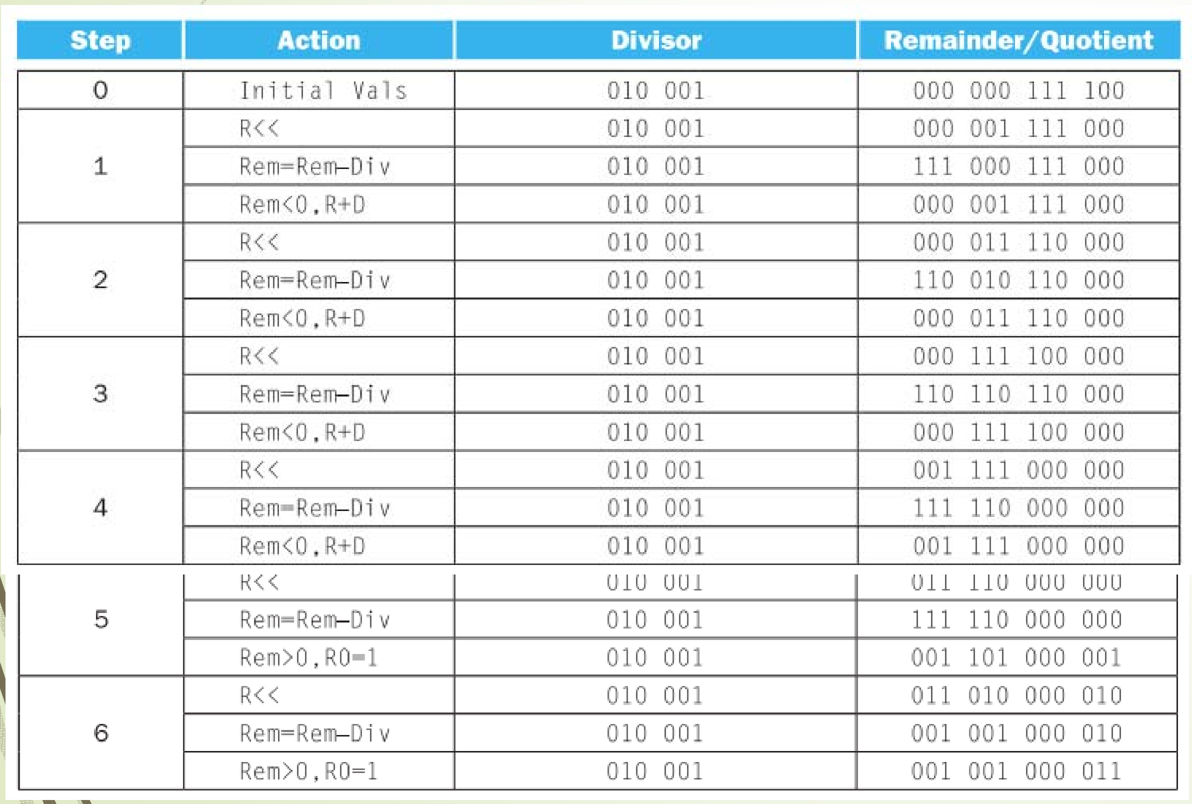
- Solution: Octal: 62 (110 010)x 12 = 764



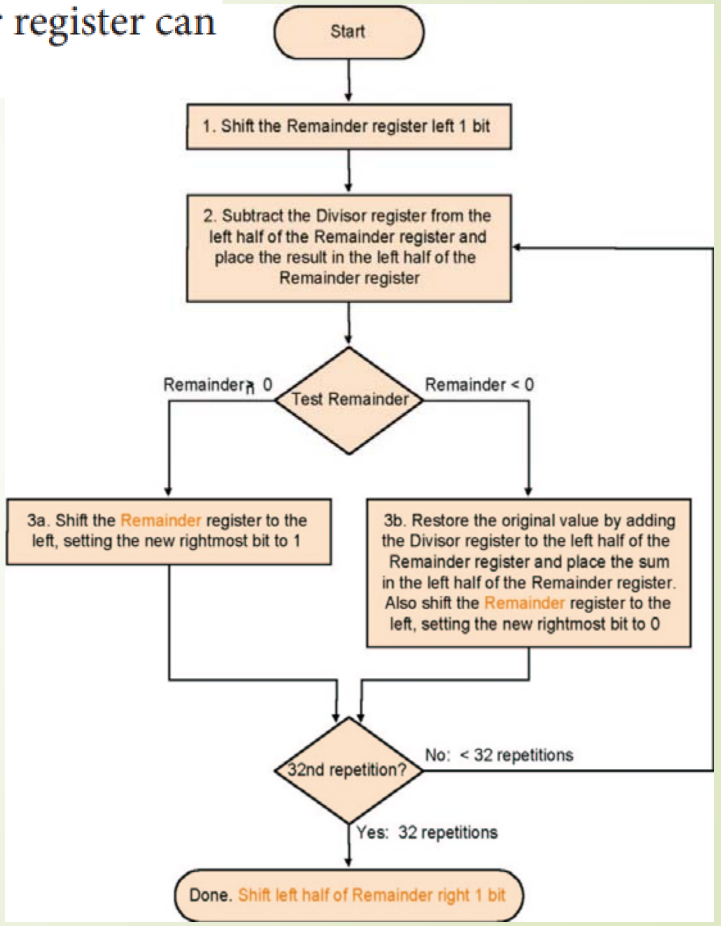
Step	Action	Multiplicand	Product/Multiplier
0	Initial Vals	110 010	000 000 001 010
1	lsb=0, no op	110 010	000 000 001 010
	Rshift Product	110 010	000 000 000 101
2	Prod=Prod+Mcand	110 010	110 010 000 101
	Rshift Mplier	110 010	011 001 000 010
3	lsb=0, no op	110 010	011 001 000 010
	Rshift Mplier	110 010	001 100 100 001
4	Prod=Prod+Mcand	110 010	111 110 100 001
	Rshift Mplier	110 010	011 111 010 000
5	lsb=0, no op	110 010	011 111 010 000
	Rshift Mplier	110 010	001 111 101 000
6	lsb=0, no op	110 010	001 111 101 000
	Rshift Mplier	110 010	000 111 110 100

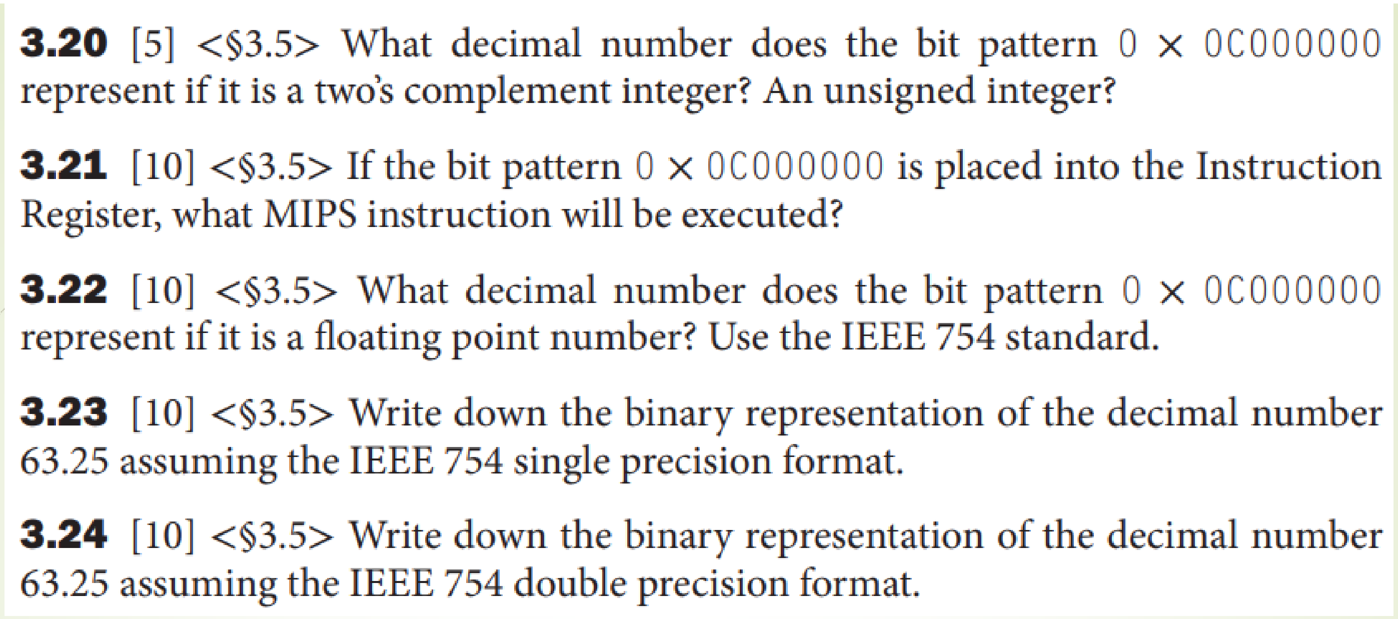
Hexadecimal: 62 (0110 0010) x 12 (0001 0010) = 0110 1110 0100 = 1764

3.19 [30] <§3.4> Using a table similar to that shown in Figure 3.10, calculate 74 divided by 21 using the hardware described in Figure 3.11. You should show the contents of each register on each step. Assume A and B are unsigned 6-bit integers. This algorithm requires a slightly different approach than that shown in Figure 3.9. You will want to think hard about this, do an experiment or two, or else go to the web to figure out how to make this work correctly. (Hint: one possible solution involves using the fact that Figure 3.11 implies the remainder register can be shifted either direction.)



Step	Action	Divisor	Remainder/Quotient
0	Initial Vals	010 001	000 000 111 100
1	R<<	010 001	000 001 111 000
	Rem=Rem–Div	010 001	111 000 111 000
	Rem<0,R+D	010 001	000 001 111 000
2	R<<	010 001	000 011 110 000
	Rem=Rem–Div	010 001	110 010 110 000
	Rem<0,R+D	010 001	000 011 110 000
3	R<<	010 001	000 111 100 000
	Rem=Rem–Div	010 001	110 110 110 000
	Rem<0,R+D	010 001	000 111 100 000
4	R<<	010 001	001 111 000 000
	Rem=Rem–Div	010 001	111 110 000 000
	Rem<0,R+D	010 001	001 111 000 000
5	R<<	010 001	011 110 000 000
	Rem=Rem–Div	010 001	111 110 000 000
	Rem>0,R0=1	010 001	001 101 000 001
6	R<<	010 001	011 010 000 010
	Rem=Rem–Div	010 001	001 001 000 010
	Rem>0,R0=1	010 001	001 001 000 011


Start
1. Shift the Remainder register left 1 bit
2. Subtract the Divisor register from the left half of the Remainder register and place the result in the left half of the Remainder register
Test Remainder
Remainder ≥ 0
Remainder < 0
3a. Shift the Remainder register to the left, setting the new rightmost bit to 1
3b. Restore the original value by adding the Divisor register to the left half of the Remainder register and place the sum in the left half of the Remainder register. Also shift the Remainder register to the left, setting the new rightmost bit to 0
32nd repetition?
No: < 32 repetitions
Yes: 32 repetitions
Done. Shift left half of Remainder right 1 bit



3.20 [5] <§3.5> What decimal number does the bit pattern 0 × 0C000000 represent if it is a two's complement integer? An unsigned integer?

3.21 [10] <§3.5> If the bit pattern 0 × 0C000000 is placed into the Instruction Register, what MIPS instruction will be executed?

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.

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

3.24 [10] <§3.5> Write down the binary representation of the decimal number 63.25 assuming the IEEE 754 double precision format.

- Solution:
- 3.20 $12*16^6$ = 201326592 in both cases.
- 3.21 jal 0x00000000
- 3.22 0x0C000000 = 0000 1100 0000 0000 0000 0000 0000 0000

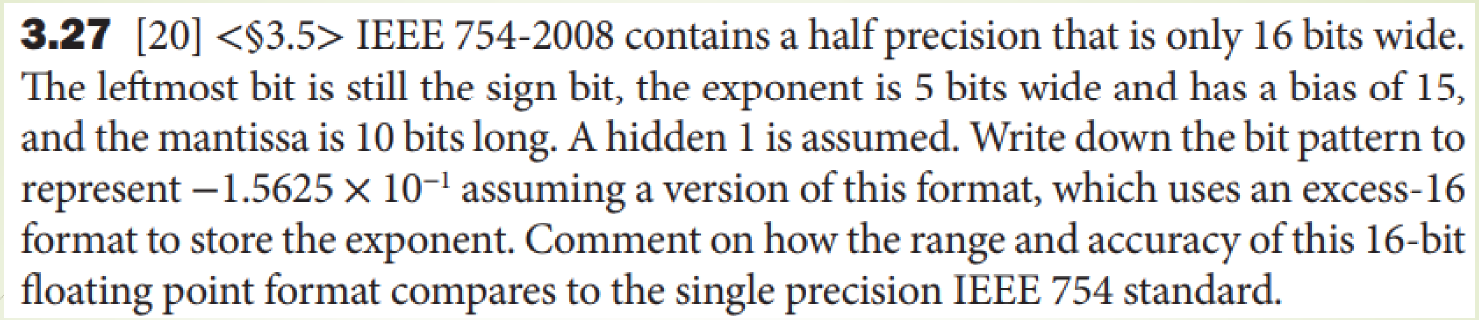
  sign is positive, exp = 24 − 127 = − 103, fraction = 0, so answer = 1.0×2^{-103}
- 3.23 63.25 = 111111.01 = 1.1111101×2^5

  sign = positive, exp = 127 + 5 = 132

  so IEEE 754 single format = 0 1000 0100 1111 1010 0000 0000 0000 000 = 0x427D0000
- 3.24 sign = positive, exp = 1023 + 5 = 1028

  so IEEE 754 double format = 0 100 0000 0100 1111 1010 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

  = 0x404FA00000000000



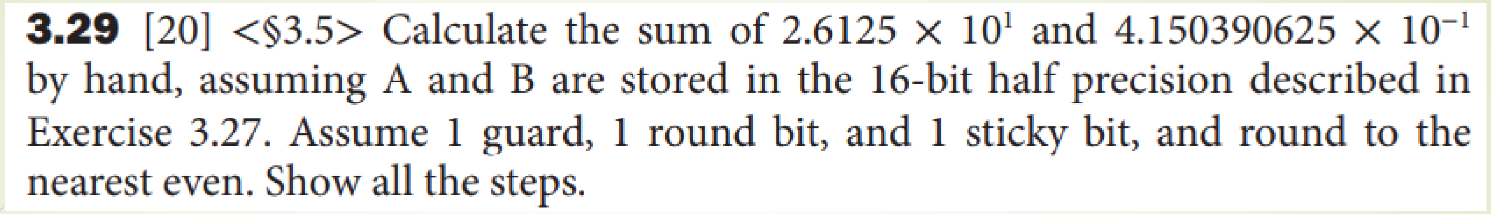
3.27 [20] <§3.5> IEEE 754-2008 contains a half precision that is only 16 bits wide. The leftmost bit is still the sign bit, the exponent is 5 bits wide and has a bias of 15, and the mantissa is 10 bits long. A hidden 1 is assumed. Write down the bit pattern to represent -1.5625×10^{-1} assuming a version 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.

- Solution:
- $-1.5625 \times 10^{-1} = -0.00101 = -1.01 \times 2^{-3}$

  exponent = $-3 + 15 = 12$: 01100,

  fraction = $-$ 0.0100000000

  answer: 1 01100 0100000000



3.29 [20] <§3.5> Calculate the sum of 2.6125×10^1 and $4.150390625 \times 10^{-1}$ 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.

- Solution:

$2.6125 \times 10^1 + 4.150390625 \times 10^{-1}$

$2.6125 \times 10^1 = 26.125 = 11010.001 = 1.1010001000 \times 2^4$

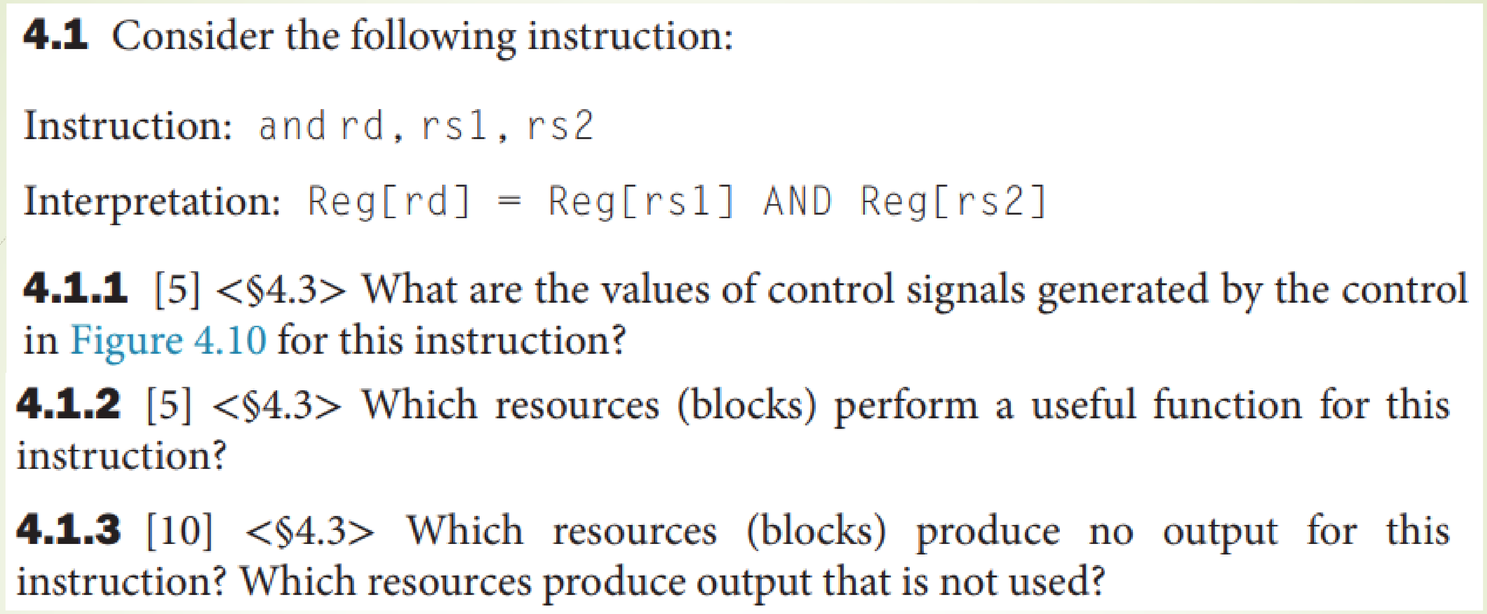
$4.150390625 \times 10^{-1} = 0.4150390625 = 0.011010100111$

$= 1.1010100111 \times 2^{-2} = 0.0000011010100111 \times 2^4$

```
                    G R
1.1010001000  0 0
0.0000011010  1 0 0111 (So Sticky is 1)
-----------------------------------
1.1010100010  1 0
```

In this case the extra bit (G,R,S) is more than half of the least significant bit (0). Thus, the value is rounded up.

So result is $1.1010100011 \times 2^4 = 11010.100011 = 26.546875 = 2.6546875 \times 10^1$


4.1 Consider the following instruction:

Instruction: `and rd, rs1, rs2`

Interpretation: `Reg[rd] = Reg[rs1] AND Reg[rs2]`

4.1.1 [5] <§4.3> What are the values of control signals generated by the control in Figure 4.10 for this instruction?

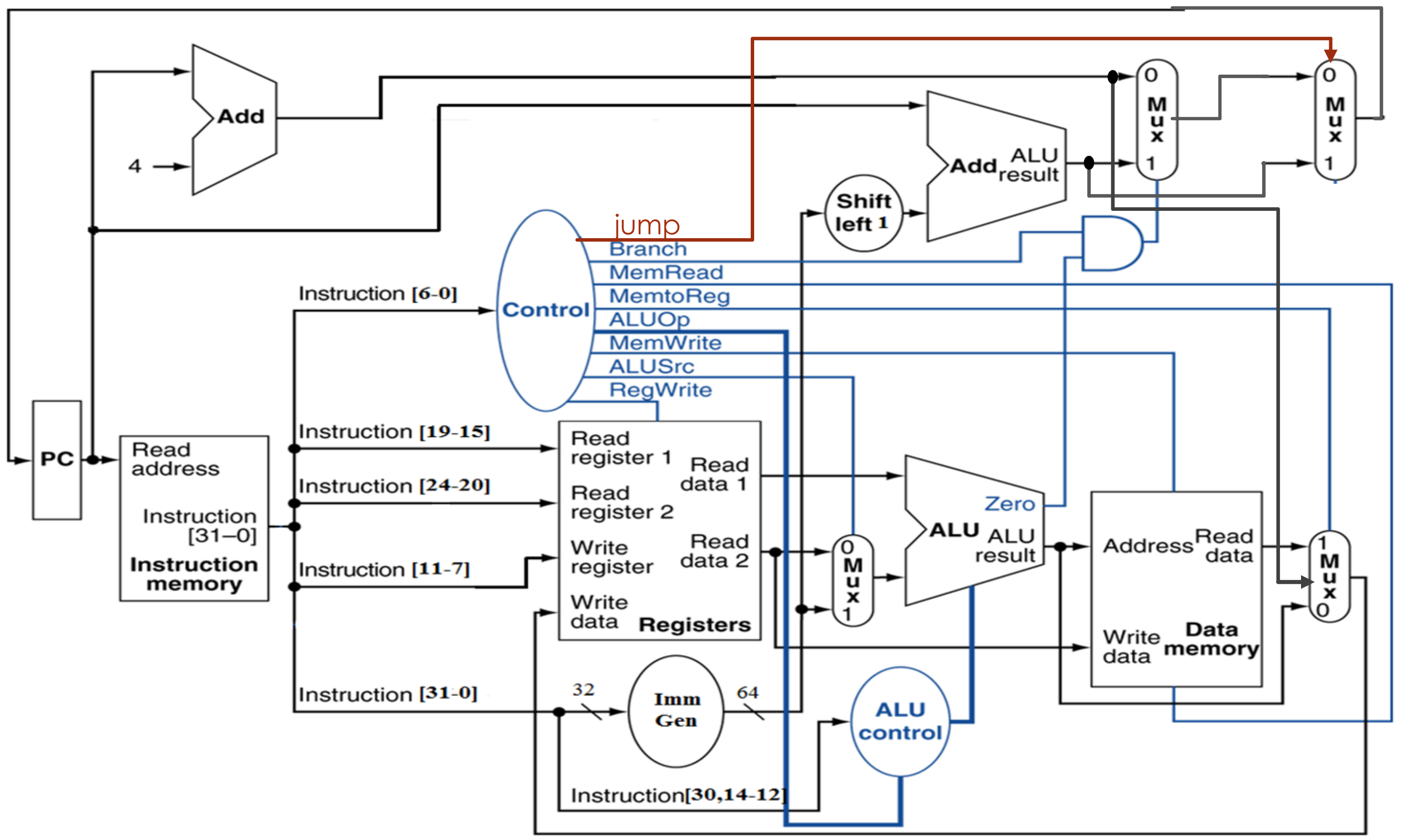
4.1.2 [5] <§4.3> Which resources (blocks) perform a useful function for this instruction?

4.1.3 [10] <§4.3> Which resources (blocks) produce no output for this instruction? Which resources produce output that is not used?

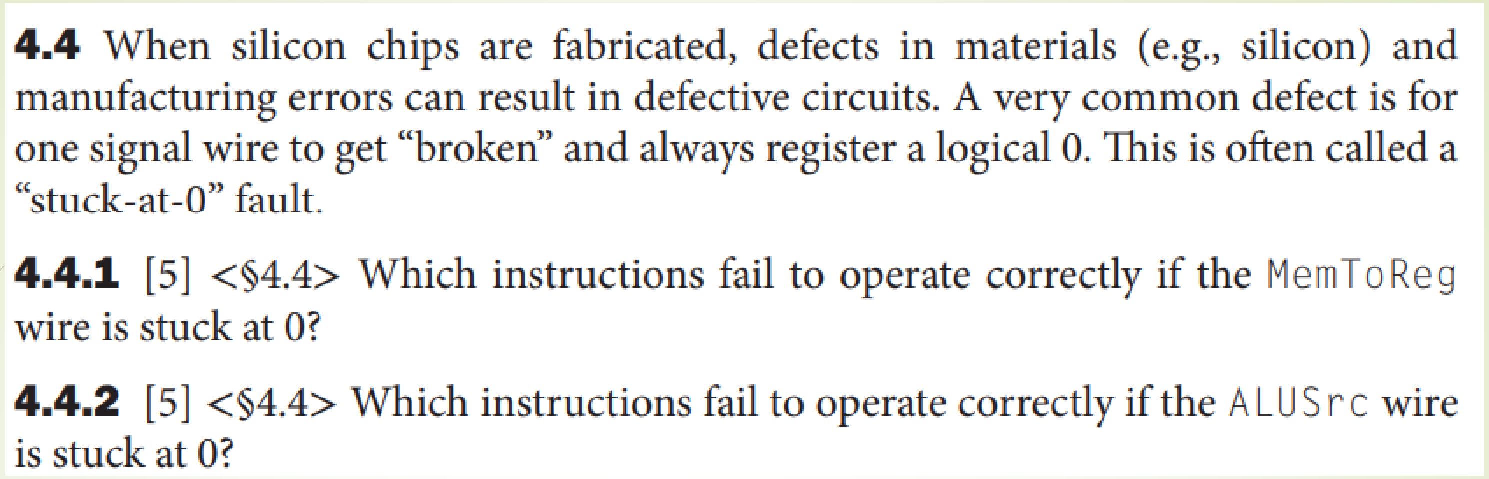
- Solution:

Instruction	ALUSrc	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU Op1	ALU Op2	ALU operation
And	0	0	1	0	0	0	0	1	0	and

- Registers, ALUSrc mux, ALU, MemtoReg mux
- All blocks produce some output. The output of Data Memory and Imm Gen are not used.



输入		输出								
Instruction	OPCode	ALUSrcB	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALUOp1	ALU Op0
R-format	0110011	0	00	1	0	0	0	0	1	0
Ld(I-Type)	0000011	1	01	1	1	0	0	0	0	0
sd(S-Type)	0100011	1	X	0	0	1	0	0	0	0
beq(SB-Type)	1100111	0	X	0	0	0	1	0	0	1
Jal(UJ-Type)	1101111	X	10	1	0	0	0	1	X	X

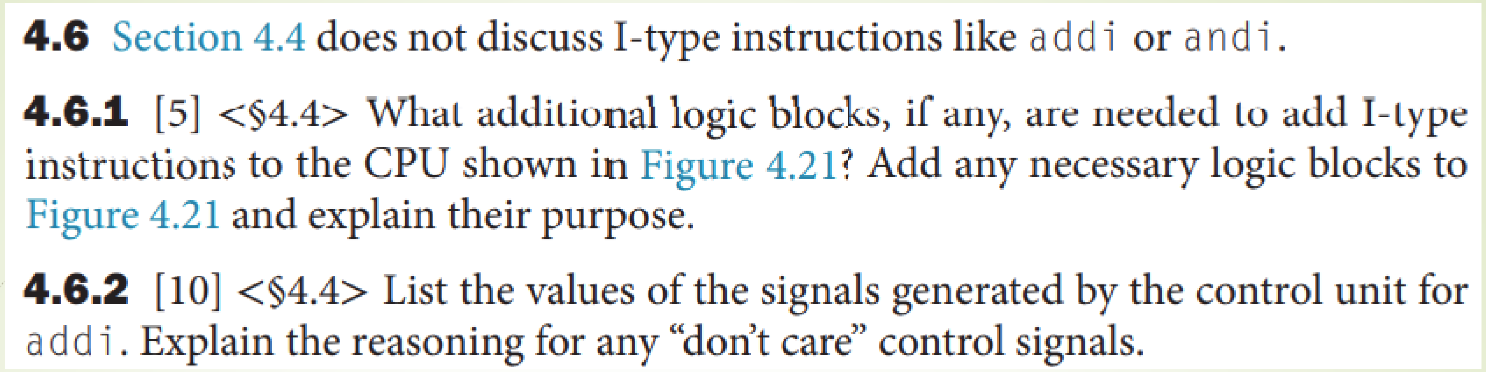


4.4 When silicon chips are fabricated, defects in materials (e.g., silicon) and manufacturing errors can result in defective circuits. A very common defect is for one signal wire to get "broken" and always register a logical 0. This is often called a "stuck-at-0" fault.

4.4.1 [5] <§4.4> Which instructions fail to operate correctly if the `MemToReg` wire is stuck at 0?

4.4.2 [5] <§4.4> Which instructions fail to operate correctly if the `ALUSrc` wire is stuck at 0?

- Solution:
  - 1. loads and jal instructions are broken
  - I-type, Loads and stores instructions are broken

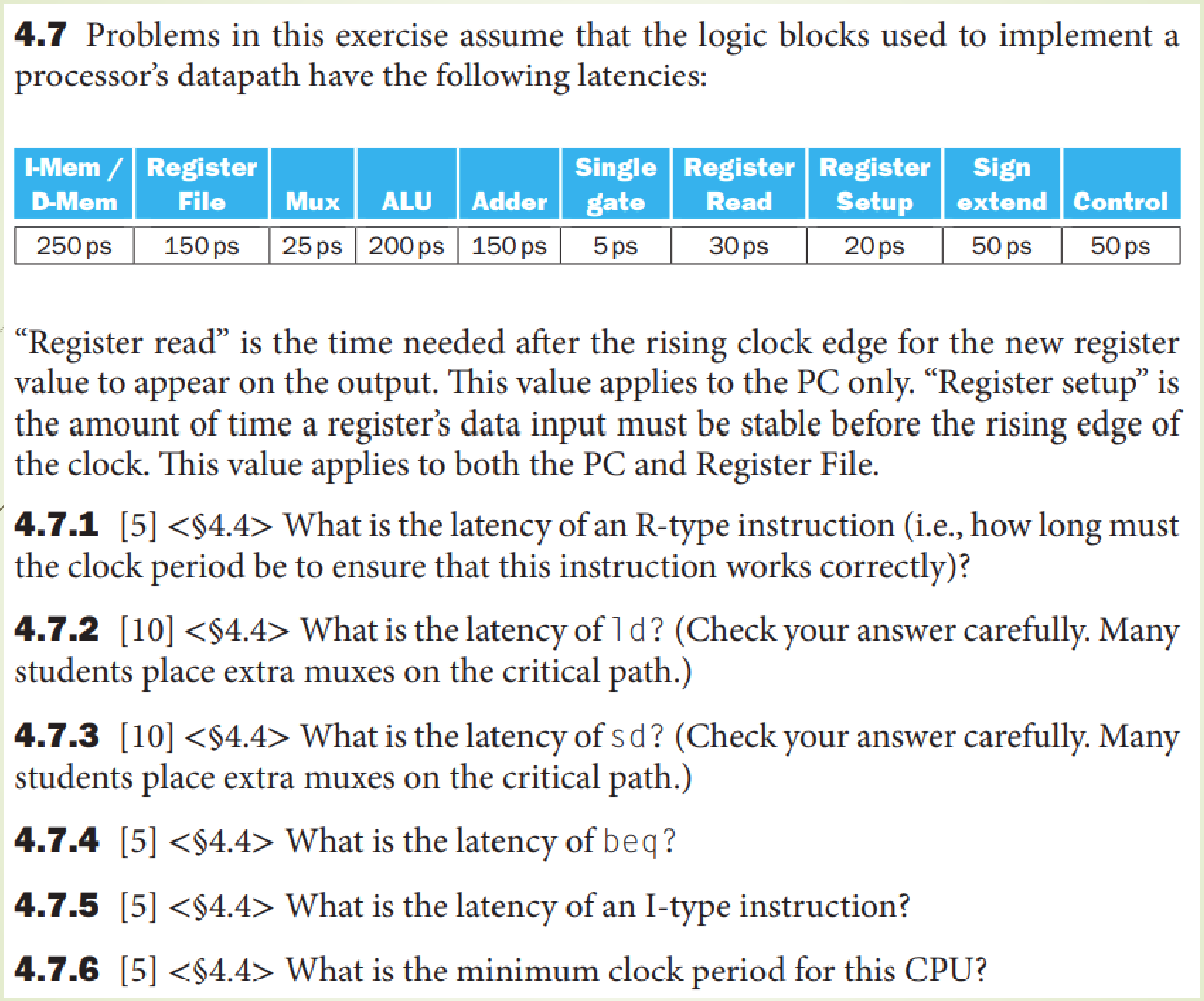

4.6 Section 4.4 does not discuss I-type instructions like `addi` or `andi`.

4.6.1 [5] <§4.4> What additional logic blocks, if any, are needed to add I-type instructions to the CPU shown in Figure 4.21? Add any necessary logic blocks to Figure 4.21 and explain their purpose.

4.6.2 [10] <§4.4> List the values of the signals generated by the control unit for `addi`. Explain the reasoning for any "don't care" control signals.

- Solution:

Instruction	ALUSrc	MemtoReg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU Op1	ALU Op2	ALU operation
addi	1	00	1	0	0	0	0	1	1	add
andi	1	00	1	0	0	0	0	1	1	and



4.7 Problems in this exercise assume that the logic blocks used to implement a processor's datapath have the following latencies:

I-Mem / D-Mem	Register File	Mux	ALU	Adder	Single gate	Register Read	Register Setup	Sign extend	Control
250 ps	150 ps	25 ps	200 ps	150 ps	5 ps	30 ps	20 ps	50 ps	50 ps

"Register read" is the time needed after the rising clock edge for the new register value to appear on the output. This value applies to the PC only. "Register setup" is the amount of time a register's data input must be stable before the rising edge of the clock. This value applies to both the PC and Register File.

4.7.1 [5] <§4.4> What is the latency of an R-type instruction (i.e., how long must the clock period be to ensure that this instruction works correctly)?

4.7.2 [10] <§4.4> What is the latency of `ld`? (Check your answer carefully. Many students place extra muxes on the critical path.)

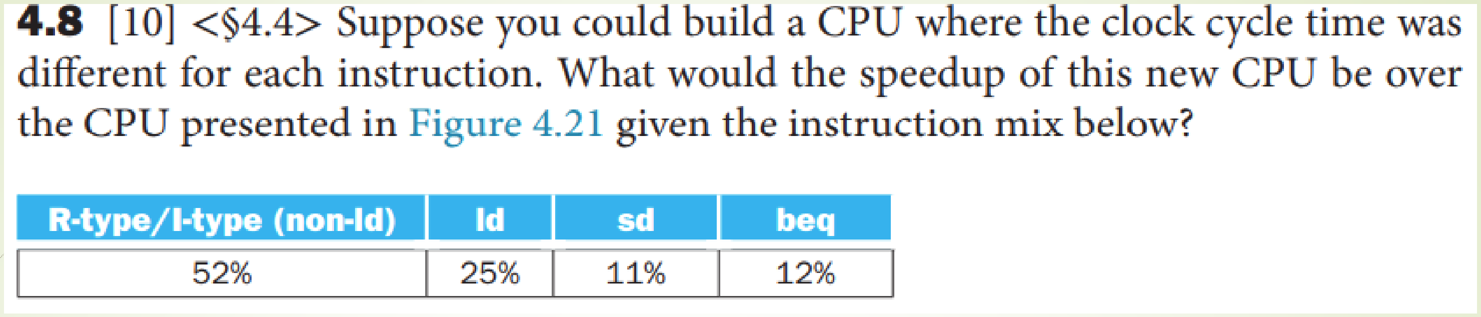
4.7.3 [10] <§4.4> What is the latency of `sd`? (Check your answer carefully. Many students place extra muxes on the critical path.)

4.7.4 [5] <§4.4> What is the latency of `beq`?

4.7.5 [5] <§4.4> What is the latency of an I-type instruction?

4.7.6 [5] <§4.4> What is the minimum clock period for this CPU?

- Solution:
- 4.7.1 R-type : 30 + 250 + 150 + 25 + 200 + 25 + 20 = 700ps
- 4.7.2 ld : 30 + 250 + 150 + 25 + 200 + 250 + 25 + 20 = 950 ps
- 4.7.3 sd : 30 + 250 + 150 + 200 + 25 + 250 = 905 ps
- 4.7.4 beq : 30 + 250 + 150 + 25 + 200 + 5 + 25 + 20 = 705 ps
- 4.7.5 I-type : 30 + 250 + 150 + 25 + 200 + 25 + 20 = 700ps
- 4.7.6 CPU clock is the longest latency: 950 ps



4.8 [10] <§4.4> Suppose you could build a CPU where the clock cycle time was different for each instruction. What would the speedup of this new CPU be over the CPU presented in Figure 4.21 given the instruction mix below?

R-type/I-type (non-ld)	ld	sd	beq
52%	25%	11%	12%

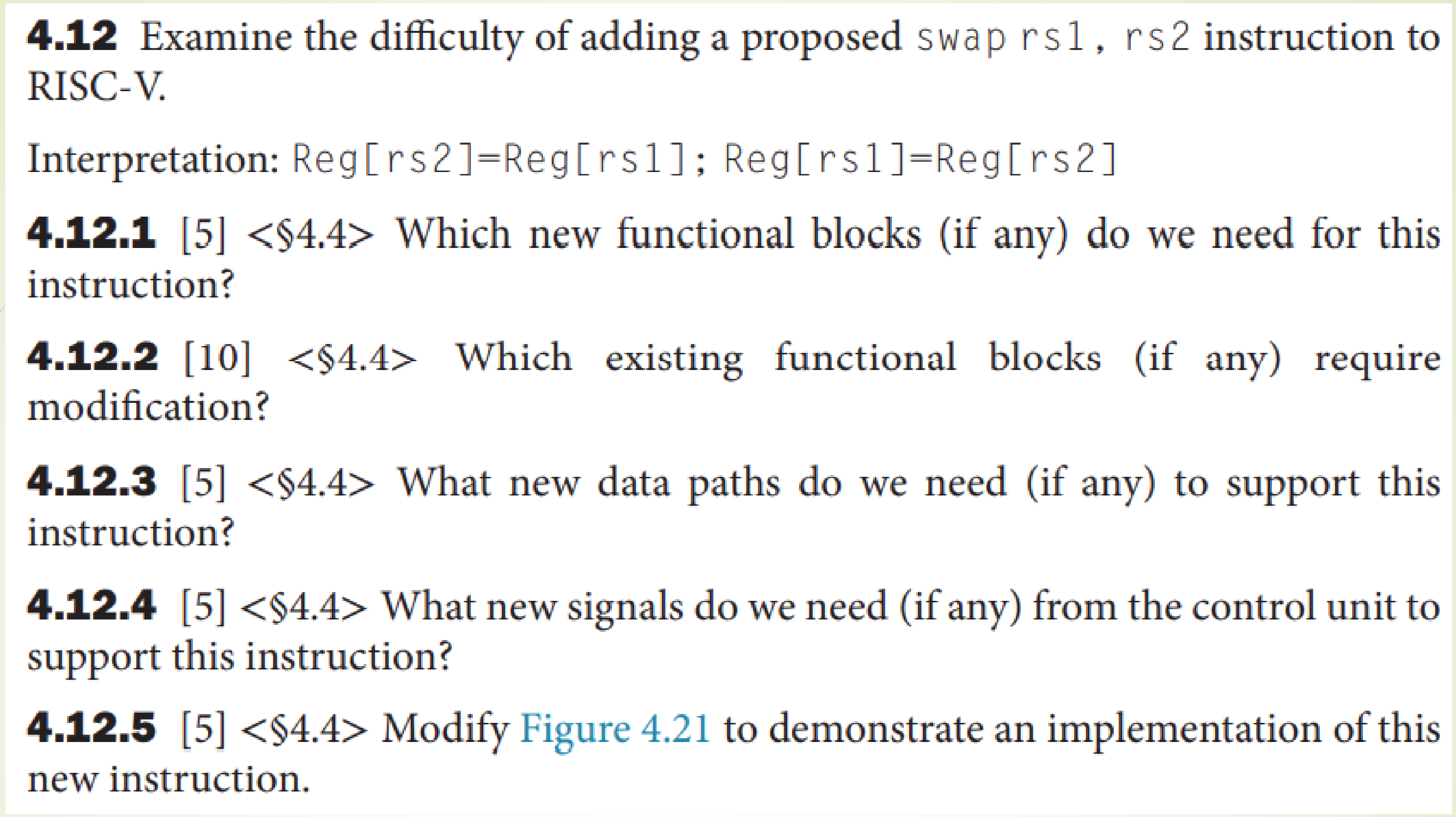
- Solution:

Using the results from Problem 4.7, we see that the average time per instruction is

.52*700 + .25*950 + .11*905 + .12 * 705 = 785.6ps

In contrast, a single-cycle CPU with a "normal" clock would require a clock cycle time of 950ps.

Thus, the speedup would be 950/785.6 = 1.209



4.12 Examine the difficulty of adding a proposed `swap rs1, rs2` instruction to RISC-V.

Interpretation: `Reg[rs2]=Reg[rs1]; Reg[rs1]=Reg[rs2]`

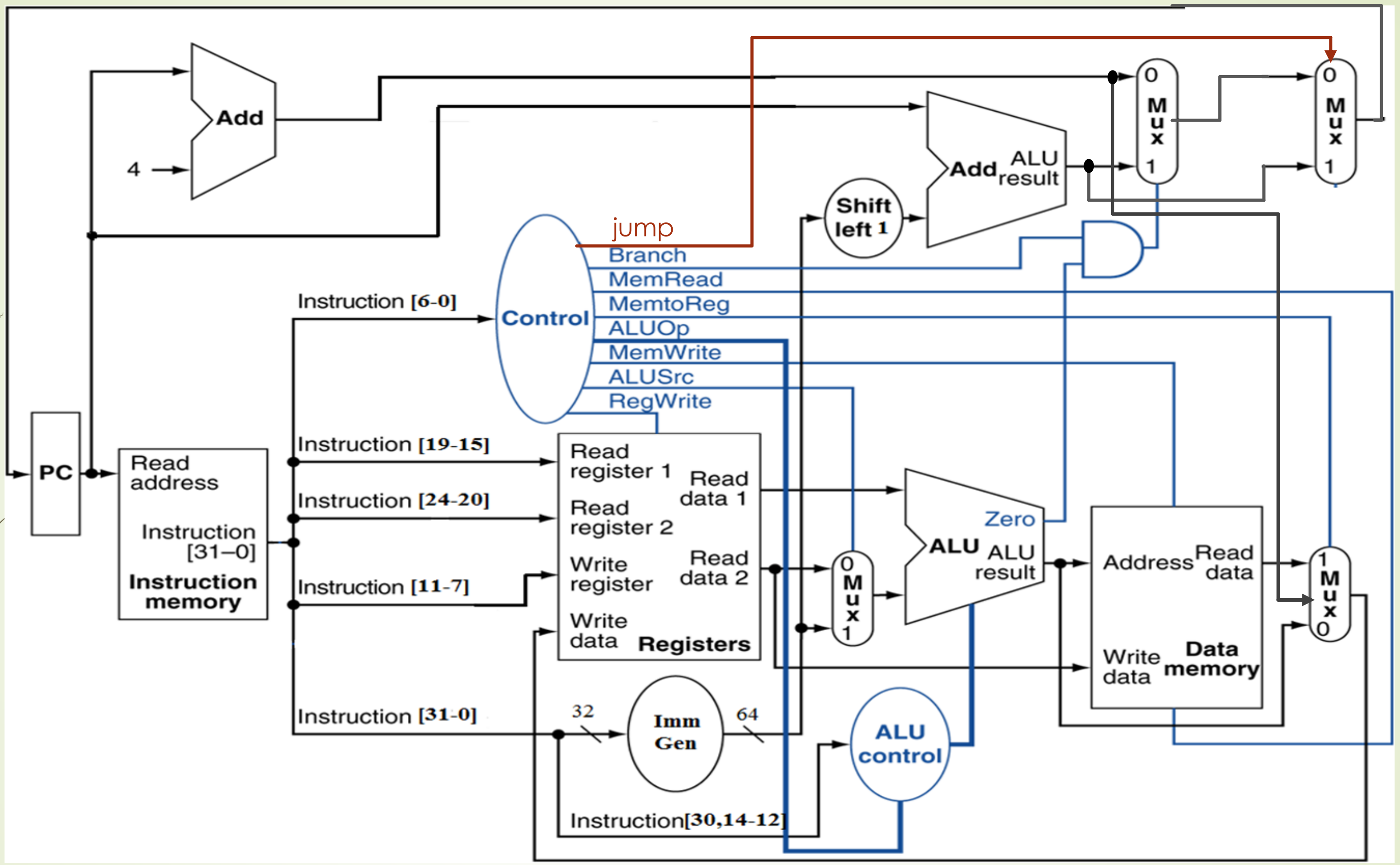
4.12.1 [5] <§4.4> Which new functional blocks (if any) do we need for this instruction?

4.12.2 [10] <§4.4> Which existing functional blocks (if any) require modification?

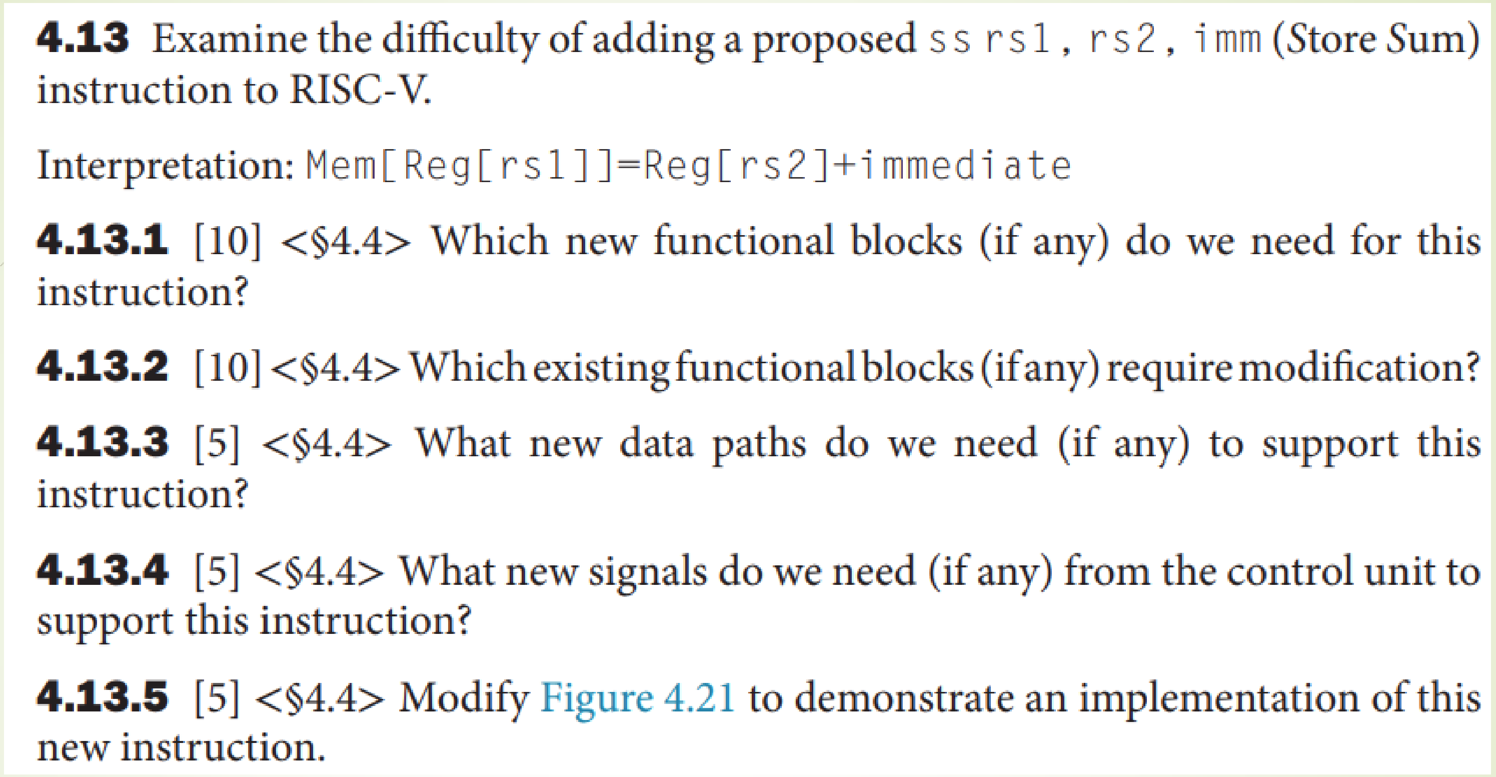
4.12.3 [5] <§4.4> What new data paths do we need (if any) to support this instruction?

4.12.4 [5] <§4.4> What new signals do we need (if any) from the control unit to support this instruction?

4.12.5 [5] <§4.4> Modify Figure 4.21 to demonstrate an implementation of this new instruction.



<table>
<tr><th colspan="2">输入</th><th colspan="9">输出</th></tr>
<tr><td>Instruction</td><td>OPCode</td><td>ALUSrcB</td><td>Memto-Reg</td><td>Reg Write</td><td>Mem Read</td><td>Mem Write</td><td>Branch</td><td>Jump</td><td>ALUOp1</td><td>ALU Op0</td></tr>
<tr><td>swap</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>



4.13 Examine the difficulty of adding a proposed `ss rs1, rs2, imm` (*Store Sum*) instruction to RISC-V.

Interpretation: `Mem[Reg[rs1]]=Reg[rs2]+immediate`

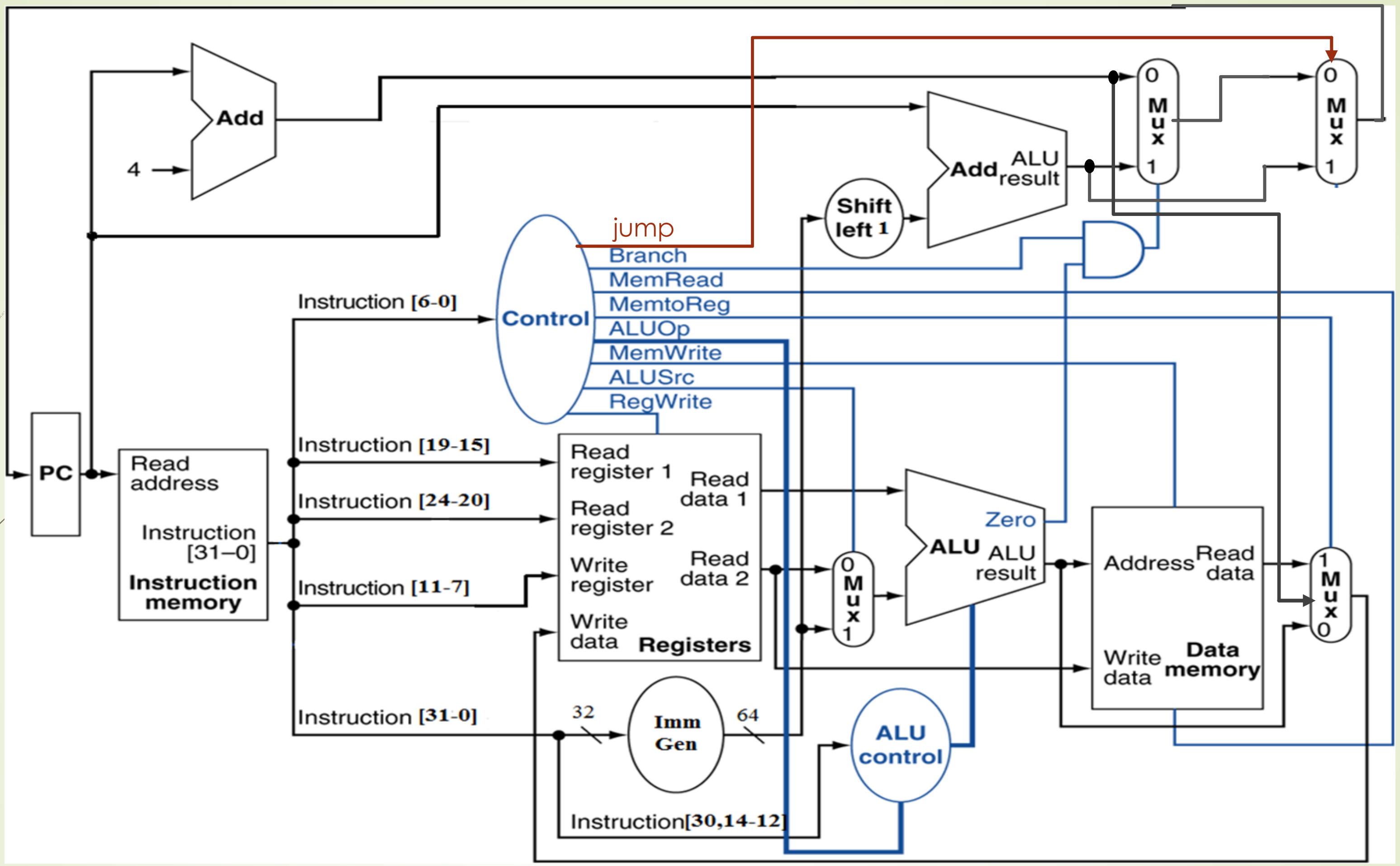
4.13.1 [10] <§4.4> Which new functional blocks (if any) do we need for this instruction?

4.13.2 [10] <§4.4> Which existing functional blocks (if any) require modification?

4.13.3 [5] <§4.4> What new data paths do we need (if any) to support this instruction?

4.13.4 [5] <§4.4> What new signals do we need (if any) from the control unit to support this instruction?

4.13.5 [5] <§4.4> Modify Figure 4.21 to demonstrate an implementation of this new instruction.



输入		输出								
Instruction	OPCode	ALUSrcB	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALUOp1	ALU Op0
ss										