Chapter 2

"Instructions : Language of the Computer"

Exercise Solution

→ 2.1:

Ans: $f = g + (h-5)$;

```
addi f, h, -5
add  f, g, f
```

→ 2.2:

Ans:
```
add f, g, h
add f, i, f
```

$f = i + (g+h)$

→ 2.3:

Ans: $B[8] = A[i-j]$

```
sub  $t0, $s3, $s4
sll  $t0, $t0, 2
add  $t0, $s6, $t0
lw   $t1, 0($t0)
sw   $t1, 32($s7)
```

→ 2.4:

Ans:
```
1. sll  $t0, $s0, 2    # $t0 = f*4;
2. add  $t0, $s6, $t0  # $t0 = &A[f];
3. sll  $t1, $s1, 2    # $t1 = g*4
4. add  $t1, $s7, $t1  # $t1 = &B[g];
5. lw   $s0, 0($t0)    # f = A[F];
6. addi $t2, $t0, 4    # $t2 = &A[f+1];
7. lw   $t0, 0($t2)    # $t0 = A[F+1];
8. add  $t0, $t0, $s0  # $t0 = A[F+1]+A[F];
9. sw   $t0, 0($t1)    # B[g] = A[F+1]+A[F];
```

overall B[g] = A[F+1] + A[F];

→ 2.5:

Ans: In the above MIPS code we can only remove the line 6 and modify line 7 by

```
7. lw  $t0, 4($t0)
8. add $t0, $t0, $s0
9. sw  $t0, 0($t1)
```

→ 2.7:

Ans: value → 0xabcdef12

Little Endian

Address	Data
3	ab
2	cd
1	ef
0	12

Big Endian

Address	Data
3	12
2	ef
1	cd
0	ab

→ 2.8:

Ans:

0xabcdef12 in decimal → 2882400018

→ 2.9:

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

```
sll  $t0, $s3, 2      # $t0 = i*4
add  $t0, $t0, $s6    # $t0 = &A[i]
lw   $t0, 0($t0)      # $t0 = A[i]
sll  $t1, $s4, 2      # $t1 = j*4
add  $t1, $t1, $s6    # $t1 = A[j]
lw   $t1, 0($t1)      # $t1 = A[j]
```

```
add $t0, $t0, $t1   # t0 = A[i] + A[j]
sw  $t0, 32($s7)    # B[8] = A[i] + A[j].
```

→ 2.10:

Ans:

```
addi $t0, $s6, 4    # $t0 = A[i]
add  $t1, $s6, $0   # $t1 = A[0]
sw   $t1, 0($t0)    # $t0 = A[0]
lw   $t0, 0($t0)    # $t0 = A[0]
add  $s0, $t1, $t0  # f = A[0] + A[0]
```

f = A[0] + A[0]

→ 2.11:

Ans:

	type	opcode	rs	rt	rd	immed
addi $t0, $s6, 4	I-type	8	22	8		4
add $t1, $s6, $0	R-type	0	22	0	9	
sw $t1, 0($t0)	I-type	43	8	9		0
lw $t0, 0($t0)	I-type	35	8	8		0
add $s0, $t1, $t0	R-type	0	9	8	16	

→ 2.12.1:

Ans:

$s0 → 0x80000000

$s1 → 0xD0000000

∴ add $t0, $s0, $s1

value of $t0 → 0x150000000

→ 2.12.2:

Ans:

the result in $t0 is overflow

→ 2.12.3:

Ans:

sub $t0, $s0, $s1

0xB0000000

→ 2.12.4:

Ans:

~~the~~ No overflow

→ 2.12.5:

Ans: 0x1D0000000.

→ 2.13.6:

Ans: Overflow.

→ 2.13.1:

Ans:

add $t0, $s0, $s1

The memory allocated to one instruction is 4 bytes

The range of numbers can be calculated as 2^{n-1}

$n = 32$

∴ The range is 2^{31} to $-$ ~~2^{31}~~ 2^{31}

= 2,147,483,647 to -2,147,483,647

∴ Range of $s1 = 2147483647 - 128

= 2147483519

→ 2.13.2

Ans:

sub $t0, $s0, $s1

Range of $s1 = 128 - 2147483647

= -2147483519

→ 2.13.3:

Ans:
sub $t0, $sL $s0

range of $s1 = -2147483648 + 128
= - 2147483520

→ 2.16:

Ans: sub $t3, $s3, $v0
op = 0, rs = 3, rt = 2, rd = 3, shamt = 0, funct = 34

$\underset{0}{\underline{000000}}$ $\underset{3}{\underline{00011}}$ $\underset{2}{\underline{00010}}$ $\underset{3}{\underline{00011}}$ $\underset{0}{\underline{00000}}$ $\underset{34}{\underline{100010}}$

op = 6bit rs = 5bit rt = 5bit rd = 5bit shamt = 5bit funct = 6bit

→ 2.17:

Ans: lw $v0, 4($v1)
op = 0x23, rs = 1, rt = 2, const = 0x4

32 16 8 4 2 1
$\underset{0x23}{\underline{100011}}$ $\underset{1}{\underline{00001}}$ $\underset{2}{\underline{00010}}$ $\underset{0x4}{\underline{0000\ 0000\ 0000\ 0100}}$

→2.18.1:

Ans: In R type instructions, opcode would be 8 bits, rs, rs, rd, fields would be 7 bit each.

→2.18.2:

Ans: In the I type instructions, opcode would be 8 bits, rs, rt would be 7 bits each.

→2.19.1:

Ans: sll $t2, $t0, 44

The instructions performs the logical left shift on $t0. but shift 44 not ~~neg~~ ligible because we can $\overset{\text{use}}{\text{only}}$ 32 registers in this case shift 30 and 14. The value of $t2

$t2 = 0xAAAAAAAB

or

$t2, $t2, $t1

Now the value of $t2 and $t2 perform the logical operation OR and store in $t2

AAAAAAAO OR 12345678

$t2= 0xBABEFEF8 .

→ 2.19.2:

Ans: srl $t2, $t0, 4

The ins srl will make the logical right shift on $t0. Then the value of $t2

$t2 = 0x AAAAAAA0

andi $t2, $t2, -1

The value of $t2 will perform the logical operations AND with immediate -1

$t2 = AAAAAAA0

→ 2.19.3:

Ans: Srl $t2, $t0, 3

The ins srl will make the logical right shift on $t0. Then the value of $t2

$t2 = 0x15555555

andi $t2, $t2, 0xFFEF

Now the value of $t2 will perform the logical AND with 0xFFEF ; 0x15555555 AND 0xFFEF

$t2 = 00005545

→ 2.25.1:

Ans: i type.

→ 2.25.2:

Ans:
```
addi $t2, $t2, -1
beq $t2, $0, loop
```

→ 2.26.1:

Ans: 20

→ 2.26.2:

Ans:
```
i = 10;
do {
    B += 2;
    i = i - 1;
} while (i > 0)
```

→ 2.26.3:

Ans: 5*N

→ 2.20:

Ans:

```
srl  $t0, $t0, 11
srl  $t0, $t0, 26
ori  $t2, $0, 0x03ff
sll  $t2, $t2, 16
ori  $t2, $t2, 0xffff
and  $t1, $t1, $t2
or   $t1, $t1, $t0
```

→ 2.21:

Ans:

```
nor $t1, $t2, $t2
```

→ 2.22:

Ans:

```
lw  $t3, 0($s1)
sll $t1, $t3, 4
```

→ 2.23:

Ans: $t2=3

→ 2.24:

Ans:

Jump: no, beq: no

→ 2.29:

Ans:

```
for(i=0; i< 100; i++) {
        result += MemArray [s0];
        s0 = s0+4;
    }
```

→ 2.30:

Ans:

```
        addi $t1, $s0, 400
Loop:   lw   $s1, 0($t1)
        add  $s2, $s2, $s1
        addi $t1, $t1, -4
        bne  $t1, $s0, Loop.
```

→ 2.32:

Ans: Due to the recursive nature of the code, it is not possible for the compiler to in line the function call.

→ 2.27 :

```
        addi  $t0, $0, 0
        beq   $0, $0, TEST1
LOOP1:  addi  $t1, $0, 0
        beq   $0, $0, TEST2
LOOP2:  add   $t3, $t0, $t1
        sll   $t2, $t1, 4
        add   $t2, $t2, $s2
        sw    $t3, ($t2)
        addi  $t1, $t1, 1
TEST2:  slt   $t2, $t1, $s1
        bne   $t2, $0, LOOP2
        addi  $t0, $t0, 1
TEST1:  slt   $t2, $t0, $s0
        bne   $t2, $0, LOOP1
```

→ 2.28 :

Ans: 14 instructions to implement and 158 instructions executed

→ 2.35:

Ans: We can use the tail-call optimization for the second call to func. but then we must restore $ra, $s0, $s1 and $sp before that call. We save only one instructions (JR, $ra)

→ 2.38:

Ans: 0x00000011

→ 2.39:

Ans: Generally all solutions are similar:

lui $t1, top-16-bits:
ori $t1, $t1, bottom 16-bits.

→ 2.40:

Ans: No jump can go upto 0x0FFFFFFC

→ 2.41:

Ans: NO, range is 0x604 + 0x1FFFC = 0x0002 0600 to 0x604 − 0x20000 = 0xFFFE 0604.

→ 2.42:

Ans: Yes range is 0x1FFFF004 + 0x1FFFC

= 0x2001F000 to 0x1FFFF004

- 0x20000 = 1FFDF004.

→ 2.45:

Ans: It is possible for one or both processors to complete this code without ever reaching the SC instructions. If only one executes SC, it completes successfully. If both reach SC, they do so in the same cycle, but one SC completes first and then the other detects this and fails.

→ 2.46.2:

Ans: 107.04%, 113.43%

→ 2.47.1:

Ans: 2.6

→ 2.47.2:

Ans: 0.88

→ 2.47.3:

Ans: 0.533333333.