2.1 For C statement, what is the corresponding MIP assembly code. Var f, g, h, and i 32 bit int (use min assembly code) $\underset{s0}{f} = \underset{s1}{g} + (\underset{s2}{h} - 5)$

```
Sub $t0, $s2, 5
add $s0, $s1, $t0
```

2.2 For following MIPS assembly instruct what is corresponding C statement

```
f = g + h
f = i + f
```

2.3 Following C statement find corresponding assembly code, var $\underset{s0}{f}, \underset{s1}{g}, \underset{s2}{h}, \underset{s3}{i}, \underset{s4}{j}$
Assume base address Array $\underset{s6}{A}$ and $\underset{s7}{B}$

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

```
Sub $t0, $s3, $s4
SLL $t0, $t0, 2      Shift logical left by two bits 2^2 = 4 byte offset   word = 4 byte
add $t0, $s6, $t0    add result of offset to base address A

LW $t1, 0($t0)    load value from A offset is 0 first element
SW $t1, 32($s7)   Store word B var assuming 8.4 = 32   4 byte = word
```

2.4 For mips assembly instruct what is corresponding C statement

```
SLL $t0, $s0, 2      # t0 = f.4
add $t0, $s6, $t0    # t0 = &A[f]
SLL $t1, $s1, 2      # t1 = g.4
add $t1, $s7, $t1    # t1 = &B[g]
LW $s0, 0($t0)       # f = Af
addi $t2, $t0, 4     # ~~add immediate~~, all 4 bytes increment for any element
LW $t0, 0($t2)       # A[f+1]
add $t0, $t0, $s0    # f = A[f+1] + f
SW $t0, 0($t1)       # B[g] = A[f+1] + f
```

2.5 MIPS rewrite instruction in exercise 2.4, minimize

```
SLL $t0, $s0, 2
add $t0, $s6, $t0
SLL $t1, $s1, 2
add $t1, $s7, $t1
LW $s0, 0($t0)   # A[f]
LW $s0, 4($t0)   # A[f+1]
add $t0, $s0, $t0  # A[f+1] + A[f] = $t0
SW $t0, 0($t1)   # B[g] = A[f+1] + A[f]
```

2.6 table 32-bits

Add	Data
24	2
38	4
32	3
36	6
40	1

2.6.1 write C code sort data from low to high, lowest val in smallest mem loc

```
 Bubble sort
int array[5] = {2, 4, 3, 6, 1};
int temp;
for(int i=0 ; i<4 ; i++)
{
   for(int j = 0; i < 4 - i ; j++)
   {
     if(array[j] > array[j+1])
     {
       temp = array[j]   // save large value on temp
       array[j] = array[j+1]   // shift small num back one element
       array[j+1] = temp     // set next element as temp which is larger val
     }
   }
}
```

2.6.2 make MIPs code Assume base address array stored reg $s6

```
 .data
 Array:  .word  2,4,3,6,1

 .text
 main:
     la $s6 , Array    # load address of array into $s6
     li $t0 , 0        # load immediate for first loop i=0
 OuterLoop:
    bge $t0, 4, endouterloop  # branch greater or equal if true jump to endouterloop  i ≥ 4
    li  $t1 , 0     # load immediate second loop j=0
InnerLoop:
   li  $t2 , 4    # $t2 = 4
   Sub  $t2 , $t2,$t0   #  ~~$t2~~: 4-i
   bge $t1 , $t2 , endinnerloop   #  $t1 = j $t2 = 4 → j ≥ 4-i
   SLL  $t3, $t1, 2    # byte offset  4 bytes → 2^2
   add $t3, $t3, $s6  #  $t3 → j·4
   Lw  $t4, 0($t3)  # load  $t4 = array[j]
   addi  $t5, $t3, 4    #  $t5 = array[j+1]
   Lw $t6, 0($t5)  # load array[j+1]
   BLE $t4, $t6, noswap   #  if array[j] < array [j+1] go to iterate j loop
   sw  $t6, 0($t3)  # array[j+1] gets array[j] val
   sw $t4, 0($t5)  # array[j]  takes the greater val from array[j+1]
Noswap:
  addi  $t1, $t1, 1  # increment  # j++
  j   innerloop   # Jump inner
endinnerloop
  addi $t0,$t0,1  # i++
  j outterloop   # jump to out
Endouterloop
  li $v0, 10   # exit program
  syscall
```

2.7 Show how 0xabcdef12 would be arranged in memory of little endian, and big endian machine
assume address stored starting at 0

```
               0       1       2       3
little endian: 0x12 | 0xef | 0xcd | 0xab
big endian:    0xab | 0xcd | 0xef | 0x12
```

2.9 Translate C code to mips assume f, g, h, i, j ($s0 $s1 $s2 $s3 $s4), array A and B in regi ($s6 $s7)
AB are 4-byte word.

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

```
SLL $t0, $s3, 2       # load offset 2^2 = 4 bytes for i
add $t0, $t0, $s6     # add $t0 = A[i]
Lw $t0, 0($t0)        # load val from $t0 to $t0
SLL $t1, $s4, 2
add $t1, $t1, $s6
sw $t1, 0($t1)
add $t2, $t0, $t1     # A[i] + A[j] store in $t2
sw $t2, 32($s7)       # store B[8] = A[i]+A[j] into $t2     8·4 = 32 bytes
```

2.10 Translate following MIPS code to C, f, g, h, i, j ($s0 $s1 $s2 $s3 $s4), Base address array A, B ($s6 $s7)

```
addi $t0, $s6, 4      # add immediate 4 bytes A[1] = $t0
add $t1, $s6, $0      # $t1 = A[0] adding zero to base regi
sw $t1, 0($t0)        # $t1 = A[1] store → A[1] = A[0]
Lw $t0, 0($t0)        # Load $t0 = A[1]
add $s0, $t1, $t0     # f = A[0] + A[1]
```

2.12 assume regi $s0 and $s1 holds 0x80000000 and 0xD0000060 respectively
2.12.1 what is value of $t0 for following assembly code
add $t0, $s0, $s1

```
overflow  1000 0000 0000 0000 0000 0000 0000 0000
  ↓       1101 0000 0000 0000 0000 0000 0000 0006
      1   0101 0000 0000 0000 0000 0000 0000 0000
           5    0    0    0    0    0    0    0
```

$\boxed{\$t0 = 0x50000000}$

2.12.2 Has $t0 desired or overflow
Overflowed in MIPS instructions are 32-bit regi, but here is 33 bit.

2.12.3 now try to sub
sub $t0, $s0, $s1

```
      ones comp
$s1  0010 1111 1111 1111 1111 1111 1111 1111 } 2's
                                           +1
$s1  0011 0000 0000 0000 0000 0000 0000 0000
$s0  1000 0000 0000 . . . . . . ,
     1011 0000 0000 0000 0000 0000 0000 0000
```

$\boxed{\$t0 = 0xB000\ 0000}$

2.12.4 Is $t0 desired or overflow
$\boxed{\text{No overflow desired.}}$

2.12.5 Perform MIPs operation

add $to, $s0, $s1 ← 0x5000000
add $to, $to, $ S0

```
     carry
$s0 ↓ 1000 0000  0000  0000  0000  0000  0000  0000
$t0  1 0101 0000  0000  0000  0000  0000  0000  0000
      1101  ↓     ↓     ↓     ↓     ↓     ↓     ↓
        D   0     0     0     0     0     0     0
```

$\boxed{\$to = 0x0000000}$

2.12.6 Desired or overflowed?
overflow occured

2.18 expand MIPs regi file to 128 regi and expand instruction set to contain four times as many instruction

32 bit → 128 bit x4

2.18.1 How does this affect size of each bit fields in R-type
$2^7 = 128$ for Rs, RT, RD
opcode increases to 8-bit and registers increment to 7 bits

2.18.2 I type instruction
opcode increase by 2 bits and registers rs, rt increase by two as well.

2.18.3 How would the two proposed changes decrease the size of MIPs assembly prog
more register means more bits per instruction and increase code size, Less regi spills

2.21 Provide minimal set of MIPs that may be used to implement the following
~~not $t1, $t2~~ // bitwise invert
$\boxed{\text{nor } \$s1, \$s2, \$s2}$

2.22 Following C statement, write minimal sequence of MIPs assembly instruction that does identical operation. $t1 = A, $t2 = B, and $s1

A = C[0] << 4;
Lw $t3, 0($s1)
SLL $t1, $t3, 4

$t1=10$

2.26 following MIPs loop $\quad 0<t1 \rightarrow \$t2=1$

```
LOOP: slt  $t2, $0, $t1
      beq  $t2, $0, DONE
      subi $t1, $t1, 1
      addi $s2, $s2, 2
      j    LOOP
DONE:
```

2.26.1 Assume regi $t1 initialized to val 10, what val in regi $s2 assuming initially it was zero

skips second line

$2 \times 10 = \boxed{20}$

decrement 10x adding immediate

2.26.2 write equivalent C code routine regi $\overset{A}{\$s1}$, $\overset{B}{\$s2}$, $\overset{I}{\$t1}$, $\overset{temp}{\$t2}$

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

2.26.3 $\boxed{5 \cdot N}$

2.29

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

2.38 $\boxed{0\times00000011}$

2.47

2.47.1 2.6

2.47.2 0.88

2.47.3 0.533