HW2 : 10、11、13、19、20、25、26、27、31、43、46

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];
```

2.10 [5] <§§2.2, 2.3> Translate the following MIPS code 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.

```
addi $t0, $s6, 4
add  $t1, $s6, $0
sw   $t1, 0($t0)
lw   $t0, 0($t0)
add  $s0, $t1, $t0
```

2.11 [5] <§§2.2, 2.5> For each MIPS instruction, show the value of the opcode (OP), source register (RS), and target register (RT) fields. For the I-type instructions, show the value of the immediate field, and for the R-type instructions, show the value of the destination register (RD) field.

```
add $t0, $t0, $s0
```

2.12.6 [5] <§2.4> Is the result in $t0 the desired result, or has there been overflow?

2.13 Assume that $s0 holds the value 128_{ten}.

2.13.1 [5] <§2.4> For the instruction add $t0, $s0, $s1, what is the range(s) of values for $s1 that would result in overflow?

2.13.2 [5] <§2.4> For the instruction sub $t0, $s0, $s1, what is the range(s) of values for $s1 that would result in overflow?

2.13.3 [5] <§2.4> For the instruction sub $t0, $s1, $s0, what is the range(s) of values for $s1 that would result in overflow?

2.14 [5] <§§2.2, 2.5> Provide the type and assembly language instruction for the following binary value: $0000\ 0010\ 0001\ 0000\ 1000\ 0000\ 0010\ 0000_{two}$

assembly program?

2.19 Assume the following register contents:

```
$t0 = 0xAAAAAAAA, $t1 = 0x12345678
```

2.19.1 [5] <§2.6> For the register values shown above, what is the value of `$t2` for the following sequence of instructions?

```
sll $t2, $t0, 44
or  $t2, $t2, $t1
```

2.19.2 [5] <§2.6> For the register values shown above, what is the value of `$t2` for the following sequence of instructions?

```
sll  $t2, $t0, 4
andi $t2, $t2, -1
```

2.19.3 [5] <§2.6> For the register values shown above, what is the value of `$t2` for the following sequence of instructions?

```
srl  $t2, $t0, 3
andi $t2, $t2, 0xFFEF
```


2.20 [5] <§2.6> Find the shortest sequence of MIPS instructions that extracts bits 16 down to 11 from register `$t0` and uses the value of this field to replace bits 31 down to 26 in register `$t1` without changing the other 26 bits of register `$t1`.

2.21 [5] <§2.6> Provide a minimal set of MIPS instructions that may be used to implement the following pseudoinstruction:

```
not $t1, $t2        // bit-wise invert
```

2.22 [5] <§2.6> For the following C statement, write a minimal sequence of MIPS

```
        j    DONE
ELSE:   addi $t2, $t2, 2
DONE:
```

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?

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?

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

2.26 Consider the following MIPS loop:

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

2.26.1 [5] <§2.7> Assume that the register $t1 is initialized to the value 10. What is the value in register $s2 assuming $s2 is initially zero?

2.26.2 [5] <§2.7> For each of the loops above, write the equivalent C code routine. Assume that the registers $s1, $s2, $t1, and $t2 are integers A, B, i, and temp, respectively.

2.26.3 [5] <§2.7> For the loops written in MIPS assembly above, assume that the register $t1 is initialized to the value N. How many MIPS instructions are executed?

2.27 [5] <§2.7> Translate the following C code to MIPS assembly code. Use a

2.26.3 [5] <§2.7> For the loops written in MIPS assembly above, assume that the register `$t1` is initialized to the value N. How many MIPS instructions are executed?

2.27 [5] <§2.7> Translate the following C code to MIPS assembly code. Use a minimum number of instructions. Assume that the values of `a`, `b`, `i`, and `j` are in registers `$s0`, `$s1`, `$t0`, and `$t1`, respectively. Also, assume that register `$s2` 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;
```

2.28 [5] <§2.7> How many MIPS instructions does it take to implement the C code from Exercise 2.27? 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?

2.30 [5] <§2.7> Rewrite the loop from Exercise 2.29 to reduce the number of MIPS instructions executed.

2.31 [5] <§2.8> Implement the following C code in MIPS assembly. What is the total number of MIPS instructions needed to execute the function?

```
int fib(int n){
    if (n==0)
       return 0;
    else if (n == 1)
       return 1;
    else
       return fib(n-1) + fib(n-2);
```

2.32 [5] <§2.8> Functions can often be implemented by compilers "in-line." An in-line function is when the body of the function is copied into the program space, allowing the overhead of the function call to be eliminated. Implement an "in-line" version of the C code above in MIPS assembly. What is the reduction in the total number of MIPS assembly instructions needed to complete the f[illegible]n? As[illegible] that the C variable n is initialized to 5.

a single branch instruction to get to the PC address as shown in Exercise 2.39?

2.42 [5] <§§2.6, 2.9> If the current value of the PC is `0x1FFFf000`, can you use a single branch instruction to get to the PC address as shown in Exercise 2.39?

2.43 [5] <§2.10> Write the MIPS assembly code to implement the following C code:

```
lock(lk);
shvar=max(shvar,x);
unlock(lk);
```

Assume that the address of the `lk` variable is in `$a0`, the address of the shvar variable is in `$a1`, and the value of variable `x` is in `$a2`. Your critical section should not contain any function calls. Use `ll/sc` instructions to implement the `lock()` operation, and the `unlock()` operation is simply an ordinary store instruction.

2.44 [5] <§2.10> Repeat Exercise 2.43, but this time use `ll/sc` to perform an atomic update of the `shvar` variable directly, without using `lock()` and `unlock()`. Note that in this problem there is no variable `lk`.

2.45 [5] <§2.10> Using your code from Exercise 2.43 as an example, explain what happens when two processors begin to execute this critical section at the same time, assuming that each processor executes exactly one instruction per cycle.

2.46 Assume for a given processor the CPI of arithmetic instructions is 1, the CPI of load/store instructions is 10, and the CPI of branch instructions is 3. Assume a program has the following instruction breakdowns: 500 million arithmetic instructions, 300 million load/store instructions, 100 million branch instructions.

2.46.1 [5] <§2.17> Suppose that new, more powerful arithmetic instructions are added to the instruction set. On average, through the use of these more powerful arithmetic instructions, we can reduce the number of arithmetic instructions needed to execute a program by 25%, and the cost of increasing the clock cycle time by only 10%. Is this a good design choice? Why?

2.46.2 [5] <§2.17> Suppose that we find a way to double the performance of arithmetic instructions. What is the overall speedup of our machine? What if we find a way to improve the performance of arithmetic instructions by 10 times?

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