Instruction class	MIPS examples	HLL correspondence	Frequency	
			Integer	Ft. pt.
Arithmetic	`add, sub, addi`	Operations in assignment statements	16%	48%
Data transfer	`lw, sw, lb, lbu, lh, lhu, sb, lui`	References to data structures, such as arrays	35%	36%
Logical	`and, or, nor, andi, ori, sll, srl`	Operations in assignment statements	12%	4%
Conditional branch	`beq, bne, slt, slti, sltiu`	*If* statements and loops	34%	8%
Jump	`j, jr, jal`	Procedure calls, returns, and *case/switch* statements	2%	0%

FIGURE 2.45 MIPS instruction classes, examples, correspondence to high-level program language constructs, and percentage of MIPS instructions executed by category for the average SPEC CPU2006 benchmarks. Figure 3.26 in Chapter 3 shows average percentage of the individual MIPS instructions executed.

2.20 Historical Perspective and Further Reading

This section surveys the history of instruction set architectures (ISAs) over time, and we give a short history of programming languages and compilers. ISAs include accumulator architectures, general-purpose register architectures, stack architectures, and a brief history of ARM and the x86. We also review the controversial subjects of high-level-language computer architectures and reduced instruction set computer architectures. The history of programming languages includes Fortran, Lisp, Algol, C, Cobol, Pascal, Simula, Smalltalk, C++, and Java, and the history of compilers includes the key milestones and the pioneers who achieved them. The rest of this section is on the CD.

2.21 Exercises

Contributed by John Oliver of Cal Poly, San Luis Obispo, with contributions from Nicole Kaiyan (University of Adelaide) and Milos Prvulovic (Georgia Tech)

Appendix B describes the MIPS simulator, which is helpful for these exercises. Although the simulator accepts pseudoinstructions, try not to use pseudoinstructions for any exercises that ask you to produce MIPS code. Your goal should be to learn the real MIPS instruction set, and if you are asked to count instructions, your count should reflect the actual instructions that will be executed and not the pseudoinstructions.

There are some cases where pseudoinstructions must be used (for example, the `la` instruction when an actual value is not known at assembly time). In many cases,

they are quite convenient and result in more readable code (for example, the `li` and `move` instructions). If you choose to use pseudoinstructions for these reasons, please add a sentence or two to your solution stating which pseudoinstructions you have used and why.

Exercise 2.1

The following problems explore translating from C to MIPS. Assume that the variables `f`, `g`, `h`, and `i` are given and could be considered 32-bit integers as declared in a C program.

a.	`f = g - h;`
b.	`f = g + (h - 5);`

2.1.1 [5] <2.2> For the C statements above, what is the corresponding MIPS assembly code? Use a minimal number of MIPS assembly instructions.

2.1.2 [5] <2.2> For the C statements above, how many MIPS assembly instructions are needed to perform the C statement?

2.1.3 [5] <2.2> If the variables `f`, `g`, `h`, and `i` have values 1, 2, 3, and 4, respectively, what is the end value of `f`?

The following problems deal with translating from MIPS to C. Assume that the variables `g`, `h`, `i`, and `j` are given and could be considered 32-bit integers as declared in a C program.

a.	`addi f, f, 4`
b.	`add  f, g, h` `add  f, i, f`

2.1.4 [5] <2.2> For the MIPS assembly instructions above, what is a corresponding C statement?

2.1.5 [5] <2.2> If the variables `f`, `g`, `h`, and `i` have values 1, 2, 3, and 4, respectively, what is the end value of `f`?

Exercise 2.2

The following problems deal with translating from C to MIPS. Assume that the variables `g`, `h`, `i`, and `j` are given and could be considered 32-bit integers as declared in a C program.

a.	`f = g - f;`
b.	`f = i + (h - 2);`

2.2.1 [5] <2.2> For the C statements above, what is the corresponding MIPS assembly code? Use a minimal number of MIPS assembly instructions.

2.2.2 [5] <2.2> For the C statements above, how many MIPS assembly instructions are needed to perform the C statement?

2.2.3 [5] <2.2> If the variables `f`, `g`, `h`, and `i` have values 1, 2, 3, and 4, respectively, what is the end value of `f`?

The following problems deal with translating from MIPS to C. For the following exercise, assume that the variables `g`, `h`, `i`, and `j` are given and could be considered 32-bit integers as declared in a C program.

a.	`addi f, f, 4`
b.	`add  f, g, h` `sub  f, i, f`

2.2.4 [5] <2.2> For the MIPS assembly instructions above, what is a corresponding C statement?

2.2.5 [5] <2.2> If the variables `f`, `g`, `h`, and `i` have values 1, 2, 3, and 4, respectively, what is the end value of `f`?

Exercise 2.3

The following problems explore translating from C to MIPS. Assume that the variables `f` and `g` are given and could be considered 32-bit integers as declared in a C program.

a.	`f = -g - f;`
b.	`f = g + (-f - 5);`

2.3.1 [5] <2.2> For the C statements above, what is the corresponding MIPS assembly code? Use a minimal number of MIPS assembly instructions.

2.3.2 [5] <2.2> For the C statements above, how many MIPS assembly instructions are needed to perform the C statement?

2.3.3 [5] <2.2> If the variables `f`, `g`, `h`, `i`, and `j` have values 1, 2, 3, 4, and 5, respectively, what is the end value of `f`?

The following problems deal with translating from MIPS to C. Assume that the variables g, h, i, and j are given and could be considered 32-bit integers as declared in a C program.

a.	`addi f, f, -4`
b.	`add  i, g, h` `add  f, i, f`

2.3.4 [5] <2.2> For the MIPS statements above, what is a corresponding C statement?

2.3.5 [5] <2.2> If the variables f, g, h, and i have values 1, 2, 3, and 4, respectively, what is the end value of f?

Exercise 2.4

The following problems deal with translating from C 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.

a.	`f = -g - A[4];`
b.	`B[8] = A[i-j];`

2.4.1 [10] <2.2, 2.3> For the C statements above, what is the corresponding MIPS assembly code?

2.4.2 [5] <2.2, 2.3> For the C statements above, how many MIPS assembly instructions are needed to perform the C statement?

2.4.3 [5] <2.2, 2.3> For the C statements above, how many different registers are needed to carry out the C statement?

The following problems deal with translating from MIPS 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.

a.	`sll  $s2, $s4,  1`
	`add  $s0, $s2,  $s3`
	`add  $s0, $s0,  $s1`

b.	`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 = A[f]`
	`addi $t2, $t0,  4`
	`lw   $t0, 0($t2)`
	`add  $t0, $t0,  $s0`
	`sw   $t0, 0($t1)`

2.4.4 [10] <2.2, 2.3> For the MIPS assembly instructions above, what is the corresponding C statement?

2.4.5 [5] <2.2, 2.3> For the MIPS assembly instructions above, rewrite the assembly code to minimize the number if MIPS instructions (if possible) needed to carry out the same function.

2.4.6 [5] <2.2, 2.3> How many registers are needed to carry out the MIPS assembly as written above? If you could rewrite the code above, what is the minimal number of registers needed?

Exercise 2.5

In the following problems, we will be investigating memory operations in the context of an MIPS processor. The table below shows the values of an array stored in memory. Assume the base address of the array is stored in register `$s6` and offset it with respect to the base address of the array.

a.

Address	Data
20	4
24	5
28	3
32	2
34	1

b.

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

2.5.1 [10] <2.2, 2.3> For the memory locations in the table above, write C code to sort the data from lowest to highest, placing the lowest value in the smallest memory location shown in the figure. Assume that the data shown represents the C variable called `Array`, which is an array of type `int`, and that the first number in the array shown is the first element in the array. Assume that this particular machine is a byte-addressable machine and a word consists of four bytes.

2.5.2 [10] <2.2, 2.3> For the memory locations in the table above, write MIPS code to sort the data from lowest to highest, placing the lowest value in the smallest memory location. Use a minimum number of MIPS instructions. Assume the base address of `Array` is stored in register `$s6`.

2.5.3 [5] <2.2, 2.3> To sort the array above, how many instructions are required for the MIPS code? If you are not allowed to use the immediate field in `lw` and `sw` instructions, how many MIPS instructions do you need?

The following problems explore the translation of hexadecimal numbers to other number formats.

a.	`0xabcdef12`
b.	`0x10203040`

2.5.4 [5] <2.3> Translate the hexadecimal numbers above into decimal.

2.5.5 [5] <2.3> Show how the data in the table would be arranged in memory of a little-endian and a big-endian machine. Assume the data is stored starting at address 0.

Exercise 2.6

The following problems deal with translating from C 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:

a.	`f = f + A[2];`
b.	`B[8] = A[i] + A[j];`

2.6.1 [10] <2.2, 2.3> For the C statements above, what is the corresponding MIPS assembly code?

2.6.2 [5] <2.2, 2.3> For the C statements above, how many MIPS assembly instructions are needed to perform the C statement?

2.6.3 [5] <2.2, 2.3> For the C statements above, how many registers are needed to carry out the C statement using MIPS assembly code?

The following problems deal with translating from MIPS 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.

a.	`sub $s0, $s0, $s1`
	`sub $s0, $s0, $s3`
	`add $s0, $s0, $s1`

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

2.6.4 [5] <2.2, 2.3> For the MIPS assembly instructions above, what is the corresponding C statement?

2.6.5 [5] <2.2, 2.3> For the MIPS assembly above, assume that the registers `$s0`, `$s1`, `$s2`, and `$s3` contain the values 0x0000000a, 0x00000014, 0x0000001e, and 0x00000028, respectively. Also, assume that register `$s6` contains the value 0x00000100, and that memory contains the following values:

Address	Value
0x00000100	0x00000064
0x00000104	0x000000c8
0x00000108	0x0000012c

Find the value of `$s0` at the end of the assembly code.

2.6.6 [10] <2.3, 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.

Exercise 2.7

The following problems explore number conversions from signed and unsigned binary numbers to decimal numbers.

a.	$0010\ 0100\ 1001\ 0010\ 0100\ 1001\ 0010\ 0100_{two}$
b.	$0101\ 1111\ 1011\ 1110\ 0100\ 0000\ 0000\ 0000_{two}$

2.7.1 [5] <2.4> For the patterns above, what base 10 number does the binary number represent, assuming that it is a two's complement integer?

2.7.2 [5] <2.4> For the patterns above, what base 10 number does the binary number represent, assuming that it is an unsigned integer?

2.7.3 [5] <2.4> For the patterns above, what hexadecimal number does it represent?

The following problems explore number conversions from decimal to signed and unsigned binary numbers.

a.	-1_{ten}
b.	1024_{ten}

2.7.4 [5] <2.4> For the base ten numbers above, convert to 2's complement binary.

2.7.5 [5] <2.4> For the base ten numbers above, convert to 2's complement hexadecimal.

2.7.6 [5] <2.4> For the base ten numbers above, convert the negated values from the table to 2's complement hexadecimal.

Exercise 2.8

The following problems deal with sign extension and overflow. Registers `$s0` and `$s1` hold the values as shown in the table below. You will be asked to perform an MIPS assembly language instruction on these registers and show the result.

a.	`$s0` = $0x80000000_{sixteen}$, `$s1` = $0xD0000000_{sixteen}$
b.	`$s0` = $0x00000001_{sixteen}$, `$s1` = $0xFFFFFFFF_{sixteen}$

2.8.1 [5] <2.4> For the contents of registers `$s0` and `$s1` as specified above, what is the value of `$t0` for the following assembly code?

```
add $t0, $s0, $s1
```

Is the result in `$t0` the desired result, or has there been overflow?

2.8.2 [5] <2.4> For the contents of registers `$s0` and `$s1` as specified above, what is the value of `$t0` for the following assembly code?

```
sub $t0, $s0, $s1
```

Is the result in `$t0` the desired result, or has there been overflow?

2.8.3 [5] <2.4> For the contents of registers `$s0` and `$s1` as specified above, what is the value of `$t0` for the following assembly code?

```
add $t0, $s0, $s1
add $t0, $t0, $s0
```

Is the result in `$t0` the desired result, or has there been overflow?

In the following problems, you will perform various MIPS operations on a pair of registers, `$s0` and `$s1`. Given the values of `$s0` and `$s1` in each of the questions below, state if there will be overflow.

a.	`add $s0, $s0, $s1` `add $s0, $s0, $s1`
b.	`add $s0, $s0, $s1` `add $s0, $s0, $s1` `add $s0, $s0, $s1`

2.8.4 [5] <2.4> Assume that register `$s0` = 0x70000000 and `$s1` = 0x10000000. For the table above, will there be overflow?

2.8.5 [5] <2.4> Assume that register `$s0` = 0x40000000 and `$s1` = 0x20000000. For the table above, will there be overflow?

2.8.6 [5] <2.4> Assume that register `$s0` = 0x8FFFFFFF and `$s1` = 0xD0000000. For the table above, will there be overflow?

Exercise 2.9

The table below contains various values for register `$s1`. You will be asked to evaluate if there would be overflow for a given operation.

a.	-1_{ten}
b.	1024_{ten}

2.9.1 [5] <2.4> Assume that register `$s0` = 0x70000000 and `$s1` has the value as given in the table. If the instruction: add `$s0`, `$s0`, `$s1` is executed, will there be overflow?

2.9.2 [5] <2.4> Assume that register `$s0` = 0x80000000 and `$s1` has the value as given in the table. If the instruction: sub `$s0`, `$s0`, `$s1` is executed, will there be overflow?

2.9.3 [5] <2.4> Assume that register `$s0` = 0x7FFFFFFF and `$s1` has the value as given in the table. If the instruction: sub `$s0`, `$s0`, `$s1` is executed, will there be overflow?

The table below contains various values for register `$s1`. You will be asked to evaluate if there would be overflow for a given operation.

a.	$0010\ 0100\ 1001\ 0010\ 0100\ 1001\ 0010\ 0100_{two}$
b.	$0101\ 1111\ 1011\ 1110\ 0100\ 0000\ 0000\ 0000_{two}$

2.9.4 [5] <2.4> Assume that register `$s0` = 0x70000000 and `$s1` has the value as given in the table. If the instruction: add `$s0`, `$s0`, `$s1` is executed, will there be overflow?

2.9.5 [5] <2.4> Assume that register `$s0` = 0x70000000 and `$s1` has the value as given in the table. If the instruction: add `$s0`, `$s0`, `$s1` is executed, what is the result in hex?

2.9.6 [5] <2.4> Assume that register `$s0` = 0x70000000 and `$s1` has the value as given in the table. If the instruction: add `$s0`, `$s0`, `$s1` is executed, what is the result in base ten?

Exercise 2.10

In the following problems, the data table contains bits that represent the opcode of an instruction. You will be asked to interpret the bits as MIPS instructions into assembly code and determine what format of MIPS instruction the bits represent.

a.	$0000\ 0010\ 0001\ 0000\ 1000\ 0000\ 0010\ 0000_{two}$
b.	$0000\ 0001\ 0100\ 1011\ 0100\ 1000\ 0010\ 0010_{two}$

2.10.1 [5] <2.5> For the binary entries above, what instruction do they represent?

2.10.2 [5] <2.5> What type (I-type, R-type, J-type) instruction do the binary entries above represent?

2.10.3 [5] <2.4, 2.5> If the binary entries above were data bits, what number would they represent in hexadecimal?

In the following problems, the data table contains MIPS instructions. You will be asked to translate the entries into the bits of the opcode and determine the MIPS instruction format.

a.	`addi $t0, $t0, 0`
b.	`sw $t1, 32($t2)`

2.10.4 [5] <2.4, 2.5> For the instructions above, show the binary then hexadecimal representation of these instructions.

2.10.5 [5] <2.5> What type (I-type, R-type, J-type) instruction do the instructions above represent?

2.10.6 [5] <2.5> What is the binary then hexadecimal representation of the opcode, Rs, and Rt fields in this instruction? For R-type instructions, what is the hexadecimal representation of the Rd and funct fields? For I-type instructions, what is the hexadecimal representation of the immediate field?

Exercise 2.11

In the following problems, the data table contains bits that represent the opcode of an instruction. You will be asked to translate the entries into assembly code and determine what format of MIPS instruction the bits represent.

a.	0x01084020
b.	0x02538822

2.11.1 [5] <2.4, 2.5> What binary number does the above hexadecimal number represent?

2.11.2 [5] <2.4, 2.5> What decimal number does the above hexadecimal number represent?

2.11.3 [5] <2.5> What instruction does the above hexadecimal number represent?

In the following problems, the data table contains the values of various fields of MIPS instructions. You will be asked to determine what the instruction is, and find the MIPS format for the instruction.

a.	op=0, rs=3, rt=2, rd=3, shamt=0, funct=34
b.	op=0x23, rs=1, rt=2, const=0x4

2.11.4 [5] <2.5> What type (I-type, R-type) instruction do the instructions above represent?

2.11.5 [5] <2.5> What is the MIPS assembly instruction described above?

2.11.6 [5] <2.4, 2.5> What is the binary representation of the instructions above?

Exercise 2.12

In the following problems, the data table contains various modifications that could be made to the MIPS instruction set architecture. You will investigate the impact of these changes on the instruction format of the MIPS architecture.

a.	128 registers
b.	Four times as many different instructions

2.12.1 [5] <2.5> If the instruction set of the MIPS processor is modified, the instruction format must also be changed. For each of the suggested changes above, show the size of the bit fields of an R-type format instruction. What is the total number of bits needed for each instruction?

2.12.2 [5] <2.5> If the instruction set of the MIPS processor is modified, the instruction format must also be changed. For each of the suggested changes above, show the size of the bit fields of an I-type format instruction. What is the total number of bits needed for each instruction?

2.12.3 [5] <2.5, 2.10> Why could the suggested change in the table above decrease the size of an MIPS assembly program? Why could the suggested change in the table above increase the size of an MIPS assembly program?

In the following problems, the data table contains hexadecimal values. You will be asked to determine what MIPS instruction the value represents, and find the MIPS instruction format.

a.	`0x01090012`
b.	`0xAD090012`

2.12.4 [5] <2.5> For the entries above, what is the value of the number in decimal?

2.12.5 [5] <2.5> For the hexadecimal entries above, what instruction do they represent?

2.12.6 [5] <2.4, 2.5> What type (I-type, R-type, J-type) instruction do the binary entries above represent? What is the value of the op field and the rt field?

Exercise 2.13

In the following problems, the data table contains the values for registers `$t0` and `$t1`. You will be asked to perform several MIPS logical operations on these registers.

a.	`$t0 = 0xAAAAAAAA, $t1 = 0x12345678`
b.	`$t0 = 0xF00DD00D, $t1 = 0x11111111`

2.13.1 [5] <2.6> For the lines above, what is the value of `$t2` for the following sequence of instructions?

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

2.13.2 [5] <2.6> For the values in the table above, what is the value of `$t2` for the following sequence of instructions?

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

2.13.3 [5] <2.6> For the lines above, what is the value of `$t2` for the following sequence of instructions?

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

In the following exercise, the data table contains various MIPS logical operations. You will be asked to find the result of these operations given values for registers `$t0` and `$t1`.

a.	`sll  $t2, $t0, 1` `andi $t2, $t2, -1`
b.	`andi $t2, $t1, 0x00F0` `srl  $t2, 2`

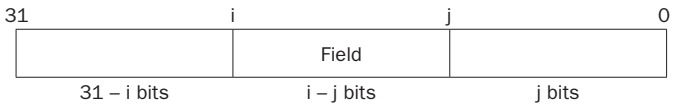
2.13.4 [5] <2.6> Assume that `$t0` = 0x0000A5A5 and `$t1` = 00005A5A. What is the value of `$t2` after the two instructions in the table?

2.13.5 [5] <2.6> Assume that `$t0` = 0xA5A50000 and `$t1` = A5A50000. What is the value of `$t2` after the two instructions in the table?

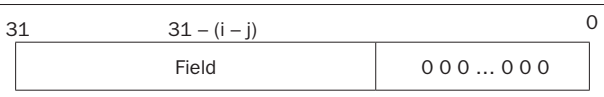
2.13.6 [5] <2.6> Assume that `$t0` = 0xA5A5FFFF and `$t1` = A5A5FFFF. What is the value of `$t2` after the two instructions in the table?

Exercise 2.14

The following figure shows the placement of a bit field in register `$t0`.



In the following problems, you will be asked to write MIPS instructions to extract the bits "Field" from register `$t0` and place them into register `$t1` at the location indicated in the following table.

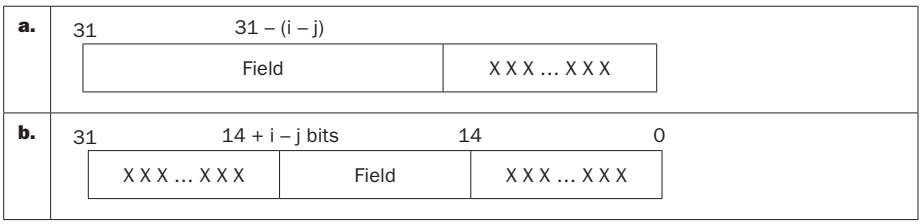
a.	
b.	

2.14.1 [20] <2.6> Find the shortest sequence of MIPS instructions that extracts a field from `$t0` for the constant values i = 22 and j = 5 and places the field into `$t1` in the format shown in the data table.

2.14.2 [5] <2.6> Find the shortest sequence of MIPS instructions that extracts a field from `$t0` for the constant values i = 4 and j = 0 and places the field into `$t1` in the format shown in the data table.

2.14.3 [5] <2.6> Find the shortest sequence of MIPS instructions that extracts a field from `$t0` for the constant values i = 31 and j = 28 and places the field into `$t1` in the format shown in the data table.

In the following problems, you will be asked to write MIPS instructions to extract the bits "Field" from register `$t0` shown in the figure and place them into register `$t1` at the location indicated in the following table. The bits shown as "`XXX`" are to remain unchanged.



2.14.4 [20] <2.6> Find the shortest sequence of MIPS instructions that extracts a field from `$t0` for the constant values i = 17 and j = 11 and places the field into `$t1` in the format shown in the data table.

2.14.5 [5] <2.6> Find the shortest sequence of MIPS instructions that extracts a field from `$t0` for the constant values i = 5 and j = 0 and places the field into `$t1` in the format shown in the data table.

2.14.6 [5] <2.6> Find the shortest sequence of MIPS instructions that extracts a field from `$t0` for the constant values i = 31 and j = 29 and places the field into `$t1` in the format shown in the data table.

Exercise 2.15

For these problems, the table holds some logical operations that are not included in the MIPS instruction set. How can these instructions be implemented?

a.	`not $t1, $t2 // bit-wise invert`
b.	`orn $t1, $t2, $t3 // bit-wise OR of $t2, !$t3`

2.15.1 [5] <2.6> The logical instructions above are not included in the MIPS instruction set, but are described above. If the value of `$t2` = 0x00FFA5A5 and the value of `$t3` = 0xFFFF003C, what is the result in `$t1`?

2.15.2 [10] <2.6> The logical instructions above are not included in the MIPS instruction set, but can be synthesized using one or more MIPS assembly instructions. Provide a minimal set of MIPS instructions that may be used in place of the instructions in the table above.

2.15.3 [5] <2.6> For your sequence of instructions in 2.15.2, show the bit-level representation of each instruction.

Various C-level logical statements are shown in the table below. In this exercise, you will be asked to evaluate the statements and implement these C statements using MIPS assembly instructions.

a.	`A = B | !A;`
b.	`A = C[0] << 4;`

2.15.4 [5] <2.6> The table above shows different C statements that use logical operators. If the memory location at C[0] contains the integer values 0x00001234, and the initial integer values of `A` and `B` are 0x00000000 and 0x00002222, what is the result value of `A`?

2.15.5 [5] <2.6> For the C statements in the table above, write a minimal sequence of MIPS assembly instructions that does the identical operation. Assume `$t1` = `A`, `$t2` = `B`, and `$s1` is the base address of `C`.

2.15.6 [5] <2.6> For your sequence of instructions in 2.15.5, show the bit-level representation of each instruction.

Exercise 2.16

For these problems, the table holds various binary values for register `$t0`. Given the value of `$t0`, you will be asked to evaluate the outcome of different branches.

a.	$0010\ 0100\ 1001\ 0010\ 0100\ 1001\ 0010\ 0100_{two}$
b.	$0101\ 1111\ 1011\ 1110\ 0100\ 0000\ 0000\ 0000_{two}$

2.16.1 [5] <2.7> Suppose that register `$t0` contains a value from above and `$t1` has the value

$$0011\ 1111\ 1111\ 1000\ 0000\ 0000\ 0000\ 0000_{two}$$

Note the result of executing these instructions on particular registers. What is the value of $t2 after the following instructions?

```
        slt  $t2, $t0, $t1
        beq  $t2, $0, ELSE
        j    DONE
ELSE: addi $t2, $0, 2
DONE:
```

2.16.2 [5] <2.7> Suppose that register $t0 contains a value from the table above and is compared against the value X, as used in the MIPS instruction below. Note the format of the slti instruction. For what values of X, if any, will $t2 be equal to 1?

```
slti $t2, $t0, X
```

2.16.3 [5] <2.7> Suppose the program counter (PC) is set to 0x0000 0020. Is it possible to use the jump MIPS assembly instruction to get set the PC to the address as shown in the data table above? Is it possible to use the branch-on-equal MIPS assembly instruction to get set the PC to the address as shown in the data table above?

For these problems, the table holds various binary values for register $t0. Given the value of $t0, you will be asked to evaluate the outcome of different branches.

a.	0x00101000
b.	0x80001000

2.16.4 [5] <2.7> Suppose that register $t0 contains a value from above. What is the value of $t2 after the following instructions?

```
        slt  $t2, $0,   $t0
        bne  $t2, $0,   ELSE
        j    DONE
ELSE: addi $t2, $t2,  2
DONE:
```

2.16.5 [5] <2.6, 2.7> Suppose that register $t0 contains a value from above. What is the value of $t2 after the following instructions?

```
sll $t0, $t0, 2
slt $t2, $t0, $0
```

2.16.6 [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 get set the PC to the

address as shown in the data table above? Is it possible to use the branch-on-equal (beq) MIPS assembly instruction to set the PC to the address as shown in the data table above? Note the format of the J-type instruction.

Exercise 2.17

For these problems, there are several instructions that are not included in the MIPS instruction set are shown.

a.	`subi $t2, $t3, 5  # R[rt] = R[rs] - SignExtImm`
b.	`rpt  $t2, loop  # if(R[rs]>0) R[rs]=R[rs]-1, PC=PC+4+BranchAddr`

2.17.1 [5] <2.7> The table above contains some instructions not included in the MIPS instruction set and the description of each instruction. Why are these instructions not included in the MIPS instruction set?

2.17.2 [5] <2.7> The table above contains some instructions not included in the MIPS instruction set and the description of each instruction. If these instructions were to be implemented in the MIPS instruction set, what is the most appropriate instruction format?

2.17.3 [5] <2.7> For each instruction in the table above, find the shortest sequence of MIPS instructions that performs the same operation.

For these problems, the table holds MIPS assembly code fragments. You will be asked to evaluate each of the code fragments, familiarizing you with the different MIPS branch instructions.

a.

```
LOOP:  addi $s2, $s2, 2
       subi $t1, $t1, 1
       bne  $t1, $0,  LOOP
DONE:
```

b.

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

2.17.4 [5] <2.7> For the loops written in MIPS assembly above, assume that the register `$t1` is initialized to the value 10. What is the value in register `$s2` assuming the `$s2` is initially zero?

2.17.5 [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.

2.17.6 [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?

Exercise 2.18

For these problems, the table holds some C code. You will be asked to evaluate these C code statements in MIPS assembly code.

a.	`for(i=0; i<a; i++)` `  a += b;`

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

2.18.1 [5] <2.7> For the table above, draw a control-flow graph of the C code.

2.18.2 [5] <2.7> For the table above, translate the 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`.

2.18.3 [5] <2.7> How many MIPS instructions does it take to implement the C code? 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?

For these problems, the table holds MIPS assembly code fragments. You will be asked to evaluate each of the code fragments, familiarizing you with the different MIPS branch instructions.

a.	`      addi $t1, $0, 50`
	`LOOP: lw   $s1, 0($s0)`
	`      add  $s2, $s2, $s1`
	`      lw   $s1, 4($s0)`
	`      add  $s2, $s2, $s1`
	`      addi $s0, $s0, 8`
	`      subi $t1, $t1, 1`
	`      bne  $t1, $0, LOOP`

b.	`      addi $t1, $0, $0`
	`LOOP: lw   $s1, 0($s0)`
	`      add  $s2, $s2, $s1`
	`      addi $s0, $s0, 4`
	`      addi $t1, $t1, 1`
	`      slti $t2, $t1, 100`
	`      bne  $t2, $s0, LOOP`

2.18.4 [5] <2.7> What is the total number of MIPS instructions executed?