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`

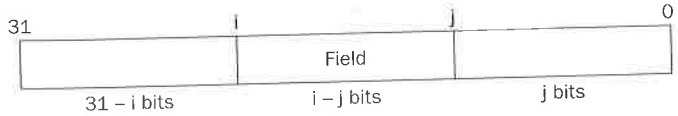
2.13.4 [5] <2.6> Assume that `$t0` = 0x0000A5A5 and `$t1` = 00005A5A. What is the value of `$t2` after the two instructions in the table?

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?

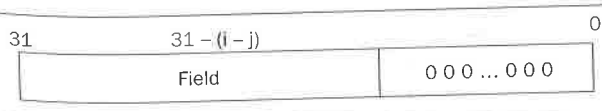
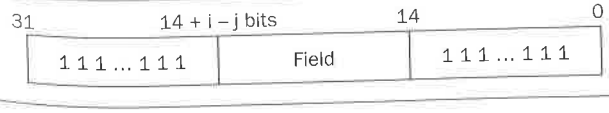
2.13.6 [5] <2.6> Assume that `$t0` = 0xA5A5FFFF and `$t1` = A5A5FFFF. What is the value of `$t2` after the two instructions in the table?

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.

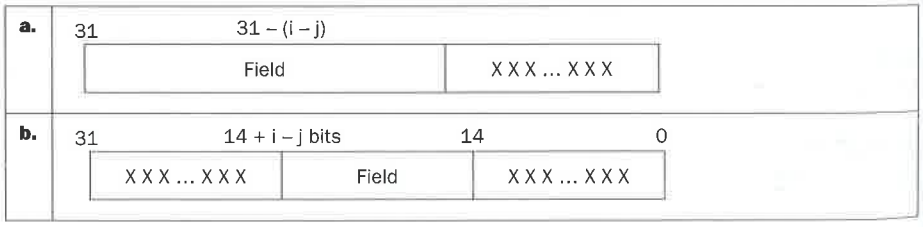
a.	
b.	

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
(beq) MIPS assembly ins
table above? Note the fo

Exercise 2.17

For these problems, ther
instruction set are show

a.	subi $t2, $t3, 5
b.	rpt $t2, loop

2.17.1 [5] <2.7> The
the MIPS instruction se
instructions not include

2.17.2 [5] <2.7> The t
MIPS instruction set an
were to be implemented
instruction format?

2.17.3 [5] <2.7> For
sequence of MIPS instru

For these problems, the
asked to evaluate each c
MIPS branch instructio

a.	LOOP: addi $s2, subi $t1, bne $t1, DONE:
b.	LOOP: slt $t2, beq $t2, subi $t1, addi $s2, j LOOP DONE:

2.17.4 [5] <2.7> For
register $t1 is initialize
the $s2 is initially zero?

2.17.5 [5] <2.7> For
tine. Assume that the r
temp, respectively.

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?

2.18.5 [5] <2.7> Tr
ger `i` is held in registe
holds the base addres

2.18.6 [5] <2.7> R
executed.

Exercise 2.19

For the following pro
function listed in the
routines into MIPS a

a.	`int fib(int n){`
	`  if (n==0)`
	`  return`
	`  else if (n`
	`  return`
	`  else`
	`  fib(n-1)`

b.	`int positive(i`
	`  if (addit(`
	`  return`
	`  else`
	`  return`
	`  }`
	`int addit(int`
	`  return a+b`
	`  }`

2.19.1 [15] <2.8>
the total number of

2.19.2 [5] <2.8>
in-line function is v
allowing the overhe
version of the the C
the total number of
Assume that the C

2.19.3 [5] <2.8>
function call is ma
and follow the regi

The following three
function `func`. The

2.18.5 [5] <2.7> Translate the loops above into C. Assume that the C-level integer `i` is held in register `$t1`, `$s2` holds the C-level integer called `result`, and `$s0` holds the base address of the integer `MemArray`.

2.18.6 [5] <2.7> Rewrite the loop to reduce the number of MIPS instructions executed.

Exercise 2.19

For the following problems, the table holds C code functions. Assume that the first function listed in the table is called first. You will be asked to translate these C code routines into MIPS assembly.

a.

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

b.

```
int positive(int a, int b) {
    if (addit(a, b) > 0)
       return 1;
    else
       return 0;
    }
int addit(int a, int b) {
    return a+b;
    }
```

2.19.1 [15] <2.8> Implement the C code in the table in MIPS assembly. What is the total number of MIPS instructions needed to execute the function?

2.19.2 [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 the C code in the table in MIPS assembly. What is the reduction in the total number of MIPS assembly instructions needed to complete the function? Assume that the C variable `n` is initialized to 5.

2.19.3 [5] <2.8> For each function call, show the contents of the stack after the function call is made. Assume the stack pointer is originally at address 0x7ffffffc, and follow the register conventions as specified in Figure 2.11.

The following three problems in this Exercise refer to a function `f` that calls another function `func`. The code for C function `func` is already compiled in another module

using the MIPS calling convention from Figure 2.14. The function declaration for `func` is "`int func(int a, int b);`". The code for function `f` is as follows:

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

b.
```
int f(int a, int b, int c, int d){
  if(a+b>c+d)
    return func(a+b,c+d);
  return func(c+d,a+b);
}
```

2.19.4 [10] <2.8> Translate function `f` into MIPS assembly language, also using the MIPS calling convention from Figure 2.14. If you need to use registers `$t0` through `$t7`, use the lower-numbered registers first.

2.19.5 [5] <2.8> Can we use the tail-call optimization in this function? If no, explain why not. If yes, what is the difference in the number of executed instructions in `f` with and without the optimization?

2.19.6 [5] <2.8> Right before your function `f` from Problem 2.19.4 returns, what do we know about contents of registers `$t5`, `$s3`, `$ra`, and `$sp`? Keep in mind that we know what the entire function `f` looks like, but for function `func` we only know its declaration.

Exercise 2.20

This exercise deals with recursive procedure calls. For the following problems, the table has an assembly code fragment that computes the factorial of a number. However, the entries in the table have errors, and you will be asked to fix these errors. For number n, factorial of n = 1 x 2 x 3 x .. .. x n.

a.
```
FACT:  sw    $ra, 4($sp)
       sw    $a0, 0($sp)
       addi  $sp, $sp, -8
       slti  $t0, $a0, 1
       beq   $t0, $0, L1
       addi  $v0, $0, 1
       addi  $sp, $sp, 8
       jr    $ra

L1:    addi  $a0, $a0, -1
       jal   FACT
       addi  $sp, $sp, 8
       lw    $a0, 0($sp)
       lw    $ra, 4($sp)
       mul   $v0, $a0, $v0
       jr    $ra
```

b.
```
FACT:  addi $sp, $sp
       sw   $ra, 4($
       sw   $a0, 0($
       add  $s0, $0,
       slti $t0, $a0
       beq  $t0, $0,
       mul  $v0, $s0
       addi $sp, $sp
       jr   $ra

L1:    addi $a0, $a0
       jal  FACT
       addi $v0, $0
       lw   $a0, 0($
       lw   $ra, 4($
       addi $sp, $sp
       jr   $ra
```

2.20.1 [5] <2.8> The MI
given input. The integer inp
in register `$v0`. In the asser

2.20.2 [10] <2.8> For th
the input is 4. Rewrite the
ner. Restrict your register
instructions used to execu
of the factorial program?

2.20.3 [5] <2.8> Show t
ing that the input is 4.

For the following problem
a Fibonacci number. How
asked to fix these errors. F

n	fibonacc
1	1
2	1
3	2
4	3
5	5
6	8
7	13
8	21

b.
```
FACT:  addi $sp, $sp, 8
       sw   $ra, 4($sp)
       sw   $a0, 0($sp)
       add  $s0, $0, $a0
       slti $t0, $a0, 2
       beq  $t0, $0, L1
       mul  $v0, $s0, $v0
       addi $sp, $sp, -8
       jr   $ra

L1:    addi $a0, $a0, -1
       jal  FACT
       addi $v0, $0, 1
       lw   $a0, 0($sp)
       lw   $ra, 4($sp)
       addi $sp, $sp, -8
       jr   $ra
```

2.20.1 [5] <2.8> The MIPS assembly program above computes the factorial of a given input. The integer input is passed through register `$a0`, and the result is returned in register `$v0`. In the assembly code, there are a few errors. Correct the MIPS errors.

2.20.2 [10] <2.8> For the recursive factorial MIPS program above, assume that the input is 4. Rewrite the factorial program to operate in a non-recursive manner. Restrict your register usage to registers `$s0-$s7`. What is the total number of instructions used to execute your solution from 2.20.2 versus the recursive version of the factorial program?

2.20.3 [5] <2.8> Show the contents of the stack after each function call, assuming that the input is 4.

For the following problems, the table has an assembly code fragment that computes a Fibonacci number. However, the entries in the table have errors, and you will be asked to fix these errors. For number n, the Fibonacci of n is calculated as follows:

n	fibonacci of n
1	1
2	1
3	2
4	3
5	5
6	8
7	13
8	21

a.

```
FIB:  addi  $sp, $sp, -12
      sw    $ra, 0($sp)
      sw    $s1, 4($sp)
      sw    $a0, 8($sp)
      slti  $t0, $a0, 1
      beq   $t0, $0, L1
      addi  $v0, $a0, $0
      j     EXIT

L1:   addi  $a0, $a0, -1
      jal   FIB
      addi  $s1, $v0, $0
      addi  $a0, $a0, -1
      jal   FIB
      add   $v0, $v0, $s1

EXIT: lw    $ra, 0($sp)
      lw    $a0, 8($sp)
      lw    $s1, 4($sp)
      addi  $sp, $sp, 12
      jr    $ra
```

b.

```
FIB:  addi  $sp, $sp, -12
      sw    $ra, 8($sp)
      sw    $s1, 4($sp)
      sw    $a0, 0($sp)
      slti  $t0, $a0, 3
      beq   $t0, $0, L1
      addi  $v0, $0, 1
      j     EXIT

L1:   addi  $a0, $a0, -1
      jal   FIB
      addi  $a0, $a0,  -2
      jal   FIB
      add   $v0, $v0, $s1

EXIT: lw    $a0, 0($sp)
      lw    $s1, 4($sp)
      lw    $ra, 8($sp)
      addi  $sp, $sp, 12
      jr    $ra
```

2.20.4 [5] <2.8> The MIPS assembly program above computes the Fibonacci of a given input. The integer input is passed through register `$a0`, and the result is returned in register `$v0`. In the assembly code, there are a few errors. Correct the MIPS errors.

2.20.5 [10] <2.8> For the recursive Fibonacci MIPS program above, assume that the input is 4. Rewrite the Fibonacci program to operate in a non-recursive manner. Restrict your register usage to registers `$s0-$s7`. What is the total number of

instructions used to exe
of the factorial progran

2.20.6 [5] <2.8> Sho
ing that the input is 4.

Exercise 2.21

Assume that the stack
global pointers start a
the calling conventio
passed using register
functions may only u

a.

```
int my_global =
main()
{
    int x = 10
    int y = 20
    int z;
    z = my_fun
}
int my_functio
{
    return x
}
```

b.

```
int my_global
main()
{
    int z;
    my_global
    z = leaf_
}
int leaf_func
{
    return x
}
```

2.21.1 [5] <2.8>

2.21.2 [5] <2.8>
each function call

2.21.3 [5] <2.8>
etc.), write the MI

instructions used to execute your solution from 2.20.2 versus the recursive version of the factorial program?

2.20.6 [5] <2.8> Show the contents of the stack after each function call, assuming that the input is 4.

Exercise 2.21

Assume that the stack and the static data segments are empty and that the stack and global pointers start at address 0x7fff fffc and 0x1000 8000, respectively. Assume the calling conventions as specified in Figure 2.11 and that function inputs are passed using registers `$a0-$a3` and returned in register `$r0`. Assume that leaf functions may only use saved registers.

a.

```
int my_global = 100;
main()
{
    int x = 10;
    int y = 20;
    int z;
    z = my_function(x, y)
}
int my_function(int x, int y)
{
    return x - y + my_global;
}
```

b.

```
int my_global = 100;
main()
{
    int z;
    my_global += 1;
    z = leaf_function(my_global);
}
int leaf_function(int x)
{
    return x + 1;
}
```

2.21.1 [5] <2.8> Write MIPS assembly code for the code in the table above.

2.21.2 [5] <2.8> Show the contents of the stack and the static data segments after each function call.

2.21.3 [5] <2.8> If the leaf function could use temporary registers (`$t0`, `$t1`, etc.), write the MIPS code for the code in the table above.

The following three problems in this Exercise refer to this function, written in MIPS assembly following the calling conventions from Figure 2.14:

a.
```
f: add  $v0,$a1,$a0
   bnez $a2,L
   sub  $v0,$a0,$a1
L: jr   $v0
```

b.
```
f: add  $a2,$a3,$a2
   slt  $a2,$a2,$a0
   move $v0,$a1
   beqz $a2, L
   jr   $ra
L: move $a0,$a1
   jal  g       ; Tail call
```

2.21.4 [10] <2.8> This code contains a mistake that violates the MIPS calling convention. What is this mistake and how should it be fixed?

2.21.5 [10] <2.8> What is the C equivalent of this code? Assume that the function's arguments are named `a`, `b`, `c`, etc. in the C version of the function.

2.21.6 [10] <2.8> At the point where this function is called register `$a0`, `$a1`, `$a2`, and `$a3` have values 1, 100, 1000, and 30, respectively. What is the value returned by this function? If another function `g` is called from `f`, assume that the value returned from `g` is always 500.

Exercise 2.22

This exercise explores ASCII and Unicode conversion.

The following table shows strings of characters.

a.	`hello world`
b.	`0123456789`

2.22.1 [5] <2.9> Translate the strings into hexadecimal ASCII byte values.

2.22.2 [5] <2.9> Translate the strings into 16-bit Unicode (using hex notation and the Basic Latin character set).

The following table shows hexadecimal ASCII character values.

a.	`41 44 44`
b.	`4D 49 50 53`

2.22.3 [5] <2.5, 2.9>

Exercise 2.23

In this exercise, you will
strings into the number

a.	positive and negative in
b.	positive hexadecimal in

2.23.1 [10] <2.9> W
ASCII number string w
Your program should e
string containing some
should compute the int
number in register `$v0`
your program should st
`$a0` points to a sequenc
"24"), then when the pr

Exercise 2.24

Assume that the registe
`$t2` contains the addr
endian addressing.

a.
```
lbu $t0, 0($t1)
sw  $t0, 0($t2)
```

b.
```
lb  $t0, 0($t1)
sh  $t0, 0($t2)
```

2.24.1 [5] <2.9> Assu

1000 0000

What value is stored at
memory location point

2.24.2 [5] <2.9> Assu

1000 0000

2.22.3 [5] <2.5, 2.9> Translate the hexadecimal ASCII values to text.

Exercise 2.23

In this exercise, you will be asked to write an MIPS assembly program that converts strings into the number format as specified in the table.

a.	positive and negative integer decimal strings
b.	positive hexadecimal integers

2.23.1 [10] <2.9> Write a program in MIPS assembly language to convert an ASCII number string with the conditions listed in the table above, to an integer. Your program should expect register `$a0` to hold the address of a null-terminated string containing some combination of the digits 0 through 9. Your program should compute the integer value equivalent to this string of digits, then place the number in register `$v0`. If a non-digit character appears anywhere in the string, your program should stop with the value –1 in register `$v0`. For example, if register `$a0` points to a sequence of three bytes 50_{ten}, 52_{ten}, 0_{ten} (the null-terminated string "24"), then when the program stops, register `$v0` should contain the value 24_{ten}.

Exercise 2.24

Assume that the register `$t1` contains the address 0x1000 0000 and the register `$t2` contains the address 0x1000 0010. Note the MIPS architecture utilizes big-endian addressing.

a.	`lbu $t0, 0($t1)` `sw $t0, 0($t2)`
b.	`lb $t0, 0($t1)` `sh $t0, 0($t2)`

2.24.1 [5] <2.9> Assume that the data (in hexadecimal) at address 0x1000 0000 is:

1000 0000	12	34	56	78

What value is stored at the address pointed to by register `$t2`? Assume that the memory location pointed to `$t2` is initialized to 0xFFFF FFFF.

2.24.2 [5] <2.9> Assume that the data (in hexadecimal) at address 0x1000 0000 is:

1000 0000	80	80	80	80

What value is stored at the address pointed to by register `$t2`? Assume that the memory location pointed to `$t2` is initialized to 0x0000 0000.

2.24.3 [5] <2.9> Assume that the data (in hexadecimal) at address 0x1000 0000 is:

1000 0000	11	00	00	FF

What value is stored at the address pointed to by register `$t2`? Assume that the memory location pointed to `$t2` is initialized to 0x5555 5555.

Exercise 2.25

In this exercise, you will explore 32-bit constants in MIPS. For the following problems, you will be using the binary data in the table below.

a.	$0010\ 0000\ 0000\ 0001\ 0100\ 1001\ 0010\ 0100_{two}$
b.	$0000\ 1111\ 1011\ 1110\ 0100\ 0000\ 0000\ 0000_{two}$

2.25.1 [10] <2.10> Write the MIPS assembly code that creates the 32-bit constants listed above and stores that value to register `$t1`.

2.25.2 [5] <2.6, 2.10> If the current value of the PC is 0x00000000, can you use a single jump instruction to get to the PC address as shown in the table above?

2.25.3 [5] <2.6, 2.10> If the current value of the PC is 0x00000600, can you use a single branch instruction to get to the PC address as shown in the table above?

2.25.4 [5] <2.6, 2.10> 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 the table above?

2.25.5 [10] <2.10> If the immediate field of an MIPS instruction was only 8 bits wide, write the MIPS code that creates the 32-bit constants listed above and stores that value to register `$t1`. Do not use the `lui` instruction.

For the following problems, you will be using the MIPS assembly code as listed in the table.

a.	`lui $t0, 0x1234` `addi $t0, $t0, 0x5678`
b.	`lui $t0, 0x1234` `andi $t0, $t0, 0x5678`

2.25.6 [5] <2.6, 2.10> W
in the table above?

2.25.7 [5] <2.6, 2.10> W
the table. Assume that the
16 bits.

Exercise 2.26

For this exercise, you will
MIPS. For the following p

a.	0x00020000
b.	0xFFFFFF00

2.26.1 [10] <2.6, 2.10>
jump instructions) do yo

2.26.2 [10] <2.6, 2.10>
instructions (no jump re
get to the target address i

2.26.3 [10] <2.6, 2.10>
designers have decided t
bits to 8 bits. If the PC is
needed to set the PC to t

For the following proble
instruction set architectu

a.	128 registers
b.	Four times as many diffe

2.26.4 [10] <2.6, 2.10>
the instruction format
above, what is the impac
that all instructions rem
format of i-type instruct
instruction.

2.26.5 [10] <2.6, 2.1
fied, the instruction fo

2.25.6 [5] <2.6, 2.10> What is the value of register `$t0` after the sequence of code in the table above?

2.25.7 [5] <2.6, 2.10> Write C code that is equivalent to the assembly code in the table. Assume that the largest constant that you can load into a 32-bit integer is 16 bits.

Exercise 2.26

For this exercise, you will explore the range of branch and jump instructions in MIPS. For the following problems, use the hexadecimal data in the table below.

a.	0x00020000
b.	0xFFFFFF00

2.26.1 [10] <2.6, 2.10> If the PC is at address 0x00000000, how many branch (no jump instructions) do you need to get to the address in the table above?

2.26.2 [10] <2.6, 2.10> If the PC is at address 0x00000000, how many jump instructions (no jump register instructions or branch instructions) are required to get to the target address in the table above?

2.26.3 [10] <2.6, 2.10> In order to reduce the size of MIPS programs, MIPS designers have decided to cut the immediate field of I-type instructions from 16 bits to 8 bits. If the PC is at address 0x0000000, how many branch instructions are needed to set the PC to the address in the table above?

For the following problems, you will be using making modifications to the MIPS instruction set architecture.

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

2.26.4 [10] <2.6, 2.10> If the instruction set of the MIPS processor is modified, the instruction format must also be changed. For each of the suggested changes above, what is the impact on the range of addresses for a beq instruction? Assume that all instructions remain 32 bits long and any changes made to the instruction format of i-type instructions only increase/decrease the immediate field of the beq instruction.

2.26.5 [10] <2.6, 2.10> If the instruction set of the MIPS processor is modified, the instruction format must also be changed. For each of the suggested

changes above, what is the impact on the range of addresses for a jump instruction? Assume that instructions remain 32 bits long and any changes made to the instruction format of J-type instructions only impact the address field of the jump instruction.

2.26.6 [10] <2.6, 2.10> If the instruction set of the MIPS processor is modified, the instruction format must also be changed. For each of the suggested changes above, what is the impact on the range of addresses for a jump register instruction, assuming that each instruction must be 32 bits.

Exercise 2.27

In the following problems, you will be exploring different addressing modes in the MIPS instruction set architecture. These different addressing modes are listed in the table below.

a.	Base or Displacement Addressing
b.	Pseudodirect Addressing

2.27.1 [5] <2.10> In the table above are different addressing modes of the MIPS instruction set. Give an example MIPS instructios that shows the MIPS addressing mode.

2.27.2 [5] <2.10> For the instructions in 2.27.1, what is the instruction format type used for the given instruction?

2.27.3 [5] <2.10> List the benefits and drawbacks of a particular MIPS addressing mode. Write MIPS code that shows these benefits and drawbacks.

In the following problems, you will be using the MIPS assembly code as listed below to explore the trade-offs of the immediate field in the MIPS I-type instructions.

a.	`0x00400000 beq $s0, $0, FAR`
	`...`
	`0x00403100 FAR: addi $s0, $s0, 1`

b.	`0x00000100 j AWAY`
	`...`
	`0x04000010 AWAY: addi $s0, $s0, 1`

2.27.4 [15] <2.10> For the MIPS statements above, show the bit-level instruction representation of each of the instructions in hexadecimal.

2.27.5 [10] <2.10> By reducing the size of the immediate fields of the I-type and J-type instructions, we can save on the number of bits needed to represent these types of instructions. If the immediate field of I-type instructions were 8 bits and the immediate field of J-type instructions were 18 bits, rewrite the MIPS code above to reflect this change. Avoid using the `lui` instruction.

2.27.6 [5] <2.10> How many extra instructions are needed to do execute your code in 2.27.5 MIPS statements in the table versus the code shown in the table above?

Exercise 2.28

The following table contains MIPS assembly code for a lock. Refer to the definition of the ll and sc pairs of MIPS instructions.

a.

```
try: MOV   R3,R4
     LL    R2,0(R2)
     ADDI  R2,R2, 1
     SC    R3,0(R1)
     BEQZ  R3,try
     MOV   R4,R2
```

2.28.1 [5] <2.11> For each test and fail of the store conditional, how many instructions need to be executed?

2.28.2 [5] <2.11> For the load locked/store conditional code above, explain why this code may fail.

2.28.3 [15] <2.11> Rewrite the code above so that the code may operate correctly. Be sure to avoid any race conditions.

Each entry in the following table has code and also shows the contents of various registers. The notation "(`$s1`)" shows the contents of a memory location pointed to by register `$s1`. The assembly code in each table is executed in the cycle shown on parallel processors with a shared memory space.

a.

Processor 1	Processor 2	Cycle	Processor 1		Mem	Processor 2	
			$t1	$t0	($s1)	$t1	$t0
		0	1	2	99	30	40
	`ll $t1, 0($s1)`	1					
`ll $t1, 0($s1)`		2					
	`sc $t0, 0($s1)`	3					
`sc $t0, 0($s1)`		4					

b.

Processor 1	Processor 2	Cycle	Processor 1		Mem	Processor 2	
			$t1	$t0	($s1)	$t1	$t0
		0	1	2	99	30	40
ll $t1,0($s1)		1					
	ll $t1,0($s1)	2					
	addi $t1,$t1,1	3					
	sc $t1,0($s1)	4					
sc $t0,0($s1)		5					

2.28.4 [5] <2.11> Fill out the table with the value of the registers for each given cycle.

Exercise 2.29

The first three problems in this Exercise refer to a critical section of the form

```
lock(lk);
operation
unlock(lk);
```

where the "operation" updates the shared variable `shvar` using the local (non-shared) variable `x` as follows:

	Operation
a.	`shvar=max(shvar,x);`
b.	`if(shvar>0)` `shvar=max(shvar,x);`

2.29.1 [10] <2.11> Write the MIPS assembly code for this critical section, assuming 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, i.e., you should include the MIPS instructions for `lock()`, `unlock()`, `max()`, and `min()` operations. Use `ll/sc` instructions to implement the `lock()` operation, and the `unlock()` operation is simply an ordinary store instruction.

2.29.2 [10] <2.11> Repeat problem 2.29.1, 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`.

2.29.3 [10] <2.11> Compare the best-case performance of your code from 2.29.1 and 2.29.2, assuming that each instruction takes one cycle to execute. Note: best-case

means that `ll/sc` always s
and if there is a branch we
executed instructions.

2.29.4 [10] <2.11> Usin
happens when two proce
time, assuming that each p

2.29.5 [10] <2.11> Expl
the address of variable `sh`
`$a2` contains the value of

2.29.6 [10] <2.11> If we
shared variables (e.g., `shv`
this easily using the appro
lock operation and the co
do this using the approac
both shared variables in
together as a single atomi

Exercise 2.30

Assembler instructions ar
in MIPS programs. The
that get translated to actu

a.	`clear $t0`
b.	`beq $t1, large, LO`

2.30.1 [5] <2.12> For
minimal sequence of act
may need to use tempor
number that requires 32
16 bits.

The table below contain
actual MIPS instructions

a.	`bltu $s0, $t1, Lo`
b.	`ulw $v0, v`

means that `ll/sc` always succeeds, the lock is always free when we want to `lock()`, and if there is a branch we take the path that completes the operation with fewer executed instructions.

2.29.4 [10] <2.11> Using your code from 2.29.2 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.

2.29.5 [10] <2.11> Explain why in your code from 2.29.2 register `$a1` contains the address of variable `shvar` and not the value of that variable, and why register `$a2` contains the value of variable `x` and not its address.

2.29.6 [10] <2.11> If we want to atomically perform the same operation on two shared variables (e.g., `shvar1` and `shvar2`) in the same critical section, we can do this easily using the approach from 2.29.1 (simply put both updates between the lock operation and the corresponding unlock operation). Explain why we cannot do this using the approach from 2.29.2. i.e., why we cannot use `ll/sc` to access both shared variables in a way that guarantees that both updates are executed together as a single atomic operation.

Exercise 2.30

Assembler instructions are not a part of the MIPS instruction set, but often appear in MIPS programs. The table below contains some MIPS assembly instructions that get translated to actual MIPS instructions.

a.	`clear $t0`
b.	`beq $t1, large, LOOP`

2.30.1 [5] <2.12> For each assembly instruction in the table above, produce a minimal sequence of actual MIPS instructions to accomplish the same thing. You may need to use temporary registers in some cases. In the table `large` refers to a number that requires 32 bits to represent and `small` to a number that can fit into 16 bits.

The table below contains some MIPS assembly instructions that get translated to actual MIPS instructions.

a.	`bltu $s0, $t1, Loop`
b.	`ulw $v0, v`

2.30.2 [5] <2.12> Does the instruction in the table above need to be edited during the link phase? Why?

Exercise 2.31

The table below contains the link-level details of two different procedures. In this exercise, you will be taking the place of the linker.

a.

Procedure A

Text Segment	Address	Instruction	
	0	`lbu $a0, 0($gp)`	
	4	`jal 0`	
Data Segment	0	(X)	
	...	...	
Relocation Info	Address	Instruction Type	Dependency
	0	`lbu`	X
	4	`jal`	B
Symbol Table	Address	Symbol	
	—	X	
	—	B	

Procedure B

Text Segment	Address	Instruction	
	0	`sw $a1, 0($gp)`	
	4	`jal 0`	
Data Segment	0	(Y)	
	...	...	
Relocation Info	Address	Instruction Type	Dependency
	0	`sw`	Y
	4	`jal`	A
Symbol Table	Address	Symbol	
	—	Y	
	—	A	

b.

Procedure A

Text Segment	Address	Instruction	
	0	`lui $at, 0`	
	4	`ori $a0, $at, 0`	
	...	...	
	0x84	`jr $ra`	
	...	...	
Data Segment	0	(X)	
	...	...	
Relocation Info	Address	Instruction Type	Dependency
	0	`lui`	X
	4	`ori`	X
Symbol Table	Address	Symbol	
	—	X	

Procedure B

Text Segment	Address	Instruction	
	0	`sw $a0, 0($gp)`	
	4	`jmp 0`	
	...	...	
	0x180	`jal 0`	
	...	...	
Data Segment	0	(Y)	
	...	...	
Relocation Info	Address	Instruction Type	Dependency
	0	`sw`	Y
	4	`jmp`	FOO
	0x180	`jal`	A
Symbol Table	Address	Symbol	
	—	Y	
	0x180	FOO	
	—	A	

2.31.1 [5] <2.12> Li
Assume that Procedur
cedure B has a text siz
allocation strategy as s

2.31.2 [5] <2.12> W

2.31.3 [5] <2.12> G
jump instructions, wh
branch and jump instr

Exercise 2.32

The first three problen
the code in Figure 2.24

a.
```
void swap(int *p
  int temp;
  temp=*p;
  *p=*q;
  *q=temp;
}
```

b.
```
void swap(int *p
  *p=*p+*q;
  *q=*p-*q;
  *p=*p-*q;
}
```

2.32.1 [10] <2.13> T

2.32.2 [5] <2.13> W

2.32.3 [5] <2.13> If
your MIPS code for `sw`

For the remaining thre
tion from Figure 2.27 i

a.	Use the `swap` functior
b.	Sort an array of n byte

2.32.4 [5] <2.13> Do
isters in Figure 2.27?

2.32.5 [10] <2.13> V
how many more (or fe

2.31.1 [5] <2.12> Link the object files above to form the executable file header. Assume that Procedure A has a text size of 0x140 and data size of 0x40 and Procedure B has a text size of 0x300 and data size of 0x50. Also assume the memory allocation strategy as shown in Figure 2.13.

2.31.2 [5] <2.12> What limitations, if any, are there on the size of an executable?

2.31.3 [5] <2.12> Given your understanding of the limitations of branch and jump instructions, why might an assembler have problems directly implementing branch and jump instructions an object file?

Exercise 2.32

The first three problems in this exercise assume that the function `swap`, instead of the code in Figure 2.24, is defined in C as follows:

a.
```
void swap(int *p, int *q){
  int temp;
  temp=*p;
  *p=*q;
  *q=temp;
}
```

b.
```
void swap(int *p, int *q){
  *p=*p+*q;
  *q=*p-*q;
  *p=*p-*q;
}
```

2.32.1 [10] <2.13> Translate this function into MIPS assembler code.

2.32.2 [5] <2.13> What needs to change in the `sort` function?

2.32.3 [5] <2.13> If we were sorting 8-bit bytes, not 32-bit words, how would your MIPS code for `swap` in 2.32.1 change?

For the remaining three problems in this Exercise, we assume that the `sort` function from Figure 2.27 is changed in the following way:

a.	Use the `swap` function from the beginning of this exercise.
b.	Sort an array of `n` bytes instead of `n` words.

2.32.4 [5] <2.13> Does this change affect the code for saving and restoring registers in Figure 2.27?

2.32.5 [10] <2.13> When sorting a 10-element array that was already sorted, how many more (or fewer) instructions are executed as a result of this change?

2.32.6 [10] <2.13> When sorting a 10-element array that was sorted in descending order (opposite of the order that `sort()` creates), how many more (or fewer) instructions are executed as a result of this change?

Exercise 2.33

The problems in this Exercise refer to the following function, given as array code:

a.
```
void copy(int a[], int b[], int n){
  int i;
  for(i=0;i!=n;i++)
  a[i]=b[i];
}
```

b.
```
void shift(int a[], int n){
  int i;
  for(i=0;i!=n-1;i++)
  a[i]=a[i+1];
}
```

2.33.1 [10] <2.14> Translate this function into MIPS assembly.

2.33.2 [10] <2.14> Convert this function into pointer-based code (in C).

2.33.3 [10] <2.14> Translate your pointer-based C code from 2.33.2 into MIPS assembly.

2.33.4 [5] <2.14> Compare the worst-case number of executed instructions per non-last loop iteration in your array-based code from 2.33.1 and your pointer-based code from 2.33.3. Note: the worst case occurs when branch conditions are such that the longest path through the code is taken, i.e., if there is an if statement, the result of the condition check is such that the path with more instructions is taken. However, if the result of the condition check would cause the loop to exit, then we assume that the path that keeps us in the loop is taken.

2.33.5 [5] <2.14> Compare the number of temporary registers (t-registers) needed for your array-based code from 2.33.1 and for your pointer-based code from 2.33.3.

2.33.6 [5] <2.14> What would change in your answer from 2.33.4 if registers `$t0-$t7` and `$a0-$a3` in the MIPS calling convention were all callee-saved, just like `$s0-$s7`?

Exercise 2.34

The table below contains
translate ARM assembly c

a.
```
ADD   r0, r1, r2
ADC   r0, r1, r2
```

b.
```
CMP   r0, #4
ADDNE r1, r1, r0
```

2.34.1 [5] <2.16> For th
assembly code. Assume th
as MIPS registers `$s0`, `$s`
(`$t0`, etc.) where necessar

2.34.2 [5] <2.16> For t
the bit fields that represen

The table below contains
translate MIPS assembly c

a.
```
nor $t0, #s0, 0
and $s1, $s1, $t0
```

b.
```
sll $s1, $s2, 16
srl $s2, $s2, 16
or  $s1, $s1, $s2
```

2.34.3 [5] <2.16> For t
sponds to the sequence of

2.34.4 [5] <2.16> Show

Exercise 2.35

The ARM processor has a
MIPS. The following prob

a.
```
LDR   r0, [r1, #4
```

b.
```
LDMIA r0!, {r1-r3}
```

2.35.1 [5] <2.16> Ident
instructions in the table a