MIPS Assembly Language Programming

Bob Britton
Chapter 4
PCSpim The MIPS Simulator

Analyzing the Data Segment

.data
prompt: .asciz "n Please Input a value for N = "
result: .asciz " The sum of the integers from 1 to N is "
bye: .asciiz " **** Adios Amigo – Have a good day ****"

This is an example of an addressing structure called Little Indian where the right most byte in a word has the smaller address.
MIPS Assembly Language Programming

Translating Assembly Language to Machine language

Use the information in Appendix C to verify that 0x3402000A is the correct machine language encoding of the instruction

ori $2, $0, 10
li $v0, 10

In Appendix C we are shown how this instruction is encoded in binary

ori Rt, Rs, Imm # RF[Rt] = RF[Rs] OR Imm

<table>
<thead>
<tr>
<th>Op Code</th>
<th>Rs</th>
<th>Rt</th>
<th>Imm</th>
</tr>
</thead>
</table>
| 001101 | ss ss ss ss ss ss ss ss ss | ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt ttt t...
Translating Assembly Language “Shift” Instruction to Machine Language

In Appendix C we are shown how Shift Right Logical is encoded in binary

\[ \text{srl} \quad \text{Rd, Rt, sa} \quad \# \text{Rd} = \text{Rt} >> \text{sa} \]

Op Code | Rd | Sd | code
---|---|---|---
000000000000000000000010

srl $23, $23, 4

0x00000000000001011111111111111111111111

$0x00 0 1 7 B 9 0 2$

---

Translating Assembly Language “addi” to Machine Language

In Appendix C we are shown how addi is encoded in binary

\[ \text{addi} \quad \text{Rt, Rs, Imm} \quad \# \text{RF}[\text{Rt}] = \text{RF}[\text{Rs}] + \text{se Imm} \]

Op Code | Rs | Rt | Imm
---|---|---|---
00100000000000000000000000000000

addi $8, $8, -1

0x01000010000100000000000000000000

$0x2 1 0 8 f f f f$

---

Translating Assembly Language “Store Byte” to Machine Language

In Appendix C we are shown how Store Byte is encoded in binary

\[ \text{sb} \quad \text{Rt, offset(Rs)} \quad \# \text{Mem}[^{\text{RF}}[\text{Rs}] + \text{Offset}] = \text{RF}[\text{Rt}] \]

Op Code | Rs | Rt | Offset
---|---|---|---
10100000000000000000000000000000

sb $4, 4($22)

101000101100010000000000000000100

$0xA 2 C 4 0 0 0 4$

---
MIPS Assembly Language Programming

Bob Britton
Chapter 5

Efficient Algorithm Development

A Function to Print a Binary Value as a Hexadecimal Number

For example suppose the following value is in $a0:
00000100101011110101001101

In Hexadecimal this value is printed as:0x04ABF54D

Algorithm:
t2 = $buffer + 10;
for (t0 = 8; t0 > 0; t0 = t0 - 1)
    t1 = a0 & 15;
    t1 = t1 + 0x30;
    a0 = a0 >> 4;
    if (t1 > 57) t1 = t1+7;
    mem(t2) = t1;
    t2 = t2-1
mem(t2) = “x”; mem(t2-1) = “0”; mem(t2-2) = 0x20;
Print the ASCII string in the buffer

Exercises 4.2

What is the character string corresponding to the following ASCII codes? Remember that for simulations running on Intel-based platforms, the characters are stored in reverse order within each word.)
0x2a2a2a2a 0x6964120 0x4120736f 0x6f67696d 0x48202d20 0x20657661

**** i d A A s o g i m H - e v a

**** Adios Amigo – Have

Exercise 4.3

// Data declaration section
.data
    msg: .asciiz "in Elapsed Time = "
.text

# Start of code section
main:
    li $s1, 0
    # Time Counter
countdown:
    li $s0, 256000
    # Adjustable Time Factor
waitloop:
    addi $s0, $s0, -1
    bnez $s0, waitloop
    addi $s1, $s1, 5
    li $s0, 4
    la $a0, msg
    syscall
    move $a0, $s1
    li $s0, 1
    syscall
    addi $s0, $s0, -60
    bnez $s0, countdown
    li $s0, 10
    syscall
Solution to Example Exam - MIPS Code

<table>
<thead>
<tr>
<th>Label</th>
<th>Op-Code</th>
<th>Dest.</th>
<th>$1, $2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>.word</td>
<td>-29, -30, 75, 34, -2, 90, -11, 98, 1, 0, 76</td>
<td></td>
<td></td>
</tr>
<tr>
<td>main</td>
<td>la</td>
<td>$a1, array</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>li</td>
<td>$a0, 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td># $a1 = &amp;array</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td># $a0 = 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>loop</td>
<td>lw</td>
<td>$t0, 0($a1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>beqz</td>
<td>$t0, done</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>addi</td>
<td>$a1, $a1, 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>andi</td>
<td>$t3, $t0, 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>bnez</td>
<td>$t3, odd</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>bltz</td>
<td>$t0, loop</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>add</td>
<td>$a0, $a0, $t0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td># $t2 = $t2 + $t0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>odd</td>
<td>bgtz</td>
<td>$t0, loop</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sub</td>
<td>$a0, $a0, $t0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td># $a0 = $a0 - $t0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>done</td>
<td>li</td>
<td>$v0, 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>syscall</td>
<td>$v0, 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>syscall</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example Exam Questions

You are given the following two numbers in two's complement representation. Perform the binary subtraction and indicate if there is signed overflow?

\[
\begin{array}{c}
10101100 \\
- \ 00001100 \\
\hline
1110100 \\
101000000
\end{array}
\]

Explain Why:

\[
10101100 \\
- \ 00001100 \\
\hline
1110100 \\
101000000
\]

You are given the following two numbers in two's complement representation. Perform the binary addition and indicate if there is signed overflow?

\[
\begin{array}{c}
10100100 \\
+ \ 11101100 \\
\hline
\end{array}
\]

Explain Why:

\[
\begin{array}{c}
10100100 \\
+ \ 11101100 \\
\hline
\end{array}
\]

Beginning of the PrintHex Function

```
# Algorithmic Description in Pseudo Code:
#
# .globl PrintHex
.data
buffer:.asciiz "0x00000000"
.text
PrintHex:
    la $a1, buffer
    addi $a1, $a1, 10 # Set pointer to end of buffer
    ...
    ...
    # Body of the Algorithm
    ...
    jr $ra
```

An Example Exam Question

Write a program that will compute the sum of the even positive values, minus the odd negative values in an array of words. Stop when a value of zero is found in the array. For Example:

array: .word -29, -30, 75, 34, -2, 90, -11, 98, 1, 0, 76

Pseudo Code for the Algorithm:

\[
\begin{array}{c}
$s1 = \&array; \\
$s0 = 0;
\end{array}
\]

loop:

\[
\begin{array}{c}
$s0 = \text{Mem}($s1); \\
\text{if} ($s0 == 0) \text{go to done}; \\
$s1 = $s1 + 4; \\
$s3 = $s0 \& 1; \\
\text{if} ($s0 >= 0 \&\& $s3 == 0) \text{ $s0 = $s0 + $s0}; \\
\text{else if} ($s0 < 0 \&\& $s3 != 0) \text{ $s0 = $s0 - $s0}; \\
\text{go to loop}
\end{array}
\]

done:

\[
\begin{array}{c}
\text{syscall(1) }<< \text{ $s0}; \\
\text{exit}
\end{array}
\]
Example Exam Questions

Sign extend the 2-digit hex number 0x90 to a 4-digit hex number. 0x_______

Show how the following PSEUDOCODE expression can be efficiently implemented in MIPS assembly language: $t0 = $s0 / 8 - 2 * $s1 + $s2;

sra $t0, $s0, 3
sll $t1, $s1, 1
sub $t0, $t0, $t1
add $t0, $t0, $s2

An Example Function

The following code segment is stored in memory starting at memory location 0x00067890. What are the two possible values for the contents of the PC after the branch instruction has executed?

Ox_________________________ Ox_________________________

andi $t3, $t1, 1  #
bnez $t3, test   #
add $t1, $s1, $s0  #
test:

loop:
blez Sa1, retzz  # If (a1 <= 0) Branch to Return
addi Sa1, Sa1, -1  # Decrement loop count
lw S10, 0(Sa0)  # Get a value from the array
addi Sa0, Sa0, 4  # Increment array pointer to next word
bltz S10, negg  # If value is negative Branch to negg
add $v0, $v0, $t1  # Add to the positive sum
b loop  # Branch to loop

negg:
add $v1, $v1, $t0  # Add to the negative sum
b loop  # Branch to loop
retzz: jr $ra  # Return
Classroom Exercises to Strengthen Your Mental Muscle

The class is presented with a challenging assembly language programming exercise. Everyone individually develops a pseudo-code solution to the exercise.

Students compare their pseudo-code and refine their algorithms.

Individually develop an assembly language solution to the exercise, and calculate a Figure of Merit (FM) for your solution.

Performance Evaluation

Time and Space Tradeoff
The Figure of Merit (FM) we will use is:

Total clock cycles required to execute the code.
Total number of memory locations required to store the code.
Assume:
• Multiplication requires 32 clock cycles
• Division requires 38 clock cycles

A Function that takes a Binary Value and prints the equivalent Decimal Representation, Right Justified

This function is similar to the PrintHex function Except you Divide by 10 instead of 16
Differences:
Negative values must be preceded by a Minus.
Need to place Leading space spaces (ASCII 20) into the print buffer.
Exercise 5.1 (MIPS Assembly Language)

<table>
<thead>
<tr>
<th>Label</th>
<th>Op-Code</th>
<th>Dest</th>
<th>$t1, $t2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>.data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>chico</td>
<td>.space</td>
<td>400</td>
<td></td>
<td></td>
</tr>
<tr>
<td>result</td>
<td>.word</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.globl</td>
<td>main</td>
<td></td>
<td></td>
</tr>
<tr>
<td>text</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>main</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>la</td>
<td>$a0, chico</td>
<td></td>
<td></td>
<td># Load address pointer</td>
</tr>
<tr>
<td>li</td>
<td>$t0, 0</td>
<td></td>
<td></td>
<td># Clear sum</td>
</tr>
<tr>
<td>li</td>
<td>$t1, 100</td>
<td></td>
<td></td>
<td># Initialize loop count</td>
</tr>
<tr>
<td>loop</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw</td>
<td>$t2, 0/($a0)</td>
<td></td>
<td></td>
<td># $t2 = Mem($a0)</td>
</tr>
<tr>
<td>add</td>
<td>$t0, $t0, $t2</td>
<td></td>
<td></td>
<td># $t0 = $t0 + $t2</td>
</tr>
<tr>
<td>addi</td>
<td>$a0, $a0, 4</td>
<td></td>
<td></td>
<td># Inc. address pointer</td>
</tr>
<tr>
<td>addi</td>
<td>$t1, $t1, -1</td>
<td></td>
<td></td>
<td># Dec. loop count</td>
</tr>
<tr>
<td>bgtz</td>
<td>$t1, loop</td>
<td></td>
<td></td>
<td># if ($t1 &gt; 0) branch</td>
</tr>
<tr>
<td>sw</td>
<td>$t0, 0/($a0)</td>
<td></td>
<td></td>
<td># Store the result</td>
</tr>
<tr>
<td>li</td>
<td>$v0, 10</td>
<td></td>
<td></td>
<td># End of program</td>
</tr>
</tbody>
</table>

Exercise 5.1

Write a MIPS assembly language program to find the Sum of the first 100 words of data in the memory data segment with the label “chico”. Store the resulting sum in the next memory location beyond the end of the array chico.

MIPS Assembly Language Programming

Bob Britton
Chapter 5 - c

Efficient Algorithm Development

Exercise 5.1 (Pseudo Code)

```
$a0 = &chico;  # "&" means "Address of"
$t0 = 0;
For ($t1 = 100; $t1 > 0; $t1 = $t1 - 1)
{
    $t0 = $t0 + Mem($a0);
    $a0 = $a0 + 4;
}
Mem($a0) = $t0;
```
Exercise 5.2 (MIPS Assembly Language)

<table>
<thead>
<tr>
<th>Label</th>
<th>Op-Code</th>
<th>Dest</th>
<th>S1, S2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>.data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SRC:</td>
<td>.space</td>
<td>400</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEST:</td>
<td>.space</td>
<td>400</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.globl</td>
<td>main</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>.text</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>main:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>la</td>
<td>$a1, SRC</td>
<td></td>
<td></td>
<td># $a1 = &amp;SRC</td>
</tr>
<tr>
<td>la</td>
<td>$a2, DEST</td>
<td></td>
<td></td>
<td># $a2 = &amp;DEST</td>
</tr>
<tr>
<td>li</td>
<td>$t0, 100</td>
<td></td>
<td></td>
<td># $t0 = 100</td>
</tr>
<tr>
<td>loop:</td>
<td>lw</td>
<td>$t1, 0($a1)</td>
<td></td>
<td># $t1 = Mem($a1)</td>
</tr>
<tr>
<td></td>
<td>sw</td>
<td>$t1, 0($a2)</td>
<td></td>
<td># Mem($a2) = $t1</td>
</tr>
<tr>
<td></td>
<td>addi</td>
<td>$a1, $a1,4</td>
<td></td>
<td># $a1 = $a1+4</td>
</tr>
<tr>
<td></td>
<td>addi</td>
<td>$a2, $a2,4</td>
<td></td>
<td># $a2 = $a2+4</td>
</tr>
<tr>
<td></td>
<td>addi</td>
<td>$t0, $t0, -1</td>
<td></td>
<td># $t0 = $t0 - 1</td>
</tr>
<tr>
<td></td>
<td>bgtz</td>
<td>$t0, loop</td>
<td></td>
<td>#Branch if $t0 &gt; 0</td>
</tr>
<tr>
<td>li</td>
<td>$v0, 10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>syscall</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exercise 5.2

Write an efficient segment of MIPS assembly language code to transfer a block of 100 words starting at memory location “SRC” to another area of memory beginning at memory location “DEST”.

Exercise 5.3

Write a MIPS function which accepts an integer word in register $a0 and returns its absolute value in $a0.

Also show an example code segment that calls the ABS function twice, to test the function.

Exercise 5.2 (Pseudo Code)

```
$ a 1 = & SRC ;   # “&” means “Address of”
$ a 2 = & DEST ;
for ($ t 0 = 100 ; $ t 0 > 0 ; $ t 0 = $ t 0 - 1 )
{ $ t 1 = Mem ( $ a 1 ) ;
  Mem ( $ a 2 ) = $ t 1 ;
  $ a 1 = $ a 1 + 4 ;
  $ a 2 = $ a 2 + 4 ;
} 
```
Exercise 5.3 (Pseudo Code)

Function ABS($a0);
if ($a0 < 0) $a0 = $0 - $a0;
return;

Exercise 5.4

Write a function PENO (&X, N, SP, SN) that will find the sum of the positive even values and the negative odd values in an array X of length “N”.

"X" the address of an array, passed through $a0.
"N" is the length of the array, passed through $a1.
The procedure should return two values:
(1) The sum of all the positive even elements in the array, passed back through $v0.
(2) The sum of all the negative odd elements in the array, passed back through $v1.

Exercise 5.4 (Pseudo Code)

$v0 = 0;
v1 = 0;
for ( ; $a1 > 0; $a1 = $a1 - 1)
{ $t0 = Mem($a0);
  $t1 = $t0 & 1;
  $a0 = $a0 + 4;
  if ($t0 >= 0 & $t1 = 0) $v0 = $v0 + $t0;
  if ($t0 < 0 & $t1 != 0) $v1 = $v1 + $t0;
}
return;

Exercise 5.3 (MIPS Assembly Language)

```
Label   Op-Code Dest   S1, S2    Comments
.text
ABS:    bgez    $a0, return # If ($a0 >= 0) go to return
        sub    $a0, $0, $a0  # $a0 = 0 - $a0 (negate $a0)
return: jr      $ra      # Return

.globl main
.text
main:   li      $a0, -9876
        jal     ABS
        li      $v0, 1  # Output result
        syscall
        li      $a0, 9876
        jal     ABS
        li      $v0, 1  # Output result
        syscall
        li      $v0, 10 # End of program
        syscall
```
Exercise 5.5 (Pseudo Code)

Function SUM (a0: input value, $v0: output value)
$v0 = a0 + 1;$
$v0 = v0 * a0;$
$v0 = v0 >> 1$  #$v0 / 2;$
return;

Exercise 5.5 (MIPS Assembly Language)

Exercise 5.4 (MIPS Assembly Language)

<table>
<thead>
<tr>
<th>Label</th>
<th>Op-Code</th>
<th>Dest.</th>
<th>S1</th>
<th>S2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>PENO:</td>
<td>li</td>
<td>$v0, 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>li</td>
<td>$v1, 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| LOOP: | lw      | $t0, 0($a0)| # Get a value from the array
|       | andi   | $t2, $t0, 1| # Extract LSB
|       | addi   | $a0, $a0, 4| # Increment array pointer
|       | bltz   | $t0, NEG | # If the value is negative go to NEG
|       | bnez   | $t2, CHK | # If the value is odd go to CHK
|       | add    | $v0, $v0, $t0| # Add value to positive even sum
|       | b      | CHK | # Branch to CHK
| NEG:  | beqz   | $t2, CHK| # If value is even go to CHK
|       | add    | $v1, $v1, $t0| # Add value to negative odd sum
| CHK:  | addi   | $a1, $a1, -1| # Decrement loop counter
|       | bgtz   | $a1, LOOP| # If loop counter > 0 go to Loop
|       | jr     | $ra | # Return

Exercise 5.5

Write a function SUM(N) to find the sum of the integers from 1 to N, making use of the multiplication and shifting operations. The value N will be passed to the procedure in $a0 and the result will be returned in the $v0 register.

Write a MIPS assembly language main program that will call the Sum function five times each time passing a different value to the function for N, and printing the results. The values for N are defined below:

.data
N: .word 9, 10, 32666, 32777, 654321
Exercise 5.6 (Pseudo Code)

Mem($a1) = 1;
Mem($a1 + 4) = 1;
for ($a0 = $a0 - 2; $a0 > 0; $a0 = $a0-1)
{
    Mem($a1+8) = Mem($a1) + Mem($a1+4);
    $a1 = $a1 + 4;
}
return;

The Main Program

.data
N:
    .word 9, 10, 32666, 32777, 654321
    .text
main:
    li  $s0, 5
    la  $s1, N
loop:
    lw  $a0, 0($s1)
    addiu $s1, $s1, 4
    jal SUM
move $a0, $v0
li $v0, 1
syscall
addi $s0, $s0, -1
bnez $s0, loop
li $v0, 10
syscall

Exercise 5.6 (MIPS Assembly Language)

<table>
<thead>
<tr>
<th>Label</th>
<th>Op-Code</th>
<th>Dest</th>
<th>S1, S2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>fib:</td>
<td>li</td>
<td>$t0, 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw</td>
<td>$t0, 0($a1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw</td>
<td>$t0, 4($a1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>addi</td>
<td>$a0, $a0, -2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>loop:</td>
<td>lw</td>
<td>$t0, 0($a1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw</td>
<td>$t1, 4($a1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add</td>
<td>$t0, $t0, $t1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw</td>
<td>$t0, 8($a1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>addi</td>
<td>$a1, $a1, 4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>addi</td>
<td>$a0, $a0, -1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bgtz</td>
<td>$a0, loop</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>jr</td>
<td>$ra</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exercise 5.6

Write a function FIB(N, &array) to store the First N elements of the Fibonacci sequence into an array in memory. The value N is passed in $a0, and the address of the array is passed in register $a1.

The first few numbers of the Fibonacci sequence are: 1, 1, 2, 3, 5, 8, 13, ...........
Exercise 5.7 (MIPS Assembly Language)

<table>
<thead>
<tr>
<th>Label</th>
<th>Op-Code</th>
<th>Dest, S1, S2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>.text</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>order:</td>
<td>ble</td>
<td>$a0, $a1, next</td>
<td></td>
</tr>
<tr>
<td></td>
<td>move</td>
<td>$t0, $a1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>move</td>
<td>$a1, $a0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>move</td>
<td>$a0, $t0</td>
<td></td>
</tr>
<tr>
<td>next:</td>
<td>ble</td>
<td>$a1, $a2, done</td>
<td></td>
</tr>
<tr>
<td></td>
<td>move</td>
<td>$t0, $a2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>move</td>
<td>$a2, $a1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>move</td>
<td>$a1, $t0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ble</td>
<td>$a0, $a1, done</td>
<td></td>
</tr>
<tr>
<td></td>
<td>move</td>
<td>$t0, $a1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>move</td>
<td>$a1, $a0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>move</td>
<td>$a0, $t0</td>
<td></td>
</tr>
<tr>
<td>done:</td>
<td>jr</td>
<td>$ra</td>
<td></td>
</tr>
</tbody>
</table>

Exercise 5.7

Write a function that receives 3 integer words in registers $a0, $a1, & $a2, and returns them in ordered form with the minimum value in $a0 and the maximum value in $a2.

Exercise 5.8

Write the complete assembly language program, including data declarations, that corresponds to the following C code fragment.

Make use of the fact that multiplication and division by powers of 2 can be performed most efficiently by shifting.

```c
int main()
{
    int K, Y;
    int Z[50];
    Y = 56;
    K = 20;
    Z[K] = Y - 16 * (K / 4 + 210);
}
```

Exercise 5.7 (Pseudo Code)

Function Order($a0,$a1,$a2);
If ($a0 > $a1) exchange $a0 and $a1;
if ($a1 > $a2) exchange $a1 and $a2 else return;
If ($a0 > $a1) exchange $a0 and $a1;
return;
Functional Descriptions of Code Modules

A functional description will provide the information anyone needs to know if they are searching for a function that would be useful is solving some larger programming assignment. The functional description only describes what the function does, not how it is done. The functional description must explain how arguments are passed to the function and how results are returned. The following is an example functional description:

Hexout($a0: value)

A 32-bit binary value is passed to the function in register $a0 and the hexadecimal equivalent is printed out right justified.

Exercise 5.8 (MIPS Assembly Language)

<table>
<thead>
<tr>
<th>Label</th>
<th>Op-Code</th>
<th>Dest</th>
<th>$S1</th>
<th>$S2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>.globl</td>
<td>main</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.space</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.space</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.space</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.text</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>main:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>la</td>
<td>$t3, K</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>li</td>
<td>$t0, 56</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw</td>
<td>$t0, 4($t3)</td>
<td></td>
<td></td>
<td></td>
<td># Y = 56</td>
</tr>
<tr>
<td>li</td>
<td>$t1, 20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw</td>
<td>$t1, 0($t3)</td>
<td></td>
<td></td>
<td></td>
<td># K = 20</td>
</tr>
<tr>
<td>sra</td>
<td>$t1, $t1, 2</td>
<td></td>
<td></td>
<td></td>
<td># K/4</td>
</tr>
<tr>
<td>addi</td>
<td>$t1, $t1, 210</td>
<td></td>
<td></td>
<td></td>
<td># K/4 + 210</td>
</tr>
<tr>
<td>sll</td>
<td>$t1, $t1, 4</td>
<td></td>
<td></td>
<td></td>
<td># --- x 16</td>
</tr>
<tr>
<td>sub</td>
<td>$t2, $t0, $t1</td>
<td></td>
<td></td>
<td></td>
<td># 2x = Y - 16 * (K / 4 + 210)</td>
</tr>
<tr>
<td>lw</td>
<td>$t1, 0($t3)</td>
<td></td>
<td></td>
<td></td>
<td># $t1 = K</td>
</tr>
<tr>
<td>sll</td>
<td>$t1, $t1, 2</td>
<td></td>
<td></td>
<td></td>
<td># scale</td>
</tr>
<tr>
<td>addu</td>
<td>$t1, $t1, $t3</td>
<td></td>
<td></td>
<td></td>
<td># $t1 = $t1 + $t3</td>
</tr>
<tr>
<td>sw</td>
<td>$t2, 8($t1)</td>
<td></td>
<td></td>
<td></td>
<td># Z[K] = Y - 16 * (K / 4 + 210)</td>
</tr>
</tbody>
</table>

Exercise 5.9

Write a function to search through an array “X” of “N” words to find the minimum and maximum values. The address of the array will be passed to the function using register $a0, and the number of words in the array will be passed in register $a1. The minimum and maximum values are returned in registers $v0, & $v1.

MIPS Assembly Language Programming

Bob Britton
Chapter 5 - d

Efficient Algorithm Development
Exercise 5.10

Write a function to find the sum of the main diagonal elements in a two dimensional N by N array of 32 bit words. The address of the array and the size N are passed to the function in registers $a0 and $a1 respectively.

The result is returned in $v0.

The values in registers $a0 and $a1 should not be modified by this function.
Calculate the number of clock cycles required to execute your algorithm, assuming N=4

Exercise 5.9 (Pseudo Code)

MaxMin ($a0: address, $a1: number of words)
$v0 = Mem($a0);
$v1 = $v0;
$a1 = $a1 - 1;
While ($a1 > 0)
{
$v0 = Mem($a0);
if ($t0 < $v0) $v0 = $t0;
else if ($t0 > $v1) $v1 = $t0;
$a1 = $a1 - 1;
}
return;

Exercise 5.10 (Pseudocode)

$v0 = Mem($a0);
$t1 = $a0;
$t3 = ($a1+1) * 4;
for ($t0 = $a1-1; $t0 > 0, $t0 = $t0-1)
{$t1 = $t1 + $t3;
$v0 = $v0 + Mem($t1) 
}
return

Exercise 5.9 (MIPS Assembly Language)

<table>
<thead>
<tr>
<th>Label</th>
<th>Op-Code</th>
<th>Dest, S1, S2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxMin:</td>
<td>lw</td>
<td>$v0, 0($a0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>move</td>
<td>$v1, $v0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>addi</td>
<td>$a1, $a1, -1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>bnez</td>
<td>$a1, ret</td>
<td></td>
</tr>
<tr>
<td>loop:</td>
<td>addi</td>
<td>$a0, $a0, 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lw</td>
<td>$t0, 0($a0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>bge</td>
<td>$t0, $v0, next</td>
<td></td>
</tr>
<tr>
<td></td>
<td>move</td>
<td>$v0, $t0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>bnez</td>
<td>$t0, chk</td>
<td></td>
</tr>
<tr>
<td>next:</td>
<td>blez</td>
<td>$t0, $v1, chk</td>
<td></td>
</tr>
<tr>
<td></td>
<td>move</td>
<td>$v1, $t0</td>
<td></td>
</tr>
<tr>
<td>chk:</td>
<td>addi</td>
<td>$a1, $a1, -1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>bgtz</td>
<td>$a1, loop</td>
<td></td>
</tr>
<tr>
<td>ret:</td>
<td>jr</td>
<td>$ra</td>
<td></td>
</tr>
</tbody>
</table>
Exercise 5.11 (Pseudocode)

\[ v_0 = \text{Mem}(a0) \cdot \text{Mem}(a0+12) - \text{Mem}(a0+4) \cdot \text{Mem}(a0+8); \]

Return

Exercise 5.10 (MIPS Assembly Language)

<table>
<thead>
<tr>
<th>Label</th>
<th>Op-Code</th>
<th>Dest, S1, S2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>.text</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mdsum:</td>
<td>lw</td>
<td>$v0, 0(a0)</td>
<td>v0 = first element</td>
</tr>
<tr>
<td></td>
<td>move</td>
<td>$t1, a0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>addi</td>
<td>$t3, a1, 1</td>
<td>compute offset</td>
</tr>
<tr>
<td></td>
<td>sll</td>
<td>$t3, $t3, 2</td>
<td>multiply by 4</td>
</tr>
<tr>
<td></td>
<td>addi</td>
<td>$t0, a1, -1</td>
<td>init. loop count</td>
</tr>
<tr>
<td></td>
<td>blez</td>
<td>$t0, return</td>
<td></td>
</tr>
<tr>
<td>loop:</td>
<td>add</td>
<td>$t1, $t1, $t3</td>
<td>calc. next address</td>
</tr>
<tr>
<td></td>
<td>lw</td>
<td>$t2, 0($t1)</td>
<td>t2=Mem(t1)</td>
</tr>
<tr>
<td></td>
<td>add</td>
<td>$v0, $v0, $t2</td>
<td>add to sum</td>
</tr>
<tr>
<td></td>
<td>addi</td>
<td>$t0, $t0, -1</td>
<td>decrement loop count</td>
</tr>
<tr>
<td></td>
<td>bgtz</td>
<td>$t0, loop</td>
<td></td>
</tr>
<tr>
<td>return:</td>
<td>jr</td>
<td>$ra</td>
<td></td>
</tr>
</tbody>
</table>

Exercise 5.11 (MIPS Assembly Language)

<table>
<thead>
<tr>
<th>Label</th>
<th>Op-Code</th>
<th>Dest, S1, S2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>.globl</td>
<td>detem2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.text</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>detem2:</td>
<td>lw</td>
<td>$t0, 0(a0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lw</td>
<td>$t1, 12(a0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mult</td>
<td>$t1, $t0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mflo</td>
<td>$v0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lw</td>
<td>$t0, 4(a0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lw</td>
<td>$t1, 8(a0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mult</td>
<td>$t1, $t0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mflo</td>
<td>$t0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sub</td>
<td>$v0, $v0, $t0</td>
<td></td>
</tr>
<tr>
<td>return:</td>
<td>jr</td>
<td>$ra</td>
<td></td>
</tr>
</tbody>
</table>

Exercise 5.11

Write a function to find the determinant of a two by two matrix (array). The address of the array is passed to the function in registers $a0 and the result is returned in $v0. The value in register $a0 should not be modified by this function.

Calculate the number of clock ticks required to execute your algorithm.
Exercise 5.12 (MIPS Assembly Language)

Label  Op-Code Dest. S1, S2 Comments
.globl count
.text
count:
    li $v0, 0
while:
    andi $t0, $a0, 1
    srl $a0, $a0, 1
    add $v0, $v0, $t0
    bnez $a0, while
    jr $ra

Exercise 5.12

Write a MIPS assembly language function that accepts a binary number in register $a0 and returns a value corresponding to the number of one’s in the binary number.

Exercise 5.13

Translate the following pseudocode expression to MIPS assembly language code. Include code to insure that there is no array bounds violation when the store word (sw) instruction is executed. Note that the array “zap” is an array containing 50 words, thus the value in register $a0 must be in the range from 0 to 196. Include code to insure that the value in register $a0 is a word address offset into the array “zap.” If an array bounds violation is detected or the value in register $a0 is not a word address offset then branch to the label “Error.”

.data
zap: .space 200
.text
...
zap[$a0] = $s0

Exercise 5.12 (Pseudocode)

$v0 = 0;
while ($a0 != 0)
    {
        $t0 = $a0 & 1;
        $a0 = $a0 >> 1;
        $v0 = $v0 + $t0;
    }
Return
Exercise 5.14

Write a function to search through an array “X” of “N” words to find how many of the values are evenly divisible by four. The address of the array will be passed to the function using register $a0, and the number of words in the array will be passed in register $a1. Return the results in register $v0.

Exercise 5.13 (Pseudocode)

\$t0 = \$a0 & 3;
if ($t0 != 0) go to Error;
if ($a0 < 0) go to Error
if ($a0 > 196) go to Error
\$t0 = &zap
\$a0 = \$a0 + \$t0
Mem($a0) = $s0;

Exercise 5.14 (Pseudocode)

\$v0 = 0;
\$t3 = 3;
For ( ; $a1 > 0; $a1=$a1-1)
{ \$t2 = Mem ($a0);
  \$a0 = \$a0 + 4;
  \$t0 = \$t2 & \$t3;
  If ($t0 == 0) \$v0 = \$v0 + 1;
}
return

Exercise 5.13 (MIPS Assembly Language)

<table>
<thead>
<tr>
<th>Label</th>
<th>Op-Code</th>
<th>Dest.</th>
<th>S1</th>
<th>S2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>.data</td>
<td>.space</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.text</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>zap:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.and</td>
<td>.a0, .a0, 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.bnez</td>
<td>.t0, Error</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.bltz</td>
<td>.a0, Error</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.li</td>
<td>.t0, 196</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.bgt</td>
<td>.a0, .t0, Error</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.la</td>
<td>.t0, zap</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.add</td>
<td>.a0, .a0, .t0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.sw</td>
<td>.s0, 0(.a0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Exercise 5.14 (MIPS Assembly Language)

<table>
<thead>
<tr>
<th>Label</th>
<th>Op-Code</th>
<th>Dest.</th>
<th>S1, S2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Div4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>li</td>
<td>$v0, 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>li</td>
<td>$t3, 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>skip</td>
<td></td>
<td></td>
</tr>
<tr>
<td>loop</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw</td>
<td>$t2, 0($a0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addi</td>
<td>$a0, $a0, 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>and</td>
<td>$t0, $t2, $t3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bnez</td>
<td>$t0, skip</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addi</td>
<td>$v0, $v0, 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>skip</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>addi</td>
<td>$a1, $a1, -1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bgez</td>
<td>$a1, loop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jr</td>
<td>$ra</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Stack is in Memory and Register $sp Points to the Top of Stack

MIPS Assembly Language Programming
Bob Britton
Chapter 6 – a

Function Calls Using the Stack
Example of Jack Saving Important Temporary Registers

```assembly
addiu $sp, $sp, -32  # Allocate More Space on the Stack
sw $t1, 0($sp)       # First In Parameter “A” at Mem[$sp]
sw $t2, 4($sp)       # Second In Parameter “B” at Mem[$sp+4]
sw $t3, 8($sp)       # Third In Parameter “C” at Mem[$sp+8]
sw $ra, 20($sp)      # Save Return address
sw $s8, 24($sp)      # Save $t8 on the stack
sw $s9, 28($sp)      # Save $t9 on the stack
jal JILL             # call the Function
lw $t8, 24($sp)      # Restore $t8 from the stack
lw $t9, 28($sp)      # Restore $t9 from the stack
lw $ra, 20($sp)      # Restore Return Address to Main Program
lw $t4, 12($sp)      # Get First Out Parameter “D” at Mem[$sp+12]
lw $t5, 16($sp)      # Get Second Out Parameter “E” at Mem[$sp+16]
addiu $sp, $sp, 32   # De-allocate Space on the Stack
```

Passing Arguments on the Stack

An Example of Jack calling Jill(A, B, C, D, E)

```assembly
addiu $sp, $sp, -24  # Allocate Space on the Stack
sw $t1, 0($sp)       # First In Parameter “A” at Mem[$sp]
sw $t2, 4($sp)       # Second In Parameter “B” at Mem[$sp+4]
sw $t3, 8($sp)       # Third In Parameter “C” at Mem[$sp+8]
sw $ra, 20($sp)      # Save Return address
jal JILL             # Call the Function
lw $ra, 20($sp)      # Restore Return Address to Main Program
lw $t4, 12($sp)      # Get First Out Parameter “D” at Mem[$sp+12]
lw $t5, 16($sp)      # Get Second Out Parameter “E” at Mem[$sp+16]
addiu $sp, $sp, 24   # De-allocate Space on the Stack
```

What if Jill Needs Additional Local Variables?

```assembly
addiu $sp, $sp, -16  # Allocate Space for a Temporary array
move $a0, $sp        # Initialize a pointer in $a0 to the array
lw $sp               # Use the array on the stack
addiu $sp, $sp, 16   # Deallocate Temporary Space
```

Example of Jill accessing the Stack

```
JILL:
lw $a0, 0($sp)       # Get First In Parameter “A” at Mem[$sp]
lw $a1, 4($sp)       # Get Second In Parameter “B” at Mem[$sp+4]
lw $a2, 8($sp)       # Get Third In Parameter “C” at Mem[$sp+8]
...                   # Body of Function
...                   # ...
sw $v0, 12($sp)      # First Out Parameter “D” at Mem[$sp+12]
sw $v1, 16($sp)      # Second Out Parameter “E” at Mem[$sp+16]
jr $ra               # Return to JACK
```
Exercise 6.1 (MIPS Assembly Language)

# An Example of calling the function MinMax (&X, N, Min, Max)

```
.data
array .space 400
.text
addiu $sp, $sp, -16
la $t0, array
sw $t0, 0($sp)
li $t0, 100
sw $t0, 4($sp)
jal MinMax
lw $t0, 8($sp)
lw $t1, 12($sp)
addiu $sp, $sp, 16
```

Exercise 6.1 (Pseudocode)

MinMax(&X: $t1, N: $t2, min: $t8, max: $t9)

$\text{\$t1} = \text{Mem}(\$sp);  
$\text{\$t2} = \text{Mem}(\$sp + 4);  
$\text{\$t8} = \text{Mem}(\$t1);  
$\text{\$t9} = \$t8;  
$\text{\$t2} = \$t2 - 1;  
\text{While (}\$t2 > 0\text{)}  
\{  
$\text{\$t1} = \$t1 + 4;  
$\text{\$t0} = \text{Mem}(\$t1);  
\text{if (}\$t0 < \$t8\text{)} \$t8 = \$t0;  
\text{else if (}\$t0 > \$t9\text{)} \$t9 = \$t0;  
$\text{\$t2} = \$t2 - 1;  
\}
\text{Mem}(\$sp + 8) = \$t8;  
\text{Mem}(\$sp + 12) = \$t9;
```

Exercise 6.1

MinMax (&X, N, Min, Max)

Write a function to search through an array 'X' of 'N' words to find the minimum and maximum values.

The parameters &X and N are passed to the function on the stack, and the minimum and maximum values are returned on the stack. (Show how MinMax is called)
Exercise 6.2 (Pseudocode)

```plaintext
$t3= Mem(sp);  \quad # get &X
$t1= Mem($sp + 4);  \quad # get N
$t0= Mem($sp + 8);  \quad # get V
$t2= t1;
for ($t2 = $t2 - 1; $t2 >= 0; $t2= $t2 - 1)
    { $t4 = mem($t3);
      $t3= $t3 + 1;
      if ( $t4 == $t0) go to found;
    }
Mem(sp + 12) = -1;
go to exit;
found:
    Mem(sp + 12) = $t1 - $t2;
exit: return;
```

Exercise 6.1 MinMax($X:$t1, $N:$t2, min:$t8, max:$t9)

<table>
<thead>
<tr>
<th>Label</th>
<th>Opcode</th>
<th>Dest.</th>
<th>S1, S2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>MinMax:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw</td>
<td>$t1, 0($sp)</td>
<td># get &amp;X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw</td>
<td>$t2, 4($sp)</td>
<td># get N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw</td>
<td>$t8, 0($t1)</td>
<td># init. min</td>
<td></td>
<td></td>
</tr>
<tr>
<td>move</td>
<td>$t9, $t8</td>
<td># init. max</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addi</td>
<td>$t2, $t2, -1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>blez</td>
<td>$t2, ret</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>loop:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>addiu</td>
<td>$t1, $t1, 4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw</td>
<td>$t0, 0($t1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bgez</td>
<td>$t0, $t8, next</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>move</td>
<td>$t8, $t0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>chk</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>next:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ble</td>
<td>$t0, $t9, chk</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>move</td>
<td>$t9, $t0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>chk:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>addi</td>
<td>$t2, $t2, -1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bgtz</td>
<td>$t2, loop</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ret:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw</td>
<td>$t8, 8($sp)</td>
<td># put min</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw</td>
<td>$t9, 12($sp)</td>
<td># put max</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jr</td>
<td>$ra</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exercise 6.2 (MIPS Assembly Language)

<table>
<thead>
<tr>
<th>Label</th>
<th>Opcode</th>
<th>Dest.</th>
<th>S1, S2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>.text</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>search:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw</td>
<td>$t3, 0($sp)</td>
<td># get &amp;X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw</td>
<td>$t1, 4($sp)</td>
<td># get N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw</td>
<td>$t0, 8($sp)</td>
<td># get V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>move</td>
<td>$t2, $t1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>addi</td>
<td>$t2, $t2, -1 # t2 = N - 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>loop:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lbu</td>
<td>$t4, 0($t3) # get a character</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>addiu</td>
<td>$t3, $t3, 1 # increment pointer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>beq</td>
<td>$t4, $t0, found</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>addi</td>
<td>$t2, $t2, -1 # decrement loop counter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bgez</td>
<td>$t2, loop</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>li</td>
<td>$t4, -1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw</td>
<td>$t4, 12($sp)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>exit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>found:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sub</td>
<td>$t1, $t1, $t2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw</td>
<td>$t1, 12($sp)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>exit:</td>
<td>jr $ra</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exercise 6.2

Search($X, N, V, L)

Write a function to sequentially search an array $X$ of $N$ bytes for the relative location $L$ of a value $V$.

The parameters $&X$, $N$, and $V$ are passed to the procedure on the stack, and the relative location $L$ (a number ranging from 1 to $N$) is returned on the stack.

If the value $V$ is not found, the value (-1) is returned for $L$. 
Exercise 6.3 (Pseudocode)
Scan(&X:$t6, N:$t2, U:$t3, L:$t4, D:$t5)
$t6 = Mem(sp)  # &X
$t2 = Mem(sp+4)  # N
$t3=$t4=$t5=0;
For (; $t2>0; $t2=$t2-1) {
  $t1 = mem($t6)  # get a byte
  $t6 = $t6 + 1
  if ( $t1 >= 65 && $t1 <= 90 )  $t3 = $t3+1;
  else if ( $t1 >= 97 && $t1 <= 122) $t4=$t4+1;
  else if ( $t1 >= 48 && $t1 <= 57 ) $t5=$t5+1;
}    
Mem(sp + 8 ) = $t3;
Mem(sp + 12 ) = $t4;
Mem(sp + 16 ) = $t5;
return;

Exercise 6.3
Scan(&X, N, U, L, D)
Write a function to scan an array 'X' of 'N' bytes counting how many bytes are ASCII codes for:
a. upper case letters - U
b. lower case letters - L
c. decimal digits - D
Return the counts on the stack. The address of the array and the number of bytes N will be passed to the function on the stack.
Write a short main program to test this function.

Exercise 6.3 (Assembly Language Initialize)
Scan(&X:$t6, N:$t2, U:$t3, L:$t4, D:$t5)

<table>
<thead>
<tr>
<th>Label</th>
<th>Op-Code</th>
<th>Dest</th>
<th>S1, S2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td>$t6, 0($sp)</td>
<td></td>
<td></td>
<td>Get &amp;X</td>
</tr>
<tr>
<td>lw</td>
<td>$t2, 4($sp)</td>
<td></td>
<td></td>
<td>Get Value N</td>
</tr>
<tr>
<td>li</td>
<td>$t3, 0</td>
<td></td>
<td></td>
<td>Count of Upper Case</td>
</tr>
<tr>
<td>li</td>
<td>$t4, 0</td>
<td></td>
<td></td>
<td>Count of Lower Case</td>
</tr>
<tr>
<td>li</td>
<td>$t5, 0</td>
<td></td>
<td></td>
<td>Count of Decimal Digits</td>
</tr>
<tr>
<td>blez</td>
<td>$t2, done</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>li</td>
<td>$t0, 48</td>
<td></td>
<td></td>
<td>ASCII “0”</td>
</tr>
<tr>
<td>li</td>
<td>$t9, 57</td>
<td></td>
<td></td>
<td>ASCII “9”</td>
</tr>
<tr>
<td>li</td>
<td>$t7, 97</td>
<td></td>
<td></td>
<td>ASCII “a”</td>
</tr>
<tr>
<td>li</td>
<td>$t8, 122</td>
<td></td>
<td></td>
<td>ASCII “z”</td>
</tr>
<tr>
<td>addiu</td>
<td>$sp, $sp,-8</td>
<td></td>
<td></td>
<td>Allocate Temp Space</td>
</tr>
<tr>
<td>sw</td>
<td>$s6, 0($sp)</td>
<td></td>
<td></td>
<td>Save s6</td>
</tr>
<tr>
<td>sw</td>
<td>$s7, 4($sp)</td>
<td></td>
<td></td>
<td>Save s7</td>
</tr>
<tr>
<td>li</td>
<td>$s6, 65</td>
<td></td>
<td></td>
<td>ASCII “A”</td>
</tr>
<tr>
<td>li</td>
<td>$s7, 90</td>
<td></td>
<td></td>
<td>ASCII “Z”</td>
</tr>
</tbody>
</table>

A Main Program to Test the Scan Function

```assembly
### Scan(&X, N, U, L, D) SASL
.data
string:.asciiz "The Quick Fox 0123456789"
.text
main:       ----
    addiu $sp, $sp, -20  # Allocate
    la $t0, string
    sw $t0, 0($sp)
    li $t0, 24
    sw $t0, 4($sp)
    jal Scan
    lw $t0, 8($sp)
    lw $t1, 12($sp)
    lw $t2, 16($sp)
    addi $sp, $sp, 20  # Deallocate
```
Exercise 6.4

AVA (&X, &Y, &Z, N, status)

Write a function to perform an absolute value vector addition. Use the stack to pass parameters. The parameters are: the starting address of three different word arrays: X, Y, Z, and an integer value N specifying the size of the vectors.

If overflow ever occurs when executing this function, an error status of “1” should be returned and the function aborts any further processing. Otherwise, return the value “0” for status. The function will perform the vector addition:

\[ X_i = |Y_i| + |Z_i| \; \text{with} \; i \; \text{going from 0 to N - 1.} \]

Also write a main program to test this function.

Exercise 6.3 (Assembly Language Body)

<table>
<thead>
<tr>
<th>Label</th>
<th>Op-Code</th>
<th>Dest, S1, S2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>loop</td>
<td>lbu</td>
<td>$t1, 0($t6)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>addi</td>
<td>$t6, $t6, 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>blt</td>
<td>$t1, $s6, num</td>
<td># “A”</td>
</tr>
<tr>
<td></td>
<td>bgt</td>
<td>$t1, $s7, lowc</td>
<td># “Z”</td>
</tr>
<tr>
<td></td>
<td>addi</td>
<td>$t3, $t3, 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>check</td>
<td></td>
</tr>
<tr>
<td>lowc</td>
<td>blt</td>
<td>$t1, $t7, check</td>
<td># “a”</td>
</tr>
<tr>
<td></td>
<td>bgt</td>
<td>$t1, $t8, check</td>
<td># “z”</td>
</tr>
<tr>
<td></td>
<td>addi</td>
<td>$t4, $t4, 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>check</td>
<td></td>
</tr>
<tr>
<td>num</td>
<td>blt</td>
<td>$t1, $t0, check</td>
<td># “0”</td>
</tr>
<tr>
<td></td>
<td>bgt</td>
<td>$t1, $t9, check</td>
<td># “9”</td>
</tr>
<tr>
<td></td>
<td>addi</td>
<td>$t5, $t5, 1</td>
<td></td>
</tr>
<tr>
<td>check</td>
<td>addi</td>
<td>$t2, $t2, -1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>bgtz</td>
<td>$t2, loop</td>
<td></td>
</tr>
</tbody>
</table>

Example code for testing the AVA function

<table>
<thead>
<tr>
<th>Label</th>
<th>Op-Code</th>
<th>Dest, S1, S2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>.data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>zig</td>
<td>.space</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>zag</td>
<td>.word</td>
<td>345, 765, 234567, 2345, 999</td>
<td></td>
</tr>
<tr>
<td>zonk</td>
<td>.word</td>
<td>-39845, 765987, 67, 3215, 444</td>
<td></td>
</tr>
<tr>
<td>msg</td>
<td>.asciz</td>
<td>“Overflow Occurred”</td>
<td># AVA (&amp;X, &amp;Y, &amp;Z, N, status)</td>
</tr>
<tr>
<td>main</td>
<td>addi</td>
<td>$sp, $sp, -20</td>
<td># Allocate</td>
</tr>
<tr>
<td></td>
<td>la</td>
<td>$s0, zig</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sw</td>
<td>$s0, 0($sp)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>la</td>
<td>$s0, zag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sw</td>
<td>$s0, 4($sp)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>la</td>
<td>$s0, zonk</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sw</td>
<td>$s0, 8($sp)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>li</td>
<td>$s0, 5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sw</td>
<td>$s0, 12($sp)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>jai</td>
<td>AVA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>lw</td>
<td>$s0, 16($sp)</td>
<td># Deallocate</td>
</tr>
<tr>
<td></td>
<td>addi</td>
<td>$sp, $sp, 20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>beqz</td>
<td>$s0, done</td>
<td></td>
</tr>
<tr>
<td></td>
<td>li</td>
<td>$v0, 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>la</td>
<td>$a0, msg</td>
<td></td>
</tr>
<tr>
<td></td>
<td>syscall</td>
<td></td>
<td></td>
</tr>
<tr>
<td>done</td>
<td>li</td>
<td>$v0, 10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>syscall</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exercise 6.3 (Assembly Language)

lw   $s6, 0($sp) # Restore s register values
lw   $s7, 4($sp)
addi $sp, $sp, 8 # De-allocate Temp space
sw   $t3, 8($sp) # U
sw   $t4, 12($sp) # L
sw   $t5, 16($sp) # D
jr   $ra
Exercise 6.4 (Pseudocode)

AVA(\&X: $t6, \&Y: $t7, \&Z: $t8, N: $t0, status)

$ t6 = Mem($sp);
$ t7 = Mem($sp+4);
$ t8 = Mem($sp+8);
$ t0 = Mem($sp+12);
for ( ; $t0 > 0 ; $t0 = $t0 - 1 )
{ $t1 = Mem($t7);
  $t7 = $t7 + 4;
  if ($t1 < 0) $t1 = 0 - $t1;
  $t2 = Mem($t8);
  $t8 = $t8 + 4;
  if ($t2 < 0) $t2 = 0 - $t2;
  $t1 = $t1 + $t2;
  if ($t1 < 0) go to ovf;
  Mem($t6) = $t1;
  $t6 = $t6 + 4;
}
Mem($sp+16) = 0;
return
ovf: Mem($sp+16) = 1; return

Exercise 6.5

Fibonacci (N, E)
Write a function to return the Nth element in the Fibonacci sequence. A value N is passed to the function on the stack, and the Nth Fibonacci number E is returned on the stack.

If N is greater than 46 overflow will occur, so return a value of 0 if N is greater than 46. Also show an example of calling this function to return the 10th element in the sequence.

The first few numbers in the Fibonacci sequence are: 0, 1, 1, 2, 3, 5, ...
Exercise 6.5 (Fibonacci Assembly Language)

<table>
<thead>
<tr>
<th>Label</th>
<th>Op-Code</th>
<th>Dest.</th>
<th>S1, S2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fib</td>
<td>lw</td>
<td>$t0, 0($sp)</td>
<td></td>
<td># Get N</td>
</tr>
<tr>
<td></td>
<td>bltz</td>
<td>$t0, error</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>addi</td>
<td>$t1, $t0, -46</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>bgtz</td>
<td>$t1, error</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>li</td>
<td>$t1, 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>li</td>
<td>$t2, 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>move</td>
<td>$t3, $t0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>addi</td>
<td>$t0, $t0, -1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>loop</td>
<td>biez</td>
<td>$t0, done</td>
<td></td>
<td></td>
</tr>
<tr>
<td>done</td>
<td>add</td>
<td>$t3, $t2, $t1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>move</td>
<td>$t1, $t2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>move</td>
<td>$t2, $t3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>addi</td>
<td>$t0, $t0, -1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>bgtz</td>
<td>$t0, loop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>error</td>
<td>sw</td>
<td>$t3, 4($sp)</td>
<td></td>
<td># Return Nth Fibonacci number</td>
</tr>
<tr>
<td></td>
<td>jr</td>
<td>$ra</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sw</td>
<td>$0, 4($sp)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>jr</td>
<td>$ra</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example Code to Test Fibonacci

<table>
<thead>
<tr>
<th>Label</th>
<th>Op-Code</th>
<th>Dest.</th>
<th>S1, S2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>.data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>msg</td>
<td></td>
<td></td>
<td></td>
<td>“Can not Compute Correct Result”</td>
</tr>
<tr>
<td>.text</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>main</td>
<td>addi</td>
<td>$sp, $sp, -8</td>
<td></td>
<td># Allocate</td>
</tr>
<tr>
<td></td>
<td>li</td>
<td>$t0, 10</td>
<td></td>
<td># Pass argument to Fib</td>
</tr>
<tr>
<td></td>
<td>sw</td>
<td>$t0, 0($sp)</td>
<td></td>
<td># Call Fibonacci</td>
</tr>
<tr>
<td></td>
<td>jal</td>
<td>Fib</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>lw</td>
<td>$a0, 4($sp)</td>
<td></td>
<td># Get result back</td>
</tr>
<tr>
<td></td>
<td>addi</td>
<td>$sp, $sp, 8</td>
<td></td>
<td># Deallocate</td>
</tr>
<tr>
<td></td>
<td>bgtz</td>
<td>$a0, done</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>li</td>
<td>$v0, 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>la</td>
<td>$a0, msg</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>syscall</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>done</td>
<td>li</td>
<td>$v0, 1</td>
<td></td>
<td># Print result</td>
</tr>
<tr>
<td></td>
<td>syscall</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>error</td>
<td>li</td>
<td>$v0, 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>syscall</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fibonacci (A Very Efficient Method)

```
.data
fibnum: .word 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229, 832040, 1346269, 2178309, 3524578, 5702887, 9277465, 14930352, 24157817, 39088169, 63245986, 102334155, 165580414, 267914296, 433494437, 701408733, 1134903170, 1836331903
.text
fib:
    lw      $t0, 0($sp)
    bltz    $t0, error
    addi    $t1, $t0, -46
    bgtz    $t1, error
    slt     $t0, $t0, 2
    la      $t1, fibnum
    addi    $t0, $t1, $t0
    lw      $t0, 0($t0)
    sw      $t0, 4($sp)
    jr      $ra
error:
    sw      $0, 4($sp)
    jr      $ra
```

Exercise 6.5 (Pseudocode)

```pseudocode
$\text{t0} = \text{Mem}($sp$);
if ($\text{t0} > 46$) {
    \{\text{Mem}($\text{sp} + 4$) = 0; Return;\}
    \{
        \text{If ($\text{t0} > 1$)}
        \{
            \{\text{\$t1 = 0; \$t2 = 1; \$t3 = \$t0;}\}
            \{\text{For ($\text{t0} = \text{t0} - 1$; \text{t0} > 0$; \text{t0} = \text{t0} - 1$)}\}
            \{\text{\$t3 = \$t2 + \$t1; \$t1 = \$t2; \$t2 = \$t3}\}
            \}
        \}
    \}
else \text{\$t3 = \$t0; Mem($sp+4) = \$t3; \text{Return}}
```
Exercise 6.6 (Assembly Language)

BubSort ($X, N$)
Write a function to sort an array of $N$ words into ascending order using the bubble sort algorithm.

The address of the array and the value $N$ will be passed to the function on the stack.

Show how the sort function is called.

Example Assembly Language Code to Call Sort($Z, 1000$)

```
addi $sp, $sp, -8
li $t0, $z
sw $t0, 0($sp)
li $t0, 1000
sw $t0, 4($sp)
jal sort
addi $sp, $sp, 8
```

Exercise 6.6 (Pseudocode)

RipSort ($X, N$)
Write a function to sort an array of $N$ words into ascending order using the ripple sort algorithm.

The address of the array and the value $N$ will be passed to the function on the stack.

BubSort ($X; t3, N; t0$)

```plaintext
$t0 = \text{Mem}($sp$+4)$;
Again:
$t0 = t0 - 1$;
$t2 = 0$;
$t3 = \text{Mem}($sp$)$;
For ($t1 = t0; t1 > 0; t1 = t1 - 1$)
  {If ( \text{Mem}($t3$) > \text{Mem}($t3+4$) ) then
    {exchange \text{Mem}($t3$) & \text{Mem}($t3+4$)
    $t2 = 1$}
  $t3 = t3 + 4$;
} If ($t2 == 1$) go to Again
else
return
```
MIPS Assembly Language Programming
Bob Britton
Chapter 6 – d
Function Calls Using the Stack

Exercise 6.7 (Pseudocode)

Function Ripsort (&X:$t3, N:$t0);
$t0= Mem($sp+4);
$t3= Mem($sp);
For ($t0 = $t0 - 1; $t0 > 0; $t0 = $t0 -1)
  {$t8 = Mem($t3);
   $t3 = $t3 + 4;
   $t4 = $t3;
   For ($t5 = $t0; $t5 > 0; $t5 = $t5 -1)
     {$t9 = Mem($t4);
      $t4 = $t4 + 4;
      if ($t8 > $t9)
        {Mem($t3 - 4) = $t9;
         Mem($t4 - 4) = $t8;}
     }
  }
return

Exercise 6.8
Write your most efficient MIPS assembly language code translation for the following function and main line calling program.
Note, all communication with the function must use a stack frame.
Make use of the fact that multiplication and division by powers of 2 can be performed most efficiently by shifting.

```c
void chico (int *X, int Y, int Z )
{X = Y / 4 - Z * 10 + *X * 8 ;}

int main()
{int J, K, L, M ;
cin >> J, K, L;
chico ( &J, K, L);
M = J - ( K + L);
cout << M;
return 0 }
```

Exercise 6.7 (Assembly Language)

<table>
<thead>
<tr>
<th>Label</th>
<th>Op-Code</th>
<th>Dest, S1, S2</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ripsort:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw</td>
<td>$t0, 4($sp)</td>
<td># Get N</td>
<td></td>
</tr>
<tr>
<td>lw</td>
<td>$t3, 0($sp)</td>
<td># Get &amp;X</td>
<td></td>
</tr>
<tr>
<td>addi</td>
<td>$t0, $t0, -1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>loop1:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw</td>
<td>$t8, 0($t3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addi</td>
<td>$t3, $t3, 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>move</td>
<td>$t4, $t3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>move</td>
<td>$t5, $t0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>loop2:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw</td>
<td>$t9, 0($t4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>addi</td>
<td>$t4, $t4, 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ble</td>
<td>$t9, $t9, check</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw</td>
<td>$t9, -4($t3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sw</td>
<td>$t8, -4($t4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>check:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>addi</td>
<td>$t5, $t5, -1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bgtz</td>
<td>$t5, loop2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>next:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>addi</td>
<td>$t0, $t0, -1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bgtz</td>
<td>$t0, loop1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jr</td>
<td>$ra</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Exercise 6.9

**MUL32 (m, n, p, f)**

Write a function MUL32(m, n, p, f) that will find the 32-bit product "p" of two arguments m and n.

If the two's complement representation of the product cannot be represented with 32 bits, then the error flag "f" should be set to 1 otherwise the error flag is set to 0.

Pass all arguments on the stack.

Exercise 6.8 (Assembly Language)

```
.text

.chico:
# Function

lw   $t0, 0($sp)    # Get address of X
lw   $t1, 0($t0)   # Get value of X
lw   $t2, 4($sp)   # Get value of Y
lw   $t3, 8($sp)   # Get value of Z
sra  $t2, $t2, 2   # Y/4
sll  $t4, $t3, 1   # Z*2
sll  $t5, $t3, 3   # Z*8
add  $t3, $t4, $t5 # Z*10 = Z*2 + Z*8 = Z*(2*8)
sub  $t1, $t1, $t5 # X*8
add  $t4, $t2, $t3
sw   $t4, 0($t0)   # X = Result
jr   $ra
```

Exercise 6.9 (Assembly Language)

```
Label   | Op-Code | Dest, S1, S2 | Comments
-------|---------|--------------|----------
Mul32:  |         |              |          
        | lw      | $t3, 0($sp)  | # Get m  
lw      | lw      | $t4, 4($sp)  | # Get n  
mult    | mult    | $t3, $t4    | # Get correct sign for prod 
xor     | xor     | $t5, $t3, $t4| # $t5 = 32 bit product 
mflo    | mflo    | $t6         | # compare signs 
xor     | xor     | $t7, $t5, $t6| # $t8 = Upper 32-bits of prod 
bltz    | bltz    | $t7, ovf    | # Check if all upper bits are 1 
mtfi     | mtfi    | $t8         | # $t8 = Upper 32-bits of prod 
bgez    | bgez    | $t5, pos    | # Check if all upper bits are 1 
addui   | addui   | $t8, $t8, 1 | # Check if all upper bits are 1 
pos:    | bnez    | $t8, ovf    | # overflow 
        | sw      | $t6, 8($sp) | # product 
        | sw      | $0, 12($sp) | # ok 
        | jr       | $ra        | # occurred 
        | li       | $t8, 1     | # overflow 
        | sw      | $t8, 12($sp)| # occurred 
        | jr       | $ra        | # overflow 
```

Exercise 6.8 (main)

```
.data

J: .space 4
K: .space 4
L: .space 4
M: .space 4

.text

main:
# cin >> J, K, L;
la   $s0, J
lw   $s2, 4($s0)  # Get value of K
sw   $s2, 4($sp)  # Pass value of K
lw   $s3, 4($s0)  # Get value of L
sw   $s3, 4($sp)  # Pass value of L
jal  chico
```
Exercise 6.10

Dynamic String Allocation
Please refer to the example in Section 6.8 of the textbook