**Homework 9: Compilers**

**CS 200 • 100 Points Total  
Due Wednesday, November 22, 2023**

**Assignment**

This one is fairly tough. It’s almost like a project. For this one, you are the compiler. You need to convert the C program given below into a MIPS program. You’ll want to use MARS and test the code you write to be sure it works correctly. This one problem demonstrates all of Section 9 and is worth the entire 100 points.

Here are the things I’m looking for in your solution:

* There are no global variables other than I/O strings. There are a few number constants (-1, 1, 2, and 20) which can be handled with instructions that use immediate values.
* You should be able to do a while loop and the if/else constructs in assembly. I presented them a couple of sections ago and the text details them in Section 9.
* The subroutine call to fib must pass values on the stack. In fact, because it is recursive, a stack frame must be used. See my suggested layout for the stack frame.
* Local variables for fib are required (result and temp). Don’t just use registers because recursive calls to fib will alter those registers and your code won’t work correctly. Again, see my suggested layout for the stack frame.
* If you use a register in main to hold ‘number’, you should preserve it before calling fib and restore it after fib returns.
* If you use registers in fib, you’ll use them in recursive calls, so preserve them in the stack frame. I suggest you use the ‘T’ registers, so you don’t need to worry about the ‘S’ registers.

Suggested stack frame:

|  |  |  |  |
| --- | --- | --- | --- |
| Pointers | Usage | $sp offset | $fp offset |
| stack pointer | temp | 0 | ? |
|  | result | 4 | ? |
|  | any registers | 8 ꜜ | -20 ꜛ |
|  | original $fp | ? | -16 |
|  | $ra | ? | -12 |
|  | number | ? | -8 |
|  | return | ? | -4 |
| frame pointer |  |  |  |

Depending on how many registers you use in fib, the stack frame size may vary, which is why $sp offsets below the registers or $fp offsets above the registers cannot be determined. Once you determine which registers you will use, you can finish the stack frame diagram.

The C program is on the following page.

#include <stdio.h> // not necessary for the assembly; you’ll use system calls

int fib(int); // forward declaration. Again, no assembly equivalent.

void main(void)  
{  
 int number = -1;

while ((number < 1) || (number > 20))  
{  
 printf("Enter a number (1 - 20): ");  
 scanf("%d", &number);

if (number < 1)  
 {  
 printf("Number must be > 0, try again.\n");  
 }

if (number > 20)  
 {  
 printf("Number must be < 21, try again.\n");  
 }  
 }

int value = fib(number);  
 printf("The fibonacci value is: %d\n", value);  
}

int fib(int n)  
{  
 int result;  
 int temp;

if (n < 2)  
 {  
 result = n;  
 }  
 else  
 {  
 temp = fib(n-2);  
 result = fib(n-1) + temp;  
 }

return result;  
}

.data

prompt\_low: .asciiz "Enter low: "

prompt\_high: .asciiz "Enter high: "

prompt\_seed: .asciiz "Enter seed (must be a large prime number): "

prompt\_count: .asciiz "Enter count of random numbers to generate: "

output\_title: .asciiz "Random Numbers:\n"

.text

.globl main

main:

# Prompt for low

li $v0, 4

la $a0, prompt\_low

syscall

# Read low

li $v0, 5

syscall

move $s0, $v0 # $s0 = low

# Prompt for high

li $v0, 4

la $a0, prompt\_high

syscall

# Read high

li $v0, 5

syscall

move $s1, $v0 # $s1 = high

# Prompt for seed

li $v0, 4

la $a0, prompt\_seed

syscall

# Read seed

li $v0, 5

syscall

move $s2, $v0 # $s2 = seed

# Prompt for count

li $v0, 4

la $a0, prompt\_count

syscall

# Read count

li $v0, 5

syscall

move $s3, $v0 # $s3 = count

# Display output title

li $v0, 4

la $a0, output\_title

syscall

# Loop to generate and print random numbers

li $t0, 0 # Loop counter

li $t1, 1103515245 # Multiplier (a large prime number)

li $t2, 12345 # Increment

li $t3, 65536 # Modulus (2^16)

generate\_random:

# Calculate next seed using LCG

multu $t1, $s2

addu $s2, $s2, $t2

mfhi $s2

# Generate pseudo-random number

move $t4, $s2 # $t4 = seed

divu $t4, $s3, $t4 # $t4 = seed % (high - low + 1)

addu $t4, $t4, $s0 # $t4 = scaled random number in [low, high]

# Print random number

li $v0, 1

move $a0, $t4

syscall

# Print newline

li $v0, 4

la $a0, "\n"

syscall

addi $t0, $t0, 1 # Increment loop counter

blt $t0, $s3, generate\_random # Loop until count is reached

# Exit

li $v0, 10

syscall