

Lab 5 : Linking

Name:

Sign the following statement:

On my honor, as an Aggie, I have neither given nor received unauthorized aid on this academic work

1 Objective

The main objective of this lab is to experiment with linking two source files together using MARS and the gcc compiler.

2 Pre-requisite

For this lab you are expected to be familiar with relocation entries and compiler operation.

3 MIPS Linking

4 Linking files using MARS

1. Type the following program in as a file called *lab5-part1.s* and load it into MARS. Do not run it yet.

```

1          .data
2          .align 2
3          .text
4          main:
5          # print out prompt
6          li $v0, 4          # system call code for printing string = 4
7          la $a0, in_string  # load address of string to be printed into $a0
8          syscall           # call operating system to perform print operation
9
10         # read integer into $s0
11         li $v0, 5          # system call code for read integer = 5
12         syscall           # call operating system
13         move $s0, $v0      # value read from keyboard returned in register $v0
14                           # transfer to $s0
15
16         sw $s0,($sp)       # push argument for Fib on stack
17         addi $sp,$sp,-4    # and decrement stack pointer
18         jal Fib           # jump to subroutine
19         addi $sp,$sp,4     # increment stack pointer
20         lw $s1,($sp)      # and pop result from stack
21
22         # print out prompt
23         li $v0, 4          # system call code for printing string = 4
24         la $a0, out_string # load address of string to be printed into $a0
25         syscall           # call operating system
26
27         # print out result (stored in $s1)
28         li $v0, 1          # system call code for printing integer = 1
29         move $a0, $s1      # move integer to be printed into $a0: $a0 = $s1
30         syscall           # call operating system to perform print
31         jr $ra

```

2. Provide the expected symbol table for the program. Which of the symbols are external?

3. Write the relocation table for the program. *Hint:* list all instructions that use absolute addresses.

4. Now, type in the following new program and save it as lab5-part2.s.

```

1      .data
2      in_string: .ascii "Input a positive integer:\n\n"
3      out_string: .ascii "The Fibonacci number is:\n\n"
4      .globl in_string # in_string is a global label
5      .globl out_string
6      .globl Fib
7      .align 2
8      .text
9      #####
10     # Fibonacci subroutine
11     # input: integer n, on stack
12     # output: Fib(n), nth Fibonacci number
13     # description: recursively computes Fib(n) = Fib(n-1) + Fib(n-2),
14     #               Fib(1) = Fib(2) = 1.
15     #
16     # uses: $t0, $t1
17     #####
18     Fib:
19     # procedure prologue:
20     sw $ra,($sp)      # save return address on stack, since recursive,
21     addi $sp,$sp,-4   # and decrement stack pointer
22     sw $fp,($sp)      # save previous frame pointer on stack
23     addi $sp,$sp,-4   # and decrement stack pointer
24     add $fp,$sp,12    # set frame pointer to point at base of stack frame
25     lw $t0,($fp)      # copy argument to $t0: $t0 = n
26     li $t1, 2
27     bgt $t0,$t1,do_recurse # if argument n >= 2, branch to recursive sequence
28     li $t0, 1         # else set result to 1
29                     # (base cases n = 1 and n = 2)
30     b epilogue        # branch to end
31     do_recurse:
32     addi $t0,$t0,-1    # $t0 = n-1
33     sw $t0,($sp)      # push argument n-1 on stack
34     addi $sp,$sp,-4   # and decrement stack pointer
35     jal Fib           # call Fibonacci with argument n-1
36     # leave result on stack for now
37     lw $t0,($fp)      # re-copy argument to $t0: $t0 = n
38     addi $t0,$t0,-2   # $t0 = n-2
39     sw $t0,($sp)      # push argument n-2 on stack

```

```

40      addi $sp,$sp,-4           # and decrement stack pointer
41      jal Fib                   # call Fibonacci with argument n-2
42      addi $sp,$sp,4           # increment stack pointer
43      lw $t0,($sp)             # and pop result of Fib(n-2) from stack into $t0
44      addi $sp,$sp,4           # increment stack pointer
45      lw $t1,($sp)             # and pop result of Fib(n-1) from stack into $t1
46      add $t0,$t0,$t1           # $t0 = Fib(n-2) + Fib(n-1); have result
47      epilogue:                # procedure epilogue: $t0 holds result
48      addi $sp,$sp,4           # increment stack pointer
49      lw $fp,($sp)             # and pop saved frame pointer into $fp
50      addi $sp,$sp,4           # increment stack pointer
51      lw $ra,($sp)             # and pop return address into $ra
52      addi $sp,$sp,4           # increment stack pointer
53      # to pop argument (n) from stack (discard)
54      sw $t0,($sp)             # push result onto stack
55      addi $sp,$sp,-4           # and decrement stack pointer
56      jr $ra                   # return to caller
57      #####
58      # end of Fibonacci #
59      #####

```

5. Provide the expected symbol table for the program. Which of the symbols are external?
6. Write the relocation table for the program. *Hint:* list all instructions that use absolute addresses.
7. Save Lab5-part1.s and lab5-part2.s into the same folder and make sure the option *Assemble all files in directory* is checked in *Settings*.

-
8. Describe all changes in the text and data segments after loading both files.

 9. Give symbol table for the combined program. Make sure the option *Show Label Window* is checked in *Settings*.

 10. Write the relocation table for combined program.

 11. Explain why the implementation given in files Lab5-part1.s and Lab5-part2.s is so inefficient.

5 Experiments with a real gcc compiler

The goal of this section is to experiment with linking two source files together.

You will use <http://dropzone.tamu.edu/350/> to perform the compilation and linking steps. Alternately, you may use a stand-alone MIPS cross compiler if you have one available.

Type the following program and save it as **Lab5-part3-1.c**. The code contains utility functions to perform IO operations:

```
char *prodMessage = "The product is ";
void print_int(int a)
{
    asm(      "li $2, 1\n\t"
        "syscall"
        : /*No outputs*/
        : "r"(a)
        : "%v0" );
}

void print_string(char *a)
{
    asm(      "li $2, 4\n\t"
        "syscall"
        : /*No outputs*/
        : "r"(a)
        : "%v0" );
}

int read_int()
{
    register unsigned long __v0 asm("$2");

    asm(      "li $2, 5\n\t"
        "syscall"
        : /*No outputs*/
        : /*No inputs*/
        : "%v0" );

    return __v0;
}
```

The asm blocks are simply a way to write assembly code within C. These functions simply set up registers for a syscall. If you examine the assembly, you will see that each is only 3 instructions long.

Type the code below and save it as **Lab5-part3-2.c**. The code below reads in two integers from the user, multiplies them together, and then prints the product to the screen.

```

extern char *prodMessage;
void print_int(int a);
void print_string(char *a);
int read_int();

int my_mul(int a, int b)
{
    int i, ret = 0;
    for(i=0; i<b; i++)
        ret = ret + a;
    return ret;
}

int main(void)
{
    print_string("Enter the first number");
    int num1 = read_int();
    print_string("Enter the second number");
    int num2 = read_int();
    print_string(prodMessage);
    print_int(my_mul(num1, num2));
    return 0;
}

```

This is the same program that you designed in lab 4. Compile the two files, and examine their object files. Look at the symbol tables for both files and fill in the following table. Write 'UND' for undefined symbols, and write 'N/A' for symbols not present in a particular file. Also include the section (.data, .text, or another section) for each symbol.

Symbol	Address in file 1	Address in file 2	Address in linked file	Section
print_int				
print_string				
read_int				
prodMessage				
my_mul				
main				

6 PIC Code

1. Open `funca` code (from the prelab) in MARS. Modify the code by adding few `nop` instruction in the beginning of the program. List places in code where the native instructions have changed and explain why the change occurred.
2. Take the code you modified in the prelab for `funca` to be position independent. Save your program as `lab5-part4.s`, and open it to make sure it is position independent in MARS. Try adding instructions before your code (such as `nop`) and make sure that program does not change.

7 Deliverables

Submit the following:

- A completed copy of this lab.
- All source code files created in this lab (with comments).