Lab 5 : Linking

Name:

Kraig Orcutt

Sign the following statement:

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

Kraig Orcutt

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.

```
                .data
                .align 2
        .text
        main:
        # print out prompt
        li $v0, 4               # system call code for printing string = 4
        la $a0, in_string       # load address of string to be printed into $a0
        syscall                 # call operating system to perform print operation

        # read integer into $s0
        li $v0, 5               # system call code for read integer = 5
        syscall                 # call operating system
        move $s0, $v0           # value read from keyboard returned in register $v0
                                # transfer to $s0

        sw $s0,($sp)            # push argument for Fib on stack
        addi $sp,$sp,-4         #   and decrement stack pointer
        jal Fib                 # jump to subroutine
        addi $sp,$sp,4          # increment stack pointer
        lw $s1,($sp)            #   and pop result from stack

        # print out prompt
        li $v0, 4               # system call code for printing string = 4
        la $a0, out_string      # load address of string to be printed into $a0
        syscall                 # call operating system

        # print out result (stored in $s1)
        li $v0, 1               # system call code for printing integer = 1
        move $a0, $s1           # move integer to be printed into $a0:  $a0 = $s1
        syscall                 # call operating system to perform print
        jr $ra
```

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

Label	addresss	
[main]	0x00400000	
[in_string]	0x00000000	external
[out_string]	0x00000000	external
[Fib]	0x00000000	external

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

Address	Instruction type
0x00400004	lui
0x00400008	ori
0x00000000	jal
0x00400034	lui
0x00400038	ori

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

```
.data
in_string:  .asciiz    "Input a positive integer:\n\n"
out_string: .asciiz    "The Fibonacci number is:\n\n"
.globl in_string # in_string is a global label
.globl out_string
.globl Fib
.align 2
.text
########################################################################
# Fibonacci subroutine
# input:  integer n, on stack
# output: Fib(n), nth Fibonacci number
# description: recursively computes   Fib(n) = Fib(n-1) + Fib(n-2),
#                                     Fib(1) = Fib(2) = 1.
#
# uses: $t0, $t1
########################################################################
Fib:
# procedure prologue:
sw $ra,($sp)            # save return address on stack, since recursive,
addi $sp,$sp,-4         #   and decrement stack pointer
sw $fp,($sp)            # save previous frame pointer on stack
addi $sp,$sp,-4         #   and decrement stack pointer
add $fp,$sp,12          # set frame pointer to point at base of stack frame
lw $t0,($fp)            # copy argument to $t0:  $t0 = n
li $t1, 2
bgt $t0,$t1,do_recurse  # if argument n >= 2, branch to recursive sequence
li $t0, 1               #   else set result to 1
                        #(base cases  n = 1 and  n = 2)
b epilogue              # branch to end
do_recurse:
addi $t0,$t0,-1         # $t0 = n-1
sw $t0,($sp)            # push argument n-1 on stack
addi $sp,$sp,-4         #   and decrement stack pointer
jal Fib                 # call Fibonacci with argument n-1
# leave result on stack for now
lw $t0,($fp)            # re-copy argument to $t0:  $t0 = n
addi $t0,$t0,-2         # $t0 = n-2
sw $t0,($sp)            # push argument n-2 on stack
```

```
        addi $sp,$sp,-4         #   and decrement stack pointer
        jal Fib                 # call Fibonacci with argument n-2
        addi $sp,$sp,4          # increment stack pointer
        lw $t0,($sp)            #   and pop result of Fib(n-2) from stack into $t0
        addi $sp,$sp,4          # increment stack pointer
        lw $t1,($sp)            #   and pop result of Fib(n-1) from stack into $t1
        add $t0,$t0,$t1         # $t0 = Fib(n-2) + Fib(n-1); have result
        epilogue:                   # procedure epilogue: $t0 holds result
        addi $sp,$sp,4          # increment stack pointer
        lw $fp,($sp)            #   and pop saved frame pointer into $fp
        addi $sp,$sp,4          # increment stack pointer
        lw $ra,($sp)            #   and pop return address into $ra
        addi $sp,$sp,4          # increment stack pointer
        #   to pop argument (n) from stack (discard)
        sw $t0,($sp)            # push result onto stack
        addi $sp,$sp,-4         #   and decrement stack pointer
        jr $ra                  # return to caller
        #####################
        # end of Fibonacci #
        #####################
```

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

Label	Address	
[main]	0x00000000	external
[Fib]	0x00400000	
[in_string]	0x10010000	
[out_string]	0x1001001c	
[do_recurse]	0x0040002c	
[epiogue]	0x00400064	

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

Address	Instruction type
0x00000038	jal
0x0000004c	jal

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.

.text has both codes from part 1 and part 2, but the addresses are changed in part 2. The .data segment has in_string and out_string from part 2.

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

Label	Address
(global)	
[Fib]	0x00400050
[in_string]	0x10010000
[out_string]	0x1001001c
lab5-part1.asm	
[main]	0x00400000

10. Write the relocation table for combined program.

Address	Instruction type
0x00400004	lui
0x00400008	ori
0x00400024	jal
0x00400034	lui
0x00400038	ori
0x00400088	jal

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

Fibbonacci is a recursive function, making it take up alot of space on the stack. It also counts from the bottom up.

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:

```c
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	00000000	UND	00400570	.text
print_string	00000010	UND	00400580	.text
read_int	00000020	UND	00400590	.text
prodMessage	00000000	UND	00410800	.data
my_mul	N/A	00000000	004005a0	.text
main	N/A	00000018	004005b8	.text

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.

   Three nop instructions adds 12 bytes to the absolute address of the loop, because an nop instruction takes up 1 word space. The immediate value of jal changed from 0x00400014 to 0x00400020.

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).