2.

Symbol	Address	External
Main	00400000	No
In_string	n/a	Yes
Out_string	n/a	Yes
Fib	n/a	Yes

3.

Address	Entry
0x00400004	lui $a0, upper(in_string)
0x00400008	ori $a0, lower(in_string)
0x00400024	jal fib
0x00400034	lui $a0, upper(out_string)
0x00400038	ori $a0, lower(out_string)

5.

Symbol	Address	External
epilogue	0x00400064	No
do_recurse	0x0040002c	No
in_string	0x10010000	Yes
out_string	0x1001000c	Yes
fib	0x00400000	Yes

6.

Address	Entry
0x00400038	Jal Fib
0x0040004c	Jal Fib

Bkpt	Address	Code	Basic	Source		
	0x00400000	0x24020004	addiu $2,$0,4	8:	li $v0, 4	# system call code for printing string = 4
	0x00400004	0x3c011001	lui $1,4097	9:	la $a0, in_string	# load address of string to be printed into $a0
	0x00400008	0x34240000	ori $4,$1,0			
	0x0040000c	0x0000000c	syscall	10:	syscall	# call operating system to perform print operation
	0x00400010	0x24020005	addiu $2,$0,5	13:	li $v0, 5	# system call code for read integer = 5
	0x00400014	0x0000000c	syscall	14:	syscall	# call operating system
	0x00400018	0x00028021	addu $16,$0,$2	15:	move $s0, $v0	# value read from keyboard returned in register $v0; transfer to $s0
	0x0040001c	0xafb00000	sw $16,0($29)	17:	sw $s0,($sp)	# push argument for Fib on stack
	0x00400020	0x23bdfffc	addi $29,$29,-4	18:	addi $sp,$sp,-4	# and decrement stack pointer
	0x00400024	0x0c100014	jal 0x00400050	19:	jal Fib	# jump to subroutine
	0x00400028	0x23bd0004	addi $29,$29,4	20:	addi $sp,$sp,4	# increment stack pointer
	0x0040002c	0x8fb10000	lw $17,0($29)	21:	lw $s1,($sp)	# and pop result from stack
	0x00400030	0x24020004	addiu $2,$0,4	24:	li $v0, 4	# system call code for printing string = 4
	0x00400034	0x3c011001	lui $1,4097	25:	la $a0, out_string	# load address of string to be printed into $a0
	0x00400038	0x3424001c	ori $4,$1,28			
	0x0040003c	0x0000000c	syscall	26:	syscall	# call operating system
	0x00400040	0x24020001	addiu $2,$0,1	29:	li $v0, 1	# system call code for printing integer = 1
	0x00400044	0x00112021	addu $4,$0,$17	30:	move $a0, $s1	# move integer to be printed into $a0: $a0 = $s1
	0x00400048	0x0000000c	syscall	31:	syscall	# call operating system to perform print
	0x0040004c	0x03e00008	jr $31	32:	jr $ra	
	0x00400050	0xafbf0000	sw $31,0($29)	24:	sw $ra,($sp)	# save return address on stack, since recursive,
	0x00400054	0x23bdfffc	addi $29,$29,-4	25:	addi $sp,$sp,-4	# and decrement stack pointer
	0x00400058	0xafbe0000	sw $30,0($29)	26:	sw $fp,($sp)	# save previous frame pointer on stack
	0x0040005c	0x23bdfffc	addi $29,$29,-4	27:	addi $sp,$sp,-4	# and decrement stack pointer
	0x00400060	0x23be000c	addi $30,$29,12	28:	add $fp,$sp,12	# set frame pointer to point at base of stack frame
	0x00400064	0x8fc80000	lw $8,0($30)	30:	lw $t0,($fp)	# copy argument to $t0: $t0 = n
	0x00400068	0x24090002	addiu $9,$0,2	31:	li $t1, 2	
	0x0040006c	0x0128082a	slt $1,$9,$8	32:	bgt $t0,$t1,do_recurse	# if argument n >= 2, branch to recursive sequence
	0x00400070	0x14200002	bne $1,$0,2			
	0x00400074	0x24080001	addiu $8,$0,1	33:	li $t0, 1	# else set result to 1 (base cases n = 1 and n = 2)
	0x00400078	0x0401000e	bgez $0,14	34:	b epilogue	# branch to end
	0x0040007c	0x2108ffff	addi $8,$8,-1	37:	addi $t0,$t0,-1	# $t0 = n-1
	0x00400080	0xafa80000	sw $8,0($29)	38:	sw $t0,($sp)	# push argument n-1 on stack
	0x00400084	0x23bdfffc	addi $29,$29,-4	39:	addi $sp,$sp,-4	# and decrement stack pointer
	0x00400088	0x0c100014	jal 0x00400050	40:	jal Fib	# call Fibonacci with argument n-1
	0x0040008c	0x8fc80000	lw $8,0($30)	42:	lw $t0,($fp)	# re-copy argument to $t0: $t0 = n
	0x00400090	0x2108fffe	addi $8,$8,-2	43:	addi $t0,$t0,-2	# $t0 = n-2
	0x00400094	0xafa80000	sw $8,0($29)	44:	sw $t0,($sp)	# push argument n-2 on stack
	0x00400098	0x23bdfffc	addi $29,$29,-4	45:	addi $sp,$sp,-4	# and decrement stack pointer
	0x0040009c	0x0c100014	jal 0x00400050	46:	jal Fib	# call Fibonacci with argument n-2
	0x004000a0	0x23bd0004	addi $29,$29,4	47:	addi $sp,$sp,4	# increment stack pointer
	0x004000a4	0x8fa80000	lw $8,0($29)	48:	lw $t0,($sp)	# and pop result of Fib(n-2) from stack into $t0
	0x004000a8	0x23bd0004	addi $29,$29,4	49:	addi $sp,$sp,4	# increment stack pointer
	0x004000ac	0x8fa90000	lw $9,0($29)	50:	lw $t1,($sp)	# and pop result of Fib(n-1) from stack into $t1
	0x004000b0	0x01094020	add $8,$8,$9	51:	add $t0,$t0,$t1	# $t0 = Fib(n-2) + Fib(n-1); have result
	0x004000b4	0x23bd0004	addi $29,$29,4	54:	addi $sp,$sp,4	# increment stack pointer
	0x004000b8	0x8fbe0000	lw $30,0($29)	55:	lw $fp,($sp)	# and pop saved frame pointer into $fp
	0x004000bc	0x23bd0004	addi $29,$29,4	56:	addi $sp,$sp,4	# increment stack pointer
	0x004000c0	0x8fbf0000	lw $31,0($29)	57:	lw $ra,($sp)	# and pop return address into $ra
	0x004000c4	0x23bd0004	addi $29,$29,4	58:	addi $sp,$sp,4	# increment stack pointer
	0x004000c8	0xafa80000	sw $8,0($29)	60:	sw $t0,($sp)	# push result onto stack
	0x004000cc	0x23bdfffc	addi $29,$29,-4	61:	addi $sp,$sp,-4	# and decrement stack pointer

8.Those lines were the only thing that changed. What the assembler does is it concatenates the two programs together to create a new program. The first part comes from lab5part1 and the second part goes after that and thus all the addresses of the lab5-part2's code gets offset by the length(part1)*4. The assembler also does this with the data section by combining the two data sections, but in lab5-part1.asm there's nothing in there so the data section remains the same.

9.

Labels

Label	Address ▲
(global)	
Fib	0x00400050
in_string	0x10010000
out_string	0x1001001c
lab5-part1.asm	
main	0x00400000
lab5-part2.asm	
do_recurse	0x0040007c
epilogue	0x004000b4

10.

Address	Entry
00400004	Lui $1, 4097
00400008	Ori $4,$1,0
00400024	Jal Fib
00400088	Jal Fib
0040009c	Jal Fib
00400034	Lui $1, 4097
00400038	Ori $4,$1,28

11.

The two files could've been combined into one file. The makes it so that the linker doesn't have to do any work to link the files together and make it so that the two files are not interdependent.

12.

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	UND	UND	00410800	data
my_mul	N/A	00000000	004005a0	text
main	N/A	00000018	004005b8	text

1.

The only instruction that changed was the j loop instruction. The native instruction changed because the nop instructions at the beginning offset the rest of the program because they took up addresses and thus the jump loop had to change to reflect the change in address of the jump. The la $t0, n didn't change because the nop instructions didn't affect the static data of the program.