CSCE 312 – Lab 5 Report

Texas A&M University

March 22, 2024

Poyi Ou

Problem 1

Assembly Code for case a:

```
irmovq $0x200, %rsp             #set stack pointer
call main                       #compile main

main:
        irmovq $0x01, %rdi      #i stored in %rdi, setting i = 1
        irmovq $0x02, %rsi      #j stored in %rsi, setting j = 2

        rrmovq %rdi, %rdx       #temp val stored in %rdx, temp = i
        subq %rsi, %rdx         #temp = temp(i) - j
        jg L4                   #jump if i > j

        irmovq $0x07, %rdi      #set i = 7
        irmovq $0x01, %rdx      #temp = 1
        addq %rdx, %rsi         #j++
        jmp end                 #jump to end

L4:
        irmovq $0x03, %rdx      #temp = 3
        subq %rdx, %rdi         #i = i - temp

        irmovq $0x04, %rdx      #temp = 4
        addq %rdx, %rsi         #j = j + temp


end:
        pushq %rdi              #push i to stack
        pushq %rsi              #push j to stack
        popq %rdi               #pop i
        popq %rsi               #pop j
        halt
```

***Note: Register of i → %rdi, Register of j → %rsi**

Results:

Case a → i = 1, j = 2; Output: i = 7, j = 3

```
[pou14@linux2 y86-code]$ ./yas prob1a.ys
[pou14@linux2 y86-code]$ ./yis prob1a.yo
Stopped in 14 steps at PC = 0x6f.  Status 'HLT', CC Z=0 S=0 O=0
Changes to registers:
%rdx:   0x0000000000000000      0x0000000000000001
%rsp:   0x0000000000000000      0x00000000000001e8
%rsi:   0x0000000000000000      0x0000000000000003
%rdi:   0x0000000000000000      0x0000000000000007

Changes to memory:
0x01e8: 0x0000000000000000      0x0000000000000003
0x01f0: 0x0000000000000000      0x0000000000000007
0x01f8: 0x0000000000000000      0x0000000000000013
```

Case b → i = 8, j = 7; Output: i = 5, j = 11

```
[pou14@linux2 y86-code]$ ./yas prob1b.ys
[pou14@linux2 y86-code]$ ./yis prob1b.yo
Stopped in 14 steps at PC = 0x6f.  Status 'HLT', CC Z=0 S=0 O=0
Changes to registers:
%rdx:   0x0000000000000000      0x0000000000000004
%rsp:   0x0000000000000000      0x00000000000001e8
%rsi:   0x0000000000000000      0x000000000000000b
%rdi:   0x0000000000000000      0x0000000000000005

Changes to memory:
0x01e8: 0x0000000000000000      0x000000000000000b
0x01f0: 0x0000000000000000      0x0000000000000005
0x01f8: 0x0000000000000000      0x0000000000000013
```

Problem 2

Assembly Code assuming j = 3, k = 5:

```
irmovq $0x200, %rsp
call main

main:
        irmovq $0x03, %r8       #j stored in %r8, setting j = 3
        irmovq $0x05, %r9       #k stored in %r9, setting k = 5
        irmovq $0x04, %r10      #i stored in %r10, setting i = 4

loop:
        rrmovq  %r10, %r11      #temp stoted in %r11, setting temp = i
        addq %r11, %r11         #temp = i + i => i * 2
        rrmovq %r11, %r8        #j = temp

        #rrmovq %r8, %r11       #temp = j
        irmovq $0x04, %r12      #temp1 stored in %r12, temp1 = 4
        subq %r12, %r11         #temp = temp - temp1 => j - 4
        rrmovq %r11, %r9        #k = temp

        irmovq $0x01, %r11      #temp = 1
        addq %r11, %r10         #i = i + temp

        rrmovq %r10, %r11       #temp = i
        irmovq $0x0A, %r12      #temp1 = 10
        subq %r12, %r11         #temp = temp - 10
        jle loop                #jump to loop again if le 0

End:
        pushq %r8               #push j to stack
        pushq %r9               #push k to stack
        popq %r8                #pop j
        popq %r9                #pop k
        halt
```

***Note: Register of j → %r8, Register of k → %r9, , Register of i → %r10**

Results:

Case 1: → j = 3, k = 5; Output: j = 20, k = 16

```
[pou14@linux2 y86-code]$ ./yas prob2.ys
[pou14@linux2 y86-code]$ ./yis prob2.yo
Stopped in 92 steps at PC = 0x6c.  Status 'HLT', CC Z=0 S=0 O=0
Changes to registers:
%rsp:   0x0000000000000000      0x00000000000001e8
%r8:    0x0000000000000000      0x0000000000000014
%r9:    0x0000000000000000      0x0000000000000010
%r10:   0x0000000000000000      0x000000000000000b
%r11:   0x0000000000000000      0x0000000000000001
%r12:   0x0000000000000000      0x000000000000000a

Changes to memory:
0x01e8: 0x0000000000000000      0x0000000000000010
0x01f0: 0x0000000000000000      0x0000000000000014
0x01f8: 0x0000000000000000      0x0000000000000013
```

Problem 3

Assembly Code for 3.1:

```
	.file	"lab5_prob3_1.c"
	.text
	.section	.rodata
.LC0:
	.string "Hello, world"
	.text
	.globl	main
	.type	main, @function
main:
.LFB0:
	.cfi_startproc
	endbr64
	pushq	%rbp
	.cfi_def_cfa_offset 16
	.cfi_offset 6, -16
	movq	%rsp, %rbp
	.cfi_def_cfa_register 6
	subq	$16, %rsp
	movl	%edi, -4(%rbp)
	movq	%rsi, -16(%rbp)
	leaq	.LC0(%rip), %rax
	movq	%rax, %rdi
	call	puts@PLT
	movl	$0, %eax
	leave
	.cfi_def_cfa 7, 8
	ret
	.cfi_endproc
.LFE0:
	.size	main, .-main
	.ident	"GCC: (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0"
	.section	.note.GNU-stack,"",@progbits
	.section	.note.gnu.property,"a"
	.align 8
	.long	1f - 0f
	.long	4f - 1f
	.long	5
0:
	.string "GNU"
1:
	.align 8
	.long	0xc0000002
	.long	3f - 2f
2:
	.long	0x3
3:
	.align 8
4:
```

Assembly Code for 3.2:

```
	.file	"lab5_prob3_2.c"
	.text
	.section	.rodata
.LC0:
	.string "The value of  i is %d\n"
	.text
	.globl	main
	.type	main, @function
main:
.LFB0:
	.cfi_startproc
	endbr64
	pushq	%rbp
	.cfi_def_cfa_offset 16
	.cfi_offset 6, -16
	movq	%rsp, %rbp
	.cfi_def_cfa_register 6
	subq	$32, %rsp
	movl	%edi, -20(%rbp)
	movq	%rsi, -32(%rbp)
	movl	$2, -4(%rbp)
	addl	$1, -4(%rbp)
	movl	-4(%rbp), %eax
	movl	%eax, %esi
	leaq	.LC0(%rip), %rax
	movq	%rax, %rdi
	movl	$0, %eax
	call	printf@PLT
	movl	$0, %eax
	leave
	.cfi_def_cfa 7, 8
	ret
	.cfi_endproc
.LFE0:
	.size	main, .-main
	.ident	"GCC: (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0"
	.section	.note.GNU-stack,"",@progbits
	.section	.note.gnu.property,"a"
	.align 8
	.long	1f - 0f
	.long	4f - 1f
	.long	5
0:
	.string "GNU"
1:
	.align 8
	.long	0xc0000002
	.long	3f - 2f
2:
	.long	0x3
3:
	.align 8
4:
```

Compare and Analysis

By comparing the lengths of the two different assembly codes, we can observe that problem 3.2 involves more steps than problem 3.1, resulting in the use of more memory in problem 3.2 compared to problem 3.1.

Both files have a **.LC0** section that stores strings. The **main** section contains all the computations. The **.LFB0** section marks the beginning of the computation, accessing registers and memory to change values. Sections .**LFE0, 0, 1, 2, 3, 4** store basic information about our compiler.

In the assembly code of problem 3.1, it simply sets up the **main** function within the **.LFB0** section and prints the "Hello, world" string.

In the assembly code of problem 3.2, additional memory is used for storing the value of "i", which is stored 4 bytes below the address of %rbp. The increment operation i++ is also implemented at the same memory address. Subsequently, the value is moved to the return register %eax.

Problem 4

Assembly Code:

```
        .file   "lab5_prob4_main.c"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    %edi, -4(%rbp)
        movq    %rsi, -16(%rbp)
        movl    $0, %eax
        call    print_hello
        movl    $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .section        .rodata
.LC0:
        .string "Hello, world"
        .text
        .globl  print_hello
        .type   print_hello, @function
```

```
print_hello:
.LFB1:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        leaq    .LC0(%rip), %rax
        movq    %rax, %rdi
        call    puts@PLT
        nop
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE1:
        .size   print_hello, .-print_hello
        .ident  "GCC: (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0"
        .section        .note.GNU-stack,"",@progbits
        .section        .note.gnu.property,"a"
        .align 8
        .long   1f - 0f
        .long   4f - 1f
        .long   5
0:
        .string "GNU"
1:
        .align 8
        .long   0xc0000002
        .long   3f - 2f
2:
        .long   0x3
3:
        .align 8
4:
```

While this code segment actually achieves the same task as problem 3.1, it introduces a new function called print_hello() to accomplish its goal.

Upon examining the assembly code, a new section called **print_hello** is created, given that it is the function name specified in C++. Within this **print_hello** section, **%rbp** is pushed onto the stack when the function is called. After printing "Hello, world" using the **puts** function, **%rbp** is then popped off the stack, and the function returns to the calling point, which is the **main** segment.

In both the assembly code for problem 3.1 and problem 4, the address of "Hello, world" is saved into respective registers, where it is stored in **%rax**. Overall, both assembly codes perform the same task by loading the address of the string into a register. In problem 3.1, it simply returns the value, while in problem 4, it returns to the main function, where it was originally located within the **print_hello** block.

Problem 5

Assembly Code for 5_main:

```
	.file	"lab5_prob5_main.c"
	.text
	.globl	main
	.type	main, @function
main:
.LFB0:
	.cfi_startproc
	endbr64
	pushq	%rbp
	.cfi_def_cfa_offset 16
	.cfi_offset 6, -16
	movq	%rsp, %rbp
	.cfi_def_cfa_register 6
	subq	$16, %rsp
	movl	%edi, -4(%rbp)
	movq	%rsi, -16(%rbp)
	movl	$0, %eax
	call	print_hello@PLT
	movl	$0, %eax
	leave
	.cfi_def_cfa 7, 8
	ret
	.cfi_endproc
.LFE0:
	.size	main, .-main
	.ident	"GCC: (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0"
	.section	.note.GNU-stack,"",@progbits
	.section	.note.gnu.property,"a"
	.align 8
	.long	1f - 0f
	.long	4f - 1f
	.long	5
0:
	.string "GNU"
1:
	.align 8
	.long	0xc0000002
	.long	3f - 2f
2:
	.long	0x3
3:
	.align 8
4:
```

Assembly Code for 5_print:

```
	.file	"lab5_prob5_print.c"
	.text
	.section	.rodata
.LC0:
	.string "Hello, world"
	.text
	.globl	print_hello
	.type	print_hello, @function
print_hello:
.LFB0:
	.cfi_startproc
	endbr64
	pushq	%rbp
	.cfi_def_cfa_offset 16
	.cfi_offset 6, -16
	movq	%rsp, %rbp
	.cfi_def_cfa_register 6
	leaq	.LC0(%rip), %rax
	movq	%rax, %rdi
	call	puts@PLT
	nop
	popq	%rbp
	.cfi_def_cfa 7, 8
	ret
	.cfi_endproc
.LFE0:
	.size	print_hello, .-print_hello
	.ident	"GCC: (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0"
	.section	.note.GNU-stack,"",@progbits
	.section	.note.gnu.property,"a"
	.align 8
	.long	1f - 0f
	.long	4f - 1f
	.long	5
0:
	.string "GNU"
1:
	.align 8
	.long	0xc0000002
	.long	3f - 2f
2:
	.long	0x3
3:
	.align 8
4:
```

While there isn't much difference between both method of calling the print function, the only obvious distinction is that when the program calls a function that resides in another file, it adds **@PLT** to the call, where **@PLT** stands for "Procedure Linkage Table". In problem 4, the syntax is **call print_hello**, while in problem 5, it's **call print_hello@PLT.**

Problem 6

Assembly code:

```
int very_fast_function(int i){

    int result;

    __asm__(
        "movl %[input_i], %%eax;"       //i stored in in eax
        "leal (%%rax,%%rax,2), %%ecx;"  //return value stored in eax, return = i * 3
        "leal (%%rcx,%%rcx,4), %%ecx;"  //return value = i * 3 * 5
        "addl $15, %%ecx;"              //return = return + 15
        "cmpl $300, %%ecx;"             //comparing return - 300
        "jle else;"                     //jump to else if temp <= 300
        "movl $0, %%eax;"               //return = 0
        "jmp end;"                      //jump to end
        "else:;"                        //else block
        "addl $1, %%eax;"               //i++
        "end:;"                         //end block
        : "=r" (result)                 //output: result in eax
        : [input_i] "r" (i)             //input: i
        : "%ecx"                            //return
    );

    return result;
}
```

*Note: Using a longward (32-bit) to implement this function because variable type integer is in 32-bit.

Output: i = 12

```
pou14@Dell-XPS-13:/mnt/c/Users/ivano/OneDrive/桌面/CSCE 312/Lab 5/lab5_srcs$ gcc lab5_prob6.c
pou14@Dell-XPS-13:/mnt/c/Users/ivano/OneDrive/桌面/CSCE 312/Lab 5/lab5_srcs$ ./a.out
The function value of i is 13
```

Output: i = 21

```
pou14@Dell-XPS-13:/mnt/c/Users/ivano/OneDrive/桌面/CSCE 312/Lab 5/lab5_srcs$ gcc lab5_prob6.c
pou14@Dell-XPS-13:/mnt/c/Users/ivano/OneDrive/桌面/CSCE 312/Lab 5/lab5_srcs$ ./a.out
The function value of i is 0
```