

CSCE 312 Lab 5

Kevin Lei

March 31, 2024

Problem 1

```
irmovq $0x100, %rsp    # set up stack pointer
call main              # run the main program
halt                  # end the program after main terminates

main:                  # main program starts here
    irmovq $8, %rdi     # int i stored in rdi
    irmovq $7, %rsi     # int j stored in rsi
    rrmovq %rsi, %r8    # move the value of j to a temporary register
    subq %rdi, %r8      # subtract i from j
    jge else            # if i ≤ j, jump to else
    irmovq $3, %r8      # move 3 to a temporary register
    subq %r8, %rdi      # subtract 3 from i
    irmovq $4, %r8      # move 4 to a temporary register
    addq %r8, %rsi      # add 4 to j
    ret                 # return, main program ends here

else:                  # else block starts here
    irmovq $7, %rdi     # set i = 7
    irmovq $1, %r8      # set 1 to a temporary register
    addq %r8, %rsi      # j++
    ret                 # return, else block ends here
```

Figure 1: Y86-64 code for Problem 1

Note: int i stored in %rdi, int j stored in %rsi

```
labs/lab5/lab5_srcs 8 → ./yas prob1.y
labs/lab5/lab5_srcs 8 → ./yis prob1.yo
Stopped in 12 steps at PC = 0x13. Status 'HLT', CC Z=0 S=0 O=0
Changes to registers:
%rsp: 0x0000000000000000 0x0000000000000100
%rsi: 0x0000000000000000 0x0000000000000003
%rdi: 0x0000000000000000 0x0000000000000007
%r8:  0x0000000000000000 0x0000000000000001

Changes to memory:
0x00f8: 0x0000000000000000 0x0000000000000013
labs/lab5/lab5_srcs 0 →
```

Figure 2: Output for Problem 1 with i = 1, j = 2

```

labs/lab5/lab5_srcs 8 → ./yas prob1.y
labs/lab5/lab5_srcs 8 → ./yis prob1.yo
Stopped in 13 steps at PC = 0x13. Status 'HLT', CC Z=0 S=0 O=0
Changes to registers:
%rsp: 0x0000000000000000 0x0000000000000100
%rsi: 0x0000000000000000 0x000000000000000b
%rdi: 0x0000000000000000 0x0000000000000005
%r8: 0x0000000000000000 0x0000000000000004

Changes to memory:
0x00f8: 0x0000000000000000 0x0000000000000013
labs/lab5/lab5_srcs 8 →

```

Figure 3: Output for Problem 1 with $i = 8$, $j = 7$

Problem 2

```

irmovq $0x100, %rsp      # set up stack pointer
call main                 # call the main program
halt                     # end of the program

main:
    irmovq $4, %rax       # int i = 4 in %rax
    jmp test              # test the for loop condition

loop:
    rrmovq %rax, %rbx     # move i into temporary register
    addq %rbx, %rbx       # double i in the temporary register
    rrmovq %rbx, %rcx     # j = 2 * i in %rcx
    irmovq $4, %rbx       # move 4 into temporary register
    rrmovq %rcx, %rdx     # k = j
    subq %rbx, %rdx       # k = k - 4 in %rdx
    irmovq $1, %rbx       # move 1 into temporary register
    addq %rbx, %rax       # i++, end of loop
    jmp test              # test the for loop condition

test:
    irmovq $10, %rbx      # move 10 into %rbx for comparison
    subq %rax, %rbx       # subtract i from 10
    jge loop              # if i ≤ 10, proceed with the loop
    halt                  # else, end the program

```

Figure 4: Y86-64 code for Problem 2

Note: `int i` stored in `%rax`, `int j` stored in `%rcx`, and `int k` stored in `%rdx`. Initial values of `j` and `k` do not matter, since they are both always overwritten by $i * 2$ and $j - 4$ (in other words $i * 2 - 4$) respectively.

```

labs/lab5/lab5_srcs 8 → ./yas prob2.y
labs/lab5/lab5_srcs 8 → ./yis prob2.yo
Stopped in 92 steps at PC = 0x65. Status 'HLT', CC Z=0 S=1 O=0
Changes to registers:
%rax: 0x0000000000000000 0x000000000000000b
%rcx: 0x0000000000000000 0x0000000000000014
%rdx: 0x0000000000000000 0x0000000000000010
%rbx: 0x0000000000000000 0xffffffffffffffff
%rsp: 0x0000000000000000 0x00000000000000f8

Changes to memory:
0x00f8: 0x0000000000000000 0x0000000000000013
labs/lab5/lab5_srcs 8 →

```

Figure 5: Output for Problem 2

Problem 3

```
.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:
```

Figure 6: x86-64 code for lab5_prob3_1.c

```
.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:
```

Figure 7: x86-64 code for lab5_prob3_2.c

The first program, `lab5_prob3_1.c`, just prints "Hello, world" to the console. Its x86-64 assembly equivalent starts with various directives and defines `.LC0` as the string "Hello, world", which will be accessed later. Then, in the `main` function, it sets up the stack frame with `pushq %rbp` and `movq %rsp, %rbp`. The function arguments `argc` and `argv` get pushed to the stack with `movl %edi, -4(%rbp)` and `movq %rsi, -16(%rbp)`, even though they are not used in the program. Finally, we get to the most important part of the program, which is the printing of "Hello, world" to the console. The address of the "Hello, world" string is loaded into `%rax` with `leaq .LC0(%rip), %rax`, and then the address is passed to `puts` with `movq %rax, %rdi` and `call puts@PLT`. The `puts` function is what actually prints the string to the console. Then, the program sets the return value to 0 with `movl $0, %eax` and exits with `leave` and `ret`. The rest of the code is irrelevant to the program's functionality, and is just information for the compiler and linker.

The second program, `lab5_prob3_2.c`, declares an integer `i` and initializes it to 2, increments it by 1, and then prints it to the console. The code is very similar to the first program, but with some minor differences. Like the first program, it sets up the stack frame and function arguments and includes similar information for the compiler and linker. Also, it declares the string "The value of i is %d\n" at `.LC0`. In the function body, the integer `i` is declared and initialized to 2 with `movl $2, -4(%rbp)`

and then incremented with `addl $1, -4(%rbp)`. The value of `i` is first loaded into `%eax` with `movl -4(%rbp), %eax` and then to `%esi` with `movl %eax, %esi`. This was just to prepare the value of `i` to be printed with the `printf` function. Finally, the address of the string is loaded into `%rdi` with `leaq .LC0(%rip), %rax` and `movq %rax, %rdi`, and then `printf` is called with `call printf@PLT`.

Problem 4

The main difference between `lab5_prob3_1.c` and `lab5_prob4.c` is the way that the "Hello world" string is printed. We can see from the C source code that `lab5_prob3_1.c` calls `printf()` directly in the main function, while `lab5_prob4.c` calls a function `print_hello()` that calls `printf()`. In the assembly generated, the effective address of the string "Hello, world" is loaded into `%rax` and moved into `%rdi`, and then `call puts@PLT` is called, which effectively prints the string to the console. In the code for `lab5_prob4.c`, the corresponding assembly code calls `call print_hello` in the main directive, and the `print_hello` function uses the same logic as `lab5_prob3_1.c` to print the string to the console.

Problem 5

```
.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:
```

Figure 8: Assembly generated from main code

```
.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:
```

Figure 9: Assembly generated from print function

```
labs/lab5/lab5_srcs → gcc -c prob5_main.s -o lab5_main.o
labs/lab5/lab5_srcs → gcc -c prob5_print.s -o lab5_print.o
labs/lab5/lab5_srcs → gcc prob5_main.o prob5_print.o -o prob5
labs/lab5/lab5_srcs →
```

Figure 10: Generating object files and linking them together

Problem 6

```
#include <stdio.h>

int very_fast_function(int i) {
    int result;
    __asm__ (
        "movl %1, %%eax;"           // move i into %eax
        "imull $18, %%eax;"         // multiply %eax by 18 (i.e. i * 18)
        "subl $3, %%eax;"           // subtract 3 from %eax (i.e. i * 18 - 3)
        "cmpl $300, %%eax;"         // compare %eax with 300
        "jle .increment;"           // if %eax ≤ 300, jump to .increment
        "movl $0, %%eax;"           // else, set %eax to 0
        "jmp .done;"                // jump to .done
        ".increment:"               // label for the increment
        "incl %1;"                  // increment i
        "movl %1, %%eax;"           // move i into %eax
        ".done:"                   // label for the end
        "movl %%eax, %0;"           // move %eax into result
        : "=r" (result)             // output gets assigned to result variable declared above
        : "r" (i)                   // input to out assembly code
        : "%eax"                    // tell gcc that %eax is the clobbered register
    );
    return result;
}

int main(int argc, char *argv[]) {
    int i;
    i = 16;
    printf("The function value of i is %d\n", very_fast_function(i));
    return 0;
}
```

Figure 11: `very_fast_function()` rewritten with inline assembly

```
labs/lab5/lab5_srcs 8 → gcc lab5_prob6.c
labs/lab5/lab5_srcs 8 → ./a.out
The function value of i is 17
labs/lab5/lab5_srcs 8 →
```

Figure 12: Compiling and running `lab5_prob6.c`