

School of Mathematical and Computational Sciences
Indian Association for the Cultivation of Science

Compiler Construction: COM 5202

Tutorial I (15 January, 2025)

M. Sc Semester IV: 2024-2025

Instructor: Goutam Biswas

Consider the following C program and the corresponding x86-64 assembly language code generated by the gcc compiler.

C Program:

```
#include <stdio.h>
int main(){ // t1.1.exer.c
    int a[10]={0}, b, i;

    printf("Enter an integer: ");
    scanf("%d", &b);
    for(i=1; i<=9; ++i)
        a[i] = 3*a[i-1]+b;
    printf("a[9]: %d\n", a[9]);
    return 0;
}
```

x86-64 Assembly Language Program:

```
.file "t1.1.exer.c" # source file name
.text #
.section .rodata # Readonly data
.LC0:
.string "Enter an integer: "
.LC1:
.string "%d"
.LC2:
.string "a[9]: %d\n"
.text
.globl main # name 'main' is global
.type main, @function # 'main' is a function
main: # 'main' starts
.LFB0:
    pushq %rbp # Save old base pointer
    movq %rsp, %rbp # Load new base pointer
    # rbp <-- rsp
    subq $64, %rsp # 64-byte stack frame
    # rsp <-- rsp-64
    # a[10]={0}
    movq $0, -48(%rbp) # 40 bytes filled with 0
    movq $0, -40(%rbp) # Mem[rbp-48] <-- 0
    movq $0, -32(%rbp)
    movq $0, -24(%rbp)
    movq $0, -16(%rbp) # Mem[rbp-16] <-- 0
    # Mem[rbp - (48 ... 9)] <-- 0
```

```

#
# printf("Enter an integer: ");
leaq    .LC0(%rip), %rdi # rdi <-- rip+.LC0
# 1st parameter to printf
# Starting address of format string
movl    $0, %eax        # eax <-- 0
call    printf@PLT      # call printf
#
# scanf("%d", &b);
leaq    -56(%rbp), %rax  #
movq    %rax, %rsi      #
#
leaq    .LC1(%rip), %rdi #
#
#
movl    $0, %eax        # eas <-- 0
call    __isoc99_scanf@PLT # call to scanf
#
# i = 1;
movl    $1, -52(%rbp)    # Mem[rbp-52] (i) <-- 1
jmp     .L2              # goto .L2
.L3:
# a[i] = 3*a[i-1]+b;
movl    -52(%rbp), %eax  # eax <-- Mem[rbp-52] (i)
subl    $1, %eax         # eax <-- eax-1 (i-1)
cltq    %rax             # rax <-- eax (31-bit to 64-bit)
# rax contains $i-1
movl    -48(%rbp,%rax,4), %edx # edx <-- Mem[rbp - 48 + 4*rax]
# edx <-- a[i-1]
movl    %edx, %eax       # eax <-- edx (a[i-1])
addl    %eax, %eax       # eax <-- eax+eax (2a[i-1])
addl    %eax, %edx       # edx <-- edx (2a[i-1]) + eax (a[i-1])
# rdx has 3*a[i-1]
movl    -56(%rbp), %eax  # eax <-- Mem[rbp-56] (b)
# eax has b
addl    %eax, %edx       # rdx <-- edx + eax
# edx <-- 3*a[i-1] + b
movl    -52(%rbp), %eax  # eax <-- Mem[rbp-52] (i)
cltq    %rax             # rax <-- i
movl    %edx, -48(%rbp,%rax,4) # Mem[rbp-48+4*rax] <-- edx
# a[i] <-- 3*a[i-1] + b
# i++
addl    $1, -52(%rbp)    # Mem[rbp-52] <-- Mem[rbp-52]+1
# i <-- i+1
.L2:
# if i <= 9 loop
cmpl    $9, -52(%rbp)    #
jle     .L3              #
movl    -12(%rbp), %eax  # eax <-- Mem[rbp-12] (a[9])
movl    %eax, %esi       # esi <-- eax

```

```

                                # 2nd param.
leaq    .LC2(%rip), %rdi # rdi <-- rip+.LC2
                                # starting address of format string

movl    $0, %eax
call    printf@PLT          # call to printf
movl    $0, %eax
leave
ret
.LFE0:
.size   main, .-main
.ident  "GCC: (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4.0"
.section .note.GNU-stack,"",@progbits
.section .note.gnu.property,"a"

```

Exercise 1.

Marks: 10

- How do you make a function name non-global?
- What is the purpose of the instruction `cltq` (convert long to quad)?
- What is the displacement of `a[5]` with respect to `rbp`?
- How do you modify the assembly language code to print `a[5]` instead of `a[9]`?
- How do you modify the assembly language code corresponding to `a[i] = 3*a[i-1]+b;` by adding one more instruction to compute `a[i] = 5*a[i-1]+b;`?
- Explain the following pair of instructions:

```

cml    $9, -52(%rbp)
jle    .L3

```

- The test `(i <= 9)` is at the end of the body of *for-loop* in the assembly language code. Modify the code to bring it at the beginning of the body of *for-loop*.
- Explain the assembly code for `scanf("%d", &b);`
- On my computer the `objdump` of the `a.out` file shows the following code at the virtual memory location `00 00 06 d9` to `00 00 06 dd` corresponding to the call of `scanf()`.

```

6d9: e8 a2 fe ff ff    callq  580 <__isoc99_scanf@plt>
6de: next instruction

```

The next instruction starts from the address `00 00 06 de`.
`e8` is the op-code for `callq`.
`a2 fe ff ff` specifies the PC-relative address of the call-location. Corresponding assembly code shows `580`. How do you relate `580` with `a2 fe ff ff`?

Send the answer to goutamamartya@gmail.com. Kindly mention **Tutorial - I** and your name in the Subject: of the mail.