School of Mathematical and Computational Sciences

Indian Association for the Cultivation of Science

Compiler Construction: COM 5202

Tutorial II (14 January, 2026)

M. Sc Semester IV: 2025-2026 *Instructor:* Goutam Biswas

Exercise 1. **[Marks: 10]**

Consider the following C program and the corresponding x86-64 assembly language code generated by the gcc compiler. Answer the questions given after the code. Explain the semantics of the assembly language code and mention its connection with the C code.

```
#include <stdio.h>

void compute(int m, int dat[]){
     int j;
     for(j=m; j>1; --j)
        dat[j] = 4*dat[j-1]+7*dat[j-2];
}

int main(){
    int d[100], n, i;

    printf("Enter a +ve integer < 100: ");
    scanf("%d", &n);
    d[n]=0;

    printf("Enter %d integer data: ", n);
    for(i=0; i<n; ++i) scanf("%d", &d[i]);
    compute(n, d);
    printf("Print %d integer data: ", n);
    for(i=0; i<=n; ++i) printf("%d ", d[i]);

    return 0;
}
```

x86-64 Assembly Language Program:

```
  .file  "ex1.c"             # file name
  .text                      # program text
  .globl  compute            # --> (b)
  .type  compute, @function  # 'compute' is a function
compute:                     # 'compute' starts
.LFB0:
  pushq  %rbp                # save old base pointer
  movq  %rsp, %rbp           # rbp <-- rsp
                             #   new base pointer
  movl  %edi, -20(%rbp)      # Mem[rbp-20] (m) <-- edi
  movq  %rsi, -32(%rbp)      # Mem[rbp-32] (dat) <-- rsi
                             #    base address of array
```

```
  movl  -20(%rbp), %eax      # --> (i)
  movl  %eax, -4(%rbp)       # Mem[rbp-4] (j) <-- m
  jmp  .L2                   # goto .L2
.L3:                         # label L3.
  movl  -4(%rbp), %eax       # eax <-- Mem[rbp-4] (j)
  cltq                       # rax <-- (64-bit) eax (j)
  salq  $2, %rax             # --> (ii)
                             #
  leaq  -4(%rax), %rdx       # --> (iii)
  movq  -32(%rbp), %rax      # rax <-- Mem[rbp-32] (dat)
  addq  %rdx, %rax           # --> (iv)
                             #
  movl  (%rax), %eax         # eax <-- Mem[rax] (dat[j-1]
  leal  0(,%rax,4), %esi     # esi <-- 4*Mem[rax]
                             #     esi <-- 4*dat[j-1]
  movl  -4(%rbp), %eax       # eax <-- Mem[ebp-4] (j)
  cltq                       # rax <-- (64-bit) eax (j)
  salq  $2, %rax             # rax <-- shift-left-arithmetic
                             #     rax (rax <-- 4j)
  leaq  -8(%rax), %rdx       # rdx <-- rax - 8 (j-2)
  movq  -32(%rbp), %rax      # rax <-- Mem[rbp-32] (dat)
  addq  %rdx, %rax           # rax < -- dat + (j-2)
                             #    rax <-- &dat[j-2]
  movl  (%rax), %edx         # edx <-- Mem[rax] (dat[j-2]
  movl  %edx, %eax           # eax <-- edx (dat[j-2])
  sall  $3, %eax             # eax <-- 3-shift-arithmetic-left
                             #     eax (eax <-- 8*dat[j-2])
  subl  %edx, %eax           # eax <-- eax  edx
                             #     8*dat[j-2] - dat[j-2]
                             # eax <-- 7*dat[j-2]
  movl  %eax, %ecx           # ecx <-- 7*dat[j-2]
  movl  -4(%rbp), %eax       # eax <-- Mem[rbp-4] (j)
  cltq                       # rax <-- (64-bit) eax (j)
  leaq  0(,%rax,4), %rdx     # rdx <-- 4*(rax)(j)
  movq  -32(%rbp), %rax      # rax <-- Mem[rbp-32] (dat)
  addq  %rdx, %rax           # rax <-- rax + rdx (dat + 4j)
                             #      rax <-- &dat[j]
  leal  (%rsi,%rcx), %edx    # edx <-- rsi (4*dat[j-1]) +
                             #       rcx (7*dat[j-2])
  movl  %edx, (%rax)         # Mem[eax] (dat[j]) <--
                             #  edx (4*dat[j-1] +
                             #       7*dat[j-2])
  subl  $1, -4(%rbp)         # Mem[rbp-4] <-- Mem[rbp-4]
                             #    j <-- j-1
.L2:                         # label .L2
  cmpl  $1, -4(%rbp)         # if Mem[rbp-4] (j) > 1
  jg  .L3                    #   goto .L3
  nop                        # no operation
  nop                        # no operation
  popq  %rbp                 # Restore old stack pointer
```

```
  ret
.LFE0:
  .size  compute, .-compute
  .section  .rodata          # read-only data
.LC0:                        # Constant strings
  .string  "Enter a +ve integer < 100: "
.LC1:
  .string  "%d"
.LC2:
  .string  "Enter %d integer data: "
.LC3:
  .string  "Print %d integer data: "
.LC4:
  .string  "%d "
  .text                      # code
  .globl  main               # 'main' is global
  .type  main, @function     # 'main' is a function
main:                        # label 'main'
.LFB1:
  pushq  %rbp                # save old base pointer
  movq  %rsp, %rbp           # rbp <-- rsp (new base pointer)
  subq  $432, %rsp           # --> (v)
  movq  %fs:40, %rax
  movq  %rax, -8(%rbp)
  xorl  %eax, %eax
  leaq  .LC0(%rip), %rax     # rax <-- address of the format
                             #    string of 1st printf
  movq  %rax, %rdi           # rdi <-- 1st parameter
  movl  $0, %eax
  call  printf@PLT           # call printf
  leaq  -424(%rbp), %rax     # -->
  movq  %rax, %rsi           # -->
                             # -->
  leaq  .LC1(%rip), %rax     # -->
                             # -->
  movq  %rax, %rdi           # --> (vi)
  call  __isoc99_scanf@PLT   #
  movl  -424(%rbp), %eax     # eax <-- Mem[rbp-424] (n)
  cltq                       # rax <-- (64-bit) eax (n)
  movl  $0, -416(%rbp,%rax,4) # Mem[rbp+4*rax] (d[n])
                             #    <-- 0
  movl  -424(%rbp), %eax     # eax <-- Mem[rbp-424] (n)
  movl  %eax, %esi           # esi <-- eax (n) 2nd parameter
  leaq  .LC2(%rip), %rax     # rax <-- address of the format
                             #    string of 2nd printf
  movq  %rax, %rdi           # rdi <-- rax, 1st parameter
  movl  $0, %eax
  call  printf@PLT           # call printf
  movl  $0, -420(%rbp)       # Mem[rbp-420] (i) <-- 0
  jmp  .L5                   # goto .L5
```

```
.L6:                            # label .L6
  leaq  -416(%rbp), %rax        # rax <-- rbp - 416 (d)
  movl  -420(%rbp), %edx        # edx <-- Mem[rbp-420] (i)
  movslq  %edx, %rdx            # rdx <-- (64 bit) edx (i)
  salq  $2, %rdx                # rdx <-- 4*rdx (4*i)
  addq  %rdx, %rax              # rax <-- rax (d) + rdx (4*i)
  movq  %rax, %rsi              # rsi <-- rax (&d[i])
  leaq  .LC1(%rip), %rax        # rax <-- address of the format
                                #    string of 2nd scanf
  movq  %rax, %rdi              # rdi <-- rax, 1st parameter
  movl  $0, %eax
  call  __isoc99_scanf@PLT      # call scanf
  addl  $1, -420(%rbp)          # Mem[rbp-420] (i) <--
                                #         Mem[rbp-420] (i) + 1
.L5:                            # Label .L5
  movl  -424(%rbp), %eax        # -->
  cmpl  %eax, -420(%rbp)        # -->
  jl  .L6                       # --> (vii)
  movl  -424(%rbp), %eax        # -->
  leaq  -416(%rbp), %rdx        # -->
  movq  %rdx, %rsi              # -->
  movl  %eax, %edi              # -->
  call  compute                 # --> (viii)
  movl  -424(%rbp), %eax
  movl  %eax, %esi
  leaq  .LC3(%rip), %rax
  movq  %rax, %rdi
  movl  $0, %eax
  call  printf@PLT
  movl  $0, -420(%rbp)
  jmp  .L7
.L8:
  movl  -420(%rbp), %eax
  cltq
  movl  -416(%rbp,%rax,4), %eax
  movl  %eax, %esi
  leaq  .LC4(%rip), %rax
  movq  %rax, %rdi
  movl  $0, %eax
  call  printf@PLT
  addl  $1, -420(%rbp)
.L7:
  movl  -424(%rbp), %eax
  cmpl  %eax, -420(%rbp)
  jle  .L8
  movl  $0, %eax
  movq  -8(%rbp), %rdx
  subq  %fs:40, %rdx
  je  .L10
  call  __stack_chk_fail@PLT
```

```
.L10:
  leave
  ret
.LFE1:
  .size  main, .-main
  .ident  "GCC: (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0"
  .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:
```

(a) An 8-bit 2's complement integer **1101 0010** is to be extended to 16-bit 2's complement integer of same value. What will be the bit pattern?

(b) Explain the following instructions.

(i) `movl  -20(%rbp), %eax`

(ii) `salq  $2, %rax`

(iii) `leaq  -4(%rax), %rdx`

(iv) `addq  %rdx, %rax`

(v) `subq  $432, %rsp`

(vi) Code corresponding to the first `scanf`

```
leaq -424(%rbp), %rax
movq  %rax, %rsi
leaq  .LC1(%rip), %rax
movq  %rax, %rdi
call  __isoc99_scanf
```

(vii) Code below `.L5:`

```
movl  -424(%rbp), %eax
cmpl  %eax, -420(%rbp)
jl    .L6
```

(viii) Code before call to compute.

```
movl  -424(%rbp), %eax
leaq  -416(%rbp), %rdx
movq  %rdx, %rsi
movl  %eax, %edi
call  compute
```

(c) Which register holds the return value of a function (if any)?

Exercise 2. **[Marks:5]**

Convert the decimal number 13.25 to 32-bit and 64-bit floating-point numerals.

Exercise 3. **[Marks: 5]**

Explain the output of the following two C programs.

```
/* f32.c */
#include <stdio.h>
int main(){
    int n = 1096024064;
    float *fp = (float *)&n;

    printf("n=%f\n", *fp);
    return 0;
}
```

Output:

```
$ ./a.out
n=13.250000
$
```

```
/* f64.c */
#include <stdio.h>
int main(){
    int nH = 1076527104, nL=0;
    double *fp = (double *)&nL;;

    printf("n=%lf\n", *fp);
    return 0;
}
```

Output: `$ ./a.out`

```
n=13.250000
$
```

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