1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2 **CPL ASSIGNMENT 4**

3

**4 Name : Darshan Dinkar Nikam**

**5 Enrollment No : BT19CSE070**

**6 Topic : Analysis of assembly code generated after compilation of .c**

**File.**

**7 Submission Date : 10/04/2021**

8

9

10 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

11

12 **C code in file named CPL\_Assign4.c:**

13

14

15

16 #include<stdio.h>

17

18

19 int main()

20 {

21 int option;

22

23 int x = 9;

24 int y = 4;

25 int ans;

26 int temp;

27 int prod;

28

29 printf("You have x = 9,y = 4\nEnter option :\n1 : Add two numbers\n2 : Calculate x^y(^ stands for power here)\n");

30

31 scanf("%d",&option);

32

33 switch(option)

34 {

35 case 1:

36 ans = x + y;

37 printf("\n\n%d",ans);

38 break;

39

40 case 2:

41 if(y == 0 || y < 0) // handling if y = 0 and setting error value in ans if y is negative as our program wants

42 // y as non-negative int

43 {

44 ans = 1;

45 }

46 else if(y == 1)

47 {

48 ans = x;

49 }

50 else

51 {

52 ans = 1;

53 temp = y;

54 prod = x;

55 while(temp != 0)

56 {

57 if(temp % 2 == 1)

58 {

59 ans = ans\*prod;

60 }

61 prod = prod \* prod;

62 temp /= 2;

63 }

64 }

65 printf("\n\n%d",ans);

66 break;

67 default:

68 printf("Wrong input");

69

70 }

71

72

73

74 }

75

76

77 **-----Command used for producing corresponding assembly file "CPL\_Assign4.s"**

78

79 "gcc -S CPL\_Assign4.c -o CPL\_Assign4.s"

80

81

82 **-----Generated Assembly code**

83

84

85

**86 .file "CPL\_Assign4.c"**

**87 .text**

**88 .section .rodata**

**89 .align 8**

**90 .LC0:**

**91 .string "You have x = 9,y = 4\nEnter option :\n1 : Add two numbers\n2 : Calculate x^y(^ stands for power here)"**

**92 .LC1:**

**93 .string "%d"**

**94 .LC2:**

**95 .string "\n\n%d"**

**96 .LC3:**

**97 .string "Wrong input"**

**98 .text**

**99 .globl main**

**100 .type main, @function**

**101 main:**

**102 .LFB0:**

**103 .cfi\_startproc**

**104 endbr64**

**105 pushq %rbp**

**106 .cfi\_def\_cfa\_offset 16**

**107 .cfi\_offset 6, -16**

**108 movq %rsp, %rbp**

**109 .cfi\_def\_cfa\_register 6**

**110 subq $32, %rsp**

**111 movq %fs:40, %rax**

**112 movq %rax, -8(%rbp)**

**113 xorl %eax, %eax**

**114 movl $9, -16(%rbp)**

**115 movl $4, -12(%rbp)**

**116 leaq .LC0(%rip), %rdi**

**117 call puts@PLT**

**118 leaq -32(%rbp), %rax**

**119 movq %rax, %rsi**

**120 leaq .LC1(%rip), %rdi**

**121 movl $0, %eax**

**122 call \_\_isoc99\_scanf@PLT**

**123 movl -32(%rbp), %eax**

**124 cmpl $1, %eax**

**125 je .L2**

**126 cmpl $2, %eax**

**127 je .L3**

**128 jmp .L15**

**129 .L2:**

**130 movl -16(%rbp), %edx**

**131 movl -12(%rbp), %eax**

**132 addl %edx, %eax**

**133 movl %eax, -28(%rbp)**

**134 movl -28(%rbp), %eax**

**135 movl %eax, %esi**

**136 leaq .LC2(%rip), %rdi**

**137 movl $0, %eax**

**138 call printf@PLT**

**139 jmp .L5**

**140 .L3:**

**141 cmpl $0, -12(%rbp)**

**142 je .L6**

**143 cmpl $0, -12(%rbp)**

**144 jns .L7**

**145 .L6:**

**146 movl $1, -28(%rbp)**

**147 jmp .L8**

**148 .L7:**

**149 cmpl $1, -12(%rbp)**

**150 jne .L9**

**151 movl -16(%rbp), %eax**

**152 movl %eax, -28(%rbp)**

**153 jmp .L8**

**154 .L9:**

**155 movl $1, -28(%rbp)**

**156 movl -12(%rbp), %eax**

**157 movl %eax, -24(%rbp)**

**158 movl -16(%rbp), %eax**

**159 movl %eax, -20(%rbp)**

**160 jmp .L10**

**161 .L12:**

**162 movl -24(%rbp), %eax**

**163 cltd**

**164 shrl $31, %edx**

**165 addl %edx, %eax**

**166 andl $1, %eax**

**167 subl %edx, %eax**

**168 cmpl $1, %eax**

**169 jne .L11**

**170 movl -28(%rbp), %eax**

**171 imull -20(%rbp), %eax**

**172 movl %eax, -28(%rbp)**

**173 .L11:**

**174 movl -20(%rbp), %eax**

**175 imull %eax, %eax**

**176 movl %eax, -20(%rbp)**

**177 movl -24(%rbp), %eax**

**178 movl %eax, %edx**

**179 shrl $31, %edx**

**180 addl %edx, %eax**

**181 sarl %eax**

**182 movl %eax, -24(%rbp)**

**183 .L10:**

**184 cmpl $0, -24(%rbp)**

**185 jne .L12**

**186 .L8:**

**187 movl -28(%rbp), %eax**

**188 movl %eax, %esi**

**189 leaq .LC2(%rip), %rdi**

**190 movl $0, %eax**

**191 call printf@PLT**

**192 jmp .L5**

**193 .L15:**

**194 leaq .LC3(%rip), %rdi**

**195 movl $0, %eax**

**196 call printf@PLT**

**197 .L5:**

**198 movl $0, %eax**

**199 movq -8(%rbp), %rcx**

**200 xorq %fs:40, %rcx**

**201 je .L14**

**202 call \_\_stack\_chk\_fail@PLT**

**203 .L14:**

**204 leave**

**205 .cfi\_def\_cfa 7, 8**

**206 ret**

**207 .cfi\_endproc**

**208 .LFE0:**

**209 .size main, .-main**

**210 .ident "GCC: (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0"**

**211 .section .note.GNU-stack,"",@progbits**

**212 .section .note.gnu.property,"a"**

**213 .align 8**

**214 .long 1f - 0f**

**215 .long 4f - 1f**

**216 .long 5**

**217 0:**

**218 .string "GNU"**

**219 1:**

**220 .align 8**

**221 .long 0xc0000002**

**222 .long 3f - 2f**

**223 2:**

**224 .long 0x3**

**225 3:**

**226 .align 8**

**227 4:**

**// sequencewise explanation of the assembly code generated**

**// format : Line No. - Explanation.**

86 - Original source file name

87 - We write things in .text section where the code goes.

88 - .rodata refers to read only data.Application can do only read operation on this data.

89 - Setting allignment byte size to 8.. more useful when we want to optimize struct placement.

90-97 - Constant strings used in printf and scanf statements.

99-100 - Gloabally visible function main declaration

102 - Function begin label

103 - Initialize new CFI table(google'd about this)

104 - End branch 64 bit

105 - Push value in base pointer register on the stack.(which is special purpose register to keep track of frame pointer) q suffix for quad(64 bits)

108 – Moving (/copying) contents of stack pointer reg. in frame pointer register

110 - Subtract 32 positions from contents of stack pointer register(preserving space for local variables)

114 - Allotting memory space for variable "int x" and assigning value 9 to it. address(offset) is -16(%rbp).

115 - Allotting memory space for variable "int y" and assigning value 4 to it. address(offset) is -12(%rbp).

116-122 - Prompting for ,scanning and storing variable "int option" at location -32(%rbp)

123 - Moving contents at -32(%rbp) location i.e variable "option" in register %eax(e stands for extended,lower 32 bits of %rax register)

124 - Compare %eax's content with immediate value 1.

125 - Jump on lable .L2 if contents match. (i.e. in higher language, execute case 1's code)

126 - Compare %eax's content with immediate value 2.

127 - Jump on lable .L3 if contents match. (i.e. in higher language execute case 2's code)

128 - Jump to lable .L15 if %eax's content(value at option) doesn't match with either 1 or 2

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Case 1: (Starts at line 129)**

130-131 - Move contents at memory location -16(%rbp)(refers to variable x) and -12(%rbp) (refers to variable y) in registers %edx and %eax respectively

(As said earlier %edx and %eax are just lower halves of corresponding 64 bit registers)

132-138 - Adding contents of register %eax and %edx and storing result back in register %eax. storing this result furthermore at location -28(%rbp) which is for variable(int ans),and,at last, printing it.

139 - Jump to label .L5 which calls Label .L14 internally.

198- 207 - (Labels .L5 and .L14) storing return value at register %eax and returning from the function

**Case 1 execution completes over here...**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Case 2:(Starts at line 140)**

141 - Compare content of location -12(%rbp) i.e. variable y with immediate value 0

142 - If they match jump on Lable .L6

**\*\*code to be executed if(y == 0):**

in Label .L6:

146-147 - Set variable ans(memory loc is -28(%rbp) to 1 and jump to label .L8

186-192 - (in Label .L8) Printing the value of ans (call printf@PLT) and JMP to label .L5 which calls Label .L14 internally.

198 - 207 - (Labels .L5 and .L14) storing return value at register %eax and returning from the function

143 - Compare content of location -12(%rbp) i.e. variable y with immediate value 0

144 - Jump on no sign(which is the case when y>0) to label .L7(i.e go to next condition check.)

**\*\*Code to be executed if(y !=0 and y < 0):**

145-147 - This is to be executed if(y<0) which is an condition after || in first if statement.Here immediate value 1 is stored at location -28(%rbp) (ans) and .L8 is called which prints the ans ..In L8 ,L5 and L14 are called which terminate the function call by returning value at %eax.

**(Note --- Here it can be noted that if the first condition in if statement is fulfilled second condition is not checked(while ORing), hence optimizing the code.)**

**\*\*Code to be executed if(y == 1)(second if statement in .c code)”**

151 - Move content of x in %eax

152 - Transfer %eax's content in variable ans(-28(%rbp)) (setting ans = x)

153 - .L8 is called which prints the ans ..In L8 ,L5 and cosequently L14 are called which terminate the function call by returning value at %eax.

**\*\*Assembly code to be executed for third if statement in .c code:**

155 - Store immediate value 1 at location of variable ans(-28(%rbp))

156-157 - Transferring content of variable y in variable temp(-24(%rbp))

158-159 - Setting prod(-20(%rbp)) = x(-16(%rbp))

160 - Jump to .L10

Lable .L10

(Line 184) if(temp (-24(%rbp) != 0) //i.e loop is to be entered..

185 -- Call .L12

In L12 -

162 - temp(-24(%rbp)) to %eax move

164-167 - Calculate (temp % 2) and store in %eax register.

168 - Compare content of %eax with immediate value 1.

169 - if they don't match goto .L11.

170-172 -- ans = ans \* prod

In L11 -

174-176 -- prod = prod \* prod

177-182 -- temp = temp / 2

(Line 185) else //i.e loop is to be exited.

Execute L8(prints ans) and L15 and exit the program.