REPORT ON ASSEMBLY LEVEL LANGUAGE OF A BASIC C PROGRAM

CPL ASSIGNMENT 4

NAME **: GOWTHAM GORREPATI**

ENROLLMENT NUMBER **: BT19CSE033**

DEPARTMENT : **Computer Science And Engineering**

COLLEGE : **Visvesvaraya National Institute of Technology, Nagpur**

Compiler used : GCC (9.3.0)

Date : 10th April 2021

The C program can do two operations :

1) Accumulate the sum of all elements from arr[0] to arr[i] of an array ‘arr[]’ in acc[i], for 0 <= i < n, where n is the size of ‘arr[]’ ans ‘acc[]’ is another array of same type and size.
2) Print the count of positive numbers, negative numbers and zeroes in the array ‘arr[]’.

These two are selected using a switch case statement.

C PROGRAM

```
// BT19CSE033, Gowtham Gorrepati, Assignment-4, .c file
#include <stdio.h>

int main()
{
	int choose = 2, n = 6, i;
	int arr[] = {3, 4, -2, 4, 1, 0};
	int acc[6]; // acc[i] accumulates the sum of all arr[j] such that j<=i
	switch(choose) {
		case 1: { // to accumulate
	  int sum = 0;
			for(i=0; i<n; i++) {
				sum += arr[i];
				acc[i] = sum;
			}
			printf("Accumulated array is : ");
			for(i=0; i<n; i++)
				printf("%d ", acc[i]);
			printf("\n");
		}
		break;
		case 2: { // to count number of +ve numbers, -ve numbers, zeroes in array
			int pc, nc, zc;
			pc = nc = zc = 0;
```

```
                for(i=0; i<n; i++) {
                        if(arr[i]<0)
                                nc++;
                        else if(arr[i]>0)
                                pc++;
                        else
                                zc++;
                }
                printf("Number of negative numbers : %d\n", nc);
                printf("Number of zeroes : %d\n", zc);
                printf("Number of positive numbers : %d\n", pc);
                printf("\n");
        }
        break;
        default : break;
    }
    return 0;
}
```

ASSEMBLY LEVEL PROGRAM

```
    .file   "BT19CSE033_Assignment_4.c"
    .text
    .section    .rodata
.LC0:
    .string "Accumulated array is : "
.LC1:
    .string "%d "
    .align 8
.LC2:
    .string "Number of negative numbers : %d\n"
.LC3:
    .string "Number of zeroes : %d\n"
    .align 8
.LC4:
    .string "Number of positive numbers : %d\n"
    .text
```

```
        .globl  main
        .type   main, @function
main:                                   # main code begins here
        .cfi_startproc
        endbr64
        pushq %rbp                      # push main function frame pointer 'rbp' on the stack
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq  %rsp, %rbp                # set stack pointer to frame pointer
        .cfi_def_cfa_register 6
        subq  $96, %rsp                 # space allocation for function on stack
        movq  %fs:40, %rax
        movq  %rax, -8(%rbp)
        xorl    %eax, %eax
        movl  $2, -72(%rbp)             # move the immediate value '2' to variable 'choose'
        movl  $6, -68(%rbp)             # move the immediate value '6' to variable 'n'
        movl  $3, -64(%rbp)             # move the immediate value '3' to variable 'arr[0]'
        movl  $4, -60(%rbp)             # move the immediate value '4' to variable 'arr[1]'
        movl  $-2, -56(%rbp)            # move the immediate value '-2' to variable 'arr[2]'
        movl  $4, -52(%rbp)             # move the immediate value '4' to variable 'arr[3]'
        movl  $1, -48(%rbp)             # move the immediate value '1' to variable 'arr[4]'
        movl  $0, -44(%rbp)             # move the immediate value '0' to variable 'arr[5]'
        cmpl  $1, -72(%rbp)             # compare immediate value '1' with 'choose'
        je      .L2                     # jump to '.L2' if 'choose' equals '1'
        cmpl  $2, -72(%rbp)             # compare immediate value '2' with 'choose'
        je      .L3                     # jump to '.L3' if 'choose' equals '2'
        jmp     .L9                     # jump to label '.L9'
.L2:                                    # Label '.L2'
        movl  $0, -88(%rbp)             # move the immediate value '0' to variable 'sum'
        movl  $0, -92(%rbp)             # move the immediate value '0' to variable 'i'
        jmp     .L5                     # jump to label '.L5'
.L6:                                    # Label '.L6'
        movl  -92(%rbp), %eax           # move the value of variable 'i' to register '%eax'
        cltq                            # convert doubleword in '%eax' to quadword in '%rax'
        movl  -64(%rbp,%rax,4), %eax    # move the value of 'arr[i]' to register '%eax'
        addl  %eax, -88(%rbp)           # add the value of 'arr[i]' to variable 'sum'
        movl  -92(%rbp), %eax           # move the value of variable 'i' to register '%eax'
        cltq                            # convert doubleword in '%eax' to quadword in '%rax'
        movl  -88(%rbp), %edx           # move the value of variable 'sum' to register '%edx'
        movl  %edx, -32(%rbp,%rax,4)    # move the value in register '%edx' to 'acc[i]'
```

```
        addl    $1, -92(%rbp)           # increment the value of variable 'i' by 1
.L5:                                    # Label '.L5'
        movl    -92(%rbp), %eax         # move the value of variable 'i' to register '%eax'
        cmpl    -68(%rbp), %eax         # compare value of variable 'i' with 'n'
        jl      .L6                     # jump to '.L6' if 'i' is less than 'n'
        leaq    .LC0(%rip), %rdi        # load effective address, memory '.LC0(%rip)' to '%rdi'
        movl    $0, %eax                # move the immediate value '0' to register '%eax'
        call    printf@PLT              # call printf
        movl    $0, -92(%rbp)           # move the immediate value '0' to variable 'i'
        jmp     .L7                     # jump to label '.L7'
.L8:                                    # Label '.L8'
        movl    -92(%rbp), %eax         # move the value of variable 'i' to register '%eax'
        cltq                            # convert doubleword in '%eax' to quadword in '%rax'
        movl    -32(%rbp,%rax,4), %eax  # move the value of 'acc[i]' to register '%eax'
        movl    %eax, %esi              # move the value in register '%eax' to register '%esi'
        leaq    .LC1(%rip), %rdi        # load effective address, memory '.LC1(%rip)' to '%rdi'
        movl    $0, %eax                # move the immediate value '0' to register '%eax'
        call    printf@PLT              # call printf
        addl    $1, -92(%rbp)           # increment the value of variable 'i' by 1
.L7:                                    # Label '.L7'
        movl    -92(%rbp), %eax         # move the value of variable 'i' to register '%eax'
        cmpl    -68(%rbp), %eax         # compare value of variable 'i' with 'n'
        jl      .L8                     # jump to '.L8' if 'i' is less than 'n'
        movl    $10, %edi               # move the immediate value '10' to register '%edi'
        call    putchar@PLT             # call putchar
        jmp     .L9                     # jump to label '.L9'
.L3:                                    # Label '.L3'
        movl    $0, -76(%rbp)           # move the immediate value '0' to variable 'zc'
        movl    -76(%rbp), %eax         # move the value of variable 'zc' to register '%eax'
        movl    %eax, -80(%rbp)         # move the value in register '%eax' to variable 'nc'
        movl    -80(%rbp), %eax         # move the value of variable 'nc' to register '%eax'
        movl    %eax, -84(%rbp)         # move the value in register '%eax' to variable 'pc'
        movl    $0, -92(%rbp)           # move the immediate value '0' to variable 'i'
        jmp     .L10                    # jump to label '.L10'
.L14:                                   # Label '.L14'
        movl    -92(%rbp), %eax         # move the value of variable 'i' to register '%eax'
        cltq                            # convert doubleword in '%eax' to quadword in '%rax'
        movl    -64(%rbp,%rax,4), %eax  # move the value of 'arr[i]' to register '%eax'
        testl   %eax, %eax              # set condition codes according to '%eax' & '%eax'
        jns     .L11                    # jump to '.L11' if 'arr[i]' is non-negative
```

```
        addl    $1, -80(%rbp)           # increment the value of variable 'nc' by 1
        jmp     .L12                    # jump to label '.L12'
.L11:                                   # Label '.L11'
        movl    -92(%rbp), %eax         # move the value of variable 'i' to register '%eax'
        cltq                            # convert doubleword in '%eax' to quadword in '%rax'
        movl    -64(%rbp,%rax,4), %eax  # move the value of 'arr[i]' to register '%eax'
        testl   %eax, %eax              # set condition codes according to '%eax' & '%eax'
        jle     .L13                    # jump to '.L13' if 'arr[i]' is <= zero
        addl    $1, -84(%rbp)           # increment the value of variable 'pc' by 1
        jmp     .L12                    # jump to label '.L12'
.L13:                                   # Label '.L13'
        addl    $1, -76(%rbp)           # increment the value of variable 'zc' by 1
.L12:                                   # Label '.L12'
        addl    $1, -92(%rbp)           # increment the value of variable 'i' by 1
.L10:                                   # Label '.L10'
        movl    -92(%rbp), %eax         # move the value of variable 'i' to register '%eax'
        cmpl    -68(%rbp), %eax         # compare value of variable 'i' with 'n'
        jl      .L14                    # jump to '.L14' if 'i' is less than 'n'
        movl    -80(%rbp), %eax         # move the value of variable 'nc' to register '%eax'
        movl    %eax, %esi              # move the value in register '%eax' to register '%esi'
        leaq    .LC2(%rip), %rdi        # load effective address, memory '.LC22(%rip)' to '%rdi'
        movl    $0, %eax                # move the immediate value '0' to register '%eax'
        call    printf@PLT              # call printf
        movl    -76(%rbp), %eax         # move the value of variable 'zc' to register '%eax'
        movl    %eax, %esi              # move the value in register '%eax' to register '%esi'
        leaq    .LC3(%rip), %rdi        # load effective address, memory '.LC3(%rip)' to '%rdi'
        movl    $0, %eax                # move the immediate value '0' to register '%eax'
        call    printf@PLT              # call printf
        movl    -84(%rbp), %eax         # move the value of variable 'pc' to register '%eax'
        movl    %eax, %esi              # move the value in register '%eax' to register '%esi'
        leaq    .LC4(%rip), %rdi        # load effective address, memory '.LC4(%rip)' to '%rdi'
        movl    $0, %eax                # move the immediate value '0' to register '%eax'
        call    printf@PLT              # call printf
        movl    $10, %edi               # move the immediate value '10' to register '%edi'
        call    putchar@PLT             # call putchar
        nop                             # no operation
.L9:                                    # Label '.L9'
        movl    $0, %eax                # move the immediate value '0' to register '%eax'
        movq    -8(%rbp), %rcx
        xorq    %fs:40, %rcx
```

```
        je      .L16
        call    __stack_chk_fail@PLT
.L16:                                       # Label '.L16'
        leave                               # set stack pointer as frame pointer,pop top of stack into '%rbp'
        .cfi_def_cfa 7, 8
        ret                                 # pop return address from stack
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0"
        .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:
```

ANALYSIS & OBSERVATIONS :

Assembly Level Program is generated by using the following command in Linux terminal :
gcc -S nameOfFile.c

The file ‘nameOfFile.s’ contains the generated Assembly Level Program.

At the beginning of the file, the labels LC0, LC1, .. LC4 stores the details required to print the strings given in printf statement.

Then the main code begins.

Here, the main function's frame pointer, i.e. %rbp is pushed onto the stack. Also, the offsets are set. Then the stack pointer, i.e. %rsp, is set to the frame pointer, i.e. %rbp.

Then, the variables like 'choose', 'n' and the array 'arr[]' are initialised using the movl instruction which assigns immediate values to the offsets of the required variables.

Variable name	**Offset**
choose	-72(%rbp)
n	-68(%rbp)
i	-92(%rbp)
arr	-64(%rbp)
acc	-32(%rbp)
sum	-88(%rbp)
pc	-84(%rbp)
nc	-80(%rbp)
zc	-76(%rbp)

As mentioned above, the variables are allocated space contiguously in the order of declaration / definition in the C program. Here, the stack grows downwards, which is indicated by the negative sign in the offset.

Then, the variable 'choose' is compared with 1 and if they are equal, the execution control jumps to .L2, else if it is equal to 2, the execution control jumps to .L3, else default statement is executed.

If the **CASE 1** is executed, the labels .L2, .L5, .L6, .L7, .L8, .L9, .L16 come into picture.

.L2 : This label initialises 'sum' and 'i' and passes the control to .L5

.L5 : It stores the value of 'i' in a temporary register %eax, and compares it with 'n'
If (i < n) control jumps to .L6
Then, the required printf statement gets executed and control jumps to .L7

.L6 : It represents the body of the for loop.
It fetches the value of arr[i] and adds the value to variable 'sum' which is used as an accumulator. Then, the value of 'sum' is stored in acc[i]. Then, the value of 'i' is incremented by 1 and according to it's value being < n, control is passed accordingly.

.L7 : It stores the value of ‘i’ in a temporary register %eax, and compares it with ‘n’
If (i < n) control jumps to .L8
Then, a new line is printed using printf and control is jumped to .L9

.L8 : It fetches the value of acc[i] and prints it’s value using printf. Then, it increments the value of ‘i’ by 1

.L9 : It finishes the execution of switch case and passes control to .L16

.L16 : It ends the main function execution by setting stack pointer as frame pointer and popping the top of the stack into ‘%rbp’.

If the **CASE 2** is executed, the labels .L3, .L10, .L11, .L12, .L13, .L14 come into picture.

.L3 : It initialises ‘zc’, ‘nc’, ‘pc’ and ‘i’ to zero and gives control to .L10

.L10 : It stores the value of ‘i’ in a temporary register %eax, and compares it with ‘n’
If (i < n) control jumps to .L14
Then, the three printf statements are printed to show the values of ‘zc’, ‘nc’ and ‘pc’.
Then, a new line is printed using printf.

.L14 : It represents the if case inside the for loop.
If (arr[i] is non-negative) control jumps to .L11
Then, the value of ‘nc’ is incremented by 1 and control jumps to .L12

.L11 : It represents the else if case inside the for loop.
If (arr[i] <= 0) control jumps to .L13
Then, the value of ‘pc’ is incremented by 1 and control jumps to .L12

.L12 : It increments the value of ‘i’ which is a counter for the ‘for loop’.

.L13 : It represents the else inside the for loop.

Then control comes to main, and then to .L9 which then passes control to .L16

.L9 : It finishes the execution of switch case and passes control to .L16

.L16 : It ends the main function execution by setting stack pointer as frame pointer and popping the top of the stack into ‘%rbp’.

THE END