Name: SUBHENDU MAJI Reg. No.: 18ETCS002121

Laboratory 6

Title of the Laboratory Exercise: Sorting

1. Introduction and Purpose of Experiment

   Students will create assembly code with sorting techniques and nested loops

2. Aim and Objectives

   Aim

   To develop assembly language program to perform sorting using nested loop structures

   Objectives

   At the end of this lab, the student will be able to

   - use nested loops in assembly
   - perform sorting in ascending/ descending order
   - Build complex looping logic in assembly language

3. Experimental Procedure

   1. Write algorithm to solve the given problem
   2. Translate the algorithm to assembly language code
   3. Run the assembly code in GNU assembler
   4. Create laboratory report documenting the work

4. Questions

   Develop an assembly language program to perform the following

   1. To design calculator to perform all arithmetic operations based on input given by user.
   2. To perform SWAP operation using Logical instructions
   3. To compute factorial of a number.
   4. To find second smallest number in an unsorted array.

5. Calculations/Computations/Algorithms

```
# arithmetic opeartion
.section .data
num1:
    .int 10
num2:
    .int 20
oprtn:
    .int 1

.section .bss

.section .text

.globl _start

# function for system exit code
_ret:
    movq    $60, %rax              # sys_exit
    movq    $0, %rdi               # exit code
    syscall

# driver function
_start:
    movl num1,%ecx;
    movl num2,%edx;

    cmp $1,oprtn
    je add_loop

    cmp $2,oprtn
    je sub_loop

    cmp $3,oprtn
    je mul_loop

    cmp $4,oprtn
    je div_loop

    add_loop:
        addl %ecx,%edx # edx = edx + ecx
        jmp _end
    sub_loop:
        subl %ecx,%edx # edx = edx - ecx
        jmp _end
    mul_loop:
        movl num2,%eax
        mull num1  # eax = eax * num1
        jmp _end
    div_loop:
        movl num2,%eax
        divl num1   # eax = eax / num1
        jmp _end

    _end:

    syscall
    call _ret           # exit
```

Figure 1 ASM code to design calculator to perform all arithmetic operations based on input given by user.

```
# SWAP
.section .data
x:
    .int 10
y:
    .int 20
.section .bss

.section .text

.globl _start

# function for system exit code
_ret:
    movq    $60, %rax               # sys_exit
    movq    $0, %rdi                # exit code
    syscall

# driver function
_start:
    movl x,%eax
    movl y,%ebx

    xorl %eax,%ebx
    xorl %ebx,%eax
    xorl %eax,%ebx

    movl %eax,x
    movl %ebx,y
    syscall
    call _ret           # exit
```

Figure 2 ASM code to perform SWAP operation using Logical instructions

```
# Factorial
.section .data
n:
    .int 5

.section .bss

.section .text

.globl _start

# function for system exit code
_ret:
    movq    $60, %rax               # sys_exit
    movq    $0, %rdi                # exit code
    syscall

# driver function
_start:
    movl n, %ebx
    movl %ebx, %eax
repeat:
    subl $1, %ebx
    mull %ebx
    cmp $1, %ebx
    je _end
    jmp repeat
_end:

    syscall
    call _ret           # exit
```

Figure 3 ASM code to compute factorial of a number.

```
# second smallest in unsorted array
.section .data
array:
    .int 7,3,5,1,6,9
array2:
    .int 0,0,0,0,0,0
sec_small:
    .int 0
.section .bss

.section .text

.globl _start

# function for system exit code
_ret:
    movq    $60, %rax               # sys_exit
    movq    $0, %rdi                # exit code
    syscall

# driver function

_start:
    movl $0,%ebx
    movl $0,%ecx
    movl $0,%edx
loop1:
    movl array(,%ecx,4),%eax
    cmpl %ebx,%eax
    je loop2
inc:
    addl $1,%ecx
    cmpl $6,%ecx
    jne loop1
    addl $1,%ebx
    movl $0,%ecx
    cmpl $16,%eax
    jne loop1
loop2:
    movl %eax,array2(,%edx,4)
    addl $1,%edx
    cmpl $6,%edx
    jne inc
    movl $1,%edx
    movl array2(,%edx,4),%eax
    movl %eax,sec_small


    syscall
    call _ret           # exit
```

Figure 4 ASM code to find second smallest number in an unsorted array.

6. Presentation of Results

```
Reading symbols from lab6a...done.
(gdb) b 55
Breakpoint 1 at 0x40011e: file lab6a.s, line 55.
(gdb) r
Starting program: /mnt/d/ruas/sem 03/MP lab/programs/lab6a

Breakpoint 1, _end () at lab6a.s:56
56          syscall
(gdb) i r eax
eax            0xc8     200
(gdb)
```

Figure 5 output to design a calculator to perform all arithmetic operations based on input given by user.

```
Reading symbols from lab6b...done.
(gdb) b 30
Breakpoint 1 at 0x4000e2: file lab6b.s, line 30.
(gdb) r
Starting program: /mnt/d/ruas/sem 03/MP lab/programs/lab6b

Breakpoint 1, _start () at lab6b.s:30
30          syscall
(gdb) p (int)x
$1 = 20
(gdb) p (int)y
$2 = 10
(gdb)
```

Figure 6 output of program to perform SWAP operation using Logical instructions

```
Reading symbols from lab6c...done.
(gdb) b 29
Breakpoint 1 at 0x4000d5: file lab6c.s, line 29.
(gdb) r
Starting program: /mnt/d/ruas/sem 03/MP lab/programs/lab6c

Breakpoint 1, _end () at lab6c.s:30
30          syscall
(gdb) i r eax
eax            0x78     120
(gdb)
```

Figure 7 output of code to compute factorial of a number

```
Reading symbols from lab6d...done.
(gdb) b 48
Breakpoint 1 at 0x400114: file lab6d.s, line 48.
(gdb) r
Starting program: /mnt/d/ruas/sem 03/MP lab/programs/lab6d

Breakpoint 1, loop2 () at lab6d.s:49
49          syscall
(gdb) p (int)sec_small
$1 = 3
(gdb)
```

Figure 8 output of program to find second smallest number in an unsorted array.

7. Analysis and Discussions

Code	`jcc address`
Example	`jne loop`
Explanation	Performs: Jumps to the address location if the condition is met Here cc = ne, e, ge, g, etc. Description: Checks the state of one or more of the status flags in the EFLAGS register (CF, OF, PF, SF, and ZF) and, if the flags are in the specified state (condition), performs a jump to the target instruction specified by the destination operand. A condition code (cc) is associated with each instruction to indicate the condition being tested for. If the condition is not satisfied, the jump is not performed and execution continues with the instruction following the Jcc instruction.

Code	`cmp op1 op2`
Example	`cmp $0, %eax`
Explanation	Performs: Compares the two operands Description: Compares the first source operand with the second source operand and sets the status flags in the EFLAGS register according to the results. The comparison is performed by subtracting the second operand from the first operand and then setting the status flags in the same manner as the SUB instruction. When an immediate value is used as an operand, it is sign-extended to the length of the first operand.

Code	`xor <source> <destination>`
Example	`xorl $20, %ebx`
Explanation	Performs: `Destination = Destination XOR Source` Description: Performs a bitwise exclusive OR (XOR) operation on the destination (first) and source (second) operands and stores the result in the destination operand location. The source operand can be an immediate, a register, or a memory location; the destination operand can be a register or a memory location. Each bit of the result is 1 if the corresponding bits of the operands are different; each bit is 0 if the corresponding bits are the same.

8. Conclusions

   In the conclusion we have learnt about the different sorting techniques and nested loops using the assembly language program for performing different operation using compare instruction and flag conditions to perform logical operations.

9. Comments

   a. Limitations of Experiments

   Every time we must declare which input from the user has to be executed to perform any arithmetic operation in calculator

   b. Limitations of Results

   None.

   c. Learning happened

   We have learnt to develop assembly language program to perform sorting using nested loop structures and perform sorting in ascending/ descending order and build complex looping logic in assembly language

Signature and date