Computer Architecture Project

Gaurav Shah[1] Gaurav Joshi[2] Rutwik More[3]

1 Project Abstract

In this project (Topic 8), we have made a Linux x/86 program that reads the contents of a file (input.txt), and prints the pairwise inverse of the string kept in it. This was done on an Ubuntu 22.1 and compiled and run on NASM assembler. For this, we externally made the input file, and then coded up an assembly code for the task. The main flow of the program is:-

1. Open the input.txt file
2. Read the string of the file to our buffer character array
3. Main loop with the algorithm for reversing the characters
4. Writing the output on the stdout
5. Closing the file and exiting the program

This implementation works well on Ubuntu, and will possibly work well with Linux, on other similar hardware architectures. This program was written in assembly language, which we had not used before so, it was a healthy challenge for us. In making this implementation, we faced many issues, but we also learned a lot about the assembly language, its semantics and how to think about programming language from the bare metal point of view.

2 Team Details

Name:	**Roll No**	**Branch**
Gaurav Shah	21110064	Ee
Gaurav Joshi	21110065	Cse
Rutwik More	21110133	Cse

3 Introduction

Figure 1: Linux logo

What is assembly language:- Linux x86 assembly language is a low-level programming language. It is used to write programs for Linux-based systems that run on x86 processors, such as AMD or intel. It is the human representation of machine code that the computer's processor executes. It gives the programmers direct control over the hardware of the computer. Some of the areas in which assembly language is used are cryptography, reverse engineering, and security testing. It provides a deep understanding of how computers work at a fundamental level [14, Wiki].

What is linux:- Linux is an open source Operating system that directly manages the systems hardware and resources like CPU, memory etc. Its design is inspired from the Unix system and it is based on the linux kernel. Linux has many popular distributions like Ubuntu, Fedora, Debian, Kali-Linux etc. [13, RedHat]

3.1 Hardware and Software Specifications

Computer model	HP
Computer's specs	x86, Windows 11
Operating system used	Ubuntu (WSL: Windows subsystem for linux)
Compiler Used	GNU

4 Code

```
.section .data
inputfile:
  .ascii "input.txt\0"
fd_in:
  .long 0
array:
  .fill 32, 1, 0

.section .text
.global _start

_start:
  # Open the input file
  movl $5, %eax
  movl $inputfile, %ebx
  movl $0, %ecx
  int $0x80

```

```
    movl %eax, fd_in

    # Read the file contents into the array
    movl $3, %eax
    movl fd_in, %ebx
    movl $array, %ecx
    movl $256, %edx
    int $0x80

    # Reverse the contents pairwise
    movl $0, %edi

  loop:
    movb array(%edi), %al
    cmpb $0, %al
    je done
    addi $1, %edi
    movb array(%edi), %ah
    cmpb $0, %ah
    je done
    xchg %al, %ah
    movb %al, array-1(%edi)
    movb %ah, array(%edi)
    addl $1, %edi
    jmp loop

  done:
    # Write the reversed contents to stdout
    movl $4, %eax
    movl $1, %ebx
    movl $array, %ecx
    movl %edi, %edx
    int $0x80

    # Close the input file
    movl $6, %eax
    movl fd_in, %ebx
    int $0x80

    # Exit the program
    movl $1, %eax
    xorl %ebx, %ebx
    int $0x80
```

Listing 1: Main code

5 Results

For testing the working of our code, we checked it for various different inputs and produced the outputs.

Project file - project.asm

Input file - input.txt

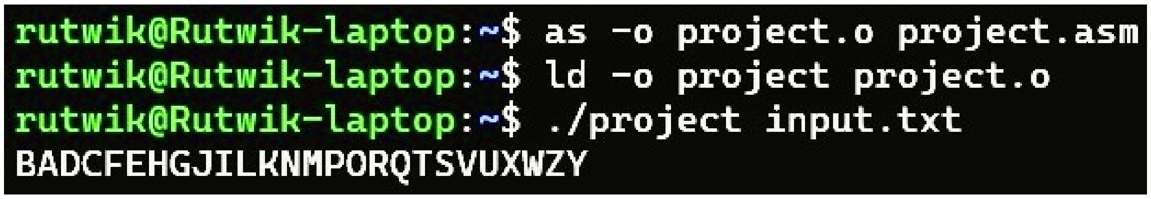
This code reads the contents of the input file named input.txt, character pairwise reverse it. And then writes the reversed contents to standard output.

The terminal command to compile and run the code is:

- as -o project.o project.asm
- ld -o project project.o
- ./project input.txt

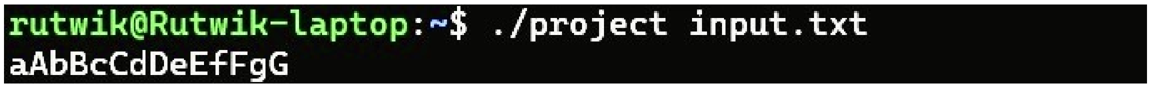
The first command is to make the assemble the code into an object code "project.o", which is then compiled into machine code. [7, Quora]

FIrst input:- ABCDEFGHIJKLMNOPQRSTUVWXYZ



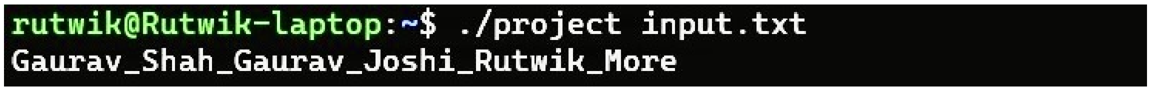
```
rutwik@Rutwik-laptop:~$ as -o project.o project.asm
rutwik@Rutwik-laptop:~$ ld -o project project.o
rutwik@Rutwik-laptop:~$ ./project input.txt
BADCFEHGJILKNMPORQTSVUXWZY
```

Secound input:- AaBbCcDdEeFfGg



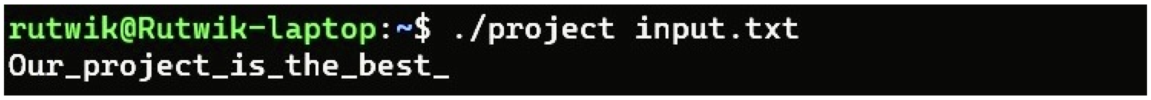
```
rutwik@Rutwik-laptop:~$ ./project input.txt
aAbBcCdDeEfFgG
```

Third input:- aGruvaS_ah_haGruvaJ_soihR_tuiw_koMer



```
rutwik@Rutwik-laptop:~$ ./project input.txt
Gaurav_Shah_Gaurav_Joshi_Rutwik_More
```

Fourth input:- uO_rrpjoce_tsit_ehb_se_t



```
rutwik@Rutwik-laptop:~$ ./project input.txt
Our_project_is_the_best_
```

6 Code Explaination

In this section we will explain all the lines and code logic used in the program. All of the read, write, close file instructions have been taken from the chromium online resource for assembly language commands [1, Chromium].

6.1 data and text sections

```
1  .section .data
2  inputfile:
3    .ascii "input.txt\0"
4  fd_in:
5    .long 0
6  array:
7    .fill 32, 1, 0
8
9  .section .text
10 .globl _start
```

Listing 2: Data and Text sections

This section is for declaring the initialised constants and, starting the text section.

A line-by line explanation is as follows:-

- 1: This creates a new section called ".data", used for declaring the initialized data
- 2: This defines a label called "inputfile" and initializes it with the ASCII string "input.txt".
- 4: This defines a label called "fd_in" and initializes it with the value 0. This label will be used later to store the file handle returned by the "open" system call.
- 6: This defines a label called "array" and reserves 256 bytes of memory for a array to store the contents of the input file.
- 9: This directive creates a new section called ".text", which is used for declaring executable instructions.
- 10: This defines a global symbol called "_start", which is the start point of the program.

[6, Tutorials Point]

6.2 Open the input.txt file

```
1  _start:
2    # Open the input file
3    movl $5, %eax
```

```
  movl $inputfile, %ebx
  movl $0, %ecx
  int $0x80
```

Listing 3: Opening File

This _start marks the beginning of the "_start", ie start of the kernel call

This block of code opens the input file using the "open" system call. Here, we set the syscall number for "open" to %eax (5), the pointer to the inputfile to %ebx (inputfile), and the flags to 0 (which means the file is opened for reading only). Then we use the "int" instruction to invoke the kernel and execute the "open" system call [1, Chromium] [5, Tutorials Point],

Some essential things

- %eax, %ebx, %ecx:
  They are 32-bit general-purpose registers in the x86 architecture. They are part of the set of six general-purpose registers in x86, which are used to hold data and addresses during program execution.
- $inputfile:
  $inputfile is a reference to the address of the string contained in "input.txt" in the .data section of the code.
- $0x80:
  int $0x80 instruction is used to invoke a system call (int means interrupt). When this instruction is executed, which triggers the kernel to execute a specific system call [8, Stack Overflow].

6.3 Read the string of the file to our buffer character array

```
_start:

  movl %eax, fd_in

  # Read the file contents into the buffer
  movl $3, %eax
  movl fd_in, %ebx
  movl $array, %ecx
  movl $256, %edx
  int $0x80
```

Listing 4: Reading File

The first code line moves the file handle returned by the "open" system call from %eax to the "fd_in" variable for later use.

This block of code reads the contents of the input file into the buffer ”array” using the ”read” system call. The ”read” system call takes four arguments: the file handle, the ”array” address, the ”array” size, and a set of flags. Here, we set the syscall number for ”read” to %eax (3), the file handle to %ebx (fd_in), the ”array” address to %ecx (array), and the ”array” size to 256 bytes. Then we use the ”int” instruction to invoke the kernel and execute the ”read” system call [1, Chromium] [5, Tutorials Point].

Some essential things

- $array:
  (”array”) is the array of size 256 bits which is used to store the string in the input file.

6.4 Main loop with the algorithm for reversing the characters

```
_start:

  # Reverse the contents pairwise
  movl $0, %edi

loop:
  movb array(%edi), %al
  cmpb $0, %al
  je done
  movb array+1(%edi), %ah
  cmpb $0, %ah
  je done
  xchg %al, %ah
  movb %al, array(%edi)
  movb %ah, array+1(%edi)
  addl $2, %edi
  jmp loop

```

Listing 5: Main Logic code

This is the main part of the code.We have made a procedure named loop which will help us traverse the array. Here we start taking %edi as an index variable (initialised to zero). We use this variable to iterate over the array. The first and second character to be reversed is taken in the % al and %ah respectively. Then the content stored in these registers is exchanged and fed into the array back. This process continues until our index variable reaches the end, checked by the je conditional instruction. The loop is run by using incrementing the value in %edi, by one-one and then finally using jmp conditional to move back to the start of the procedure [1, Chromium] [5, Tutorials Point].

Some essential things

- procedure:
  The loop which we have made is a procedure. Procedures or subroutines are very important in assembly language, as the assembly language programs tend to be large in size, and they help simplify the code and make proper flows of code. [4, Tutorials Point]
- cmpb $0, %al:
  compares the lower 8 bits of %al register with the value 0. This instruction is used to check if the index variable is reached at the end of the input string or not. As last of the input string is represented by null character. If the value is zero, then it sets the zero flag as one.
- je done:
  It checks the zero flag (ZF) in the processor's flags register. If the zero flag is set, it means that the comparison (cmpb $0, %al) in the previous instruction resulted in equality. In other words, the current character in %al register is a null character, which marks the end of the file contents in the array. We get out of the current loop. [3, Tutorials Point]
- movb and movl:
  movb is for moving 8 bits and movl is for moving 32 bits, [2, Quora]
- xchg %al, %ah:
  The instruction xchg %al, %ah exchanges the values in the al and ah registers. Since al holds the current character and ah holds the next character in the array, this instruction swaps the characters pairwise to reverse them.
- jmp loop:
  When one pair of characters is reversed and index variable is not reached at the end of the character array then this instrucution is executed. This logical instruction make program to jump back to the loop. [3, Tutorials Point]

6.5 Writing the output on the stdout

```
_start:

done:
  # Write the reversed contents to stdout
  movl $4, %eax
  movl $1, %ebx
  movl $array, %ecx
  movl %edi, %edx
  int $0x80
```

Listing 6: Writing Output

These lines perform a write syscall to output the pairwise reversed array to stdout. This part of the code writes the contents of the "array" to stdout, which is represented by file handle 1. The "array" address is stored in %ecx, and the "array" size is stored in %edx, which is the number of bytes to be written [1, Chromium] [5, Tutorials Point].

6.6 Closing the file and exiting the program

```
_start:

  # Close the input file
  movl $6, %eax
  movl fd_in, %ebx
  int $0x80

  # Exit the program
  movl $1, %eax
  xorl %ebx, %ebx
  int $0x80

```

Listing 7: Closing file and program

These lines perform a close syscall to close the input file. The file handle is passed in %ebx. These lines perform an exit syscall to exit the program. $1 is the exit number used in the %eax register. The exit code i.e. (0) is passed in %ebx. In this case, the exit code is set to 0 to indicate successful closing of the program [1, Chromium] [5, Tutorials Point].

7 Errors Encountered

- Segmentation fault:
  We encountered many segmentation faults due to syntax error. When we forgot to write the one of the directives in the top of the code or many other syntax then it resulted in the segmentation fault.
- Wrong String Output:
  We had some issues in our code logic as, we mistakenly didn't shifted the index variable to next variable and because of which it resulted into printin A at alternate positions.
  So for the input "ABCDEFGHIJKLMNOPQRSTUVWXYZ" , we were getting "ABADAFAHAJALANAPARATAVAXAZA"
- Odd string Length:
  A problem in the code, is for a string of odd length, in which case, all of the string is pairwise flipped in the correct way, but the last character of the string is printed in a new line.(This is due to the fact that the last character gets exchanged with enter)

8 Possible Extensions

There are a lot of possibilities of extending this project, than the currently given question.

- Making the input file using Assembly:
  In this project, we have made the input file manually, and we also add and remove text to it manually. A good alternative approach could be to write an assembly program that creates a text file input.txt, and writes a string of characters into it.
- Extending to triplets:
  We can also extend this approach to produce some similar and equally interesting patterns like making a code which flips two characters and then leaves one and then flips two eg. ”abcdef” to ”bacedf”

9 Work Distribution

Name	**Work**
Gaurav Shah	Provided background information on the problem the program aims to solve. Studied the Linux toolchain and explained it to the other group members. Helped in the debugging of the code.
Gaurav Joshi	Summarized the project and its key findings. Checked if there are any limitations of the program. He also asked if there were any possible extensions of this project. He also contributed to the report writing of the project using the latex software. Helped in debugging the code.
Rutwik More	Wrote the main skeleton of the program. Explained how to approach the problem. Collected essential resources from the internet for the deep learning of the project and assembly language.

10 Acknowledgments

We would like to express our gratitude to all of our team members who have put immense effort in completing this project well before the deadline. Secondly we would like to thanks Prof. Rajat Moona and Prof. Sameer Kulkarni for providing us a opportunity to work on this project with necessary guidance. Lastly we would like to thank all the educators who put up the available online resources, tutorials and Youtube videos which helped us with better understanding of the project and deepened our knowledge in linux x86 assembly programming.

References

[1] ChromiumOs docs, Website link

[2] Quora Answer Search, ”What is the difference between movq and movl assembly instruction?” Website link

[3] Tutorials Point, Assembly-Logical Instructions, Website Link

[4] Tutorials Point, Assembly-Procedures Website Link

[5] Tutorials Point, Assembly-File Management Website Link

[6] Tutorials Point, Assembly-Basic Syntax Website Link

[7] Quora Answer Search, ”How do you compile in Assembly” Website link

[8] StackOverflow, ”What does ”int 0x80” mean in assembly code?” Website link

[9] Youtube Video, File Handling in Assembly, Video Link

[10] Youtube Video, You Can Learn x86 Assembly in 10 Minutes, Video Link

[11] Book, x86-64 Assembly Language Programming with Ubuntu, Pdf link

[12] Book,Programming from the Ground Up by Jonathan Bartlett

[13] Redhat Website, Website Link

[14] ”Assembly Language”, Wikipedia, Wiki link