

END- SEMESTER PROJECT REPORT 2025

PREPARED BY:
LOUAFI SOHAIB

PRESENTED BY:
MECHIR ISLAM

NASM Mini-Project Report

I-Reverse Array :

1. Objective:

The goal of this project is to implement and optimize low-level array operations using NASM x86-64 assembly. Through this exercise, we aim to enhance our understanding of memory manipulation, register usage, and system-level programming.

2. Project Description:

This project includes the development of two fundamental routines in NASM:

1. reverse_array_asm

- **Description:** Reverses the contents of a 32-bit integer array in place.
- **Inputs:**
 - RDI: pointer to the array
 - RSI: number of elements
- **Behavior:**
The function swaps elements from both ends moving toward the center. It handles edge cases where the array is empty or has only one element.

2. find_second_largest_asm

- **Description:** Identifies the second largest value in an integer array.
- **Inputs:**
 - RDI: pointer to the array
 - RSI: number of elements
- **Output:**
 - EAX: second largest value, or -1 if invalid input
- **Behavior:**
Traverses the array while tracking the largest and second largest values. It handles the case of insufficient array length by returning -1.

All functions are implemented using the System V AMD64 calling convention, making them callable from C programs.

3. Environment Setup:

- **Operating System:** Kali Linux
- **Assembler:** NASM
- **Compiler/Linker:** GCC
- **Text Editor:** nano/vim
- **Build Automation:** Optional use of Makefile

4. Testing and Validation:

Each function was tested using a dedicated C test file. Test inputs included both normal and edge cases, such as:

- Arrays of size 0 and 1
- Arrays with repeated elements
- Negative values

Example test in C:

```
int arr[] = {7, 2, 10, 5};

reverse_array_asm(arr, 4);

// arr becomes {5, 10, 2, 7}

int result = find_second_largest_asm(arr, 4);

// result = 10
```

5. Challenges Encountered:

- **Memory Addressing:** Computing offsets correctly using `rdi + rcx*4`
- **Edge Case Handling:** Avoiding invalid memory access on small arrays
- **Register Management:** Preserving necessary registers as per the calling convention
- **Flow Control:** Designing clean conditional logic with `cmp`, `jge`, `je`, etc.

6. Optimizations and Design Decisions:

- Efficient swapping using `mov` and `xchg` instructions
- Pointer arithmetic to reduce memory access overhead
- Use of manual loop control for clarity and predictability
- Return values via `eax` to match C function expectations

7. Conclusion:

This project offered a hands-on opportunity to work at the hardware-near level using assembly language. It allowed us to practice low-level debugging, understand data layout in memory, and manage performance-critical control flow.

Key Achievements:

- Wrote and tested two NASM routines
- Ensured compliance with Linux calling conventions
- Validated correctness through integration with C

II-Find second largest element in array

1. Project Objective

The goal of this project is to implement and optimize low-level array operations using NASM x86-64 assembly language. This exercise provides hands-on experience with:

- Memory manipulation at the system level
- Efficient register usage
- System-level programming concepts

- Integration with high-level languages (C)

2. Project Description

The project implements two core array operations in x86-64 assembly:

`reverse_array_asm`

Purpose: Reverses a 32-bit integer array in place

Inputs:

- RDI: Pointer to array
- RSI: Number of elements

Behavior:

- Swaps elements from both ends moving toward center
- Handles edge cases (empty/single-element arrays)
- Performs in-place modification

`find_second_largest_asm`

Purpose: Finds the second largest value in an integer array

Inputs:

- RDI: Pointer to array
- RSI: Number of elements

Output:

- EAX: Second largest value (-1 for invalid input)

Behavior:

- Tracks largest and second largest during traversal
- Returns -1 if array has fewer than 2 elements
- Handles duplicate values

3. Development Environment

- **OS:** Kali Linux
- **Assembler:** NASM (Netwide Assembler)
- **Compiler/Linker:** GCC
- **Editor:** nano/vim
- **Build:** Makefile for automation

4. Testing Methodology

Test Cases Included:

- Empty arrays and single-element arrays
- Arrays with duplicate values
- Arrays containing negative numbers
- Typical use cases

Sample Test (C Driver Program):

- `int arr[] = {7, 2, 10, 5};`
- `reverse_array_asm(arr, 4);` // Result: {5, 10, 2, 7}
- `int result = find_second_largest_asm(arr, 4);` // Returns 10

5. Technical Challenges

1. **Memory Addressing:**
 - Correct calculation of element offsets using `rdi + rcx*4`
 - Ensuring proper alignment for 32-bit operations
2. **Edge Case Handling:**
 - Preventing invalid memory access on small arrays
 - Special handling for arrays with <2 elements
3. **Register Management:**
 - Compliance with System V AMD64 calling convention
 - Efficient use of available registers
4. **Flow Control:**
 - Clean conditional logic using `cmp`, `jge`, `je`
 - Optimized loop structures

6. Optimizations & Design Choices

- **Efficient Swapping:** Used `mov` and `xchg` instructions
- **Pointer Arithmetic:** Minimized memory access overhead
- **Manual Loop Control:** Improved predictability
- **Register Usage:** Maximized register-based operations
- **Return Values:** Used `EAX` for C compatibility

7. Key Learnings & Outcomes

- Gained practical experience with low-level memory manipulation
- Developed understanding of x86-64 calling conventions
- Learned to integrate assembly with high-level languages
- Improved debugging skills at the assembly level

III-Factorial

Purpose:

To compute the **factorial** of a non-negative integer using x86-64 assembly ($n! = n \times (n-1) \times \dots \times 1$).

Input:

- EDI: an integer n whose factorial is to be calculated.

Output:

- RAX: the result of $n!$ (as a 64-bit integer).

Logic Overview:

1. The function initializes the result in EAX to 1, assuming $0! = 1$ by definition.
2. It checks if the input number is **less than or equal to 0**:
 - If so, it skips the loop and directly returns 1.
3. If the input is greater than 0:
 - It copies the input value into the counter register ECX.
4. The function enters a loop:
 - On each iteration, it multiplies the current result (RAX) by the counter (RCX).
 - It then decrements the counter.
 - The loop continues until the counter becomes zero.
5. Once finished, the function returns with RAX holding the computed factorial value.

Example:

Input:

`int n = 5;`

Calculations:

$RAX = 1 \times 5 \times 4 \times 3 \times 2 \times 1 = 120$

Output:

$RAX = 120$

IV- Reverse a string in place

Purpose:

To reverse a null-terminated string **in place**—i.e., modify the original string so that the characters appear in reverse order.

Input:

- RDI: a pointer to the input string (`char*`)

Output:

- The original string is modified directly in memory to become reversed.

- The function does not return a value; the change happens in place.

Logic Overview:

1. **Calculate the length of the string** by calling a helper function (`strlen_asm`). The length does not include the null terminator.
2. If the string is **empty** or has **only one character**, there's nothing to reverse, so the function exits immediately.
3. Two pointers are initialized:
 - One pointing to the **first character** of the string.
 - One pointing to the **last character** (just before the null terminator).
4. The function performs **in-place swapping**:
 - It swaps the character at the beginning with the one at the end.
 - Then it moves the start pointer forward and the end pointer backward.
 - This process repeats until both pointers meet or cross, which means the reversal is complete.
5. The function ends, and the string is now reversed in memory.

Example:

Input:

```
char str[] = "hello";
```

Output (after `strrev_asm(str)`):

```
str → "olleh"
```

VI- Calculate length of null-terminated string

Purpose:

To compute the length of a null-terminated string (just like the C standard function `strlen`), without including the null terminator (`\0`).

Input:

- `RDI`: A pointer to the start of the null-terminated string.

Output:

- `RAX`: The number of characters in the string (excluding the null terminator).

Logic Overview:

1. The function starts by clearing `RAX`, which will serve as a **counter** for the string length.
2. It then enters a loop that:
 - Checks the byte at the current position (`[RDI + RAX]`).
 - If this byte is **not zero**, it increments the counter (`RAX`) and continues.

- If the byte **is zero** (i.e., the null terminator `\0` is reached), the loop ends.
- 3. When the loop finishes, RAX holds the length of the string (number of characters before the null terminator).
- 4. The function returns with the length in RAX.

Example:

Input:

```
char* str = "NASM";
```

Memory layout:

Address → Content

[0] = 'N'

[1] = 'A'

[2] = 'S'

[3] = 'M'

[4] = '\0'

Output:

RAX = 4

VI-debugging

step:

1 Understand the Problem

- Read the error message, log, or unexpected output.
- Reproduce the bug consistently (if possible).
- Identify **where** and **when** the error occurs.

2 Set Up the Debugging Environment

- Use tools like:
 - gdb for C/C++
 - Built-in debuggers in IDEs (e.g., Visual Studio, Eclipse, VSCode)
 - Logging (e.g., printf, console.log)
- If you're using Assembly, tools like **GDB** or **objdump** are useful.

3 Use Breakpoints

- Place breakpoints to **pause** the program at specific lines.
- Inspect variable values and program flow step by step.

4 Trace and Analyze

- Step through the code **line by line**.
- Watch variable changes, register values (in assembly), or memory content.
- Check if logic and control flow behave as expected.

5 Isolate the Bug

- Narrow down the part of code where the issue originates.
- Remove or comment out unrelated sections to focus on the bug.

6 Fix the Bug

- Correct the logic, syntax, or algorithm causing the error.
- Ensure that the fix doesn't break other parts of the code

window.__oai_logHTML?

window.__oai_logHTML():window.__oai_SSR_HTML=window.__oai_SSR_HTML | | Date.now();requestAnimationFrame((function(){window.__oai_logTTI?

window.__oai_logTTI():window.__oai_SSR_TTI=window.__oai_SSR_TTI | | Date.now()))

- step1:

```
moncif@moncif-NS5x-NS7xAU:~/Desktop/0s2_project/arrays$ gdb ./my_arrays_program
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./my_arrays_program...
(gdb) break sum_of_array
Breakpoint 1 at 0x401150
(gdb) disassemble sum_of_array
Dump of assembler code for function sum_of_array:
   0x0000000000401150 <+0>:    xor     %rcx,%rcx
   0x0000000000401153 <+3>:    xor     %rax,%rax
End of assembler dump.
(gdb) run
Starting program: /home/moncif/Desktop/0s2_project/arrays/my_arrays_program

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for system-supplied DSO at 0x7ffff7fc3000
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Array operations:
Original array: 1 2 3 4 5
```

- step2:

```
Breakpoint 1, 0x0000000000401150 in sum_of_array ()
(gdb) info registers
rax                0x0                0
rbx                0x7fffffffde78       140737488346744
rcx                0x7ffffffd1c574     140737351107956
rdx                0x0                0
rsi                0x4052a0            4215456
rdi                0x7ffff7e05710     140737352062736
rbp                0x7fffffffdd50     0x7fffffffdd50
rsp                0x7fffffffdd38     0x7fffffffdd38
r8                 0x64                100
r9                 0x0                0
r10                0x7ffff7c0ebf0     140737350003696
r11                0x202              514
r12                0x1                1
r13                0x0                0
r14                0x403e00            4210176
r15                0x7ffff7ffd000     140737354125312
rip                0x401150            0x401150 <sum_of_array>
eflags             0x202              [ IF ]
cs                 0x33                51
ss                 0x2b                43
ds                 0x0                0
es                 0x0                0
fs                 0x0                0
gs                 0x0                0
fs_base            0x7ffff7faa740     140737353787200
gs_base            0x0                0
(gdb)
```

- step3:

```

moncif@moncif-NS5x-NS7xAU:~/Desktop/0s2_project/strings$ gdb ./strings_program
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./strings_program...
(gdb) break reverse_string
Breakpoint 1 at 0x4011b1
(gdb) run
Starting program: /home/moncif/Desktop/0s2_project/strings/strings_program

This GDB supports auto-downloading debuginfo from the following URLs:
<https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Integer to string:
Copied string: Hello
Concatenated string: Hello World!

Breakpoint 1, 0x000000004011b1 in reverse_string ()
(gdb) disassemble reverse_string
Dump of assembler code for function reverse_string:
=> 0x000000004011b1 <+0>:    push    %rsi
0x000000004011b2 <+1>:    mov     %rdi,%rsi
0x000000004011b5 <+4>:    call   0x401171 <string_len>
0x000000004011ba <+9>:    pop     %rdi
0x000000004011bb <+10>:   test    %rax,%rax
0x000000004011be <+13>:   je      0x4011d8 <reverse_string.empty>
0x000000004011c0 <+15>:   lea     -0x1(%rsi,%rax,1),%rsi
0x000000004011c5 <+20>:   xor     %rcx,%rcx

```

- stpe4:

```

b) info register
      0x7fffffffdc10      140737488346128
      0x7fffffffde78      140737488346744
      0x0                  0
      0x7fffffffddcd0     140737488346320
      0x7fffffffddcd0     140737488346320
      0x7fffffffdc10      140737488346128
      0x7fffffffdd50      0x7fffffffdd50
      0x7fffffffdbed8     0x7fffffffdbed8
      0x73                 115
      0x0                  0
      0xffffffff          4294967295
      0x202               514
      0x1                 1
      0x0                 0
      0x403e00            4210176
      0x7ffff7ffd000      140737354125312
      0x4011b1            0x4011b1 <reverse_string>
ags  0x246               [ PF ZF IF ]
      0x33                51
      0x2b                43
      0x0                 0
      0x0                 0
      0x0                 0
      0x0                 0
base  0x7ffff7faa740      140737353787200
base  0x0                 0
b) print (char*)$rdi
= 0x7fffffffdc10 "Hello World!"
b) print (char*)$rsi
= 0x7fffffffddcd0 ""
b) stepi
0000000004011b2 in reverse_string ()
b) print (char*)$rdi
= 0x7fffffffdc10 "Hello World!"
b) print (char*)$rdi
= 0x7fffffffdc10 "Hello World!"
b) print (char*)$rsi
= 0x7fffffffddcd0 ""
b) nexti
0000000004011b5 in reverse_string ()
b) print (char*)$rsi
= 0x7fffffffdc10 "Hello World!"

```