GROUP ASSIGNMENT

TECHNOLOGY PARK MALAYSIA

CT124-3-1-ICS

INTEGRATED COMPUTER SYSTEMS

Lecturer Name: DR. HASSAN JAMIL SYED

Intake Code: APU1F2403CS(CYB)

Group Number: Group 28

Group Members:

NO.	NAME	TP NO.
1.	Lee Yi Chern (L)	TP081340
2.	Voon Yue Cheng	TP080509
3.	Bardia Momeni	TP077883
4.	Hwang XiaoShun	TP077723

Date Assigned: 5 October 2024

Date Completed: 26 November 2024

1.0 Table of Contents

2.0 Introduction

This assignment investigates critical aspects of computer system performance and optimization, focusing on low-level programming and system management techniques. There are two primary sections to it: The objective of Section A, which focuses on assembly programming, is to create a Shape Generation System that relies on assembly language to effectively draw simple shapes like triangles, circles, rectangles, and lines. To guarantee flawless operation, the system will integrate error-handling methods and communicate directly with hardware. Section B delves into the research and analysis of branch prediction and caching techniques, emphasizing their importance in maximizing CPU performance, memory management, and I/O operations in resource-constrained environments. A comparison of high-level programming techniques and low-level assembly programming is also included in this section, with a focus on operating system development and a focus on usability, security, and performance. The assignment aims to give students a thorough understanding of both low-level system design and performance-enhancing strategies in contemporary computing by fusing theoretical research with real-world programming problems.

3.0 Assembly Program and Analysis

3.1 Explanation of Assembly Code

```
section .data
    prompt_shape db '          Shape Drawing Implementation', 10, "-----------------------------------------------", 10, '1. Line', 10, '2. Rectangular', >
    prompt_choice db 'Enter your choice (1-6): ', 0
    invalid_choice_msg db 'Invalid choice. Please enter a number between 1 and 5.', 10, 0
    prompt_line_length db "Enter the length of the line (1-9): ", 0
    prompt_rec_length db "Enter the length of rectangle (1-9): ", 0
    prompt_rec_width db "Enter the width of rectangle (1-9): ", 0
    prompt_square db "Enter the length of square (1-9): ", 0
    prompt_circle db "Enter the radius of the circle (1-9): ", 0
    prompt_tri db "Enter the row of triangle (1-9): ", 0
    invalid_length_msg db "Invalid length. Please enter a number between 1 and 9.", 10, 0
    star db "* ", 0
    space db "  ", 0
    newline db 10, 0
```

In section .data, variables with messages wanted to be printed are declared.

```
section .bss
    buffer resb 3                   ; Buffer to store 2 digit (at most) input
    choice resb 1                   ; variable to store menu choice
    line_length resb 1              ; variable to store line length
    rec_length resb 1               ; variable to store rectangle length
    rec_width resb 1                ; variable to store rectangle width
    rec_ori_length resb 1           ; variable to store original length for resetting
    square_length resb 1            ; variable to store square side length
    square_ori_length resb 1        ; variable to store original square length for resetting
    square_width resb 1             ; variable to store square width
    radius resd 1                   ; variable to store circle radius
    x resd 1                        ; variable to store circle x coordinate
    y resd 1                        ; variable to store circle y coordinate
    tri_space resb 1                ; variable to store number of triangle space
    tri_star resb 1                 ; variable to store number of triangle star
    tri_ori_space resb 1            ; variable to store original space number
    tri_ori_star resb 1             ; variable to store original star number
```

In section .bss, variables used to save user's input are declared.

```
section .text
    global _start
```

In section .text, executable code is declared.

```
_start:
    ; Print the shape menu
    mov eax, 4                      ; sys_write
    mov ebx, 1                      ; file descriptor: stdout
    mov ecx, prompt_shape           ; pointer to message
    mov edx, 151                    ; message length
    int 0x80                        ; call kernel

    mov eax, 4
    mov ebx, 1
    mov ecx, prompt_choice
    mov edx, 25
    int 0x80

    ; Read user input for the choice
    mov eax, 3                      ; sys_read
    mov ebx, 0                      ; file descriptor: stdin
    mov ecx, buffer                 ; buffer to store choice
    mov edx, 3                      ; read up to 2 bytes (1-2 digit input + blank)
    int 0x80                        ; call kernel

    ; Call convert_loop to process input in buffer
    call convert_loop               ; convert buffer to integer
    mov [choice], al                ; store the final integer choice in choice

    ; Validate choice and jump to the corresponding function
    cmp al, 1
    je draw_line                    ; choice 1: draw line
    cmp al, 2
    je draw_rectangle               ; choice 2: draw rectangle
    cmp al, 3
    je draw_circle                  ; choice 3: draw circle
    cmp al, 4
    je draw_triangle                ; choice 4: draw triangle
    cmp al, 5
    je draw_square                  ; choice 5: draw square
    cmp al, 6
    je exit
    ; If no match, go to invalid_choice
    jmp invalid_choice
```

In _start, which is when the program is executed, the system shows a message asking the user for the shapes wanted to be printed, then accepts user's choice.

```
convert_loop:
    ; Convert ASCII characters in buffer to integer
    xor eax, eax                    ; clear eax (to accumulate result)
    xor ecx, ecx                    ; clear ecx (index counter for buffer)

convert_digit_loop:
    movzx edx, byte [buffer + ecx]  ; load the current character from buffer
    cmp edx, 0xA                    ; check if we reached a newline character (ASCII LF)
    je convert_done                 ; if newline, end conversion

    sub edx, '0'                    ; convert ASCII character to integer
    imul eax, eax, 10               ; multiply current result by 10
    add eax, edx                    ; add the new digit
    inc ecx                         ; move to the next character in buffer
    jmp convert_digit_loop          ; repeat for next digit

convert_done:
    ret                             ; return to the calling function with result in eax
```

After that, the system brings the choice to convert_loop, where the choice is changed from ASCII character to its numeric value. And it also handles 2-digit user input.

```
exit:
    ; Exit the program
    mov eax, 1                      ; syscall: sys_exit
    xor ebx, ebx                    ; status 0
    int 0x80                        ; make the syscall
```

Then, the system compares the choice with the choices set (1-6). If the user types 1-5, the system shows the user the corresponding message. If the user types 6, the system exits the program.

```
invalid_choice:
    ; Print invalid choice message
    mov eax, 4                      ; sys_write
    mov ebx, 1                      ; file descriptor: stdout
    mov ecx, invalid_choice_msg     ; pointer to invalid choice message
    mov edx, 54                     ; message length
    int 0x80                        ; call kernel

    ; Print two new line
    mov eax, 4
    mov ebx, 1
    mov ecx, newline
    mov edx, 1
    int 0x80
    mov eax, 4
    mov ebx, 1
    mov ecx, newline
    mov edx, 1
    int 0x80

    jmp _start
```

If the user types anything else besides 1-6, the system jumps to invalid_choice, which tells the user that the input is invalid, then loop back to the start and ask user for another input.

```
draw_line:
    ; Prompt for line length
    mov eax, 4                      ; syscall: sys_write
    mov ebx, 1                      ; file descriptor: stdout
    mov ecx, prompt_line_length     ; pointer to prompt line message
    mov edx, 35                     ; Length of the prompt message
    int 0x80

    ; Read line length input
    mov eax, 3                      ; syscall: sys_read
    mov ebx, 0                      ; file descriptor: stdin
    mov ecx, buffer                 ; buffer to store length input
    mov edx, 3                      ; Allow for 1-2 characters input + newline
    int 0x80                        ; call kernel

    ; Convert input to integer
    call convert_loop               ; convert buffer to integer in eax
    mov [line_length], al           ; store the final integer line length in line_length

    ; Validate line length (1-9)
    cmp al, 1
    jl invalid_length               ; if less than 1, invalid length
    cmp al, 9
    jg invalid_length               ; if greater than 9, invalid length

    ; Print the line
    jmp print_line_loop
```

If the user chooses 1, which is line, the system jumps to draw_line, which asks the user the length of the line, and converts the input from ASCII to its numeric value.

```
print_line_loop:
    mov al, [line_length]           ; Get the line length
    cmp al, 0                       ; Check if we need to print more stars
    jle line_print_newline          ; If no more stars, jump to done

    ; Print a star
    mov eax, 4
    mov ebx, 1                      ; stdout
    mov ecx, star                   ; Pointer to star
    mov edx, 2                      ; Length of "* "
    int 0x80                        ; Call kernel

    sub byte [line_length], 1       ; Decrease star count
    jmp print_line_loop             ; Repeat

line_print_newline:
    ; Print new line
    mov eax, 4
    mov ebx, 1
    mov ecx, newline
    mov edx, 1
    int 0x80
    jmp line_done

line_done:
    jmp _start                      ; Go to exit
```

After that, the system compares the length, if the length is (1-9), the system jumps to print_line_loop, which prints the line with * symbol, which number of * shows the length the user has entered. After the line is printed, the system jumps back to the start of the program.

```
invalid_length:
    ; Print invalid length message
    mov eax, 4                      ; syscall: sys_write
    mov ebx, 1                      ; file descriptor: stdout
    mov ecx, invalid_length_msg     ; pointer to invalid length message
    mov edx, 54                     ; message length
    int 0x80                        ; call kernel

    ; Print two new line
    mov eax, 4
    mov ebx, 1
    mov ecx, newline
    mov edx, 1
    int 0x80

    mov eax, 4
    mov ebx, 1
    mov ecx, newline
    mov edx, 1
    int 0x80

    jmp _start
```

However if the user entered length more than 9 or less than 1, the system jumps to invalid_length, where the system tells the user that the length entered is invalid, then jumps back to the start of the program.

```
draw_rectangle:
    ; Prompt for length
    mov eax, 4
    mov ebx, 1
    mov ecx, prompt_rec_length      ; pointer to prompt rectangle length message
    mov edx, 36                     ; Length of the prompt message
    int 0x80

    ; Read length input
    mov eax, 3
    mov ebx, 0
    mov ecx, buffer
    mov edx, 3                      ; Allow for 2 characters input
    int 0x80

    call convert_loop               ; convert buffer to integer in eax
    mov [rec_length], al            ; store the final integer line length in line_length
    mov [rec_ori_length], al        ; Save original length

    ; Validate line length (1-9)
    cmp al, 1
    jl invalid_length               ; if less than 1, invalid length
    cmp al, 9
    jg invalid_length               ; if greater than 9, invalid length

    ; Prompt for width
    mov eax, 4
    mov ebx, 1
    mov ecx, prompt_rec_width
    mov edx, 35                     ; Length of the prompt message
    int 0x80

    ; Read width input
    mov eax, 3
    mov ebx, 0
    mov ecx, buffer
    mov edx, 3                      ; Allow for 2 characters input
    int 0x80

    call convert_loop               ; convert buffer to integer in eax
    mov [rec_width], al             ; store the final integer line length in line_length

    ; Validate line length (1-9)
    cmp al, 1
    jl invalid_length               ; if less than 1, invalid length
    cmp al, 9
    jg invalid_length               ; if greater than 9, invalid length
```

If the user chooses 2, which is rectangle. The system asks the user to enter the length of the rectangle and change the length from ASCII to its numeric value. If the length is less than 1 or more than 9, the system jumps to invalid_length. If the length is 1-9, then the system asks the user to enter the width of the rectangle and change the width from ASCII to its numeric value.

```
rectangle_loop_width:
    mov al, [rec_width]
    cmp al, 0
    jle rectangle_done              ; If width is zero, exit

rectangle_loop_length:
    mov al, [rec_length]
    cmp al, 0
    jle rectangle_newline           ; If length is zero, print newline

    ; Print a star
    mov eax, 4
    mov ebx, 1
    mov ecx, star
    mov edx, 2                      ; Length of "* "
    int 0x80

    ; Decrement length
    sub byte [rec_length], 1        ; Decrease star count
    jmp rectangle_loop_length       ; Repeat
```

```
rectangle_newline:
    ; Print a newline character
    mov eax, 4
    mov ebx, 1
    mov ecx, newline
    mov edx, 1
    int 0x80

    ; Reset length to original value for the next line
    mov al, [rec_ori_length]
    mov [rec_length], al

    ; Decrement width
    sub byte [rec_width], 1         ; Decrease width count
    jmp rectangle_loop_width

rectangle_done:
    jmp _start
```

If the width is (1-9), the system jumps to rectangle_loop_width, which prints the rectangle row by row. After the rectangle is printed, the system loops back to the start of the program.

```
draw_circle:
    ; Print the circle prompt
    mov eax, 4                      ; sys_write
    mov ebx, 1                      ; file descriptor (stdout)
    mov ecx, prompt_circle          ; pointer to the prompt string
    mov edx, 37                     ; length of the prompt string
    int 0x80

    ; Read the radius from the user
    mov eax, 3                      ; sys_read
    mov ebx, 0                      ; file descriptor (stdin)
    mov ecx, buffer                 ; buffer to store radius
    mov edx, 3                      ; read up to 2 bytes
    int 0x80

    ; Convert input to integer
    call convert_loop               ; convert buffer to integer
    mov [radius], al                ; store the final integer radius in "radius"

    ; Validate line length (1-9)
    cmp al, 1
    jl invalid_length               ; if less than 1, invalid length
    cmp al, 9
    jg invalid_length               ; if greater than 9, invalid length

    ;set y = -radius
    mov ecx, [radius]               ; ecx = radius
    neg ecx                         ; ecx = -radius
    mov [y], ecx                    ; y = -radius
```

If the user chooses 3, which is circle. The system asks the user for the radius of the circle and converts it from ASCII to its numeric value. If the radius is more than 9 or less than 1, then the system jumps to invalid_length.

```
outer_loop:
    ; Check if y > radius
    mov eax, [y]
    cmp eax, [radius]
    jg end_outer_loop

    ; Inner loop for x = -radius to radius
    mov ecx, [radius]               ; ecx = radius
    neg ecx                         ; ecx = -radius
    mov [x], ecx                    ; x = -radius
```

```
inner_loop:
    ; Check if x > radius
    mov eax, [x]
    cmp eax, [radius]
    jg end_inner_loop

    ; Calculate x^2 + y^2
    mov eax, [x]                    ; eax = x
    imul eax, eax                   ; eax = x^2
    mov ebx, eax                    ; ebx = x^2
    mov eax, [y]                    ; eax = y
    imul eax, eax                   ; eax = y^2
    add eax, ebx                    ; eax = x^2 + y^2

    ; Compare with radius^2
    mov ebx, [radius]               ; ebx = radius
    imul ebx, ebx                   ; ebx = radius^2
    cmp eax, ebx
    jg print_space                  ; if x^2 + y^2 > radius^2, print space

    ; Print '* ' (inside the circle)
    mov eax, 4                      ; sys_write
    mov ebx, 1                      ; file descriptor (stdout)
    mov ecx, star                   ; pointer to the '*' character
    mov edx, 2                      ; length of the character
    int 0x80
    jmp next_x
```

```
print_space:
    ; Print ' ' (outside the circle)
    mov eax, 4                      ; sys_write
    mov ebx, 1                      ; file descriptor (stdout)
    mov ecx, space                  ; pointer to the space character
    mov edx, 2                      ; length of the character
    int 0x80

next_x:
    ; Increment x
    inc dword [x]
    jmp inner_loop                  ; continue inner loop

end_inner_loop:
    ; Print newline
    mov eax, 4                      ; sys_write
    mov ebx, 1                      ; file descriptor (stdout)
    mov ecx, newline                ; pointer to the newline character
    mov edx, 1                      ; length of the newline character
    int 0x80

    ; Increment y
    inc dword [y]
    jmp outer_loop                  ; continue outer loop

end_outer_loop:
    jmp _start                      ; go to exit
```

If the radius is (1-9), the system jumps to outer_loop and prints the circle row by row. After the circle is printed, the system loops back to the start of the program.

```
draw_triangle:
    ; Prompt for side length
    mov eax, 4                      ; sys_write
    mov ebx, 1                      ; file descriptor (stdout)
    mov ecx, prompt_tri             ; pointer to the triangle prmopt
    mov edx, 33                     ; length of the prompt message
    int 0x80

    ; Read side length input
    mov eax, 3                      ; sys_read
    mov ebx, 0                      ; file descriptor (stdin)
    mov ecx, buffer                 ; buffer to store radius
    mov edx, 3                      ; read up to 2 bytes
    int 0x80

    ; Convert input to integer
    call convert_loop               ; convert buffer to integer
    mov [tri_ori_space], al

    ; Validate line length (1-9)
    cmp al, 1
    jl invalid_length               ; if less than 1, invalid length
    cmp al, 9
    jg invalid_length               ; if greater than 9, invalid length

    ; Set initial value
    sub byte [tri_ori_space], 1     ; initial space = row - 1
    mov eax, 1                      ; initial star = 1
    mov [tri_ori_star], eax
```

If the user chooses 4, which is triangle. The system asks the user to enter the height(row) of the triangle and converts it from ASCII to its numeric value. If the height this more than 9 or less than 1, then the system jumps to invalid_length.

```
triangle_load_space:
    ; Load space number to tri_space
    mov al, [tri_ori_space]
    mov [tri_space], al

triangle_print_space:
    mov al, [tri_space]
    cmp al, 0
    jle triangle_load_star          ; jump triangle_load_star after printing all space

    mov eax, 4
    mov ebx, 1
    mov ecx, space
    mov edx, 2
    int 0x80

    sub byte [tri_space], 1         ; Decrement tri_space
    jmp triangle_print_space        ; loop again

triangle_load_star:
    ; Load star number to tri_star
    mov al,[tri_ori_star]
    mov [tri_star], al
```

```
triangle_load_star:
    ; Load star number to tri_star
    mov al,[tri_ori_star]
    mov [tri_star], al

triangle_print_star:
    mov al, [tri_star]
    cmp al, 0
    jle triangle_print_newline      ; jump triangle_print_line after printing alll star

    mov eax, 4
    mov ebx, 1
    mov ecx, star
    mov edx, 2
    int 0x80

    sub byte [tri_star], 1          ; Decrement tri_star
    jmp triangle_print_star         ; loop again

triangle_print_newline:
    mov eax, 4
    mov ebx, 1
    mov ecx, newline
    mov edx , 1
    int 0x80

    mov al, [tri_ori_space]
    cmp al, 0
    jle triangle_done               ; jump to triangle_done if all rows printed

    sub byte [tri_ori_space], 1     ; Decrement tri_ori_space
    add byte [tri_ori_star], 2      ; Add 2 for the tri_ori_star
    jmp triangle_load_space         ; loop form the begining

triangle_done:
    jmp _start                      ; go to exit
```

If the side length is (1-9), the system jumps to outer_loop and prints the triangle row by row. After the triangle is printed, the system loops back to the start of the program.

```
draw_square:
    ; Prompt for side length
    mov eax, 4
    mov ebx, 1
    mov ecx, prompt_square
    mov edx, 34                     ; Length of the updated prompt message
    int 0x80

    ; Read side length input
    mov eax, 3
    mov ebx, 0
    mov ecx, buffer
    mov edx, 3
    int 0x80

    call convert_loop               ; convert buffer to integer in eax
    mov [square_length], al         ; store the final integer line length in line_length
    mov [square_ori_length], al     ; Save original length
    mov [square_width], al

    ; Validate line length (1-9)
    cmp al, 1
    jl invalid_length               ; if less than 1, invalid length
    cmp al, 9
    jg invalid_length               ; if greater than 9, invalid length
```

If the user chooses 5, which is square. The system asks the user to enter the length of the square and converts it from ASCII to its numeric value. If the length is more than 9 or less than 1, then the system jumps to invalid_length.

```
square_loop_rows:
    mov al, [square_width]
    cmp al, 0
    je square_done                  ; If width is zero, exit

square_loop_columns:
    mov al, [square_length]
    cmp al, 0
    je square_print_newline         ; If length is zero, print newline

    ; Print a star
    mov eax, 4
    mov ebx, 1
    mov ecx, star
    mov edx, 2                      ; Length of "* "
    int 0x80

    ; Decrement length
    sub byte [square_length], 1
    jmp square_loop_columns
```

```
square_print_newline:
    ; Print a newline character
    mov eax, 4
    mov ebx, 1
    mov ecx, newline                ; Use updated name for newline
    mov edx, 1
    int 0x80

    ; Reset length to original value for the next line
    mov al, [square_ori_length]
    mov [square_length], al

    ; Decrement the row count
    sub byte [square_width], 1      ; Decrement square_width
    jmp square_loop_rows

square_done:
    jmp _start
```

If the side length is (1-9), the system jumps to outer_loop and prints the triangle row by row. After the triangle is printed, the system loops back to the start of the program.

3.2 Sample Input & Output

```
          Shape Drawing Implementation
-----------------------------------------------
1. Line
2. Rectangular
3. Circle
4. Triangle
5. Square
6. Exit
Enter your choice (1-6):
```

When the program starts

```
          Shape Drawing Implementation
-----------------------------------------------
1. Line
2. Rectangular
3. Circle
4. Triangle
5. Square
6. Exit
Enter your choice (1-6): 7
Invalid choice. Please enter a number between 1 and 5.

          Shape Drawing Implementation
-----------------------------------------------
1. Line
2. Rectangular
3. Circle
4. Triangle
5. Square
6. Exit
Enter your choice (1-6):
```

Enter number not (1-6)

```
          Shape Drawing Implementation
-----------------------------------------------
1. Line
2. Rectangular
3. Circle
4. Triangle
5. Square
6. Exit
Enter your choice (1-6): 1
Enter the length of the line (1-9):
```

Enter 1 (Line)

```
          Shape Drawing Implementation
-----------------------------------------------
1. Line
2. Rectangular
3. Circle
4. Triangle
5. Square
6. Exit
Enter your choice (1-6): 1
Enter the length of the line (1-9):6
* * * * * *
          Shape Drawing Implementation
-----------------------------------------------
1. Line
2. Rectangular
3. Circle
4. Triangle
5. Square
6. Exit
Enter your choice (1-6):
```

Line: Enter length = 6

```
          Shape Drawing Implementation
-----------------------------------------------
1. Line
2. Rectangular
3. Circle
4. Triangle
5. Square
6. Exit
Enter your choice (1-6): 2
Enter the length of rectangle (1-9):
```

Enter 2 (Rectangle)

```
          Shape Drawing Implementation
-----------------------------------------------
1. Line
2. Rectangular
3. Circle
4. Triangle
5. Square
6. Exit
Enter your choice (1-6): 2
Enter the length of rectangle (1-9):5
Enter the width of rectangle (1-9):
```

Rectangle: Enter length = 5

```
          Shape Drawing Implementation
-----------------------------------------------
1. Line
2. Rectangular
3. Circle
4. Triangle
5. Square
6. Exit
Enter your choice (1-6): 2
Enter the length of rectangle (1-9):5
Enter the width of rectangle (1-9):7
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
          Shape Drawing Implementation
-----------------------------------------------
1. Line
2. Rectangular
3. Circle
4. Triangle
5. Square
6. Exit
Enter your choice (1-6):
```

Rectangle: Enter width = 7

```
          Shape Drawing Implementation
-----------------------------------------------
1. Line
2. Rectangular
3. Circle
4. Triangle
5. Square
6. Exit
Enter your choice (1-6): 3
Enter the radius of the circle (1-9):
```

Enter 3 (Circle)

```
          Shape Drawing Implementation
-----------------------------------------------
1. Line
2. Rectangular
3. Circle
4. Triangle
5. Square
6. Exit
Enter your choice (1-6): 3
Enter the radius of the circle (1-9):4
        *
    * * * * *
  * * * * * * *
  * * * * * * *
* * * * * * * * *
  * * * * * * *
  * * * * * * *
    * * * * *
        *
          Shape Drawing Implementation
-----------------------------------------------
1. Line
2. Rectangular
3. Circle
4. Triangle
5. Square
6. Exit
Enter your choice (1-6):
```

Circle: Enter radius = 4

```
          Shape Drawing Implementation
-----------------------------------------------
1. Line
2. Rectangular
3. Circle
4. Triangle
5. Square
6. Exit
Enter your choice (1-6): 4
Enter the row of triangle (1-9):
```

Enter 4 (Triangle)

```
          Shape Drawing Implementation
-----------------------------------------------
1. Line
2. Rectangular
3. Circle
4. Triangle
5. Square
6. Exit
Enter your choice (1-6): 4
Enter the row of triangle (1-9): 4
      *
    * * *
  * * * * *
* * * * * * *
          Shape Drawing Implementation
-----------------------------------------------
1. Line
2. Rectangular
3. Circle
4. Triangle
5. Square
6. Exit
Enter your choice (1-6):
```

Triangle: Enter side length(row) = 4

```
          Shape Drawing Implementation
-----------------------------------------------
1. Line
2. Rectangular
3. Circle
4. Triangle
5. Square
6. Exit
Enter your choice (1-6): 5
Enter the length of square (1-9):
```

Enter 5 (Square)

```
          Shape Drawing Implementation
-----------------------------------------------
1. Line
2. Rectangular
3. Circle
4. Triangle
5. Square
6. Exit
Enter your choice (1-6): 5
Enter the length of square (1-9): 4
* * * *
* * * *
* * * *
* * * *
          Shape Drawing Implementation
-----------------------------------------------
1. Line
2. Rectangular
3. Circle
4. Triangle
5. Square
6. Exit
Enter your choice (1-6):
```

Square: Enter length = 4

```
          Shape Drawing Implementation
-----------------------------------------------
1. Line
2. Rectangular
3. Circle
4. Triangle
5. Square
6. Exit
Enter your choice (1-6): 6
yuecheng@LAPTOP-OJSF6NC1:~$
```

Enter 6 (Exit)

4.0 Research and Analysis

4.1 Branch Prediction and Cashing

Branch misprediction has been a well – recognized problem area in high – performance processors, which most of today's microprocessors face the methods of coping with branch related evils that Martin and Goodman (1988) expounded are the ways and means of dealing with the concept of delay in its three facets – temporal, spatial, and logical. Here, I encountered them while reading Sganzerla et al. (2019) and their mention of how they can manifest in Machine Learning and interact with algorithms trained by Data Flows. Carreras et al (2007) did this for architectures with very high instruction level parallelism and proved that branch predictor does impact on the smoothness of the flow of instruction in the pipelines Khanna et al (2010) carried out modification of Branch Target Buffer (BTB) structures to reduce the penalties caused by branches in superscalar processors. These authors' investigation verifies this and all those presented to satisfy the target mechanisms have also been suggested and all thereof agree that better performance as well as throughput rates can be achieved.

Memory management involves the integration of caching mechanisms which are designed to ease data localization processes and improve efficiency of the whole system. It has been observed that client-side caching can prove useful in enhancing the scale & rate in the client-server DB systems without incurring a loss of the transactions' semantics and dependability (Franklin, 1993). Cache memory in the CPU is assistive of escaping the gap between the speeds of the processor and main memory through retrieving frequently requested data that would otherwise be obtained from the main memory. However, while query caching involves loading items from cache memory, it is asserted that aging based LFU techniques assist in managing the cache within data warehouses (Hassan et al., 2022). Mobile ad hoc networks (MANETs) benefit from cooperative caching both in the aspect of high efficiency in accessing data and in the aspect of high availability of data by utilizing the resources that wander in the mobile devices (Kumar & Chauhan, 2009). The fact that these various systems are capable of using data caching techniques in performing their activities is an indication that data caching is an effective method of enhancing the effectiveness of any system in delivering access to data however A modern deep pipelined microprocessor cannot be imagined without the correct prediction of branch instructions because branch prediction is the most critical component which comes in instruction fetch cycle. As a result of the fact that the branch predictor is one of the paths that lie within the instrumental fetch path, a single-cycle prediction must be done. However, as the features go down, and the clock frequency goes up, the latencies will affect the size of the branch predictor than can be implemented in one cycle. Therefore, accuracy and latency are two factors in which branch prediction is directly proportional with each other. Therefore, with the introduction of deeper pipelines, there is also an improvement on the overall system performance, and it is made possible to increase the clock rate but not without some performance penalty through increased

in the branch misprediction penalties. This makes this particular aspect particularly ironical because the branch predictor is provided with less clock time in low period clock systems in which to attempt to make these predictions: thus, the system has to scale down on the branch predictor which translates to low accuracy and therefore less overall benefits that accrue from high clock speeds. And Several methodologies are also considered in which there is a chance to obtain the balance of high accuracy and low operational latencies of branch predictors. It is possible to divide our methods into two large sets or strategies which include a first order hierarchical predictor that has to rely solely on the available hardware and a second order predictor also known as a cooperative predictor that will have to rely on both the available hardware components and a compiler. New hierarchical organizations are described which expand the capability of standard schemes for prediction. This is followed by a specification of one high scalability branch predictor design that results from a new kind of neural learning. Relying on this research paper, this study is mainly aimed at the hierarchical model. (Jiménez, D. A., & Lin, C. (2002)).

On other side These days, Branch Predictors has become one of the most crucial units in each and every modern-day processor. It looks like the modern predictors can work with a reasonable level of accuracy, however, misprediction appears to be quite costly. Hence it is possible to understand why research effort is now aimed at improving the operation of branch predictors within fewer resource capacities. In the past, as ILP increased, computer architectures now often use speculation to enhance performance. The importance of accurate basic prediction techniques grows with the extension of the pipeline depths and with the number of instructions that are issued per cycle. In this study, we concentrate on some branch predictors and do performance analysis of the predictors with some benchmark applications. The present study was conducted by Ambashankar, A., Chandrasekar, G., R, C. A., & S, S in the year 2022. Today Alao Branch prediction technique had been incorporated in the superscalar and also in the deeper pipelined.

However, with this flexibility, the distributed instructions though can cause disruption of the flow of execution that already is part of the pipeline stages and therefore have a negative impact on the latter. In branching, some of the issues that this paper looks at include; Branch prediction and further; some of the issues and challenges linked with branching as well as methods possibly to improve the processor efficiency. Furthermore, this paper also dwells on the consequences regarding branch predictions in various types of processors and their attributes. The record of data which is used in parallel processors to enhance the speed of conditional branch instructions and hence augment the performance of the CPU is something that forms the subject of this paper. In addition to but not limited to branch predictor techniques, this paper also brings into focus the characteristics of branch predictor techniques, challenges/ issues emanating from branch predictor techniques, and future techniques on branch predictor. As is accessible in Nain & Chaudhary (2021).

4.2 Comparative Study of Programming Approaches

In the development of operating systems, there is ongoing discussion whether to use low-level assembly programming or high-level programming, noting that both approaches have their setbacks. In this part, we are going to compare both low-level assembly programming and high-level programming according to their performance, ease of use, security and stability and the application in OS development.

(I) Performance

When designing operating systems, there is always a debate regarding whether to employ level assembly or high-level programmer, noting that each has its drawbacks. However, assembly provides more control to the part of the hardware resources, thus, the higher-level languages are easier to understand, extent (Bryant, 2002). It is the opinion of Landau (1976) that in the area of compiling codes for overlapped instruction execution high level languages are possibly more proficient than the structure assembly codes. This claim then seems a little more reasonable, however, if many hierarchical levels of memory are introduced including operating systems or multiprogramming the cache may be drastically changed especially when employing large caches (Agarwal et al., 1988). With more enhancement in the internal architecture of the processors that were introduced like out of order execution and boosted up clock rates, the memory systems are most likely to be the biggest bottleneck because they will be keeping the CPU idle for over half of its execution time. One would expect, however, that a small fraction will increase even further in the machines of the future; because much larger caches and latency hiding techniques will prevent this. Further challenges batter multiprocessor systems, around 30-70% of OS service time on multiprocessor is high compared to uniprocessor due to coherence misses resulting from inter-processor communication (Rosenblum et al., 1995).

On the other hand, regarding the utilization of an operating system, high-level programming language and the low-level assembly language also have advantages as well as disadvantages. Bryant (2002) defined that languages at high level are better and preferable for cross-platform while, at the same time, assembly language gives more control over the hardware in use. According to Landau (1976), there is an expectation he has for high level languages to provide better execution of overlaid instructions that can be done through manual coding of assembly operands. Al the same though, the impacts of multiprogramming and operating systems can also be substantial whenever they are used in a larger cache since this utilizes the cache when used (Agarwal et al. 1988). In addition, as the out of order execution ability gains entry into the design of present-day CPU, this results in CPU's take most of their time in idle time, this state renders the memory systems as the largest impediment in any architecture since the idle time easily goes over fifty percent for most of the execution time. Surprisingly, the rate at which the stall percentage increases does not necessarily grow in the subsequent systems because of the availability of a larger amount of cache as well as other forms of latency hiding. Other issues

peculiar to multiprocessor systems includes the otherwise reasonable concern that majority of system resources are consumed by most OS functionary at an estimate ratio of 30-70% (Rosenblum et al. 1995).

The operating system has very specific and important roles in application memory management and optimization from the time when the concept is being formulated. Another typical suspect that means in Chip Multiprocessor affects an application performance is the on-chip memory organization. Also, several software-controlled optimization techniques are also very useful in improving the efficiency of the on-chip memory hierarchy as are the many hardware optimization methods; they seem to be general for wide application. This is as is due to the fact that in most cases, the software-controlled approaches have a good outlook of other workloads that are live concurrently. Incorporation of the hardware measures with sub-goals of memory control and optimization on-chip with standard OS rules and techniques offer a more suitable approach towards the resolution of the memory performance challenge. Cessor impacts an application's overall performance in the on-chip memory hierarchy. Also, several optimization techniques which are software-controlled are very useful in enhancing the efficiency of the on-chip memory hierarchy in addition to many hardware optimization methods which seem to be general one for all types of applications. This is the case because, in most situations, software-controlled approaches have a good perspective of the other workloads that are operating concurrently. Since traditional hardware solutions can control and optimizing opportunities of the on-chip memory hierarchy and the possibility of applying operating system rules and methods are based on the higher level, the better solution to the memory performance problem is to combine both of them. This study provides a comprehensive review of techniques and work of all such systems that have addressed the integration of operation system methods and policies with on chip cache optimization strategies. Other methods that could be researched in order to predict their effectiveness are suggested. Apoorva: Khatoon, H., Altaf, T., & Mirza, S. H. (2011).

(II) Ease of Life

While low level assembly programming has been favored for its simplicity, high level programming languages have been fevered for their high level of abstraction, safety and point access (Bryant 2002). Low-level control and while being often required to fulfil systems programmers' needs that are not addressed by the language, brings an either-or choice between managing the underlying hardware, or making it accessible as part of the language (Frampton, et. al., 2009). As we have postulated in this paper, an assembly language is by no way the simplest language to learn but for activities like programming device drivers they are unrivaled by Hyde, (K., 2003). Several authors viewed the above-mentioned problem of the gap between low level and high-level languages, and have advanced a solution, namely the integration of low-level knowledge into high level languages. This maintains the purity of type but enables SSP with scoped semantic regimes, architectural width datatypes, ++ and intrinsic methods (Frampton et al. 2009). Modern materially typed functional languages can also support datatypes which layouts have to be strict, as e.g. needed for device drivers or operating system. (Diatchki 2007). These developments aim at embracing both the high-level languages and ordinary low-level programming and at the same time they make the low-level languages favorable for developers.

However, to focus on the main aspect of the single-processor computer let stick by the following statement. Every program at nucleus is coded using a higher order language, that is Modula-21 while no programming is done in assembly language. Brief about nucleus is introduced and how the concept of the nucleus can be applied is demonstrated in the most basic and simple way. The nuclear content is also offered systematically and the manner in which nuclear content is presented sequentially is also explained systematically. In fact, the observations made in the contributions reveal that the use of the high-level language in the programming of kernel is much better than the assembly language. Therefore, let's intentionally concentrate on the part of computers which has only one processor. It should be noted that no similitude of assembly code, which is characteristic of numerous other equivalent computer programs and applications, can be found anywhere in the Nucleus Computer Software in question, but is to the contrary premised on the much more elevated Modula-21 computer language. In this article, the basic concepts of the nucleus are described along with an example of the ensemble basic nucleus application. There is also a methodical structure of the nuclear content. Thus, it has been explained that it is better to program kernel in some high-level language rather than assembly language.

The performance efficiency of the resulting nucleus is in the range of other assembly code nuclei. It is worth stating educational and ill-taught items or ideas as stated by Hoppe, J. (1980). It was observed that some of the courses such as physics, mathematics and computer science are naturally difficult courses. These teachers always attempt various strategies in a bid to help students learn these fields. Apparently, there is a correlation to find, and that is: the more advanced a language, the more advanced concepts and more abstractions are applied in computer programming. As can be expected, it is not beyond the realm of expectation to propose that it is with ease languages closer to natural languages that computer programming is made. Other

subjects are based on other ideas which also are quite similar to each other. This, however, seems to be in opposition to the manner in which concepts are learned by most people, which is that it is much easier to learn second level concepts once the first level concepts have been grasped. Does it hold water then to state that it is possible, in some ways, to grasp ideas that are often regarded as more complex without having a solid means by which to grasp many more basic ideas at one's disposal? The current assignment presents two basic problems which are the heart of modern education: Since low level languages like assembly and machine code are considered to be difficult. (Gallego-Durán, F. J., Villagrá-Arnedo, C. J., Satorre-Cuerda, R., Compañ-Rosique, P., & Llorens-Largo, F. (2018))

(III) Security and Stability

Branch prediction can improve the performance of the entire system, but it also has some security risks, especially execution-related vulnerabilities, such as Specter and Meltdown. The vulnerabilities exploit the CPU's ability to access and store data ahead of time while computing certain instructions, allowing side-channel attacks to access normally protected data in the system. These attacks affect the privacy and data integrity of the entire system, not just on public devices but also on others, especially servers and personal devices that handle sensitive information.

In operating system development, you need to consider how to avoid memory leaks, buffer overflows and other problems through programming methods.

- Memory leaks: Memory leaks will slowly use up system resources, affecting performance or causing the system to crash directly. Memory leaks can be effectively avoided through smart pointers and garbage collection mechanisms. In particular, high-level languages (such as Java and Python) have built-in garbage collection mechanisms, which can ensure memory safety to a certain extent.
- Buffer overflow: Buffer overflow is one of the common security issues and can be exploited to execute malicious code. Many programming languages and compilers (such as modern C++ and GCC) provide boundary checking features or avoid overflows through type-safe design. This control method can effectively enhance the system's security and stability.

Comprehensive Security Approaches for Mitigating Vulnerabilities

(i) Security Measures for Hardware Component Vulnerabilities

To mitigate the risks that arise from some hardware threats such as Spectre or Meltdown, manufacturers of the hardware components developed certain hardware security features directly embedded in the CPUs. Among them are barriers to speculative execution and more stiff isolation approaches applied to sensitive regions in the memory. The partitioning of memory access is, in turn, further complemented by the application of speculative store bypass technologies to such designs in order to make it theoretically impossible for even speculative execution to ever cause physical data leakage.

(ii) Mechanisms for Correcting or Avoiding the Negative Effects of Speculation

Speculation control register bit flags for the operating system may be used to include a Speculative Store Bypass Disable (SSBD) flag which signals to the CPU that certain operations should not be done via speculative execution (*Measuring the Impact of Spectre and Meltdown*, 2018). Because OS-level measures allow applications to disable speculative execution in high-security environments such as the access to the cryptographic keys, the attack geometry for side-channel vulnerabilities is narrowed significantly.

(iii) Assumptive Anomaly Detection

It is also possible to deploy algorithms capable of automating the process of detecting robust signs of speculative execution monitoring and control that can signal potential threats from the activity. Such algorithms with the use of monitoring the performance parameters of CPU and memory access during the run of applications and their changes can guard against application abuses as well as warn the OS of possible intrusions and automatically initiate necessary protective interventions to restrict the relevant speculation of the affected applications.

(iv) Enhancements in the Security at the Level of the Operating System

OS developers can employ Data Execution Prevention (DEP) and Address Space Layout Randomization (ASLR) in a bid to reinforce their systems against possible attacks invasions such as the buffer overflow or code injection. Thus, DEP forbids execution in non-executable sections of memory, while ASLR, which is more common, also randomizes bits in order to make it more difficult for the intruder to target memory components or even functions.

(v) Increased Strength of Isolation through Sandboxing

Snack boxing is the process of confinement of applications and their resources, so that a smaller scope of inter-application attack surfaces is available. In a sandbox model, access to system memory is strictly controlled, thus averting the probability of any unauthorized application trying to gain access and controlling memory usage from one process to another process. Sandboxing is essential for applications where there is a high presence of untrusted data such as web applications or other publicly accessible applications.

(vi) Adoption of Rust and other Memory-Safe Languages

Programming in memory-safe languages, such as Rust, can have the effect of increasing the stability and the security of the program, as it will minimize the risks of manual memory management. Rust's ownership model means that certain memory must be adhered to – this results in predictable behaviors and prevents NULL dereference or use after free problems. Developing essential elements of OS in languages safe for memory addresses less the questions of memory management without the loss of performance.

(IV) Application in OS Development

Operating system development consists of various very basic parts: Kernal programming, System Calls and Driver Development.

Kernel programming

Kernel programming is responsible for controlling critical system parts such as memory and device control and the management of processes.

Some low-level languages such as C and C++ are often used for kernel programming because efficiency and speed are very important for programming. Although these languages have excellent management of hardware, memory management still requires careful checks to avoid problems that should not occur. Such as the Linux kernel, it is built in C and thus provides fine-grained control over the interactions between hardware.

System Calls

System calls provide a way for user-space programs to communicate with the kernel.

The system call interface should be explicit and uninterruptable to allow user space applications unique access to system resources. For example, the Windows system call interface is designed for compatibility and security.

Driver Development

Drivers are developed to better enable people to manage hardware devices so that the operating system can correctly identify and use the hardware. Drivers must interact directly with hardware elements and usually in real-time. The use of C is usually implemented because of efficiency in low-level programming. Windows implements the Windows Driver Model, WDM, a standardized interface that augments the reliability of managing devices. (*A Coordinated Approach for Practical OS-Level Cache Management in Multi-core Real-Time Systems*, 2013)

Use of Programming Techniques in OS Development

(i) Memory Safety with Kernel Development Tools

Kernel programming can involve a lot of memory operations, but there is a movement to kernel programming in languages that are more memory safe such as Rust, to prevent problems such as null pointer dereference and buffer overflow. For example, there is no problem if Rust has been gradually incorporated into the Linux kernel where it is expected to provide advantages in memory safety assurances as the risks associated with memory management are mitigated but performance does not suffer.

(ii) Real Time Scheduling and Critical Task Separation

Language such as C or Assembly as used by OS developers enable real-time scheduling to be executed as well as critical tasks to be separated. For example, low level optimizations are added in Linux Real Time (RT) patches to guarantee predictable task control that is very important for all embedded and mission critical applications.

Additional Aspects in Operating System Development

(i) Concurrency Control

This is a crucial facet in OS development and much more so in multi-threaded applications. All low-level concurrency control techniques including spinlocks, mutexes and semaphores, in C and assembly, and any concurrency mechanisms in the kernel are implemented with these methods. Higher-level operating systems such as FreeBSD provide thorough support of multi cores processors through using more advanced techniques like read-write locks.

(ii) Portability and Cross-Platform Compatibility

As is the case with other modern operating systems, such diversity must be achieved as a target. It is by encouraging cross-platform languages, porting libraries with POSIX compliant APIs in Unix based systems that allow a standard environment for making system calls or performing kernel functions on different devices. This needs more attention especially for embedded and mobile devices which necessitates the operating system to be, and multiple architectures, in an effective manner.

(iii) Security-Sensitive OS Modules Written in a Particular Language

Due to the context of the modules' usage, languages with advanced safety features such as Rust or Go are rather common. For example, the Windows OS now has Rust foundations integrated into it which are used in preventing memory safety problems. Furthermore, these languages also incorporate capability-based security models that enhance access control.

4.3 Future Enhancements and Proposals

In order to improve system efficiency, both branch prediction and caching are highly required to enhance with two different levels such as the hardware and software. For now, here are the proposed improvements listed.

Future Improvements in Branch Prediction and Caching Techniques

(i) Hardware level improvements

Improved branch prediction algorithm: Currently, state-of-the-art branch prediction relies on shallower historical data. By expanding the depth of tracking branch history or applying machine learning algorithms, the accuracy of prediction can be greatly improved, and performance losses brought about by incorrect predictions will be reduced. In the future, CPUs may use deep neural network models to analyze more complex branch patterns to improve prediction accuracy.

Cache hierarchy optimization: In the future, we can consider more different types of third-level caches, and even design different cache structures for specific jobs so that they can access data faster. More flexible data prefetching algorithms that consider data prefetching with more different information can improve cache hit rates and lower latency.

(ii) Software improvement level

Optimization by the compiler: The compilers can do smart rearrangements of code based on characteristics of the program to reduce the number of branch instructions or to optimize branches for the branch prediction. As an example, the compiler may insert comments within the code to force the processor to make predictions for selected chunks of code.

Cache management tools: Realize smarter cache management utilities that will enable the operating system to observe and adapt the cache policies at runtime-such as enabling finer-grained data cleaning policies-increasingly efficient use of caches.

Benefits of Future Enhancement: Low-Level and High-Level Programming Approaches

(i) Performances

Low-level programming methods are able to enhance the operating system in directly manipulating hardware resources, which is very important for core tasks such as process management or memory allocation. High-level programming languages are at an advantage in terms of security and development efficiency. By doing so, it will help the development of future operating systems find a better balance between performance and development efficiency. For example, Rust has been adopted in OS to reduce common errors in memory management, thus enlarging system security.

(ii) Resource management flexibility

Details controlled by low-level resource management are more appropriately done with relatively low-level languages, whereas high-level languages are suitable for the writing of high-level OS services and application program interfaces to support readability and scalability of code. A design like microkernel can be utilized in designing future OSs to modularize the basic functions of the system while using high-level languages to implement high-level user interfaces and services yet retaining the efficiency and stability of the underlying system. (Mittal, 2018)

Future Improvements: Other Approaches

A more specific context for a program is not always included in the current branch predictors, which may lead to an incorrect guess on complex programs that need context. Context-aware prediction might enable branch prediction to have the ability to figure out what the context is of the application (e.g., type of application, workload characteristics) and customize the prediction. This means a CPU could use different prediction patterns for different tasks, for example, predicting image processing performance and web page browsing performance.

(i) Consider / Explore the Use of Heterogeneous Cache Structures

A heterogeneous cache structure could be uneven so that different types of applications can be catered for. For instance, some L3 cache blocks could be meant to be used for high throughput applications while others are meant for faster access. Thus, improving the efficiency of the E/cache across the different tasks.

(ii) Machine Learning That Interfaced with Hardware for Automating the Changes

Hardware can be embedded to learn instructions through a unique learning algorithm that gets better with time. For instance, a machine learning unit can be utilized to control cache and branch prediction by actively interfacing with the processor and utilizing its data. It would therefore mean that the ML-enabled chip will adjust prediction rate depending on program behavior patterns.

(iii) Compiler-Assisted Branch Optimization Hints

Apart from mere code rearrangements, optimized branch hints based on testing phases profiling could also be created by the compiler. This information could assist the processors with branch prediction in a more efficient manner. For instance, it would also be possible to inform the compiler to attach some likelihoods for certain branches, which the CPUs would then use to determine which branches to pursue in the actual application usage model.

(iv) Dynamic Cache Partitioning and Access Control

There could also be a dynamic control of the cache where the OS could provide a partition on the cache resources on the basis of the access pattern or the priority of the processes. This would mean that at a given time, several tasks could be executed simultaneously and a dedicated area within the cache could be reserved for the tasks of high priority, or the cache retention times of selected

data could be modified. Such an approach would lead to improved cache management especially in multitasking settings where various programs are executed at the same time, and all compete for memory bandwidth.

5.0 Conclusion

To summarize, this assignment has provided useful insights into both the practical and theoretical aspects of system optimization. The section on assembly programming effectively puts into reality a form generation system that uses memory-efficient techniques to create simple shapes, communicates directly with hardware, and has strong error-handling features. While the study in Section B demonstrated how important branch prediction and caching are for improving CPU performance and memory management, especially in settings with limited resources. A more thorough grasp of their use in operating system development was provided by the comparison of low-level assembly programming and high-level programming techniques, which brought to light the trade-offs in security, usability, and performance. The results imply that creating effective, safe, and responsive operating systems requires both high-level programming methods and low-level optimizations. Future developments in OS systems and performance management will be greatly influenced by improvements in the hardware-level integration of branch prediction and caching as well as a well-rounded programming style.

6.0 References

Das, M., Sardar, B., & Banerjee, A. (2015). Attacks on Branch Predictors: An Empirical Exploration. In *Lecture notes in computer science* (pp. 511–520). https://doi.org/10.1007/978-3-319-26961-0_30

Falcón, A., Santana, O. J., Ramírez, A., & Valero, M. (n.d.). *International Journal of High Performance Computing and Networking (ijhpcn) a Latency-conscious Smt Branch Prediction Architecture*. https://www.semanticscholar.org/paper/International-Journal-of-High-Performance-Computing-Falc%C3%B3n-Santana/b03e3055f2afff01adfb4748425c7fea3d430cd9

Implementation of branch delay in Superscalar processors by reducing branch penalties. (2010). *IEEE Conference Publication | IEEE Xplore*. https://ieeexplore.ieee.org/document/5423045?denied=

Franklin, M. (1993). *Caching and Memory management in Client-Server database systems*. https://www.semanticscholar.org/paper/Caching-and-Memory-Management-in-Client-Server-Franklin/77040ed665cbe72b7bfc831373c8890752163bf5

Abayomi, A. O., Olukayode, A. A., & Olakunle, G. O. (2020). An overview of cache memory in memory management. *Automation Control and Intelligent Systems*, *8*(3), 24. https://doi.org/10.11648/j.acis.20200803.11

Hassan, C. a. U., Hammad, M., Uddin, M., Iqbal, J., Sahi, J., Hussain, S., & Ullah, S. S. (2022). Optimizing the performance of data warehouse by query cache mechanism. *IEEE Access*, *10*, 13472–13480. https://doi.org/10.1109/access.2022.3148131

Kumar, P. S., & Chauhan, N. (2009). *A review of Cooperative cache Management Techniques in MANETs*. https://www.semanticscholar.org/paper/A-Review-of-Cooperative-Cache-Management-Techniques-Kumar-Chauhan/f820a5703a69407838bbfe6c23a2096fc526bd60

Jiménez, D. A., & Lin, C. (2002). *Delay-sensitive branch predictors for future technologies*. https://www.semanticscholar.org/paper/Delay-sensitive-branch-predictors-for-future-Jim%C3%A9nez-Lin/be770b561effc412f3dbe1915374bccaa5eb8fce

Ambashankar, A., Chandrasekar, G., R, C. A., & S, S. (2022). Exploration of Performance of Dynamic Branch Predictors used in Mitigating Cost of Branching. *2022 Third*

International Conference on Intelligent Computing Instrumentation and Control Technologies (ICICICT), 574–579. https://doi.org/10.1109/icicict54557.2022.9917915

Nain, S., & Chaudhary, P. (2021). Towards the improvement of branch instructions identification in high- performance processors: Issues, challenges and techniques. *Recent Advances in Computer Science and Communications*, *15*(7). https://doi.org/10.2174/2666255814666210210164146

Bryant, R. (2002). *CS:APP Chapter 3: Machine-Level Representations of Programs*. https://www.semanticscholar.org/paper/CS%3AAPP-Chapter-3%3A-Machine-Level-Representations-of-Bryant/11abd8a8f0c017279b7c9ef8c110468b88f022a7

Landau, C. (1976). On high-level languages for system programming. *ACM SIGPLAN Notices*, *11*(1), 30–31. https://doi.org/10.1145/987324.987328

Agarwal, A., Hennessy, J., & Horowitz, M. (1988). Cache performance of operating system and multiprogramming workloads. *ACM Transactions on Computer Systems*, *6*(4), 393–431. https://doi.org/10.1145/48012.48037

Agarwal, A., Hennessy, J., & Horowitz, M. (1988b). Cache performance of operating system and multiprogramming workloads. *ACM Transactions on Computer Systems*, *6*(4), 393–431. https://doi.org/10.1145/48012.48037

Rosenblum, M., Bugnion, E., Herrod, S. A., Witchel, E., & Gupta, A. (1995). The impact of architectural trends on operating system performance. *The Impact of Architectural Trends on Operating System Performance*. https://doi.org/10.1145/224056.224078

Bryant, R. (2002b). *CS:APP Chapter 3: Machine-Level Representations of Programs*. https://www.semanticscholar.org/paper/CS%3AAPP-Chapter-3%3A-Machine-Level-Representations-of-Bryant/11abd8a8f0c017279b7c9ef8c110468b88f022a7

Frampton, D., Blackburn, S. M., Cheng, P., Garner, R. J., Grove, D., Moss, J. E. B., & Salishev, S. I. (2009). Demystifying magic. *Demystifying Magic*, *7*, 81–90. https://doi.org/10.1145/1508293.1508305

The art of assembly language. (2004). *Choice Reviews Online*, *41*(08), 41–4714. https://doi.org/10.5860/choice.41-4714

Landau, C. (1976b). On high-level languages for system programming. *ACM SIGPLAN Notices*, *11*(1), 30–31. https://doi.org/10.1145/987324.987328

Bryant, R. (2002c). *CS:APP Chapter 3: Machine-Level Representations of Programs*. https://www.semanticscholar.org/paper/CS%3AAPP-Chapter-3%3A-Machine-Level-Representations-of-Bryant/11abd8a8f0c017279b7c9ef8c110468b88f022a7?utm_source=direct_link

Agarwal, A., Hennessy, J., & Horowitz, M. (1988c). Cache performance of operating system and multiprogramming workloads. *ACM Transactions on Computer Systems*, *6*(4), 393–431. https://doi.org/10.1145/48012.48037

Rosenblum, M., Bugnion, E., Herrod, S. A., Witchel, E., & Gupta, A. (1995b). The impact of architectural trends on operating system performance. *The Impact of Architectural Trends on Operating System Performance*. https://doi.org/10.1145/224056.224078

Frampton, D., Blackburn, S. M., Cheng, P., Garner, R. J., Grove, D., Moss, J. E. B., & Salishev, S. I. (2009b). Demystifying magic. *Demystifying Magic*, *7*, 81–90. https://doi.org/10.1145/1508293.1508305

Diatchki, I. S. (2007). *High-Level abstractions for Low-Level programming*. https://www.semanticscholar.org/paper/High-Level-Abstractions-for-Low-Level-Programming-Diatchki/a373ff043c46af11041ada2aa399209afcf981e9

Khatoon, H., Mirza, S. H., & Altaf, T. (2011). Operating System-Aware Cache Optimization Techniques for Multi Core Processors. *Operating System-Aware Cache Optimization Techniques for Multi Core Processors*, *35*, 99–105. https://doi.org/10.1109/fit.2011.26

Hoppe, J. (1980). A simple nucleus written in modula-2: A case study. *Software Practice and Experience*, *10*(9), 697–706. https://doi.org/10.1002/spe.4380100903

Gallego-Durán, F. J., Villagrá-Arnedo, C. J., Satorre-Cuerda, R., Compañ-Rosique, P., & Llorens-Largo, F. (2018). Effects of Low-Level Development on Learning to Program. In *Lecture notes in computer science* (pp. 431–445). https://doi.org/10.1007/978-3-319-91152-6_33

Measuring the impact of spectre and meltdown. (2018, September 1). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/8547554

A coordinated approach for practical OS-Level cache management in multi-core Real-Time systems. (2013b, July 1). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/6602090

Mittal, S. (2018). A survey of techniques for dynamic branch prediction. *Concurrency and Computation Practice and Experience*, *31*(1). https://doi.org/10.1002/cpe.4666

7.0 Workload Matrix

Student Name	Lee Yi Chern	Voon Yue Cheng	Bardia Momeni	Hwang XiaoShun
Student TP	(TP081340)	(TP080509)	(TP077883)	(TP077723)
Table of Contents	/			
Introduction	/			
Assembly Program and Analysis	/	/		
Research and Analysis			/	/
Conclusion	/			
References	/	/	/	/
Workload Matrix	/			
Contribution percentage (%)	25%	25%	25%	25%
Signature