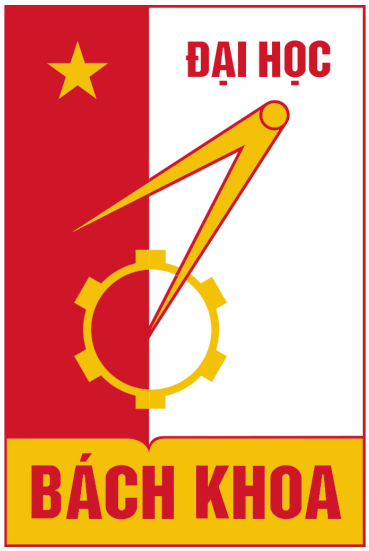
**Hanoi University of Science and Technology**
School of Information and Communication Technology

---




# IT3280E - Assembly Language and Computer Architecture Lab

Final Project: Reading BMP Files and Displaying on Bitmap Screen

Student Name: Nguyen Duc Anh
Student ID: 20235890
Class: ICT-02 K68
Group: 3

## 1 Source Code:

```
.eqv BITMAP_DISPLAY, 0x10010000   # Memmory address of Bitmap Display

.data
    # Message for user input
    input_prompt: .asciz "Enter location and .bmp image to display (Ex: D:/image/image.bmp): "
    # Message for resolution error handling
    error_scale: .asciz "Invalid image scale (512x512)!\n"
    # Message for image format error handling
    error_format: .asciz "Invalid image format (.bmp)!\n"
    # Storing user input for file path
    input_buffer: .space 256
    # Memmory for debugging or fixing the code (Addition)
    debug_mem: .space 1100000
    # Memmory to store the file content
    main_mem: .space 1100000
    # Message for open file error handling
    error_open: .asciz "Cannot find the file!"

.text
# Main Program
.global main
main:

    # Get file input
    get_input:
        li a7, 4            # Service number to print string
        la a0, input_prompt # Address of input_prompt
        ecall               # System Call

        li a7, 8            # Service number for input string
        la a0, input_buffer # Address hold the file path
        li a1, 256          # Length of path input
        ecall               # System Call

    # Replace "\n" with "\0"
        la t0, input_buffer         # Load the starting address of the input_buffer
    remove_newline:
        lb t1, (t0)                 # Load a byte from the buffer
        beq t1, zero, open_file     # Branch If the byte is '\0' -> jump to open_file
        li t2, 10                   # Load ASCII value of newline ('\n')
        beq t1, t2, replace_newline # Branch If the byte is '\n" -> jump to replace_newline
        addi t0, t0, 1              # Move to the next byte
        j remove_newline            # Repeat the process

    # Replace the character
    replace_newline:
        sb zero, (t0)               # Replace '\n' with '\0'

    # Open the file
    open_file:
        li a7, 1024             # Service number to open the file
        la a0, input_buffer     # Load address of the file path
        li a1, 0                # Open file in Read-only mode
        ecall                   # System Call
        addi t0, a0, 0          # Store the file descriptor in t0
        blt t0, zero, file_error # Branch If file descriptor is negative -> jump to file_error

    # Read the file header
    read_file:
        li   t1, 54             # BMP file header size is 54 bytes
```

```
        la   a1, main_mem       # Load address of main_mem to store the file header
        addi   a0, t0, 0        # Load file descriptor
        addi   a2, t1, 0        # Specify the length of bytes to read
        li   a7, 63             # Load syscall number for "read file"
        ecall                   # System Call
        blt  a0, t1, file_error # Branch If read fails -> jump to file_error handling

    # Check file format
    check_file_format:
        la   t2, main_mem         # Load address of the file header in memory
        lbu  t3, 0(t2)            # Load the first byte of the header
        lbu  t4, 1(t2)            # Load the second byte of the header
        li   t5, 'B'              # ASCII value for 'B'
        bne  t3, t5, format_error # Branch If first byte is not 'B' -> jump to format_error
        li   t5, 'M'              # ASCII value for 'M'
        bne  t4, t5, format_error # Branch If second byte is not 'M' -> jump to format_error

    # Check image resolution
    check_image_res:
        addi t3, t2, 18     # Move to the address of the width field in the header
        lw   t4, 0(t3)      # Load the width of the image
        addi t3, t2, 22     # Move to the address of the height field in the header
        lw   t5, 0(t3)      # Load the height of the image

        li t6, 512                  # Maximum scale for the image allowed is 512x512
        bgt t4, t6, resolution_error # Branch If width is invalid -> jump to resolution_error
        bgt t5, t6, resolution_error # Branch If height is invalid -> jump to resolution_error

    # Get begin position of pixel data
    get_position:
        addi t3, t2, 10   # Move to the address of the pixel data offset in the header
        lw   t6, 0(t3)    # Load the starting position of pixel data

    # Move the pointer to the position of begin pixel data
    file_seek:
        addi   a0, t0, 0   # Load file descriptor
        addi   a1, t6, 0   # Load the pixel data offset
        li   a2, 0         # Specify SEEK_SET (absolute positioning)
        li   a7, 62        # Service number for seek
        ecall              # System Call
        blt  a0, zero, end # Branch If seek fails -> jump to end

    # Calculate pixel data for display on Bitmap Display
    pixel_calculate:
        li    s10, 3       # Each pixel is 3 bytes (RGB format)
        mul   s7, t4, s10  # Calculate RowSize (Not include padding)
        mul   s8, s7, t5   # Calculate PixelData = RowSize x Height

    # Read the pixel data
    read_pixel:
        la   a1, main_mem   # Load address of main_mem to store pixel data
        addi   a0, t0, 0    # Load file descriptor
        addi   a2, s8, 0    # Specify the total size of pixel data
        li   a7, 63         # Service number for read file
        ecall               # System Call
        blt  a0, s8, end    # Branch If read fails -> jump to end

    # Initial Bitmap Display for output
    init:
        li a3, BITMAP_DISPLAY  # Load the Bitmap Display base address
        addi s1, t5, 0         # Height of the image
        addi s2, t4, 0         # Width of the image

```

```
#-------------------------------------------------------------------------------------------------
# Algorithm for displaying:                                                                      |
#                                                                                                |
# s2 = width of the image.                                                                       |
# s1 = height of the image.                                                                      |
#                                                                                                |
# Loop from bottom up for each rows.                                                             |
# This purpose is to access first row of bitmap image.                                          |
#                                                                                                |
# For each rows, loop from left to right to access each columns refers for each pixels.          |
# For each columns (aka pixels) in the row, process the color and display immediately to the     |
# display.                                                                                       |
#                                                                                                |
# The default color format of bitmap image it's BGR.                                            |
# Implementation of pixel in a row of bitmap image: [B1 G1 R1] [B2 G2 R2] [B3 G3 R3] ...        |
# To display image on Bitmap Display, we need to convert BGR color format into RGB color format|
# It means convert 0x00BBGGRR -> 0x00RRGGBB.                                                     |
# Each value on the hexadecimal value it's 4 bit.                                               |
#                                                                                                |
# To do that, we use shift left logic and bitwise operation to evaluate the exactly value of    |
# the color refers to RGB format to display the image with right color as accurately as          |
# possible.                                                                                      |
#                                                                                                |
#-------------------------------------------------------------------------------------------------

    # Algorithm implement
    display:
        loop_rows:
            addi s1, s1, -1 # Decrease the row counter
            blt  s1, zero, end # Branch If all rows of the image processed -> jump to end
            mul s9, s1, s7 # Calculate the offset to the current row's pixel data
            la  t3, main_mem # Get the address of the image header to access each pixel
            add t3, t3, s9 # Get the begin address of the pixel data of the current row
            addi s4, s2, 0 # Reset the width of the row after access each columns (aka pixels)
            addi s5, t3, 0 # Init pointer to process each pixels in the row

        loop_cols:
            beq s4, zero, next_row # Branch if all pixels in a row processed -> jump to next row
            lbu t1, 0(s5)      # Get the B value 0xBB
            lbu t2, 1(s5)      # Get the G value 0xGG
            lbu s11, 2(s5)     # Get the R value 0xRR
            slli s11, s11, 16 # Shift left logical s10 = 0x000000RR -> 0x00RR0000
            slli t2, t2, 8     # Shift left logical t2 = 0x000000GG -> 0x0000GG00
            or   s11, s11, t2 # Logical OR: s10 or t2 -> 0x00RRGG00
            or   s11, s11, t1 # Logical OR: s10 or t1 -> 0x00RRGGBB
            sw  s11, 0(a3) # Store the value into Bitmap Display to display
            addi a3, a3, 4 # Move to the next pixel on Bitmap Display (each 4 bytes)
            addi s5, s5, 3 # Move to the next pixel on image after process 3 byte RGB format
            addi s4, s4, -1 # Decrease the pixel counter of the image
            j loop_cols      # Jump to next pixel

        next_row:
            j loop_rows      # Jump to next row

end:
    # Close the file
    close_file:
        addi a0, t0, 0  # Load the file descriptor
        li a7, 57       # Service number for clode the file
        ecall           # System Call

    # Exit the Program
    exit:
```

```
        li a0, 0         # Load exit code (0)
        li a7, 10        # Service number for exit program
        ecall            # System Call

# Error Handling
# Open file error handle
file_error:
    li a7, 4           # Service number for print string
    la a0, error_open  # Load address of error_open Message
    ecall              # System Call
    j end              # Jump to end

# Resolution of image error handle
resolution_error:
    li a7, 4            # Service number for print string
    la a0, error_scale  # Load address of error_scale Message
    ecall               # System Call
    j end               # Jump to end

# Format file error handle
format_error:
    li a7, 4            # Service number for print string
    la a0, error_format # Load address of error_format Message
    ecall               # System Call
    j end               # Jump to end

```

## 2 Explanation:

### 2.1 Declare Variables

#### 2.1.1 Memmory address of Bitmap Display

- `.eqv BITMAP_DISPLAY, 0x10010000`: Defines the memory-mapped address of the bitmap display.

#### 2.1.2 Input Variables

- `input_prompt: .asciz "Enter location and .bmp image to display (Ex: D:/image/image.bmp): "`: String to prompt the user to input a BMP file path.
- `input_buffer: .space 256`: Allocates 256 bytes for storing user input.

#### 2.1.3 Error Message

```
    error_scale: .asciz "Invalid image scale (512x512)!\n"
    error_format: .asciz "Invalid image format (.bmp)!\n"
    error_open: .asciz "Cannot find the file!"
```

- Strings for error messages to handle different error conditions (resolution, format, file opening).

#### 2.1.4 Memmory Allocates

- `main_mem: .space 1100000`: Allocates a large memmory region to storing the BMP file content.
- `debug_mem: .space 1100000`: Allocates a large memmory region to debugging. In addition, the purpose of this is to prevent overwriting other data, align memmory and simulate reserve space.

### 2.2 Main Program

```
.global main
main:
```

- Defines the main entry point for the program.

#### 2.2.1 Input Handling

```
# Get file input
get_input:
    li a7, 4            # Service number to print string
    la a0, input_prompt # Address of input_prompt
    ecall               # System Call
```

- Displays the input prompt using a syscall for printing a string.

```
li a7, 8            # Service number for input string
la a0, input_buffer # Address hold the file path
li a1, 256          # Length of path input
ecall               # System Call
```

- Accepts the user's file path input and stores it in `input_buffer`.

#### 2.2.2 Replace Newline with Null Terminator

```
# Replace "\n" with "\0"
    la t0, input_buffer         # Load the starting address of the input_buffer
remove_newline:
    lb t1, (t0)                 # Load a byte from the buffer
    beq t1, zero, open_file     # Branch If the byte is '\0' -> jump to open_file
    li t2, 10                   # Load ASCII value of newline ('\n')
    beq t1, t2, replace_newline # Branch If the byte is '\n" -> jump to replace_newline
    addi t0, t0, 1              # Move to the next byte
    j remove_newline            # Repeat the process

# Replace the character
replace_newline:
    sb zero, (t0)               # Replace '\n' with '\0'
```

- Iterates through `input_buffer` to find and replace the newline character (`\n`, ASCII 10) with a null terminator `\0`.

#### 2.2.3 Open File

```
# Open the file
open_file:
    li a7, 1024             # Service number to open the file
    la a0, input_buffer     # Load address of the file path
    li a1, 0                # Open file in Read-only mode
    ecall                   # System Call
    addi t0, a0, 0          # Store the file descriptor in t0
    blt t0, zero, file_error # Branch If file descriptor is negative -> jump to file_error
```

- Opens the BMP file in read-only mode. If the file descriptor is negative, it jumps to `file_error`.

### 2.2.4 Read File Header

```
    # Read the file header
    read_file:
        li   t1, 54             # BMP file header size is 54 bytes
        la   a1, main_mem       # Load address of main_mem to store the file header
        addi   a0, t0, 0        # Load file descriptor
        addi   a2, t1, 0        # Specify the length of bytes to read
        li   a7, 63             # Load syscall number for "read file"
        ecall                   # System Call
        blt  a0, t1, file_error # Branch If read fails -> jump to file_error handling
```

- Reads the first 54 bytes (header) of the BMP file into `main_mem`.

### 2.2.5 Validate File Format

```
    # Check file format
    check_file_format:
        la   t2, main_mem         # Load address of the file header in memory
        lbu  t3, 0(t2)            # Load the first byte of the header
        lbu  t4, 1(t2)            # Load the second byte of the header
        li   t5, 'B'              # ASCII value for 'B'
        bne  t3, t5, format_error # Branch If first byte is not 'B' -> jump to format_error
        li   t5, 'M'              # ASCII value for 'M'
        bne  t4, t5, format_error # Branch If second byte is not 'M' -> jump to format_error
```

- Checks the first two bytes of the BMP header to ensure they are 'B' and 'M'.

### 2.2.6 Validate Image Resolution

```
    # Check image resolution
    check_image_res:
        addi t3, t2, 18     # Move to the address of the width field in the header
        lw   t4, 0(t3)      # Load the width of the image
        addi t3, t2, 22     # Move to the address of the height field in the header
        lw   t5, 0(t3)      # Load the height of the image

        li t6, 512                  # Maximum scale for the image allowed is 512x512
        bgt t4, t6, resolution_error # Branch If width is invalid -> jump to resolution_error
        bgt t5, t6, resolution_error # Branch If height is invalid -> jump to resolution_error
```

- Validates the width and height of the image (both must be 512).

### 2.2.7 Read Pixel Data

```
    # Get begin position of pixel data
    get_position:
        addi t3, t2, 10   # Move to the address of the pixel data offset in the header
        lw   t6, 0(t3)    # Load the starting position of pixel data
```

- Extracts the offset to the pixel data from the header.

```
    # Move the pointer to the position of begin pixel data
    file_seek:
        addi   a0, t0, 0   # Load file descriptor
        addi   a1, t6, 0   # Load the pixel data offset
        li   a2, 0         # Specify SEEK_SET (absolute positioning)
```

```
        li   a7, 62        # Service number for seek
        ecall              # System Call
        blt  a0, zero, end # Branch If seek fails -> jump to end
```

- Moves the file pointer to the pixel data.

```
    # Calculate pixel data for display on Bitmap Display
    pixel_calculate:
        li    s10, 3       # Each pixel is 3 bytes (RGB format)
        mul   s7, t4, s10  # Calculate RowSize (Not include padding)
        mul   s8, s7, t5   # Calculate PixelData = RowSize x Height
```

- Calculates the total size of the pixel data.

```
    # Read the pixel data
    read_pixel:
        la   a1, main_mem   # Load address of main_mem to store pixel data
        addi   a0, t0, 0    # Load file descriptor
        addi   a2, s8, 0    # Specify the total size of pixel data
        li   a7, 63         # Service number for read file
        ecall               # System Call
        blt  a0, s8, end    # Branch If read fails -> jump to end
```

- Reads the pixel data into `main_mem`.

### 2.2.8 Display Pixel Data

```
    # Initial Bitmap Display for output
    init:
        li a3, BITMAP_DISPLAY  # Load the Bitmap Display base address
        addi s1, t5, 0         # Height of the image
        addi s2, t4, 0         # Width of the image
```

- Initializes the base address for the bitmap display and stores image dimensions.

```
#-------------------------------------------------------------------------------------------
# Algorithm for displaying:                                                                |
#                                                                                          |
# s2 = width of the image.                                                                 |
# s1 = height of the image.                                                                |
#                                                                                          |
# Loop from bottom up for each rows.                                                       |
# This purpose is to access first row of bitmap image.                                     |
#                                                                                          |
# For each rows, loop from left to right to access each columns refers for each pixels.    |
# For each columns (aka pixels) in the row, process the color and display immediately to the   |
# display.                                                                                 |
#                                                                                          |
# The default color format of bitmap image it's BGR.                                       |
# Implementation of pixel in a row of bitmap image: [B1 G1 R1] [B2 G2 R2] [B3 G3 R3] ...   |
# To display image on Bitmap Display, we need to convert BGR color format into RGB color format|
# It means convert 0x00BBGGRR -> 0x00RRGGBB.                                               |
# Each value on the hexadecimal value it's 4 bit.                                          |
#                                                                                          |
# To do that, we use shift left logic and bitwise operation to evaluate the exactly value of   |
# the color refers to RGB format to display the image with right color as accurately as    |
# possible.                                                                                |
```

```
#                                                                                                    |
#----------------------------------------------------------------------------------------------------
```

```
    display:
        loop_rows:
            addi s1, s1, -1 # Decrease the row counter
            blt  s1, zero, end # Branch If all rows of the image processed -> jump to end
            mul s9, s1, s7 # Calculate the offset to the current row's pixel data
            la  t3, main_mem # Get the address of the image header to access each pixel
            add t3, t3, s9 # Get the begin address of the pixel data of the current row
            addi s4, s2, 0 # Reset the width of the row after access each columns (aka pixels)
            addi s5, t3, 0 # Init pointer to process each pixels in the row
```

- Iterates through the rows of the image, starting from the bottom.

```
    loop_cols:
            beq s4, zero, next_row # Branch if all pixels in a row processed -> jump to next row
            lbu t1, 0(s5)      # Get the B value 0xBB
            lbu t2, 1(s5)      # Get the G value 0xGG
            lbu s11, 2(s5)     # Get the R value 0xRR
            slli s11, s11, 16 # Shift left logical s10 = 0x000000RR -> 0x00RR0000
            slli t2, t2, 8     # Shift left logical t2 = 0x000000GG -> 0x0000GG00
            or   s11, s11, t2 # Logical OR: s10 or t2 -> 0x00RRGG00
            or   s11, s11, t1 # Logical OR: s10 or t1 -> 0x00RRGGBB
            sw  s11, 0(a3) # Store the value into Bitmap Display to display
            addi a3, a3, 4 # Move to the next pixel on Bitmap Display (each 4 bytes)
            addi s5, s5, 3 # Move to the next pixel on image after process 3 byte RGB format
            addi s4, s4, -1 # Decrease the pixel counter of the image
            j loop_cols      # Jump to next pixel
```

- Converts the pixel format from BGR to RGB and writes it to the display.

```
    next_row:
            j loop_rows      # Jump to next row
```

- Moves to the next row of the image.

### 2.2.9 End Program

```
end:
    # Close the file
    close_file:
        addi a0, t0, 0  # Load the file descriptor
        li a7, 57       # Service number for clode the file
        ecall           # System Call
```

- Execute close the file after processed.

```
    # Exit the Program
    exit:
        li a0, 0        # Load exit code (0)
        li a7, 10       # Service number for exit program
        ecall           # System Call
```

- Exit the program after finished all.

### 2.3 Error Handling

```
# Error Handling
# Open file error handle
file_error:
    li a7, 4            # Service number for print string
    la a0, error_open   # Load address of error_open Message
    ecall               # System Call
    j end               # Jump to end

# Resolution of image error handle
resolution_error:
    li a7, 4            # Service number for print string
    la a0, error_scale  # Load address of error_scale Message
    ecall               # System Call
    j end               # Jump to end

# Format file error handle
format_error:
    li a7, 4            # Service number for print string
    la a0, error_format # Load address of error_format Message
    ecall               # System Call
    j end               # Jump to end
```

- Prints the corresponding error message and jump to end process.

## 3 Result Observe: