

# 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:

```
1  .eqv BITMAP_DISPLAY, 0x10010000    # Memmory address of Bitmap Display
2
3  .data
4      # Message for user input
5      input_prompt: .asciz "Enter location and .bmp image to display (Ex: D:/image/image.bmp): "
6      # Message for resolution error handling
7      error_scale: .asciz "Invalid image scale (512x512)!\n"
8      # Message for image format error handling
9      error_format: .asciz "Invalid image format (.bmp)!\n"
10     # Storing user input for file path
11     input_buffer: .space 256
12     # Memmory for debugging or fixing the code (Addition)
13     debug_mem: .space 1100000
14     # Memmory to store the file content
15     main_mem: .space 1100000
16     # Message for open file error handling
17     error_open: .asciz "Cannot find the file!"
18
19 .text
20 # Main Program
21 .global main
22 main:
23
24     # Get file input
25     get_input:
26         li a7, 4          # Service number to print string
27         la a0, input_prompt # Address of input_prompt
28         ecall             # System Call
29
30         li a7, 8          # Service number for input string
31         la a0, input_buffer # Address hold the file path
32         li a1, 256        # Length of path input
33         ecall             # System Call
34
35     # Replace "\n" with "\0"
36     la t0, input_buffer   # Load the starting address of the input_buffer
37     remove_newline:
38         lb t1, (t0)        # Load a byte from the buffer
39         beq t1, zero, open_file # Branch If the byte is '\0' -> jump to open_file
40         li t2, 10         # Load ASCII value of newline ('\n')
41         beq t1, t2, replace_newline # Branch If the byte is '\n' -> jump to replace_newline
42         addi t0, t0, 1     # Move to the next byte
43         j remove_newline  # Repeat the process
44
45     # Replace the character
46     replace_newline:
47         sb zero, (t0)      # Replace '\n' with '\0'
48
49     # Open the file
50     open_file:
51         li a7, 1024        # Service number to open the file
52         la a0, input_buffer # Load address of the file path
53         li a1, 0           # Open file in Read-only mode
54         ecall             # System Call
55         addi t0, a0, 0     # Store the file descriptor in t0
56         blt t0, zero, file_error # Branch If file descriptor is negative -> jump to file_error
57
58     # Read the file header
59     read_file:
60         li t1, 54         # BMP file header size is 54 bytes
```

```

61     la    a1, main_mem      # Load address of main_mem to store the file header
62     addi  a0, t0, 0         # Load file descriptor
63     addi  a2, t1, 0         # Specify the length of bytes to read
64     li    a7, 63            # Load syscall number for "read file"
65     ecall                          # System Call
66     blt   a0, t1, file_error # Branch If read fails -> jump to file_error handling
67
68 # Check file format
69 check_file_format:
70     la    t2, main_mem      # Load address of the file header in memory
71     lbu   t3, 0(t2)         # Load the first byte of the header
72     lbu   t4, 1(t2)         # Load the second byte of the header
73     li    t5, 'B'           # ASCII value for 'B'
74     bne   t3, t5, format_error # Branch If first byte is not 'B' -> jump to format_error
75     li    t5, 'M'           # ASCII value for 'M'
76     bne   t4, t5, format_error # Branch If second byte is not 'M' -> jump to format_error
77
78 # Check image resolution
79 check_image_res:
80     addi  t3, t2, 18        # Move to the address of the width field in the header
81     lw    t4, 0(t3)         # Load the width of the image
82     addi  t3, t2, 22        # Move to the address of the height field in the header
83     lw    t5, 0(t3)         # Load the height of the image
84
85     li    t6, 512           # Maximum scale for the image allowed is 512x512
86     bgt   t4, t6, resolution_error # Branch If width is invalid -> jump to resolution_error
87     bgt   t5, t6, resolution_error # Branch If height is invalid -> jump to resolution_error
88
89 # Get begin position of pixel data
90 get_position:
91     addi  t3, t2, 10        # Move to the address of the pixel data offset in the header
92     lw    t6, 0(t3)         # Load the starting position of pixel data
93
94 # Move the pointer to the position of begin pixel data
95 file_seek:
96     addi  a0, t0, 0         # Load file descriptor
97     addi  a1, t6, 0         # Load the pixel data offset
98     li    a2, 0             # Specify SEEK_SET (absolute positioning)
99     li    a7, 62            # Service number for seek
100    ecall                          # System Call
101    blt   a0, zero, end      # Branch If seek fails -> jump to end
102
103 # Calculate pixel data for display on Bitmap Display
104 pixel_calculate:
105     li    s10, 3            # Each pixel is 3 bytes (RGB format)
106     mul    s7, t4, s10       # Calculate RowSize (Not include padding)
107     mul    s8, s7, t5        # Calculate PixelData = RowSize x Height
108
109 # Read the pixel data
110 read_pixel:
111     la    a1, main_mem      # Load address of main_mem to store pixel data
112     addi  a0, t0, 0         # Load file descriptor
113     addi  a2, s8, 0         # Specify the total size of pixel data
114     li    a7, 63            # Service number for read file
115     ecall                          # System Call
116     blt   a0, s8, end       # Branch If read fails -> jump to end
117
118 # Initial Bitmap Display for output
119 init:
120     li    a3, BITMAP_DISPLAY # Load the Bitmap Display base address
121     addi  s1, t5, 0          # Height of the image
122     addi  s2, t4, 0          # Width of the image
123

```

```

124 #-----
125 # Algorithm for displaying:
126 #
127 # s2 = width of the image.
128 # s1 = height of the image.
129 #
130 # Loop from bottom up for each rows.
131 # This purpose is to access first row of bitmap image.
132 #
133 # For each rows, loop from left to right to access each columns refers for each pixels.
134 # For each columns (aka pixels) in the row, process the color and display immediately to the
135 # display.
136 #
137 # The default color format of bitmap image it's BGR.
138 # Implementation of pixel in a row of bitmap image: [B1 G1 R1] [B2 G2 R2] [B3 G3 R3] ...
139 # To display image on Bitmap Display, we need to convert BGR color format into RGB color format
140 # It means convert 0x00BGGRR -> 0x00RRGGBB.
141 # Each value on the hexadecimal value it's 4 bit.
142 #
143 # To do that, we use shift left logic and bitwise operation to evaluate the exactly value of
144 # the color refers to RGB format to display the image with right color as accurately as
145 # possible.
146 #
147 #-----
148
149 # Algorithm implement
150 display:
151     loop_rows:
152         addi s1, s1, -1 # Decrease the row counter
153         blt s1, zero, end # Branch If all rows of the image processed -> jump to end
154         mul s9, s1, s7 # Calculate the offset to the current row's pixel data
155         la t3, main_mem # Get the address of the image header to access each pixel
156         add t3, t3, s9 # Get the begin address of the pixel data of the current row
157         addi s4, s2, 0 # Reset the width of the row after access each columns (aka pixels)
158         addi s5, t3, 0 # Init pointer to process each pixels in the row
159
160     loop_cols:
161         beq s4, zero, next_row # Branch if all pixels in a row processed -> jump to next row
162         lbu t1, 0(s5) # Get the B value 0xBB
163         lbu t2, 1(s5) # Get the G value 0xGG
164         lbu s11, 2(s5) # Get the R value 0xRR
165         slli s11, s11, 16 # Shift left logical s10 = 0x000000RR -> 0x00RR0000
166         slli t2, t2, 8 # Shift left logical t2 = 0x000000GG -> 0x0000GG00
167         or s11, s11, t2 # Logical OR: s10 or t2 -> 0x00RRGG00
168         or s11, s11, t1 # Logical OR: s10 or t1 -> 0x00RRGGBB
169         sw s11, 0(a3) # Store the value into Bitmap Display to display
170         addi a3, a3, 4 # Move to the next pixel on Bitmap Display (each 4 bytes)
171         addi s5, s5, 3 # Move to the next pixel on image after process 3 byte RGB format
172         addi s4, s4, -1 # Decrease the pixel counter of the image
173         j loop_cols # Jump to next pixel
174
175     next_row:
176         j loop_rows # Jump to next row
177
178 end:
179 # Close the file
180 close_file:
181     addi a0, t0, 0 # Load the file descriptor
182     li a7, 57 # Service number for clode the file
183     ecall # System Call
184
185 # Exit the Program
186 exit:

```

```

187         li a0, 0          # Load exit code (0)
188         li a7, 10         # Service number for exit program
189         ecall             # System Call
190
191     # Error Handling
192     # Open file error handle
193     file_error:
194         li a7, 4           # Service number for print string
195         la a0, error_open  # Load address of error_open Message
196         ecall             # System Call
197         j end             # Jump to end
198
199     # Resolution of image error handle
200     resolution_error:
201         li a7, 4           # Service number for print string
202         la a0, error_scale # Load address of error_scale Message
203         ecall             # System Call
204         j end             # Jump to end
205
206     # Format file error handle
207     format_error:
208         li a7, 4           # Service number for print string
209         la a0, error_format # Load address of error_format Message
210         ecall             # System Call
211         j end             # Jump to end
212

```

## 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

```

1     error_scale: .asciz "Invalid image scale (512x512)!\n"
2     error_format: .asciz "Invalid image format (.bmp)!\n"
3     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

```
1  .global main
2  main:
```

- Defines the main entry point for the program.

### 2.2.1 Input Handling

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

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

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

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

### 2.2.2 Replace Newline with Null Terminator

```
1  # Replace "\n" with "\0"
2      la t0, input_buffer          # Load the starting address of the input_buffer
3  remove_newline:
4      lb t1, (t0)                  # Load a byte from the buffer
5      beq t1, zero, open_file      # Branch If the byte is '\0' -> jump to open_file
6      li t2, 10                    # Load ASCII value of newline ('\n')
7      beq t1, t2, replace_newline  # Branch If the byte is '\n' -> jump to replace_newline
8      addi t0, t0, 1               # Move to the next byte
9      j remove_newline             # Repeat the process
10
11 # Replace the character
12 replace_newline:
13     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

```
1  # Open the file
2  open_file:
3      li a7, 1024                # Service number to open the file
4      la a0, input_buffer        # Load address of the file path
5      li a1, 0                   # Open file in Read-only mode
6      ecall                     # System Call
7      addi t0, a0, 0             # Store the file descriptor in t0
8      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

```
1  # Read the file header
2  read_file:
3      li    t1, 54          # BMP file header size is 54 bytes
4      la    a1, main_mem    # Load address of main_mem to store the file header
5      addi  a0, t0, 0        # Load file descriptor
6      addi  a2, t1, 0        # Specify the length of bytes to read
7      li    a7, 63          # Load syscall number for "read file"
8      ecall                          # System Call
9      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

```
1  # Check file format
2  check_file_format:
3      la    t2, main_mem    # Load address of the file header in memory
4      lbu   t3, 0(t2)        # Load the first byte of the header
5      lbu   t4, 1(t2)        # Load the second byte of the header
6      li    t5, 'B'          # ASCII value for 'B'
7      bne   t3, t5, format_error # Branch If first byte is not 'B' -> jump to format_error
8      li    t5, 'M'          # ASCII value for 'M'
9      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

```
1  # Check image resolution
2  check_image_res:
3      addi  t3, t2, 18        # Move to the address of the width field in the header
4      lw    t4, 0(t3)         # Load the width of the image
5      addi  t3, t2, 22        # Move to the address of the height field in the header
6      lw    t5, 0(t3)         # Load the height of the image
7
8      li    t6, 512           # Maximum scale for the image allowed is 512x512
9      bgt   t4, t6, resolution_error # Branch If width is invalid -> jump to resolution_error
10     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

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

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

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

```

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

```

- Moves the file pointer to the pixel data.

```

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

```

- Calculates the total size of the pixel data.

```

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

```

- Reads the pixel data into `main_mem`.

## 2.2.8 Display Pixel Data

```

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

```

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

```

1  #-----
2  # Algorithm for displaying:                                     /
3  #                                                                 /
4  # s2 = width of the image.                                     /
5  # s1 = height of the image.                                   /
6  #                                                                 /
7  # Loop from bottom up for each rows.                           /
8  # This purpose is to access first row of bitmap image.         /
9  #                                                                 /
10 # For each rows, loop from left to right to access each columns refers for each pixels. /
11 # For each columns (aka pixels) in the row, process the color and display immediately to the /
12 # display.                                                       /
13 #                                                                 /
14 # The default color format of bitmap image it's BGR.           /
15 # Implementation of pixel in a row of bitmap image: [B1 G1 R1] [B2 G2 R2] [B3 G3 R3] ... /
16 # To display image on Bitmap Display, we need to convert BGR color format into RGB color format /
17 # It means convert 0x00BBGGRR -> 0x00RRGGBB.                   /
18 # Each value on the hexadecimal value it's 4 bit.             /
19 #                                                                 /
20 # To do that, we use shift left logic and bitwise operation to evaluate the exactly value of /
21 # the color refers to RGB format to display the image with right color as accurately as /
22 # possible.                                                       /

```



```

23 # /
24 #-----

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

```

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

```

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

```

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

```

1  next_row:
2      j loop_rows # Jump to next row

```

- Moves to the next row of the image.

## 2.2.9 End Program

```

1  end:
2      # Close the file
3      close_file:
4          addi a0, t0, 0 # Load the file descriptor
5          li a7, 57 # Service number for close the file
6          ecall # System Call

```

- Execute close the file after processed.

```

1  # Exit the Program
2  exit:
3      li a0, 0 # Load exit code (0)
4      li a7, 10 # Service number for exit program
5      ecall # System Call

```

- Exit the program after finished all.

## 2.3 Error Handling

```

1  # Error Handling
2  # Open file error handle
3  file_error:
4      li a7, 4          # Service number for print string
5      la a0, error_open # Load address of error_open Message
6      ecall            # System Call
7      j end            # Jump to end
8
9  # Resolution of image error handle
10 resolution_error:
11     li a7, 4          # Service number for print string
12     la a0, error_scale # Load address of error_scale Message
13     ecall            # System Call
14     j end            # Jump to end
15
16 # Format file error handle
17 format_error:
18     li a7, 4          # Service number for print string
19     la a0, error_format # Load address of error_format Message
20     ecall            # System Call
21     j end            # Jump to end

```

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

## 3 Result Observe:

