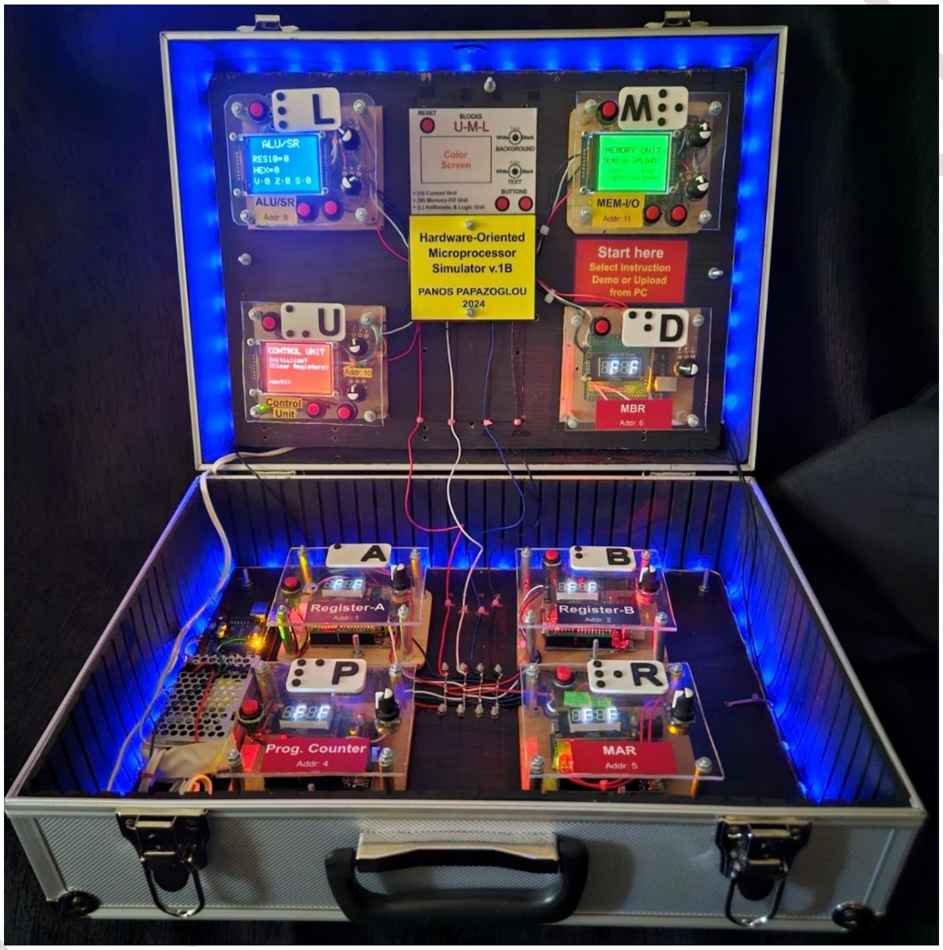
HARDWARE-ORIENTED MICROPROCESSOR SIMULATOR (HOMS v.1B)

OPEN-SOURCE PROJECT



Dr. Panayotis (Panos) Papazoglou

8/2024

Contents

HARDWARE-ORIENTED MICROPROCESSOR SIMULATOR (HOMS v.1B)

At-A-Glance:

- A Handmade and Open-Source project
- A full-working hardware-oriented simulator
- Microprocessor/ Microcomputer simulator
- Visual Aids (Contrast adjustment, Brightness adjustment, Braille Labels, Color selection)
- Educational tool
- Based on Arduino platform
- Easy reproduction
- Custom educational scenarios
- Suitable for academic teachers and researchers in the field of engineering education

Keywords
Microprocessor simulator, Arduino-based educational tool, Hardware-oriented simulator, Open-source simulator

CHAPTER 1
General information

The HOMS v.1B system, constitutes a very different approach regarding the microprocessor simulation and faces effectively the corresponding limitations of the other relevant tools. Table 1.1 shows some differences between existing tools and the HOMS system.

Table 1.1

Feature	**Software simulation**	**FPGA technology**	**HOMS system**
Hardware point of view	NO	YES	YES
Hands on	NO	YES	YES
Complexity	LOW	HIGH	LOW
Architectural point of view (component level)	YES	NO	YES
Ease of use	YES	NO	YES
Custom architecture	NO	YES	YES
Custom assembly instructions	NO	YES	YES
Touch and Feel	NO	NO	YES
Platform type	PC	PC and board	Autonomous and PC

Prior to HOMS system (versions 1 and 1B), a novel hybrid simulation platform has been proposed in the literature from the same author (Papazoglou, P., 2018). This platform is based on original designed PCBs with SMD technology. On the other hand, the above proposed implementation (Papazoglou, P., 2018) has limitations such as board assembly complexity, high cost, different board technologies and complex operation software. While the question for replacing microprocessor software simulators with hybrid approaches remains, a new educational tool for studying microprocessor architecture and programming has to be proposed for facing effectively all the previous tools limitations regarding construction, programming and operation complexity.
In this project, a fully working and mature educational tool for learning microprocessors is proposed for higher education in the field of computer science and computer engineering. The proposed educational tool faces effectively every limitation of the previous versions, is based on open-source hardware and can be reproduced by everyone. Figure 1.1 shows the implementation of the HOMS v.1B tool which is an 8bit microprocessor/system model. This model consists of similar blocks that represent

microprocessor internal components. The HOMS tool has also a memory/output unit for supporting memory data entry and data output. Is a full working custom system, where the corresponding developer can build its own assembly language and choose their desired microprocessor components.

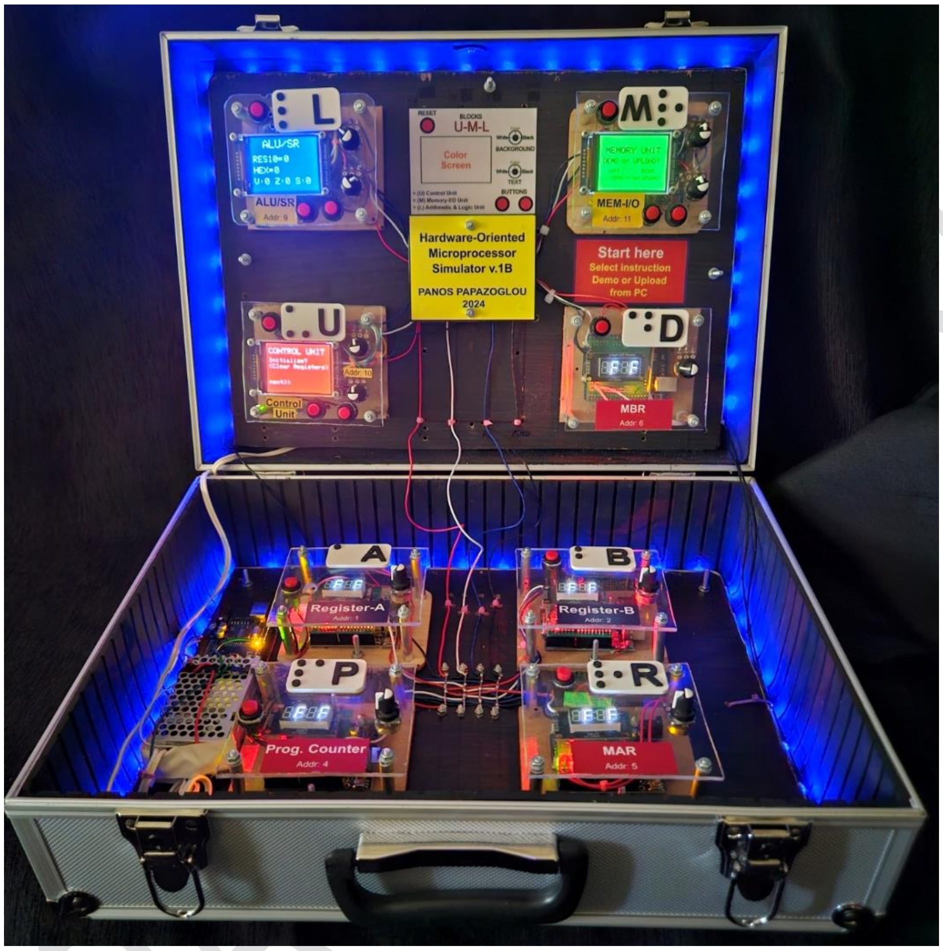


Fig. 1.1 The HOMS v.1B tool

CHAPTER 2
System description

2.1 Introduction

A microprocessor consists of various internal units for performing instruction execution. The internal units interact with each other by exchanging data. On the other hand, microprocessor fetches instructions from external memory and if needed, the corresponding results are returned to memory again. For building an operational microsystem (microcomputer), a microprocessor and a memory unit are needed as well as an input and an output unit. Figure 2.1 shows a typical model that represents a simple microprocessor.
Based on this model, the proposed hardware-oriented educational tool (HOMS) consists of the necessary units that form a simple microprocessor, a memory and supports data entry in memory as well as data output.

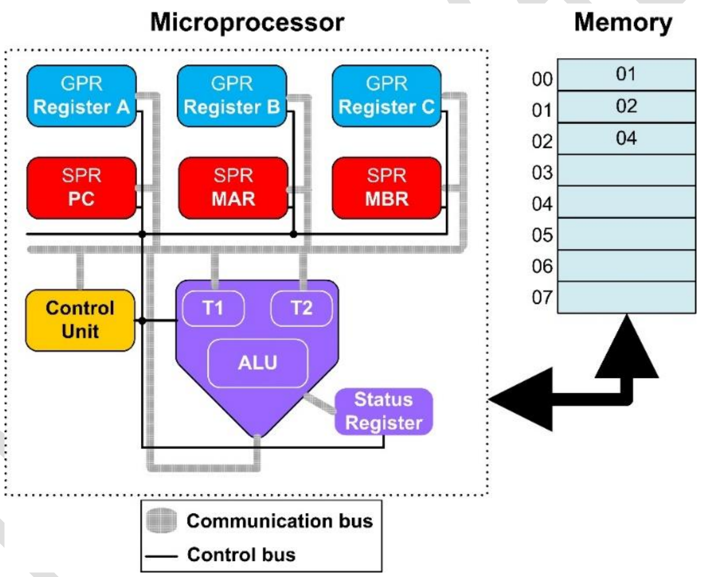


Fig. 2.1 Typical components of a single microprocessor

The main goal of a microprocessor is the instruction execution. Instructions are part of the program which is stored in main memory. Instructions have to be transferred from memory to microprocessor (fetching) via data bus. The instruction execution procedure is simplified in steps, as follows (assume that the addition A+B will be performed):

Step 1: Read PC (Program counter) for finding where the next instruction address is

Step 2: Store this address to MAR (Memory Access Register) in order to place the desired memory address on address bus

Step 3: Fetch instruction code via data bus and store this code in MBR (Memory Buffer register)

Step 4: Fetch parameter code via data bus and store this code in MBR (Memory Buffer register)

Step 5: Decode instruction data and execute, a) copy the contents of registers A and B to ALU registers T1 and T2 respectively, b) perform the addition T1+T2 and update SR (Status Register), c) store result to register A

Step 6: Update PC (Program Counter) for fetching next instruction code from memory

There is no hardware tool/kit in the market that shows the above procedure step by step. The proposed hardware-oriented tool (HOMS) consists of real hardware components that represent the microprocessor internal units, as shown in fig. 2.1.

2.2 HOMS tool Overview

The HOMS is a fully working prototype that offers unique features as compared to similar tools. Table 2.1 summarizes the supported features.

Table 2.1
HOMS tool unique features

Customizable architecture	The teacher or student can use any number or type of blocks for building the preferred microprocessor architecture.
Block reusability	The microprocessor units are based on the same board (e.g. Arduino UNO). For example, the HOMS v.1B tool consists of eight identical blocks. Thus, the embedded software determines the block functionality (same block, different functionality based on software).
Programmable functionality	Based on the embedded software, a block operates like a register or control unit or ALU or special register, etc.
Experimental architecture	Based on the number and type of blocks, a teacher or student can test a prototype architecture or to expand an operation to smaller steps by using more blocks.
Assembly instructions development	The existing blocks support functionality that is controlled by the control unit. The HOMS user is free to build any assembly instruction which is supported by the software inside control unit.
Student skills emerge	Block building and assembly instructions can be developed by students. Students

	use their mind and hands and learn to perform accurate manipulations and movements to bring the hardware to life.
Educational scenarios	Using the default HOMS tool architecture, teachers can develop the desired assembly instructions for building different educational scenarios. Based on the embedded software (inside blocks), the behavior of the blocks can be adapted to the desired educational scenarios.
Complete approach	Using the HOMS v.1B tool, the microprocessor can be approached from many different points of view/operation: a) a programmer/user develops and tests assembly code using exclusively the available system instructions b) a software/hardware developer adapts the embedded software for supporting the desired microprocessor functionality/operation c) a teacher or student selects the desired blocks and builds an entirely new architecture d) a student is simply watching/studying the instruction execution procedure/sequence
Multiple points of view	Based on the different points of view as mentioned previously, every student can extract information from the desired microprocessor feature.
Hardware point of view	The HOMS tool, emphasizes the hardware layer which is hidden in the existing simulation tools. Thus the "connection" of instruction, operation and hardware implementation is more clear in the student's minds.
Standalone tool	The HOMS v.1B tool does not need a PC and can be operated autonomously. Thus, constitutes a mobile laboratory system unit.
Easy reproduction	The hardware components of the HOMS tool can be found easily in any market. On the other hand, the multiple identical

	blocks support easily the reproduction procedure.
Open features	The main advantage of the implemented HOMS tool is the object-oriented approach and the open-source hardware which gives the freedom to any developer not only to reproduce the same tool but also to implement the whole simulator using different blocks (with or without an LCD, buttons, etc). Note that the embedded software makes the difference.

Based on the mentioned functionality and features, it is obvious that the HOMS v.1B tool is more suitable for laboratory exercises in higher education in the field of computer science and engineering. ***Laboratory academic stuff may use the proposed prototype HOMS tool for building multiple boards and fully support a semester course. Moreover, the existing HOMS tool can be extended under the development of a thesis or a lab assignment or even under a research program for exploring new methods and tools in engineering education.***

CHAPTER 3
Hardware components

3.1 Microprocessor blocks

Figure 3.1 shows the hardware block types that are used as internal microprocessor components and external memory. There are two types of blocks: a) **Register blocks** (Register-A, Register-B, Prog. Counter, MAR and MBR), b) **Special blocks** (Control Unit, ALU/SR and Memory-I/O). All blocks are based on Arduino UNO. Register blocks use seven-segment displays and special blocks use TFT color displays.

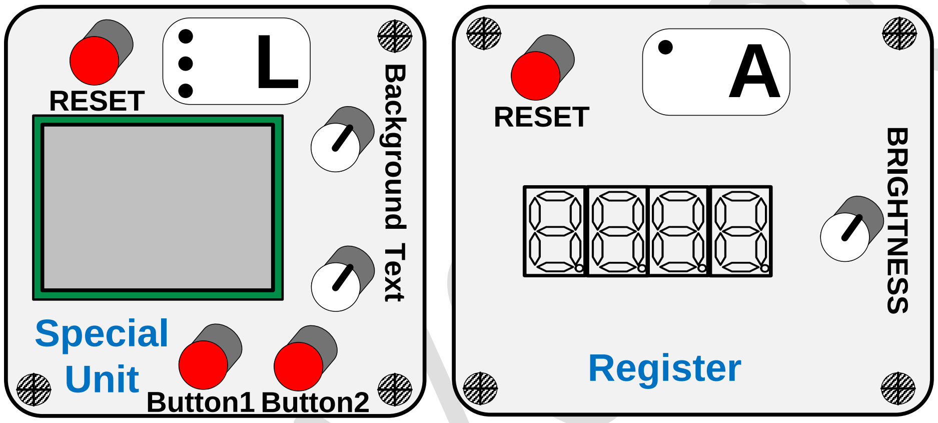


Fig. 3.1 Block/Component types

Figures 3.2 and 3.3 show the circuit connections between Arduino UNO and the seven-segment module as well as the corresponding physical implementation. The seven-segment module is based on TM1637. The same implementation is used for Registers A, B, PC (Prog. Counter), MAR and MBR.

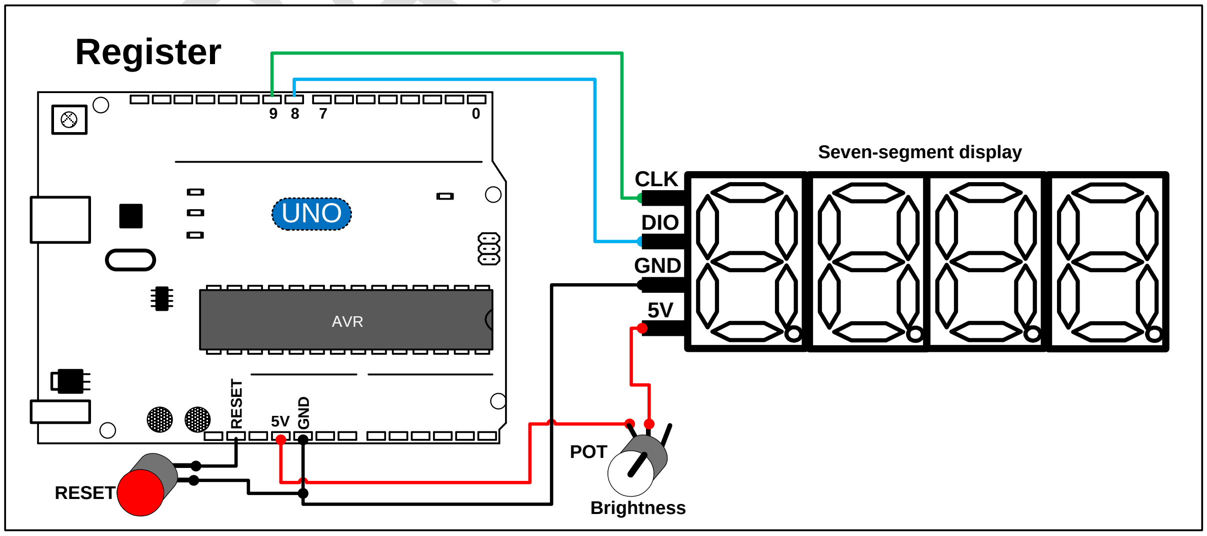


Fig. 3.2 Register circuit

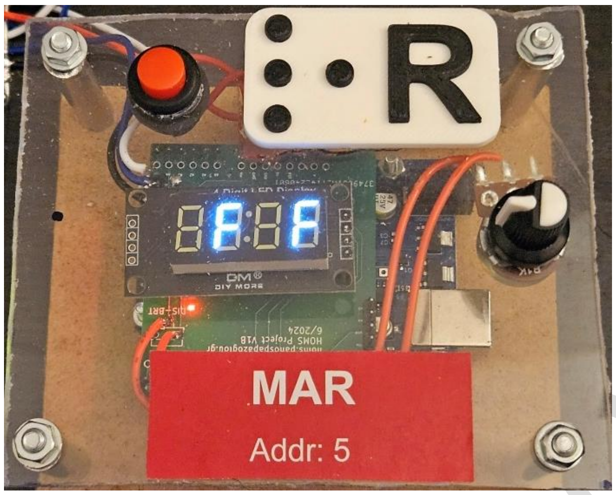


Fig. 3.3 Physical implementation (e.g. MAR register)

As shown in fig. 3.2, one Reset button and a brightness adjustment knob are used. The Reset button connects the RESET pin of Arduino to GND and the POT is used as variable resistor for controlling the current flow in 5V line. Thus, no code is needed for the corresponding operation of the button and the brightness knob.
Figures 3.4 and 3.5 show the circuit of the special block as well as the corresponding physical implementation. The same implementation is used in special blocks such as Control Unit, Memory-I/O and ALU/SR.

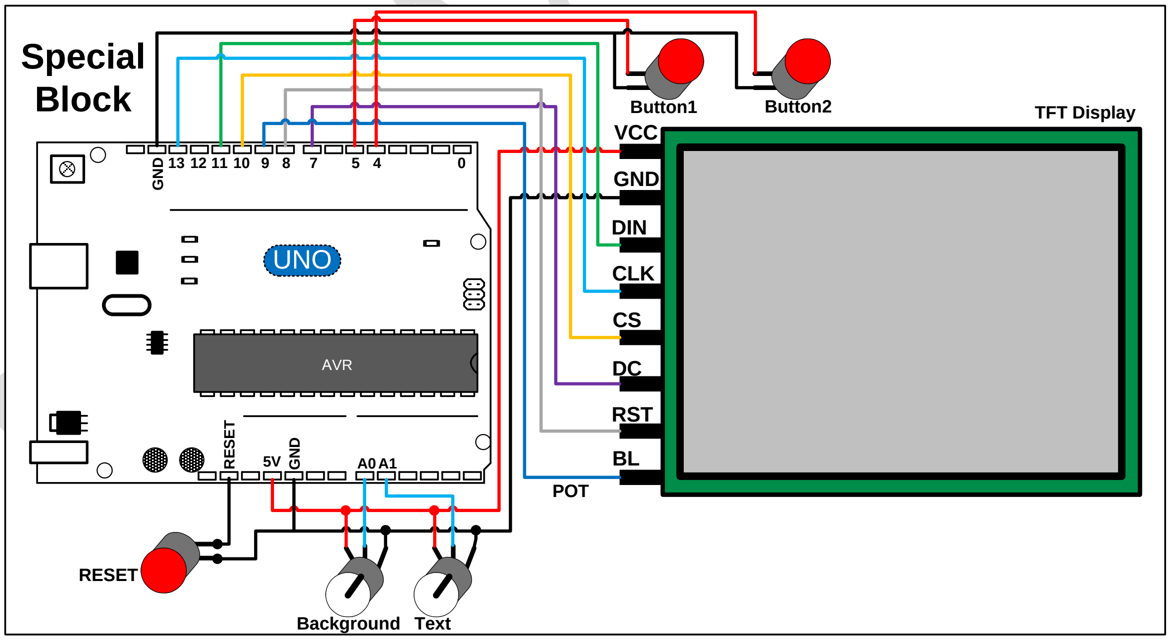


Fig. 3.4 Special block circuit

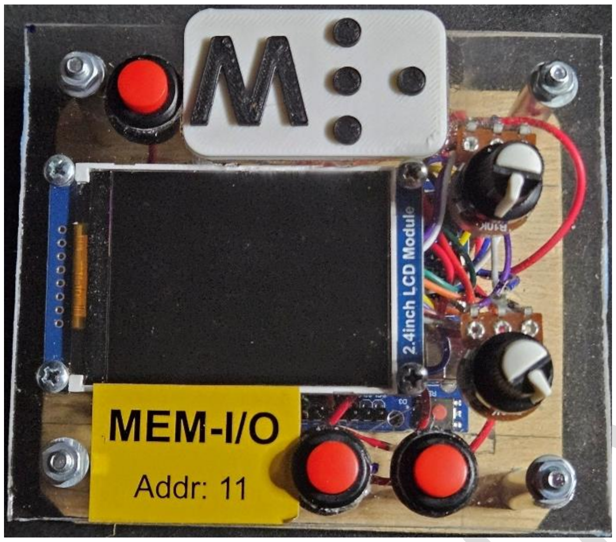


Fig. 3.5 Physical implementation (e.g. MEM-I/O Unit)

As shown in fig. 3.4, there are three buttons and two POTs. One button is for resetting Arduino and the other two for user selection based on display instructions. The button operation is supported through the pull-up internal resistors of the microcontroller that are activated within the code. On the other hand, two POTs are used for adjusting background color and text color respectively. The POTs are connected to analog inputs A0 and A1. When the POT in A0 is turned all the way left the resulting color is BLACK and if the POT is turned all the way right, then the resulting color is WHITE. For any other position of the POT, the resulting color has a predefined color value. Thus, High-Contrast display can be achieved.

3.2 RGB LED-Strip frame

Figure 3.6 shows the RGB LED-Strip frame around the suitcase and fig. 3.7 the physical view of the installed hardware components.

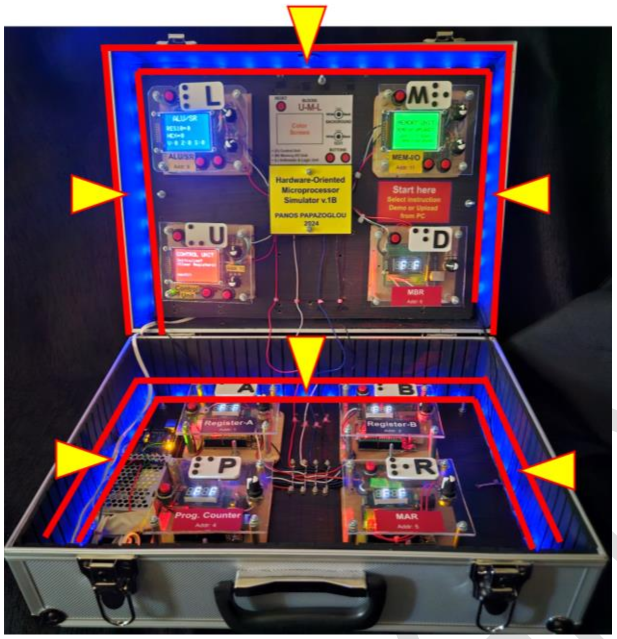


Fig. 3.6 RGB LED-Strip frame

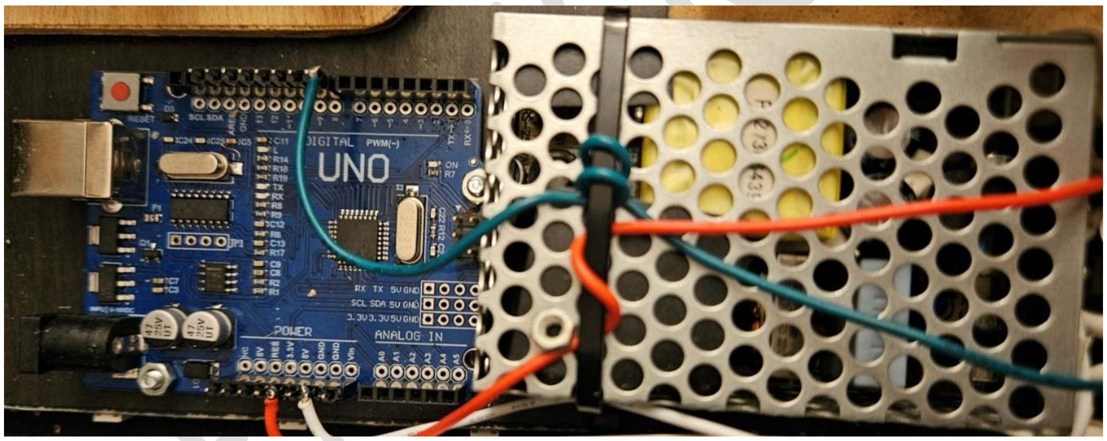


Fig. 3.7 RGB LED-Strip controller with power supply

Figure 3.8 shows the electrical circuit for controlling the RGB LED-Stripe. Note that the external power supply gives power also to Arduino. Thus, the RGB LED-Strip frame does not depend on the HOMS system.

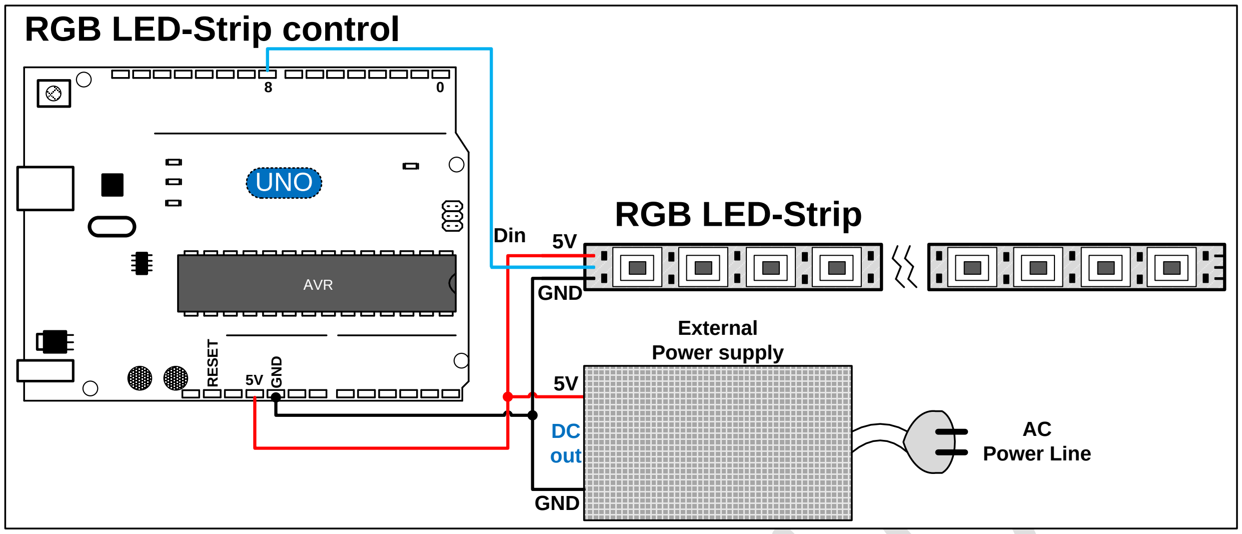


Fig. 3.8 RGB LED-Strip control circuit

3.3 PCB design (optional)

For simplifying the register block assembly, a PCB has been designed as Arduino shield. The shield hosts the seven-segment module without the need for wiring and can be plugged directly to Arduino pin-sockets. The PCB was designed for hosting the two-digit seven-segment (RED) from the version 1 of the HOMS system. Figure 3.9 shows the PCB-Shield connected to Arduino for hosting the seven-segment display from the version 1 of HOMS system.

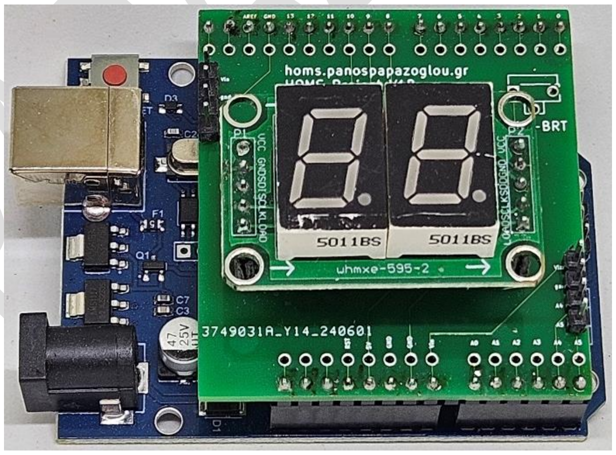


Fig. 3.9 Arduino shield for two-digit seven segment display

After some tests, the final decision was to replace the two-digit RED display with a four-digit BW display, but the pins of the new display did not match the pins of the shield (different dimensions and different pins). The double 5V and GND pins of the shield regarding the display module was not connected together, because this is achieved through the module itself. For solving this problem, two wires are installed for connecting the pins together (5V-5V and GND-GND). On the other hand, the new display module has four pins and not five like the two-digit module. For matching fewer pins, power and data/clock lines, the new module has been placed with 180 degrees of rotation. Figure 3.10 shows the two soldered wires for 5V and GND.

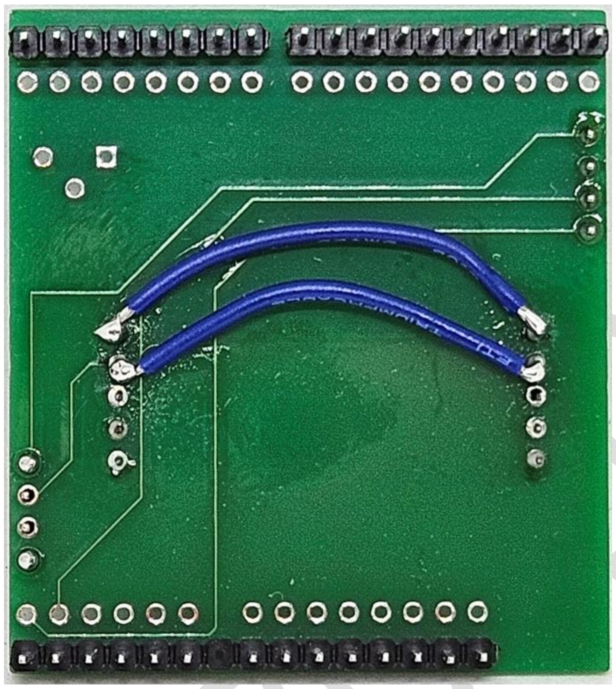


Fig. 3.10 Additional wires for the new module

After the necessary shield modifications, the four-digit module is finally installed (fig. 3.11).

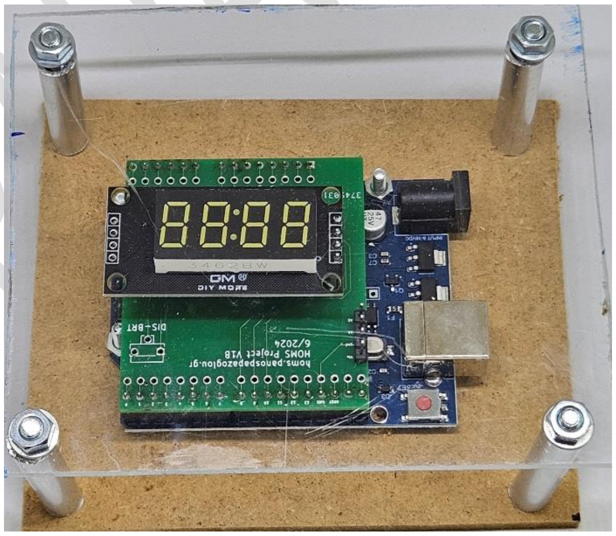


Fig. 3.11 Using the PCB shield for installing the new seven-segment module

***Important note:* please visit the web site of the HOMS project for finding the experimental version of the PCB design.**

CHAPTER 4
HOMS software & GUI

***Important note 1:* only indicative instructions and operations have been implemented in the following source-codes for supporting basic system functionality.**

***Important note 2:* the only code updates as compared to HOMS v.1 are mainly for supporting the new display modules (four digit seven-segment display and TFT display) as well as the new messages that can be displayed in the larger TFT of 2.4 inches.**

4.1 Register Unit

Every register Unit (Register block, fig. 4.1) contains identical code for supporting instructions (operations) such as LOAD, READ, INC, DEC, SHIFT and RESET. Three basic functions (fig. 4.2) support the I^{2}C communication (onReceive, onRequest) and seven-segment display operation (display).

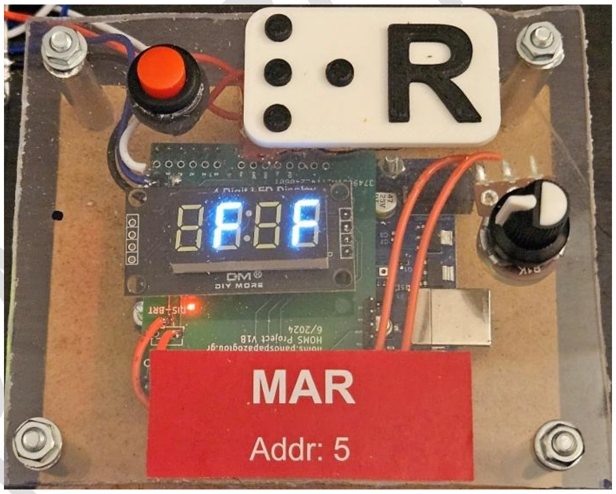


Fig. 4.1 Register Unit (e.g. MAR)

When a single byte is received for executing an instruction regarding the register content, the function onReceive is activated (interrupt-based process). On the other hand, when the control unit requests an answer (e.g. READ register content), the function onRequest is activated (interrupt-based process). Finally, the function display is activated when the seven-segment display value has to be updated.

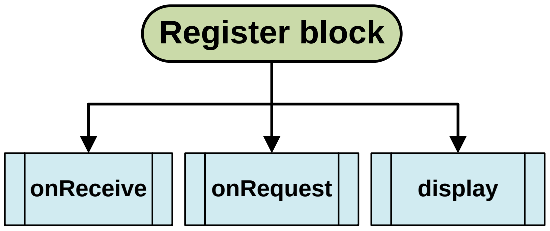


Fig. 4.2 Register functions

Figure 4.3 shows the operation of the function onReceive. In the case of the LOAD instruction, two bytes are used: one for the LOAD instruction (previous byte) and one for the value (current byte) that will be stored in the register. The function checks always the instruction code value for executing the corresponding operation (fig. 4.3).

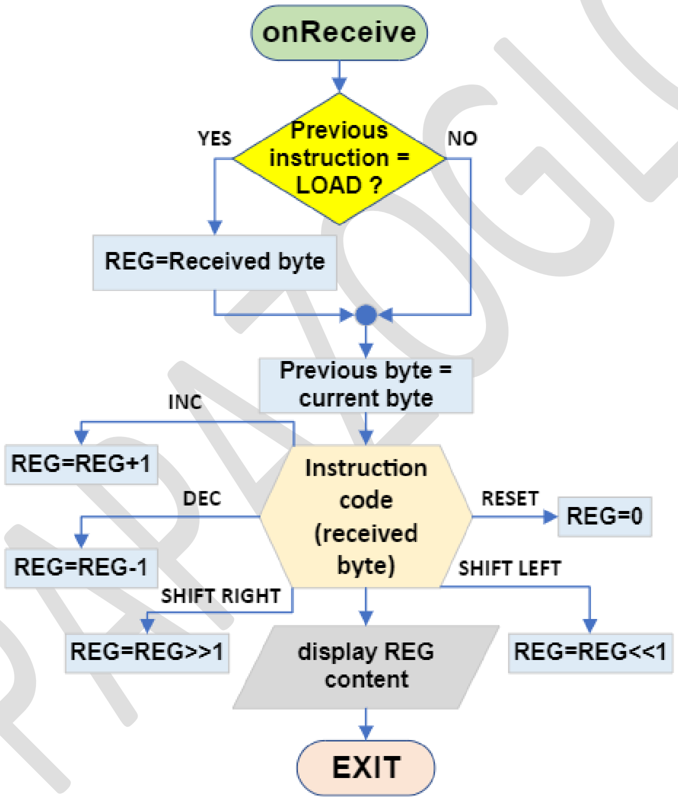


Fig. 4.3 onReceive function operation

Figure 4.4 shows the operation of the function onRequest.

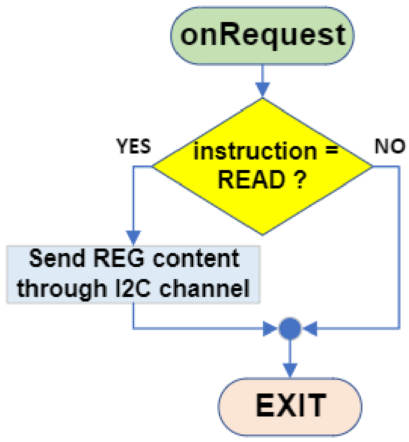


Fig. 4.4 onRequest function operation

SOURCE-CODE:

```
/********************************************
REGISTER UNIT
All registers execute the same code,
the only difference is the I2C address
********************************************

HOMS version 1B
Hardware-Oriented Microprocessor
Simulator

(C) Panayotis (Panos) Papazoglou
https://homs.panospapazoglou.gr/

LICENSE:
Creative Commons
CC BY NC SA
International License
********************************************/

//Change address based on block that is used
#define ADDRESS 6


//new seven segment
#include <Arduino.h>
#include <TM1637Display.h>

//Seven segment pins
```

```
#define CLK 9
#define DIO 8

TM1637Display display(CLK, DIO);
uint8_t data[] = { 0x00, 0x00, 0x00, 0x00 };

//Include I2C library
#include <Wire.h>

//Define symbolic names for instructions codes
//between blocks
#define READ 65
#define RESET 66
#define SHIFT_L 67
#define SHIFT_R 68
#define DEC 69
#define INC 70
#define LOAD 71
#define IDON 99
#define IDOFF 98
#define IDOK 97

int identify = 0;

//byte that is received via I2C communication
byte rV;

//Previous received byte. Is used when a sequence of bytes
//has to be checked
byte rVprev = 0;

//Initial register content
byte REG = 0xff;

//Starting function (after reset or power on), runs once
void setup() {

  display.setBrightness(0x0f);

  //Initialize Serial communication
  //for displaying debugging messages
  Serial.begin(9600);
  Serial.println("hello from MBR");

  //Display register content
  displayR(REG);

  //Initialize I2C communication:
  //Set address
  //Enable ISR for receiving bytes and requests
  Wire.begin(ADDRESS);
  Wire.onReceive(onReceive);
  Wire.onRequest(onRequest);
}

void loop() {
  //Nothing here
}

/*******************************************
Action Routine that is activated when
a byte is received
*******************************************/
void onReceive(int a)

{

  //Read received byte
  rV = Wire.read();
  Serial.println("Byte received!");
  //If previous byte is 71 (LOAD), then the current byte
  //is the value that will be loaded in register
  if (rVprev == LOAD) REG = rV;

  //Update variable for previous value
  rVprev = rV;

  //Actions based on received byte
  if (rV == INC) REG++;
  if (rV == DEC) REG--;
  if (rV == SHIFT_R) REG = REG >> 1;
  if (rV == SHIFT_L) REG = REG << 1;
  if (rV == RESET) REG = 00;

  //Display register content after performed action
  displayR(REG);
}

/*******************************************
Action Routine that is activated when
```

```
a request is received
*******************************************/
void onRequest() {

  //If the received byte represents
  //the READ command, then the
  //register content is sent as answer
  if (rV == READ) Wire.write(REG);
  if (rV == IDON) Wire.write(IDOK);


  //Display register content
  displayR(REG);
}

/*******************************************
Display register content as HEX number on
seven segment module
Digit manipulation for left and right
seven segment display unit
*******************************************/
void displayR(byte REGnum) {

  //Convert decimal number REGnum to HEX
  String Shex = String(REGnum, HEX);

  //Variables for left and right digit
  char LL;
  char RR;

  //If the hex number has only one digit, then
  //the left digit is zero ('0') and the right digit
  //is the first character of the string Shex
  if (Shex.length() == 1) {
    LL = '0';
    RR = Shex[0];
  }

  //Otherwise, update left and right digit variables from
  //the whole string Shex
  else {
    LL = Shex[0];
    RR = Shex[1];
  }
  //variables for accessing later the hex[] array
  //in order to activate the correct segment
  //on seven segment module
  byte left, right;

  if (LL >= 'a' && LL <= 'f') left = LL - 87;
  else if (LL >= '0' && LL <= '9') left = LL - 48;

  if (RR >= 'a' && RR <= 'f') right = RR - 87;
  else if (RR >= '0' && RR <= '9') right = RR - 48;

  display2(left, right);
}

/******************************************
Digit manipulation
final display on
seven segment unit
******************************************/
void display2(int d1, int d3) {
  display.clear();
  data[1] = display.encodeDigit(d1);
  data[3] = display.encodeDigit(d3);
  display.setSegments(data);
}
```

4.2 Arithmetic & Logic Unit (ALU) and Status Register (SR)

In current code version, only the ADD instruction is implemented.

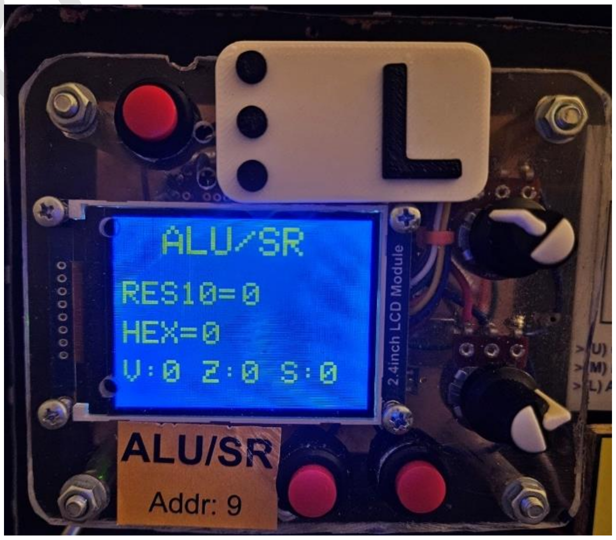


Fig 4.5 ALU-SR unit

The result is displayed on the TFT screen (fig 4.5) and the Status Register is updated. The result is available inside the CPU model (simulator) through the internal data bus.

The following pseudo-code represents the above steps for the main function operation:

```
START
      Byte received?
      YES
        *     Instruction=ADD
        *     YES
        *       *     T1=Read Register A
        *       *     T2=Read Register B
        *       *     Result=T1+T2
        *       *     Update Status Register (SR)
        *       *     Display REG content
        *     NO
        *       -
      NO
       -
END
```

The status register (SR) is declared as a three element array (*byte SR[]={0,0,0};*)

SOURCE-CODE:

```
/*******************************************
ALU (Arithmetic and Logic Unit)
SR (Status Register)
UNIT
*******************************************
HOMS version 1B
Hardware-Oriented Microprocessor
Simulator

(C) Panayotis (Panos) Papazoglou
https://homs.panospapazoglou.gr/

LICENSE:
Creative Commons
CC BY NC SA
International License
*******************************************/

//UNIT ADDRESS = 9
```

```
#define ADDRESS 9

//Include I2C library
#include <Wire.h>
#define FREQ 10000
//new tft section

#define ILI9341_BLACK 0x0000
#define ILI9341_BLUE 0x001F
#define ILI9341_WHITE 0xFFFF

#include "SPI.h"
#include "Adafruit_GFX.h"
#include "Adafruit_ILI9341.h"

//TFT pins
#define TFT_CLK 13
#define TFT_MISO 12
#define TFT_MOSI 11
#define TFT_DC 7
#define TFT_CS 10
#define TFT_RST 8

Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC, TFT_MOSI, TFT_CLK,
TFT_RST, TFT_MISO);

//This UNIT Receives/Sends commands from/to other units as bytes
#define READ 65
#define ADD 72
#define IDON 99
#define IDOFF 98
#define IDOK 97

#define GREEN_button 4
#define RED_button 5

//Variable that is used inside ISR
volatile byte rvTRUE = 0;

//Byte received from other units
byte rV;

//Store result of the numerical calculation
byte result = 0;
```

```
//Store the content of temporarily registers T1 and T2
byte T1 = 0;
byte T2 = 0;

//Flag array
byte SR[] = { 0, 0, 0 };  //VF (oVerflow), ZF (Zero), SF (Sign)

int bcolor = ILI9341_BLACK, fcolor = ILI9341_WHITE;


//Starting function (after reset or power on), runs once
void setup() {

  init_buttons();
  tft.begin();
  tft.setRotation(135);

  //Read POT for background color
  bcolor = analogRead(0);

  if (bcolor < 50) bcolor = ILI9341_BLACK;
  else if (bcolor > 975) bcolor = ILI9341_WHITE;
  else
    bcolor = ILI9341_BLUE;

  //Read POT for text color
  fcolor = analogRead(1);

  if (fcolor < 50) fcolor = ILI9341_BLACK;
  else if (fcolor > 975) fcolor = ILI9341_WHITE;
  else
    fcolor = ILI9341_YELLOW;

  tft.fillScreen(bcolor);

  //Start I2C communication
  Wire.begin(ADDRESS);
  Wire.setClock(FREQ);

  //Activate Receive & Request ISR routines
  Wire.onReceive(onReceive);
  Wire.onRequest(onRequest);

  //Display result and flag status
  DISPLAY_RES();
```

```
}

void loop() {
  //If a byte is received and is an ADD command,
  //then execute addition between the contents of Registers A and B
  //Store result in a variable, update SR[] array
  //and display result/SR contents

  if (rvTRUE == 1) {
    if (rV == ADD) {
      T1 = READ_REG(1);
      delay(100);
      T2 = READ_REG(2);
      result = T1 + T2;
      UPDATE_SR(result);
      DISPLAY_RES();
    }
    rvTRUE = 0;
  }
}

/********************************************
ISR routine
Is activated when a byte is received
via I2C communication
********************************************/
void onReceive(int a) {
  rV = Wire.read();

  //Flag for activating code in order to read Reg. A and REg. B contents
  //I2C functions can not be called within an active I2C routine
  rvTRUE = 1;
}

/********************************************
ISR routine
Is activated when a request is received
via I2C communication
********************************************/
void onRequest() {
  //Send result (ALU calculation) as answer to an I2C request

  if (rV == IDON) Wire.write(IDOK);
  else
    Wire.write(result);
```

```
}

/*******************************************
Read the content of a Register
with address addr
*******************************************/
byte READ_REG(int addr) {
  byte ans;

  Wire.beginTransmission(addr);
  Wire.write(READ);
  Wire.endTransmission();

  //Receive answer
  Wire.requestFrom(addr, 1);
  if (Wire.available())
    ans = Wire.read();

  return ans;
}

/*******************************************
Update Status Register Flags based
on result status
*******************************************/
void UPDATE_SR(byte res) {

  if (res == 0) SR[1] = 1;
  else SR[1] = 0;
  if (res < 0) SR[2] = 1;
  else SR[2] = 0;
  if (res > 255) {
    SR[0] = 1;
    result = 0;
  } else SR[0] = 0;
}

/*******************************************
Display result and Status Register Flags
on LCD screen
*******************************************/
void DISPLAY_RES() {

  display(60, 10, 5, 0, "ALU/SR");
  int y = 80;
  display(10, y, 4, 0, "RES10=");
  display(160, y, 4, 0, String(result));

  y += 50;
  display(10, y, 4, 0, "HEX=");
  display(110, y, 4, 0, String(result, HEX));

  y += 50;
  display(10, y, 4, 0, "V:");
  display(60, y, 4, 0, String(SR[0]));
  display(110, y, 4, 0, "Z:");
  display(160, y, 4, 0, String(SR[1]));
  display(210, y, 4, 0, "S:");
  display(260, y, 4, 0, String(SR[2]));
}

/******************************************
Unit title
******************************************/
void tft_welcome() {
  tft.fillScreen(bcolor);
  display(30, 50, 4, ILI9341_GREEN, "Memory Unit");
}

/******************************************
Display text on TFT
on LCD screen

col: column
row: row
txsize: text size
txcolor: not used here (optional for debugging purposes)
val: text to be displayed
******************************************/
void display(int col, int row, int txsize, int txcolor, String val) {

  tft.setCursor(col, row);
  tft.setTextSize(txsize);
  tft.setTextColor(fcolor);
  tft.print(val);
}

/******************************************
Initialize buttons by activating
internal pull-up resistors
```

```
**********************************************/
void init_buttons() {
  pinMode(GREEN_button, INPUT_PULLUP);
  pinMode(RED_button, INPUT_PULLUP);
}
```

***Important note:* The GREEN_button and RED_button pins in source code, represent the Left and Right buttons in current hardware blocks respectively (the GREEN and RED buttons were used in HOMS version 1).**

4.3 Memory and I/O system

The HOMS system executes the assembly instructions that are stored inside the memory. Thus, a memory system is implemented. Additionally, the HOMS supports user input for entering assembly instructions in memory through a computer-based application. Instructions are entered within the GUI environment and then are uploaded to memory unit through the USB connection. Also, a demo instruction is already stored inside the memory unit, and thus can be executed without the need of a computer.

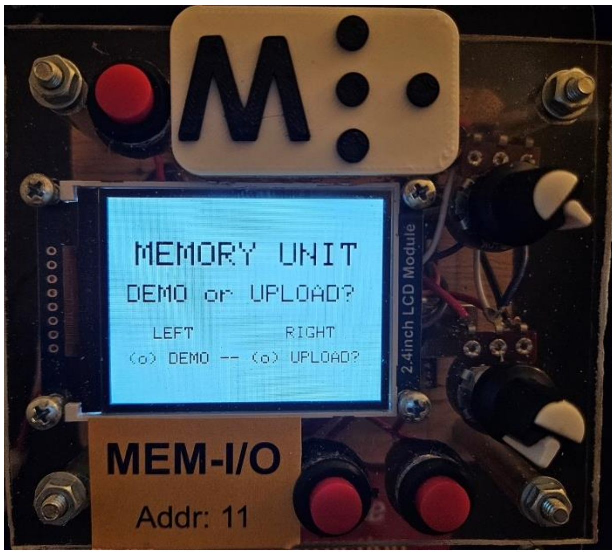


Fig. 4.6 MEM, I/O unit

SOURCE-CODE:

```
/*********************************************
MEMORY & OUTPUT UNIT
*********************************************
HOMS version 1B
```

```
Hardware-Oriented Microprocessor
Simulator

(C) Panayotis (Panos) Papazoglou
https://homs.panospapazoglou.gr/

LICENSE:
Creative Commons
CC BY NC SA
International License
********************************************/

#define ILI9341_BLACK 0x0000
#define ILI9341_DARKGREEN 0x03E0
#define ILI9341_GREEN 0x07E0
#define ILI9341_WHITE 0xFFFF

#include "SPI.h"
#include "Adafruit_GFX.h"
#include "Adafruit_ILI9341.h"

//TFT pins
#define TFT_CLK 13
#define TFT_MISO 12
#define TFT_MOSI 11
#define TFT_DC 7
#define TFT_CS 10
#define TFT_RST 8

Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC, TFT_MOSI, TFT_CLK,
TFT_RST, TFT_MISO);

//Include I2C library
#include <Wire.h>
#define FREQ 10000

//Set symbolic names for button/LED pins
#define GREEN_button 4
#define RED_button 5


//A byte is read from memory when the LOAD command is received
#define LOAD 71
#define ADDRESS 11
#define IDON 99
```

```
#define IDOFF 98
#define IDOK 97


int bcolor = ILI9341_BLACK, fcolor = ILI9341_WHITE;

//Memory: 256 locations, starting address 0, ending address 255 (FF hex)
int mem[256];

//Serial received string
String rs = "";

//variables for serial communication
int i = 0;
int c = 0;
int r = 40;

//Received data from control unit
byte rV = 0;

//Previous received data from control unit
byte rVprev = 0;

//Flag that is activated when an address is received
byte ADDRflag = 0;

//Declare next array if you want to have a preloaded code in memory
byte prog[] = { 4, 3, 17, 0 }; //MOV A,4 ; HALT

//Variable for choosing demo mode or upload from PC
int demo = 0;


//Starting function (after reset or power on), runs once
void setup(void) {
  Serial.begin(9600);
  Serial.println("HELLO!");

  init_buttons();

  //Read POT for background color
  bcolor = analogRead(0);
  Serial.print("bcolor=");
  Serial.println(bcolor);
  if (bcolor < 50) bcolor = ILI9341_BLACK;
```

```
else if (bcolor > 975) bcolor = ILI9341_WHITE;
else
  bcolor = ILI9341_GREEN;

//Read POT for text color
fcolor = analogRead(1);
Serial.print("fcolor=");
Serial.println(fcolor);
if (fcolor < 50) fcolor = ILI9341_BLACK;
else if (fcolor > 975) fcolor = ILI9341_WHITE;
else
  fcolor = ILI9341_BLACK;

//initialize I2C communication (address 11)
Wire.begin(ADDRESS);
Wire.setClock(FREQ);

//Initialize TFT
tft.begin();
tft.setRotation(135);

tft_welcome();


//Enable ISR routines for receiving bytes and requests
Wire.onReceive(onReceive);
Wire.onRequest(onRequest);

//Initialize Serial communication
//for displaying debugging messages
Serial.begin(9600);
init_mem(256);
demo_or_upload();

if (demo) {
  set_prog();

  tft.fillScreen(bcolor);
  display(50, 30, 4, 0, "D E M O");
  display(40, 70, 3, 0, "Instruction:");
  display(50, 100, 3, 0, "MOV A,3");
  display(20, 160, 3, 0, "Use CU to start");


  while (1);
  }

  tft.fillScreen(bcolor);
  display(50, 100, 3, ILI9341_WHITE, "Waiting for");
  display(70, 150, 3, ILI9341_WHITE, "Upload...");
}


/********************************************
Main function (always active)
********************************************/
void loop(void) {

  //Section added for serial communication with the PC
  if (Serial.available() > 0) {
    rs = Serial.readStringUntil('\n');
    if (rs == "new") {
      tft.fillScreen(bcolor);

      i = 0;
      c = 0;
      display(15, 0, 3, ILI9341_BLUE, "Prog. bytes:");
      while (rs != "end") {
        if (Serial.available() > 0) {
          rs = Serial.readStringUntil('\n');
          mem[i] = rs.toInt();
          if (c > 300) {
            r += 40;
            c = 0;
          }

          if ((mem[i] == 0) && (i % 4 == 0)) {
            display(c, r, 2, ILI9341_WHITE, rs);
            c += 30;
          }
          if (mem[i] > 0) {
            display(c, r, 2, ILI9341_WHITE, rs);
            c += 30;
          }
          if (mem[i] == 17) rs = "end";
          i++;
        }
      }
      display(15, 100, 3, ILI9341_GREEN, "Upload complete");
      display(20, 150, 3, ILI9341_RED, "Use CU to start");
      rs = "stop";
    }

    else if (rs = "ready") {
      Serial.println("Sending from arduino...");
      for (int p = 0; p < i; p++) {
        //Serial.println("Hello from Arduino"); rs="stop";
        Serial.println(mem[p]);
      }
    }
  }
}





/********************************************
initialize memory with size of locations

locations: total array locations
********************************************/
void init_mem(int locations) {
  for (int i = 0; i < locations; i++)
    mem[i] = 0;
}

/********************************************
When data are received from control unit
********************************************/
void onReceive(int a)

{
  rV = Wire.read();
  Serial.print("rV:");
  Serial.println(rV);

  //If previous command is LOAD, then the current byte represents an address
  if (rVprev == LOAD) ADDRflag = 1;
  rVprev = rV;
}

/********************************************
When data are requested from control unit
********************************************/
```

```
void onRequest() {
  //If the contents of an address are requested
  if (ADDRflag == 1) {
    Serial.println("ADDRflag=1");
    Serial.print("rV:");
    Serial.print(rV);
    Serial.print(", content:");
    Serial.println(mem[rV]);
    Wire.write(mem[rV]);
    ADDRflag = 0;
    rVprev = 0;
  }
  if (rV == IDON) Wire.write(IDOK);
}

/********************************************
Preload program bytes in memory.
Update mem[] array from prog[] array
********************************************/
void set_prog() {
  for (int i = 0; i < 4; i++) mem[i] = prog[i];
}

/********************************************
Unit title
********************************************/
void tft_welcome() {
  tft.fillScreen(bcolor);
  display(30, 50, 4, ILI9341_GREEN, "MEMORY UNIT");
}

/********************************************
Display text on TFT
on LCD screen

col: column
row: row
txsize: text size
txcolor: not used here (optional for debugging purposes)
val: text to be displayed
********************************************/
void display(int col, int row, int txsize, int txcolor, String val) {

  tft.setCursor(col, row);
  tft.setTextSize(txsize);
  tft.setTextColor(fcolor);
  tft.print(val);
}

/********************************************
User selects Demo or Upload from PC
********************************************/
void demo_or_upload() {

  display(20, 100, 3, ILI9341_WHITE, "DEMO or UPLOAD?");
  display(50, 150, 2, ILI9341_YELLOW, "LEFT         RIGHT");
  display(20, 180, 2, ILI9341_GREEN, "(o) DEMO -- (o) UPLOAD?");

  int Gb = HIGH;
  int Rb = HIGH;
  while ((Gb == HIGH) && (Rb == HIGH)) {
    Gb = digitalRead(GREEN_button);
    Rb = digitalRead(RED_button);
  }
  if (Gb == LOW) demo = 1;
  else
    demo = 0;
}

/********************************************
Wait for user input
********************************************/
void next(String txt) {
  display(0, 0, 4, ILI9341_WHITE, "(o) Next (o)--");
  while (digitalRead(GREEN_button) == HIGH) { ; }
}

/********************************************
Initialize PULL-UP resistors
Every button is activated when
a PULL-UP pin goes to LOW (GND)
********************************************/
void init_buttons() {
  pinMode(GREEN_button, INPUT_PULLUP);
  pinMode(RED_button, INPUT_PULLUP);
}
```

Important note: ***The GREEN_button and RED_button pins, represent the Left and Right buttons in current block respectively (the GREEN and RED buttons were used in HOMS version 1).***

4.4 Control Unit (CU)

The control unit constitutes the most important component of the HOMS system. This unit fetches instruction data from the memory unit, decodes each instruction (type and parameters) and finally sends commands to other components for supporting the execution cycle.

In other words, synchronizes the HOMS components operation for supporting a fully working system regarding the instruction execution. The required steps for each instruction execution are implemented within the CU.

Important note: ***For creating new assembly instructions, new source code has to be added inside the CU.***

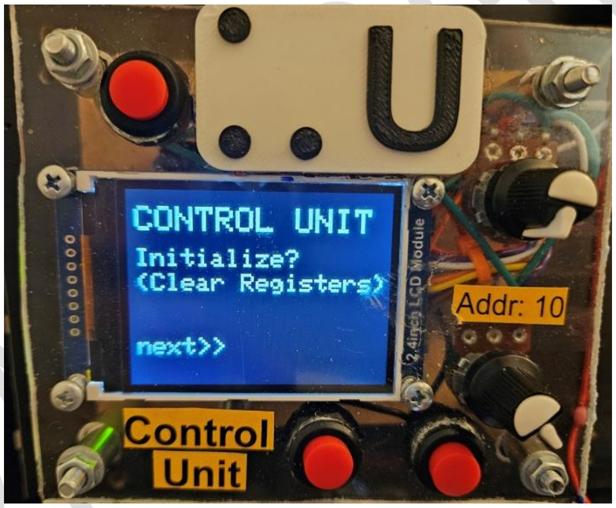
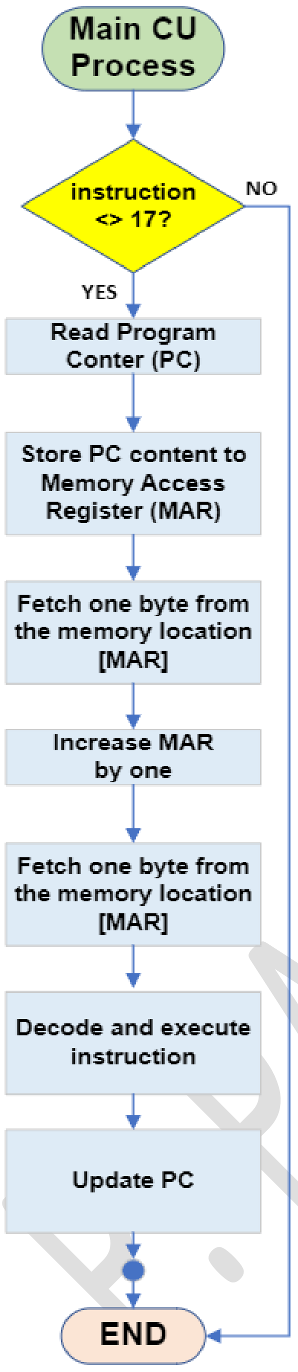


Fig. 4.7 CU unit



Main CU Process
instruction <> 17?
NO
YES
Read Program Conter (PC)
Store PC content to Memory Access Register (MAR)
Fetch one byte from the memory location [MAR]
Increase MAR by one
Fetch one byte from the memory location [MAR]
Decode and execute instruction
Update PC
END

Fig. 4.8 Execution process

The flow chart (fig 4.8) shows the whole procedure for (a) fetching instruction data from memory and (b) executing instruction through the control unit. This procedure can be described in steps as follows:

Step 1: Read PC (Program counter) for finding where the next instruction address is

Step 2: Store this address to MAR (Memory Access Register) in order to place the desired memory address on address bus

Step 3: Fetch instruction code via data bus and store this code in MBR (Memory Buffer register) and increase MAR by one

Step 4: Fetch parameter code via data bus and store this code in MBR (Memory Buffer register)

Step 5: Decode instruction data and execute, a) copy the contents of registers A and B to ALU variables T1 and T2 respectively, b) perform the addition T1+T2 and update SR (Status Register), c) store result to register A

Step 6: Update PC (Program Counter) for fetching next instruction code from memory

SOURCE-CODE:

```
/*******************************************
CONTROL UNIT
I2C MASTER
*******************************************
HOMS version 1B
Hardware-Oriented Microprocessor
Simulator

(C) Panayotis (Panos) Papazoglou
https://homs.panospapazoglou.gr/

LICENSE:
Creative Commons
CC BY NC SA
International License
*******************************************/
//Please ignore any compilation warnings
//due to function overload (the same function name
//can be used but with different parameters).
//the MCU executes the function which matches
//with parameters

//Define symbolic names for sending commands to other blocks
//The I2C communication is based on one byte transmit/receive
#define READ 65
#define RESET 66
#define SHIFT_L 67
#define SHIFT_R 68
#define DEC 69
#define INC 70
#define LOAD 71
#define ADD 72
#define IDON 99
```

```
#define IDOFF 98
#define IDOK 97

//Set symbolic names for button/LED pins
#define GREEN_button 4
#define RED_button 5
//#define GREEN_LED 10
//#define RED_LED 9

//Include I2C library
//The I2C function are called using the prefix Wire.
#include <Wire.h>

#define FREQ 10000
//new tft section

#define ILI9341_BLACK 0x0000
#define ILI9341_RED 0xF800
#define ILI9341_WHITE 0xFFFF


#include "SPI.h"
#include "Adafruit_GFX.h"
#include "Adafruit_ILI9341.h"

//TFT pins
#define TFT_CLK 13
#define TFT_MISO 12
#define TFT_MOSI 11
#define TFT_DC 7
#define TFT_CS 10
#define TFT_RST 8

Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC, TFT_MOSI,
TFT_CLK, TFT_RST, TFT_MISO);

//Declare variables for the needs of the CU unit (locally)
byte RA = 0;
byte RB = 0;
byte RC = 0;
byte PC = 0;
byte MAR = 0;
byte MBR = 0;
byte T1 = 0;
byte T2 = 0;
byte RS = 0;
byte command = 0;
byte param = 0;
String hexval = "";
int blockADDR;

//Delay between execution process steps
int d = 3000;

int bcolor = ILI9341_BLACK, fcolor = ILI9341_WHITE;

//Starting function (after reset or power on), runs once
void setup() {
  //Initialize Serial communication
  //for displaying debugging messages
  Serial.begin(9600);

  init_buttons();
  tft.begin();
  tft.setRotation(135);

  //Read POT for background color
  bcolor = analogRead(0);
  Serial.print("bcolor=");
  Serial.println(bcolor);
  if (bcolor < 50) bcolor = ILI9341_BLACK;
  else if (bcolor > 975) bcolor = ILI9341_WHITE;
  else
    bcolor = ILI9341_RED;

  //Read POT for text color
  fcolor = analogRead(1);
  Serial.print("fcolor=");
  Serial.println(fcolor);
  if (fcolor < 50) fcolor = ILI9341_BLACK;
  else if (fcolor > 975) fcolor = ILI9341_WHITE;
  else
    fcolor = ILI9341_WHITE;

  //Initialize I2C communication
  Wire.begin();
  //Wire.setClock(FREQ);

  //Initialize buttons & LED-buttons
  init_buttons();

  Serial.println("init ok");

  //Display first message and read user input
  run();

  //Initial value for entering the while loop
  command = 0;

  //Clear registers (content=00)
  //Wait for RED button to start execution
  INIT();

  /****************************
  MAIN EXECUTION PROCESS
  *****************************/
  while (command != 17) {
    READ_PC();      //Read PC content for instruction starting
address
    PC2MAR();       //Update MAR for accessibg memory address
    FETCH1();       //Fetch 1st byte from memory (instruction code)
    UPDATE_MAR();   //Update MAR for fetching 2nd byte
    FETCH2();       //Fetch 2nd byte (instruction parameter)
    DECODE_EXEC();  //Decode instruction and execute
    UPDATE_PC();    //Update PC content for next instruction

    next("Next >>");
  }

  //Out of while loop. The following code is executed when the
instruction code is 17
  //which corresponds to STOP instruction
  tft.fillScreen(bcolor);
  display(50, 100, 3, 0, "END OF PROGRAM");
  display(50, 150, 3, 0, "EXECUTION");

}  //End of SETUP


/*********************************************
READ PC (Program Counter), PC block address=4
*********************************************/

void READ_PC() {
  //Display message on LCD
  tft.fillScreen(bcolor);
  display(50, 30, 4, 0, "READING");
  display(40, 70, 3, 0, "Prog. Counter");
  display(40, 110, 3, 0, "(PC Register)");

  next("next>>");

  //Read PC content. Store content in PC variable
  PC = READ_REG(4);

  //Display READ results on LCD
  tft.fillScreen(bcolor);
  display(10, 10, 4, 0, "Prog.Counter");
  display(10, 60, 3, 0, "PC10=");
  display(100, 60, 3, 0, String(PC));
  display(10, 90, 3, 0, "PC16=");
  hexval = String(PC, HEX);
  display(100, 90, 3, 0, hexval);

  //Wait before next step
  next("next>>");
```

```
}  //End of READ_PC

/********************************************
Store PC contents to register MAR
for accessing memory address
MAR=PC (MAR block address=5)
*********************************************/
void PC2MAR() {
  //Display message on LCD
  tft.fillScreen(bcolor);
  display(20, 10, 4, 0, "MAR <-- PC");
  display(0, 60, 3, 0, "MAR=Memory Access Register");
  display(0, 125, 3, 0, "PC=Prog. Counter");
  next("next>>");

  //LOAD PC content in MAR register
  WRITE_REG(5, PC);

  //Update variable
  MAR = PC;

  //Display message on LCD
  tft.fillScreen(bcolor);
  display(10, 10, 4, 0, "Check MAR");
  display(0, 60, 3, 0, "MAR=Memory Access Register");
  next("next>>");
}  //End of PC2MAR

/********************************************
FETCH 1st byte from memory, mem[MAR]
Instruction code
*********************************************/
void FETCH1() {
  //Display message on LCD
  tft.fillScreen(bcolor);
  display(10, 10, 4, 0, "FETCH");
  display(140, 10, 3, 0, "byte 1>");
  display(10, 50, 3, 0, "Reading Instruction code");
  display(10, 110, 3, 0, "from mem[");
  display(175, 110, 3, 0, String(MAR));
  display(220, 110, 3, 0, "]");
  next("next>>");

  //LOAD MAR content in MEM/OUT Unit
  //for accessing the corresponding address
  WRITE_REG(11, MAR);

  //Receive memory content
  //Store content in MBR variable
  blockADDR = 11;
  Wire.requestFrom(blockADDR, 1);
  if (Wire.available())
    MBR = Wire.read();

  //Store instruction code in command variable
  command = MBR;

  //Display information (MAR, MBR) on LCD
  tft.fillScreen(bcolor);
  display(10, 10, 3, 0, "MBR<--mem[");
  display(190, 10, 3, 0, String(MAR));
  display(230, 10, 3, 0, "]");
  display(10, 40, 3, 0, "(10)=");
  display(100, 40, 3, 0, String(MBR));
  display(10, 70, 3, 0, "(16)=");
  hexval = String(MBR, HEX);
  display(100, 70, 3, 0, hexval);
  next("next>>");

  //Update MBR register from MBR variable
  WRITE_REG(6, MBR);

  //Display message on LCD
  tft.fillScreen(bcolor);
  display(10, 10, 4, 0, "Check MBR");
  next("next>>");
  tft.fillScreen(bcolor);
```

```
  display(10, 10, 4, 0, "command is");
  display(120, 50, 4, 0, String(command));
  next("next>>");
}  //End of FETCH1


/********************************************
Updating MAR for accessing
MAR=PC+1 (MAR address=5)
********************************************/
void UPDATE_MAR() {
  //Display message on LCD
  tft.fillScreen(bcolor);
  display(10, 10, 4, 0, "Updating");
  display(10, 60, 4, 0, "MAR...");

  //Update MAR register for fetching the next byte from memory
  //(from next address)
  WRITE_REG(5, PC + 1);
  next("next>>");

  //Update variable
  MAR = PC + 1;

  //Display message on LCD
  tft.fillScreen(bcolor);
  display(10, 10, 4, 0, "Check MAR");
  next("next>>");
}


/********************************************
FETCH 2nd byte from memory, mem[MAR]
Parameter code
********************************************/
void FETCH2() {
  //Display message on LCD
  tft.fillScreen(bcolor);
  display(10, 10, 4, 0, "FETCH");
  display(140, 10, 3, 0, "byte 2>");

  //LOAD MAR content in MEM/OUT Unit
  //for accessing the corresponding address
  WRITE_REG(11, MAR);
  next("next>>");


  //Receive memory content
  //Store content in MBR variable
  blockADDR = 11;
  Wire.requestFrom(blockADDR, 1);
  if (Wire.available())
    MBR = Wire.read();

  //Store parameter code in param variable
  param = MBR;

  //Display information (MAR, MBR) on LCD
  tft.fillScreen(bcolor);
  display(10, 10, 3, 0, "MBR<--mem[");
  display(190, 10, 3, 0, String(MAR));
  display(230, 10, 3, 0, "]");
  display(10, 40, 3, 0, "(10)=");
  display(100, 40, 3, 0, String(MBR));
  display(10, 70, 3, 0, "(16)=");
  hexval = String(MBR, HEX);
  display(100, 70, 3, 0, hexval);
  next("next>>");

  //update register MBR
  WRITE_REG(6, MBR);

  //Display messages on LCD
  tft.fillScreen(bcolor);
  display(10, 10, 3, 0, "Check MBR");
  next("next>>");
  tft.fillScreen(bcolor);
  display(10, 10, 3, 0, "param is");
```

```
  display(180, 10, 3, 0, String(param));
  next("next>>");
}  //END of FETCH2


/*******************************************
Decoding instruction and execution
Instruction block: [command][param]

command=instruction code
param=parameter code
********************************************/
//command,param
void DECODE_EXEC() {

/****************************
MOV A,i (A=i)
*****************************/
  if (command == 4) {
    //Display messages on LCD
    tft.fillScreen(bcolor);
    display(10, 10, 3, 0, "CMD:MOV A,");
    display(190, 10, 3, 0, String(param));
    display(10, 40, 3, 0, "Executing...");
    next("next>>");

    //LOAD param (num) in Register A
    WRITE_REG(1, param);
  }


/****************************
MOV B,i (B=i)
*****************************/
  if (command == 3) {
    //Display messages on LCD
    tft.fillScreen(bcolor);
    display(10, 10, 3, 0, "CMD:MOV B,");
    display(100, 10, 3, 0, String(param));
    display(10, 40, 3, 0, "Executing...");
    next("next>>");

    //LOAD param (num) in Register A
    WRITE_REG(2, param);
  }


/****************************
INC A (A=A+1)
*****************************/
  if (command == 10) {
    //Display messages on LCD
    tft.fillScreen(bcolor);
    display(10, 10, 3, 0, "CMD:INC A");
    display(10, 40, 3, 0, "Executing...");
    next("next>>");

    //Send INC command to register A
    blockADDR = 1;
    Wire.beginTransmission(blockADDR);
    Wire.write(INC);
    Wire.endTransmission();

    //Display messages on LCD
    tft.fillScreen(bcolor);
    display(10, 10, 3, 0, "Check REG-A");
  }

/****************************
INC B (B=B+1)
*****************************/
  if (command == 5) {
    //Display messages on LCD
    tft.fillScreen(bcolor);
    display(10, 10, 3, 0, "CMD:INC B");
    display(10, 40, 3, 0, "Executing...");
    next("next>>");
    //Send INC command to register B
    blockADDR = 2;
    Wire.beginTransmission(blockADDR);
    Wire.write(INC);
    Wire.endTransmission();

    //Display messages on LCD
    tft.fillScreen(bcolor);
    display(10, 10, 3, 0, "Check REG-B");
  }

/****************************
DEC A (A=A-1)
*****************************/
  if (command == 6) {
    //Display messages on LCD
    tft.fillScreen(bcolor);
    display(10, 10, 3, 0, "CMD:DEC A");
    display(10, 40, 3, 0, "Executing...");
    next("next>>");

    //Send DEC command to register A
    blockADDR = 1;
    Wire.beginTransmission(blockADDR);
    Wire.write(DEC);
    Wire.endTransmission();

    //Display messages on LCD
    tft.fillScreen(bcolor);
    display(10, 10, 3, 0, "Check REG-A");
  }


/****************************
DEC B (B=B-1)
*****************************/
  if (command == 7) {
    //Display messages on LCD
    tft.fillScreen(bcolor);
    display(10, 10, 3, 0, "CMD:DEC B");
    display(10, 40, 3, 0, "Executing...");
    next("next>>");

    //Send DEC command to register B
    blockADDR = 2;
    Wire.beginTransmission(blockADDR);
    Wire.write(DEC);
    Wire.endTransmission();

    //Display messages on LCD
    tft.fillScreen(bcolor);
    display(10, 10, 3, 0, "Check REG-B");
  }

/*****************************
INC B (B=B+1)
*****************************/
  if (command == 5) {
    //Display messages on LCD
    tft.fillScreen(bcolor);
    display(10, 10, 3, 0, "CMD:INC B");
    display(10, 40, 3, 0, "Executing...");
    next("next>>");

    //Send INC command to register B
    blockADDR = 2;
    Wire.beginTransmission(blockADDR);
    Wire.write(INC);
    Wire.endTransmission();

    //Display messages on LCD
    tft.fillScreen(bcolor);
    display(10, 10, 3, 0, "Check REG-B");
  }
/*****************************
MOV B,A (B=A)
*****************************/
  if (command == 99)  //DISABLED INSTRUCTION
  {
    //Display messages on LCD
    tft.fillScreen(bcolor);
    display(10, 10, 3, 0, "CMD:MOV B,A");
    display(10, 10, 3, 0, "Executing...");
    next("next>>");

    //Read from register A
    //Store content to variable RA
    RA = READ_REG(1);
    Serial.print("Read=RA:");
    Serial.println(RA);

    //Store register A content (RA) in register B
    WRITE_REG(2, RA);

    //Display message on LCD
    tft.fillScreen(bcolor);
    display(10, 10, 3, 0, "Check REG-B");
  }

/*****************************
ADD A,B
*****************************/
  if (command == 1) {
    //Display message on LCD
    tft.fillScreen(bcolor);
    display(10, 10, 3, 0, "CMD:ADD A,B");
    next("next>>");

    //READ from register A, Store in variable RA
    RA = READ_REG(1);

    //READ from register B, Store in variable RB
    RB = READ_REG(2);

    //Store register A content in register T1
    WRITE_REG(7, RA);
    //Store register B content in register T2
    WRITE_REG(8, RB);

    //Display messages on LCD
    tft.fillScreen(bcolor);
    display(10, 10, 3, 0, "Check T1,T2");
    display(10, 40, 3, 0, "ALUEXEC...");
    next("next>>");

    //Send ADD instruction to ALU/SR Unit
    byte RES = SEND_ALU(9, ADD);

    //Store ALU result in register A
    WRITE_REG(1, RES);

    //Display message on LCD
    tft.fillScreen(bcolor);
    display(10, 10, 3, 0, "Check RA,ALU,SR");
    next("next>>");
  }


}  //End of DECODE_EXEC

/*******************************************
Update PC register content
for fetching next instruction bytes
from memory
*******************************************/
void UPDATE_PC() {

  //Send two INC commands to PC register
  Wire.beginTransmission(4);
  Wire.write(INC);
  Wire.endTransmission();
  delay(50);
  Wire.beginTransmission(4);
  Wire.write(INC);
  Wire.endTransmission();
  delay(50);

  //Update variable
  PC = PC + 2;
}

void loop() {
  //Nothing here
}



/*******************************************
Initialize PULL-UP resistors
Every button is activated when
a PULL-UP pin goes to LOW (GND)
********************************************/
void init_buttons() {
  pinMode(GREEN_button, INPUT_PULLUP);
  pinMode(RED_button, INPUT_PULLUP);
}



/*******************************************
Clear all registers (set content=0)

Register address = 1 = Register A
Register address = 2 = Register B
Register address = 4 = Register PC
Register address = 5 = Register MAR
Register address = 6 = Register MBR
********************************************/
void clear_REGS() {
  Serial.println("Clearing registers...");
  int validADDR[] = { 1, 2, 4, 5, 6, 7 };
  for (int i = 0; i < 6; i++) {
    Wire.beginTransmission(validADDR[i]);
    Wire.write(RESET);
    Wire.endTransmission();
    delay(150);
  }
  Serial.println("OK done!");
}

/******************************************
Display messages on TFT
and call clear register function
******************************************/
void INIT() {
  tft.fillScreen(bcolor);
  display(10, 10, 3, 0, "RESET REGISTERS");

  clear_REGS();
  display(10, 50, 3, 0, "Done!");
  display(10, 80, 3, 0, "Program loaded?");
  display(10, 110, 3, 0, "Start EXECUTION?");
  next("next>>");
}

/******************************************
Read register content
Send READ command
Register address = addr
Return content
******************************************/
byte READ_REG(int addr) {
  byte ans;

  Wire.beginTransmission(addr);
  Wire.write(READ);
  Wire.endTransmission();
  delay(200);
  //Receive answer
  Wire.requestFrom(addr, 1);
  if (Wire.available())
    ans = Wire.read();
  return ans;
}

/********************************************
Send command to ALU/SR unit
Address = addr
command to ALU/SR = instruction
Return ALU result
********************************************/
byte SEND_ALU(int addr, int instruction) {
  byte ans;

  Wire.beginTransmission(addr);
  Wire.write(instruction);
  Wire.endTransmission();
  delay(300);
  //Receive answer (PC content)

  Wire.requestFrom(addr, 1);
  if (Wire.available())
    ans = Wire.read();

  return ans;
}


/********************************************
Load a number to a specific register
Register address = addr
Number = data
********************************************/
void WRITE_REG(byte addr, byte data) {

  Wire.beginTransmission(addr);
  Wire.write(LOAD);
  Wire.endTransmission();
  delay(200);
  Wire.beginTransmission(addr);
  Wire.write(data);
  Wire.endTransmission();
}


/*******************************************
Wait for a button press (LEFT button)
********************************************/
void next(String txt) {

  display(10, 180, 3, 0, txt);
  while (digitalRead(GREEN_button) == HIGH) { ; }
}

/*******************************************
Display first message and wait for user input
********************************************/
void run() {
  Serial.print("inside run");
  tft.fillScreen(bcolor);
  display(10, 20, 4, 0, "CONTROL UNIT");
  display(10, 70, 3, 0, "Initialize?");
  display(10, 100, 3, 0, "(Clear Registers)");


  next("next>>");
}

/*******************************************
Display text on TFT
on LCD screen

col: column
row: row
txsize: text size
txcolor: not used here (optional for debugging purposes)
val: text to be displayed
********************************************/
```

```
void display(int col, int row, int txsize, int txcolor, String val) {

  tft.setCursor(col, row);
  tft.setTextSize(txsize);
  tft.setTextColor(fcolor);
  tft.print(val);
}
```

4.5 RGB LED-Strip control

At the edges of the suitcase, an RGB LED-Strip has been installed for offering supplementary lighting (fig. 4.9). The LED-Strip is controlled by an Arduino which is also installed in the suitcase as well as an additional DC power supply of 5V/3A.

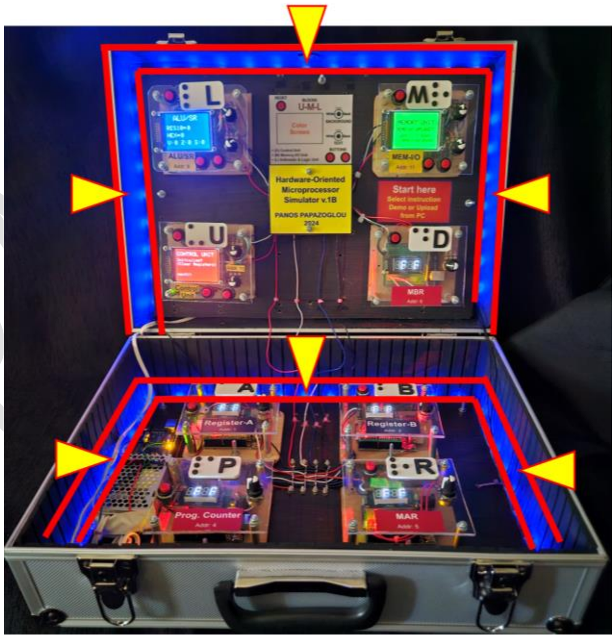


Fig. 4.9 RGB LED-Strip frame

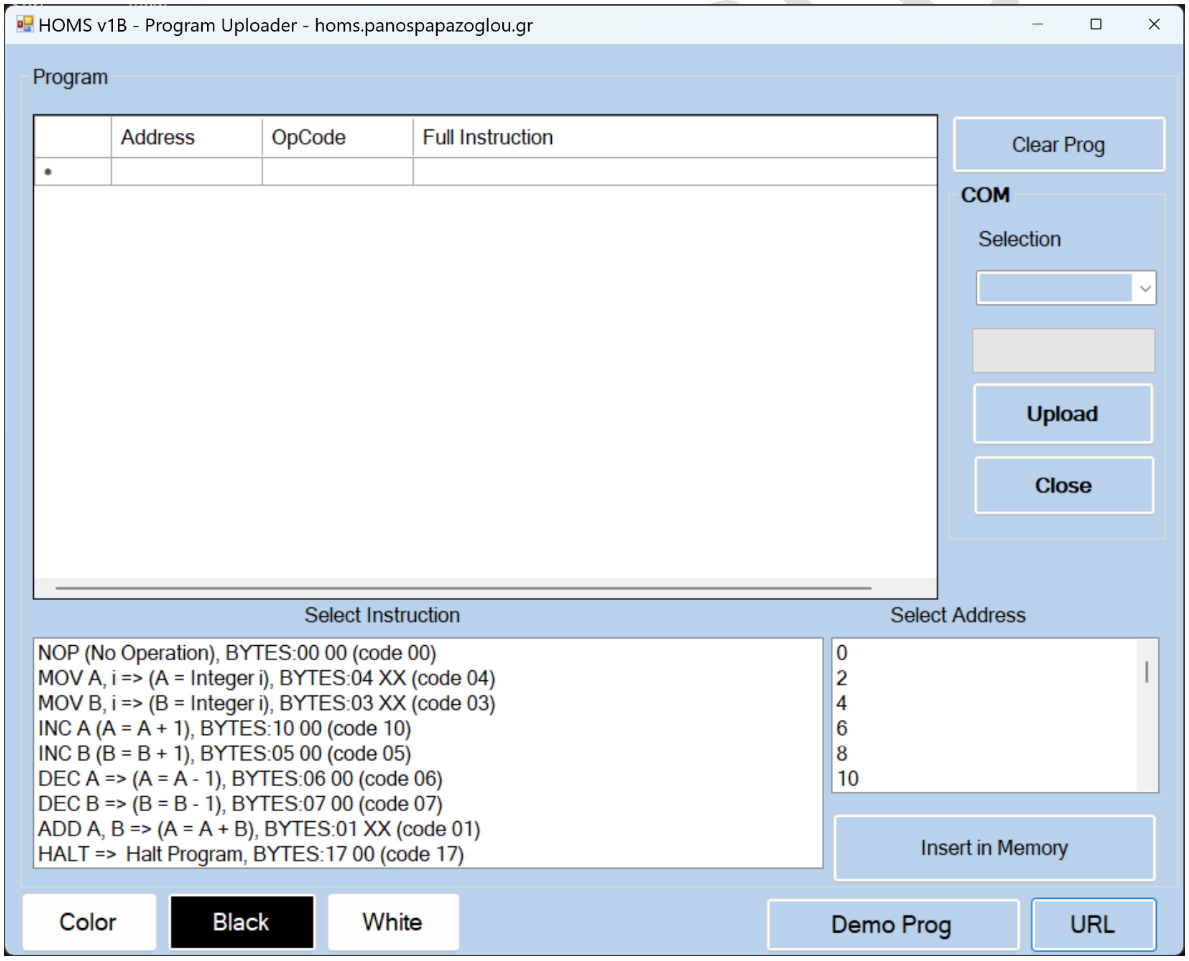
SOURCE-CODE:

```
/*******************************************
RGB LED-STRIP FRAME CONTROL
*******************************************
HOMS version 1B
Hardware-Oriented Microprocessor
Simulator

(C) Panayotis (Panos) Papazoglou
https://homs.panospapazoglou.gr/

LICENSE:
Creative Commons
CC BY NC SA
International License
*******************************************/
#include <FastLED.h>
#include <Adafruit_NeoPixel.h>

//Set control Pin
#define LED_PIN 8

//Set number of LEDs
#define NUM_LEDS 57

CRGB leds[NUM_LEDS];

void setup() {
  //Initialize LEDs
  FastLED.addLeds<WS2812, LED_PIN, GRB>(leds, NUM_LEDS);

  //Set RED color only with low intensity
  int r = 000, g = 0, b = 50;

  //Set the above color intensities to all LEDs
  for (int i = 0; i <= 56; i++) {
    leds[i] = CRGB(r, g, b);

    //Light-up LEDs
    FastLED.show();
  }
}
```

```
void loop() {

}
```

4.6 GUI PC application

As mentioned before, a GUI environment has been implemented for inserting and uploading programs to memory unit. This application is developed in Visual Basic within the free Microsoft Visual Studio Community environment. The developed GUI consists of two main parts: (a) application form, where interactive visual objects are embedded and (b) visual basic code for implementing the object functionality. Figure 4.10 shows the developed GUI environment.



NOP (No Operation), BYTES:00 00 (code 00)
MOV A, i => (A = Integer i), BYTES:04 XX (code 04)
MOV B, i => (B = Integer i), BYTES:03 XX (code 03)
INC A (A = A + 1), BYTES:10 00 (code 10)
INC B (B = B + 1), BYTES:05 00 (code 05)
DEC A => (A = A - 1), BYTES:06 00 (code 06)
DEC B => (B = B - 1), BYTES:07 00 (code 07)
ADD A, B => (A = A + B), BYTES:01 XX (code 01)
HALT => Halt Program, BYTES:17 00 (code 17)

0
2
4
6
8
10

Fig. 4.10 The GUI environment (developed in Visual Basic)

SOURCE-CODE:

```
'********************************************
'GUI for developing And uploading
'program to physical memory unit
'a Visual Basic application
'*********************************************
'HOMS version 1B
'Hardware-Oriented Microprocessor
'Simulator
'
'(C) Panayotis (Panos) Papazoglou
'https://homs.panospapazoglou.gr/
'
'LICENSE:
'Creative Commons
'CC BY NC SA
'International License
'*********************************************/

Imports System.Windows.Forms.VisualStyles.VisualStyleElement.ProgressBar
Imports System
Imports System.IO.Ports
'Imports System.Reflection.Emit
'Imports System.Resources
'Imports System.Diagnostics.Eventing.Reader
'Imports System.Threading
'Imports System.Drawing

Public Class Form1


    '*******************
    '*** GLOBAL VARS ***
    '*******************
    Dim comPORT As String
    Dim count As Integer = 0
    Dim url As String = "https://homs.panospapazoglou.gr"
    Private fn(20) As String
    Private iset(20) As String
    Private code(20) As Integer
    Private mem(100) As Integer
    Private opcode = 0, addr = 0, procounter = 0, a = 0, b = 0, TX = 0, RX = 0
    '****************************
    '*** AFTER APP FORM LOAD ***
    '****************************
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load

        iset(0) = "NOP (No Operation), BYTES:00 00 (code 00)"
        iset(1) = "MOV A, i => (A = Integer i), BYTES:04 XX (code 04)"
        iset(2) = "MOV B, i => (B = Integer i), BYTES:03 XX (code 03)"
        iset(3) = "INC A (A = A + 1), BYTES:10 00 (code 10)"
        iset(4) = "INC B (B = B + 1), BYTES:05 00 (code 05)"
        iset(5) = "DEC A => (A = A - 1), BYTES:06 00 (code 06)"
        iset(6) = "DEC B => (B = B - 1), BYTES:07 00 (code 07)"
        iset(7) = "ADD A, B => (A = A + B), BYTES:01 XX (code 01)"
        iset(8) = "HALT =>  Halt Program, BYTES:17 00 (code 17)"

        code(0) = 0      'NOP
        code(1) = 4      'MOV A,i
        code(2) = 3      'MOV B,i
        code(3) = 10     'INC A
        code(4) = 5      'INC B
        code(5) = 6      'DEC A
        code(6) = 7      'DEC B
        code(7) = 1      'ADD A,B
        code(8) = 17     'HALT

        fn(0) = "NOP (No Operation), BYTES:00 00 (code 00)"
        fn(4) = "MOV A, i => (A = Integer i), BYTES:04 XX (code 04)"
        fn(3) = "MOV B, i => (B = Integer i), BYTES:03 XX (code 03)"
        fn(10) = "INC A (A = A + 1), BYTES:10 00 (code 10)"
        fn(5) = "INC B (B = B + 1), BYTES:05 00 (code 05)"
        fn(6) = "DEC A => (A = A - 1), BYTES:06 00 (code 06)"
        fn(7) = "DEC B => (B = B - 1), BYTES:07 00 00 (code 07)"
        fn(1) = "ADD A, B => (A = A + B), BYTES:01 XX (code 01)"
        fn(17) = "HALT =>   Halt Program, BYTES:17 00 (code 17)"

        Setcolors(1)
        display_instructions_listbox1()

        For i = 0 To 100 Step 2
            mem_address_list.Items.Add(i)
        Next
        Timer1.Enabled = False
        comPORT = ""
        For Each sp As String In My.Computer.Ports.SerialPortNames
            ComboBox1.Items.Add(sp)
        Next
    End Sub


    '*******************************
    'Set colors for all form objects
    '*******************************

    Private Sub Setcolors(mode As Integer)
        Dim bcolor As Color = Color.FromArgb(185, 209, 234)
        Dim fcolor As Color = Color.FromArgb(0, 0, 0)
        Dim wcolor As Color = Color.FromArgb(255, 255, 255)

        'Set default colors
        If (mode = 1) Then
            bcolor = Color.FromArgb(185, 209, 234)
            fcolor = Color.FromArgb(0, 0, 0)
            'overwrites back color for boxes within the app form
            wcolor = Color.FromArgb(255, 255, 255)
        End If

        'Set colors for high contrast theme (black on white)
        If (mode = 2) Then
            bcolor = Color.FromArgb(255, 255, 255)
            fcolor = Color.FromArgb(0, 0, 0)
            wcolor = bcolor
        End If

        'Set colors for high contrast theme (white on black)
        If (mode = 3) Then
            bcolor = Color.FromArgb(0, 0, 0)
            fcolor = Color.FromArgb(255, 255, 255)
            wcolor = bcolor
        End If

        'set the same back/fore colors for every form object
        clear_mem_button.BackColor = bcolor
        clear_mem_button.ForeColor = fcolor

        uploadbutton.BackColor = bcolor
        uploadbutton.ForeColor = fcolor
        stopbutton.BackColor = bcolor
        stopbutton.ForeColor = fcolor

        demo_button.BackColor = bcolor
        demo_button.ForeColor = fcolor

        goto_url.BackColor = bcolor
        goto_url.ForeColor = fcolor

        Insert_memory_button.BackColor = bcolor
        Insert_memory_button.ForeColor = fcolor

        Me.BackColor = bcolor

        Instruction_box_list.BackColor = wcolor
        Instruction_box_list.ForeColor = fcolor

        mem_address_list.BackColor = wcolor
        mem_address_list.ForeColor = fcolor

        memory.BackColor = wcolor
        memory.ForeColor = fcolor

        GroupBox5.BackColor = bcolor
        GroupBox5.ForeColor = fcolor

        GroupBox3.BackColor = bcolor
        GroupBox3.ForeColor = fcolor

        ComboBox1.BackColor = bcolor
        ComboBox1.ForeColor = fcolor


        pbar1.BackColor = bcolor
        pbar1.ForeColor = fcolor


        'unlock datagridbox colors from default
        memory.EnableHeadersVisualStyles = False

        memory.BackgroundColor = wcolor
        memory.ForeColor = fcolor

        memory.DefaultCellStyle.BackColor = wcolor
        memory.DefaultCellStyle.ForeColor = fcolor

        memory.ColumnHeadersDefaultCellStyle.BackColor = wcolor
        memory.ColumnHeadersDefaultCellStyle.ForeColor = fcolor

        memory.DefaultCellStyle.ForeColor = fcolor

        memory.RowHeadersDefaultCellStyle.BackColor = wcolor
        memory.RowHeadersDefaultCellStyle.ForeColor = fcolor

    End Sub

    '*******************************
    'Select COM port from list
    '*******************************
    Private Sub ComboBox1_SelectedIndexChanged(sender As Object, e As
EventArgs) Handles ComboBox1.SelectedIndexChanged
        If (ComboBox1.SelectedItem <> "") Then
            comPORT = ComboBox1.SelectedItem
        End If
    End Sub

    '*******************************
    'Clear Prog button operation
    '*******************************
    Private Sub Button3_Click(sender As Object, e As EventArgs) Handles
clear_mem_button.Click
        clear_memory()
    End Sub

    '******************************************
    'Clear Prog array/set each location to zero
    '******************************************
    Private Sub clear_memory()
        memory.Rows.Clear()
        For i = 0 To 100
            mem(i) = 0

        Next
        display_memory()
    End Sub

    '*******************************
    'Display available instructions
    '*******************************
    Private Sub display_instructions_listbox1()
        Instruction_box_list.Items.Clear()
        For i = 0 To 8
            Instruction_box_list.Items.Add(iset(i))
        Next
    End Sub

    '********************************************
    '*** Insert automatically a demo program ***
    '********************************************
    Private Sub demobutton_Click(sender As Object, e As EventArgs) Handles
demo_button.Click
        clear_memory()
        mem(0) = 4
        mem(1) = 4
        mem(2) = 10
        mem(3) = 0
        mem(4) = 3
        mem(5) = 7
        mem(6) = 7
        mem(7) = 0
        mem(8) = 17
        mem(9) = 0
        display_memory()
    End Sub

    '***************************
    '*** Open URL in browser ***
    '***************************
    Private Sub Button9_Click_1(sender As Object, e As EventArgs) Handles
goto_url.Click
        System.Diagnostics.Process.Start(url)
    End Sub

    '***********************************
    '*** Configure and open COM port ***
    '***********************************
    Private Sub open_port()
        If (comPORT <> "") Then
            Timer1.Enabled = False
            SerialPort1.Close()
            SerialPort1.PortName = comPORT
            SerialPort1.BaudRate = 9600
            SerialPort1.DataBits = 8
            SerialPort1.Parity = Parity.None
            SerialPort1.StopBits = StopBits.One
            SerialPort1.Handshake = Handshake.None
            SerialPort1.Encoding = System.Text.Encoding.Default
            SerialPort1.ReadTimeout = 10000
            Try
                SerialPort1.Open()
            Catch ex As System.UnauthorizedAccessException
                MsgBox("Access denied")
                Exit Sub
            End Try
        End If
    End Sub

    '***************************************
    '***    UPLOAD CODE TO ARDUINO      ***
    '*** enable transmitting in Timer 1 ***
    '***************************************
    Private Sub Button1_Click_1(sender As Object, e As EventArgs) Handles
uploadbutton.Click
        open_port()
        TX = 1 : RX = 0
        Timer1.Enabled = True


    End Sub


    '**************************************
    '*** button for default GUI colors ***
    '**************************************
    Private Sub Defbutton_Click(sender As Object, e As EventArgs) Handles
Defbutton.Click
        Setcolors(1)
    End Sub

    '**************************************************
    '*** button for GUI HC colors (black on white) ***
    '**************************************************
    Private Sub Whitebutton_Click(sender As Object, e As EventArgs) Handles
Whitebutton.Click
        Setcolors(2)
    End Sub

    '**************************************************
    '*** button for GUI HC colors (white on black) ***
    '**************************************************
    Private Sub Blackbutton_Click(sender As Object, e As EventArgs) Handles
Blackbutton.Click
        Setcolors(3)
    End Sub

    '**********************************************
    '*** EVENT TIMER for Serial communicatinon ***
    '**********************************************
    Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
Timer1.Tick
        If (TX = 1) Then

            SerialPort1.WriteLine("fakelineafterreset")
            Waitfake()
            SerialPort1.WriteLine("new")
            For i = 0 To 100
                pbar1.Value = i
                SerialPort1.WriteLine(mem(i).ToString())
                If (mem(i) <> 0) Then

                End If
            Next
            SerialPort1.WriteLine("end")

            TX = 0
        End If
    End Sub

    '**********************************************
    '***     Wait after Arduino auto reset     ***
    '**********************************************
    Private Sub Waitfake()
        For x = 1 To 10000000
            mem(100) = Math.Floor(Math.Sqrt(x / 100000) * Rnd())
        Next
    End Sub

    '********************************
    '*** Close com port and stop ***
    '********************************
    Private Sub stopbutton_Click(sender As Object, e As EventArgs) Handles
stopbutton.Click
        TX = 0
        SerialPort1.Close()
    End Sub

    '******************************
    '*** DISPLAY MEMORY CONTENTS ***
    '******************************
    Private Sub display_memory()

        'Clear RAM display area
        memory.Rows.Clear()

        For i = 0 To 100
            If (i Mod 2 = 0) Then
                memory.Rows.Add(i, mem(i), fn(mem(i)))
            Else
                memory.Rows.Add(i, mem(i), mem(i))
            End If
        Next
    End Sub


    '****************************
    '*** INSERT CODE IN MEMORY ***
    '****************************
    Private Sub InsertButton_Click(sender As Object, e As EventArgs) Handles
Insert_memory_button.Click

        Dim addrselect As Integer = mem_address_list.SelectedIndex * 2
        Dim insselect As Integer = Instruction_box_list.SelectedIndex

        If addrselect <> -1 Then
            addr = addrselect
        Else
            addr = 0

        End If

        If insselect <> -1 Then
            opcode = code(insselect)
        Else
            opcode = 0
        End If

        If addr < 0 Then
            addr = 0
        End If
        Dim intnum As Integer
        Select Case opcode
            Case 4
                intnum = read_int("MOV A,X //X=")
                mem(addr + 1) = intnum
            Case 33 'idle
                intnum = read_int("MOV B,X //B=")
                mem(addr + 1) = intnum
            Case 38 'idle
                intnum = read_int("JNZ (A), Address=")
                mem(addr + 1) = intnum
            Case 39 'idle
                intnum = read_int("JNZ (B), Address=")
                mem(addr + 1) = intnum
        End Select
        mem(addr) = opcode
        display_memory()
    End Sub

    '************************************************
    '*** READ INT VALUE AS INSTRUCTION PARAMETER ***
    '************************************************
    Function read_int(msg As String) As Integer
        Dim str = InputBox(msg)
        If (str = "") Then
            MsgBox("Value not selected, set to default=0")
            str = "0"
        End If
        Return Convert.ToInt32(str)
    End Function


End Class
```

CHAPTER 5
System Operation

5.1 HOMS as a system

The previous mentioned components (blocks) have to be reused in order to form a working microprocessor model. The working model consists of General and Special Purpose Registers (GPR-SPR, Register blocks) and special blocks such as, Arithmetic and Logic Unit (ALU), Control Unit (CU) and memory/output system unit (MEM-I/O). The real implemented model includes five (5) blocks as registers, one (1) block for Control Unit, one (1) block for ALU and Status Register and one (1) block for the memory/output system. Figure 5.1 shows how the hardware components constitute a working system (microprocessor, memory with user data input and output).

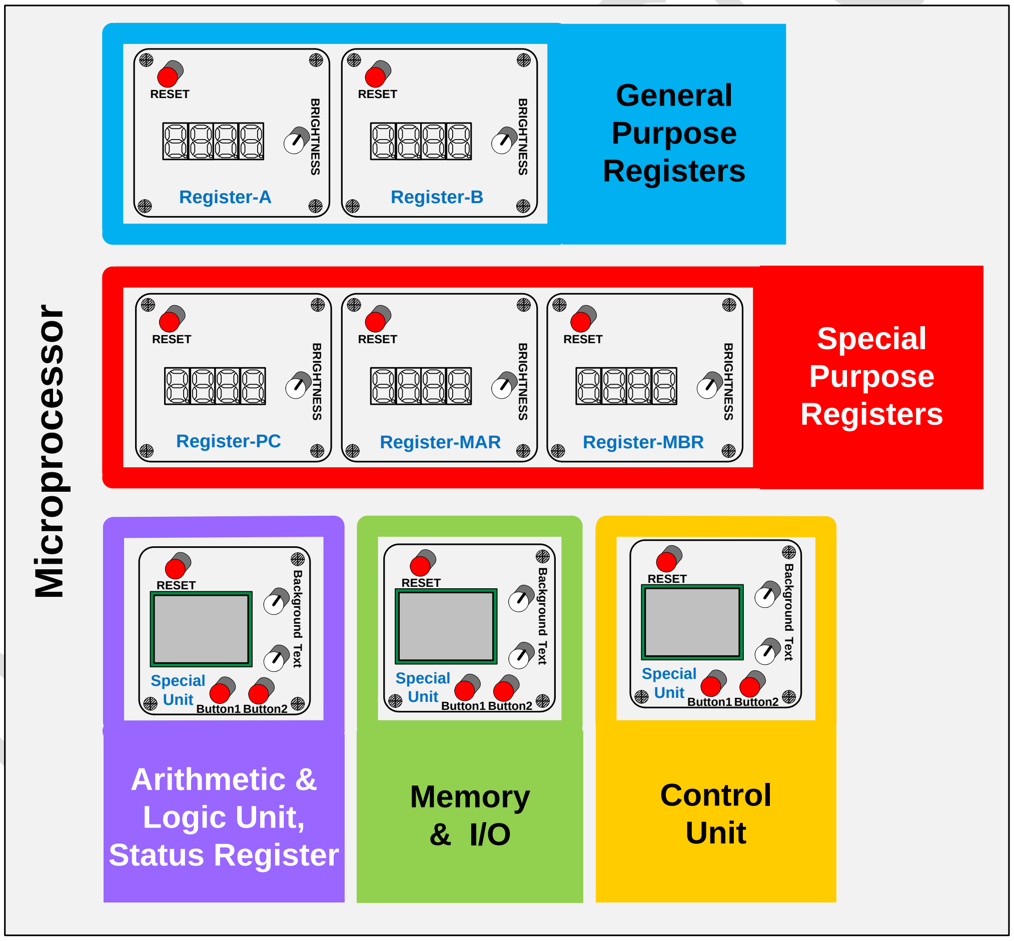


Fig. 5.1 Block organization for a working system

As shown in fig. 5.2, the data transfer between units (e.g. registers, memory) is performed via communication buses. This communication is implemented in the real model by using the I^{2}C serial protocol. This practical approach is chosen in order to simplify the physical connections between blocks. There are two pairs of common connections between all blocks (SDA-Serial Data, SCL-Serial Clock). Figure 5.2 shows the I^{2}C connections as well as the power channels (5V and GND).

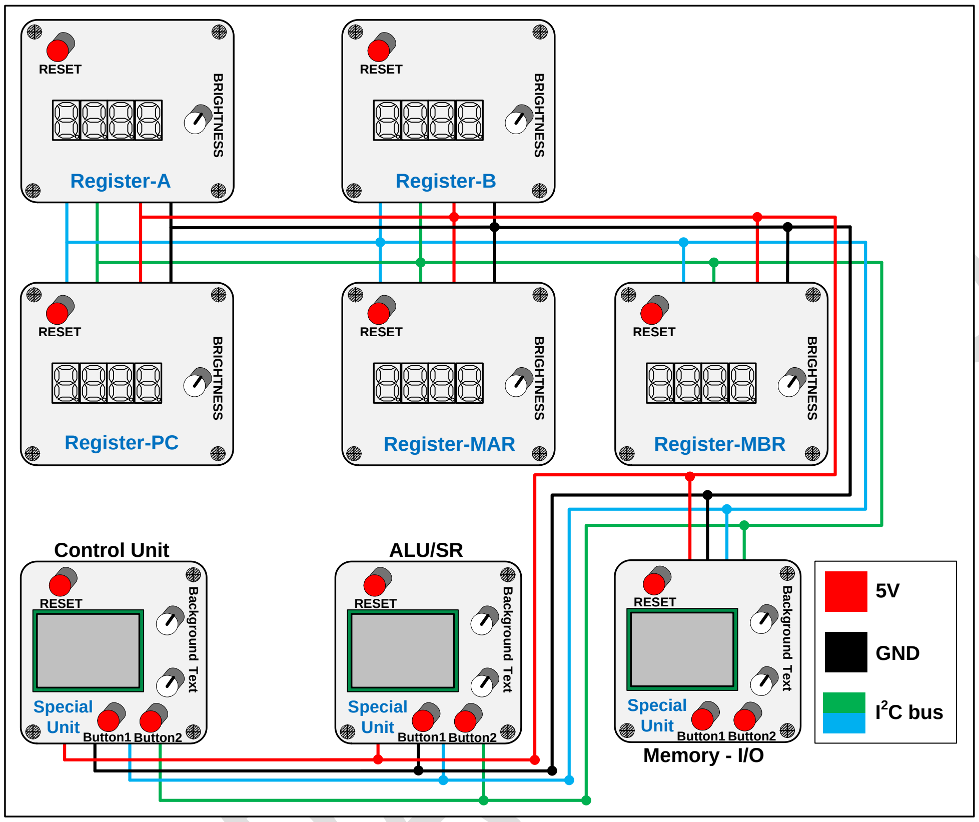


Fig. 5.2 Common connections between blocks

5.2 Assembly program execution

5.2.1 Introduction

As mentioned before, the HOMS tool constitutes a “working microprocessor” that interacts with the memory unit for executing assembly instructions. All the available HOMS blocks have to communicate to each other via the I^{2}C bus. The HOMS is based on an 8bit architecture. Thus, registers, memory contents and addresses are all 8bit. The memory unit just holds the instruction data (the first byte for the command and the second byte for the parameter). The control unit ensures that in each execution step, two bytes will be transferred to “microprocessor” starting from the current address that the PC register points to. The value of instruction bytes (command and parameter) makes sense only for the control unit in order to perform the needed actions (instruction execution). Inside the control unit, an execution loop takes place. Figure 5.3 shows the flow chart for the execution loop. Based on this process, the memory locations are scanned and the control unit executes the corresponding instructions. When a HALT instruction (code value 17) is found, the execution process is terminated. Within the loop, the following tasks are performed:

- *READ_PC*. The starting address of the next instruction to be executed is retrieved from PC register. Every assembly instruction that is stored in memory, has a constant length of two bytes. Thus, the fetching from memory can be implemented in a simple way.
- *PC2MAR.* Inside a microprocessor, the MAR (Memory Access Register) is directly connected to address bus, in order to activate a specific address for reading or writing data. At this step, the content of PC register is copied to MAR.
- *FETCH1.* Based on the MAR content, the first instruction byte is fetched from the memory. This byte is transferred to MBR (Memory Buffer Register) in order to be available to the control unit.
- *UPDATE_MAR.* The content of MAR register is updated (MAR=MAR+1) in order to point to the next address. Thus, the next byte fetching is prepared correctly.
- *FETCH2.* Based on the new MAR content, the second instruction byte is fetched from the memory. This byte is also transferred to MBR (Memory Buffer Register) in order to be available to the control unit.
- *DECODE_EXEC.* Now the command block is completed. The first byte represents the command code and the second byte the parameter. If an assembly instruction has no parameter, this byte is zero but is transferred from the memory based on the above steps.
- *UPDATE_PC.* After instruction execution, the contents of PC register are updated (adding the number 2) for fetching the next instruction from memory.

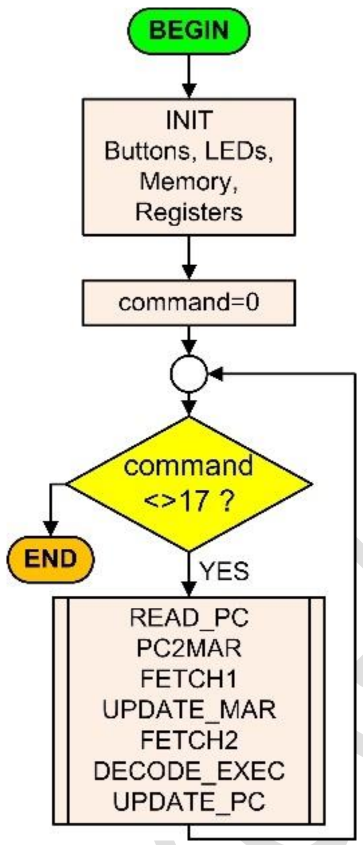
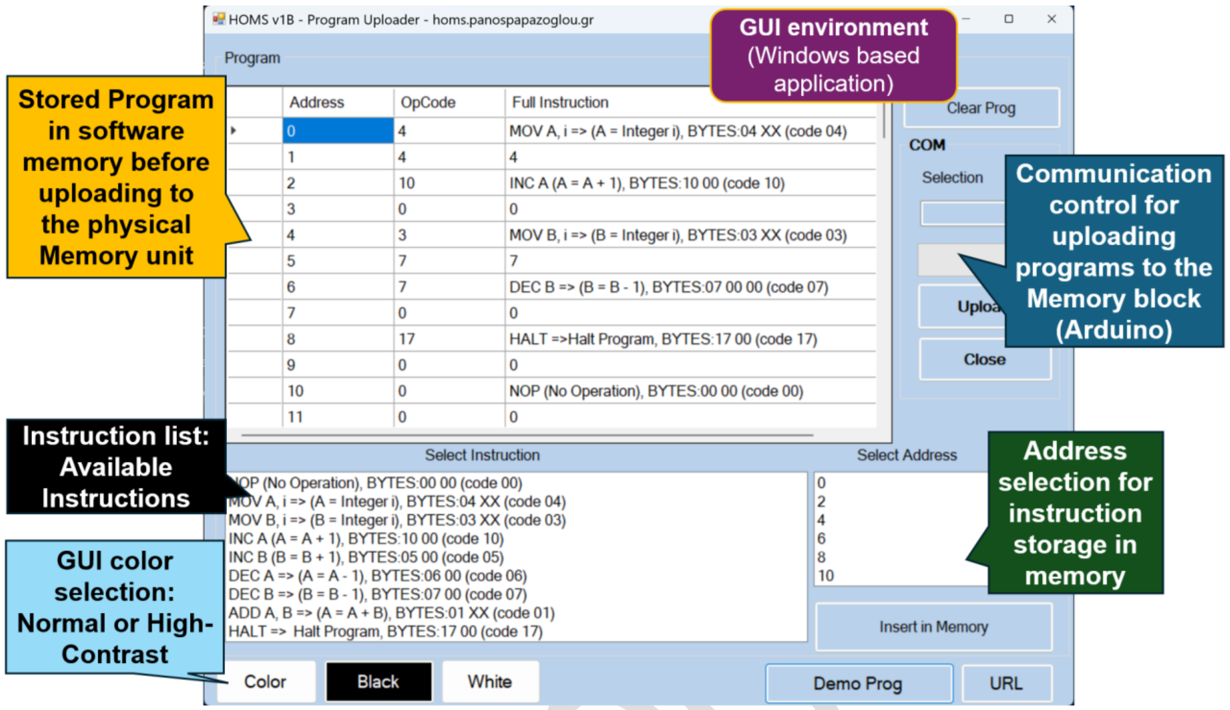


Fig. 5.3 Execution loop

5.2.2 Executing a real program

For performing a program execution, the corresponding byte codes have to be stored in the memory module. When the system starts, the user selects the **Demo** or the **Upload** operation. When the **Demo** option is selected, then the instruction **MOV A,3** is loaded in the memory unit and can be immediately executed. On the other hand, the program can be developed within the GUI environment which is a computer-based application. After the program development, the corresponding byte codes are uploaded to memory unit through a USB connection.

Figure 5.4 shows the computer-based application where the program can be developed. The GUI is organized in sections based on the corresponding functionality.


HOMS v1B - Program Uploader - homs.panospapazoglou.gr
GUI environment (Windows based application)
Stored Program in software memory before uploading to the physical Memory unit
Communication control for uploading programs to the Memory block (Arduino)
Instruction list: Available Instructions
Address selection for instruction storage in memory
GUI color selection: Normal or High-Contrast
Program
Clear Prog
COM
Selection
Uploa
Close
Select Instruction
Select Address
Insert in Memory
Color
Black
White
Demo Prog
URL

Address	OpCode	Full Instruction
0	4	MOV A, i => (A = Integer i), BYTES:04 XX (code 04)
1	4	4
2	10	INC A (A = A + 1), BYTES:10 00 (code 10)
3	0	0
4	3	MOV B, i => (B = Integer i), BYTES:03 XX (code 03)
5	7	7
6	7	DEC B => (B = B - 1), BYTES:07 00 00 (code 07)
7	0	0
8	17	HALT =>Halt Program, BYTES:17 00 (code 17)
9	0	0
10	0	NOP (No Operation), BYTES:00 00 (code 00)
11	0	0

```
NOP (No Operation), BYTES:00 00 (code 00)
MOV A, i => (A = Integer i), BYTES:04 XX (code 04)
MOV B, i => (B = Integer i), BYTES:03 XX (code 03)
INC A (A = A + 1), BYTES:10 00 (code 10)
INC B (B = B + 1), BYTES:05 00 (code 05)
DEC A => (A = A - 1), BYTES:06 00 (code 06)
DEC B => (B = B - 1), BYTES:07 00 (code 07)
ADD A, B => (A = A + B), BYTES:01 XX (code 01)
HALT => Halt Program, BYTES:17 00 (code 17)
```

```
0
2
4
6
8
10
```

Fig. 5.4 GUI environment for program development

5.2.3 Testing the demo instruction

The first thing to do after HOMS activation, is to select the demo execution or the upload from the PC. Fig. 5.5 shows the available options for preparing program execution. The first step for program execution is to load program instruction codes in memory unit.

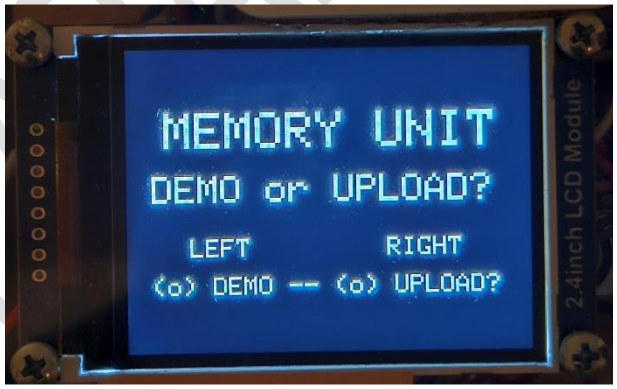
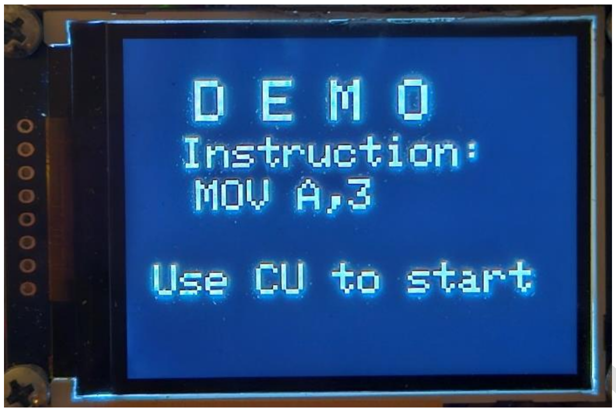


Fig. 5.5 Starting from memory unit

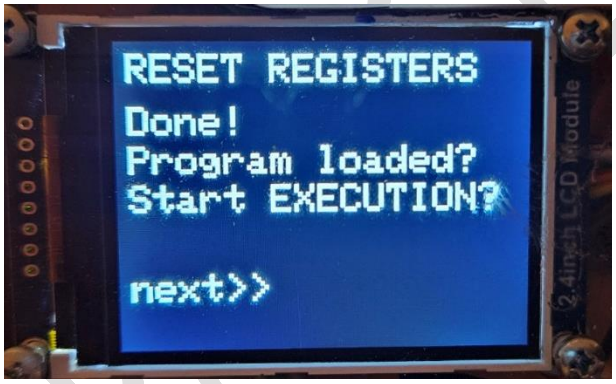
According to fig. 5.6 the user has selected the **Demo** option, where the instruction **MOV A,3** will be loaded in memory.


D E M O
Instruction:
MOV A,3

Use CU to start

Fig. 5.6 The instruction MOV A,3 will be loaded in memory

After instruction load, the control unit will be used for starting the execution process (fig. 5.7). The first step is to Reset all the HOMS registers. The initial value for all registers is FF (hexadecimal value) and after Reset, the new content will be zero (fig. 5.8).


RESET REGISTERS
Done!
Program loaded?
Start EXECUTION?

next>>

Fig. 5.7 Starting the execution process

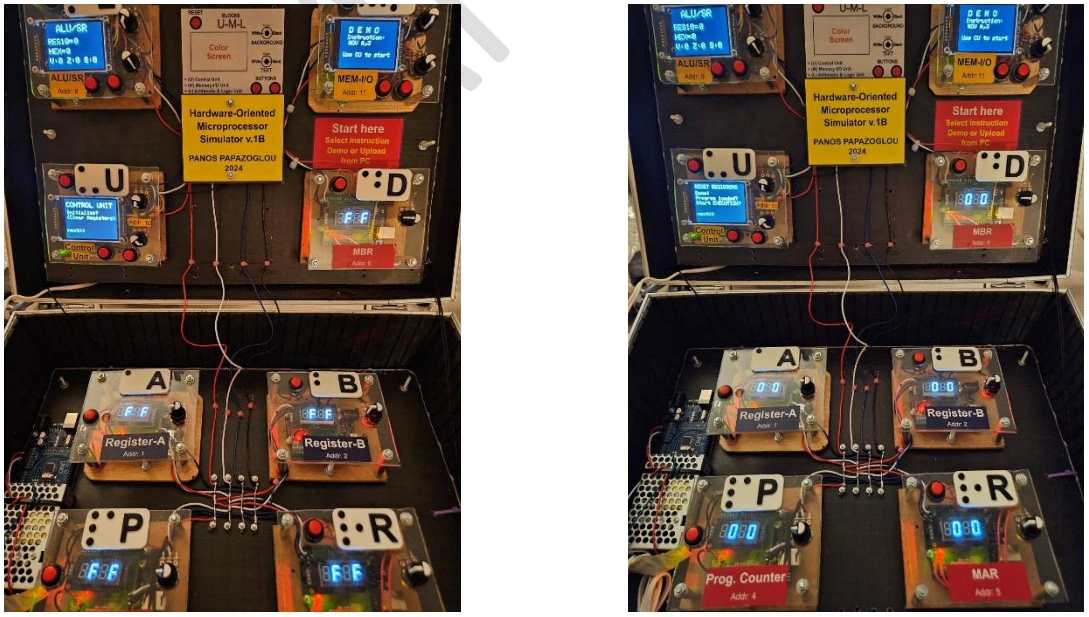


Fig. 5.8 Register Rest

5.2.4 Testing a program from PC

For a full demonstration of the HOMS v. 1B tool operation, an assembly program will be developed within the GUI environment and uploaded to the memory module. Table 5.1 shows the demo program (symbolic instruction, byte code and memory contents).

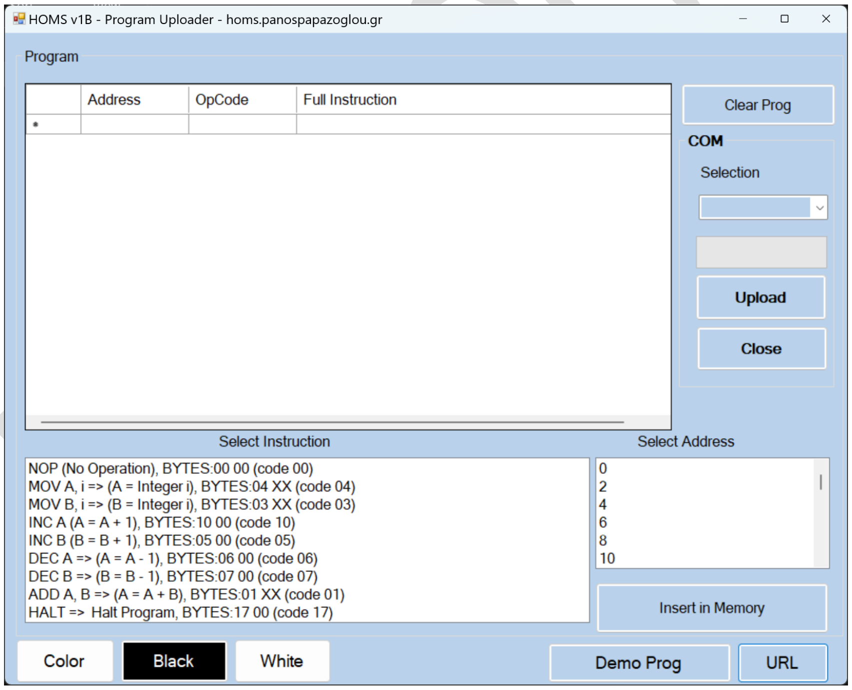
Table 5.1
Demo program

Instruction	Byte code	Address (content) (in decimal)
MOV A, 6	(dec) 04 04, (hex) 04 04	00* (04), 01 (04)
INC A	(dec) 10 00, (hex) 0A 00	02* (10), 03 (00)
MOV B, 7	(dec) 03 07, (hex) 03 07	04* (03), 05 (07)
DEC B	(dec) 07 00, (hex) 07 00	06* (07), 07 (00)
HALT	(dec) 17 00, (hex) 11 00	08* (17), 09 (00)

* Instruction starting address (PC content)

STEP 1 - GUI Application execution

After the PC application execution, the GUI environment will be activated (fig. 5.9).



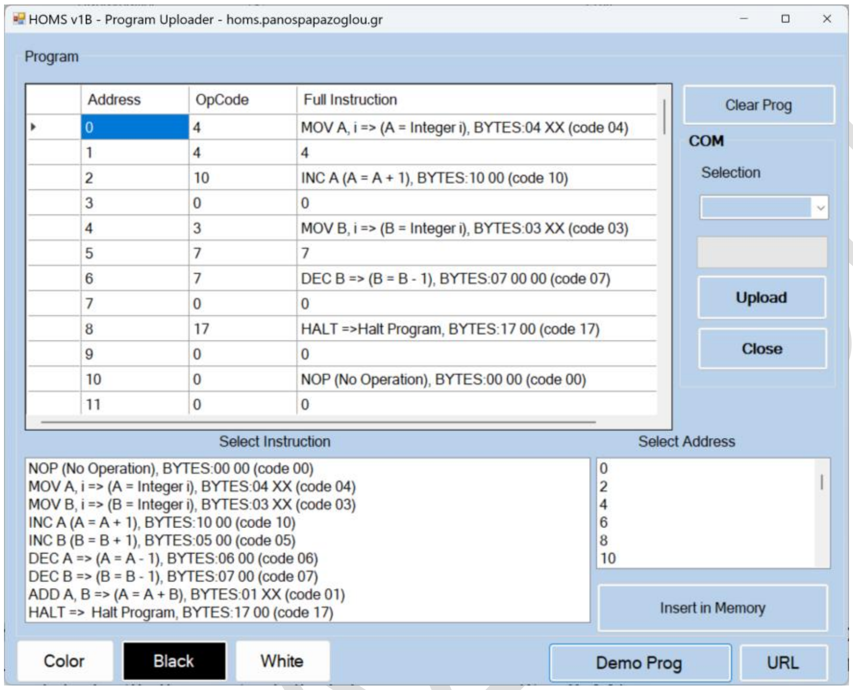
NOP (No Operation), BYTES:00 00 (code 00)
MOV A, i => (A = Integer i), BYTES:04 XX (code 04)
MOV B, i => (B = Integer i), BYTES:03 XX (code 03)
INC A (A = A + 1), BYTES:10 00 (code 10)
INC B (B = B + 1), BYTES:05 00 (code 05)
DEC A => (A = A - 1), BYTES:06 00 (code 06)
DEC B => (B = B - 1), BYTES:07 00 (code 07)
ADD A, B => (A = A + B), BYTES:01 XX (code 01)
HALT => Halt Program, BYTES:17 00 (code 17)

0
2
4
6
8
10

Fig. 5.9 The GUI environment

STEP 2 - Inserting the program

The program can be entered step by step, but we will use the **Demo Prog** option for automatic program insertion. When the button **Demo Prog** is pressed, the program table area is populated with the preinstalled demo program (fig. 5.10).


HOMS v1B - Program Uploader - homs.panospapazoglou.gr

Program

Address	OpCode	Full Instruction
0	4	MOV A, i => (A = Integer i), BYTES:04 XX (code 04)
1	4	4
2	10	INC A (A = A + 1), BYTES:10 00 (code 10)
3	0	0
4	3	MOV B, i => (B = Integer i), BYTES:03 XX (code 03)
5	7	7
6	7	DEC B => (B = B - 1), BYTES:07 00 00 (code 07)
7	0	0
8	17	HALT =>Halt Program, BYTES:17 00 (code 17)
9	0	0
10	0	NOP (No Operation), BYTES:00 00 (code 00)
11	0	0

Select Instruction

```
NOP (No Operation), BYTES:00 00 (code 00)
MOV A, i => (A = Integer i), BYTES:04 XX (code 04)
MOV B, i => (B = Integer i), BYTES:03 XX (code 03)
INC A (A = A + 1), BYTES:10 00 (code 10)
INC B (B = B + 1), BYTES:05 00 (code 05)
DEC A => (A = A - 1), BYTES:06 00 (code 06)
DEC B => (B = B - 1), BYTES:07 00 (code 07)
ADD A, B => (A = A + B), BYTES:01 XX (code 01)
HALT =>  Halt Program, BYTES:17 00 (code 17)
```

Select Address

```
0
2
4
6
8
10
```

Fig. 5.10 The program is inserted

STEP 3 - Activating HOMS Tool and Upload option

The USB cable is plugged from PC into the memory unit (fig. 5.11) and the option UPLOAD is chosen (fig. 5.12a, 5.12b).

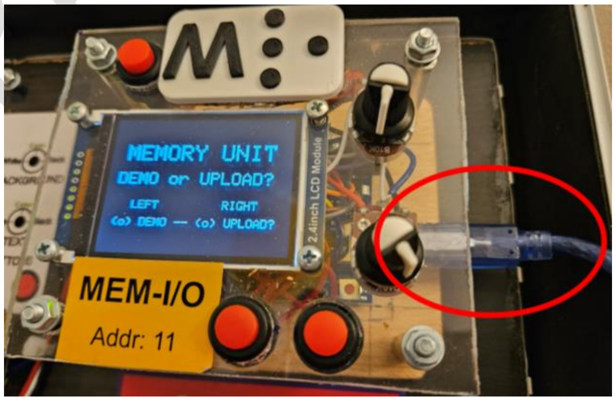


Fig. 5.11 USB connection

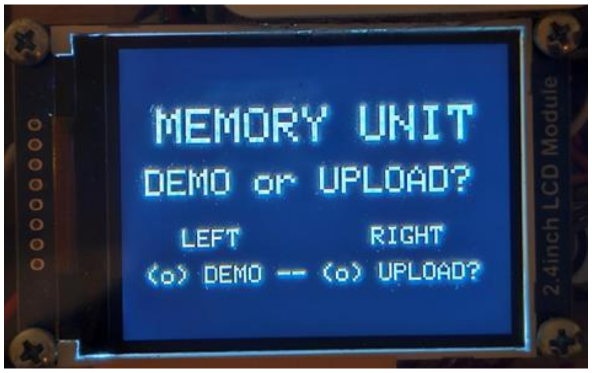


fig. 5.12a Main menu in memory unit

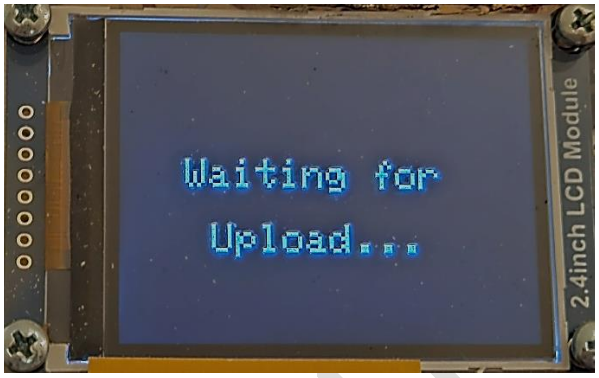
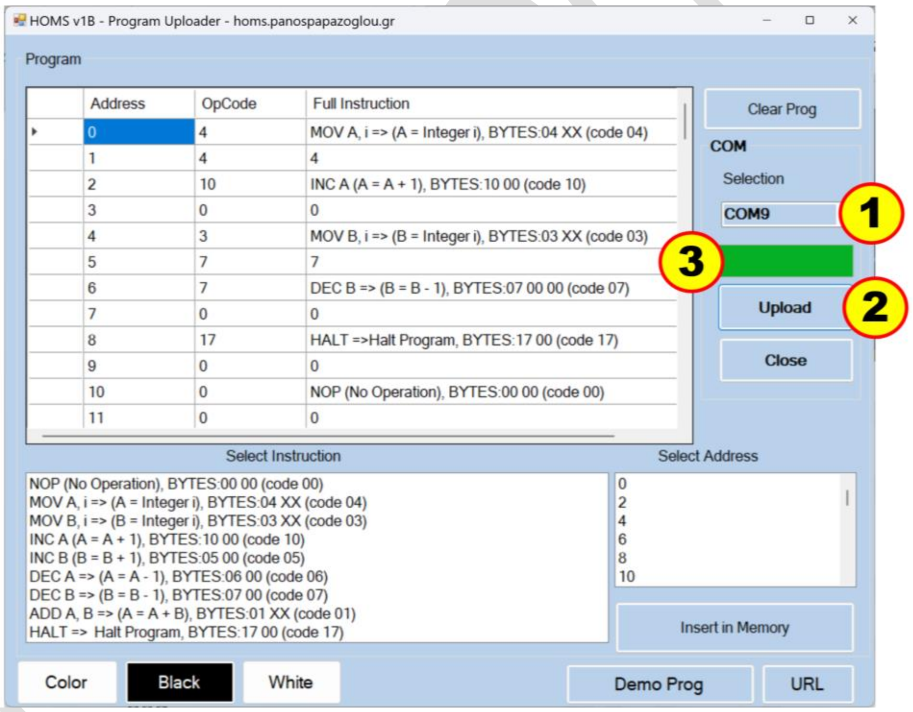


fig. 5.12b Memory unit is waiting to receive from PC

STEP 4 - Uploading from PC

Firstly, we select the COM port where the memory unit (Arduino) is connected (1). The next step is to press the **Upload** button (2). The upload process is confirmed through the green bar (3). Figure 5.13 shows the above steps.



COM9

Address	OpCode	Full Instruction
0	4	MOV A, i => (A = Integer i), BYTES:04 XX (code 04)
1	4	4
2	10	INC A (A = A + 1), BYTES:10 00 (code 10)
3	0	0
4	3	MOV B, i => (B = Integer i), BYTES:03 XX (code 03)
5	7	7
6	7	DEC B => (B = B - 1), BYTES:07 00 00 (code 07)
7	0	0
8	17	HALT =>Halt Program, BYTES:17 00 (code 17)
9	0	0
10	0	NOP (No Operation), BYTES:00 00 (code 00)
11	0	0

Select Instruction

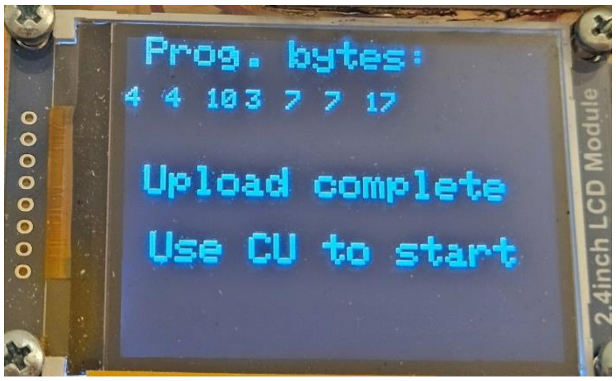
NOP (No Operation), BYTES:00 00 (code 00)
MOV A, i => (A = Integer i), BYTES:04 XX (code 04)
MOV B, i => (B = Integer i), BYTES:03 XX (code 03)
INC A (A = A + 1), BYTES:10 00 (code 10)
INC B (B = B + 1), BYTES:05 00 (code 05)
DEC A => (A = A - 1), BYTES:06 00 (code 06)
DEC B => (B = B - 1), BYTES:07 00 (code 07)
ADD A, B => (A = A + B), BYTES:01 XX (code 01)
HALT => Halt Program, BYTES:17 00 (code 17)

Select Address

0
2
4
6
8
10

Fig. 5.13 Upload process

If the program upload is successful, then the corresponding bytes will be appeared on the TFT display of the memory unit (fig. 5.14).



Prog. bytes:
4 4 103 7 7 17
Upload complete
Use CU to start

Fig. 5.14 Upload successful

STEP 5 – Program Execution through the Control Unit

Now, the uploaded program can be executed step by step using the on-display instructions at the control unit.

The execution steps inside the HOMS tool are described in the following table.

Table 5.2 shows as an example, how the instruction **MOV A,4** is executed.

Table 5.2
Execution steps for instruction MOV A,4

Register	1	2	3	4	5	6	7
PC	00					DECODE	
MAR		00		01			
MBR			04		04		
A							04

As shown in table 5.2, the instruction *MOV A,4* is executed as follows:

STEP 1: The PC shows the starting address of the instruction to be executed (MOV A,4)
STEP 2: The starting address of the instruction is stored in MAR register
STEP 3: The first instruction byte is fetched and is stored in MBR register
STEP 4: The MAR address is increased by one, in order to point to the next address where the second byte of the instruction is stored (parameter)
STEP 5: The second instruction byte is fetched and is stored in MBR register
STEP 6: The control unit decodes the instruction bytes and starts to execute the instruction
STEP 7: The content of register A is now 04

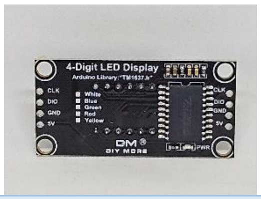
The above steps can be now confirmed inside the real HOMS tool environment.

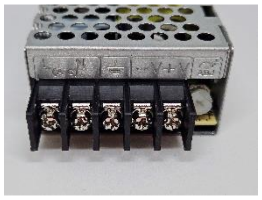
***Important note:* please visit the web site of the HOMS project for viewing the corresponding videos.**

CHAPTER 6
Basic Electronic and other components

All the necessary components for building the HOMS v. 1B tool are very common and can be found in any local or international market. On the other hand, the block dimensions, the TFT screens, the buttons, etc, can be very different as compared to the presented HOMS tool implementation based on designer's choices. Current implementation shows how the concept of the object-oriented approach can be applied. Table 6.1 shows some basic components that have been used in HOMS tool version 1B.

Table 6.1
Components

Component	Physical form	Number of items
Microcontroller platform	Arduino UNO 16MHz (compatible board)	**9** 5: register blocks 3: special blocks 1: RGB LED-Strip control
4-digit Display		**5** 5: register blocks
TFT Display		**3** 3: special blocks

	2.4 Inch, LCDTFT	
Power Supply	 5V/5A	**1** 1: RGB LED-Strip
Power Supply	5V/3A	**1** 1: HOMS tool
Pot knobs	Plastic	**11** 5: register blocks 6: special blocks
Potentiometer		**11** 5: 1KΩ linear (register blocks) 6: 10KΩ linear (special blocks)
Button		**14** 5: register blocks 9: special blocks

***Important note:* for building instructions, please read the complete technical reference of HOMS version 1 (sections 4.3 and 4.4).**

About the author [Dr. Panayotis (Panos) Papazoglou]

Associate professor, Department of Digital Arts and Cinema, National and Kapodistrian University of Athens.
He worked also as Lecturer, Assistant Professor and Associate Professor at Technological Educational Institutes of Athens, Lamia and Central Greece, Departments of Electronics, Computer Engineering (Head of Department 2015–2016) and Electrical Engineering respectively. He teaches Computer Architecture and Microprocessor programming for more than 25 years with a total academic experience more than 27 years. Dr P. Papazoglou is the author of 14 scientific-technical books (12 in Greek and 2 in English -Amazon, USA-) and has more than 50 publications in international journals, book chapters and conferences. He is the author of the Greek best seller book "Application Development with Arduino" and the most popular book about microprocessors.

This project did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

References

1. P.M.Papazoglou, A Hybrid Simulation Platform for Learning Microprocessors, Computer Applications in Engineering Education, 10.1002/cae.21921, (pp 655-674) WILEY, 2018
2. HOMS Project version 1 (https://homs.panospapazoglou.gr/)

Website

https://homs.panospapazoglou.gr/