# Assembler

את הפרוייקט הזה חילקנו ל2 חלקים:

- Static Definitions – הגדרה סטטית כל כל הרגיסטרים והפקודות.
- Static Definitions – פונקציות עזר.
- Main – הלוגיקה של האסמבלר והפעולה שלו.

Static Definition

ראשית, הגדרנו קבועים ומבנים הקשורים לריגסטרים ולפעולות:

```
#define NUMBER_OF_REGISTERS 16
#define NUMBER_OF_OPCODES 22
#include <string.h>

typedef struct
{
    char* RegisterName;
    char* RegisterNumber;
}Register;

typedef struct
{
    char* OpcodeName;
    char* OpcodeNumber;
}Opcode;
```

מוגדרים לנו מספר הרגיסטרים ומספר הפעולות האפשריות.

עבור כל רגיסטר הוגדר השם שלו והמספר שלו, עבור כל פעולה הוגדר השם. בגלל שאנו קוראים וכותבים את הפקודות והרגיסטרים כטקסט אז התייחסנו להכל כמחרוזות.

לאחר מכן בנינו מיפוי סטטי של כל הפקודות והמחרוזות:

```
Register RegisterMapping[NUMBER_OF_REGISTERS] = {
    {"$zero", "0"}, // Constant zero
    {"$imm", "1"},  // Sign extended immediate
    {"$v0", "2"},   // Result value
    {"$a0", "3"},   // Argument register
    {"$a1", "4"},   // Argument register
    {"$t0", "5"},   // Temporary register
    {"$t1", "6"},   // Temporary register
    {"$t2", "7"},   // Temporary register
    {"$t3", "8"},   // Temporary register
    {"$s0", "9"},   // Saved register
    {"$s1", "A"},   // Saved register
    {"$s2", "B"},   // Saved register
    {"$gp", "C"},   // Global pointer (static data)
    {"$sp", "D"},   // Stack pointer
    {"$fp", "E"},   // Frame Pointer
    {"$ra", "F"},   // Return address
};
```

```
Opcode OpcodeMapping[NUMBER_OF_OPCODES] = {
    {"add", "00"},
    {"sub", "01"},
    {"and", "02"},
    {"or", "03"},
    {"xor", "04"},
    {"mul", "05"},
    {"sll", "06"},
    {"sra", "07"},
    {"srl", "08"},
    {"beq", "09"},
    {"bne", "0A"},
    {"blt", "0B"},
    {"bgt", "0C"},
    {"ble", "0D"},
    {"bge", "0E"},
    {"jal", "0F"},
    {"lw", "10"},
    {"sw", "11"},
    {"reti", "12"},
    {"in", "13"},
    {"out", "14"},
    {"halt", "15"},
};
```

בסוף, הגדרנו 2 פונקציות עזר לחיפוש רגיסטר/פקודה לפי השם שלה לקבל מספר בHEX:

```
/*Getting register hex value number by name.
Return -1 if not found
*/
char* GetRegisterNumber(char* name) { ... }

/*Getting opcode hex value number by name.
Return -1 if not found
*/
char* GetOpcodeNumber(char* name) {
    for (int i = 0; i < NUMBER_OF_OPCODES; i++) { ... }

    return -1;
}
```

Helpers

```
#ifndef HELPER
#define HELPER

typedef unsigned int uint;
typedef struct {
    char* LabelName;
    uint LabelLine;
}Label;

void RemoveLastChar(char* str);
int GetDecimalFromHex(char* hexValue);
int GetDecimalValueFromString(char* str);
void GetHexValueOfConstant(uint num, char* hexVal, int numOfBytes);

int HasImmediate(char* rd, char* rs, char* rt);

#endif
```

**Label –** Struct המכיל 2 שדות – השם של הלייבל ומספר השורה שבו הוא נמצא. בלולאה שלנו כל פעם שנזהה label ניצור את המבנה עם הפרמטרים הנ"ל.

**RemoveLastChar** – מוחק את האות האחרונה של המחרוזת.

**GetHexValueOfConstant** – בהנתן קבוע מחזיר את הערך שלו בHEX(הפונקציה מקבלת את מספר הבתים שאותם אנו רוצים בHEX ועושה הארכת סימן בהתאם).

**GetDecimalFromHex** – בהנתן ערך בHEX מחוזירה את הערך הדצימלי שלו.

**HasImmediate** – בודק אם אחד משלושת הרגיסטרים הוא הערך $imm ומחזיר TRUE אם אכן מתקיים.

Main

```
int main(int argc, char* argv[]) {
    if (argc != 4)
        return 1;

    asmFile = fopen(argv[1], "r");
    imemFile = fopen(argv[2], "w");
    dmemFile = fopen(argv[3], "w");
    if (asmFile == NULL || imemFile == NULL || dmemFile == NULL) {
        printf("Error! opening file");
        exit(1);
    }

    InitMemoryAddress();

    ExtractLabels();
    ExtractCommands();
    ExtractMemoryFile();

    FreeAllMemory();

    fclose(asmFile);
    fclose(imemFile);
    fclose(dmemFile);
    return 0;
}
```

הלולאה המרכזית מבצעת 2 מעברים על הקוד אסמבלי:

1) מוצאת את כל הLabels בקוד ושומרת אותו במערך של מבנה של Label.
2) עושה Parse לכל שורה בקוד בהתאם לפקודות של הקוד אסמבלי וכותבת אותו לקבצי הפלט הרצויים.
3) ממיר את מערך תמונת הזיכרון שלנו וממלא את קובץ היציאה.

אלו הן ההגדרות הסטטיות של הקובץ:

```
#define _CRT_SECURE_NO_WARNINGS
#define MAX_LINE_LENGTH 500
#define MEMORY_ADDRESS_NUM 4096

#include <stdio.h>
#include <stdlib.h>
#include <string.h>


#include "Misc/Helpers.h"
#include "Misc/StaticDefinitions.h"

// Files
FILE* asmFile;
FILE* imemFile;
FILE* dmemFile;

// List of labels
Label* LabelsLocations = NULL;
uint labelsCounter = 0;

// Memory
char* memoryAddress[MEMORY_ADDRESS_NUM];
```

פונקציות העזר שהגדרנו לטיפול בקוד האסמבלי:

```
/* Init memory pointers to NULL */
void InitMemoryAddress() { ... }

int InsideLabelList(char* potentialLabel) { ... }

int GetLabelLine(char* label) { ... }

void AddLabelToList(char* label, uint lineCounter) { ... }

void ExtractLabels() { ... }

void HandleWordCommand(char* location, char* value) { ... }

void HandleCommand(char* opcode, char* rd, char* rs, char* rt, char* immVal) { ... }

void ExtractCommands() { ... }

void ExtractMemoryFile() { ... }

void FreeAllMemory() { ... }
```

**InitMemoryArray** – מאתחל את תמונת הזיכרון לאפסים בתחילת הריצה.

**InsideLableList** – בודק האם הLabel כבר נמצא ברשימה שלנו. אם כן מחזיר TRUE, אחרת FALSE.

**GetLabelLine** – בהנתן Label מחזיר את מספר השורה שלו עפ"י המעבר הראשוני שעשינו על הקובץ.

**AddLabelToList** – כאשר נמצא את הlabel נכניס אותו לרשימת הlabels שלנו.

**ExtractLabels** – אנו עוברים שורה שורה בקובץ הasm. ומחפשים בכל שורה תבנית של Label. במקביל, בכל מעבר אנו סופרים כמה שורות עברנו. במידה ומצאנו Label נכניס אותו לרשימה עם השם שלו ומספר השורה.

**HandleWordCommand** – במידה ומצאנו שורה הכוללת פקודת Word אנו מטפלים בה בפונקציה הזו(מעדכנים את תמונת הזיכרון).

**HandleCommand** – במידה ומצאנו שורה הכוללת פקודה אנו דואגים להכניס אותה בצורתה הנכונה בקובץ הפלט.

**ExctarctCommand** – עוברים שורה שורה בקובץ הasm. ומחפשים תבנית של פקודה. במידה ומצאנו תבנית מתאימה נדפיס אותו לקובץ עם תרגום של כל שם רגיסטר לערך המייצג אותו לפני המיפוי שבנינו.

כמו כן, אנו בודקים האם קיבלנו את הפקודה השמורה word. . אם כן, נוסיף את הערך המתאים למערך המדמה את תמונת הזיכרון שלנו.

**ExctractMemoryFile** משחררים את כל ההקצאות הדינמיות שנעשו במהלך הריצה.

**FreeAllMemory** – עוברים על כל המערך המדמה את תמונת הזיכרון. כותבים את הערכים שאותחלו בקובץ הasm במידה ונכתבו, אם לאו אז נמלא אפסים.

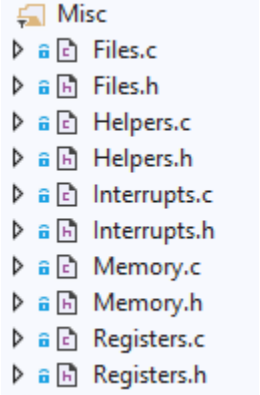
# Simulator

החלק הזה בפרוייקט מעט יותר מורכב ומחולק ליותר חלקים. חילקנו את הפרוייקט לשלושה חלקים מרכזיים –

1) IO
2) Misc
3) Main

## Misc

חלק זה כולל הגדרות גלובליות לכל הפרוייקט או פונקציות כלליות שנשתמש בהם בלולאה המרכזית.



Registers:

דומה לקובץ הStaticDefinitions שעשינו באסמבלר בצורה מורחבת שיתאים לנו לסימולטור.

```
#define REGISTERS

#define NUMBER_OF_REGISTERS 16
#define NUMBER_OF_IO_REGISTERS 22
#define NUMBER_OF_OPCODES 22
#define INSTRUCTION_COUNT (InstructionCommands + InstructionCounter - 1)->PCLocation
#include "Helpers.h"

typedef enum { ... } IORegisterNames;

typedef struct { ... } Register;
typedef struct { ... } IORegister;
typedef struct { ... } Opcode;

typedef struct { ... } InstructionCommand;

InstructionCommand* InstructionCommands;
uint InstructionCounter;

Register RegisterMapping[NUMBER_OF_REGISTERS];
IORegister IORegisterMapping[NUMBER_OF_IO_REGISTERS];
Opcode OpcodeMapping[NUMBER_OF_OPCODES];

void InstructionInit(void);
InstructionCommand* GetInstructionCommand(uint pc);
uint IncreasePCAmount(InstructionCommand command);

void FreeInstructionsCommandArray(void);

#endif
```

**InstructionCommand** – מערך של כל הפקודות שאותן אנו מריצים.

**InstructionCounter** – גודל מערך הפקודות.

**InstructionCount** – המספר של הפקודה האחרונה.

נסתכל על המבנים החשובים:

```c
typedef struct
{
    char* RegisterName;
    char* RegisterNumber;
    uint RegisterValue;
}Register;

typedef struct
{
    char* RegisterName;
    char* RegisterNumber;
    int NumberOfBits;
    uint RegisterValue;
}IORegister;

typedef struct
{
    char* OpcodeName;
    char* OpcodeNumber;
    void (*OperationFunc)(uint rd, uint rs, uint rt);
}Opcode;
```

**Register** – מגדיר רגיסטר חומרתי של המעבד.

**IORegister** – מגדיר רגיטסר IO ואת מספר הביטים שיש לו.

**Opcode** – מגדיר פקודה ומצביע לפונקציה שתבצע את הפקודה המתאימה.

בשילוב כל מבנים אלו נגדיר שורת פקודה בצורה הבאה:

```
typedef struct
{
    uint PCLocation;
    Opcode opcode;
    uint rd;
    uint rs;
    uint rt;
    int ImmValue;
    int HasImmediate;
    char* Name;
}InstructionCommand;
```

**PCLocation** – מספר הPC של הפקודה.

**Opcode** – סוג הפקודה.

**Rd** – רגיסטר Rd.

**Rs** – רגיסטר Rs.

**Rt** – רגיסטר Rt.

**ImmValue -** ערך immediate אם קיים עבור פקודה זו.

**HasImmediate** – האם זו פקודת Immediate.

Name – שם הפקודה.

בהמשך הקובץ מוגדרים שלושה מיפויים – רגיסטרי החומרה, רגיסטרי IO ולפקודות. המיפויים האלו גלובליים ומשומשים בשאר חלקי המערכת.

**InstructionInit** – עוברים על הקובץ שאותו המרנו דרך האסמבלר ויוצרים את מערך הפקודות שלנו.

**GetInstructionCommand** – בהנתן PC מחזיר את מבנה הפקודה המתאים מתוך המערך.

**IncreasePCAmount** – בהנתן פקודה מעלה את הPC לפי התנאי האם הפקודה היא מסוג Immediate או לא.

**FreeInstructionsCommandArray** – משחרר את כל ההקצאות הדינמיות הקשורות במסמך זה.

Helpers:

הגדרה של פונקציות עזר ומשתנים סטטים גלובליים.

```
#ifndef HELPER
#define HELPER

typedef unsigned int uint;

uint ProgramCounter;
uint ClockCycles;
uint TotalInstructionsCommand;

int GetDecimalFromHex(char* hexValue);
int GetDecimalFromHex2Comp(char* hexValue);
void RemoveLastChar(char* str);
#endif
```

**ProgramCounter** – PC של כל התוכנית.

**ClocklCycles** - מספר מחזורי השעון שהתוכנית רצה.

**TotalInstructionsCommad** – מספר הפקודות שהתוכנית ביצעה.

**GetDecimalFromHex** – קבלת מספר דצימלי מHEX בלי הארכת סימן.

**GetDecimalFromHex2Comp** – קבלת מספר דצימלי מHEX לפי משלים ל2.

**RemoveLastChar** – מוריד את האות האחרונה של מחרוזת.

**Files:**

```
#include "Registers.h"

FILE* ImemInFile;
FILE* DmemInFile;
FILE* DiskInFile;
FILE* Irq2InFile;
FILE* DmemOutFile;
FILE* RegOutFile;
FILE* TraceFile;
FILE* HwRegTraceFile;
FILE* CyclesFile;
FILE* LedsFile;
FILE* MonitorFile;
FILE* MonitorYuvFile;
FILE* DiskOutFile;

int OpenFiles(char* argv[]);
void CloseFiles(void);

void WriteTrace(InstructionCommand command);
void WriteRegistersToFile(void);
void WriteCyclesToFile(void);

#endif
```

ראשית, יש לנו הגדרה גלובלית של כל הקבצים שאותם נכתוב/נקרא במהלך הריצה.

**OpenFiles** – פותח את כל הקבצים בהתאם לתפקיד שלו בריצה ובודק שאכן נפתחו בצורה תקינה.

**CloseFiles** – סוגר את כל הקבצים.

**WriteTrace** – בהנתן שורת פקודה כותב שורה אחת לקובץ הTrace.

**WriteRgitersToFile** – כותב את קובץ הRegOut .

**WriteCyclesToFile** – כותב את קובץ הCycles.

**Interrupts:**

קובץ האחראי על ניהול הפסיקות שלנו.

```
#ifndef INTERRUPTS
#define INTERRUPTS

#include "Helpers.h"

uint InterruptBusy;

// General
void InitInterrupts();
uint GetIrqSignal(void);
void HandleInterrupt(void);
void ExecuteInterrupts(uint incrementValue);

void FreeInterruptsMemory(void);

#endif
```

**InterruptsBusy** – דגל שמסמן האם ישנה פסיקה בטיפול.

**InitInterrupts** – מאתחל את הדגל להיות אפס וקורא את כל ערכי הזמנים שבהם ייקרו זמני הפסיקות של irq2.

**HandleInterrupt** – בודק האם הפסיקה התסיימה. במידה וכן, מוריד את הדגל ומחזיר את הPC לפני היציאה לפסיקה.

**ExecuteInterrupts** – בודק את הסטטוס של כל אחת מהפסיקות, ומטפל בהן עפ"י הצורך.

**FreeInterruptsMemory** – משחרר את כל ההקצאות הדינמיות הקשורות במסמך זה.

**Memory:**

תמונת הזיכרון של המעבד.

```
#ifndef MEMORY
#define MEMORY
#define MEMORY_SIZE 4096
#include "Helpers.h"

uint Memory[MEMORY_SIZE];

void MemoryInit(void);
void WriteMemoryToFile(void);

#endif
```

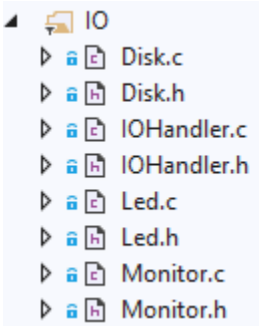
**Memory** – מערך המדמה את תמונת הזיכרון של המעבד.

**MemoryInit** – פונקציה הקוראת את תמונת הזיכרון מקובץ הקלט אל המערך שלנו.

**WriteMemoryToFile** – פונקציה הכותבת תמונת הזיכרון בסוף הריצה אל קובץ הפלט.

## IO

כולל את כל רכיבי החומרה החיצוניים של הסימולטור.




IOHandler:

```
#ifndef IO_HANDLER
#define IO_HANDLER
#include "../Misc/Helpers.h"

void HandleIOs(uint incrementValue);

#endif
```

**HandleIOs** – הפקודה היחידה בחלק זה. עובר על כל מכשירי הIO ומטפל בהם עפ"י הצורך.

Disk:

```
#ifndef DISK
#define DISK
#define SECTOR_NUMBER 128
#define SECTOR_SIZE 128

#include <stdio.h>
#include "../Misc/Helpers.h"

uint DiskTimer;
uint DiskSectorMemory[SECTOR_NUMBER][SECTOR_SIZE];

void InitDiskMemory(void);
void WriteDiskMemory(void);

int DiskCommand(uint timerIncrement);

#endif
```

ראשית, אנו מגדירים בצורה סטטית את הגודל של הדיסקט שלנו – מספר הסקטורים ומספר האיברים בכל סקטור.

**DiskTimer** – ברגע שמגיעה פקודת IO לדיסק זהו הטיימר שסופר שעברו 1024 מחזורי שעון.

**InitDiskMemory** – מאתחל את זכרון הדיסק שלנו לפי קובץ הקלט לתוכנית.

**WriteDiskMemory** – כותב את תמונת זיכרון הדיסק בסוף הריצה.

:LED

```
#ifndef LED
#define LED
#include "../Misc/Helpers.h"

uint LEDValue;
void WriteLEDStatus();

#endif
```

**LEDValue** – שומר את כל ערכי הלדים. ברגע שיש שינוי אנו מזהים שהערך בתוך הרגיסטר שונה מהערך השמור(שהוא בעצם הערך הקודם).

**WriteLEDStatus** – ברגע שזיהינו שינוי בערך הלדים אנו כותבים בפונקציה זו את ערכי הלדים הנוכחיים לקובץ.

:Monitor

```
#ifndef MONITOR
#define MONITOR
#define NUMBER_OF_PIXEL_X 352
#define NUMBER_OF_PIXEL_Y 288

#include <stdio.h>
#include "../Misc/Helpers.h"

uint MonitorData[NUMBER_OF_PIXEL_X][NUMBER_OF_PIXEL_Y];

void InitMonitor(void);
void MonitorCommand(void);
void WriteMonitorData();
#endif
```

אנו שומרים כקבועים את ערכי גודל המסך שלנו.

**InitMonitor** – מאתחלים בתחילת הריצה את המוניטור להיות מערך דו מימדי של מסכים.

**MonitorCommand** – פקודה שנקראת כאשר רגיסטר Monitor CMD איננו 0. מעתיקים מהבאפר את הערך ולאיזו נקודה במסך אנו כותבים.

**WriteMonitorData** – בסוף ריצה אנו מעתיקים את הערך האחרון של המסך לקובץ טקסט ולקובץ yuv.

## Main

ריצת הסימולור.

```
int main(int argc, char* argv[]) {
    if (argc != 14)
        return 1;

    if (OpenFiles(argv))
    {
        printf("Error! opening file");
        exit(1);
    }

    // Init Stage
    Init();

    // Execute Stage
    MainLoop();

    // Exit Stage
    Exit();
}
```

סדר הפעולות שלנו –

- פותחים את כל הקבצים שלנו ובודקים את תקינותם.
- Init:

```
/*Initiate all relevant modules*/
void Init()
{
    ProgramCounter = 0;
    ClockCycles = 0;

    MemoryInit();
    InitDiskMemory();
    InstructionInit();
    InitInterrupts();
}
```

מאתחלים את הPC ואת ClockCycles לאפס. מאתחלים מודולים רלוונטיים.

:MainLoop -

```
void MainLoop()
{
    while (ProgramCounter <= INSTRUCTION_COUNT)
    {
        // Get instruction command
        InstructionCommand* command = GetInstructionCommand(ProgramCounter);
        if (command->HasImmediate)
        {
            RegisterMapping[1].RegisterValue = command->ImmValue;
        }

        // Writing current trace
        WriteTrace(*command);

        // Increase PC -
        // PC update is on this location because the jump/branch commands need the updated value
        uint incrementPCValue = IncreasePCAmount(*command);
        ProgramCounter += incrementPCValue;

        // Execute command
        command->opcode.OperationFunc(command->rd, command->rs, command->rt);

        // Handle IOs
        HandleIOs(incrementPCValue);

        // Increase ClockCycle
        ClockCycles += incrementPCValue;

        // Interrupts execution
        ExecuteInterrupts(incrementPCValue);

        // Check interrupts status
        if (GetIrqSignal())
        {
            HandleInterrupt();
        }

        // Increase instructions command
        TotalInstructionsCommand++;
    }
}
```

- רצים כל עוד לא עברנו את מספר הפקודה האחרונה.
- מביאים את הפקודה הבאה ברשימת הפקודות שלנו.
- אם בפקודה קיים ערך Immediate מעדכנים את רגיסטר 1.
- מעדכנים את הTrace
- מעדכנים את הPC.
- מבצעים את הפקודה.
- מטפלים בIO.
- מטפלים בפסיקות.
- מעדנים מחזורי שעון(לפי האם היה ערך Immediate או לא) ומעלים את מספר הפקודות שביצענו.

- Exit:

```
void Exit()
{
    // Write out files
    WriteMemoryToFile();
    WriteMonitorData();
    WriteCyclesToFile();
    WriteRegistersToFile();
    WriteDiskMemory();

    CloseFiles();

    // Free instructionCommand memory
    FreeInstructionsCommandArray();
    // Free Irq2Array
    FreeInterruptsMemory();
}
```

- מבצעים כתיבה של כל הקבצים וסוגרים אותם.
- מבצעים שחרור של כל הזכרונות בקובץ.