# CPU Architecture

# Task 3 Report File

עומר לוקסמבורג 205500390
עילי נוריאל 312538580

## מטרת המטלה

מטרת מטלה זו היא ביצוע תכנון, סינתזה וניתוח של CPU פשוט מסוג MIPS, עם שילוב של Mapped I/O, וכמו כן הבנת מבנה הזיכרון של רכיב ה- Cyclone II FPGA.

## הגדרת תכנון המערכת

ה-CPU אותו תכננו עובד בתצורת Single Cycle, ומבצע פקודות מה-ISA של ה-MIPS.

כמו כן ל-CPU יש register file סטנדרטי של MIPS, וה- top level entity מתוכנן בתצורת structural.

הפקודות שמימשנו הן: BEQ, BNQ, JUMP, SLL, SRL, AND, ANDI, OR, ORI, ADD, ADDI, ADDU, XOR, XORI, SLT והן תוכננו בהתאם לפורמט הפקודות המתואר ב-Table 1:

| Type | -31- | | | format (bits) | | -0- |
|---|---|---|---|---|---|---|
| R | opcode (6) | rs (5) | rt (5) | rd (5) | shamt (5) | funct (6) |
| I | opcode (6) | rs (5) | rt (5) | immediate (16) | | |
| J | opcode (6) | address (26) | | | | |

**Table 1 : MIPS Instruction format**

התכנון אותו ממשנו תואם לדיאגרמת הבלוקים המתוארת ב-Figure 1:

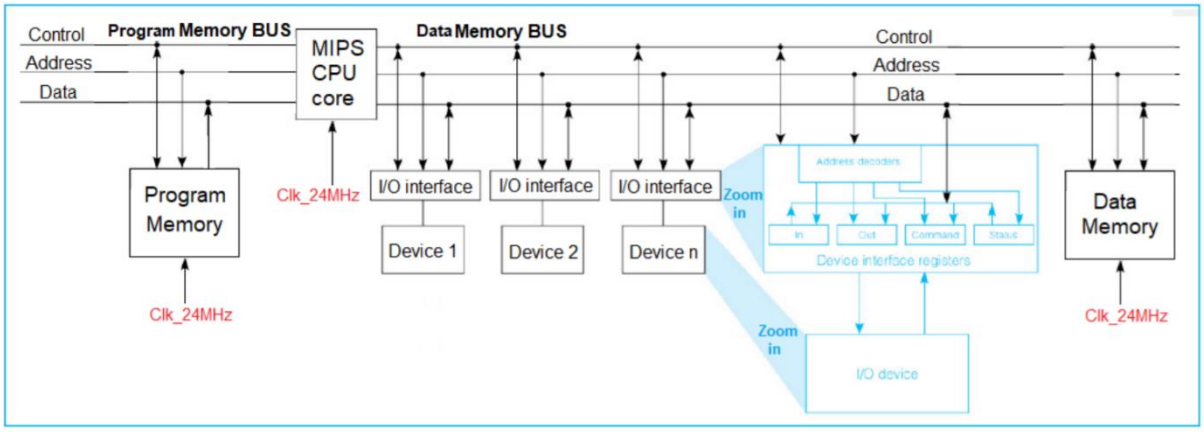



**Figure 1 : System architecture**

כמו כן התכנון שלנו מותאם ל- Mapped I/O המקושר למרחב הכתובות המתואר ב- Figure 2 :

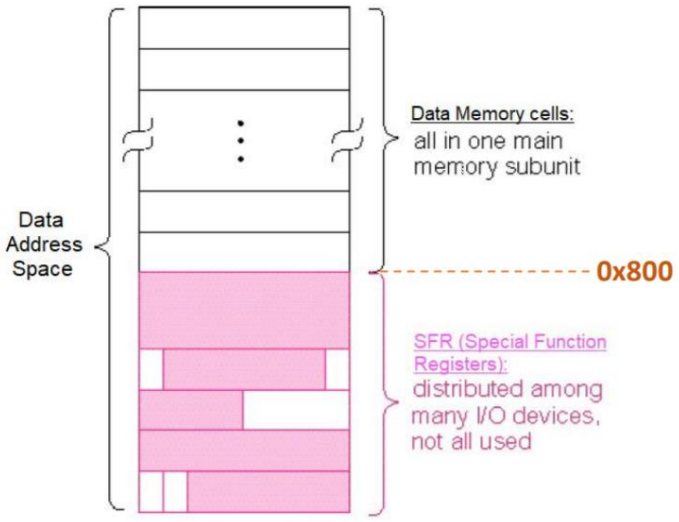



**Figure 2: Address Space of a Computer Using Memory Mapped I/O**

## בדיקת התכנון והמערכת

ניתוח עבודת המערכת מתבצע ע״י שני ה- test benchים הבאים :

1. היה עלינו לכתוב קוד אסמבלי (התואם ל-ISA של ה-MIPS) אשר ממיין מערך של 16 מספרים שלמים המאוחסן בזיכרון, ולהציג את המערך הממוין על גבי הלדים ותצוגת ה-7-Seg.
2. תמיכה והרצה תקינה של הקוד בקובץ ה-test.asm שניתן לנו (הצגת ערכי ה-switches על גבי הלדים והתצוגה והזזה של הערך שמאלה בכל מחזור).

```
#-------------------- MEMORY Mapped I/O -----------------------
#define PORT_LEDG[7-0] 0x800 - LSB byte (Output Mode)
#define PORT_LEDR[7-0] 0x804 - LSB byte (Output Mode)
#define PORT_HEX0[7-0] 0x808 - LSB byte (Output Mode)
#define PORT_HEX1[7-0] 0x80C - LSB byte (Output Mode)
#define PORT_HEX2[7-0] 0x810 - LSB byte (Output Mode)
#define PORT_HEX3[7-0] 0x814 - LSB byte (Output Mode)
#define PORT_SW[7-0]   0x818 - LSB byte (Input Mode)
#---------------------------------------------------------------
.data
    Array:  .word 7,6,2,3,12,16,1,4,5,9,8,13,14,10,11,15
    N:  .word 0x5B8D8   # 1.5Mhz clock, 4 commands per dleay = 375,000 times
.text
    addi $s1, $zero, 1       # $s1 = 1 - to compare with
    la   $t0, Array          # $t0 is the base address. "END" of the array

outLoop:                     # outLoop - used to know if we end sorting
    add  $t1, $zero, $zero   # $t1 = 0 when the list is sorted
    la   $s0, Array          # Set $s0 to base address.
    addi $s0, $s0, 60        # Set $s0 to last element, start sorting from there
inLoop:                      # inLoop - will iterate over the Array checking if a swap is needed
    lw   $t3, 0($s0)         # t3 = Array[i]
    lw   $t2, -4($s0)        # t2 = Array[i-1]
    slt  $t5, $t3, $t2       # $t5 = 1 if $t3 < $t2 A[i]< A[i-1]
    beq  $t5, $zero, continue# if $t5 = 1, then swap them, meaning DONT jump to continue
    addi $t1, $zero, 1       # If we have swapped, the array is NOT sorted yet. its sorted only when we didnt swap at all
    sw   $t3, -4($s0)        # store the bigger numbers contents in the higher position in array (swap)
    sw   $t2, 0($s0)         # store the smaller numbers contents in the lower position in array (swap)
continue:
    addi $s0, $s0, -4        # advance the array to start at the next location from last time
    slt  $t6, $t0, $s0       # $t6 = 1 if $t0 < $s0 that means $s0 is not the start of the Array
    beq  $t6, $s1, inLoop    # if $t6 = 1 (which is s1), than we didnt finish the current loop, $s0 != start of the Array
            #bne  $s0, $t0, inLoop    # If $s0 != the start of Array, jump back to inLoop
    beq  $t1, $s1, outLoop   # if $t1 = 1 (which is s1), another pass is needed, jump back to outLoop
            #bne  $t1, $zero, outLoop # $t1 = 1, another pass is needed, jump back to outLoop

# -------- LED SHOW ---------
    addi $s0, $s0, 0
    addi $s1, $s1, 16
    lw   $t3,N
```

```
    la   $t5,Array  # array start - $t5 is the address
Loop:   lw   $t0,0($t5) # $t0 has the value of M[$t5]
    sw   $t0,0x800  # write to PORT_LEDG[7-0]
    sw   $t0,0x804  # write to PORT_LEDR[7-0]
    sw   $t0,0x808  # write to PORT_HEX0[7-0]
    sw   $t0,0x80C  # write to PORT_HEX1[7-0]
    sw   $t0,0x810  # write to PORT_HEX2[7-0]
    sw   $t0,0x814  # write to PORT_HEX3[7-0]
    addi $s0,$s0,1   # $s0++
    beq  $s0,$s1,END # till the last element
    addi $t5,$t5,4  # updating $t5 to new address
    move $t1,$zero  # $t1=0
delay:  addi $t1,$t1,1  # $t1=$t1+1
    slt  $t2,$t1,$t3      #if $t1<N than $t2=1
    beq  $t2,$zero,Loop  #if $t1>=N then go to Loop label
    j    delay
END:    addi $t0,$zero,0
    sw   $t0,0x800  # write to PORT_LEDG[7-0]
    sw   $t0,0x804  # write to PORT_LEDR[7-0]
    sw   $t0,0x808  # write to PORT_HEX0[7-0]
    sw   $t0,0x80C  # write to PORT_HEX1[7-0]
    sw   $t0,0x810  # write to PORT_HEX2[7-0]
    sw   $t0,0x814  # write to PORT_HEX3[7-0]
    j    END
```

# ניתוח עבודת המערכת

- **תדר השעון המקסימלי** שקיבלנו הוא $380\ Mhz$:

**Slow Model Fmax Summary**

| | Fmax | Restricted Fmax | Clock Name | Note |
|---|---|---|---|---|
| 1 | 790.51 MHz | 380.08 MHz | clk_50Mhz | limit due to minimum period restriction (max I/O toggle rate) |

**Figure 3: Fmax analysis**

כפי שניתן לראות בהערה בעמודה הכי ימנית, ככל הנראה הקריאה מהמתגים מגבילה את מהירות המערכת. ייתכן כי נוכל לשפר את המהירות אם נשנה את אופן העבודה מול המתגים.

- **ניתוח המסלול הקריטי:**

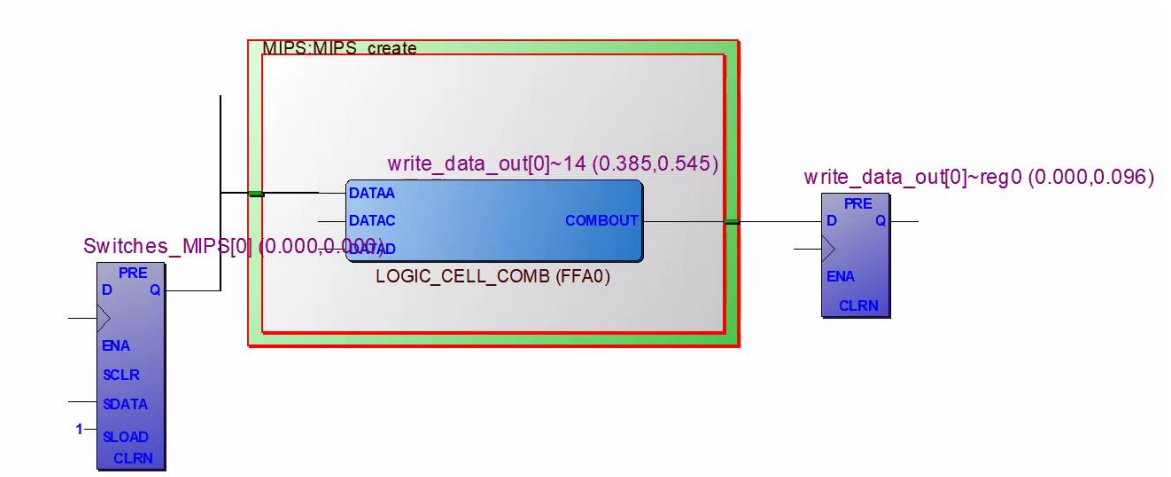



**Figure 4: Critical Path**

המסלול הקריטי הוא של קריאת הערכים מה-switchים (Load word), ככל הנראה בעקבות השהיות שנוצרות מהקריאה שלהם.

- **ניתוח המסלול המינימלי:**

אנו משערים כי המסלול הכי קצר המערכת הוא של אחת מפעולות ה-bitwise, OR או AND (XOR ככל הנראה מעט ארוכה יותר), זאת מכיוון שהן מבוצעות ע"י מימוש פשוט של שער לוגי וללא פעולות אריתמטיות מסובכות, וכמו כן הן נכתבות לרגיסטר ולא לזיכרון.
באופן כללי – כלל הפעולות מבצעות את 4 השלבים של המערכת (כאשר יש פעולות שמבצעות Write back לרגיסטרים ויש פעולות הכותבות לזיכרון) כלומר, אין פעולה קצרה משמעותית מפני שכולן עוברות בשלבי המערכת כפי שלמדנו על מעבד single cycle.

- **ניתוח השימוש הלוגי:**

של ה-MIPS כולו:

| Analysis & Synthesis Summary | |
|---|---|
| Analysis & Synthesis Status | Successful - Sun Jul 26 19:05:07 2020 |
| Quartus II 64-Bit Version | 12.1 Build 177 11/07/2012 SJ Web Edition |
| Revision Name | Project3 |
| Top-level Entity Name | MIPS |
| Family | Cyclone II |
| Total logic elements | 6,501 |
| Total combinational functions | 3,290 |
| Dedicated logic registers | 4,116 |
| Total registers | 4116 |
| Total pins | 230 |
| Total virtual pins | 0 |
| Total memory bits | 124,928 |
| Embedded Multiplier 9-bit elements | 0 |
| Total PLLs | 0 |

**Figure 5: MIPS Logic Usage**

Fetch:

| Analysis & Synthesis Summary | |
|---|---|
| Analysis & Synthesis Status | Successful - Sun Jul 26 19:56:04 2020 |
| Quartus II 64-Bit Version | 12.1 Build 177 11/07/2012 SJ Web Edition |
| Revision Name | Project3 |
| Top-level Entity Name | Ifetch |
| Family | Cyclone II |
| Total logic elements | 3,583 |
| Total combinational functions | 1,150 |
| Dedicated logic registers | 3,120 |
| Total registers | 3120 |
| Total pins | 66 |
| Total virtual pins | 0 |
| Total memory bits | 92,160 |
| Embedded Multiplier 9-bit elements | 0 |
| Total PLLs | 0 |

**Figure 6: Ifetch Logic Usage**

Decode:

| Analysis & Synthesis Summary | |
|---|---|
| Analysis & Synthesis Status | Successful - Sun Jul 26 20:01:25 2020 |
| Quartus II 64-Bit Version | 12.1 Build 177 11/07/2012 SJ Web Edition |
| Revision Name | Project3 |
| Top-level Entity Name | Idecode |
| Family | Cyclone II |
| Total logic elements | 5,961 |
| Total combinational functions | 2,573 |
| Dedicated logic registers | 4,104 |
| Total registers | 4104 |
| Total pins | 197 |
| Total virtual pins | 0 |
| Total memory bits | 59,392 |
| Embedded Multiplier 9-bit elements | 0 |
| Total PLLs | 0 |

**Figure 7: Idecode Logic Usage**

: Execute

| Analysis & Synthesis Summary | |
|---|---|
| Analysis & Synthesis Status | Successful - Sun Jul 26 20:07:24 2020 |
| Quartus II 64-Bit Version | 12.1 Build 177 11/07/2012 SJ Web Edition |
| Revision Name | Project3 |
| Top-level Entity Name | Execute |
| Family | Cyclone II |
| Total logic elements | 4,055 |
| Total combinational functions | 1,622 |
| Dedicated logic registers | 3,112 |
| Total registers | 3112 |
| Total pins | 165 |
| Total virtual pins | 0 |
| Total memory bits | 59,392 |
| Embedded Multiplier 9-bit elements | 0 |
| Total PLLs | 0 |

**Figure 8: Execute Logic Usage**

: Memory

| Analysis & Synthesis Summary | |
|---|---|
| Analysis & Synthesis Status | Successful - Sun Jul 26 20:05:31 2020 |
| Quartus II 64-Bit Version | 12.1 Build 177 11/07/2012 SJ Web Edition |
| Revision Name | Project3 |
| Top-level Entity Name | dmemory |
| Family | Cyclone II |
| Total logic elements | 3,567 |
| Total combinational functions | 1,134 |
| Dedicated logic registers | 3,112 |
| Total registers | 3112 |
| Total pins | 78 |
| Total virtual pins | 0 |
| Total memory bits | 92,160 |
| Embedded Multiplier 9-bit elements | 0 |
| Total PLLs | 0 |

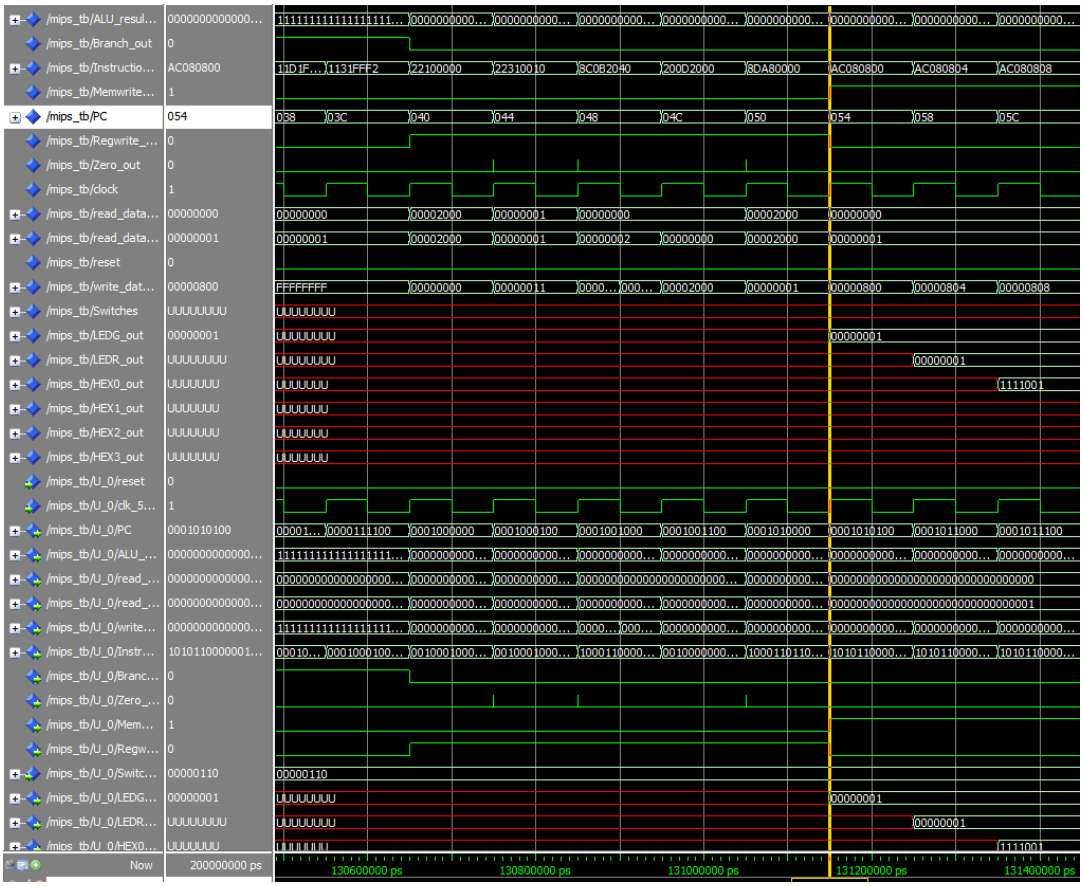
**Figure 9: dmemory Logic Usage**

: Control

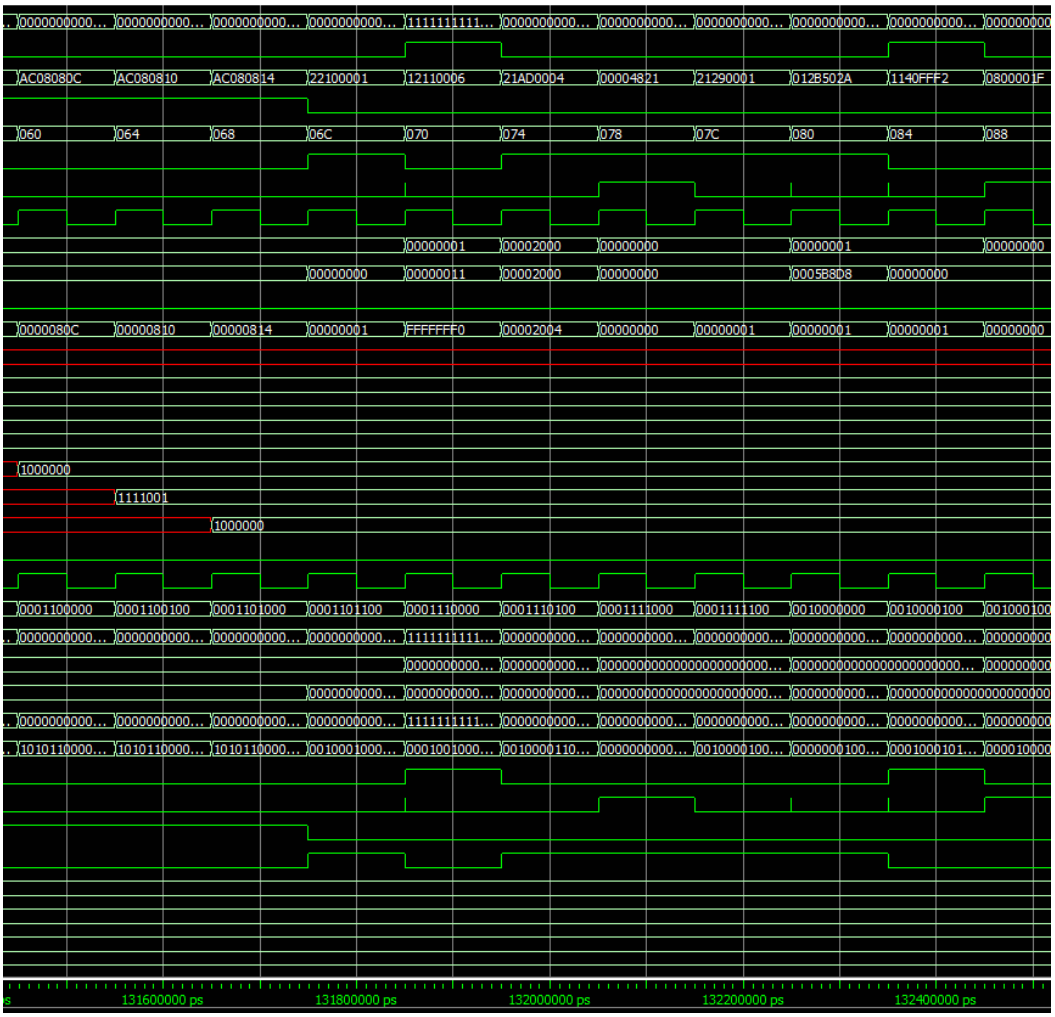
| Analysis & Synthesis Summary | |
|---|---|
| Analysis & Synthesis Status | Successful - Sun Jul 26 20:04:09 2020 |
| Quartus II 64-Bit Version | 12.1 Build 177 11/07/2012 SJ Web Edition |
| Revision Name | Project3 |
| Top-level Entity Name | control |
| Family | Cyclone II |
| Total logic elements | 3,580 |
| Total combinational functions | 1,147 |
| Dedicated logic registers | 3,112 |
| Total registers | 3112 |
| Total pins | 19 |
| Total virtual pins | 0 |
| Total memory bits | 59,392 |
| Embedded Multiplier 9-bit elements | 0 |
| Total PLLs | 0 |

**Figure 10: control Logic Usage**

- **Wave form: זוהי תצוגה של הקוד מיון**




**Figures 11-12: Wave form analysis**



הערך הקטן ביותר במערך שלנו הוא 1. ניתן לראות בתמונה _ (בסימון הצהוב) כי מתבצעת הפקודה:

```
42:      sw    $t0,0x800   # write to PORT_LEDG[7-0]
```

ואכן ניתן לראות כי ערך ה-LEDG_ OUT משתנה ל-1, כלומר הLEDים יציגו את המספר 1.

כמו כן מצורפת תמונה של הערכים בזיכרון לאחר המיון:
הערך האחרון הוא ערך לחישוב הזמן בכדי להמתין שניה אחת בין השינוי תצוגה ב-LEDים – חשוב ע"י תדר השעון בו אנו משתמשים (לאחר חלוקה) ולפי מספר הפקודות עבור ביצוע מחזור השהייה אחד.

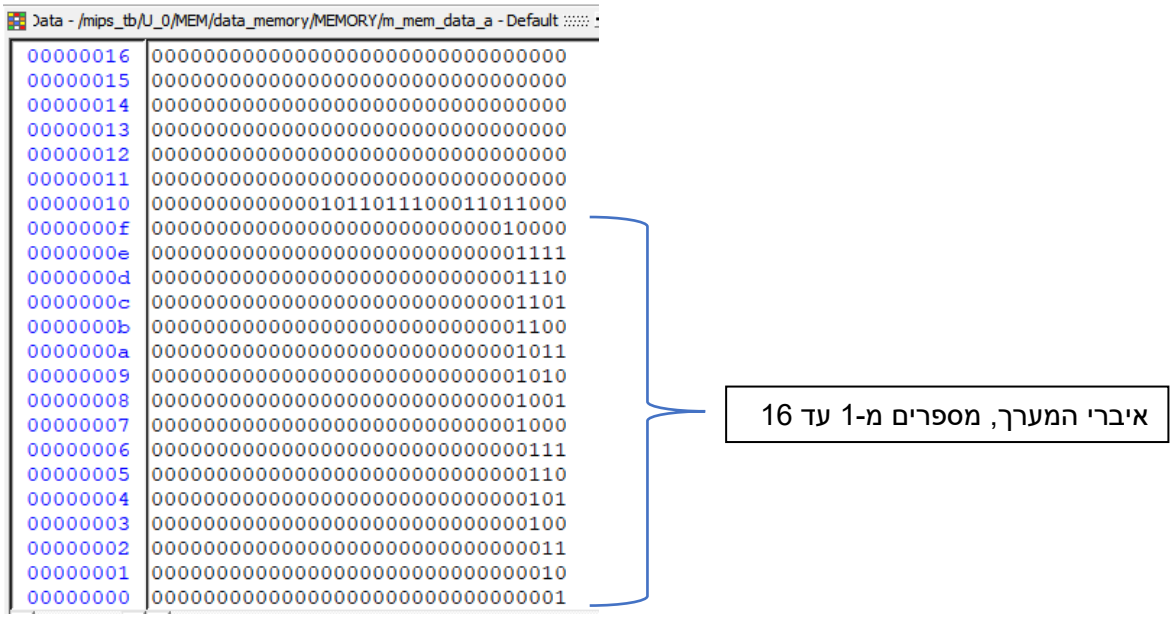



**Figure 13: Memory view**

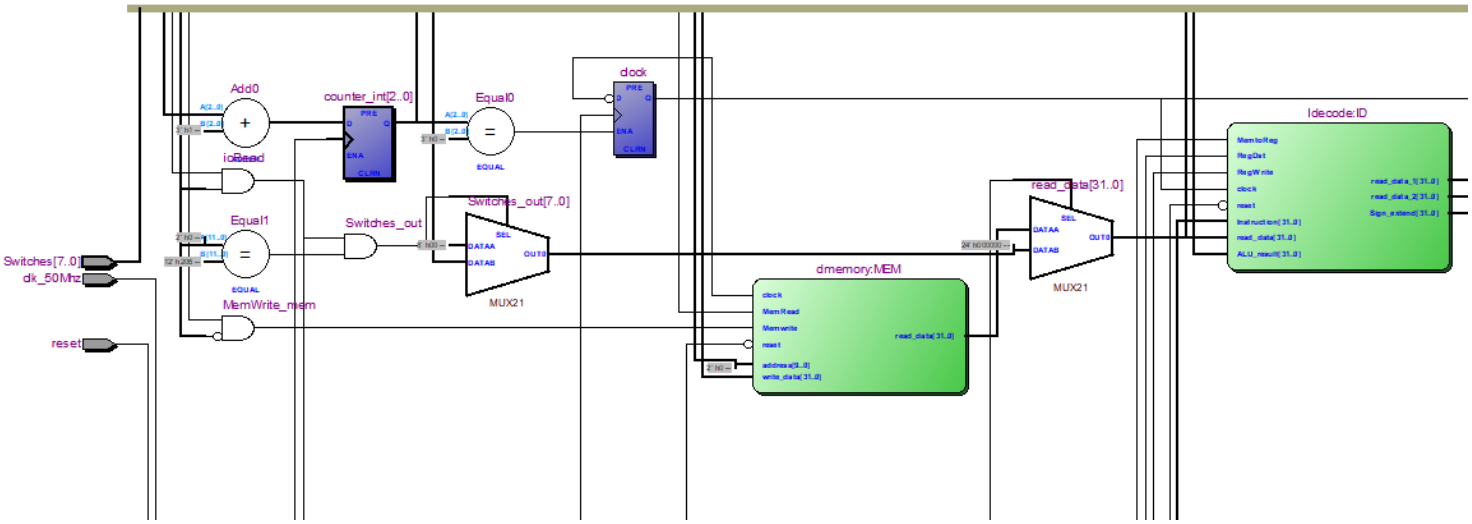
- **RTL viewer:**
  ניתן להבחין ביחידות לוגיות המטפלות בניתוב הקריאה והכתיבה בין כתובות זיכרון רגילות לבין כתובות זיכרון הממופות ל- I\O:

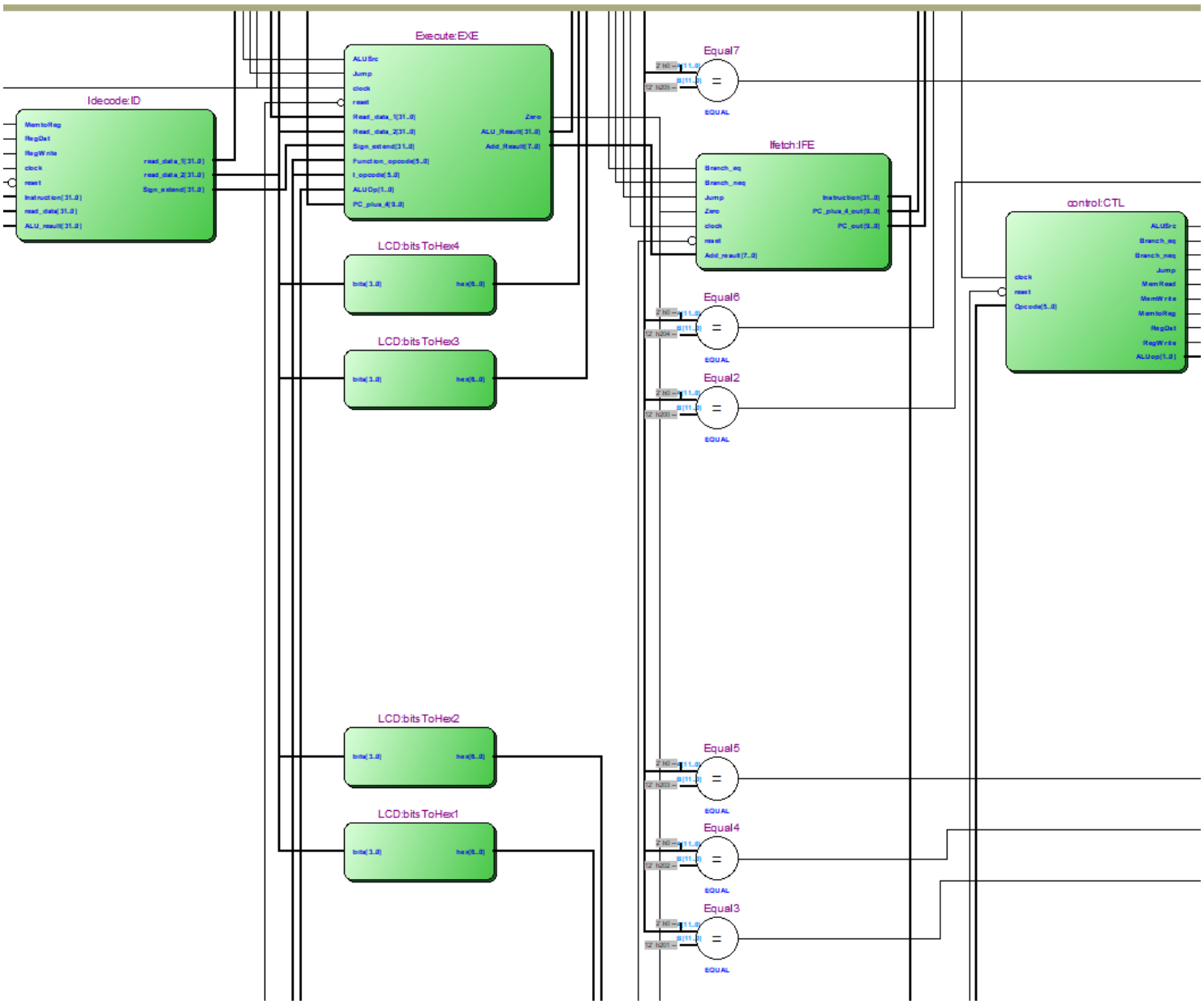
```
-- IO / MEM
MemWrite_mem <= MemWrite AND (NOT(ALU_Result(11)));          -- Writing to mem / IO
--MemRead_mem    <= MemRead AND (NOT(ALU_Result(11)));
ioWrite      <= MemWrite AND ALU_Result(11);                 -- Writing to IO - see after components
ioRead       <= MemRead AND ALU_Result(11);
-- Added this line to get the switches value into register - memory >= 800h
read_data <= X"000000" & Switches_out WHEN ioRead = '1' -- read_data from IO switches
        ELSE read_data_tmp;                              -- normal read_data from DMEMORY
-- temp signal for ALU_res to address:
Address_ALU_res <= ALU_Result(9 DOWNTO 2) & "00";
```

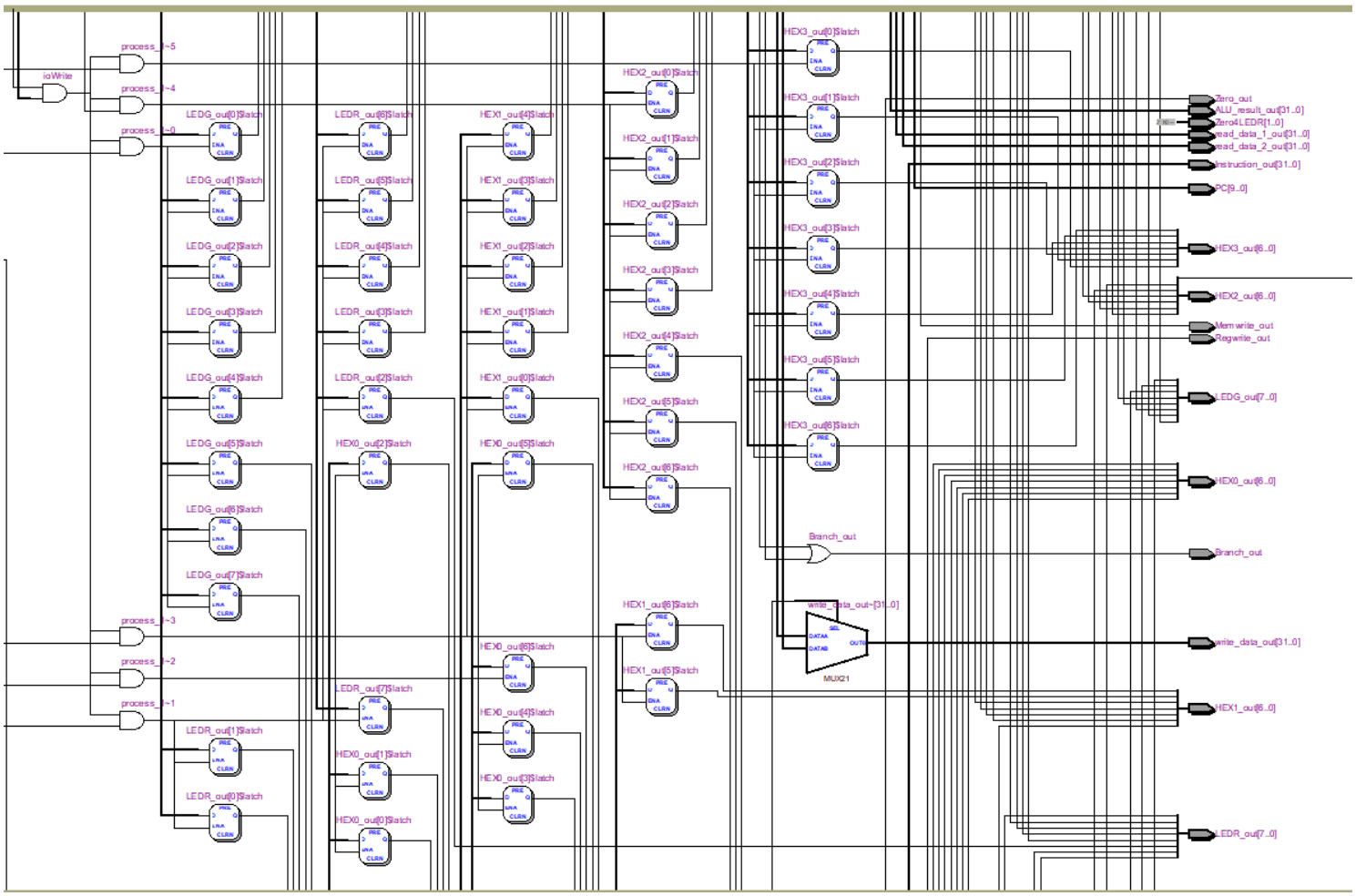
**Figure 14: Sorting Assembly Code**

ניתן לראות כי הוספנו קווי בקרה בעזרתם אנו יודעים האם פונים ל- I\O.

**בשני העמודים הבאים ניתן לראות את דיאגרמת ה-RTL המלאה של המערכת, חילקנו אותה ל-3 תמונות. במקור התמונות מחוברות לאורך.**


Add0
counter_int[2..0]
Equal0
clock
Idecode:ID
ioRead
Equal1
Switches_out
Switches_out[7..0]
read_data[31..0]
Switches[7..0]
dk_50Mhz
MemWrite_mem
reset
dmemory:MEM
MUX21
MUX21


Execute:EXE
Idecode:ID
Equal7
Ifetch:IFE
control:CTL
LCD:bits ToHex4
LCD:bits ToHex3
Equal6
Equal2
LCD:bits ToHex2
LCD:bits ToHex1
Equal5
Equal4
Equal3




**Figures 15-17: RTL viewer**

להלן הבלוקים המתוארים בדיאגרמת ה-RTL בפירוט גדול יותר – כולל כניסות ויציאות, בהתאם לדיאגרמה המתארת את המערכת.

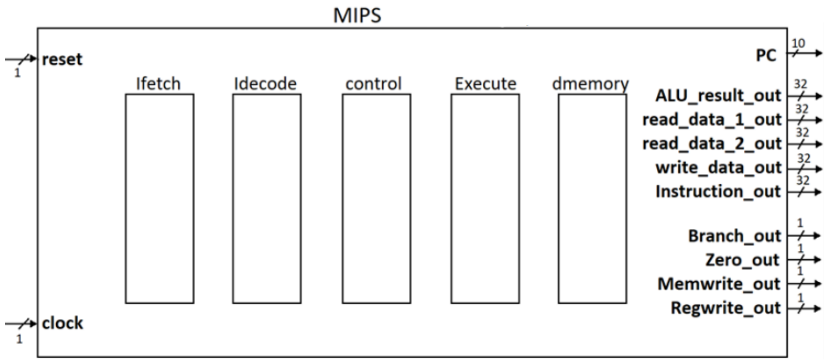



**Figure 18: Top Block Diagram**

כמו כן ניתן לראות כי הוספנו קווי בקרה ושינינו חלק מלוגיקת הבלוק Execute – לדוגמא הוספנו קווי בקרה ל-BEQ, BNQ, Jump. כמו כן כפי שהסברנו בעמוד הקודם, הוספנו התמודדות עם כתיבה לכתובות בזיכרון הממופות ל-I\O לעומת כתובות רגילות בזיכרון.

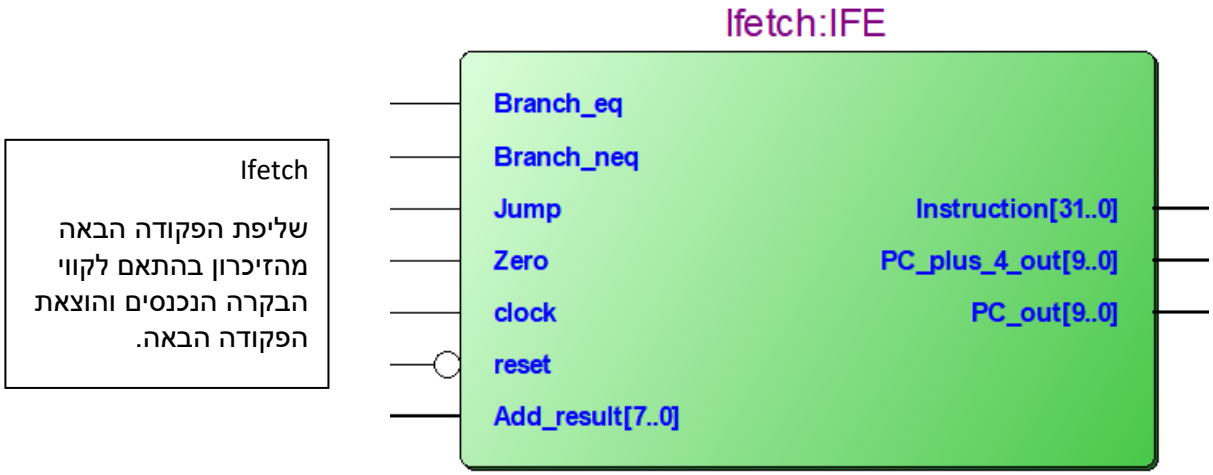



**Figure 19: Ifetch block**

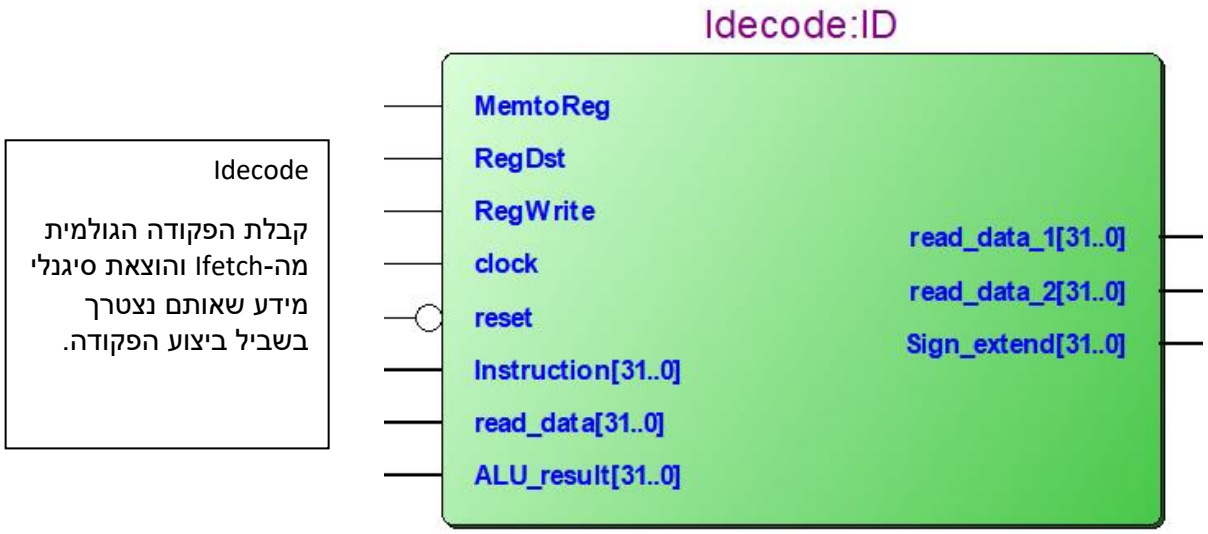



**Figure 20: Idecode block**

Execute

ביצוע הפקודה עצמה, כלומר פעולה אריתמטית (מתוך אחת מהפעולות הנתמכות במעבד). והוצאת תוצאה ודגלים מתאימים.

Add_Result – תוצאה המייצגת כתובת ולכן הוא רק באורך של שמונה ביטים.

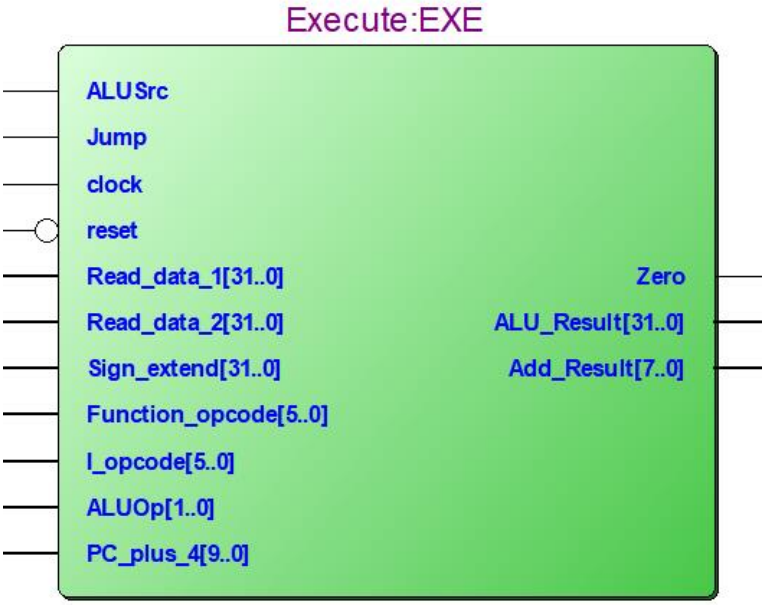



**Figure 21: Execute block**

dmemory

מבצע קריאה או כתיבה לזכרון, מלבד לכתובות הממופות לרכיבי ה-O\I

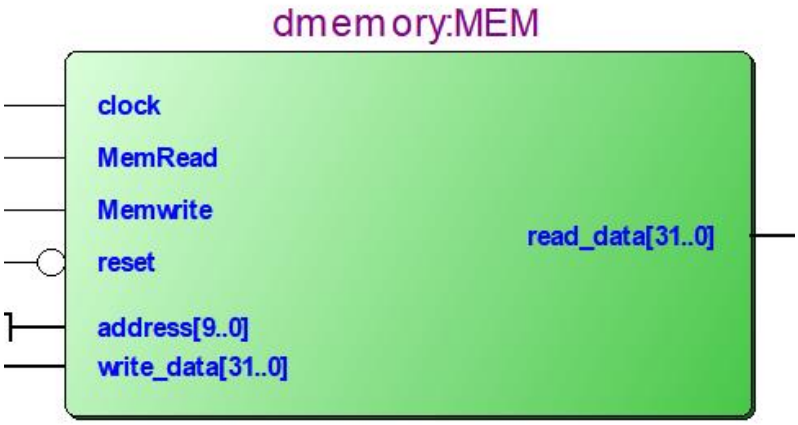



**Figure 22: dmemory block**

control

הוצאת קווי הבקרה המתאימים לפקודה (לפי ה-opcode המגיע מה-fetch)

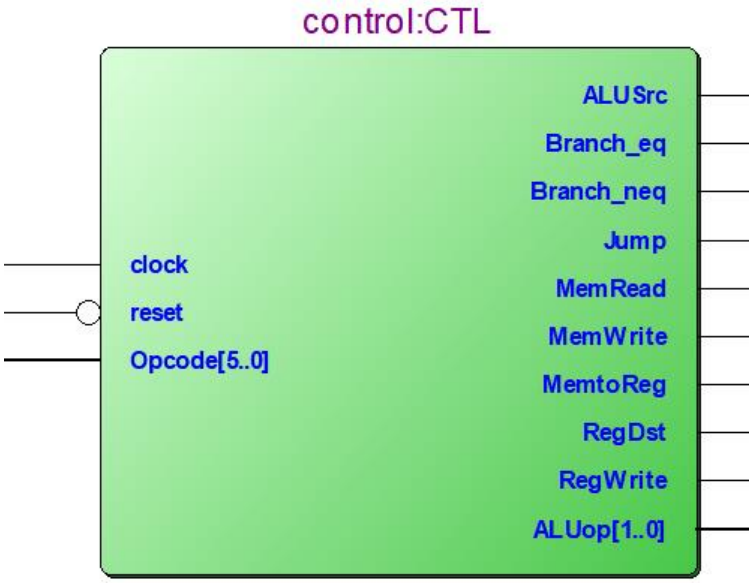



**Figure 23: control block**

- **Port Table:** מצורפות טבלאות המתארות את הגודל, כיוון ופונקציונאליות של הקווים היוצאים מכל בלוק במערכת

| MIPS | | | |
|---|---|---|---|
| Port | Direction | Size | Functionality |
| reset | in | 1 | |
| clk_50Mhz | in | 1 | * We used 24Mhz clock. |
| PC | out | 10 | |
| ALU_result_out | out | 32 | |
| read_data_1_out | out | 32 | |
| read_data_2_out | out | 32 | |
| write_data_out | out | 32 | |
| Instruction_out | out | 32 | |
| Branch_out | out | 1 | |
| Zero_out | out | 1 | |
| Memwrite_out | out | 1 | |
| Regwrite_out | out | 1 | |
| Switches | in | 8 | |
| LEDG_out | buffer | 8 | |
| LEDR_out | buffer | 8 | |
| HEX0_out | buffer | 7 | |
| HEX1_out | buffer | 7 | |
| HEX2_out | buffer | 7 | |
| HEX3_out | buffer | 7 | |
| HEX4_out | buffer | 7 | |
| Zero4LEDR | out | 2 | To initiate LEDR 8 and 9 to 0 value. |

**Table 2: MIPS ports**

| IFETCH | | | |
|---|---|---|---|
| Port | Direction | Size | Functionality |
| Instruction | out | 32 | |
| PC_plus_4_out | out | 10 | The next command address |
| Add_result | in | 8 | Address value to branch / jump to |
| Branch_eq | in | 1 | Control signal indicates BEQ command |
| Branch_neq | in | 1 | Control signal indicates BNQ command |
| Zero | in | 1 | |
| Jump | in | 1 | Control signal indicates Jump command |
| PC_out | out | 10 | |
| clock | in | 1 | |
| reset | in | 1 | |

**Table 3: Ifetch ports**

| IDECODE | | | |
|---|---|---|---|
| Port | Direction | Size | Functionality |
| read_data_1 | out | 32 | Data #1 to use in execute |
| read_data_2 | out | 32 | Data #2 to use in execute |
| Instruction | in | 32 | |
| read_data | in | 32 | |
| ALU_result | in | 32 | |
| RegWrite | in | 1 | |
| MemtoReg | in | 1 | |
| Reg_dst | in | 1 | |
| Sign_extend | out | 32 | Sign extended immediate |
| clock | in | 1 | |
| reset | in | 1 | |

**Table 4: Idecode ports**

| CONTROL | | | |
|---|---|---|---|
| Port | Direction | Size | Functionality |
| Opcode | in | 6 | |
| RegDst | out | 1 | What register to write to |
| ALUSrc | out | 1 | Binput selector |
| MemtoReg | out | 1 | |
| RegWrite | out | 1 | |
| MemRead | out | 1 | Read control |
| MemWrite | out | 1 | Write control |
| Branch_eq | out | 1 | |
| Branch_neq | out | 1 | |
| ALUop | out | 2 | |
| Jump | out | 1 | |
| clock | in | 1 | |
| reset | in | 1 | |

**Table 5: Control ports**

| Execute | | | |
|---|---|---|---|
| Port | Direction | Size | Functionality |
| Read_data_1 | in | 32 | Data #1 to read from |
| Read_data_2 | in | 32 | Data #2 to read from |
| Sign_extend | in | 32 | Sign extended immediate |
| Function_opcode | in | 6 | For “0” opcode |
| I_opcode | in | 6 | Opcode number |
| ALUOp | in | 2 | With Function_opcode, I_opcode – decides which ALU operation will be executed |
| Jump | in | 1 | |
| ALUSrc | in | 1 | |
| Zero | out | 1 | |
| ALU_Result | out | 32 | The result of the ALU (shift included) |
| Add_Result | out | 8 | |
| PC_plus_4 | in | 9 | |
| Clock | in | 1 | |
| Reset | in | 1 | |

**Table 6: Execute ports**

| DMEMORY | | | |
|---|---|---|---|
| Port | Direction | Size | Functionality |
| read_data | out | 32 | Memory data read |
| address | in | 10 | Address number in |
| write_data | in | 32 | Data to write |
| MemRead | in | 1 | ONLY for memory read (no I/O) |
| Memwrite | in | 1 | ONLY for memory write (no I/O) |
| clock | in | 1 | |
| reset | in | 1 | |

**Table 7: Dmemory ports**

| I/O Memory (ports in MIPS.vhdl file) | | | |
|---|---|---|---|
| Port | Direction | Size | Functionality |
| MemWrite_mem | signal | 1 | ‘1’ when Memwrite=’1’ and ALU_result is **not** the address of the I/O. |
| ioWrite | signal | 1 | ‘1’ when Memwrite=’1’ and ALU_result **is** the address of the I/O. |
| ioRead | signal | 1 | ‘1’ when MemRead=’1’ and ALU_result **is** the address of the I/O. |
| read_data | signal | 32 | Out will be from switches, when ioRead=’1’, else it will be “ DMEMORY - read_data” signal. |

**Table 8: I/O memory ports**

- **Proof of work – Signal Tap Screenshots:**

של קוד המיון אותו כתבנו :

| Address | Code | Basic | Source |
|---|---|---|---|
| 0x00000000 | 0x20110001 | addi $17,$0,0x00000001 | 14: addi $s1, $zero, 1 # $s1 = 1 - to compare with |
| 0x00000004 | 0x20082000 | addi $8,$0,0x00002000 | 15: la $t0, Array # $t0 is the base address. "END" of the array |
| 0x00000008 | 0x00004820 | add $9,$0,$0 | 17: add $t1, $zero, $zero # $t1 = 0 when the list is sorted |
| 0x0000000c | 0x20102000 | addi $16,$0,0x00002000 | 18: la $s0, Array # Set $s0 to base address. |
| 0x00000010 | 0x2210003c | addi $16,$16,0x0000... | 19: addi $s0, $s0, 60 # Set $s0 to last element, start sorting from there |
| 0x00000014 | 0x8e0b0000 | lw $11,0x00000000($16) | 21: lw $t3, 0($s0) # t3 = Array[i] |
| 0x00000018 | 0x8e0afffc | lw $10,0xfffffffc($16) | 22: lw $t2, -4($s0) # t2 = Array[i-1] |
| 0x0000001c | 0x016a682a | slt $13,$11,$10 | 23: slt $t5, $t3, $t2 # $t5 = 1 if $t3 < $t2 A[i]< A[i-1] |
| 0x00000020 | 0x11a00003 | beq $13,$0,0x00000003 | 24: beq $t5, $zero, continue# if $t5 = 1, then swap them, meaning DONT jump to continue |
| 0x00000024 | 0x20090001 | addi $9,$0,0x00000001 | 25: addi $t1, $zero, 1 # If we have swapped, the array is NOT sorted yet. its sorted only when we didnt swap at all |
| 0x00000028 | 0xae0bfffc | sw $11,0xfffffffc($16) | 26: sw $t3, -4($s0) # store the bigger numbers contents in the higher position in array (swap) |
| 0x0000002c | 0xae0a0000 | sw $10,0x00000000($16) | 27: sw $t2, 0($s0) # store the smaller numbers contents in the lower position in array (swap) |
| 0x00000030 | 0x2210fffc | addi $16,$16,0xffff... | 29: addi $s0, $s0, -4 # advance the array to start at the next location from last time |
| 0x00000034 | 0x0110702a | slt $14,$8,$16 | 30: slt $t6, $t0, $s0 # $t6 = 1 if $t0 < $s0 that means $s0 is not the start of the Array |
| 0x00000038 | 0x11d1fff6 | beq $14,$17,0xfffffff6 | 31: beq $t6, $s1, inLoop # if $t6 = 1 (which is s1), than we didnt finish the current loop, $s0 != start of the Array |
| 0x0000003c | 0x1131fff2 | beq $9,$17,0xfffffff2 | 33: beq $t1, $s1, outLoop # if $t1 = 1 (which is s1), another pass is needed, jump back to outLoop |

**Figure 24: Assembly Sorting Code**

ניתן לראות כי הפקודה הראשונה שמופיעה לנו ב-SIGNAL TAP היא אכן הפקודה הראשונה בקוד האסמבלי שלנו. בה נכתב לרגיסטר $17 הערך 1 בעזרת פקודת addi.

ניתן לראות כי הפקודה הזו היא פעולת ה-add בה מכניסים לרגיסטר $9 את הערך 0+0 ולכן גם נקבל Zero_out עם ערך 1.

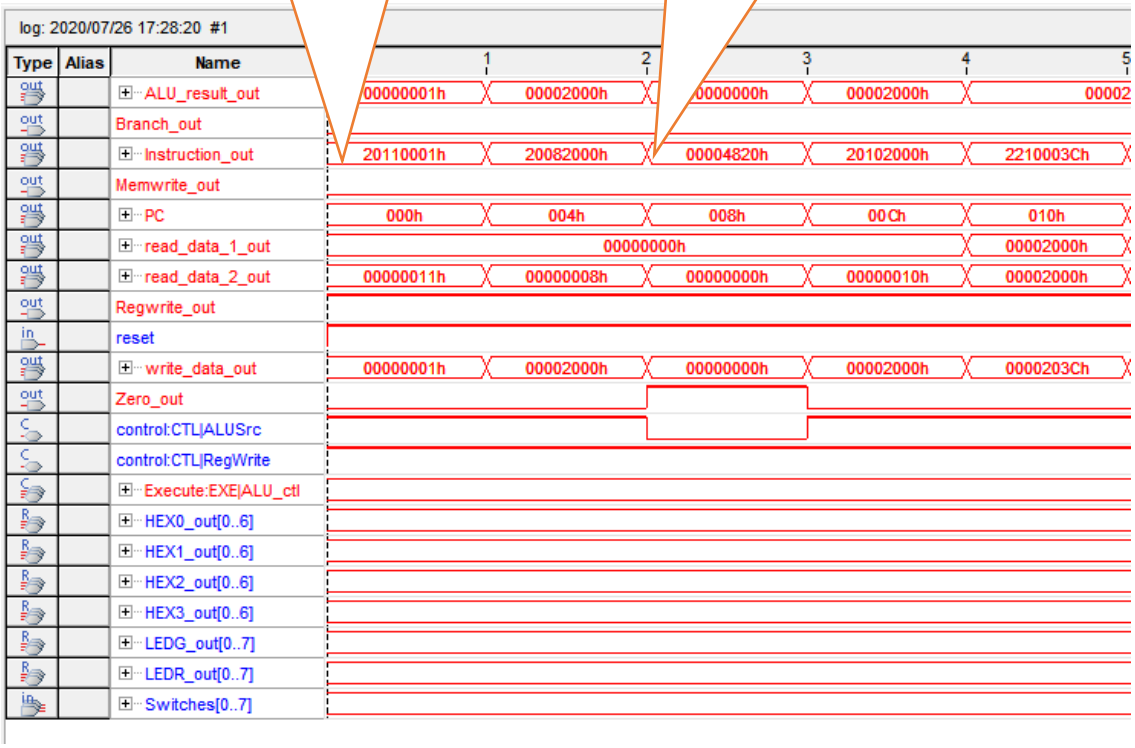



**Figure 25: Signal Tap 1**

בהמשך ריצת התוכנית, לאחר מיון המערך, אנו מגיעים לשלב בו אנו מזינים את הערכים לזיכרון המתאים ל-I/O עליו אנו רוצים להציג את המספרים:

בפקודה זו (שורה 42 בקוד שלנו) אנו רושמים את הערך שבמקום הראשון במערך ל-LEDG הממופה לכתובת 800h (ניתן לראות זאת ב-write_data_out)

בפקודה זו (שורה 44 בקוד שלנו) אנו רושמים את הערך שבמקום הראשון במערך ל-7-Seg הכי ימני הממופה לכתובת 808h (ניתן לראות זאת ב-write_data_out)

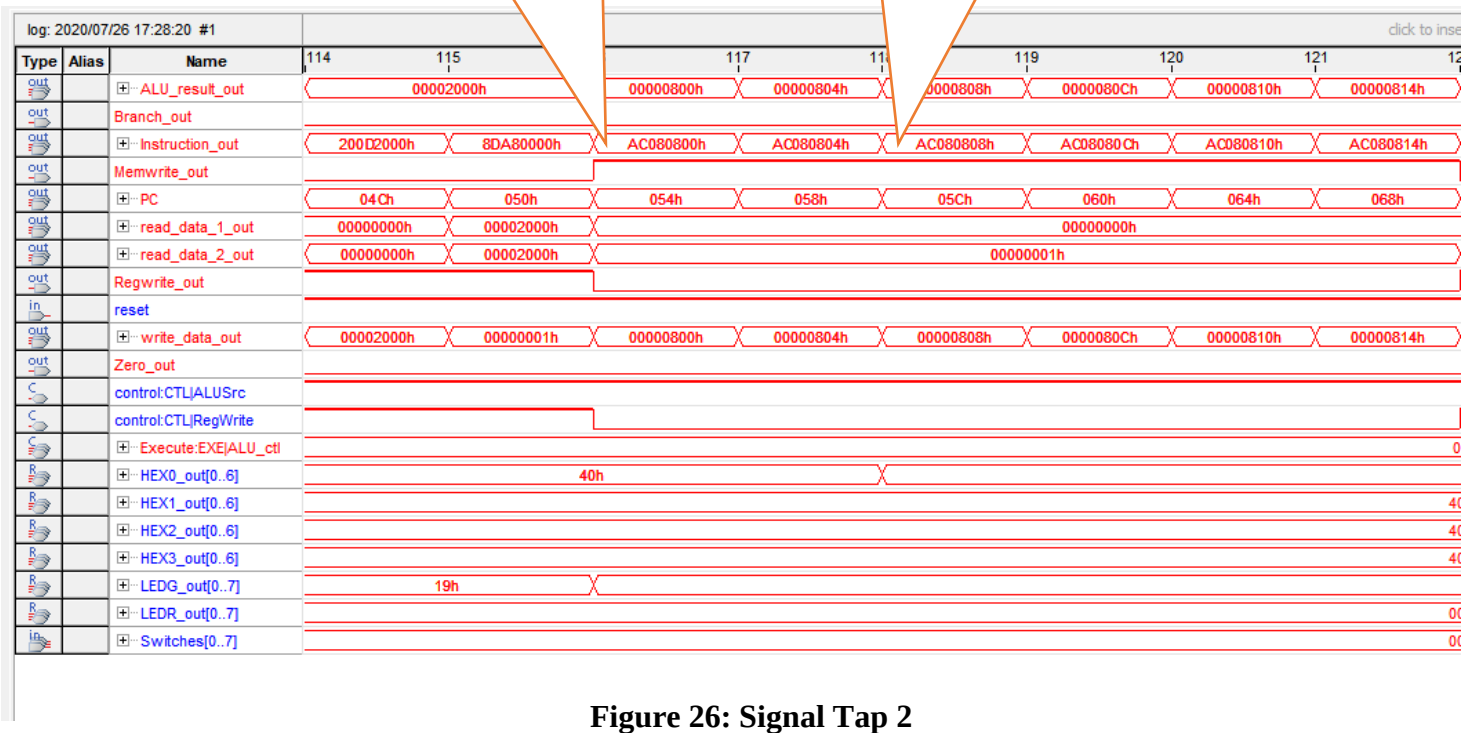



**Figure 26: Signal Tap 2**