

Procesor MIPS Ciclu unic

RAPORT

Student: Dunca Denisa Mihaela

Grupa: 30227

Prof. îndrumător: Constantin Nandra

Cele 4 instrucțiuni alese: multu, xor, ori, bltz

MULTU – unsigned multiplication

Descriere: înmulțește două registre și pune rezultatul în al treilea

Operație: $d<- $s \* $t; PC <-PC + 1;

Sintaxa: multu $d, $s, $t

Format: 000\_sss\_ttt\_ddd\_0\_110

Ex: multu $1, $2, $3 cod mașina: 000\_010\_011\_001\_0\_110

XOR – logical XOR

Descriere: SAU exclusiv logic între doua registre, memorează rezultatul în alt registru

Operație: $d<- $s ^ $t; PC <-PC + 1;

Sintaxa: xor $d, $s, $t

Format: 000\_sss\_ttt\_ddd\_0\_111

Ex: xor $1, $2, $3 cod mașina: 000\_010\_011\_001\_0\_111

ORI– or immediat

Descriere: SAU logic între un registru registre și o valoare imediată, memorează rezultatul în alt registru

Operație: $t<- $s | imm; PC <-PC + 1;

Sintaxa: ori $t, $s, imm

Format: 000\_sss\_ttt\_iiiiiii

Ex: ori $1, $2, 1 cod mașina: 000\_010\_001\_0000001

BLTZ -Branch on less than zero

Descriere: Salt condiționat dacă valoarea dintr-un registru este mai mică decât 0

Operație: if $s > 0 PC <- PC + 1 + offset ; else PC <- PC + 1;

Sintaxa: bltz $s, offset

Format: 111\_sss\_000\_iiiiiii

Ex: bltz $1, 1 cod mașina: 111\_001\_000\_0000001

1. Tabelul cu valorile semnalelor de control pentru toate instrucțiunile:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instr | Opcode | RegDst | ExtOp | ALUSrc | Branch | BrLTZ | Jump | Mem Write | Memto  Reg | Reg  Write | AluOp  (2:0) | func | AluCtrl  (2:0) |
| add | 000 | 1 | x | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 000 | 000 | 000 (+) |
| sub | 000 | 1 | x | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 000 | 001 | 001 (-) |
| sll | 000 | 1 | x | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 000 | 010 | 010 (<<l) |
| srl | 000 | 1 | x | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 000 | 011 | 011 (>>l) |
| and | 000 | 1 | x | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 000 | 100 | 100 (&) |
| or | 000 | 1 | x | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 000 | 101 | 101 (|) |
| multu | 000 | 1 | x | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 000 | 110 | 110 (\*) |
| xor | 000 | 1 | x | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 000 | 111 | 111 (^) |
| addi | 001 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 001 | X | 000 (+) |
| lw | 010 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 010 | X | 000 (+) |
| sw | 011 | x | 1 | 1 | 0 | 0 | 0 | 1 | x | 0 | 011 | X | 000 (+) |
| beq | 100 | x | 0 | 0 | 1 | 0 | 0 | 0 | x | 0 | 100 | X | 001 (-) |
| ori | 101 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 101 | X | 101 (|) |
| bltz | 111 | x | 0 | 0 | 1 | 1 | 0 | 0 | x | 0 | 111 | X | 001 (-) |
| j | 110 | x | x | x | x | x | 1 | 0 | x | 0 | 110 | X | 000 (+) |

c.

Pentru codul în C am ales să fac o buclă în funcție de i care este o variabilă dată de noi, în cazul de față este -2. Bucla am reprezentat-o printr-un while care verifică dacă i -ul dat este mai mic decât 0. În cazul în care este mai mic atunci intră în while și execută operațiile de acolo. Am creat o variabilă numită rez în care îmi tot adaug la fiecare iterație rezultatul unei expresii în fucție de I, iar apoi incrementez i-ul.

Când i -ul ajunge să fie pozitiv, se iese din buclă și se afișează rezultatul final.

#include <stdio.h>

int main(){

int rez = 0;

int i = -2;

while (i < 0){

rez = rez + (6 \* 3 - i) / 2;

i++;

}

printf("%d\n", rez);

}

xor $0, $0, $0 ------initializez reg 0 cu 0 - 000\_000\_000\_000\_0\_111

addi $1, $0, 0 ------rez - 001\_000\_001\_0000000

addi $2, $0, -2 ---- i - 001\_000\_010\_1111110

bltz $2, 10 - 111\_010\_000\_0001010

addi $3, $0, 6 -----in reg 3 am 6 - 001\_000\_011\_0000110

addi $4, $0, 3 -----in reg 4 am 3 - 001\_000\_100\_0000011

addi $5, $0, 2 -----in reg 5 am 2 - 001\_000\_101\_0000010

lw $6, 0($3) ------pun 6 in reg 6 - 010\_011\_110\_0000000

multu $6,$6,$4 ----inmultesc ce am in reg 6 cu 3 - 000\_110\_100\_110\_0\_110

sub $6, $6, $2 -----scad din reg 6 pe i - 000\_110\_010\_110\_0\_001

srl $6, $6, 0 -----impart reg 6 la 2 - 000\_000\_110\_110\_0\_011

add $1, $1, $6 -----adun la rezultat ce am in reg 6 - 000\_001\_110\_001\_0\_000

addi $2, $2, 1 ------incrementez i cu 1 - 001\_010\_010\_0000001

jmp 4 - 110\_0000000000100

sw $7, 0($1) ----- dau store in reg 7 la rez - 011\_001\_111\_0000000

d.

0. xor $0, $0, $0

instr = 000 ; PC = 1 ; RD1 = x; RD2 = x ; WD = 0 ; EXT = x; ALUres = 0 ; MEMDATA = 0;

1. addi $1, $0, 0

instr = 001; PC = 2; RD1 = 0; RD2 = x; WD = 0; EXT = 1; ALUres = 0; MEMDATA =0;

1. addi $2, $0, -2

instr = 001; PC = 3; RD1 = 0; RD2 = x; WD = -2 ; EXT =1 ; ALUres = -2 ; MEMDATA =0;

--- Iterația unu ---

1. bltz $2, 10

instr = 111 ; PC = 4; RD1 = -2; RD2 = 0; WD = -2; EXT = 0 ; ALUres = -2 ; MEMDATA =0;

1. addi $3, $0, 6

instr = 001; PC = 5; RD1 = 0; RD2 = x; WD = 6 ; EXT = 1; ALUres = 6; MEMDATA =0;

1. addi $4, $0, 3

instr = 001 ; PC = 6; RD1 = 0; RD2 = x; WD = 3; EXT = 1 ; ALUres = 3 ; MEMDATA =0;

1. addi $5, $0, 2

instr = 001; PC = 7; RD1 = 0; RD2 = x; WD = 2 ; EXT = 1; ALUres = 2; MEMDATA =0;

1. lw $6, 0($3)

instr = 010 ; PC = 8; RD1 = 6; RD2 = x; WD = 6 ; EXT = 1; ALUres = 6; MEMDATA =0;

1. multu $6,$6,$4

instr = 000; PC = 9; RD1 = 6; RD2 = 3; WD = 18; EXT = x; ALUres = 18; MEMDATA =0;

1. sub $6, $6, $2

instr = 000; PC = 10; RD1 = 18; RD2 = -2; WD = 20; EXT = x; ALUres = 20; MEMDATA=0 ;

1. srl $6, $6, 0

instr = 000; PC = 11; RD1 = 0; RD2 = 20; WD = 10; EXT = x; ALUres = 10 ; MEMDATA =0;

1. add $1, $1, $6

instr = 000 ; PC = 12; RD1 = 0 ; RD2 =10 ; WD = 10; EXT =x ; ALUres = 10; MEMDATA =0;

1. addi $2, $2, 1

instr = 001; PC = 13; RD1 = -2; RD2 = 1; WD = -1 ; EXT = 1; ALUres = -1; MEMDATA =0;

1. jmp 4

instr = 110; PC = 3; RD1 = 0; RD2 = 0; WD = 0; EXT = x; ALUres = 0; MEMDATA =0;

---Iteratia a doua ---

3. bltz $2, 10

instr = 111 ; PC = 4; RD1 = -1; RD2 = 0; WD = -1 ; EXT =0 ; ALUres = -1 ; MEMDATA =0;

4. addi $3, $0, 6

instr = 001; PC = 5; RD1 = 0; RD2 = x; WD = 6; EXT =1 ; ALUres = 6; MEMDATA =0;

5. addi $4, $0, 3

instr = 001 ; PC = 6; RD1 = 0; RD2 = x; WD = 3; EXT =1 ; ALUres = 3 ; MEMDATA =0;

6. addi $5, $0, 2

instr = 001; PC = 7; RD1 = 0; RD2 = x; WD = 2; EXT = 1; ALUres = 2; MEMDATA =0;

7. lw $6, 0($3)

instr = 010 ; PC = 8; RD1 = 6; RD2 = x; WD =0 ; EXT =1 ; ALUres = 6; MEMDATA =0;

8. multu $6,$6,$4

instr = 000; PC = 9; RD1 = 6; RD2 = 3; WD = 18; EXT = x; ALUres = 18; MEMDATA =0;

9. sub $6, $6, $2

instr = 000; PC = 10; RD1 = 18; RD2 =-1 ; WD =19 ; EXT =x ; ALUres = 19; MEMDATA =0;

10. srl $6, $6, 0

instr = 000; PC = 11; RD1 = 0; RD2 = 19; WD = 9 ; EXT =x ; ALUres = 9 ; MEMDATA =0;

11. add $1, $1, $6

instr = 000 ; PC = 12; RD1 = 1 ; RD2 =9 ; WD = 19; EXT =x ; ALUres = 19; MEMDATA =0;

12. addi $2, $2, 1

instr = 001; PC = 13; RD1 = -1; RD2 = 1; WD = 0; EXT = 1; ALUres = 0; MEMDATA =0;

1. jmp 4

instr = 110; PC = 3; RD1 = 0; RD2 = 0; WD =0 ; EXT =x ; ALUres = 0; MEMDATA =0;

---Iterația a treia---

3. bltz $2, 10

instr = 111 ; PC = 14 ; RD1 = 0; RD2 = 0; WD = 0; EXT =0 ; ALUres = 0 ; MEMDATA =0;

---Ieșire din bucla---

1. sw $7, 0($1)

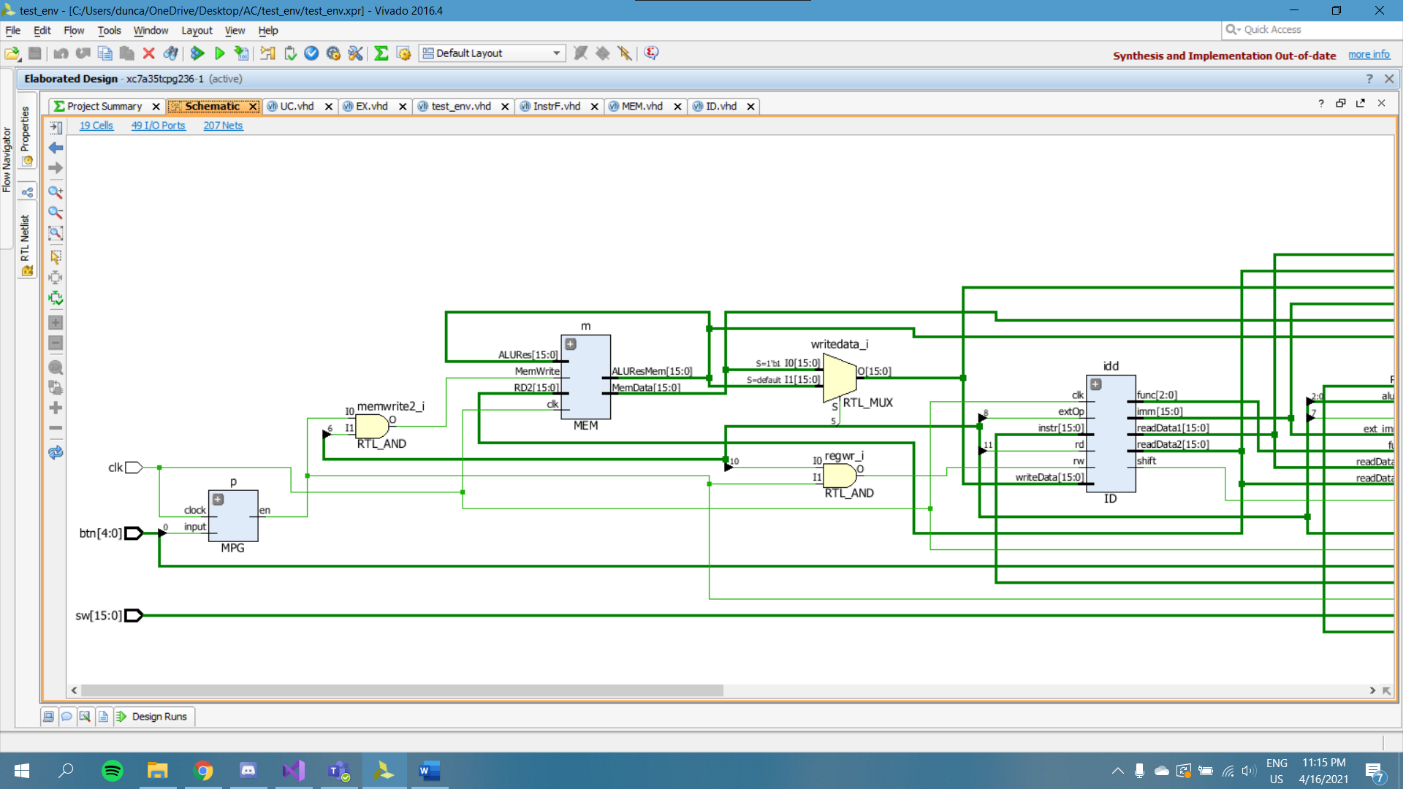
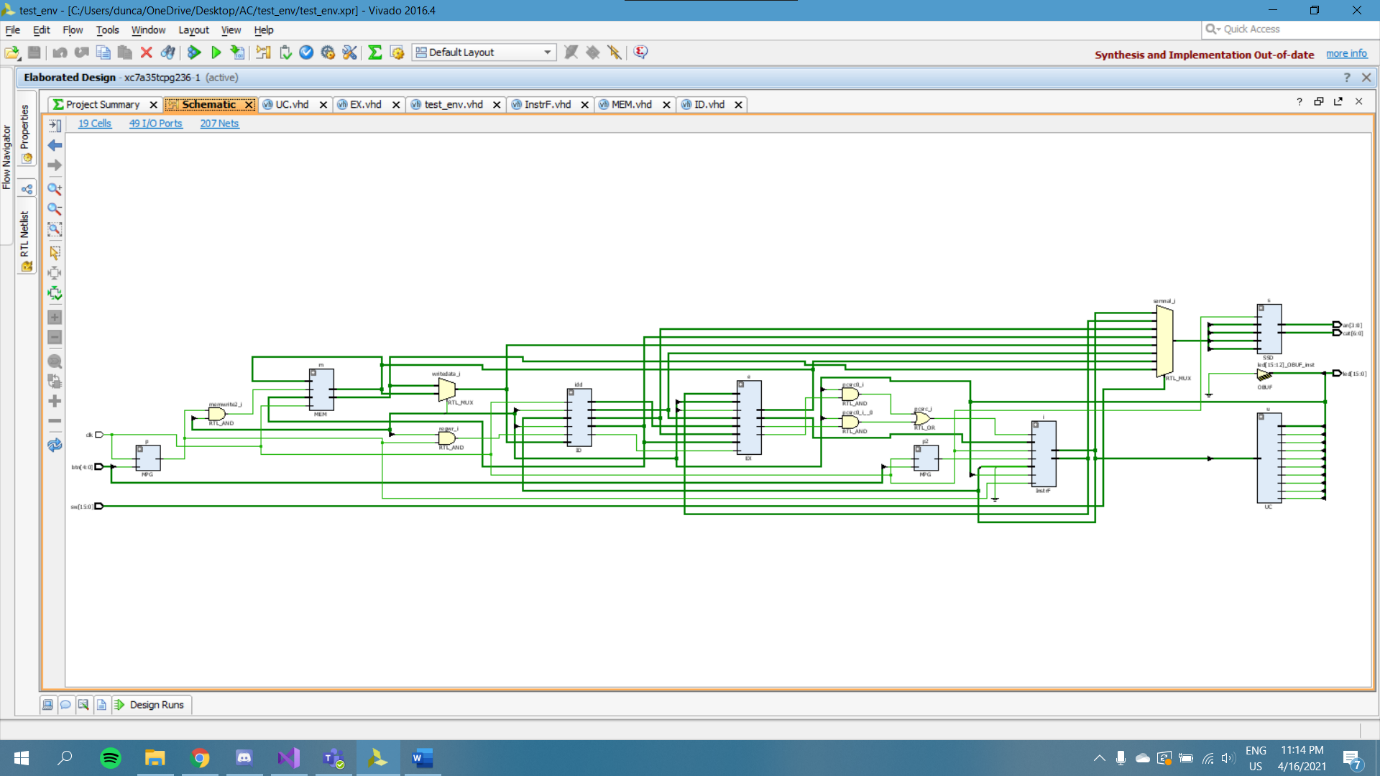
instr = 011; PC = 15; RD1 = 19; RD2 = x; WD = 19; EXT = 1; ALUres = 19 ; MEMDATA =19;

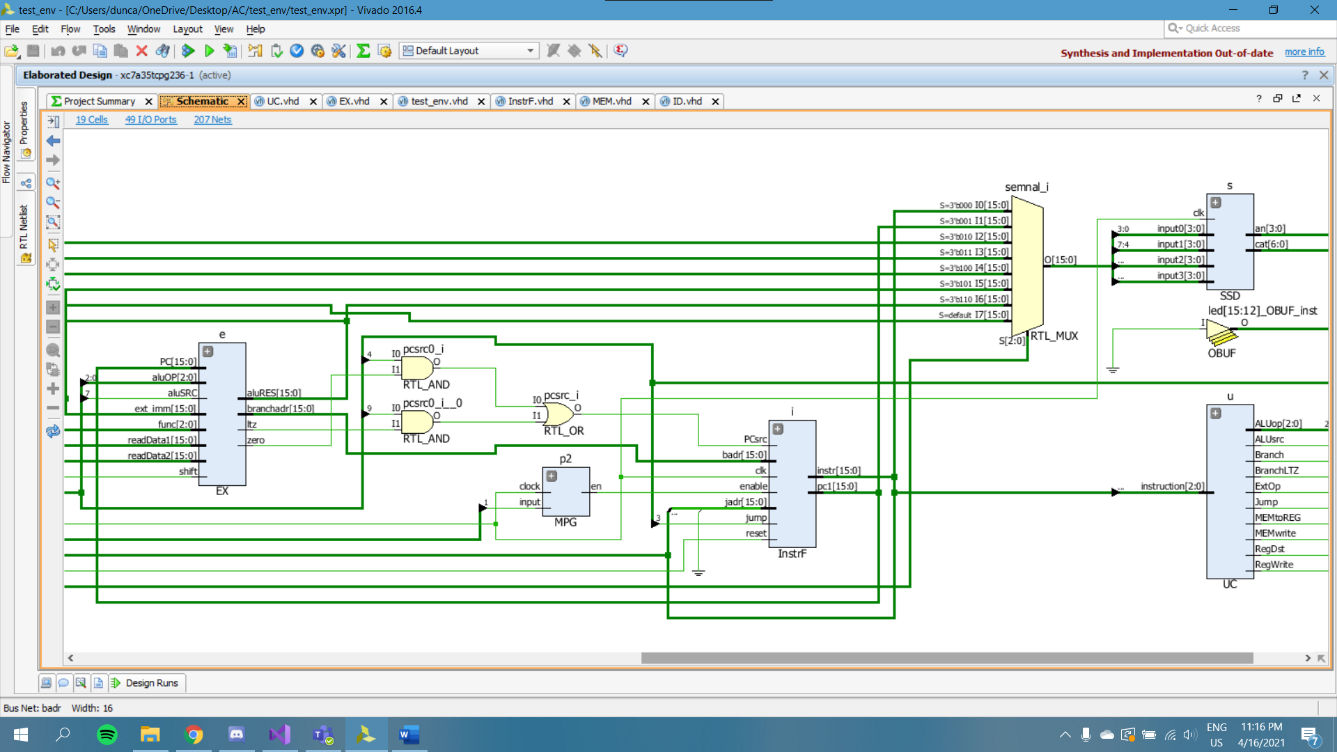
e.

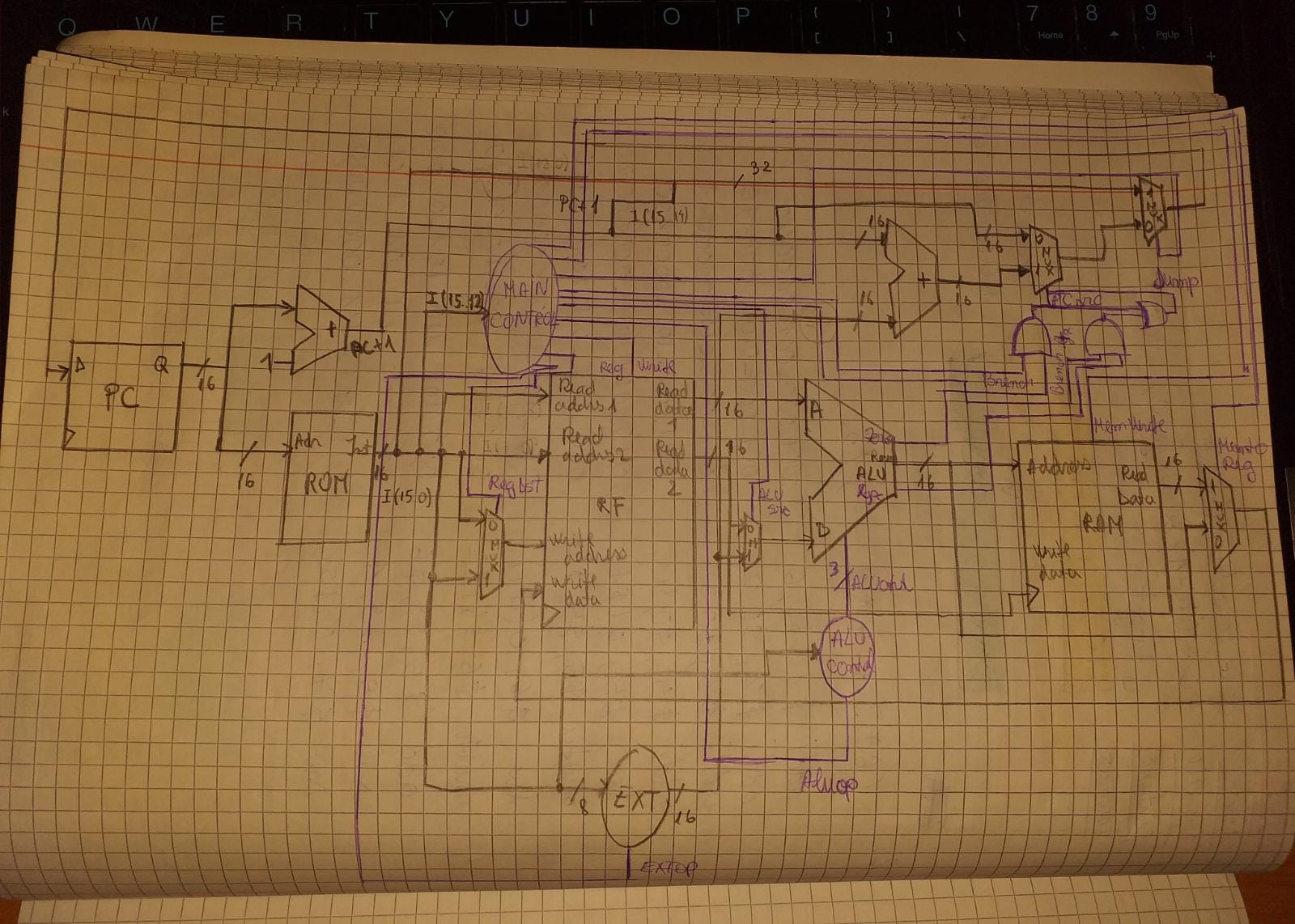
Nu exista activitati incomplete din laboratoarele 4-7.

f.

Nu sunt erori în codul vhdl prin urmare am generat fisierul \*.bit

Schema cu analiza RTL:



Schema cu tot MIPS ul ciclu unic pe 16 biți, făcută în caiet.