Pop Ruxandra Maria
30226

# MIPS 16
# Ciclu unic

## *Instrucțiunile alese suplimentar:*

1. XOR -> realizează operația XOR între biți celor două registre, și pune rezultatul în primul registru

ASM

xor $d,$s,$t

ex: xor $1,$1,$2

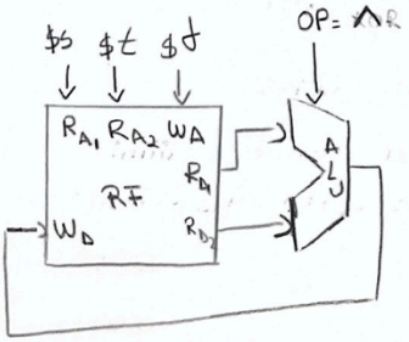
RTL

$d <- $s ^ $t; PC <- PC+1

Cod mașină

B"000_sss_ttt_ddd_0_110"

ex: B"000_001_010_001_0_110"




2. SLT (Set on less than (signed)) - dacă $s<$t, $d este inițializat cu 1, altfel cu 0

ASM

slt $d,$s,$t

ex: slt $1,$1,$2

RTL

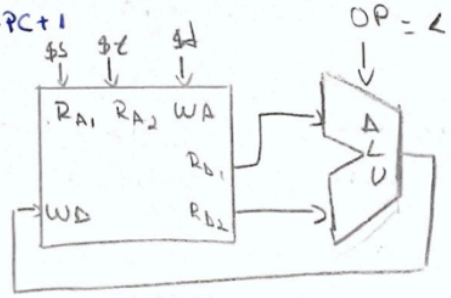
if $s<$t $d <- 1; else $d <- 0

P <- PC+1

Cod mașină

B"000_sss_ttt_ddd_0_111"

B"000_001_010_001_0_111"



3. BNE (Branch on not equal) salt conditionat dacă 2 registre sunt diferite. Dacă cele 2 registre sunt diferite sare la adresa ~~imm~~ dată de imm

ASM

bne $s, $t, imm

ex: bne $2, $3, 3

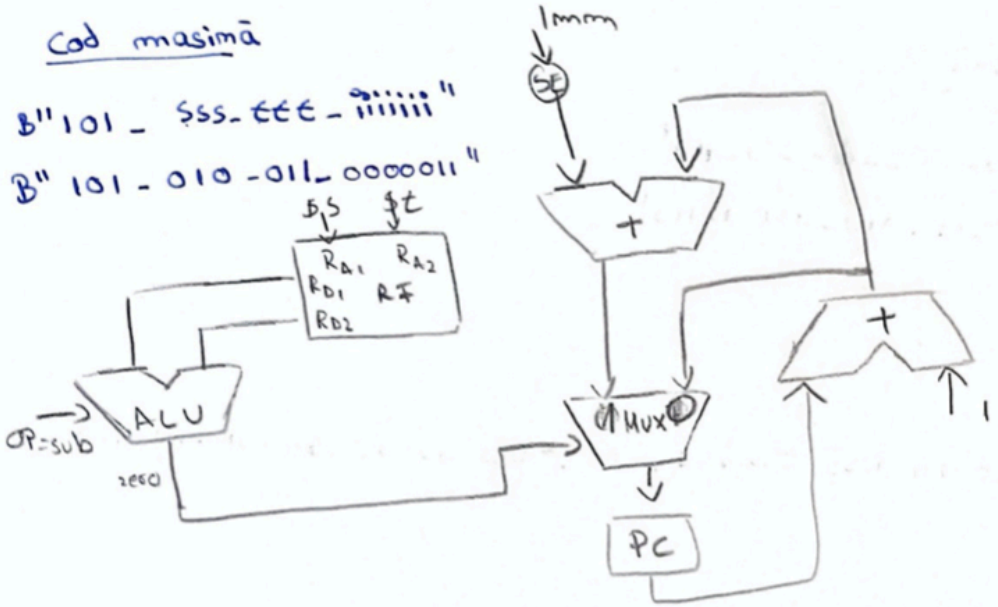
RTL

if ($s != $t) PC <- PC+1+SE(imm)
else PC <- PC+1

Cod mașină

B"101_sss_ttt_iiiiiiii"

B"101-010-011_00000011"




4. BGEZ (Branch on greater than on equal to zero), salt conditionat dacă un registru este mai mare sau egal cu 0

ASM

bgez $s, imm

ex bgez $4, 12

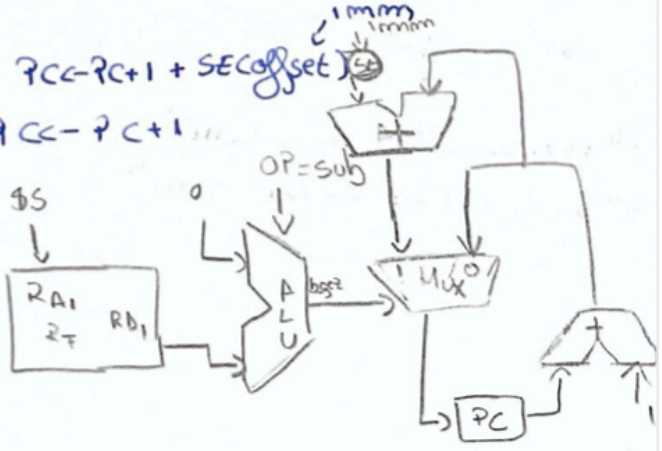
RTL

if $s ≥ 0 PC <- PC+1+SE(offset)
else PC <- PC+1

Cod mașină

B"110_sss_001_iiiiiiii"

B"110-100-001-00001100"



## *Semnalele de control:*

| Instrucțiune | Opcode *Instr(15-13)* | RegDst | ExtOp | ALUSrc | Branch | <Br?> (opțional) | Jump | JmpR (opțional) | MemWrite | MemtoReg | RegWrite | ALUOp (1:0) | func *Instr(2-0)* | ALUCtrl (2:0) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADD | -000- | 1 | 0 | 0 | 0 | | 0 | | 0 | 0 | 1 | 00(R) | -000- | 000(+) |
| SUB | -000- | 1 | 0 | 0 | 0 | | 0 | | 0 | 0 | 1 | 00(R) | -001- | 001(-) |
| SLL | -000- | 1 | 0 | 0 | 0 | | 0 | | 0 | 0 | 1 | 00(R) | -010- | 010(<<l) |
| SRL | -000- | 1 | 0 | 0 | 0 | | 0 | | 0 | 0 | 1 | 00(R) | -011- | 011(>>l) |
| AND | -000- | 1 | 0 | 0 | 0 | | 0 | | 0 | 0 | 1 | 00(R) | -100- | 100(&) |
| OR | -000- | 1 | 0 | 0 | 0 | | 0 | | 0 | 0 | 1 | 00(R) | -101- | 101(\|) |
| XOR | -000- | 1 | 0 | 0 | 0 | | 0 | | 0 | 0 | 1 | 00(R) | -110- | 110(^) |
| SLT | -000- | 1 | 0 | 0 | 0 | | 0 | | 0 | 0 | 1 | 00(R) | -111- | 111(<) |
| ADDI | -001- | 0 | 1 | 1 | 0 | | 0 | | 0 | 0 | 1 | 01(+) | | 000(+) |
| LW | -010- | 0 | 1 | 1 | 0 | | 0 | | 0 | 1 | 1 | 01(+) | | 000(+) |
| SW | -011- | 0 | 1 | 1 | 0 | | 0 | | 1 | 0 | 0 | 01(+) | | 000(+) |
| BEQ | -100- | 0 | 1 | 0 | 1 | | 0 | | 0 | 0 | 0 | 10(-) | | 001(-) |
| BNE | -101- | 0 | 1 | 0 | 0 | BNE 1 | 0 | | 0 | 0 | 0 | 10(-) | | 001(-) |
| BGEZ | -110- | 0 | 1 | 0 | 0 | BGEZ 1 | 0 | | 0 | 0 | 0 | 10(-) | | 001(-) |
| J | 111 | 0 | 0 | 0 | 0 | | 1 | | 0 | 0 | 0 | xx | | xxx |

Primele 8 instrucțiuni sunt de tip R ,după cum se poate observa acestea au și un semnal func ,pentru a putea face diferența între ele în EX.vhd,doarece au acelasi opcode .Am mai adăugat pentru BNE și BGEZ ,un semnal de control.

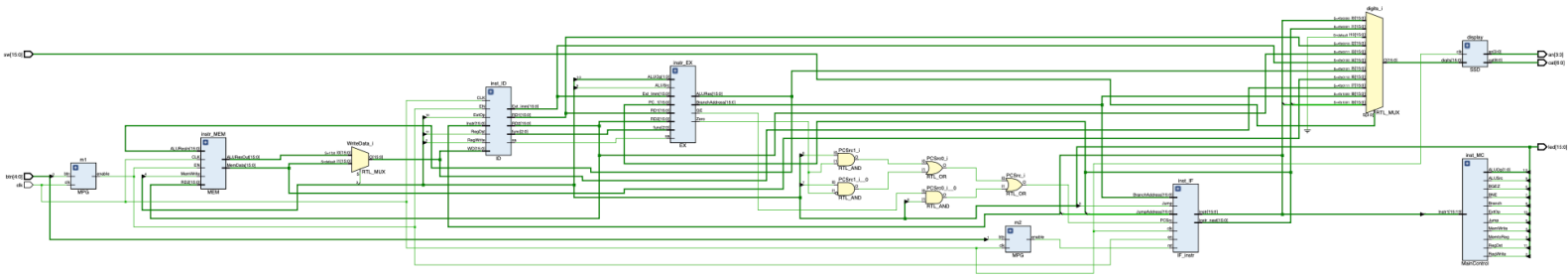
***Trasarea execuției programului:***

*Am mai adaugat un bit pentru switch ,pentru a putea vizualiza pe placă, și adresa de branch și de jump.*

| Pas | SW(7:5) | “0000” | “0001” | “0010” | “0011” | “0100” | “0101” | “0110” | “0111” | 1000/1001 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Instr (*în asamblare*) | Instr (*hexa*) | PC+1 | RD1 | RD2 | Ext_Imm | ALURes | MemData | WD | BranchAddr | JumpAddr |
| 0 | ADDI $1, $0,0 | x”2080” | x”0001” | x”0000” | x”0000” | x”0000” | x”0000” | x”0000” | x”0000” | x”” | x”” |
| 1 | ADDI $2, $0,4 | x”2104” | x”0002” | x”0000” | x”0000” | x”0004” | x”0004” | x”0000” | x”0004” | | |
| 2 | ADD $3, $0,$0 | x”0030” | x”0003” | x”0000” | x”0000” | x”0030” | x”0000” | x”0000” | x”0000” | | |
| 3 | ADDI $4, $0,1 | x”2201” | x”0004” | x”0000” | x”0000” | x”0001” | x”0001” | x”0000” | x”0001” | | |
| 4 | ADDI $5, $0,12 | x”228C” | x”0005” | x”0000” | x”0000” | x”000C” | x”000C” | x”0000” | x”000C” | | |
| 5 | ADDI $6, $0,1 | x”2301” | x”0006” | x”0000” | x”0000” | x”0001” | x”0001” | x”0000” | x”0001” | | |
| 6 | BEG $1, $5,8 | x”8688” | x”0007” | x”0000” | x”000C” | x”0008” | x”FFF4” | x”0000” | x”FFF4” | x”000F” | |
| 7 | SW $1, 0($3) | x”6C80” | x”0008” | x”0000” | x”0000” | x”0000” | x”0000” | x”0000” | x”0000” | | |
| 8 | SW $2, 0($4) | x”7100” | x”0009” | x”0001” | x”0004” | x”0000” | x”0001” | x”0000” | x”0001” | | |
| 9 | LW $1, 0($3) | x”4C80” | x”000A” | x”0000” | x”0000” | x”0000” | x”0000” | x”0000” | x”0000” | | |
| 10 | LW $2, 0($4) | x”5100” | x”000B” | x”0001” | x”0004” | x”0000” | x”0001” | x”0004” | x”0004” | | |
| 11 | ADD $1, $1,$6 | x”0710” | x”000C” | x”0000” | x”0001” | x”0010” | x”0001” | x”0004” | x”0001” | | |
| 12 | ADD $2, $2,$1 | x”08A0” | x”000D” | x”0004” | x”0001” | x”0020” | x”0005” | x”0000” | x”0005” | | |
| 13 | SUB $5, $5,$6 | x”1751” | x”000E” | x”000C” | x”0001” | x”0051” | x”000B” | x”0000” | x”000B” | | |
| 14 | J 6 | x”E006” | x”000F” | x”0000” | x”0000” | x”0006” | x”0000” | x”0000” | x”0000” | | x”0006" |

Programul nu are un scop anume.Conține o buclă care se va executa de 5 ori ,mai exact până când valorea conținută de registrul 1 este egală cu valoarea conținută de registrul 5(ambele trebuie să aibe valoarea 6). Un așa zis “rol” al programului este de a calcula valoarea registrului 2 ,care la final ar trebui să fie 25 , iar pe placuță ar trebui să se afișeze “0019”, deoarece numărul 25 este reprezentat în binar ca fiind 11001.Ceea ce se întâmplă într-o bucla este descris în IF.vhd,dar ca principiu , valoarea registrului 1 se incrementează cu 1 , registrul 2 va lua valoarea curentă din registrul 1 adunată cu valoarea la momentul actual din registrul 2 (R2<-R2+R1),iar registrul 5 se decrementează cu unul .După care se sare la instrucțiunea de la adresa 6, unde se va compara valoarea registrelor, și se va analiza dacă se sare peste 9 instrucțiuni sau se continuă bucla.

## *RTL schematic*



Nu am activități incomplete ,am reușit să implementez toate componentele,după cum se poate observa în schemă.Pe părcursul laboratoarelor am întâmpinat unele neclarități sau probleme ,dar cu ajutorul profesorului de laborator am reusit să le rezolv ,astfel că în final totul funcționează corect ,fără erori.

## Descrierea componentelor

***1.IF.vhd*** *-Unitatea de extragere a instrucțiunilor MIPS16*
*-Instruction Fetch*
*Această componentă generează instrucțiunea curentă și adresa urmatoarei instrucțiuni.*
*Aici se gaseste memoria ROM care conține programul scris de mine,instrucțiuniile pe care le va executa procesorul.Programul conține 15 instrucțiuni, acestea sunt scrise sub codificare binară,iar pentru fiecare instrucțiune este prezent câte un comentăriu care prezintă codificarea instrucțiuni în hexazecimal ,specificația în limbaj de asamblare ,și ceea ce face .*

***2.ID.vhd*** *- Unitatea de decodificare a instrucțiunilor MIPS16*
*- Instruction Decode*

*Această componentă extrage valoriile pentru RD1,RD2,Ext_imm,func,sa.Conține și un register file.*
*Aici se citeste din reg_file ,se scrie în reg_file în funcție de semnalul RegDst a fiecărei instrucțiuni. Totodată aici se efectuează exitinderea cu 0 și extinderea cu semn ,pentru instrucțiunile care prezinta ExtOp.*

```
Ext_imm<=Instr(6)&Instr(6) & Instr(6)&Instr(6)&Instr(6)&Instr(6) &Instr(6)&Instr(6)&
Instr(6 downto 0) when ExtOp='1' else "000000000" & Instr(6 downto 0);
```

- *reg_file.vhd*

*Toate registrele au fost inițializate cu 0.*
*Prezintă o scriere sincronă,și o citire asincronă.*
*Scrierea in registru necesită și un semnal de enable.*

***3.MainControl.vhd.*** *-unitatea de control*

*Furnizează cu ajutorul opcode-ului (Instr(15-13)) valorile semnalelor descrise în tabel pentru fiecare instrucțiune.La început, fiecărui semnal i-am atribuit valoarea 0, îar pe urmă în funcție de Instr ,se activează semnalele corespunzătpare fiecărei operații.*
***Obs****.Instrucțiunile de tip R au fost luate toate impreună deoarece prezintă același opcode.*

***4.EX.vhd -*** *unitatea de execuție*
*-Instruction Execute*

*Această componentă furnizează rezultatul ALU,adresă de branch precum și adresa de branch pentru instrucțiunea BGEZ.Totodată aici se implementează și logica pentru flag-ul zero (isZero).*
*Aici sunt implementate funcționarea fiecarei operații prezente in tabel.*

***5.MEM.vhd*** *- memoria de date*

*Prezintă o scriere sincronă și o citire asincronă.*
*Scrierea în memorie este influențată de semnalul de enable, astfel se scrie doar dacă acesta este activ.*

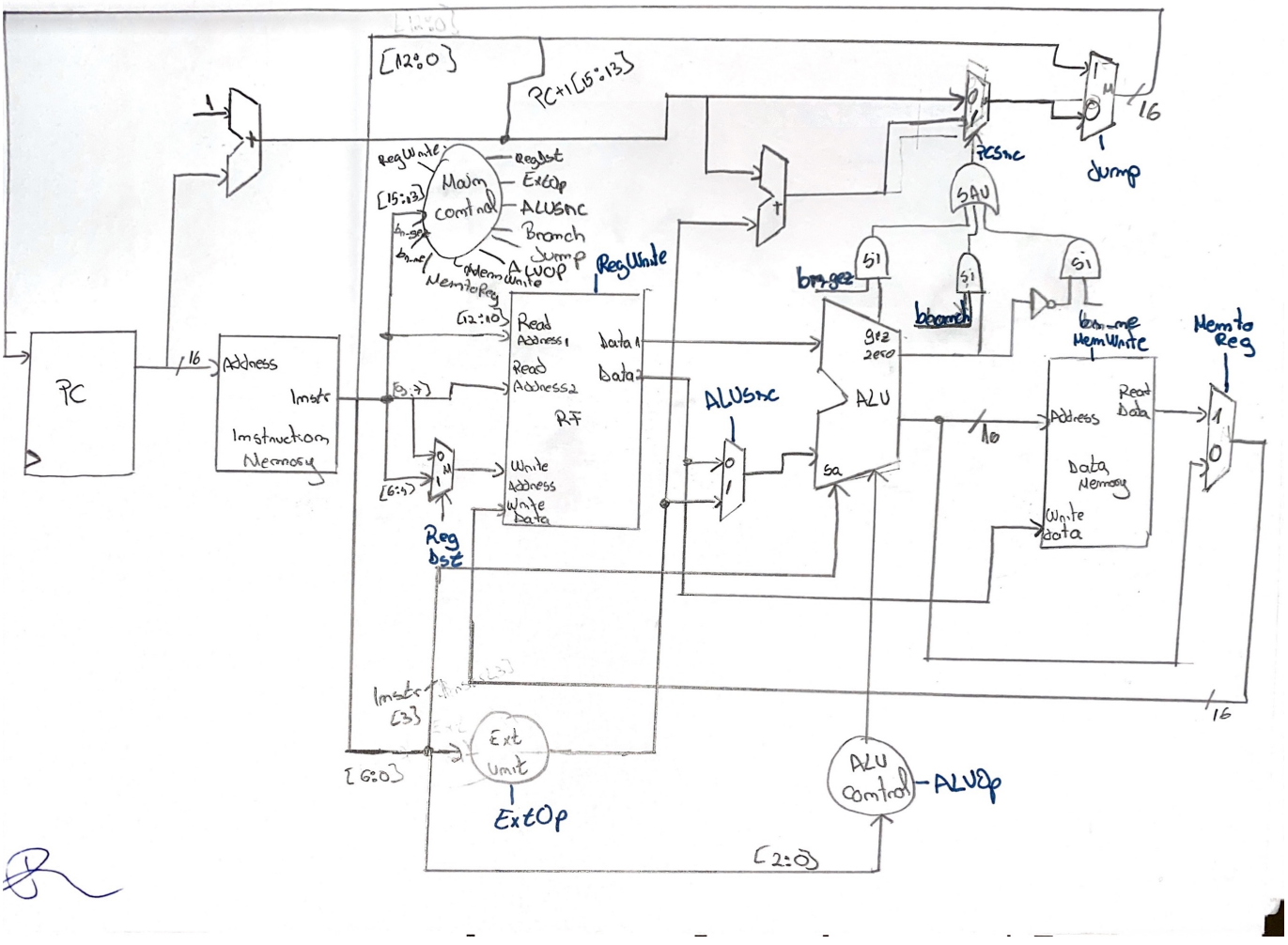
***6.MPG.vhd -****generator Monopuls Sincron*

*Prezintă 3 registre (Q1,Q2,Q3) .*
*Prezintă un numărator sincron pe 16 biți ,care la un semnal de clock se incrementează.*

***7.SSD.vhd*** *-afișor pe 7 segmente cu 4 cifre*
*Pentru multiplexoare am folosit procese ,care sunt combinaționale.De la număratorul intern folosesc doar ultimi 2 biți pentru a determina care anod este activ la un moment dat ,și totodată am setat și catozi în funcție de crifa care dorește să fie afisată pe SSD.*

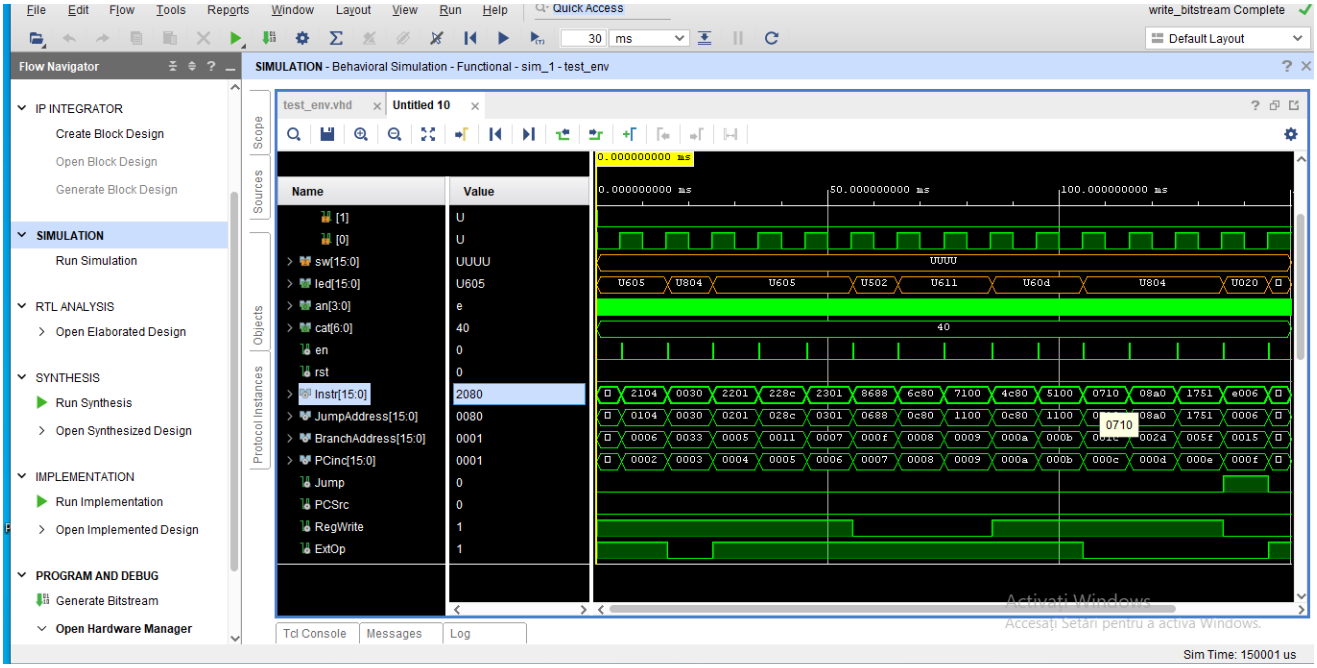
***8.test_env.vhd -****proiectul final care conține toate lagăturile*

*În această entitate s-au legat toate componentele precedente ,pentru a realiză procesorul MIPS16 ,conform schemei căii de date,realizate de noi în primul laborator referitor la MIPS16.*
*Pentru a se afisa pe SSD ,valorile dorite am folosit o instrucțiune concurentă de tipul “with ..select”,care va fi controlată cu ajutorul switch-urilor(7-4).*
*Cum am mai amintit în acest raport ,am folosit încă un switch pentru a putea reprezenta și adresa de branch și adresa da jump*



## *Testarea corectitudinii:*

*Pentru a testa funcționalitatea programului,am analizat ce ar trebui să fie afișat cu ajutorul SSD-ului,după care am completat tabelul de trasare a execuției.*
*Următorul pas a fost analizarea funcționari cu ajutorul simulatorului din Vivado.*
*Aici se poate observa un print din fereastra de simulare.*




*După care am efectuat și testarea cu ajutorul  plăcuței Basys 3.Totul a funcționat exact cum trebuia ,fără erori sau diferențe între ceea ce am scris în tabel și ceea ce este afisat pe placă.*

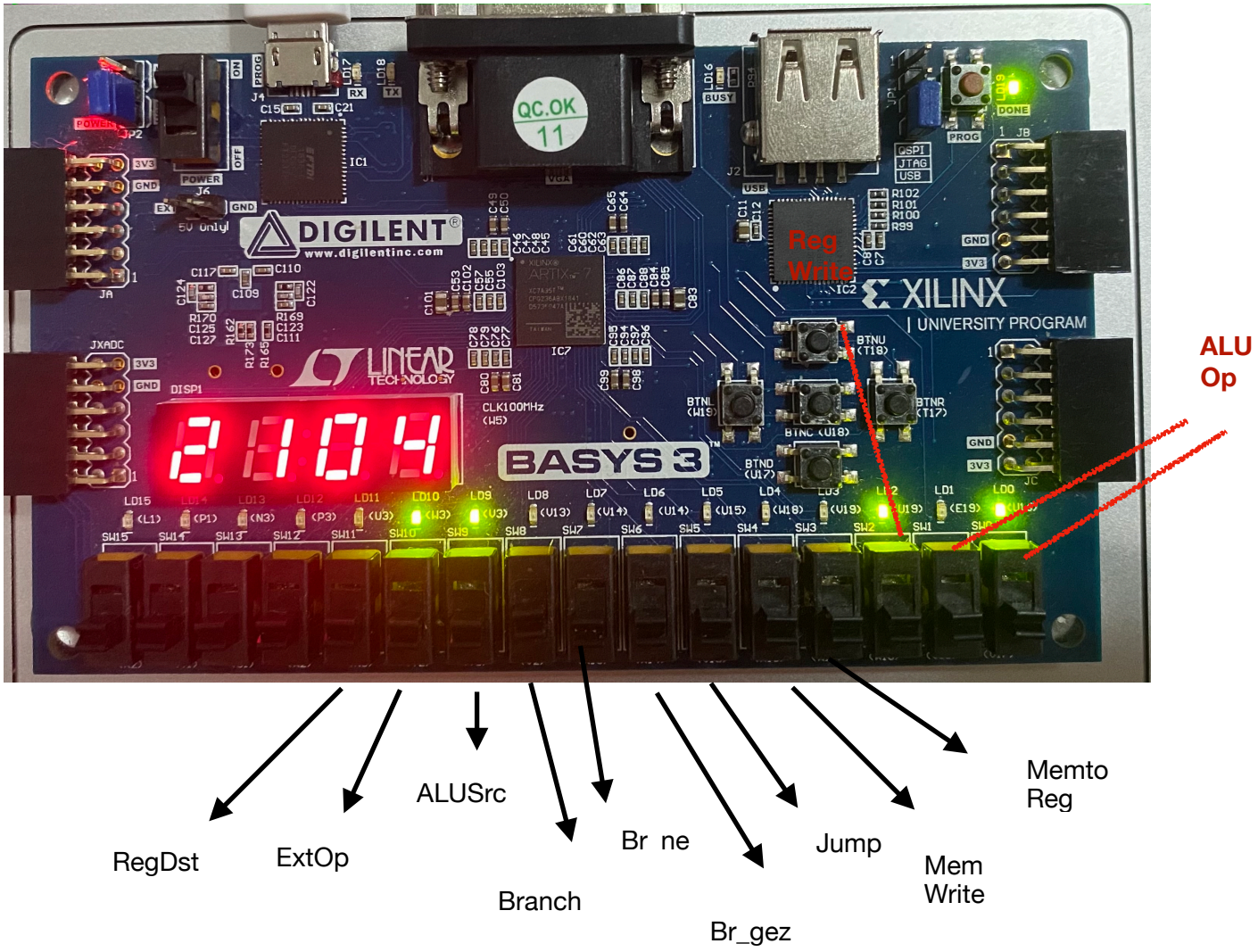
*Mai jos am pus o imagine cu testarea pe placuța.Se poate observa codul întrucțiuni în hexadecimal.”2104” corespunde celei de a doua înstrucțiune din program ,și mai exact este o înstrucțiune  addi.*

*Se poate observa cum se aprind led-urile în funcție de flag-urile active ,corespunzătoare tipului de instrucțiune.Addi este o instrucțiune de tip I și prezintă urmatoarele semnale de control (Se gasesc și în tabelul cu semnalele de control).*

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADDI | -001- | 0 | 1 | 1 | 0 | | 0 | | 0 | 0 | 1 | 01(+) | | 000(+) |

*Iar pe led-urile placuței se gasesc în aceiași ordinea:*

```
led(11 downto 0)<=RegDst &ExtOp&ALUSrc&Branch&br_ne &br_gez &Jump&MemWrite &MemtoReg & RegWrite &ALUOp
```




**Obs!**

ALUOp este pe 2 biți