MINISTRY OF EDUCATION AND RESEARCH
TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

Arhitectura Calculatoarelor

Procesor MIPS 16 biți

-Xilinx Vivado Suite 2016.4, VHDL -

Realizat de:

Tanul Gabriel-Ștefan

an 2 grupa 30222

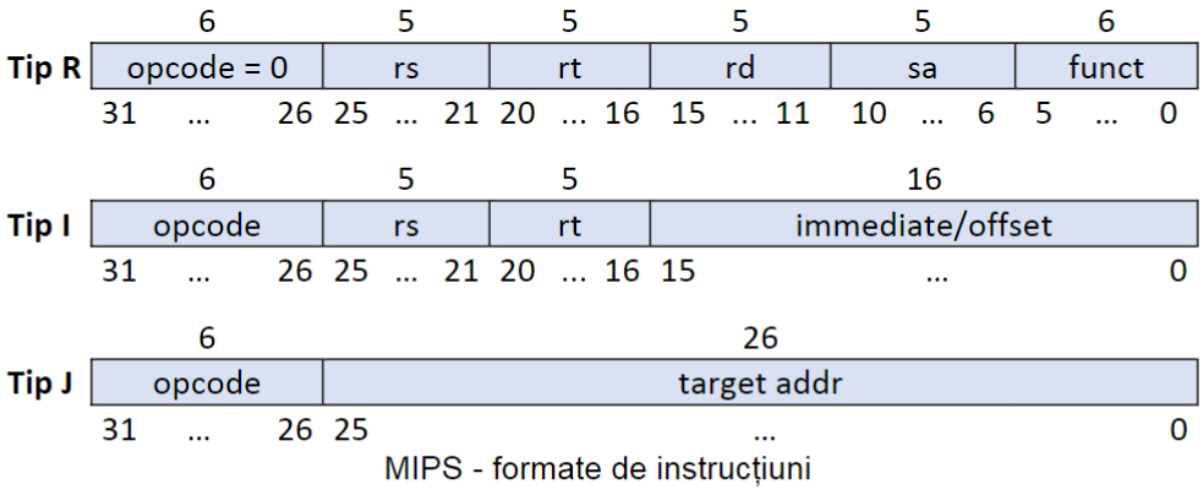
I. Descriere

I.1. Componente

II. Testare

I. Descriere

- MIPS- Microprocessor without Interlocked Pipeline Stages
- Conține mai multe tipuri de instrucțiuni, regiștri în care se memorează valori numerice și codificări pentru anumite operaț



MIPS - formate de instrucțiuni

În cazul MIPS 16 biți codificările sunt după cum urmează

Tip R:

3 biti 3 biti 3 biti 3 biti 3 biti 3 biti

opcode –----------rs---------- –rt --------– rd------- –sa ----------func

Tip I:

3 biti 3 biti 3 biti 7 biti

opcode –----------rs---------- –rt ----------immediate----------

Tip J:

3 biti 13 biti

opcode –---------- immediate----------

I.1. Componente

Procesorul conține o componentă **Instruction_Fetch** care parcurge instrucțiunile aflate în memoria ROM și transmite ca semnal PC+1 și instrucțiunea de prelucrat.

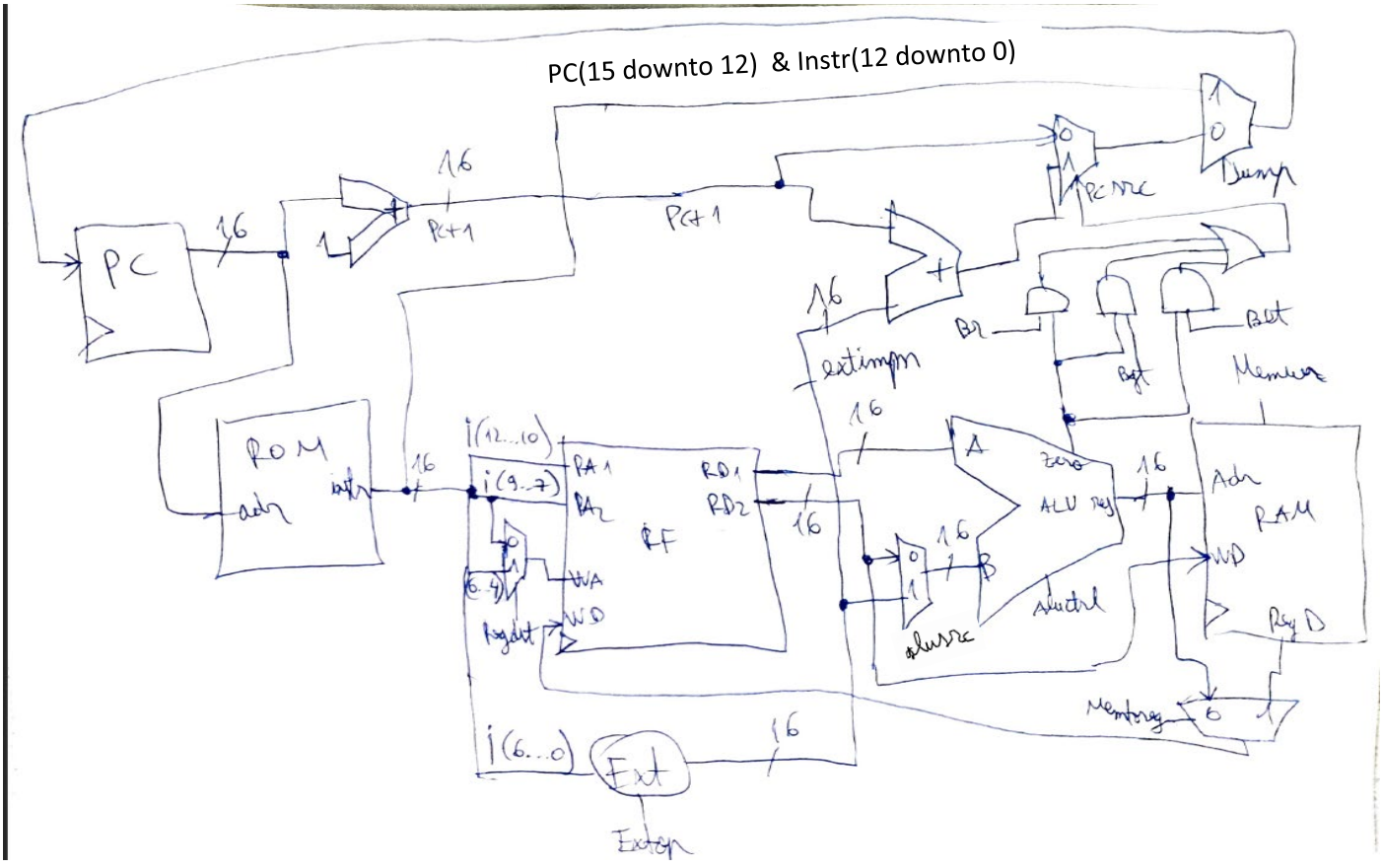
Instruction_Decode este responsabilă de decodificarea instrucțiunii în formatul standard MIPS și transmiterea adreselor de regiștri, ALUctrl și salt.

ALU execută operațiile necesare în funcție de ALUctrl.

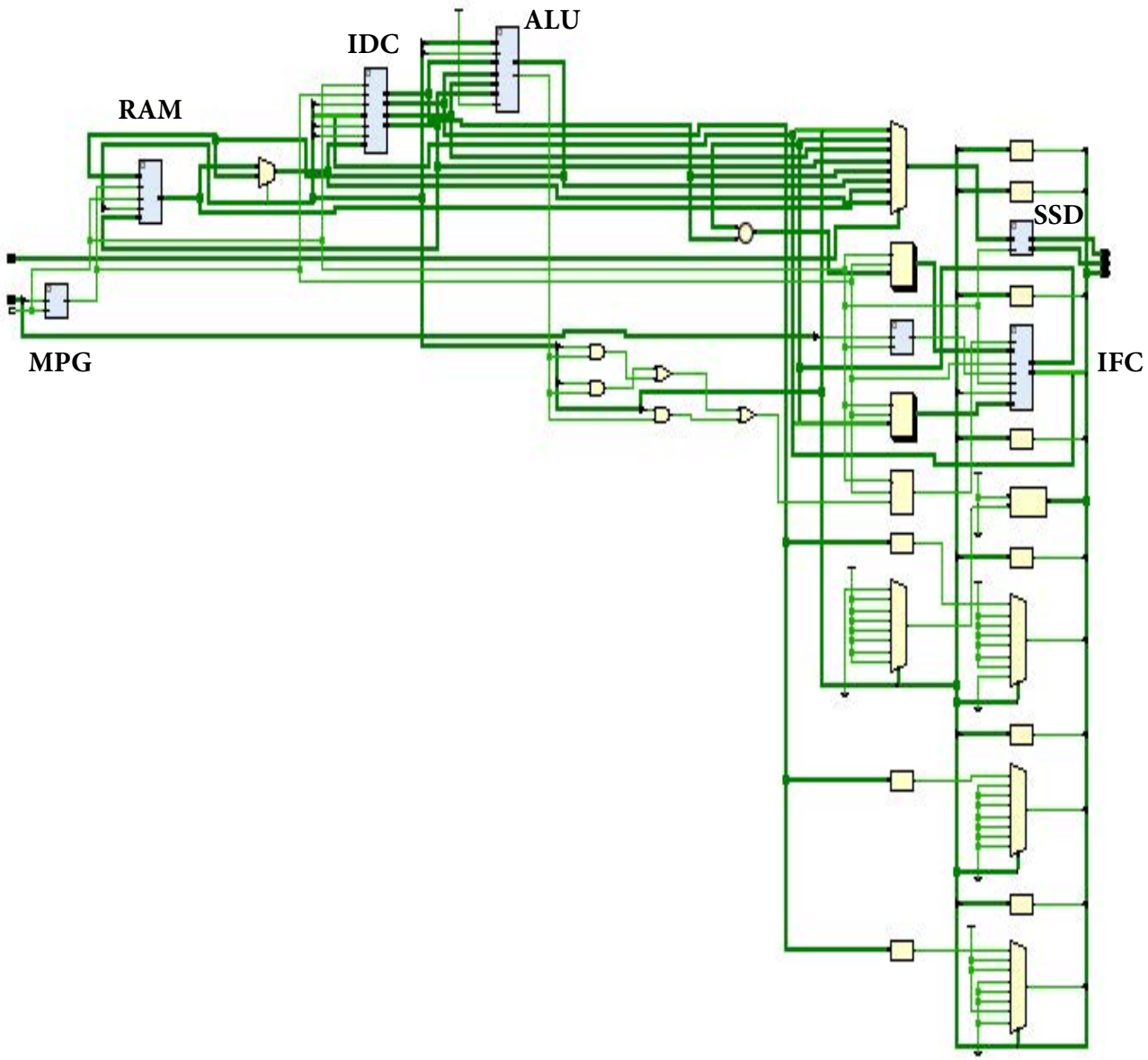
RAM memorează date și de asemenea oferă acces la acestea.

REG_BLOCK este blocul principal de regiștri (8 regiștri, cu registrul 0 setat pe 0 standard) în care se memorează și din care se citesc valori numerice.

Așa arată schema bloc a procesorului MIPS de 16 biți după codificarea standard în funcție de tipul operației suportate în arhitectura acestuia.



Schema implementata Vivado



Toate semnalele fac capabile rularea procesorului pentru un anumit număr de operații declarate în memoria **ROM**.

Semnale folosite:

- **Pc**- contorul de operații – este asemănător unui pointer care indică adresa operației curente
- **Instr-** este semnalul ce preia instrucțiunea și o sparge în formatul prezentat mai sus. (opcode – rs –rt – rd –sa –func etc)
- **Instr(12..10) rs**
- **Instr(9...7) rt**
- **Instr(6....4) rd**
- **ALURes-** rezultatu operației executată de unitatea ALU (Unitatea Aritmetică-Logică)
- **ALUCtrl-** semnal pentru tipul operației care se execută în ALU
- **Instr(6.....0) immediate-** care necesită extins la 16 biți si merge în Ext unde are semnal de control **ExtOp**
- **RA1-** adresa de citire a registrului **rs**
- **RA2-** adresa de citire a registrului **rt**
- **WA-** adresa de scriere în registrul **rd**
- **WD-** valoarea de scriere în regitrul **rd**
- **RD1, RD2-** valorile regiștrilor **rs, rt** transmise către ALU
- **AdrRAM-** adresa din/în care se citește/scrie
- **WDRAM-** valoarea care se scrie în RAM
- **Br, BGT (branch on greater than), BLT (branch on lower than)-** semnale care determină dacă se execută un salt condiționat
- **JUMP-** semnal care determină saltul forțat
- **Memtoreg-** semnal care permite transmiterea în registru a unei valori din memorie
- **Memwr-** semnal care permite scrierea în memorie
- **PCsrc-** semnal care decide dacă se transmite adresa unui salt condiționat sau adresa instrucțiunii următoare (PC+1)

II. Testare

M-am folosit de o placa de dezvoltare FPGA Basys 3 de la Xilinx iar ca software am utilizat Xilinx Vivado Suite 2016.4 unde am descris în cod VHDL funcționalitățile procesorului.

Am construit un mic algoritm de împărțire a 2 numere prin scăderi repetate care necesită buclă.

Ex: 21/5

21 – 5 = 16 , catul = 1

16 – 5 = 11 catul = 2

11 – 5 = 6 catul = 3

6 – 5 = 1 < 5 , catul = 4 și restul 1

B"001_000_101_0000011", --**lw $5, 0 (mem(0) ----- 2283**

B"001_000_001_0000001", --**lw $1,21 (mem(1) --deimpartitul -------2081**

B"001_000_010_0000010", -- **lw $2,5 (mem(2) --impartitorul ------2102**

B"011_010_001_0000100", -- beq $2,$1 **, jmp(7)---------- 6884**

B"000_001_010_001_0_010", -- **sub $1,$1,$2 (scadere repetata) bucla-eticheta ---0512**

B"110_101_101_0000001", -- **addi $5,%5, 1 (incrementare cat) ----------D681**

B"100_001_010_1111101", **--bgt $1,$2 (daca e mai mare inseamna ca trebuie facut branch inapoi la scadere ---------857D**

B"010_000_101_0000000", **-- sw memorie(0), %1 --restul -------4280**

B"010_000_001_0001001", **--sw memorie(9), $5 --catul ------------4089**

B"111_0100000000000", **--jmp pentru blocare**

B"111_0100000000001", **--jmp pentru blocare**

Video testare împărțire pe placa Basys3

Tanul Gabriel-Stefan-30222

Semnale control MIPS16 pentru Anexa 5

Opcode 000 - tip R
Func 000 – xnor
001 – add
010 - sub
011 – sll
100 - srl
101 – and
110 - or
111 – xor

tip I
Opcode:
001 – lw
010 - sw
011 - beq
100 - bgt
101 – blt
110 - addi

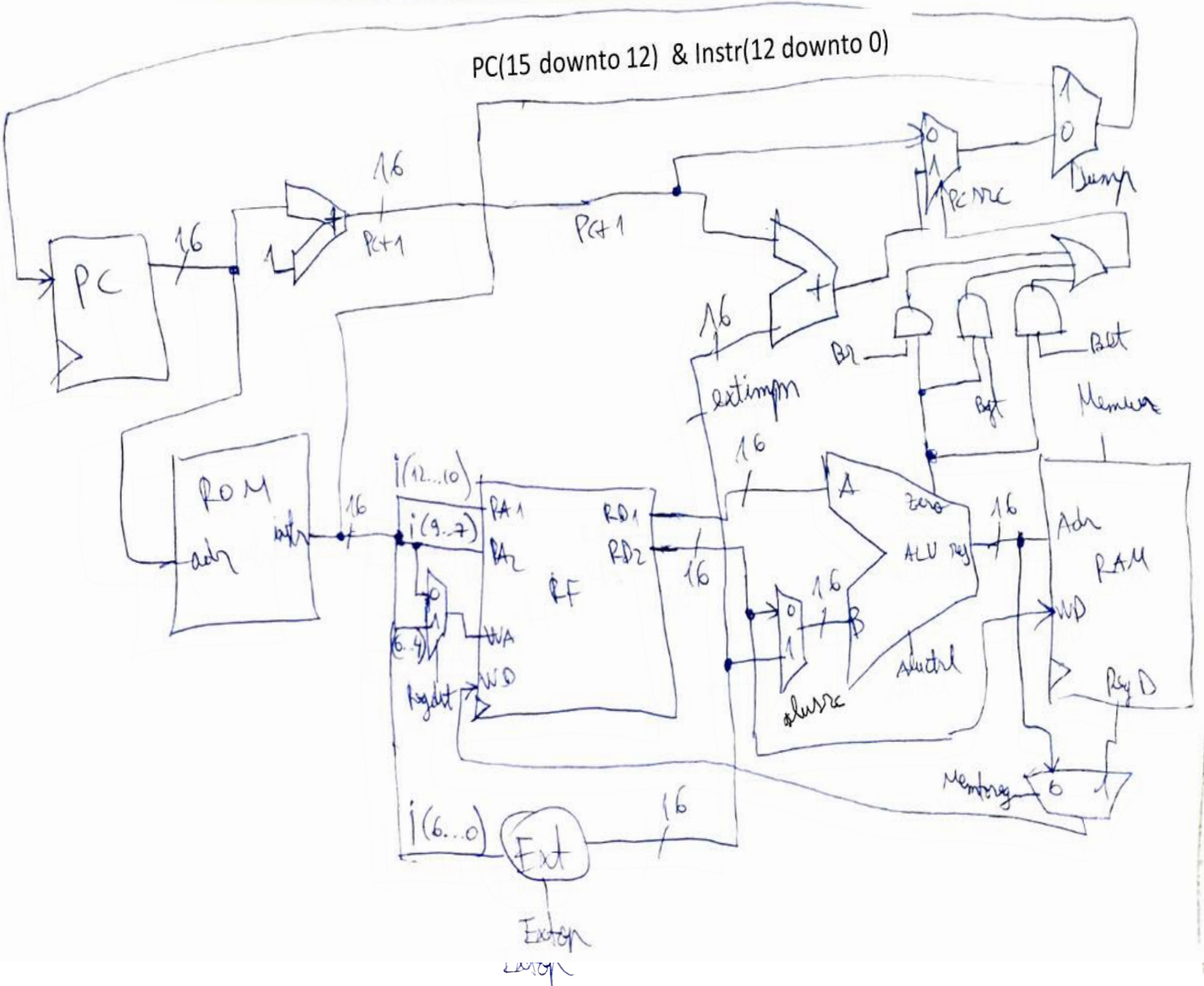
tip J
Opcode:
111 - jump

Instructiune	**Opcode** *Instr(15-13)*	**RegDst**	**ExtOp**	**ALUSrc**	**Branch**	**Blt**	**Bgt**	**Jump**	**Mem Write**	**Memto Reg**	**Reg Write**	**func** *Instr(2-0)*
add	000	1	0	0	0	0	0	0	0	0	1	001
sub	000	1	0	0	0	0	0	0	0	0	1	010
sll	000	0	1	1	0	0	0	0	0	0	1	011
srl	000	0	1	1	0	0	0	0	0	0	1	100
and	000	1	0	0	0	0	0	0	0	0	1	101
or	000	1	0	0	0	0	0	0	0	0	1	110
xor	000	1	0	0	0	0	0	0	0	0	1	111
xnor	000	1	0	0	0	0	0	0	0	0	1	000
lw	001	1	1	1	0	0	0	0	0	1	1	---
sw	010	0	1	0	0	0	0	0	1	0	0	---
beq	011	0	1	0	1	0	0	0	0	0	0	---
bgt	100	0	1	0	0	0	1	0	0	0	0	---
blt	101	0	1	0	0	1	0	0	0	0	0	---
addi	110	0	1	1	0	0	0	0	0	0	1	---
Jump	111	0	0	0	0	0	0	1	0	0	0	---

Bibliografie

https://users.utcluj.ro/~onigaf/files/AC.html

https://reference.digilentinc.com/reference/programmable-logic/basys-3/start


PC(15 downto 12) & Instr(12 downto 0)
PC
16
PC+1
PC+1
PCSrc
Jump
16
extimm
Br
Bne
Bgt
Memtoreg
ROM
adr
instr
16
i(12...10)
i(9...7)
RA1
RA2
RF
RD1
RD2
16
A
Zero
ALU
16
Adr
RAM
WD
RegD
16
16
ALUSrc
ALUctrl
i(6...0)
Ext
16
ExtOp
Memtoreg

Impărțire

21/5

```
0  lw  $5, 1 mem(3)
1  lw  $1, 21 mem(1)
2  lw  $2, 5 , mem(2)
3  beq $2, $1, jmp(7)  (nr. egale, catul 1)
4     sub $1, $1, $2 -- diferența deîmpărțit-împărțitor
5     addi $5, $0, 1 -- incrementare cat
6  bgt $1, $2, jmp(4)
7  sw  mem(0), $1 -- restul
8  sw  mem(9), $5, -- catul
9  jmp out (adresă mare 2^13)
```

21 - 5 = 16 C = 1 R = 16

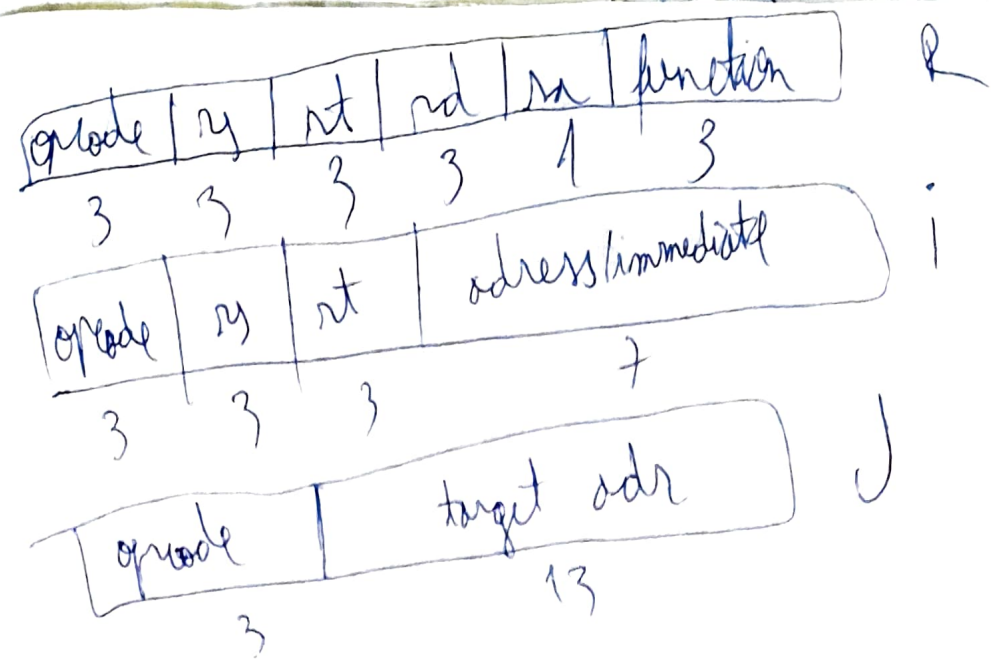
16 - 5 = 11 C = 2 R = 11

11 - 5 = 6 C = 3 R = 6

6 - 5 = 1 C = 4 > R = 1

1 < 5, final

instr de tip R	addition	add
	subtraction	sub
	shift left logical (ca sa)	sll
	shift right logical	srl
	logical AND	and
	logical OR	or
	logical XOR	xor
	logical XNOR	xnor
instr de tip I	add immediate	addi
	load word	lw
	store word	sw
	branch on equal	beq
	branch on greater than	bgt
	branch on less than	blt
instr. de tip J		jump j



add

Descriere	Adună 2 reg și pune în al 3-lea
Operație	$d ← $s + $t; PC ← PC + 2
Sintaxă	add $d, $s, $t
Format	000_sss_ttt_ddd_0_001

add $2, $3, $4

B"000_011_100_010_0_001

sule

Descriere	Scade 2 reg și pune în al 3-lea
Operație	$d ← $s - $t; PC ← PC + 2
Sintaxă	sule $d, $s, $t
Format	000_sss_ttt_ddd_0_010

sule $3, $4, $5

B"000_100_101_011_0_010

sll

Descriere	Deplasează la stânga cu sa
Operație	$d ← $t << sa; PC ← PC + 2
Sintaxă	sll $d, $t, sa
Format	000_sss_ttt_ddd_sa_011

sll $5, $3, 1

000_000_011_101_1_011

srl

Descriere	Deplasare la dreapta cu sa
Operație	$d ← $t >> sa
Sintaxă	srl $d, $t, sa
format	000_111_ttt_ddd_sa_100

srl $6, $5, 1
000_000_101_110_1 100

and

Descriere	AND logic pe 2 reg; mem ul 3-lea
Operație	$d ← $s & $t; PC ← PC+2
Sintaxă	and $d, $s, $t
Format	000_sss_ttt_ddd_0_101

and $3, $1, $2
000_001_010_011_0_101

or

Descriere	sau logic pe 2 reg, mem ul 3-lea
Operație	$d ← $s | $t, PC ← PC+2
Sintaxă	or $d, $s, $t
format	000_sss_ttt_ddd_0_110

or $4, $1, $5
000_001_101_100_0_110

xor

Descriere	xor logic pe 2 reg, mem ul 3-lea
Operație	$d ← $s ^ $t, PC ← PC+2
Sintaxă	xor $d, $s, $t
Format	000_sss_ttt_ddd_0_111

xor $5, $1, $2
000_001_010_101_0_111

xnor

Descriere	xnor logic pe 2 reg, mem ul 3-lea
Operație	$d ← $s xnor $t, PC ← PC+2
Sintaxă	xnor $d, $s, $t
Format	000_sss_ttt_ddd_0_000

xnor $6, $5, $4
000_101_100_110_0_000

addi

addi $3, $4, 5
110_100-011-0000101

Descriere	adunare imediată cu o Const
Operație	$t ← $s + imm; PC ← PC+2
Sintaxă	addi $t, $s, imm
Format	110_sss_ttt_iiiiiii

lw

lw $5, offset($6)
001_110_101_xxxxxxx

Descriere	un cuvânt încărcat în reg
Operație	$t ← MEM[$s+offset]; PC ← PC+2
Sintaxă	lw $t, offset($s)
Format	001_sss_ttt_iiiiiii

sw

sw offset($6), $5
010_101_110_xxxxxxx

Descriere	un reg stocat în memorie
Operație	MEM[$t+offset] ← $s; PC ← PC+2
Sintaxă	sw offset($t), $s
Format	010_sss_ttt_iiiiiii

beq

Descriere	Salt condiționat dacă este = Rotire 2 Reg
Operație	if $s == $t then PC ← PC+2+(offset<<1) else PC ← PC+2;
Sintaxă	beq $s, $t, offset
Format	011_sss_ttt_iiiiiii

beq $6, $7, offset
011_110_111_offset

bgt

Descriere	Salt cond. dacă reg a mai mare reg b
Operație	if (\$s > \$t) then PC ← PC+2+(offset<<1) else PC ← PC+2;
Sintaxă	bgt \$s, \$t, offset
Format	100-ssss-tttt-iiiiiiii

bgt \$5, \$7, offset
100-101-111-offset

blt

Descriere	Salt condiționat dacă reg a mai mic decât b
Operație	if (\$s < \$t) then PC ← PC+2+offset(<<1) else PC ← PC+2;
Sintaxă	blt \$5, \$t, offset
Format	101-sss1-~~tttt~~-iiiiiiii

blt \$6, \$7, offset
101-110-111-offset

j

Descriere	Salt la adresă
Operație	PC ← (PC+2) & 0xf0000000 | (target<<1)
Sintaxă	j target
Format	111-iiiiiiiiiiiiii

j offset
111-offset