Procesorul MIPS, pipeline
– versiune pe 16 biți –

Student: Marcu Ariana-Mălina

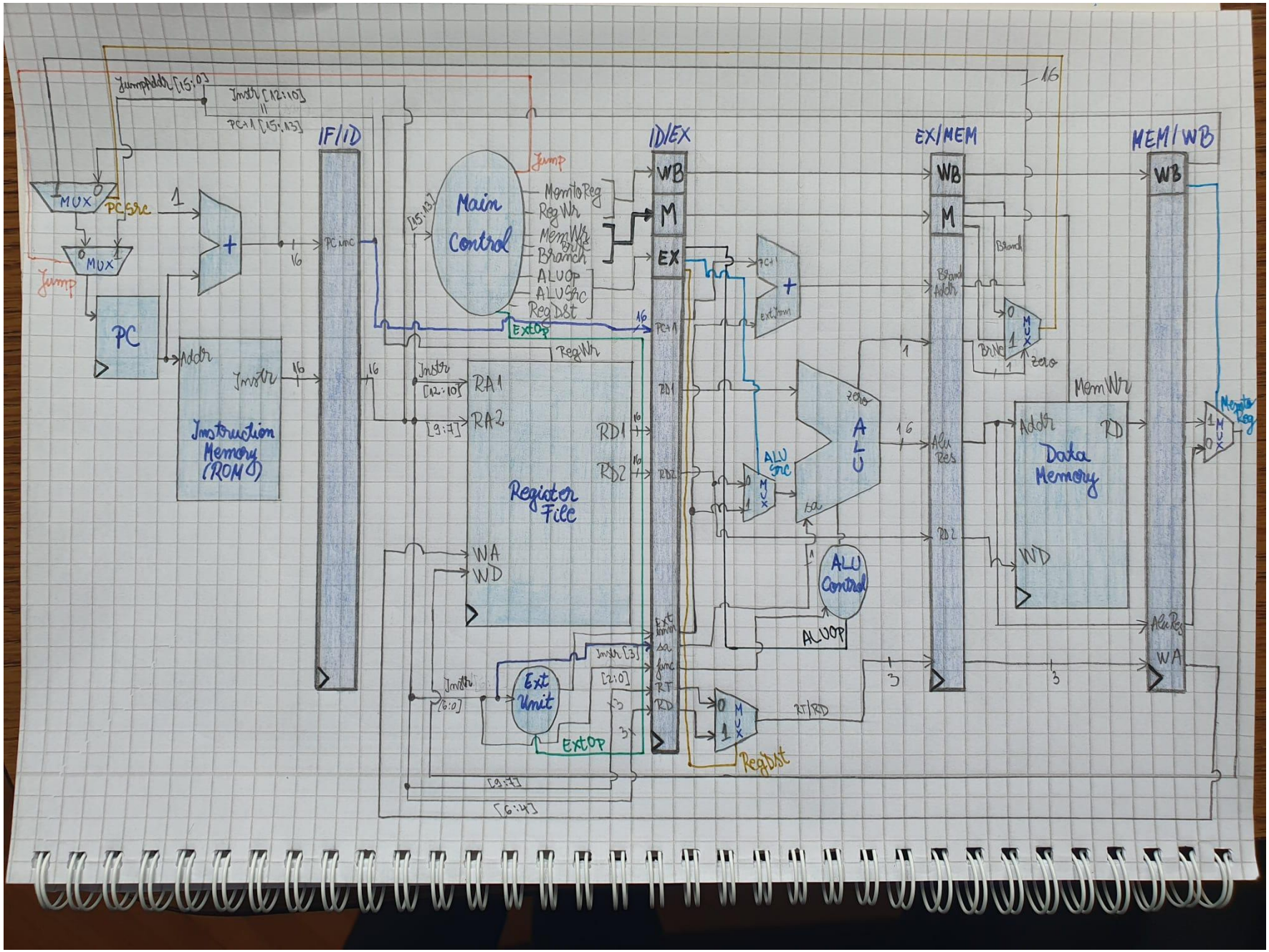
Grupa: 30222

1. Configurare registre MIPS16 Pipeline

Se introduc pe coloane numele utilizate în codul VHDL pentru semnalele de date și control implementate ca registre, pe categorii. Se introduce în paranteză dimensiunea în biți.

IF/ID (31:0)	ID/EX (83:0)	EX/MEM (56:0)	MEM/WB (36:0)
Instruction_IF_ID (16)	PCinc_ID_EX (16)	BranchAddress_EX_MEM (16)	MemData_MEM_WB (16)
PCinc_IF_ID (16)	RD1_ID_EX (16)	ALURes_EX_MEM (16)	ALURes_MEM_WB (16)
	RD2_ID_EX (16)	RD2_EX_MEM (16)	rd_MEM_WB (3)
	Ext_imm_ID_EX (16)	rd_EX_MEM (3)	MemtoReg_MEM_WB (1)
	func_ID_EX (3)	zero_EX_MEM (1)	RegWrite_MEM_WB (1)
	rt_ID_EX (3)	MemtoReg_EX_MEM (1)	
	rd_ID_EX (3)	RegWrite_EX_MEM (1)	
	ALUOp_ID_EX (3)	MemWrite_EX_MEM (1)	
	sa_ID_EX (1)	Branch_EX_MEM (1)	
	MemtoReg_ID_EX (1)	BrNe_EX_MEM (1)	
	RegWrite_ID_EX (1)		
	MemWrite_ID_EX (1)		
	Branch_ID_EX (1)		
	BrNe_ID_EX (1)		
	ALUSrc_ID_EX (1)		
	RegDst_ID_EX (1)		

2. Schema MIPS 16 Pipeline



3. Tabel cu diagrama de executie pipeline, determinarea hazardurilor si solutia modificata pentru eliminarea hazardurilor

Adr.	Instructiune/C	CC1	CC2	CC3	CC4	CC5	CC6	CC5	CC8	CC9	CC10	CC11	CC12	CC13	CC14	CC15	CC16	CC17	CC18	CC19	CC20	CC21
0	**add $1, $0, $0**	IF	ID	EX	MEM	WB																
1	**lw $2, 0($0)**		IF	IF	IF	IF	IF															
2	**lw $3, 1($0)**			IF	ID	EX	MEM	WB														
3	**lw $4, 2($0)**				IF	ID	EX	MEM	WB													
4	**add $5, $0, $0**					IF	ID	EX	MEM	WB												
5	**beq $2, $3, 3**						IF	ID	EX	MEM	WB											
6	**add $1, $1, $2**							IF	ID	EX	MEM	WB										
7	**addi $2, $2, 1**								IF	ID	EX	MEM	WB									
8	**j 5**									IF	ID	EX	MEM	WB								
9	**beq $4, $1, 6**										IF	ID	EX	MEM	WB							
10	**slt $5, $4, $1**											IF	ID	EX	MEM	WB						
11	**bne $5, $0, 0**												IF	ID	EX	MEM	WB					
12	**sub $1, $1, $4**													IF	ID	EX	MEM	WB				
13	**j 9**														IF	ID	EX	MEM	WB			
14	**sub $4, $4, $1**															IF	ID	EX	MEM	WB		
15	**j 9**																IF	ID	EX	MEM	WB	
16	**sw $4, 3($0)**																	IF	ID	EX	MEM	WB

Asa arata programul inainte de rezolvarea hazardurilor. Se observa in principal hazardurile de control, care apar in cazul instructiunilor de salt conditionat (in cazul meu beq si bne), asa ca am ales cea mai simpla metoda(dar nu cea mai eficienta), de a adauga 3 NoOp-uri dupa fiecare din acestea. In cazul instructiunilor de salt neconditionat(jump), acestea se finalizeaza cand sunt la etajul ID, deci intra in executie instructiunea imediat urmatoare => solutia cea mai simpla => adaug un singur NoOp dupa j.

Hazardurile structurale apar atunci cand, pe acelasi ciclu de ceas, instructiuni aflate pe etaje diferite din pipeline, incearca sa foloseasca aceeasi componenta din procesor. Ca solutie posibila in evitarea eventualelor hazarduri structurale aici ar fi modificarea blocului de registre RF astfel incat scrierea sa se faca in mijlocul perioadei de ceas (falling_edge(CLK)).

Ca si hazard de date(RAW), regasim cazul instructiunii SLT. Acest tip de hazard apare la instructiunile care folosesc ca operanzi sursa, valori care sunt in curs de calculare de catre instructiuni anterioare, nefiind actualizate inca, aici fiind vorba despre registrul $5. Ca si solutie, am adaugat 2 NoOp, pentru a intarzia executia instructiunii urmatoare (BEQ) cu doua cicluri de ceas.

Varianta de program fără hazarduri

Adr.	Instrucțiune/Clk	CC1	CC2	CC3	CC4	CC5	CC6	CC5	CC8	CC9	CC10	CC11	CC12	CC13	CC14	CC15	CC16	CC17	CC18	CC19	CC20	CC21	CC22	CC23	CC24	CC25	CC26	CC27	CC28	CC29	CC30	CC31	CC32	CC33	CC34	CC35
0	**add $1, $0, $0**	IF	ID	EX	MEM	WB																														
1	**lw $2, 0($0)**		IF	ID	EX	MEM	WB $2																														
2	**lw $3, 1($0)**			IF	ID	EX	MEM	WB $3																													
3	**lw $4, 2($0)**				IF	ID	EX	MEM	WB																												
4	**add $5, $0, $0**					IF	ID	EX	MEM	WB																											
5	**beq $2, $3, 3**						IF	ID($2,$3)	EX	MEM(C)	WB																											
6	NoOp							IF	ID	EX	MEM	WB																								
7	NoOp								IF	ID	EX	MEM	WB																							
8	NoOp									IF	ID	EX	MEM	WB																						
9	**add $1, $1, $2**										IF	ID	EX	MEM	WB $1																					
10	**addi $2, $2, 1**											IF	ID	EX	MEM	WB																					
11	**j 5**												IF	ID(C)	EX	MEM	WB																				
12	NoOp													IF	ID	EX	MEM	WB																		
13	**beq $4, $1, 6**														IF	ID($4,$1)	EX	MEM(C)	WB																	
14	NoOp															IF	ID	EX	MEM	WB																
15	NoOp																IF	ID	EX	MEM	WB															
16	NoOp																	IF	ID	EX	MEM	WB														
17	**slt $5, $4, $1**																		IF	ID	EX	MEM	WB $5													
18	NoOp																			IF	ID	EX	MEM	WB												
19	NoOp																				IF	ID	EX	MEM	WB											
20	**bne $5, $0, 0**																					IF	ID($5,$0)	EX	MEM(C)	WB										
21	NoOp																						IF	ID	EX	MEM	WB									
22	NoOp																							IF	ID	EX	MEM	WB								
23	NoOp																								IF	ID	EX	MEM	WB							
24	**sub $1, $1, $4**																									IF	ID	EX	MEM	WB $1						
25	**j 9**																										IF	ID(C)	EX	MEM	WB					
26	NoOp																											IF	ID	EX	MEM	WB				
27	**sub $4, $4, $1**																												IF	ID	EX	MEM	WB $4			
28	**j 9**																													IF	ID(C)	EX	MEM	WB		
29	NoOp																														IF	ID	EX	MEM	WB	
30	**sw $4, 3($0)**																															IF	ID	EX	MEM	WB

Dupa actualizarea offset/imm, ale instructiunilor de salt, codul complet functional va arata astfel:

```
B"000_000_000_001_0_011"    --0   add $1, $0, $0          X"0013"
B"101_000_010_0000000"      --1    lw $2, 0($0)          X"A100"
B"101_000_011_0000001"      --2    lw $3, 1($0)          X"A181"
B"101_000_100_0000010"      --3    lw $4, 2($0)          X"A202"
B"000_000_000_101_0_011"    --4    add $5, $0, $0        X"0053"
B"110_010_011_0000111"      --5   beq $2, $3, 7          X"C987"
B"000_000_000_000_0_000"    --6   NoOp (add $0, $0, $0)  X"0000"
B"000_000_000_000_0_000"    --7   NoOp (add $0, $0, $0)  X"0000"
B"000_000_000_000_0_000"    --8   NoOp (add $0, $0, $0)  X"0000"
B"000_001_010_001_0_011"    --9   add $1, $1, $2         X"0513"
B"001_010_010_0000001"      --10   addi $2, $2, 1        X"2901"
B"111_0000000000101"        --11   j 5                   X"E005"
B"000_000_000_000_0_000"    --12  NoOp (add $0, $0, $0)  X"0000"
B"110_100_001_0010000"      --13  beq $4, $1, 16         X"D090"
B"000_000_000_000_0_000"    --14  NoOp (add $0, $0, $0)  X"0000"
B"000_000_000_000_0_000"    --15  NoOp (add $0, $0, $0)  X"0000"
B"000_000_000_000_0_000"    --16  NoOp (add $0, $0, $0)  X"0000"
B"000_100_001_101_0_101"    --17  slt $5, $4, $1         X"10D5"
B"000_000_000_000_0_000"    --18  NoOp (add $0, $0, $0)  X"0000"
B"000_000_000_000_0_000"    --19  NoOp (add $0, $0, $0)  X"0000"
B"100_101_000_0000011"      --20  bne $5, $0, 3          X"9403"
B"000_000_000_000_0_000"    --21  NoOp (add $0, $0, $0)  X"0000"
B"000_000_000_000_0_000"    --22  NoOp (add $0, $0, $0)  X"0000"
B"000_000_000_000_0_000"    --23  NoOp (add $0, $0, $0)  X"0000"
B"000_001_100_001_0_000"    --24  sub $1, $1, $4         X"0610"
B"111_0000000001101"        --25   j 13                  X"E00D"
B"000_000_000_000_0_000"    --26  NoOp (add $0, $0, $0)  X"0000"
B"000_100_001_100_0_000"    --27  sub $4, $4, $1         X"10C0"
B"111_0000000001101"        --28   j 13                  X"E00D"
B"000_000_000_000_0_000"    --29  NoOp (add $0, $0, $0)  X"0000"
B"010_000_100_0000011"      --30  sw $4, 3($0)           X"4203"
```

In final, programul a fost testat si functioneaza complet toate instructiunile, realizand cel mai mare divizor comun dintre suma numerelor din intervalul [3, 8) si numarul dat din memorie, pe care l-am initializat cu 5 => 3+4+5+6+7=25. CMMDC=5