DOCUMENTATIE

PROCESOR MIPS-PIPELINE

Irimes David
Gr. 7

Transformarea unui miniprocesor MIPS monociclu într-unul pipeline necesită adăugarea de registre între etapele de execuție a instrucțiunilor. Aceste registre permit paralelizarea execuției instrucțiunilor (viteza marita), fara a modifica in rest, logica instructiunilor. Cu toate acestea, pot aparea hazarduri de mai multe tipuri, care trebuie tratate.

Tabel registre intermediare:

IF/ID[63:0]	ID/EX[162:0]	EX/MEM[105:0]	MEM/WB[70:0]
Instr[63:32]	Instr(25:21)[162:158]	Branch[104]	RegWrite[70]
Pcp4[31:0]	RegDst[157]	MemWrite[103]	MemtoReg[69]
	ALUSrc[156]	RegWrite[102]	MemData[68:37]
	ALUOp[155:153]	MemtoReg[101]	ALURes1[36:5]
	Branch[152]	BranchAddr[100:69]	EX/MEM(4:0)[4:0]
	MemWrite[151]	ALURes[68:37]	
	RegWrite[150]	RD2[36:5]	
	MemtoReg[149]	WriteAddr[4:0]	
	Pcp4[148:117]	Zero[105]	
	RD1[116:85]		
	RD2[84:53]		
	sa[52:48]		
	Ext_Imm[47:16]		
	func[15:10]		
	Instr(20:16)[9:5]		
	Instr(15:11)[4:0]		

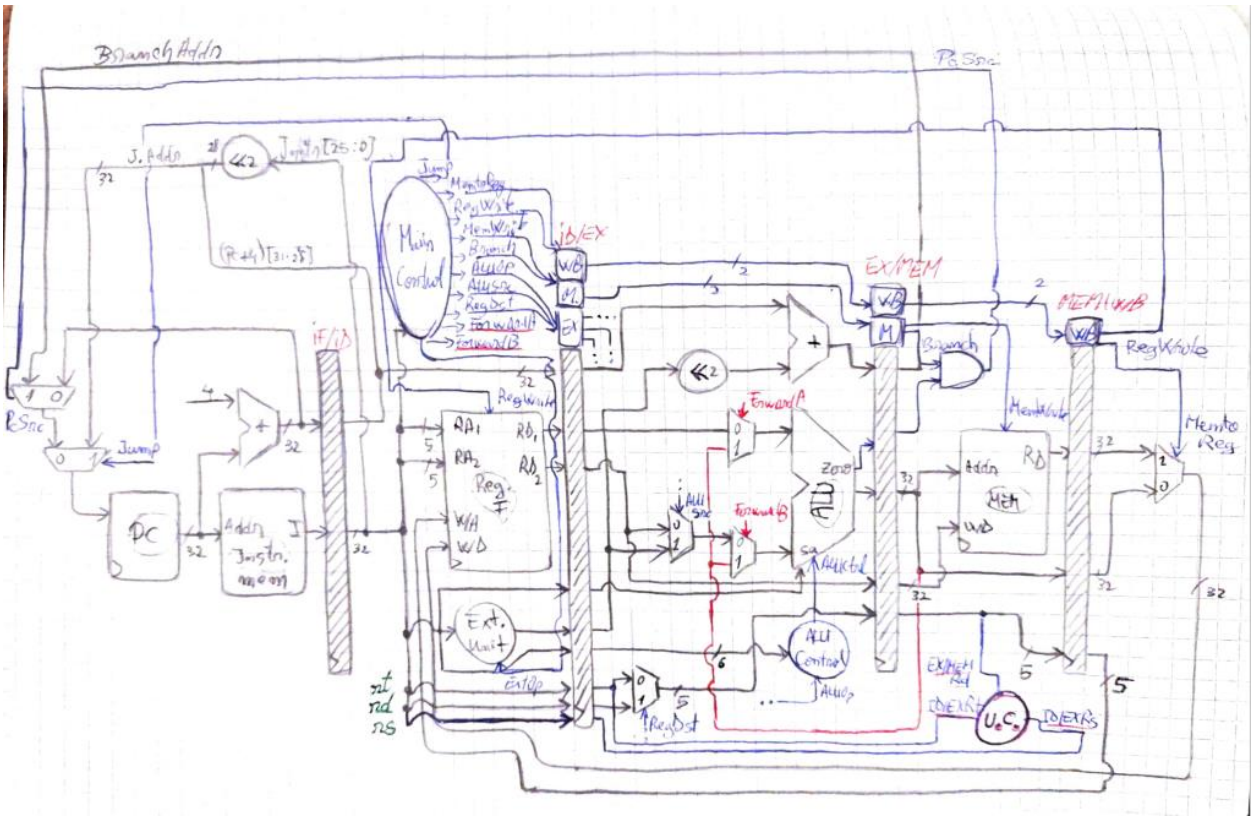
Cod ASM_MIPS initial (calculeaza termenii sirului lui Fibonacci):

00	ADDI $1, $0, 0
01	ADDI $2, $0, 1
02,	ADDI $3, $0,
03	ADDI $4, $0, 4
04	SW $2, 0($4)
05	SW $1, 0($3) ←hazard structural
06	LW $1, 0($3)
07	LW $2, 0($4)
08	ADD $5, $1, $2
09	ADD $1, $0, $2
10	ADD $2, $0, $5 ←hazard de date
11	J 8 ←hazard de control

Cod ASM_MIPS dupa tratarea hazardurilor (hazardurile de date pe care le consideram sunt cele de tip RAW → read after write):

00	ADDI $1, $0, 0
01	ADDI $2, $0, 1
02	ADDI $3, $0, 0
03	ADDI $4, $0, 4
04	SW $2, 0($4)
05	SW $1, 0($3)
06	LW $1, 0($3)
07	LW $2, 0($4)
08	NOOP
09	NOOP
10	ADD $5, $1, $2
11	ADD $1, $0, $2
12	J 10
13	ADD $2, $0, $5

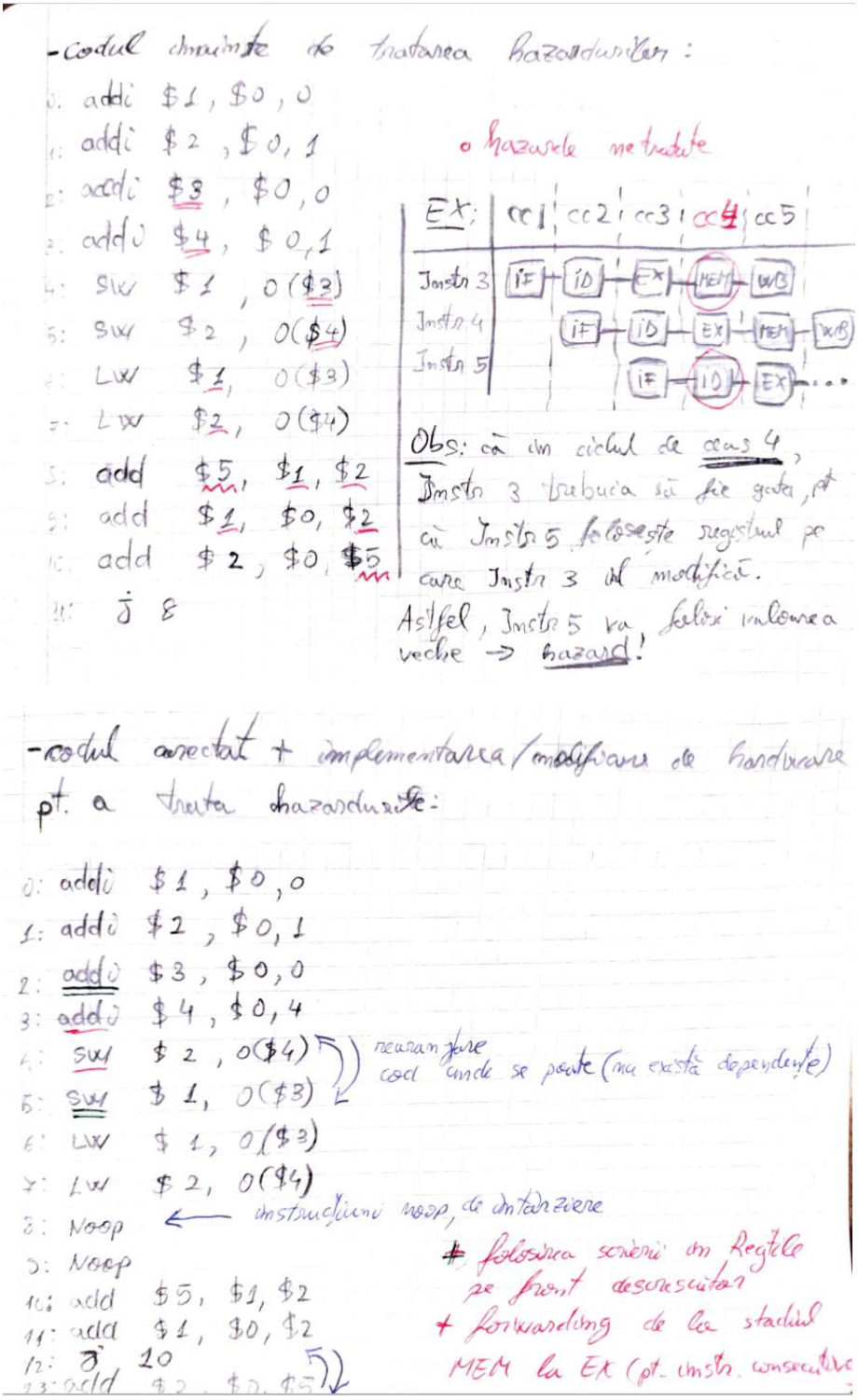
Arhitectura MIPS dupa tratarea hazardurilor:

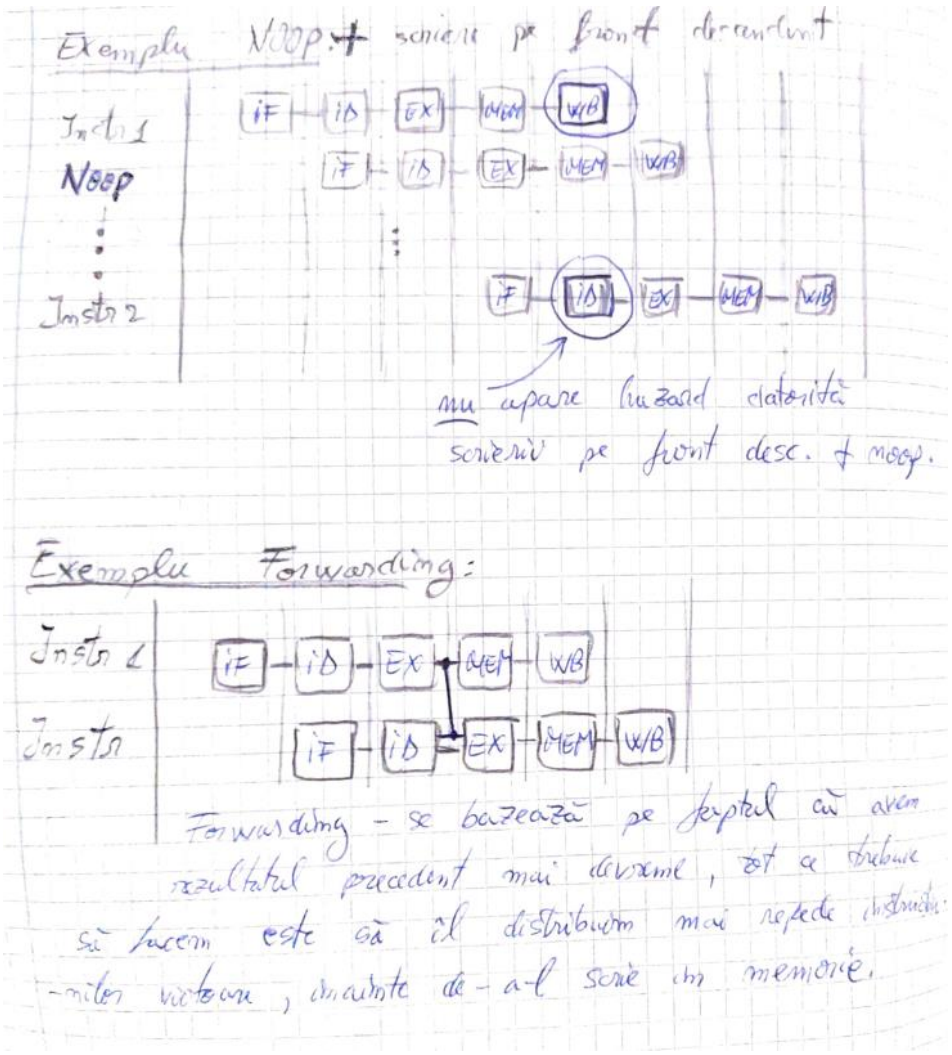


Hazardurile de date și de control sunt rezolvate prin utilizarea de instrucțiuni NOOP și prin rearanjarea instrucțiunilor existente, astfel încât să se minimizeze dependențele de date și să se prevină condițiile de salt care ar putea afecta execuția corectă a instrucțiunilor ulterioare.

Hazardurile structurale sunt abordate prin modificarea modului de scriere în register file, astfel încât operațiile de scriere să fie realizate pe frontul descrescător al semnalului de ceas, evitând astfel conflictele de acces la resursele hardware comune (mai explicit, cand o instructiune se afla in etapa WB, si o instructiune succesiva care foloseste acelasi registru se afla in ID, datele folosite nu o sa fie cele vechi, hazardul se evita). În plus, sunt adăugate legături pentru forwarding între etapele de acces la memorie (MEM) și execuție (EX) pentru a facilita transferul rapid al datelor între instrucțiunile consecutive, contribuind la evitarea hazardurilor de date(obs: se rezolva hazardul dintre instructiuniile strict consecutive; in cazul in care aveam o instructiune in EX si una in WB, nici frontul descrescător nici forwarding nu ar fi functionat, in acest caz fiind nevoiti sa folosim NOOP; desi se putea implementa forwarding si de la WB la EX, dar arhitectura se complica prea mult).

Exemplificare si tratare hazarduri:





Tabel Excel:

Coloană1	Coloană2	Coloană3	Coloană4	Coloană5	Coloană6	Coloană7	Coloană8	Coloană9	Coloană10	Coloană11	Coloană12	Coloană13	Coloană14	Coloană15	Coloană16	Coloană17	Coloană18	Coloană19
INSTR	cc1	cc2	cc3	cc4	cc5	cc6	cc7	cc8	cc9	cc10	cc11	cc12	cc13	cc14	cc15	cc16	cc17	cc18
ADDI $1, $0, 0	IF	ID	EX	MEM	WB													
ADDI $2, $0, 1		IF	ID	EX	MEM	WB												
ADDI $3, $0, 0			IF	ID	EX	MEM	WB											
ADDI $4, $0, 4				IF	ID	EX	MEM	WB										
SW $2, 0($4)					IF	ID	EX	MEM	WB									
SW $1, 0($3)						IF	ID	EX	MEM	WB								
LW $1, 0($3)							IF	ID	EX	MEM	WB							
LW $2, 0($4)								IF	ID	EX	MEM	WB						
NOOP									IF	ID	EX	MEM	WB					
NOOP										IF	ID	EX	MEM	WB				
ADD $5, $1, $2											IF	ID	EX	MEM	WB			
ADD $1, $0, $2												IF	ID	EX	MEM	WB		
J 10													IF	ID	EX	MEM	WB	
ADD $2, $0, $5														IF	ID	EX	MEM	WB