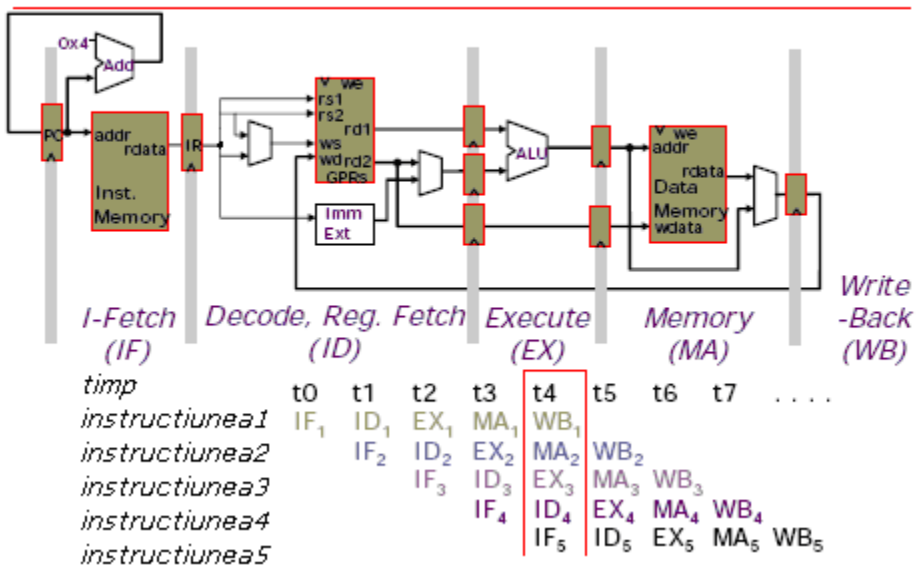
Solutii pentru Rezolvarea Hazardelor in Unitatea de Executie in Banda de Asamblare

http://www.csg.csail.mit.edu/6.823

Unitatea de Executie in Banda de Asamblare cu 5 etaje.



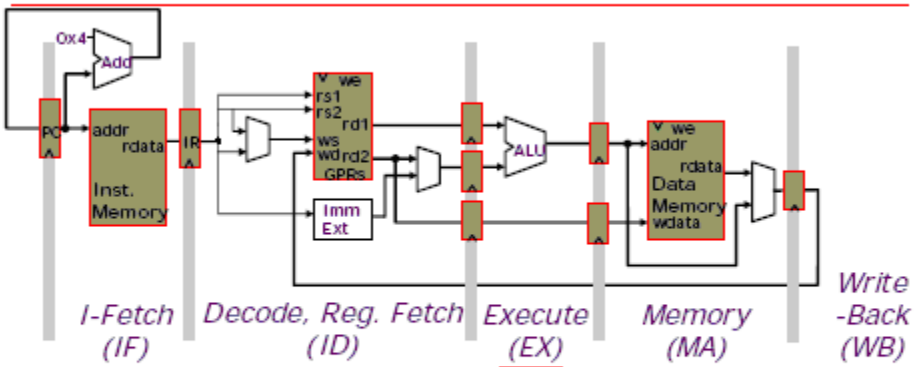
timp	t0	t1	t2	t3	t4	t5	t6	t7	. . . .
instructiunea1	IF_1	ID_1	EX_1	MA_1	WB_1				
instructiunea2		IF_2	ID_2	EX_2	MA_2	WB_2			
instructiunea3			IF_3	ID_3	EX_3	MA_3	WB_3		
instructiunea4				IF_4	ID_4	EX_4	MA_4	WB_4	
instructiunea5					IF_5	ID_5	EX_5	MA_5	WB_5

Se fac urmatoarele aprecieri privind intarzierile la nivelul diferitelor etaje ale benzii de asamblare:

$$t_{IM} \approx t_{RF} \approx t_{ALU} \approx t_{DM} \approx t_{RW}$$

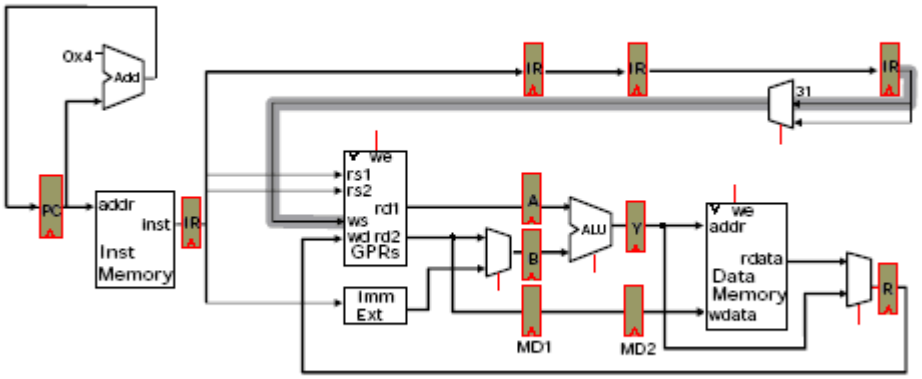
Unitatea de executie in Banda de Asamblare cu 5 etaje

Diagrama Utilizarii Resurselor



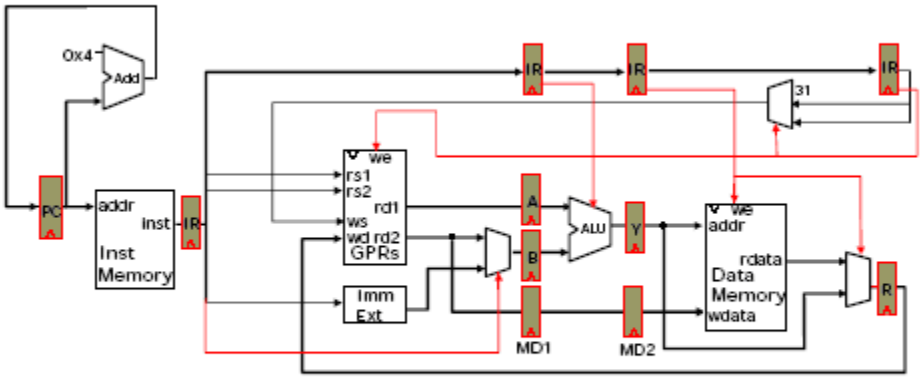
Resurse	*timp*	t0	t1	t2	t3	t4	t5	t6	t7	. . . .
	IF	I_1	I_2	I_3	I_4	I_5				
	ID		I_1	I_2	I_3	I_4	I_5			
	EX			I_1	I_2	I_3	I_4	I_5		
	MA				I_1	I_2	I_3	I_4	I_5	
	WB					I_1	I_2	I_3	I_4	I_5

Executia Instructiunilor UAL in Banda de Asamblare



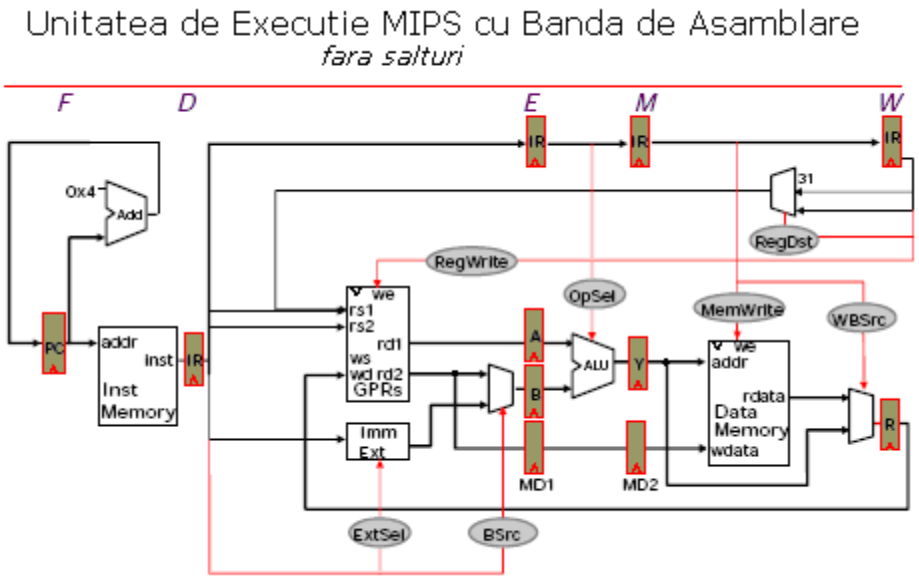
Pentru fiecare etaj este necesar un Registru al Instructiunii (IR)

Registrele Instructiunilor si Punctele de Control



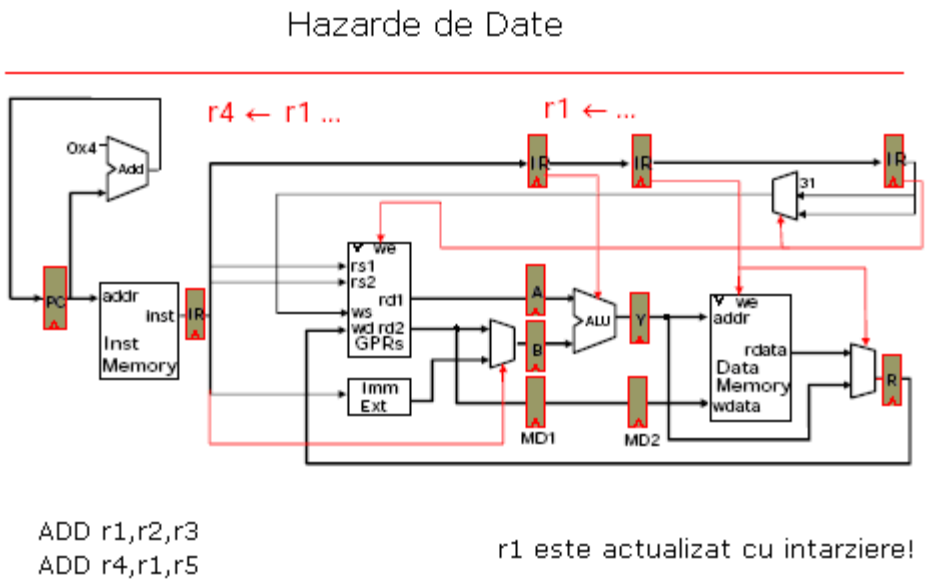
Punctele de aplicare a semnalelor de comanda pentru:

- instructiunile UAL
- instructiunile Load/Store
- Write back



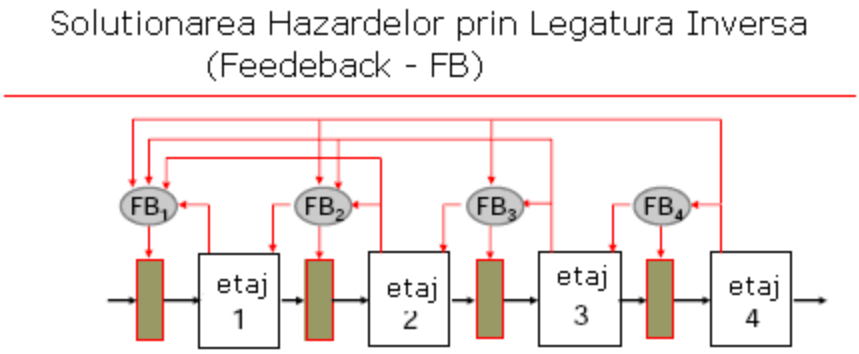
Interactiunile instructiunilor in Banda de Asamblare (BA)pot conduce la aparitia hazardelor:

- *structurale*: o instructiune din BA solicita o resursa hardware utilizata de catre o alta instructiune;
- *de date*: o instructiune poate produce date care sunt utilizate de catre alte instructiuni din BA;
- *de control*: intr-un caz extrem o instructiune poate stabili instructiunea care urmeaza sa se execute.

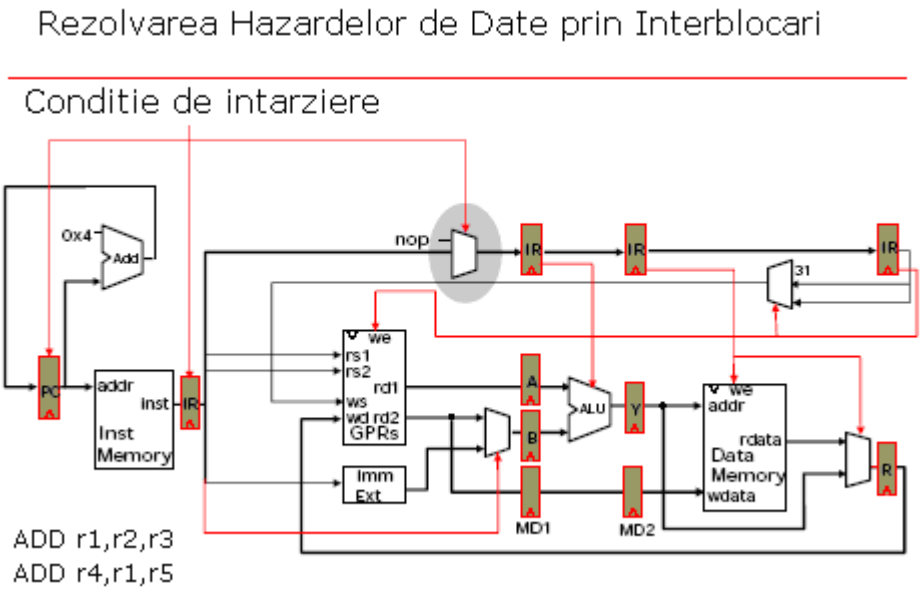


Solutii pentru Hazardele de Date:

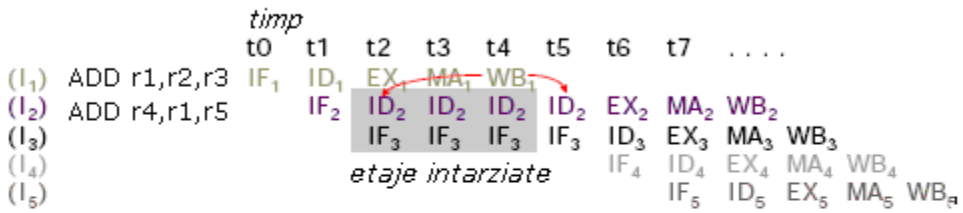
- **Interblocari:** se blocheaza etajele anterioare ale benzii de asamblare pana cand data devine disponibila;
- **Ocolire:** daca data este disponibila intr-un etaj al unitatii de executie se asigura o cale de ocolore (bypass) pentru a o aduce la etajul corespunzator;
- **Anulare:** se speculeaza asupra rezolvarii hazardului, iar in cazul unei speculatii eronate instructiunea este anulata/anihilata.

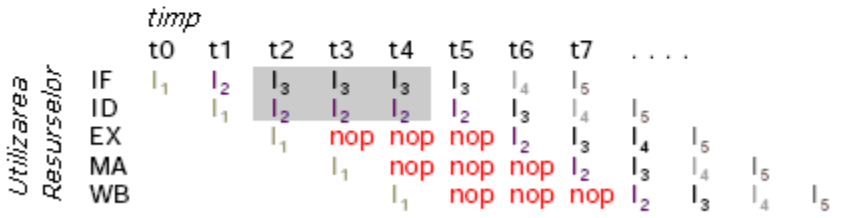


- Se detecteaza un hazard si se realizeaza o reactie/legatura inversa, catre etajele precedente pentru a intarzia sau anula instructiunile;
- Controlarea unei benzi de asamblare in aceasta maniera este realizabila numai daca instructiunea de la etajul i+1 se poate termina fara interferente din partea instructiunilor de la etajele 1 la i; in caz contrar pot aparea blocari.



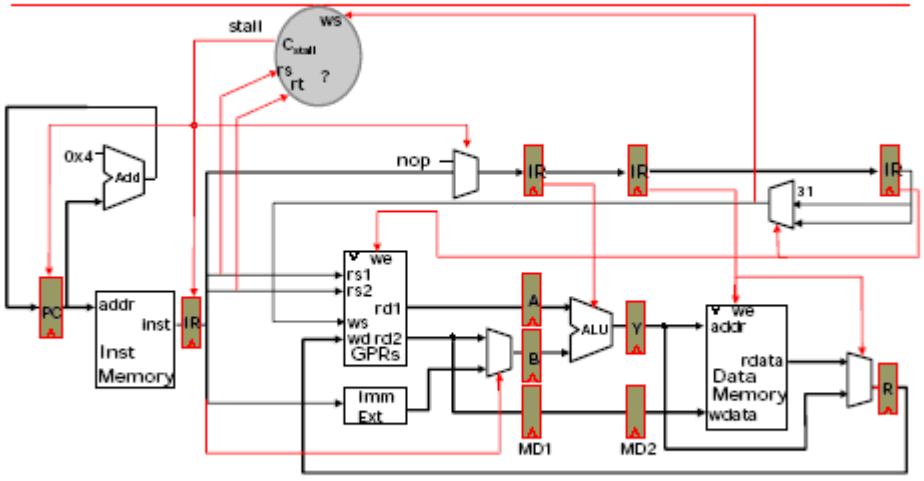
Etaje intarziate si Pipeline Bubbles





nop ⇒ *pipeline bubble*

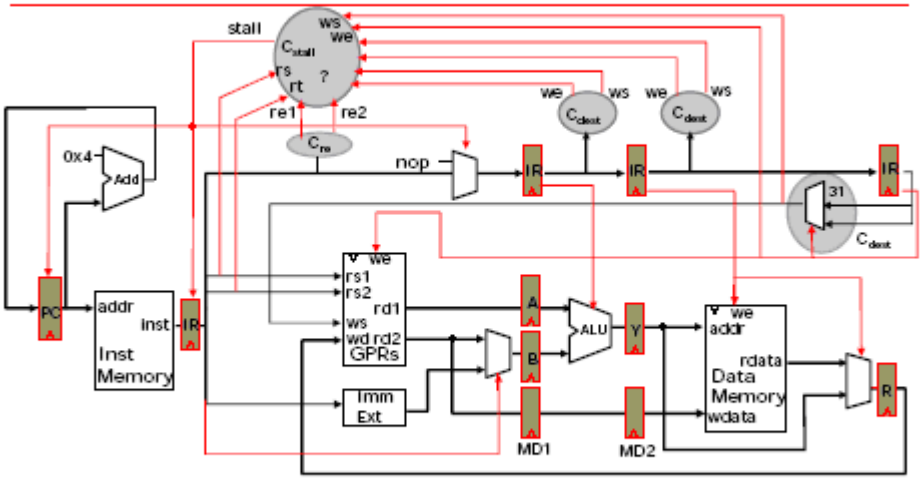
Logica de Control pentru Interblocare



Compara registrele sursa ale instructiunii aflate in etajul ID cu registrul destinatie alinstructiunilor neterminate

Logica de Control a Interblocarilor

se ignora salturile si ramificarile



Trebuie introduse intarzieri de fiecare data atunci cand campul rs coincide cu rd?

citirea unui registru nu este necesara in cazul fiecarei instructiuni ⇒ we

scrierea unui registru nu este necesara in cazul fiecarei instructiuni ⇒ re

Registre sursa si Registre Destinatie

Tip R:

op	rs	rt	rd		func

Tip I:

op	rs	rt	imediat 16

Tip J:

op	imediat 26

		surse	*destinatie*
ALU	rd ← (rs) func (rt)	rs, rt	rd
ALUi	rt ← (rs) op im	rs	rt
LW	rt ← M [(rs) + im]	rs	rt
SW	M [(rs) + im] ← (rt)	rs, rt	
BZ	*cond* (rs)		
	adevarat: PC ← (PC) + im	rs	
	fals: PC ← (PC) + 4	rs	
J	PC ← (PC) + imm		
JAL	r31 ← (PC), PC ← (PC) + im		31
JR	PC ← (rs)	rs	
JALR	r31 ← (PC), PC ← (rs)	rs	31

Generarea Semnalelor pentru Intarziere (Stall)

C_{dest}

ws = *Case* opcode
- ALU ⇒ rd
- ALUi, LW ⇒ rt
- JAL, JALR ⇒ R31

we = *Case* opcode
- ALU, ALUi, LW ⇒ (ws ≠ 0)
- JAL, JALR ⇒ on
- … ⇒ off

C_{re}

re1 = *Case* opcode
- ALU, ALUi, LW, SW, BZ, JR, JALR ⇒ on
- J, JAL ⇒ off

re2 = *Case* opcode
- ALU, SW ⇒ on
- … ⇒ off

C_{stall}

$$stall = ((rs_D = ws_E).we_E + (rs_D = ws_M).we_M + (rs_D = ws_W).we_W) . re1_D + ((rt_D = ws_E).we_E + (rt_D = ws_M).we_M + (rt_D = ws_W).we_W) . re2_D$$

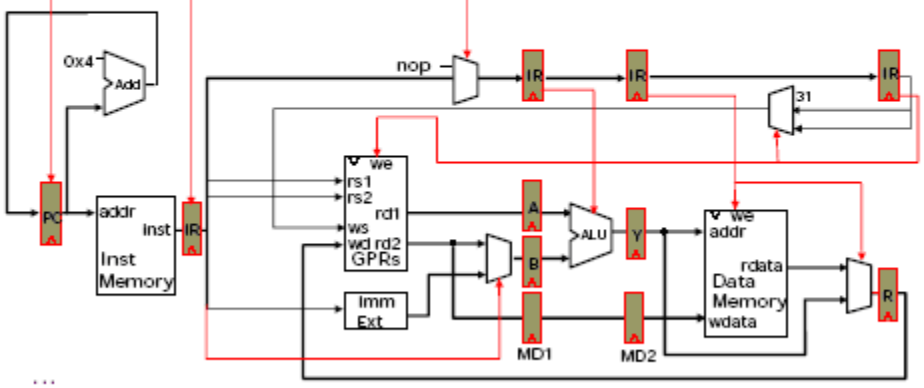
Tratarea este incompleta

Hazarde provocate de Instructiunile Incarca/Load si Memoreaza/Store

Conditie de Stall

Ce se intampla daca

(r1)+7 = (r3)+5 ?



…

M[(r1)+7] ← (r2)

r4 ← M[(r3)+5]

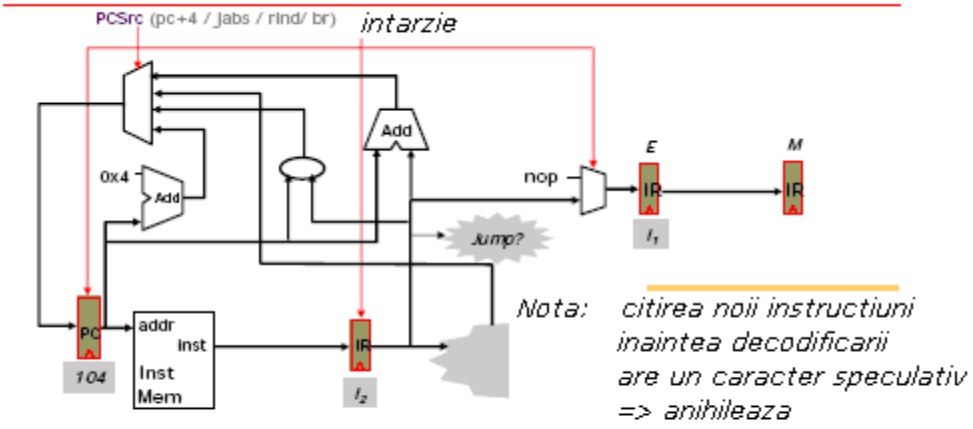
Exista in aceasta secventa un posibil hazard de date?

```
...
M[(r1)+7] ← (r2)
r4 ← M[(r3)+5]
...
```

(r1)+7 = (r3)+5 ⇒ *hazard de date*

Hazardul este evitat, deoarece sistemul de memorie termina operatia de scriere intr-un singur ciclu. Hazardele Load/Store, cand apar, sunt solutionate la nivelul hardware-lui memoriei.

Complicatii introduse de instructiunile de Salt/Jump

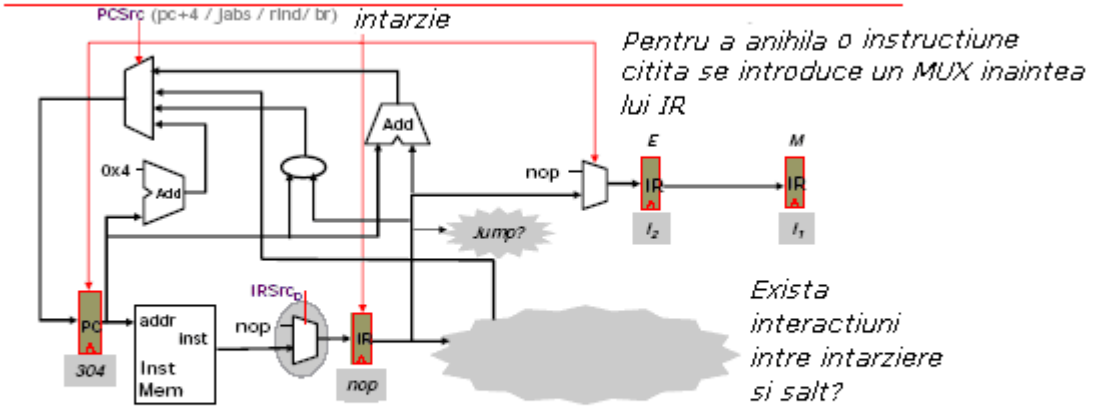


I_1	096	ADD	
I_2	100	J	200
I_3	~~104~~	~~ADD~~	*anihileaza*
I_4	304	ADD	

O instructiune de salt/jump anihileaza (nu intarzie) urmatoarea instructiune.

Cum?

Instructiuni de Salt in Banda de Asamblare



I_1	096	ADD	
I_2	100	J	200
I_3	~~104~~	~~ADD~~	*anihileaza*
I_4	304	ADD	

$IRSrc_D$ = *Case* $opcode_D$	
J, JAL	⇒ nop
...	⇒ IM

Diagramele Instructiunii Salt/Jump in Banda de asamblare

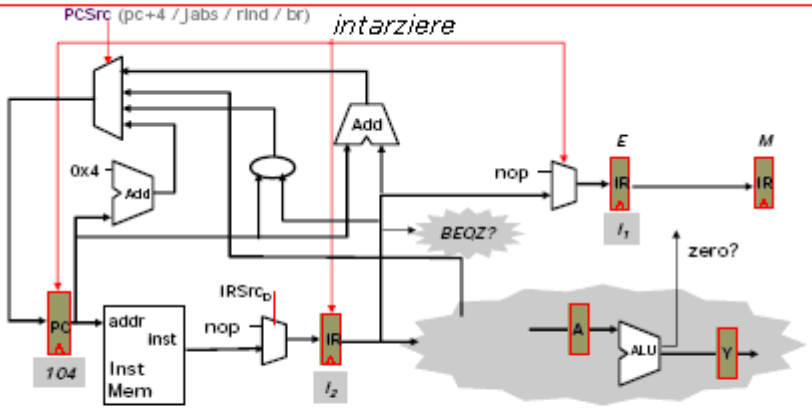
	timp t0	t1	t2	t3	t4	t5	t6	t7	. . . .
(I_1) 096: ADD	IF_1	ID_1	EX_1	MA_1	WB_1				
(I_2) 100: J 200		IF_2	ID_2	EX_2	MA_2	WB_2			
(I_3) 104: ADD			IF_3	nop	nop	nop	nop		
(I_4) 304: ADD				IF_4	ID_4	EX_4	MA_4	WB_4	

Utilizarea Resurselor

	timp t0	t1	t2	t3	t4	t5	t6	t7	. . . .
IF	I_1	I_2	I_3	I_4	I_5				
ID		I_1	I_2	nop	I_4	I_5			
EX			I_1	I_2	nop	I_4	I_5		
MA				I_1	I_2	nop	I_4	I_5	
WB					I_1	I_2	nop	I_4	I_5

nop ⇒ *pipeline bubble*

Ramificatiile Conditionate in Banda de Asamblare

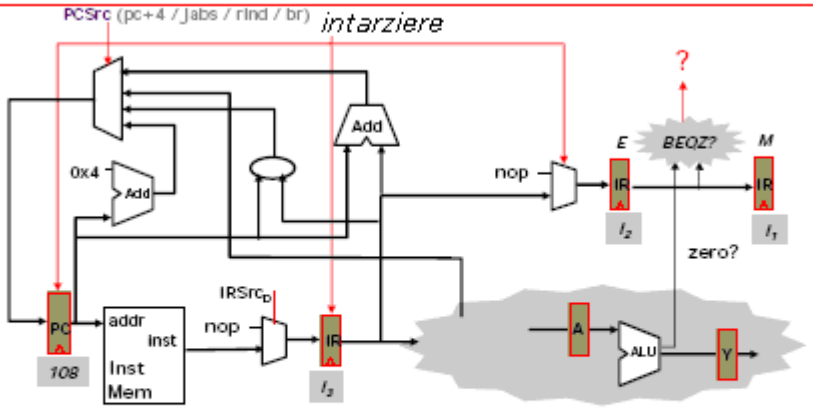


I_1	096	ADD
I_2	100	BEQZ r1 200
I_3	104	ADD
I_4	304	ADD

Conditia de Ramificare nu este cunoscuta decat in etajul de executie

ce actiune trebuie intreprinsa in etajul de decodificare?

Ramificatiile Conditionate in Banda de Asamblare

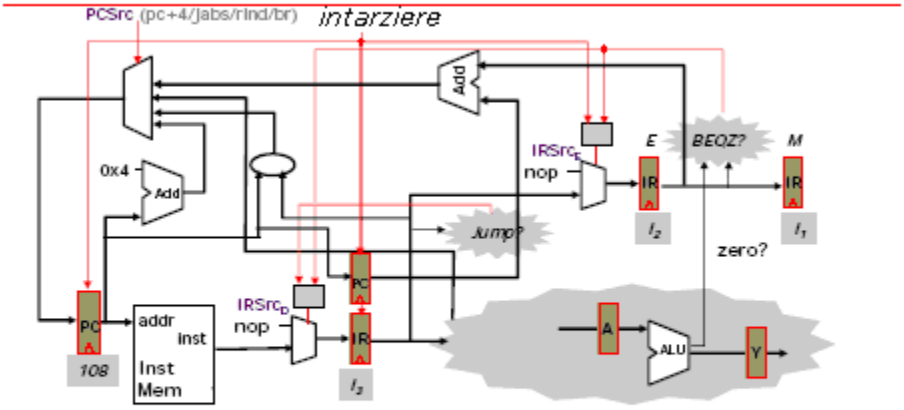


I_1	096	ADD
I_2	100	BEQZ r1 200
I_3	104	ADD
I_4	304	ADD

Daca se efectueaza ramificarea:
- anihileaza urmatoarele doua instructiuni;
- instructiunea aflata in etajul de decodificare nu este valida.

=> semnalul intarziere/stall nu este valid

Ramificatiile Conditionate in Banda de Asamblare



I_1	096	ADD
I_2	100	BEQZ r1 200
I_3	104	ADD
I_4	304	ADD

Daca se efectueaza ramificarea:
- anihileaza urmatoarele doua instructiuni;
- instructiunea aflata in etajul de decodificare nu este valida.

=> semnalul intarziere/stall nu este valid

Semnal de Intarziere/Stall nou

$$stall = (\ ((rs_D = ws_E).we_E + (rs_D = ws_M).we_M + (rs_D = ws_W).we_W).re1_D$$
$$+ ((rt_D = ws_E).we_E + (rt_D = ws_M).we_M + (rt_D = ws_W).we_W).re2_D\ )$$
$$.\ !((opcode_E = BEQZ).z + (opcode_E = BNEZ).!z)$$

Daca ramificarea este efectuata nu se introduce intarziere. De ce?

Instructiunea din etajul de decodificare este invalida

Ecuatiile Logice pentru comenzile PC si ale Multiplexoarelor atasate la IR

$PCSrc$ = *Case* $opcode_E$
- BEQZ.z, BNEZ.!z ⇒ br
- ... ⇒
  - *Case* $opcode_D$
    - J, JAL ⇒ jabs
    - JR, JALR ⇒ rind
    - ... ⇒ pc+4

$IRSrc_D$ = *Case* $opcode_E$
- BEQZ.z, BNEZ.!z ⇒ nop
- ... ⇒
  - *Case* $opcode_D$
    - J, JAL, JR, JALR ⇒ nop
    - ... ⇒ IM

$IRSrc_E$ = *Case* $opcode_E$
- BEQZ.z, BNEZ.!z ⇒ nop
- ... ⇒ $stall.nop + !stall.IR_D$

Se da prioritate instructiunii mai vechi, adica instructiunii din etajul Ex fata de instructiunea din etajul ID

Diagramele Ramificarii/Branch in Banda de Asamblare (solutionate in etajul de executie)

	timp t0	t1	t2	t3	t4	t5	t6	t7	. . . .
(I_1) 096: ADD	IF_1	ID_1	EX_1	MA_1	WB_1				
(I_2) 100: BEQZ 200		IF_2	ID_2	EX_2	MA_2	WB_2			
(I_3) 104: ADD			IF_3	ID_3	nop	nop	nop		
(I_4) 108:				IF_4	nop	nop	nop	nop	
(I_5) 304: ADD					IF_5	ID_5	EX_5	MA_5	WB_5

Utilizarea Resurselor

	timp t0	t1	t2	t3	t4	t5	t6	t7	. . . .
IF	I_1	I_2	I_3	I_4	I_5				
ID		I_1	I_2	I_3	nop	I_5			
EX			I_1	I_2	nop	nop	I_5		
MA				I_1	I_2	nop	nop	I_5	
WB					I_1	I_2	nop	nop	I_5

nop ⇒ *pipeline bubble*

Reducerea Penalizarii pentru Ramificare/Branch (solutionata in etajul de decodificare)

- Se poate elimina o bula/bubble daca se utilizeaza un comparator suplimentar in etajul de Decodificare

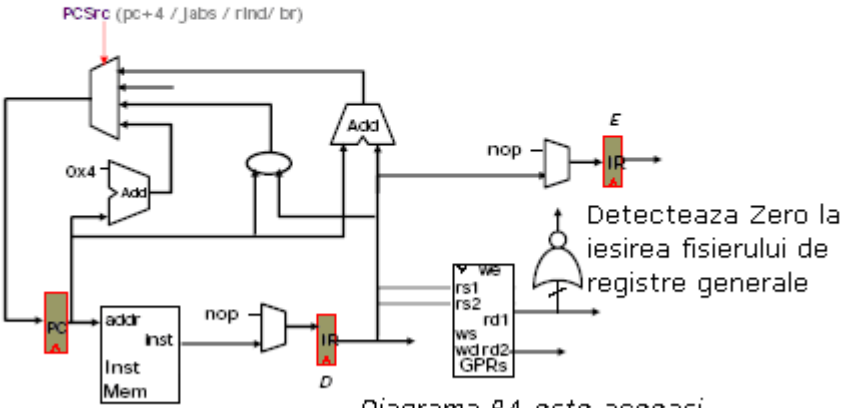


Diagrama BA este aceeasi ca si pentru salturi

Nise/Slots de Intarziere a Ramificarii (controlul hazardului prin software)

- Schimba semantica astfel incat instructiunea care urmeaza dupa salt sa fie intotdeauna executata

(ofera posibilitatea compilatorului de a plasa o instructiune utila in locul in care ar fi trebuit plasata o bula in BA)

I_1	096	ADD
I_2	100	BEQZ r1 200
I_3	104	ADD
I_4	304	ADD

Instructiunea din nisa de intarziere este executata indiferent de rezultatul ramificarii

- alte tehnici presupun predictia ramificarii, cu rezultate foarte bune in ceea ce priveste reducerea penalizarii

Ocolire/Bypass

timp	t0	t1	t2	t3	t4	t5	t6	t7	. . . .
(I_1) r1 ← r0 + 10	IF_1	ID_1	EX_1	MA_1	WB_1				
(I_2) r4 ← r1 + 17		IF_2	ID_2	ID_2	ID_2	ID_2	EX_2	MA_2	WB_2
(I_3)			IF_3	IF_3	IF_3	IF_3	ID_3	EX_3	MA_3
(I_4)			*etaje intarziate*				IF_4	ID_4	EX_4
(I_5)								IF_5	ID_5

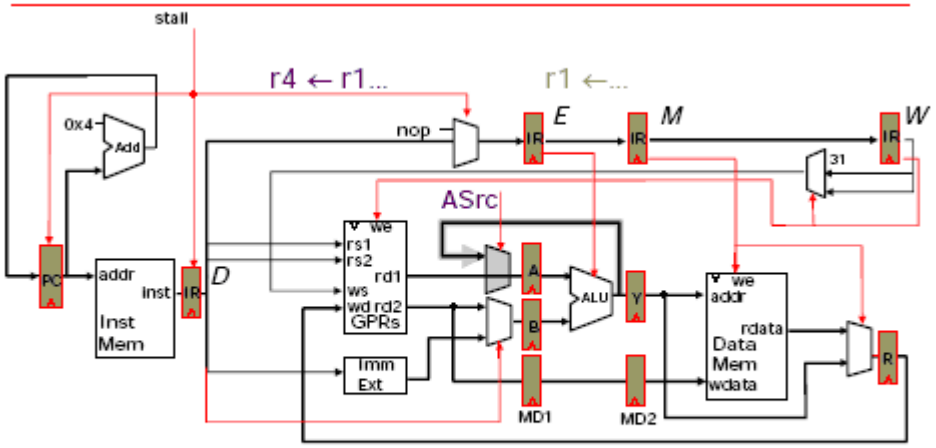
Fiecare intarzier/stall sau anihilare/kill introduce o bula in BA

=> CPI > 1

O conexiune inversa/ocolire bypass poate aduce data de la iesirea UAL la intrarea acesteia

timp	t0	t1	t2	t3	t4	t5	t6	t7	. . . .
(I_1) r1 ← r0 + 10		IF_1	ID_1	EX_1	MA_1	WB_1			
(I_2) r4 ← r1 + 17			IF_2	ID_2	EX_2	MA_2	WB_2		
(I_3)				IF_3	ID_3	EX_3	MA_3	WB_3	
(I_4)					IF_4	ID_4	EX_4	MA_4	WB_4
(I_5)						IF_5	ID_5	EX_5	MA_5 WB_5

Adaugarea Conexiunii Inverse/Ocolirii/Baypass-ului



Cand ajuta bypass-ul?

	...		
(I_1)	r1 ← r0 + 10	r1 ← M[r0 + 10]	JAL 500
(I_2)	r4 ← r1 + 17	r4 ← r1 + 17	r4 ← r31 + 17
	da	*nu*	*nu*

Semnalul Bypass

Generarea lui din semnalul Stall

$$stall = (\cancel{((rs_D = ws_E).we_E} + (rs_D = ws_M).we_M + (rs_D = ws_W).we_W).re1_D$$
$$+((rt_D = ws_E).we_E + (rt_D = ws_M).we_M + (rt_D = ws_W).we_W).re2_D)$$

ws = *Case* opcode
- ALU ⇒ rd
- ALUi, LW ⇒ rt
- JAL, JALR ⇒ R31

we = *Case* opcode
- ALU, ALUi, LW ⇒ (ws ≠ 0)
- JAL, JALR ⇒ on
- ... ⇒ off

$ASrc = (rs_D = ws_E).we_E.re1_D$

Este corect?

Nu, deoarece de acest bypass pot beneficia numai instructiunile ALU si ALUi.

Se descompune we_E in doua componente: we-bypass, we-stall

Semnalele Bypass si Stall

Se descompune we_E in doua componente: we-bypass, we-stall

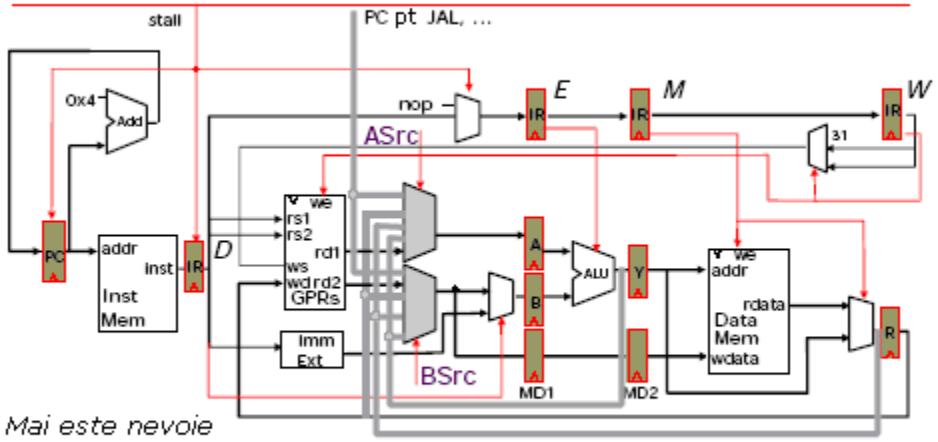
$we\text{-}bypass_E$ = *Case* $opcode_E$
- ALU, ALUi ⇒ $(ws \neq 0)$
- ... ⇒ off

$we\text{-}stall_E$ = *Case* $opcode_E$
- LW ⇒ $(ws \neq 0)$
- JAL, JALR ⇒ on
- ... ⇒ off

$ASrc = (rs_D = ws_E).we\text{-}bypass_E . re1_D$

$stall = ((rs_D = ws_E).we\text{-}stall_E + (rs_D = ws_M).we_M + (rs_D = ws_W).we_W).re1_D + ((rt_D = ws_E).we_E + (rt_D = ws_M).we_M + (rt_D = ws_W).we_W).re2_D$

Unitate de Executie cu Bypass complet



Mai este nevoie de semnalul stall?

$stall = (rs_D = ws_E).(opcode_E = LW_E).(ws_E \neq 0).re1_D + (rt_D = ws_E).(opcode_E = LW_E).(ws_E \neq 0).re2_D$

De ce o instructiune nu poate fi transmisa in fiecare ciclu (CPI > 1)?

- Implementarea Bypass-ului complet poate fi prea costisitoare:
  - sunt implementate in mod tipic numai conexiunile mai frecvente;
  - unele conexiuni inverse mai putin frecvente pot creste durata ciclului, ceea ce contravine aportului adus prin reducerea CPI;

- Incarcarile pot avea o latenta de doua cicluri:
  - instructiunea care urmeaza Incarcarii/Load nu poate folosi rezultatul acesteia
  - ISA (Arhitectura Setului de Instructiuni) MIPs-1 defineste nise de intarziere pentru incarcare, un hazard de BA vizibil software; pentru a evita hazardul, compilatorul planifica o instructiune independenta sau face insertii de NOP. In MIPS-II, nu mai este prezenta aceasta caracteristica.
- Ramificarile conditionate pot produce bule:
  - elimina urmatoarea instructiune/urmatoarele instructiuni daca nu sunt nise de intarziere.

Procesoarele cu nise de intarziere vizibile software pot executa numeroase instructiuni NOP introduse de catre compilator.