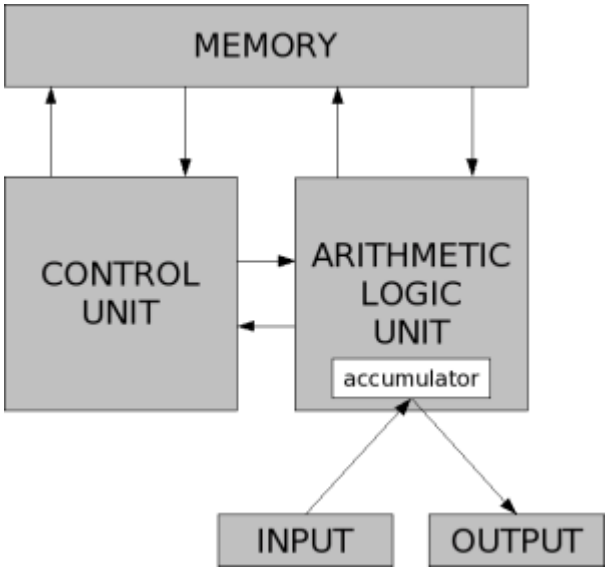
Compilatoare

Microarhitecturi
Dependență
Planificare

Arhitectura ‘Von Neumann’

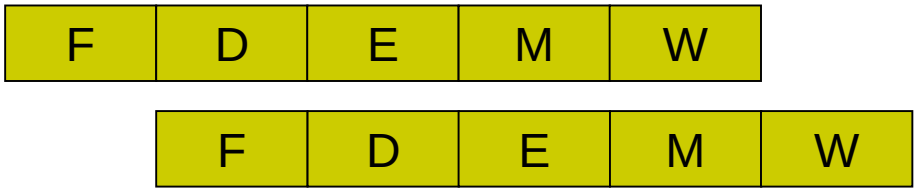


Imbunatatirea vitezei de procesare

- Suntem aproape de limita fizica
  - Ce distanta poate parcurge unda electromagnetica intr-un ciclu, la o frecventa de 3 GHz?
  - Legea lui Moore – densitate, nu viteza / consum
- Pentru a sustine cresterea performantei trebuie exploatat paralelismul
  - ILP
    - Hardware – pipelines, forwarding, dynamic scheduling&renaming, branch prediction, multiple issue, speculare
    - Software – optimizari(unrolling, scheduling etc.)
  - SIMD, instructiuni vectoriale, MIMD chiar

Pipelining

- Mai multe instruct. se executa simultan



- Bazat pe faptul ca intructiunile executa operatii diferite pe parcursul executiei
- Important ca stagiile sa fie echilibrate
  - Altfel e nevoie de buffere
- Pipelining-ul adauga overhead
  - 1/Frecv >clock skew+latch overhead

RISC vs. CISC

- RISC
  - Permite un pipeline echilibrat
  - Faciliteaza superpipelining – frecvente mari de executie
  - Cost hardware mai mic, mai simplu de implementat
- CISC
  - Permite cod mai compact
  - Nu prea poate fi “pipelined”
    - Solutia x86 – transformarea “on the fly” in hardware a instructiunilor CISC in micro-instructiuni RISC
    - Solutia Transmeta – transformarea in software, executia pe VLIW
      - **Low power**

CPL
IR crunch

Hazards

- Sunt situatii care impiedica rularea cursiva – apar ‘pipeline stalls’
- Structural hazards – conflicte de resurse
  - De ex – memorie cu un singur port, pe pipeline-ul precedent
  - Apar din motive de ‘cost reduction’
- Data hazards – dependenta de rezultatul instructiunii precedente
- Control hazards – din instructiunile de salt (trebuie golit pipeline-ul)

Data hazards

- Read After Write (mai rar WAW, WAR)
- Forwarding (“bypassing”, “short-circuiting”)
  - Citirea rezultatului unei instructiuni direct din latch-urile intermediare, fara a astepta scrierea rezultatului in registru
- Pipeline interlock
  - Cand nu se poate rezolva din Forwarding
- Lasa rezolvarea in software
  - “e ilegal sa se foloseasca rezultatul”
- NUAL (Non-Unit Assumed Latency)

Control hazards

- Procesorul nu stie de unde sa faca ‘fetch’
  - ‘pipeline freeze’ sau ‘flush’
  - La ‘flush’ tb. avut grija sa nu schimbam ‘starea procesorului’
    - Compilatorul poate imbunatati performanta rearanjand codul
  - Delayed branches – o forma de NUAL
    - Cancelling/nullifying branches
  - Branch prediction
    - Prezice taken/not taken
    - Prezice unde se sare
  - Predication

Branch prediction

- Predictori simpli – tabela de ‘istorie’ care tine minte daca e ‘taken’ sau ‘not taken’
  - 1 bit vs. 2 bit
- Predictori corelati (pe 2 nivele)
  - Tine minte daca ultimul salt executat a fost ‘taken’ sau ‘not taken’
  - Practic am 2 predictori simpli, aleg pe baza istoriei globale
- Branch target buffers
  - Memoreaza si adresa la care se sare

Branch prediction

- Sugestii din partea compilatorului
- Eliminarea branch-urilor prin predicare

```
CMP R1,R2
ADDNE R3,R4,R5
```

  - Mai multe coduri de conditii

```
CMPW CR0, R1,R2
```

  - Registri de predicate – IA64

```
cmp.ne p2,p3 = r5,r0 ;;
(p2) add r6 = 8, r5
(p3) add r6 = 12, r5
```

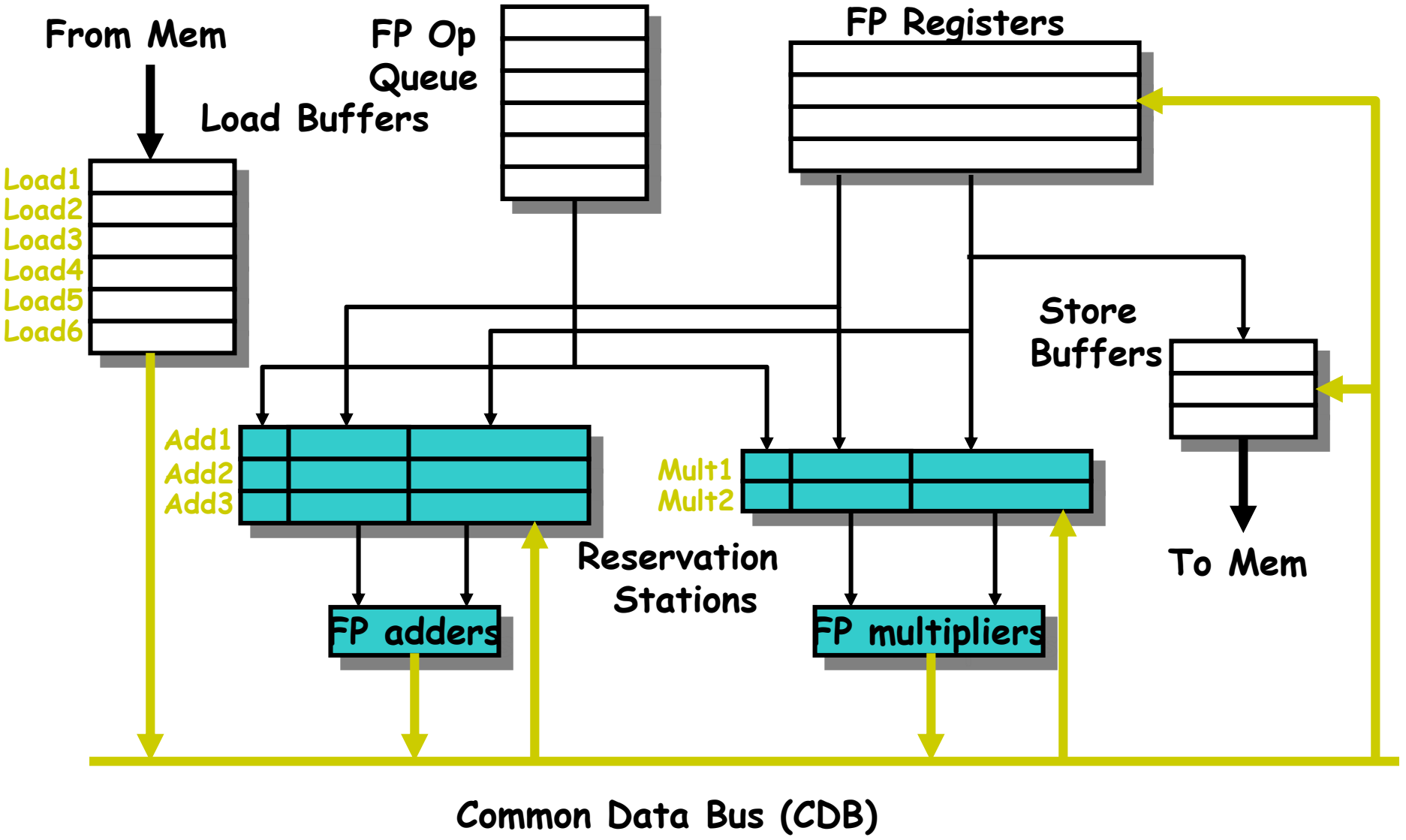
Probleme ridicate de pipelining

- Tratarea exceptiilor
  - Mai multe instruct. simultan, exceptiile pot aparea ‘out of order’
- Set complex de instructiuni, instructiuni multiciclu
  - Se face pipeline pe microinstructiuni
  - Instruct. multiciclu ‘bucleaza’ pe acelasi stagiu
  - Pt. exceptii – history file/future file

Planificarea în hardware

- Pe baza de ‘scoreboard’
  - Issue in-order, execute out-of-order, complete in-order
  - Se executa doar daca nu depinde de instruct. precedente
- Pe baza de ‘reservation station’
  - Face ‘register renaming’
  - “algoritmul lui Tomasulo”
  - Controlul e distribuit

Algoritmul lui Tomasulo



Reservation Station Components

Op: Operatia ce se executa (de ex., + sau –)

Vj, Vk: Valoarea operanzilor sursa

- Store buffers au un singur V

Qj, Qk: RS care produc registrii sursa (valoarea din operanzi)

- Nota: Nu exista ‘ready flag’; Qj,Qk=0 => ready
- Store buffers au un singur Qi

Busy: Indica RS sau FU ocupat

Register result status— Indica ce FU va scrie fiecare registru (daca e cazul). E gol cand nu exista instructiuni in executie ce urmeaza sa scrie acel registru

Stagii in Algoritmul Tomasulo

1. Issue—ia instructiuni din FP Op Queue
   Daca RS e liber (nu exista hazard structural) lanseaza instructiunea & trimite operanzii
2. Execution— executa operatia propriu-zisa (EX)
   Cand valoarea ambilor operanzi e cunoscuta;
   daca nu e cunoscuta, urmareste Common Data Bus
3. Write result—termina executia(WB)
   Scrie rezultatul pe Common Data Bus pentru toate unitatile care asteapta; marcheaza RS ca 'ready'

- Normal data bus: data + destination ("go to" bus)
- Common data bus: data + source ("come from" bus)
  - 64 bits data + 4 bits Functional Unit source address

Exemplu (Tomasulo)

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2			
LD	F2	45+	R3			
MULTD	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
0	*FU*									

Exemplu(Tomasulo) ciclul 1

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1		
LD	F2	45+	R3			
MULTD	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	*Address*
Load1	Yes	34+R2
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
1	*FU*				Load1					

Exemplu(Tomasulo) ciclul 2

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1		
LD	F2	45+	R3	2		
MULTD	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	Yes	34+R2
Load2	Yes	45+R3
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
2	*FU*		Load2		Load1					

Pot exista mai multe instructiuni load in executie

Exemplu(Tomasulo) ciclul 3

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	
LD	F2	45+	R3	2		
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	Yes	34+R2
Load2	Yes	45+R3
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	MULTD		R(F4)	Load2	
	Mult2	No					

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
3	FU	Mult1	Load2		Load1					

- Nota: registrii nu mai sunt folositi ulterior (“register renaming”)
- Load1 se termina; cine asteapta dupa Load1?

Exemplu(Tomasulo) ciclul 4

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4		
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	No	
Load2	Yes	45+R3
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	Yes	SUBD	M(A1)			Load2
	Add2	No					
	Add3	No					
	Mult1	Yes	MULTD		R(F4)	Load2	
	Mult2	No					

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
4	*FU*	Mult1	Load2		M(A1)	Add1				

- Load2 se termina; cine asteapta dupa Load2?

Exemplu(Tomasulo) ciclul 5

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4		
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2			

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
2	Add1	Yes	SUBD	M(A1)	M(A2)		
	Add2	No					
	Add3	No					
10	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
5	*FU*	Mult1	M(A2)		M(A1)	Add1	Mult2			

Exemplu(Tomasulo) ciclul 6

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4		
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
1	Add1	Yes	SUBD	M(A1)	M(A2)		
	Add2	Yes	ADDD		M(A2)	Add1	
	Add3	No					
9	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
6	*FU*	Mult1	M(A2)		Add2	Add1	Mult2			

- La scoreboard, ADDD nu ar fi putut fi executata

Exemplu(Tomasulo) ciclul 7

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
0	Add1	Yes	SUBD	M(A1)	M(A2)		
	Add2	Yes	ADDD		M(A2)	Add1	
	Add3	No					
8	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
7	*FU*	Mult1	M(A2)		Add2	Add1	Mult2			

- Add1 se termina; cine asteapta dupa el?

Exemplu(Tomasulo) ciclul 8

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
2	Add2	Yes	ADDD	(M-M)	M(A2)		
	Add3	No					
7	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
8	*FU*	Mult1	M(A2)		Add2	(M-M)	Mult2			

Exemplu(Tomasulo) ciclul 9

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
1	Add2	Yes	ADDD	(M-M)	M(A2)		
	Add3	No					
6	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
9	*FU*	Mult1	M(A2)		Add2	(M-M)	Mult2			

Exemplu(Tomasulo) ciclul 10

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
0	Add2	Yes	ADDD	(M-M)	M(A2)		
	Add3	No					
5	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
10	*FU*	Mult1	M(A2)		Add2	(M-M)	Mult2			

- Add2 se termina; cine asteapta rezultatul?

Exemplu(Tomasulo) ciclul 11

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
4	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
11	*FU*	Mult1	M(A2)		(M-M+M	(M-M)	Mult2			

- ADDD poate scrie rezultatul acum(la scoreboard nu)
- Toate instructiunile ‘rapide’ se termina acum

Exemplu(Tomasulo) ciclul 12

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
3	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
12	*FU*	Mult1	M(A2)		(M-M+M	(M-M)	Mult2			

Exemplu(Tomasulo) ciclul 13

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
2	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
13	*FU*	Mult1	M(A2)		(M-M+M	(M-M)	Mult2			

Exemplu(Tomasulo) ciclul 14

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
1	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
14	*FU*	Mult1	M(A2)		(M-M+M	(M-M)	Mult2			

Exemplu(Tomasulo) ciclul 15

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
0	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
15	*FU*	Mult1	M(A2)		(M-M+M	(M-M)	Mult2			

Exemplu(Tomasulo) ciclul 16

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
40	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
16	*FU*	M*F4	M(A2)		(M-M+M	(M-M)	Mult2			

Exemplu(Tomasulo)

- s.a.m.d – se termina si DIV-ul
- Tomasulo:
  - Issue in-order
  - Executie, completare out-of-order
- Slabiciunea principala – CDB
  - Multe scrieri asociative, la viteza mare

Multiple issue

- Implicit (Dinamic, Hw)
- Explicit(static, Sw) VLIW/VLES/EPIC
  - TI/StarCore/Itanium (IA64)
  - Instructiuni EPIC - semantica + restrictii
- SuperScalar
  - Poate fi out-of-order sau nu!
  - N-way superscalar – cate poate lansa in executie simultan
  - Ciclul de ‘issue’ e pe critical path. Cum poate fi impartit?

Executia speculativa

- Executia speculativa.
  - Retragerea instructiunilor in ordine
  - Probleme: exceptii
    - Ex: acces invalid la memorie
    - Unele exceptii trebuie sa fie precise (ex. floating point)
  - Implementare: Reorder Buffer
    - Issue, Execute, Write, Commit
    - Ce se intampla cand un branch e prezis incorect?

CPL
IR crunch

Planificarea in software

- Scop: reordoneaza instructiunile din program, cu scopul de a minimiza numarul de stall-uri
- Constrangeri:
  - dependente de date
    - De flux, de iesire, antidependente
    - Pe reg – simplu; pe memorie – analiza de alias, analiza de dependenta
  - dependente de control
  - constrangeri de resurse

Tipuri de dependente

- De date/de control; ne vom concentra pe cele de date (dar e nevoie de ambele pentru a determina corectitudinea)
- Exemplu simplu de dependenta de date:

```
S1   PI = 3.14
S2   R = 5.0
S3   C = 2 * PI * R * R
```

  - S_3 nu poate fi mutata inaintea S_1 sau S_2
- Ex. de dependenta de control

```
S1  IF(X>0)
S2     A[i]=0
```

  - S2 nu poate fi mutata inaintea S1

Dependente

- Formal:

  Exista dependenta de date de la S1 la S2 (S_2 depinde de S_1) daca:

  1. Ambele acceseaza aceeasi locatie de memorie
  2. Cel putin una dintre ele scrie locatia de memorie
  3. Exista o cale de executie fezabila de la S_1 la S_2

  Nu se poate determina in cazul cel mai general daca exista o cale de executie la rulare care le include pe amandoua

- Cale fezabila = cale in CFG
- Pt. locatii volatile, se renunta la conditia 2(avem dependenta si daca ambele sunt citiri)
  - Nu ne intereseaza in practica sa optimizam accesele la variabile volatile

Clasificarea dependentelor

- Clasificarea in functie de ordinea de citire-scriere:
  1. Dependente adevarate (de flux, RAW) X =
     - S_2 depide de S_1, notat si $S_1\ \delta\ S_2$ = X
  2. Antidependente (WAR) = X
     - S_2 depinde de S_1,cu notatia $S_1\ \delta^{-1}\ S_2$ X =
  3. Dependenta de iesire (WAW) X =
     - S_2 depinde de S_1, notat $S_1\ \delta^{0}\ S_2$ X =
  4. Dependenta de intrare (RAR) = X
     - S_2 depinde de S_1, notat $S_1\ \delta^{i}\ S_2$ = X
- 4 nu ne intereseaza in practica; 2,3 sunt legate de constrangeri de reprezentare si nu de fluxul valorilor (pot fi evitate de multe ori)

Durata si latenta

- Durata D(x) = nr. de cicli dupa care se poate folosi din nou unitatea dupa executia instructiunii x
  - 1 pt unitati pipelined, >1 non-pipelined
- Latenta L(x,y) = nr. de cicli dupa care instructiunea y poate folosi rezultatele din x
  - L(x) = nr. de cicli dupa care rezultatul lui x este scris

Dependenta	VLIW	Superscalar
RAW	L(x,y)	L(x,y)
WAR	1-L(y)	0
WAW	1+L(x)-L(y)	L(x)

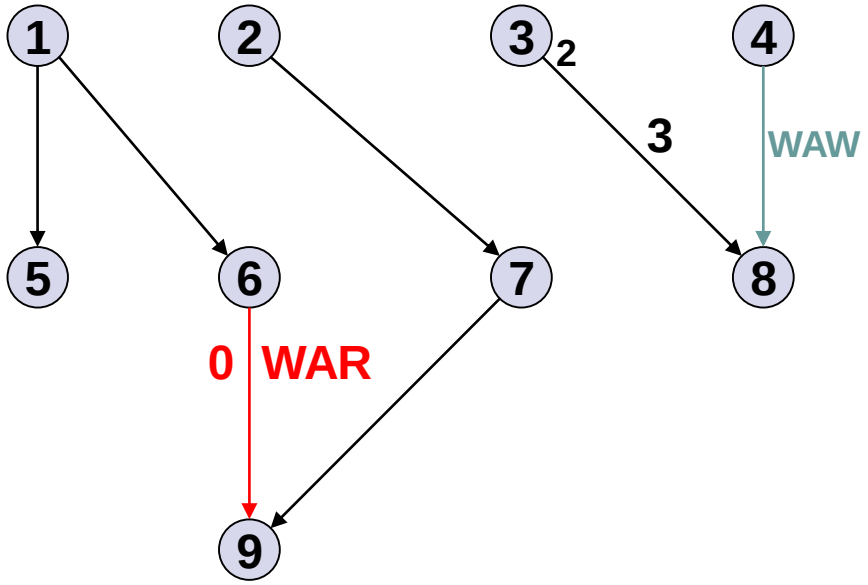
Reprezentarea dependentelor

CPL
IR crunch

- Graful de dependenta
  - Noduri = instructiuni (etichetate cu nr. de cicli necesari instructiunii)
  - Arce = dependente (etichetate cu latenta)

Exemplu

1. MOV R1,10
2. ADD R2,R2,5
3. DIV R5,R5,45
4. LOAD R6,[A]
5. MOV R3,R1
6. ADD R8,R4,R1
7. MUL R7,R2,2
8. ADD R6,R5,1
9. LOAD R4,[R7]



DIV – are latenta de 3 cicli, nu este pipelined, unitatea este ocupata pt 2 cicli

List scheduling

- Sorteaza topologic graful de dependenta
- Planifica o instructiune daca nu cauzeaza nici un stall si toti predecesorii au fost deja planificati
- Optimal list scheduling e NP-completa; se folosesc euristici la selectie
- Algoritm:
  1. Creaza graful de dependenta (DAG) al bb-ului.
  2. Sorteaza graful topologic
  3. READY = noduri fara predecesori
  4. Executa 5-6 pana cand lista READY e goala
  5. Scoate din lista si planifica nodurile din READY care nu cauzeaza stall-uri;
  6. READY += nodurile ale carori predecesori au fost deja planificati
- Ordinea in care sunt planificate nodurile din READY conteaza!

Euristici pentru alegerea din ready list

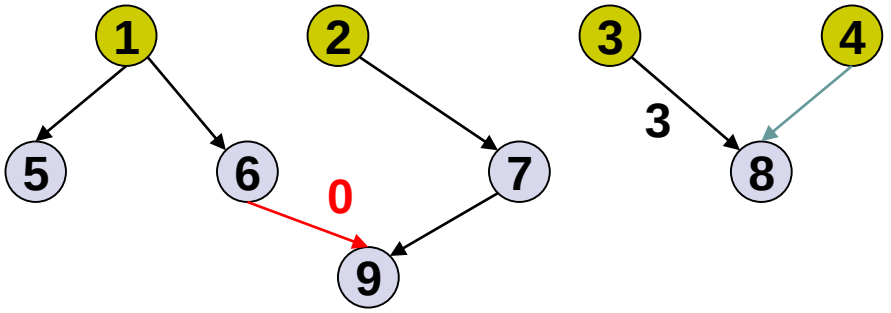
CPL IR crunch

- Alege nodul cu cea mai lunga cale in graful de dependenta (nodul de pe ‘critical path’)
- Algoritm (pentru nodul x)
  - Daca x nu are succesori $d_x = 0$
  - $d_x = MAX(d_y + c_{xy})$ pentru toti succesorii y ai lui x
  - (se foloseste traversarea in ordine inversa BFS)

- Alege nodul cu cu cei mai multi succesori imediati
- Alege un nod care foloseste mai putine resurse
- Etc.

Exemplu

1. MOV R1,10
2. **ADD R2,R2,5**
3. **DIV R5,R5,45**
4. **LOAD R6,[A]**
5. MOV R3,R1
6. ADD R8,R4,R1
7. MUL R7,R2,2
8. ADD R6,R5,1
9. LOAD R4,[R7]

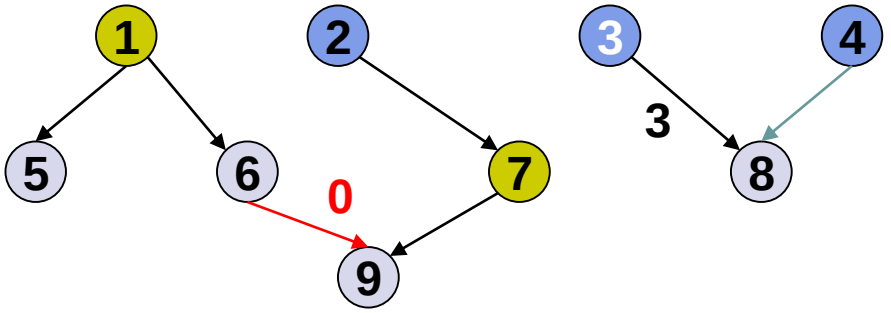


Aritmetic	Aritmetic	Load/Store
ADD R2,R2,5	DIV R5,R5,45	LOAD R6,[A]

2, 3 sunt pe calea critica. 4 e singura instructiune de load/store disponibila

Exemplu

1. **MOV R1,10**
2. **ADD R2,R2,5**
3. **DIV R5,R5,45**
4. **LOAD R6,[A]**
5. MOV R3,R1
6. ADD R8,R4,R1
7. MUL R7,R2,2
8. ADD R6,R5,1
9. LOAD R4,[R7]

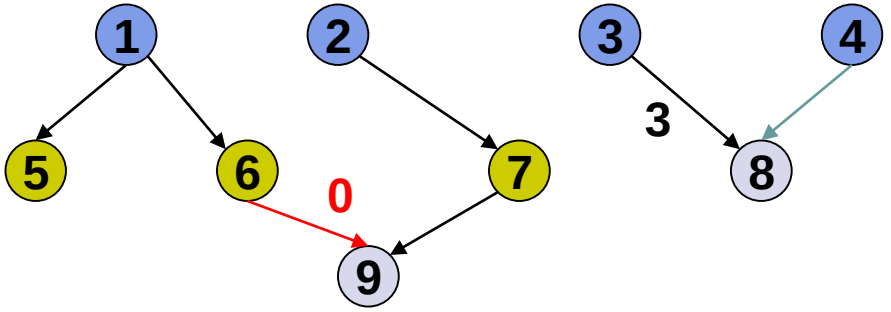


Aritmetic	Aritmetic	Load/Store
ADD R2,R2,5	DIV R5,R5,45	LOAD R6,[A]
MOV R1,10		

1 are mai multi succesori ca 7 . Nu exista instructiuni load/store in ready

Exemplu

1. **MOV R1,10**
2. **ADD R2,R2,5**
3. **DIV R5,R5,45**
4. **LOAD R6,[A]**
5. MOV R3,R1
6. **ADD R8,R4,R1**
7. **MUL R7,R2,2**
8. ADD R6,R5,1
9. LOAD R4,[R7]

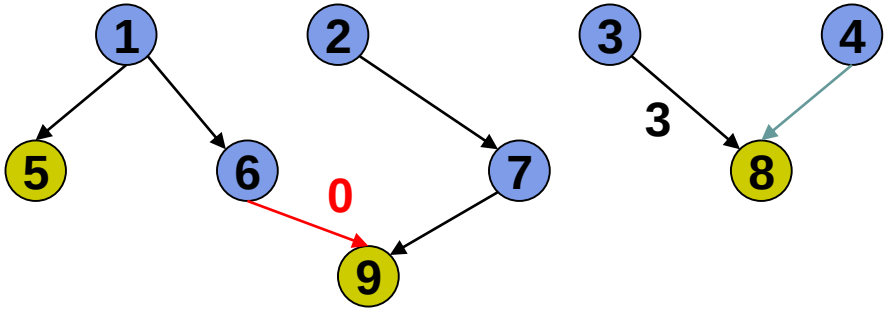


Aritmetic	Aritmetic	Load/Store
ADD R2,R2,5	DIV R5,R5,45	LOAD R6,[A]
MOV R1,10		
ADD R8,R4,R1	MUL R7,R2,2	

8 nu este in READY, pt ca rezultatul lui 3 nu e inca disponibil

Exemplu

1. **MOV R1,10**
2. **ADD R2,R2,5**
3. **DIV R5,R5,45**
4. **LOAD R6,[A]**
5. **MOV R3,R1**
6. **ADD R8,R4,R1**
7. **MUL R7,R2,2**
8. **ADD R6,R5,1**
9. **LOAD R4,[R7]**

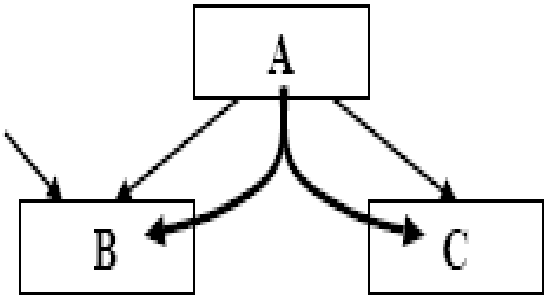


Aritmetic	Aritmetic	Load/Store
ADD R2,R2,5	DIV R5,R5,45	LOAD R6,[A]
MOV R1,10		
ADD R8,R4,R1	MUL R7,R2,2	
MOV R3,R1	ADD R6,R5,1	LOAD R4,[R7]

Planificare peste basic block-uri

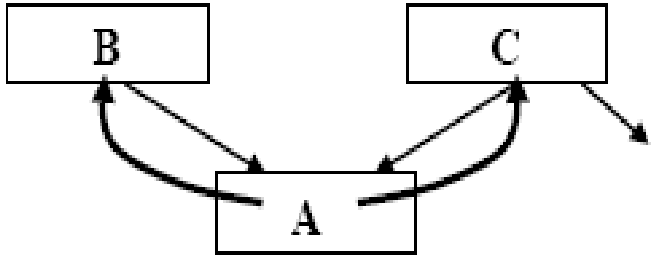
- Trace scheduling, extended BB cu duplicare de cod

Downward to adjacent basic block



A path to B that does not execute A?

Upward to adjacent basic block

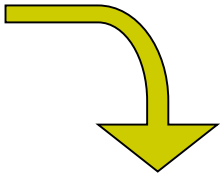


A path from C that does not reach A?

Planificarea in bucle

- Unrolling -> renaming

```
for(i=0; i<300; i++ ) {
  x = a[i];
  y = calc(x);
  b[i] = y;
}
```



```
for(i=0; i<300; i+=3 ) {
  x0 = a[i];
  x1 = a[i+1]; y0 = calc(x0);
  x2 = a[i+2]; y1 = calc(x1); b[i] = y0;
               y2 = calc(x2); b[i+1] = y1;
                              b[i+2] = y2;
}
```

Software pipelining

```
for(i=0; i<300; i++ ) {
  x = a[i];
  y = calc(x);
  b[i] = y;
}
```

```
for(i=0; i<300; i+=3 ) {
  x0 = a[i];
  y0 = calc(x0); x1 = a[i+1];
  b[i] = y0;     y1 = calc(x1); x2 = a[i+2];
                 b[i+1] = y1;   y2 = calc(x2);
                                b[i+2] = y2;
}
```

```
                                x = a[0];
                 y = calc(x);   x = a[1];
for(i=0; i<298; i++ ) {
  b[i]   = y;    y = calc(x);   x = a[i+2];
}
  b[299] = y;    y = calc(x);
  b[300] = y;
```

Implementare: Modulo scheduling (ex. “Iterative Modulo Scheduling”, Rau, 1995)

Backup slides

Memorii

- Principiul localitatii
- Registri – Cache – Memorie – I/O
- Tipul de cache
  - Mapat direct, complet asociativ, set-asociativ pe n cai
  - Politici de inlocuire – random, LRU, FIFO
  - Write through, write back

Ierarhia de memorie

- Ne intereseaza cache pt. a optimiza accesul la variabile
  - Instructiuni de control explicit al cache-ului
- Reg – de multe ori sunt mai multi reg. hw. decat “registri vizibili arhitectural”.
- I/O – memorie virtuala