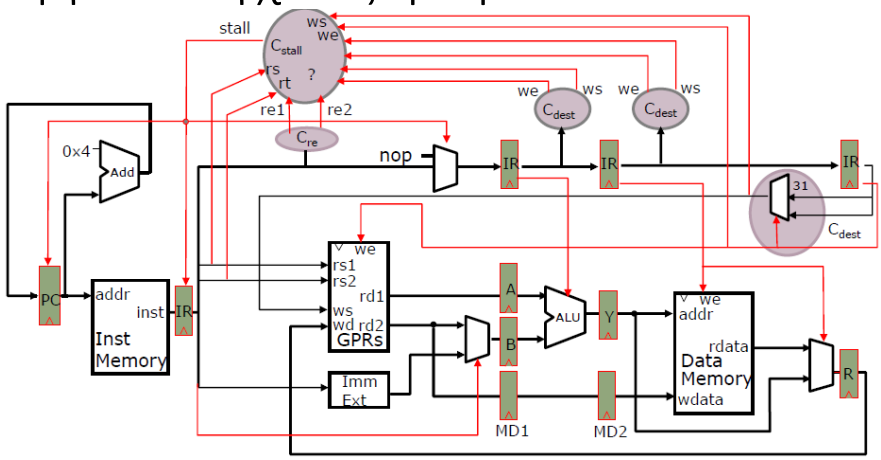
Εκτέλεση Εντολών με Pipelining

Επίλυση Εξάρτησης Δεδομένων

- Τρόπος #1: Αναμονή μέχρι το αποτέλεσμα να είναι διαθέσιμο, με πάγωμα των σταδίων pipeline(stall)
- Τρόπος #2: Προώθηση του αποτελέσματος μόλις υπολογισθεί στο κατάλληλο στάδιο του pipeline που το χρειάζεται (bypass)
- Τρόπος #3: Μάντεψε την εξάρτηση
  - Δύο περιπτώσεις:
  - Σωστή υπόθεση -> μην κάνεις τίποτα
  - Λάθος υπόθεση -> σταμάτα και ξανάρχισε την εκτέλεση

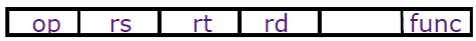
Λογική για Έλεγχο Εξαρτήσεων



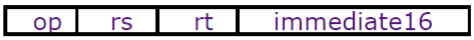
- Πρέπει πάντα να παγώνει το pipeline όταν rs είναι ίσο με κάποιο rd ?
  - Δεν γράφουν όλες οι εντολές το rd, ούτε όλες διαβάζουν $re_{1,2}$

Καταχωρητές Πηγής και Προορισμού

R-type:



I-type:



J-type:

		source(s)	*destination*
ALU	rd ← (rs) func (rt)	rs, rt	rd
ALUi	rt ← (rs) op imm	rs	rt
LW	rt ← M [(rs) + imm]	rs	rt
SW	M [(rs) + imm] ← (rt)	rs, rt	
BZ	*cond* (rs)		
	true: PC ← (PC) + imm	rs	
	false: PC ← (PC) + 4	rs	
J	PC ← (PC) + imm		
JAL	r31 ← (PC), PC ← (PC) + imm		31
JR	PC ← (rs)	rs	
JALR	r31 ← (PC), PC ← (rs)	rs	31

Κίνδυνοι Λόγω Load και Store

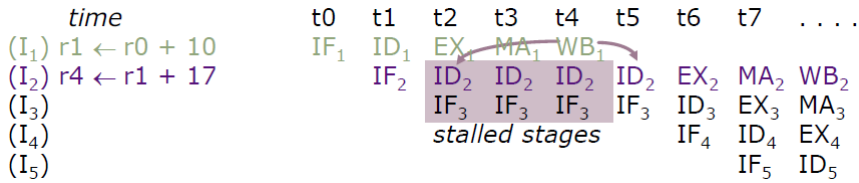
- ...
- M[(r1)+7] ← (r2)
- r4 ← M[(r3)+5]
- ...

(r1)+7 = (r3)+5 : *data hazard*

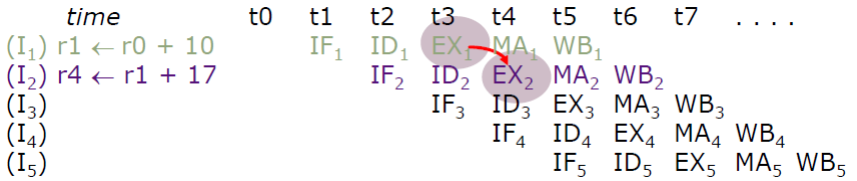
- Τέτοιος κίνδυνος όμως αποφεύγεται επειδή η μνήμη ολοκληρώνει μία εγγραφή σε ένα κύκλο
- Κίνδυνος εξάρτησης καταχωρητών επιλύεται μέσω της λογικής ελέγχου για πάγωμα ή προσπέραση

5

Εξαρτήσεις: Επίλυση με Προώθηση Τιμών



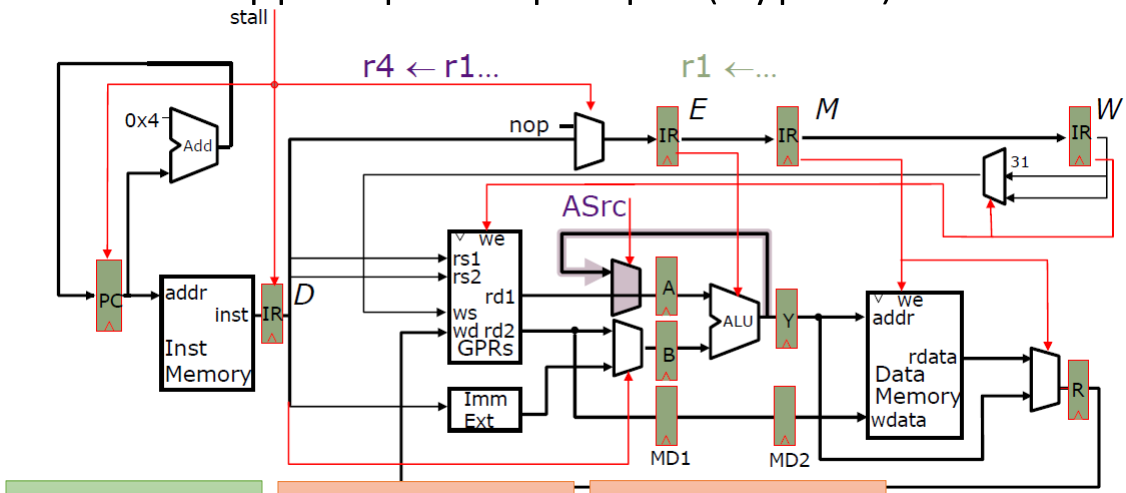
- Κάθε καθυστέρηση/σταμάτημα ή ακύρωση συνεπάγεται μία φυσαλίδα => CPI > 1
- Τα αποτελέσματα/δεδομένα είναι ουσιστικά διαθέσιμα στο στάδιο EX



- Πρακτικά χρειάζεται ένα νέο μονοπάτι (μία προσπέραση) τα δεδομένα να είνι διαθέσιμα από την έξοδο της ALU στην είσοδο

6

Επέκταση με Προσπέρασμα (bypass)



(I1) r1 ← r0 + 10
(I2) r4 ← r1 + 13

ο

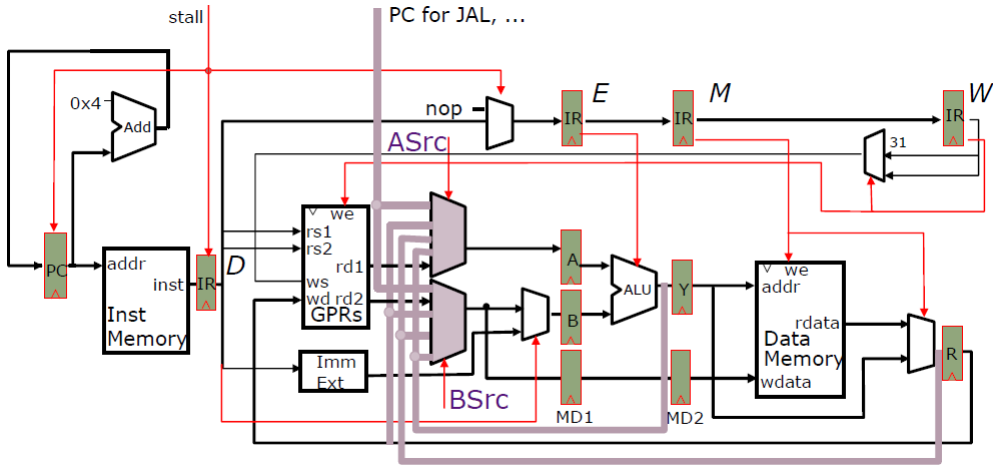
(I1) r1 ← M[r0 + 10]
(I2) r4 ← r1 + 13

ος

(I1) jal 300
(I2) r4 ← r31 + 13

7

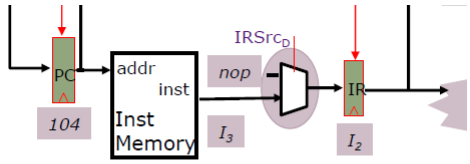
Datapath με Προσπεράσματα



8

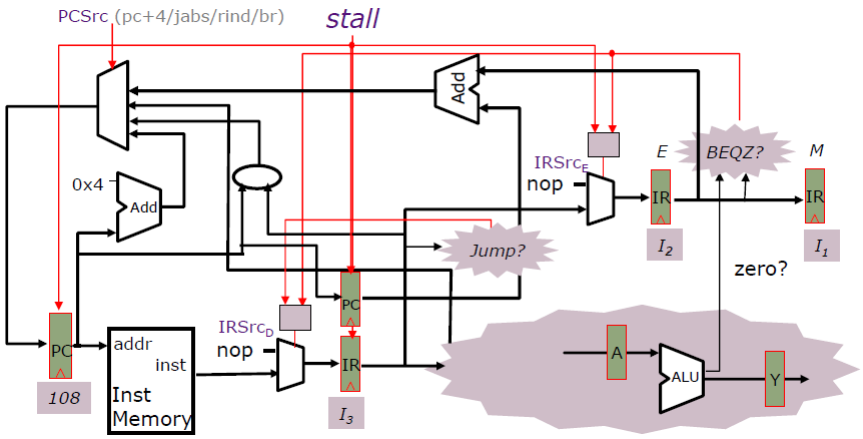
Pipeline για Εντολές Jump

	time 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	



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

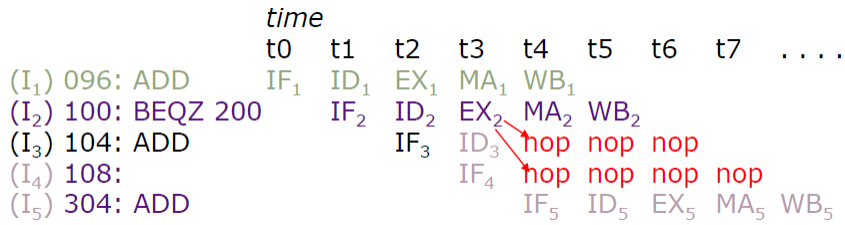
Διακλαδώσεις υπό Συνθήκη



- I1 096 ADD
- I2 100 BEQZ r1 200
- I3 104 ADD
- I4 304 ADD

Αν η διακλάδωση είναι επιτυχής, τότε οι δύο επόμενες εντολές πρέπει να ακυρωθούν

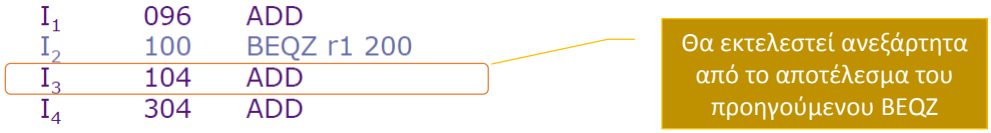
Διακλάδωση με Pipelined Μορφή



- Περιπτώσεις ανάλογα με το Opcode στον IR:
  - PC+4 : υπόθεση
  - Beq/bne, J, Jal, Jr : επανεκκίνηση του pipeline, nop

Καθυστέρηση σε Εντολή Διακλάδωσης

- Δώσε την ευκαιρία ώστε να εκτελείται πάντα η επόμενη εντολή μετά ένα jump ή ένα beq (branch delay slot).
  - Πρέπει όμως ο compiler να βάζει μία χρήσιμη εντολή στο slot αυτό, εκεί που κανονικά θα έπρεπε να μπει μία φυσαλίδα pipeline για καθυστέρηση.



- Άλλες τεχνικές περιλαμβάνουν πρόβλεψη διακλάδωσης, που μπορεί να βελτιώσει σημαντικά τις καθυστερήσεις!

Γιατί μία Εντολή Είναι Δύσκολο να Εκτελείται σε Pipeline σε Κάθε Κύκλο ? (CPI = 1)

- Πλήρη προσπεράσματα σε όλες τις περιπτώσεις είναι ακριβό!
  - Συνήθως υλοποιούνται οι πιο συχνές περιπτώσεις
  - Οι σπάνιες περιπτώσεις μπορεί να απαιτούν μεγάλη λογική που αυξάνει τον κύκλο ρολογιού
- Αναγνώσεις από την μνήμη έχουν απαραίτητα δύο κύκλους
  - Μία εντολή μετά από ανάγνωση δεν μπορεί να χρησιμοποιεί την τιμή που διαβάστηκε
- Οι διακλαδώσεις υπό συνθήκη προκαλούν φυσαλίδες καθυστέρησης
  - Είτε ακυρώνεται η επόμενη εντολή είτε εφαρμόζεται ένα delay slot

13

Pipelining σε Μοντέρνα Συστήματα Υψηλών Επιδόσεων

- Συστήματα υψηλών επιδόσεων επιχειρούν να εκμεταλευτούν:
  - Τεχνολογίες, καλύτερη σχεδίαση και ολοκλήρωση λογικής και μνημών
  - Παραλληλία, στην αποκωδικοποίηση, εκτέλεση πράξεων/λειτουργιών
  - Κρύψιμο καθυστερήσεων, με αναδιάταξη εντολών, υπο-λειτουργιών

14

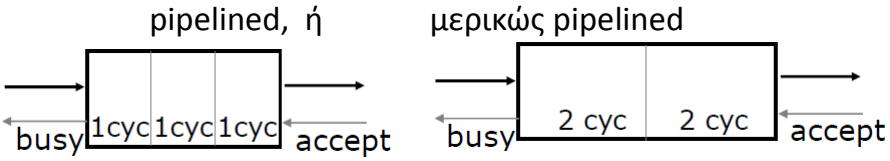
Πολυπλοκότητα Pipelining και Αίτια

- Τεχνικές σχεδιασμού για υψηλές επιδόσεις περιλαμβάνουν:
  - Λειτουργίες που διαρκούν πολλούς κύκλους, όπως:
    - Μονάδες υπολογισμών για αριθμούς κινητής υποδιαστολής, με πλήρες ή μερικό pipelining
    - Λειτουργίες που διαρκούν πολύ χρόνο (π.χ. Διαίρεση)
  - Λειτουργίες που έχουν μεταβλητή καθυστέρηση (την ώρα εκτέλεσης)
    - Συστήματα μνήμης με μεταβλητό χρόνο απόκρισης
  - Μονάδες με πολλά αντίγραφα
    - Πολλαπλές μονάδες για πράξεις με αριθμούς κινητής υποδιιαστολής ή ελεγκτές μνήμης

15

Μονάδα Κινητής Υποδιαστολής (FPU)

- Πολύ μεγαλύτερη, πολύπλοκο hardware, σε σχέση με μία μονάδα ακεραίων
- Είναι σύνηθες να υπάρχουν μερικές μονάδες κινητής υποδιαστολής, ίσως διαφορετικού τύπου: FADD, FMUL, FDIV, …
- Μία μονάδα κινητής υποδιαστολής μπορεί να είναι:



- Για να λειτουργούν αρκετές FPU ταυτόχρονα πρέπει το αρχείο καταχωρητών να έχει πολλές πόρτες για ανάγνωση και αποθήκευση

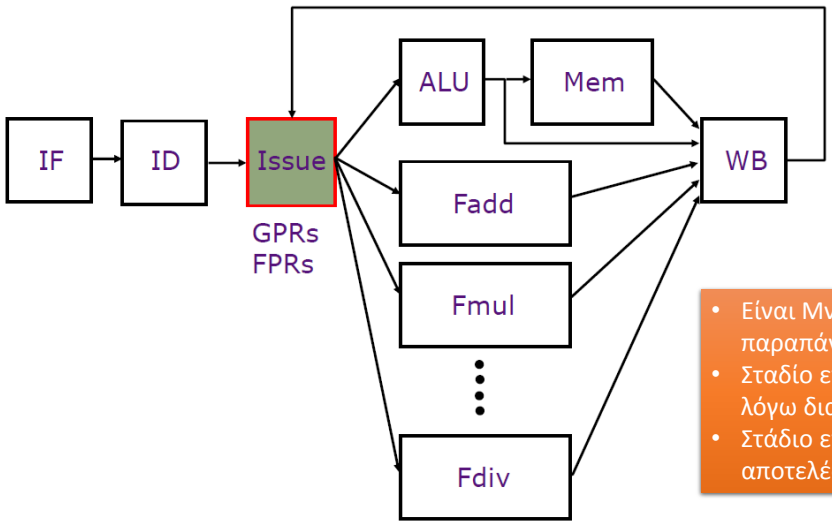
16

Πραγματικά Συστήματα Μνήμης

- Η καθυστέρηση πρόσβασης στην κύρια μνήμη είναι συνήθως μεγαλύτερη από ένα κύκλο ρολογιού και συχνά απρόβλεπτη...
  - Η λύση αυτού του προβλήματος είναι ένα βασικό πρόβλημα για τους αρχιτέκτονες
- Συνήθεις τεχνικές για την βελτίωση των επιδόσεων της μνήμης
  - Ξεχωριστές πόρτες για εντολές και για δεδομένα
    - ⇒ κώδικας που τροποποιεί τον εαυτό του δεν είναι εφικτό
  - Κρυφές μνήμες (caches)
    - Επιδίωξη είναι η πρόσβαση σε 1 κύκλο εκτός και υπάρξει miss
  - Μνήμη με διαφύλλωση (interleaved)
    - Πολλαπλές προσβάσεις στην μνήμη ⇒ μπορεί να προκαλούν συγκρούσεις στις memory banks
  - Λειτουργίες στην μνήμη σε φάσεις
    - ⇒ απόκριση της μνήμης σε διαφορετική σειρά (out-of-order)

17

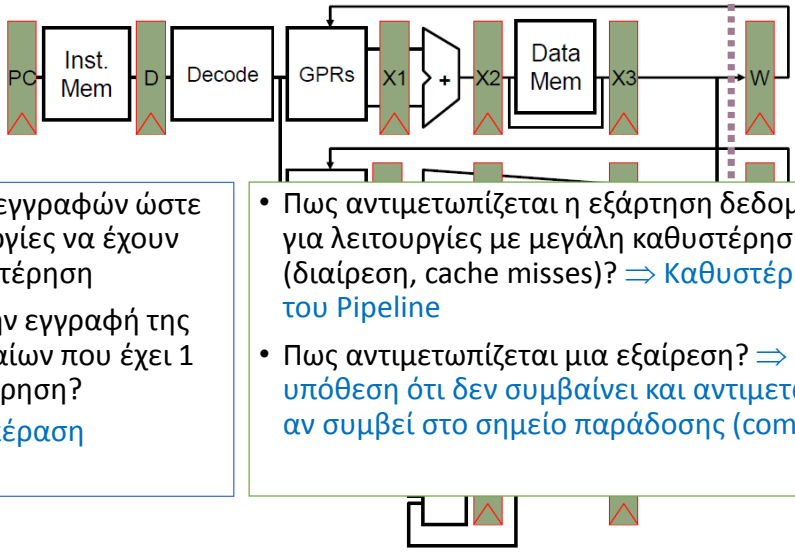
Δομές Επεξεργασίας με Πολύπλοκο Pipeline



- Είναι Μνήμη και FPU pipelined? Ή απαιτούν παραπάνω από 1 κύκλο?
- Σταδίο εγγραφής αποτελέσματος πολύπλοκο λόγω διαφορετικών καθυστερήσεων των FPU.
- Στάδιο εγγραφής με out-of-order αποτελέσματα

18

Πολύπλοκο Pipeline In-Order



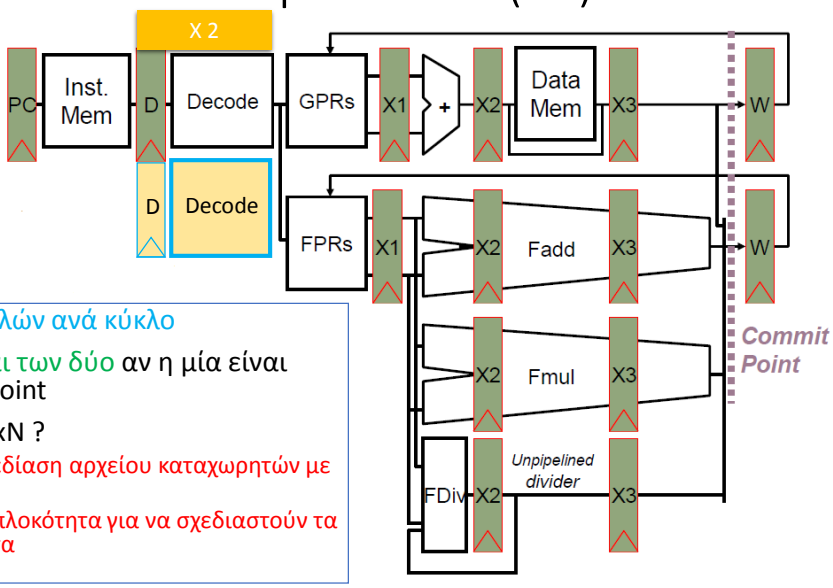
- Καθυστέρηση εγγραφών ώστε όλες οι λειτουργίες να έχουν την ίδια καθυστέρηση
- Τι γίνεται με την εγγραφή της μονάδας ακεραίων που έχει 1 κύκλο καθυστέρηση?

  ⇒ προσπέραση

- Πως αντιμετωπίζεται η εξάρτηση δεδομένων για λειτουργίες με μεγάλη καθυστέρηση (διαίρεση, cache misses)? ⇒ Καθυστέρηση του Pipeline
- Πως αντιμετωπίζεται μια εξαίρεση? ⇒ υπόθεση ότι δεν συμβαίνει και αντιμετώπιση αν συμβεί στο σημείο παράδοσης (commit)

19

Pipeline In-Order: Superscalar (x2)



- Διάβασμα 2 εντολών ανά κύκλο
- Ενεργοποίηση και των δύο αν η μία είναι εντολή floating-point
- Superscalar x4 ? xN ?
  - Πολύπλοκη σχεδίαση αρχείου καταχωρητών με πολλές πόρτες
  - Εκθετική πολυπλοκότητα για να σχεδιαστούν τα προσπεράσματα

20

Εξαρτήσεις εν όψη Παραλληλισμού Εντολών

- Έστω η εκτέλεση ακολουθίας εντολών της μορφής: $r_k \leftarrow (r_i)\ op\ (r_j)$

Εξάρτηση Δεδομένων
$r_3 \leftarrow (r_1)\ op\ (r_2)$
$r_5 \leftarrow (r_3)\ op\ (r_4)$

Αντι-Εξάρτηση Δεδομένων
$r_3 \leftarrow (r_1)\ op\ (r_2)$
$r_1 \leftarrow (r_5)\ op\ (r_4)$

Εξάρτηση Δεδομένων στην Έξοδο
$r_3 \leftarrow (r_1)\ op\ (r_2)$
$r_3 \leftarrow (r_7)\ op\ (r_4)$

Κίνδυνοι αν εκτελεστεί πριν η εντολή j από την εντολή i, την στιγμή που η εντολή j βρίσκεται μετά από την i στο πρόγραμμα, με $Rj \cap Ri \neq \varnothing$

Εξαρτήσεις σε Καταχωρητές και Μνήμη

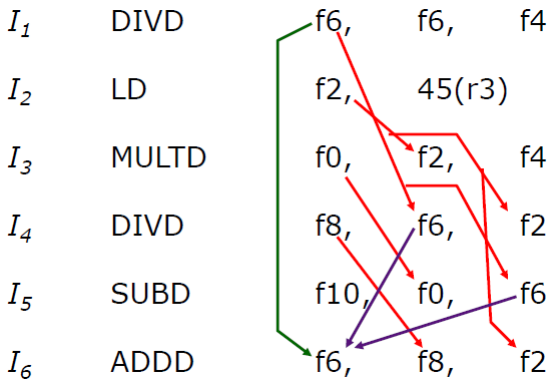
- Εξαρτήσεις δεδομένων λόγω πράξεων με καταχωρητές μπορούν να ανιχνευθούν στο στάδιο αποκωδικοποίησης των εντολών
- Εξαρτήσεις δεδομένων λόγω λειτουργιών στη μνήμη μπορούν να ανιχνευθούν μόνο μετά που θα υπολογιστεί η τελική διεύθυνση μνήμης

Αποθήκευση: M[(r1) + immediate1] ← (r2)
Διάβασμα: r3 ← M[(r4) + immediate2]

Έλεγχος αν (r1) + immediate1 == (r4) + immediate2, πότε?

22

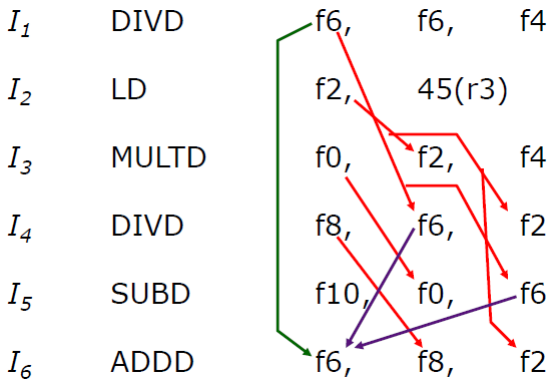
Εξαρτήσεις Δεδομένων - Παράδειγμα



- Κίνδυνοι εξαρτήσεων:
- Read-after-Write: RAW
- Write-after-Read: WAR
- Write-after-Write: WAW

23

Χρονοδρομολόγηση Εντολών



- Έγκυρη σειρά εκτέλεσης:
  - In-order: I1 I2 I3 I4 I5 I6
  - Out-of-order: I2 I1 I3 I4 I5 I6
  - Out-of-order: I1 I2 I3 I5 I4 I6

24

Δρομολογηση In-Order, Ολοκλήρωση Out-of-Order

					Latency
I_1	DIVD	f6,	f6,	f4	*4*
I_2	LD	f2,	45(r3)		*1*
I_3	MULTD	f0,	f2,	f4	*3*
I_4	DIVD	f8,	f6,	f2	*4*
I_5	SUBD	f10,	f0,	f6	*1*
I_6	ADDD	f6,	f8,	f2	*1*

Κύκλος Ρολογιού	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Ολοκλήρωση In-Order	1	2			1	2	3	4		3	5	4	6	5	6
Ολοκλήρωση Out-of-Order	1	2	2	3	1	4	3	5	5	4	6	6			

Πιθανά προβλήματα: εξαρτήσεις δεδομένων !!

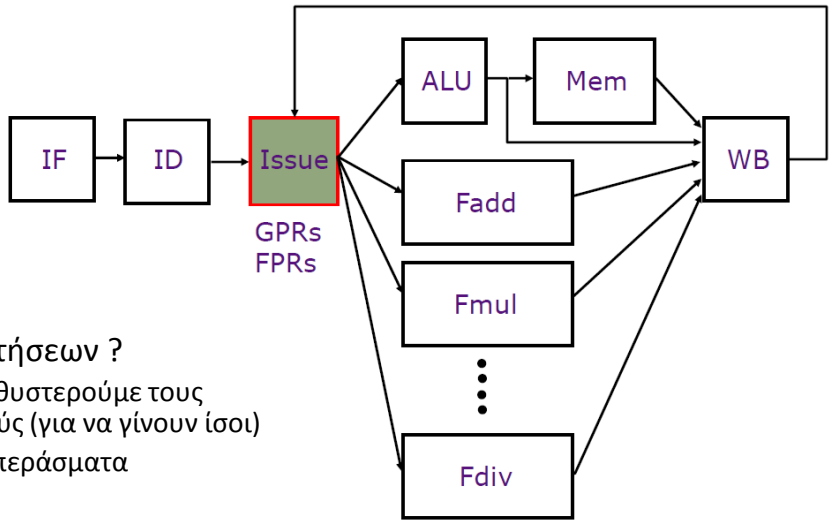
25

ScoreBoard

Δομή δεδομένων σε υλικό για να ανιχνεύονται οι εξαρτήσεις δυναμικά !!!

26

Πολύπλοκο Pipeline



- Επίλυση εξαρτήσεων ?
  - Χωρίς να καθυστερούμε τους υπολογισμούς (για να γίνουν ίσοι)
  - Χωρίς προσπεράσματα

27

Πότε Είναι Ασφαλές να Ξεκινήσει η Εκτέλεση?

- Γενική Αρχή: καθυστέρηση της εκτέλεσης εντολών με στόχο να υπάρχει εγγύηση ότι δεν θα προκύψει εξάρτηση
- Έστω μία δομή δεδομένων που κρατά τις εντολές που βρίσκονται υπό εκτέλεση στις μονάδες υπολογισμού

- Οι έλεγχοι που πρέπει να γίνουν πριν ξεκινήσει η εκτέλεση
  - Ελεύθερη η μονάδα υπολογισμού ή κατειλημμένη?
  - Δεδομένα εισόδου διαθέσιμα ? (κίνδυνος RAW)
  - Είναι ασφαλές να αποθηκευτεί το αποτέλεσμα ? (WAR, WAW)
  - Υπάρχει σύγκρουση στην αποθήκευση στο στάδιο WB ?

28

Δομή Δεδομένων για Ορθή Δρομολόγηση

Εντολή	Απασχολημένο	Op	Dest	Src1	Src2
Int					
Mem					
Add1					
Add2					
Add3					
Mult1					
Mult2					
Div					

- Η εντολή i μόλις διαβαστεί συμβουλεύεται τον πίνακα αυτόν:
  - Είναι διαθέσιμη η μονάδα FU? Έλεγχος της στήλης «Απασχολημένο»
  - RAW? Αναζήτηση στην στήλη "Dest" αν υπάρχουν οι καταχωρητές πηγής της εντολής i
  - WAR? Αναζήτηση στην στήλη "SrcX" αν υπάρχει ο καταχωρητής προορισμού της εντολής i
  - WAW? Αναζήτηση στην στήλη "Dest" αν υπάρχει ο καταχωρητής προορισμού της εντολής I
- Αν βρεθεί εξάρτηση νέα εισαγωγή στον πίνακα
- Μετά το στάδιο write-back η εισαγωγή σβήνεται από τον πίνακα

29

Απλοποιήσεις στη Δομή Δεδομένων

- Το στάδιο δρομολόγησης εντολών παγώνει εντολή όταν ανιχνεύσει εξάρτηση WAW
  - Ένας καταχώρητης πρέπει να εμφανίζεται μόνο MIA φορά στη στήλη Dest
- Αποθήκευση των καταχωρητών που θα χρησιμοποιηθούν για αποθήκευση (εκκρεμείς καταχωρητές): bit vector => WP[#Reg]
  - Ενεργοποίηση σε '1' στο στάδιο issue όταν χρησιμοποιείται ένας καταχωρητής, σε '0' στο στάδιο write-back
- In-order δρομολόγηση => δεν έχουμε εξάρτηση WAR => δεν χρειάζονται οι στήλες "Src1" , "Src2"
- Διαθεσιμότητα απασχολημένων μονάδων: bit vector => Busy[#FU]
  - (FU = Int, Add, Mult, Div)

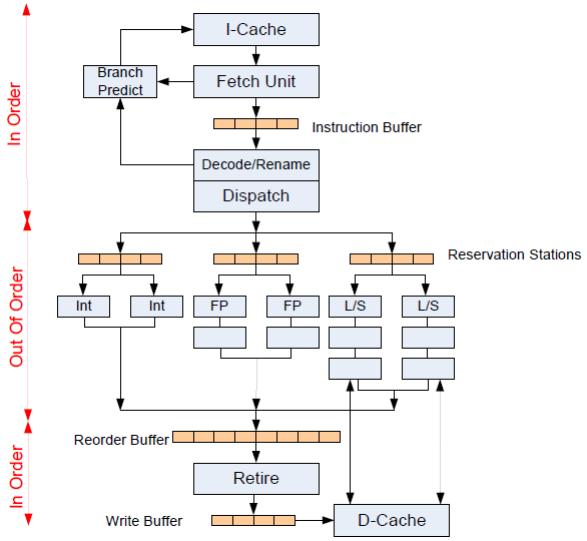
30

Παράδειγμα

	Functional Unit Status										Registers Reserved for Writes	
	Int(1)	Add(1)	Mult(3)			Div(4)				WB		
t0 I_1						f6					f6	
t1 I_2	f2						f6				f6, f2	
t2								f6		f2	f6, f2	I_2
t3 I_3			f0						f6		f6, f0	
t4				f0						f6	f6, f0	I_1
t5 I_4					f0	f8					f0, f8	
t6							f8			f0	f0, f8	I_3
t7 I_5		f10						f8			f8, f10	
t8									f8	f10	f8, f10	I_5
t9										f8	f8	I_4
t10 I_6		f6									f6	
t11										f6	f6	I_6

I_1	DIVD	f6,	f6,	f4
I_2	LD	f2,	45(r3)	
I_3	MULTD	f0,	f2,	f4
I_4	DIVD	f8,	f6,	f2
I_5	SUBD	f10,	f0,	f6
I_6	ADDD	f6,	f8,	f2

Ανατομία ενός Μοντέρνου Πυρήνα Superscalar Out-of-Order



Συμπεράσματα....

- Superscalar: φόρτωση/έκδοση πολλαπλών εντολών (4-6 max)
  - Εκμετάλλευση παραλληλισμού στις εντολές (instruction level parallelism) επιπλέον της τεχνικής Pipelining
  - Βελτίωση CPI με κόστος στη συχνότητα ρολογιού, χώρο και ενέργεια

33