Προηγμένη Αρχιτεκτονική Υπολογιστών

Παραλληλία Επιπέδου Εντολής (Instruction Level Parallelism)

Μιχάλης Ψαράκης

Εισαγωγή

- Instruction Level Parallelism (ILP):
  - Επικάλυψη της εκτέλεσης των εντολών για αύξηση της απόδοσης του επεξεργαστή
- Pipelining (διοχέτευση):
  - Κλασική τεχνική που εκμεταλλεύεται την παραλληλία επιπέδου εντολής
- Επιπλέον, δύο βασικές προσεγγίσεις:
  - Στατικές τεχνικές που βασίζονται στο λογισμικό (τεχνικές μεταγλωττιστή)
    - Χρησιμοποιούνται περισσότερο στην αγορά των ενσωματωμένων επεξεργαστών
  - Δυναμικές τεχνικές που βασίζονται στο υλικό
    - Χρησιμοποιούνται σε desktop PCs & servers

Απόδοση της διοχέτευσης

- Ο σκοπός είναι να μειώσουμε το CPI (clocks per instruction)
- Pipeline CPI = Ideal pipeline CPI +

  Structural stalls + Data hazard stalls + Control stalls
- Δομικοί κίνδυνοι (structural hazards):
  - Το υλικό δεν μπορεί να υποστηρίξει το συνδυασμό εντολών που έχουν οριστεί να εκτελεστούν σε έναν κύκλο
- Κίνδυνοι δεδομένων (data hazards):
  - Μία εντολή δεν μπορεί να εκτελεστεί στον κατάλληλο κύκλο επειδή χρειάζεται δεδομένα που δεν είναι ακόμα διαθέσιμα
- Κίνδυνοι ελέγχου (control hazards):
  - Δεν μπορεί να εκτελεστεί η κατάλληλη εντολή στον κατάλληλο κύκλο επειδή η ροή των εντολών δεν είναι αυτή που περίμενε η διοχέτευση

Εξάρτηση δεδομένων

- Εξάρτηση δεδομένων (data dependency):
  - Η εντολή *j* έχει εξάρτηση δεδομένων με την εντολή *i* εάν
    - Η εντολή *i* παράγει ένα αποτέλεσμα που μπορεί να χρησιμοποιηθεί από την εντολή *j*

```
i: add r1,r2,r3
j: sub r4,r1,r3
```

    - Η εντολή *j* έχει εξάρτηση δεδομένων με την εντολή *k* και η εντολή *k* έχει εξάρτηση δεδομένων με την *i*
- Η εξαρτημένες εντολές δεν μπορούν να εκτελεστούν παράλληλα
- Εάν η εξάρτηση δεδομένων προκαλέσει κίνδυνο στη διοχέτευση, ονομάζεται Read After Write (RAW) hazard

Εξάρτηση δεδομένων και ILP

- Οι εξαρτήσεις είναι ιδιότητα των προγραμμάτων
  - Πρέπει να διατηρήσουμε τη σειρά του προγράμματος:
    - οι εντολές θα πρέπει να εκτελεστούν με την σειρά με την οποία εμφανίζονται στον πηγαίο κώδικα
  - Η παρουσία εξάρτησης υποδηλώνει την πιθανότητα κινδύνου (hazard), αλλά τον εάν θα υπάρξει πραγματικός κίνδυνος και καθυστέρηση (stall) εξαρτάται από τη διοχέτευση
- Οι εξαρτήσεις δεδομένων θέτουν ένα άνω όριο στην ILP που μπορεί να επιτευχθεί
- Στόχος: εκμετάλλευση της παραλληλίας διατηρώντας τη σειρά του προγράμματος μόνο όπου επηρεάζει το αποτέλεσμα του προγράμματος
- Μία εξάρτηση μπορεί να ξεπεραστεί με δύο τρόπους:
  - Τηρώντας την εξάρτηση αλλά αποφεύγοντας τον κίνδυνο (π.χ. με χρονοπρογραμματισμό του κώδικα)
  - Εξαλείφοντας την εξάρτηση με μετασχηματισμό του κώδικα

Εξάρτηση ονόματος: αντι-εξάρτηση

- Εξάρτηση ονόματος (name dependence):
  - Όταν 2 εντολές χρησιμοποιούν τον ίδιο καταχωρητή ή θέση μνήμης (όνομα), αλλά δεν υπάρχει ροή δεδομένων μεταξύ των εντολών που να σχετίζεται με αυτό το όνομα
  - Υπάρχουν 2 περιπτώσεις εξάρτησης ονόματος
- Η *εντολή j* γράψει τον τελεστέο *πριν* η *εντολή i* τον διαβάσει

```
I: sub r4,r1,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

- Ονομάζεται αντι-εξάρτηση (anti-dependence)
  - Προκύπτει από την επαναχρησιμοποίηση του r1
- Εάν η αντι-εξάρτηση προκαλέσει κίνδυνο στη διοχέτευση, ονομάζεται Write After Read (WAR) hazard

Εξάρτηση ονόματος: εξάρτηση εξόδου

- Η *εντολή j* γράφει τον τελεστέο *πριν* η *εντολή i* τον γράψει

```
I: sub r1,r4,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

- Ονομάζεται εξάρτηση εξόδου (output dependence)
  - Προκύπτει από την επαναχρησιμοποίηση του r1
- Εάν η εξάρτηση εξόδου προκαλέσει κίνδυνο στη διοχέτευση, ονομάζεται Write After Write (WAW) hazard

Εξάρτηση ονόματος

- Δεν είναι πραγματική εξάρτηση δεδομένων, αλλά αποτελεί πρόβλημα όταν αλλάζουμε τη σειρά εκτέλεσης των εντολών
- Οι εντολές που εμπλέκονται σε εξάρτηση ονόματος μπορούν να εκτελεστούν παράλληλα εάν το όνομα που χρησιμοποιείται αλλάξει ώστε να μην συγκρούονται οι εντολές
  - Για την επίλυση της εξάρτησης ονόματος σε καταχωρητές χρησιμοποιείται μετονομασία καταχωρητών (register renaming)
  - Είτε στο λογισμικό (μεταγλωττιστής) είτε στο υλικό

Παραλληλία επιπέδου βρόχου

- Η παραλληλία μέσα σε μια βασική ενότητα (basic block) είναι περιορισμένη
  - Τυπικό μέγεθος της βασικής ενότητας = 7-10 εντολές
    - Πιθανή εξάρτηση μεταξύ των εντολών της ενότητας
  - Πρέπει να εκμεταλλευτούμε την παραλληλία μεταξύ διαφορετικών βασικών ενοτήτων
- Παραλληλία επιπέδου βρόχου (loop-level parallelism)
  - Εκμετάλλευση της παραλληλίας μεταξύ διαφορετικών επαναλήψεων

```
for (i=1; i<=1000; i=i+1)
     x[i] = x[i] + y[i];
```

  - Ξετύλιγμα του βρόχου (loop unrolling)
    - Στατικά (με τη βοήθεια του μεταγλωττιστή)
    - Δυναμικά (με πρόβλεψη της διακλάδωσης)

Τεχνικές λογισμικού για ILP

- Παράδειγμα: πρόσθεση τιμής σε διάνυσμα:

```
for (i=1000; i>0; i=i–1)
   x[i] = x[i] + s;
```

- Υποθέστε τις παρακάτω καθυστερήσεις:

Εντολή που παράγει	*Εντολή που καταναλώνει*	*Καθυστέρηση σε κύκλους*
FP ALU op	Άλλη FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0
Load int	Int ALU op	1
Int ALU op	Int ALU op	0

Που είναι οι κίνδυνοι

- Μετάφραση του βρόχου σε κώδικα MIPS:
  - Για απλότητα, υποθέστε ότι το τελευταίο στοιχείο που ενημερώνεται είναι στη διεύθυνση 8

```
Loop:L.D      F0,0(R1)      ;F0=vector element
     ADD.D    F4,F0,F2      ;add scalar from F2
     S.D      0(R1),F4      ;store result
     DADDUI   R1,R1,-8      ;decrement pointer 8B
     BNEZ     R1,Loop       ;branch R1!=zero
```


Καθυστερήσεις στον βρόχο

```
1 Loop: L.D    F0,0(R1)  ;F0=vector element
2       stall
3       ADD.D  F4,F0,F2  ;add scalar in F2
4       stall
5       stall
6       S.D    0(R1),F4  ;store result
7       DADDUI R1,R1,-8  ;decrement pointer 8B (DW)
8       stall
9       BNEZ   R1,Loop   ;branch R1!=zero
```

- 9 κύκλοι: Αναδιάταξη κώδικα για μείωση των καθυστερήσεων

Μείωση των καθυστερήσεων

```
1 Loop: L.D    F0,0(R1)
2       DADDUI R1,R1,-8
3       ADD.D  F4,F0,F2
4       stall
5       stall
6       S.D    8(R1),F4 ;altered offset when move DSUBUI
7       BNEZ   R1,Loop
```

- Εναλλαγή των DADDUI και S.D αλλάζοντας την διεύθυνση του S.D
- 7 κύκλοι, αλλά μόνο 3 για εκτέλεση (L.D, ADD.D,S.D), και 4 επιβάρυνση του βρόχου
  - Πώς θα κάνουμε την εκτέλεση του βρόχου πιο γρήγορη?

Loop unrolling: 4 φορές

```
1 Loop:L.D    F0,0(R1)
3      ADD.D  F4,F0,F2
6      S.D    0(R1),F4     ;drop DSUBUI & BNEZ
7      L.D    F6,-8(R1)
9      ADD.D  F8,F6,F2
12     S.D    -8(R1),F8    ;drop DSUBUI & BNEZ
13     L.D    F10,-16(R1)
15     ADD.D  F12,F10,F2
18     S.D    -16(R1),F12  ;drop DSUBUI & BNEZ
19     L.D    F14,-24(R1)
21     ADD.D  F16,F14,F2
24     S.D    -24(R1),F16
25     DADDUI R1,R1,#-32   ;alter to 4*8
27     BNEZ   R1,LOOP
```

1 cycle stall

2 cycles stall

- 27 κύκλοι ή 6.75 ανά επανάληψη
  - Υποθέτουμε ότι ο R1 είναι πολλαπλάσιο του 4
- Πώς θα γράψουμε το βρόχο για να μειώσουμε τις καθυστερήσεις ?

Loop unrolling: μείωση καθυστερήσεων

```
1 Loop:L.D    F0,0(R1)
2       L.D    F6,-8(R1)
3       L.D    F10,-16(R1)
4       L.D    F14,-24(R1)
5       ADD.D  F4,F0,F2
6       ADD.D  F8,F6,F2
7       ADD.D  F12,F10,F2
8       ADD.D  F16,F14,F2
9       S.D    0(R1),F4
10      S.D    -8(R1),F8
11      S.D    -16(R1),F12
12      DSUBUI R1,R1,#32
13      S.D    8(R1),F16        ; 8-32 = -24
14      BNEZ   R1,LOOP
```

- 14 κύκλοι ή 3.5 ανά επανάληψη

Λεπτομέρειες του loop unrolling

- Δεν γνωρίζουμε συνήθως το άνω όριο του βρόχου
- Υποθέστε ότι είναι n, και ότι θέλουμε να ξετυλίξουμε το βρόχο k φορές
- Αντί για έναν απλό ξετυλιγμένο βρόχο, παράγουμε δύο συνεχόμενους βρόχους
  - Ο πρώτος έχει το ξετυλιγμένο σώμα και επαναλαμβάνεται n/k times
  - Ο δεύτερος επαναλαμβάνεται n mod k φορές και το σώμα του βρόχου είναι το ίδιο με το αρχικό
- Για μεγάλες τιμές του n, ο περισσότερος χρόνος καταναλώνεται στην εκτέλεση του ξετυλιγμένου βρόχου

Περιορισμοί του loop unrolling

1. Μείωση της επιβάρυνσης του βρόχου που γλιτώνουμε με κάθε επιπλέον ξετύλιγμα
   - Amdahl's Law
2. Αύξηση του μεγέθους του κώδικα
   - Για μεγαλύτερους βρόχους, η αύξηση του μεγέθους μπορεί να προκαλέσει αύξηση στο ρυθμός αστοχίας της κρυφής μνήμης
3. Πίεση καταχωρητών (register pressure): πιθανή έλλειψη καταχωρητών από το επιθετικό ξετύλιγμα και το χρονοπρογραμματισμό
   - Εάν δεν είναι εφικτό να κατανείμουμε όλες τις «ενεργές» μεταβλητές σε καταχωρητές μπορεί να χάσουμε τα πλεονεκτήματα της τεχνικής

Δυναμικός Χρονοπρογραμματισμός

- Δυναμικός χρονοπρογραμματισμός ή χρονοδρομολόγηση (dynamic scheduling):
  το υλικό αναδιατάσσει την εκτέλεση των εντολών για να μειώσει τις καθυστερήσεις ενώ διατηρεί τη ροή δεδομένων (data flow) και τη συμπεριφορά των εξαιρέσεων (exception behavior)
  - Σε αντίθεση με τους στατικά χρονοπρογραμματιζόμενους επεξεργαστές που βασίζονται σε τεχνικές του μεταγλωττιστή για να αυξήσουν το ILP
- Πλεονεκτήματα:
  - Χειρίζεται περιπτώσεις όπου οι εξαρτήσεις δεν είναι γνωστές κατά τη διάρκεια της μεταγλώττισης (π.χ. περιέχουν αναφορές στη μνήμη)
    - Επιτρέπει στον επεξεργαστή, όταν συμβούν μεγάλες καθυστερήσεις (π.χ. από αστοχίες της κρυφής μνήμης) να εκμεταλλευτεί το χρόνο μέχρι να επιλυθεί η αστοχία ώστε να εκτελέσει κάποιο άλλο κώδικα
  - Απλοποιεί τη μεταγλώττιση
  - Επιτρέπει στον κώδικα που έχει εκτελεστεί για ένα συγκεκριμένο μηχανισμό διοχέτευσης να εκτελεστεί αποδοτικά και σε άλλη διοχέτευση

Βασική ιδέα

- Περιορισμός της απλής διοχέτευσης: σειριακή εκκίνηση (issue) και εκτέλεση (execution) εντολών
  - Εάν μία εντολή j εξαρτάται από μία εντολή i (με μεγάλο χρόνο εκτέλεσης), τότε όλες οι εντολές μετά την j πρέπει να καθυστερήσουν μέχρι να τελειώσει η i
  - Εάν υπάρχουν πολλαπλές λειτουργικές μονάδες θα μείνουν ανενεργές

```
DIV.D F0,F2,F4
ADD.D F10,F0,F8
SUB.D F12,F8,F14
```

- Βασική ιδέα του δυναμικού χρονοπρογραμματισμού:
  - Επιτρέπει σε εντολές να συνεχίσουν παρά την ύπαρξη των κινδύνων
  - Επιτρέπει την εκτέλεση εκτός-σειράς (out-of-order execution)

Εκτέλεση εκτός σειράς

- Επιτρέπει την εκτέλεση εκτός-σειράς (out-of-order execution) και υπονοεί την ολοκλήρωση εκτός σειράς (out-of-order completion), π.χ. SUBD
  - Η εκκίνηση των εντολών ωστόσο εξακολουθεί να είναι εντός σειράς (in-order issue)
- Δημιουργεί την πιθανότητα κινδύνων WAR and WAW

```
DIV.D F0,F2,F4
ADD.D F6,F0,F8
SUB.D F8,F10,F14  (antidependence)
MUL.D F6,F10,F8   (output dependence)
```

- Κάνει πιο δύσκολο των χειρισμό των εξαιρέσεων
  - Καμία εντολή δεν μπορεί να προκαλέσει εξαίρεση έως ότου διαπιστωθεί ότι η εντολή αυτή θα εκτελεστεί πραγματικά
  - Μπορεί να προκαλέσει ανακριβείς εξαιρέσεις (imprecise exceptions)

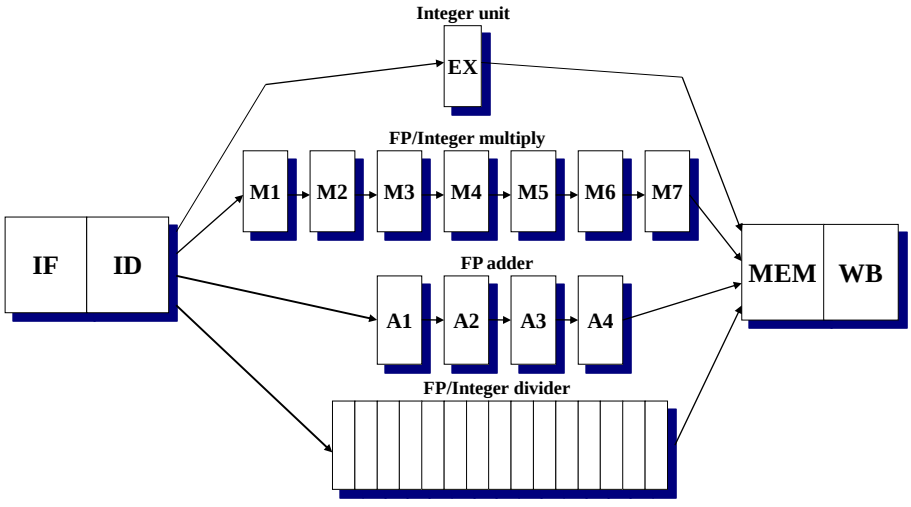
Εκκίνηση εντολής

- Η απλή διοχέτευση ελέγχει για δομικούς κινδύνους και κινδύνους δεδομένων στο στάδιο προσκόμισης εντολής (Instruction Decode (ID), ή αλλιώς Instruction Issue)
- Στην εκτέλεση εκτός σειράς χωρίζουμε το στάδιο ID σε 2 στάδια:
  - *Εκκίνηση (Issue):* Αποκωδικοποιεί την εντολή και ελέγχει για δομικούς κινδύνους
  - *Ανάγνωση τελεστέων (Read operands):* Περιμένει μέχρι να μην υπάρχουν κίνδυνοι δεδομένων, και έπειτα διαβάζει τους τελεστέους
- Διακρίνει πότε μια εντολή *αρχίζει την εκτέλεση* και πότε *ολοκληρώνει την εκτέλεση*
  - Μεταξύ των δύο αυτών στιγμών, η εντολή είναι *υπό εκτέλεση*

Εκτέλεση εντολής

- Πολλαπλές εντολές βρίσκονται ταυτόχρονα υπό εκτέλεση
  - Η ταυτόχρονη εκτέλεση απαιτεί πολλαπλές λειτουργικές μονάδες ή λειτουργικές μονάδες με διοχέτευση ή και τα δύο
- Το στάδιο IF προηγείται του σταδίου εκκίνησης (issue) και αποθηκεύει την εντολή σε ουρά αναμονής
- Οι εντολές διέρχονται από το στάδιο issue εντός σειράς (in-order issue)
- Οι εντολές μπορεί να περιμένουν (stall) ή να παρακάμψουν (bypass) άλλες εντολές και να εισέλθουν στην εκτέλεση εκτός σειράς (out-of-order execution)
- Το στάδιο EXE ακολουθεί το στάδιο της ανάγνωσης τελεστέων (read operands)
  - Μπορεί να χρειαστεί πολλούς κύκλους
  - Ολοκληρώνει την εκτέλεση εκτός σειράς (out-of-order completion)

Πολλαπλές λειτουργικές μονάδες



Προσέγγιση Tomasulo

- Προτάθηκε από τον Robert Tomasulo (1967) για τη μονάδα κινητής υποδιαστολής του IBM 360/91
- Στόχος: υψηλή απόδοση κινητής υποδιαστολής χωρίς εξειδικευμένους μεταγλωττιστές (συμβατότητα για ολόκληρη της οικογένεια 360)
- Χαρακτηριστικά του IBM 360/91
  - Δεν είχε κρυφές μνήμες
    - Μεγάλες καθυστερήσεις στις προσπελάσεις μνήμης
  - Μεγάλες καθυστερήσεις στις πράξεις κινητής υποδιαστολής
  - Μικρός αριθμός καταχωρητών κινητής υποδιαστολής
    - Δεν επέτρεπε αποτελεσματικό χρονοπρογραμματισμό από το μεταγλωττιστή
- Η προσέγγιση χρησιμοποιήθηκε ευρέως πολλά χρόνια αργότερα
  - Alpha 21264, Pentium 4, AMD Opteron, Power 5, …

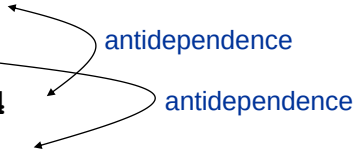
Βασική ιδέα

- Η εκτέλεση εκτός σειράς δημιουργεί κινδύνους WAR και WAW
- Η προσέγγιση Tomasulo επιχειρεί να επιλύσει τους κινδύνους:
  - Κίνδυνοι RAW: μία εντολή εκτελείται μόνο όταν οι τελεστέοι της είναι διαθέσιμοι
  - Κίνδυνοι WAR και WAW: χρησιμοποιείται μετονομασία καταχωρητών (register renaming)

Μετονομασία καταχωρητών (1)

- Παράδειγμα:

```
DIV.D F0,F2,F4
ADD.D F6,F0,F8
S.D   F6,0(R1)
SUB.D F8,F10,F14
MUL.D F6,F10,F8
```



+ output dependence (F6)

Μετονομασία καταχωρητών (2)

- Παράδειγμα:

```
DIV.D F0,F2,F4
ADD.D S,F0,F8
S.D   S,0(R1)
SUB.D T,F10,F14
MUL.D F6,F10,T
```

- Τώρα παραμένουν μόνο κίνδυνοι που θα καθορίσουν τη σειρά εκτέλεσης
- Οι μετέπειτα χρήσεις του F8 πρέπει να αντικατασταθούν από τον καταχωρητή T
  - Είτε με πολύπλοκη ανάλυση στον μεταγλωττιστή είτε στο υλικό

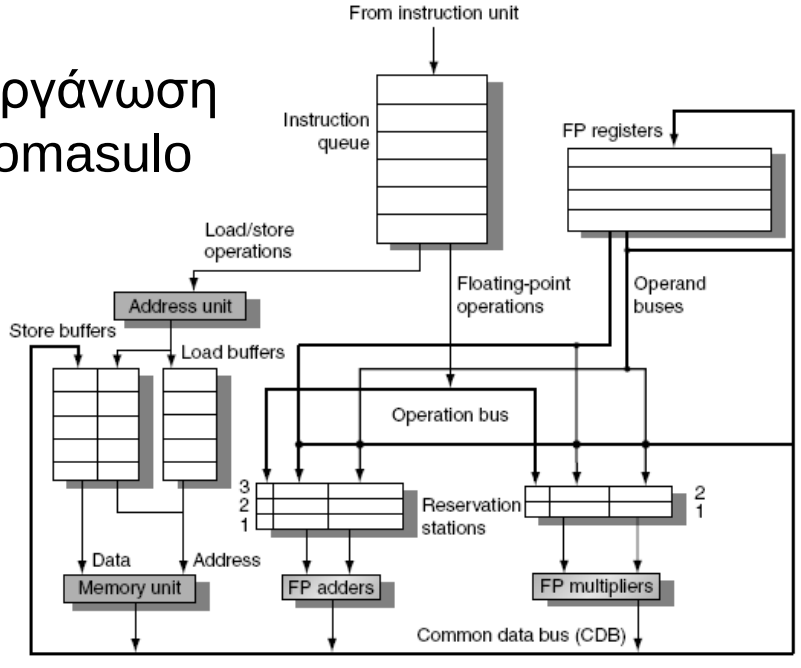
Αλγόριθμος Tomasulo (1)

- Η μετονομασία καταχωρητών υλοποιείται με τους σταθμούς κράτησης (reservation stations – RS)
- Ένας σταθμός κράτησης περιέχει:
  - Την εντολή
  - Προσωρινή αποθήκευση των τιμών των τελεστέων (όταν γίνουν διαθέσιμοι)
  - Τους αριθμούς των σταθμών κράτησης που θα παρέχουν τις τιμές των τελεστέων
- Ένας σταθμός κράτησης προσκομίζει και αποθηκεύει προσωρινά έναν τελεστέο μόλις αυτός είναι διαθέσιμος (όχι απαραίτητα από το αρχείο καταχωρητών)

Αλγόριθμος Tomasulo (2)

- Οι εκκρεμείς εντολές προσδιορίζουν το σταθμό κράτησης στον οποίο θα στείλουν την έξοδό τους
  - Τα αποτελέσματα τοποθετούνται σε έναν δίαυλο δεδομένων (common data bus – CDB)
- Με την εκκίνηση των εντολών, οι καταχωρητές μετονομάζονται με χρήση του σταθμού κράτησης
- Περισσότεροι σταθμοί κράτησης από καταχωρητές
- Προσωρινές μνήμες φόρτωσης και αποθήκευσης (load and store buffers)
  - Περιέχουν δεδομένα και διευθύνσεις και λειτουργούν όπως οι σταθμοί κράτησης

Οργάνωση Tomasulo



Πεδία του σταθμού κράτησης

- Op: Πράξη που θα εκτελεστεί στη μονάδα (π.χ., + ή –)
- Vj, Vk: Τιμές των τελεστεών
  - Για φορτώσεις το Vk χρησιμοποιείται για το πεδίο μετατόπισης
- Qj, Qk: Σταθμοί κράτησης που θα παράγουν τους τελεστέους
  - Σημείωση: Qj,Qk=0 => ο τελεστέος είναι διαθέσιμος
- A: Κρατάει πληροφορίες για τον υπολογισμό της διεύθυνσης μνήμης για φόρτωση ή αποθήκευση
- Busy: Υποδηλώνει ότι ο σταθμός κράτησης και η αντίστοιχη λειτουργική μονάδα είναι απασχολημένοι

Το αρχείο καταχωρητών έχει ένα πεδίο :

- Register status: υποδηλώνει ποια λειτουργική μονάδα θα γράψει κάθε καταχωρητή.
  - Κενό σημαίνει ότι δεν υπάρχουν εκκρεμείς εντολές που θα γράψουν σε αυτό τον καταχωρητή

Τα 3 βήματα του Tomasulo

- Εκκίνηση (Issue) ή έκδοση
  - Παίρνει την επόμενη εντολή από την ουρά εντολών (FIFO queue)
  - Εάν υπάρχει διαθέσιμος RS, στέλνει την εντολή στον RS με τις τιμές των τελεστέων (εάν είναι διαθέσιμες)
  - Εάν δεν υπάρχει διαθέσιμος RS τότε υπάρχει structural hazards και η εντολή καθυστερεί (stall)
  - Εάν οι τελεστέοι δεν βρίσκονται στους καταχωρητές η εντολή καθυστερεί (stall) και παρακολουθεί τις λειτουργικές μονάδες που θα τους παράγουν
    - Έτσι λειτουργεί η μετονομασία και αποφεύγονται οι κίνδυνοι WAR και WAW
- Εκτέλεση (Execute)
  - Όταν ένας τελεστέος γίνει διαθέσιμος, τον αποθηκεύει στους RS που τον περιμένουν
  - Όταν όλοι οι τελεστέοι είναι διαθέσιμοι, η εντολή μπορεί να εκτελεστεί
    - Έτσι αποφεύγονται οι κίνδυνοι RAW
- Εγγραφή αποτελέσματος (Write result)
  - Γράφει το αποτέλεσμα στον CDB και από κει στους RS (και στους store buffers) και στους καταχωρητές
    - Οι αποθηκεύσεις γράφουν δεδομένα στη μνήμη κατά τη διάρκεια αυτού του βήματος

Προσπελάσεις μνήμης και εξαιρέσεις

- Οι φορτώσεις και οι αποθηκεύσεις διατηρούνται στη σειρά προγράμματος
  - Απαιτούν διαδικασία 2 βημάτων:
    - 1ο βήμα: υπολογίζει την τελική διεύθυνση και την τοποθετεί στις προσωρινές μνήμες (buffers)
    - 2ο βήμα: εκτελούνται μόλις η μονάδα μνήμης είναι διαθέσιμη και υπολογιστεί η τιμή προς εγγραφή (για τις αποθηκεύσεις)
- Για να διατηρήσουμε τη συμπεριφορά εξαιρέσεων καμία εντολή δεν επιτρέπεται να ξεκινήσει εκτέλεση έως ότου όλες οι διακλαδώσεις που προηγούνται της εντολής στο πρόγραμμα έχουν ολοκληρωθεί
  - Για να αποφύγουν αυτόν τον περιορισμό, υπάρχουν μέθοδοι που καταγράφουν την παρουσία της εξαίρεσης χωρίς να την παράγουν
    - Επιτρέπουν να ξεκινήσει η εκτέλεση της εντολής και να περιμένει στην εγγραφή αποτελέσματος

Παράδειγμα Tomasulo

Instruction stream

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	

3 Load/Buffers

Reservation Stations:

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

FU count down

3 FP Adder RS
2 FP Mult RS

Register result status:

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

Clock cycle counter

Το παράδειγμα Tomasulo είναι από τις διαφάνειες του Prof. David Patterson για το μάθημα "Graduate Computer Architecture", Electrical Engineering and Computer Sciences, University of California, Berkeley

Παράδειγμα: κύκλος 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					

- Εκκίνηση της LD (issue)

Παράδειγμα: κύκλος 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					

- Εκκίνηση της LD (issue)
- Σημείωση: Μπορεί να υπάρχουν πολλές εκκρεμείς φορτώσεις

Παράδειγμα: κύκλος 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					

- Εκκίνηση της MULT (issue)
- Σημείωση: τα ονόματα των καταχωρητών δεν διατηρούνται στα RS
- Ολοκλήρωση της Load1. Ποιος περιμένει την Load1 ?

Παράδειγμα: κύκλος 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				

- Εκκίνηση της SUBD (issue)
- Ολοκλήρωση της Load2. Ποιος περιμένει την Load2 ?

Παράδειγμα: κύκλος 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			

- Εκκίνηση της DIVD (issue)
- Ξεκινούν οι μετρητές για την Add1 (2 κύκλοι) και την Mult1 (10 κύκλοι)

Παράδειγμα: κύκλος 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			

- Εκκίνηση της ADDD (issue) παρά την εξάρτηση ονόματος λόγω του F6

Παράδειγμα: κύκλος 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			

- Ολοκλήρωση της Add1 (SUBD). Ποιος περιμένει την Add1?

Παράδειγμα: κύκλος 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			

- Ξεκινάει ο μετρητής για την Add2 (2 κύκλοι)

Παράδειγμα: κύκλος 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			

- Συνεχίζουν την εκτέλεση οι Add2 και Mult1

Παράδειγμα: κύκλος 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			

- Ολοκλήρωση της Add2(ADDD). Ποιος περιμένει την Add2?

Παράδειγμα: κύκλος 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			

- Εγγραφή του αποτελέσματος της ADDD
- Οι «γρήγορες» εντολές ολοκληρώθηκαν (SUBD, ADDD)

Παράδειγμα: κύκλος 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			

- Συνεχίζει την εκτέλεση η Mult1

Παράδειγμα: κύκλος 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			

- Συνεχίζει την εκτέλεση η Mult1

Παράδειγμα: κύκλος 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			

- Συνεχίζει την εκτέλεση η Mult1

Παράδειγμα: κύκλος 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			

- Ολοκλήρωση της Mult1 (MULTD). Ποιος περιμένει την Mult1?

Παράδειγμα: κύκλος 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			

- Ξεκινάει ο μετρητής της Mult2 (40 κύκλοι)

Παραλείπουμε τους επόμενους κύκλους

Παράδειγμα: κύκλος 55

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					
1	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

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

Παράδειγμα: κύκλος 56

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	56	
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					
0	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

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

- Ολοκλήρωση της Mult2 (DIVD). Ποιος περιμένει την Mult2?

Παράδειγμα: κύκλος 57

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	56	57
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					
	Mult2	No					

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...
57	FU	M*F4	M(A2)		(M-M+M	(M-M)	Result		

- Εκκίνηση εντός σειράς (in-order issue), εκτέλεση εκτός σειράς (out-of-order execution) και ολοκλήρωση εκτός σειράς (out-of-order completion)

Μειονεκτήματα του Tomasulo

- Πολυπλοκότητα
  - Κάθε RS πρέπει να περιέχει ένα συσχετιστικό χώρο αποθήκευσης υψηλής ταχύτητας και πολύπλοκη λογική ελέγχου
- Περιορισμός απόδοσης λόγω CDB (Common Data Bus)
  - Κάθε CDB πρέπει να συνδέεται σε πολλαπλές λειτουργικές μονάδες
    ⇒ πολλές διασυνδέσεις
  - Ο αριθμός των λειτουργικών μονάδων που μπορούν να ολοκληρώνουν την εκτέλεση ανά κύκλο = 1!
    - Πολλαπλοί CDB ⇒ περισσότερη λογική ελέγχου στις λειτουργικές μονάδες

Εικασία για αύξηση της ILP

- Για να πετύχουμε περισσότερη παραλληλία επιπέδου εντολής (ILP) πρέπει να ξεπεράσουμε τις εξαρτήσεις ελέγχου (control dependence)
  - Δεν αρκεί η πρόβλεψη διακλάδωσης (branch prediction)
    - Μειώνει τις καθυστερήσεις που αφορούν τις εντολές διακλάδωσης αλλά στους επεξεργαστές που εκτελούν πολλές εντολές ανά κύκλο η ακρίβεια πρόβλεψης δεν αρκεί
- Λύση: Εικασία βασισμένη στο υλικό (Hardware speculation)
  - Εικάζει το αποτέλεσμα των διακλαδώσεων και εκτελεί το πρόγραμμα σαν οι εικασίες να είναι σωστές
  - Μηχανισμός για την περίπτωση που η εικασία είναι λανθασμένη
- Σε τι διαφέρει ο δυναμικός χρονοπρογραμματισμός με ή χωρίς εικασία;
  - Χωρίς εικασία: Προσκομίζει (fetch) και εκκινεί (issue) εντολές
  - Με εικασία: Προσκομίζει (fetch), εκκινεί (issue), και εκτελεί (execute) εντολές σαν οι προβλέψεις των διακλαδώσεων να είναι σωστές

Hardware-based speculation

- Συνδυάζει 3 βασικές ιδέες:
  - Δυναμική πρόβλεψη διακλάδωσης για να επιλέξει ποιες εντολές θα εκτελέσει
  - Εικασία για να επιτρέψει την εκτέλεση των εντολών προτού επιλυθούν οι εξαρτήσεις ελέγχου
    - + δυνατότητα να αναιρέσει τα αποτελέσματα εντολών που ακολουθούν λανθασμένες εικασίες
  - Δυναμικές τεχνικές για να χρονοπρογραμματίσει και να εκτελέσει παράλληλα πολλές βασικές ενότητες (basic blocks)
- Δυναμικός χρονοπρογραμματισμός χωρίς εικασία:
  - Απλά επικαλύπτει τις βασικές ενότητες
    - π.χ. σε έναν βρόχο απαιτείται η επίλυση της διακλάδωσης προτού εκτελεστούν οι εντολές της επόμενης επανάληψης

Προσθήκη εικασίας στον Tomasulo

- Πρέπει να διαχωρίσουμε την εκτέλεση μιας εντολής από το να επιτρέψουμε σε μια εντολή να τελειώσει ή να «υποβληθεί» (ή επικυρωθεί) (commit)
- Επιτρέπουμε σε μια εντολή να εκτελεστεί και να προωθήσει τα αποτελέσματά της σε άλλες εντολές χωρίς να επιτρέπουμε να κάνει ενημερώσεις που δεν μπορούν να αναιρεθούν
  - Η χρήση της προώθησης είναι μια κατ' εικασία ανάγνωση καταχωρητή
- Όταν μια εντολή δεν είναι πλέον εικασία, της επιτρέπουμε να ενημερώσει το αρχείο καταχωρητών ή τη μνήμη
  - Αυτό το επιπλέον βήμα ονομάζεται υποβολή εντολής (instruction commit)

Υποβολή εντός σειράς

- Στην εικασία επιτρέπεται η εκτέλεση εκτός σειράς (out-of-order execution) αλλά η υποβολή γίνεται με τη σειρά (in-order commit)
- Η προσθήκη της φάσης υποβολής (commit) απαιτεί επιπλέον προσωρινή μνήμη
- Προσωρινή μνήμη αναδιάταξης (Re-Order Buffer, ROB)
  - Αποθηκεύει τα αποτελέσματα των εντολών που έχουν εκτελεστεί αλλά δεν έχουν υποβληθεί
  - Χρησιμοποιείται επίσης για να περάσει αποτελέσματα μεταξύ εντολών που εικάζονται

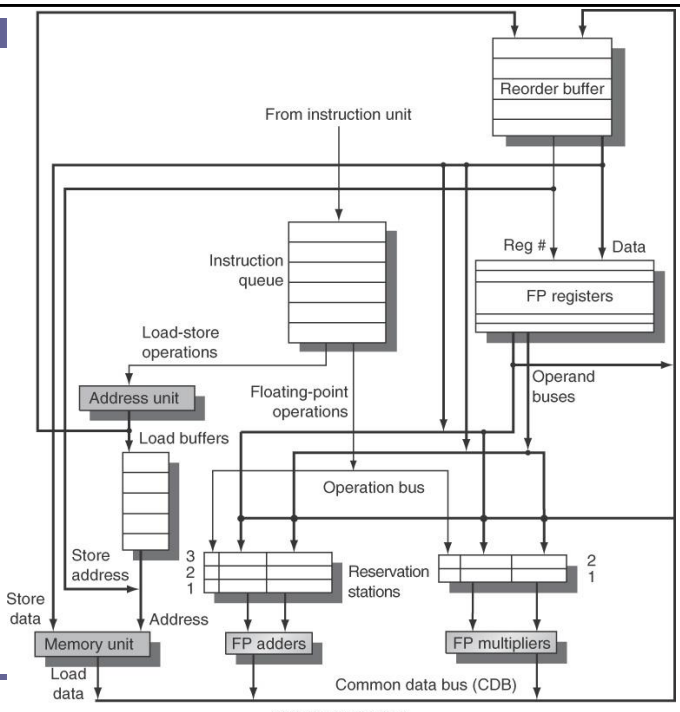
Reorder buffer (ROB)

- Στον αλγόριθμο Tomasulo, μόλις μια εντολή γράψει το αποτέλεσμά της (write result stage), οι εντολές που ακολουθούν και έχουν περάσει το στάδιο εκκίνησης θα βρουν το αποτέλεσμα στο αρχείο καταχωρητών
- Στην εικασία, το αρχείο καταχωρητών δεν ενημερώνεται έως ότου η εντολή υποβληθεί
  - Και άρα γνωρίζουμε οριστικά ότι η εντολή θα πρέπει να εκτελεστεί
- Επομένως, ο ROB παρέχει τελεστέους στο διάστημα μεταξύ της ολοκλήρωσης της εντολής και της υποβολής της εντολής
  - Ο ROB είναι μια πηγή τελεστέων για εντολές, όπως ακριβώς οι σταθμοί κράτησης (RS) για τον αλγόριθμο Tomasulo
  - Ο ROB επεκτείνει τον αριθμό των καταχωρητών όπως οι RS

Πεδία του ROB

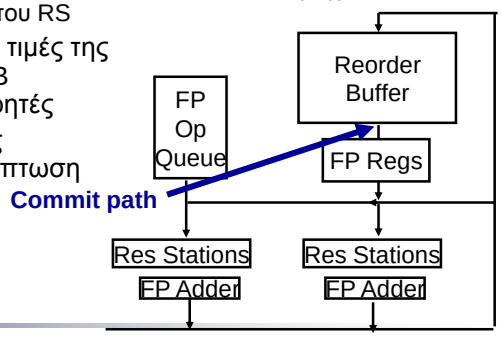
- Κάθε καταχώρηση του ROB περιέχει 4 πεδία:
- Τύπος εντολής (instruction type)
  - Διακλάδωση (δεν έχει προορισμό για το αποτέλεσμα), αποθήκευση (έχει προορισμό μια διεύθυνση μνήμης), ή λειτουργία καταχωρητή (λειτουργία ALU ή φόρτωση, που έχουν προορισμό έναν καταχωρητή)
- Προορισμός (destination)
  - Αριθμός καταχωρητή (για φορτώσεις και πράξεις ALU) ή διεύθυνση μνήμης (για αποθηκεύσεις)
    - Όπου θα γραφτεί το αποτέλεσμα της εντολής
- Τιμή (value)
  - Τιμή του αποτελέσματος της εντολής έως ότου υποβληθεί
- Ετοιμότητα (ready)
  - Δείχνει ότι η εντολή έχει ολοκληρώσει την εκτέλεση και η τιμή είναι έτοιμη

- Δομή μονάδας FP με Tomasulo και εικασία



Λειτουργία του ROB

- Αποθηκεύει τις εντολές ως FIFO
  - Με τη σειρά που εκτελούν το στάδιο issue
- Όταν η εντολή ολοκληρωθεί, τα αποτελέσματα τοποθετούνται στον ROB
  - Τροφοδοτεί τους τελεστέους στις υπόλοιπες εντολές όταν είναι μεταξύ ολοκλήρωσης εκτέλεσης (execution complete) και υποβολής (commit)
  - Οι ετικέτες στα αποτελέσματα έχουν τον αριθμό καταχώρησης του ROB αντί για τον αριθμό του RS
- Υποβολή εντολής (commit): οι τιμές της εντολής στην κορυφή του ROB τοποθετούνται στους καταχωρητές
- Μπορούμε να αναιρέσουμε τις εικαζόμενες εντολές στην περίπτωση λάθος πρόβλεψης ή σε εξαίρεση
  - Παρέχει ακριβείς εξαιρέσεις (precise exceptions)



4 βήματα του Tomasulo με εικασία

- 1. Εκκίνηση (Issue): παίρνει την επόμενη εντολή από την ουρά των εντολών
  - Εκκινεί την εντολή εάν υπάρχουν διαθέσιμος RS και κενή θέση στον ROB
  - Στέλνει τους τελεστέους στον RS εάν είναι διαθέσιμοι στον ROB ή στους καταχωρητές
  - Στέλνει στον RS τον αριθμό του ROB που δεσμεύεται για το αποτέλεσμα
  - Αυτό το στάδιο ονομάζεται και αποστολή (dispatch)
- 2. Εκτέλεση (Execution): εκτελεί τη λειτουργία
  - Όταν και οι δυο τελεστέοι είναι διαθέσιμοι εκτελεί τη λειτουργία
  - Εάν δεν είναι διαθέσιμοι, παρακολουθεί τον CDB περιμένοντας το αποτέλεσμα
    - Ελέγχει για RAW

4 βήματα του Tomasulo με εικασία

- 3. Εγγραφή αποτελέσματος (Write result): ολοκληρώνει την εκτέλεση
  - Γράφει το αποτέλεσμα στον CDB και από κει σε όλους τους RS που περιμένουν και στον ROB
  - Σημειώνει τον RS ως διαθέσιμο
- 4. Υποβολή (Commit): ενημερώνει τον προορισμό με το αποτέλεσμα
  - Όταν η εντολή φτάσει στην κορυφή του ROB και το αποτέλεσμα είναι διαθέσιμο, τότε ενημερώνει τον προορισμό (καταχωρητή ή μνήμη) και απομακρύνει την εντολή από τον ROB
  - Σε περίπτωση διακλάδωσης με λανθασμένη πρόβλεψη αδειάζει τον ROB
    - Καλείται και αποφοίτηση (graduation)
    - Πολλοί επεξεργαστές επανέρχονται αμέσως μετά από μια λανθασμένη πρόβλεψη (προτού η διακλάδωση φτάσει στην κορυφή του ROB)
      - Επιτρέπουν στις εντολές που προηγούνται στον ROB να συνεχίσουν και ακυρώνουν αυτές που ακολουθούν

Παράδειγμα 1

- Εκτέλεση του παρακάτω κώδικα σε:
  - έναν επεξεργαστή με μηχανισμό Tomasulo χωρίς εικασία και
  - έναν επεξεργαστή με μηχανισμό Tomasulo και υποστήριξη εικασίας
- Ποια θα είναι η κατάσταση του επεξεργαστή όταν η εντολή MUL.D ολοκληρώσει την εκτέλεσή της;
- Καθυστερήσεις πράξεων κινητής υποδιαστολής:
  - Πρόσθεση 2 κύκλοι ρολογιού, πολλαπλασιασμός 6 κύκλοι, και διαίρεση 12 κύκλοι

```
L.D F6,32(R2)
L.D F2,44(R3)
MUL.D F0,F2,F4
SUB.D F8,F6,F2
DIV.D F10,F0,F6
ADD.D F6,F8,F2
```


Tomasulo χωρίς εικασία

Instructions	Issues at clock number	Executes (start-to-end)	Memory access at clock number	Writes CDB at clock number
L.D F6,32(R2)	1	2	3	4
L.D F2,44(R3)	2	3	4	5
MUL.D F0,F2,F4	3	6-11		
SUB.D F8,F6,F2	4	6-7		8
DIV.D F10,F0,F6	5			
ADD.D F6,F8,F2	6	9-10		11

- Tomasulo χωρίς εικασία
- Εκτός των MUL.D και DIV.D οι υπόλοιπες εντολές έχουν γράψει το αποτέλεσμά τους

		Instruction status		
Instruction		Issue	Execute	Write Result
L.D	F6,32(R2)	√	√	√
L.D	F2,44(R3)	√	√	√
MUL.D	F0,F2,F4	√	√	
SUB.D	F8,F2,F6	√	√	√
DIV.D	F10,F0,F6	√		
ADD.D	F6,F8,F2	√	√	√

	Reservation stations							
Name	Busy	Op	Vj	Vk		Qj	Qk	A
Load1	no							
Load2	no							
Add1	no							
Add2	no							
Add3	no							
Mult1	yes	MUL	Mem[45 + Regs[R3]]	Regs[F4]				
Mult2	yes	DIV		Mem[34 + Regs[R2]]		Mult1		

	Register status								
Field	F0	F2	F4	F6	F8	F10	F12	...	F30
Qi	Mult1					Mult2			

Tomasulo με εικασία

Instructions	Issues at clock number	Executes (start-to-end)	Memory access at clock number	Writes CDB at clock number	Commits at clock number
L.D F6,32(R2)	1	2	3	4	5
L.D F2,44(R3)	2	3	4	5	6
MUL.D F0,F2,F4	3	6-11			
SUB.D F8,F6,F2	4	6-7		8	
DIV.D F10,F0,F6	5				
ADD.D F6,F8,F2	6	9-10		11	

Tomasulo με εικασία

- Μόνο οι δύο εντολές L.D έχουν υποβληθεί, αν και άλλες έχουν ολοκληρώσει την εκτέλεσή τους.

Reorder buffer

Entry	Busy	Instruction	State	Destination	Value
1	no	L.D F6,32(R2)	Commit	F6	Mem[34 + Regs[R2]]
2	no	L.D F2,44(R3)	Commit	F2	Mem[45 + Regs[R3]]
3	yes	MUL.D F0,F2,F4	Write result	F0	#2 × Regs[F4]
4	yes	SUB.D F8,F2,F6	Write result	F8	#2 – #1
5	yes	DIV.D F10,F0,F6	Execute	F10	
6	yes	ADD.D F6,F8,F2	Write result	F6	#4 + #2

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Load1	no							
Load2	no							
Add1	no							
Add2	no							
Add3	no							
Mult1	no	MUL.D	Mem[45 + Regs[R3]]	Regs[F4]			#3	
Mult2	yes	DIV.D		Mem[34 + Regs[R2]]	#3		#5	

FP register status

Field	F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
Reorder #	3						6		4	5
Busy	yes	no	no	no	no	no	yes	...	yes	yes

- Οι SUB.D και ADD.D δεν θα υποβληθούν έως ότου υποβληθεί η MUL.D, αν και τα αποτελέσματά τους είναι διαθέσιμα για άλλες εντολές

Παράδειγμα 2

- Εκτέλεση του παρακάτω βρόχου σε έναν επεξεργαστή με μηχανισμό Tomasulo και υποστήριξη εικασίας
  - Καθυστερήσεις πράξεων κινητής υποδιαστολής:
    - Πρόσθεση 2 κύκλοι ρολογιού, πολλαπλασιασμός 6 κύκλοι, και διαίρεση 12 κύκλοι
- Υποθέστε ότι έχουν εκκινήσει όλες οι εντολές δύο επαναλήψεων του βρόχου και μόνο οι εντολές L.D και MUL.D της πρώτης επανάληψης έχουν υποβληθεί
  - τι συμβαίνει με τις υπόλοιπες εντολές;

```
Loop: L.D      F0,0(R1)
      MUL.D    F4,F0,F2
      S.D      F4,0(R1)
      DADDIU   R1,R1,#-8
      BNE      R1,R2,Loop ;branches if R1≠R2
```

Tomasulo με εικασία

Instructions	Issues at clock number	Executes (start-to-end)	Memory access at clock number	Writes CDB at clock number	Commits at clock number
`L.D F0,0(R1)`	1	2	3	4	5
`MUL.D F4,F0,F2`	2	5-10		11	12
`S.D F4,0(R1)`	3	4	12		
`DADDIU R1,R1,#-8`	4	5		6	
`BNE R1,R2,Loop`	5	7			
`L.D F0,0(R1)`	6	7	8	9	
`MUL.D F4,F0,F2`	7	10-			
`S.D F4,0(R1)`	8	9			
`DADDIU R1,R1,#-8`	9	10		11	

Reorder buffer

Entry	Busy	Instruction	State	Destination	Value
1	no	`L.D F0,0(R1)`	Commit	F0	Mem[0 + Regs[R1]]
2	no	`MUL.D F4,F0,F2`	Commit	F4	#1 × Regs[F2]
3	yes	`S.D F4,0(R1)`	Write result	0 + Regs[R1]	#2
4	yes	`DADDIU R1,R1,#-8`	Write result	R1	Regs[R1] – 8
5	yes	`BNE R1,R2,Loop`	Write result		
6	yes	`L.D F0,0(R1)`	Write result	F0	Mem[#4]
7	yes	`MUL.D F4,F0,F2`	Write result	F4	#6 × Regs[F2]
8	yes	`S.D F4,0(R1)`	Write result	0 + #4	#7
9	yes	`DADDIU R1,R1,#-8`	Write result	R1	#4 – 8
10	yes	`BNE R1,R2,Loop`	Write result		

FP register status

Field	F0	F1	F2	F3	F4	F5	F6	F7	F8
Reorder #	6				7				
Busy	yes	no	no	no	yes	no	no	...	no

- Τι θα συμβεί εάν η διακλάδωση της πρώτης επανάληψης δεν ληφθεί?

Πολλαπλή εκκίνηση

- Για να επιτύχουμε CPI < 1, πρέπει να ολοκληρώνονται (ή να ξεκινούν) πολλαπλές εντολές ανά κύκλο ρολογιού
- Λύσεις:
  - Στατικά χρονοπρογραμματιζόμενοι υπερβαθμωτοί επεξεργαστές (Statically scheduled superscalar processors)
  - Επεξεργαστές VLIW (very long instruction word)
  - Δυναμικά χρονοπρογραμματιζόμενοι υπερβαθμωτοί επεξεργαστές (dynamically scheduled superscalar processors)

Πολλαπλή εκκίνηση

- Οι υπερβαθμωτοί επεξεργαστές ξεκινούν πολλαπλές εντολές (όχι σταθερό αριθμό) ανά κύκλο ρολογιού. Δύο τύποι:
  - Χρησιμοποιούν εκτέλεση σε σειρά εάν χρονοπρογραμματίζονται στατικά ή
  - Εκτέλεση εκτός σειράς εάν χρονοπρογραμματίζονται δυναμικά
- Οι επεξεργαστές VLIW ξεκινούν σταθερό αριθμό εντολών που σχηματίζουν είτε μια μεγάλη εντολή είτε ένα σταθερό πακέτο εντολών όπου ο παραλληλισμός μεταξύ των εντολών υποδηλώνεται ρητά
  - Intel/HP IA-64: EPIC (explicitly parallel instruction computer)
- Οι στατικά χρονοπρογραμματιζόμενοι επεξεργαστές και οι VLIW βασίζονται στο μεταγλωττιστή

Επεξεργαστές πολλαπλής εκκίνησης

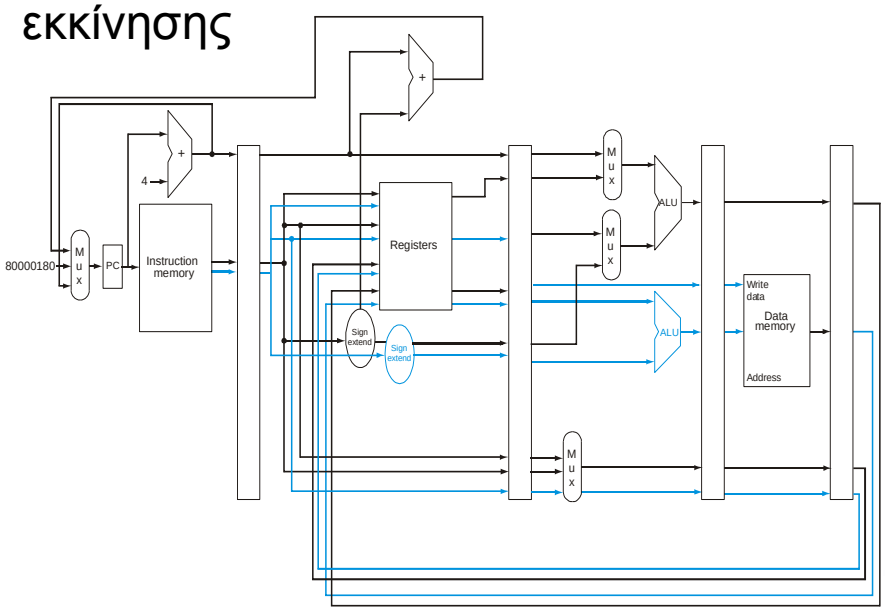
Common name	Issue structure	Hazard detection	Scheduling	Distinguishing characteristic	Examples
Superscalar (static)	Dynamic	Hardware	Static	In-order execution	Mostly in the embedded space: MIPS and ARM, including the ARM Coretex A8
Superscalar (dynamic)	Dynamic	Hardware	Dynamic	Some out-of-order execution, but no speculation	None at the present
Superscalar (speculative)	Dynamic	Hardware	Dynamic with speculation	Out-of-order execution with speculation	Intel Core i3, i5, i7; AMD Phenom; IBM Power 7
VLIW/LIW	Static	Primarily software	Static	All hazards determined and indicated by compiler (often implicitly)	Most examples are in signal processing, such as the TI C6x
EPIC	Primarily static	Primarily software	Mostly static	All hazards determined and indicated explicitly by the compiler	Itanium

Διοχέτευση στατικής διπλής εκκίνησης

- Εντολή 1: Εντολή ALU ή διακλάδωση
- Εντολή 2: Εντολή φόρτωσης ή αποθήκευσης
- Προσκόμιση και αποκωδικοποίηση εντολών 64-bit

Τύπος εντολής	Στάδια διοχέτευσης							
Εντολή ALU ή διακλάδωσης	IF	ID	EX	MEM	WB			
Εντολή φόρτωσης ή αποθήκευσης	IF	ID	EX	MEM	WB			
Εντολή ALU ή διακλάδωσης		IF	ID	EX	MEM	WB		
Εντολή φόρτωσης ή αποθήκευσης		IF	ID	EX	MEM	WB		
Εντολή ALU ή διακλάδωσης			IF	ID	EX	MEM	WB	
Εντολή φόρτωσης ή αποθήκευσης			IF	ID	EX	MEM	WB	
Εντολή ALU ή διακλάδωσης				IF	ID	EX	MEM	WB
Εντολή φόρτωσης ή αποθήκευσης				IF	ID	EX	MEM	WB

Στατική διαδρομή δεδομένων διπλής εκκίνησης



Χρονοπρογραμματισμός κώδικα πολλαπλής εκκίνησης

```
Loop:   lw   $t0,0($s1)
        addu $t0,$t0,$s2
        sw   $t0,0($s1)
        addi $s1,$s1,-4
        bne  $s1,$zero,Loop
```

- Για να επιτύχουμε αύξηση του ILP εφαρμόζουμε loop unrolling & code scheduling
- Υποθέστε ότι οι κίνδυνοι ελέγχου διαχειρίζονται από το υλικό

Χρονοπρογραμματισμός κώδικα πολλαπλής εκκίνησης

```
Loop:   lw   $t0,0($s1)
        addi $s1,$s1,-4
        addu $t0,$t0,$s2
        sw   $t0,4($s1)
        bne  $s1,$zero,Loop
```

	Εντολή ALU ή διακλάδωσης	Εντολή μεταφοράς δεδομένων	Κύκλος ρολογιού
Loop:		lw $t0,0($s1)	1
	addi $s1,$s1,-4		2
	addu $t0,$t0,$s2		3
	bne $s1,$zero,Loop	sw $t0,4($s1)	4

Loop unrolling

```
Loop:  lw   $t0,0($s1)
       addu $t0,$t0,$s2
       sw   $t0,0($s1)
       addi $s1,$s1,-4
       bne $s1,$zero,Loop
```

```
Loop:  lw   $t0,0($s1)
       addu $t0,$t0,$s2
       sw   $t0,0($s1)
       addi $s1,$s1,-16
       lw   $t0,12($s1)
       addu $t0,$t0,$s2
       sw   $t0,12($s1)
       lw   $t0,8($s1)
       addu $t0,$t0,$s2
       sw   $t0,8($s1)
       lw   $t0,4($s1)
       addu $t0,$t0,$s2
       sw   $t0,4($s1)
       bne s1,$zero,Loop
```

Μετονομασία καταχωρητών

```
Loop:  lw   $t0,0($s1)        Loop:  lw   $t0,0($s1)
       addu $t0,$t0,$s2              addu $t0,$t0,$s2
       sw   $t0,0($s1)               sw   $t0,0($s1)
       addi $s1,$s1,-16              addi $s1,$s1,-16
       lw   $t0,12($s1)              lw   $t1,12($s1)
       addu $t0,$t0,$s2              addu $t1,$t1,$s2
       sw   $t0,12($s1)              sw   $t1,12($s1)
       lw   $t0,8($s1)               lw   $t2,8($s1)
       addu $t0,$t0,$s2              addu $t2,$t2,$s2
       sw   $t0,8($s1)               sw   $t2,8($s1)
       lw   $t0,4($s1)               lw   $t3,4($s1)
       addu $t0,$t0,$s2              addu $t3,$t3,$s2
       sw   $t0,4($s1)               sw   $t3,4($s1)
       bne  $s1,$zero,Loop           bne  $s1,$zero,Loop
```

Χρονοπρογραμματισμός κώδικα πολλαπλής εκκίνησης

	Εντολή ALU ή διακλάδωσης	Εντολή μεταφοράς δεδομένων	Κύκλος ρολογιού
Loop:	addi $s1,$s1,-16	lw $t0,0($s1)	1
		lw $t1,12($s1)	2
	addu $t0,$t0,$s2	lw $t2,8($s1)	3
	addu $t1,$t1,$s2	lw $t3,4($s1)	4
	addu $t2,$t2,$s2	sw $t0,16($s1)	5
	addu $t3,$t3,$s2	sw $t1,12($s1)	6
		sw $t2,8($s1)	7
	bne $s1,$zero,Loop	sw $t3,4($s1)	8

Επεξεργαστές VLIW

- Ομαδοποιούν πολλαπλές λειτουργίες σε μια εντολή
- Παράδειγμα VLIW:
  - Μια εντολή ακεραίων (ή διακλάδωση)
  - Δύο ανεξάρτητες λειτουργίες κινητής υποδιαστολής
  - Δύο ανεξάρτητες αναφορές μνήμης
  - Ένα σύνολο πεδίων για κάθε λειτουργία
    - Π.χ. 16-20 bit ανά μονάδα, μήκος εντολής 80-120 bit
- Πρέπει να υπάρχει αρκετή παραλληλία στον κώδικα ώστε να γεμίσουν οι διαθέσιμες υποδοχές εκκίνησης (issue slots)
  - Χρησιμοποιούνται loop unrolling & code scheduling

Παράδειγμα

- Πρόσθεση τιμής σε διάνυσμα:

```
for (i=1000; i>0; i=i-1)
   x[i] = x[i] + s;
```

- Χρονοπρογραμματισμός του βρόχου σε έναν επεξεργαστή VLIW που υποστηρίζει το πακέτο εντολών:
  - Δύο λειτουργίες FP
  - Δύο προσπελάσεις μνήμης
  - Μία εντολή ακεραίων ή διακλάδωσης
- Ξετύλιγμα του βρόχου ώστε να αποφύγουμε καθυστερήσεις (δλδ. κύκλους με κενές υποδοχές εκκίνησης)

Κώδικας MIPS

- Μετάφραση του βρόχου σε κώδικα MIPS:
  - Για απλότητα, υποθέστε ότι το τελευταίο στοιχείο που ενημερώνεται είναι στη διεύθυνση 8

```
Loop:L.D      F0,0(R1)      ;F0=vector element
     ADD.D    F4,F0,F2      ;add scalar from F2
     S.D      0(R1),F4      ;store result
     DADDUI   R1,R1,-8      ;decrement pointer 8B
     BNEZ     R1,Loop       ;branch R1!=zero
```


Θυμηθείτε το χρονοπρογραμματισμό του κώδικα

```
1  Loop:L.D    F0,0(R1)
2       L.D    F6,-8(R1)
3       L.D    F10,-16(R1)
4       L.D    F14,-24(R1)
5       ADD.D  F4,F0,F2
6       ADD.D  F8,F6,F2
7       ADD.D  F12,F10,F2
8       ADD.D  F16,F14,F2
9       S.D    0(R1),F4
10      S.D    -8(R1),F8
11      S.D    -16(R1),F12
12      DSUBUI R1,R1,#32
13      S.D    8(R1),F16       ; 8-32 = -24
14      BNEZ   R1,LOOP
```

- 14 κύκλοι ή 3.5 ανά επανάληψη

Εκτέλεση σε επεξεργαστή VLIW

Memory reference 1	Memory reference 2	FP operation 1	FP operation 2	Integer operation/branch
L.D F0,0(R1)	L.D F6,-8(R1)			
L.D F10,-16(R1)	L.D F14,-24(R1)			
L.D F18,-32(R1)	L.D F22,-40(R1)	ADD.D F4,F0,F2	ADD.D F8,F6,F2	
L.D F26,-48(R1)		ADD.D F12,F10,F2	ADD.D F16,F14,F2	
		ADD.D F20,F18,F2	ADD.D F24,F22,F2	
S.D F4,0(R1)	S.D F8,-8(R1)	ADD.D F28,F26,F2		
S.D F12,-16(R1)	S.D F16,-24(R1)			DADDUI R1,R1,#-56
S.D F20,24(R1)	S.D F24,16(R1)			
S.D F28,8(R1)				BNE R1,R2,Loop

- Ξετύλιγμα 7 φορές
- Εκτέλεση σε 9 κύκλους
  - 23 λειτουργίες σε 9 κύκλους, IPC=2.5
  - Αποδοτικότητα = 50% (ποσοστό υποδοχών που περιέχουν λειτουργία)
- Μεγάλος αριθμός καταχωρητών

Μειονεκτήματα

- Να βρούμε παραλληλία με στατική μέθοδο
  - Μετά το ξεδίπλωμα βρόχου, τοπικές τεχνικές χρονοπρογραμματισμού σε βασικές ενότητες
  - Σφαιρικές τεχνικές για χρονοπρογραμματισμό κώδικα σε πολλές επαναλήψεις του βρόχου
- Αύξηση μεγέθους κώδικα
  - Λόγω του «επιθετικού» ξετυλίγματος βρόχων
    - Χρήση τεχνικών λογισμικού (software pipelining) για να πετύχουμε ξεδίπλωμα χωρίς αύξηση του κώδικα
  - Λόγω των αχρησιμοποίητων μονάδων στα πακέτα υποδοχών που δεν είναι γεμάτα
    - Χρήση «έξυπνων» τεχνικών κωδικοποίησης για μείωση του μεγέθους κώδικα
    - Χρήση τεχνικών συμπίεσης κώδικα

Μειονεκτήματα

- Δεν περιέχει υλικό ανίχνευσης κινδύνων
  - Καθυστέρηση σε μια λειτουργική μονάδα οδηγεί στην καθυστέρηση ολόκληρου του επεξεργαστή
  - Ο μεταγλωττιστής μπορεί να προνοήσει για την καθυστέρηση των λειτουργικών μονάδων αλλά όχι για τις αστοχίες κρυφής μνήμης
    - Για να αντιμετωπίσει αυτά τα προβλήματα η αρχιτεκτονική EPIC χρησιμοποιεί και δυναμικές τεχνικές (π.χ. dynamic branch prediction, register renaming)
- Συμβατότητα δυαδικού κώδικα
  - Σε αυστηρούς VLIW επεξεργαστές
    - Διαφορετικός αριθμός λειτουργικών μονάδων και διαφορετικές καθυστερήσεις απαιτούν διαφορετικές εκδόσεις του κώδικα

Συνδυασμός όλων των τεχνικών

- Δυναμικός χρονοπρογραμματισμός + Πολλαπλή εκκίνηση + Εικασία
- Ας υποθέσουμε έναν επεξεργαστή διπλής εκκίνησης (2-issue processor)
  - Επιτρέπει την εκκίνηση οποιουδήποτε συνδυασμού δύο εντολών σε έναν κύκλο
  - Το υλικό για το χρονοπρογραμματισμό αναθέτει τις λειτουργίες στους σταθμούς κράτησης
  - Δύο προσεγγίσεις:
    - Ανάθεση εντολών σε μισούς κύκλους ρολογιού ⇒ 2 εντολές/κύκλο
    - Επιπλέον λογική για να χειριστεί τις πιθανές εξαρτήσεις μεταξύ των εντολών
  - Επιπλέον λογική για να υποστηρίξει την ταυτόχρονη ολοκλήρωση και υποβολή (commit) πολλαπλών εντολών

Παράδειγμα

```
Loop:LD R2,0(R1)     ;R2=array element
     DADDIU R2,R2,#1;increment R2
     SD R2,0(R1)     ;store result
     DADDIU R1,R1,#8;increment pointer
     BNE R2,R3,LOOP ;if not last element
```

- Εκτέλεση του κώδικα σε επεξεργαστή διπλής εκκίνησης: (α) χωρίς υποστήριξη εικασίας, (β) με εικασία
  - Ξεχωριστές μονάδες για υπολογισμό διεύθυνσης μνήμης, λειτουργίες ALU και επίλυση διακλάδωσης
  - Για τις 3 πρώτες επαναλήψεις του βρόχου

Χωρίς εικασία

Iteration number	Instructions	Issues at clock cycle number	Executes at clock cycle number	Memory access at clock cycle number	Write CDB at clock cycle number	Comment
1	LD R2,0(R1)	1	2	3	4	First issue
1	DADDIU R2,R2,#1	1	5		6	Wait for LW
1	SD R2,0(R1)	2	3	7		Wait for DADDIU
1	DADDIU R1,R1,#8	2	3		4	Execute directly
1	BNE R2,R3,LOOP	3	7			Wait for DADDIU
2	LD R2,0(R1)	4	8	9	10	Wait for BNE
2	DADDIU R2,R2,#1	4	11		12	Wait for LW
2	SD R2,0(R1)	5	9	13		Wait for DADDIU
2	DADDIU R1,R1,#8	5	8		9	Wait for BNE
2	BNE R2,R3,LOOP	6	13			Wait for DADDIU
3	LD R2,0(R1)	7	14	15	16	Wait for BNE
3	DADDIU R2,R2,#1	7	17		18	Wait for LW
3	SD R2,0(R1)	8	15	19		Wait for DADDIU
3	DADDIU R1,R1,#8	8	14		15	Wait for BNE
3	BNE R2,R3,LOOP	9	19			Wait for DADDIU

Με εικασία

Iteration number	Instructions		Issues at clock number	Executes at clock number	Read access at clock number	Write CDB at clock number	Commits at clock number	Comment
1	LD	R2,0(R1)	1	2	3	4	5	First issue
1	DADDIU	R2,R2,#1	1	5		6	7	Wait for LW
1	SD	R2,0(R1)	2	3			7	Wait for DADDIU
1	DADDIU	R1,R1,#8	2	3		4	8	Commit in order
1	BNE	R2,R3,LOOP	3	7			8	Wait for DADDIU
2	LD	R2,0(R1)	4	5	6	7	9	No execute delay
2	DADDIU	R2,R2,#1	4	8		9	10	Wait for LW
2	SD	R2,0(R1)	5	6			10	Wait for DADDIU
2	DADDIU	R1,R1,#8	5	6		7	11	Commit in order
2	BNE	R2,R3,LOOP	6	10			11	Wait for DADDIU
3	LD	R2,0(R1)	7	8	9	10	12	Earliest possible
3	DADDIU	R2,R2,#1	7	11		12	13	Wait for LW
3	SD	R2,0(R1)	8	9			13	Wait for DADDIU
3	DADDIU	R1,R1,#8	8	9		10	14	Executes earlier
3	BNE	R2,R3,LOOP	9	13			14	Wait for DADDIU