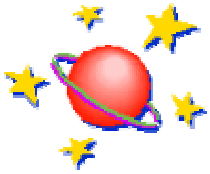

Ενότητα 1η: *Παραλληλία και τεχνικές*

Σκοπός **Ο σκοπός της ενότητας αυτής είναι να παρουσιάσει την έννοια της παραλληλίας σε επίπεδο εντολής και σε επίπεδο βρόχου. Στη συνέχεια παρουσιάζονται δύο τεχνικές, ο προγραμματισμός σωλήνωσης και η ανάπτυξη βρόχου, οι οποίες μας επιτρέπουν να αυξήσουμε το βαθμό παραλληλίας.**

Προσδοκώμενα Αποτελέσματα **Όταν θα έχετε μελετήσει την ενότητα, θα είστε σε θέση:**

<u>να περιγράφετε την παραλληλία σε επίπεδο εντολής και σε επίπεδο βρόχου,</u>

<u>να εφαρμόζετε τις τεχνικές του προγραμματισμού σωλήνωσης και της ανάπτυξης βρόχου.</u>

παραλληλία σε επίπεδο εντολής, παραλληλία σε επίπεδο βρόχου, προγραμματισμός σωλήνωσης, ανάπτυξη βρόχου

Παραλληλία σε επίπεδο εντολής και βρόχου

Στη σωλήνωση μπορεί να υπάρξει επικάλυψη στην εκτέλεση εντολών, όταν αυτές είναι ανεξάρτητες μεταξύ τους και μπορούν να εκτελεστούν παράλληλα. Η επικάλυψη αυτών των εντολών ονομάζεται **παραλληλία σε επίπεδο εντολής** (ILP).
Για να αυξήσουμε το βαθμό της παραλληλίας που είναι διαθέσιμο μεταξύ των εντολών, αξιοποιούμε την παραλληλία μεταξύ των επαναλήψεων ενός βρόχου. Αυτή η μορφή παραλληλίας, συχνά καλείται **παραλληλία σε επίπεδο βρόχου**.

Παράδειγμα

Παράδειγμα βρόχου, στο οποίο προστίθενται δύο πίνακες 1000 στοιχείων, σε παραλληλία :

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

Κάθε επανάληψη του βρόχου μπορεί να επικαλυφθεί με οποιαδήποτε άλλη επανάληψη.

Το λογισμικό είναι υπεύθυνο για τον προγραμματισμό των εντολών, με σκοπό εξαρτώμενες εντολές να γίνουν ανεξάρτητες και να ελαχιστοποιηθεί το πλήθος των πραγματικών κινδύνων και των καθυστερήσεων που προκαλούνται στη σωλήνωση.

Δύο είναι οι **τεχνικές μετασχηματισμού** παραλληλίας επιπέδου βρόχου, σε παραλληλία επιπέδου εντολής, οι οποίες ανήκουν επίσης στη περίπτωση του στατικού προγραμματισμού**:**

- Προγραμματισμός σωλήνωσης.
- Ανάπτυξη βρόχου.

ΔΡΑΣΤΗΡΙΟΤΗΤΑ 1

- **Θυμάστε τι είναι ο προγραμματισμός σωλήνωσης και τι πετυχαίνουμε με αυτή την τεχνική;**
- **Να εφαρμόσετε την τεχνική του προγραμματισμού σωλήνωσης, δίνοντας ένα παράδειγμα.**
- **Για περισσότερες λεπτομέρειες καλό θα ήταν να ανατρέξετε στο 1^{ο} κεφάλαιο και συγκεκριμένα στην 5^{η} ενότητα: «Κίνδυνοι Δεδομένων που απαιτούν καθυστερήσεις».**

ΑΠΑΝΤΗΣΗ ΔΡΑΣΤΗΡΙΟΤΗΤΑΣ 1

Ο μεταγλωττιστής σε έναν σωληνωμένο υπολογιστή προσπαθεί να προγραμματίσει τη σωλήνωση, έτσι ώστε να αποφύγει τις καθυστερήσεις, αλλάζοντας τη σειρά εκτέλεσης των εντολών για να εξαλείψει των κίνδυνο δεδομένων. Η τεχνική αυτή ονομάζεται προγραμματισμός σωλήνωσης.

<u>Παράδειγμα</u>:

Έστω η παρακάτω ακολουθία:

a = b * c;

d = e + f ;

όπου a, b, c, d, e, f είναι διευθύνσεις μνήμης. Υποθέστε ότι οι φορτώσεις έχουν αδράνεια ενός κύκλου ρολογιού.

Ο κώδικας για τη δεδομένη ακολουθία εντολών , ώστε να αποφεύγονται οι καθυστερήσεις είναι:

```
LW   Rb, b
LW   Rc, c
LW   Re, e        ; ανταλλαγή της θέσης των εντολών για αποφυγή της καθυστέρησης
MUL Ra, Rb, Rc
LW   Rf, f
SW a, Ra          ; ανταλλαγή αποθήκευσης /φόρτωσης αποφυγή καθυστερήσεων
SUB Rb, Re, Rf
SW   d, Rd
```

Παρατηρούμε ότι η χρήση διαφορετικών καταχωρητών για την πρώτη και δεύτερη δήλωση είναι απαραίτητη, για να είναι σωστός ο προγραμματισμός. Αν η μεταβλητή e φορτωνόταν στον ίδιο καταχωρητή με τη μεταβλητή b ή τη μεταβλητή c, ο προγραμματισμός δε θα ήταν σωστός.

Προγραμματισμός σωλήνωσης και ανάπτυξη βρόχου.

Για να διατηρήσουμε την σωλήνωση πλήρη βρίσκουμε ακολουθίες εντολών που να μη σχετίζονται μεταξύ τους, έτσι ώστε να μπορούν να επικαλυφθούν. Για να αποφύγουμε καθυστερήσεις στην σωλήνωση, διαχωρίζουμε κάθε εξαρτώμενη εντολή από την πηγαία εντολή, σε απόσταση, ίση με την καθυστέρηση της πηγαίας εντολής στην σωλήνωση, μετρημένη σε κύκλους ρολογιού.

Η δυνατότητα του μεταγλωττιστή να εκτελέσει τον προγραμματισμό σωλήνωσης εξαρτάται από:

- Το διαθέσιμο βαθμό της παραλληλίας σε επίπεδο προγράμματος.
- Την καθυστέρηση των λειτουργικών μονάδων της σωλήνωσης.

Υποθέτουμε ότι διαθέτουμε **:**

- Ακέραια σωλήνωση με καθυστέρηση διακλάδωσης έναν κύκλο ρολογιού.
- Οι λειτουργικές μονάδες είναι πλήρως σωληνωμένες ή επαναλαμβάνονται όσο το βάθος της σωλήνωσης, ώστε να μην υπάρχουν κατασκευαστικοί κίνδυνοι.

Στο παρακάτω σχήμα δίνονται οι καθυστερήσεις για τις εντολές κινητής υποδιαστολής (λειτουργίες FP) **:**

Εντολή που δίνει το αποτέλεσμα (πηγαία εντολή)	**Εντολή που χρησιμοποιεί το αποτέλεσμα**	**Καθυστέρηση σε κύκλους ρολογιού**
FP ALU op	Άλλη FP ALU op	3
FP ALU op	Αποθήκευση double	2
Φόρτωση double	FP ALU op	1
Φόρτωση double	Αποθήκευση double	0

Σχήμα 2.1.1 - Στην πρώτη στήλη φαίνεται ο τύπος της πηγαίας εντολής, στην δεύτερη στήλη ο τύπος της εντολής που εξαρτάται από το αποτέλεσμα, ενώ στην τρίτη φαίνεται η καθυστέρηση σε κύκλους ρολογιού, που είναι αναγκαία για την αποφυγή της κατάστασης κινδύνου.

Παράδειγμα:

Παράδειγμα απλού βρόχου, που προσθέτει μία βαθμωτή τιμή σε έναν πίνακα, στην μνήμη.

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

Απάντηση:

Παρατηρούμε ότι το κύριο μέρος κάθε επανάληψης είναι ανεξάρτητο και άρα ο βρόχος είναι παράλληλος. Τώρα θα εξετάσουμε πώς μπορούμε να χρησιμοποιήσουμε αυτήν την ιδιότητα, για να βελτιώσουμε την παραλληλία στον DLX, χρησιμοποιώντας τις καθυστερήσεις που αναφέραμε στο σχήμα 2.1.1.
Μεταφράζουμε το βρόχο σε γλώσσα Assembly. Στο τμήμα κώδικα που ακολουθεί, το οποίο δεν είναι ειδικά προγραμματισμένο για σωλήνωση, ο R1 περιέχει αρχικά τη διεύθυνση του στοιχείου στον πίνακα με την μεγαλύτερη διεύθυνση και ο F2 περιέχει τη βαθμωτή τιμή s. Υποθέτουμε ότι η χαμηλότερη διεύθυνση είναι στο 0, γιατί διαφορετικά, ο βρόχος θα απαιτούσε μία παραπάνω εντολή που θα εκτελούσε την σύγκριση με τον R1. 'Εχουμε, λοιπόν **:**

```
loop: LD      F0, 0 (R1)  ;  F0 = στοιχείο πίνακα
      ADDD    F4, F0, F2  ;  πρόσθεση με το βαθμωτό μέγεθος του F2
      SD      0 (R1), F4  ;  αποθήκευση του αποτελέσματος
      SUBI    R1, R1, #8  ;  μείωση δείκτη 8 bytes (για κάθε DW)
      BNEZ    R1, loop    ;  διακλάδωση R1!= μηδέν
```

ΔΡΑΣΤΗΡΙΟΤΗΤΑ 2

Μπορείτε να βρείτε πόσοι κύκλοι ρολογιού χρειάζονται για την εκτέλεση του βρόχου που βρίσκεται στο παραπάνω παράδειγμα, χωρίς τη χρήση προγραμματισμού σωλήνωσης; Να αιτιολογήσετε την απάντησή σας και να τη συγκρίνετε με την απάντηση που δίνεται στο επόμενο παράδειγμα.

Στο παράδειγμα που ακολουθεί εξετάζουμε πόσο γρήγορα θα εκτελεστεί ο παραπάνω βρόχος σε απλή σωλήνωση του DLX, λαμβάνοντας υπόψη τις καθυστερήσεις που αναφέρθηκαν στο σχήμα 2.1.1.

Παράδειγμα:

Να δείξετε, πώς ο παραπάνω βρόχος θα ήταν για τον DLX, χωρίς προγραμματισμό και με προγραμματισμό, περικλείοντας τις καθυστερήσεις που προκύπτουν ή τους αδρανείς κύκλους ρολογιού. Ο προγραμματισμός να γίνει και για τις καθυστερήσεις που προκύπτουν από εφαρμογές κινητής υποδιαστολής αλλά και από καθυστερημένες διακλαδώσεις.

Απάντηση:

Χωρίς προγραμματισμό, ο βρόχος θα εκτελεστεί ως εξής **:**

			Κύκλοι ρολογιού
loop**:**	LD	F0, 0 (R1)	1
	stall		2
	ADDD	F4, F0, F2	3
	stall		4
	stall		5
	SD	0 (R1), F4	6
	SUBI	R1, R1, #8	7
	stall		8
	BNEZ	R1,loop	9
	stall		10

Παρατηρούμε ότι χρειάζονται 10 κύκλοι ρολογιού, ανά επανάληψη**:**

- Μία καθυστέρηση για την εντολή LD.
- Δύο για την εντολή ADDD.
- Μία για την εντολή SUBI (μια και η διακλάδωση διαβάζει τον τελεστέο στο στάδιο ID).
- Μία για την εντολή διακλάδωσης BNEZ.

Με προγραμματισμό έχουμε **:**

			Κύκλοι ρολογιού
loop**:**	**LD**	**F0, 0 (R1)**	**1**
	SUBI	**R1, R1, #8**	**2**
	ADDD	**F4, F0, F2**	**3**
	stall		**4**
	BNEZ	**R1, loop**	**5**
	SD	**8 (R1), F4**	**6**

Ο χρόνος εκτέλεσης μειώθηκε από 10 σε 6 κύκλους ρολογιού. Η καθυστέρηση μετά την εντολή ADDD χρειάζεται λόγω της χρήσης του F4 από την εντολή SD.

Για τον προγραμματισμό της καθυστερημένης διακλάδωσης, ο μεταγλωττιστής καθορίζει την ανταλλαγή των εντολών SUBI και SD, αλλάζοντας και τη διεύθυνση που η SD αποθηκεύει το αποτέλεσμα της από 0 (R1) σε 8 (R1). Οι περισσότεροι μεταγλωττιστές, γνωρίζουν ότι η εντολή SD εξαρτάται από την εντολή SUBI και δεν υλοποιούν την ανταλλαγή αυτή. Η αλυσίδα των εξαρτημένων εντολών, από την εντολή LD στην ADDD και μετά στην SD, καθορίζει το πλήθος κύκλων ρολογιού του βρόχου.

Σε κάθε επανάληψη αποθηκεύουμε ένα στοιχείο του πίνακα, κάθε 6 κύκλους ρολογιού, αλλά ο πραγματικός χρόνος είναι 3 μόνο κύκλοι από τους 6, για LD, ADD, SD. Οι υπόλοιποι τρεις, διατίθενται για τις εντολές BNEZ, SUBI και stall. Για να εξαλείψουμε αυτούς τους τρεις τελευταίους κύκλους ρολογιού, θα χρειαστεί να υλοποιήσουμε περισσότερες εντολές μέσα στο κύριο μέρος του βρόχου ως προς την εντολή διακλάδωσης (BNEZ R1,loop) και την εντολή επιβάρυνσης (SUBI R1,R1,#8).

Ανάπτυξη βρόχου είναι ένα απλό σχήμα για την αύξηση του αριθμού των εντολών που σχετίζονται με την εντολή διακλάδωσης και την εντολή επιβάρυνσης.

Επιτυγχάνεται με την επανάληψη του κυρίως μέρους του βρόχου και με προσαρμογή των εντολών που τελειώνουν το βρόχο (στο παράδειγμα BNEZ, SUBI). Η ανάπτυξη του βρόχου βελτιώνει επίσης τον προγραμματισμό, περιορίζοντας την εντολή διακλάδωσης και ομαδοποιώντας εντολές από διαφορετικές επαναλήψεις.

ΔΡΑΣΤΗΡΙΟΤΗΤΑ 3

- **Μπορείτε να εξηγήσετε πώς επηρεάζει η δημιουργία εντολών που δεν προκαλούν κινδύνους δεδομένων, την καθυστέρηση της σωλήνωσης;**
- **Θεωρείτε πως απαιτείται ο ίδιος αριθμός καταχωρητών για τον προγραμματισμό με ανάπτυξη βρόχου;**

ΑΠΑΝΤΗΣΗ ΔΡΑΣΤΗΡΙΟΤΗΤΑΣ 3

- **Η δημιουργία επιπλέον εντολών, που δεν έχουν κινδύνους δεδομένων, μειώνει τη καθυστέρηση των εντολών φόρτωσης. Αυτό συμβαίνει επειδή οι εντολές που χρησιμοποιούν το αποτέλεσμα των εντολών φόρτωσης παραλείπεται στην επανάληψη των εντολών του βρόχου.**
- **Για ικανοποιητικό προγραμματισμό του βρόχου απαιτούνται περισσότεροι καταχωρητές, έτσι ώστε να αποθηκεύονται τα αποτελέσματα των εντολών του βρόχου, οι οποίες επαναλαμβάνονται.**

Παράδειγμα

Αναπτύξτε τον παραπάνω βρόχο, χωρίς προγραμματισμό και με προγραμματισμό, ώστε να υπάρχουν τέσσερα αντίγραφα του σώματος του βρόχου (LD, ADDD, SD) υποθέτοντας ότι ο R1 είναι αρχικά πολλαπλάσιο του 32, δηλαδή ότι ο αριθμός των επαναλήψεων του βρόχου είναι πολλαπλάσιο του 4. Μειώστε τυχόν περιττούς υπολογισμούς και χρησιμοποιήστε κάθε καταχωρητή μία φορά.

Απάντηση:

Με ανάπτυξη του βρόχου και χωρίς προγραμματισμό, έχουμε:

```
loop: LD      F0 , 0 (R1)
      stall
      ADDD    F4 , F0 , F2
      stall
      stall
      SD      0 (R1) , F4     ; παράλειψη των SUBI και BNEZ

      LD      F6 , -8 (R1)
      stall
      ADDD    F8 , F6 , F2
      stall
      stall
      SD      -8 (R1) , F8    ; παράλειψη των SUBI και BNEZ

      LD      F10 , -16 (R1)
      stall
      ADDD    F12 , F10 , F2
      stall
```

```
stall

SD      -16 (R1) , F12 ; παράλειψη των SUBI και BNEZ

LD      F14 , -24 (R1)
stall
ADDD    F16 , F14 , F2
stall
stall
SD      -24 (R1),  F16
SUBI    R1 , R1 , #32
stall
BNEZ    R1 , loop
stall
```

Συγχωνεύουμε τις SUBI εντολές και παραλείπουμε τις μη αναγκαίες BNEZ εντολές. Έχουμε εξαλείψει τρεις εντολές διακλάδωσης (BNEZ) και τρεις εντολές ελάττωσης (SUBI) του R1. Οι διευθύνσεις των εντολών φόρτωσης (loads) και αποθήκευσης (stores) μας επέτρεψαν να συγχωνεύσουμε τις SUBI εντολές με κατάλληλη τιμή στον R1. Χωρίς προγραμματισμό, κάθε εντολή ακολουθείται από μία εξαρτώμενη εντολή, με αποτέλεσμα τη δημιουργία καθυστέρησης. Αυτός ο βρόχος, θα εκτελεστεί σε 28 κύκλους ρολογιού αφού**:**

- Κάθε εντολή LD έχει καθυστέρηση κατά ένα κύκλο ρολογιού.
- Κάθε εντολή ADDD έχει καθυστέρηση κατά δύο κύκλους ρολογιού.
- Η εντολή SUBI έχει καθυστέρηση κατά ένα κύκλο ρολογιού.
- Η εντολή διακλάδωσης BNEZ έχει καθυστέρηση κατά ένα κύκλο ρολογιού.

Επιπλέον, χρειάζονται 14 κύκλοι ρολογιού για την εκτέλεση των εντολών. Συνεπώς χρειάζονται 7 κύκλοι ρολογιού για κάθε ένα από τα τέσσερα αντίγραφα του βρόχου. Αυτή η ανάπτυξη βρόχου είναι πιο αργή από τον αρχικό βρόχο με προγραμματισμό. Εάν προγραμματίσουμε τον ανεπτυγμένο βρόχο, τότε θα έχουμε βελτίωση της επίδοσης.

Με ανάπτυξη του βρόχου και προγραμματισμό, έχουμε:

```
loop: LD      F0 , 0(R1)
      LD      F6 , -8 (R1)
      LD      F10 , -16 (R1)
      LD      F14 , -24 (R1)

      ADDD    F4 , F0 , F2
      ADDD    F8 , F6 , F2
      ADDD    F12 , F10 , F2
      ADDD    F16 , F14 , F2

      SD      0 (R1) , F4
      SD      -8(R1), F8
      SUBI    R1 , R1 , #32
      SD      -16(R1) , F12
      BNEZ    R1 , loop
      SD      8(R1), F16 ; 8-32 = -24
```

Συγκεντρωτικά έχουμε **:**

	Κύκλοι ρολογιού
Χωρίς προγραμματισμό και χωρίς ανάπτυξη βρόχου	10
Με προγραμματισμό και χωρίς ανάπτυξη βρόχου	6
Με ανάπτυξη βρόχου και χωρίς προγραμματισμό	28 ή 7 ανά αντίγραφο
Με ανάπτυξη βρόχου και με προγραμματισμό	14 ή 3.5 ανά αντίγραφο

ΔΡΑΣΤΗΡΙΟΤΗΤΑ 4

Να επιλέξετε την καλύτερη τεχνική για την αποφυγή κινδύνων στη σωλήνωση, από αυτές που παρουσιάστηκαν στην ενότητα αυτή. Να αιτιολογήσετε την απάντησή σας.

ΑΠΑΝΤΗΣΗ ΔΡΑΣΤΗΡΙΟΤΗΤΑΣ 4

Η τεχνική της ανάπτυξης βρόχου με προγραμματισμό είναι καλύτερη από τις άλλες τεχνικές, γιατί με την ανάπτυξη έχουμε δημιουργία πολλών εντολών, τις οποίες μπορούμε να προγραμματίσουμε, ώστε να μειωθούν οι καθυστερήσεις που οφείλονται σε κινδύνους δεδομένων.

ΔΡΑΣΤΗΡΙΟΤΗΤΑ 5

Έστω το παρακάτω τμήμα κώδικα:

```
main:  lw r3,0(r7)
       lw r4,0(r7)
       add r1,r3,r4
       lw r5,0(r7)
       lw r6,0(r7)
       add r2,r5,r6
```

- **Μπορείτε να εντοπίσετε σε ποιες εντολές του προγράμματος προκύπτουν κίνδυνοι δεδομένων;**
- **Να τροποποιήσετε κατάλληλα τον κώδικα έτσι ώστε να μειωθούν οι καθυστερήσεις κατά την εκτέλεση του προγράμματος.**
- **Να εκτελέσετε των κώδικα του παραδείγματος χρησιμοποιώντας το λογισμικό προσομοίωσης του DLX, Windlx, το οποίο παρατίθεται στις ιστοσελίδες:**
  http://www.gup.uni-linz.ac.at/downloads/ti3/windlx

  http://www.mkp.com/pub/dlx
- **Να συγκρίνετε την απάντησή σας με τα αποτελέσματα από την εκτέλεση του προγράμματος από τον προσομοιωτή Wndlx.**
- **Για περισσότερες λεπτομέρειες καλό θα ήταν να συμβουλευόσασταν το παράρτημα Α: «Εργαλεία Προσομοίωσης του DLX».**

ΑΠΑΝΤΗΣΗ ΔΡΑΣΤΗΡΙΟΤΗΤΑΣ 5

- **Παρατηρούμε πως προκύπτουν κίνδυνοι δεδομένων στις δύο εντολές add, οι οποίες χρειάζονται τις τιμές των καταχωρητών πριν την ολοκλήρωση των εντολών lw.**
- **Για να αποφευχθούν οι κίνδυνοι δεδομένων, τροποποιούμε τον κώδικα ως εξής:**

```
main:  lw r3,0(r7)
       lw r4,0(r7)
       lw r5,0(r7)
       lw r6,0(r7)
       add r1,r3,r4
       add r2,r5,r6
```

Ανακεφαλαιώνοντας λοιπόν…

Στη σωλήνωση μπορεί να υπάρξει επικάλυψη στην εκτέλεση εντολών όταν αυτές είναι ανεξάρτητες. Αυτή η επικάλυψη, ονομάζεται παραλληλία σε επίπεδο εντολής.

Η δυνατότητα παραλληλίας μεταξύ των επαναλήψεων ενός βρόχου, ονομάζεται παραλληλία σε επίπεδο βρόχου.

Δύο τεχνικές που μας επιτρέπουν να αυξήσουμε το διαθέσιμο βαθμό παραλληλίας είναι ο Προγραμματισμός Σωλήνωσης και η Ανάπτυξη Βρόχου.

Στον προγραμματισμό σωλήνωσης για να αποφύγουμε καθυστερήσεις στην σωλήνωση, διαχωρίζουμε κάθε εξαρτώμενη εντολή από την πηγαία εντολή, σε απόσταση, ίση με την καθυστέρηση της πηγαίας εντολής στην σωλήνωση, μετρημένη σε κύκλους ρολογιού.

Η ανάπτυξη του βρόχου επιτυγχάνεται με την επανάληψη του κυρίως μέρους