ΠΑΡΑΛΛΗΛΗ ΕΠΕΞΕΡΓΑΣΙΑ

ΑΓΩΓΟΙ & ΔΙΑΝΥΣΜΑΤΙΚΟΙ ΥΠΟΛΟΓΙΣΤΕΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΕΡΓΑΣΤΗΡΙΟ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ ΥΨΗΛΩΝ ΕΠΙΔΟΣΕΩΝ

Αγωγοί Υπολογιστές

Βασικές Έννοιες (pipelining)

διανυσματικές εντολές - διάσπαση και εκτέλεση τους σε αγωγό

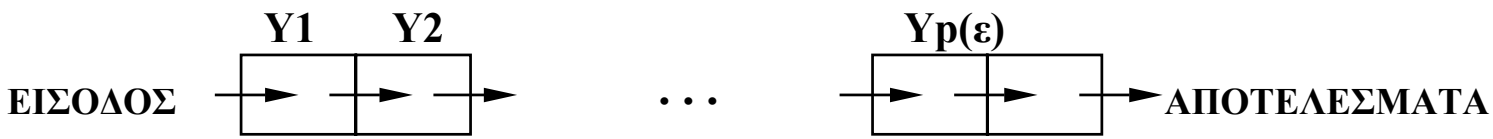
Μία διανυσματική εντολή ε μπορεί να διασπαστεί σε $p(\varepsilon)$ στοιχειώδεις υποεντολές, οι οποίες μπορούν με τη σειρά τους να εκτελεστούν σε αγωγό αποτελούμενο από $p(\varepsilon)$ επεξεργαστές αντίστοιχα. Υποθέτουμε ότι η κάθε μια από τις $p(\varepsilon)$ υποεντολές χρειάζεται χρόνο t^0 για να εκτελεστεί. Αν μία από τις $p(\varepsilon)$ εντολές χρειάζονται χρόνο $t^{0\prime} > t^0$ τότε, θέτουμε σαν ελάχιστο χρόνο του αγωγού, το χρόνο που χρειάζεται για να εκτελεστεί η πιο χρονοβόρα εντολή, δηλαδή $t^0 = t^{0\prime}$. Αν υποθέσουμε ότι έχουμε μία τέτοια διανυσματική εντολή μήκους n τότε για να εκτελεστούν ακολουθιακά οι $p(\varepsilon)$ υποεντολές της, χρειάζεται χρόνος t_{seq} που δίνεται από την σχέση:

$$t_{seq}(\varepsilon,n) = n*p(\varepsilon)*t^0$$

Βασικές Έννοιες (pipelining)

Μέθοδος Αγωγού (Pipelining)

Στους υπολογιστές με αγωγούς, υπάρχουν *p(ε)* διασυνδεδεμένα επεξεργαστικά στοιχεία κάθε ένα από τα οποία είναι προγραμματισμένο να εκτελεί μία συγκεκριμένη διαδικασία. Όλα μαζί στη σειρά, σχηματίζουν ένα νοητό αγωγό δημιουργώντας έτσι μία συνεχόμενη ή με συνθήκες ροή στοιχείων μέσα στον αγωγό. Το κάθε *p(ε),* περνά το αποτέλεσμα του στο επόμενο μέχρι να ολοκληρωθεί η εκτέλεση όλων των υποεντολών, όπως φαίνεται και στο ακόλουθο σχήμα.



Βασικές Έννοιες (pipelining)

Παράδειγμα: Στην παρακάτω εικόνα φαίνεται το διάγραμμα χώρου-χρόνου για εκτέλεση μίας διανυσματικής εντολής μήκους $n=6$ σε αγωγό μήκους $p(\varepsilon)=5$.

Εξαγωγή Αποτελέσματος					1	2	3	4	5	6
Y5					**1**	**2**	**3**	**4**	**5**	**6**
Y4				**1**	**2**	**3**	**4**	**5**	**6**	
Y3			**1**	**2**	**3**	**4**	**5**	**6**		
Y2		**1**	**2**	**3**	**4**	**5**	**6**			
Y1	**1**	**2**	**3**	**4**	**5**	**6**				
	0	1	2	3	4	5	6	7	8	9

10
χρόνος

Ο αγωγός που αντιστοιχεί στο παραπάνω διάγραμμα γεμίζει μετά από χρόνο $4t^0 = (p(\varepsilon)-1)t^0$, το πρώτο αποτέλεσμα παράγεται τη χρονική στιγμή $5t^0 = p(\varepsilon)t^0$, ενώ το τελευταίο αποτέλεσμα θα παραχθεί τη χρονική στιγμή, "$p(\varepsilon)t^0$(για το πρώτο) + (n-1)t^0(για τα υπόλοιπα n-1) = $(p(\varepsilon)-1+n)t^0$. Πρέπει όμως να συνυπολογίσουμε και το χρόνο αρχικοποίησης αt^0 ή $a(\varepsilon)t^0$. Έτσι ο συνολικός χρόνος είναι: $t_{pipe}(\varepsilon,n) = (\alpha(\varepsilon)+ p(\varepsilon)+n-1)t^0$.

Βασικές Έννοιες (pipelining)

Διάνυσμα Επεξεργαστών

Έστω ότι χρησιμοποιούνται K επεξεργαστές (ή K αθροιστές - αν έχουμε διανυσματική πρόσθεση), οι οποίοι μπορούν να υπολογίζουν ταυτόχρονα και ανεξάρτητα. Δηλαδή μπορούν να γίνονται K ταυτόχρονες εκτελέσεις. Τότε ο συνολικός χρόνος εκτέλεσης είναι ίσος με το χρόνο εκτέλεσης της μίας εντολής:

$$t_{διαν}(\varepsilon,n) = \lceil n/K \rceil * p(\varepsilon)t^0$$

Υπάρχουν όμως και χρονικές επιβαρύνεις με τις οποίες θα ασχοληθούμε στα επόμενα.

Βασικές Έννοιες (pipelining)

Από αυτά που μέχρι τώρα είδαμε για αγωγούς με χρήση διανυσματικών εντολών μπορούμε να πούμε ότι η ιδέα των αγωγών είναι η χρονική επικάλυψη της εκτέλεσης με ή και χωρίς υποδιαίρεση μίας μεγαλύτερης διαδικασίας.

Συμβολισμοί: Έστω ότι συμβολίζουμε με T_i $i=1,\ldots,k$ τα επεξεργαστικά στοιχεία και τα υποέργα ενός έργου T του αγωγού. Επιπλέον υποθέτουμε ότι έχει γίνει και τοπολογική διάταξη, δηλαδή το T_j δεν μπορεί να αρχίσει την εκτέλεση του αν πρώτα δεν ολοκληρώσει την εκτέλεση του το υποέργο T_i, δηλαδή, $i<j$. Χρησιμοποιούμε επίσης τον συμβολισμό $T_i \rightarrow T_j$ ή $i \rightarrow j$ και εννοούμε ότι το T_i προηγείται του T_j.

Βασικές Έννοιες (pipelining)

Παράδειγμα: Έστω ότι το έργο T είναι να υπολογιστεί η τιμή της εξίσωσης

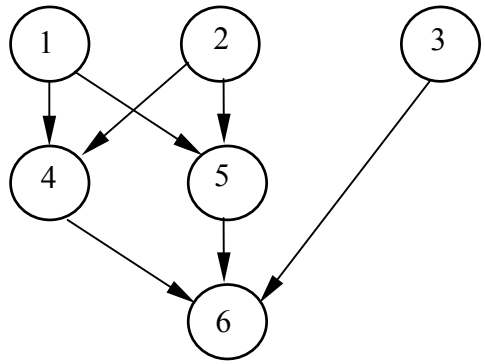
$$z=x*y*u*v+(a+b)/c+(x/u)*(y/v)$$

Μία πιθανή διάσπαση σε υποέργα είναι

T_1:$x*y$, T_2:$u*v$, T_3:$(a+b)/c:=A$,

T_4: $T_1 * T_2 :=B$, T_5:$(x*y)/(u*v):=C$,

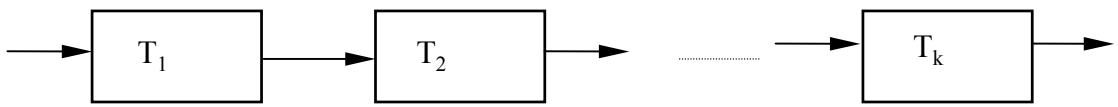
T_6:$z=C+A+B$, $T_6= T$



Το γράφημα προτεραιότητας που αντιστοιχεί στην παραπάνω διάσπαση είναι τοπολογικά διατεταγμένο. Υπάρχει όμως η πιθανότητα να ισχύει ότι $i<j$ χωρίς όμως το T_i να προηγείται του T_j. Στο παράδειγμα μας 3<4 χωρίς όμως το T_3 να προηγείται το T_4

Βασικές Έννοιες (pipelining)

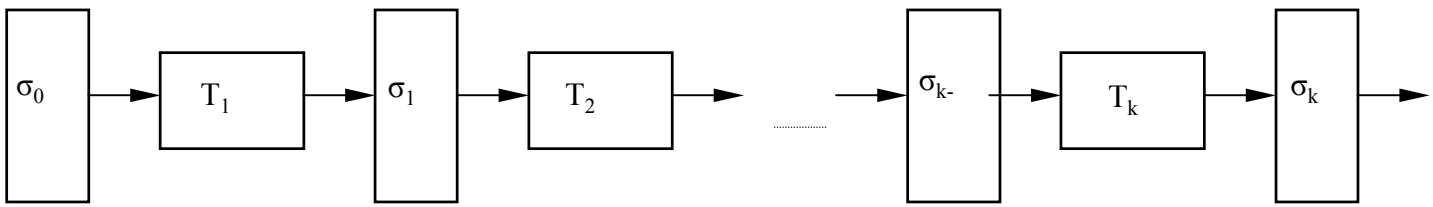
Η απεικόνιση της διάσπασης του προηγούμενου παραδείγματος σε αγωγό δεν προσφέρεται εξ' αιτίας της ιδιότητας: ότι ενώ υπάρχουν διαδικασίες που είναι τοπολογικά διατεταγμένες τα υποέργα δεν ακολουθούν απόλυτα την τοπολογική διάταξη. Αποτέλεσμα αυτού είναι η μη εύκολη (ή αδύνατη) απεικόνιση τέτοιων διαδικασιών σε γραμμικούς αγωγούς. Όπως ο αγωγός που φαίνεται στο παρακάτω σχήμα.



Αγωγοί

Σε πραγματικούς αγωγούς συνήθως μεταξύ των επεξεργαστών παρεμβάλλεται ένας καταχωρητής – συνδετήρας *σ* (latch: μάνταλο). Η χρήση του καταχωρητή είναι να κρατά τα αποτελέσματα της επεξεργασίας από τον προηγούμενο επεξεργαστή ή και να δημιουργεί μία τεχνητή καθυστέρηση.

Η καθυστέρηση χρησιμοποιείται για λόγους συγχρονισμού, δηλαδή καθυστερούνται ενδιάμεσα αποτελέσματα ώστε να μεταβιβάζονται στον επόμενο επεξεργαστή σε προκαθορισμένες χρονικές στιγμές. Στο επόμενο σχήμα φαίνεται ένας τέτοιος γραμμικός αγωγός με καταχωρητές.



Αγωγοί

Στην περίπτωση πραγματικών αγωγών ο αρχιτέκτονας του αγωγού πρέπει να ορίσει μία στοιχειώδη χρονική μονάδα t^0 ώστε οι μεταβιβάσεις δεδομένων να γίνονται σε τακτά χρονικά διαστήματα μήκους t^0. Αν για την εκτέλεση μιας εντολής T_i οι χρόνοι t_i $i= 1, \ldots, k$ είναι ίδιοι, τότε μπορούμε να θέσουμε σε όλους τους συνδετήρες την ίδια χρονική καθυστέρηση τ, ενώ μία καλή επιλογή γι' αυτή την περίπτωση θα ήταν:

$$t^0 := \tau + max\{t_i \mid i= 1, \ldots, k\}$$

Αγωγοί

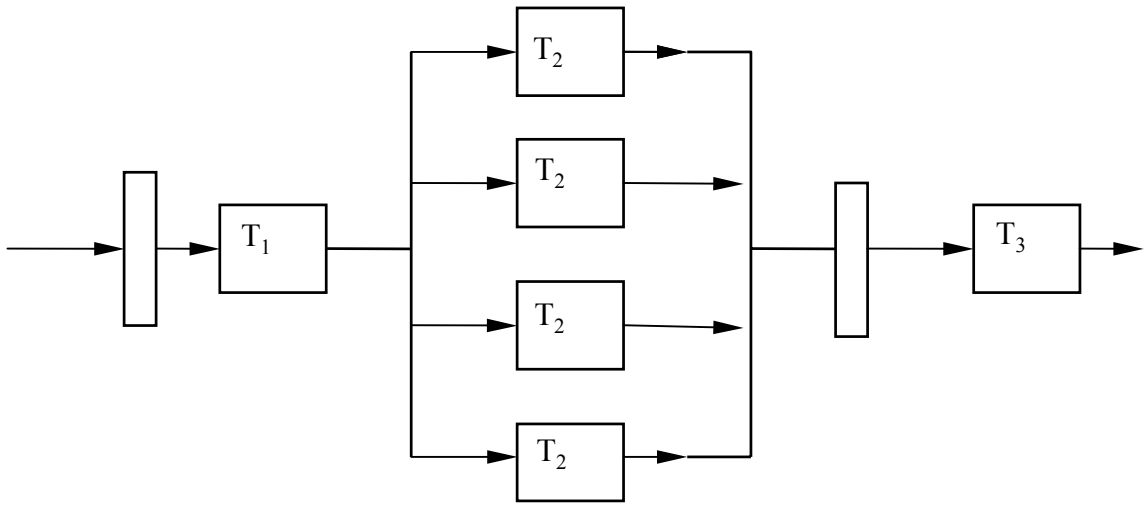
Αν όμως υπάρχουν έργα με πολύ διαφορετικούς χρόνους, δηλαδή $\tau_1 << \tau_2$ τότε είτε θα πρέπει να καθυστερήσουμε τεχνητά το τ_1, είτε θα προκύψει συμφόρηση με εισροή πολλών αποτελεσμάτων από τον επεξεργαστή T_1 στον T_2 ο οποίος δεν θα προλαβαίνει να δεχθεί και να επεξεργαστεί τα στοιχεία που θα προωθούνται από τον T_1. Θα έχουμε δηλαδή πρόβλημα συσσώρευσης αποτελεσμάτων.

Λύσεις στο πρόβλημα της συσσώρευσης:

Μία λύση είναι να διασπάσουμε σε περισσότερα υποέργα τη διαδικασία που υποέργου T_2, αν αυτό είναι εφικτό. Η λύση αυτή δίνει μακρύτερο αγωγό και μειώνει το *max* της προηγούμενης σχέσης. Έτσι το t^0 γίνεται πιο λογικό και πλησιάζει το χρόνο τον γρήγορων επεξεργαστών (διαδικασιών).

Αγωγοί

Μια άλλη λύση στο πρόβλημα της συσσώρευσης είναι να χρησιμοποιηθούν πολλά αντίγραφα του αργού επεξεργαστή (ή αντίστοιχα της χρονοβόρας διαδικασίας) T_i. Αν για παράδειγμα $t_2 = 4\ t_1$ τότε η συμφόρηση μπορεί να αποφευχθεί με 4 αντίγραφα του T_2, όπως φαίνεται και στο επόμενο σχήμα.



Αγωγοί

Συνέχεια του προηγούμενου παραδείγματος:

Αφού καθοριστεί ο t^0, ο χρόνος για την εκτέλεση του έργου $T(n)$ είναι:

$$t_{pipe}(T,n) = (\alpha + k - 1 + n)\, t^0.$$

Σύμφωνα με αυτά που αναφέρθηκαν στα εισαγωγικά περί αγωγών.

Όπου $k=p(\varepsilon)$.

Αγωγοί

Είδη αγωγών:

Ανάλογα με τη λειτουργία και την οργάνωση τους υπάρχουν *αριθμητικοί*, *εντολικοί* και *αγωγοί επεξεργαστών*, οι οποίοι χωρίζονται σε *μονολειτουργικούς* και *πολυλειτουργικούς* και οι οποίοι με τη σειρά τους χωρίζονται σε *δυναμικούς* και *στατικούς*.

ΔΙΑΝΥΣΜΑΤΙΚΕΣ ΕΝΤΟΛΕΣ

Μία διανυσματική εντολή μήκους *n*, όπως είναι η πρόσθεση δύο διανυσμάτων, μπορεί να εκτελεστεί από ειδικά σχεδιασμένους αγωγούς.

Αφού γίνει διαθέσιμη η σχετική διανυσματική εντολή σε ένα υπολογιστή, ο αγωγός παραλαμβάνει τις απαραίτητες συνιστώσες των διανυσμάτων κατά τακτά χρονικά διαστήματα και εκτελεί τις σχετικές πράξεις στις συνιστώσες. Αν ο αγωγός διασπάσει την εντολή σε *p(ε)* υποεντολές, ο χρόνος εκτέλεσης της διανυσματικής εντολής μήκος *n* είναι

$$t_{pipe}(\varepsilon, n) = (a + p(\varepsilon) - 1 + n)t^0$$

Αν θεωρήσουμε τώρα δύο ανεξάρτητες διανυσματικές εντολές ε_1, ε_2 εν γένει διαφορετικές, και για απλούστευση θεωρούμε ότι $p(\varepsilon_1)=p(\varepsilon_2)=p$. Οι διανυσματικές αυτές εντολές μπορούν να εκτελεστούν με χρονική επικάλυψη από δύο κατάλληλους αγωγούς. Υποθέτουμε βεβαίως ότι η δεύτερη εντολή μπορεί να αρχίσει αμέσως μετά την εξαγωγή του πρώτου αποτελέσματος της πρώτης εντολής. Τότε αν οι αρχικοί χρόνοι είναι $\alpha_1 t^0$ και $\alpha_2 t^0$,ο συνολικός χρόνος εκτέλεσης θα είναι: $t_{total} = (a + p + n)t^0$ όπου $a := max\{a_1, a_2\}$

ΔΙΑΝΥΣΜΑΤΙΚΕΣ ΕΝΤΟΛΕΣ

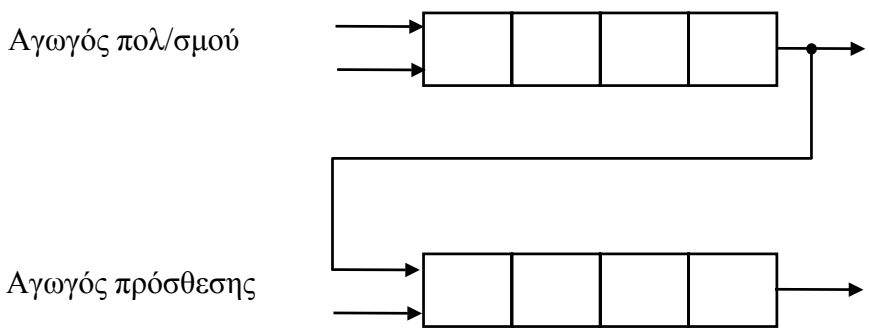
Όταν όμως οι δύο εντολές ε_1, ε_2 δεν είναι ανεξάρτητες αλλά η ε_2 χρειάζεται τα αποτελέσματα από την ε_1, τότε εφαρμόζουμε την τεχνική της "αλύσωσης" (chaining). Στην περίπτωση αυτή κάθε αποτέλεσμα αντιγράφου της συνιστώσας ε_1 από τον αγωγό που την εκτελεί στέλνεται αμέσως στην είσοδο του αγωγού που εκτελεί την ε_2.

Παράδειγμα: "inner product step"

ε1: $C = A * B$, [πολλαπλασιασμός ανά συνιστώσα: $C(i) = A(i)B(i)$]

ε2: $E = C + D$, [πρόσθεση ανά συνιστώσα: $E(i) = C(i) + D(i)$]

$$E(i) = A(i)B(i) + D(i), \qquad i = 1, \ldots, n$$



Αριθμητικοί Αγωγοί

Όλοι οι σύγχρονοι επεξεργαστές διαθέτουν αριθμητικούς αγωγούς για την εκτέλεση αριθμητικών πράξεων. Σαν παράδειγμα περιγράφουμε αγωγό για πρόσθεση κινητής υποδιαστολής.

Αλγόριθμος: Πρόσθεση Κινητής Υποδιαστολής

Αν σε κανονικοποιημένη μορφή οι δύο προσθετέοι είναι $x=a*2^p$ και $y=b*2^q$, τότε το αποτέλεσμα τους υπολογίζεται από την ακόλουθη σχέση.

$$z = x+y = c*2^{\max(p,q)} = d*2^{\varepsilon}, (0.5 \le d < 1).$$

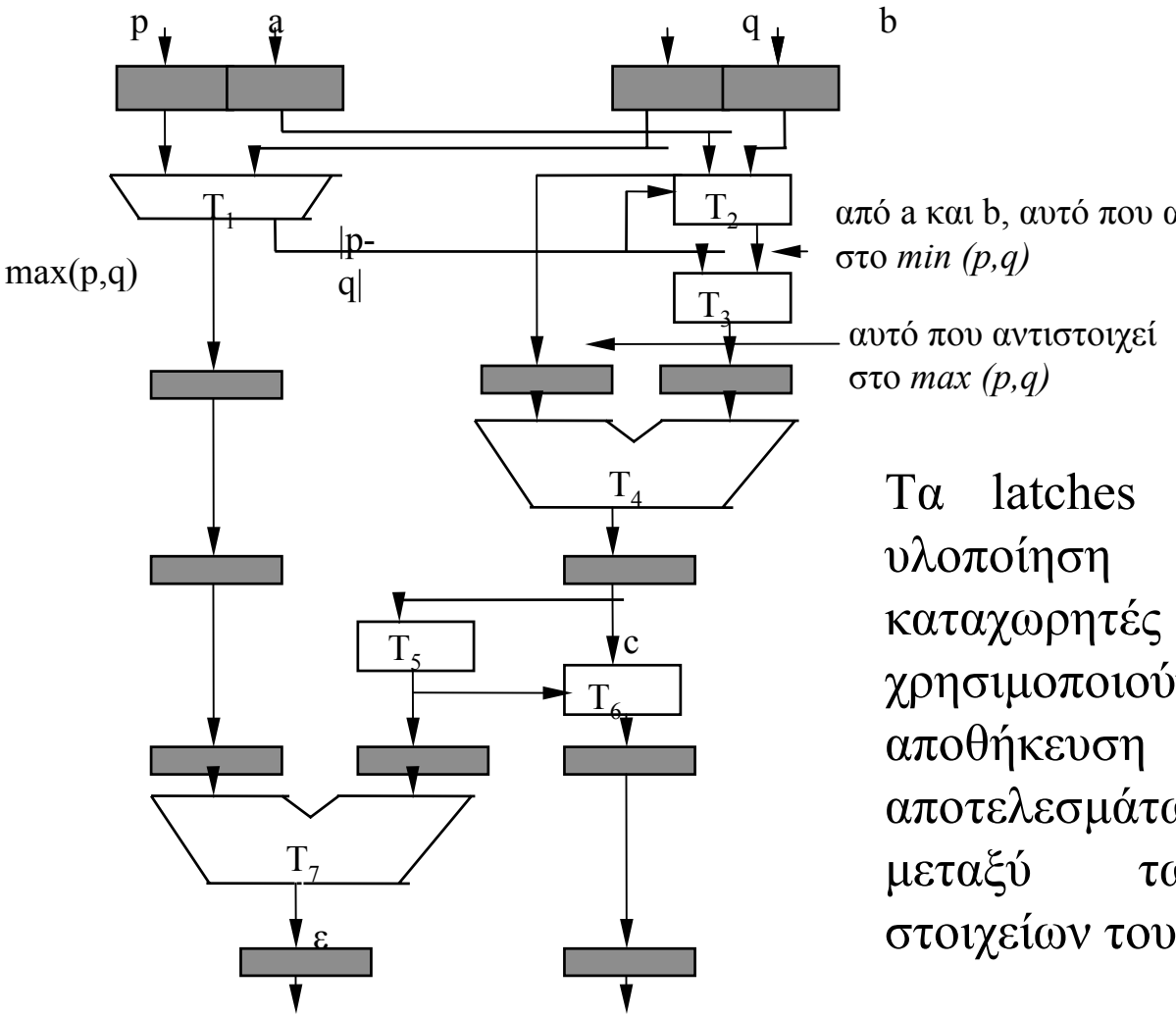
Στον αγωγό δίνονται οι αριθμοί *a, p, b, q* σε κανονικοποιημένη μορφή και πρέπει να υπολογιστούν οι *d* και *ε*.

Αγωγός για πρόσθεση κινητής υποδιαστολής

Πρώτα κάνουμε τη διάσπαση των πράξεων που απαιτούνται από τον αλγόριθμο σε υποέργα:

- T_1: Αφαίρεση των εκθετών p, q, για να υπολογίσουμε το $max(p,q)$ και τη θετική διαφορά $|p-q$.
- T_2: Επιλογή μεταξύ a, b, αυτού που αντιστοιχεί στο $max(p,q)$ και αυτού που αντιστοιχεί στο $min(p,q)$.
- T_3: Μετακίνηση κατά $(p-q)$ θέσεις, σε αυτόν που αντιστοιχεί στο $min(p,q)$.
- T_4: Πρόσθεση των a, b, (το ένα μετακινημένο από το T_3). Το αποτέλεσμα είναι c.
- T_5: Μέτρημα των κύριων μηδενικών στο c, έστω ότι αυτά είναι μ.
- T_6: Μετακίνηση κατά μ θέσεις του c, οπότε προκύπτει το d.
- T_7: Αφαίρεση εκθετών $max(p,q)$ και μ, οπότε προκύπτει ο ε.

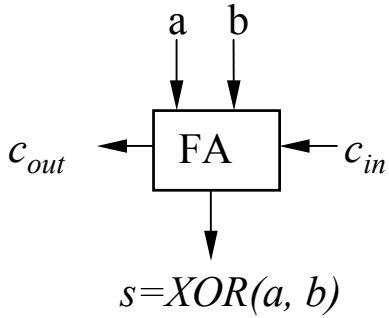
Αγωγός για πρόσθεση κινητής υποδιαστολής



Τα latches που φαίνονται στην υλοποίηση του αγωγού είναι καταχωρητές οι οποίοι χρησιμοποιούνται για την αποθήκευση των ενδιάμεσων αποτελεσμάτων και το συγχρονισμό μεταξύ των επεξεργαστικών στοιχείων του αγωγού.

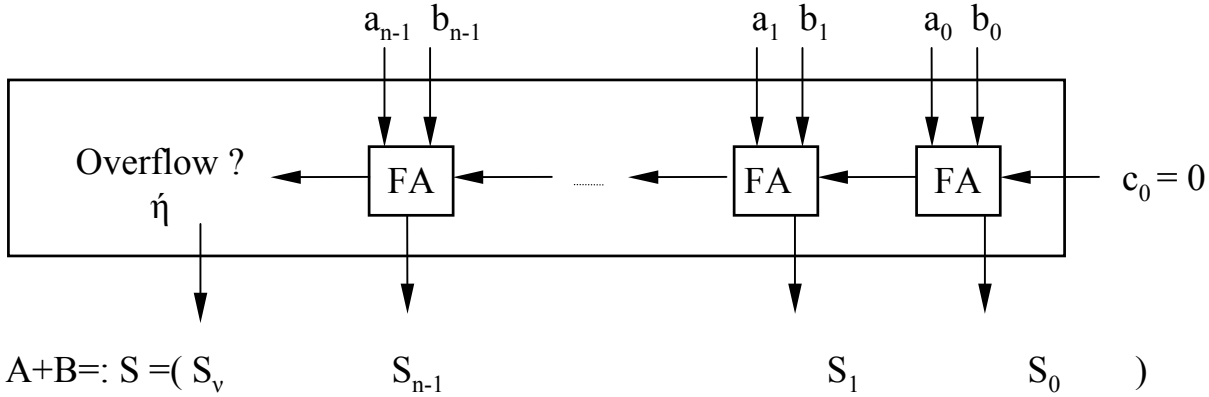
Αγωγός για διανυσματική πρόσθεση με διατάξεις πλήρους αθροιστή (Full Adder)

Στην επόμενη εικόνα απεικονίζεται σχηματικά ένας πλήρης αθροιστής (full Adder)



Για να προσθέσουμε δύο αριθμούς των *n* bits χρειαζόμαστε ένα αθροιστή διάδοσης κρατουμένων CPA (Carry Propagation Adder). Ο απαιτούμενος CPA υλοποιείται από την γραμμική σύνδεση *n* Full Adder. Ένας CPA δέχεται δύο αριθμούς των *n* bits και παράγει ως άθροισμα, των δύο αυτών αριθμών, ένα αριθμό των *n* ή *n+1* bits (ανάλογα με το αν έχουμε ή όχι παραγωγή υπολοίπου στην τελευταία πράξη της πρόσθεσης).

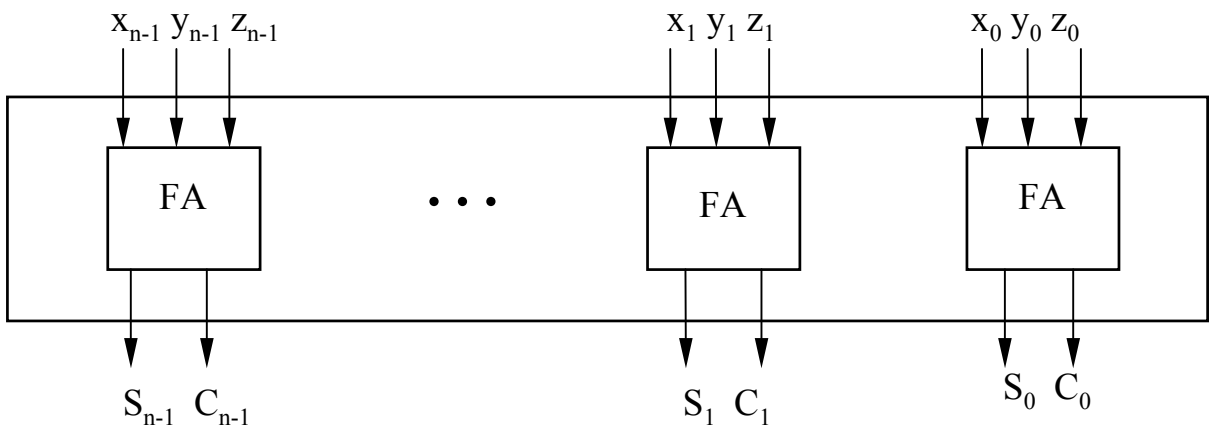
Αγωγός για διανυσματική πρόσθεση με διατάξεις πλήρους αθροιστή (Full Adder)



Είναι προφανές ότι τρεις αριθμοί μπορούν να προστεθούν με δύο CPAs, ή και με ένα CPA που θα χρησιμοποιηθεί όμως δύο φορές διαδοχικά.

Αγωγός διανυσματικής πρόσθεσης με CSAs

Υπάρχει διάταξη από Full Adders που μπορεί να αθροίσει τρεις αριθμούς ταυτόχρονα. Η διάταξη αυτή ονομάζεται Carry Save Adder (CSA). Στους CSAs οι FAs είναι οι ίδιοι με τους FAs των CPAs, αλλά σε διαφορετική διάταξη και μεταξύ τους ανεξάρτητοι όπως φαίνεται και στο σχήμα.



Αγωγός διανυσματικής πρόσθεσης με CSAs

Πρόσθεση 16 αριθμών με "δέντρο CSA