ΥΠΟΥΡΓΕΙΟ ΕΘΝΙΚΗΣ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ ΕΠΕΑΕΚ
ΕΥΡΩΠΑΪΚΗ ΕΝΩΣΗ
ΣΥΓΧΡΗΜΑΤΟΔΟΤΗΣΗ
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
ΕΛΛΑΔΑ 2008
Ανάπτυξη παντού. Ανάπτυξη για όλους.
Η ΠΑΙΔΕΙΑ ΣΤΗΝ ΚΟΡΥΦΗ
Επιχειρησιακό Πρόγραμμα
Εκπαίδευσης και Αρχικής
Επαγγελματικής Κατάρτισης

Αλγόριθμοι για Χρονικό Προγραμματισμό (Scheduling)

Εισαγωγή

Ο χρονικός προγραμματισμός είναι σημαντικό πρόβλημα:

- ✓ Καθορίζει τον ακριβή χρόνο έναρξης κάθε λειτουργίας με βάση τις εξαρτήσεις του ακολουθιακού γράφου.
- ✓ Καθορίζει τον παραλληλισμό της υλοποίησης και άρα την απόδοση.
- ✓ Επηρεάζει το υλικό: ο αριθμός των λειτουργιών που προγραμματίζονται σε ένα βήμα καθορίζει το υλικό που απαιτείται.
- ✓ Λαμβάνει υπόψη περιορισμούς σε υλικό για την απόδοση λύσεων.
- ✓ Για διάφορες τιμές του κύκλου ρολογιού μπορεί να δώσει area/latency trade off.
- ✓ Διαφορετικές αρχιτεκτονικές επηρεάζουν το Scheduling (πχ pipelining).
- ✓ Οι γλωσσικές δομές επηρεάζουν το Scheduling (πχ loops).

Μοντέλο προβλημάτων Scheduling

Βασικά στοιχεία μοντελοποίησης προβλήματος:

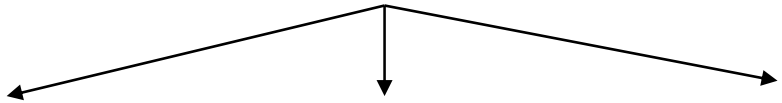
- Χρησιμοποιούνται μη-ιεραρχικοί ακολουθιακοί γράφοι με τις κορυφές v_i να αντιστοιχούν στις λειτουργίες και οι ακμές σε εξαρτήσεις.
- Η καθυστέρηση κάθε λειτουργίας v_i είναι ίση με d_i.
- Θεωρούμε ότι οι καθυστερήσεις είναι ανεξάρτητες δεδομένων και γνωστές. Επίσης το ρολόι είναι μονοφασικό (edge-triggered ffs).
- Ο χρόνος έναρξης κάθε λειτουργίας είναι t_i.
- Το latency είναι η διαφορά χρόνου έναρξης της καταβόθρας από την πηγή $\lambda=t_n-t_0$. (Θεωρούμε μοναδιαίους χρόνους και $t_0=1$)
- **Βασική σχέση**: ο χρόνος έναρξης μίας λειτουργίας είναι ίσος ή μεγαλύτερος από τον χρόνο έναρξης των οδηγών κορυφών + καθυστέρηση εκτέλεσης της λειτουργίας:

$$\mathbf{t_i \geq t_j + d_j \ \forall\ i, j: (v_j, v_i) \in E\ (\sigma\acute{\upsilon}\nu o\lambda o\ \alpha\kappa\mu\acute{\omega}\nu)}$$

Μοντέλο προβλημάτων Scheduling

Υποθέσεις (αρχικές)

- Δεν έχουμε δομές απόφασης ή επανάληψης στην περιγραφή.
- Κάθε λειτουργία εκτελείται σε ακριβώς ένα βήμα ελέγχου.
- Κάθε τύπος λειτουργίας μπορεί να εκτελεστεί από ένα και μόνο τύπο λειτουργικής μονάδας.
- Είναι δοσμένη μία βιβλιοθήκη με λειτουργικές μονάδες με γνωστά χαρακτηριστικά (area – delay).
- Είναι γνωστό το μήκος του βήματος ελέγχου.

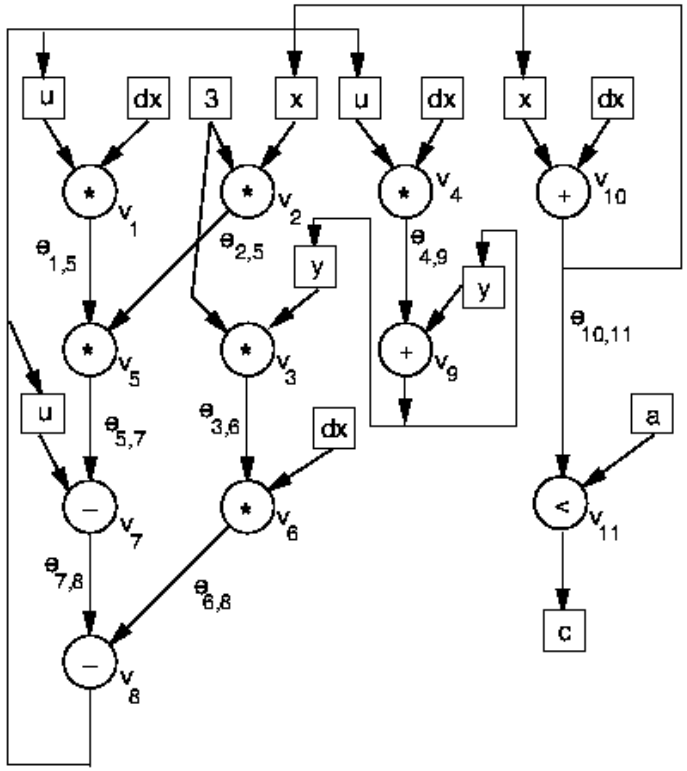


Unconstrained	*Time Constrained*	*Resource Constrained*
Χωρίς περιορισμούς	*Ελαχιστοποίηση αριθμού λειτουργικών μονάδων για δεδομένο αριθμό βημάτων*	*Ελαχιστοποίηση αριθμού βημάτων για δεδομένο κόστος σχεδίασης*

Scheduling χωρίς περιορισμούς

```
while (x < a)   loop
   x1 := x + dx;
   u1 := u – (3 * x * u * dx) – (3 * y * dx);
   y1 := y + (u*dx);
   x  := x1; u := u1; y := y1;
end loop;
```

VHDL Behavior



DFG Representation

Scheduling χωρίς περιορισμούς

ASAP αλγόριθμος: ο χρόνος εκκίνησης κάθε λειτουργίας είναι ο μικρότερος δυνατός.

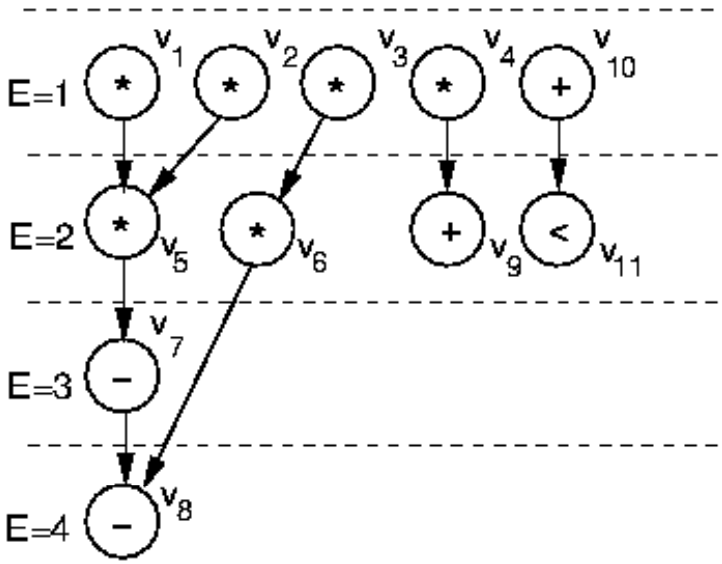
ALAP αλγόριθμος: ο χρόνος εκκίνησης μίας λειτουργίας είναι ο μεγαλύτερος δυνατός για latency λ.

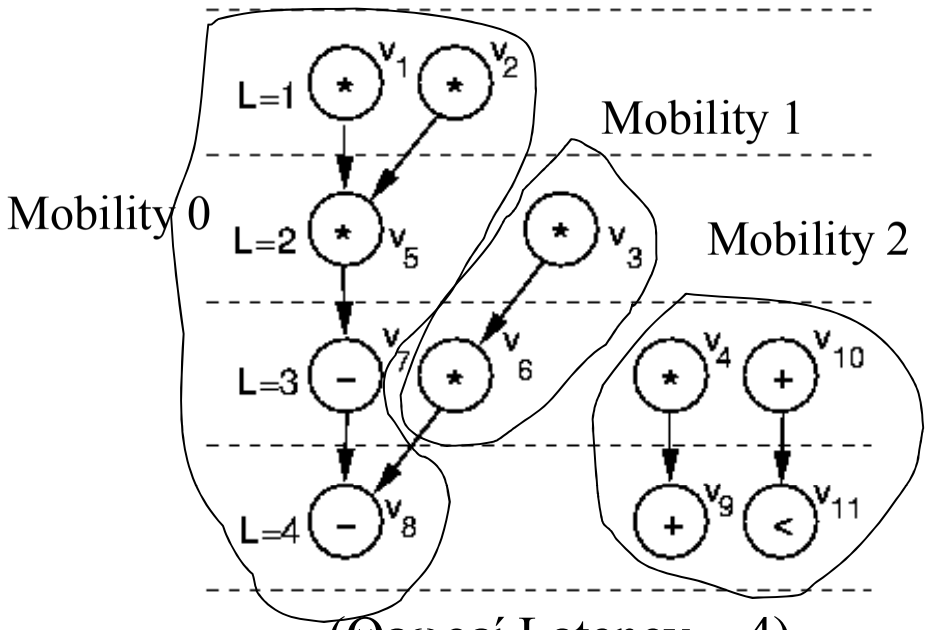
Mobility ή slack λειτουργίας: η διαφορά των χρόνων εκκίνησης μίας λειτουργίας ASAP, ALAP.

Mobility = 0 $\Rightarrow$ *η λειτουργία μπορεί να ξεκινήσει στο δεδομένο χρόνο και μόνο*

(Θεωρούν ότι δεν υπάρχει περιορισμός σε υλικό - Unconstrained)

Scheduling χωρίς περιορισμούς





(Θεωρεί Latency = 4)

ASAP Schedule

4 πολλαπλασιαστές

1 αθροιστή / αφαιρέτη

1 συγκριτή

ALAP Schedule

2 πολλαπλασιαστές

1 αθροιστή + 1 αφαιρέτη

1 συγκριτή

Scheduling με χρονικούς περιορισμούς

- ✓ Πρέπει να ικανοποιηθούν deadlines ή/και release times λειτουργιών.
- ✓ Τα deadlines, release times είναι περιορισμοί **απόλυτου χρόνου**.
- ✓ Περιορισμοί **σχετικού χρόνου** (Relative timing constraints): καθορισμός του χρόνου που μεσολαβεί ανάμεσα σε δύο λειτουργίες. (παράδειγμα: πρωτόκολλα επικοινωνίας)

Ορισμός: Οι περιορισμοί σχετικού χρόνου ανάμεσα σε λειτουργίες v_i, v_j είναι θετικοί ακέραιοι l_{ij} τέτοιοι ώστε:

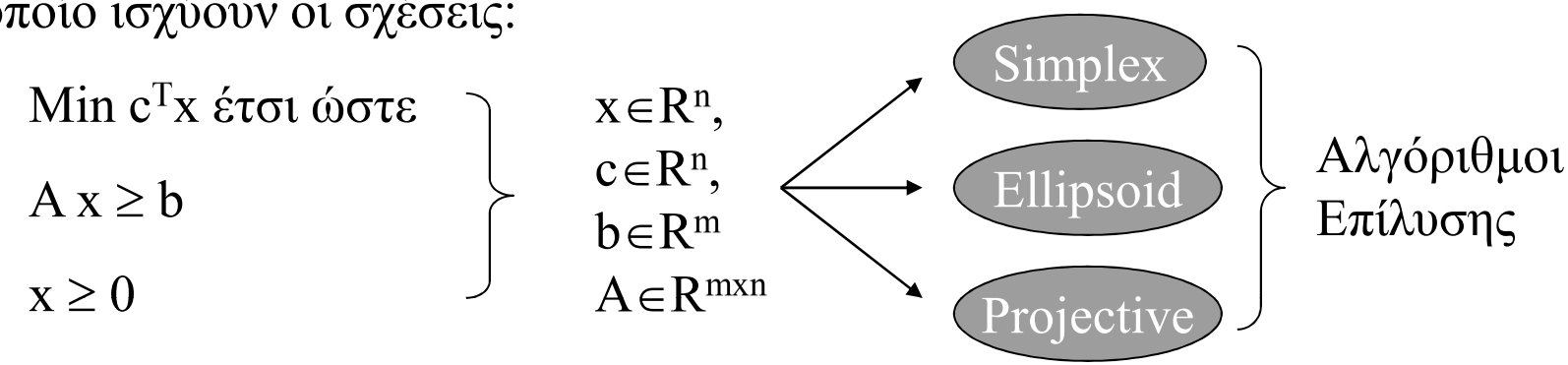
Ελάχιστος χρονικός περιορισμός $l_{ij} \geq 0$ απαιτεί $t_j \geq t_i + l_{ij}$

Μέγιστος χρονικός περιορισμός $u_{ij} \geq 0$ απαιτεί $t_j \leq t_i + u_{ij}$

Μέθοδοι
- Mathematical : *Integer Linear Programming*
- Constructive : *Force Directed*
- Iterative Refinement

Γραμμικά & Ακέραια Προβλήματα

Πολλά προβλήματα κατατάσσονται στην βελτιστοποίηση γραμμικών συναρτήσεων υπό γραμμικούς περιορισμούς. Γενικά ζητάμε ένα διάνυσμα x για το οποίο ισχύουν οι σχέσεις:

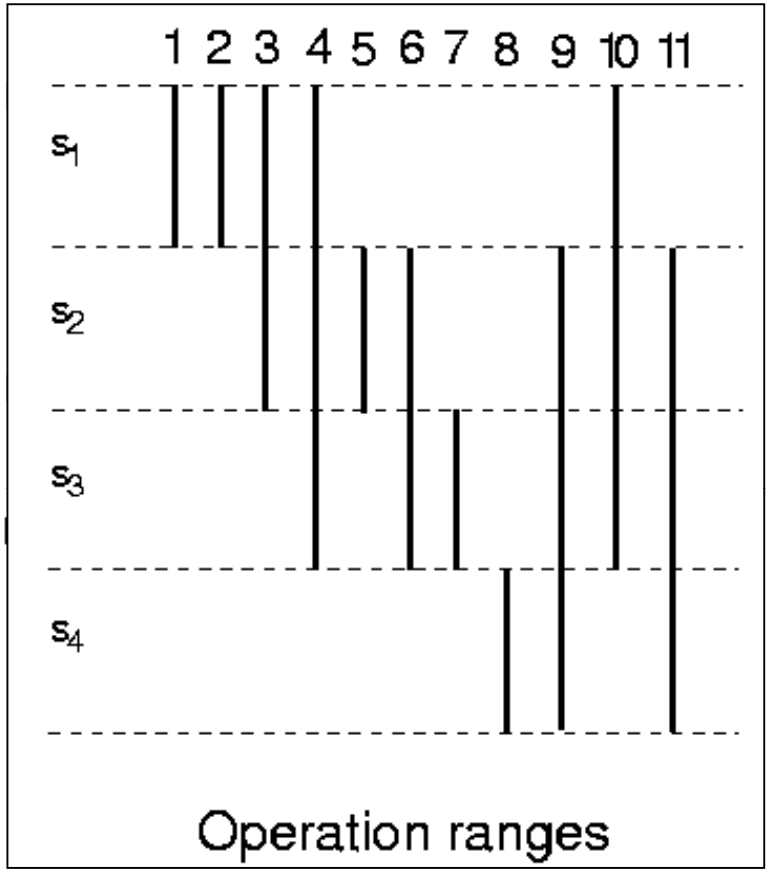


Η τυποποίηση αυτή ονομάζεται γραμμικό πρόγραμμα.

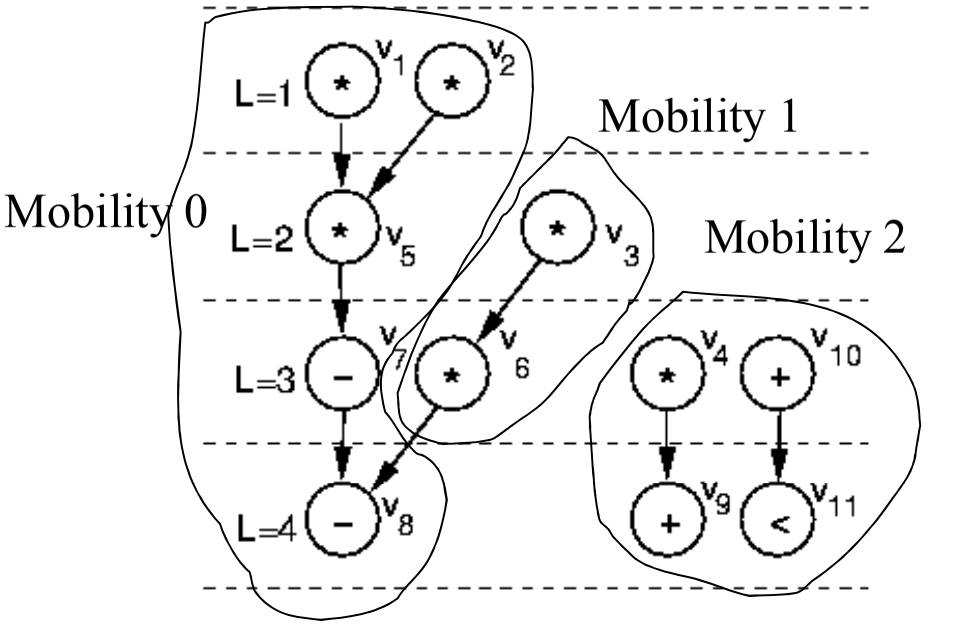
Για βελτιστοποίηση συνδυαστικών προβλημάτων το διάνυσμα x παίρνει ακέραιες τιμές ($x \in Z^n$) και έχουμε το Ακέραιο Γραμμικό πρόγραμμα (ILP – Integer Linear Programming).

ZOLP: Πρόβλημα απόφασης ILP που μαζί με τις μεταβλητές περιορίζεται σε δυαδικές τιμές 0, 1 (binary linear problems).

Integer Linear Programming



Διάστημα Κινητικότητας



o_k λειτουργία → ASAP → s_{Ek} βήμα

o_k λειτουργία → ALAP → s_{Lk} βήμα

$E_k \leq L_k$

Κάθε αποδεκτό schedule τοποθετεί την o_k μεταξύ των βημάτων s_{Ek} και s_{Lk}

Integer Linear Programming

T= σύνολο των m τύπων λειτουργιών

$S=\{s_j \mid 1 \le j \le r\}$ σύνολο βημάτων ελέγχου

$x_{ij} = 1$: η λειτουργία o_i στο βήμα s_j.

$OP= \Sigma o_i$ το σύνολο των n λειτουργιών

$t_i = type(o_i)$ ο τύπος της λειτουργίας o_i.

OP_{tk} σύνολο λειτουργιών τύπου t_k.

$INDEX_{tk}$ δείκτες του συνόλου OP_{tk}.

N_{tk} σύνολο μονάδων τύπου t_k.

C_{tk} κόστος μονάδας τύπου t_k.

Integer Linear Programming

Scheduling Formulation

Ελαχιστοποίησε κόστος υλικού $C_{t1}N_{t1}+C_{t2}N_{t2}+\ldots+C_{tk}N_{tk}$ υπό τις ακόλουθες προϋποθέσεις:

Συνάρτηση Α:

Κάθε λειτουργία προγραμματίζεται στο διάστημα κινητικότητας της μία και μόνο φορά

$$\forall i, 1 \le i \le n \quad \sum_{E_i \le j \le L_i} x_{ij} = 1$$

Συνάρτηση Β:

Σε κανένα βήμα δεν προγραμματίζονται περισσότερες από N_{tk} λειτουργίες τύπου t_k.

$$\forall j, 1 \le j \le r, \forall k, 1 \le k \le m \quad \sum_{i \in INDEX_{tk}} x_{ij} \le N_{tk}$$

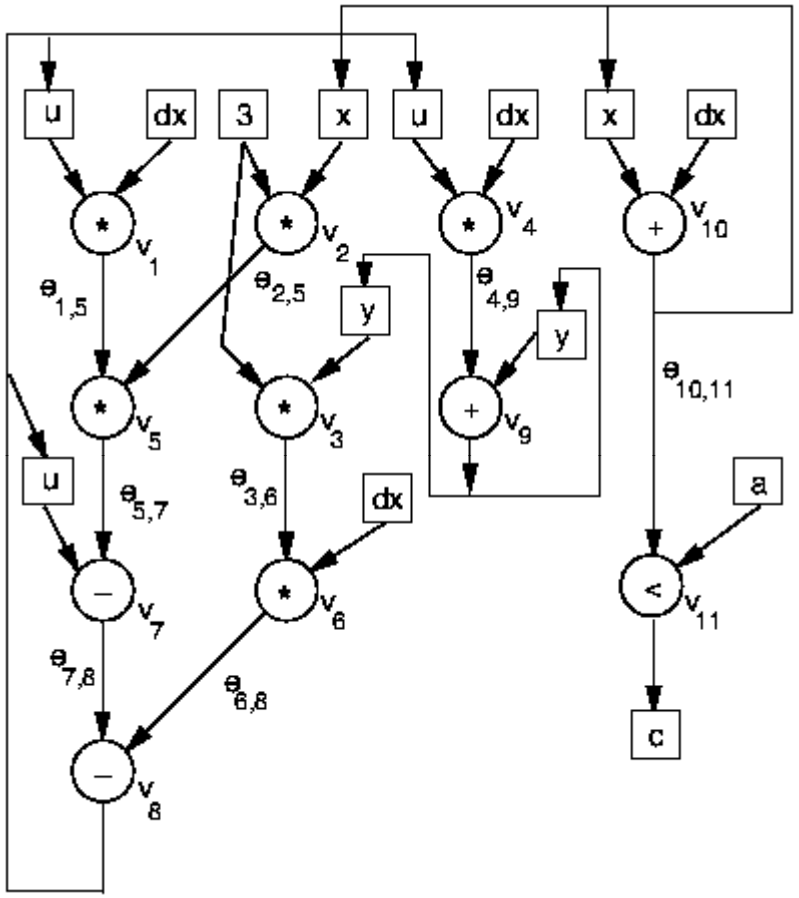
Integer Linear Programming

Συνάρτηση Γ:

Όλες οι λειτουργίες o_i που είναι προκάτοχοι της o_j προγραμματίζονται σε προηγούμενο βήμα από την o_j (αν $x_{i,k}=x_{j,l}=1$ τότε $k<l$)

$$\forall i, j \; o_i \in \Pr ed_{o_j} \left(\sum_{E_i \le k \le L_i} k \cdot x_{i,k} - \sum_{E_j \le l \le L_j} l \cdot x_{j,l} \right) \le -1$$

Παράδειγμα



Πολλαπλασιαστές: C_m, N_m

Αθροιστές: C_a, N_a

Αφαιρέτες: C_s, N_s

Συγκριτές: C_c, N_c

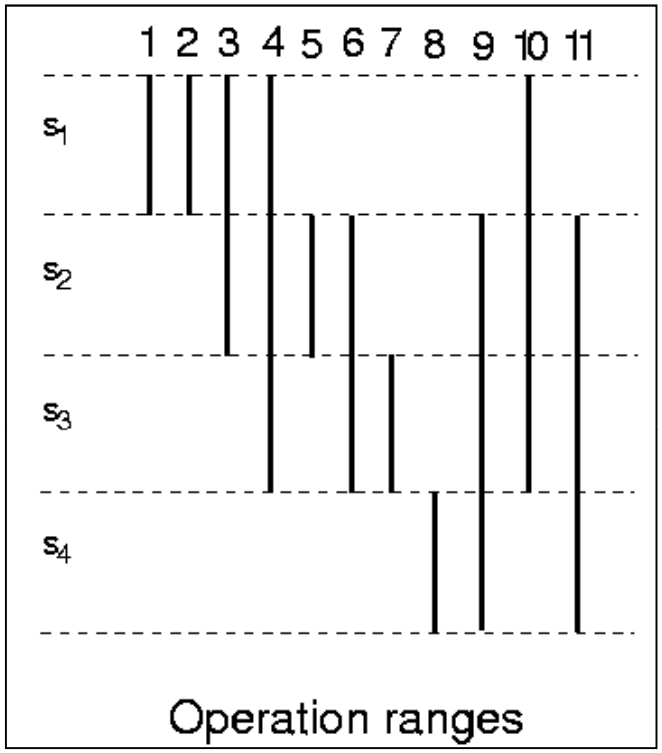
ILP:

Ελαχιστοποίηση του

$$C_mN_m + C_aN_a + C_sN_s + C_cN_c$$

Με τις ακόλουθες προϋποθέσεις:

Integer Linear Programming



Συνάρτηση Α:

Κάθε λειτουργία προγραμματίζεται στο διάστημα κινητικότητας της μία και μόνο φορά

$X_{1,1} = 1$

$X_{2,1} = 1$

$X_{3,1}+X_{3,2} = 1$

$X_{4,1}+X_{4,2}+X_{4,3} = 1$

$X_{5,2} = 1$

$X_{6,2}+X_{6,3} = 1$

$X_{7,3} = 1$

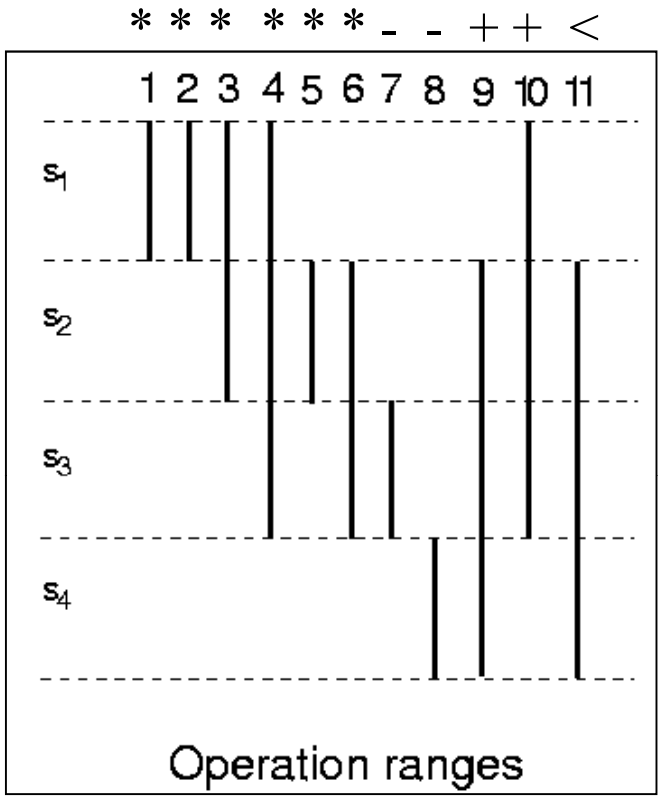
$X_{8,4} = 1$

$X_{9,2}+X_{9,3}+X_{9,4} = 1$

$X_{10,1}+X_{10,2}+X_{10,3} = 1$

$X_{11,2}+X_{11,3}+X_{11,4} = 1$

Integer Linear Programming



Συνάρτηση B:

Σε κανένα βήμα δεν προγραμματίζονται περισσότερες από N_{tk} λειτουργίες τύπου t_k.

$X_{1,1}+X_{2,1}+X_{3,1}+X_{4,1} \le N_m$

$X_{3,2}+X_{4,2}+X_{5,2}+X_{6,2} \le N_m$

$X_{4,3}+X_{6,3} \le N_m$

$X_{7,3} \le N_s$

$X_{8,4} \le N_s$

$X_{10,1} \le N_a$

$X_{9,2}+X_{10,2} \le N_a$

$X_{9,3}+X_{10,3} \le N_a$

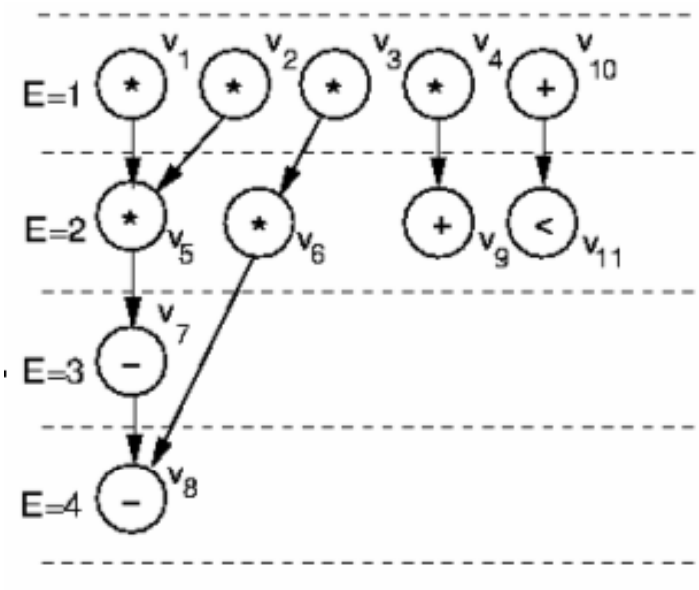
$X_{9,4} \le N_a$

$X_{11,2} \le N_c$

$X_{11,3} \le N_c$

$X_{11,4} \le N_c$

Integer Linear Programming



Συνάρτηση Γ:

Όλες οι λειτουργίες προκάτοχοι της o_j προγραμματίζονται σε προηγούμενο βήμα από την o_j (αν $x_{i,k}=x_{j,l}=1$ τότε $k<l$)

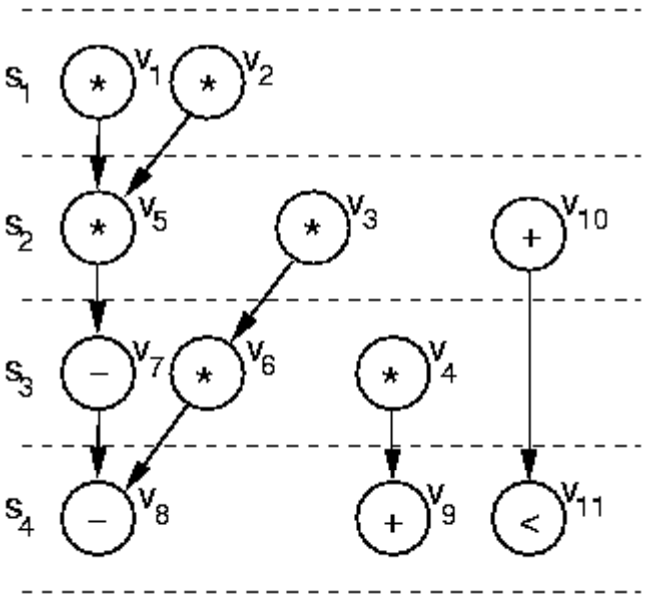
Μόνο για όσες έχουν κινητικότητα

$$1X_{3,1}+2X_{3,2}-2X_{6,2}-3X_{6,3} \leq -1$$

$$1X_{4,1}+2X_{4,2}+3X_{4,3}-2X_{9,2}-3X_{9,3}-4X_{9,4} \leq -1$$

$$1X_{10,1}+2X_{10,2}+3X_{10,3}-2X_{11,2}-3X_{11,3}-4X_{11,4} \leq -1$$

Integer Linear Programming



Final Schedule

Θεωρώντας ότι διαθέτουμε:

- 2 πολλαπλασιαστές (κόστος 2 καθένας)
- 1 αθροιστή (κόστος 1)
- 1 αφαιρέτη (κόστος 1)
- 1 συγκριτή (κόστος 1)

Η πολυπλοκότητα του ILP αυξάνει ραγδαία με την αύξηση της πολυπλοκότητας των δεδομένων.

⇩

Μόνο για μικρές περιγραφές

Force Directed Scheduling

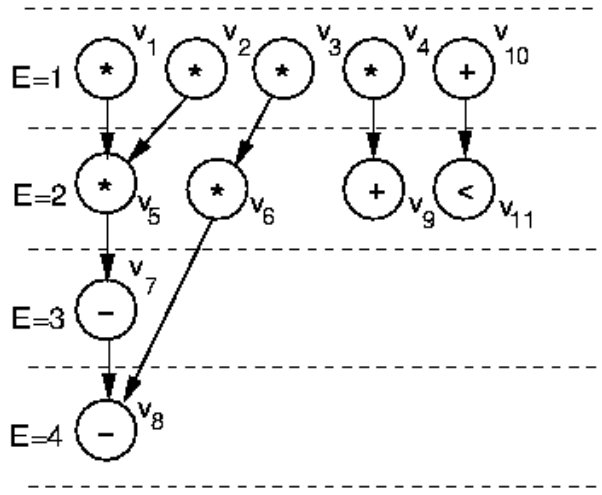
- ✓ Είναι η Constructive μέθοδος για Scheduling με χρονικούς περιορισμούς.
- ✓ Ονομάζεται Constructive γιατί κατασκευάζει λύση χωρίς backtracking.
- ✓ Στόχος είναι η μείωση του συνολικού αριθμού λειτουργικών μονάδων που χρησιμοποιούνται.
- ✓ Κατανέμει τις λειτουργίες ίδιου τύπου ομοιόμορφα σε όλες τις καταστάσεις.
- ✓ Έτσι πετυχαίνει υψηλή αξιοποίηση των resources σε όλους τους χρόνους.

Formulation

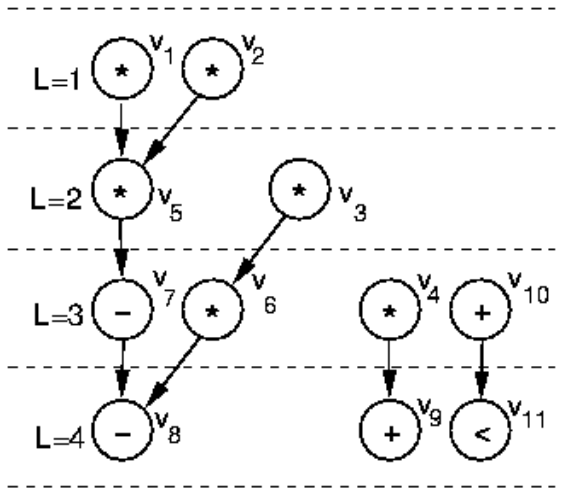
- ✓ Με τις τιμές ASAP,ALAP καθορίζει το εύρος $[E_i, L_i]$ κάθε λειτουργίας.
- ✓ Θεωρεί ότι κάθε λειτουργία μπορεί να προγραμματιστεί ισοπίθανα στο εύρος της και με πιθανότητα 0 αλλού.

$E_i \le j \le L_i$ η λειτουργία o_i μπορεί να προγρ. στο βήμα s_j με $p_j(o_i)=1/(L_i-E_i+1)$

Force Directed Scheduling



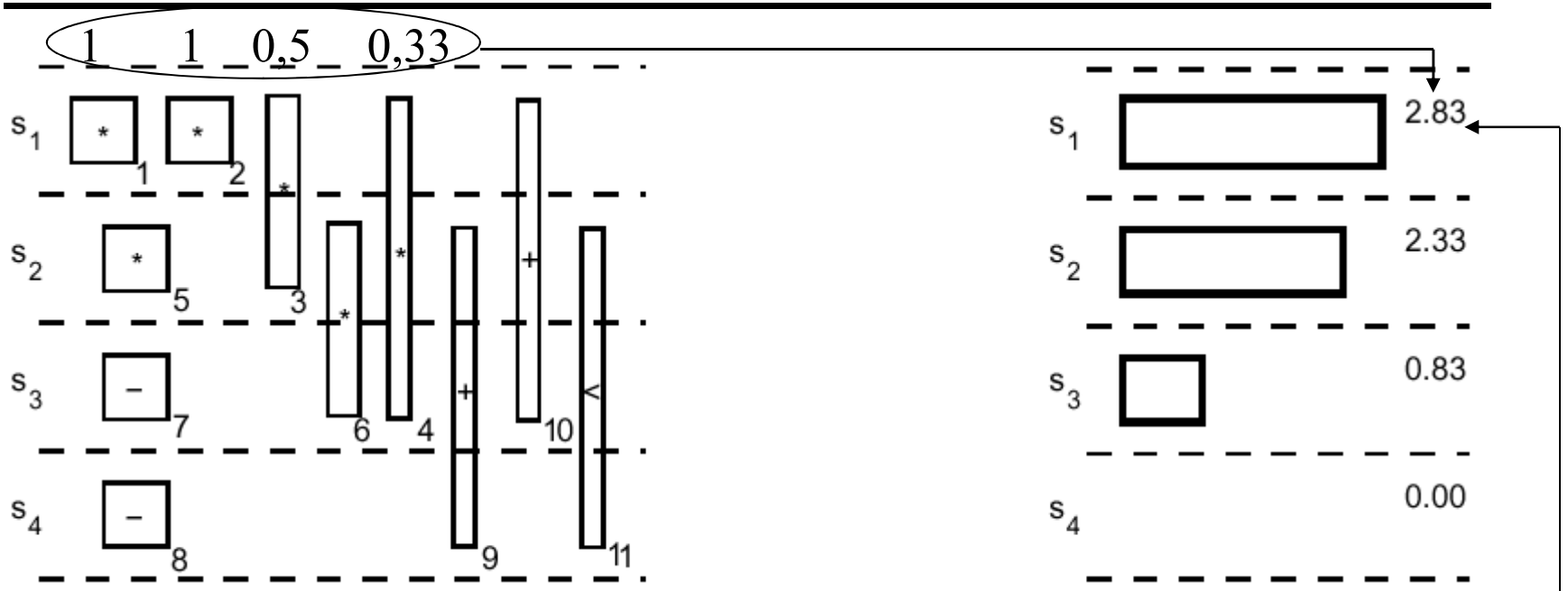
ASAP Schedule



ALAP Schedule

- ✓ Οι λειτουργίες 1, 2, 5, 7, 8 έχουν πιθανότητα 1 να προγραμματιστούν στα βήματα 1, 1, 2, 3, 4 αντίστοιχα.
- ✓ Οι λειτουργίες 3, 6 έχουν πιθανότητα 0,5 να μπουν στο 1, 2 ή 2, 3 βήμα αντίστοιχα
- ✓ Οι λειτουργίες 4, 9, 10, 11 έχουν πιθανότητα 0,333 να μπουν στο βήμα 1,2,3 (οι 4 και 10) ή 2,3,4 (οι 9, 11) αντίστοιχα

Force Directed Scheduling



- ✓ Το πάχος κάθε παραλ/μου είναι η πιθανότητα να μπει σε αυτό το βήμα η αντίστοιχη λειτουργία.
- ✓ Οι μπάρες αναπαριστούν το αναμενόμενο κόστος ενός τελεστή (εδώ πολλαπλ.) σε αυτό το βήμα (πχ s_1, *: $c_k(p_1(o_1)+p_1(o_2)+p_1(o_3)+p_1(o_4))$

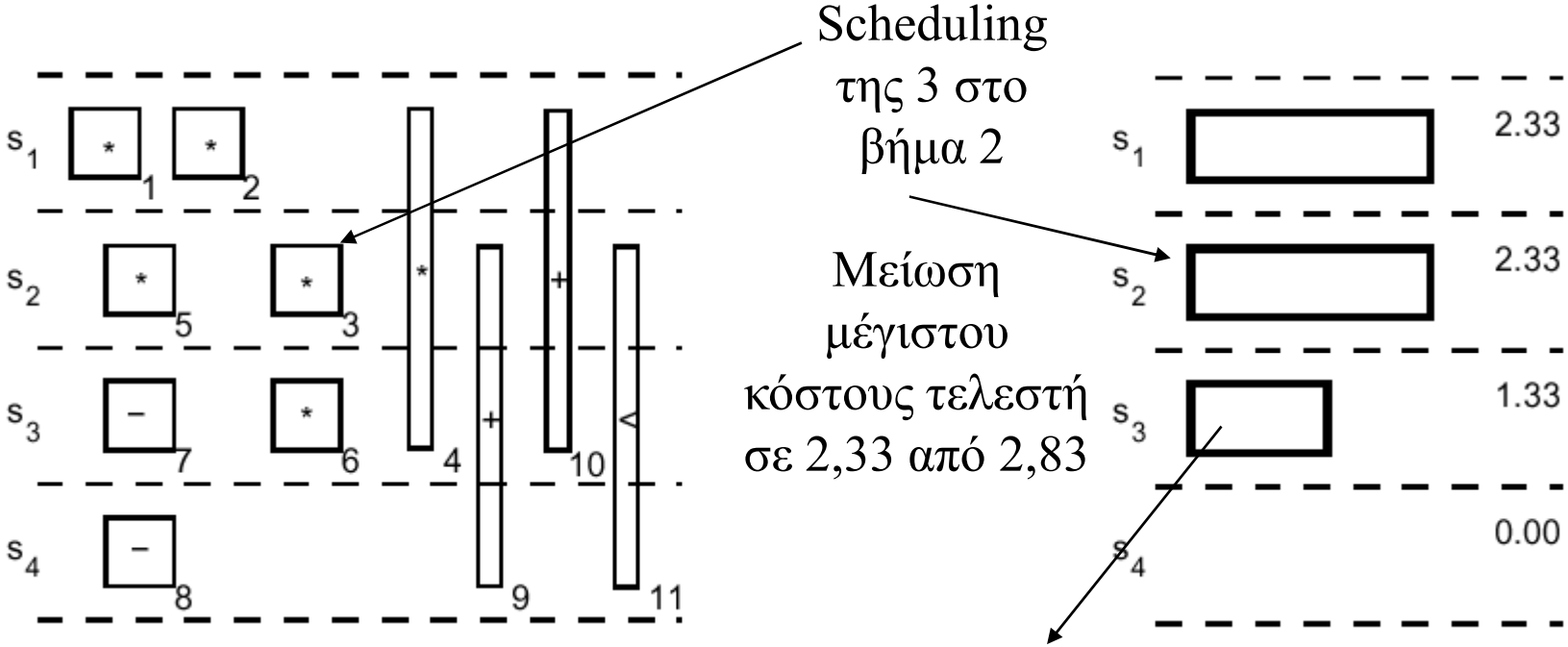
Κόστος Τελεστή

Άθροισμα p πολλαπλ. σε αυτό το βήμα

Force Directed Scheduling

- ✓ Τέτοιες μπάρες φτιάχνονται για όλους τους τελεστές.
- ✓ Όσο μεγαλύτερο το κόστος ενός τελεστή σε κάποιο βήμα, τόσο πιθανότερο είναι ο τελεστής να πρέπει να υλοποιηθεί σε αυτό το βήμα και πιθανώς να χρειαστούν πολλές λειτουργικές μονάδες.
- ✓ Όσο μικρότερο το κόστος ενός τελεστή σε κάποιο βήμα, τόσο πιθανότερο είναι ο τελεστής να μην υλοποιηθεί σε αυτό το βήμα και οι λειτουργικές μονάδες να μην αξιοποιηθούν.
- ✓ Μας ενδιαφέρει το μέγιστο κόστος ενός τελεστή το οποίο προσπαθούμε να μειώσουμε διαμοιράζοντας το σε όλα τα βήματα εξίσου.
- ✓ Δοκιμάζονται διάφορα schedules.
- ✓ Σε κάθε scheduling μίας λειτουργίας επαναϋπολογίζονται οι πιθανότητες των λειτουργιών (λόγω εξαρτήσεων).
- ✓ Το schedule που οδηγεί στην ελάχιστη τιμή αναμενόμενου κόστους για κάποιον τελεστή επιλέγεται.

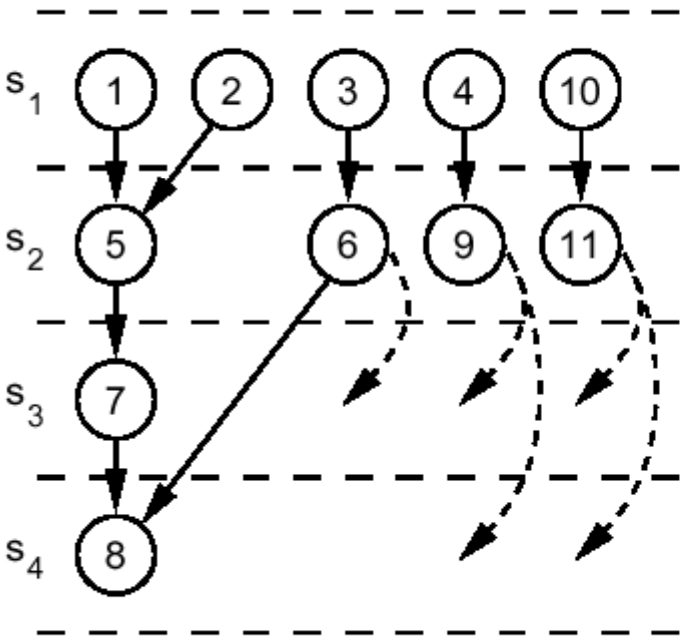
Force Directed Scheduling



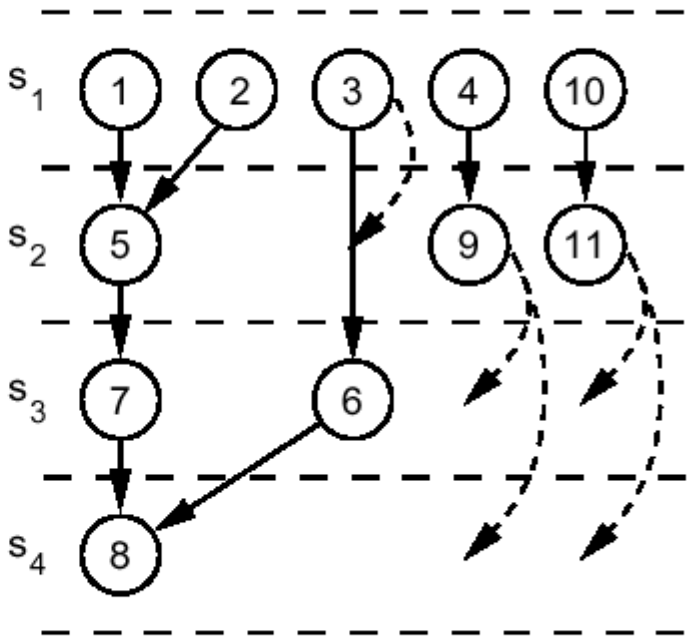
Αύξηση του τελεστή σε αυτό το βήμα αλλά δεν ενοχλεί αφού οι μονάδες υπάρχουν από άλλα βήματα με μεγαλύτερο τελεστή.

Iterative Refinement Scheduling

- ✓ Ένα πρώτο schedule επιτυγχάνεται με κάποιον αλγόριθμο.
- ✓ Νέα schedules προκύπτουν με rescheduling μίας λειτουργίας την φορά.
- ✓ Πρέπει να τηρούνται οι περιορισμοί εξάρτησης δεδομένων



Initial Schedule



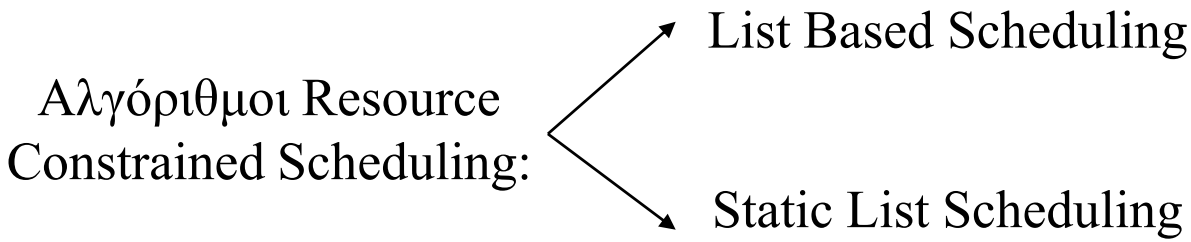
After Op 6 is moved and locked

Iterative Refinement Scheduling

- ✓ Ο αλγόριθμος προχωράει και εκτελεί μετακινήσεις έως ότου όλες οι λειτουργίες κλειδωθούν.
- ✓ Βρίσκουμε την υπο-ακολουθία των k μετακινήσεων που μεγιστοποιεί το συνολικό κέρδος.
- ✓ Εάν βρεθεί υπο-ακολουθία με θετικό κέρδος, την εφαρμόζουμε, ξεκλειδώνουμε λειτουργίες και επαναλαμβάνουμε.
- ✓ Μπορούμε να εκτελέσουμε τον αλγόριθμο πολλές φορές ξεκινώντας από διαφορετικά αρχικά schedules.

Resource Constrained Scheduling

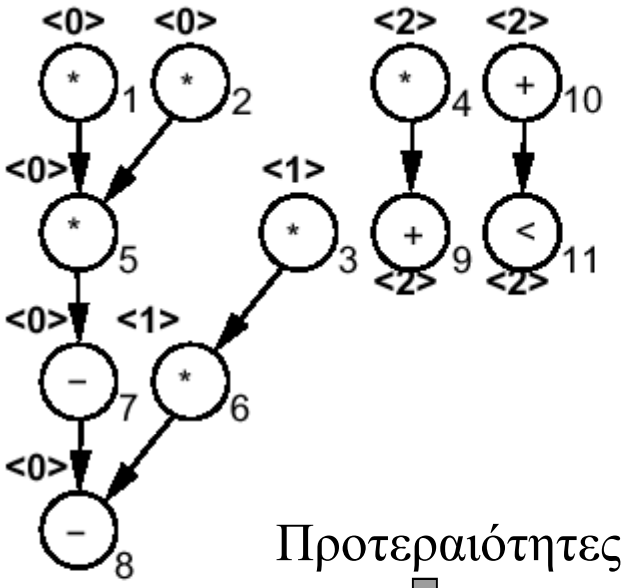
- ✓ Σε αυτό το είδος scheduling ξεκινάμε με περιορισμούς στην επιφάνεια.
- ✓ Ο περιορισμός μπορεί να είναι είτε αριθμός λειτουργικών μονάδων, είτε συνολική επιφάνεια.
- ✓ Όταν ο περιορισμός είναι σε επιφάνεια, ο αλγόριθμος επιλέγει και λειτουργικές μονάδες.
- ✓ Ο περιορισμός επιφάνειας ικανοποιείται όταν οι λειτουργίες που προγραμματίζονται σε ένα βήμα δεν παραβιάζουν τον περιορισμό.
- ✓ Οι εξαρτήσεις δεδομένων προφανώς πρέπει να τηρούνται.



List Based Scheduling

- ✓ Είναι γενίκευση του αλγόριθμου ASAP.
- ✓ Διατηρεί μία λίστα προτεραιότητας των “έτοιμων” κόμβων.
- ✓ Έτοιμος είναι ο κόμβος του οποίου όλοι οι προκάτοχοι έχουν προγραμματιστεί.
- ✓ Σε κάθε επανάληψη προγραμματίζονται οι λειτουργίες με την υψηλότερη προτεραιότητα (αρχή λίστας) μέχρις ότου χρησιμοποιηθούν όλα τα resources.
- ✓ Ο προγραμματισμός μίας λειτουργίας θέτει κάποιες άλλες έτοιμες, οι οποίες εισάγονται στην λίστα με βάση την προτεραιότητα τους.
- ✓ Κρίσιμος παράγοντας απόδοσης είναι η επιλογή της συνάρτησης προτεραιότητας

List Based Scheduling



Προτεραιότητες

DFG with mobilities

$PList_{*}$: 1<0>, 2<0>, 3<1>, 4<2> | 3<0>, 5<0>, 4<1>

$PList_{+}$: 10<2> | 11<1>

$PList_{-}$: NIL

$PList_{<}$: NIL

Ready List

Επανάληψη 2

Επανάληψη 1

$Resources_{*}$: 2 ⟶ 1, 2

$Resources_{+}$: 1 ⟶ 10

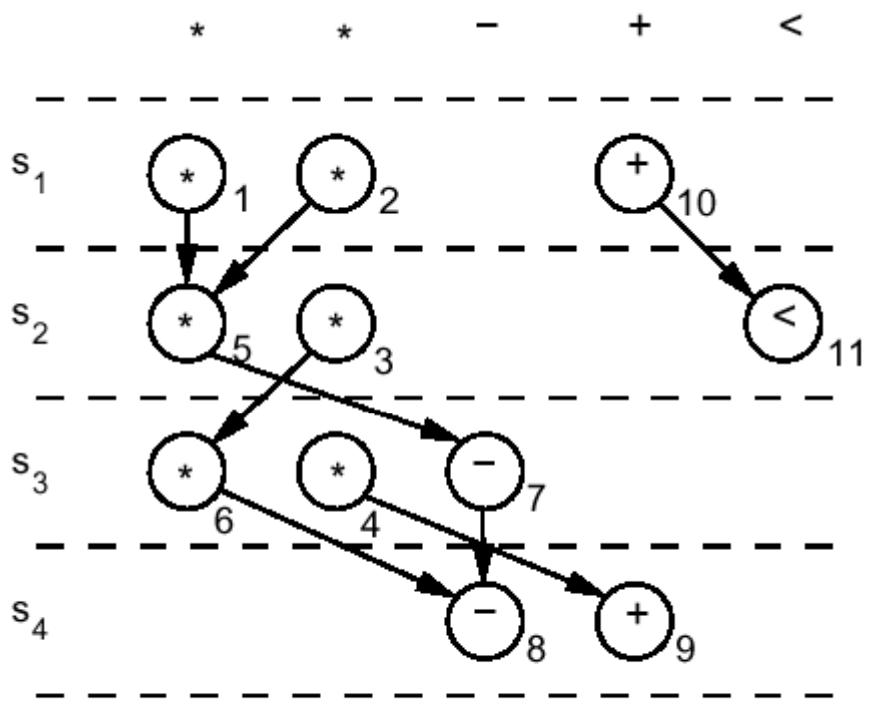
$Resources_{-}$: 1

$Resources_{<}$: 1

Resource Constraints

- ✓ Σαν προτεραιότητα χρησιμοποιείται η κινητικότητα (μικρή κινητικότητα = μεγάλη προτεραιότητα)

List Based Scheduling



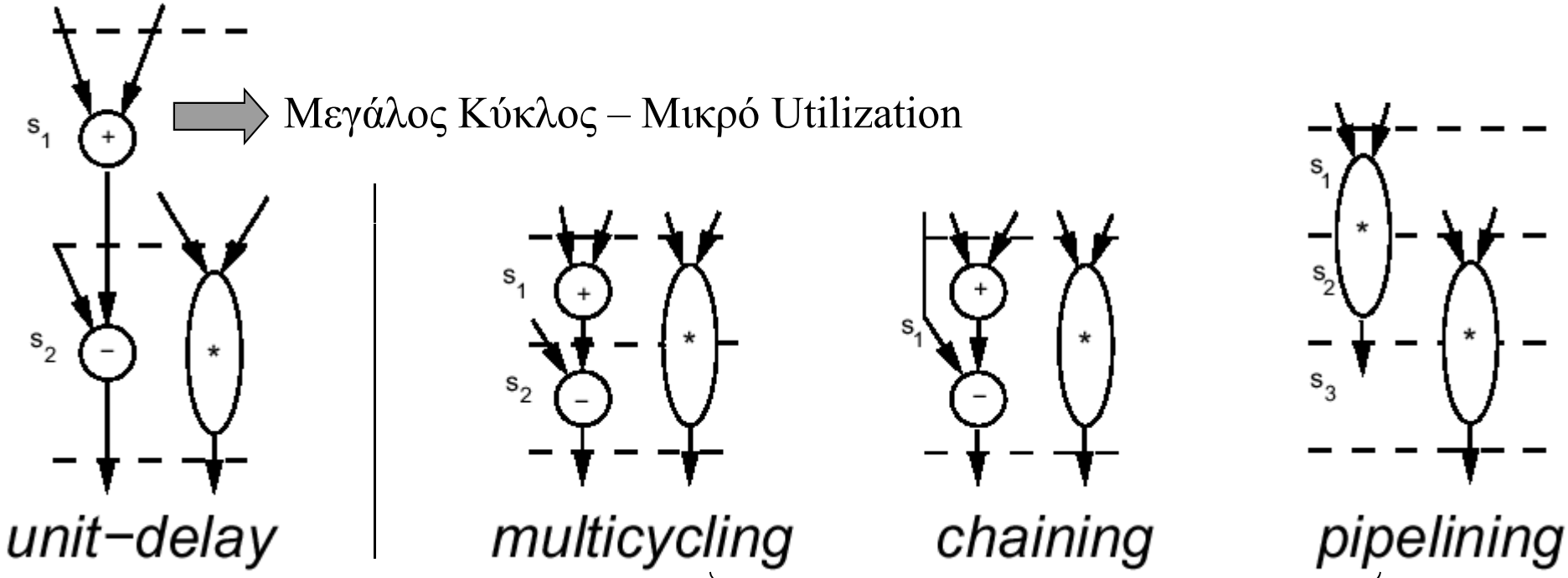
Scheduled DFG

Scheduling με Ρεαλιστικές Υποθέσεις

- ✓ Οι υποθέσεις που κάναμε δεν ανταποκρίνονται σε ρεαλιστικά σενάρια.
- ✓ Η πραγματικότητα είναι πιο περίπλοκη.
- ✓ Κάποιες πιο ρεαλιστικές υποθέσεις είναι οι ακόλουθες:
  1. Λειτουργικές μονάδες με ποικίλες καθυστερήσεις
  2. Λειτουργικές μονάδες με περισσότερες από μία λειτουργίες
  3. Πιο περίπλοκες περιγραφές συμπεριφοράς.

Λειτουργικές Μονάδες Ποικίλων Καθυστερ.

Η καθυστέρηση μίας λειτουργικής μονάδας εξαρτάται από την σχεδίαση της (πχ. ο πολλαπλασιαστής κινητής υποδιαστολής είναι πιο αργός από αθροιστή σταθερής υποδιαστολής).



Λειτουργικές Μονάδες Ποικίλων Καθυστερ.

Multicycling:

- ✓ Ο κύκλος ρολογιού μικραίνει έτσι ώστε οι ταχείες λειτουργίες να ολοκληρώνονται σε έναν κύκλο.
- ✓ Οι πιο αργές λειτουργίες απαιτούν περισσότερους κύκλους.
- ✓ Αυξάνεται η αξιοποίηση των ταχέων μονάδων αφού δεν μένουν ανενεργές κατά την διάρκεια των κύκλων.
- ✓ Στις εισόδους των μονάδων πολλαπλών κύκλων απαιτούνται latches για να διατηρούν τα δεδομένα έως ότου τα αποτελέσματα να είναι διαθέσιμα.
- ✓ Ο αυξημένος αριθμός βημάτων ελέγχου, αυξάνει και την πολυπλοκότητα του γράφου CDFG.

Λειτουργικές Μονάδες Ποικίλων Καθυστερ.

Chaining:

- ✓ Επιτρέπουμε την σειριακή εκτέλεση λειτουργιών σε ένα βήμα.
- ✓ Το αποτέλεσμα μίας μονάδας τροφοδοτείται σε μία άλλη μονάδα κατευθείαν.
- ✓ Απαιτεί συνδέσεις άμεσες των μονάδων (χωρίς παρεμβολή latches).

Pipelining:

- ✓ Είναι απλή και αποδοτική τεχνική αύξησης παραλληλισμού.
- ✓ Επηρεάζει τον τρόπο υπολογισμού του Schedule.
- ✓ Δύο λειτουργίες μπορούν να μοιράζονται την ίδια μονάδα και να εκτελούνται ταυτόχρονα (σε διαφορετικές βαθμίδες).

Λειτ. Μονάδες Πολλαπλών Λειτουργιών.

- ✓ Οι πολυλειτουργικές μονάδες χρησιμοποιούνται συχνά καθώς στοιχίζουν λιγότερο από ένα ισοδύναμο σύνολο μονο-λειτουργικών μονάδων.

  Παράδειγμα είναι ο αθροιστής/αφαιρέτης.

- ✓ Οι λειτουργικές μονάδες έχουν πολλές πιθανές υλοποιήσεις βιβλιοθήκης.
- ✓ Κάθε υλοποίηση έχει διαφορετικά χαρακτηριστικά area/delay.

  Παράδειγμα είναι ο αθροιστής ριπής και πρόβλεψης κρατουμένου

- ✓ Ο scheduler έχει να επιλέξει και τις μονάδες που θα χρησιμοποιήσει.
- ✓ Συνήθως επιλέγουμε γρήγορες μονάδες για το κρίσιμο μονοπάτι και αργές (αλλά φθηνές) για τα υπόλοιπα.

Ρεαλιστικές Περιγραφές.

- ✓ Θεωρήσαμε ότι έχουμε κώδικα χωρίς διακλαδώσεις (straight line).
- ✓ Οι δομές συνθήκης και επανάληψης χρησιμοποιούνται πολύ συχνά.

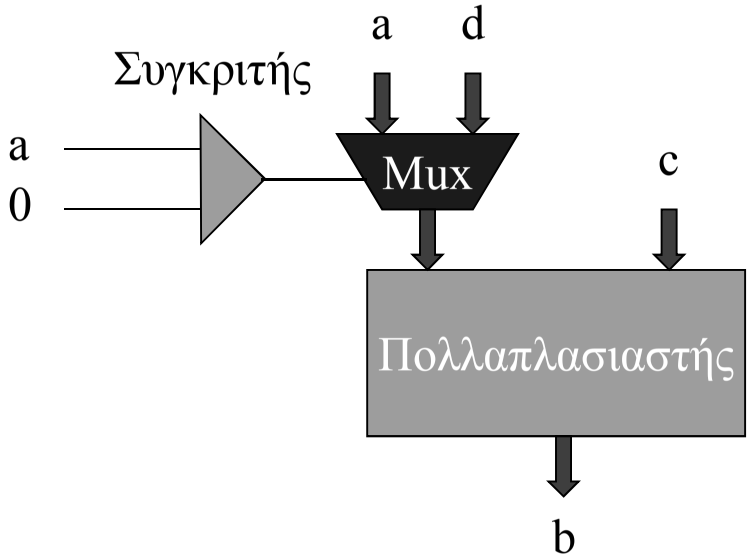
Δομές Συνθήκης

- ✓ Αντιστοιχούν στις εντολές if-case.
- ✓ Οδηγούν σε πολλαπλές διακλαδώσεις αμοιβαία αποκλειόμενες.
- ✓ Οι αποδοτικοί αλγόριθμοι διαμοιράζουν resources ανάμεσα στις αμοιβαία αποκλειόμενες λειτουργίες.

Ρεαλιστικές Περιγραφές.

Παράδειγμα: ο παρακάτω κώδικας απαιτεί ένα μόνο πολλαπλασιαστή και ένα βήμα εκτέλεσης:

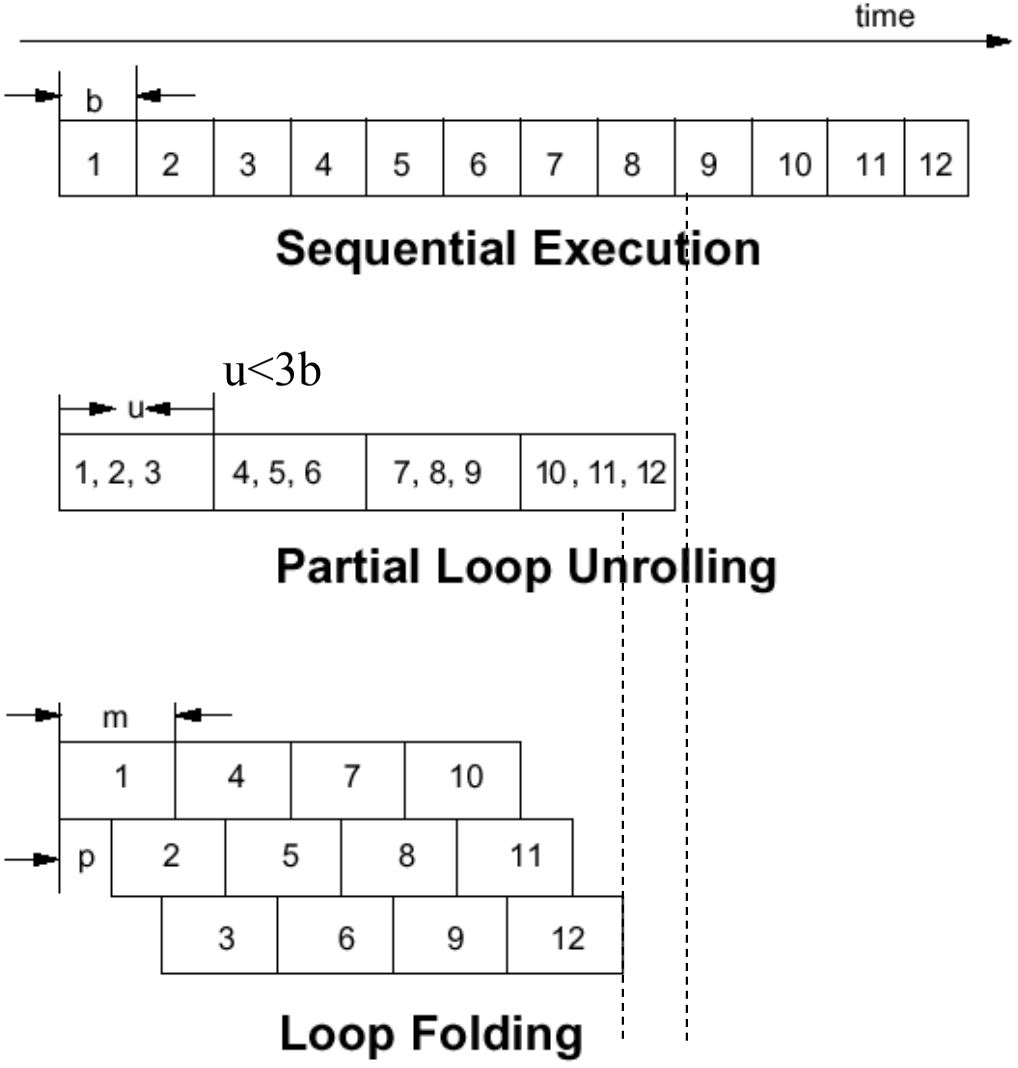
*if (a>0) then b:=a*c else b:=d*c endif*



Ρεαλιστικές Περιγραφές.

Δομές Επανάληψης

- ✓ Σε δομές επανάληψης μπορούμε να εισάγουμε αρκετό παραλληλισμό.
- ✓ Μπορούμε να εκτελούμε πολλαπλές επαναλήψεις του loop ταυτόχρονα.



Εσωτερικός παραλληλισμός με pipelining

Scheduling για Pipelined κυκλώματα

- ✓ Τα pipelined κυκλώματα έχουν ένα μοντέλο ακολουθιακού γράφου και έναν ρυθμό δεδομένων.
- ✓ **Structural Pipeline**: Ένα είδος pipeline είναι η χρήση pipelined resources.
- ✓ **Functional Pipeline**: Άλλο είδος pipeline είναι η χρήση non-pipelined resources και η επίτευξη του pipelining με το scheduling του γράφου.

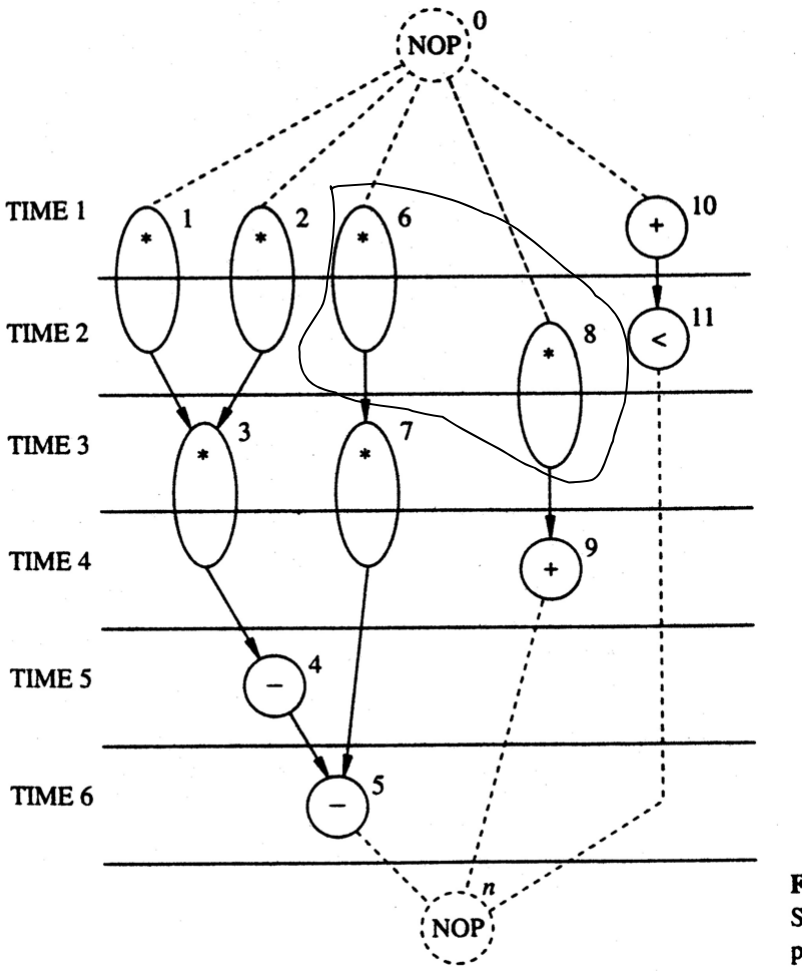
A. Structural Pipeline

- ✓ **Data introduction intervals**: τα διαστήματα που καταναλώνουν και παράγουν δεδομένα οι pipelined resources (θεωρούνται σταθερά).
- ✓ Οι pipelined resources μπορούν να διαμοιραστούν ακόμη και όταν οι λειτουργίες είναι επικαλυπτόμενες. Υπάρχουν δύο προϋποθέσεις:

  Α. Δεν υπάρχει εξάρτηση δεδομένων μεταξύ των λειτουργιών

  Β. Οι λειτουργίες δεν ξεκινούν στο ίδιο βήμα χρόνου

Scheduling για Pipelined κυκλώματα



Για Pipeline Resources:

α_1=3 πολλαπλ, α_2=1 ALU

$t_1 = 2$, $t_2 = 1$,

πολλαπλ=pipelined με unit interval

Scheduling για Pipelined κυκλώματα