Προηγμένα Θέματα Θεωρητικής Πληροφορικής

Χρονοπρογραμματισμός κώδικα και βελτιστοποιήσεις εξαρτημένες από την αρχιτεκτονική

Νικόλαος Καββαδίας
nkavv@uop.gr

12 Μαΐου 2010

Βελτιστοποιήσεις εξαρτημένες από την αρχιτεκτονική

- Χρονοπρογραμματισμός κώδικα: δρομολόγηση των λειτουργιών (εντολών) του επεξεργαστή σε χρονοθυρίδες δεσμεύοντας αντίστοιχες λειτουργικές μονάδες με στόχο τη βελτίωση των επιδόσεων εκτέλεσης του προγράμματος
- Τεχνικές
  - Χρονοπρογραμματισμός χωρίς περιορισμούς (ASAP, ALAP)
  - Χρονοπρογραμματισμός με περιορισμούς (όπως χρονοπρογραμματισμός λίστας: list scheduling)
  - Ευριστικές τεχνικές: με ακέραιο γραμμικό προγραμματισμό, προσομοιωμένη ανόπτηση, γενετικούς αλγορίθμους
- Βελτιστοποίηση κλειδαρότρυπας (peephole optimization): βελτιστοποίηση με αντικατάσταση ομάδας εντολών βάσει δοθέντος κανόνα εξετάζοντας μικρό τμήμα κώδικα
- Υπερβελτιστοποίηση (superoptimization): παραγωγή του βέλτιστου κώδικα χαμηλού επιπέδου με εξαντλητική εξέταση όλων των πιθανών περιπτώσεων

Χρονοπρογραμματισμός κώδικα (code scheduling) [Aho, 2008, (μετφρ. Ελληνικά)]

- Κατά το χρονοπρογραμματισμό, κάθε λειτουργία ανατίθεται σε ένα μοναδικό βήμα ελέγχου
- Κατηγοριοποίηση των αντίστοιχων τεχνικών
  - χωρίς περιορισμό (UCS: unconstrained scheduling)
    - δεν τίθενται εξωτερικοί περιορισμοί
    - λύνεται με την τεχνική ASAP (As Soon As Possible) και ALAP (As Late As Possible) όπου κάθε λειτουργία αντιστοιχίζεται στο πρώτο ή στο τελευταίο βήμα ελέγχου που μπορεί να ανατεθεί, αντίστοιχα
  - υπό περιορισμό πόρων (RCS: Resource-Constrained Scheduling)
    - χρονοπρογραμματισμός λίστας
  - υπό περιορισμό χρόνου (TCS: Time-Constrained Scheduling)
    - αλγόριθμος χρονοπρογραμματισμού κατευθυνόμενου από δύναμη (force-directed scheduling) ο οποίος προσπαθεί να ισοσταθμίσει τις χρήσεις των υφιστάμενων πόρων
  - υπό περιορισμό πόρων και χρόνου
    - Ευριστικές (heuristic) τεχνικές

Παραλληλία επιπέδου εντολών (ILP: Instruction-Level Parallelism)

- Σήμερα, οι περισσότερες αρχιτεκτονικές επεξεργαστή είναι ικανές για την ταυτόχρονη εκτέλεση περισσότερων της μίας εντολών
- $IPC \geq 1$ (IPC: Instructions per Clock Cycle)
  - αρχιτεκτονικές πολλαπλής έκδοσης εντολών (multiple-issue architectures): εντολές ανακαλούνται από τη μνήμη, αποκωδικοποιούνται και γίνονται διαθέσιμες σε λειτουργικές μονάδες του επεξεργαστή, ταυτόχρονα
  - νέες εντολές εκδίδονται καθόσον οι προηγηθείσες εντολές βρίσκονται ακόμη σε εξέλιξη (αρχιτεκτονικές με διοχέτευση)
  - συνδυασμός των παραπάνω τεχνικών
- Παράλληλες αρχιτεκτονικές: διαφοροποίηση στον τρόπο έκδοσης εντολών
  - στατικός καθορισμός από το μεταγλωττιστή: (VLIW: Very-Long Instruction Word) αρχιτεκτονικές
  - δυναμικός καθορισμός από το υλικό: υπερβαθμωτές (superscalar) αρχιτεκτονικές

Εξαρτήσεις εντολών (instruction dependencies)

- Εξάρτηση εντολής: μία εντολή i_2 εξαρτάται από την εντολή i_1 όταν δεν είναι εφικτό να εκτελεστεί η i_2 πριν την i_1 χωρίς τη μεταβολή της συμπεριφοράς του προγράμματος
- Εξάρτηση δεδομένων: η εντολή i_2 χρειάζεται τουλάχιστον μία τιμή που υπολογίζεται από την i_1
- Είδη εξάρτησης δεδομένων
  - πραγματική εξάρτηση τύπου RAW (Read After Write): η i_2 διαβάζει μία τιμή η οποία γράφεται από την i_1
  - αντι-εξάρτηση τύπου WAR (Write After Read): η i_2 γράφει μία τιμή η οποία διαβάζεται από την i_1
  - αντι-εξάρτηση τύπου WAW (Write After Write): η i_2 γράφει μία τιμή η οποία γράφεται από την i_1
- Οι αντι-εξαρτήσεις εμφανίζονται λόγω της εγγραφής στην ίδια θέση αποθήκευσης (π.χ. στον ίδιο καταχωρητή). Μπορούν να απομακρυνθούν με επανονομασία των καταχωρητών

Η αρχή της διοχέτευσης εντολών

- Διαχωρισμός μιας λειτουργίας σε μία ακολουθία σταδίων, ιδανικά με τον ίδιο χρόνο καθυστέρησης
- Επιτρέπει την εκτέλεση λειτουργιών με επικάλυψη
- Τρόπος εκμετάλλευσης παραλληλίας στο χρόνο, ενώ η αρχή VLIW εκμεταλλεύεται την παραλληλία στο χώρο
- Διοχέτευση τόσο εντολών όσο και εντός της ίδιας λειτουργικής μονάδας (micro-pipelining)

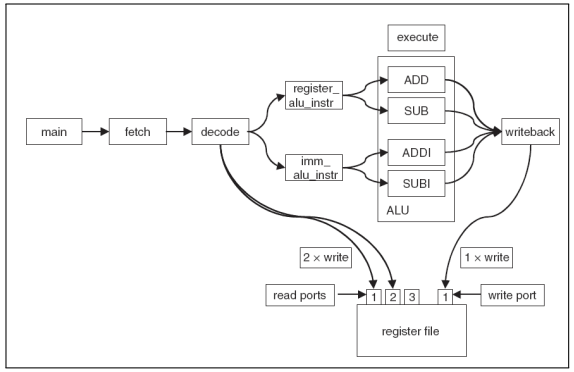
Pipeline stage	cycle 1	2	3	4	5	6	7
1	B_1	B_2	B_3	B_4			
2		B_1	B_2	B_3	B_4		
3			B_1	B_2	B_3	B_4	
4				B_1	B_2	B_3	B_4

Χαρακτηριστικά μιας αρχιτεκτονικής διοχέτευσης εντολών

- Περισσότερες της μιας εντολές βρίσκονται ταυτόχρονα σε διαφορετικά στάδια εκτέλεσης
- Πιθανή οργάνωση μιας αρχιτεκτονικής διοχέτευσης
  - Ανάκληση εντολής
  - Αποκωδικοποίηση εντολής
  - Ανάκληση ορισμάτων
  - Εκτέλεση εντολής με ενδεχόμενη προσπέλαση μνήμης
  - Εγγραφή ορισμάτων στο αρχείο καταχωρητών
- Κίνδυνοι (hazards) σε μία αρχιτεκτονική διοχέτευσης
  - Data hazards: λόγω πραγματικών εξαρτήσεων, π.χ. ορίσματα ανάγνωσης που δεν είναι διαθέσιμα
  - Structural hazards: συγκρούσεις για τη δέσμευση λειτουργικών μονάδων (δύο ή περισσότερες εντολές χρειάζονται την ίδια μονάδα)
  - Control hazards: άλματα υπό συνθήκη, των οποίων η τιμή της συνθήκης δεν έχει ακόμη υπολογιστεί

Επεξεργαστές RISC με αρχιτεκτονική διοχέτευσης

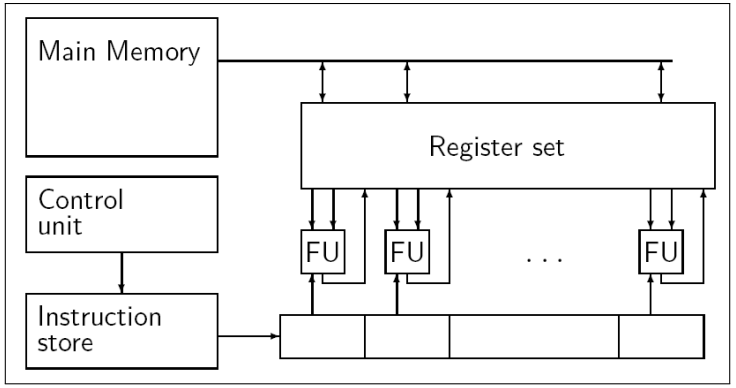
- Περιορισμένο ρεπερτόριο εντολών με λίγους τρόπους διευθυνσιοδότησης, σταθερό μήκος εντολής, ομογενής αρχιτεκτονική καταχωρητών
- Εντολές πάνω σε καταχωρητές, ξεχωριστές εντολές για φόρτωση και αποθήκευση από τη μνήμη



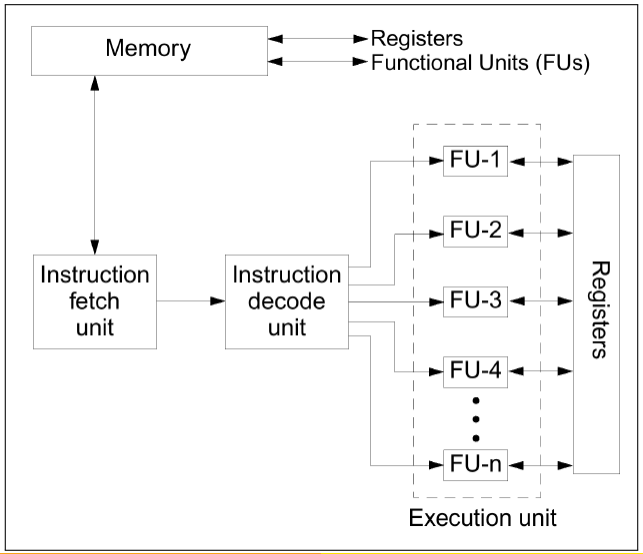
Η αρχιτεκτονική VLIW (1)

- Πολλαπλές λειτουργικές μονάδες (FU: Functional Unit), ιδανικά του ίδιου τύπου αν και αυτό δεν ισχύει στην πράξη (αριθμητική-λογική μονάδα, μονάδα φόρτωσης-αποθήκευσης, πολλαπλασιαστής, διαιρέτης, μονάδα διακλάδωσης)
- Κάθε εντολή απαρτίζεται από μία ομάδα λειτουργιών με κάθε μία από αυτές να εξυπηρετείται από αντίστοιχη λειτουργική μονάδα
- Οι FUs συνδέονται με συστοιχίες καταχωρητών. Η απαίτηση για πολλαπλές θύρες ανάγνωσης/εγγραφής επιβάλλει πολλές φορές το διαμερισμό του αρχείου καταχωρητών

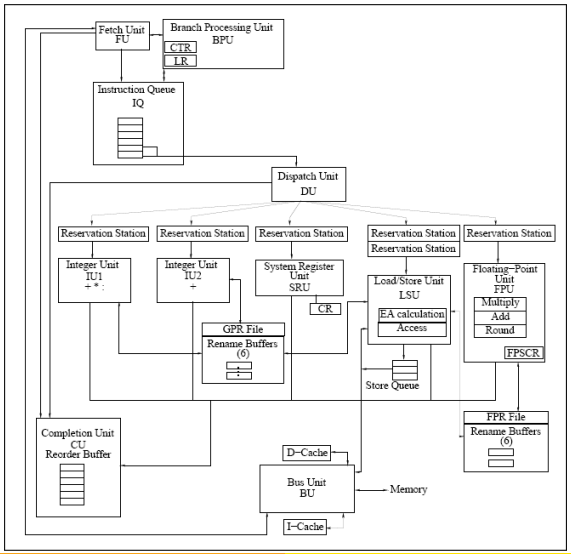
Η αρχιτεκτονική VLIW (2)



Αρχιτεκτονικό περίγραμμα μιας απλής VLIW αρχιτεκτονικής



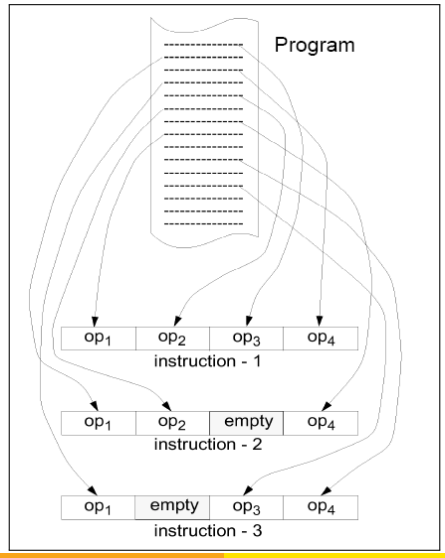
Παράδειγμα RISC/VLIW μικροαρχιτεκτονικής: Επεξεργαστής PowerPC



Αρχές στατικού και δυναμικού χρονοπρογραμματισμού

Στατικός χρονοπρογραμματισμός	**Δυναμικός χρονοπρογραμματισμός**
Σε χρόνο μεταγλώττισης	Επίλυση στο υλικό
Εφικτή η καθολική εμβέλεια (σε όλο το πρόγραμμα)	Τοπική εμβέλεια
Σε κάθε βήμα: έλεγχος των εξαρτήσεων των υποψηφίων για δρομολόγηση εντολών με εντολές που έχουν δρομολογηθεί προηγουμένως. Δρομολόγηση των κατάλληλων εντολών με την πρέπουσα καθυστέρηση	Με ανάλυση των τοπικών εξαρτήσεων: έλεγχος των εξαρτήσεων των υποψηφίων για δρομολόγηση εντολών με εντολές που ήδη εκτελούνται ή υπόκεινται σε προγραμματισμένη καθυστέρηση. Αποδέσμευση για δρομολόγηση ή συγκράτηση των υποψηφίων εντολών
Η εμβέλεια συνήθως είναι: βασικό μπλοκ, ακολουθία βασικών μπλοκ (π.χ. από ίχνος εκτέλεσης προγράμματος), εσώτεροι βρόχοι	Η εμβέλεια συνήθως είναι: μικρό παράθυρο μέχρι 10-12 εντολές, ώστε οι εξαρτήσεις να είναι επιλύσιμες δυναμικά στο υλικό

Δρομολόγηση εντολών με στατικό χρονοπρογραμματισμό σε επεξεργαστή VLIW



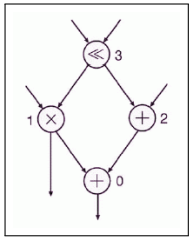
Εξαρτήσεις δεδομένων σε βασικά μπλοκ

- Οι εξαρτήσεις δεδομένων καταγράφονται με τη μορφή γράφου εξάρτησης δεδομένων (DDG: Data-Dependence Graph) ο οποίος είναι DAG
- Ιδιότητες:
  - όλες οι άμεσες εξαρτήσεις αναπαρίστανται από ακμές
  - η εξάρτηση έχει μεταβατική ιδιότητα, όχι ρητά εκπεφρασμένη

Αλγόριθμος ακολουθιακής δρομολόγησης με τοπολογική κατάταξη (topological sort)

- Τοπολογική κατάταξη (topological sorting): Οι κορυφές ενός κατευθυνόμενου ακυκλικού γράφου $G(V, E)$ διατάσσονται έτσι ώστε αν ο G περιέχει ακμή $(u, v) \in E$ τότε ο u να εμφανίζεται μετά τον v στην κατάταξη
- Γενικά υπάρχουν περισσότερες από μία έγκυρες τοπολογικές κατατάξεις ενός DAG
- Στο χρονοπρόγραμμα: μία λειτουργία σε κάθε βήμα ελέγχου

```
SEQUENTIAL(G(V, E))
Topological sort on G;
Schedule u_0 by setting t_0(S) = 0;
repeat
{
   Schedule u_i in increasing topological order;
}
until (u_n is scheduled);
return (t_i(S) for all i);
```



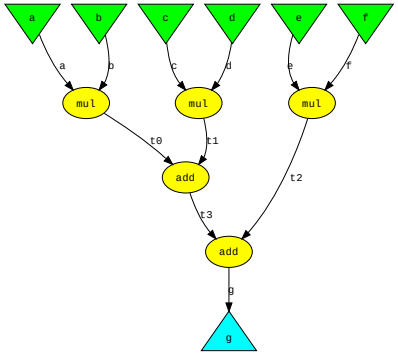
Παράδειγμα 1

- Η σύνθετη έκφραση

```
G = A*B + C*D + E*F
```

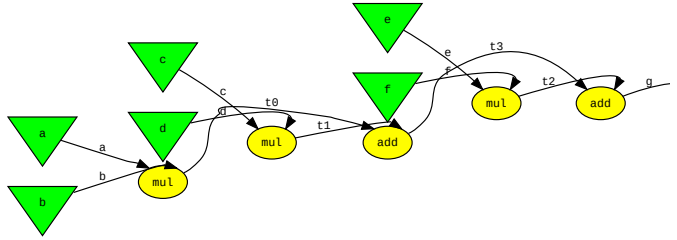
- Ενδεικτικός κώδικας τριών διευθύνσεων

```
t0 = A * B;
t1 = C * D;
t2 = E * F;
t3 = t0 + t1;
G  = t2 + t3;
```



Παράδειγμα 1: Ακολουθιακή δρομολόγηση

Τοπολογική κατάταξη



Παραγόμενος κώδικας συμβολομεταφραστή (υποθέτουμε ότι οι μεταβλητές `a, b, c, d, e, f` βρίσκονται ήδη σε καταχωρητές)

```
L1:
  mul  Rt0, Ra, Rb
  mul  Rt1, Rc, Rd
  add  Rt3, Rt0, Rt1
  mul  Rt2, Re, Rf
  add  Rg, Rt3, Rt2
```

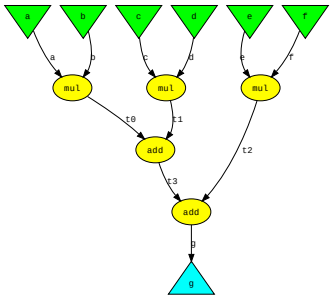

Ο αλγόριθμος ASAP (As Soon As Possible)

- Επιλύει το πρόβλημα του χρονοπρογραμματισμού χωρίς περιορισμούς
- Υποτίθεται ότι το δοσμένο DAG είναι πολικός γράφος (polar graph)

ASAP($G(V,E)$)
Schedule u_0 by setting $t_0(S) = 0$;
repeat
{
  Select a vertex u_i whose predecessors are all scheduled;
  Schedule u_i setting $t_i(S) = max_{j:(u_j,u_i)\in E}(t_j(S) + d_j)$;
}
until (u_n is scheduled);
return ($t_i(S)$ for all i);

Παράδειγμα 1: Χρονοπρογραμματισμός ASAP

Χρονοπρόγραμμα ASAP



Παραγόμενος κώδικας συμβολομεταφραστή (οι διπλές κάθετες μπάρες υποδηλώνουν παράλληλη εκτέλεση)

```
L1:
  mul  Rt0, Ra, Rb || mul  Rt1, Rc, Rd || mul  Rt2, Re, Rf
  add  Rt3, Rt0, Rt1
  add  Rg, Rt3, Rt2
```

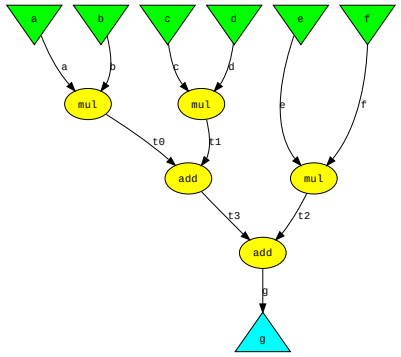
Ο αλγόριθμος ALAP (As Late As Possible)

- Επιλύει το πρόβλημα του χρονοπρογραμματισμού χωρίς περιορισμούς και υπό περιορισμό χρόνου
- Χρησιμοποιεί την έννοια του ορίου καθυστέρησης (latency bound): $\overline{\lambda} = t_n(s) - t_0(s)$

ALAP($G(V, E)$, $\overline{\lambda}$)
Schedule u_n by setting $t_n(L) = \overline{\lambda} + 1$;
repeat
{
  Select a vertex u_i whose successors are all scheduled;
  Schedule u_i setting $t_i(L) = min_{j:(u_i,u_j)\in E}(t_j(L) - d_i)$;
}
until (u_0 is scheduled);
return ($t_i(L)$ for all i);

Παράδειγμα 1: Χρονοπρογραμματισμός ALAP

Χρονοπρόγραμμα ALAP



Παραγόμενος κώδικας συμβολομεταφραστή

```
L1:
  mul  Rt0, Ra, Rb || mul  Rt1, Rc, Rd
  mul  Rt2, Re, Rf || add  Rt3, Rt0, Rt1
  add  Rg, Rt3, Rt2
```


Η έννοια της ευκινησίας λειτουργίας (operation mobility)

- Η ευκινησία μιας λειτουργίας ορίζεται ως η διαφορά μεταξύ των χρόνων έναρξης (start times) όπως αυτοί υπολογίζονται από το χρονοπρογραμματισμό με τους αλγορίθμους ASAP και ALAP
- Δίνεται από τη σχέση: $\mu_i = t_i(L) - t_i(S)$ για τη λειτουργία i
- Μηδενική ευκινησία υπονοεί ότι η συγκεκριμένη λειτουργία μπορεί να ξεκινήσει μόνο στο συγκεκριμένο χρονικό σημείο προκειμένου να ικανοποιούνται οι απαιτήσεις του περιορισμού χρόνου
- Μη μηδενική ευκινησία περιγράφει τα διαθέσιμα χρονικά σημεία (βήματα ελέγχου του χρονοπρογράμματος) στα οποία μπορεί να ξεκινήσει η αντίστοιχη λειτουργία

Ο αλγόριθμος χρονοπρογραμματισμού λίστας (1)

- Επαναληπτικός αλγόριθμος, σε κάθε επανάληψη του οποίου επιλέγεται η καταλληλότερη λειτουργία από μία δεξαμενή αδέσμευτων λειτουργιών ώστε αυτή να ανατεθεί στο πρώτο βήμα ελέγχου για το οποίο δεν παραβιάζονται οι περιορισμοί πόρων
- Οι λειτουργίες κρίνονται με βάση μία συνάρτηση προτεραιότητας, η οποία ανταποκρίνεται στις απαιτήσεις του προγραμματιστή
- Στον αλγόριθμο διατηρούνται δύο λίστες:
  - Λίστα **ready**: διαθέσιμες εντολές που μπορούν να δρομολογηθούν κατά σειρά προτεραιότητας
  - Λίστα **active**: εντολές που βρίσκονται σε εκτέλεση
- Σε κάθε βήμα, η εντολή υψηλότερης προτεραιότητας από τη λίστα ready δρομολογείται και μετακινείται στην active όπου μένει για χρόνο εκτέλεσης ίσο με τον απαιτούμενο αριθμό κύκλων μηχανής

Ο αλγόριθμος χρονοπρογραμματισμού λίστας (2)

- Συναρτήσεις προτεραιότητας
  - Η πιο κοινή συνάρτηση προτεραιότητας είναι το μήκος του μακρύτερου μονοπατιού από τον τρέχοντα κόμβο-εντολή μέχρι τον εικονικό κόμβο-ρίζα του ακυκλικού γράφου εξαρτήσεων (data-dependence DAG)
  - Άλλες συναρτήσεις προτεραιότητας είναι δυνατόν να χρησιμοποιηθούν όπως για παράδειγμα ο αριθμός των άμεσων διάδοχων κόμβων για την τρέχουσα εντολή (number of immediate successor nodes)

Ο αλγόριθμος χρονοπρογραμματισμού λίστας (3)

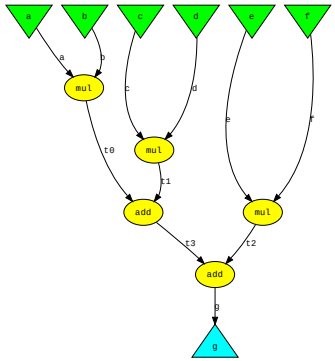
```
cycle ← 1;
ready ← { leaf nodes of DDG };
active ← ∅;

while (ready ∪ active ≠ ∅)
   if (ready ≠ ∅) then
      remove an operation from ready;
      Schedule(operation) ← cycle;
      active ← active ∪ operation;
   endif
   cycle ← cycle + 1;
   update the ready queue;
endwhile
```


Παράδειγμα 1: Χρονοπρογραμματισμός λίστας με περιορισμούς πόρων

☞Θεωρούμε ότι υπάρχει διαθεσιμότητα ενός πολλαπλασιαστή και ενός αθροιστή

Τελικό χρονοπρόγραμμα λίστας



Παραγόμενος κώδικας συμβολομεταφραστή

```
L1:
  mul  Rt0, Ra, Rb
  mul  Rt1, Rc, Rd
  mul  Rt2, Re, Rf || add  Rt3, Rt0, Rt1
  add  Rg, Rt3, Rt2
```


Βελτιστοποίηση κλειδαρότρυπας [McKeeman, 1965]

- Γίνεται μετά τη μεταγλώττιση, δηλαδή τη μετατροπή της ενδιάμεσης αναπαράστασης σε τελικό κώδικα
- Η βασική ιδέα
  - Ανακάλυψη τοπικών βελτιστοποιήσεων εξετάζοντας κάθε φορά ένα παράθυρο κώδικα
  - Αντικατάσταση του τρέχοντος μοτίβου εντολών (ακολουθία ή υπογράφος εντολών) εφόσον υπάρχει αντίστοιχος κανόνας αντικατάστασης
- Κανόνες αντικατάστασης

Έκφραση	Αντικατάσταση
`mov r1, r1`	–
`shiftr r1, r2, 0`	–
`add r1, r1, constant` `lw r2, 0(r1)`	`lw r2, constant(r1)`
`neg r1, r2`	`sub r1, $zero, r2`

`abs r10, r11`	`sra $at, r11, 31`
	`xor r10, r11, $at`
	`subu r10, r10, $at`

Το εργαλείο copt [copt]

- Απλό εργαλείο αντικατάστασης μοτίβων (patterns) από λεκτικές μονάδες σύμφωνα με κανόνα αντικατάστασης
- Όταν ένας κανόνας αναγνωριστεί, εφαρμόζεται πάντα
- Αντιμετωπίζει το πρόγραμμα χαμηλού επιπέδου ως κείμενο
- Μορφή κανόνων στο copt

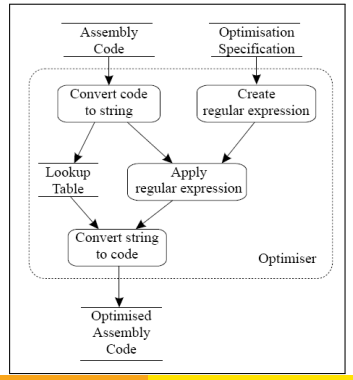
```
<pattern for input line 1>
...
<pattern for input line n>
=
<pattern for output line 1>
...
<pattern for output line m>
<blank line>
```

- Παράδειγμα

```
abs %0, %1
=
sra $at, %1, 31
xor %0, %1, $at
subu %0, %0, $at
```


Το εργαλείο peep [Spinellis, 1999]

- Οι κανόνες ταύτισης και αντικατάστασης περιγράφονται σε δηλωτική μορφή
- Το πρόγραμμα θεωρείται σαν μία συμβολοσειρά
- Υποστηρίζει τη χρήση κανονικών εκφράσεων, με χαρακτήρες μπαλαντέρ (wildcards) σε μία αναζήτηση



Υπερβελτιστοποίηση [Massalin, 1987]

- Δοθέντος ενός τμήματος κώδικα και του συνόλου εντολών του στοχευόμενου επεξεργαστή, ο υπερβελτιστοποιητής παράγει το βέλτιστο δυνατό κώδικα
- Το πρόβλημα της γέννησης βέλτιστου κώδικα είναι δυσεπίλυτο και έτσι συνήθως η αναζήτηση γίνεται με εφαρμογή ωμής δύναμης (brute-force search) σε συνδυασμό με τεχνικές αποκοπής υποπεριπτώσεων που η περαιτέρω διερεύνησή τους δεν μπορεί να οδηγήσει σε βέλτιστο κώδικα
- Παραδείγματα εφαρμογής: συνάρτηση εξαγωγής προσήμου (signum function), απόλυτη τιμή ακεραίου, μέγιστο και ελάχιστο δύο ακεραίων, πολλαπλασιασμοί/διαιρέσεις με σταθερά
- Εργαλεία υπερβελτιστοποίησης
  - GNU superopt [superopt]
  - Aha superoptimizer [Aha!]

Το εργαλείο superopt (1)

- Χρησιμοποιεί πραγματικά σύνολα εντολών (PowerPC, SPARC) ή ένα γενικό σύνολο εντολών με εντολές όπως: `adc, add, and, copy, exchange, ior`
- Παράδειγμα: τμήμα κώδικα για την επιλογή υπό συνθήκη σε C

```
r = (unsigned int) (v0 != 0 ? v1 : v2);
```

- Κανόνας στο superopt

```
DEF_GOAL (SELECT, 3, "select", { r = (unsigned_word) v0 != 0 ? v1 : v2; })
```

- Εάν ο χρήστης ζητήσει όλες τις ακολουθίες με 5 ή λιγότερες εντολές, το superopt θα επιστρέψει 276 ακολουθίες εντολών που υλοποιούν τον κώδικα select, όπως:

```
1: r3:=sub(r2,r1)
r4:=add_co(r0,-1)
r5:=adc_cio(r5,r5)
r6:=and(r5,r3)
r7:=add_co(r6,r1)
```

Το εργαλείο superopt (2)

- Πραγματοποιεί εξαντλητικό έλεγχο για την εύρεση της βραχύτερης ακολουθίας
- Δέχεται ως είσοδο την έκφραση προς βελτιστοποίηση, καθώς και το μέγιστο αριθμό επιτρεπόμενων εντολών στην παραγόμενη ακολουθία
- Δεν μπορεί να βελτιστοποιήσει κώδικα παράγοντας μεγάλες ακολουθίες καθώς η πολυπλοκότητα του χρησιμοποιούμενου αλγορίθμου είναι εκθετική: $O(m \cdot n^{2n})$ όπου m είναι ο αριθμός των εντολών της στοχευόμενης αρχιτεκτονικής και n είναι η ακολουθία ελάχιστου μήκους για το τμήμα κώδικα υπό βελτιστοποίηση
- Το πρακτικό μέγεθος ακολουθίας εντολών κυμαίνεται μεταξύ 2 και 7 εντολών
- Η υπερβελτιστοποίηση είναι χρήσιμη μόνο για ορισμένα τμήματα κώδικα που εμφανίζονται σε εσώτερους βρόχους

Αναφορές του μαθήματος I

- A. V. Aho, R. Sethi, and J. D. Ullman, *Μεταγλωττιστές: Αρχές, Τεχνικές και Εργαλεία*, με την επιμέλεια των: Άγγελος Σπ. Βώρος και Νικόλαος Σπ. Βώρος και Κων/νος Γ. Μασσέλος, **κεφάλαια 8.7, 10, 10.1–10.3**, Εκδόσεις Νέων Τεχνολογιών, 2008. Website for the English version: `http://dragonbook.stanford.edu`
- W. M. McKeeman, "Peephole optimization," *Communications of the ACM*, vol. 8, no. 7, pp. 443–444, July 1965.
- H. Massalin, "Superoptimizer: A look at the smallest program," in *Proceedings of the Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS II)*, January 1987, pp. 122–126.
- D. Spinellis, "Declarative peephole optimization using string pattern matching," *ACM SIGPLAN Notices*, vol. 34, no. 2, pp. 47–51, February 1999.
- A simple retargetable peephole optimizer. [Online]. Available: `ftp://ftp.cs.princeton.edu/pub/packages/lcc/contrib/copt.shar`

Αναφορές του μαθήματος II

- The GNU superoptimizer. [Online]. Available: `ftp://ftp.gnu.org/pub/gnu/superopt/`
- The Aha! superoptimizer. [Online]. Available: `http://www.hackersdelight.org`