# Ενσωματωμένα Συστήματα

**Ενότητα 12:** Σχεδίαση και Ανάλυση Προγράμματος.

Δρ. Μηνάς Δασυγένης
*mdasyg@ieee.org*
Εργαστήριο Ψηφιακών Συστημάτων και Αρχιτεκτονικής Υπολογιστών
*http://arch.icte.uowm.gr/mdasyg*

# Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

# Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ψηφιακά Μαθήματα στο Πανεπιστήμιο Δυτικής Μακεδονίας**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

# Σκοπός ενότητας

- Η κατανόηση της διαδικασίας σχεδίασης και υλοποίησης μιας εφαρμογής ενσωματωμένων συστημάτων.
- Η κατανόηση των τρόπων βελτιστοποίησης της ενέργειας και της απόδοσης μιας ενσωματωμένης εφαρμογής.

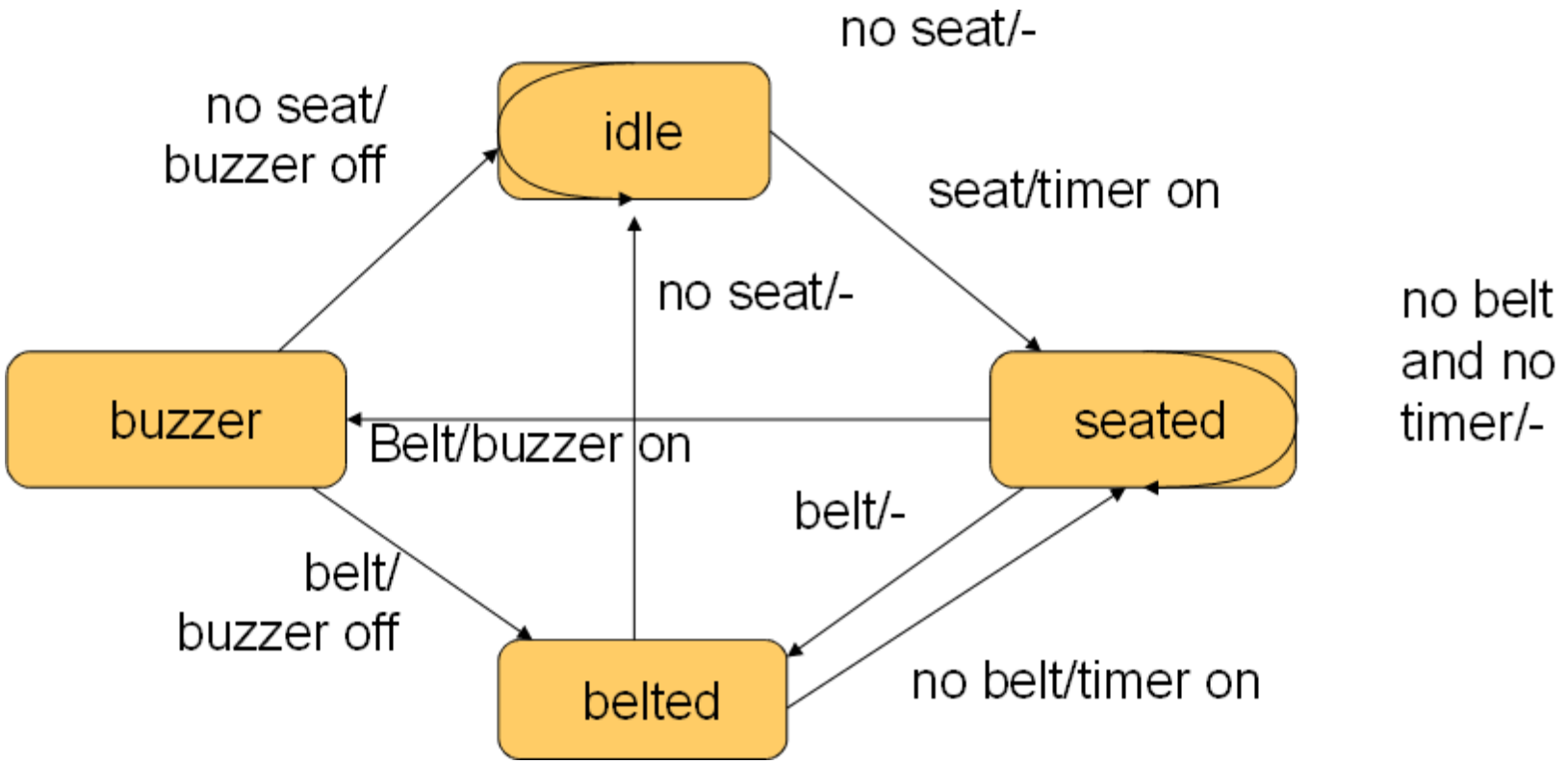
# Σχεδίαση Προγράμματος

# Μηχανή καταστάσεων λογισμικού

- Η μηχανή καταστάσεων κρατάει την τρέχουσα κατάσταση σαν μεταβλητή, και αλλάζει κατάσταση βασιζόμενη στις εισόδους.

- Χρησιμοποιεί:
  - κώδικα συνεχών ελέγχων .
  - συστήματα που αλληλεπιδρούν με το περιβάλλον.

# Παράδειγμα μηχανής καταστάσεων



# Υλοποίηση σε C

```
#define IDLE 0
#define SEATED 1
#define BELTED 2
#define BUZZER 3
switch (state) {
        case IDLE: if (seat) { state = SEATED; timer_on = TRUE; }
                break;
        case SEATED: if (belt) state = BELTED;
                        else if (timer) state = BUZZER;
                break;
        …
}
```

# Επεξεργασία σήματος και κυκλική περιοχή προσωρινής αποθήκευσης (1/3)

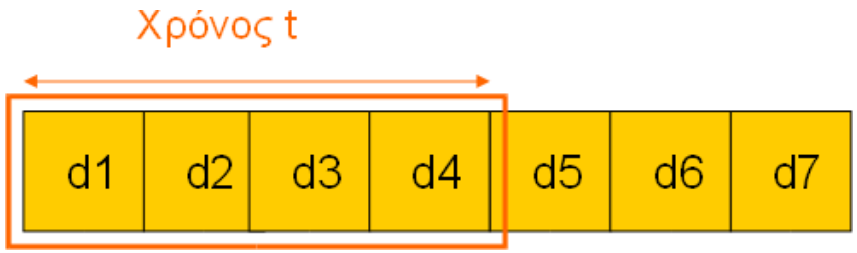
- Χρησιμοποιείται συνήθως στην επεξεργασία σήματος:
  - νέα δεδομένα συνεχώς καταφθάνουν,
  - κάθε δεδομένο έχει περιορισμένη διάρκεια ζωής.

- Χρήση μίας κυκλικής περιοχής προσωρινής αποθήκευσης για να αποθηκεύει τη ροή δεδομένων.

# Επεξεργασία σήματος και κυκλική περιοχή προσωρινής αποθήκευσης (2/3)

- Χρησιμοποιείται συνήθως στην επεξεργασία σήματος:
  - νέα δεδομένα συνεχώς καταφθάνουν,
  - κάθε δεδομένο έχει περιορισμένη διάρκεια ζωής.

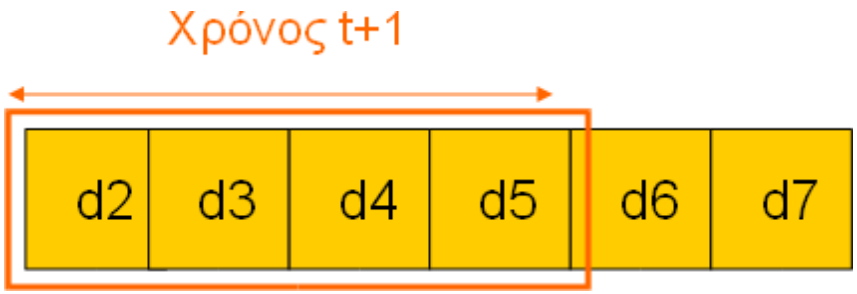
- Χρήση μίας κυκλικής περιοχής προσωρινής αποθήκευσης για να αποθηκεύει τη ροή δεδομένων.



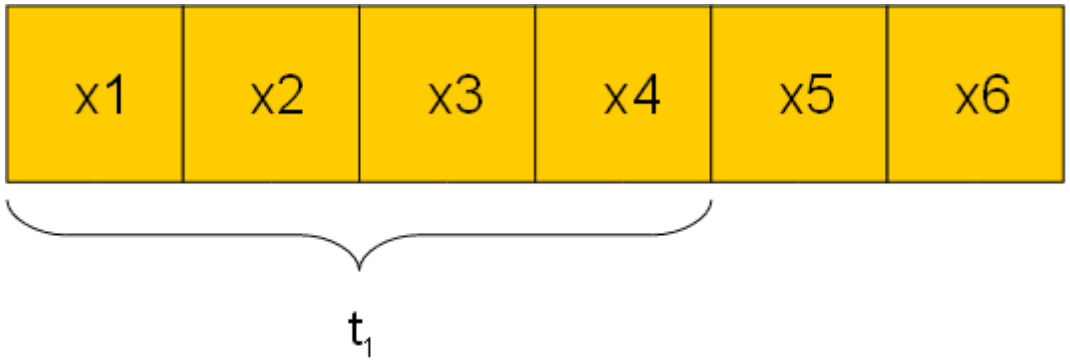
# Επεξεργασία σήματος και κυκλική περιοχή προσωρινής αποθήκευσης (3/3)

- Χρησιμοποιείται συνήθως στην επεξεργασία σήματος:
  - νέα δεδομένα συνεχώς καταφθάνουν,
  - κάθε δεδομένο έχει περιορισμένη διάρκεια ζωής.

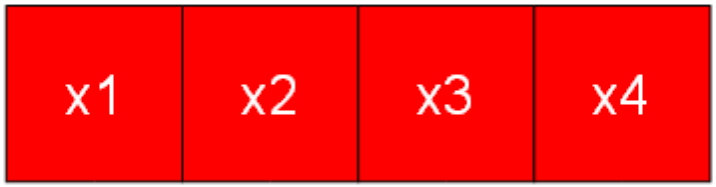
- Χρήση μίας κυκλικής περιοχής προσωρινής αποθήκευσης για να αποθηκεύει τη ροή δεδομένων.



# Κυκλική περιοχή προσωρινής αποθήκευσης (1/8)

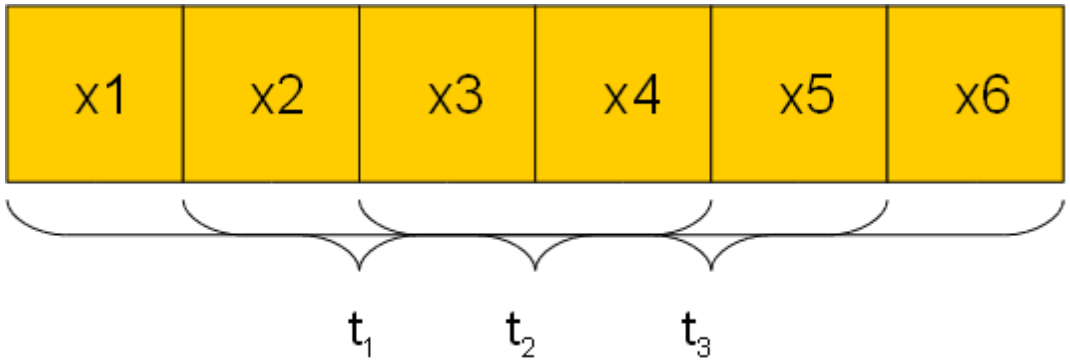



Ροή δεδομένων




Κυκλική περιοχή προσωρινής αποθήκευσης

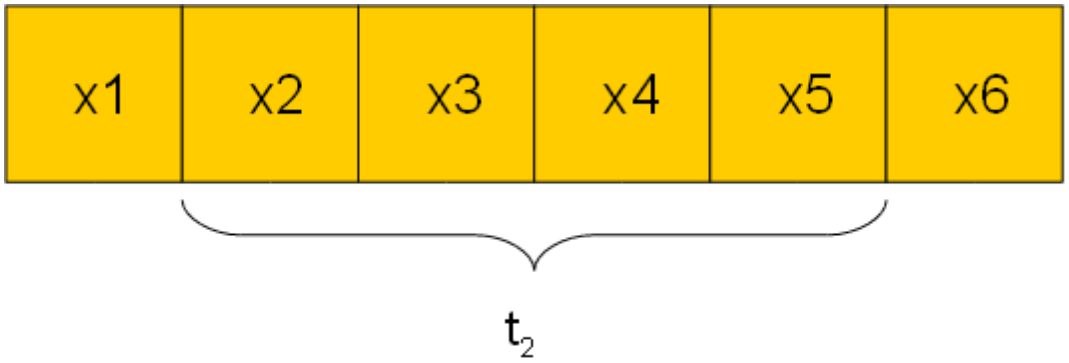
# Κυκλική περιοχή προσωρινής αποθήκευσης (2/8)



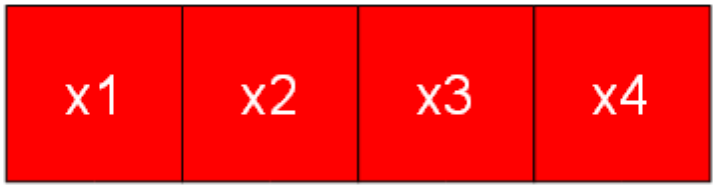

Ροή δεδομένων


Κυκλική περιοχή προσωρινής αποθήκευσης

# Κυκλική περιοχή προσωρινής αποθήκευσης (3/8)

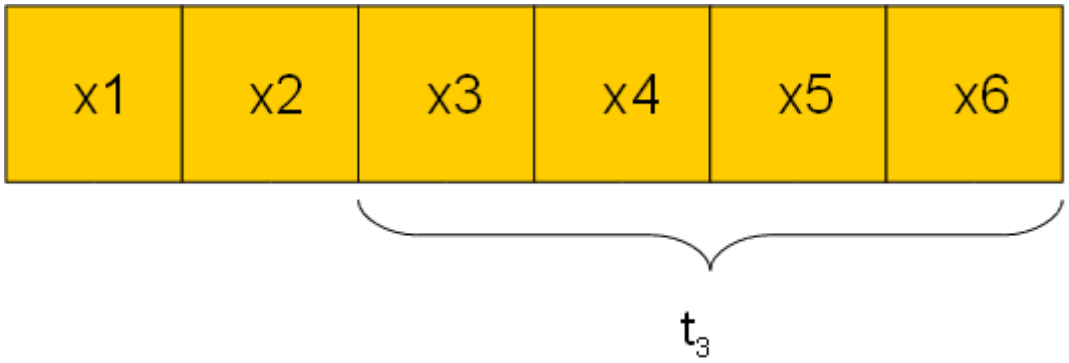



Ροή δεδομένων

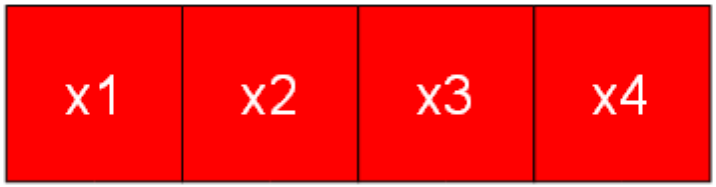



Κυκλική περιοχή προσωρινής αποθήκευσης

# Κυκλική περιοχή προσωρινής αποθήκευσης (4/8)



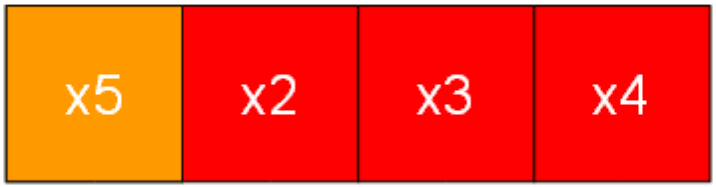

Ροή δεδομένων




Κυκλική περιοχή προσωρινής αποθήκευσης

# Κυκλική περιοχή προσωρινής αποθήκευσης (5/8)

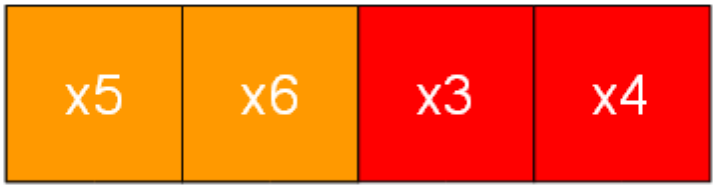

Ροή δεδομένων




Κυκλική περιοχή προσωρινής αποθήκευσης

# Κυκλική περιοχή προσωρινής αποθήκευσης (6/8)


Ροή δεδομένων




Κυκλική περιοχή προσωρινής αποθήκευσης

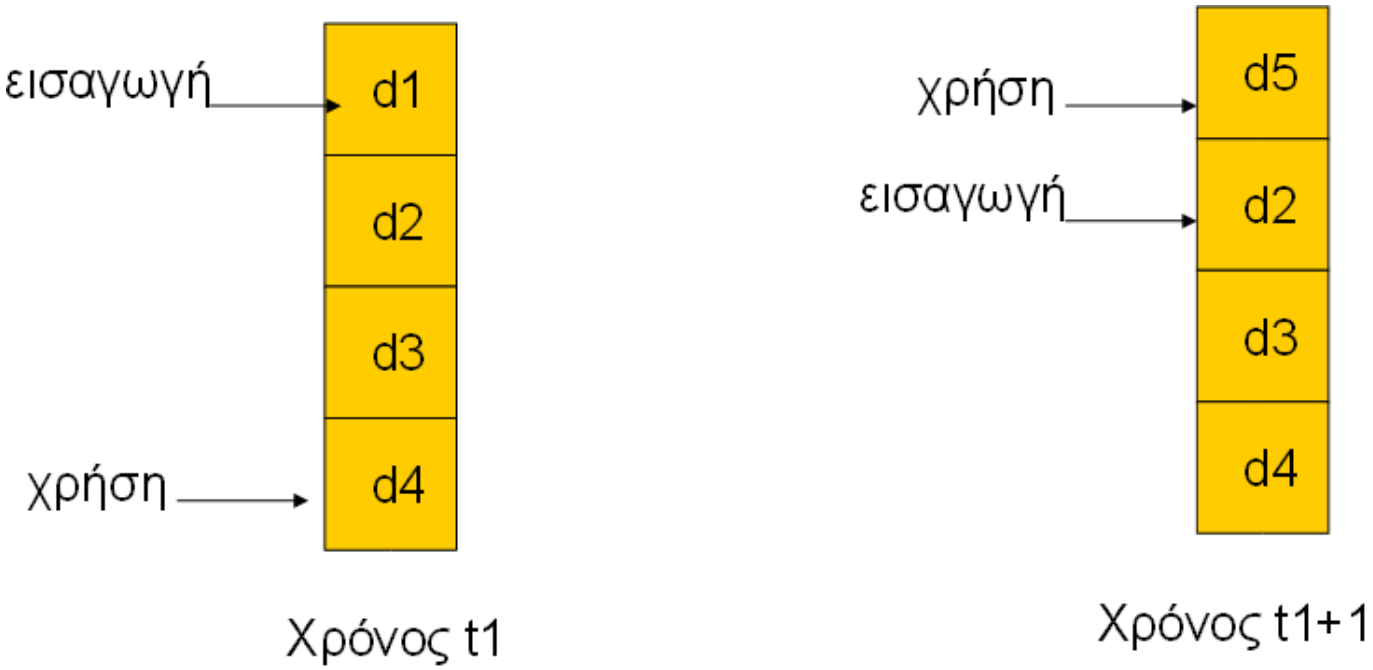
# Κυκλική περιοχή προσωρινής αποθήκευσης (7/8)


Ροή δεδομένων


Κυκλική περιοχή προσωρινής αποθήκευσης

# Κυκλικές περιοχές προσωρινής αποθήκευσης (8/8)

- Οι δείκτες δείχνουν στη θέση που θα τοποθετηθεί το επόμενο δείγμα, αντικαθιστώντας το παλαιότερο δείγμα:



# Υλοποίηση κυκλικής περιοχής προσωρινής αποθήκευσης: Φίλτρο FIR

```
int circ_buffer[N], circ_buffer_head = 0;
int c[N]; /* coefficients */
…
int ibuf, ic;
for (f=0, ibuff=circ_buff_head, ic=0;
        ic<N; ibuff=(ibuff==N-1?0:ibuff++), ic++)
f = f + c[ic]*circ_buffer[ibuf];
```

# Ουρές

- Ευέλικτη περιοχή προσωρινής αποθήκευσης: αποθηκεύει τα δεδομένα που καταφθάνουν ακανόνιστα.

# Ουρές βασιζόμενες σε περιοχές προσωρινής αποθήκευσης

```
#define Q_SIZE 32
#define Q_MAX (Q_SIZE-1)
int q[Q_MAX], head, tail;
void initialize_queue() { head = tail = 0; }
void enqueue(int val) {
        if (((tail+1)%Q_SIZE) == head) error();
        q[tail]=val;
        if (tail == Q_MAX) tail = 0;
else tail++;
}
```

```
int dequeue() {
        int returnval;
        if (head == tail) error();
        returnval = q[head];
        if (head == Q_MAX)
        head = 0;
                else head++;
        return returnval;
}
```

# Μοντέλα Προγραμμάτων

# Μοντέλα Προγραμμάτων

- Ο πηγαίος κώδικας δεν αποτελεί καλή αναπαράσταση για πρόγραμμα:
  - ασύντακτος
  - αφήνει αρκετή πληροφορία να εννοηθεί.
- Οι μεταγλωττιστές αντλούν ενδιάμεσες αναπαραστάσεις για να χειριστούν και να βελτιστοποιήσουν το πρόγραμμα.

# Γράφημα ροής δεδομένων (1/2)

- Γράφημα ροής δεδομένων (data flow graph): **DFG**
- Χωρίς συνθήκες ελέγχου.
- Βασικό μπλοκ μοντέλου: τμήμα κώδικα μόνο ένα σημείο εισόδου και ένα σημείο εξόδου.
- Περιγράφει τις ελάχιστες απαιτήσεις των εντολών για κάποια λειτουργία.

# Μορφή μονής ανάθεσης

```
x = a + b;
y = c - d;
z = x * y;
y = b + d;
```

Ένα βασικό μπλοκ στην C.

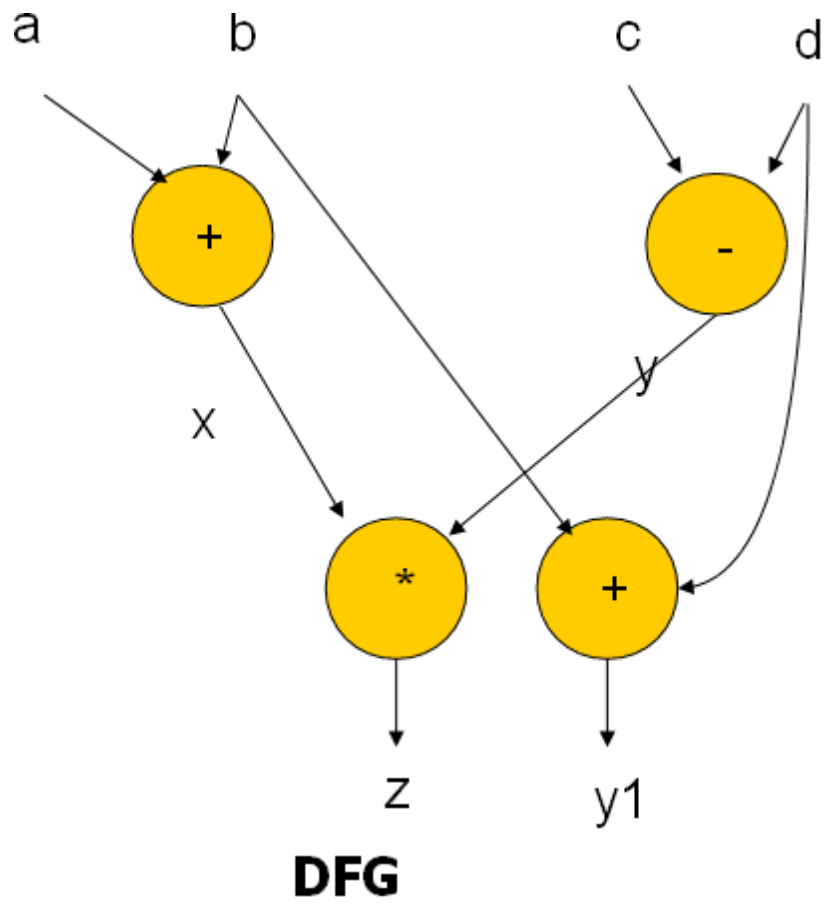
```
x = a + b;
y = c - d;
z = x * y;
y1 = b + d;
```

Το βασικό μπλοκ σε μορφή απλής (ή μονής) ανάθεσης (single assigment).

# Γράφημα ροής δεδομένων (2/2)

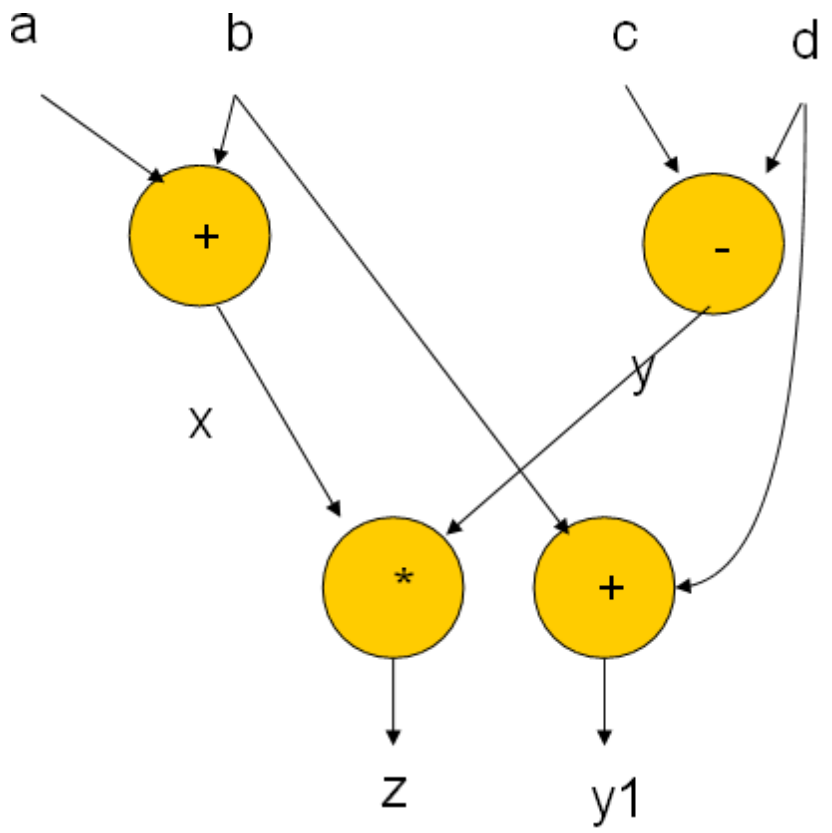
```
x = a + b;
y = c - d;
z = x * y;
y1 = b + d;
```

μορφή μονής ανάθεσης.




DFG

# DFGs και μερική ταξινόμηση




Μερική ταξινόμηση:

- a+b, c-d; b+d x*y

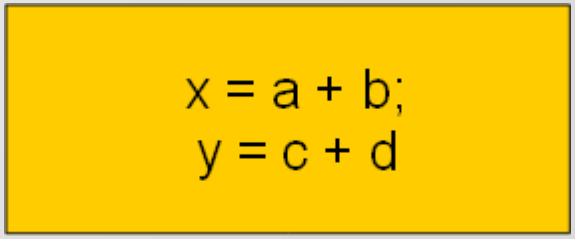
  Αρκετές πιθανές διατάξεις οι οποίες ικανοποιούν την απαίτηση.

# Γράφημα ροής ελέγχου/δεδομένων

- Γράφημα ροής ελέγχου/δεδομένων (control/data flow graph): **CDFG**
- Αναπαριστά λειτουργίες δεδομένων και λειτουργίες ελέγχου.
- Χρησιμοποιεί γραφήματα ροής δεδομένων σαν συστατικά.
- Δύο τύποι κόμβων:
  - Απόφασης,
  - ροής δεδομένων.

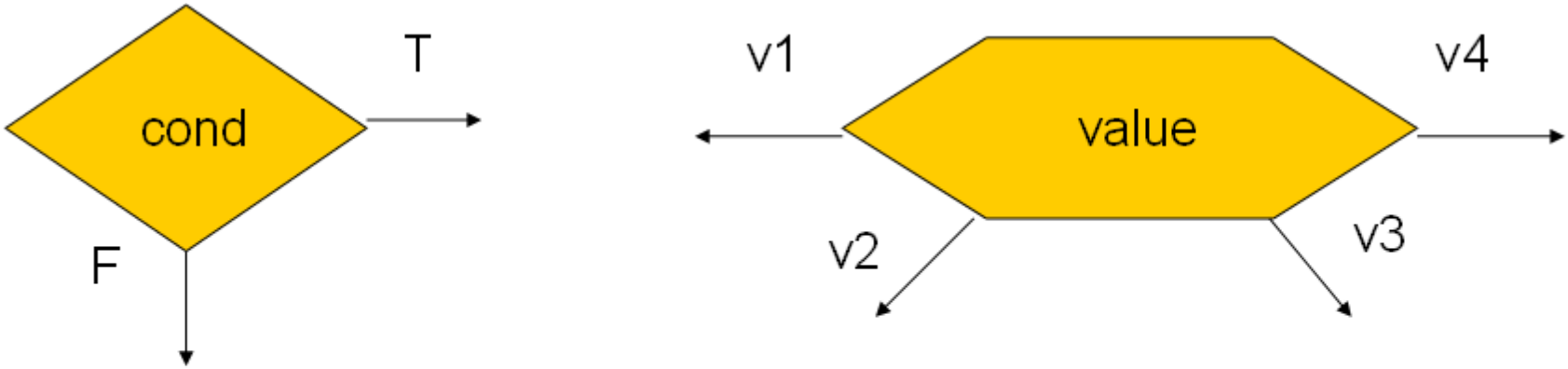
# Κόμβος ροής δεδομένων (data flow node)

Εμπεριέχει ένα πλήρες γράφημα ροής δεδομένων:




Εγγραφή λειτουργιών σε μορφή βασικού μπλοκ για απλότητα.

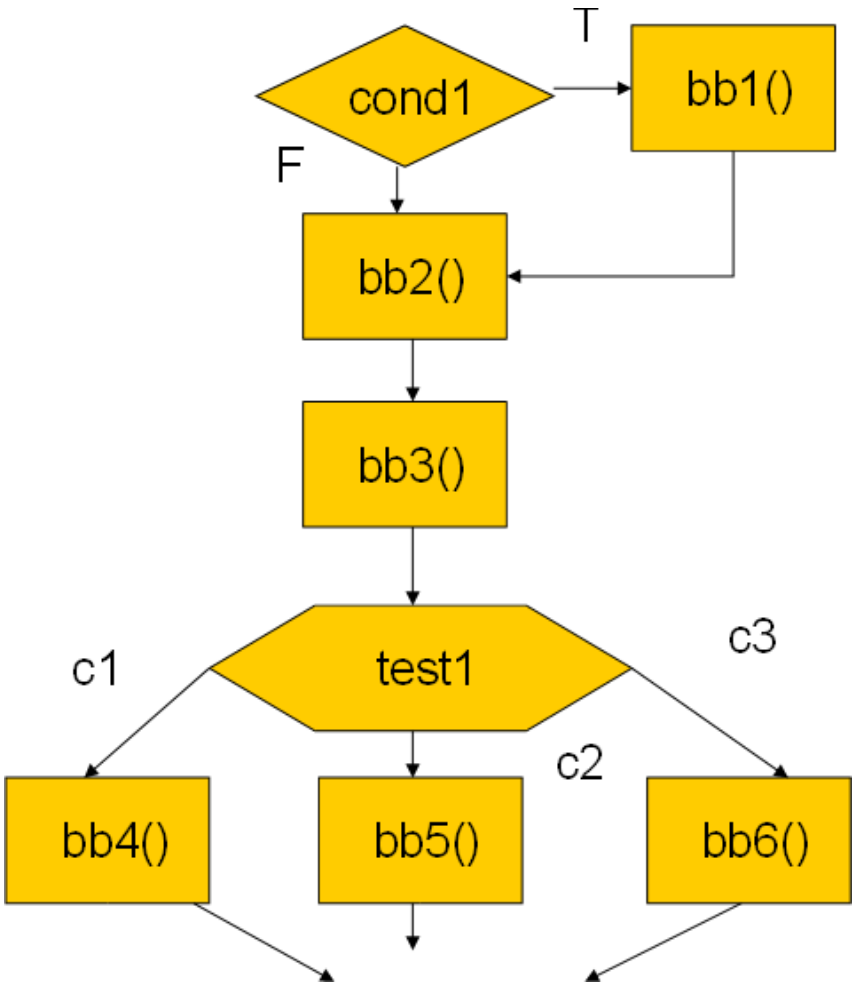
# Έλεγχος




**Ισοδύναμες μορφές**

# Παράδειγμα CDFG

```
if (cond1) bb1();
 else bb2();
bb3();
switch (test1) {
        case c1: bb4(); break;
        case c2: bb5(); break;
        case c3: bb6(); break;
}
```



# Ο βρόχος for

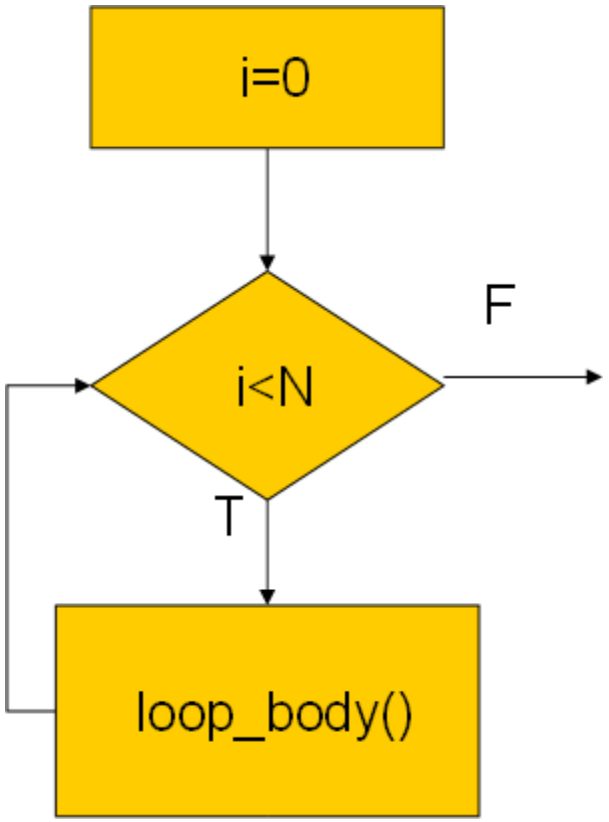
```
for (i=0; i<N; i++)
        loop_body();
```

*βρόχος for*

.....

```
i=0;
while (i<N) {
        loop_body(); i++; }
```

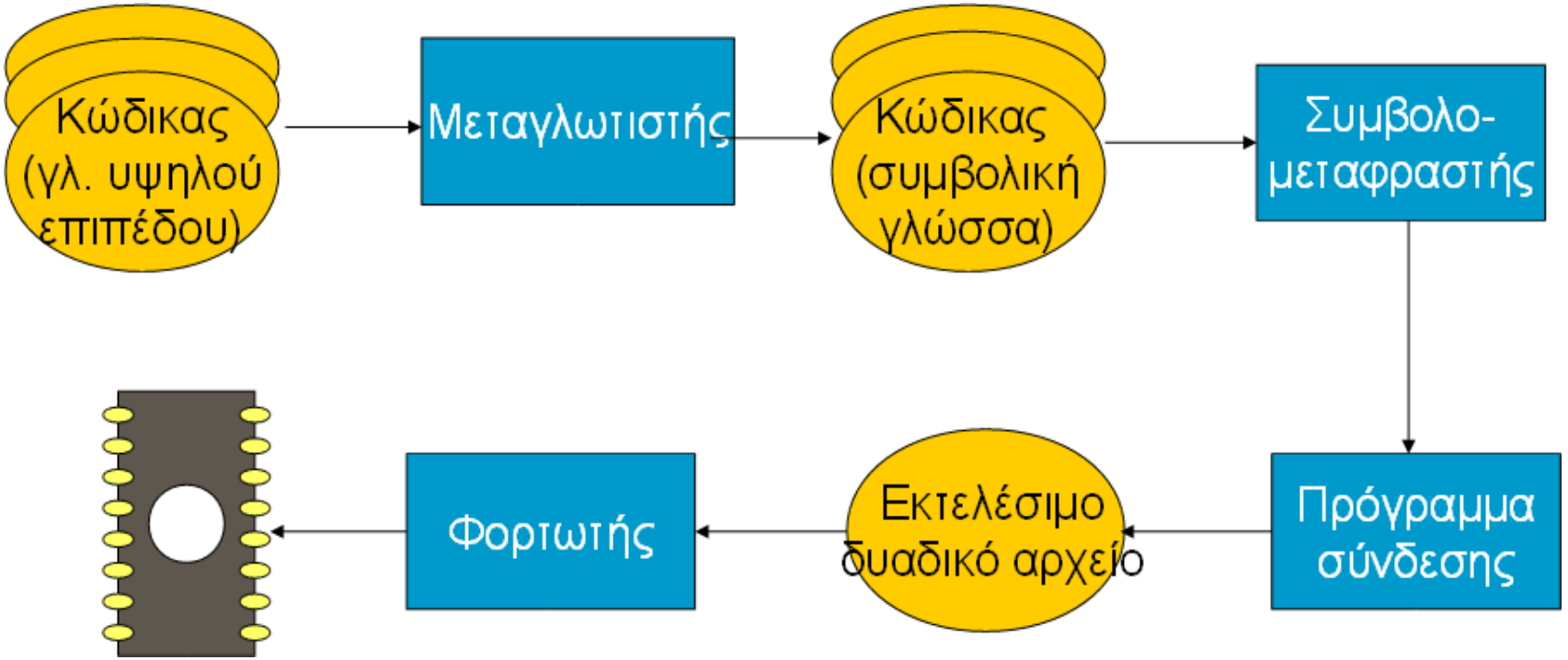
*ισοδύναμο*



# Συμβολομετάφραση και Σύνδεση.

# Συμβολομετάφραση και Σύνδεση

- Τελευταία βήματα της διαδικασίας μεταγλώττισης:



# Προγράμματα πολλαπλών αρχείων

- Το πρόγραμμα μπορεί να αποτελείται από πολλά αρχεία (υπομονάδες).
- Οι διευθύνσεις γίνονται πιο συγκεκριμένες κατά την επεξεργασία:
  - Οι **σχετικές διευθύνσεις** υπολογίζονται σχετικά ως προς τη διεύθυνση έναρξης της υπομονάδας.
  - Οι **απόλυτες διευθύνσεις** υπολογίζονται σχετικά ως προς τη διεύθυνση έναρξης του προγράμματος.

# Συμβολομεταφραστές

- Κύριες εργασίες:
  - Μετάφραση της συμβολικής γλώσσας σε δυαδική αναπαράσταση.
  - Μετάφραση των ετικετών σε διευθύνσεις.
  - Χειρισμός ψευδολειτουργιών (δεδομένα, κλπ.).
- Γενικά μετάφραση ένα-προς-ένα.
- Ετικέτες συμβολικής γλώσσας:

```
            ORG 100
label1      ADR r4,c
```

# Πίνακας συμβόλων

```
        ADD r0,r1,r2
xx      ADD r3,r4,r5
        CMP r0,r3
yy      SUB r5,r6,r7
```

**Κώδικας σε συμβολική Γλώσσα.**

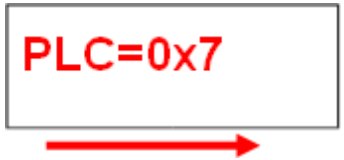
| | |
|---|---|
| xx | 0x8 |
| yy | 0x10 |

**Πίνακας συμβόλων.**

# Παραγωγή πίνακα συμβόλων

- Χρήση μετρητή τοποθεσίας προγράμματος (program location counter – **PLC**) για τον καθορισμό της διεύθυνσης.
- Σάρωση του προγράμματος, ο PLC ενημερώνεται για τη θέση.
- Οι διευθύνσεις παράγονται κατά τον χρόνο συμβολομετάφρασης, όχι κατά την εκτέλεση.

# Παράδειγμα πίνακα συμβόλων (1/4)



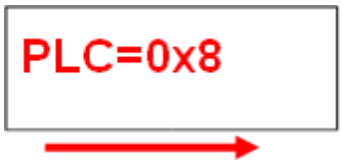

ADD r0,r1,r2

xx ADD r3,r4,r5

CMP r0,r3

yy SUB r5,r6,r7

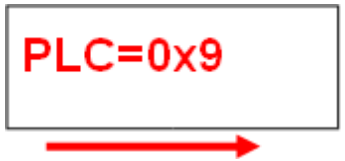
# Παράδειγμα πίνακα συμβόλων (2/4)




```
ADD r0,r1,r2
xx      ADD r3,r4,r5
        CMP r0,r3
yy      SUB r5,r6,r7
```

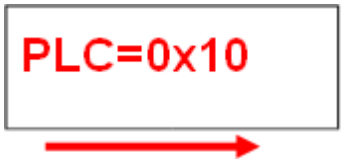
| | |
|---|---|
| xx | 0x8 |

# Παράδειγμα πίνακα συμβόλων (3/4)




| | | | |
|---|---|---|---|
| | ADD r0,r1,r2 | xx | 0x8 |
| xx | ADD r3,r4,r5 | | |
| | CMP r0,r3 | | |
| yy | SUB r5,r6,r7 | | |

# Παράδειγμα πίνακα συμβόλων (4/4)




```
     ADD r0,r1,r2
xx   ADD r3,r4,r5
     CMP r0,r3
yy   SUB r5,r6,r7
```

| | |
|---|---|
| xx | 0x8 |
| yy | 0x10 |

# Συμβολομετάφραση 2 περασμάτων

- Πέρασμα 1:
  - παραγωγή πίνακα συμβόλων.
- Πέρασμα 2:
  - παραγωγή δυαδικών εντολών.

# Παραγωγή σχετικών διευθύνσεων

- Μερικές τιμές ετικετών μπορεί να μην είναι γνωστές κατά τη διάρκεια της συμβολομετάφρασης.
- Οι εντολές της υπομονάδας πρέπει να είναι σε σχετική μορφή.
- Πρέπει να παρακολουθούνται οι εξωτερικές ετικέτες---δεν μπορεί να παραχθεί πλήρης δυαδική μορφή από εντολές που κάνουν χρήση εξωτερικών ετικετών.

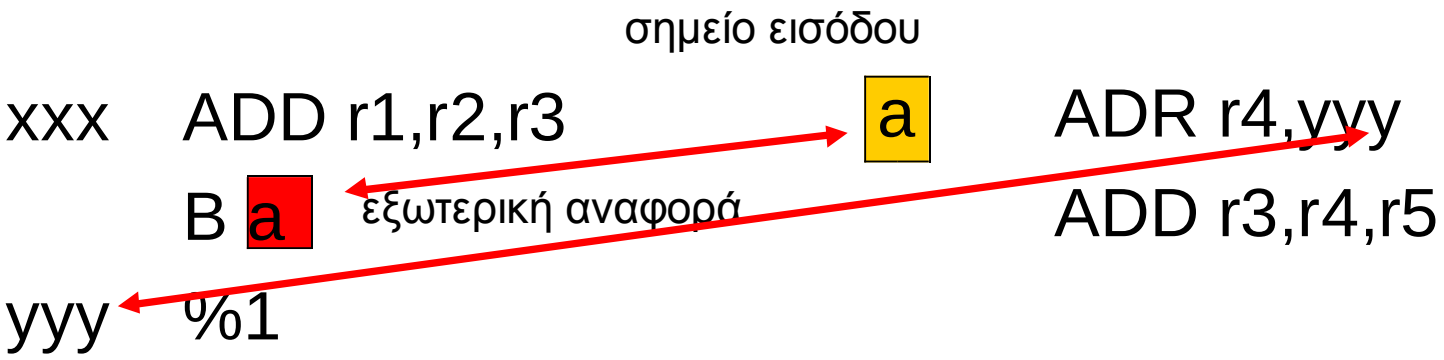
# Ψευδολειτουργίες

- Οι ψευδολειτουργίες δεν παράγουν εντολές:
  - **ORG** θέτει την αρχή του προγράμματος.
  - **EQU** επιτρέπει την προσθήκη ετικετών στον πίνακα συμβόλων χωρίς να καταλαμβάνουν χώρο στη μνήμη προγράμματος.
  - **Data statements** (δηλώσεις δεδομένων) καθορίζουν τα μπλοκ δεδομένων.

# Σύνδεση

- Επιτρέπει την σύνθεση των επί μέρους μικρότερων τμημάτων ενός προγράμματος σε ένα ενιαίο πρόγραμμα.
- Εργασίες:
  - τοποθέτηση των τμημάτων σε σειρά,
  - αποσαφήνιση των ετικετών των τμημάτων.

# Εξωτερικές αναφορές και σημεία εισόδου



# Διάταξη αρχείων

- Τα αρχεία κώδικα πρέπει να τοποθετούνται σε ακριβείς θέσεις στη μνήμη.
- Ο **χάρτης φόρτωσης (load map)** ή συνδετικές παράμετροι (linker parameters) ορίζουν την σειρά με την οποία τα αρχεία πρέπει να τοποθετηθούν στη μνήμη.

# Δυναμική σύνδεση

- Μερικά λειτουργικά συστήματα συνδέουν τα αρχεία δυναμικά κατά τον χρόνο εκτέλεσης:
  - διαμοιρασμός ενός αντιγράφου βιβλιοθήκης μεταξύ των προγραμμάτων που εκτελούνται,
  - επιτρέπει στα προγράμματα που χρησιμοποιούν αυτές τις βιβλιοθήκες να ενημερώνονται εύκολα.

# Σχεδιασμός και ανάλυση προγράμματος

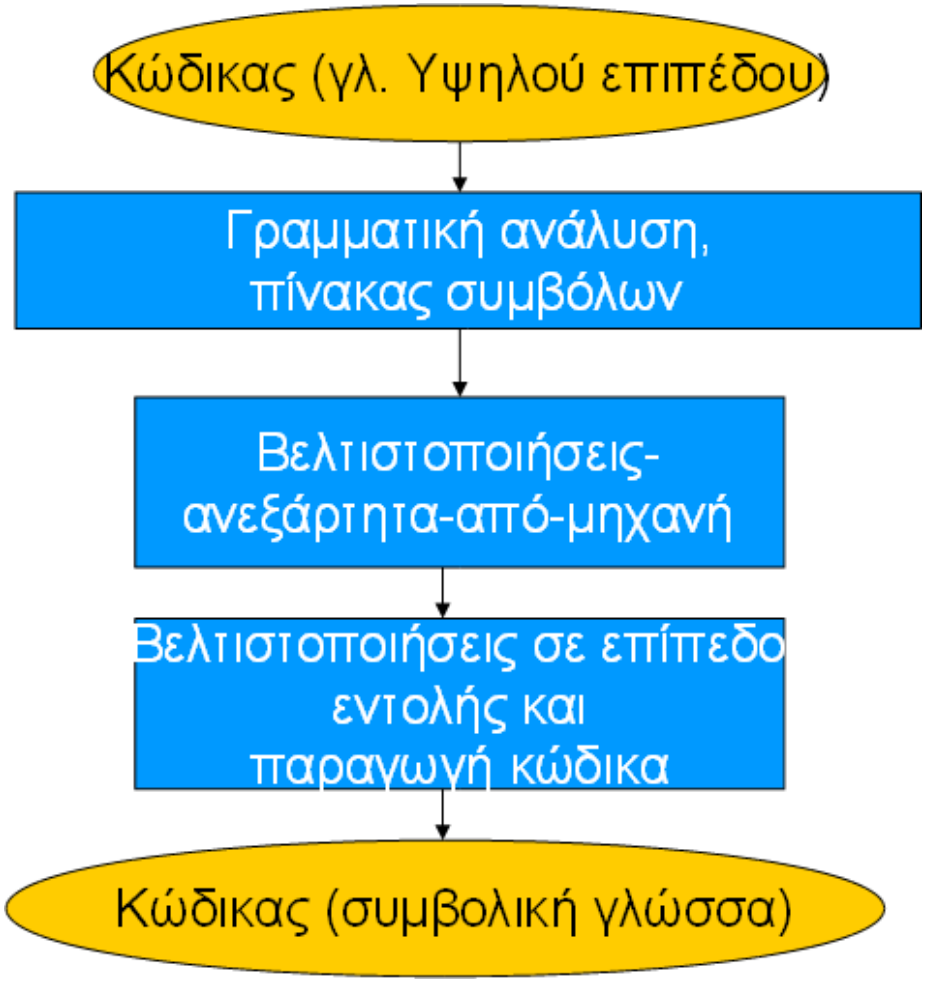
- Ροή μεταγλώττισης.
- Βασική μετάφραση εντολών.
- Βασικές βελτιστοποιήσεις.
- Ερμηνευτές και μεταγλωττιστές JIT (just-in-time).

# Μεταγλώττιση

- Στρατηγική μεταγλώττισης:
  - μεταγλώττιση = μετάφραση + βελτιστοποίηση
- Ο μεταγλωττιστής καθορίζει την ποιότητα του κώδικα:
  - χρήση των πόρων της CPU.
  - προγραμματισμός προσπελάσεων μνήμης.
  - μέγεθος κώδικα.

# Βασικές Τεχνικές Μεταγλώττισης

# Βασικές φάσεις μεταγλώττισης



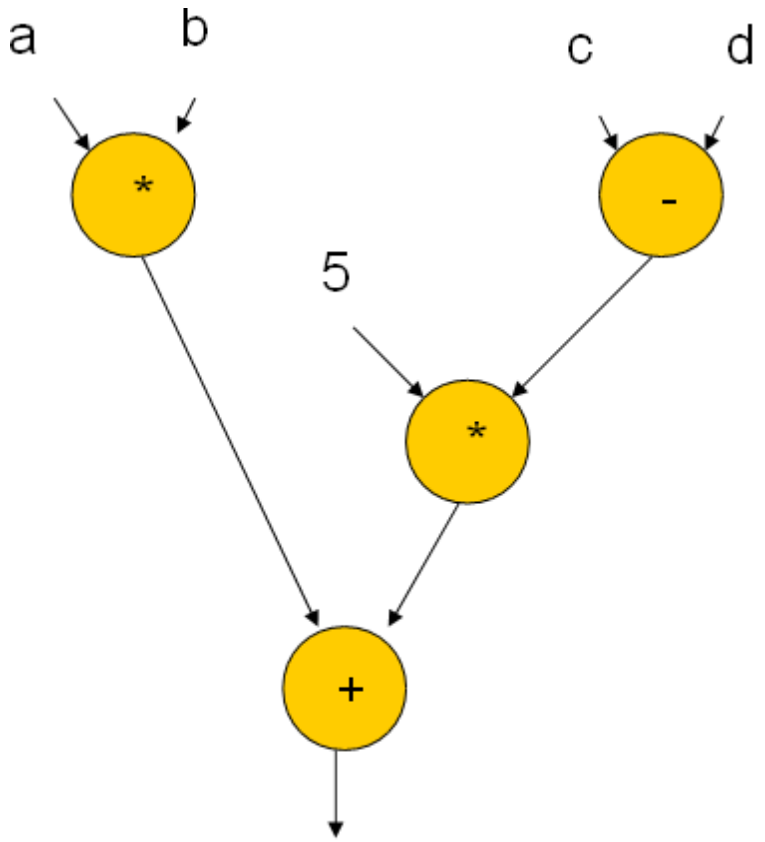
# Μετάφραση εντολών και βελτιστοποίηση

- Ο πηγαίος κώδικας μεταφράζεται σε ενδιάμεση μορφή όπως το CDFG.
- Το CDFG μετασχηματίζεται/βελτιστοποιείται.
- Το CDFG μεταφράζεται σε εντολές με αποφάσεις βελτιστοποίησης.
- Οι εντολές βελτιστοποιούνται περαιτέρω.

# Αριθμητικές εκφράσεις (1/2)

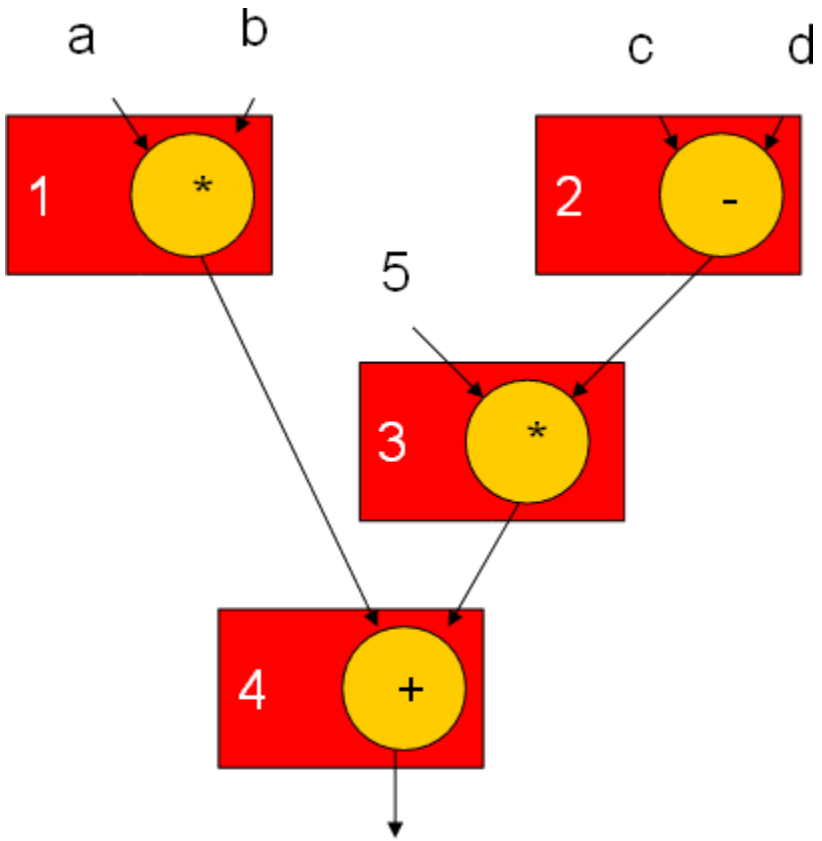
**a*b + 5*(c-d)**

**έκφραση**




DFG

# Αριθμητικές εκφράσεις (2/2)



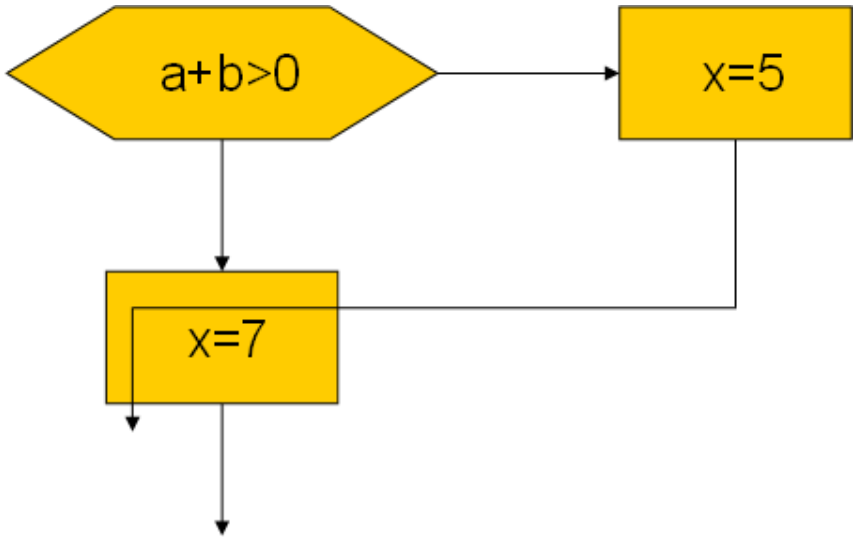

DFG

```
ADR r4,a
MOV r1,[r4]
ADR r4,b
MOV r2,[r4]
ADR r3,r1,r2

ADR r4,c
MOV r1,[r4]
ADR r4,d
MOV r5,[r4]
SUB r6,r4,r5

MUL r7,r6,#5
ADD r8,r7,r3
```

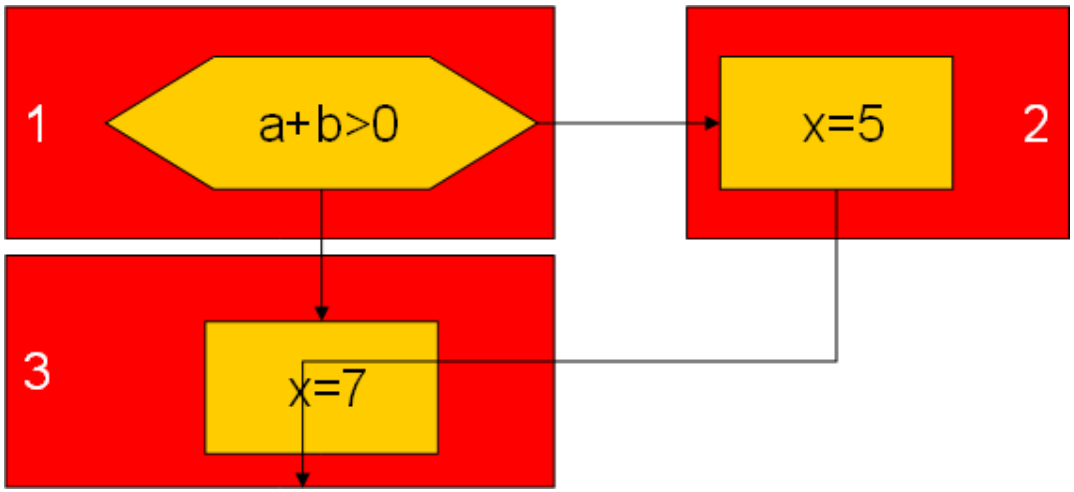
κώδικας

# Παραγωγή κώδικα για μια συνθήκη (1/2)

```
if (a+b > 0)
    x = 5;
else
    x = 7;
```



# Παραγωγή κώδικα για μια συνθήκη (2/2)

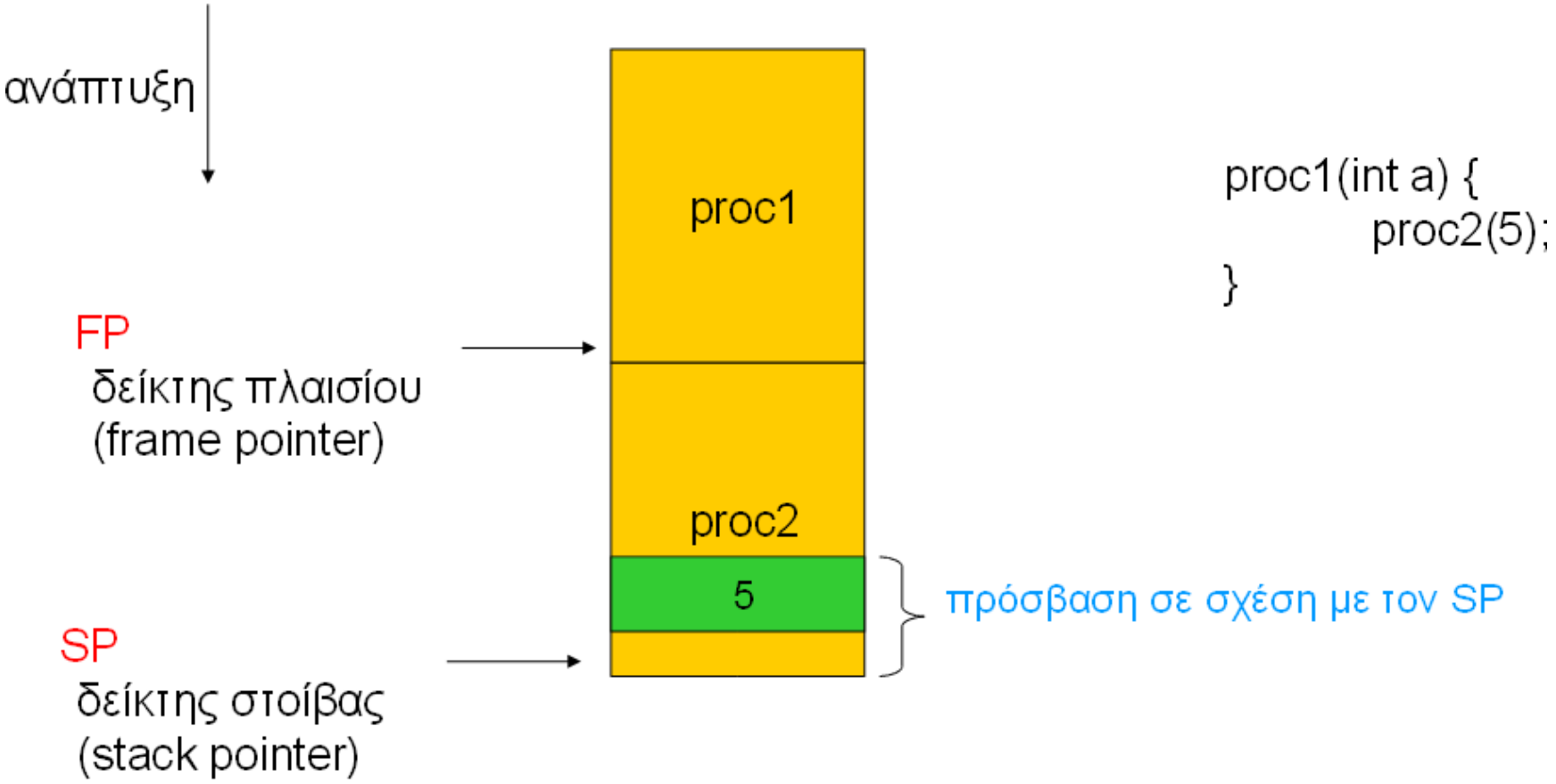



```
        ADR r5,a
        LDR r1,[r5]
        ADR r5,b
        LDR r2,b
        ADD r3,r1,r2
        BLE label3

        LDR r3,#5
        ADR r5,x
        STR r3,[r5]
        B stmtent

        LDR r3,#7
        ADR r5,x
        STR r3,[r5]

stmtent ...
```

# Διαδικασία σύνδεσης

- Απαιτείται κώδικας για:
  - κλήση και επιστροφή.
  - πέρασμα παραμέτρων και αποτελεσμάτων.
- Οι παράμετροι και οι επιστροφές περνάνε στη στοίβα.
  - Διαδικασίες με λίγες παραμέτρους ίσως χρησιμοποιούν καταχωρητές.

# Στοίβες διαδικασίας



# Διαδικασία σύνδεσης ARM

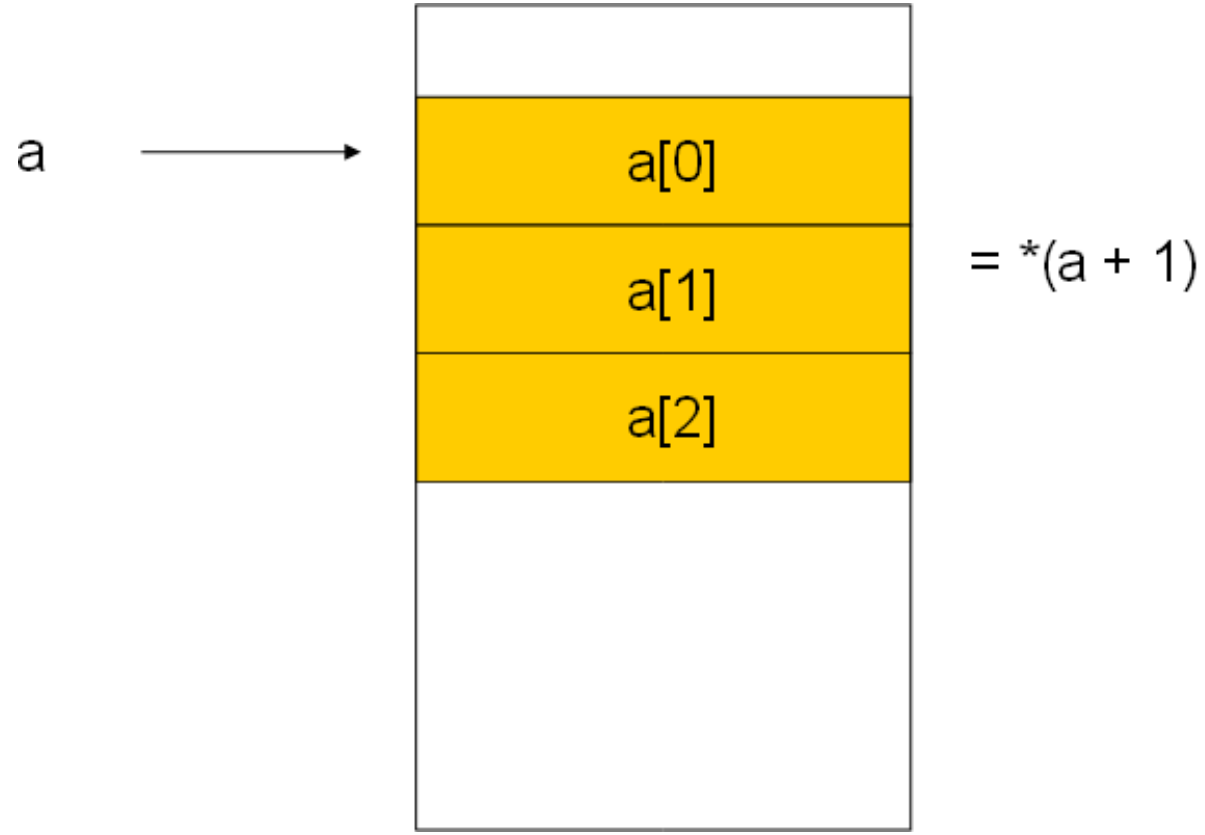
- Το πρότυπο κλήσης διαδικασιών του ARM (ARM Procedure Call Standard - APCS):
  - r0-r3 χρησιμοποιούνται για το πέρασμα των παραμέτρων στις διαδικασίες. Επιπλέον παράμετροι τοποθετούνται στο πλαίσιο στοίβας.
  - r0 χρησιμοποιείται επίσης για να κρατά την τιμή επιστροφής.
  - r4-r7 κρατούν μεταβλητές καταχωρητών.
  - R11 είναι ο δείκτης πλαισίου, r13 είναι ο δείκτης στοίβας.
  - r10 κρατά τον περιορισμό διεύθυνσης για το μέγεθος της στοίβας το οποίο χρησιμοποιείται για να ελέγχει την υπερχείλιση της στοίβας.

# Δομές δεδομένων

- Διαφορετικά είδη δομών δεδομένων χρησιμοποιούν διαφορετικές μορφές δεδομένων.
- Μερικές μετατοπίσεις στις δομές δεδομένων μπορούν να υπολογιστούν κατά τον χρόνο μεταγλώττισης, ενώ άλλες πρέπει να γίνουν κατά τον χρόνο εκτέλεσης.

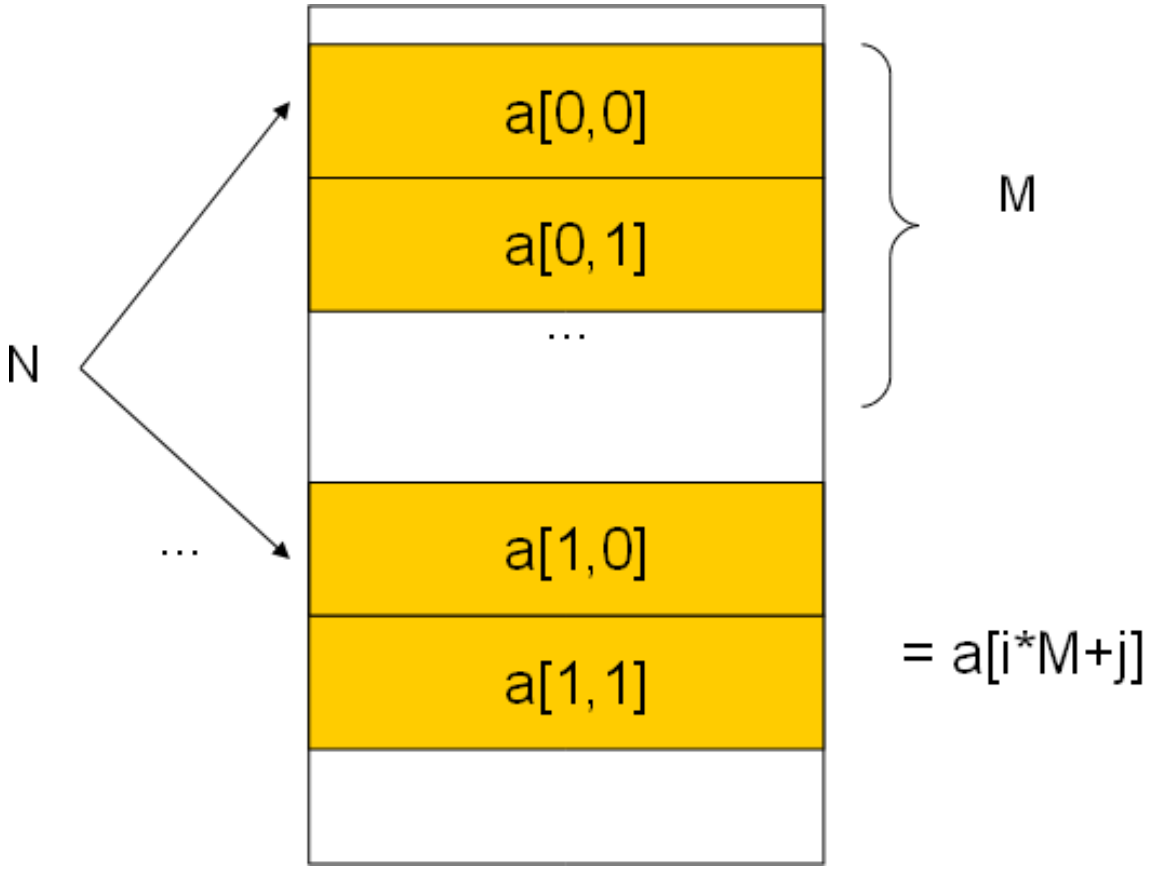
# Μονοδιάστατοι πίνακες

- Το όνομα του πίνακα δείχνει στο μηδενικό στοιχείο:



# Δισδιάστατοι πίνακες

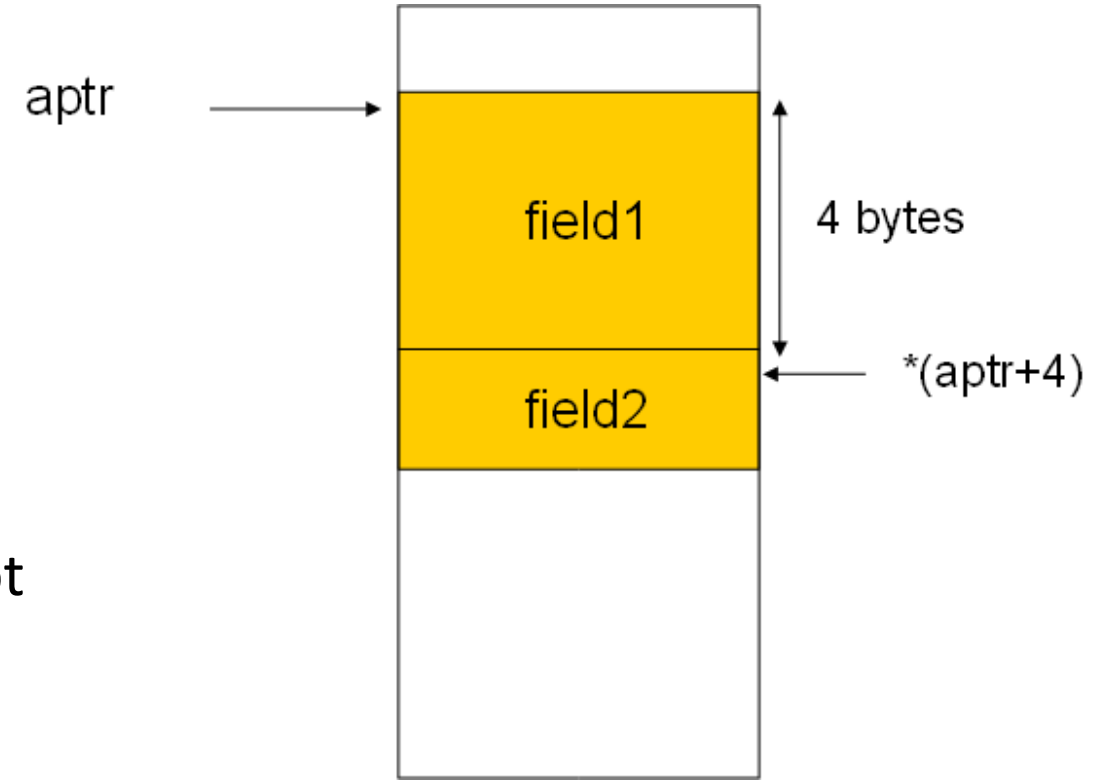
- Μορφή μείζονος στήλης (column-major):



# Δομές

Τα πεδία της δομής μπορούν να προσπελαστούν χρησιμοποιώντας σταθερές μετατοπίσεις:

```
struct {
   int field1;
   char field2;
} mystruct;

struct mystruct a, *apt
```



# Απλοποίηση εκφράσεων

- Υπολογισμός σταθερών εκφράσεων (Constant folding):
  - 8+1 = 9
- Αλγεβρικά:
  - a*b + a*c = a*(b+c)
- Μείωση έντασης (strength reduction):
  - a*2 = a<<1

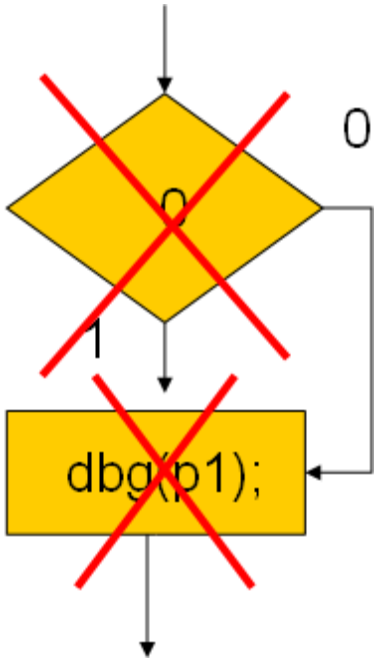
# Απαλοιφή νεκρού κώδικα

- Νεκρός κώδικας

  #define DEBUG 0

  if (DEBUG) dbg(p1);

- Μπορεί να εξαλειφθεί με ανάλυση ελέγχου ροής, και constant folding.



# Ενσωμάτωση διαδικασιών στον κώδικα (procedure inlining)

- Εξαλείφει την επιβάρυνση σύνδεσης της διαδικασίας.

```
int foo(a,b,c) { return a + b - c;}
z = foo(w,x,y);
        ->
z = w + x + y;
```

# Μετασχηματισμοί Βρόχων

- Στόχοι:
  - μείωση επιβάρυνσης βρόχου,
  - αύξηση των δυνατοτήτων διασωλήνωσης,
  - βελτίωση της απόδοσης του συστήματος μνήμης.

# Ξετύλιγμα βρόχου

- Μειώνει την επιβάρυνση βρόχου, επιτρέπει μερικές ακόμα βελτιστοποιήσεις.

```
for (i=0; i<4; i++)
a[i] = b[i] * c[i];
```

**->**

```
for (i=0; i<2; i++) {
a[i*2] = b[i*2] * c[i*2];
a[i*2+1] = b[i*2+1] * c[i*2+1];
}
```

# Συγχώνευση βρόχων / Κατανομή βρόχου

- Η συγχώνευση βρόχων συνδυάζει δύο ή περισσότερους βρόχους σε έναν ενιαίο βρόχο:

```
for (i=0; i<N; i++) a[i] = b[i] * 5;
for (j=0; j<N; j++) w[j] = c[j] * d[j];
-> for (i=0; i<N; i++) {
a[i] = b[i] * 5; w[i] = c[i] * d[i];
}
```

- Η κατανομή βρόχου αποσυνθέτει έναν βρόχο σε περισσότερους βρόχους.
- Αλλάζει τις βελτιστοποιήσεις εντός του βρόχου.

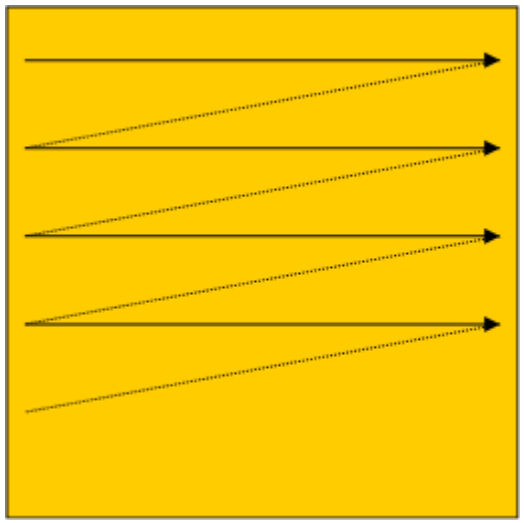
# Τεμαχισμός βρόχου (loop tiling)

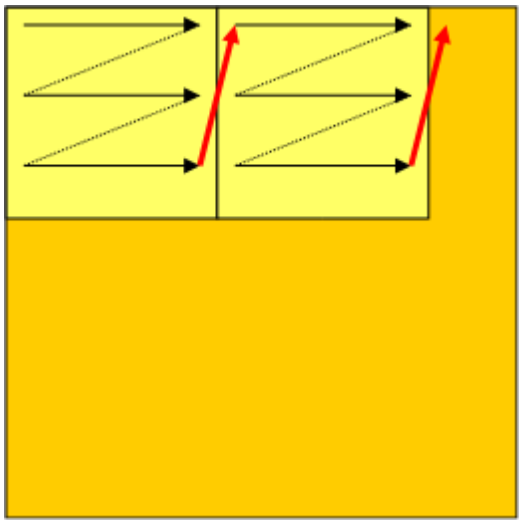
- Τεμαχίζει έναν βρόχο σε ένα σύνολο φωλιασμένων βρόχων.
- Αλλάζει την σειρά με την οποία προσπελάζονται τα στοιχεία του πίνακα.
  - Αλλάζει τη συμπεριφορά της κρυφής μνήμης.

# Τεμαχισμός βρόχου παράδειγμα

```
for (i=0; i<N; i++)
        for (j=0; j<N; j++)
                c[i] = a[i,j]*b[i];
```

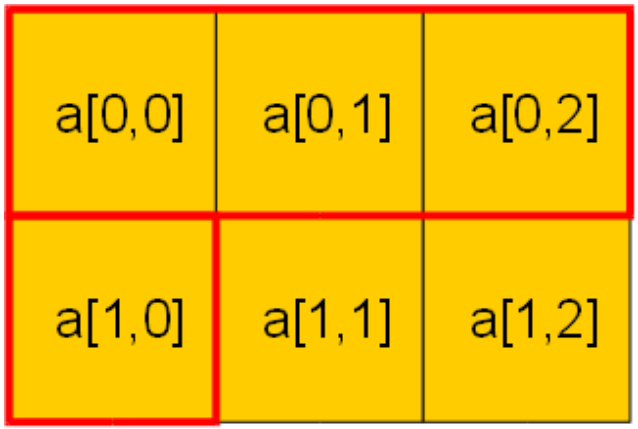
```
for (i=0; i<N; i+=2)
   for (j=0; j<N; j+=2)
      for (ii=0; ii<min(i+2,n); ii++)
         for (jj=0; jj<min(j+2,N); jj++)
                c[ii] = a[ii,jj]*b[ii];
```



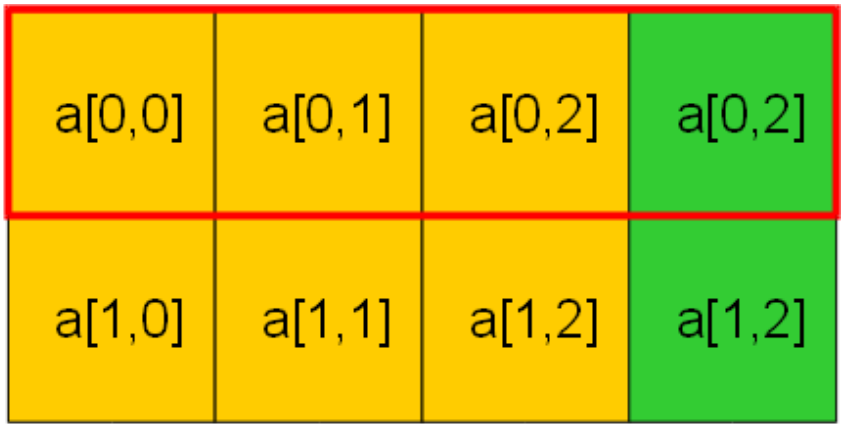


# Συμπλήρωση πίνακα

- Προσθέτει εικονικά στοιχεία δεδομένων στον βρόχο προκειμένου να αλλάξει τη μορφή του πίνακα στην κρυφή μνήμη:




πριν




μετά

# Κατανομή καταχωρητών

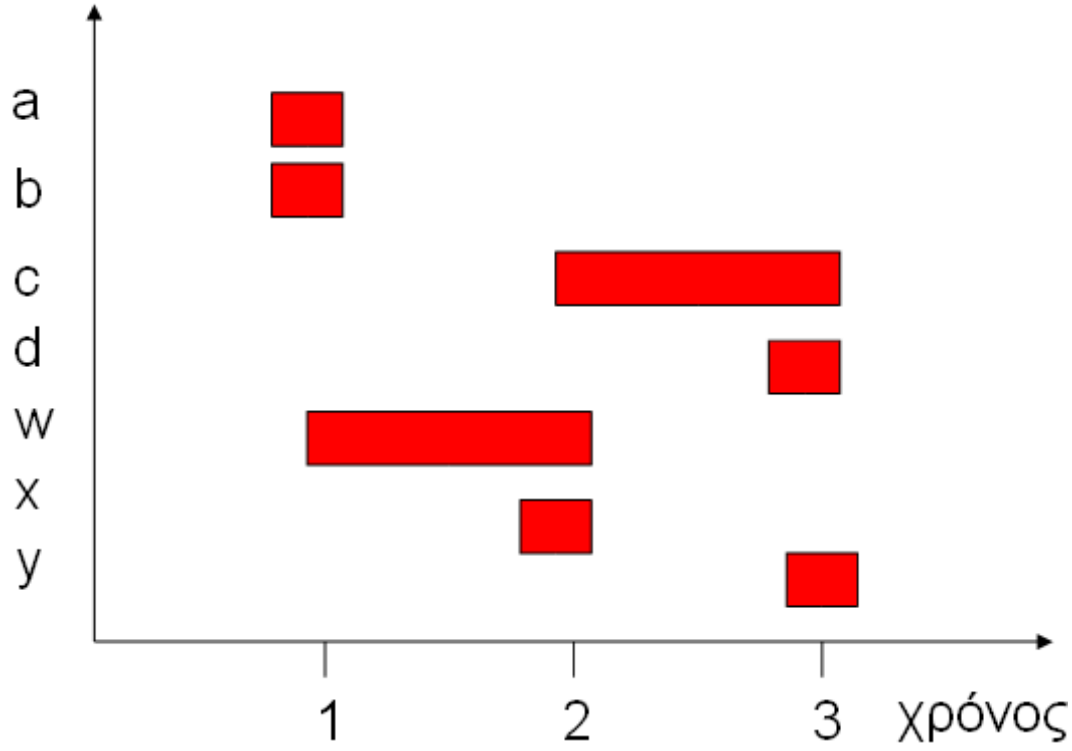
- Στόχοι:
  - επιλογή καταχωρητή που θα κρατάει κάθε μεταβλητή,
  - καθορισμός της διάρκειας ζωής της μεταβλητής στον καταχωρητή.
- Βασική περίπτωση: εντός του βασικού μπλοκ.

# Γραφική παράσταση διάρκειας ζωής καταχωρητή

w = a + b; t=1

x = c + w; t=2

y = c + d; t=3



# Χρονοδρομολόγηση εντολών

- Οι μη διασωληνωμένες μηχανές δεν χρειάζονται χρονοδρομολόγηση εντολών: οποιαδήποτε διάταξη εντολών ικανοποιεί τις εξαρτήσεις των δεδομένων τρέχει εξίσου γρήγορα.
- Σε διασωληνωμένες μηχανές, ο χρόνος εκτέλεσης μιας εντολής εξαρτάται από τις γειτονικές εντολές: **κώδικας λειτουργίας, τελεστές.**

# Πίνακας δέσμευσης

- Ένας πίνακας δέσμευσης αφορά στις εντολές/χρόνος και πόροι της CPU.

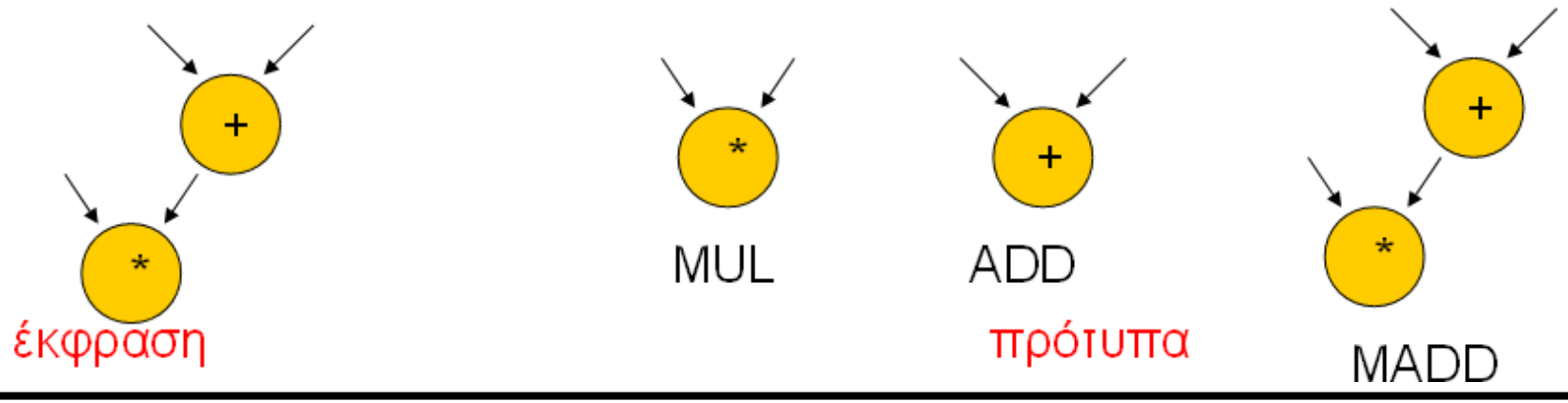
| Time/instr | A | B |
|---|---|---|
| instr1 | X | |
| instr2 | X | X |
| instr3 | X | |
| instr4 | | X |

# Διοχέτευση λογισμικού (software pipeline)

- Τεχνική επαναδιάταξης των εντολών στις διάφορες επαναλήψεις βρόχων.
- Μειώνει την καθυστέρηση των εντολών στην επανάληψη i εισάγοντας εντολές από την επανάληψη i+1.

# Επιλογή εντολών

- Μπορεί να υπάρχουν αρκετοί τρόποι για την υλοποίηση μιας λειτουργίας ή ακολουθίας λειτουργιών.
- Αναπαράσταση λειτουργιών σε γραφήματα, ταίριασμα πιθανών ακολουθιών εντολών σε γράφημα.



# Χρήση του μεταγλωττιστή σας

- Μπορείτε να πειραματιστείτε με τα διαφορετικά επίπεδα βελτιστοποίησης (-O1, -O2, κλπ.)
- Μπορείτε να δείτε την έξοδο του μεταγλωττιστή/συμβολομεταφραστή.
- Η τροποποίηση της εξόδου του μεταγλωττιστή απαιτεί προσοχή:
  - ορθότητα
  - Πιθανή απώλεια του κώδικα που τροποποιεί ο χρήστης (hand-tweaked), αν ξαναγίνει compile.

# Ερμηνευτές και Μεταγλωττιστές JIT

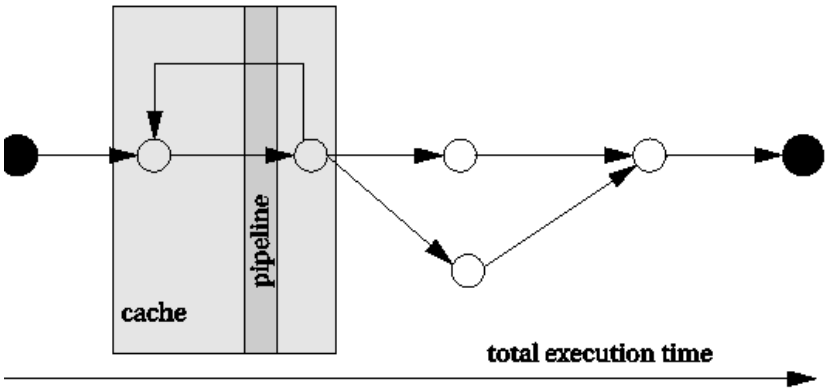
- **Ερμηνευτής:** μεταφράζει εντολές του προγράμματος μία τη φορά.
- **Μεταγλωττιστής ακριβώς-στην-ώρα (just-in-time):** μεταγλωττίζει μικρά τμήματα κώδικα σε εντολές κατά τη διάρκεια της εκτέλεσης του προγράμματος.
  - Εξαλείφει κάποια επιβάρυνση μετάφρασης.
  - Συχνά απαιτεί περισσότερη μνήμη.

# Ανάλυση και βελτιστοποίηση του Χρόνου Εκτέλεσης

# Σχεδίαση και ανάλυση προγράμματος

- Ανάλυση απόδοσης επιπέδου προγράμματος.
- Βελτιστοποίηση σε:
  - Χρόνο εκτέλεσης.
  - Ενέργεια/ισχύς.
  - Μέγεθος προγράμματος.
- Επικύρωση και δοκιμή του προγράμματος.

# Ανάλυση απόδοσης σε επίπεδο προγράμματος




- Πρέπει να κατανοήσουμε λεπτομερώς την απόδοση:
  - Συμπεριφορά πραγματικού χρόνου, όχι απλά τυπικού.
  - Πολύπλοκες πλατφόρμες.
- Απόδοση προγράμματος ≠ απόδοση CPU:
  - Διοχέτευση, οι κρυφές μνήμες είναι παράθυρα στο πρόγραμμα.
  - Πρέπει να αναλύσουμε ολόκληρο το πρόγραμμα.

# Πολυπλοκότητες της απόδοσης προγράμματος

- Ποικίλει από τις τιμές των δεδομένων εισόδου:
  - Διαδρομές διαφορετικού μήκους.
- Επιδράσεις κρυφής μνήμης.
- Διακυμάνσεις απόδοσης σε επίπεδο εντολών:
  - Εξαρτήσεις και παύσεις διασωλήνωσης (pipeline interlocks).
  - Χρόνοι προσκόμισης.

# Πως μετράμε την απόδοση ενός προγράμματος;

- Προσομοίωση της εκτέλεσης στη CPU.
  - Καταστεί ορατή την κατάσταση της CPU.
- Μέτρηση σε πραγματική CPU, με τη χρήση χρονομέτρου.
  - Απαιτεί τροποποίηση του προγράμματος για έλεγχο του χρονομέτρου.
- Μέτρηση σε πραγματική CPU, με τη χρήση λογικού αναλυτή.
  - Απαιτεί τα γεγονότα να είναι ορατά από τους ακροδέκτες.

# Μετρικές απόδοσης του προγράμματος

- Χρόνος εκτέλεσης τυπικής περίπτωσης.
  - Τυπικά χρησιμοποιείται στον προγραμματισμό εφαρμογών.
- Χρόνος εκτέλεσης χειρότερης περίπτωσης.
  - Ένα συστατικό για την ικανοποίηση της προθεσμίας για την εκτέλεση λειτουργίας.
- Χρόνος εκτέλεσης βέλτιστης περίπτωσης.
  - Οι αλληλεπιδράσεις σε επίπεδο εργασιών μπορεί να οδηγήσουν την συμπεριφορά βέλτιστης περίπτωσης προγράμματος σε συμπεριφορά χειρότερης περίπτωσης συστήματος.

# Στοιχεία της απόδοσης του προγράμματος

- Βασικός τύπος χρόνου εκτέλεσης προγράμματος:
  - χρόνος εκτέλεσης = διαδρομή προγράμματος + χρόνος εντολής.
- Επίλυση αυτών των προβλημάτων ανεξάρτητα βοηθάει στην απλοποίηση της ανάλυσης.
  - Ευκολότερο να διαχωριστούν σε πιο απλές CPU.
- Ακριβής ανάλυση απόδοσης απαιτεί:
  - Συμβολικό/δυαδικό κώδικα.
  - Πλατφόρμα εκτέλεσης.

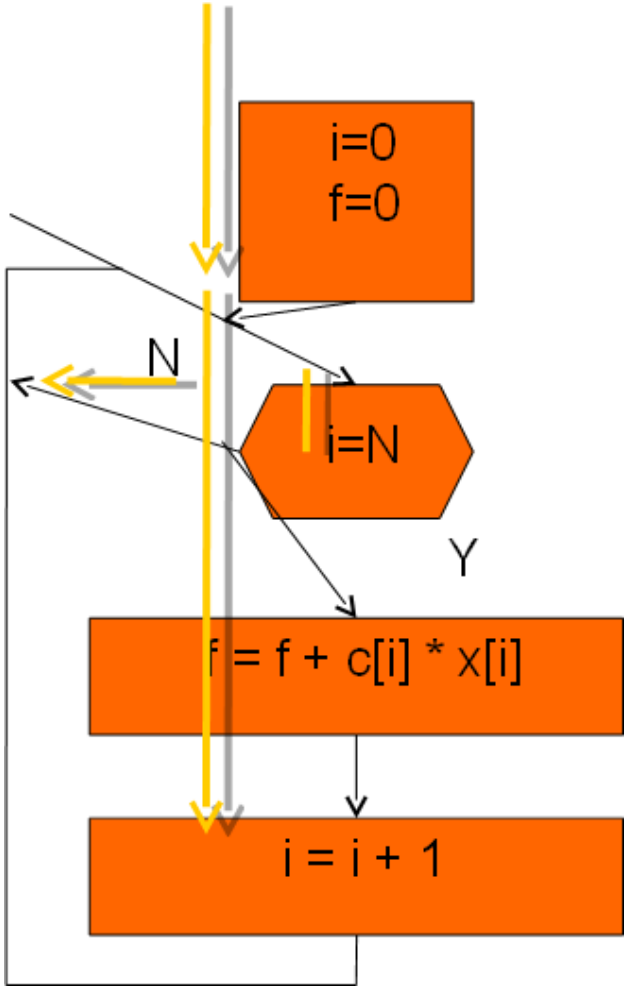
# Διαδρομές προγράμματος εξαρτώμενες από δεδομένα σε εντολές if

```
if (a || b) { /* T1 */
            if ( c ) /* T2 */
            x = r*s+t; /* A1 */
            else y=r+s; /* A2 */
            z = r+s+u; /* A3 */
            }
else {
            if ( c ) /* T3 */
                y = r-t; /* A4 */
}
```

| a | b | c | διαδρομή |
|---|---|---|---|
| 0 | 0 | 0 | T1=F, T3=F: no assignments |
| 0 | 0 | 1 | T1=F, T3=T: A4 |
| 0 | 1 | 0 | T1=T, T2=F: A2, A3 |
| 0 | 1 | 1 | T1=T, T2=T: A1, A3 |
| 1 | 0 | 0 | T1=T, T2=F: A2, A3 |
| 1 | 0 | 1 | T1=T, T2=T: A1, A3 |
| 1 | 1 | 0 | T1=T, T2=F: A2, A3 |
| 1 | 1 | 1 | T1=T, T2=T: A1, A3 |

# Διαδρομές σε ένα βρόχο

```
for (i=0, f=0; i<N; i++)
    f = f + c[i] * x[i];
```



# Χρονισμός εντολής

- Δεν απαιτούν όλες οι εντολές τον ίδιο χρόνο εκτέλεσης.
  - Εντολές πολλαπλών κύκλων εκτέλεσης.
  - Προσκομίσεις.
- Οι χρόνοι εκτέλεσης των εντολών δεν είναι ανεξάρτητοι.
  - Pipeline interlocks.
  - Επιδράσεις κρυφής μνήμης.
- Ο χρόνος εκτέλεσης μιας εντολής μπορεί να εξαρτάται από τις τιμές του τελεστή.
  - Εντολές κινητής υποδιαστολής.
  - Μερικές λειτουργίες ακεραίων πολλαπλών κύκλων.

# Ανάλυση απόδοσης οδηγούμενη από μέτρηση

- Όχι τόσο εύκολο όσο ακούγεται:
  - Πρέπει πραγματικά να υπάρχει πρόσβαση στη CPU.
  - Πρέπει να είναι γνωστές οι είσοδοι δεδομένων που οδηγούν σε απόδοση χειρότερης/καλύτερης περίπτωσης.
  - Πρέπει να καθίσταται ορατή η κατάσταση.
  - Σημαντική μέθοδος για ανάλυση απόδοσης.

# Τροφοδοτώντας το πρόγραμμα

- Πρέπει να είναι γνωστές οι επιθυμητές τιμές εισόδου.
- Ίσως χρειαστεί να γραφτεί επιπρόσθετο λογισμικό παραγωγής τιμών (software scaffolding) για να παραχθούν οι τιμές εισόδου.
- Το software scaffolding ίσως χρειαστεί επίσης να εξετάσει τις εξόδους για παραγωγή εισόδων οδηγούμενες από ανατροφοδότηση.

# Μέτρηση οδηγούμενη από ίχνος

- Οδηγούμενη από ίχνος:
  - Μέτρηση του προγράμματος.
  - Καταγραφή της διαδρομής εκτέλεσης.
- Απαιτεί τροποποίηση του προγράμματος.
- Τα ίχνη γενικά είναι μεγάλα.
- Ευρέως χρησιμοποιούμενη για ανάλυση της κρυφής μνήμης.

# Φυσική μέτρηση

- Εξομοιωτής μέσα στο κύκλωμα επιτρέπει την καταγραφή ιχνών.
  - Επηρεάζει τον χρονισμό εκτέλεσης.
- Ο λογικός αναλυτής μπορεί να μετρήσει τη συμπεριφορά στα pins.
  - Η διεύθυνση διαύλου μπορεί να αναλυθεί για να ψάχνει για γεγονότα.
  - Ο κώδικας μπορεί να τροποποιηθεί για να κάνει τα γεγονότα ορατά.
- Ιδιαίτερα σημαντικό για ροές εισόδου πραγματικού κόσμου.

# Προσομοίωση της CPU

- Μερικοί προσομοιωτές είναι λιγότερο ακριβείς.
- Προσομοιωτής ακριβού αριθμού κύκλων παρέχει ακριβή χρονισμό κύκλου ρολογιού.
  - Ο προσομοιωτής μοντελοποιεί την εσωτερική λειτουργία μιας CPU.
  - Ο συγγραφέας ενός προσομοιωτή πρέπει να γνωρίζει με λεπτομέρεια πως λειτουργεί η CPU.

# Προσομοίωση FIR φίλτρου στο SimpleScalar

```
int x[N] = {8, 17, … };
int c[N] = {1, 2, … };
main() {
   int i, k, f;
    for (k=0; k<COUNT; k++)
            for (i=0; i<N; i++)
                    f += c[i]*x[i];
}
```

| N | total sim cycles | sim cycles per filter execution |
|---|---|---|
| 100 | 25854 | 259 |
| 1,000 | 155759 | 156 |
| 1,0000 | 1451840 | 145 |

# Κίνητρο βελτιστοποίησης απόδοσης

- Τα ενσωματωμένα συστήματα πρέπει συχνά να συναντούν τις προθεσμίες.
  - Το γρηγορότερο μπορεί να μην είναι αρκετά γρήγορο.
- Ανάγκη ικανότητας ανάλυσης του χρόνου εκτέλεσης.
  - Χειρότερη περίπτωση, όχι τυπική.
- Ανάγκη τεχνικών για βελτίωση του χρόνου εκτέλεσης αξιόπιστα.

# Προγράμματα και ανάλυση απόδοσης

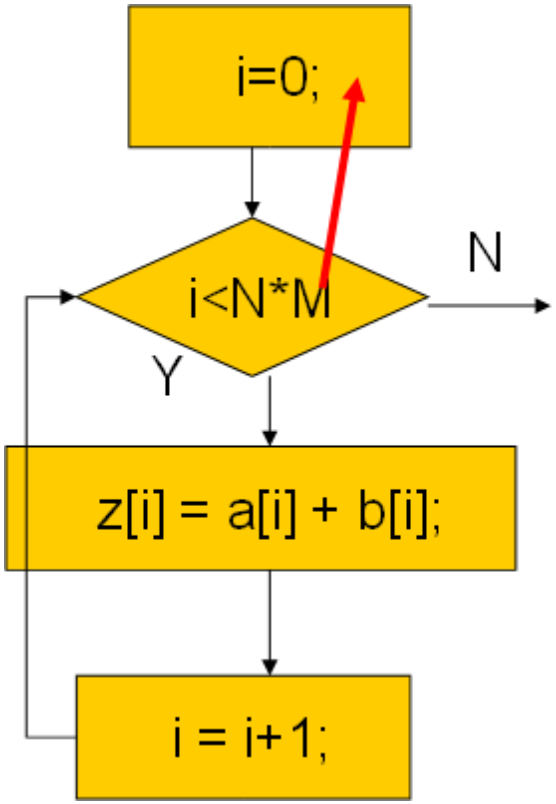
- Τα καλύτερα αποτελέσματα προκύπτουν από ανάλυση βελτιστοποιημένων εντολών, όχι κώδικα γλώσσας υψηλού επιπέδου:
  - Μη προφανείς μεταφράσεις καταστάσεων γλώσσας υψηλού επιπέδου σε εντολές.
  - Ο κώδικας μπορεί να μετακινηθεί.
  - Οι επιδράσεις της κρυφής μνήμης είναι δύσκολο να προβλεφθούν.

# Βελτιστοποιήσεις βρόχων

- Οι βρόχοι είναι σημαντικοί στόχοι βελτιστοποίησης.
- Βασικές βελτιστοποιήσεις βρόχων:
  - κίνηση κώδικα,
  - εξάλειψη επαγωγικής μεταβλητής,
  - μείωση έντασης (x*2 -> x<<1).

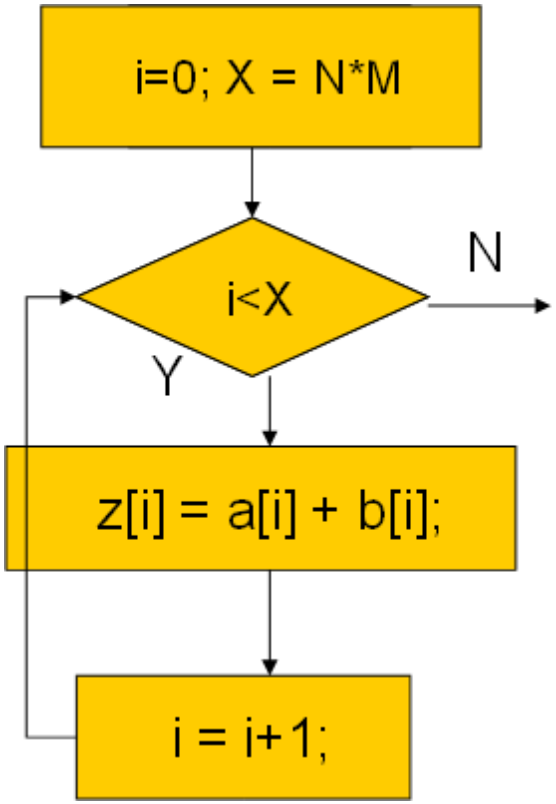
# Μετακίνηση κώδικα (1/2)

```
for (i=0; i<N*M; i++)
        z[i] = a[i] + b[i];
```



# Μετακίνηση κώδικα (2/2)

```
for (i=0; i<N*M; i++)
    z[i] = a[i] + b[i];
```



# Εξάλειψη επαγωγικής μεταβλητής

- Επαγωγική μεταβλητή: δείκτης βρόχου.

Θεωρήστε τον βρόχο:

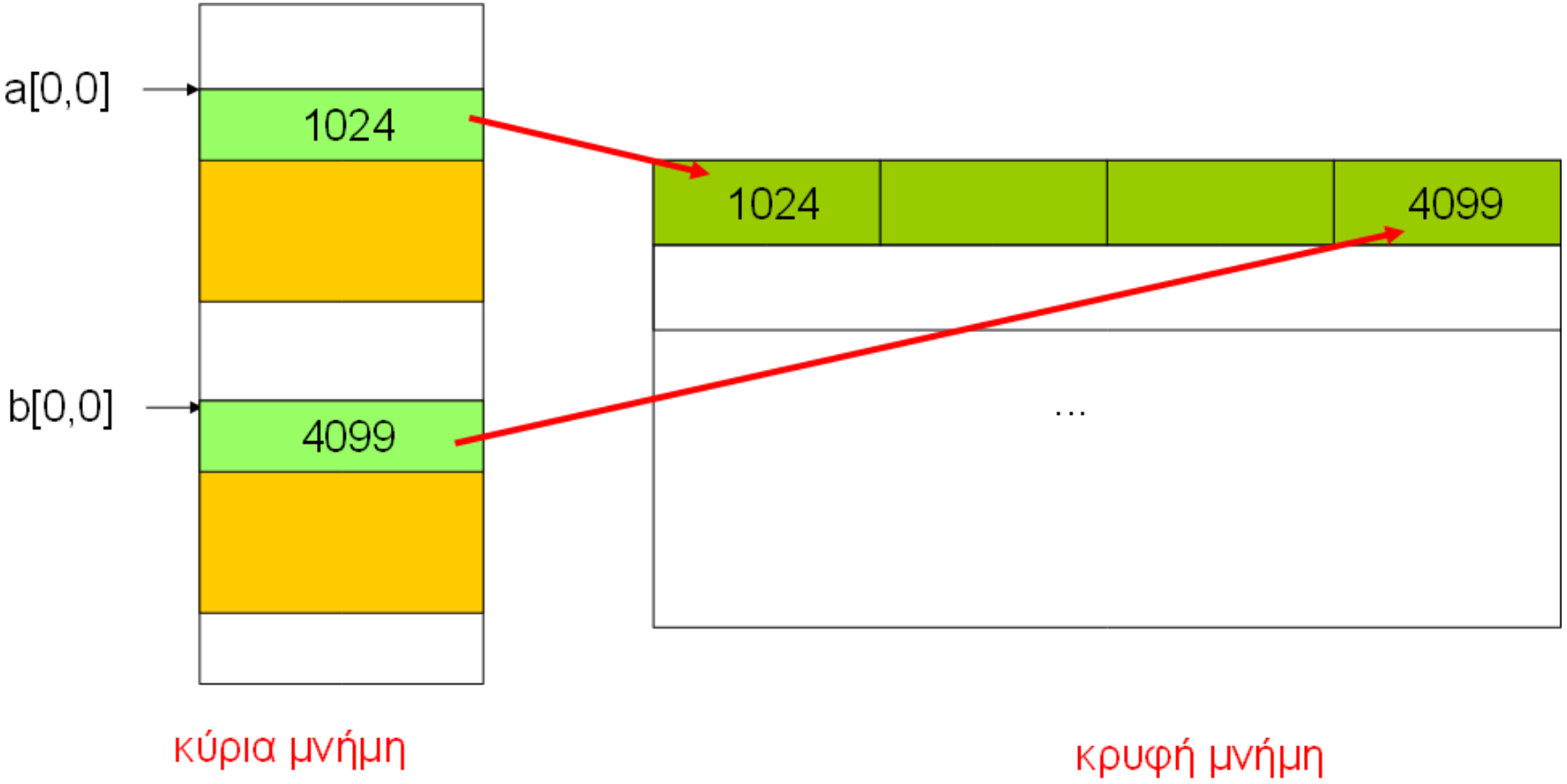
```
for (i=0; i<N; i++)
        for (j=0; j<M; j++)
                z[i,j] = b[i,j];
```

- Από το να υπολογίζουμε το i*M+j για κάθε πίνακα σε κάθε επανάληψη, θα χρησιμοποιήσουμε μια κοινή επαγωγική μεταβλητή για τους πίνακες, η οποία αυξάνεται στο τέλος του βρόχου.

# Ανάλυση κρυφής μνήμης

- Εμφωλευμένος βρόχος: σύνολο βρόχων, ο ένας μέσα σε άλλο.
- Τέλειος εμφωλευμένος βρόχος: χωρίς συνθήκες στο εσωτερικό του.
- Επειδή οι βρόχοι χρησιμοποιούν μεγάλες ποσότητες δεδομένων, οι συγκρούσεις της κρυφής μνήμης είναι συνηθισμένες.

# Συγκρούσεις πίνακα κρυφής μνήμης (1/2)



# Συγκρούσεις πίνακα κρυφής μνήμης (2/2)

- Τα στοιχεία του πίνακα συγκρούονται επειδή αντιστοιχίζονται στην ίδια γραμμή, ακόμα και αν δεν αντιστοιχίζονται ακριβώς στην ίδια τοποθεσία.
- Λύσεις:
  - μετακίνηση ενός πίνακα.
  - συμπλήρωση πίνακα.

# Συμβουλές βελτιστοποίησης απόδοσης

- Χρησιμοποίηση των καταχωρητών αποτελεσματικά.
- Χρησιμοποίηση προσβάσεων στο σύστημα μνήμης με τρόπο λειτουργίας σελίδας (page mode accesses).
- Ανάλυση της συμπεριφοράς της κρυφής μνήμης:
  - Οι συγκρούσεις εντολών μπορούν να χειριστούν ξαναγράφοντας τον κώδικα, αναδιάρθρωση.
  - Οι συγκρούσεις βαθμωτών δεδομένων μπορούν εύκολα να μετακινηθούν.
  - Οι συγκρούσεις δεδομένων σε πίνακες μπορούν να μετακινηθούν, τεχνική συμπλήρωσης.

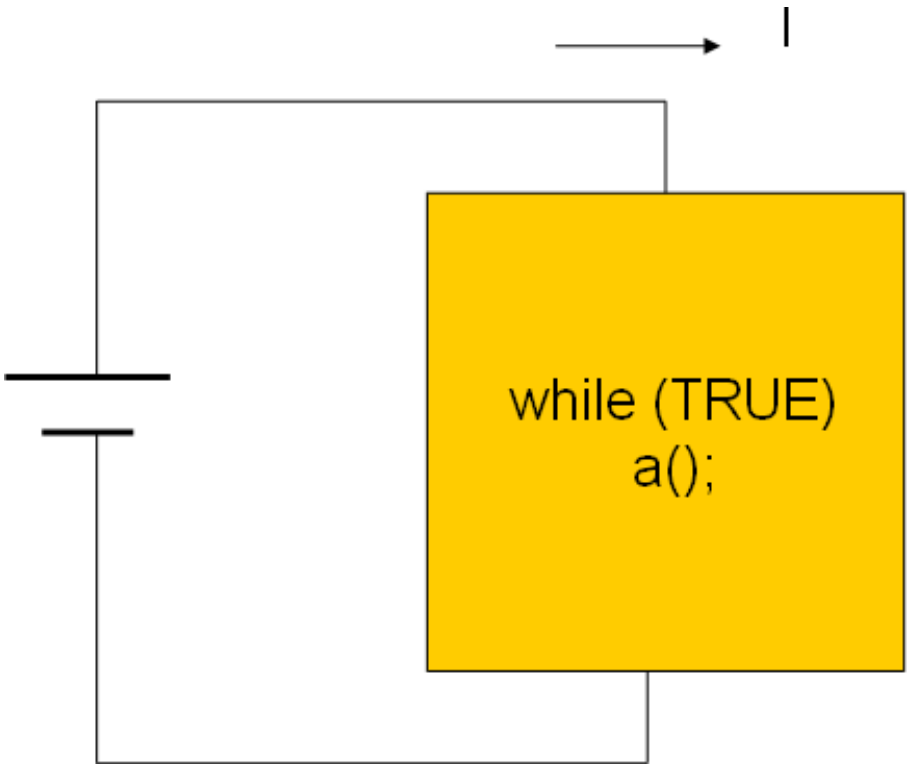
# Ανάλυση και Βελτιστοποίηση Ενέργειας και Ισχύος

# Βελτιστοποίηση ενέργειας / ισχύος

- Ενέργεια: ικανότητα παραγωγής έργου.
  - Ιδιαίτερα σημαντική για συστήματα που τροφοδοτούνται από μπαταρίες.
- Ισχύς: ρυθμός παραγωγής έργου.
  - Σημαντική ακόμα και σε συστήματα πρίζας---η ισχύς μετατρέπεται σε θερμότητα.

# Μέτρηση κατανάλωσης ενέργειας

- Εκτέλεση ενός μικρού τμήματος κώδικα βρόχου, μέτρηση της ροής του ρεύματος:



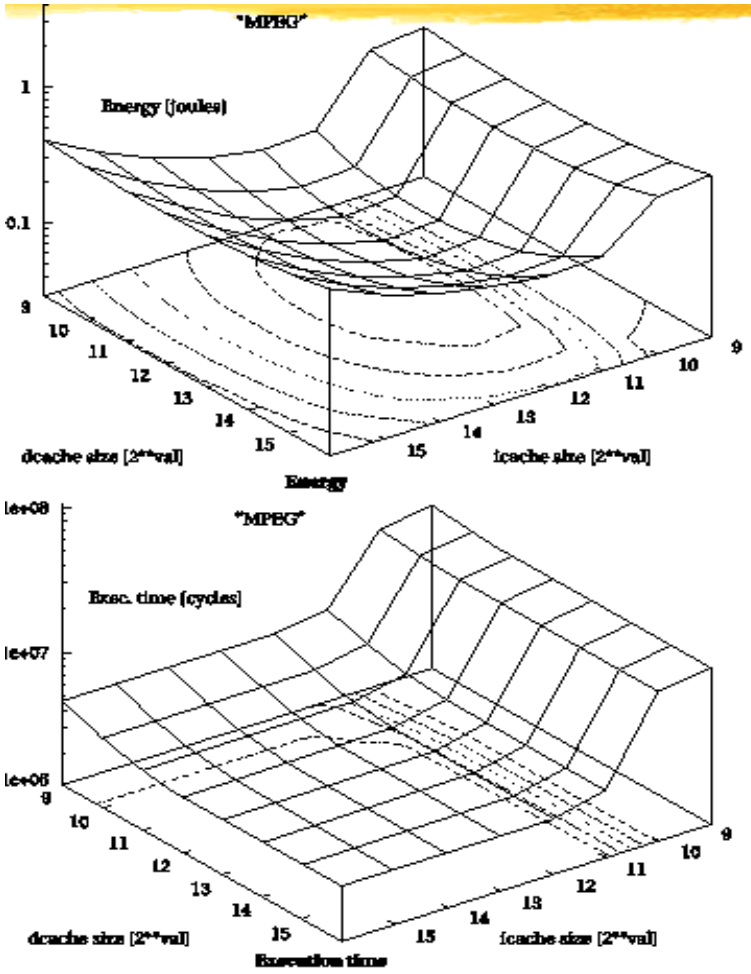
# Πηγές κατανάλωσης ενέργειας

- Σχετική ενέργεια ανά λειτουργία (Catthoor et al):
  - Μεταφορά μνήμης: 33
  - Προσπέλαση εξωτερικής Ε/Ε: 10
  - Εγγραφή SRAM: 9
  - Ανάγνωση SRAM: 4.4
  - Πολλαπλασιασμός: 3.6
  - Πρόσθεση: 1

# Η συμπεριφορά της κρυφής μνήμης είναι σημαντική

- Η σχέση της κατανάλωσης ενέργειας και του χρόνου εκτέλεσης μεταβάλλεται καθώς το μέγεθος της κρυφής μνήμης αλλάζει:
  - κρυφή μνήμη πολύ μικρή: το πρόγραμμα τρέχει αργά, το σύστημα καταναλώνει πολύ ενέργεια λόγω του υψηλού κόστους των προσβάσεων στην κύρια μνήμη.
  - Κρυφή μνήμη πολύ μεγάλη: η κατανάλωση ισχύος είναι πολύ μεγάλη.

# Ενέργεια και χρόνος εκτέλεσης σε συνάρτηση του μεγέθους της κρυφής μνήμης




[Li98] © 1998 IEEE

# Βελτιστοποίηση ενέργειας (1/2)

- Πρώτης τάξης βελτιστοποίηση:
  - υψηλή απόδοση = χαμηλή ενέργεια.
- Δεν υπάρχουν πολλές εντολές που ανταλλάσσουν ταχύτητα για ενέργεια.

# Βελτιστοποίηση ενέργειας (2/2)

- Αποτελεσματική χρήση των καταχωρητών.
- Εντοπισμός και εξάλειψη των συγκρούσεων κρυφής μνήμης.
- Το περιορισμένο ξετύλιγμα βρόχου μειώνει κάποια από την επιβάρυνση ελέγχου του βρόχου.
- Μείωση σταματημάτων (stall) διοχέτευσης.
- Η ενσωμάτωση διαδικασιών μπορεί να βοηθήσει: μειώνει τη σύνδεση, ωστόσο μπορεί να αυξήσει το thrashing (~μείωση απόδοσης) στην κρυφή μνήμη.

# Αποδοτικοί βρόχοι

- Γενικοί κανόνες:
  - Μη χρησιμοποιείτε κλήσεις συναρτήσεων.
  - Κρατήστε το σώμα του βρόχου μικρό για να επιτρέψετε την τοπική επανάληψη (μόνο διακλαδώσεις προς τα εμπρός).
  - Χρησιμοποιείστε μη προσημασμένο ακέραιο για μετρητή βρόχου.
  - Χρησιμοποιείστε <= για έλεγχο του μετρητή βρόχου.
  - Χρησιμοποιείστε καλύτερα το μεταγλωττιστή--- καθολική βελτίωση, διοχέτευση λογισμικού.

# Παράδειγμα επανάληψης βρόχου μόνης εντολής

```
STM #4000h,AR2
   ; φόρτωση δείκτη στην πηγή
STM #100h,AR3
    ; φόρτωση δείκτη στον προορισμό
RPT #(1024-1)
MVDD *AR2+,*AR3+
    ; μετακίνηση
```

# Ανάλυση και Βελτιστοποίηση Μεγέθους Προγράμματος

# Βελτιστοποίηση μεγέθους προγράμματος

- Στόχος:
  - μείωση κόστους υλικού της μνήμης,
  - μείωση κατανάλωσης ισχύος των μονάδων μνήμης.
- Δύο δυνατότητες:
  - Δεδομένα,
  - Εντολές.

# Ελαχιστοποίηση μεγέθους δεδομένων

- Επαναχρησιμοποίηση σταθερών, μεταβλητών, περιοχές προσωρινής αποθήκευσης δεδομένων σε αρκετά διαφορετικά σημεία στο πρόγραμμα.
  - Απαιτεί προσεκτική επαλήθευση για ορθότητα.
- Παραγωγή δεδομένων από τη χρησιμοποίηση των εντολών.

# Μείωση μεγέθους κώδικα

- Αποφυγή ενσωμάτωσης λειτουργιών.
- Επιλογή CPU με συμπαγείς εντολές.
- Χρήση ειδικευμένων εντολών όπου είναι δυνατόν.

# Επικύρωση και Δοκιμή Προγράμματος

# Επικύρωση και δοκιμή του προγράμματος

- Λειτουργεί;
- Σε αυτό το μέρος επικεντρωνόμαστε στη λειτουργική επικύρωση.
- Κύριοι τύποι στρατηγικών δοκιμών:
  - Το μαύρο κουτί δεν εξετάζει τον πηγαίο κώδικα.
  - Το διαφανές κουτί (άσπρο κουτί) εξετάζει τον πηγαίο κώδικα.

# Δοκιμή διαφανούς κουτιού

- Εξετάζει τον πηγαίο κώδικα για να καθορίσει εάν λειτουργεί:
  - Μπορείτε να δοκιμάσετε πραγματικά μια διαδρομή;
  - Λαμβάνετε την τιμή που περιμένατε κατά μήκος μιας διαδρομής;
- Διαδικασία δοκιμής:
  - Ελεγξιμότητα: παροχή των εισόδων στο πρόγραμμα.
  - Εκτέλεση.
  - Παρατηρησιμότητα: εξέταση αποτελεσμάτων.

# Ελέγχοντας και παρατηρώντας τα προγράμματα

```
firout = 0.0;
for (j=curr, k=0; j<N; j++, k++)
        firout += buff[j] * c[k];
for (j=0; j<curr; j++, k++)
        firout += buff[j] * c[k];
if (firout > 100.0) firout = 100.0;
if (firout < -100.0) firout = -100.0;
```

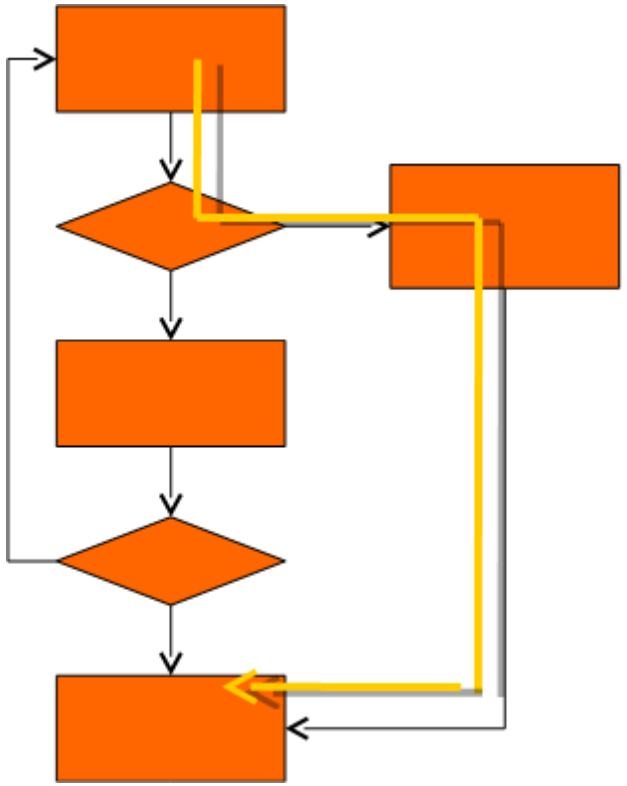
- Ελεγξιμότητα:
  - Πρέπει να γεμίσουμε την προσωρινή περιοχή προσωρινής αποθήκευσης με N τιμές στο κατάλληλο εύρος.
  - Διαφορετικός κώδικας ρυθμίζει πως αποκτούμε πρόσβαση στην προσωρινή περιοχή προσωρινής αποθήκευσης .
- Παρατηρησιμότητα:
  - Εξετάζουμε την τιμή του firout προτού εκτελεσθεί ο κώδικας περιορισμού.

# Διαδρομές εκτέλεσης και δοκιμές

- Οι διαδρομές εκτέλεσης είναι σημαντικές στις δοκιμές, όσο και η ανάλυση απόδοσης.
- Γενικά, ένας εκθετικός αριθμός διαδρομών μέσω του προγράμματος.
  - Ένδειξη ότι ορισμένες διαδρομές κυριαρχούν έναντι άλλων.
  - Ευρετικός περιορισμός των διαδρομών.

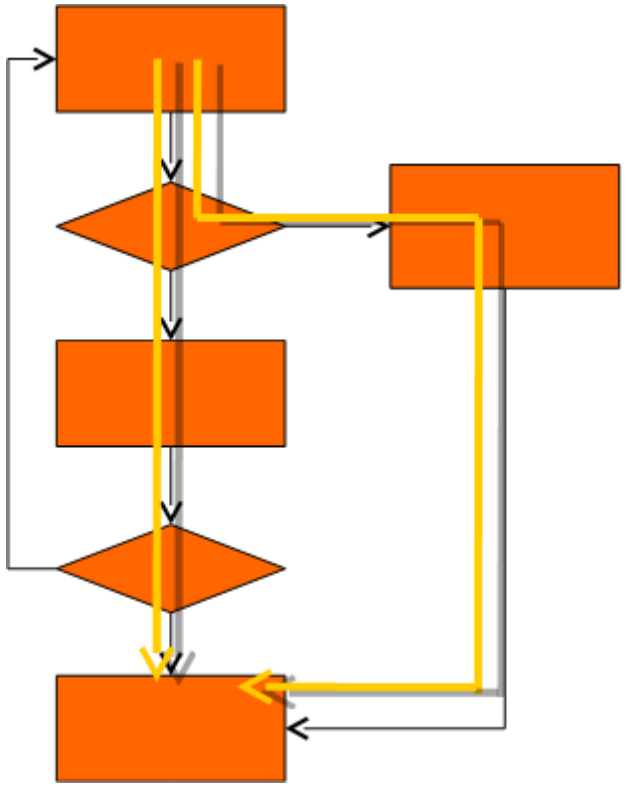
# Επιλέγοντας τις διαδρομές για δοκιμή (1/3)

- Δύο λογικές επιλογές:
  - Να εκτελέσουμε κάθε εντολή τουλάχιστον μία φορά.
  - Να εκτελέσουμε κάθε διεύθυνση μιας διακλάδωσης τουλάχιστον μία φορά.
- Αυτές οι συνθήκες είναι ισοδύναμες για τις δομημένες γλώσσες προγραμματισμού χωρίς gotos.



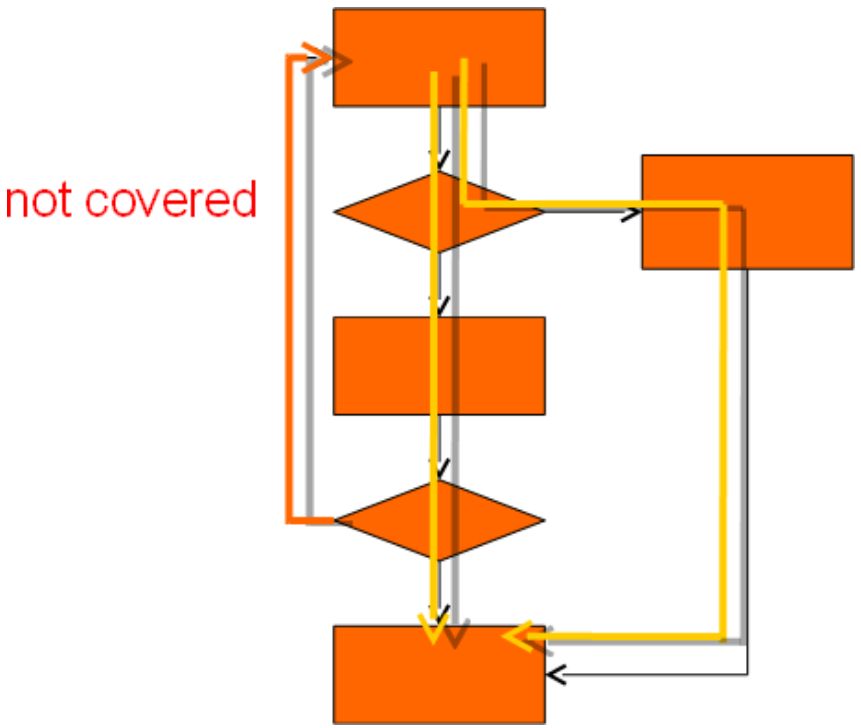
# Επιλέγοντας τις διαδρομές για δοκιμή (2/3)

- Δύο λογικές επιλογές:
  - Να εκτελέσουμε κάθε εντολή τουλάχιστον μία φορά.
  - Να εκτελέσουμε κάθε διεύθυνση μιας διακλάδωσης τουλάχιστον μία φορά.
- Αυτές οι συνθήκες είναι ισοδύναμες για τις δομημένες γλώσσες προγραμματισμού χωρίς gotos.



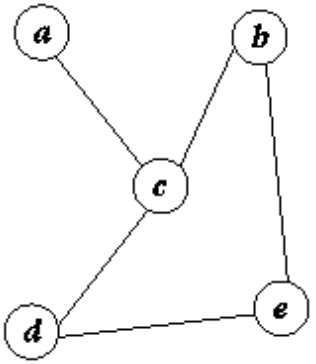
# Επιλέγοντας τις διαδρομές για δοκιμή (3/3)

- Δύο λογικές επιλογές:
  - Να εκτελέσουμε κάθε εντολή τουλάχιστον μία φορά.
  - Να εκτελέσουμε κάθε διεύθυνση μιας διακλάδωσης τουλάχιστον μία φορά.
- Αυτές οι συνθήκες είναι ισοδύναμες για τις δομημένες γλώσσες προγραμματισμού χωρίς gotos.



# Διαδρομές βάσης

- Προσεγγιστικά CDFG με μη προσανατολισμένο γράφημα.
- Σε μη προσανατολισμένα γραφήματα, μπορούμε να εφαρμόσουμε διαδρομές βάσης.
  - Όλες οι διαδρομές είναι γραμμικός συνδυασμός των διαδρομών βάσης.




Graph

| | a | b | c | d | e |
|---|---|---|---|---|---|
| a | 0 | 0 | 1 | 0 | 0 |
| b | 0 | 0 | 1 | 0 | 1 |
| c | 1 | 1 | 0 | 1 | 0 |
| d | 0 | 0 | 1 | 0 | 1 |
| e | 0 | 1 | 0 | 1 | 0 |

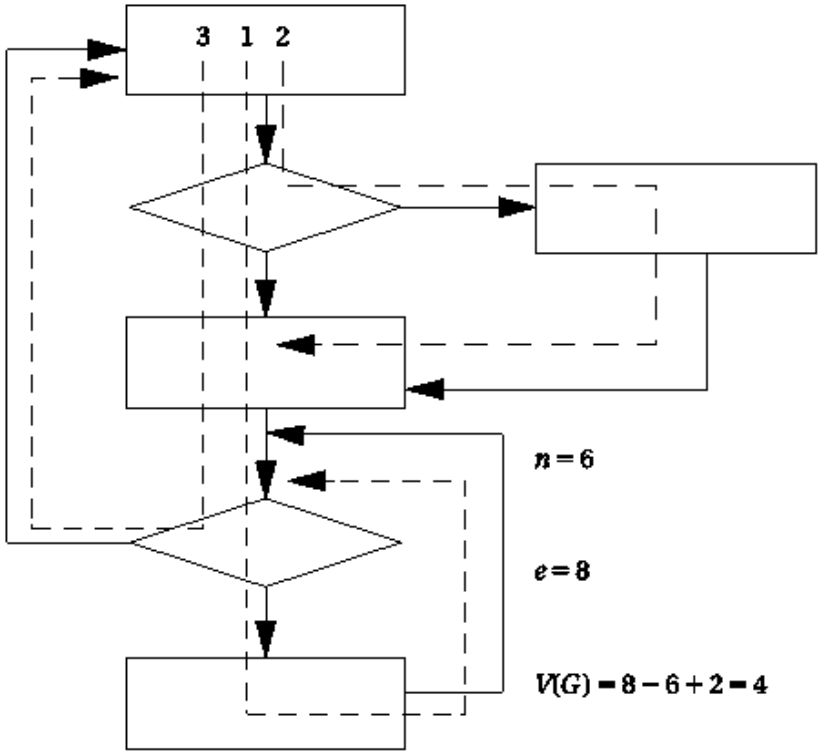
Incidence matrix

| | | | | | |
|---|---|---|---|---|---|
| a | 1 | 0 | 0 | 0 | 0 |
| b | 0 | 1 | 0 | 0 | 0 |
| c | 0 | 0 | 1 | 0 | 0 |
| d | 0 | 0 | 0 | 1 | 0 |
| e | 0 | 0 | 0 | 0 | 1 |

Basis set

# Κυκλοματική πολυπλοκότητα

- Η κυκλωματική πολυπλοκότητα είναι ένα άνω όριο στο μέγεθος του συνόλου βάσης:
  - e = # ακμών
  - n = # κόμβων
  - p = αριθμός στοιχείων στο γράφημα
  - M = e – n + 2p.



# Δοκιμή διακλάδωσης

- Ευρετική για δοκιμές διακλαδώσεων.
  - Δοκιμή αληθών και ψευδών διακλαδώσεων μιας συνθήκης.
  - Δοκιμή κάθε απλής συνθήκης τουλάχιστον μία φορά.

# Παράδειγμα δοκιμής διακλάδωσης 1

- Αληθές:
  - if (a || (b >= c)) { printf(“OK\n”); }
- Ψευδές:
  - if (a && (b >= c)) { printf(“OK\n”); }

- Δοκιμή:
  - a = F
  - (b >=c) = T
- Παράδειγμα:
  - Αληθές: [0 || (3 >= 2)] = T
  - Ψευδές: [0 && (3 >= 2)] = F

# Παράδειγμα δοκιμής διακλάδωσης 2

- Αληθές:

```
if ((x == good_pointer) && x->field1 == 3)) { printf("got the value\n"); }
```

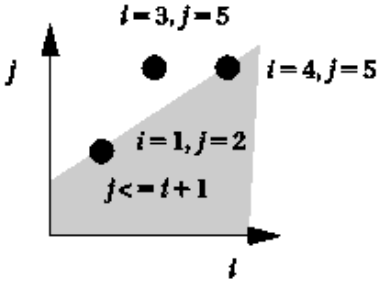
- Ψευδές:

```
if ((x = good_pointer) && x->field1 == 3)) { printf("got the value\n"); }
```

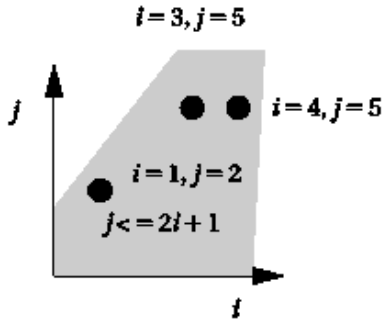
- Λάθος κώδικας αλλάζει τον δείκτη.
  - Η εκχώρηση επιστρέφει νέα LHS στη C.
- Δοκιμή που συλλαμβάνει σφάλμα:
  - (x != good_pointer) && x->field1 = 3)

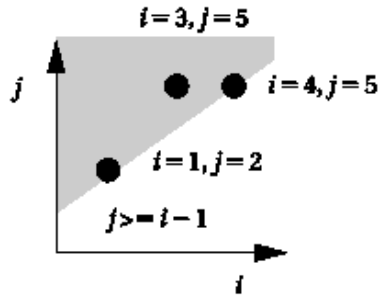
# Δοκιμή περιοχής

- Εμπειρική δοκιμή που επικεντρώνεται στις γραμμικές ανισότητες.
  - Δοκιμή κάθε πλευράς + όριο της ανισότητας.




Correct test

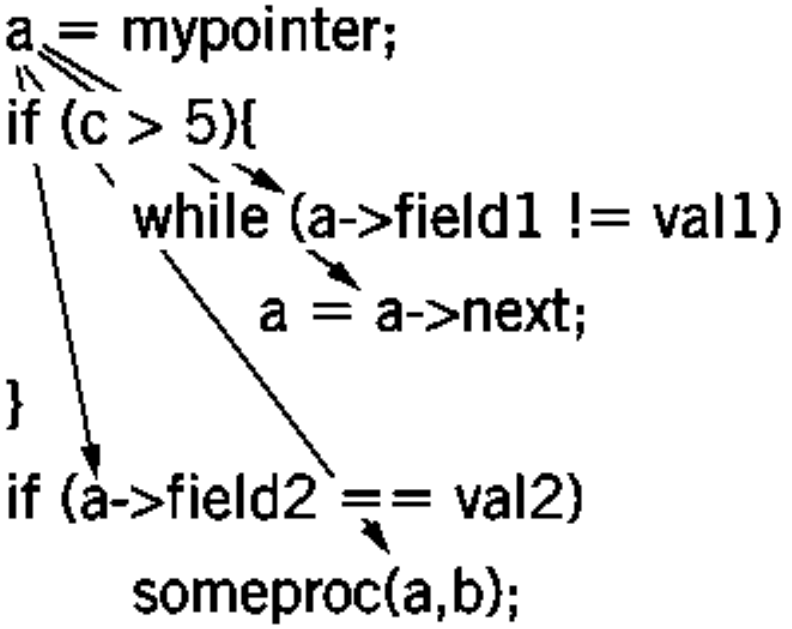



Incorrect tests

# Ζευγάρια ορισμού-χρήσης

- Μεταβλητή ορισμού-χρήσης:
  - Καθορίζεται όταν γίνεται μια ανάθεση στην μεταβλητή.
  - Χρησιμοποιείται όταν εμφανίζεται στη δεξιά πλευρά μιας ανάθεσης.
- Δοκιμή κάθε ζευγαριού ορισμού-χρήσης.
  - Απαιτεί δοκιμή της σωστής διαδρομής.



# Δοκιμές βρόχων

- Οι βρόχοι απαιτούν ειδικευμένες δοκιμές για να δοκιμαστούν αποδοτικά.
- Εμπειρική στρατηγική δοκιμής:
  - Παράκαμψη του βρόχου εξ ολοκλήρου.
  - Μία επανάληψη του βρόχου.
  - Δύο επαναλήψεις του βρόχου.
  - # επαναλήψεων σημαντικά κάτω από τον μέγιστο αριθμό.
  - n-1, n, n+1 επαναλήψεις όπου n ο μέγιστος αριθμός.

# Δοκιμή μαύρου κουτιού

- Συμπληρώνει τη δοκιμή διαφανούς κουτιού.
  - Ίσως απαιτήσει πολλές δοκιμές.
- Δοκιμές λογισμικού με διαφορετικούς τρόπους.

# Διανύσματα δοκιμής μαύρου κουτιού

- Τυχαίες δοκιμές.
  - Μπορεί η κατανομή του βάρους να βασίζεται στις προδιαγραφές λογισμικού.
- Δοκιμές παλινδρόμησης.
  - Δοκιμές προηγούμενων εκδόσεων, σφαλμάτων κλπ.
  - Μπορεί να είναι δοκιμές προηγούμενων εκδόσεων διαφανούς κουτιού.

# Πόση δοκιμή είναι αρκετή;

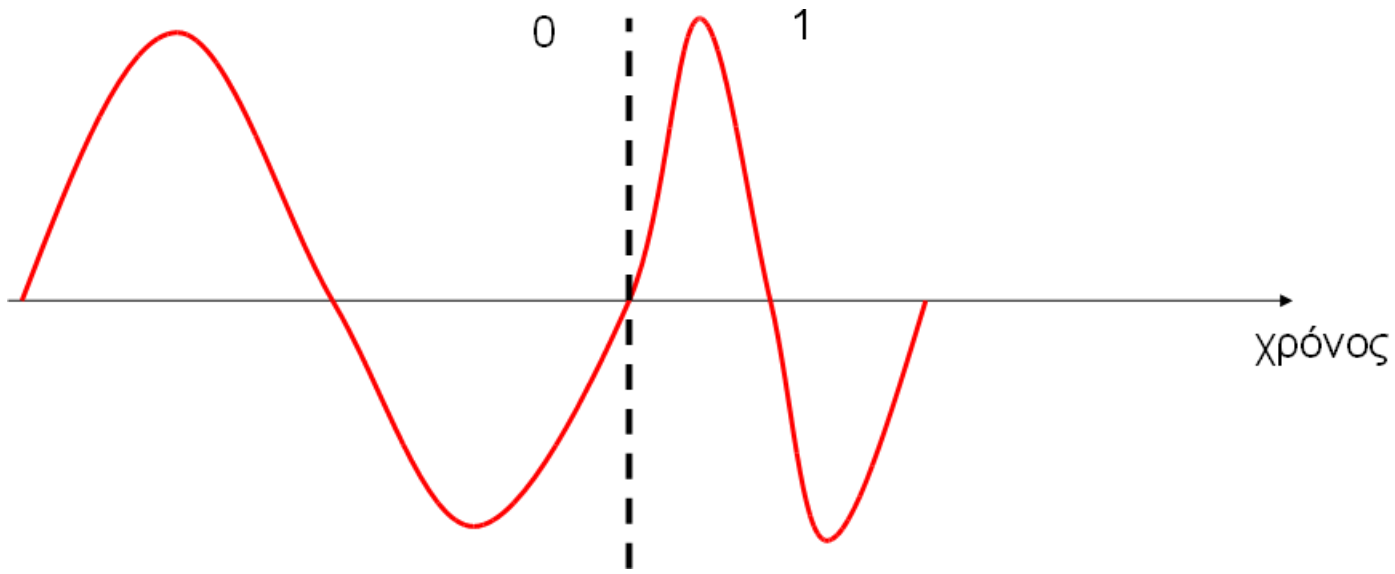
- Οι εξαντλητικές δοκιμές είναι κάτι μη πρακτικό.
- Μια σημαντική μέτρηση είναι η δοκιμή απόδοσης---διαφυγή σφαλμάτων στο πεδίο.
- Καλές οργανώσεις μπορούν να δοκιμάσουν το λογισμικό να δώσουν πολύ χαμηλά ποσοστά αναφοράς σφαλμάτων πεδίου.
- Η έγχυση σφάλματος μετράει την ποιότητα της δοκιμής:
  - Προσθήκη γνωστών σφαλμάτων.
  - Τρέξιμο των δοκιμών.
  - Καθορισμός % σφαλμάτων έγχυσης που πιάστηκαν.

# Παράρτημα

# Παράδειγμα Σχεδίασης: Modem λογισμικού

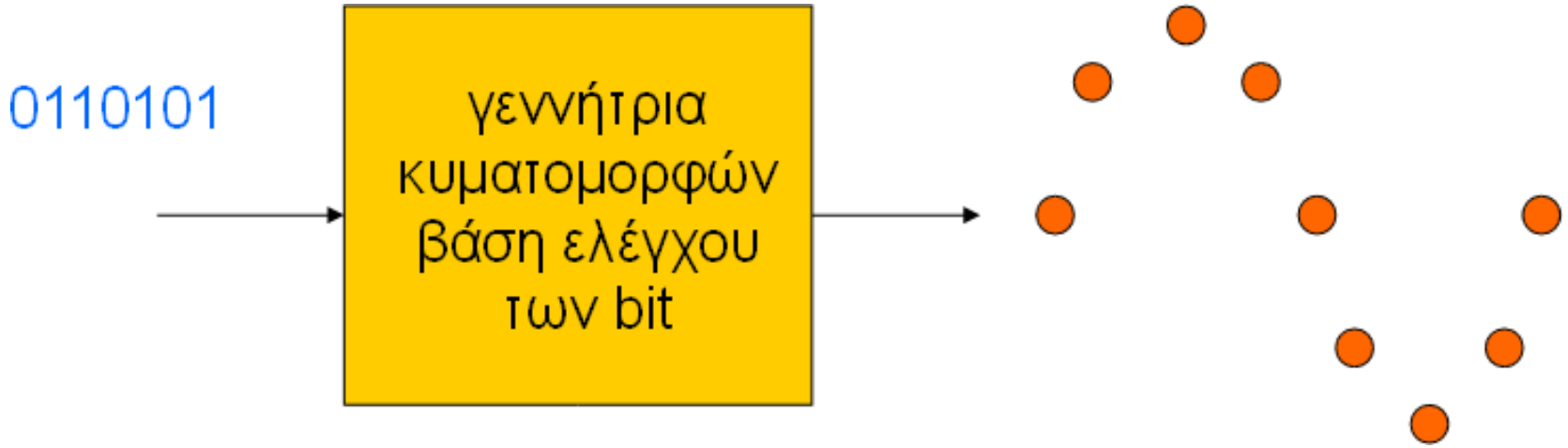
# Θεωρία της λειτουργίας

- Διαμόρφωση μετατόπισης συχνότητας (frequency-shift keying - FSK),
- διαφορετικές συχνότητες από 0 και 1.

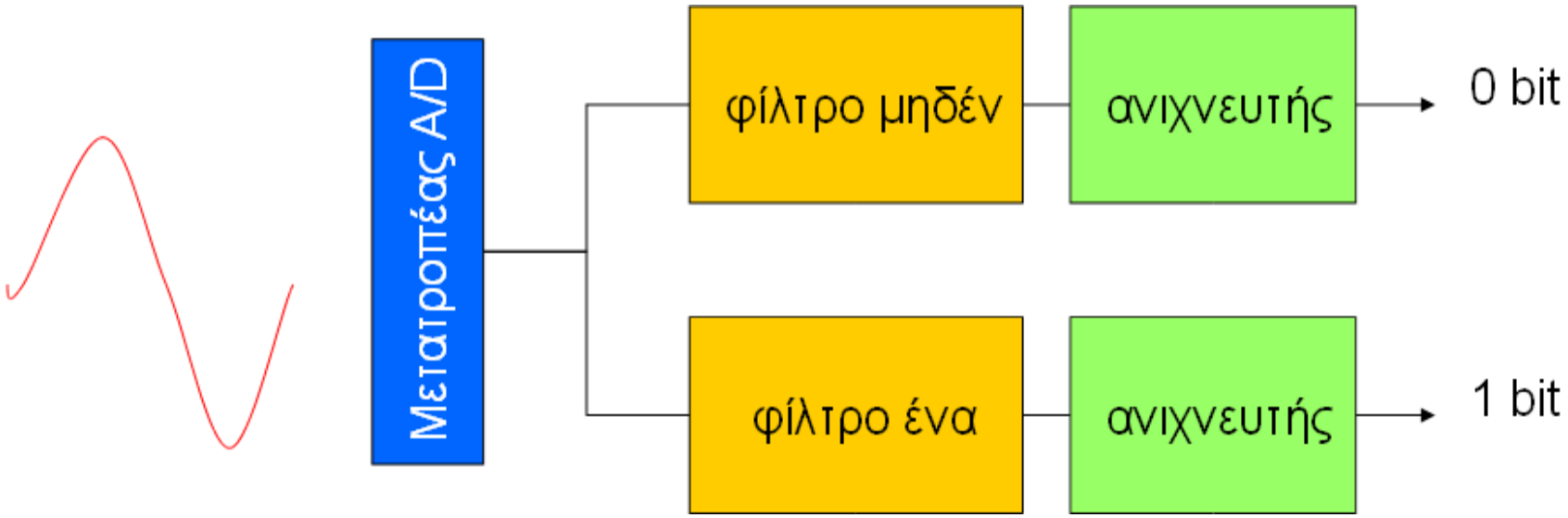


# Κωδικοποίηση FSK

- Παραγωγή κυματομορφών βασισμένες στα τρέχοντα δυαδικά ψηφία (bit):

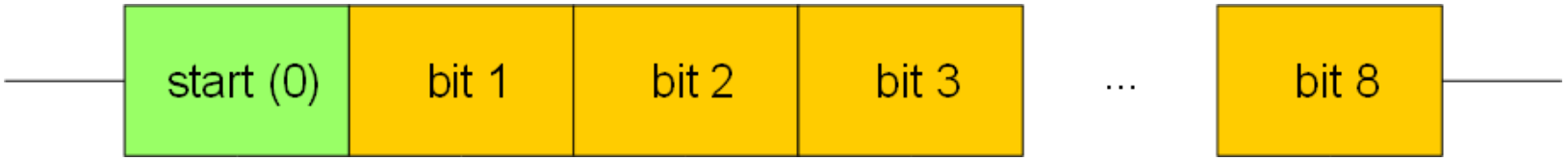


# Αποκωδικοποίηση FSK



# Σχήμα μετάδοσης

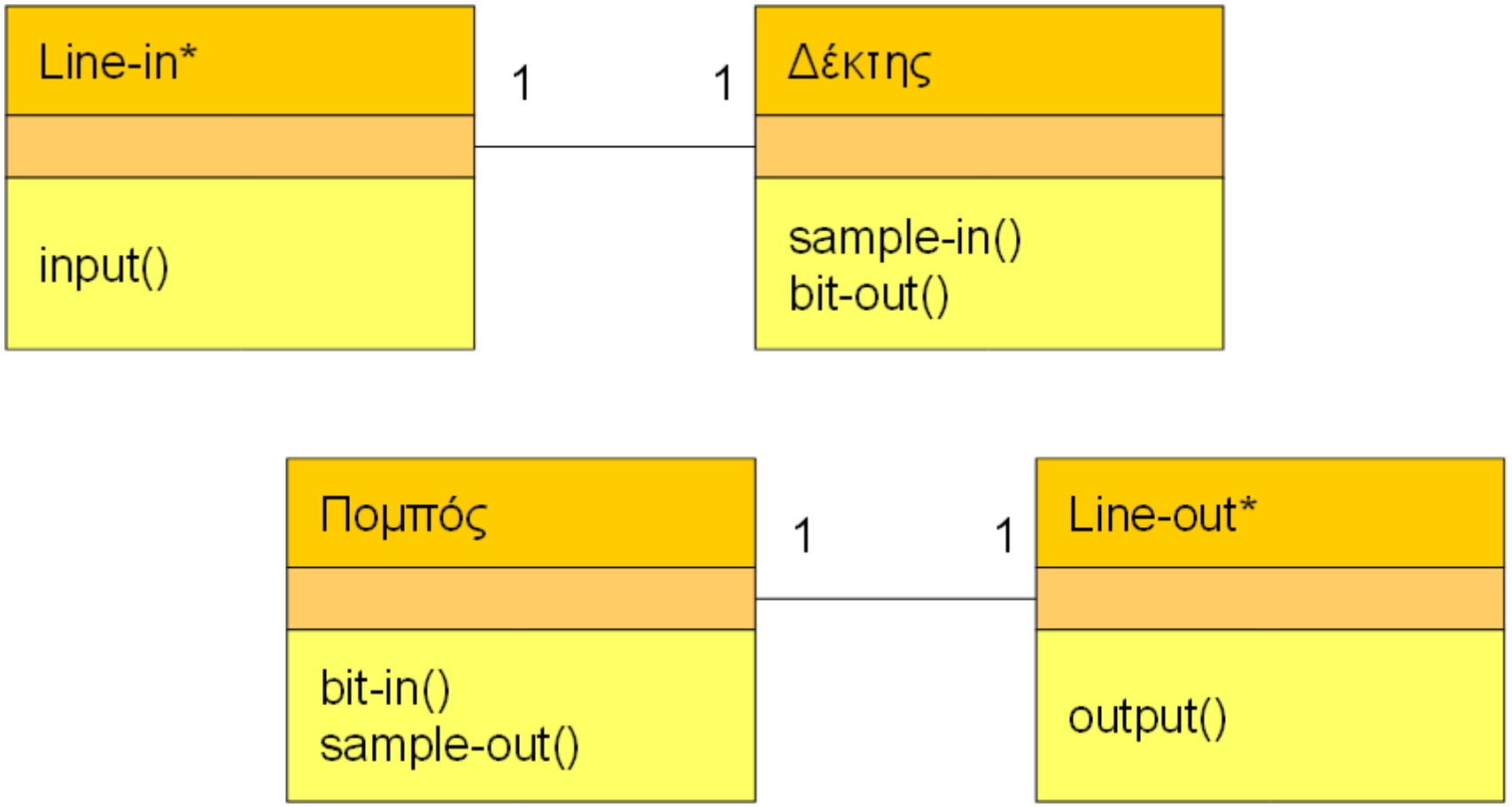
- Αποστολή δεδομένων σε bytes των 8-bit. Αυθαίρετη απόσταση μεταξύ των bytes.
- Το byte ξεκινά, με το bit έναρξης 0.
- Ο δέκτης μετράει το μήκος του bit έναρξης για να συγχρονιστεί στα υπόλοιπα 8 bits.



# Απαιτήσεις

| | |
|---|---|
| Είσοδοι | Είσοδος αναλογικού ήχου, κουμπί reset. |
| Έξοδοι | Έξοδος αναλογικού ήχου, οθόνη LED bit. |
| Λειτουργίες | Πομπός: Στέλνει δεδομένα σε bytes των 8-bit συν το bit έναρξης.<br>Δέκτης: Ανιχνεύει αυτόματα τα bytes. Εμφανίζει το τρέχον ληφθέν bit σε LED. |
| Απόδοση | 1200 baud. |
| Κόστος κατασκευής | Κυριαρχείται από τον μικροεπεξεργαστή και την αναλογική Ε/Ε. |
| Ισχύς | Τροφοδοτείται από AC. |
| Φυσικό μέγεθος | Αρκετά μικρό και ελαφρύ. |

# Προσδιορισμός

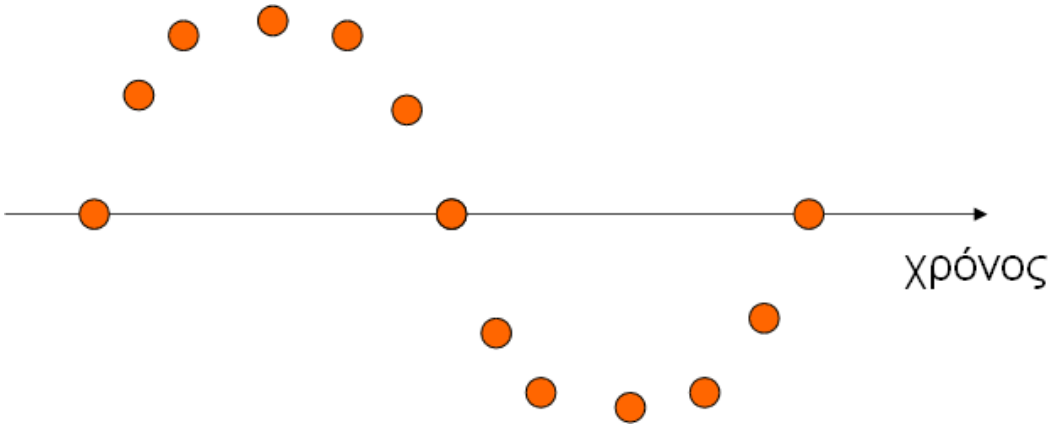


# Αρχιτεκτονική συστήματος

- Χειριστές διακοπών για τα δείγματα:
  - είσοδος και έξοδος.
- Πομπός.
- Δέκτης.

# Πομπός

- Παραγωγή κυματομορφής μέσω αναζήτησης σε πίνακα.
- float sine_wave[N_SAMP] = { 0.0, 0.5, 0.866, 1, 0.866, 0.5, 0.0, -0.5, -0.866, -1.0, -0.866, -0.5, 0};



# Δέκτης

- Τα φίλτρα (FIR για απλότητα) χρησιμοποιούν κυκλικές περιοχές προσωρινής αποθήκευσης για να αποθηκεύουν τα δεδομένα.
- Ένα χρονόμετρο μετράει το μήκος bit.
- Μια μηχανή κατάστασης αναγνωρίζει τα bit έναρξης, τα bit δεδομένων.

# Πλατφόρμα υλικού

- CPU.
- Μετατροπέας A/D.
- Μετατροπέας D/A.
- Χρονόμετρο.

# Σχεδίαση συστατικών και δοκιμή

- Εύκολη η δοκιμή πομπού και δέκτη στην πλατφόρμα υπολογιστή υπηρεσίας.
- Ο πομπός μπορεί να επαληθευθεί στις εξόδους των ηχείων.
- Λειτουργίες επαλήθευσης δέκτη:
  - αναγνώριση bit έναρξης,
  - αναγνώριση bit δεδομένων.

# Ολοκλήρωση συστήματος και δοκιμή

- Χρήση κατάστασης βρόχου επιστροφής για τη δοκιμή στοιχείων έναντι ενός άλλου.
  - Βρόχος επιστροφής στο λογισμικό ή συνδέοντας D/A και A/D μετατροπείς.

# Βιβλιογραφία

Χρησιμοποιήθηκε υλικό από παρουσιάσεις των:

- Wayne Wolf, Overheads for *Computers as Components 2nd ed.* ,2008 [5.2 έως 5.10]

# Τέλος Ενότητας