Συστήματα Παράλληλης & Κατανεμημένης Επεξεργασίας

Ενότητα 13: Είδη συστοιχιών. Διανυσματικοί Υπολογιστές. Επεξεργαστές μητρώου, Επεξεργαστές Blitzen, Επεξεργαστές Transputters

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

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 της Ενότητας

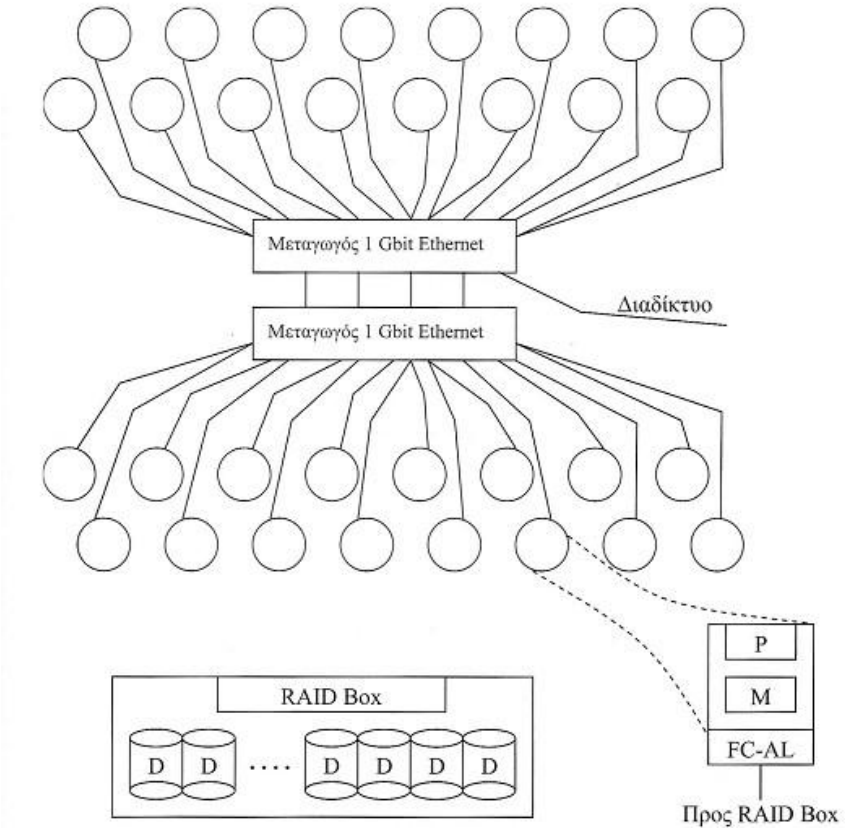
- Η κατανόηση της διανυσματοποίησης.
- Η περιγραφή των επεξεργαστών μητρώου.

Είδη Συστοιχιών

- Στα βασικά κριτήρια σχεδίασης μιας συστοιχίας ανήκουν:
  - Κόστος απόκτησης.
  - Κόστος συντήρησης.
  - Εφαρμογές που θα χρησιμοποιηθούν (οι επιστημονικές εφαρμογές απαιτούν υπολογιστική ισχύ, οι εφαρμογές βάσεων δεδομένων και επεξεργασίας συναλλαγών απαιτούν αποθηκευτικό χώρο).
  - Υπάρχουν 3 βασικές αρχιτεκτονικές...

Αρχιτεκτονική 1 από 3: Ύπαρξη τοπικών δίσκων

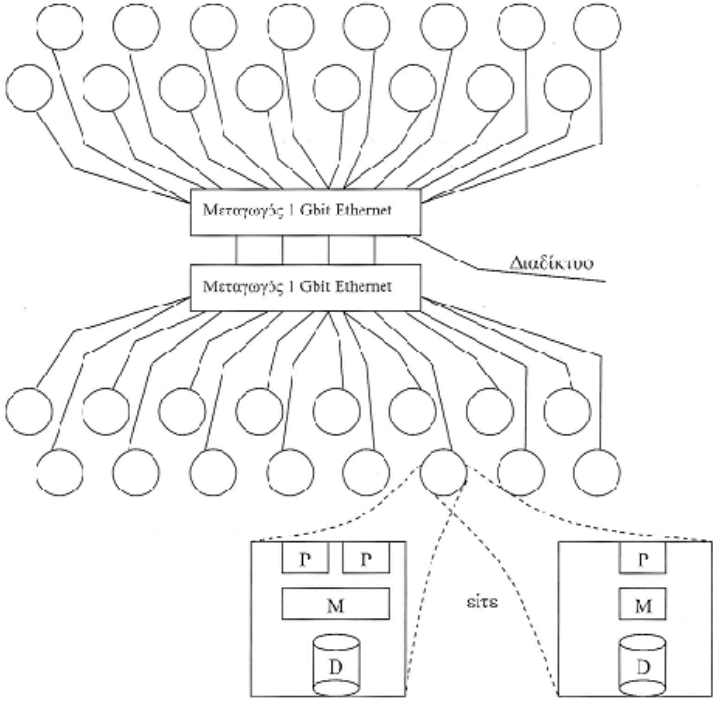
- Χαμηλό κόστος απόκτησης.
- Κάθε κόμβος είναι πλήρης υπολογιστής.
- Χρησιμοποιούνται πολλαπλοί πυρήνες ανά κόμβο.
- Χρησιμοποιείται GB ethernet.

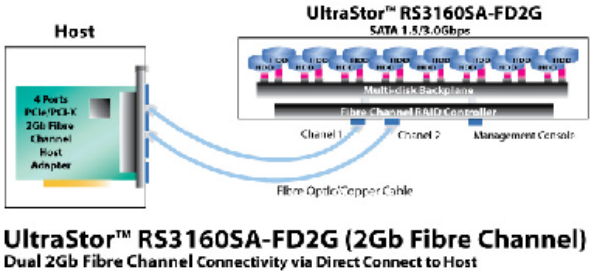


Αρχιτεκτονική 2 από 3: Ύπαρξη δικτυακού χώρου

- Χρησιμοποιείται όπου είναι κρίσιμη η συνέπεια των δεδομένων.
- Χρήση RAID.
- Χρησιμοποιούνται FCAL Host Adapter για σύνδεση κάθε κόμβου.
- Χρησιμοποιούνται οπτικές ίνες.

ATTO Celerity FC-42ES
4GB/second

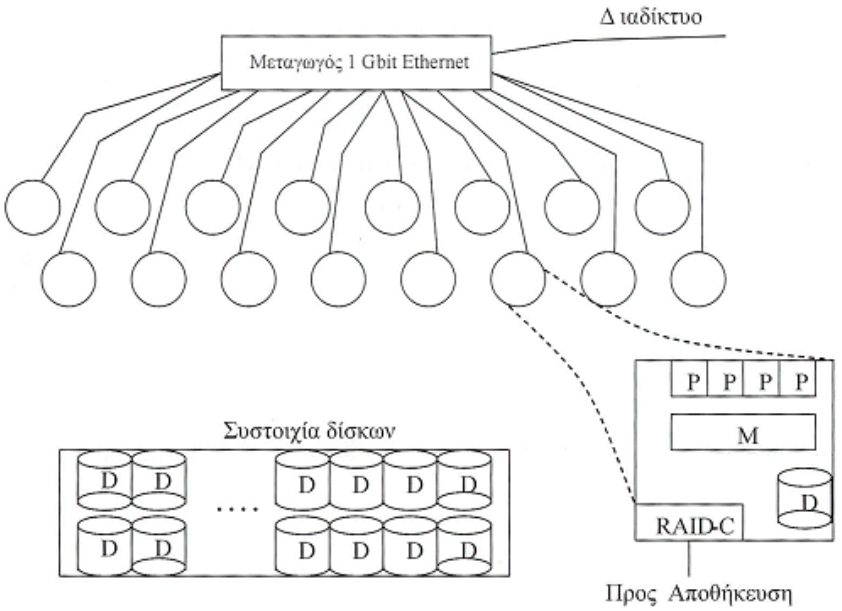




UltraStor™ RS3160SA-FD2G (2Gb Fibre Channel)
Dual 2Gb Fibre Channel Connectivity via Direct Connect to Host

Αρχιτεκτονική 3 από 3: Επεξεργασία Συναλλαγών

- Υπάρχει τοπικός δίσκος για cache, OS, κτλ.
- Υπάρχει δικτυακή αποθήκευση στη μεγάλη κοινή βάση δεδομένων.



Παράδειγμα: Google Search Engine

- Κάθε ερώτημα ξεκινάει αναζήτηση σε βάση δεδομένων εκατοντάδων TB.
- Η συστοιχία έχει πάνω από 15000 υπολογιστές από pentium 544Mhz έως Xeon.
- Κάθε υπολογιστής έχει 1 ή 2 δίσκους από 80GB.
- Οι υπολογιστές τοποθετούνται συρταρωτά σε ικριώματα.
- Κάθε ικρίωμα έχει 80 υπολογιστές.

Προβλήματα Google Search Engine

- Διαχείριση ενέργειας.
- Απαγωγή θερμότητας.
- Ίση κατανομή φορτίου (λαμβάνοντας υπόψιν τις διαφορετικές γενιές).
- Ανοχή στα σφάλματα, διαχείριση κόμβων, αντικατάσταση.
- Συνεχής επέκταση του συστήματος.
- Υψηλές απαιτήσεις:
  - Ανταπόκριση σε λιγότερο από 0.5sec.
  - Αναζήτηση σε δισεκατομμύρια διευθύνσεις URL.
  - 24/7 λειτουργία.

Παράδειγμα: Jaguar (Top500@2010)

- Jaguar Cray XT5.
- 18688 γενικής επεξεργασίας κόμβους.
- Ειδικοί κόμβοι για login/service.
- Κάθε κόμβος έχει 6core Opteron 2435 @ 2.6Ghz και μνήμη 16GB.
  - Συνολικά: 224256 πυρήνες, 300 PetaByte, 2.3 PetaFlops.
- Λειτουργικό Σύστημα Cray Linux.
- Δίκτυο διασύνδεσης: SeaStar 2+ (57,6 GB/second).

Διανυσματικοί Υπολογιστές

Τι είναι διανυσματοποίηση (vectorization); (1/2)

- Είναι μια ειδική τεχνική της παραλληλοποίησης, κατά την οποία λειτουργίες που θα εκτελούνται ατομικά και μια-μια σε ένα νήμα, τροποποιούνται ώστε να εκτελεστούν παράλληλα σε ομάδες.
- Απαιτείται υποστήριξη υλικού.
- Στο παρελθόν υπήρχαν υπολογιστές που εκτελούσαν μόνο διανυσματικές πράξεις (και ονομάζονταν διανυσματικοί υπολογιστές).
- Οι σύγχρονοι επεξεργαστές έχουν υιοθετήσει κάποια στοιχεία αυτών των υπολογιστών.

Τι είναι διανυσματοποίηση (vectorization); (2/2)

- Επίσης, η διανυσματοποίηση ονομάζεται η τεχνική της μετατροπής ενός βαθμωτού προγράμματος σε διανυσματικό, προκειμένου να αυξηθεί η απόδοση εκτέλεσης σε υπολογιστή που υποστηρίζει διανυσματοποίηση.
- Υπάρχουν πολύ σημαντικά οφέλη από τη χρήση τόσο στους προσωπικούς υπολογιστές, όσο και στους παράλληλους.
- Αποτελεί ενεργή ερευνητική περιοχή.

Χαρακτηριστικά Διανυσματικών Υπολογιστών

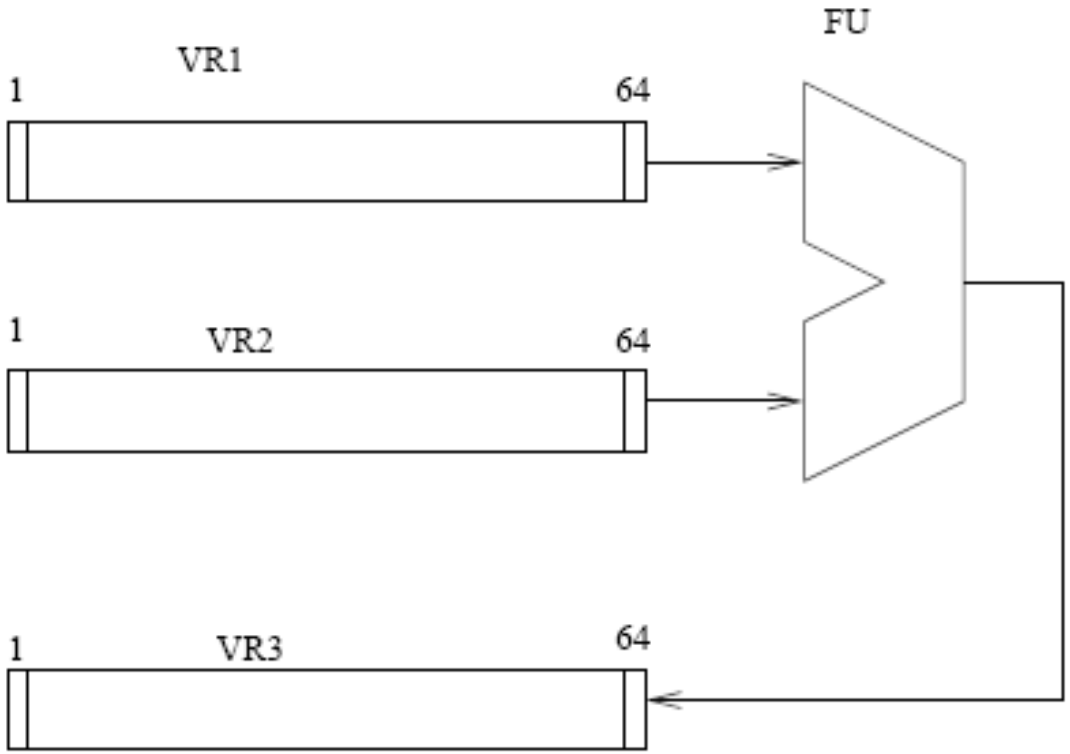
- Οι διανυσματικοί επεξεργαστές παρέχουν λειτουργίες πολύ μεγάλης ταχύτητας σε διανύσματα, δηλαδή σε γραμμικούς πίνακες με αριθμούς.
- Χρησιμοποιούν τον παραλληλισμό αγωγού.
  - Αγωγοί εντολών για φάσεις εκτέλεσης εντολών.
  - Αριθμητικοί Αγωγοί για αποτίμηση αριθμητικών εκφράσεων.
- Είναι πιο κοντά στο ακολουθιακό μοντέλο Von Neumann (έννοια του αποθηκευμένου προγράμματος).
- Δεν έχουν προβλήματα συγχρονισμού.
- Για μεγάλο διάστημα ήταν οι TOP υπολογιστές, ιδίως για προβλήματα επιστημονικών υπολογισμών.

Παράδειγμα εντολής vector processor

- Μια τυπική λειτουργία διανύσματος ήταν η πρόσθεση δύο διανυσμάτων πραγματικών στοιχείων 64bit.
- Σε τυπικούς υπολογιστές απαιτείται βρόχος επανάληψης.
- Σε διανυσματικούς υπολογιστές ήταν μια εντολή.

Μια διανυσματική εντολή αντιστοιχούσε σε δεκάδες ή εκατοντάδες εντολές σε τυπικό υπολογιστή.

Βασική Αρχιτεκτονική Διανυσματικού Επεξεργαστή



Ποια είναι τα πλεονεκτήματα της διανυσματικής επεξεργασίας

- Η χρήση των ειδικών διανυσματικών εντολών από τον προγραμματιστή, έδειχνε στο υλικό ότι μπορούσαν να γίνουν παράλληλα αυτές οι πράξεις οπότε δεν απαιτούνται ειδικό υλικό για να ελέγχει εξαρτήσεις.
- Υπήρχε υλικό όμως για να ελέγχει για εξαρτήσεις ανάμεσα σε διανυσματικές εντολές, το οποίο δεν απαιτούνται να είναι τόσο πολύπλοκο. Ανάμεσα όμως στον έλεγχο εξαρτήσεων μπορούσαν να γίνουν πάρα πολλές πράξεις.
- Η πρόσβαση στη μνήμη των διανυσματικών εντολών ακολουθεί κάποια γνωστά πρότυπα. Μπορεί να χρησιμοποιηθεί ειδικού τύπου interleaved memory. Υπάρχει λοιπόν πολύ μικρή καθυστέρηση στη μεταφορά δεδομένων από τη μνήμη.
- Επειδή ολόκληροι βρόχοι επανάληψης αντικαθίστανται από μια εντολή, τότε εξαφανίζονται οι κίνδυνοι ελέγχου (control hazards) από το pipeline.

Διανυσματικοί επεξεργαστές

- Οι περισσότεροι διανυσματικοί επεξεργαστές επέτρεπαν πολλαπλές ταυτόχρονες διανυσματικές εντολές, οπότε επιτυγχάνονταν ακόμη μεγαλύτερες αποδόσεις.
- Χρησιμοποιήθηκαν ευρέως σε επιστημονικές εφαρμογές, ανάλυση συγκρούσεων, πρόβλεψη καιρού, κ.α.
- Αν και δε χρησιμοποιούνται πια, εντούτοις στοιχεία διανυσματικής επεξεργασίας (MMX, SSE, GPU, CELL) υπάρχουν σε σημερινούς επεξεργαστές.
- Είναι της μορφής SIMD.

Διαφορά Βαθμωτού (scalar)/ διανυσματικού (vector) CPU (1/2)

- Scalar processor

```
execute this loop 10 times
  read the next instruction and decode it
  fetch this number
  fetch that number
  add them
  put the result here
end loop
```

- Vector processor

```
read instruction and decode it
fetch these 10 numbers
fetch those 10 numbers
add them
put the results here
```

Διαφορά Βαθμωτού (scalar)/ διανυσματικού (vector) CPU (2/2)

- Οι διανυσματικοί επεξεργαστές μπορούν να εκτελούν και πιο σύνθετες πράξεις ταυτόχρονα (πρόσθεση και πολλαπλασιασμό):

```
read instruction and decode it
fetch these 10 numbers
fetch those 10 numbers
fetch another 10 numbers
add and multiply them
put the results here
```

Vector Processing @ IA32

- Η Intel το 1999 εισήγαγε τις εντολές SSE (Streaming SIMD Extensions) στο Pentium III.
- Χρησιμοποιούσαν 128bit καταχωρητές.
- Παράλληλη επεξεργασία αυτών.
- Η AMD εισήγαγε τις εντολές 3DNow! Που ήταν παρόμοιες στο Athlon (αλλά τις απέσυρε το 2010).

```
vec_res.x = v1.x + v2.x;
vec_res.y = v1.y + v2.y;
vec_res.z = v1.z + v2.z;
vec_res.w = v1.w + v2.w;
```

Χωρίς SIMD: 4 εντολές FPU για πρόσθεση

Με SIMD: 1 εντολή FPU για πρόσθεση

```
movaps xmm0,address-of-v1          ;xmm0=v1.w | v1.z | v1.y | v1.x
addps xmm0,address-of-v2           ;xmm0=v1.w+v2.w | v1.z+v2.z | v1.y+v2.y | v1.x+v2.x
movaps address-of-vec_res,xmm0
```

Vector Processing (SSE)

```
//SSE simd function for vectorised multiplication of 2 arrays with single-precision floatingpoint numbers
//1st param pointer on source/destination array, 2nd param 2. source array, 3th param number of floats per array
 void mul_asm(float* out, float* in, unsigned int leng)
 {    unsigned int count, rest;

      //compute if array is big enough for vector operation
      rest  = (leng*4)%16;
      count = (leng*4)-rest;

     // vectorised part; 4 floats per loop iteration
      if (count>0){
      __asm __volatile__  (".intel_syntax noprefix\n\t"
      "loop:                  \n\t"
      "movups xmm0,[ebx+ecx] ;loads 4 floats in first register (xmm0)\n\t"
      "movups xmm1,[eax+ecx] ;loads 4 floats in second register (xmm1)\n\t"
      "mulps xmm0,xmm1       ;multiplies both vector registers\n\t"
      "movups [eax+ecx],xmm0 ;write back the result to memory\n\t"
      "sub ecx,16            ;increase address pointer by 4 floats\n\t"
      "jns loop              \n\t"
      ".att_syntax prefix    \n\t"
        : : "a" (out), "b" (in), "c"(count), "d"(rest): "xmm0","xmm1");
      }

      // scalar part; 1 float per loop iteration
      if (rest!=0)
      {
       __asm __volatile__  (".intel_syntax noprefix\n\t"
      "add eax,ecx           \n\t"
      "add ebx,ecx           \n\t"

      "rest:                 \n\t"
      "movss xmm0,[ebx+edx]  ;load 1 float in first register (xmm0)\n\t"
      "movss xmm1,[eax+edx]  ;load 1 float in second register (xmm1)\n\t"
      "mulss xmm0,xmm1       ;multiplies both scalar parts of registers\n\t"
      "movss [eax+edx],xmm0  ;write back the result\n\t"
      "sub edx,4             \n\t"
      "jns rest              \n\t"
      ".att_syntax prefix    \n\t"
        : : "a" (out), "b" (in), "c"(count), "d"(rest): "xmm0","xmm1");
      }
      return;
 }
```

Ένα ακόμη παράδειγμα διανυσματοποίησης

Ordinary C

```
unsigned short a[N];
unsigned short b[N];
…
unsigned int val = 0;
for( i = 0; i < N; ++i ) {
      val += a[i] * b[i];
```

```
.L28:
    lhzu    r7, 2(r10)
    lhzu    r8, 2(r12)
    mullw    r0, r7, r8
    add    r11, r11, r0
    clrlwi    r11, r11, 16
    lhzu    r7, 2(r10)
    lhzu    r8, 2(r12)
    mullw    r0, r7, r8
    add    r11, r11, r0
    clrlwi    r11, r11, 16
    lhzu    r7, 2(r10)
    lhzu    r8, 2(r12)
    mullw    r0, r7, r8
    add    r11, r11, r0
    clrlwi    r11, r11, 16
    lhzu    r7, 2(r10)
    lhzu    r8, 2(r12)
    mullw    r0, r7, r8
    add    r11, r11, r0
    clrlwi    r11, r11, 16
    bdnz    .L28
```

= 21 cycles

Using Vectorization

```
#pragma alignvar(16)
unsigned short a[N];
#pragma alignvar(16)
unsigned short b[N];
…
unsigned int val = 0;
#pragma ghs vectorize
  for( i = 0; i < N; ++i ) {
  val += a[i] * b[i];
```

code added for vectorization

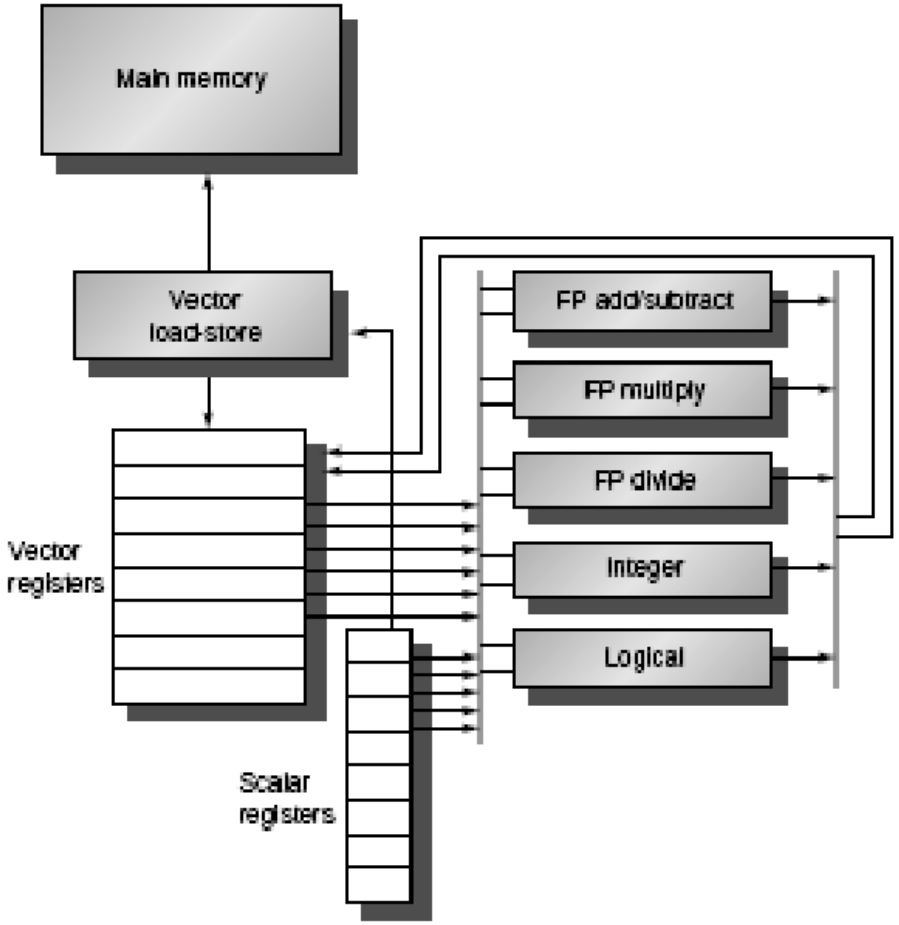
```
.L28:
    lvx    v17, r11, r12
    lvx    v18, r10, r12
    addi    r12, r12, 16
    vmladduhm    v19, v17, v18, v19
    bdnz    .L28
```

= 5 cycles

Κατηγορίες διανυσματικών επεξεργαστών

- Επεξεργαστές διανύσματος – καταχωρητή:
  - Όλες οι πράξεις γίνονται με καταχωρητές (load/store).
  - Κυριάρχησε από το 1980 και έπειτα.
- Επεξεργαστές διανύσματος μνήμης-μνήμης:
  - Όλες οι πράξεις γίνονται απευθείας στη μνήμη.
  - Χρησιμοποιήθηκε στους πρώτους διανυσματικούς επεξεργαστές.

Μια τυπική διανυσματική αρχιτεκτονική: VMIPS



- Vector registers multiple read/write (64bits)
- Multiple Vector pipelined functional units
- Control Unit (detect hazards)
- Vector Load Store
- Scalar Registers

Χαρακτηριστικά κάποιων διανυσματικών επεξεργαστών

Processor (year)	Clock rate (MHz)	Vector registers	Elements per register (64-bit elements)	Vector arithmetic units	Vector load-store units	Lanes
Cray-1 (1976)	80	8	64	6: FP add, FP multiply, FP reciprocal, integer add, logical, shift	1	1
Cray X-MP (1983) Cray Y-MP (1988)	118 166	8	64	8: FP add, FP multiply, FP reciprocal, integer add, 2 logical, shift, population count/parity	2 loads 1 store	1
Cray-2 (1985)	244	8	64	5: FP add, FP multiply, FP reciprocal/ sqrt, integer add/shift/population count, logical	1	1
Fujitsu VP100/ VP200 (1982)	133	8–256	32–1024	3: FP or integer add/logical, multiply, divide	2	1 (VP100) 2 (VP200)
Hitachi S810/ S820 (1983)	71	32	256	4: FP multiply-add, FP multiply/ divide-add unit, 2 integer add/logical	3 loads 1 store	1 (S810) 2 (S820)
Convex C-1 (1985)	10	8	128	2: FP or integer multiply/divide, add/ logical	1	1 (64 bit) 2 (32 bit)
NEC SX/2 (1985)	167	8 + 32	256	4: FP multiply/divide, FP add, integer add/logical, shift	1	4
Cray C90 (1991) Cray T90 (1995)	240 460	8	128	8: FP add, FP multiply, FP reciprocal, integer add, 2 logical, shift, population count/parity	2 loads 1 store	2

Ένα παράδειγμα διανυσματικής επεξεργασίας

- Y = a x X + Y
- Αυτή είναι μια πράξη SAXPY ή DAXPY (single/double-precision a times X plus Y).
- Ας υποθέσουμε ότι έχουμε 64 στοιχεία (όσα υποστηρίζονται από τον επεξεργαστή μας).

Υλοποίηση της DAXPY

```
        L.D       F0,a          ;load scalar a
        DADDIU    R4,Rx,#512    ;last address to load
Loop:   L.D       F2,0(Rx)      ;load X(i)
        MUL.D     F2,F2,F0      ;a × X(i)
        L.D       F4,0(Ry)      ;load Y(i)
        ADD.D     F4,F4,F2      ;a × X(i) + Y(i)
        S.D       0(Ry),F4      ;store into Y(i)
        DADDIU    Rx,Rx,#8      ;increment index to X
        DADDIU    Ry,Ry,#8      ;increment index to Y
        DSUBU     R20,R4,Rx     ;compute bound
        BNEZ      R20,Loop      ;check if done
```

Κώδικας MIPS
600 εντολές

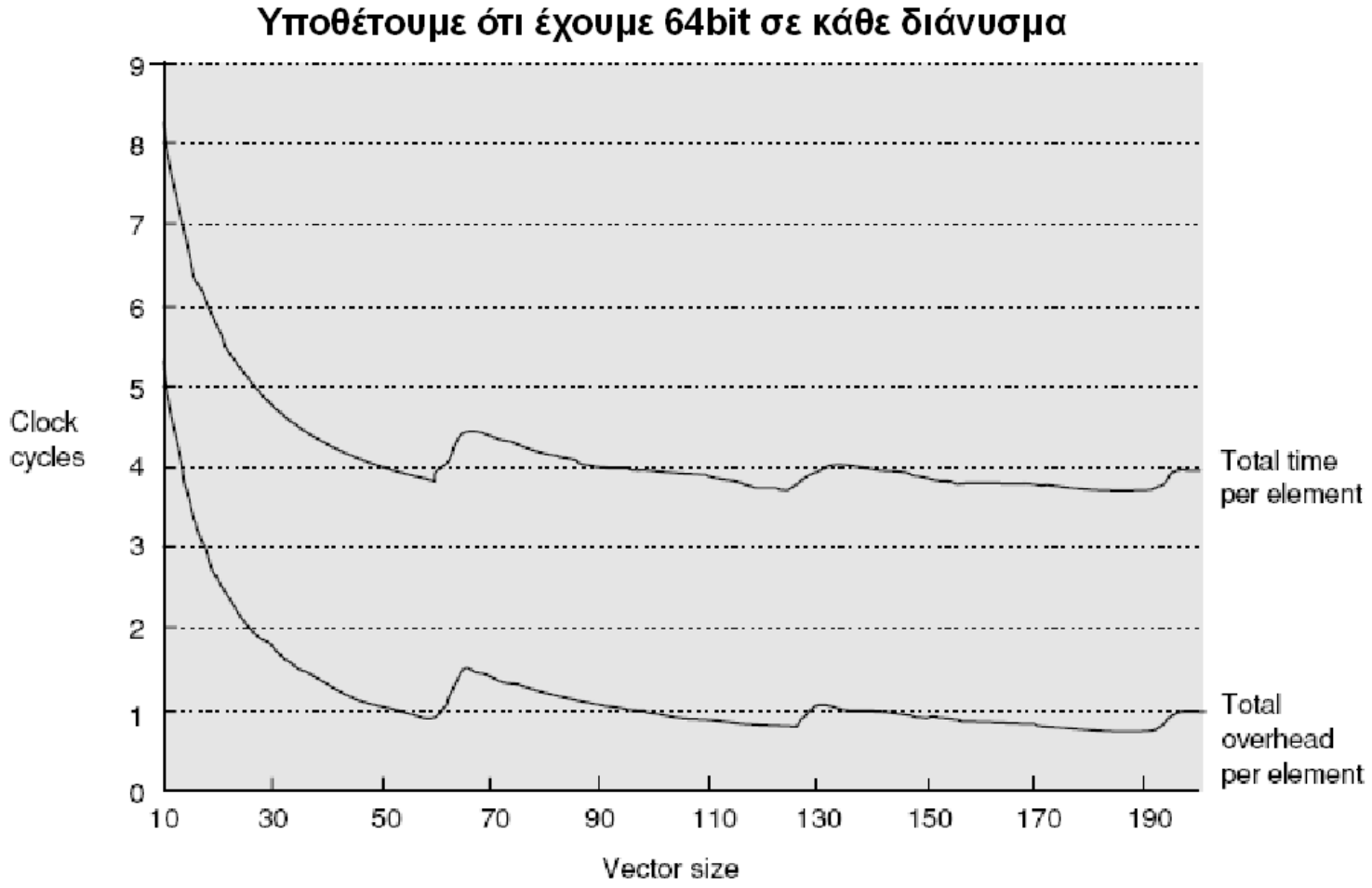
Κώδικας VMIPS
6 εντολές

```
L.D       F0,a          ;load scalar a
LV        V1,Rx         ;load vector X
MULVS.D   V2,V1,F0      ;vector-scalar multiply
LV        V3,Ry         ;load vector Y
ADDV.D    V4,V2,V3      ;add
SV        Ry,V4         ;store the result
```

Σύγκριση κωδικών MIPS/VMIPS

- MIPS.
  - 600 εντολές.
  - Προβλήματα στο pipeline:
    - Κάθε ADD.D πρέπει να περιμένει το MUL.D.
    - Κάθε S.D πρέπει να περιμένει το ADD.D.
    - 64 φορές περισσότερα stall από ότι στο VMIPS.
    - Μπορεί να τροποποιηθεί ο κώδικας για να απομακρυνθούν τα περισσότερα stalls, αλλά το instruction bandwidth δε θα μειωθεί.
- VMIPS
  - 6 εντολές.
  - Στο pipeline θα υπάρχει μόνο μια φορά καθυστέρηση στο να έρθει το πρώτο στοιχείο του διανύσματος.

Οι διανυσματικοί επεξεργαστές εξαρτώνται από το μέγεθος διανύσματος



Τεχνικές βελτίωσης διανυσματικών επεξεργαστών: chaining (1/2)

- Chaining (αλυσιδωτή σύνδεση).
- Έστω έχουμε τις πράξεις:

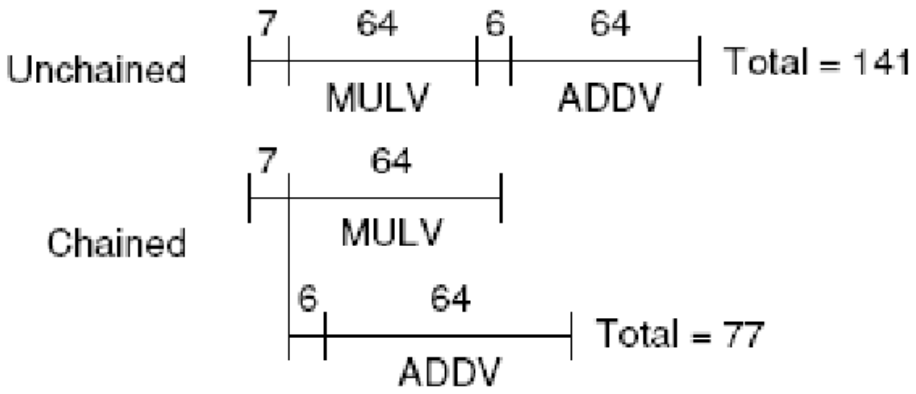
  MULV.D V1,V2,V3

  ADDV.D V4,V1,V5
- Η 2η εντολή εξαρτάται από την 1η.
- Αν όμως το V1 δεν το δούμε ως μια ολόκληρη οντότητα αλλά ως συλλογή από στοιχεία, τότε μόλις υπολογιστούν τα πρώτα στοιχεία του V1, μπορούν να τροφοδοτηθούν στην επόμενη εντολή. Έτσι θα αρχίσει να εκτελείται μαζί με τη MULV.D και η ADDV.D μετά από λίγους κύκλους.
- Θα πρέπει ο καταχωρητής να υποστηρίζει εγγραφή και ανάγνωση ταυτόχρονα (σε διαφορετικές θέσεις).
- Υπάρχουν ειδικές εντολές για chaining.

Τεχνικές βελτίωσης διανυσματικών επεξεργαστών: chaining (2/2)

- Απόδοση του chaining.
- Συνολική απόδοση:

$$\text{Vector length} + \text{Start-up time}_{\text{ADDV}} + \text{Start-up time}_{\text{MULV}}$$



- Υπάρχει μια σημαντική αύξηση των επιδόσεων.

Τεχνικές βελτίωσης διανυσματικών επεξεργαστών: μάσκα (1/2)

- Υπάρχουν βρόχοι με διακλαδώσεις που δε μπορούν να εκτελεστούν αποδοτικά.
- Ένα παράδειγμα είναι:

```
      do 100 i = 1, 64
            if (A(i).ne. 0) then
                  A(i) = A(i) – B(i)
            endif
100   continue
```

- Αν μπορούσαμε να εκτελέσουμε την πράξη μόνο για τις τιμές του διανύσματος A(i) που δεν είναι ίσες με μηδέν τότε θα μπορούσε να εκτελεστεί αποδοτικά.
- Γίνεται με τη χρήση των ειδικών εντολών μάσκας.

Τεχνικές βελτίωσης διανυσματικών επεξεργαστών: μάσκα (2/2)

- Έτσι ορίζουμε μια μάσκα και εκτελούμε μια πράξη για τις εντολές που υπακούν σε αυτή την μάσκα:

```
LV          V1,Ra        ;load vector A into V1
LV          V2,Rb        ;load vector B
L.D         F0,#0        ;load FP zero into F0
SNEVS.D     V1,F0        ;sets VM(i) to 1 if V1(i)!=F0
SUBV.D      V1,V1,V2     ;subtract under vector mask
CVM                      ;set the vector mask to all 1s
SV          Ra,V1        ;store the result in A
```

- Ασφαλώς η επεξεργασία της μάσκας απαιτεί κάποιους κύκλους. Εντούτοις η απαλοιφή της διακλάδωσης μας αυξάνει τις επιδόσεις.
- Συνήθως η μάσκα απενεργοποιεί την αποθήκευση. Η πράξη γίνεται κανονικά.

Τεχνικές βελτίωσης διανυσματικών επεξεργαστών: scatter/gather (1/3)

- Υπάρχουν περιπτώσεις που γίνεται πρόσβαση σε μη συνεχόμενα στοιχεία (sparse arrays), όπως όταν έχουμε έμμεση πρόσβαση με δείκτες:

```
      do      100 i = 1,n
100           A(K(i)) = A(K(i)) + C(M(i))
```

- Χρησιμοποιείται η τεχνική scatter / gather.
  - Με τη gather μαζεύουμε όλα τα στοιχεία σε ένα vector.
  - Επεξεργαζόμαστε το vector.
  - Με τη scatter αποθηκεύουμε τα στοιχεία στις θέσεις τους.

Τεχνικές βελτίωσης διανυσματικών επεξεργαστών: scatter/gather (2/3)

- Χρησιμοποιούνται οι εντολές LVI (gather) και SVI (store).
- Ο compiler δε μπορεί να χρησιμοποιήσει αυτές τις εντολές, γιατί δε γνωρίζει αν υπάρχουν εξαρτήσεις.

```
LV        Vk,Rk          ;load K
LVI       Va,(Ra+Vk)     ;load A(K(I))
LV        Vm,Rm          ;load M
LVI       Vc,(Rc+Vm)     ;load C(M(I))
ADDV.D    Va,Va,Vc       ;add them
SVI       (Ra+Vk),Va     ;store A(K(I))
```

- Ο προγραμματιστής πρέπει να τοποθετήσει αυτές τις εντολές.
- Υπάρχει επιβάρυνση ασφαλώς για τις εντολές, αλλά εντούτοις η απόδοση είναι καλύτερη από την απλή βαθμωτή εκτέλεση.

Τεχνικές βελτίωσης διανυσματικών επεξεργαστών: scatter/gather (3/3)

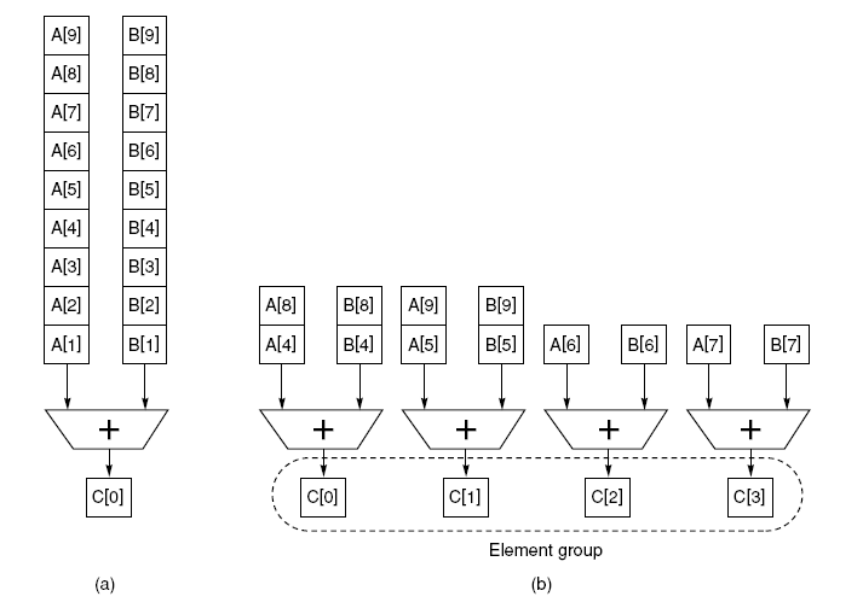
- Μπορούμε να χρησιμοποιήσουμε και τις λειτουργίες της μάσκας, αν υποστηρίζεται όπως:

```
LV          V1,Ra         ;load vector A into V1
L.D         F0,#0         ;load FP zero into F0
SNEVS.D     V1,F0         ;sets the VM to 1 if V1(i)!=F0
CVI         V2,#8         ;generates indices in V2
POP         R1,VM         ;find the number of 1's in VM
MTC1        VLR,R1        ;load vector-length register
CVM                       ;clears the mask
LVI         V3,(Ra+V2)    ;load the nonzero A elements
LVI         V4,(Rb+V2)    ;load corresponding B elements
SUBV.D      V3,V3,V4      ;do the subtract
SVI         (Ra+V2),V3    ;store A back
```

- Ποια τεχνική είναι καλύτερη; Εξαρτάται από τη συχνότητα με την οποία ισχύει η συνθήκη και το κόστος των πράξεων.
- Συνήθως είναι αυτή καλύτερη, εκτός αν επαναχρησιμοποιούνται τα scatter/gather.

Τεχνικές βελτίωσης διανυσματικών επεξεργαστών: πολλαπλές ροές

- Οι επιδόσεις μπορούν να αυξηθούν με τη χρήση πολλαπλών ροών και πολλαπλών λειτουργικών μονάδων:



- Είναι εύκολο να προστεθούν νέες ροές.
- Βελτιώνεται το peak performance αλλά δεν αλλάζει το startup-latency.

Τι καθορίζει το χρόνο εκτέλεσης μιας διανυσματικής πράξης;

- Ο χρόνος εκτέλεσης μιας διανυσματικής πράξης καθορίζεται από:
  - Την επιβάρυνση για εκκίνηση της σωλήνωσης και συμπλήρωση των καταχωρητών.
  - Ένα κύκλο ανά στοιχείο διανύσματος.

Τι επηρεάζει την επιτυχία εκτέλεσης σε διανυσματικό επεξεργαστή;

- Δύο στοιχεία επηρεάζουν την επιτυχία (δηλαδή την αύξηση της απόδοσης) ενός προγράμματος όταν εκτελείται σε διανυσματικό επεξεργαστή.
  - Τη δομή του αλγόριθμου (εξαρτήσεις):
    - εξαρτάται από τον αλγόριθμο και,
    - από τον τρόπο που έχει γραφεί το πρόγραμμα.
  - Ικανότητα του συμβολομεταφραστή (compiler):
    - Υπάρχουν ποικίλες διαφοροποιήσεις στην ικανότητα των compiler να βρίσκουν πότε ένας βρόχος μπορεί να διανυσματοποιηθεί.

Έμπειρος προγραμματιστής VS compiler

Benchmark name	Operations executed in vector mode, compiler-optimized	Operations executed in vector mode, hand-optimized	Speedup from hand optimization
BDNA	96.1%	97.2%	1.52
MG3D	95.1%	94.5%	1.00
FLO52	91.5%	88.7%	N/A
ARC3D	91.1%	92.0%	1.01
SPEC77	90.3%	90.4%	1.07
MDG	87.7%	94.2%	1.49
TRFD	69.8%	73.7%	1.67
DYFESM	68.8%	65.6%	N/A
ADM	42.9%	59.6%	3.60
OCEAN	42.8%	91.2%	3.92
TRACK	14.4%	54.6%	2.52
SPICE	11.5%	79.9%	4.06
QCD	4.2%	75.1%	2.15

Υπάρχουν καλοί και όχι καλοί compiler

Processor	Compiler	Completely vectorized	Partially vectorized	Not vectorized
CDC CYBER 205	VAST-2 V2.21	62	5	33
Convex C-series	FC5.0	69	5	26
Cray X-MP	CFT77 V3.0	69	3	28
Cray X-MP	CFT V1.15	50	1	49
Cray-2	CFT2 V3.1a	27	1	72
ETA-10	FTN 77 V1.0	62	7	31
Hitachi S810/820	FORT77/HAP V20-2B	67	4	29
IBM 3090/VF	VS FORTRAN V2.4	52	4	44
NEC SX/2	FORTRAN77 / SX V.040	66	5	29

Τι ισχύει για τους διανυσματικούς επεξεργαστές; (1/2)

- Δεν είναι μόνο το peak performance σημαντικό. Είναι και το start-up overhead.
- Σε ένα διανυσματικό επεξεργαστή πρέπει να έχουμε και γρήγορη επεξεργασία βαθμωτών (scalar) μεγεθών.
- Δε γίνεται να έχουμε μεγάλες αποδόσεις στους διανυσματικούς επεξεργαστές, αν δεν παρέχουμε μεγάλο εύρος ζώνης μνήμης.
- Ύστερα από 30 χρόνια διανυσματικής επεξεργασίας, παραμένει η φιλοσοφία στους σύγχρονους επεξεργαστές.
- Δεν υπάρχουν πια διανυσματικοί επεξεργαστές.

Τι ισχύει για τους διανυσματικούς επεξεργαστές; (2/2)

- Οι διανυσματικοί επεξεργαστές δε μπόρεσαν να εκμεταλλευτούν:
  - μεγαλύτερες διοχετεύσεις,
  - τεχνικές πολλαπλής έκδοσης εντολών, εκτέλεσης εκτός σειράς, πρόγνωσης βρόχων,
  - τεχνολογίες κατασκευής (cmos).

Διανυσματοποίηση βρόχων (1/2)

- Είναι η διαδικασία κατά την οποία βρόχοι που εκτελούν μια λειτουργία σε βαθμωτά στοιχεία μετατρέπονται από το συμβολομεταφραστή σε βρόχους που εκτελούν σε διανυσματικά στοιχεία, με πολύ μεγάλα οφέλη.
- Η συμπεριφορά του προγράμματος δεν αλλάζει, μόνο η ταχύτητα.
- Δεν υπάρχει παραβίαση των εξαρτήσεων.
- Δεν υπάρχει παραβίαση ακρίβειας (δηλαδή, διατηρούνται τα μεγέθη σε αρ. Bit των βαθμωτών στοιχείων).

Διανυσματοποίηση βρόχων (2/2)

- Στην ανάλυση του βρόχου για διανυσματοποίηση, κοιτάμε ένα παράθυρο εξαρτήσεων δεδομένων όσο είναι το μέγεθος του διανύσματος που θα χρησιμοποιήσουμε.
- Αν χρησιμοποιούμε διάνυσμα 128bit και ακέραιους αριθμούς 32bit, τότε πρέπει το παράθυρο διανυσματοποίησης θα έχει 4 ακεραίους.
- Έτσι:

  a[i] = a[i+16]; // 16 > 4, δε μας ενδιαφέρει η εξάρτηση

  a[i] = a[i+1]; // 1 < 4, μας ενδιαφέρει η εξάρτηση

Διανυσματοποίηση βρόχων - Ομαδοποίηση

- Χρησιμοποιώντας το γράφο εξάρτησης ο compiler μπορεί να ομαδοποιήσει συγκεκριμένες εντολές.
- Για παράδειγμα αν υπάρχουν τα μπλοκ εντολών (S1+S2), S3, S4 και μόνο το S3 μπορεί να διανυσματοποιηθεί, τότε θα δημιουργήσει 3 βρόχους, από τους οποίους μόνο ο S3 θα έχει διανυσματικές εντολές.

Διανυσματοποίηση βρόχων - Στάδια

- Υπάρχουν 4 στάδια στη διανυσματοποίηση:
  - Αρχή (prelude): Μεταφορά των βαθμωτών τιμών σε διανύσματα.
  - Βρόχος: Εκτέλεση του βρόχου.
  - Κλείσιμο (postlude): Αν απαιτείται κάποια βαθμωτή ενέργεια μετά τη διανυσματική πράξη.
  - Τακτοποίηση εκτός βρόχου (clean-up): Αν οι επαναλήψεις δεν ήταν πολλαπλάσιο του μεγέθους διανύσματος, τότε τα στοιχεία που δεν τοποθετήθηκαν σε διάνυσμα θα ακολουθήσουν μια βαθμωτή επεξεργασία.

Δυσκολία της διανυσματοποίησης με δείκτες

- Το παρακάτω κομμάτι κώδικα μπορεί να διανυσματοποιηθεί αυτόματα, γιατί είναι ξεκάθαρες οι εξαρτήσεις:

- Αυτό το κομμάτι κώδικα δε μπορεί να διανυσματοποιηθεί, γιατί χρησιμοποιεί δείκτες.

```
int a[128];
int b[128];
// initialize b

for (i = 0; i<128; i++)
  a[i] = b[i] + 5;
```

```
int *a = malloc(128*sizeof(int));
int *b = malloc(128*sizeof(int));
// initialize b

for (i = 0; i<128; i++, a++, b++)
  *a = *b + 5;
// ...
// ...
// ...
free(b);
free(a);
```

Παράδειγμα διανυσματοποίησης (1/4)

- Πολλαπλασιασμός δυο βαθμωτών στοιχείων σε βρόχο:

```
for (i = 0; i < 1024; i++)
   C[i] = A[i]*B[i];
```

- Μετατρέπεται

σε διανυσματικό βρόχο:

```
for (i = 0; i < 1024; i+=4)
   C[i:i+3] = A[i:i+3]*B[i:i+3];
```

Δεδομένου ότι μια διανυσματική λειτουργία απαιτεί σχεδόν ίδιο χρόνο με μια βαθμωτή λειτουργία, πόσο είναι το κέρδος μας;

Παράδειγμα διανυσματοποίησης (2/4)

- Ο προηγούμενος βρόχος διανυσματοποιείται ως εξής σε μια σειρά από στάδια:
  1. Εκτέλεση μετατροπής stripmining (δημιουργία υπο-βρόχων μεγέθους όσο το μέγεθος του διανύσματος).

```
for (i = 0; i < 1024; i+=4)
    for (ii = 0; ii < 4; ii++)
        C[i+ii] = A[i+ii]*B[i+ii];
```

Παράδειγμα διανυσματοποίησης (3/4)

2. Εκτέλεση μετατροπής loop-distribution (ή loop-fussion) για τη διαίρεση του βρόχου σε υπο-βρόχους χρησιμοποιώντας προσωρινούς πίνακες:

```
for (i = 0; i < 1024; i+=4)
{
  for (ii = 0; ii < 4; ii++) tA[ii] = A[i+ii];
  for (ii = 0; ii < 4; ii++) tB[ii] = B[i+ii];
  for (ii = 0; ii < 4; ii++) tC[ii] = tA[ii]*tB[ii];
  for (ii = 0; ii < 4; ii++) C[i+ii] = tC[ii];
}
```

Παράδειγμα διανυσματοποίησης (4/4)

3. Μετατροπή των βαθμωτών εντολών σε διανυσματικές εντολές:

```
for (i = 0; i < 1024; i+=4)
  {
    vA = vec_ld( &A[i] );
    vB = vec_ld( &B[i] );
    vC = vec_mul( vA, vB );
    vec_st( vC, &C[i] );
  }
```

Διανυσματοποίηση στους σύγχρονους επεξεργαστές (1/3)

- Όλοι οι σύγχρονοι επεξεργαστές υποστηρίζουν SSE (διανυσματικές) εντολές.
- Υπάρχουν ειδικοί διανυσματικοί καταχωρητές 128bit με ονόματα xmm0-xmm7.
- Μπορεί να χρησιμοποιηθούν για πράξεις:
  - 2 των 64bit διπλής ακρίβειας πραγματικό αριθμό.
  - 2 των 64bit ακέραιων αριθμών.
  - 4 των 32bit ακέραιων αριθμών.
  - 8 των 16bit σύντομων ακεραίων.
  - 16 των 8bit χαρακτήρων ή 1Byte αριθμών.
- Προσεχώς οι καταχωρητές διανυσμάτων θα είναι 256bit.

Διανυσματοποίηση στους σύγχρονους επεξεργαστές (2/3)

- Έστω πρόσθεση 4 ακεραίων αριθμών των 32bit.

- Χωρίς διανυσματοποίηση:

```
vec_res.x = v1.x + v2.x;
vec_res.y = v1.y + v2.y;
vec_res.z = v1.z + v2.z;
vec_res.w = v1.w + v2.w;
```

- Με διανυσματοποίηση:

```
movaps xmm0, [v1]          ;xmm0 = v1.w | v1.z | v1.y | v1.x
addps xmm0, [v2]           ;xmm0 = v1.w+v2.w | v1.z+v2.z | v1.y+v2.y | v1.x+v2.x
movaps [vec_res], xmm0
```

Διανυσματοποίηση στους σύγχρονους επεξεργαστές (3/3)

- Οι συμβολομεταφραστές υποστηρίζουν αυτόματη διανυσματοποίηση με τη χρήση των κατάλληλων παραμέτρων.
- π.χ. για το gcc είναι η παράμετρος:
  - -**ftree-vectorize**

    (Επίσης, ενεργοποιείται με την παράμετρο **-O3**)

Προγραμματισμός φιλικός για τη διανυσματοποίηση (1/2)

- Ένα τμήμα κώδικα με αριθμούς διπλής ακρίβειας:

```
crd[mvatm][X] = (double)crdtemp[X] + d[X] * k[X][X] + d[Y] * k[X][Y] + d[Z] * k[X][Z];
crd[mvatm][Y] = (double)crdtemp[Y] + d[X] * k[Y][X] + d[Y] * k[Y][Y] + d[Z] * k[Y][Z];
crd[mvatm][Z] = (double)crdtemp[Z] + d[X] * k[Z][X] + d[Y] * k[Z][Y] + d[Z] * k[Z][Z];
```

- Δε μπορεί να διανυσματοποιηθεί αυτόματα όπως φαίνεται ως εξής:
  - Αν εκτελέσουμε το compiler, με τη σημαία -O3 και την εκτύπωση πληροφοριών διανυσματοποίησης (δηλαδή, ποιο κομμάτι κώδικα μπόρεσε να διανυσματοποιηθεί).
  - Από τη assembly αν ζητήσουμε να μας την εμφανίσει σε ένα αρχείο.

Προγραμματισμός φιλικός για τη διανυσματοποίηση (2/2)

- Το πρόβλημα είναι ότι έχουμε διανύσματα 128bit, οπότε η διανυσματοποίηση γίνεται ανά 2.
- Εμείς έχουμε 3 αριθμούς διπλής ακρίβειας, οπότε δε μπορεί να γίνει διανυσματοποίηση.
- Η λύση είναι να προσθέσουμε ένα “ψεύτικο” πίνακα ώστε να καταλήξουμε σε ζυγό αριθμό πράξεων διπλής ακρίβειας, ως εξής:

```
crdmod[mvatm][X] = (double)crdtemp[X] + d[X] * k[X][X] + d[Y] * k[Y][X] + d[Z] * k[Z][X];
crdmod[mvatm][Y] = (double)crdtemp[Y] + d[X] * k[X][Y] + d[Y] * k[Y][Y] + d[Z] * k[Z][Y];
crdmod[mvatm][Z] = (double)crdtemp[Z] + d[X] * k[X][Z] + d[Y] * k[Y][Z] + d[Z] * k[Z][Z];
crdmod[mvatm][S] = (double)crdtemp[S] + d[X] * k[X][S] + d[Y] * k[Y][S] + d[Z] * k[Z][S];
```

- Μπορούμε να επιβεβαιώσουμε με το compiler ότι το παραπάνω κομμάτι διανυσματοποιείται.

Επεξεργαστές Μητρώου

Τι είναι και από τι αποτελείται ένας επεξεργαστής μητρώου;

- Οι επεξεργαστές μητρώου (array processor) ανήκουν στην κατηγορία SIMD.
- Αποτελούνται από:
  - ένα μεγάλο αριθμό μονάδων επεξεργασίας (processing elements PE):
    - Αριθμητική λογική μονάδα.
    - Καταχωρητές.
    - Μονάδα ελέγχου.
    - Αποκωδικοποιητή εντολών.
  - Ένα δίκτυο διασυνδέσεως.
  - Μια μονάδα ελέγχου.

Array Processors

- Οι PE εκτελούν συγχρόνως την ίδια εντολή αλλά υπάρχει η δυνατότητα απενεργοποιήσεως κάποιου υποσυνόλου τους.
- Το Δίκτυο Διασύνδεσης επιτρέπει την απευθείας μεταφορά δεδομένων μεταξύ των PE.
- Ένας επεξεργαστής χρησιμοποιείται για την επικοινωνία με τον έξω κόσμο (είσοδο / έξοδο).
- Εφαρμογή σε περιοχές που απαιτούν ομοειδή επεξεργασία πολλαπλών δεδομένων (π.χ. επεξεργασία εικόνας, γραφικά, AI).

Ιστορία των επεξεργαστών μητρώου

- Illiak IV (64 PE) @1960.
- DAP ICL (64 x 64 PE) @1983.
- MPP Goodyear Aerospace (128 x 128 PE) @1979.
- Blitzen @1989.

Χαρακτηριστικά Blitzen

- Physically small.
- Massively parallel.
- High performance machine.
- SIMD.
- 128 PE (processing elements).
- 128KB μνήμη κάθε PE.
- 20Mhz.
- 450MFlops (128 chips με 128PE).
- 1,25μm.

Μόνο ένα πρωτότυπο chip κατασκευάστηκε;.

Χαρακτηριστικά διασύνδεσης Blitzen

- Bit serial PE.
- RAM onchip for each PE.
- Bus orienteed I/O.
- Local modification of RAM addressing.
- Local conditional control of arithmetic/logic.
- X-grid interconnect (8 neighbors).
- 1bit ALU.
- 1bit registers A, B, (γενικής χρήσης) C P K G (ειδικής χρήσης).
- Καταχωρητής ολίσθησης (2 - 30bit).

Λειτουργικό Διάγραμμα PE

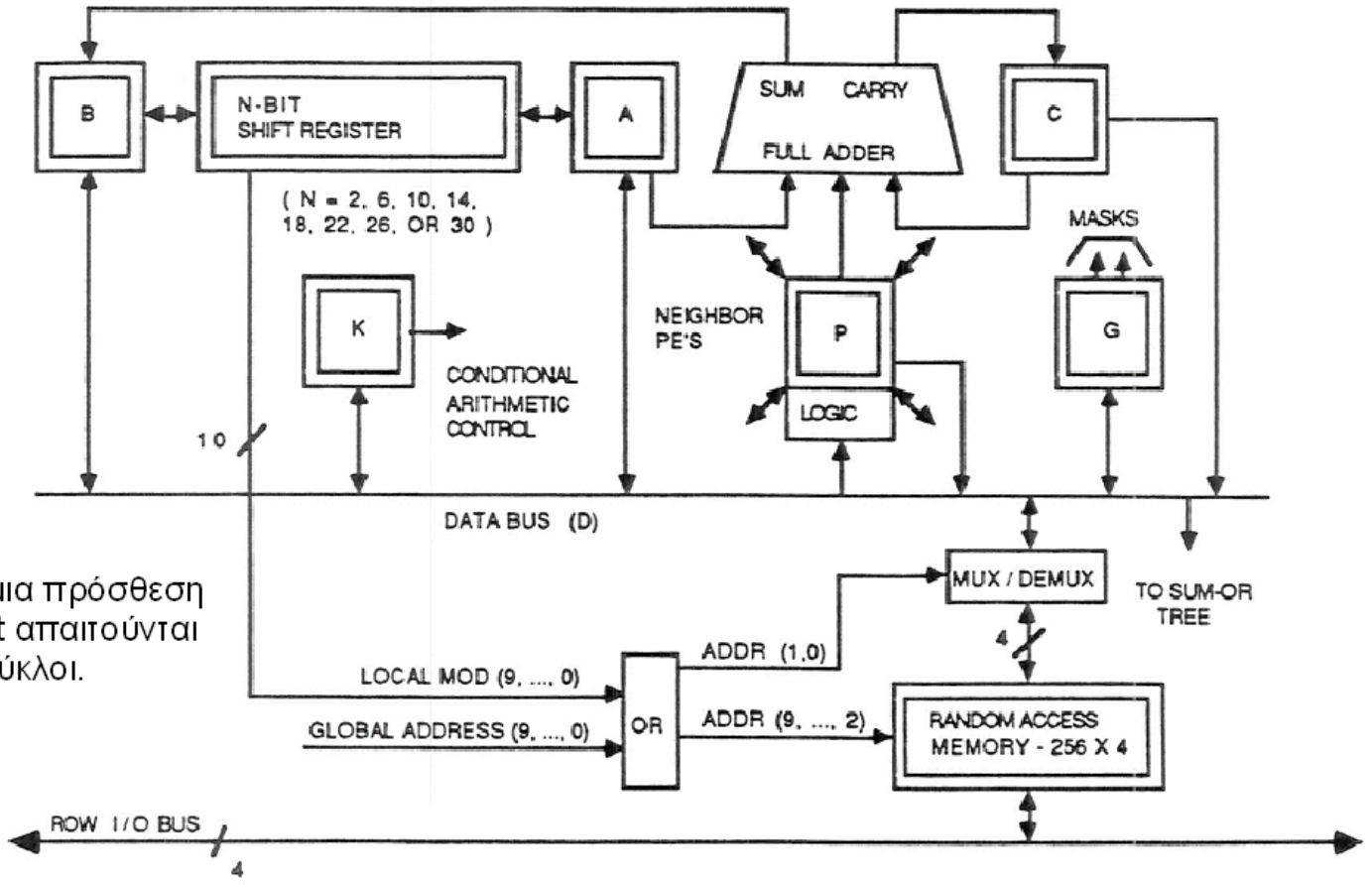


FIG. 2. Functional elements of one BLITZEN PE.

Για μια πρόσθεση 32bit απαιτούνται 96 κύκλοι.

Η διασύνδεση X-grid

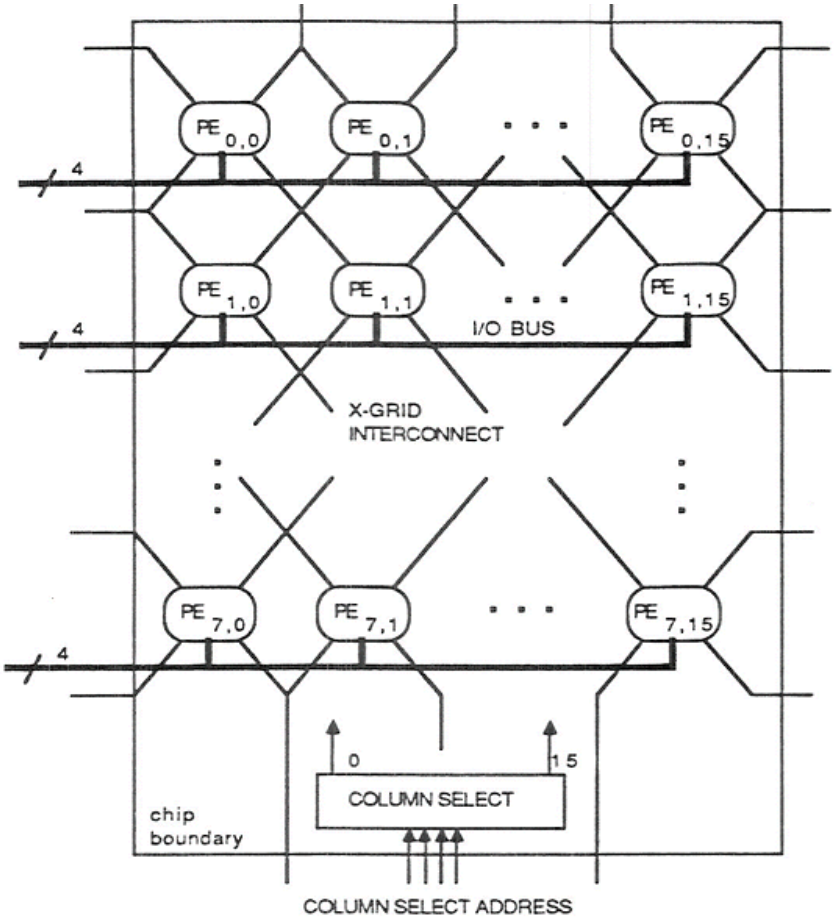
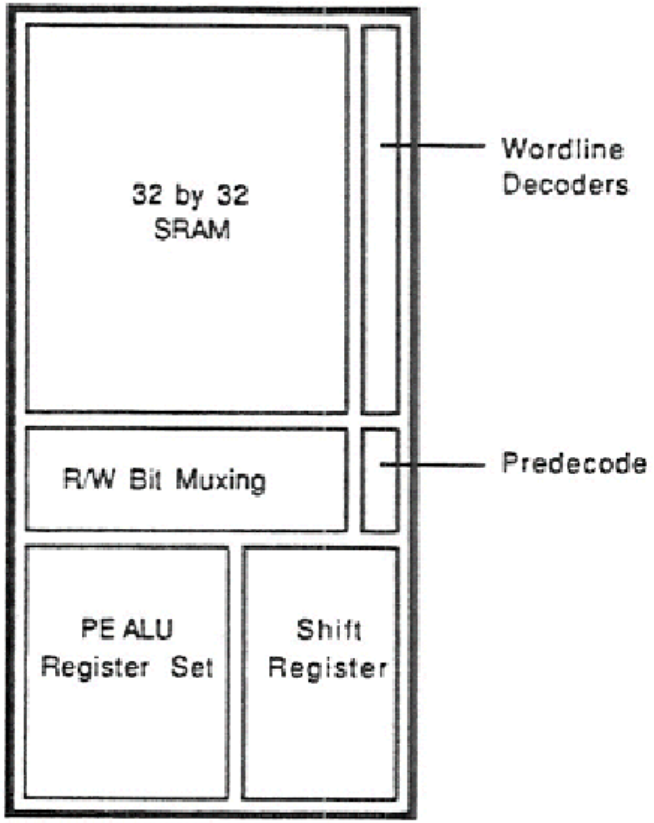


FIG. 6. BLITZEN chip architecture.

Το VLSI floorplan ενός στοιχείου PE



1130 transistor

FIG. 12. VLSI design floorplan for one PE.

Διάγραμμα συστήματος

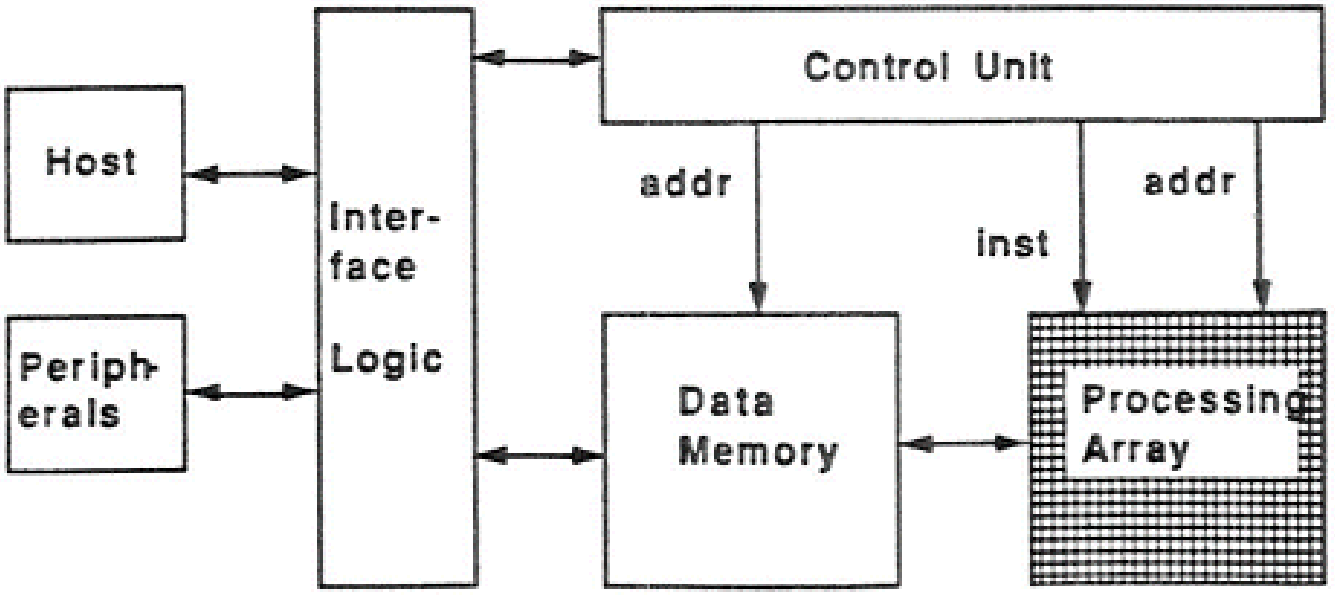


FIG. 1. System block diagram.

Μορφή εντολών blitzen (23bit)

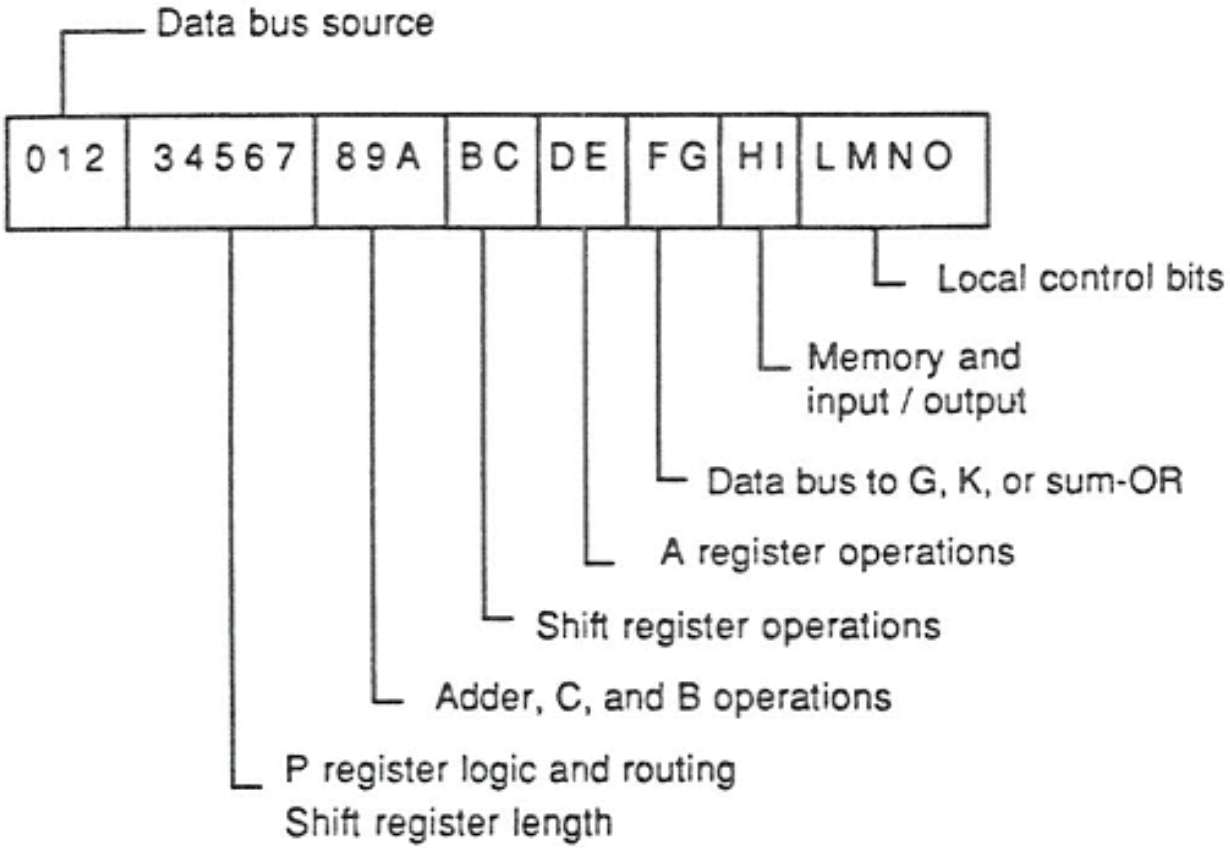


FIG. 9. Operation code format.

Προγραμματισμός με C (1/2)

- Για τον προγραμματισμό σε Blitzen χρησιμοποιείται η γλώσσα C με κάποια include.

  #include “blitzen.h”.

  #include “loadsave.h”.

- Χρησιμοποιούνται ειδικές συναρτήσεις:

  main() {set_route(GRID);

  load_file(“input.ism”,workspace,workspace,8)

  increment(workspace,8);

  save_file(“output.osm”,workspace,workspace,8)

  zyg_end()}

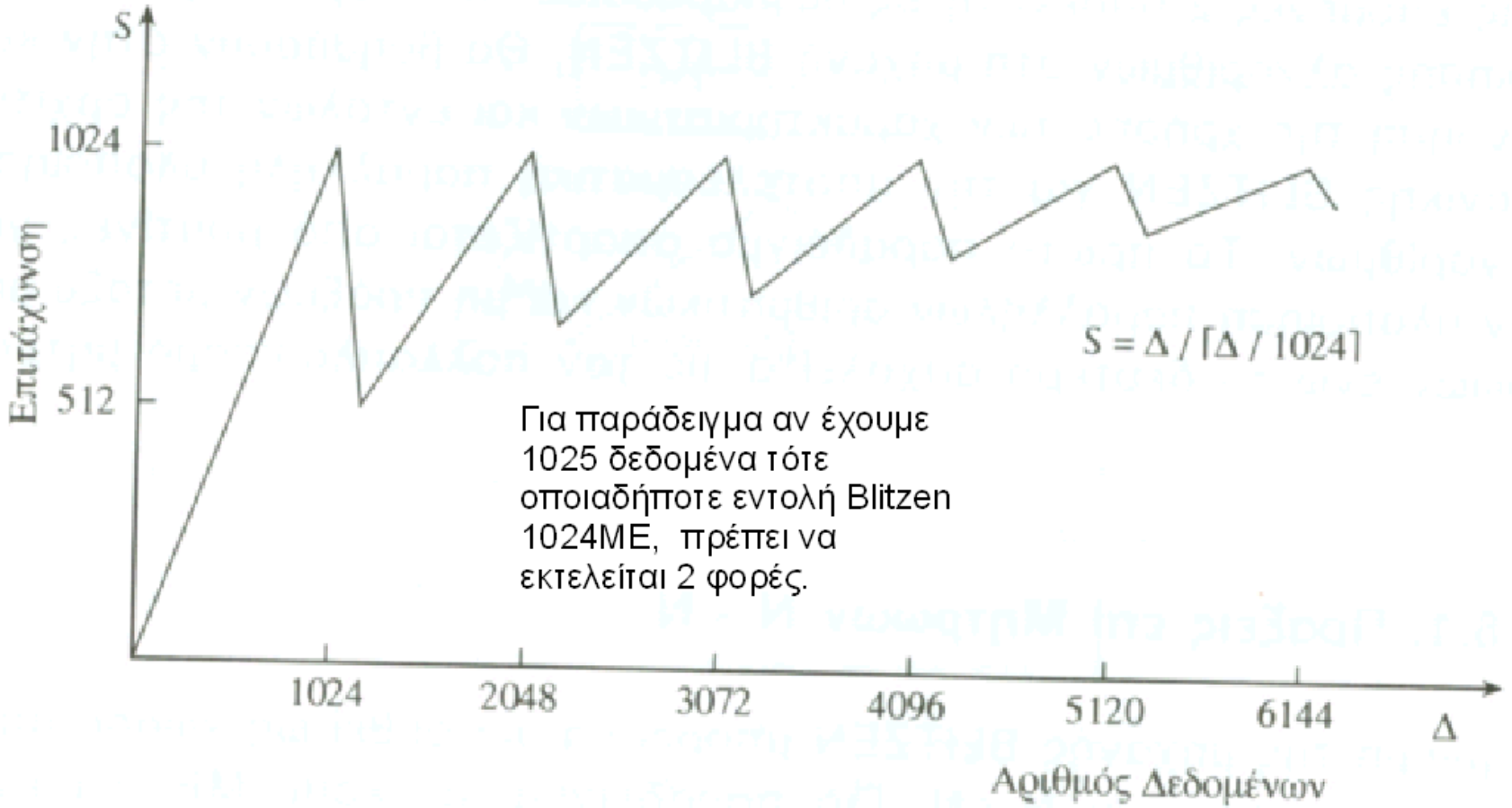
Προγραμματισμός με C (2/2)

```
Void increment(int addr,int numbit)
{
int i;
SET_C; /*1 στο C κάθε PE/
END;
for(i=0;i<numbits;i++){
MOV_MD(addr+i);
MOV_DA;/*τοποθέτηση στον A του αριθμού*/
END;
HADD;   /*hal add A+C αποτέλεσμα στο B /
END;
MOV_BD;/* τοποθέτηση αποτελέσματος στο B στη μνήμη
*/MOV_DB(addr+i);
END;
}
}
```

Τι ισχύει για την απόδοση της SIMD Blitzen;

- Εφαρμογές με ομοιόμορφη επεξεργασία διαφορετικών δεδομένων μπορούν να επιτύχουν υψηλή απόδοση.
- Όσο διαφοροποιείται η επεξεργασία των δεδομένων και όσο αυξάνεται η αλληλεξάρτηση τόσο μειώνεται η απόδοση.
- Η επιτάχυνση έχει τη μέγιστη απόδοση όταν ο αριθμός των δεδομένων είναι ακριβές πολλαπλάσιο του αριθμού των PE (π.χ. Αν έχουμε 130 στοιχεία για 128 PE τότε απαιτείται δύο φορές να εκτελείται η κάθε εντολή).

Η επιτάχυνση έχει τη μέγιστη τιμή της όταν ο αριθμός των δεδομένων είναι ακριβές πολλαπλάσιο του αριθμού ΜΕ



Ποια είναι τα χαρακτηριστικά του SMP;

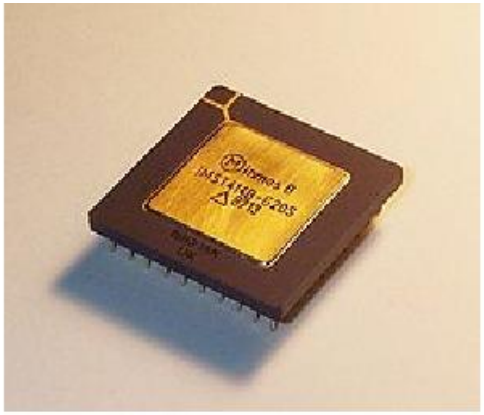
- Υπάρχουν δύο ή περισσότεροι όμοιοι επεξεργαστές με συγκρίσιμες ικανότητες.
- Οι επεξεργαστές διαμοιράζονται την ίδια κύρια μνήμη, τις διεπαφές I/O και διασυνδέονται με μια αρτηρία ή με κάποια άλλη μορφή εσωτερικής σύνδεσης.
- Ο χρόνος προσπέλασης στη μνήμη είναι ίδιος για όλους τους επεξεργαστές.
- Όλοι οι επεξεργαστές μπορούν να εκτελέσουν τις ίδιες λειτουργίες (*“συμμετρικοί”*).
- Το σύστημα ελέγχεται από ένα ολοκληρωμένο λειτουργικό σύστημα που προσφέρει αλληλεπίδραση μεταξύ επεξεργαστών.

Υπολογιστές transputers

Η αρχιτεκτονική transputer

- Πρωτοποριακή αρχιτεκτονική του 1980.
- Υπήρχαν σειριακοί σύνδεσμοι και ενσωματωμένη μνήμη.
- Προορίζονταν αποκλειστικά για παράλληλη επεξεργασία.
- Κατασκευάζονταν από την βρετανική εταιρία Inmos.
- Πίστευαν το 1980 ότι ήταν το μέλλον στην παράλληλη επεξεργασία.
- Αν και δεν το κατάφερε, εντούτοις χρησιμοποιήθηκαν κάποιες ιδέες και στους σύγχρονους παράλληλους υπολογιστές.

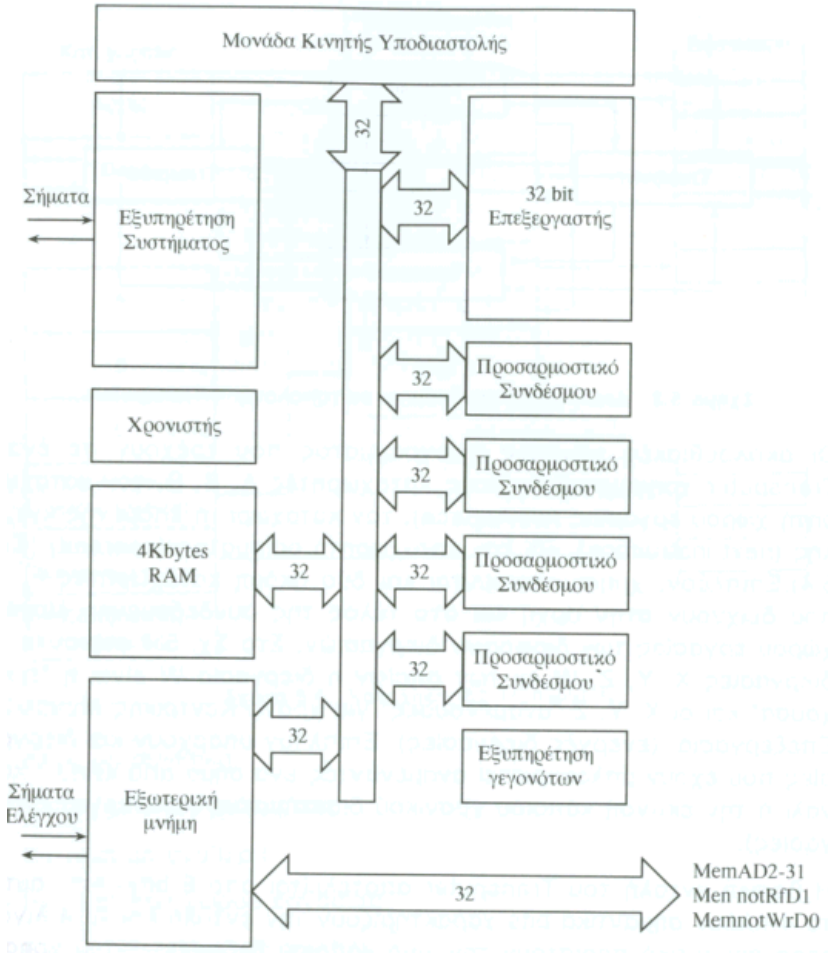
Ο επεξεργαστής transputer και η κάρτα τοποθέτησης



Στις αρχές του 1980 πολλοί πίστευαν ότι ο transputer ήταν το μέλλον

- Transputer (=transistor + computer).
- Το όνομα υποδήλωνε ότι μπορούσε να χρησιμοποιηθεί ως βασικό δομικό στοιχείο μεγάλων υπολογιστών.
- Ήθελαν να κρατήσουν το κόστος σε λίγα $$ για να μπορούν να χρησιμοποιηθούν παντού.
- Εύκολη διασύνδεση χωρίς πολύπλοκο δίαυλο.
- Απαιτούνταν μόνο τροφοδοσία και ρολόι.
- Χρησιμοποιήθηκε μικροκώδικας και μνήμη ROM.
- Πολύ γρήγορο ρολόι: 20Mhz.

Γενική Αρχιτεκτονική transputer (1/2)

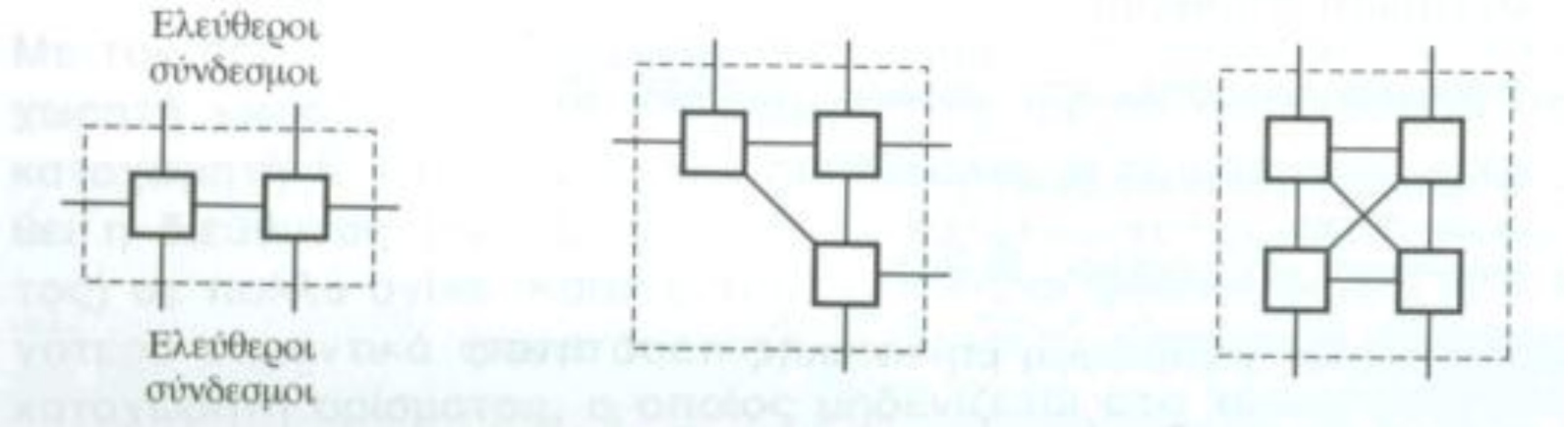


Γενική Αρχιτεκτονική transputer (2/2)

- Υπήρχαν σειριακοί σύνδεσμοι διπλής κατεύθυνσης.
- Μπορούσε εύκολα να συνδεθεί με 4 γειτονικούς transputers σε ταχύτητες 5,10,20Mbit/sec.
- Για τη διασύνδεση χρησιμοποιήθηκε διακόπτης (switch) μηδενικής καθυστέρησης για τη σύνδεση 32 transputers.
- Μπορούσε να συνδεθεί και αλυσιδωτά.
- Μπορούσε να κάνει bootup από το δίκτυο.
- Υπήρχε ένας scheduler οπότε δε χρειάζονταν το λειτουργικό σύστημα να κάνει τη χρονοδρομολόγηση.

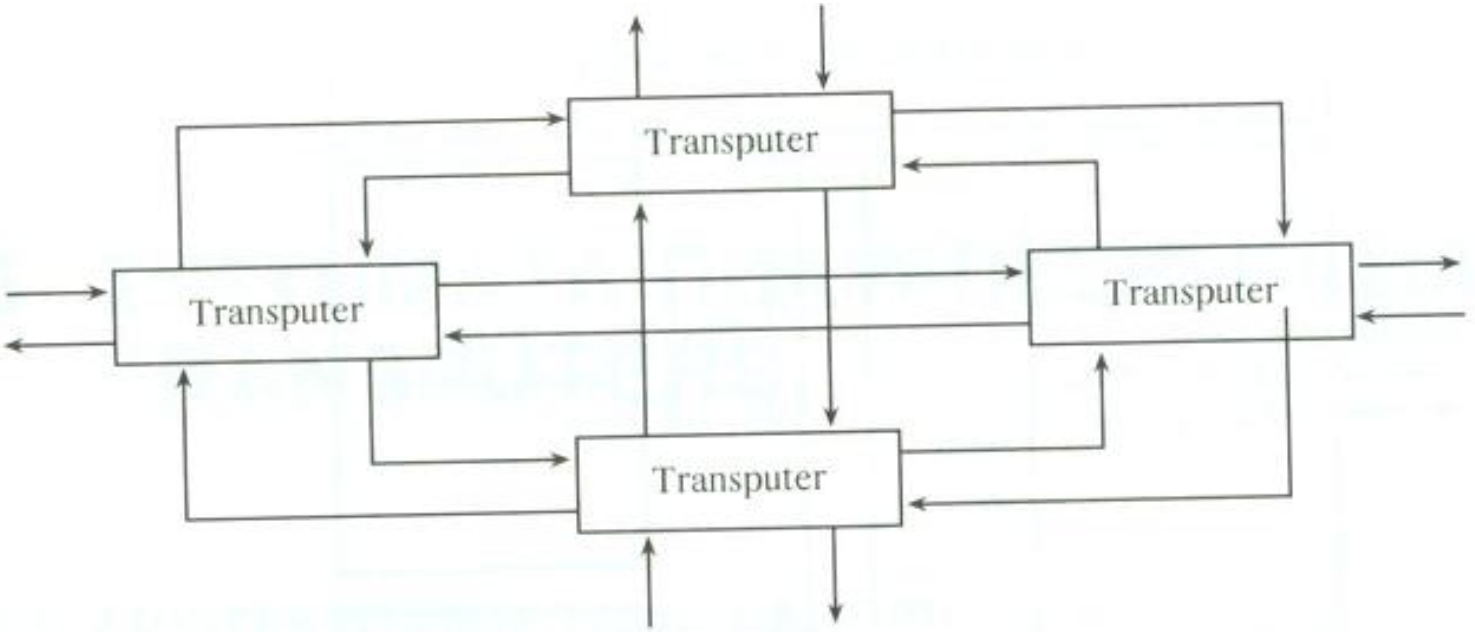
Διασύνδεση transputer (1/2)

- Μέγιστη διασύνδεση ενός transputer με 4 γείτονες.



Διασύνδεση transputer (2/2)

- Διασύνδεση συνδέσμων σε τοπολογία 4 transputer.



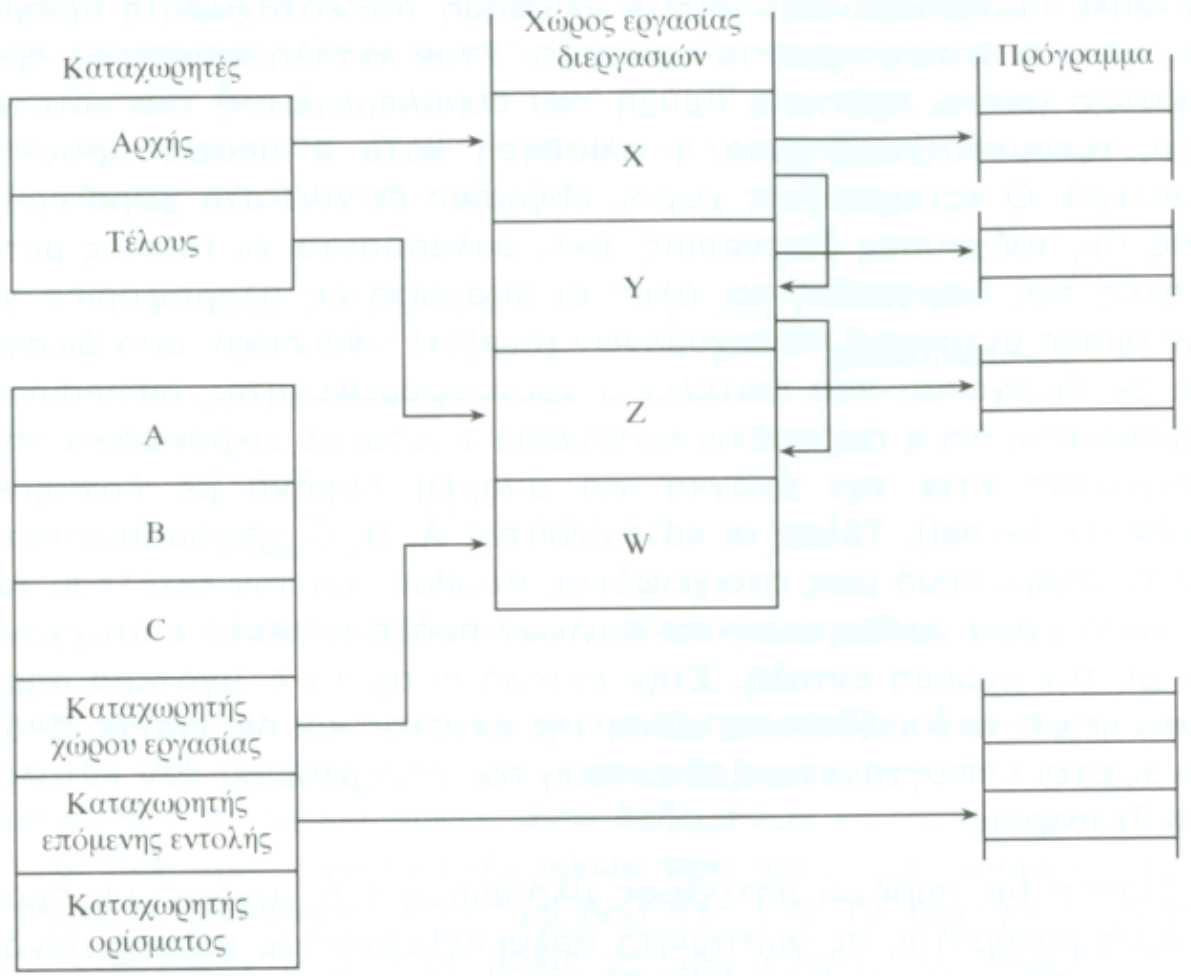
Η αρχιτεκτονική των transputter

- Ελάχιστοι καταχωρητές.
- Προγραμματισμός κυρίως με τη γλώσσα OCCAM.
- Δεν είχε υποστήριξη για virtual memory (και δε μπόρεσε να μεταφερθεί έτσι το UNIX).
- Αναπτύχθηκε το ΛΣ HELIOS (unix-like) ειδικά για τους transputers.
- 16bit, 32bit, 32bit(64bit FPU).
- Χρησιμοποιήθηκε η έννοια SoC (System on Chip).
- Αρχικά κόστιζαν πολύ (400$) για αυτό και δεν κυριάρχησαν στην αγορά.

Εσωτερική Περιγραφή του transputer

- 3 καταχωρητές (A,B,C).
- Καταχωρητής χώρου εργασίας.
- Καταχωρητής επόμενης εντολής.
- Καταχωρητής ορίσματος.
- Καταχωρητές συνδεδεμένης λίστας διεργασιών (F,B).
- Εντολές των 8bit.
- Υποστήριξη για μεταβίβαση μηνυμάτων (εσωτερικά –στον ίδιο transputter-- και εξωτερικά --σε άλλο transputer--).

Καταχωρητές transputer



Τέλος Ενότητας

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης