

Φαρδιές Ποσότητες, Φαρδιές Μνήμες:
Little-Endian versus Big-Endian,
Ευθυγράμμιση για Ταχύτητα Πρόσβασης

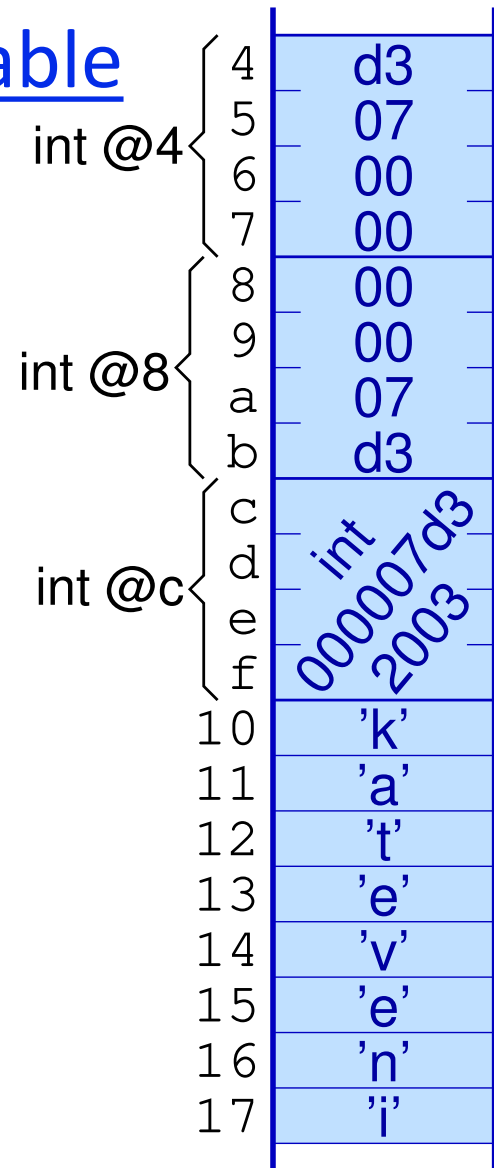
03b (§3.2-3.3) – 26/2-1/3/2021 – Μανόλης Κατεβαίνης

Ποσότη. >1 By σε Mn. Byte-Addressable

- Μνήμη Byte-Addressable
- Πίνακες κατά αύξουσες & συνεχόμενες διευθύνσεις, πάντα
 - π.χ. array of char @ base addr. 10

Ποσότητες μεγαλύτερες από 1 Byte:

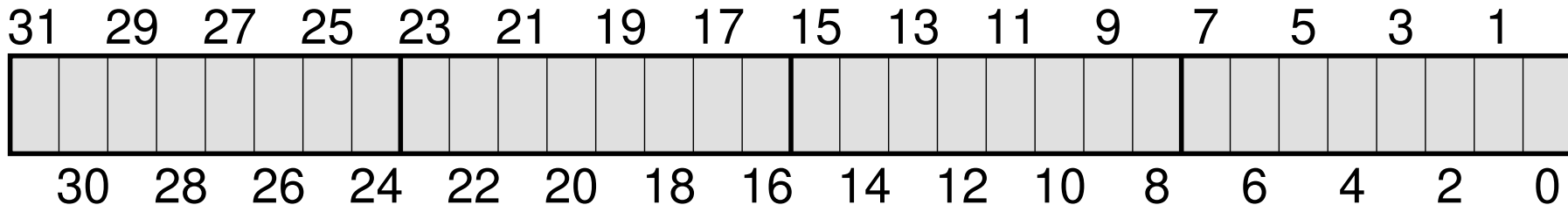
- Διεύθυνση = η διεύθυνση εκείνου από τα Bytes τους που έχει τη μικρότερη διεύθυνση, πάντα
- Τα Bytes μέσα στην ποσότητα: πού;;
 - π.χ. ακέραιος 2003 (δεκαεξαδικό 7d3)
 - πού πάνε τα Bytes 00, 00, 07, d3 ?



Αρίθμηση των bits μέσα σε μία Λέξη Ακεραίου

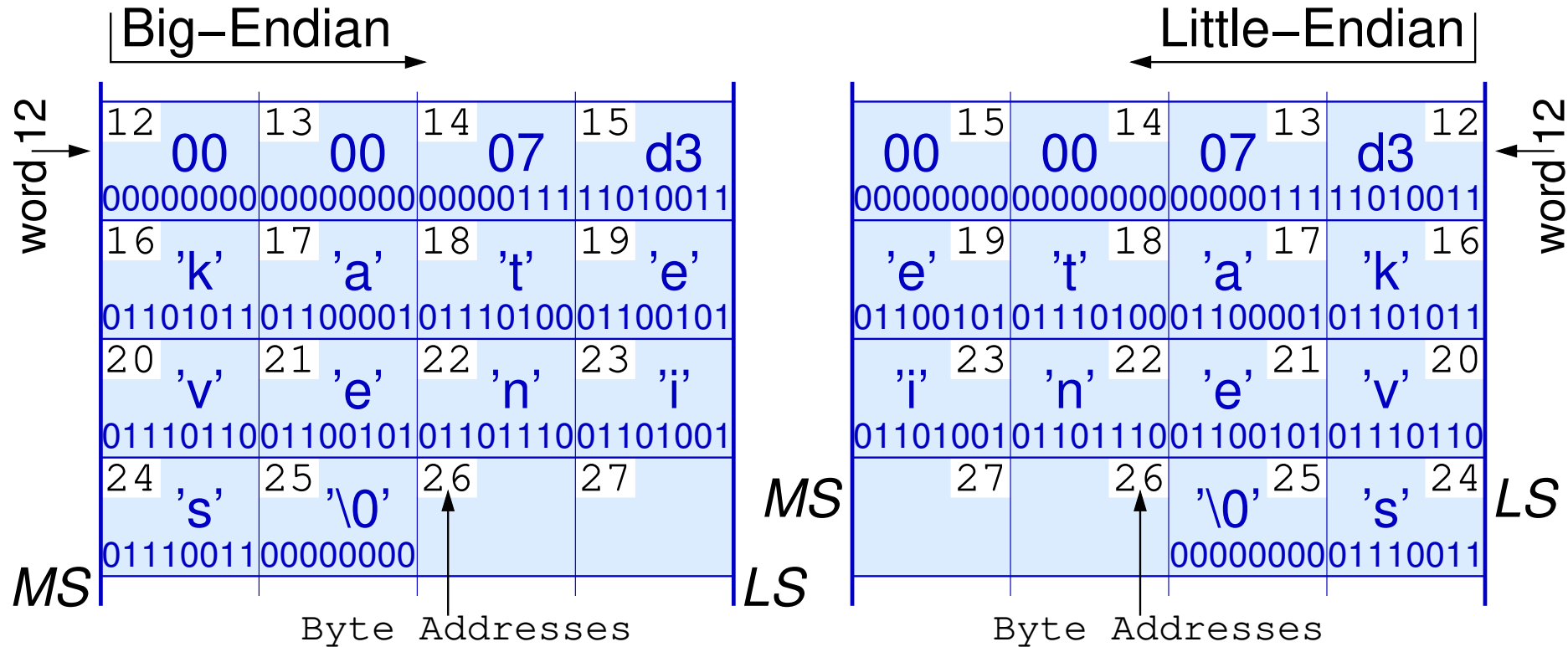
$$\text{Ακέραιος} = b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + \dots + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$$

Most Significant (MS)



Least Significant (LS)

Αρίθμηση των Bytes μέσα σε μία Λέξη Ακεραίου;



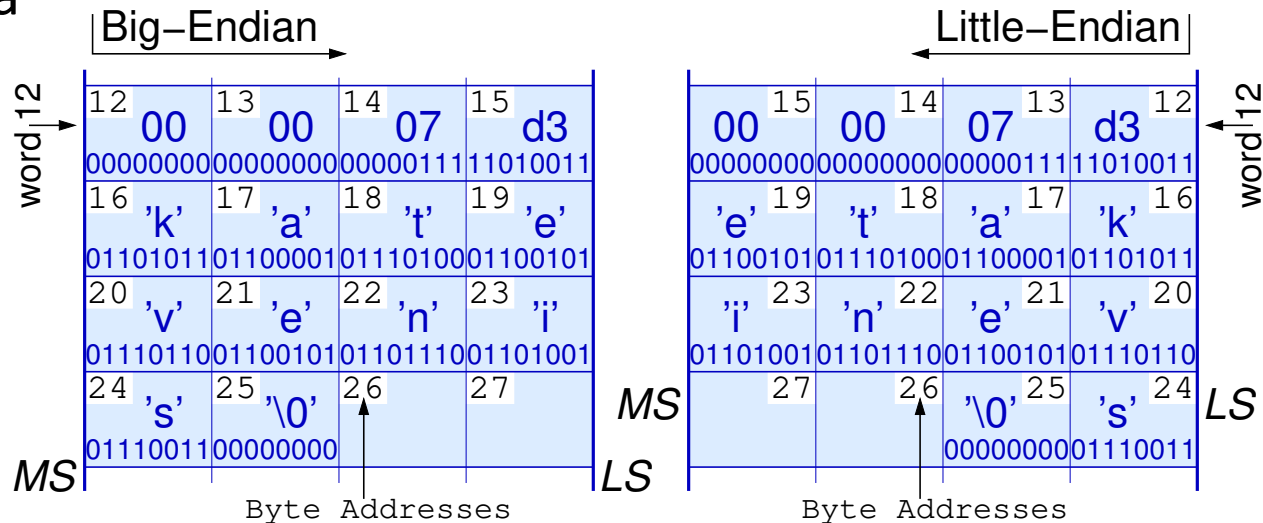
Δύο στρατόπεδα κατασκευαστών («Ιερός πόλεμος»)

Big-Endians versus Little-Endians: ο ιερός πόλεμος

- Big-Endians: «τα strings να μπορούμε να τα διαβάζουμε»!
- Little-Endians: «όπως αριθμούμε τα bits, έτσι & τα Bytes»!
- rfc-editor.org/ien/ien137.txt “On Holy Wars and a Plea for Peace”
- *Big-Endians*: IBM S/360, Motorola 68000, early PowerPC
- *Bi-Endians* (configured as either): ARM, PowerPC, MIPS
- *Little-Endians*: Intel x86, AMD, RISC-V, early ARM

Τι με νοιάζει εμένα; – (1) κανονικά προγράμματα

- char buf[10] με base addr. 16
- γράφω buf[0] = 'k'
- γράφω buf[1] = 'a'
- διαβάζω buf[0] → 'k'
- διαβάζω buf[1] → 'a'
- ίδια, σωστή συμπεριφορά σε ό,τι μηχανή και να τρέξω το πρόγραμμά μου
- int year=2003 στη διεύθυνση 12
- διαβάζω int από 12, βρίσκω 2003
- ίδια, σωστή συμπεριφορά σε ό,τι μηχανή και να τρέξω

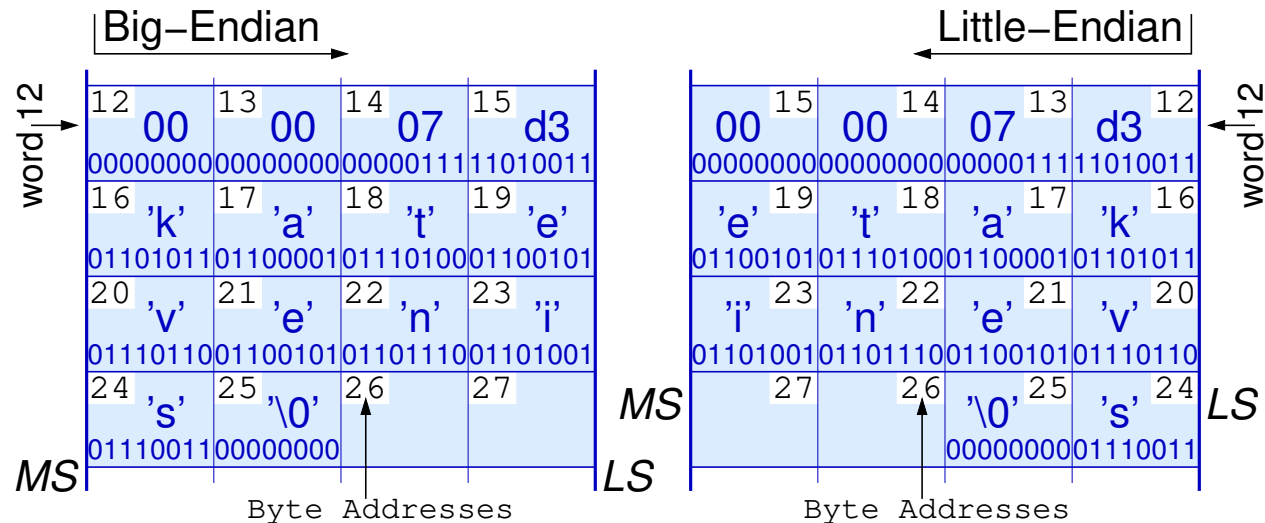


Τι με νοιάζει εμένα; – (2) «παράξενα» προγράμματα

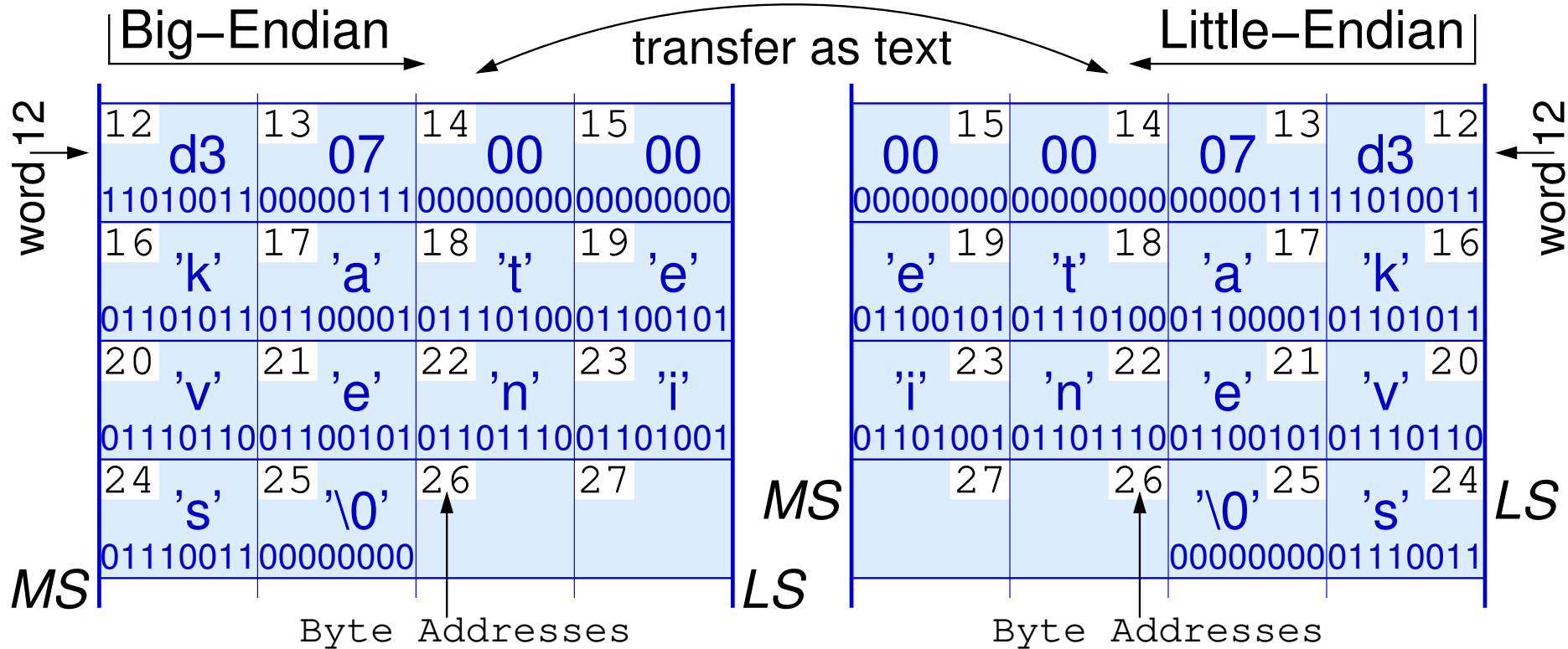
- γράφω string “katevenis” ξεκινώντας από τη διευθ. 16
 - διαβάζω int (!) από δ. 16:
 - αλλού (big-e.) → 6b617465
 - αλλού (little-e.) → 6574616b
- γράφω int year=2003 στη δ. 12
 - διαβάζω char (!) από διευθ. 12:
 - αλλού (big-e.) βρίσκω: ‘\0’ (00)
 - αλλού (little-e.) βρίσκω: ‘Σ’ (d3)

• non-portable program !

- η συμπεριφορά αλλάζει από μηχανή σε μηχανή
- (και μάλλον ανόητο πρόγραμμα)

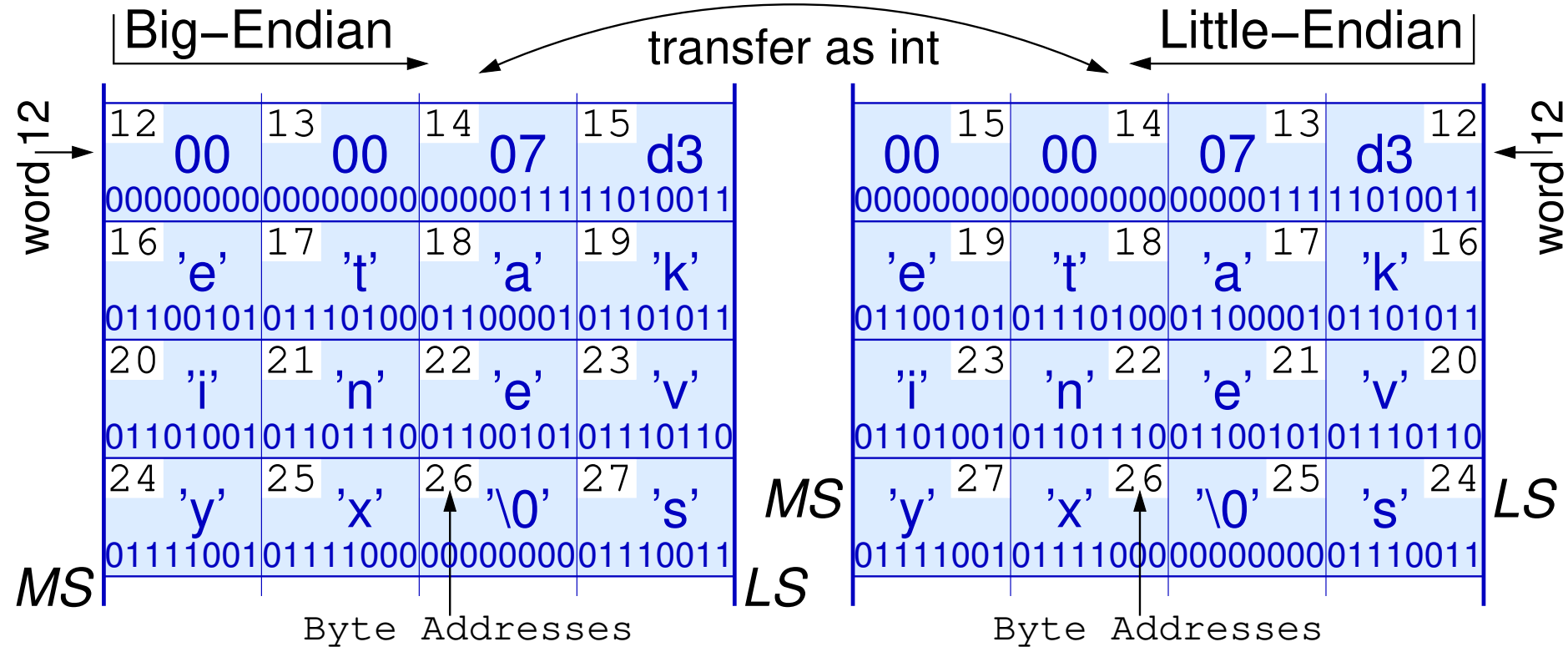


Τι με νοιάζει εμένα; – (3) μεταφορά αρχείου ως text



Ο ακέραιος 2003 γίνεται -754.515.968 στην άλλη μηχανή!

Τι με νοιάζει εμένα; – (4) μπτ. αρχείου ως binary (int)



To string @16 γίνεται “etakinevnx” στην άλλη μηχανή!

Προσπελάσεις σε Φαρδιές Μνήμες: Ευθυγράμμιση

- Οι πραγματικές μνήμες συνήθως φαρδιές (32, 64,... bits)
 - γιά λόγους ταχύτητας: περισσότερα Bytes σε κάθε πρόσβαση
 - εάν ο επεξεργαστής χρειάζεται λιγότερα Bytes, η μνήμη του δίνει όλα τα Bytes στην «γραμμή» εκείνη, και ο επεξεργαστής, εσωτερικά, επιλέγει και κρατά εκείνα που θέλει
 - χωριστό σήμα *writeEnable* γιά κάθε Byte position: γιά να γράψουμε λιγότερα Bytes από μία ολόκληρη «γραμμή», ανάβουμε μόνον τα *wrEnab's* εκεί που θέλουμε να γράψουμε
- Η πραγματική μνήμη προσπελάζει πάντα ευθυγραμμισμένες «γραμμές», στην κάθε πρόσβαση
- Πόσες προσπελάσεις χρειάζονται γιά ποσότητα n Bytes?

2-Byte "Half Words" Aligned on Addresses that are integer multiples of 2

| | |
|----|-----------|
| 1 | 0 |
| 3 | 2 |
| 5 | 4 |
| 7 | 6 |
| 9 | 8 |
| 11 | 10 |
| 13 | 12 |
| 15 | 14 |
| 17 | 16 |
| 19 | 18 |
| 21 | 20 |
| 23 | 22 |

Addresses drawn assuming: *Little Endian layout*

| | | | |
|----|-----------|----|-----------|
| 3 | 2 | 1 | 0 |
| 7 | 6 | 5 | 4 |
| 11 | 10 | 9 | 8 |
| 15 | 14 | 13 | 12 |
| 19 | 18 | 17 | 16 |
| 23 | 22 | 21 | 20 |

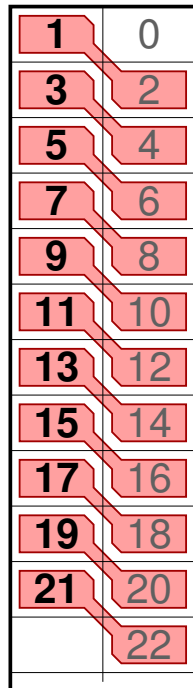
| | | | | | | | |
|----|-----------|----|-----------|----|-----------|----|-----------|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |

← 64 bits = 8 Bytes →

- Numbers inside boxes are Byte Addresses –NOT Contents
- The address of each 2-Byte "half word" is shown in Bold

(the address of a multi-Byte quantity is the address of that Byte inside it that has the smallest address among all the Bytes inside the quantity)

← 16 b
2 By →



← 32 bits = 4 Bytes →

2-Byte "Half Words" at Addresses that are NOT integer multiples of 2

| | | | |
|-----------|-----------|-----------|-----------|
| 3 | 2 | 1 | 0 |
| 7 | 6 | 5 | 4 |
| 11 | 10 | 9 | 8 |
| 15 | 14 | 13 | 12 |
| 19 | 18 | 17 | 16 |
| | 22 | 21 | 20 |

| | | | | | | | |
|-----------|-----------|-----------|-----------|-----------|----|----|----|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| | 22 | 21 | 20 | 19 | 18 | 17 | 16 |

← 64 bits = 8 Bytes →

- Some (even if not all) of these 2-Byte half-w. incur a performance penalty when accessed

4-Byte "Words" Aligned on Addresses that are integer multiples of 4

| | |
|----|-----------|
| 1 | 0 |
| 3 | 2 |
| 5 | 4 |
| 7 | 6 |
| 9 | 8 |
| 11 | 10 |
| 13 | 12 |
| 15 | 14 |
| 17 | 16 |
| 19 | 18 |
| 21 | 20 |
| 23 | 22 |

← 16 b
2 By →

Addresses drawn assuming: *Little Endian layout*

| | |
|-----------|----|
| 1 | 0 |
| 3 | 2 |
| 5 | 4 |
| 7 | 6 |
| 9 | 8 |
| 11 | 10 |
| 13 | 12 |
| 15 | 14 |
| 17 | 16 |
| 19 | 18 |
| | 20 |

| | | | |
|----|----|----|-----------|
| 3 | 2 | 1 | 0 |
| 7 | 6 | 5 | 4 |
| 11 | 10 | 9 | 8 |
| 15 | 14 | 13 | 12 |
| 19 | 18 | 17 | 16 |
| 23 | 22 | 21 | 20 |

← 32 bits = 4 Bytes →

(the address of a multi-Byte quantity is the address of that Byte inside it that has the smallest address among all the Bytes inside the quantity)

| | | | | | | | |
|----|----|----|-----------|----|----|----|-----------|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |

← 64 bits = 8 Bytes →

- Numbers inside boxes are Byte Addresses –NOT Contents
- The address of each 4-Byte "word" is shown in Bold

4-Byte "Words" at Addresses that are NOT multiples of 4, but are 1-off, i.e. $\text{Addr mod } 4 == 1$

| | | | |
|----|----|-----------|----|
| 3 | 2 | 1 | 0 |
| 7 | 6 | 5 | 4 |
| 11 | 10 | 9 | 8 |
| 15 | 14 | 13 | 12 |
| 19 | 18 | 17 | 16 |
| | | | 20 |

| | | | | | | | |
|----|----|-----------|----|----|----|-----------|----|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| | | | 20 | 19 | 18 | 17 | 16 |

← 64 bits = 8 Bytes →

- Some (even if not all) of these 4-Byte words incur a performance penalty when accessed

4-Byte "Words" Aligned on Addresses that are integer multiples of 4

| | |
|----|-----------|
| 1 | 0 |
| 3 | 2 |
| 5 | 4 |
| 7 | 6 |
| 9 | 8 |
| 11 | 10 |
| 13 | 12 |
| 15 | 14 |
| 17 | 16 |
| 19 | 18 |
| 21 | 20 |
| 23 | 22 |

← 16 b
2 By →

Addresses drawn assuming: *Little Endian layout*

| | |
|----|-----------|
| 1 | 0 |
| 3 | 2 |
| 5 | 4 |
| 7 | 6 |
| 9 | 8 |
| 11 | 10 |
| 13 | 12 |
| 15 | 14 |
| 17 | 16 |
| 19 | 18 |
| 21 | 20 |

4-Byte words at multiples of 2 but not of 4 are OK in 2-Byte wide memories, but NOT in wider!

| | | | |
|----|----|----|-----------|
| 3 | 2 | 1 | 0 |
| 7 | 6 | 5 | 4 |
| 11 | 10 | 9 | 8 |
| 15 | 14 | 13 | 12 |
| 19 | 18 | 17 | 16 |
| 23 | 22 | 21 | 20 |

← 32 bits = 4 Bytes →

(the address of a multi-Byte quantity is the address of that Byte inside it that has the smallest address among all the Bytes inside the quantity)

4-Byte "Words" at Addresses that are multiples of 2, but NOT multiples of 4 (i.e. $\text{Addr mod } 4 == 2$)

| | | | |
|----|-----------|----|----|
| 3 | 2 | 1 | 0 |
| 7 | 6 | 5 | 4 |
| 11 | 10 | 9 | 8 |
| 15 | 14 | 13 | 12 |
| 19 | 18 | 17 | 16 |
| | | 21 | 20 |

| | | | | | | | |
|----|----|----|-----------|----|----|----|-----------|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |

← 64 bits = 8 Bytes →

- Numbers inside boxes are Byte Addresses –NOT Contents
- The address of each 4-Byte "word" is shown in Bold

| | | | | | | | |
|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| | | 21 | 20 | 19 | 18 | 17 | 16 |

← 64 bits = 8 Bytes →

- Some (even if not all) of these 4-Byte words incur a performance penalty when accessed

8-Byte "Double Words" Aligned on Addresses that are integer multiples of 8

| | |
|---|----------|
| 1 | 0 |
| 3 | 2 |
| 5 | 4 |
| 7 | 6 |

Addresses drawn assuming: *Little Endian layout*

| | | | |
|----|----|----|-----------|
| 3 | 2 | 1 | 0 |
| 7 | 6 | 5 | 4 |
| 11 | 10 | 9 | 8 |
| 15 | 14 | 13 | 12 |
| 19 | 18 | 17 | 16 |
| 23 | 22 | 21 | 20 |

| | | | | | | | |
|----|----|----|----|----|----|----|-----------|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |

64 bits = 8 Bytes

- Numbers inside boxes are Byte Addresses –NOT Contents
- The address of each 8-Byte "double word" is shown in Bold

(the address of a multi-Byte quantity is the address of that Byte inside it that has the smallest address among all the Bytes inside the quantity)

| | |
|----|-----------|
| 9 | 8 |
| 11 | 10 |
| 13 | 12 |
| 15 | 14 |
| 17 | 16 |
| 19 | 18 |
| 21 | 20 |
| 23 | 22 |

| | |
|----|-----------|
| 1 | 0 |
| 3 | 2 |
| 5 | 4 |
| 7 | 6 |
| 9 | 8 |
| 11 | 10 |
| 13 | 12 |
| 15 | 14 |
| 17 | 16 |
| 19 | 18 |
| 21 | 20 |
| 23 | 22 |

32 bits = 4 Bytes

8-Byte "Doubles" at Addresses that are multiples of 2, but NOT multiples of 4 or 8 (i.e. $\text{Addr mod } 8 == 2$)

16 b
2 By

8-Byte doubles at multiples of 2 but not of 4 or 8 are OK in 2-Byte wide memories, but NOT in wider!

| | | | |
|----|-----------|----|----|
| 3 | 2 | 1 | 0 |
| 7 | 6 | 5 | 4 |
| 11 | 10 | 9 | 8 |
| 15 | 14 | 13 | 12 |
| 19 | 18 | 17 | 16 |
| 23 | 22 | 21 | 20 |

| | | | | | | | |
|----|----|----|----|----|-----------|----|----|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |

64 bits = 8 Bytes

- In 4- and 8-wide memories, all of these doubles incur a performance penalty when accessed

8-Byte "Double Words" Aligned on Addresses that are integer multiples of 8

| | |
|---|----------|
| 1 | 0 |
| 3 | 2 |
| 5 | 4 |
| 7 | 6 |

Addresses drawn assuming: *Little Endian layout*

| | | | |
|----|----|----|-----------|
| 3 | 2 | 1 | 0 |
| 7 | 6 | 5 | 4 |
| 11 | 10 | 9 | 8 |
| 15 | 14 | 13 | 12 |
| 19 | 18 | 17 | 16 |
| 23 | 22 | 21 | 20 |

← 32 bits = 4 Bytes →

| | | | | | | | |
|----|----|----|----|----|----|----|-----------|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |

← 64 bits = 8 Bytes →

- Numbers inside boxes are Byte Addresses –NOT Contents
- The address of each 8-Byte "Double word" is shown in Bold

(the address of a multi-Byte quantity is the address of that Byte inside it that has the smallest address among all the Bytes inside the quantity)

| | |
|----|-----------|
| 9 | 8 |
| 11 | 10 |
| 13 | 12 |
| 15 | 14 |
| 17 | 16 |
| 19 | 18 |
| 21 | 20 |
| 23 | 22 |

← 16 b
2 By →

| | |
|----|-----------|
| 1 | 0 |
| 3 | 2 |
| 5 | 4 |
| 7 | 6 |
| 9 | 8 |
| 11 | 10 |
| 13 | 12 |
| 15 | 14 |
| 17 | 16 |
| 19 | 18 |
| 21 | 20 |
| 23 | 22 |

8-Byte "Doubles" at Addresses that are multiples of 4, but NOT multiples of 8 (i.e. $\text{Addr mod } 8 == 4$)

| | | | |
|----|----|----|-----------|
| 3 | 2 | 1 | 0 |
| 7 | 6 | 5 | 4 |
| 11 | 10 | 9 | 8 |
| 15 | 14 | 13 | 12 |
| 19 | 18 | 17 | 16 |
| 23 | 22 | 21 | 20 |

| | | | | | | | |
|----|----|----|-----------|----|----|----|----|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |

← 64 bits = 8 Bytes →

- In 8-Byte wide memories, these Doubles incur a performance penalty when accessed

8-Byte Doubles at multiples of 4 but not of 8 are OK in 2- & 4-wide memories, but NOT in wider!

Γιά ταχύτητα: Ευθυγράμμιση στα «φυσικά όρια»

Ηθικόν Δίδαγμα:

- Γιά τον ελάχιστο δυνατό αριθμό προσπελάσεων μνήμης,
- σε μνήμη οιουδήποτε πλάτους (πάντα δύναμη του 2)
 - δηλαδή γιά προγράμματα με “portable efficiency”:
- Half-words (2 Bytes) σε διευθ. ακέραια πολλαπλ. του 2
- Words (4 Bytes) σε διευθύνσεις ακέραια πολλαπλ. του 4
- Double-words (8 Bytes) σε διευθ. ακερ. πολλαπλ. του 8
- Γενικά: Ευθυγράμμιση της κάθε ποσότητας (2^n By) στα «φυσικά» της όρια (ακερ. πολ. του 2^n)

Οδηγίες στον Assembler για Ευθυγράμμιση

- Σε μερικούς επεξ. υποχρεωτική η ευθυγράμ. (π.χ. MIPS)
- Σε άλλους προαιρετική: **RISC-V** (κ.α.)
 - παντού συμφέρει σε ταχύτητα, έστω και με κάποια κενά στη χρησιμοπ. του χώρου μνήμης (λιγότερα εάν βελτιστοποιηθούν)
- Οδηγίες Assembler να ευθυγραμμίσει το επόμενο item:
 - .align 2 \Rightarrow σε ακέραιο πολλαπλ. του $2^2 = 4$
 - .align 3 \Rightarrow σε ακέραιο πολλαπλ. του $2^3 = 8$
κάνε τη διεύθ. όπου θα αρχίσει το επόμενο πράγμα στο data segment ακερ. πολ. του 2^n (αυξάνοντας την λίγο αν δεν είναι ήδη)
 - .space 32 \Rightarrow κράτησε εδώ χώρο 32 Bytes (άφησε κενό)

Παράδειγμα: .align .space

```
mystr: .asciz "katevenis"  
       .align 3 # πήδα στο επόμενο  
              ακέρ. πολ. του 23=8  
  
p_dbl: .space 8 # ονόμασε τη θέση  
         "p_dbl" και κράτα  
         χώρο 8 Bytes εκεί  
  
i_wrd: .space 4 # επόμ. θέση ονόματι  
         "i_wrd" και κράτα  
         χώρο 4 Bytes εκεί
```

| | | |
|--------|-----|----|
| mystr= | 400 | k |
| | 401 | a |
| | 402 | t |
| | 403 | e |
| | 404 | v |
| | 405 | e |
| | 406 | n |
| | 407 | i |
| | 408 | s |
| | 409 | \0 |
| | 410 | |
| | 411 | |
| | 412 | |
| | 413 | |
| | 414 | |
| | 415 | |
| p_dbl= | 416 | |
| | 417 | |
| | 418 | |
| | 419 | |
| | 420 | |
| | 421 | |
| | 422 | |
| | 423 | |
| i_wrd= | 424 | |
| | 425 | |
| | 426 | |
| | 427 | |
| | 428 | |

Ευθυγραμμισμένες Ποσότητες & τα bits Διεύθυνσης

| | |
|----------------|----------------|
| 00000 1 | 00000 0 |
| 00001 1 | 00001 0 |
| 00010 1 | 00010 0 |

| | | | |
|----------------|----------------|----------------|----------------|
| 0000 11 | 0000 10 | 0000 01 | 0000 00 |
| 0001 11 | 0001 10 | 0001 01 | 0001 00 |
| 0010 11 | 0010 10 | 0010 01 | 0010 00 |

| | | | | | | | |
|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 000 111 | 000 110 | 000 101 | 000 100 | 000 011 | 000 010 | 000 001 | 000 000 |
| 001 111 | 001 110 | 001 101 | 001 100 | 001 011 | 001 010 | 001 001 | 001 000 |
| 010 111 | 010 110 | 010 101 | 010 100 | 010 011 | 010 010 | 010 001 | 010 000 |

- Στις ευθυγραμμισμένες ποσότητες μεγέθους 2^n Bytes, τα n LS bits της διεύθυνσης του κάθε Byte τους δείχνουν τη θέση του Byte μέσα στην ποσότητα, και τα bits αυτά είναι όλα 0 στη διεύθυνση της ποσότητας, ενώ τα υπόλοιπα bits της διεύθ. είναι όλα ίδια σε όλα τα Bytes της ποσότητας
- Ομοίως: «Γραμμές» Κρυφών Μνημών, «Σελίδες» Εικονικής Μνήμης