

Βρόχοι και
Επικοινωνία Κονσόλας στον RARS,
Προσημασμένοι και Επέκταση Προσήμου

02α (§2) – 13 Φεβ. 2026 – Μανόλης Κατεβαίνης

Διακλαδώσεις υπό Συνθήκη στον RISC-V

- `beq x26, x27, label` # if ($x26 == x27$) goto label
- `bne x26, x27, label` # if ($x26 \neq x27$) goto label
- `blt x26, x27, label` # if ($x26 < x27$) goto label
- `bge x26, x27, label` # if ($x26 \geq x27$) goto label
- Σύνθεση άλλων συγκρίσεων μέσω αυτών (ασκ. 5.2)
- Δεν υπάρχουν συγκρίσεις καταχωρητή-σταθεράς
—γιατί; (ασκ. 5.2)

[ιστιq ποση]

Παράδειγμα Βρόχου: $1+2+3+\dots+(n-1)$

compute $s = 1+2+3+\dots+(n-1)$, for $n \geq 2$

register x26: n

register x27: s

register x28: i

$s=0; i=1;$

do { $s = s+i;$

$i = i+1;$

} while ($i \neq n$)

```
22      add      x27, x0, x0      # s=0;
23      addi     x28, x0, 1       # i=1;
24  loop:                               # (4) LOOP starts here
25      add      x27, x27, x28    # s=s+i;
26      addi     x28, x28, 1      # i=i+1;
27      bne      x28, x26, loop   # repeat while (i!=n)
28                               # LOOP ENDS HERE
```

I/O: Environment (System) Calls

- Υποτυπώδης επικοινωνία κονσόλας στον RARS
- `scanf()`, `printf()` είναι διαδικασίες βιβλιοθήκης
 - όταν χρειαστεί, καλούν το λειτουργικό σύστημα γιά να μιλήσουν πραγματικά σε πληκτρολόγιο, οθόνη, αρχεία, κλπ.
- *Environment (System) Call* (κάλεσμα περιβάλλοντος/λειτουργικού)
 - σαν κάλεσμα διαδικασίας, αλλά με προστασία!
 - μόνον από την «κύρια είσοδο» του Λειτουργικού Συσ.
 - ώστε να μην παρακάμπονται οι εκεί έλεγχοι ασφαλείας
 - ⇒ εντολή `ecall` χωρίς ορίσματα (χωρίς διεύθυνση)
 - ποιό απ' όλα τα καλέσματα περιβάλλοντος;; → `x17`

ecall υπ' αριθμόν 1: Print Integer

```
addi    x17, x0, 1      # environment call code for print_int
add     x10, x27, x0     # copy argument s to x10
ecall                   # print the integer in x10 (s)
```

- ποιό απ' όλα τα καλέσματα περιβάλλοντος;
— $x17 = 1$;
- Ποιόν Ακέραιο;
— $x10 = s$; /* ο ακέραιος που θέλουμε να τυπωθεί */
- Τα ορίσματα πρέπει να δοθούν μέσω των $x17$ και $x10$
συγκεκριμένα: Σύμβαση Κλήσης Περιβάλλοντος

ecall υπ' αριθμόν 4: Print String

```
addi    x17, x0, 4      # environment call code for print_string
la      x10, str_n      # pseudo-instruction: address of string
ecall                               # print the string from str_n
```

- ποιό απ' όλα τα καλέσματα περιβάλλοντος;
 - $x17 = 4$;
- Ποιό String?
 - array of char's $\Rightarrow$ στη μνήμη – όχι σε καταχωρητή/τες
 - σε ποιά διεύθυνση μνήμης; $\rightarrow$ όρισμα στον $x10$
 - label = διεύθυνση μνήμης (pointer)
 - la (load address): Ψευδοεντολή ... πολλά bits η διευθ.

Θυμηθείτε / Σκεφτείτε:

- `ecall` (*environment call*):
 - κλήση σε ποια διεύθυνση;
- Σύμβαση κλήσης – ορίσματα, επιστρεφόμενη τιμή:
 - τι υπηρεσία ζητάμε;
 - πρώτο όρισμα για αυτό που ζητάμε;
- `la` (*load address*):
 - κανονική εντολή;
 - διεύθυνση ξέρουμε; πώς την δίνουμε;
 - χωράει; δεν χωράει; μία ή δύο εντολές;

Θυμηθείτε / Σκεφτείτε:

- `ecall` (*environment call*):
 - κλήση σε ποια διεύθυνση; → **όχι εμείς – το hardware (προστασία!)**
- Σύμβαση κλήσης – ορίσματα, επιστρεφόμενη τιμή:
 - τι υπηρεσία ζητάμε; → **x17**
 - πρώτο όρισμα για αυτό που ζητάμε; → **x10**
- `la` (*load address*):
 - κανονική εντολή; → **όχι – «ψευδοεντολή»**
 - διεύθυνση ξέρουμε; πώς την δίνουμε; → **label (ορισμένο αλλού)**
 - χωράει; δεν χωράει; μία ή δύο εντολές; → **ο Assembler φροντίζει**

Αρχικοποιημένα Data στη Μνήμη

```
6      .data                # init. data memory with the strings needed
7  str_n: .asciz "n = "
8  str_s: .asciz "      s = "
9  str_nl: .asciz "\n"
10
11      .text                # program memory:
12
13  main:                    # (1) PRINT A PROMPT:
```

- Assembler Directives (“directive”)
- `.data` → τα επόμ. στο Data Segm. `.text` → στο Text Segm.
- `.asciz` → zero-terminated ASCII string αρχικοποιημ. μν.
- `label:` → ορίζει συμβολικό όνομα για αυτή τη διεύθυνση

ecall υπ' αριθμόν 5: Read Integer

```
addi    x17, x0, 5      # environment call code for read_int
ecall                                # read a line containing an integer
add     x26, x10, x0    # copy returned int from x10 to n
```

- ποιό απ' όλα τα καλέσματα περιβάλλοντος;
 - $x17 = 5$;
- Κάλεσμα περιβάλλοντος να διαβάσει από πληκτρολόγιο
- Πού θα μου επιστρέψει την τιμή που ζήτησα;
 - $x10 \leftarrow$ η επιστρεφόμενη τιμή από το κάλεσμα περιβ.
 - την έχει ήδη μετατρέψει από string σε int

```

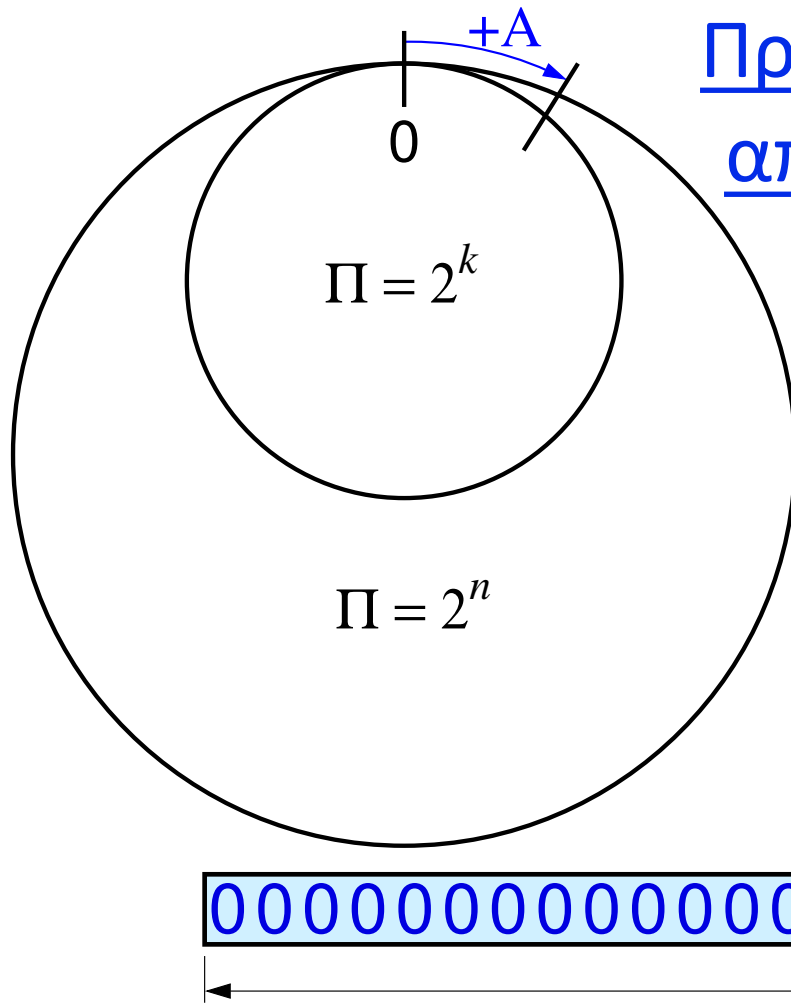
1      # compute s = 1+2+3+...+(n-1), for n>=2
2      # register x26: n
3      # register x27: s
4      # register x28: i
5
6      .data          # init. data memory with the strings needed:
7  str_n: .asciz "n = "
8  str_s: .asciz "      s = "
9  str_nl: .asciz "\n"
10
11     .text          # program memory:
12
13  main:              # (1) PRINT A PROMPT:
14      addi    x17, x0, 4      # environment call code for print_string
15      la      x10, str_n     # pseudo-instruction: address of string
16      ecall                   # print the string from str_n
17      # (2) READ n (MUST be n>=2 --not checked!):
18      addi    x17, x0, 5      # environment call code for read_int
19      ecall                   # read a line containing an integer
20      add     x26, x10, x0     # copy returned int from x10 to n
21      # (3) INITIALIZE s and i:
22      add     x27, x0, x0     # s=0;
23      addi    x28, x0, 1      # i=1;
24  loop:              # (4) LOOP starts here
25      add     x27, x27, x28    # s=s+i;
26      addi    x28, x28, 1      # i=i+1;
27      bne     x28, x26, loop   # repeat while (i!=n)
28      # LOOP ENDS HERE
29      # (5) PRINT THE RESULT:
30      addi    x17, x0, 4      # environment call code for print_string
31      la      x10, str_s     # pseudo-instruction: address of string
32      ecall                   # print the string from str_s
33      addi    x17, x0, 1      # environment call code for print_int
34      add     x10, x27, x0     # copy argument s to x10
35      ecall                   # print the integer in x10 (s)
36      addi    x17, x0, 4      # environment call code for print_string
37      la      x10, str_nl    # pseudo-instruction: address of string
38      ecall                   # print a new-line
39      # (6) START ALL OVER AGAIN (infinite loop)
40      j       main           # unconditionally jump back to main

```

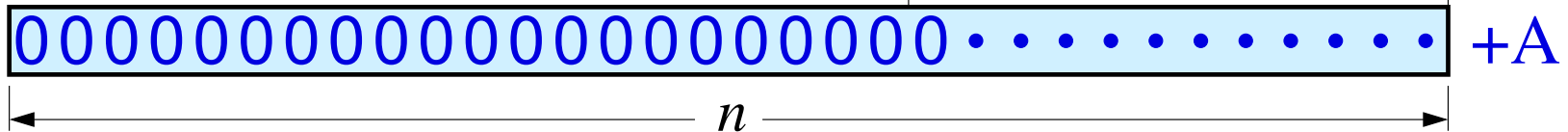
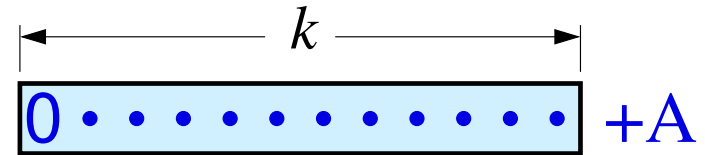
Μετατροπές Ακεραίων από λίγα σε πολλά bits

- Απρόσημοι (Unsigned) ακέραιοι (≥ 0) (HY-120 Εργ.5):
 - $-b_{n-1} \times 2^{n-1} + \dots + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$
 - πάντα ο ίδιος αριθμός, οσαδήποτε μηδενικά και αν προστεθούν αριστερά από τον MS άσσο
- Προσημασμένοι (Signed) ακέραιοι σε 2's Complement:
 - Όταν MS bit == 0 $\Rightarrow \geq 0$, ίδιος όπως εάν unsigned
 - Όταν MS bit == 1 $\Rightarrow < 0$, όσος εάν unsigned μείον 2^j
όπου j το πλήθος των bits του αριθμού
 - Όταν το j αλλάζει??

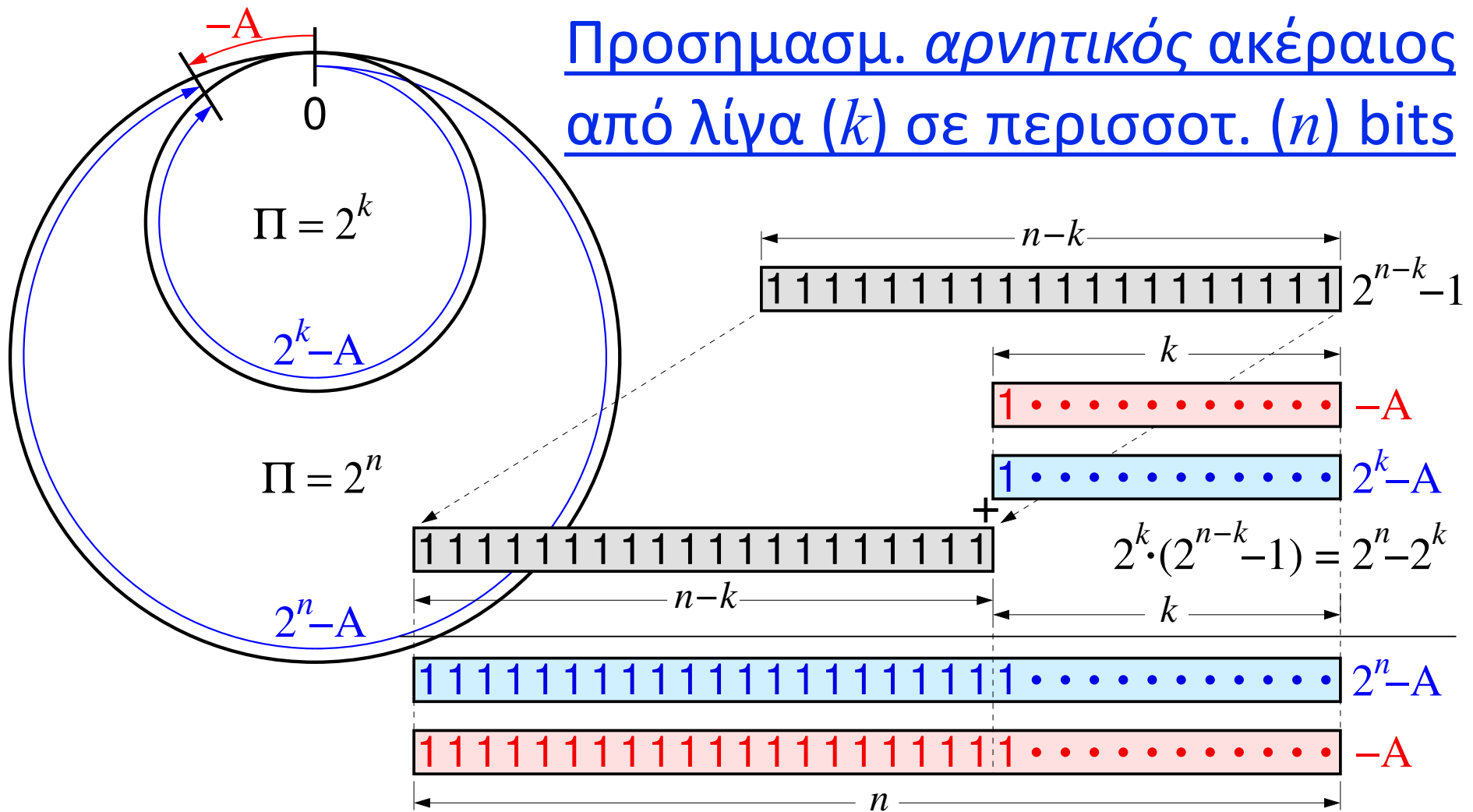
Προσημασμένος θετικός ακέραιος από λίγα (k) σε περισσοτ. (n) bits



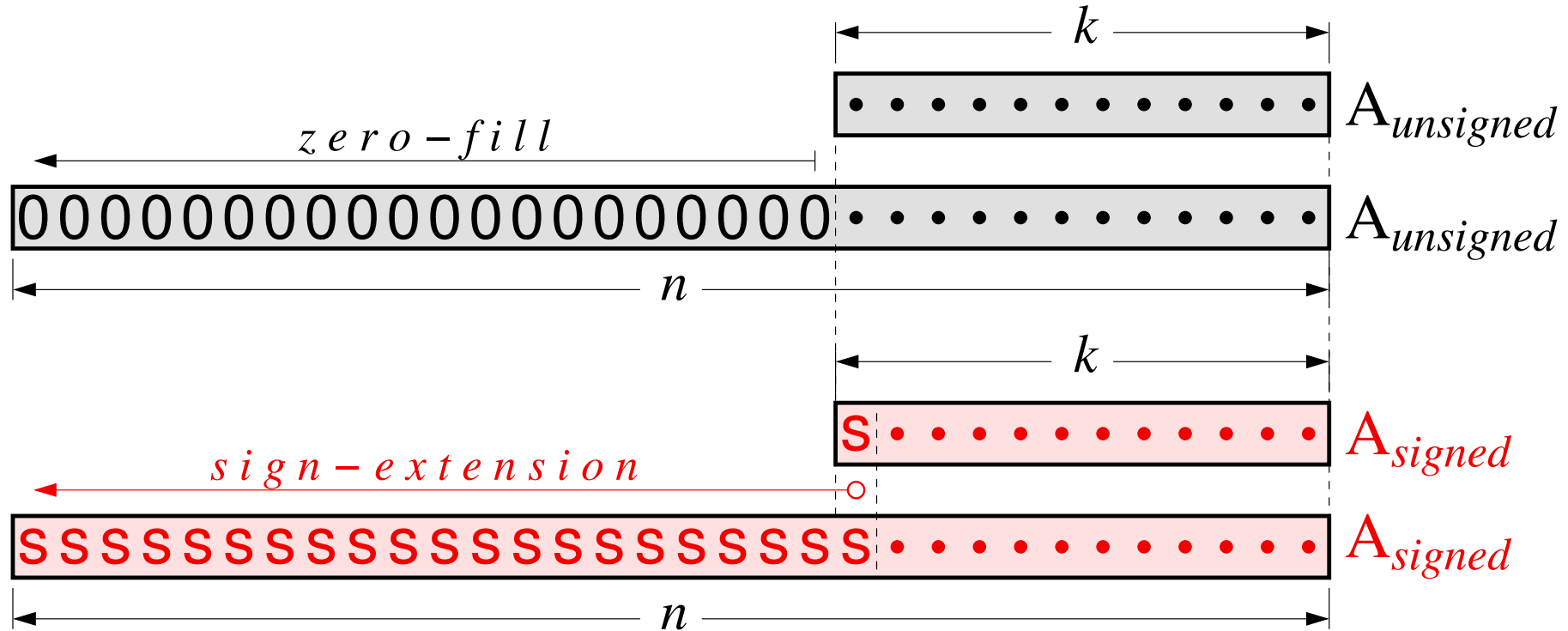
- MS bit == 0
- ίδιος αριθμός όπως και απρόσημος
- ίδιος οσαδήποτε μηδενικά αριστερά



Προσημασμ. αρνητικός ακέραιος από λίγα (k) σε περισσοτ. (n) bits



Προσημασμένοι σε περισ. bits: Επέκταση Προσήμου



- Απρόσημοι: Συμπλήρωση Μηδενικών (ανεξαρτήτως MS bit)
- Προσημασμένοι (όλα τα *Immed.* στον *RV*): Επέκταση Προσήμου