Διακλαδώσεις υπό Συνθήκη, PC-relative Addressing, Βελησνεκές, Relocatable Code, If-then-else, Βρόχοι

05a (§5.1-5.6) – 5-8 Μαρτίου 2021 – Μανόλης Κατεβαίνης

Διακλαδώσεις υπό Συνθήκη

- `beq rs1, rs2, label` # if (rs1==rs2) goto label
- `bne rs1, rs2, label` # if (rs1 ≠ rs2) goto label
- `blt rs1, rs2, label` # if (rs1 < rs2) goto label
- `bge rs1, rs2, label` # if (rs1 ≥ rs2) goto label
- `bltu  bgeu` → unsigned variants
  - υποθέτουν ότι οι καταχ. περιέχουν μη προσημασμ. ακεραίους
  - οι αρνητικοί σε signed είναι μεγάλοι θετικοί σε unsigned
- γιατί δεν χρειάζονται `bequ`, `bneu` ? (άσκηση…)
- γιατί δεν χρειάζονται `ble` (≤), `bgt` (>) ? (άσκηση…)

Array bounds checking (in C) via unsigned compare

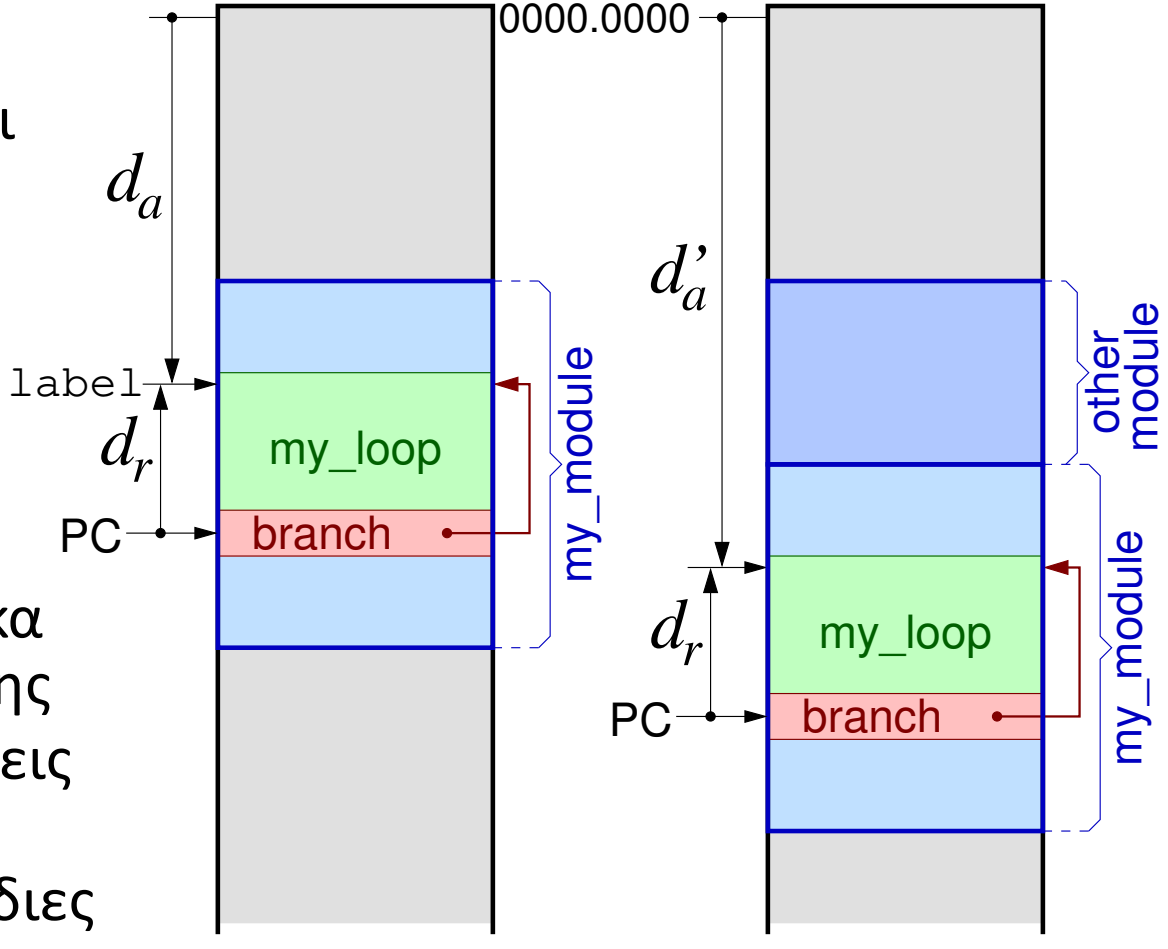
- register int N, i; /* i is signed */ my_type array[N];
- access array[i] → is i in-bounds? ⇔ $0 \le i < N$

```
bgeu i, N, out_of_bounds
```

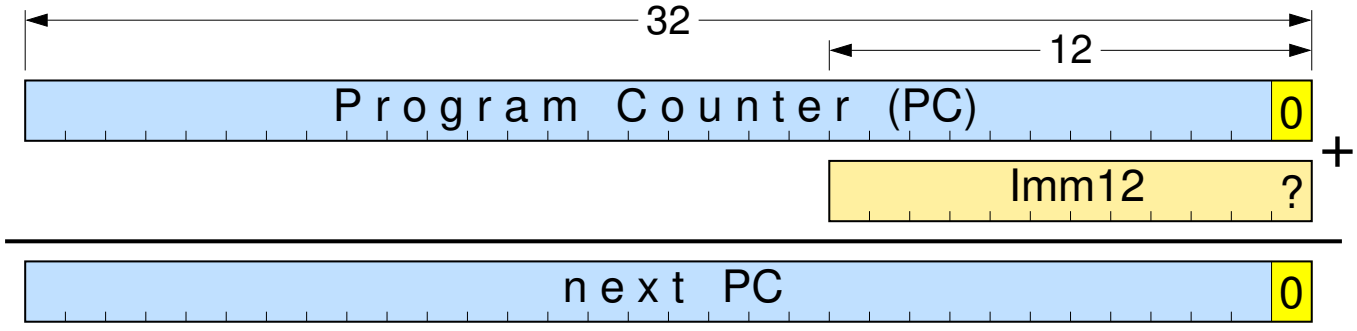
- εάν i θετικός και i ≥ N, τότε όντως i is out_of_bounds
- εάν i αρνητικός, άρα out_of_bounds, τότε σε σύγκριση μη προσημασμένων το i εμφανίζεται ως πολύ μεγάλος θετικός (i ≥ 2^{31} = 2G γιά 32-μπιτο υπολογιστή), άρα πάλι αληθεύει η συνθήκη `bgeu i, N`
  - επειδή στην C οι πίνακες αρχίζουν από το 0, και υποθ. N < 2^{31}…

PC-relative Addressing Mode

- $d_r \ll d_a$
  - συνήθως βρόχοι και if-then-else είναι μικρά σε μέγεθος
  - συνήθως το text είναι σε σχετικά μεγάλες διευθυν.
- Relocation
  - φόρτωση του κώδικα σε άλλη θέση μνήμης
  - απόλυτες διευθύνσεις d_a αλλάζουν σε d_a'
  - σχετικές διευθ. d_r ίδιες

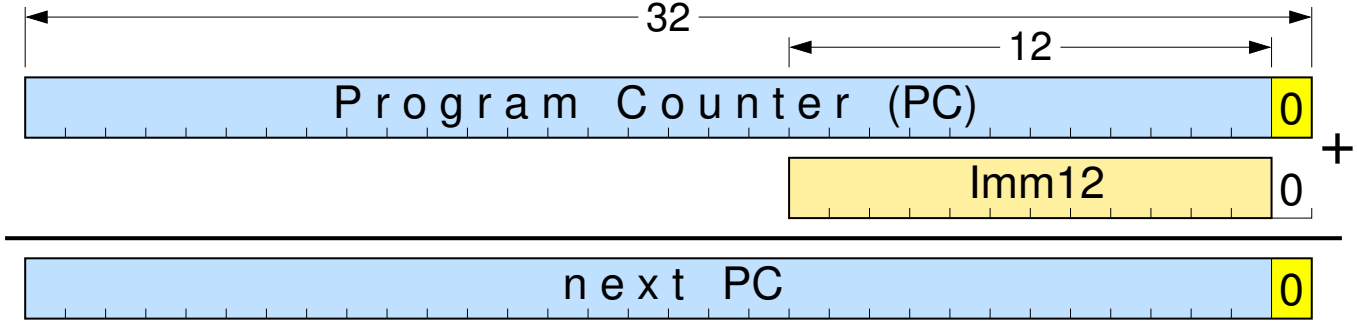


PC-relative addr.: Ευθυγράμμιση εντολών & Βελησνεκές



Απλοϊκή πρόσθεση ⇒ Βελησνεκές = ±2 KBytes

Ευθυγράμμιση εντ. RISC-V πάντα σε όρια half-words ⇒ δεξιό bit PC πάντα 0 ⇒ δεξιό bit σταθεράς υποχρεωτικά 0 → σπατάλη πληροφ.



Πρόσθεση διπλασίου ⇒ Βελησνεκές = ±2 KHalfWrd = ±4 KBytes

Το Imm12 μετρά σε μονάδες Half-Words

Διακλαδώσεις RISC-V: PC-relative Addressing

`beq rs1, rs2, Imm12` $\Rightarrow$

if (rs1==rs2) {PC $\leftarrow$ PC+2×Imm12} else {PC $\leftarrow$ PC+4}

- Η σταθερά Imm12 (offset) πάντα προσημασμένη
- Θετικά offsets: συνήθως if-then-else
- Αρνητικά offsets: συνήθως βρόχοι

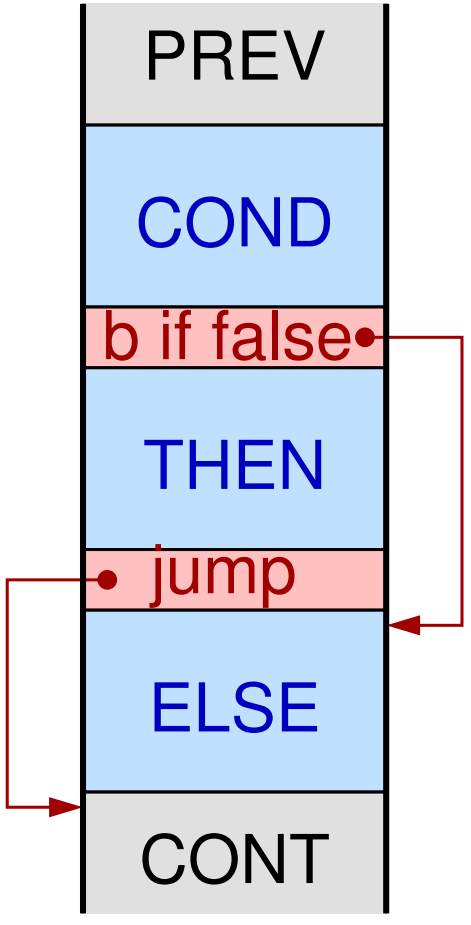
Μετάφραση του if-then-else σε Assembly

register int i, j, f, g, h;

if (i==j) { f = g+h; } else { f = g-h; }

h = 0;

```
        bne i, j, else1
        add f, g, h
        j  cont1          #jump
else1:  sub f, g, h
cont1:  add h, zero, zero
```

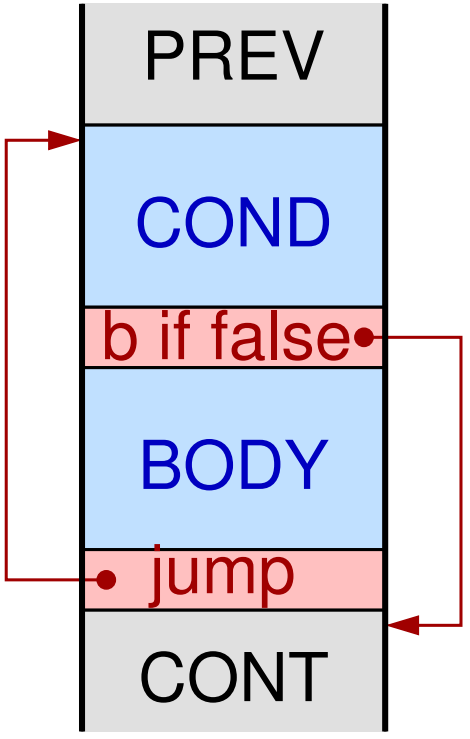


Μετάφραση Βρόχου while σε Assembly

```
long long int  i, key, table[N];
i = 0;
while ( table[i] != key ) { i++; }
      /* table[] must contain at least one element==key */
```

t1: i
t2: key
s0: table

```
        add  t1, x0, x0   # i=0;
loop1:  slli t0, t1, 3    # 8*i
        add  t0, s0, t0   # &table[i]
        ld   t0, 0(t0)    # table[i]
        beq  t0, t2, exit1
        addi t1, t1, 1    # i++;
        j  loop1
exit1:  ...
```



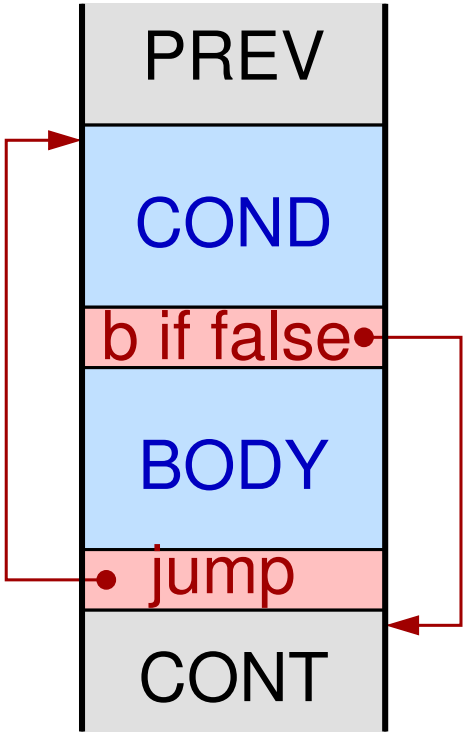
Βελτιστοποίηση Βρόχου με pointers

```
long long int  *p, key, table[N];
p = table;
while ( *p != key ) { p++; }
```

t1: p
t2: key
s0: table

/* table[] must contain at least one element==key */

```
        add   t1, s0, x0   # p= table;
loop1:  ld    t0, 0(t1)    # t0=*p
        beq   t0, t2, exit1
        addi  t1, t1, 8    # p++;
        j   loop1
exit1:  ...
```



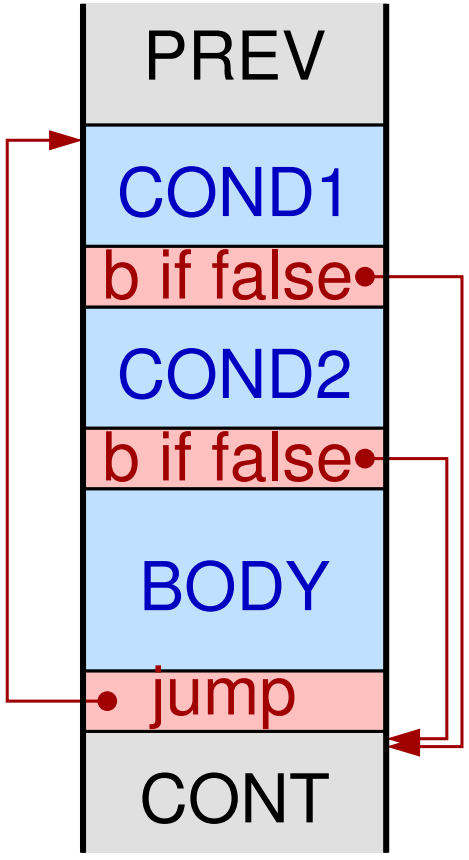
Άσκηση 5.5: αλλάξτε τη χωροθέτηση των στοιχείων του βρόχου ούτως ώστε οι ανακυκλώσεις μετά την είσοδο να εκτελούν μόνο μία εντολή CTI καθεμία

Σύνθετη Συνθήκη με υπολογισμό "Short-Circuit"

```
struct node {int value; struct node *next;} *p;
while ( p!=NULL && p->value != key )
        { p = p->next; }
    /* assume 32-bit RISC-V here */
```

t1: p
t2: key

```
loop2: beq  t1, x0, exit2
       lw   t0, 0(t1)    # p->value
       beq  t0, t2, exit2
       lw   t1, 4(t1)    # p=p->next;
       j    loop2
exit2: ...
```



- Εάν ο p είναι NULL, *απαγορεύεται* να υπολογίσουμε (δηλ. προσπελάσουμε) το p->value