Άλματα σε σταθερή ή μεταβλητή διεύθυνση, Κλήση & Επιστροφή από Διαδικασία, Switch, Συγκρίσεις με αποτελ. Boolean

05b (§5.7-5.10) – 4-9 Μαρτίου 2022 – Μανόλης Κατεβαίνης

Άλματα: PC-relative Addr., όπως και οι διακλαδώσεις

- Όπως οι διακλαδώσεις υπό συνθήκη στον RISC-V:

  `beq rs1, rs2, Imm12` ⇒

  if (rs1==rs2) {PC ← PC + 2×Imm12} else {PC ← PC+4}

- Έτσι και τα άλματα χωρίς συνθήκη, γιά τους ίδιους λόγους:

  `j ImmXX` ⇒ PC ← PC + 2×ImmXX

- Η σταθερά ImmXX (offset) πάντα προσημασμένη
- Πόσα bits "XX" η σταθερά??
  - θα μπορούσε να έχει «πολλά» bits, αλλά άχρηστο γιά τις «σκέτες» jump: χρησιμοποιούνται εντός διαδικασίας, μόνον…

Άλμα με αποθ. Διεύθ. Επιστροφής: Κλήση Διαδικασίας

- Άλματα σε μεγαλύτερη απόσταση: Κλήση Διαδικασίας
- "Jump-and-Link": απλό άλμα + σωσ. Διεύθ. Επιστροφής:

`jal rd, Imm20` $\Rightarrow$ $rd \leftarrow PC_{old} + 4$,

$PC_{new} \leftarrow PC_{old} + 2 \times Imm20$

- Η «από κάτω» εντολή στο $PC_{old}+4$ = Διεύθυνση Επιστροφής
- Η σταθερά Imm20 (offset) – πάντα *προσημασμένη*:
  - 20 bits: όσο χωρούσε στο J-format
  - επαρκής γιά κάλεσμα έως ±0.5 MHalfWords = ±1 Mbyte
  - υπερκαλύπτει και τις συνηθισμένες ανάγκες της «σκέτης» jump

Πού αποθηκεύουμε τη Διεύθυνση Επιστροφής;

- *CISC*: στη μνήμη, στη στοίβα (πιθανόν μαζί και με σωζόμενους καταχωρητές, «γιά υποστήριξη αναδρομής»)
- *RISC*: Διεύθυνση Επιστροφής *σε Καταχωρητή*:
  1. Μόνον οι εντολές store γράφουν στη μνήμη
  2. Ταχύτερο!
     - μόνον όσες διαδικασίες καλούν άλλην χρειαζ. στη στοίβα
     - μεγάλο ποσοστό των «δυναμικά» εκτελούμενων (≠ «στατικά» στο προγρ.) διαδικασιών δεν καλούν άλλη διαδικασία
  3. Σώσιμο άλλων μεταβλητών; ⟶ βλ. §6

Jump (Ψευδοεντολή): ειδική περίπτωση Κλήσης Διαδ.

- `x1` (αλλιώς: `ra`): συνήθης καταχ. γιά διευθ. επιστροφής
- Συνήθης κλήση διαδικασίας:

  `jal x1, Imm20` ⇒ $x1 \leftarrow PC_{old} + 4$,
  $PC_{new} \leftarrow PC_{old} + 2 \times Imm20$

- Ψευδοεντολή: **`j Imm20`** =

  `jal x0, Imm20` ⇒ $PC_{new} \leftarrow PC_{old} + 2 \times Imm20$

- Εγγραφή στον x0 αγνοείται: «Δεν με ενδιαφέρει η διεύθυνση επιστροφής – μην την κρατήσεις»

Άλματα σε μεταβλητές Διευθύνσεις: *Jump Register*

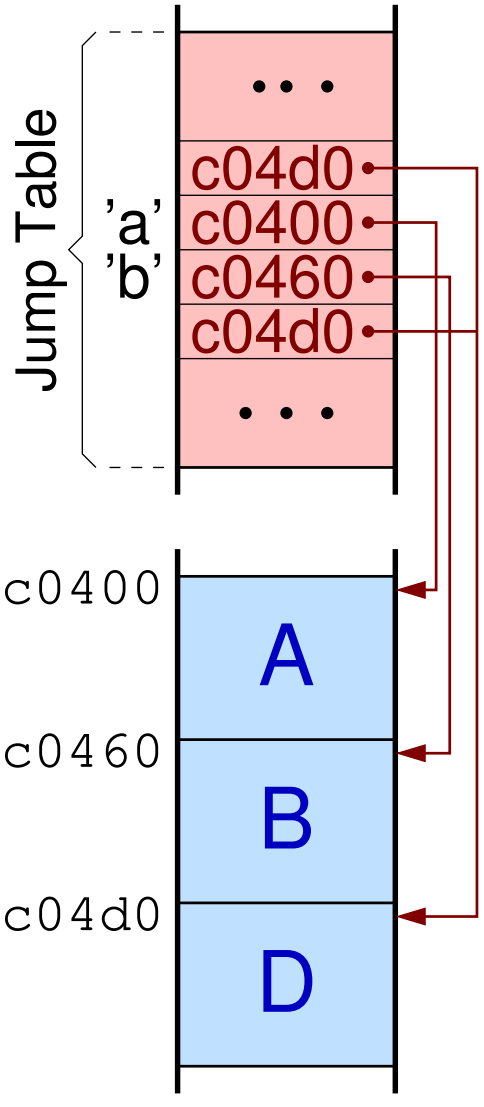
`jr rs1` ⇒ PC ← rs1

- Γιά Επιστροφή από Διαδικασία: `jr x1` ή `jr ra`
  - Μεταβλητή διεύθυνση διότι με καλούν από διάφορα μέρη
- Γιά άλμα σε αυθαίρετη διεύθυνση, οσοδήποτε μακριά
  - σύνθεση αυθαίρετης σταθ. σε καταχ. από προηγ. εντ. – βλ. παρακ.
- Switch statement – multi-way "computed" branch
- Περισσεύουν πολλά bits μέσα στο format της εντολής:
  - *Ψευδοεντολή*, ειδική περίπτωση άλλης γενικότερης…

Switch statement μέσω `jr`

```
char in_c;
switch (in_c) {
    case ’a’:  { A; }
    case ’b’:  { B; }
    default:   { D; }
}
```

```
x5 ← JumpTable[in_c]
jr  x5
```



jr: Ειδική περίπτωση της *Jump-and-Link-Register*

- Μιάς και περισσεύουν bits στην `jr`, γενικεύουμε σε άλλη εντολή:

  `jalr rd, Imm12(rs1)` ⇒

  $rd \leftarrow PC_{old} + 4$; $PC_{new} \leftarrow Imm12 + rs1$

- Και Κλήση Διαδικασιών σε αυθαίρετη/μεταβλητή διεύθυνση – όχι μόνο άλματα σε τέτοιες διευθύνσεις
  - Object-Oriented: type-dependent procedure @variable address
- I-format: διαθέσιμη και η σταθερά Imm12, εάν χρειαστεί
  - Ίδιο addressing mode με *load*: signed Imm12, όχι διπλασιασμός
  - Μαζί με μιά προηγ. εντολή Upper Imm20 συνθέτει αυθαιρ. σταθ.

Dynamic Linking of Library Procedures

- Προβλήματα στατικής συνένωσης βιβλιοθηκών:
  - οι σύγχρονες βιβλιοθήκες (με διαδικασίες) είναι τεράστιες
  - οι βιβλιοθήκες αλλάζουν συχνά – ενημερώσεις, εκσυγχρονισμός
  - εάν όλες οι (εν δυνάμει!) καλούμενες διαδικασίες βιβλιοθήκης έχουν γίνει στατικά linked μέσα στο εκτελέσιμο αρχείο (.out), τότε *(α)* αυτό γίνεται πολύ μεγάλο, και *(β)* απαιτείται re-linking σε κάθε αλλαγή του version μιάς χρησιμοποιούμενης βιβλιοθ.
- Δυναμική συνένωση (dynamic linking) βιβλιοθηκών:
  - Την ώρα που τρέχει το εκτελέσιμο (run-time), εάν και όταν καλέσει γιά πρώτη φορά μιά διαδικασία βιβλιοθήκης, τότε και μόνο τότε συνενώνεται (linked) αυτή με το εκτελέσιμο

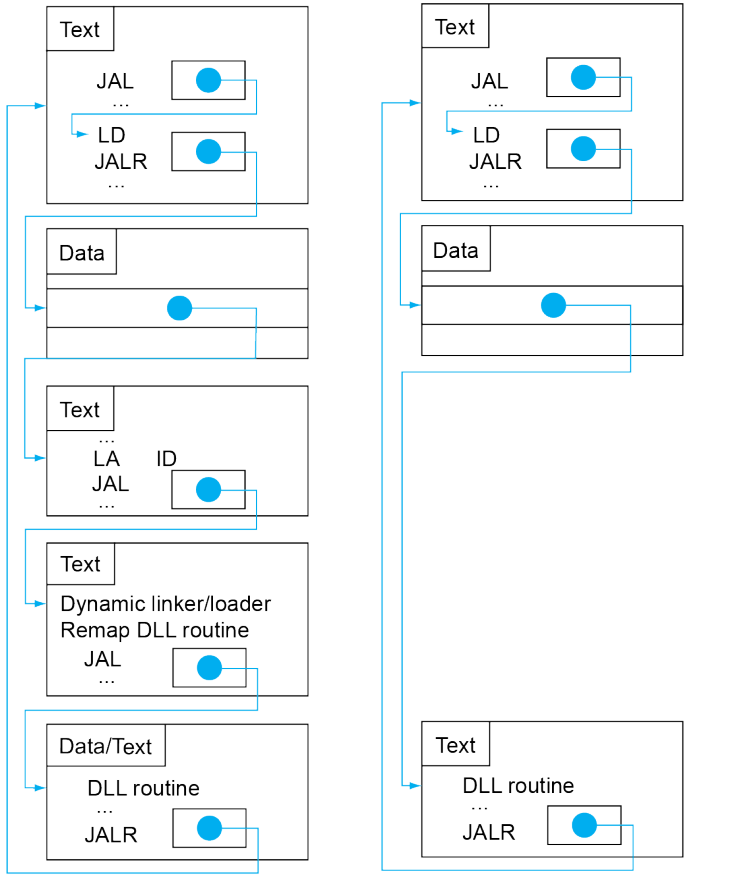
Lazy Linkage

Indirection table

Stub: Loads routine ID, Jump to linker/loader

Linker/loader code

Dynamically mapped code



(a) First call to DLL routine (b) Subsequent calls to DLL routine

Relocatable code, Add Upper Immediate to PC (`auipc`)

- *Relocatable code*:
  κώδικας ("text") (πιθανώς συνοδευόμενος και από στατικά δεδομένα) που είναι εύκολο –ή και που μπορεί αυτόματα χωρίς καμία αλλαγή– να συνενωθεί (linking) με άλλον κώδικα, δηλαδή να φορτωθεί σε νέα, αυθαίρετη θέση μνήμης
- PC-relative addressing mode: το κατάλληλο γιά relocatable
- Εντολή *Add Upper Immediate to PC* (**`auipc`**) (U-format):
  - Κατασκευάζει την ίδια σταθερά όπως και η lui, προσθέτει τον PC σε αυτήν, και γράφει το αποτέλεσμα στον rd
  - Σκοπός: relocatable code – βλ. επόμενη διαφάνεια

Κλήση/Άλμα/Load/Store PC-relative & οσοδήποτε μακριά

- Κάλεσμα PC-relative, σε αυθαίρετη απόσταση:

```
auipc t0, HI(ή HI+1); jalr ra, LO(t0)
```

- Άλμα PC-relative, σε αυθαίρετη απόσταση:

```
auipc t0, HI(ή HI+1); jalr x0, LO(t0)
```

- Load PC-relative, από αυθαίρετη απόσταση:

```
auipc t0, HI(ή HI+1);  ld rd,  LO(t0)
```

- Store PC-relative, σε αυθαίρετη απόσταση:

```
auipc t0, HI(ή HI+1);  sd rs2, LO(t0)
```

Διακλαδώσεις οσοδήποτε μακριά (εάν χρειαστεί)

- if (i ≠ j) goto *farAway*; /* σπάνιο, αλλά εάν χρειαστεί */
  continue…

```
      beq i, j, cnt          # "else" continue…
      auipc t0, Imm20        # upper 20 bits of offset
      jalr x0, Imm12(t0)     # low 12 bits of farAway
cnt:  continue…
```

Πράξεις Σύγκρισης με αποτέλεσμα Boolean

- `slt  rd, rs1, rs2` # rd ← (rs1<rs2) – sign'd
- `sltu  rd, rs1, rs2` # rd ← (rs1<rs2) – uns.
- `slti  rd, rs1, Imm12` # rd ← (rs1<Imm12) s.
- `sltiu rd, rs1, Imm12` # rd ← (rs1<Imm12) uns.
- "Set-if-less-than": πράξη ALU, όχι διακλάδωση
- Το αποτέλεσμα είναι Boolean:
  - True = 000…0001
  - False = 000…0000
- Γιατί μόνον αυτές; – βλ. άσκηση 5.11
  - Ρεπερτόριο εντολών: Δομικοί λίθοι γιά σύνθεση πιό πολύπλ.

Το `Imm12` είναι πάντα signed, ακόμα και στην `sltiu`