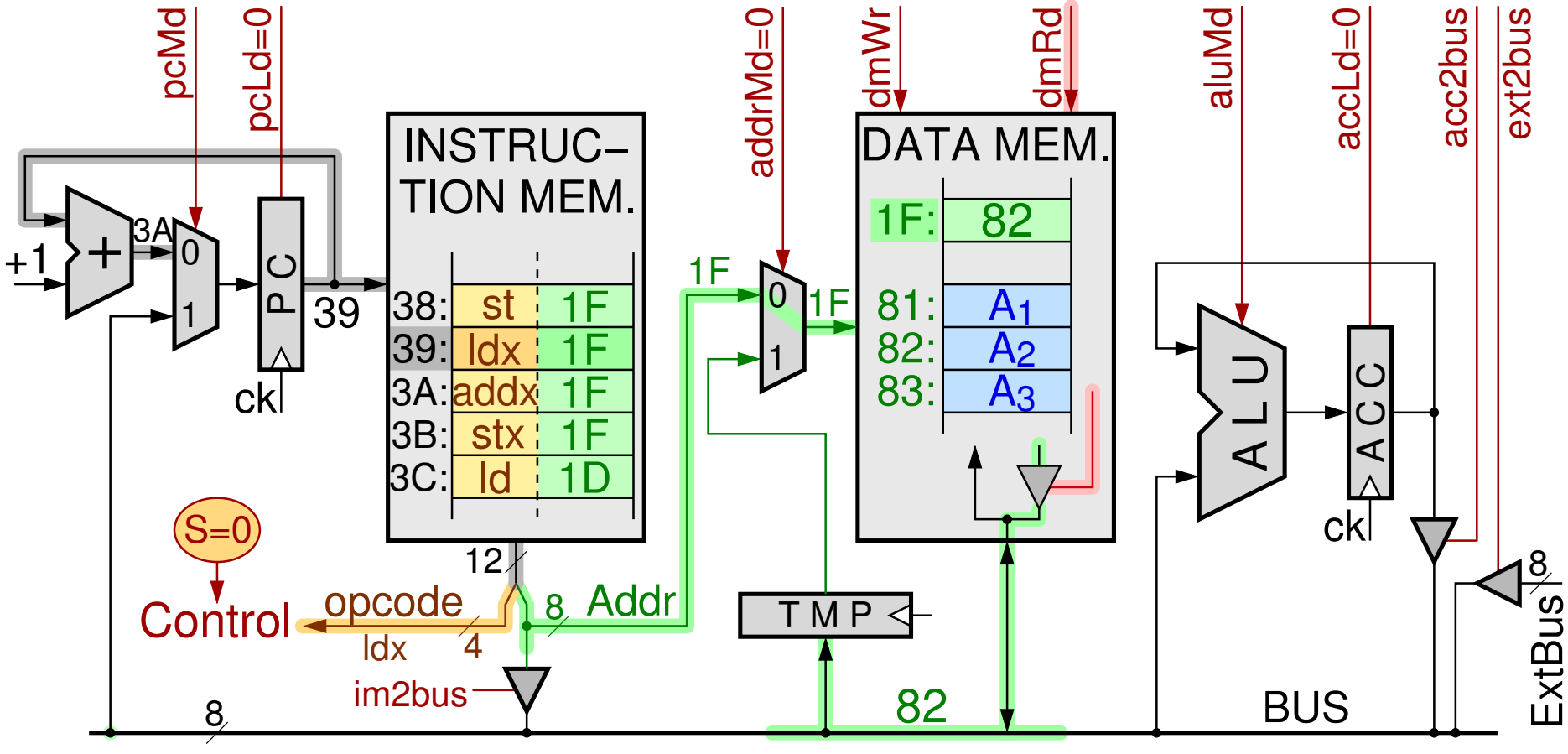
Καταχωρητές Γενικού Σκοπού, Αρχιτεκτονικές RISC και το Ρεπερτόριο RISC-V, Γλώσσα Assembly και ο Προσομοιωτής RARS

01a (§1) – 14-18 Φεβ. 2022 – Μανόλης Κατεβαίνης

Επανάληψη: Υπολογιστές, Επεξεργαστής, Εντολές

- Υπολογιστής: Επεξεργαστής, Μνήμη, Περιφερειακά
- Επεξεργαστής:
  - PC (Program Counter) → Instruction fetch (read from memory)
  - Εντολή (Instruction): Opcode (τι), Operands (σε ποιούς)
  - Read source Operands: από εντολή, ή καταχωρητές, ή μνήμη
    - σταθερός αρ., ή βαθμωτή μεταβλ., ή στοιχείο δομής μέσω pointer
  - Operate: ALU (Arithmetic/Logic Unit)
  - Write destination Operand: σε μνήμη/καταχωρητή
  - Update PC: καθορισμός «διαδόχου» – είτε η «από κάτω», ή κάποια άλλη εντολή → Διακλαδώσεις υπό συνθήκη, άλματα
  - Ξανά από την αρχή γιά την επόμενη εντολή

Επανάληψη: ο απλός επεξεργαστής του HY-120



Μεγάλες Ιδέες στην Αρχιτεκτονική Υπολογιστών

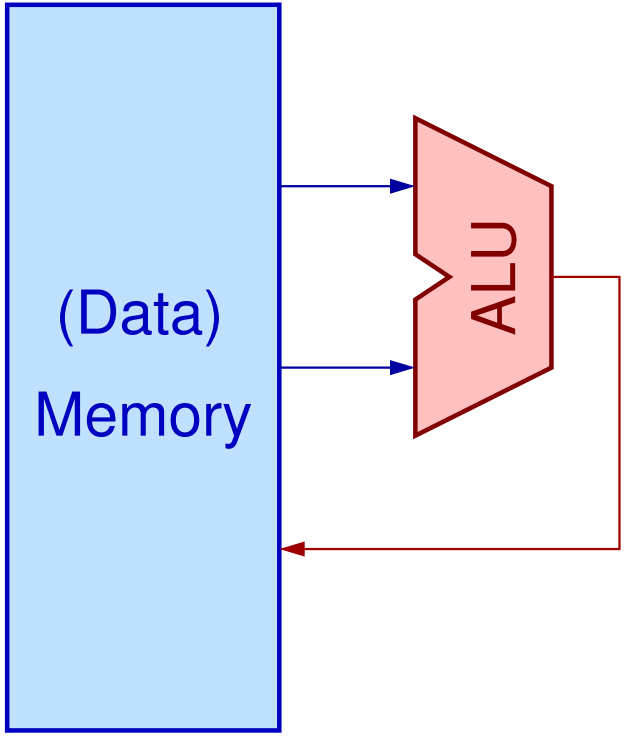
Όσο πιό μικρό και απλό ένα κύκλωμα, τόσο πιό γρήγορο:

- Μικρές / τοπικές μνήμες: γρήγορες
- Μεγάλες / απομακρυσμένες μνήμες: αργές
- Ομοιομορφία & απλότητα: γρήγορα κυκλώματα

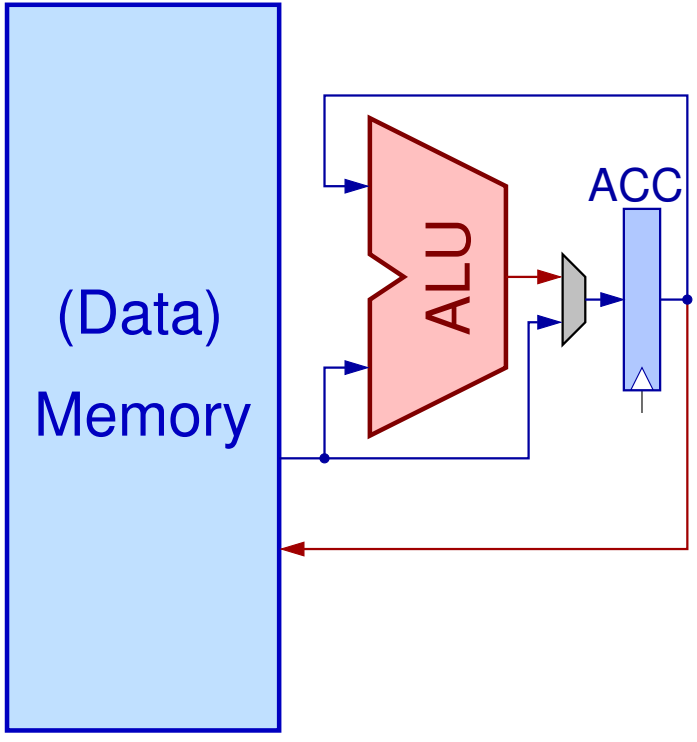
Παραλληλισμός:

- Pipelining (ομοχειρία): γραμμή συναρμολόγησης
- Multiple issue (superscalar): πολλαπλές εντολές ταυτοχρ.
- Multicore (πολυπύρηνοι): παράλληλα προγράμματα

Συνήθως 3 τελεστέοι ανά πράξη: Πού είναι αυτοί;

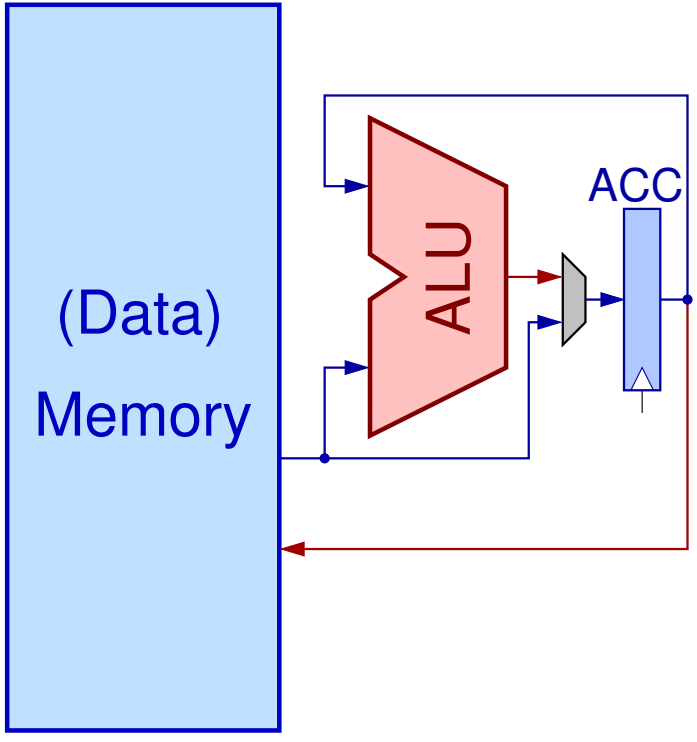


Εν γένει, στη μνήμη
(συνήθως μονόπορτη)

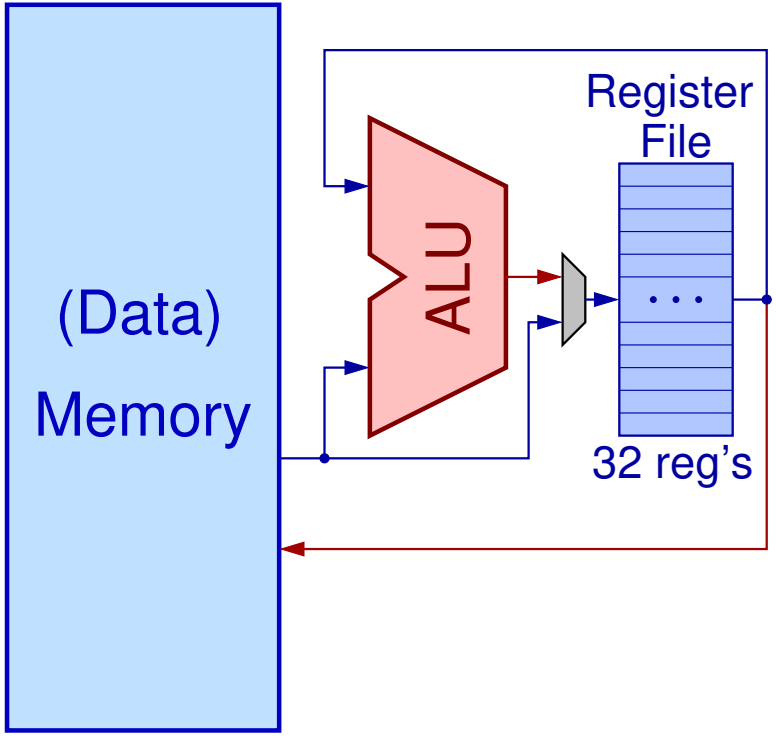


Στο HY-120, γιά απλότητα:
ένας βοηθητικός καταχωρητής

Ένας καταχωρητής είναι πολύ λίγος: Θέλουμε 32



Με έναν μόνο «συσσωρευτή» πρέπει πολύ συχνά να τον σώζουμε στη μνήμη



32 καταχωρητές είναι καλή ισορροπία μεταξύ μεγέθ.-ταχύτητας & πλήθους προσωρ. βαθμ. μεταβλ.

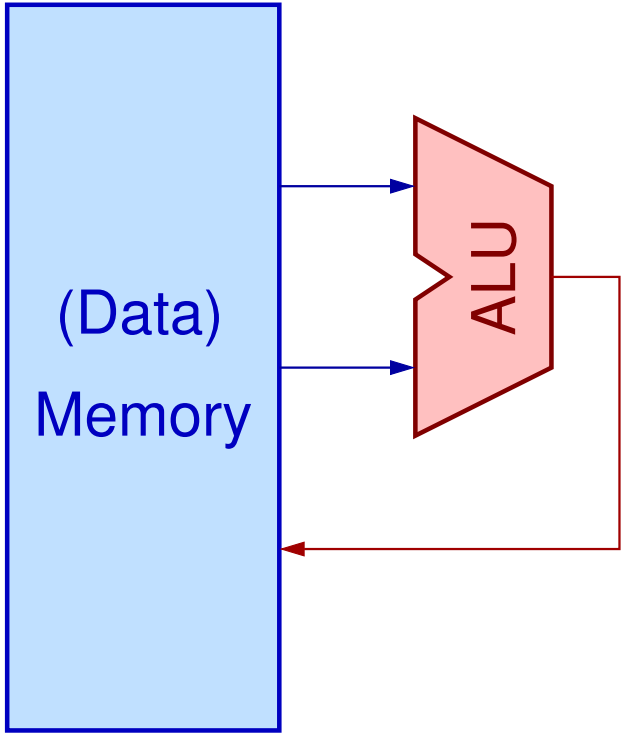
Πλεονεκτήματα Καταχωρητών έναντι Μνήμης

- Πολύ-πολύ λιγότεροι από (δισεκατομ.) λέξεις Μνήμης
  ⇒ πολύ-πολύ ταχύτεροι από «κεντρική» μνήμη
- Λιγότεροι από (χιλιάδες) λέξεις «Κρυφής Μνήμης»
  ⇒ ταχύτεροι/πολύπορτοι από (μονόπορτη) «κρυφή μνήμη»
  - «Κρυφή Μνήμη» = μικρή άρα γρήγορη μνήμη που κρατά τα συχνότερον χρησιμοποιούμενα δεδομένα της «κεντρικής» μνήμης
- Μόνον 5 bits διεύθυνσης, έναντι 32 ή 64 γιά Μνήμη
  ⇒ χωράνε >1 διευθύνσεις καταχωρητών σε 32-μπιτη εντολή
- Απλή διευθυνσιοδότηση, χωρίς αριθμητική
  – σταθερή διεύθ. καταχωρητή ⇒ βαθμωτές μεταβλητές μόνον

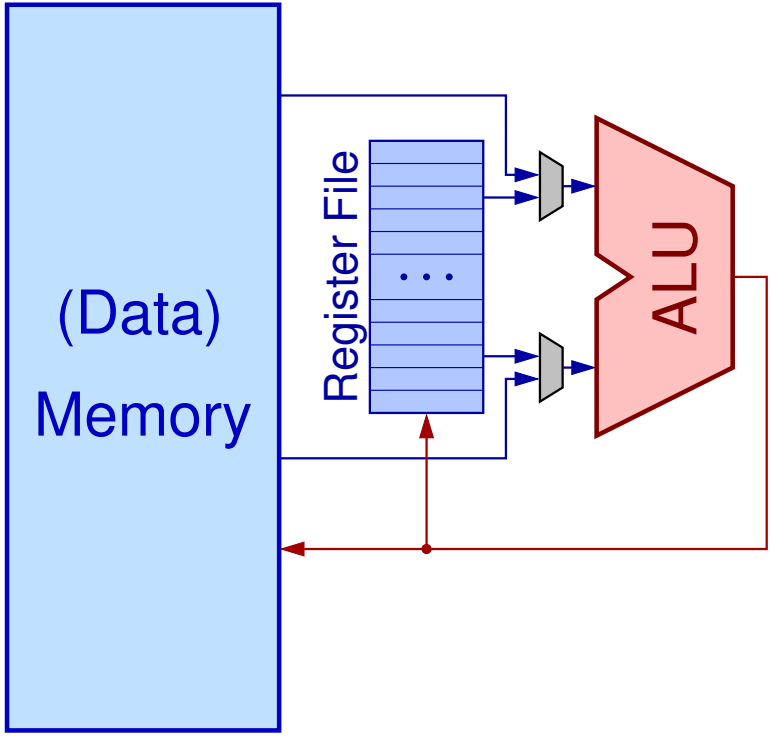
Περιορισμοί Καταχωρητών έναντι (κρυφής) Μνήμης

- «Ορατοί» από τον προγραμματιστή γλώσσας μηχανής
  ⇒ ο Μεταφραστής (παλαιότερα: ο προγραμματιστής) πρέπει να επιλέγει ποιές μεταβλητές θα μεταφέρει εκεί και πότε
  - η «κρυφή μνήμη», που είναι και αυτή πολύ ταχύτερη από την «κεντρική», είναι *αόρατη* γιά το λογισμικό – την διαχειρίζεται το hardware, δηλ. επιλέγει αυτόματα τι θα μεταφέρει εκεί και πότε
- Μόνον Βαθμωτές Μεταβλητές στους καταχωρητές
  - πολύ λίγοι ⇒ δεν χωρούν δομές δεδομένων εκεί
  - μόνον προσωρινό αντίγραφο ενός στοιχείου δομής κάθε φορά
  - σταθερή διεύθυνση («αριθμός») καταχωρητή μέσα στην εντολή ⇒ ο ίδιος καταχωρητής στην κάθε ανακύκλωση βρόχου

Συνήθως 3 τελεστέοι ανά πράξη: Πού είναι αυτοί;

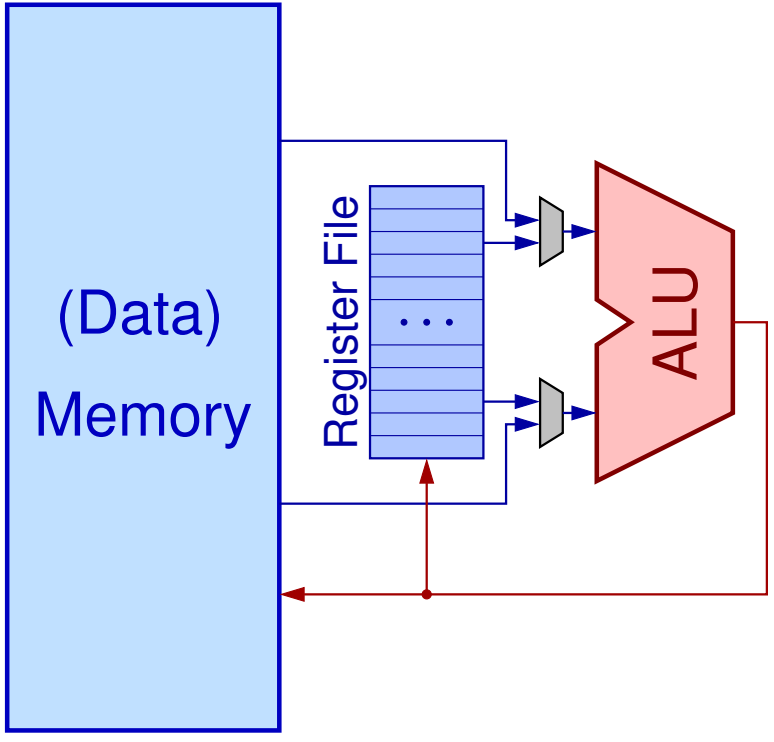


Αν δεν είχαμε καταχωρητές…

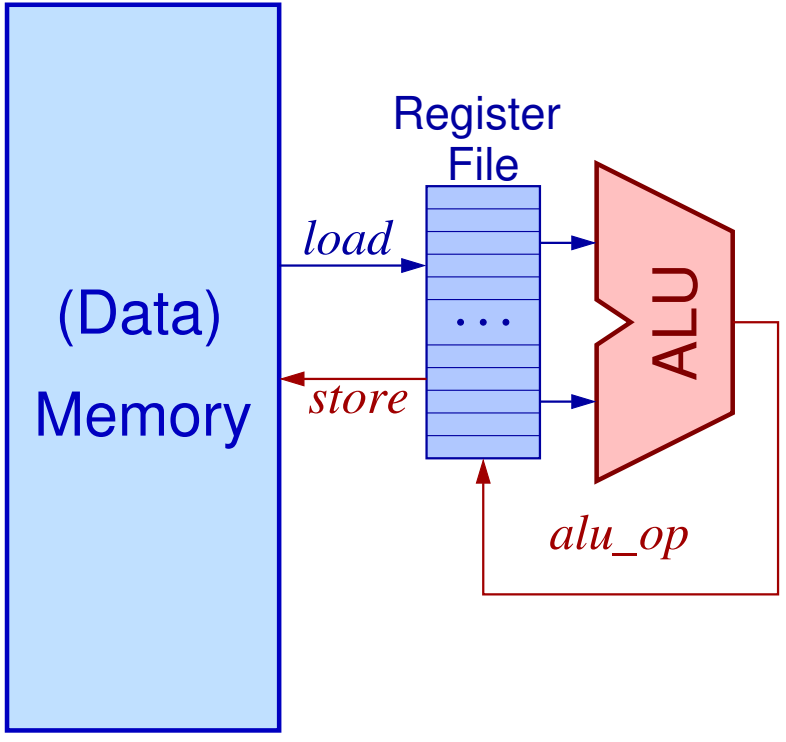


Τώρα που έχουμε και καταχωρητές;;

Τελεστέοι: Οπουδήποτε ή σε Καταχωρητές μόνον;

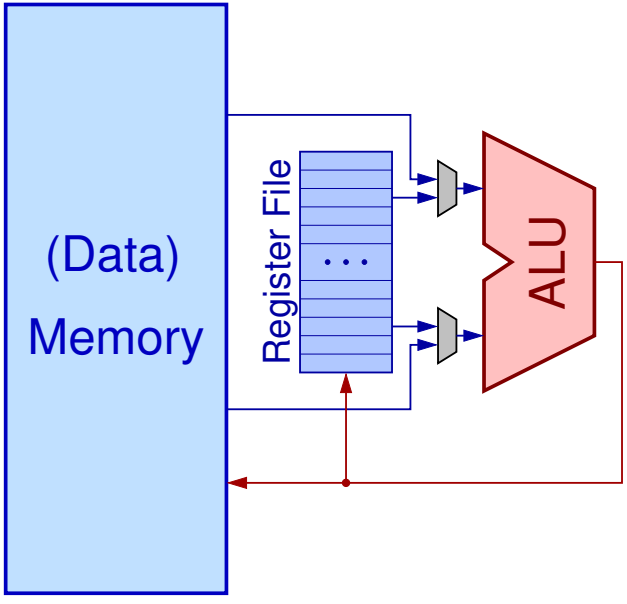


Οι τελεστέοι οπουδήποτε:
"**CISC**"
(Complex Instr. Set Computer)



Οι τελεστέοι μόνο σε καταχωρητές:
"**RISC**"
(Reduced Instr. Set Computer)

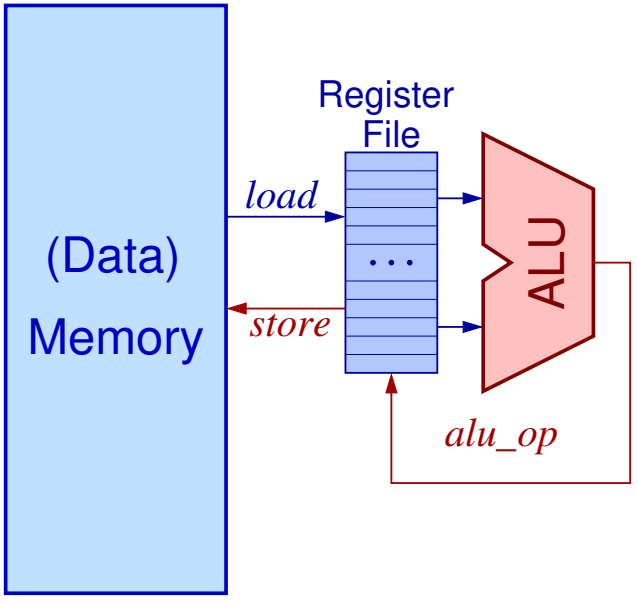
CISC: Παλαιότερα Ρεπερτόρια Εντ.



- Τάση των 70's έως mid-80's
  - Αποκορύφωμα ο VAX της DEC
  - Τότε βγήκε και ο 8086 της Intel
  - Το Ρεπερτόριο x86 τον ακολουθεί λόγω *backwards compatibility*… ☹
  - Υπήρχε η εντύπωση (& marketing) ότι ο,τιδήποτε εκτελείται «σε μία μόνον εντολή» είναι και γρήγορο
  - VAX: 1 εντολή υπολογ. τιμής πολυωνύμου (10-100'δες κύκλοι)
- 1980 – 84: η έρευνα απέδειξε ότι δεν είναι έτσι
  - Πολύπλοκο ρεπερτόριο ⇒ πολύπλ. κύκλωμα ⇒ αργό κύκλωμα
  - «Μία εντολή», αλλά πόσοι κύκλοι, πόσο αργού ρολογιού;

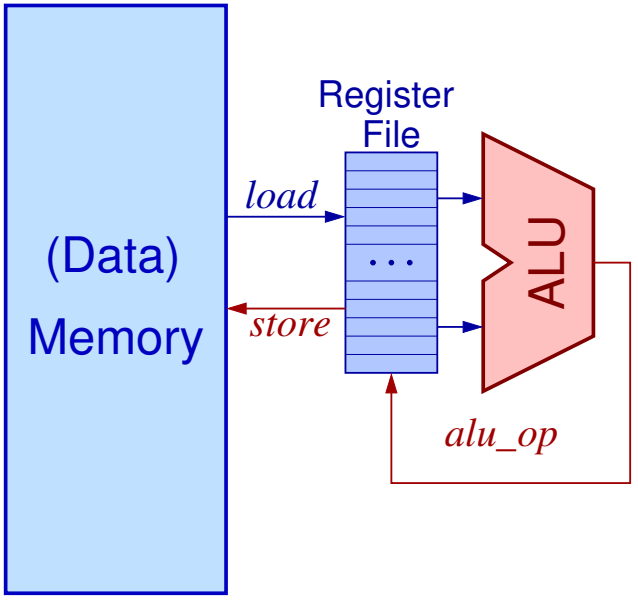
RISC: Νεότερα Ρεπερτ. Εντολών

- Απλότητα και Ομοιομορφία
  - ⇒ Απλούστερο κύκλωμα, μικρότεροι πολυπλέκτες, ευκολότερο pipelining (§9)
  - ⇒ Ταχύτερο Ρολόϊ
- Επιτάχυνση των συχνών περιπτωσ.
- Pipelining με Προώθηση αποτελ.:
  - Πολύπλοκες εργασίες αποτελούμενες από σειριακά βήματα συντίθενται εξ ίσου γρήγορα μέσω πολλαπλών απλών εντολών
- Superscalar (multiple-issue) processors (§14)
  - Πολύπλοκες εργασίες αποτελούμενες από παράλληλα βήματα εξ ίσου γρήγορα μέσω πολλαπλών εντολών εν παραλλήλω



RISC vs. CISC: το τοπίο 1990-2020

- Intel/x86 δεσπόζει σε Servers
  - προσπάθησε και απλό ρεπερτ. (IA64)
  - η κληρονομιά x86 την βαραίνει πάντα
- Η ARM δεσπόζει σε Mobile & Emb.
  - ARM = "Advanced RISC Machine"
  - Αγγλική, την αγόρασε η Ιαπων. SoftBank (& brexit), μετά προσπ. αγόρ. η Nvidia



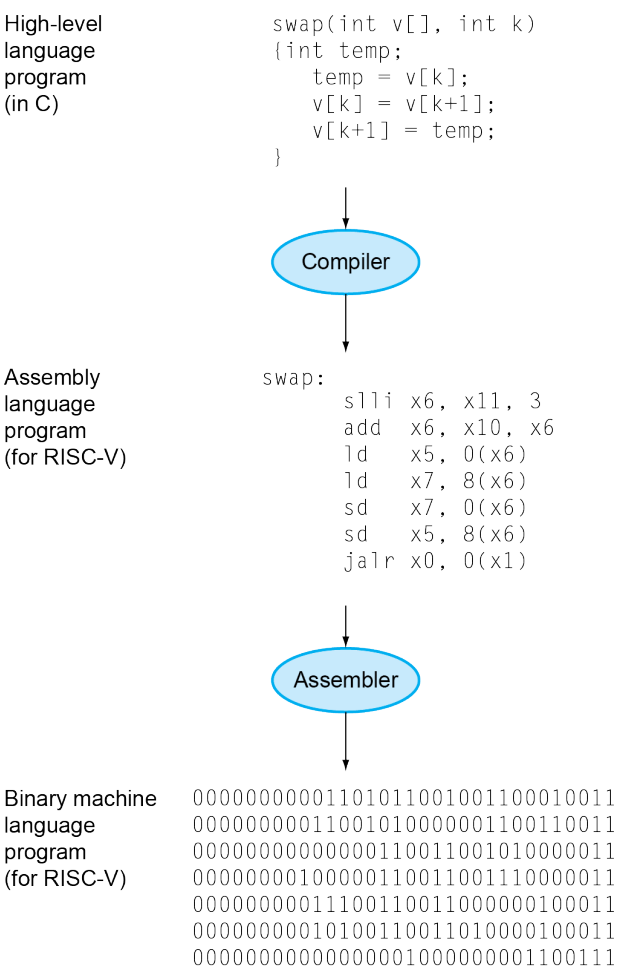
- RISC-V: Ανοικτό Ρεπερτόριο Εντολών (και RISC-style)
  - Από την παράδοση των RISC-I & RISC-II του U.C.Berkeley (1981-83) υπό τον Patterson, & MIPS από Stanford υπό τον Hennessy
  - Αύξουσα δημοτικότητα, Ανοικτό, τώρα εδρεύει στην Ελβετία
- Άλλοι RISC: MIPS, PowerPC, και ιστορικά: SPARC, Alpha

Τεχνολογ. Ανεξαρτησία & ο Ευρωπαϊκός Επεξεργαστής

- Οι επεξεργαστές είναι πλέον παντού, περιλαμβανομένων κρίσιμων υποδομών και άμυνας
  - θέματα ασφάλειας κεφαλαιώδους σημασίας (βλ. & backdoors)
  - η Ευρώπη επιδιώκει ανεξαρτησία και αυτάρκεια έναντι τρίτων
- https://www.european-processor-initiative.eu/
  - Αρχικά βασισμένος σε ARM, λόγω χαμηλής ενεργειακής κατανάλωσης και τέως Ευρωπαϊκού background
  - Αλλά περιλαμβάνει και υποσύνολο RISC-V αύξουσας σημασίας
  - Κρήτη (ΙΤΕ-ΙΠ-CARV): 7ος σε μέγεθος partner μεταξύ 28 (RISC-V Cache υπό Β. Παπαευσταθίου, Systems S/W υπό Μ. Μαραζάκη)
  - Νέο project “eProcessor” (high-end RISC-V): 2ος σε μεγ. partner

Levels of Program Code

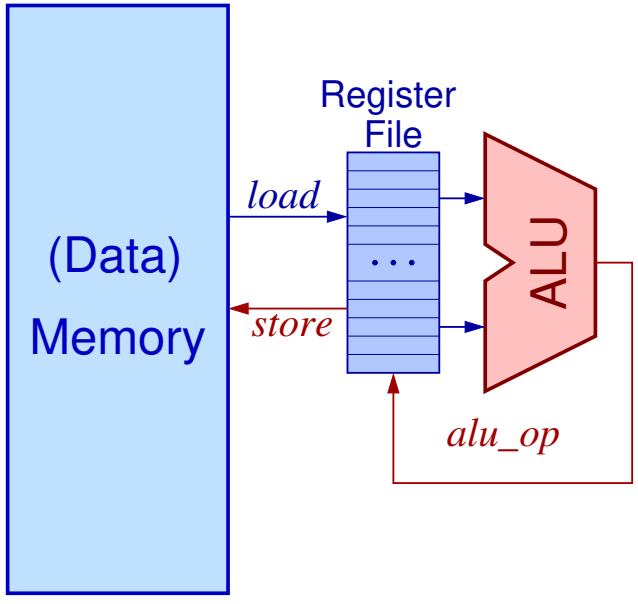
- High-level language
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- Assembly language
  - Textual representation of instructions
- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data



ISA - Instruction Set Architecture

- **ISA**: Ρεπερτόριο Εντολών μιάς *Οικογένειας* επεξεργαστών
  - η διεπαφή (interface) μεταξύ Λογισμικού και Υλικού
  - ό,τι πρέπει να ξέρει ο Compiler/Assembler γιά το hardware
- Οικογένειες επεξεργαστών "*Binary Compatible*" (ίδιο ISA)
  - διαφορετικές υλοποιήσεις επεξεργαστών (κόστος, ταχύτητα, …) που όλες «τρέχουν» τα ίδια εκτελέσιμα αρχεία ("object code")
- Επεκτάσεις Ρεπερτορίων με *Backwards Compatibility*
  - προσθήκη νέων εντολών με διατήρηση και των προηγουμένων
  - η οικογένεια x86 της Intel είναι η πιό φημισμένη τέτοια που διατηρεί ευλαβικά την συμβατότητα με τα παλαιότερα ISA της (η οποία και την βαραίνει σε μεγάλο βαθμό)

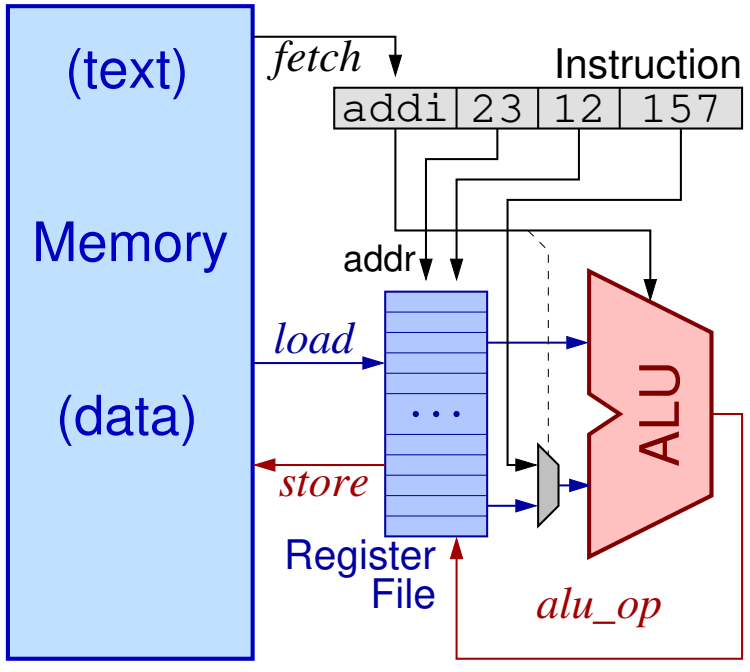
Εντολές reg2reg στον RISC-V



- `add x23, x12, x14`
  - σημαίνει: x23 ← x12 + x14
  - *xNN* σημαίνει «το περιεχόμενο του καταχωρητή υπ' αριθμ. *NN*»
  - καταχωρητές: x0, x1,... εώς και x31
  - ο x0 είναι «ειδικός»: x0 ≡ 0 πάντα!
  - επιτρέπονται εγγγραφές στον x0, αλλά αγνοούνται!
  - βλ. σχόλια και εξηγήσεις «γιατί» παρακάτω
- `sub x23, x12, x14` σημαίνει: x23 ← x12 - x14
- ομοίως: `and x23, x12, x14` `or` ..., `xor` ..., `mul` ...

Σταθεροί αριθμοί στον RISC-V

- `addi x23, x12, 157`
  - "add immediate (constant)"
  - σημαίνει: x23 ← x12 + 157
  - `addi x23, x12, 14`
    προσθέτει τον αριθμό 14, *όχι* το περιεχόμενο του καταχωρητή x14, στο περιεχόμενο του καταχωρ. x12
  - οι σταθερές είναι πάντα προσημασμένες στον RISC-V



- *Δεν* υπάρχει "subi x23, x12, 14" ⇔ addi x23, x12, -14
- *Δεν* υπάρχει "subii x23, 157, x14": πολύ σπάνια, δεν αξίζει

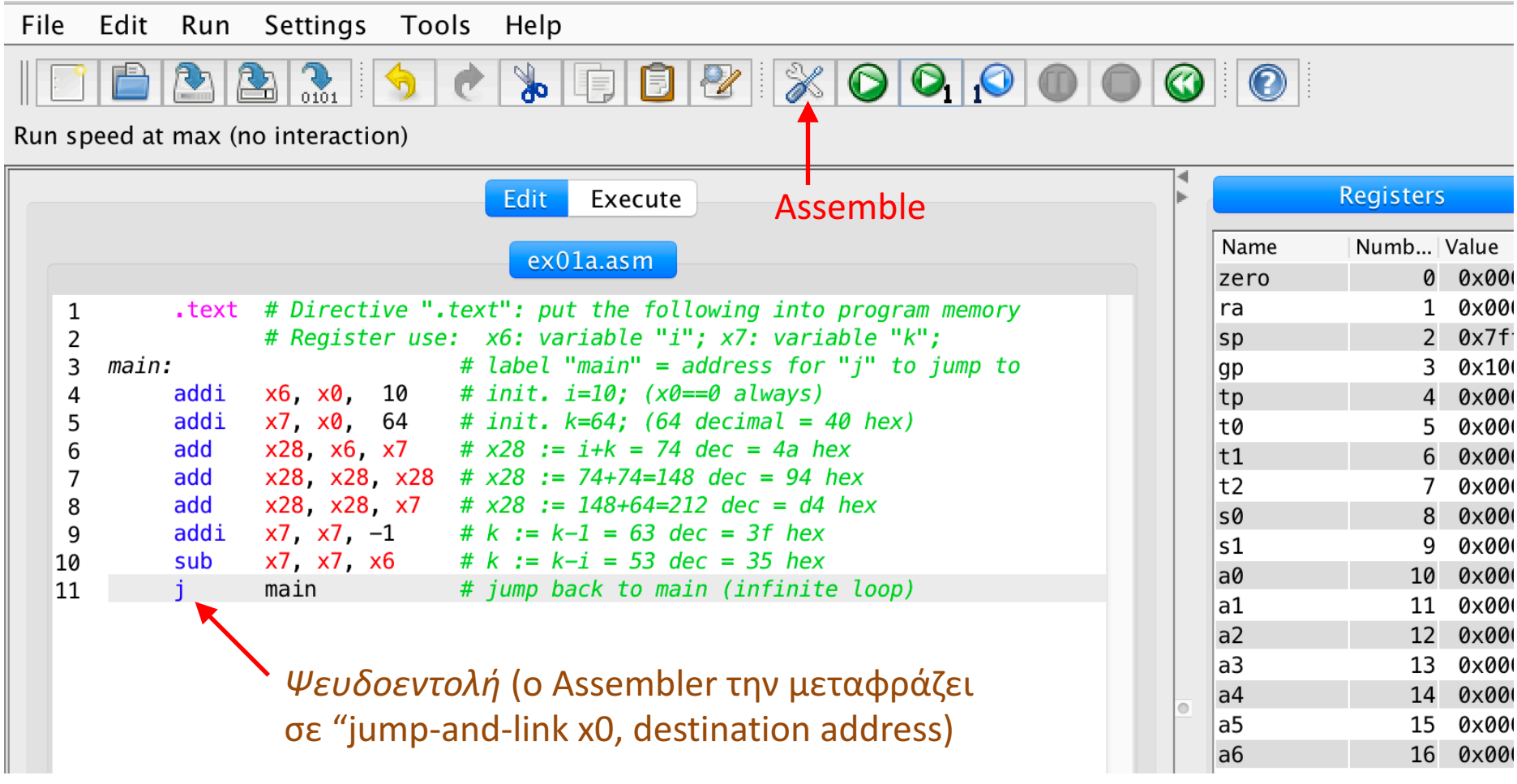
Σύνθεση άλλων τύπων εντολών σε RISC-style ISA's

- add x8, x0, x0 # i=0; αρχικοποίηση (μτβλ. i στον x8)
- addi x8, x0, 0 # i=0; εναλλακτική αρχικοποίηση
- addi x8, x0, 1 # i=1; αρχικοπ. σε μη μηδενική τιμή
- add x8, x9, x0 # i=j; αντιγραφή μεταβλητών (j στον x9)
- addi x8, x9, 0 # i=j; εναλλακτική αντιγραφή
- addi x8, x8, 1 # i=i+1;
- add x8, x8, x9 # i=i+j; "two-operand add"
- sub x8, x0, x8 # i=-i; αλγεβρικό αντίθετο

Γιατί «σπαταλάμε» μία πρόσθεση όταν δεν χρειάζεται;

- Αρχικοποιήσεις και αντιγραφές μεταβλητών κάνουν μία πρόσθεση που κανονικά δεν χρειάζεται – καθυστέρηση;;
  - addi x8, x0, 13 # i=13; add x8, x9, x0 # i=j;
- Θα διαπιστώσουμε, όταν δούμε την υλοποίηση με pipelining ότι η πρόσθεση κοστίζει έναν μόνο κύκλο, τον ίδιο που χρειάζεται ούτως ή άλλως και γιά το fetch, και *δεν* προκαλεί καθυστέρηση χάρις στην «προώθηση»
  - άλλες εργασίες εν παραλλήλω μέσω “multiple issue”

RARS: RISC-V Assembler & Simulator



Ψευδοεντολή (ο Assembler την μεταφράζει σε “jump-and-link x0, destination address)

Το παράδειγμα προγράμματος Ασκ. 1 στον RARS

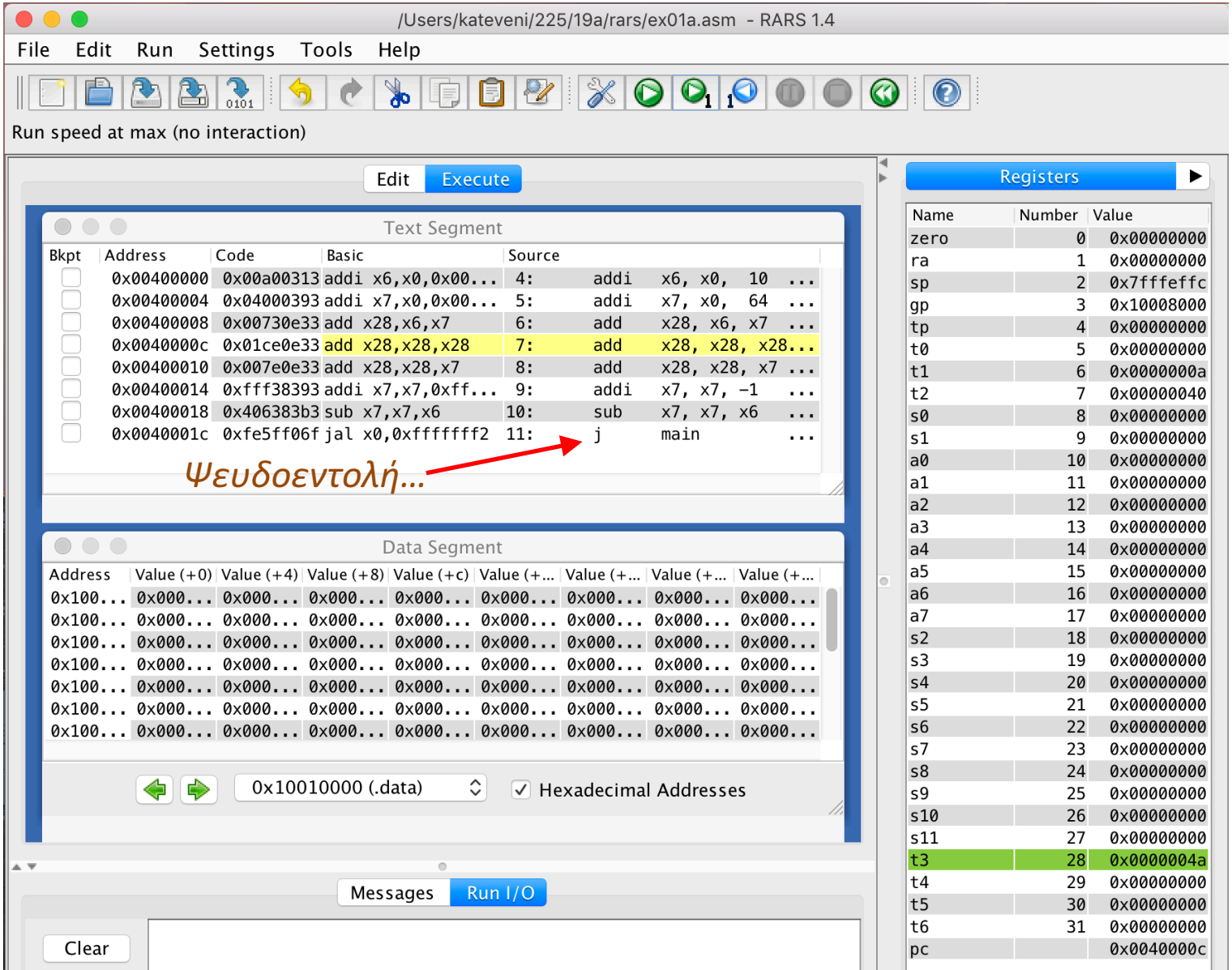
```
1        .text  # Directive ".text": put the following into program memory
2               # Register use:  x6: variable "i"; x7: variable "k";
3   main:                       # label "main" = address for "j" to jump to
4        addi   x6, x0,  10     # init. i=10; (x0==0 always)
5        addi   x7, x0,  64     # init. k=64; (64 decimal = 40 hex)
6        add    x28, x6, x7     # x28 := i+k = 74 dec = 4a hex
7        add    x28, x28, x28   # x28 := 74+74=148 dec = 94 hex
8        add    x28, x28, x7    # x28 := 148+64=212 dec = d4 hex
9        addi   x7, x7, -1      # k := k-1 = 63 dec = 3f hex
10       sub    x7, x7, x6      # k := k-i = 53 dec = 35 hex
11       j      main            # jump back to main (infinite loop)
```

- Assembler Directives: αρχίζουν με “.”, απευθύνονται στον Assembler

  `.text` # “Text Segment” = portion of address space for program (instr.)

  `.data` # “Data Segment” = portion for data – “#”: start of comments

- *Label:* left-justified, with “:” → def. symbolic name for address here


/Users/kateveni/225/19a/rars/ex01a.asm - RARS 1.4
File Edit Run Settings Tools Help
Run speed at max (no interaction)
Edit Execute
Text Segment
Bkpt Address Code Basic Source
0x00400000 0x00a00313 addi x6,x0,0x00... 4: addi x6, x0, 10 ...
0x00400004 0x04000393 addi x7,x0,0x00... 5: addi x7, x0, 64 ...
0x00400008 0x00730e33 add x28,x6,x7 6: add x28, x6, x7 ...
0x0040000c 0x01ce0e33 add x28,x28,x28 7: add x28, x28, x28...
0x00400010 0x007e0e33 add x28,x28,x7 8: add x28, x28, x7 ...
0x00400014 0xfff38393 addi x7,x7,0xff... 9: addi x7, x7, -1 ...
0x00400018 0x406383b3 sub x7,x7,x6 10: sub x7, x7, x6 ...
0x0040001c 0xfe5ff06f jal x0,0xfffffff2 11: j main ...
Ψευδοεντολή...
Data Segment
Address Value (+0) Value (+4) Value (+8) Value (+c) Value (+... Value (+... Value (+... Value (+...
0x100... 0x000... 0x000... 0x000... 0x000... 0x000... 0x000... 0x000... 0x000...
0x10010000 (.data)
Hexadecimal Addresses
Messages Run I/O
Clear
Registers
Name Number Value
zero 0 0x00000000
ra 1 0x00000000
sp 2 0x7fffeffc
gp 3 0x10008000
tp 4 0x00000000
t0 5 0x00000000
t1 6 0x0000000a
t2 7 0x00000040
s0 8 0x00000000
s1 9 0x00000000
a0 10 0x00000000
a1 11 0x00000000
a2 12 0x00000000
a3 13 0x00000000
a4 14 0x00000000
a5 15 0x00000000
a6 16 0x00000000
a7 17 0x00000000
s2 18 0x00000000
s3 19 0x00000000
s4 20 0x00000000
s5 21 0x00000000
s6 22 0x00000000
s7 23 0x00000000
s8 24 0x00000000
s9 25 0x00000000
s10 26 0x00000000
s11 27 0x00000000
t3 28 0x0000004a
t4 29 0x00000000
t5 30 0x00000000
t6 31 0x00000000
pc 0x0040000c