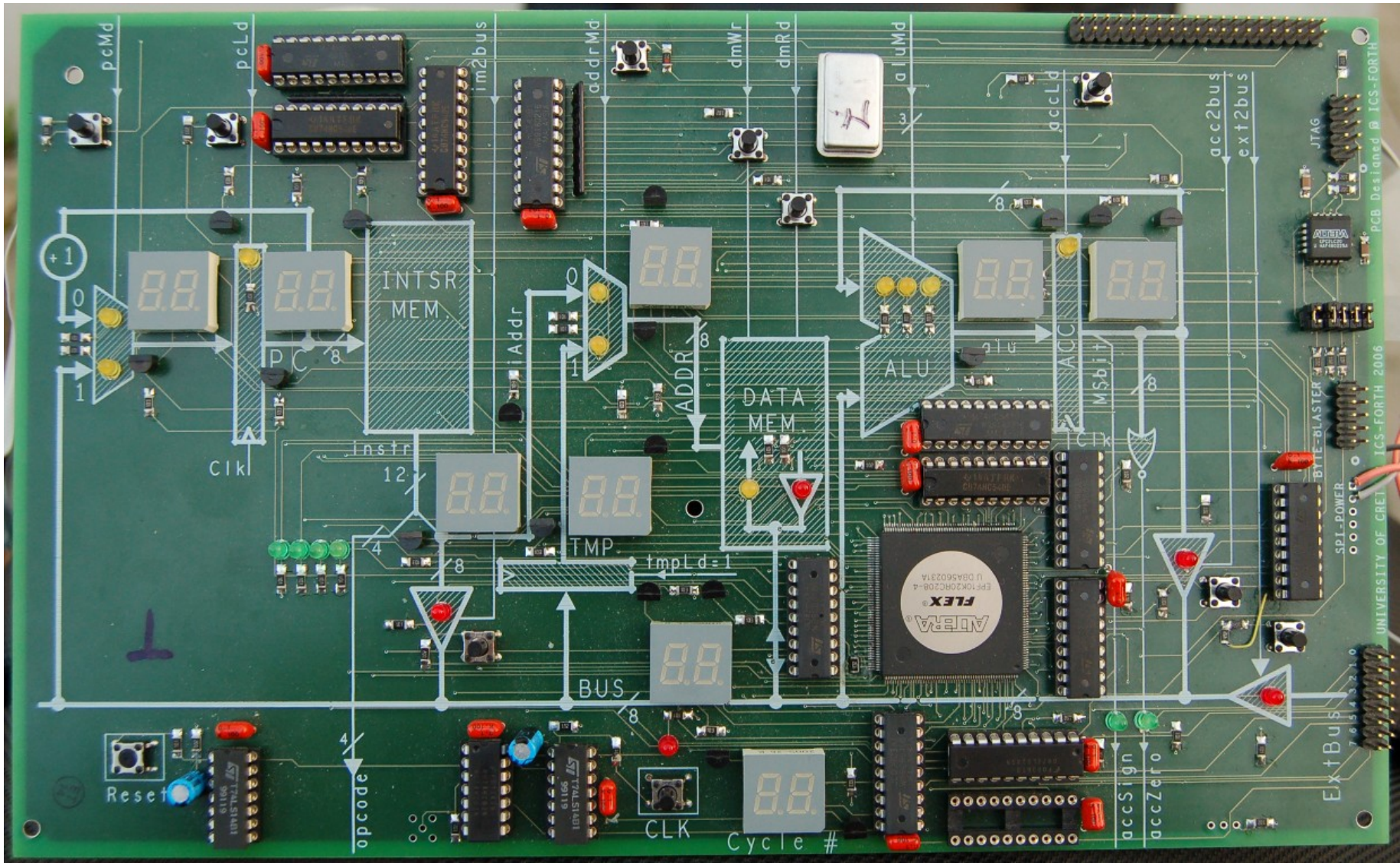


Ένας απλός Υπολογιστής: Datath & Εντολές Πράξεων

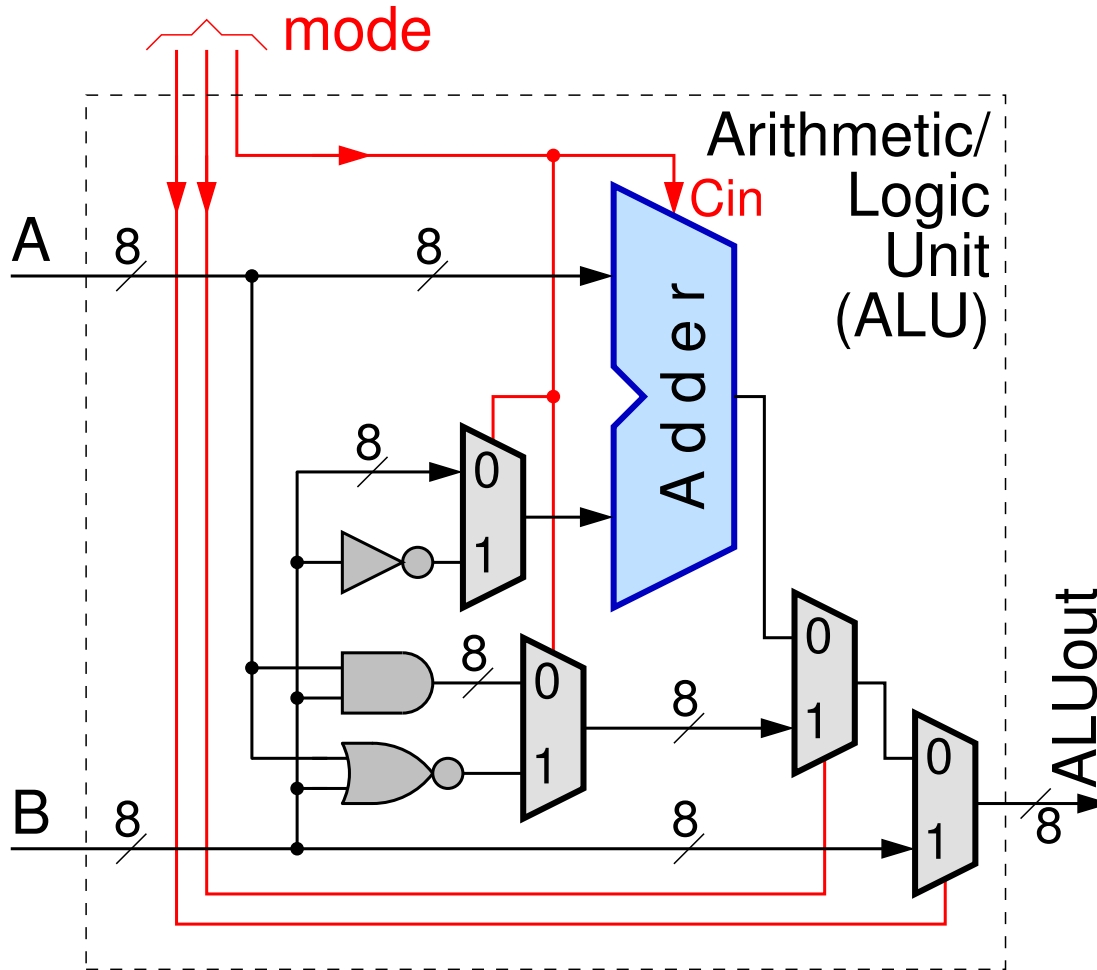
11α (§11.1 - 11.11) – 4-9 Δεκ. 2020 – Μανόλης Κατεβαίνης

Έναν Απλό Υπολογιστή με αυτά που μάθαμε μόνον!

- Απλός: πολύ αργός
- Υπολογιστής:
 - μπορεί να εκτελεί (περίπου) όλα τα προγράμματα
 - Όλες τις δομές δεδομένων
 - Κλήση διαδικασιών & επιστροφή
 - Είσοδος/Εξοδος (I/O – Input/Output): εξαιρετικά υποτυπώδης
 - Πολλαπλασιασμός, διαίρεση, αρ. κινητής υποδ.: in software
 - Χωρίς πολυπρογραμματισμό / προστατευμένες διεργασίες
- Με αυτά που μάθαμε μόνον: βασικά, όλοι οι υπολογιστές αποτελούνται από μνήμες, πολυπλέκτες, και αθροιστές

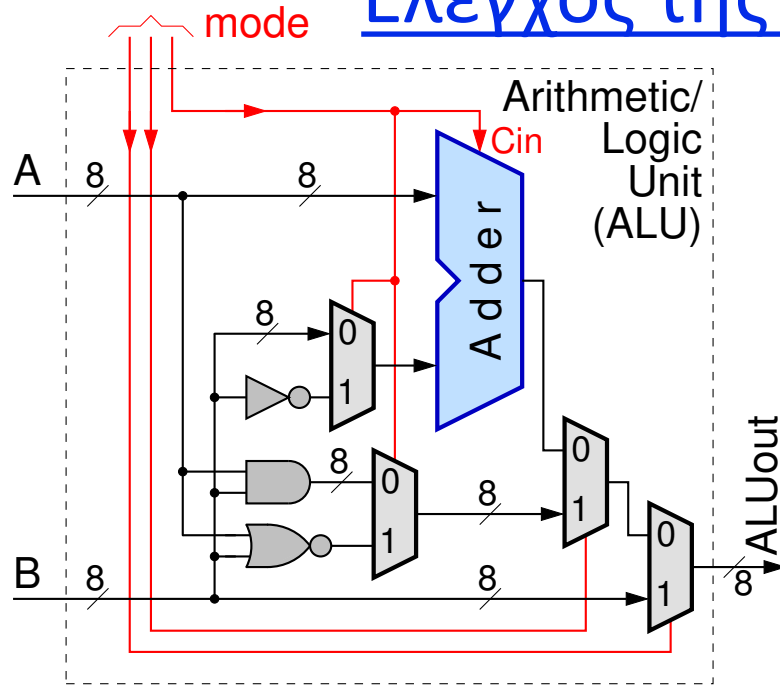


Κάπου οι αρ. πράξεις: Αριθμητική/Λογική Μονάδα



- Οκτάμπιτος μόνον υπολογ., για εμάς
- Πρόσθεση-Αφαίρεση προσημασμένων ακεραίων, όπως την μάθαμε (Εργ. 6)
- Bitwise AND, NOR
- Λειτουργία “PassB”
– θα δούμε γιατί

Έλεγχος της ALU, και επιλογή Πράξεων



mode	Πράξη	Όνομα
000	$ALU_{out} = A+B$	add
001	$ALU_{out} = A-B$	sub
010	$ALU_{out} = A \text{ AND } B$	and
011	$ALU_{out} = \text{NOT}(A \text{ OR } B)$	nor
1xx	$ALU_{out} = B$	passB

- Περιορισμένο πλήθος εντολών (16 μόνον), για απλότητα – έτσι:
- Δεν χωρούσαν πάνω από 4 εντολές πράξεων, άρα:
- Αντί για AND, OR, NOT: μόνον AND και NOR
- **NOT (A) = A NOR 0; A OR B = NOT (A NOR B)**

(To CircuitVerse έχει άλλες Πράξεις & mode encodings)

A and B are two Operands.

SLT (Set Less Than) : If A is less than B Output = 1, Otherwise 0.

Opcode	Opcode Value	Operation
000	0	A & B
001	1	A B
010	2	A + B
100	4	A & ~B
101	5	A ~B
110	6	A - B
111	7	SLT

mode	Πράξη	Όνομα
000	ALUout = A+B	add
001	ALUout = A-B	sub
010	ALUout = A AND B	and
011	ALUout = NOT(A OR B)	nor
1xx	ALUout = B	passB

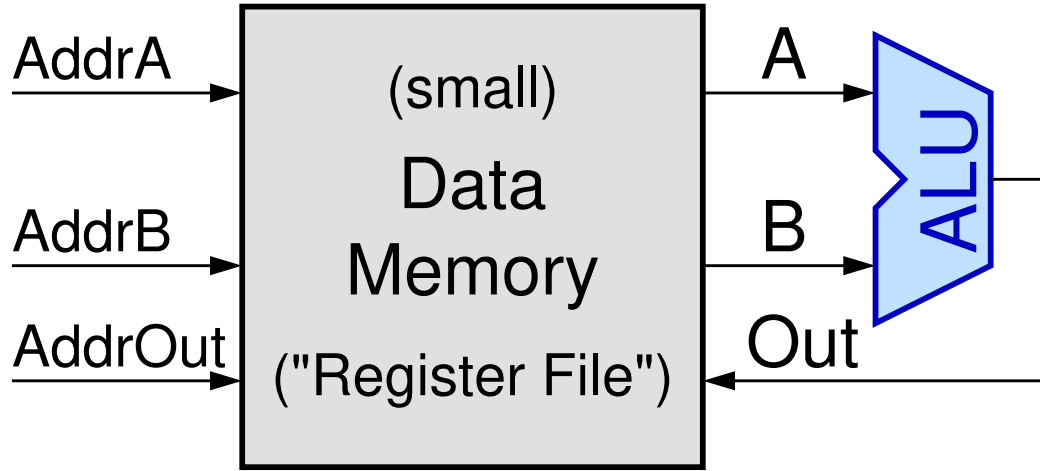
- Θα χρειαστεί με κάποιον τρόπο να προσαρμοστούμε, στις Ασκήσεις σε Προσομοιωτή...

Πρόγραμμα και Δεδομένα στη Μνήμη ή Μνήμες

Στην ίδια μνήμη το Πρόγραμμα και τα Δεδομένα;

- Πραγματικοί Υπολογιστές: στην ίδια («κεντρική») Μνήμη
 - Διότι υπάρχουν μεγάλα προγράμματα με λίγα δεδομένα, και υπάρχουν και μικρά προγράμματα με πολλά δεδομένα
 - Αλλά έχουν και μικρές & γρήγορες («κρυφές») μνήμες που είναι χωριστές για πρόγραμμα και δεδομένα, για να μπορούν ταυτόχρονα να διαβάζουν τωρινά δεδομένα & επόμενη εντολή
- Εμείς εδώ: σε χωριστές μνήμες, για απλότητα
 - προτιμάμε να διαβάζουμε την εντολή και τα δεδομένα της στον ίδιο (μακρύ) κύκλο ρολογιού από χωριστές μνήμες, αντί από την ίδια μνήμη σε δύο (βραχύτερους) κύκλους ρολογιού

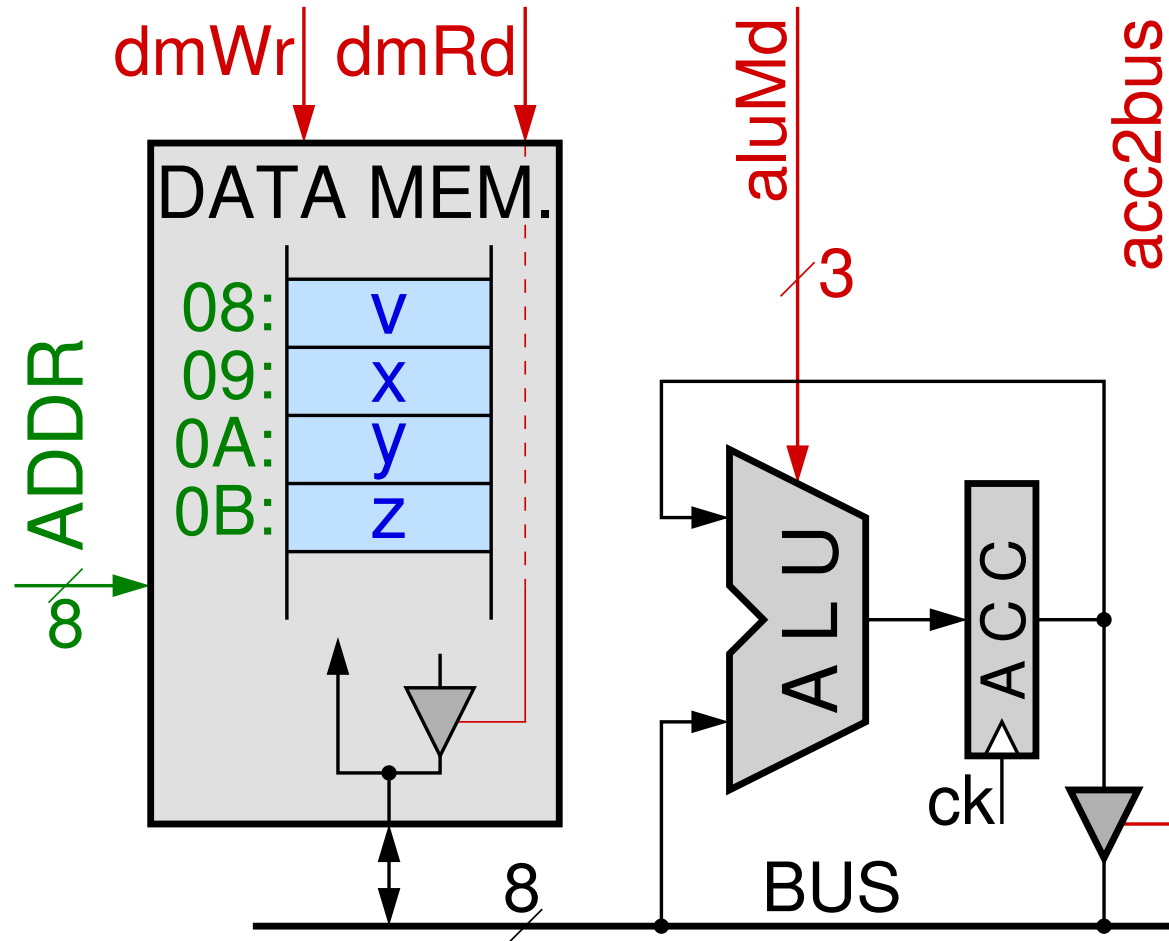
Δεδομένα για πράξη ALU: πόσες λέξεις και πού;



- Εμείς εδώ, για απλότητα: μονόπορτη μνήμη δεδομένων και ένας κύκλος ρολογιού
⇒ «Συσσωρευτής» (πρωτόγονο)

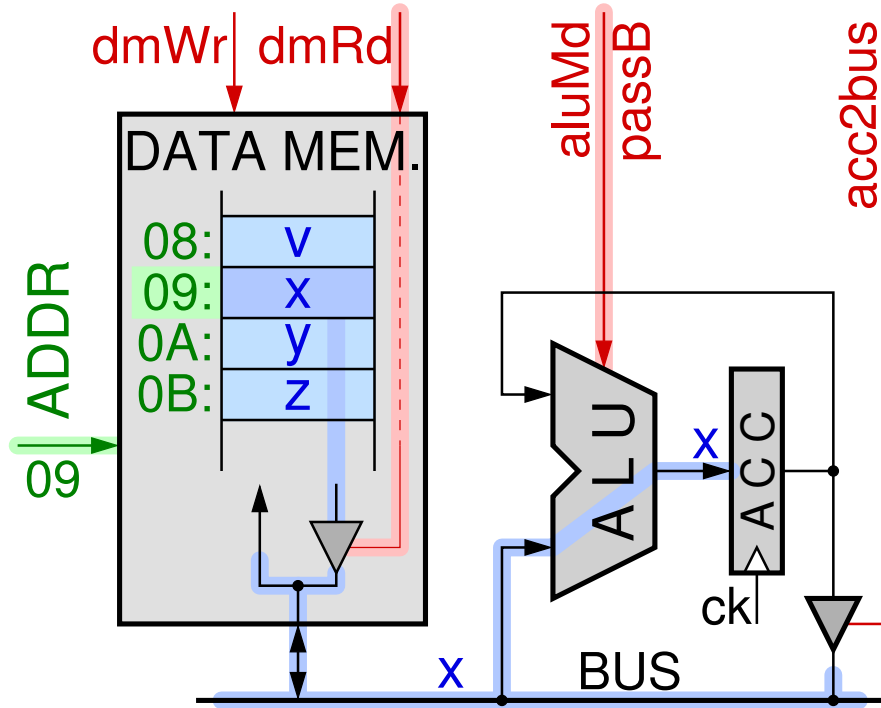
- Οι κανονικοί υπολογιστές: τρίπορτη, μικρή & γρήγορη μνήμη «συχνών» δεδομένων («αρχείο καταχωρητών»)
- Εναλλακτικά θα έπρεπε: 3 ή 4 κύκλοι ρολογιού από μονόπορτη μνήμη δεδομ.

Πράξεις ALU σε Επεξ. τ. Συσσωρευτή (Accumulator)

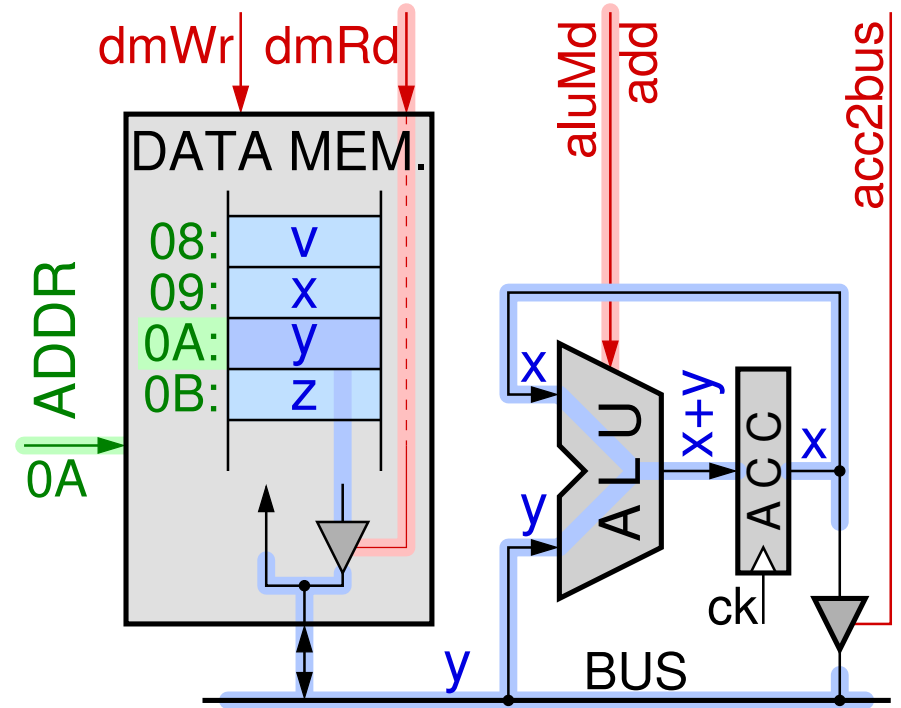


- Δύο από τους «τελεστέους» εξυπακούονται
- Προηγούμενο αποτέλεσμα από τον Συσσωρευτή
- Πράξη με κάτι νέο από μνήμη
- Αποτέλεσμα στον Συσσωρευτή

Παράδειγμα: $v = x + y - z$; (εντολές 1, 2 από 4)

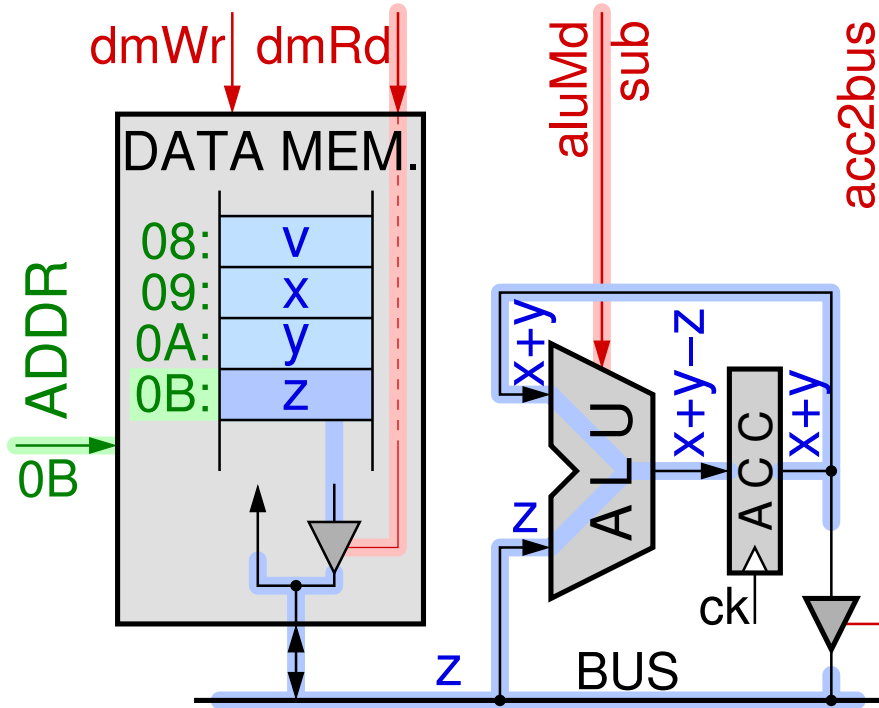


- $ACC \leftarrow x$
"load 09"

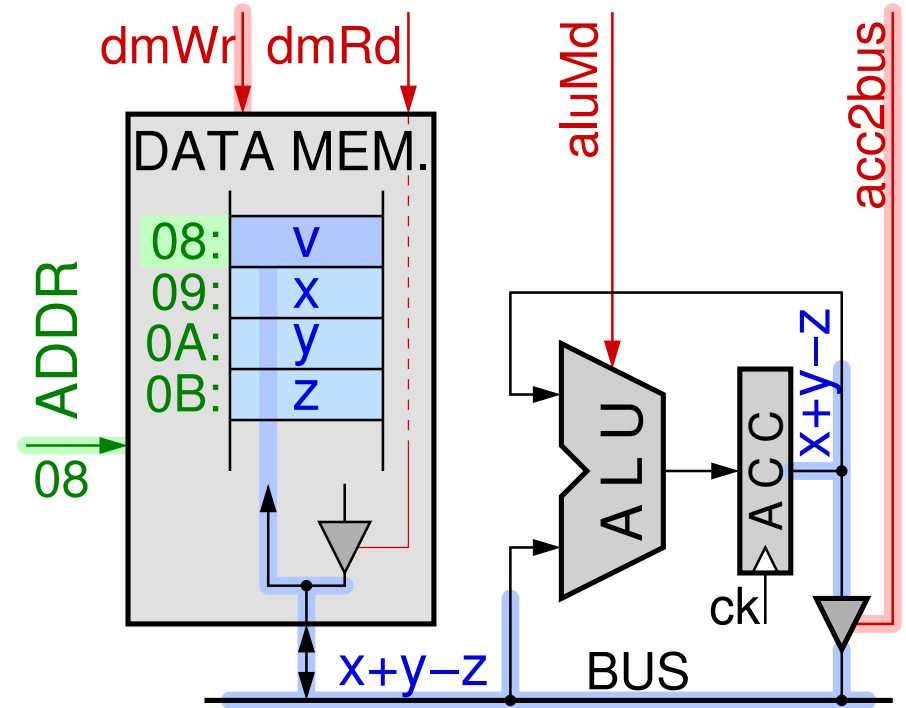


- $ACC \leftarrow ACC + y$
"add 0A"

Παράδειγμα: $v = x + y - z$; (εντολές 3, 4 από 4)



- $ACC \leftarrow ACC - z$
“sub 0B”



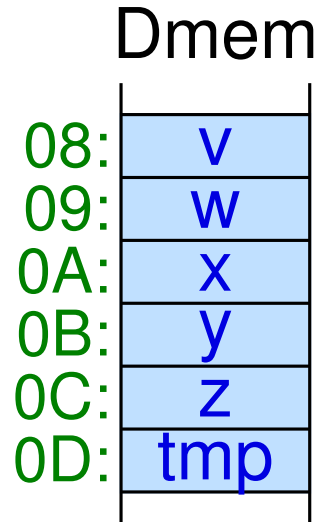
- $v \leftarrow ACC$
“store 08”

Πολυπλοκότερες εκφράσεις: ενδιάμεσα αποτελέσματα

$$V = X + Y * Z;$$

(έστω ότι έχουμε εντολή
πολλαπλασιασμού: *mul*)

```
load 0B    # y
mul 0C     # z
add 0A     # x
store 08   # v
```



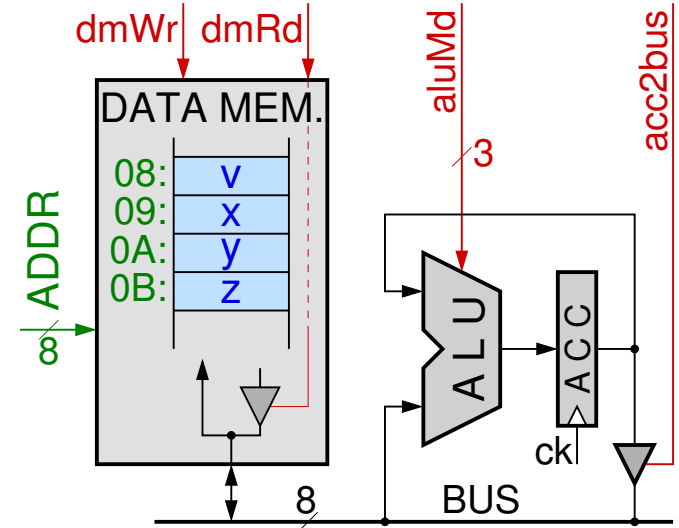
$$V = X * W + Y * Z;$$

```
load 0A    # x
mul 09     # w
store 0D   # tmp
load 0B    # y
mul 0C     # z
add 0D     # tmp
store 08   # v
```

(θα μπορούσε και το *V* να παίξει το ρόλο του *tmp*, αφού δεν εμφανίζεται το *V* στη δεξιά πλευρά της εκχώρησης)

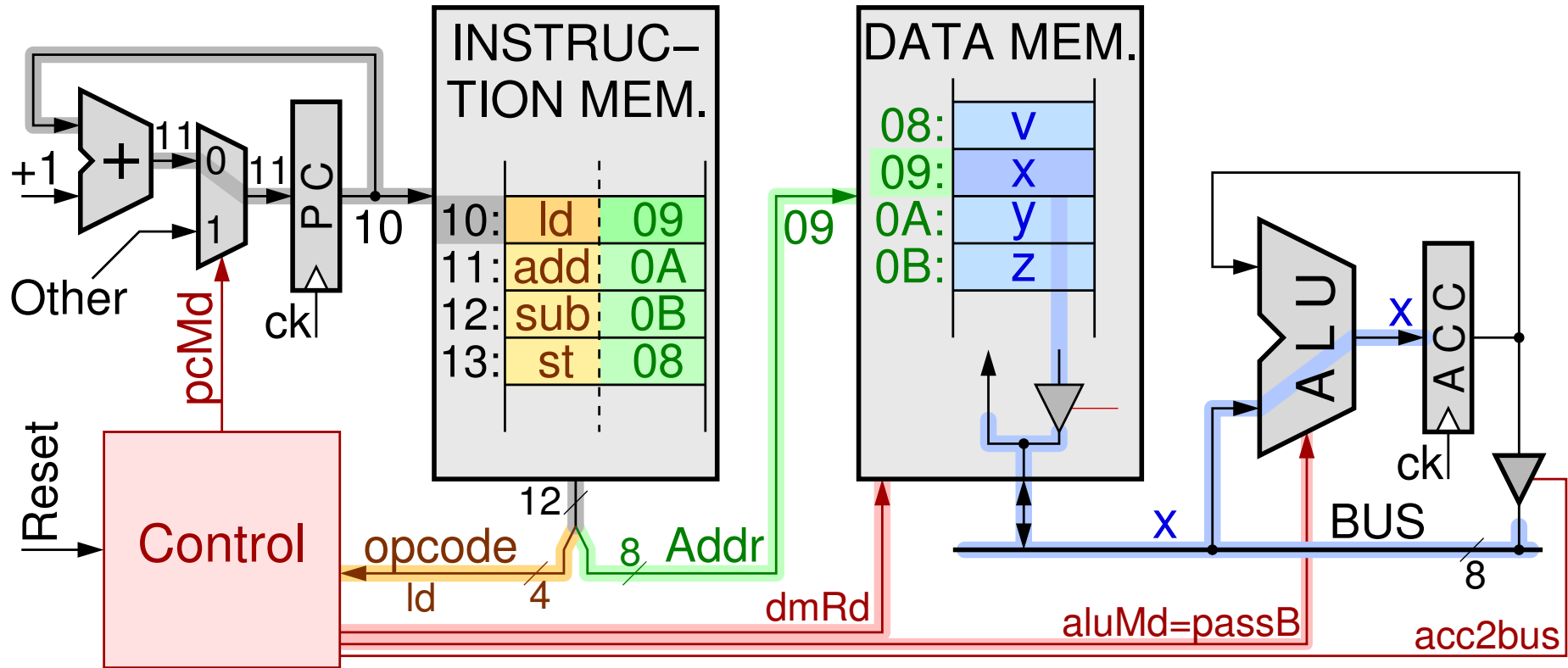
Εντολές: η μονάδα για τον έλεγχο της λειτουργίας

- Για να εκτελούνται οι πράξεις που συζητήσαμε, η μία μετά την άλλη, πρέπει κάποιος να δίνει τις σωστές διευθύνσεις των μεταβλητών στη μνήμη δεδομένων και τις σωστές τιμές στα σήματα ελέγχου



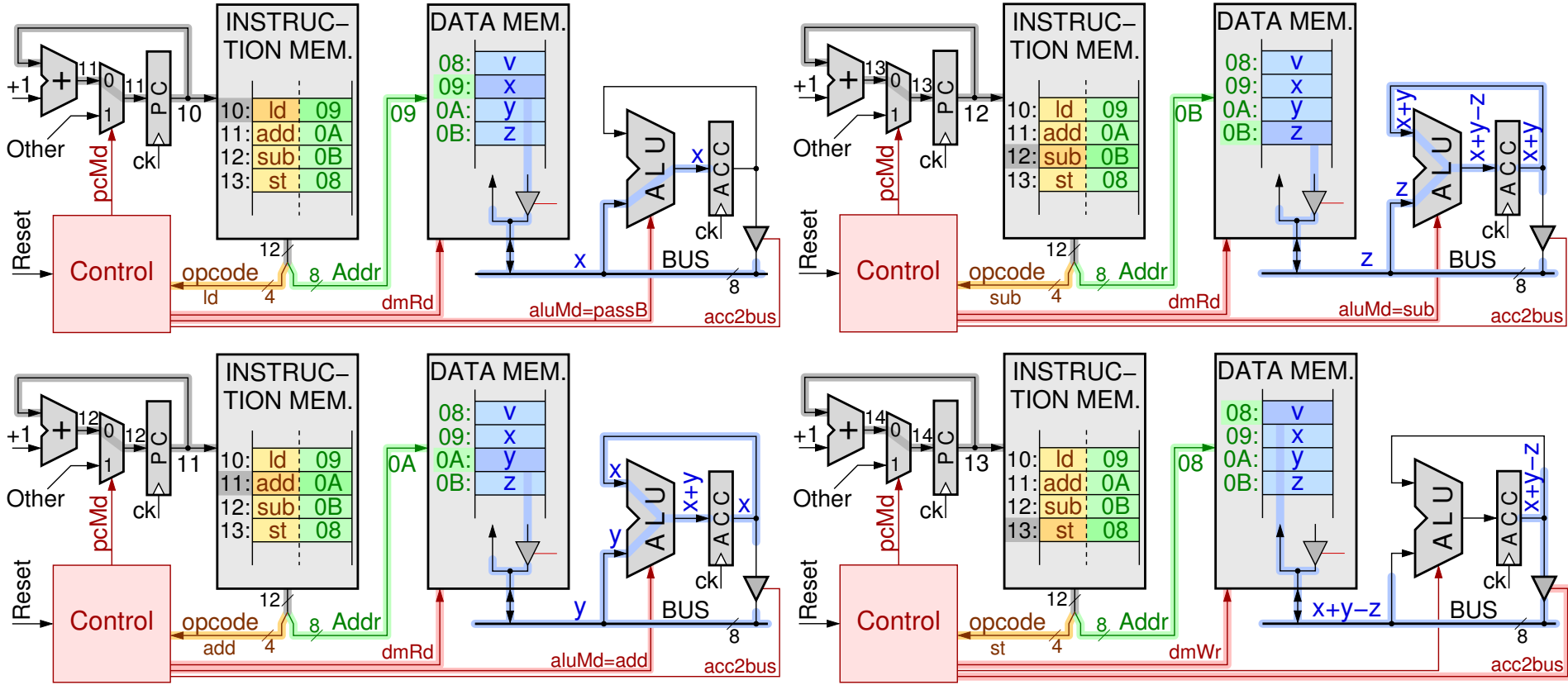
- Για την κάθε πράξη, αυτές οι οδηγίες πηγάζουν από την *Εντολή (Instruction)*, που είναι η μονάδα σύνθεσης των προγραμμάτων (Λογισμικού) στην *Γλώσσα Μηχανής*
- Κάτι ανάλογο προς τα *Statements* στις Γλώσσες ψηλού επιπέδου

Εντολές: Τι να κάνουμε, σε ποιόν να το κάνουμε



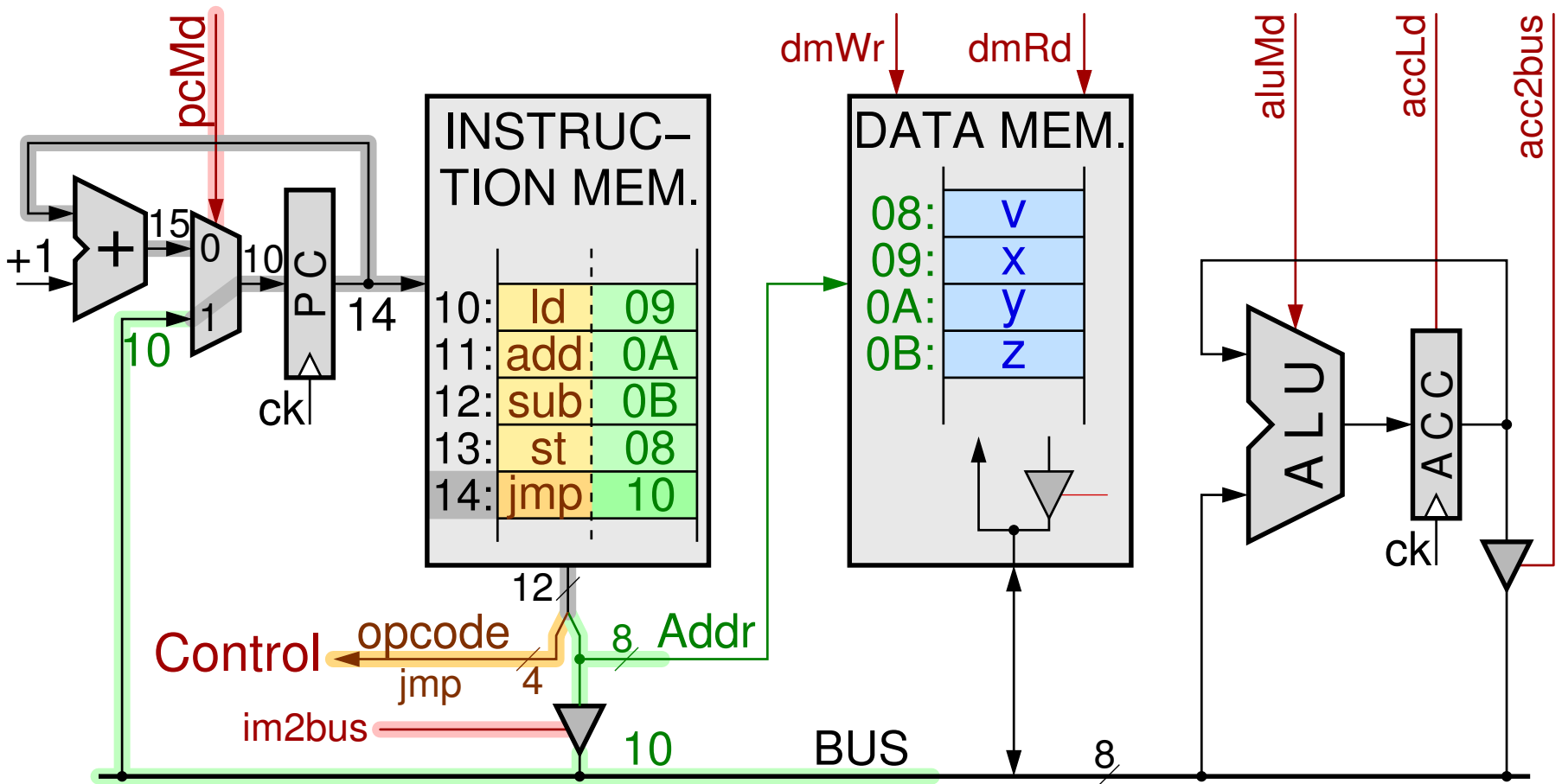
- Τι: *opcode* – Έλεγχος
- Σε ποιόν: Διεύθυνση τελεστή

Ανάκληση & Εκτέλεση Εντολών, και του Προγράμματος

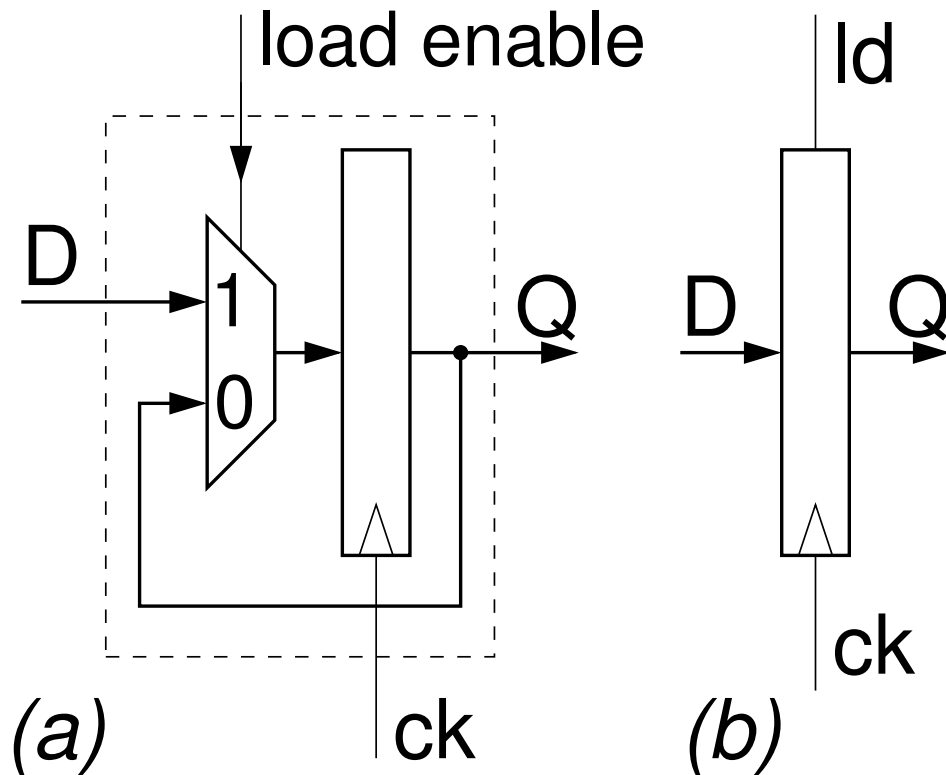


- Program Counter (PC): ανάγνωση των εντολών του προγράμματος, μία-μία με τη σειρά από τη μνήμη εντολών, ώστε ο υπολογιστής να εκτελέσει ό,τι λέει η καθεμία

Jump: Επόμενη εντολή όχι «η από κάτω»

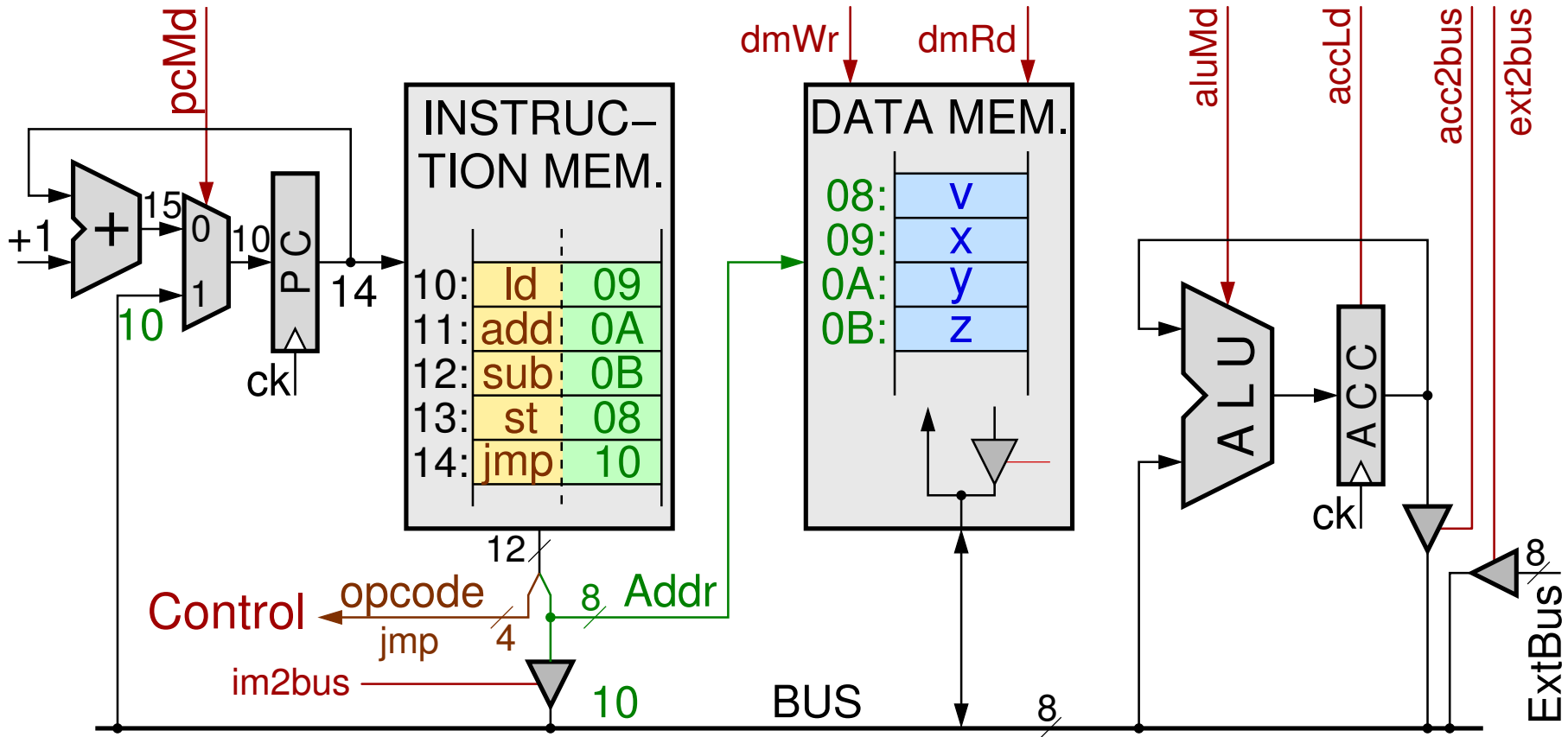


Καταχωρητές με Επίτρεψη Φόρτωσης



- Δεν θέλουμε πάντα όλοι οι ακμοπυροδότητοι καταχωρητές μας να φορτώνονται στην κάθε ακμή ρολογιού:
- Σήμα ελέγχου σε ποιές ακμές να φορτώνονται και σε ποιές όχι
- Π.χ. ο ACC στις εντολές *Store*, *Jump* να διατηρεί το περιεχόμενό του

Εντολή Input: από εξωτερική είσοδο στη μνήμη



- Μία ακόμα είσοδος προς το BUS, δεξιά, από περιφ. συσκ.

Ρεπερτόριο Εντολών (Instruction Set) – πρώτο μισό

Ρεπερτόριο Εντολών (Instruction Set):

Γλώσσα Μηχανής: Γλ. **Assembly**: Σημαίνει:

0000αααααααα	<i>add</i>	<i>ADDR</i>	$ACC \leftarrow ACC + DM[ADDR]$
0001αααααααα	<i>sub</i>	<i>ADDR</i>	$ACC \leftarrow ACC - DM[ADDR]$
0010αααααααα	<i>and</i>	<i>ADDR</i>	$ACC \leftarrow ACC \mathbf{AND} DM[ADDR]$
0011αααααααα	<i>nor</i>	<i>ADDR</i>	$ACC \leftarrow \mathbf{NOT} (ACC \mathbf{OR} DM[ADDR])$
0100αααααααα	<i>input</i>	<i>ADDR</i>	$DM[ADDR] \leftarrow$ εξωτερική είσοδος από πληκτρολόγιο (§ 11.8)
0101αααααααα	<i>load</i>	<i>ADDR</i>	$ACC \leftarrow DM[ADDR]$
0110αααααααα	<i>store</i>	<i>ADDR</i>	$DM[ADDR] \leftarrow ACC$
0111αααααααα	<i>jump</i>	<i>ADDR</i>	$PC \leftarrow ADDR$ (επόμεν. εντ. σε ADDR: § 11.9)

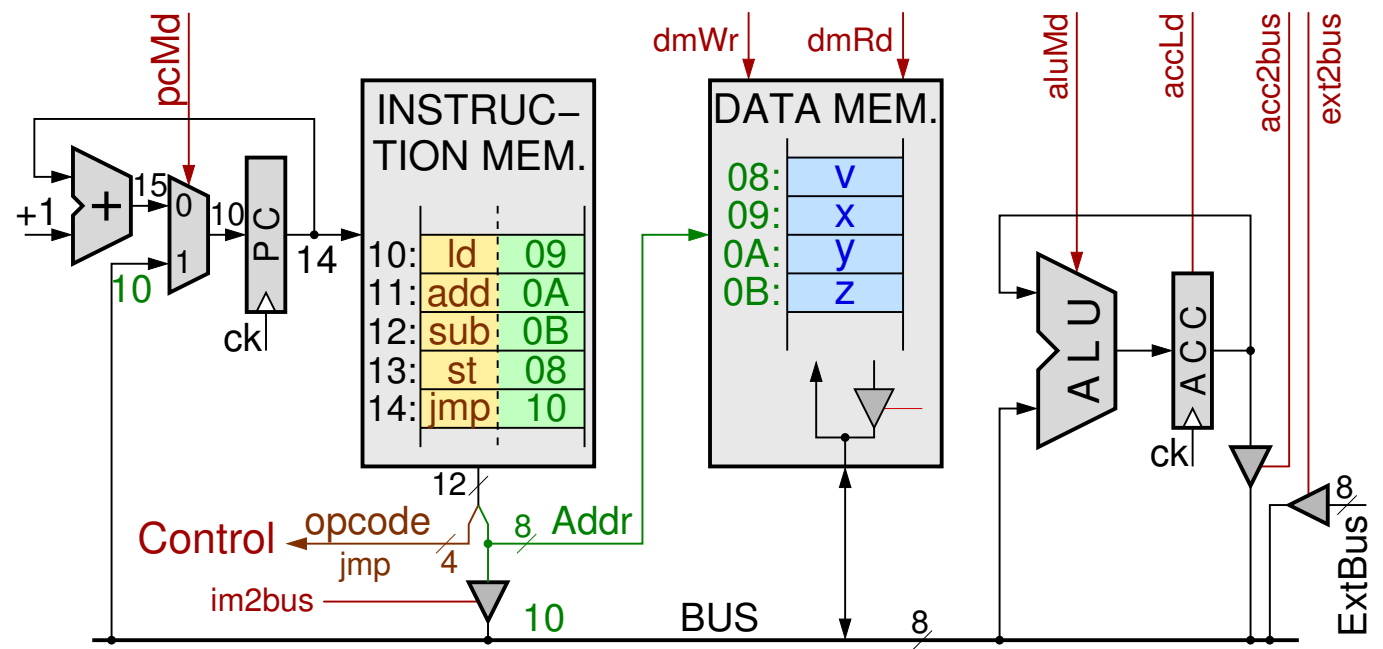
- **DM[addr]** = η λέξη στη διεύθυνση *addr* της **Data Memory**
- Εντολή *Input*: όπως η *Store* αλλά από εξωτερ. είσοδο αντί ACC
- Εντολή *Jump*: επόμενη εντολή άλλη από την «από κάτω (+1)»

Το (συνδυαστικό) Κύκλωμα Ελέγχου

<i>opcode:</i>	<i>Λειτουργία:</i>	<i>aluMd</i>	<i>acc2bus</i>	<i>dmWr</i>	<i>pcMd</i>				
		<i>dmRd</i>	<i>accLd</i>	<i>ext2bus</i>	<i>im2bus</i>				
0000 (add)	ACC:=ACC+DM[A]	1	000	1	0	0	0	0	0
0001 (sub)	ACC:=ACC-DM[A]	1	001	1	0	0	0	0	0
0010 (and)	ACC:=ACCandDM[A]	1	010	1	0	0	0	0	0
0011 (nor)	ACC:=ACCnorDM[A]	1	011	1	0	0	0	0	0
0100 (input)	DM[A]:=ExtBus	0	xxx	0	0	1	1	0	0
0101 (load)	ACC:=DM[A]	1	1xx	1	0	0	0	0	0
0110 (store)	DM[A]:=ACC	0	xxx	0	1	0	1	0	0
0111 (jump)	PC := A	0	xxx	0	0	0	0	1	1

Πάντα για όλες αυτές τις εντολές: $pcLd = 1$; $addr_md = 0$;

<i>opcode:</i>	<i>Λειτουργία:</i>	<i>dmRd</i>	<i>aluMd</i>	<i>acc2bus</i>	<i>dmWr</i>	<i>pcMd</i>
			<i>accLd</i>	<i>ext2bus</i>		<i>im2bus</i>
0000 (add)	ACC:=ACC+DM[A]	1	000	1	0	0
0001 (sub)	ACC:=ACC-DM[A]	1	001	1	0	0
0010 (and)	ACC:=ACCandDM[A]	1	010	1	0	0
0011 (nor)	ACC:=ACCnorDM[A]	1	011	1	0	0
0100 (input)	DM[A]:=ExtBus	0	xxx	0	1	1
0101 (load)	ACC:=DM[A]	1	1xx	1	0	0
0110 (store)	DM[A]:=ACC	0	xxx	0	1	1
0111 (jump)	PC := A	0	xxx	0	0	1



Το πλήρες Datapath και σήματα ελέγχου

