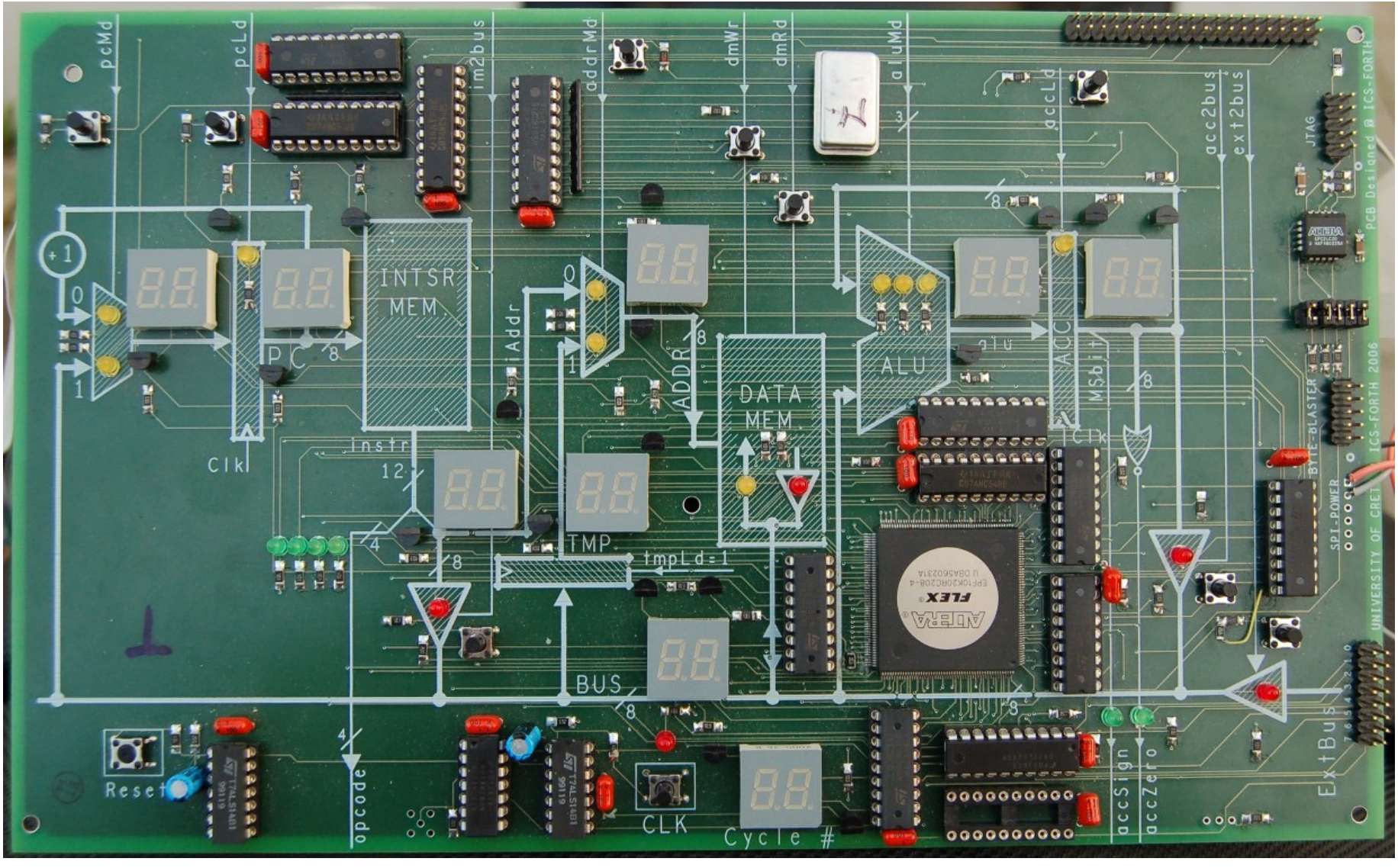
Ένας απλός Υπολογιστής: Datapath & Εντολές Πράξεων

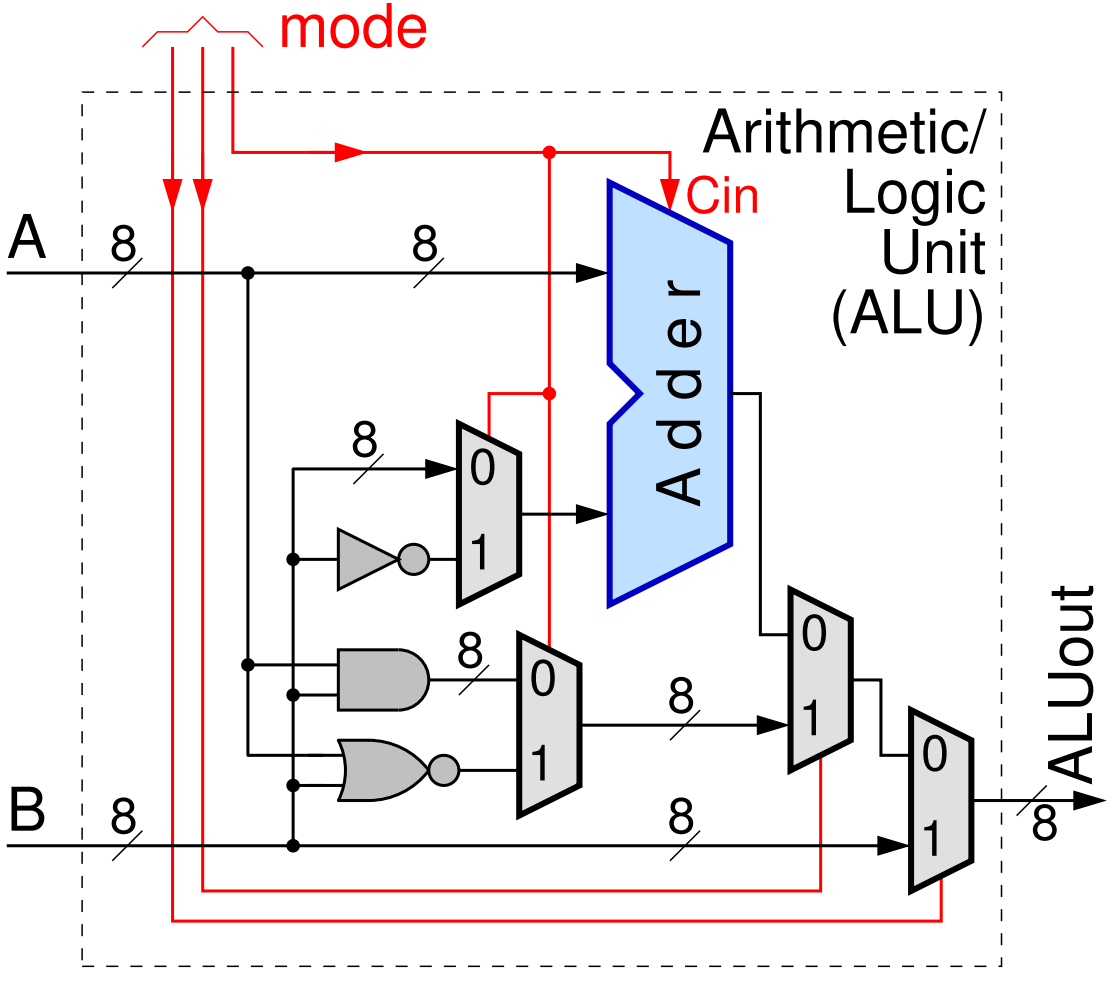
11α (§11.1 - 11.11) – 1-6 Δεκ. 2023 – Μανόλης Κατεβαίνης

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/Έξοδος (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

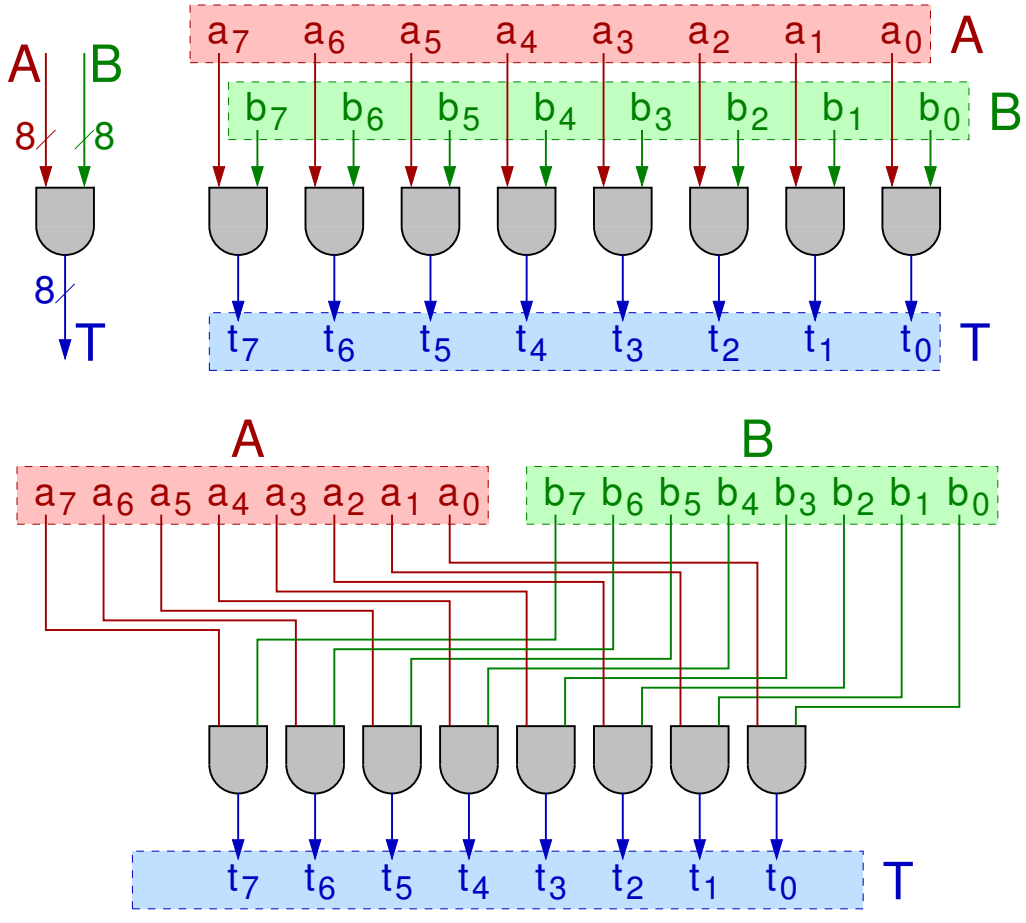

pcMd
pcLd
im2bus
addrMd
dmWr
dmRd
aluMd
accLd
acc2bus
ext2bus
JTAG
PCB Designed @ ICS-FORTH
UNIVERSITY OF CRETE ICS-FORTH 2006
BYTE-BLASTER
SPI-POWER
+1
INTSR MEM
PC
Clk
iAddr
ADDR
DATA MEM
ALU
ACC
MSbit
Clk
Instr
12
TMP
tmpLd=1
BUS
Reset
opcode
CLK
Cycle #
accSign
accZero
ExtBus

Κάπου οι αρ. πράξεις: Αριθμητική/Λογική Μονάδα



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

Bitwise Logical Operations



Παραδείγματα Χρήσεων:

- Bit field isolation

```
01101010   a
00001111   mask
--------
00001010   a & mask
```

- Bit test

```
010b0110   a
00010000   msk
--------
000b0000   a&msk != 0
```

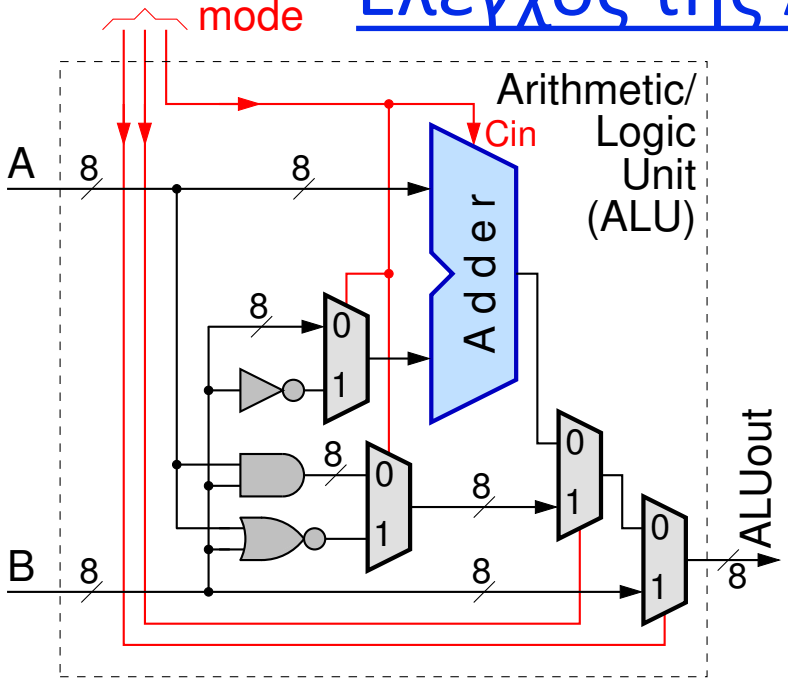
- Bit set

```
01110x01   a
00000100   mask
--------
01110101   a | mask
```

- Bit clear

```
01101x10   a
11111011   mask
--------
01101010   a & mask
```

Έλεγχος της ALU, και επιλογή Πράξεων



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

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

(Το 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

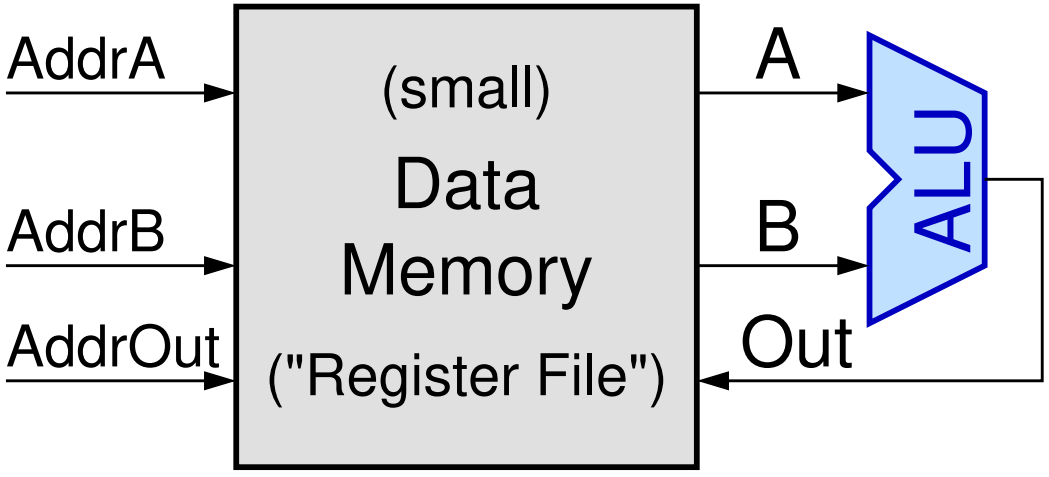
- Θα χρειαστεί με κάποιον τρόπο
  να προσαρμοστούμε, στις Ασκήσεις σε Προσομοιωτή…

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 γιά πρόγραμμα και δεδομένα, γιά να μπορούν *ταυτόχρονα* να διαβάζουν τωρινά δεδομένα & επόμενη εντολή
- Εμείς εδώ: σε χωριστές μνήμες, γιά 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 γιά πράξη ALU: πόσες λέξεις και πού;

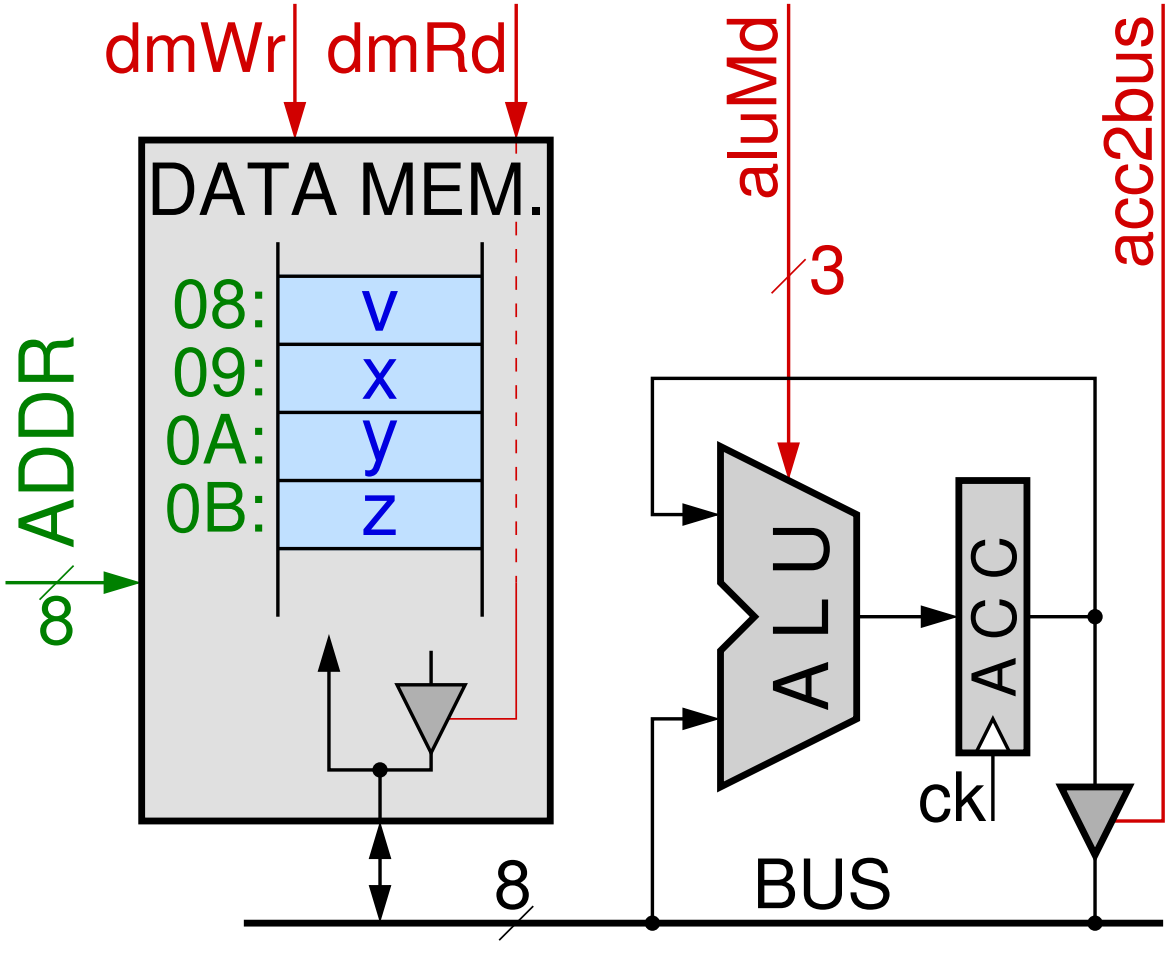


- Εμείς εδώ, γιά απλότητα: μονόπορτη μνήμη δεδομένων και ένας κύκλος ρολογιού

  ⇒ «Συσσωρευτής» (πρωτόγονο)

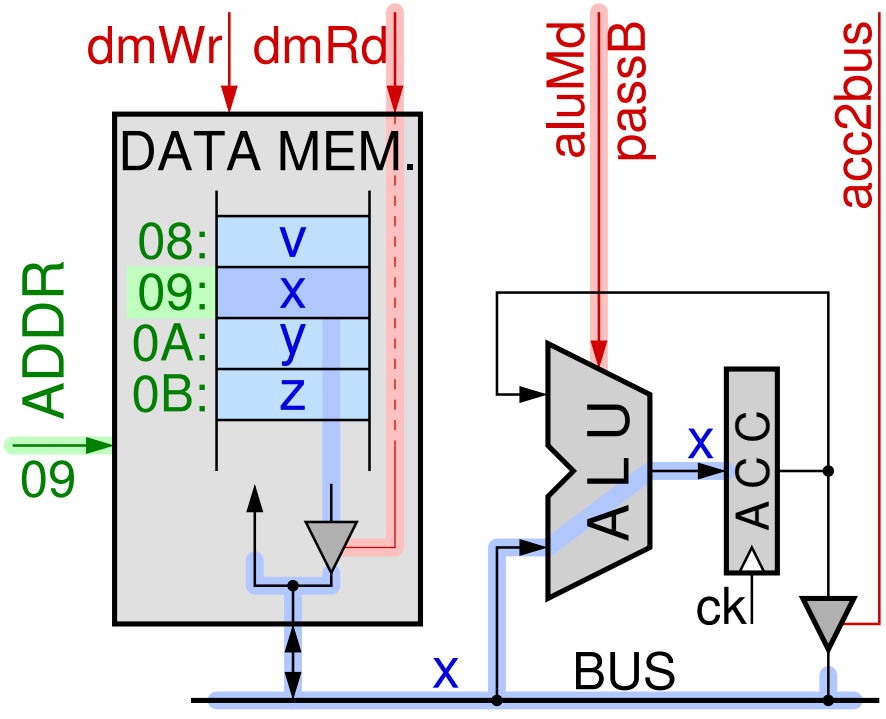
- Οι κανονικοί υπολογιστές: τρίπορτη, μικρή & γρήγορη μνήμη «συχνών» δεδομένων («αρχείο καταχωρητών»)
- Εναλλακτικά θα έπρεπε: 3 ή 4 κύκλοι ρολογιού από μονοπορτη μνήμη δεδομ.

Πράξεις ALU σε Επεξ. τ. Συσσωρευτή (Accumulator)



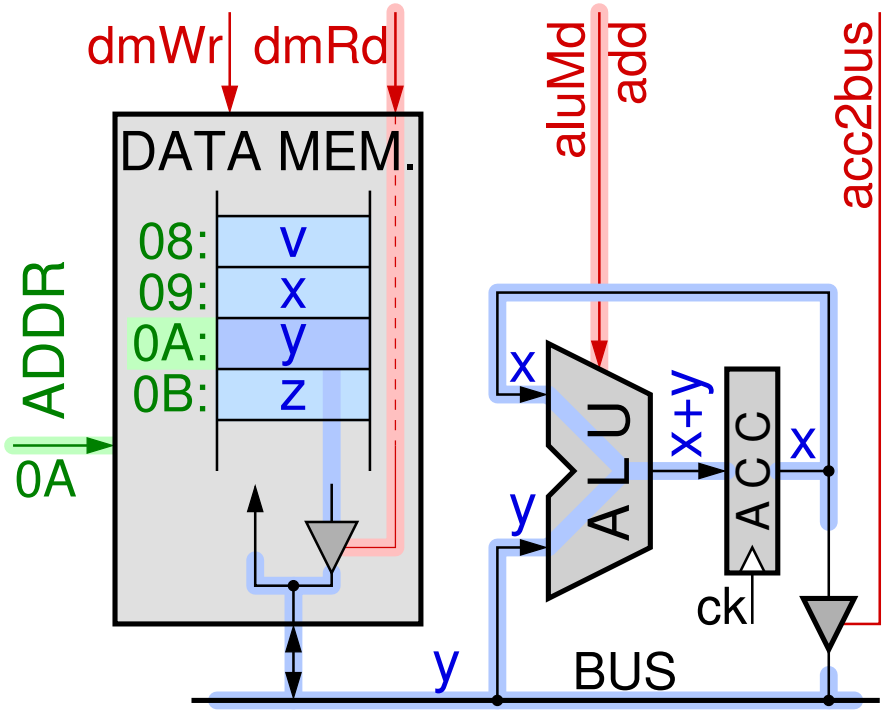
- Δύο από τους «τελεστέους» εξυπακούονται
- Προηγούμενο αποτέλεσμα από τον Συσσω-ρευτή
- Πράξη με κάτι νέο από μνήμη
- Αποτέλεσμα στον Συσσω-ρευτή

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



- ACC ← x

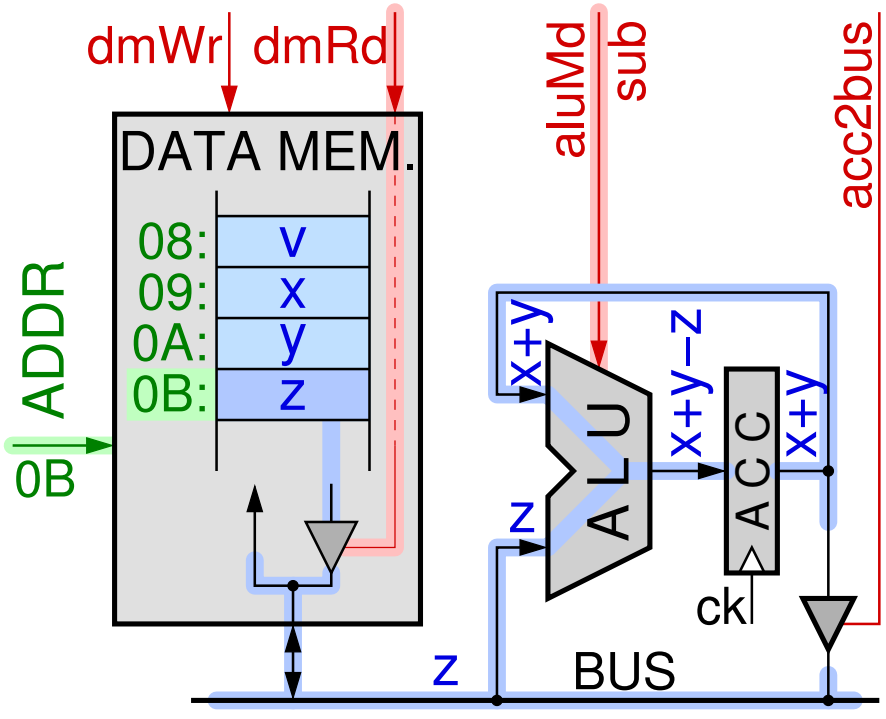
"load 09"

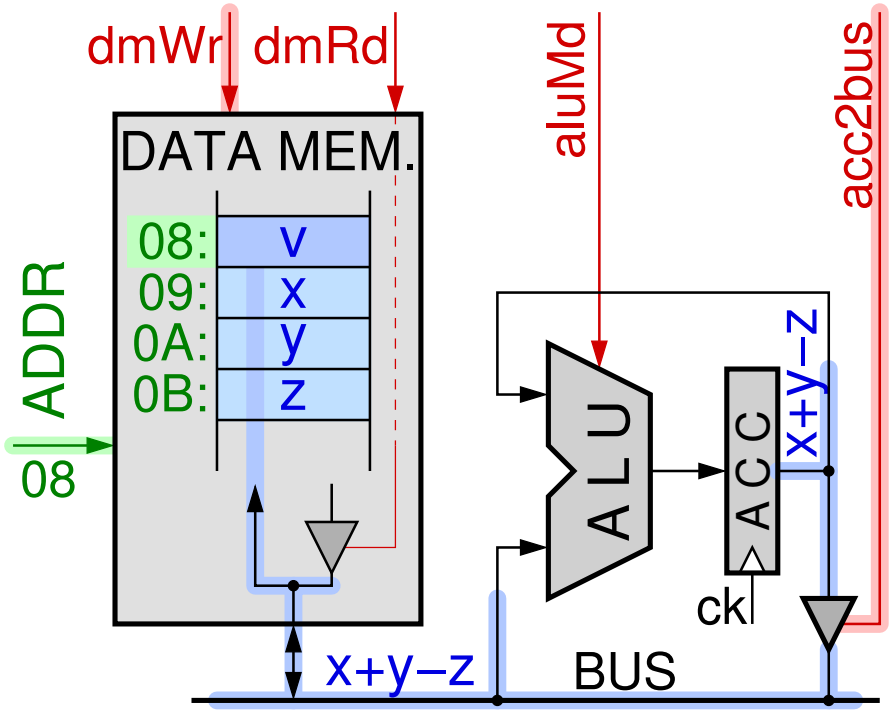


- ACC ← ACC + y

"add 0A"

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





- ACC ← ACC – z

"sub 0B"

- v ← ACC

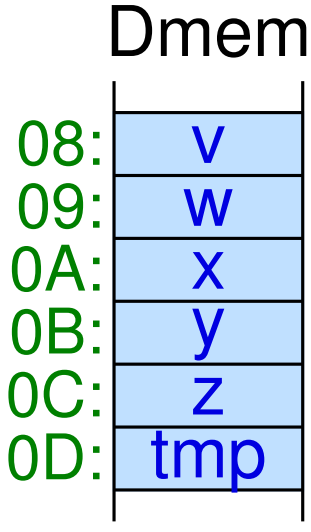
"store 08"

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
```



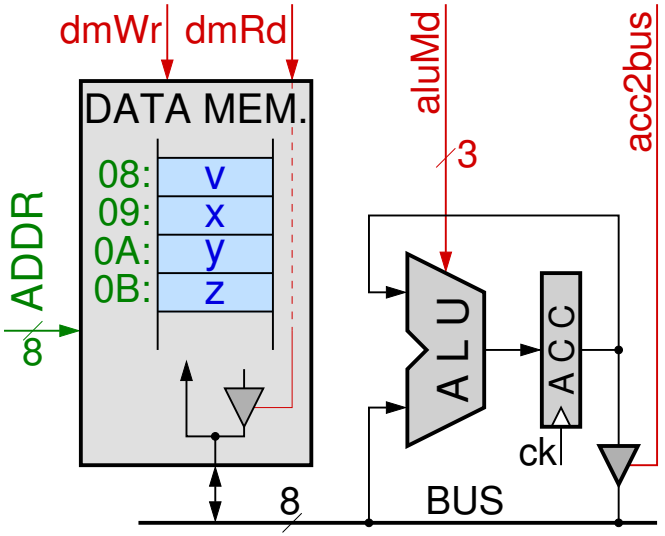
v = x∗w + y∗z;

```
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)

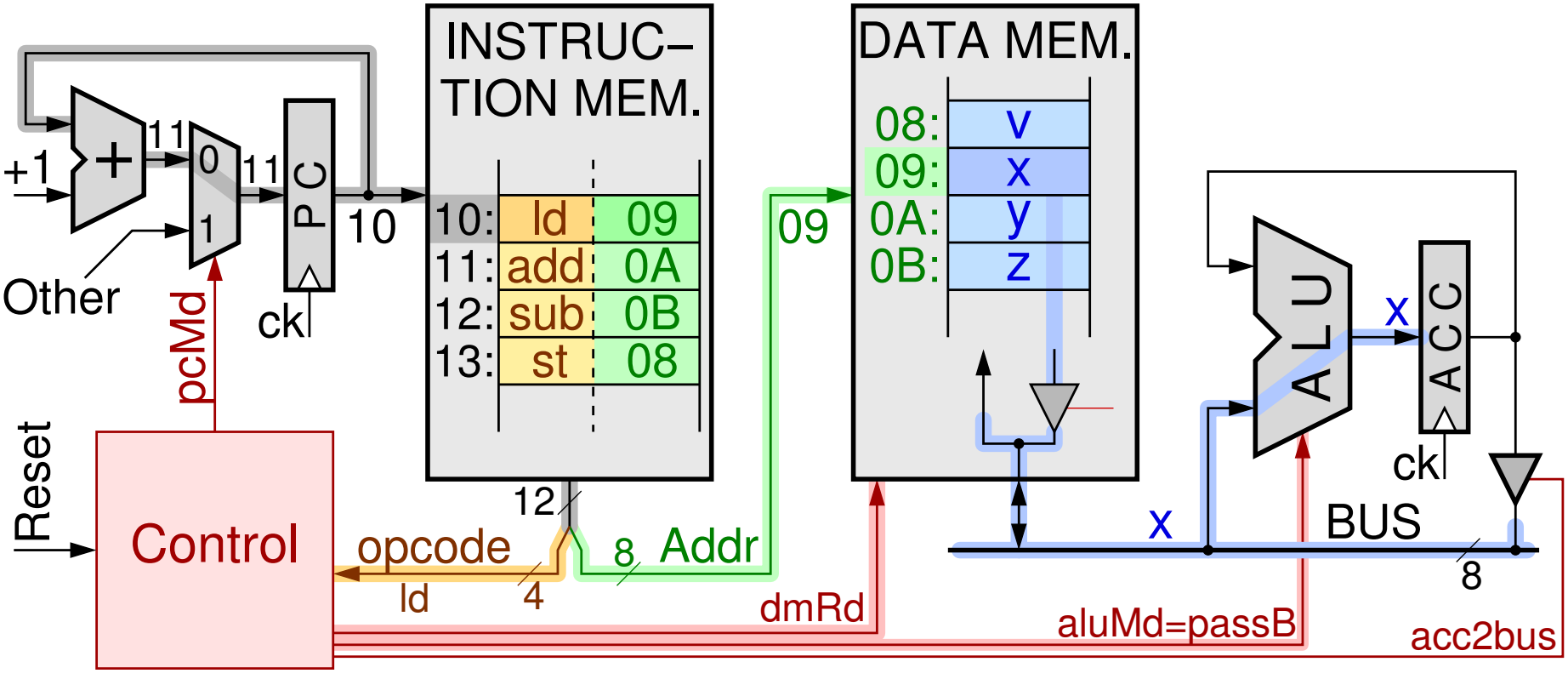
Εντολές: η μονάδα γιά τον έλεγχο της λειτουργίας

- Γιά 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



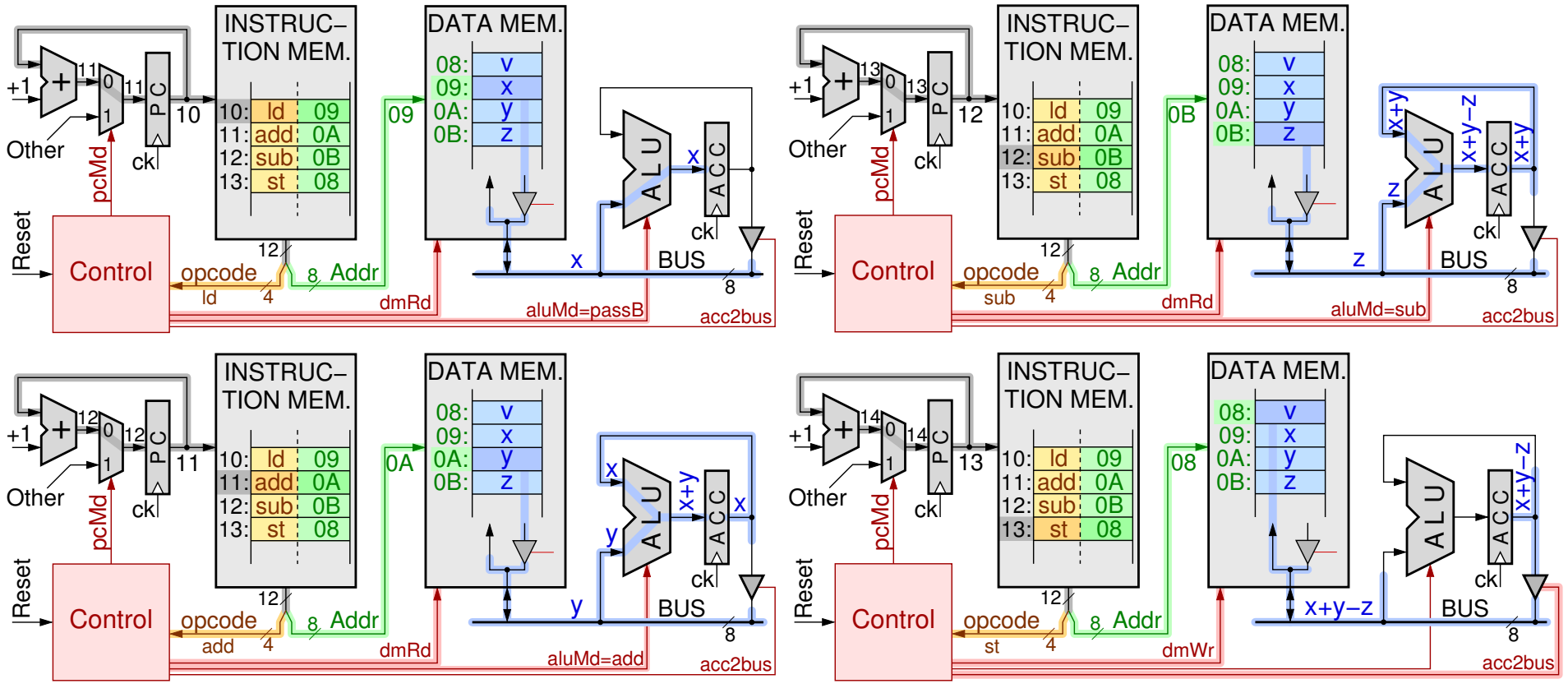
- Γιά την κάθε πράξη, αυτές οι οδηγίες πηγάζουν από την *Εντολή* (*Instruction*), που είναι η μονάδα σύνθεσης των προγραμμάτων (Λογισμικού) στην *Γλώσσα Μηχανής*
- Κάτι ανάλογο προς τα *Statements* στις Γλώσσες ψηλού επιπέδου

Εντολές: Τι να κάνουμε, σε ποιόν να το κάνουμε



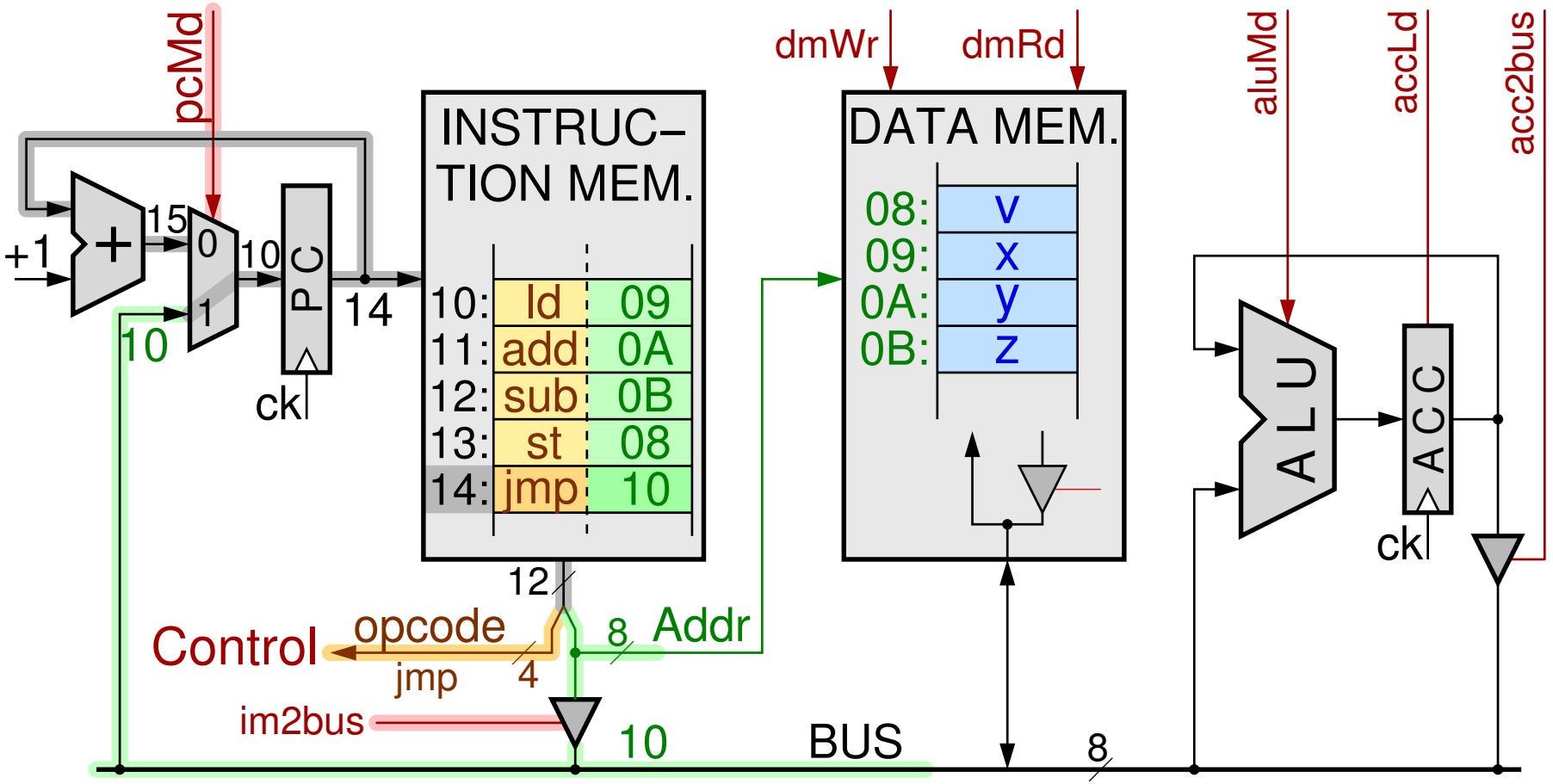
- Τι: *opcode* – Έλεγχος
- Σε ποιόν: Διεύθυνση τελεστέου

Ανάκληση & Εκτέλεση Εντολών, και του Προγράμματος

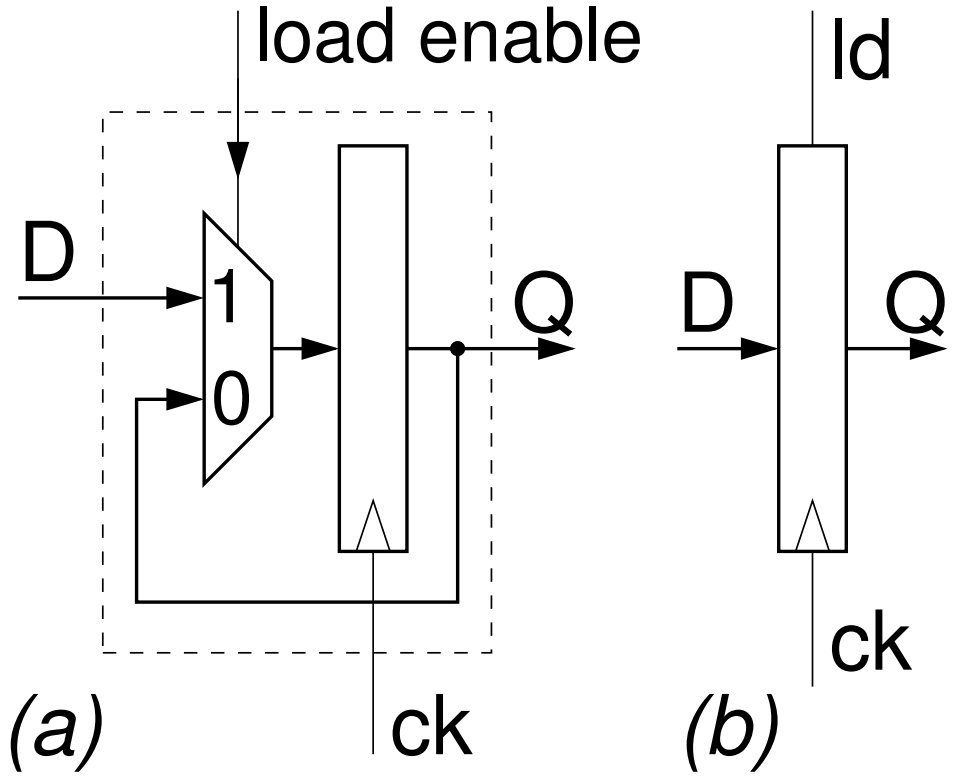


- Program Counter (PC): ανάγνωση των εντολών του προγράμματος, μία-μία με τη σειρά από τη μνήμη εντολών, ώστε ο υπολογιστής να εκτελέσει ό,τι λέει η καθεμία

Jump: Επόμενη εντολή όχι «η από κάτω»

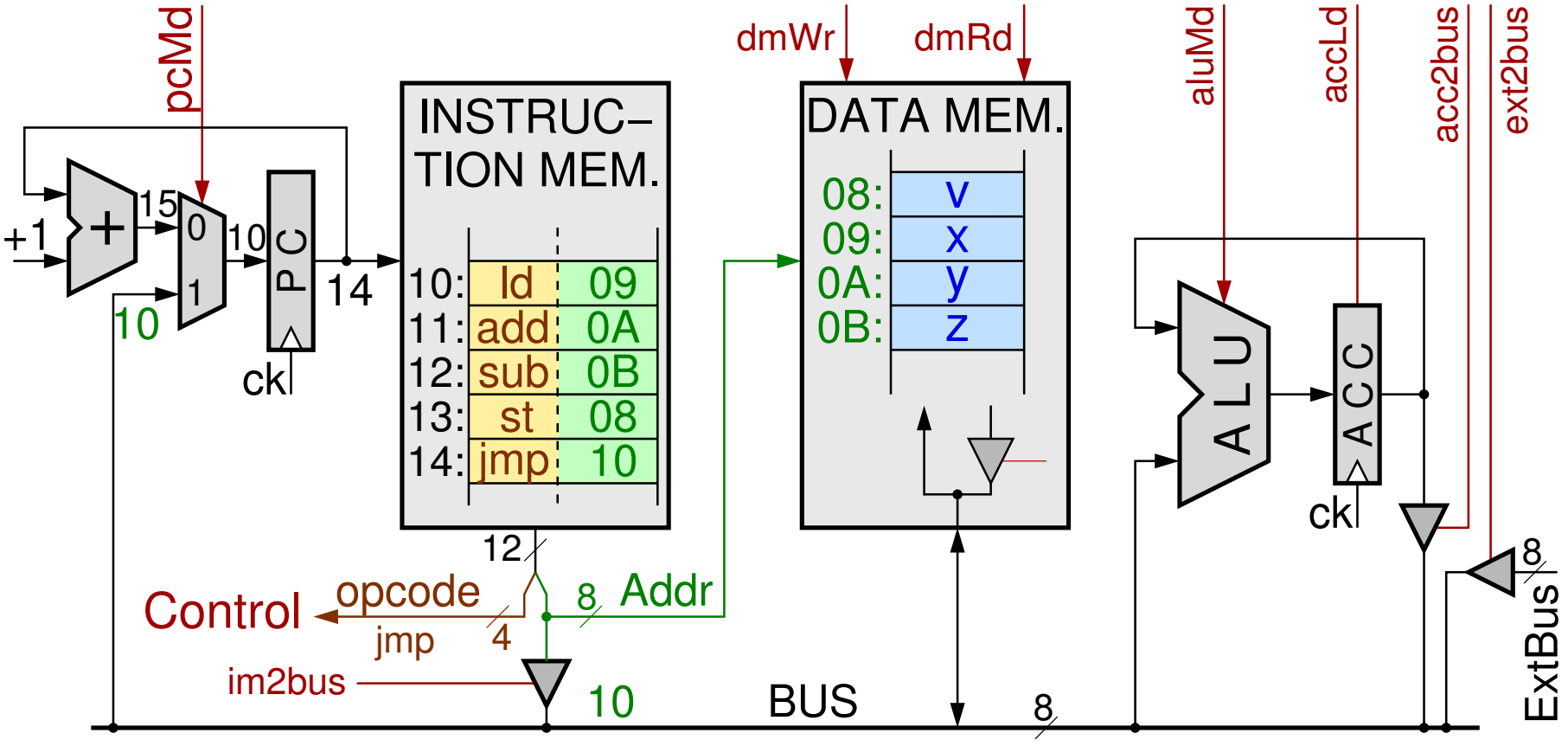


Καταχωρητές με Επίτρεψη Φόρτωσης



- Δεν θέλουμε πάντα όλοι οι ακμοπυροδότητοι καταχωρητές μας να φορτώνουνται 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:**
0000aaaaaaaa	*add*	*ADDR*	ACC <- ACC + DM[ADDR]
0001aaaaaaaa	*sub*	*ADDR*	ACC <- ACC - DM[ADDR]
0010aaaaaaaa	*and*	*ADDR*	ACC <- ACC **AND** DM[ADDR]
0011aaaaaaaa	*nor*	*ADDR*	ACC <- **NOT** (ACC **OR** DM[ADDR])
0100aaaaaaaa	*input*	*ADDR*	DM[ADDR] <- εξωτερική είσοδος από πληκτρολόγιο (§11.8)
0101aaaaaaaa	*load*	*ADDR*	ACC <- DM[ADDR]
0110aaaaaaaa	*store*	*ADDR*	DM[ADDR] <- ACC
0111aaaaaaaa	*jump*	*ADDR*	PC <- ADDR (επόμ.εντ.σε ADDR: §11.9)

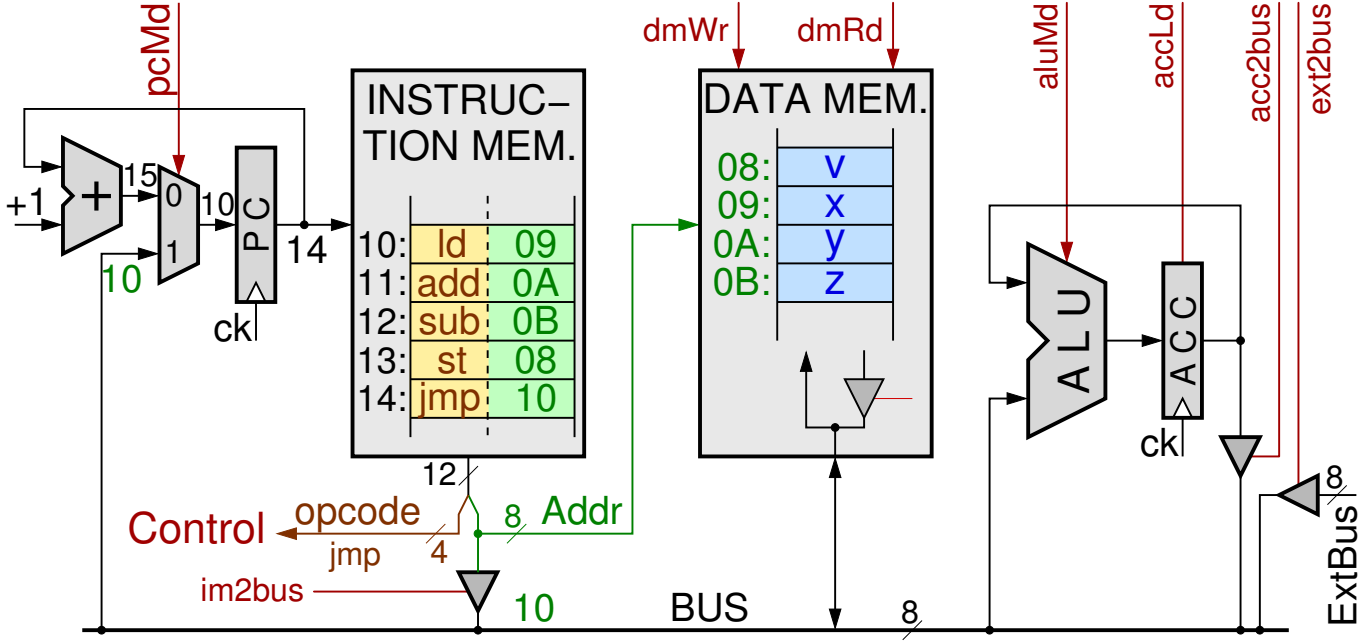
- **DM**[*addr*] = η λέξη στη διεύθυνση *addr* της **D**ata **M**emory
- Εντολή *Input*: όπως η *Store* αλλά από εξωτερ. είσοδο αντί ACC
- Εντολή Jump: επόμενη εντολή άλλη από την «από κάτω (+1)»

Το (συνδυαστικό) Κύκλωμα Ελέγχου

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

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

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



Το πλήρες Datapath και σήματα ελέγχου