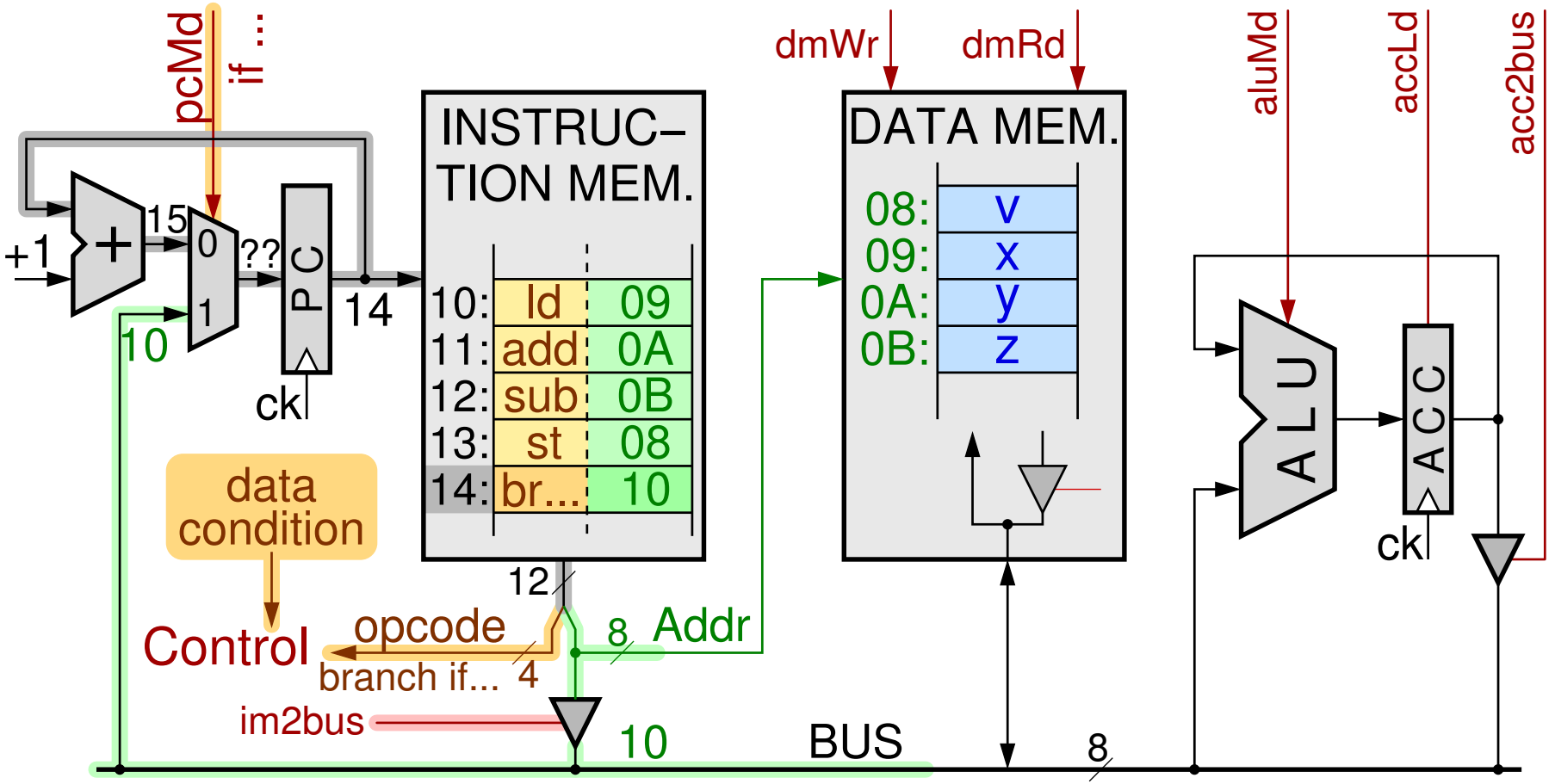
Απλός Υπολογιστής: Διακλαδώσεις και Έμμεσες Προσπελάσεις (Pointers)

12α (§12.1 - 12.7) – 9-14 Δεκ. 2020 – Μανόλης Κατεβαίνης

Διακλαδώσεις: Εάν... Επόμενη εντολή όχι «η από κάτω»



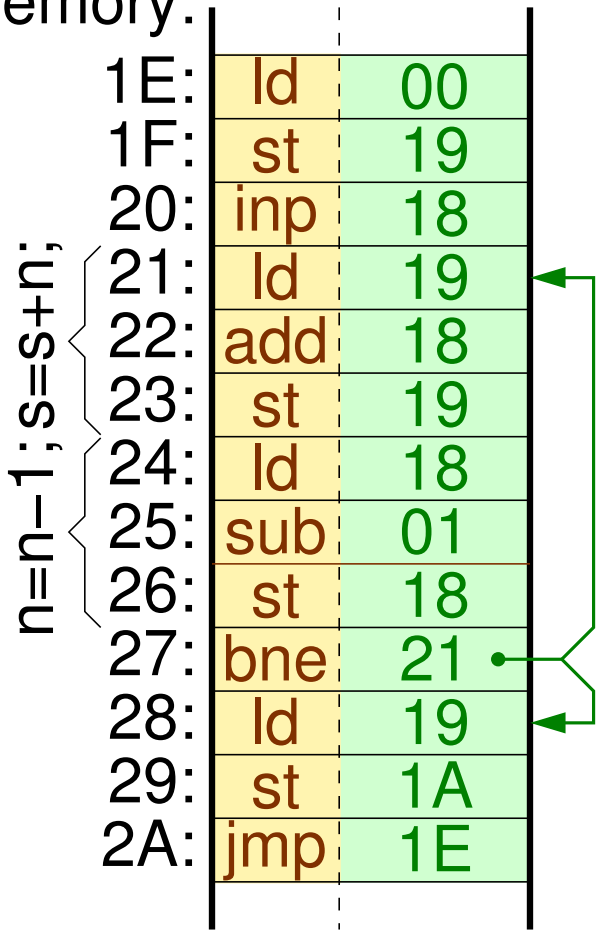
Απλός βρόχος: Άθροισμα (n)+(n-1)+(n-2)+…+(2)+(1)

```
s = 0;  n = input();
do { s = s+n;  n = n–1;
     } while (n != 0)
S = s;  /* print sum */
```

Data Memory:

00:	0
01:	1
	• • •
18:	n = ~~8~~ ~~7~~ 6
19:	s = ~~0~~ ~~8~~ F
1A:	S

Instr. Memory:



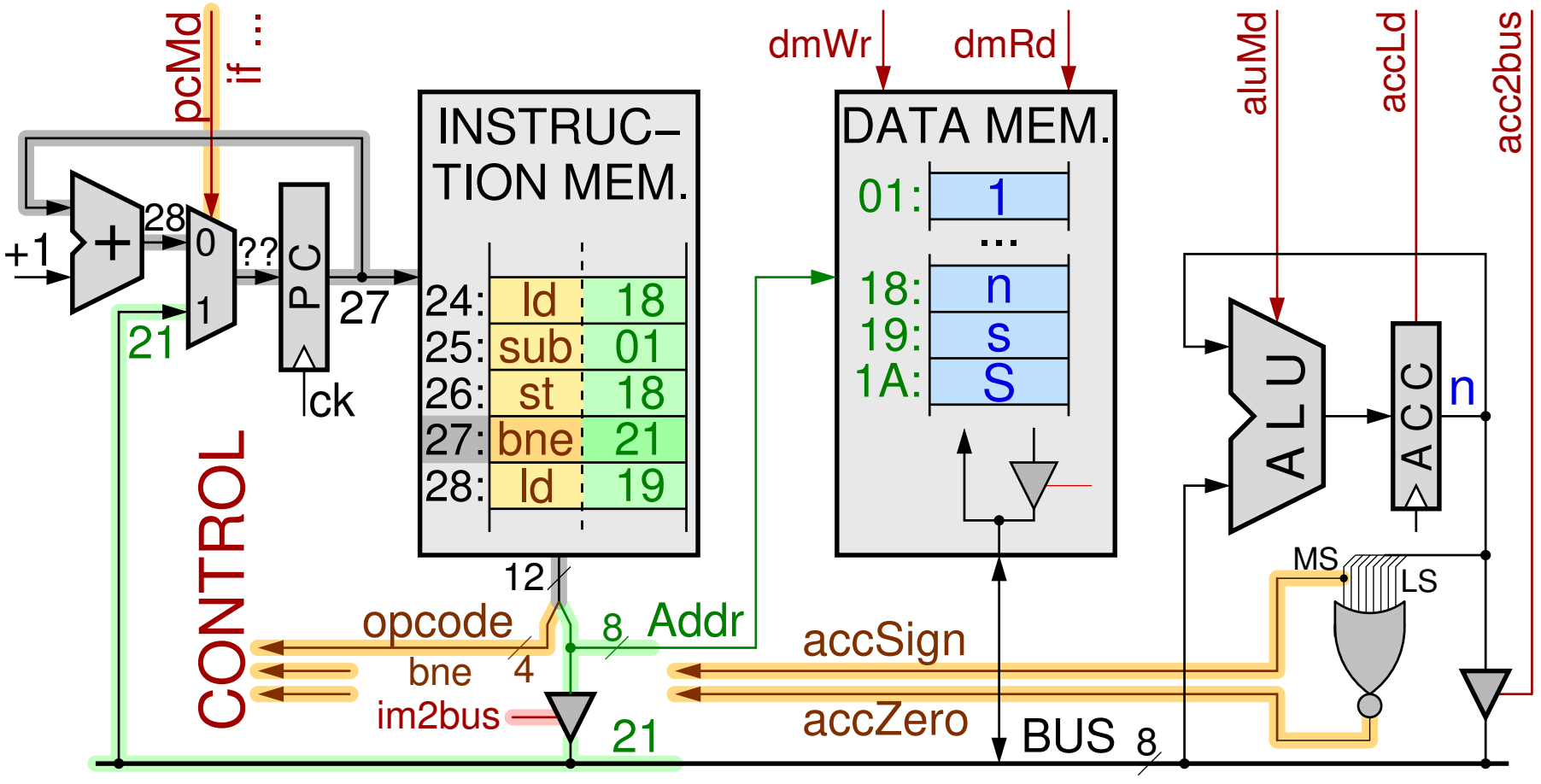
- *bne*: branch if not equal
- Εξυπακούεται: ACC, to zero
- Εάν ACC != 0, τότε επόμενη εντολή από 21
- Αλλιώς, επόμενη εντολή η «από κάτω»

Οι 4 εντολές διακλάδωσης του απλού υπολογιστή

- beq – branch if equal
  - if ACC == 0
- bne – branch if not equal
  - if ACC != 0
- blt – branch if less than
  - if ACC < 0
- bge – branch if greater or equal
  - if ACC ≥ 0

- Ομοίως και στους πραγματικούς επεξεργαστές, αλλά με δύο συγκρινόμενους τελεστέους σε καταχωρητές
- Σύγκριση ≤ δεν χρειάζεται: αντιμεταθέτουμε τους δύο τελεστέους στη σύγκριση ≥
- Ομοίως η > μέσω της <

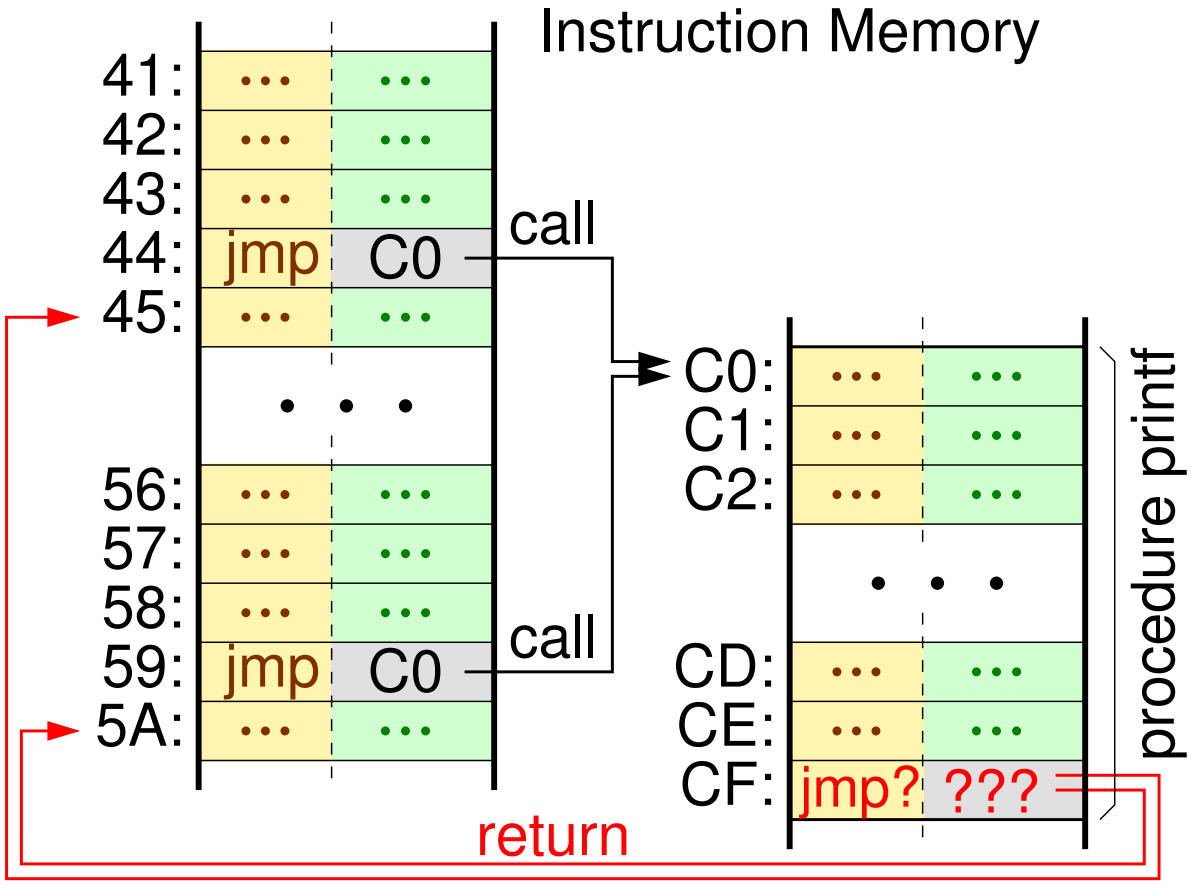
Τα σήματα Συνθήκης Δεδ. γιά τον έλεγχο Διακλάδωσης



Το κύκλωμα Ελέγχου γιά τις Διακλαδώσεις

opcode:		accZero:	accSign:	Λειτουργία:	dmRd	aluMd	accLd	acc2bus	ext2bus	dmWr	im2bus	pcMd
0000	(add)	x	x	ACC:=ACC+DM[A]	1	000	1	0	0	0	0	0
0001	(sub)	x	x	ACC:=ACC-DM[A]	1	001	1	0	0	0	0	0
0010	(and)	x	x	ACC:=ACCandDM[A]	1	010	1	0	0	0	0	0
0011	(nor)	x	x	ACC:=ACCnorDM[A]	1	011	1	0	0	0	0	0
0100	(inp)	x	x	DM[A]:=IO_BUS	0	xxx	0	0	1	1	0	0
0101	(ld)	x	x	ACC:=DM[A]	1	1xx	1	0	0	0	0	0
0110	(st)	x	x	DM[A]:=ACC	0	xxx	0	1	0	1	0	0
0111	(jmp)	x	x	PC := A	0	xxx	0	0	0	0	1	1
1111	(jmpx)	x	x	PC := DM[A]	1	xxx	0	0	0	0	0	1
1000	(beq)	0	x	PC := PC+1	0	xxx	0	0	0	0	x	0
1000		1	x	PC := A	0	xxx	0	0	0	0	1	1
1001	(bne)	0	x	PC := A	0	xxx	0	0	0	0	1	1
1001		1	x	PC := PC+1	0	xxx	0	0	0	0	x	0
1010	(blt)	x	0	PC := PC+1	0	xxx	0	0	0	0	x	0
1010		x	1	PC := A	0	xxx	0	0	0	0	1	1
1011	(bge)	x	0	PC := A	0	xxx	0	0	0	0	1	1
1011		x	1	PC := PC+1	0	xxx	0	0	0	0	x	0

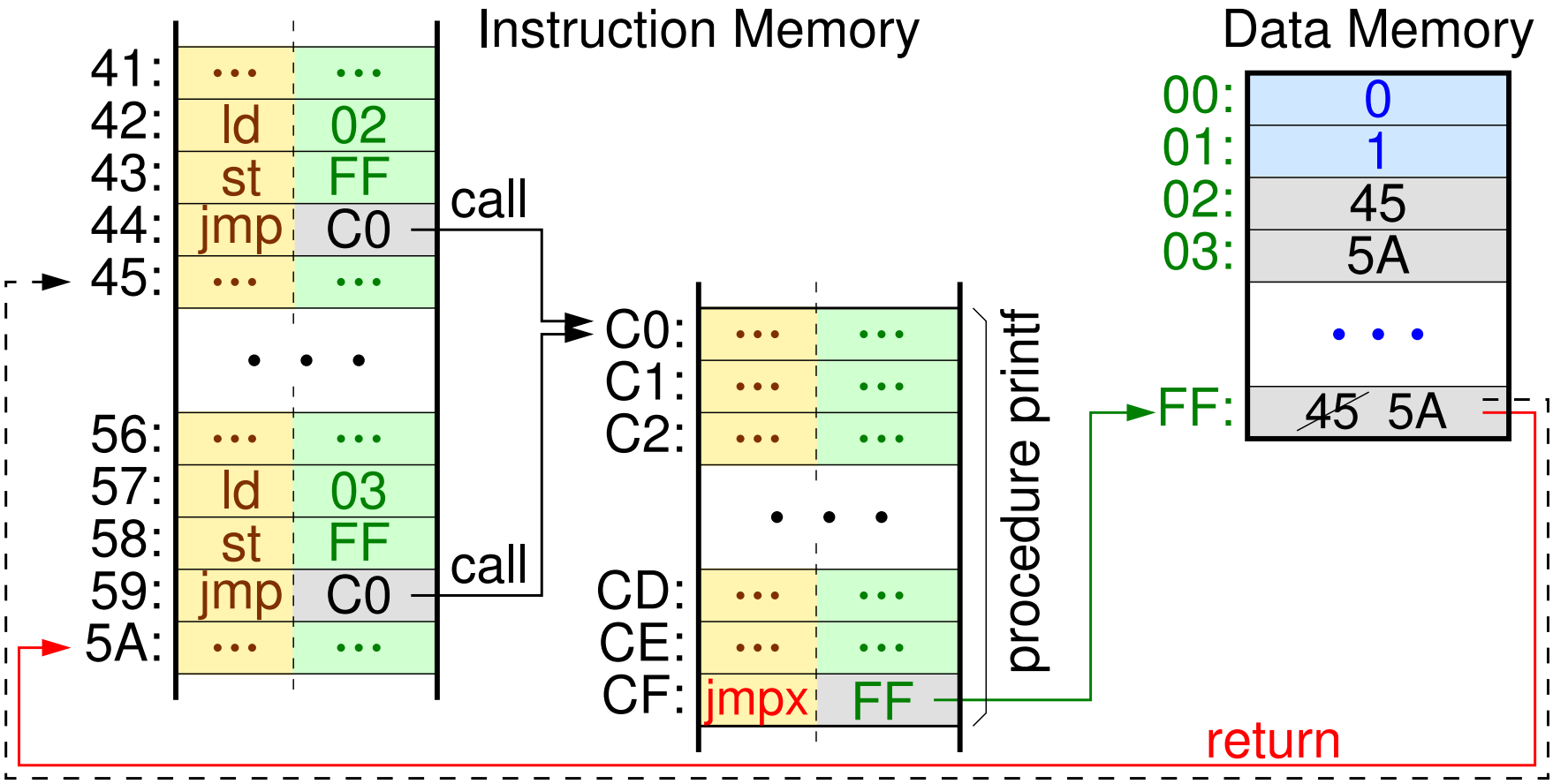
Κάλεσμα Διαδικασιών: Πώς επιστρέφουμε;



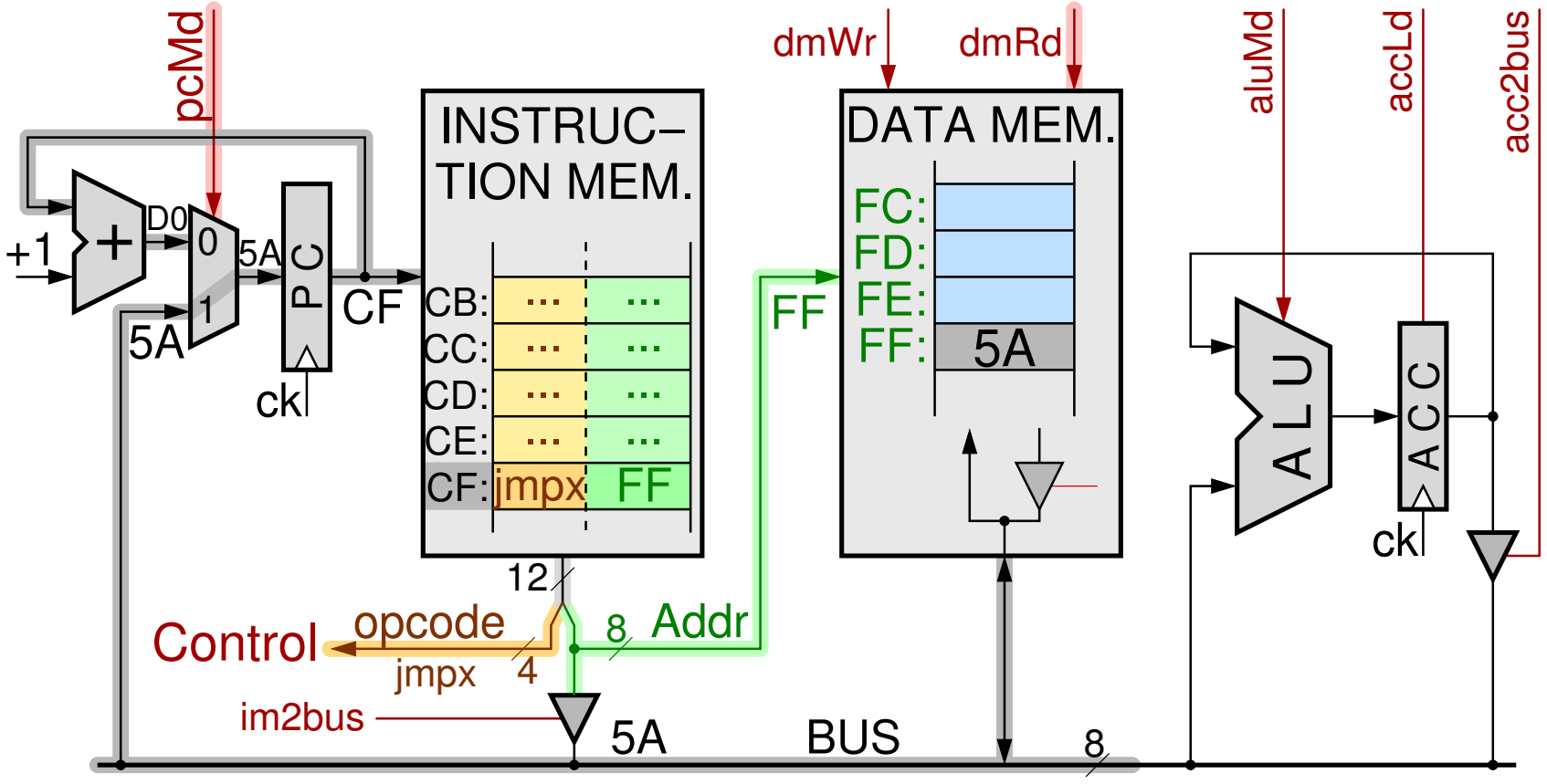
- Το πεδίο Addr μέσα στις εντολές είναι *σταθερό*
- Δεν μπορεί να αλλάζει κάθε φορά που η διαδικασία επιστρέφει

⇒ Πρέπει η διεύθυνση επιστροφής να έρχεται από Data Mem.

Jump Indexed: Εμμέσως μέσω Data Memory



Υλοποίηση της *Jumpx*



- Όπως η απλή *Jump*, αλλά ανάβει το *dmRd* αντί του *Im2bus*

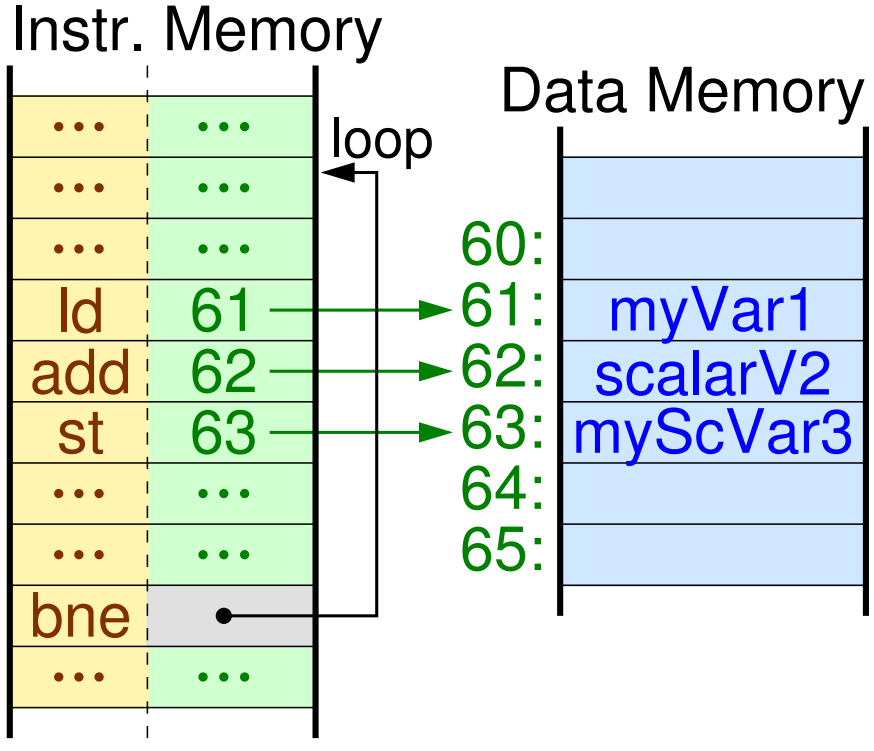
Πού να γράφουμε τη Διεύθυνση Επιστροφής;

- Εάν πάντα στην ίδια θέση (π.χ. FF), τότε η πρώτη διαδικασία που καλούμε δεν μπορεί να καλέσει δεύτερη
- Η κάθε διαδικασία σε «δική της» θέση;
  - όχι, γιά δύο λόγους:
  - υπάρχουν πολλές διαδικασίες «στατικά» ορισμένες, αλλά λίγες από αυτές είναι «δυναμικά» ενεργές κάθε στιγμή
  - δεν θα μπορούσε μία διαδικασία να καλέσει τον εαυτό της «αναδρομικά»!...
- Θα μάθουμε σε άλλα μαθήματα: στην «Στοίβα»...

Σταθερές Διευθύνσεις ⇒ Βαθμωτές Μεταβλητές μόνον

- Μη αυτομεταβαλόμενα προγράμματα
- Σταθερές οι Διευθύνσεις μέσα στις εντολές

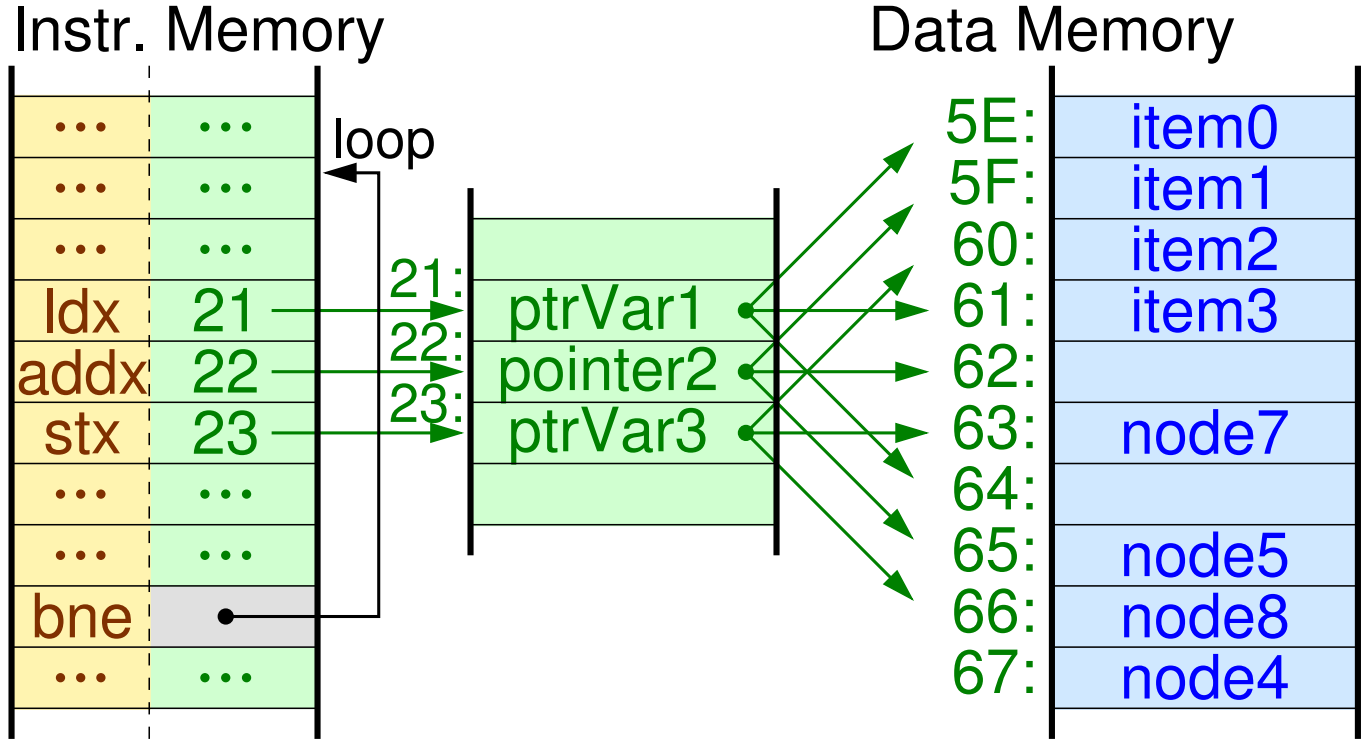
⇒ Όλες οι επαναλήψεις των βρόχων προσπελά-ζουν πάντα τις ίδιες (*βαθμωτές*) μεταβλητές με τις μέχρι τώρα εντολές



- Γιά την επεξεργασία στοιχείων Δομών Δεδομένων απαιτούνται *μεταβλητές* Διευθύνσεις

Έμμεση Πρόσβαση γιά μεταβλητές Διευθύνσεις

Νέες εντολές "indexed": προσπε-λάζουν τα δεδομένα εμμέσως, μέσω μετα-βλητών δεί-κτη (pointer)

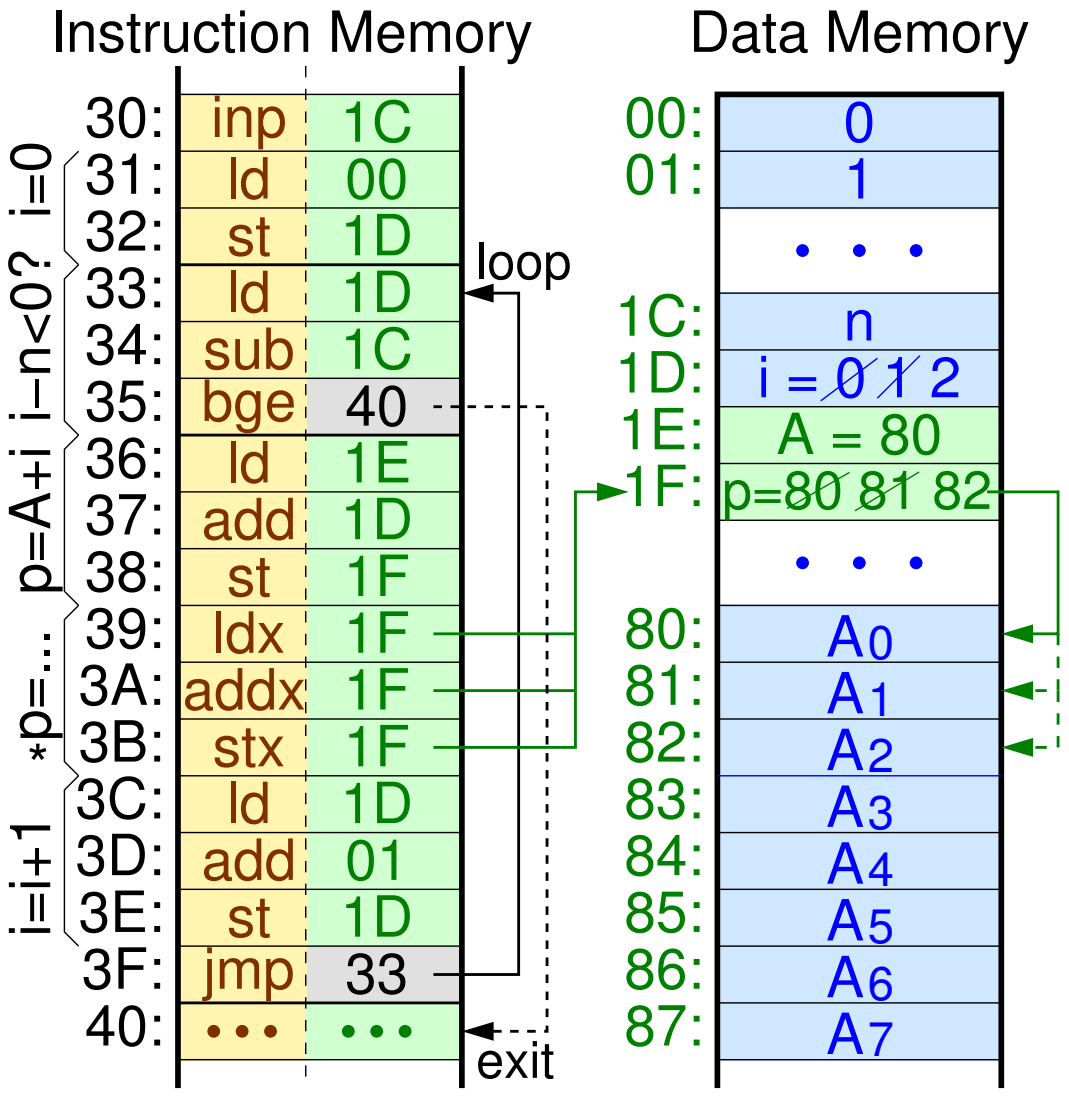


Ένας φημισμένος αφορισμός από τον David Wheeler ισχυρίζεται ότι: "All problems in computer science can be solved by another level of indirection". Μερικοί συμπληρώνουν σκωπτικά: "...except for the problem of too many layers of indirection".

Βρόχος επεξεργασίας Πίνακα

```
n=input();  i=0;
while ( i<n ) {
    A[i] = 2*A[i];
    i = i+1;
}
```

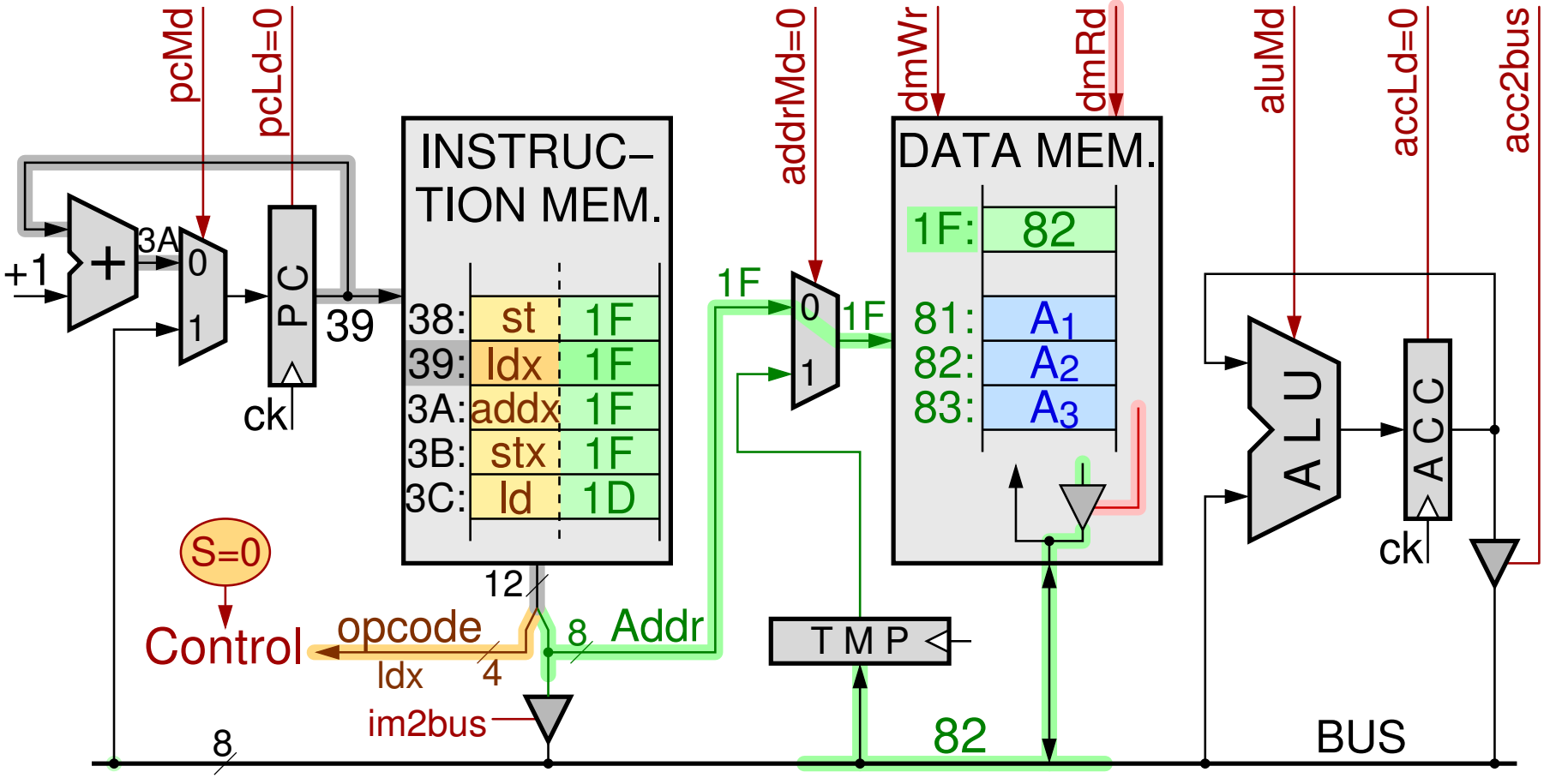
- p = διεύθυνση του A[i]
  = διευθ. του A[0] + i
- Εντολές Indexed:
  «πήγαινε να ρωτήσεις να σου πούν πού είναι η μεταβλητή που θέλω»



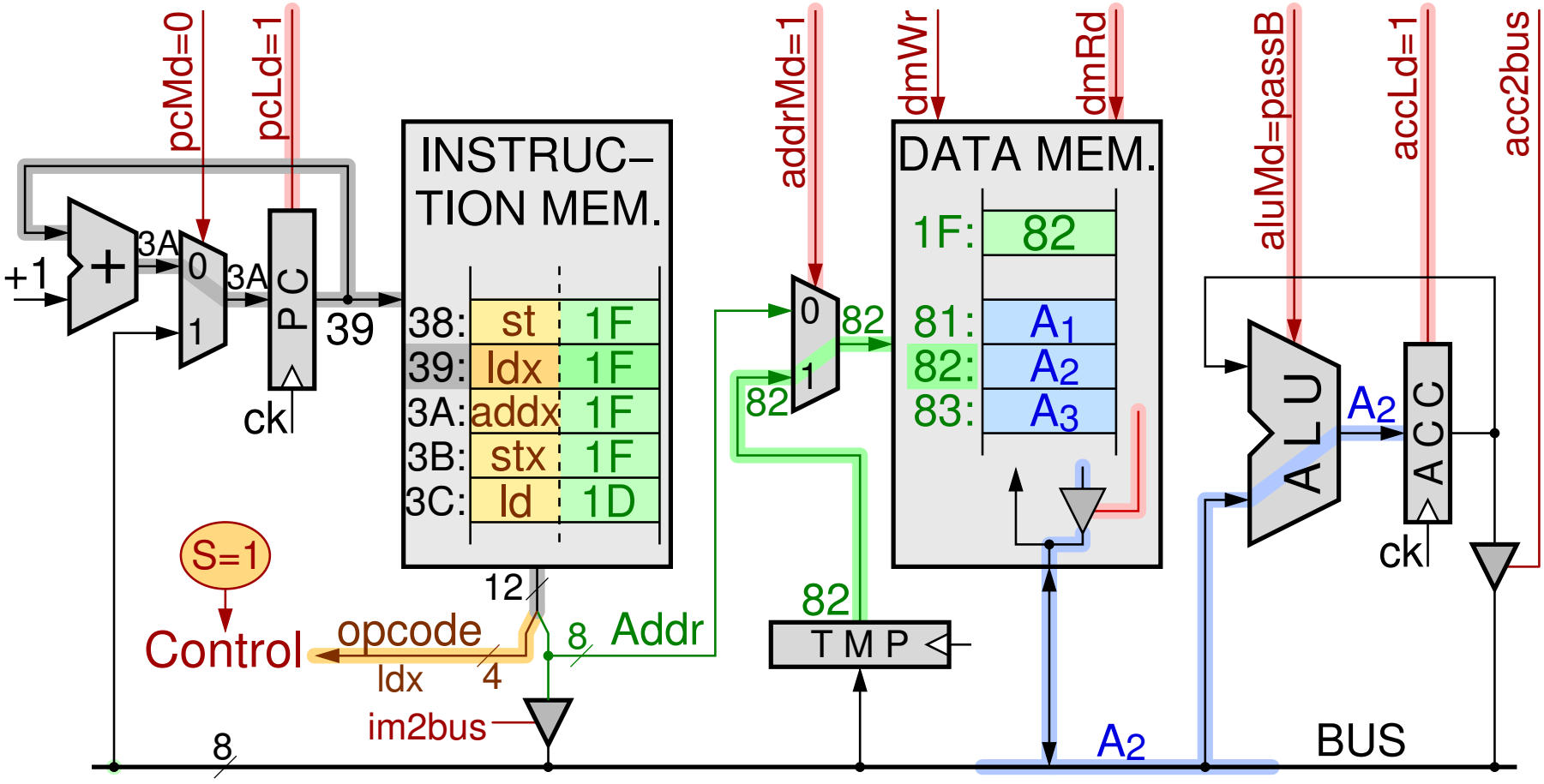
Οιαδήποτε Δομή Δεδομένων μέσω loadx & storex

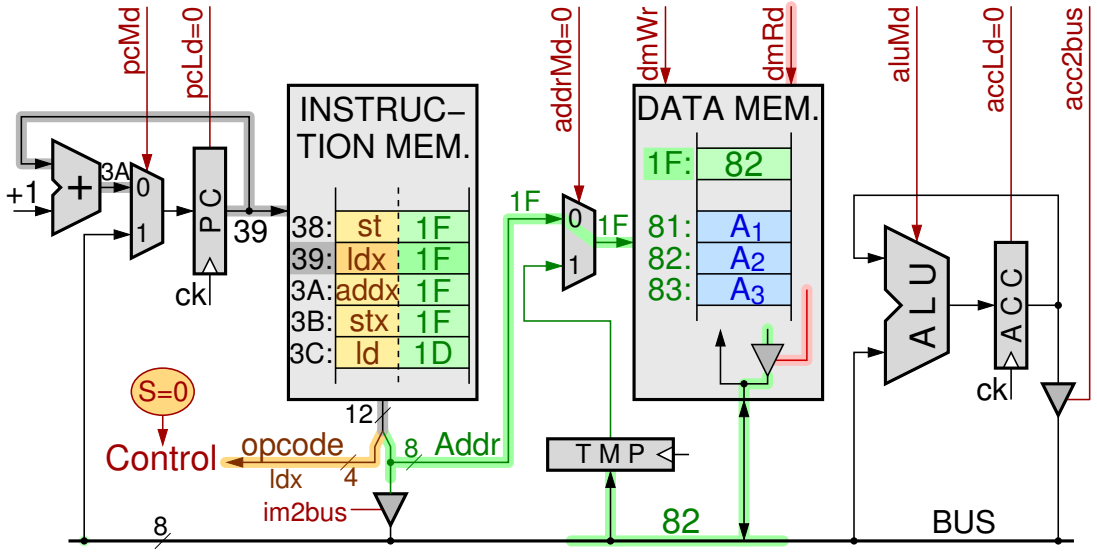
- Η load-indexed είναι απαραίτητη γιά να διαβάζουμε στοιχεία δομών και να τα φέρνουμε σε «προσωρινές» θέσεις
- Η store-indexed είναι απαραίτητη γιά να γράφουμε αποτελέσματα από «προσωρινές» θέσεις σε στοιχεία δομών
- Οι πράξεις μπορούν και να γίνονται στις «προσωρινές» θέσεις, εάν δεν έχουμε add-indexed κλπ. γιά κάθε πράξη
- *Οιαδήποτε* Δομή Δεδομένων υλοπ. με τις loadx και storex:
  - πρώτα κάνουμε σε «προσωρινές» θέσεις τον οιοδήποτε υπολογισμό διεύθυνσης, οιουδήποτε στοιχείου δομής
  - με αυτήν: ldx/stx από/σε δομή ↔ «προσωρινές» θέσεις

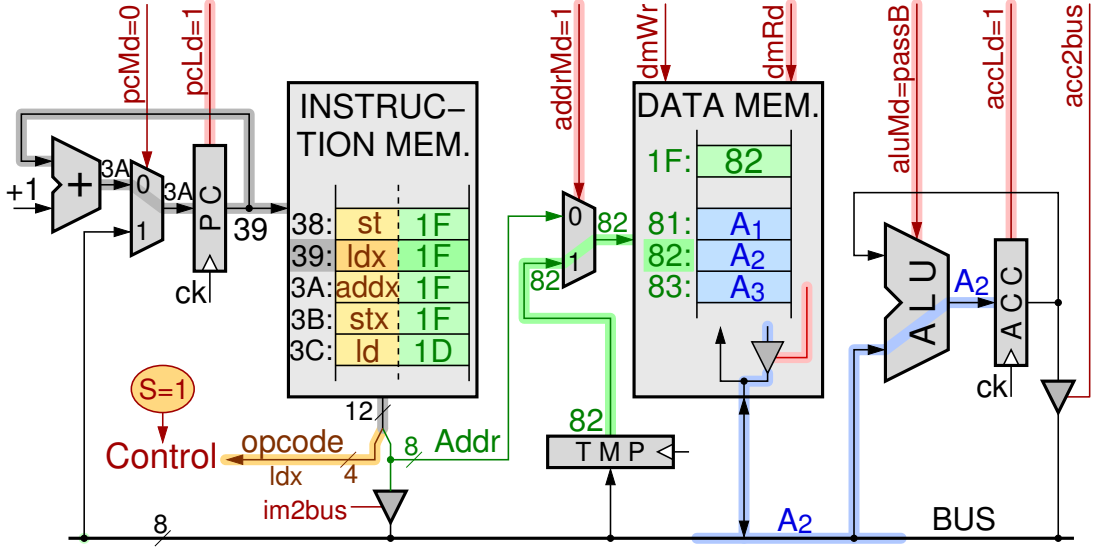
Υλοποίηση εντολής Indexed: Πρώτος Κύκλος



Υλοποίηση εντολής Indexed: Δεύτερος Κύκλος





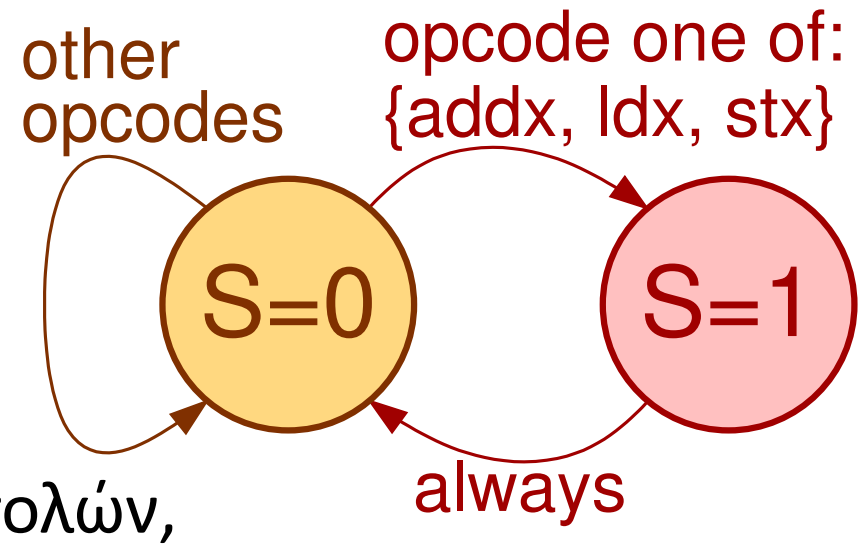


Οι δύο κύκλοι συγκριτικά

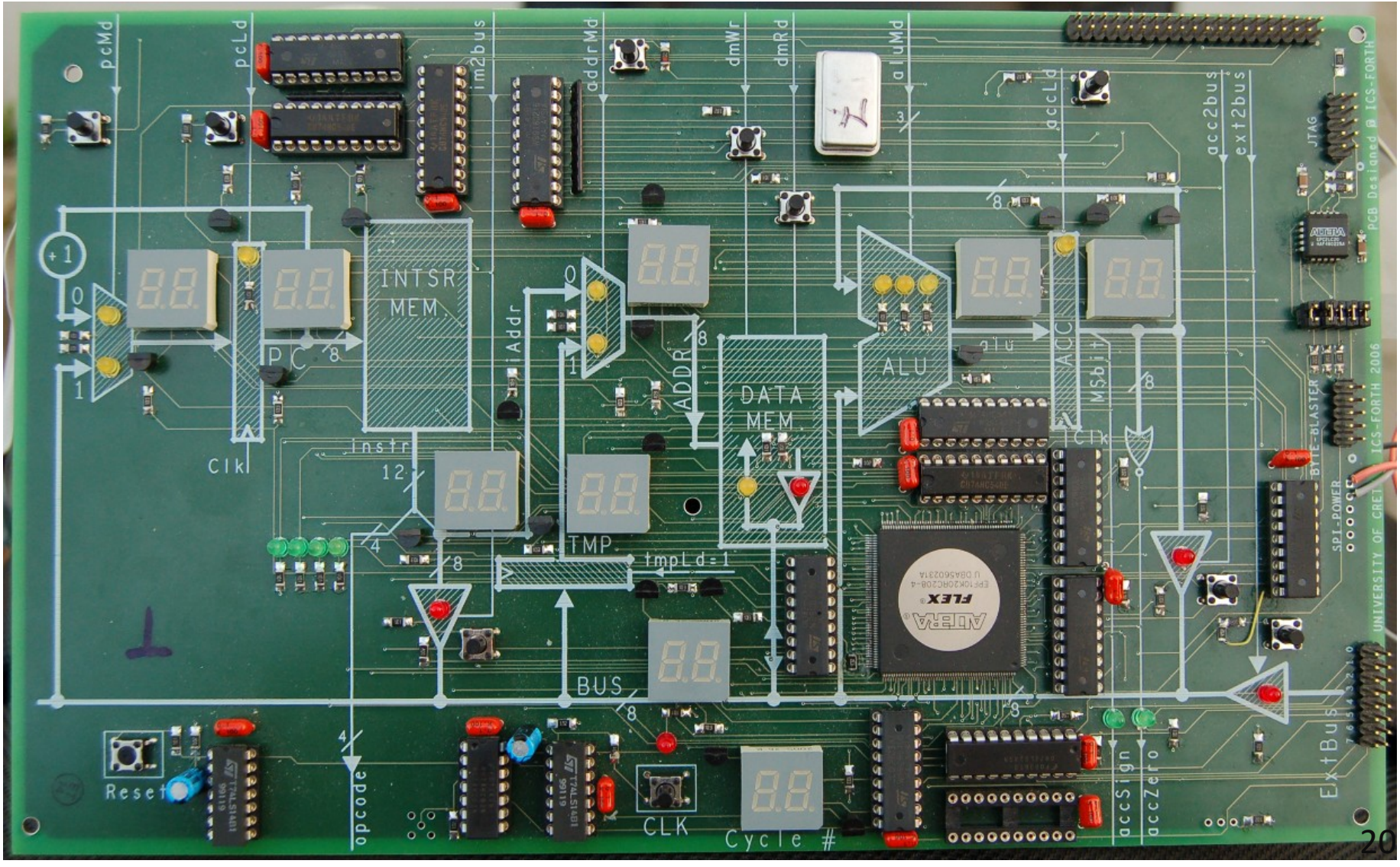
- Δύο προσπελάσεις στην Data Mem. ⇒ δύο κύκλοι
- Ακολουθ. Έλεγχος (FSM)
- Πρώτος κύκλος (S=0):
  - pcLd=0: εντολή δεν τελείωσε
  - AccLd=0: όχι ακόμα τα σωστά data, δεν χαλάμε τα παλαιά
  - addrMd=0: ανάγνωση του pointer από διεύθ. εντολής
- Δεύτερος κύκλος (S=1):
  - pcLd=1: εντολή τελειώνει
  - AccLd=1: σωστά νέα data
  - TMP=pointer (BUS προηγ. κ.)
  - addrMd=1: ανάγν. data

1ος – 2ος κύκλος εντολών Indexed: η FSM Ελέγχου

- Ο Έλεγχος είναι πλέον ακολουθιακό κύκλωμα, όχι συνδυαστικό όπως πριν
- S=0 είναι η κατάσταση (ο κύκλος ρολογιού) εκτέλεσης όλων των προηγουμένων εντολών, καθώς και ο πρώτος κύκλος εκτέλεσης των νέων
- S=1 είναι ο δεύτερος (και τελευταίος) κύκλος εκτέλεσης των νέων εντολών Indexed load/store & αριθμητικών



opcode:	S:	Λειτουργία:	addrMd nS:		dmRd	aluMd	acc2bus accLd		dmWr ext2bus		pcMd im2bus		pcLd
0000 (add)	x	ACC:=ACC+DM[A]	0	0	1	000	1	0	0	0	0	0	1
0001 (sub)	x	ACC:=ACC-DM[A]	0	0	1	001	1	0	0	0	0	0	1
0010 (and)	x	ACC:=ACCandDM[A]	0	0	1	010	1	0	0	0	0	0	1
0011 (nor)	x	ACC:=ACCnorDM[A]	0	0	1	011	1	0	0	0	0	0	1
0100 (inp)	x	DM[A]:=IO_BUS	0	0	0	xxx	0	0	1	1	0	0	1
0101 (ld)	x	ACC:=DM[A]	0	0	1	1xx	1	0	0	0	0	0	1
0110 (st)	x	DM[A]:=ACC	0	0	0	xxx	0	1	0	1	0	0	1
0111 (jmp)	x	PC := A	0	0	0	xxx	0	0	0	0	1	1	1
1000 (beq)	x	PC := A or PC+1	0	0	0	xxx	0	0	0	0	1	(*)	1
1001 (bne)	x	PC := A or PC+1	0	0	0	xxx	0	0	0	0	1	(*)	1
1010 (blt)	x	PC := A or PC+1	0	0	0	xxx	0	0	0	0	1	(*)	1
1011 (bge)	x	PC := A or PC+1	0	0	0	xxx	0	0	0	0	1	(*)	1
1100 (addx)	0	TMP:=DM[A]	1	0	1	xxx	0	0	0	0	0	x	0
1100	1	ACC:=ACC+DM[TMP]	0	1	1	000	1	0	0	0	0	0	1
1101 (ldx)	0	TMP:=DM[A]	1	0	1	xxx	0	0	0	0	0	x	0
1101	1	ACC:= DM[TMP]	0	1	1	1xx	1	0	0	0	0	0	1
1110 (stx)	0	TMP:=DM[A]	1	0	1	xxx	0	0	0	0	0	x	0
1110	1	DM[TMP]:=ACC	0	1	0	xxx	0	1	0	1	0	0	1
1111 (jmpx)	x	PC := DM[A]	0	0	1	xxx	0	0	0	0	0	1	1


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

Το πλήρες Datapath και σήματα ελέγχου