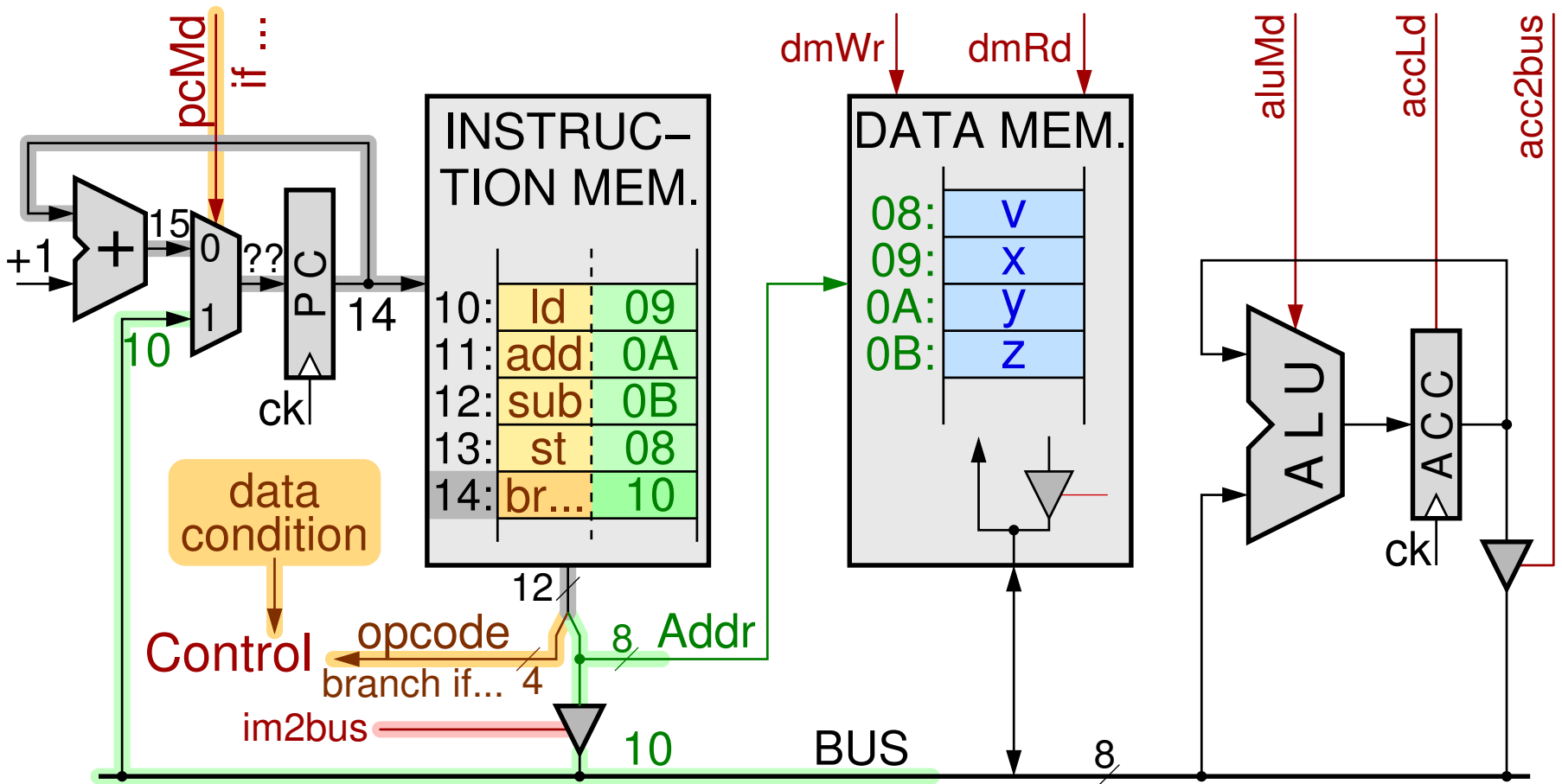


Απλός Υπολογιστής: Διακλαδώσεις και Έμμεσες Προσπελάσεις (Pointers)

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

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



Απλός βρόχος: Άθροισμα $(n)+(n-1)+(n-2)+\dots+(2)+(1)$

```

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:

1E:	ld	00
1F:	st	19
20:	inp	18
21:	ld	19
22:	add	18
23:	st	19
24:	ld	18
25:	sub	01
26:	st	18
27:	bne	21
28:	ld	19
29:	st	1A
2A:	jmp	1E

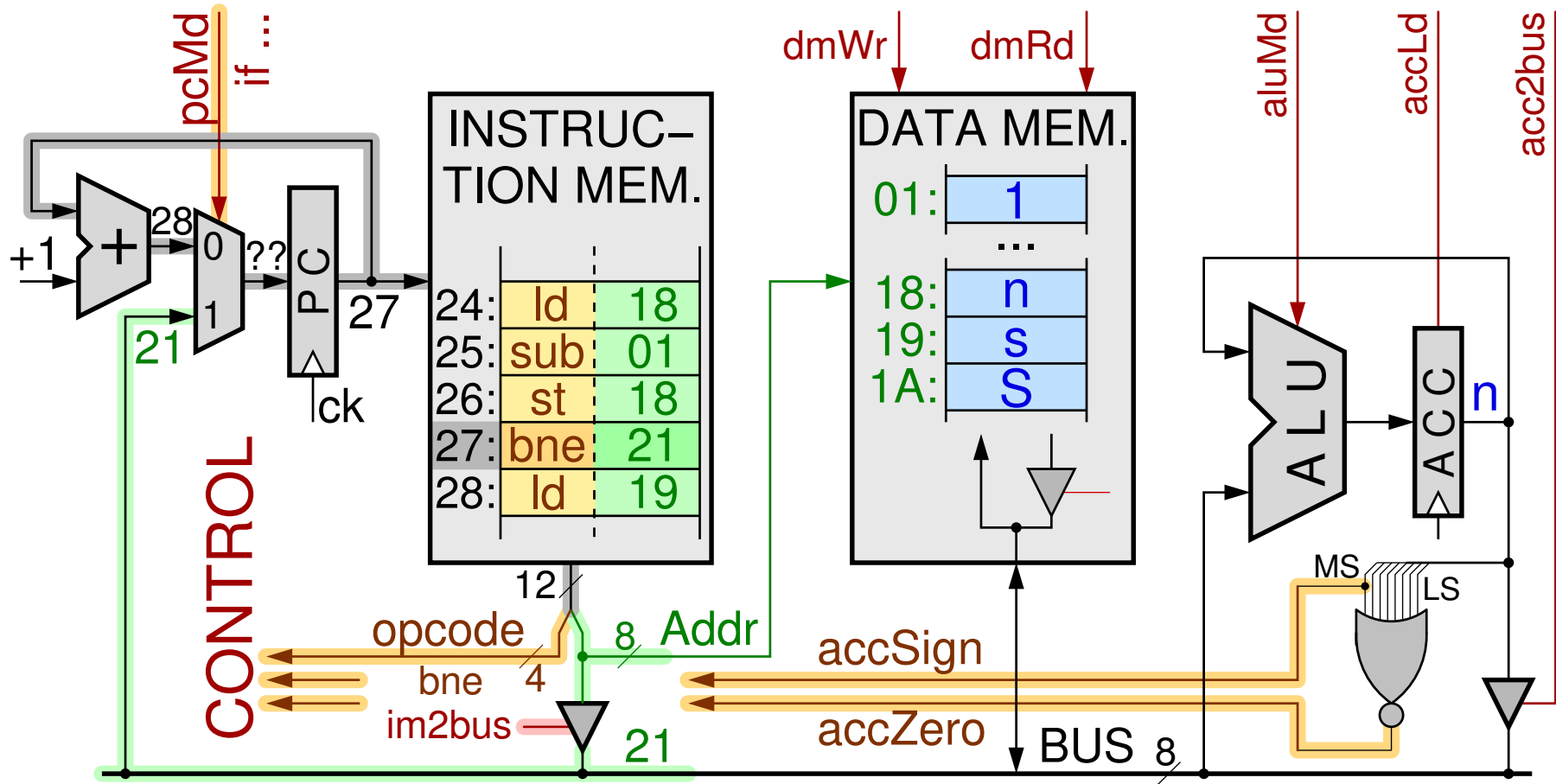
$n=n-1; s=s+n;$

- *bne*: branch if not equal
- Εξυπακούεται: ACC, to zero
- Εάν ACC $\neq 0$, 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 $ACC \geq 0$
- Ομοίως και στους πραγματικούς επεξεργαστές, αλλά με δύο συγκρινόμενους τελεστές σε καταχωρητές
- Σύγκριση \leq δεν χρειάζεται: αντιμετωθούμε τους δύο τελεστές στη σύγκριση \geq
- Ομοίως η $>$ μέσω της $<$

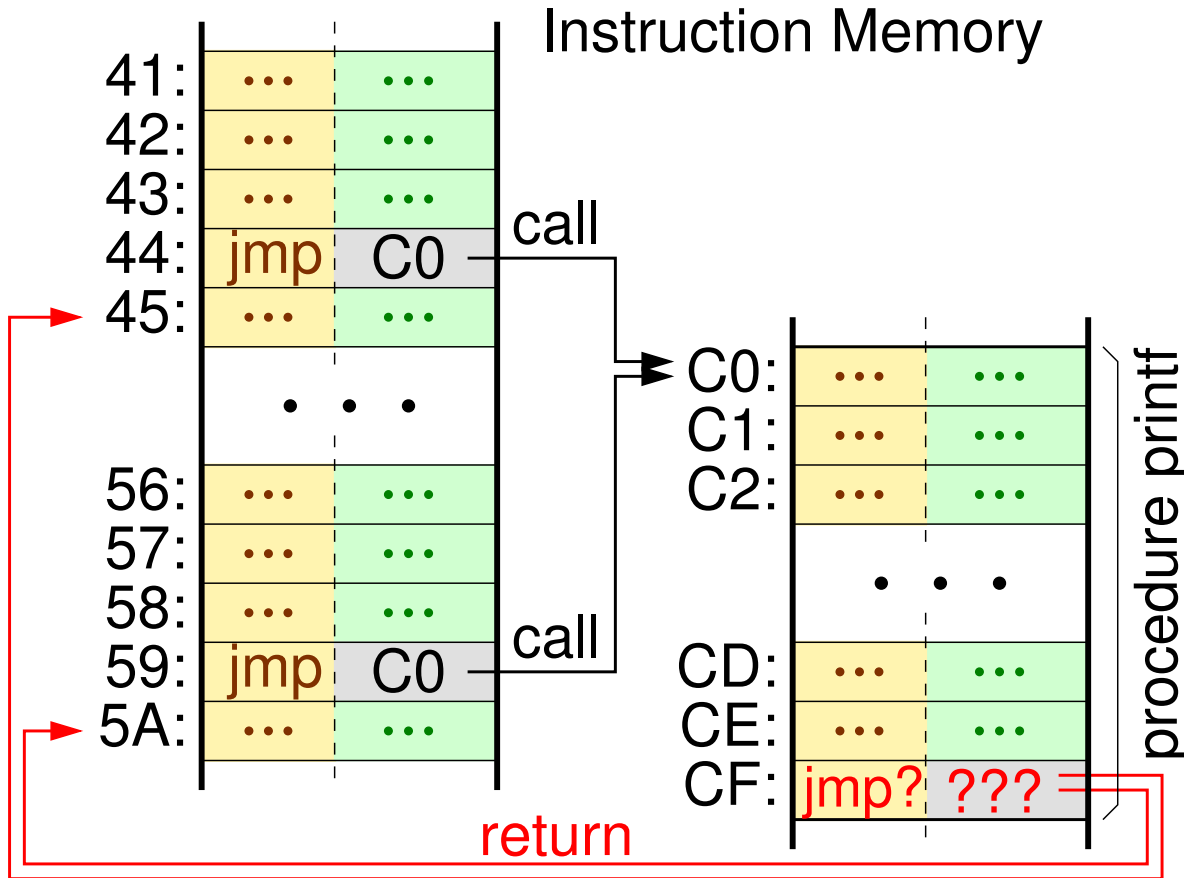
Τα σήματα Συνθήκης Δεδ. για τον έλεγχο Διακλάδωσης



Το κύκλωμα Ελέγχου για τις Διακλαδώσεις

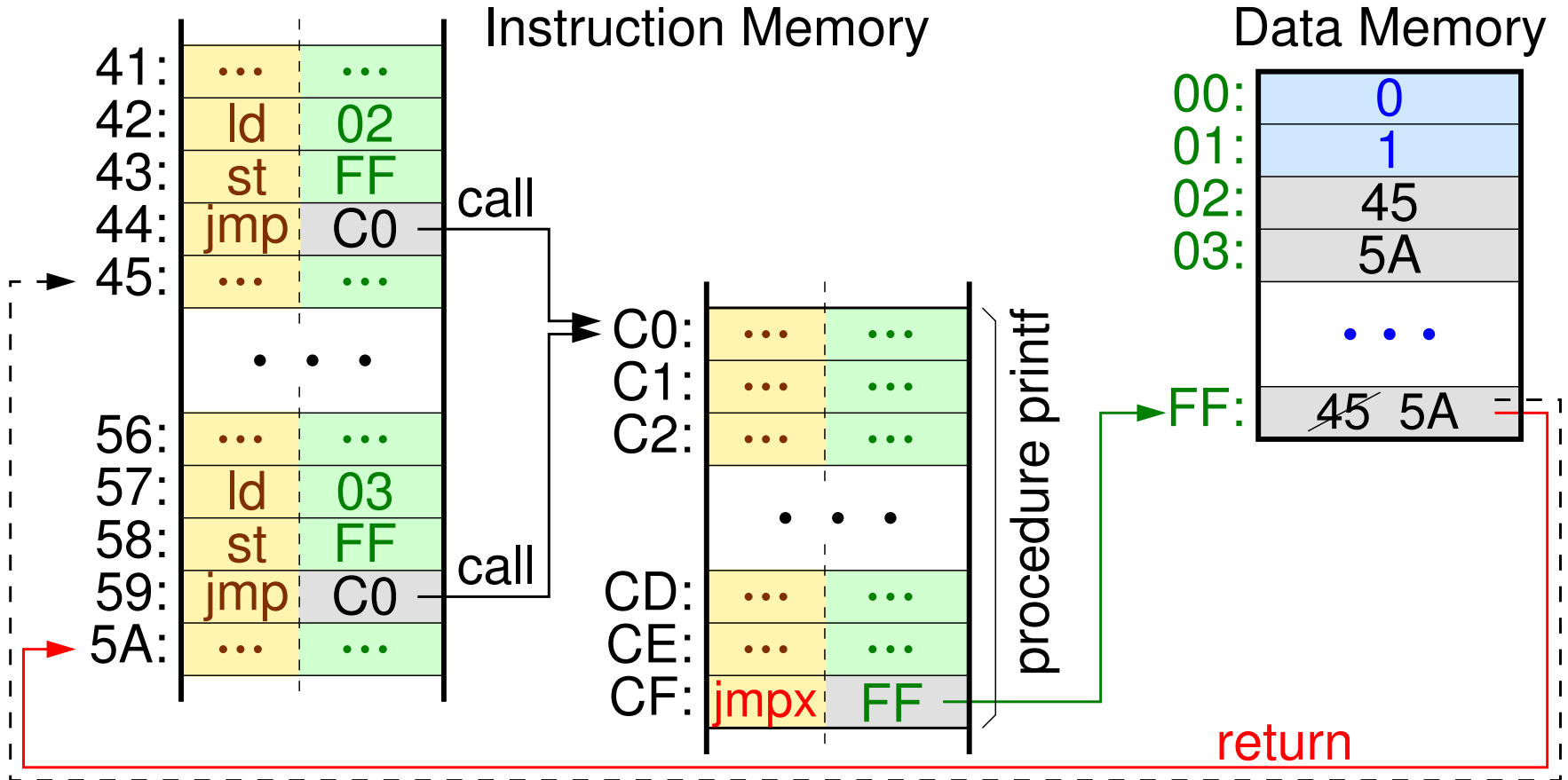
<i>accZero:</i>		<i>Λειτουργία:</i>		<i>aluMd</i>	<i>acc2bus</i>	<i>dmWr</i>	<i>pcMd</i>
<i>opcode:</i>	<i>accSign:</i>			<i>dmRd</i>	<i>accLd</i>	<i>ext2bus</i>	<i>im2bus</i>
0000 (add)	x x	ACC := ACC + DM[A]	1	000	1	0	0
0001 (sub)	x x	ACC := ACC - DM[A]	1	001	1	0	0
0010 (and)	x x	ACC := ACC and DM[A]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

Κάλεσμα Διαδικασιών: Πώς επιστρέφουμε;

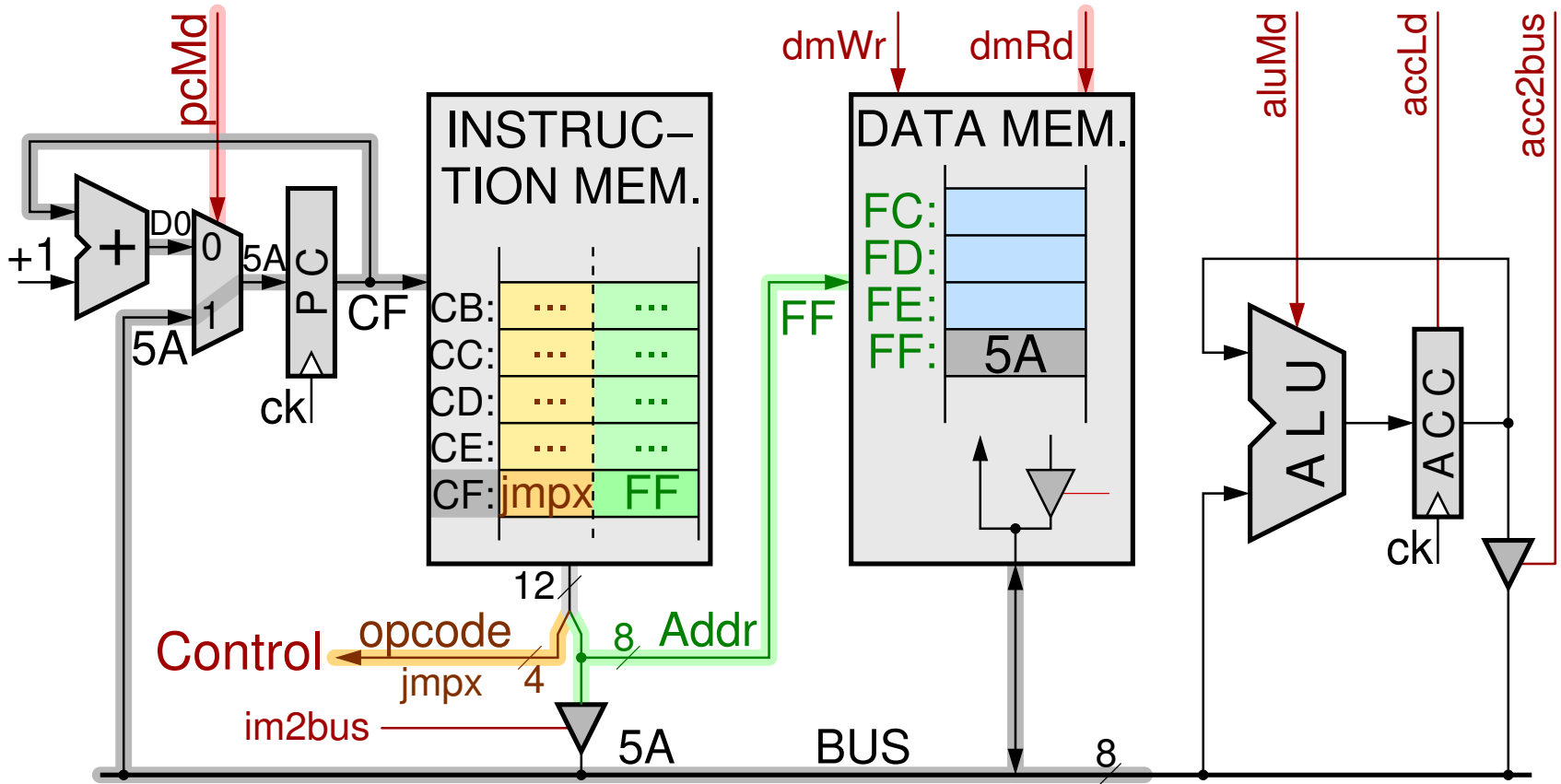


- Το πεδίο Addr μέσα στις εντολές είναι σταθερό
 - Δεν μπορεί να αλλάζει κάθε φορά που η διαδικασία επιστρέφει
- ⇒ Πρέπει η διεύθυνση επιστροφής να έρχεται από Data Mem.

Jump Indexed: Εμμέσως μέσω Data Memory



Υλοποίηση της *Jmpx*



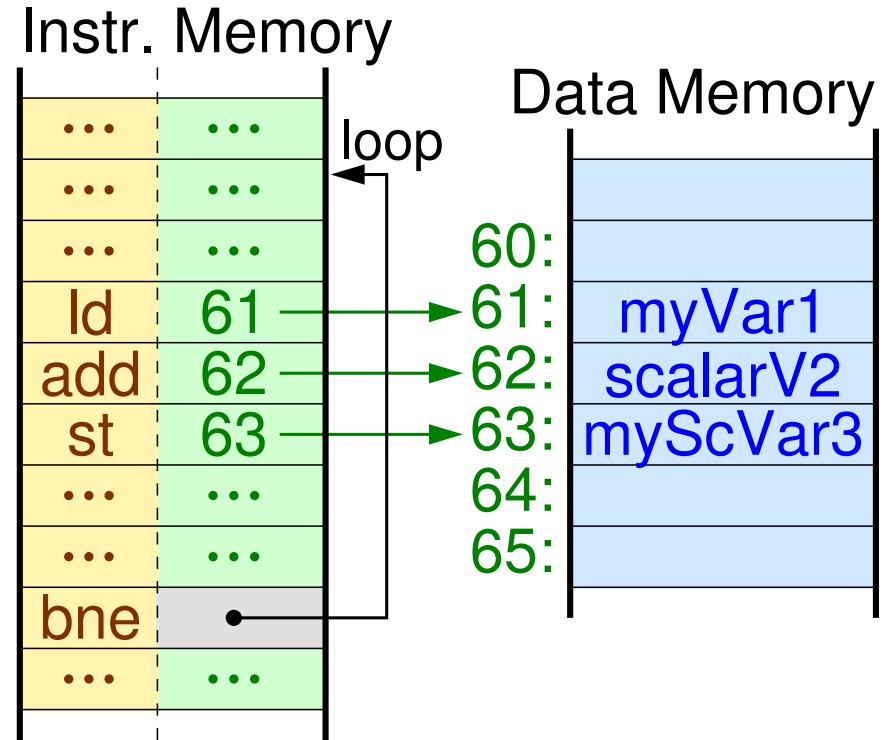
- Όπως η απλή *Jmp*, αλλά ανάβει το *dmRd* αντί του *Im2bus*

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, για 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 «Στοιίβα»...

Σταθερές Διευθύνσεις \Rightarrow Βαθμωτές Μεταβλητές μόνον

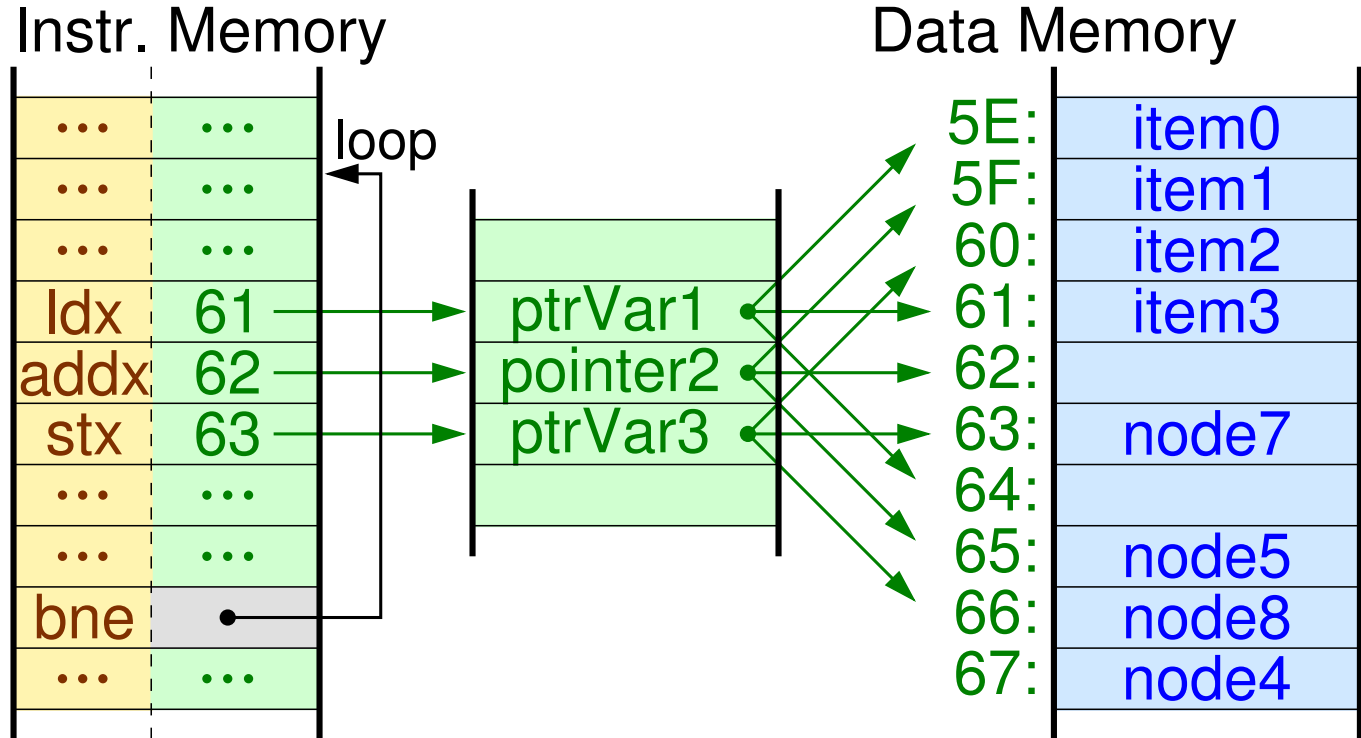
- Μη αυτομεταβαλλόμενα προγράμματα
 - Σταθερές οι Διευθύνσεις μέσα στις εντολές
- \Rightarrow Όλες οι επαναλήψεις των βρόχων προσπελάζουν πάντα τις ίδιες (βαθμωτές) μεταβλητές με τις μέχρι τώρα εντολές



- Για την επεξεργασία στοιχείων Δομών Δεδομένων απαιτούνται μεταβλητές Διευθύνσεις

Έμμεση Πρόσβαση για 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»

Instruction Memory

30:	inp	1C	
31:	ld	00	
32:	st	1D	
33:	ld	1D	← loop
34:	sub	1C	
35:	bge	40	
36:	ld	1E	
37:	add	1D	
38:	st	1F	
39:	ldx	1F	
3A:	addx	1F	
3B:	stx	1F	
3C:	ld	1D	
3D:	add	01	
3E:	st	1D	
3F:	jmp	33	
40:	← exit

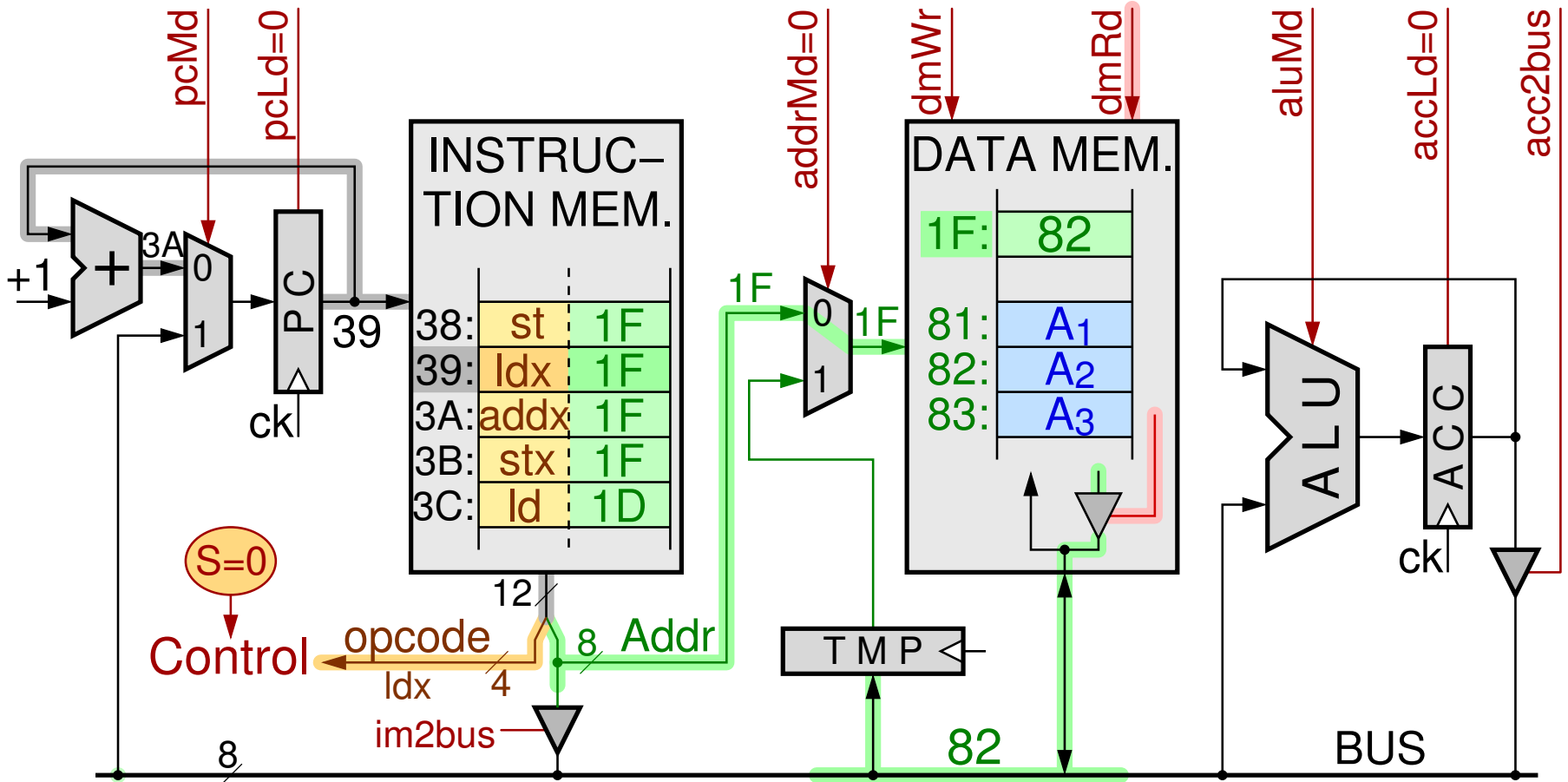
Data Memory

00:	0
01:	1
	...
1C:	n
1D:	i = 0 2
1E:	A = 80
1F:	p = 80 81 82
	...
80:	A ₀
81:	A ₁
82:	A ₂
83:	A ₃
84:	A ₄
85:	A ₅
86:	A ₆
87:	A ₇

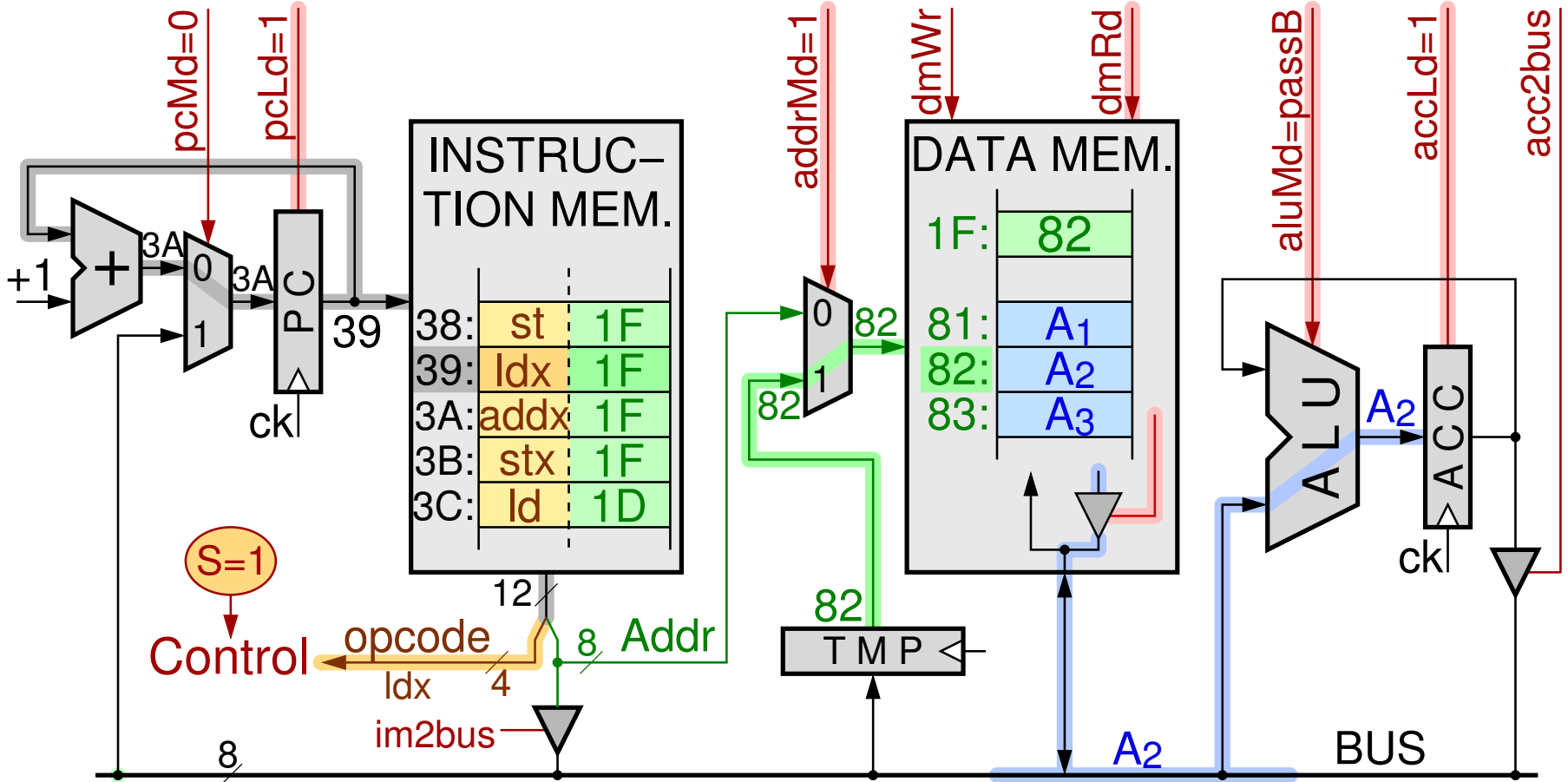
Οιαδήποτε Δομή Δεδομένων μέσω loadx & storex

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

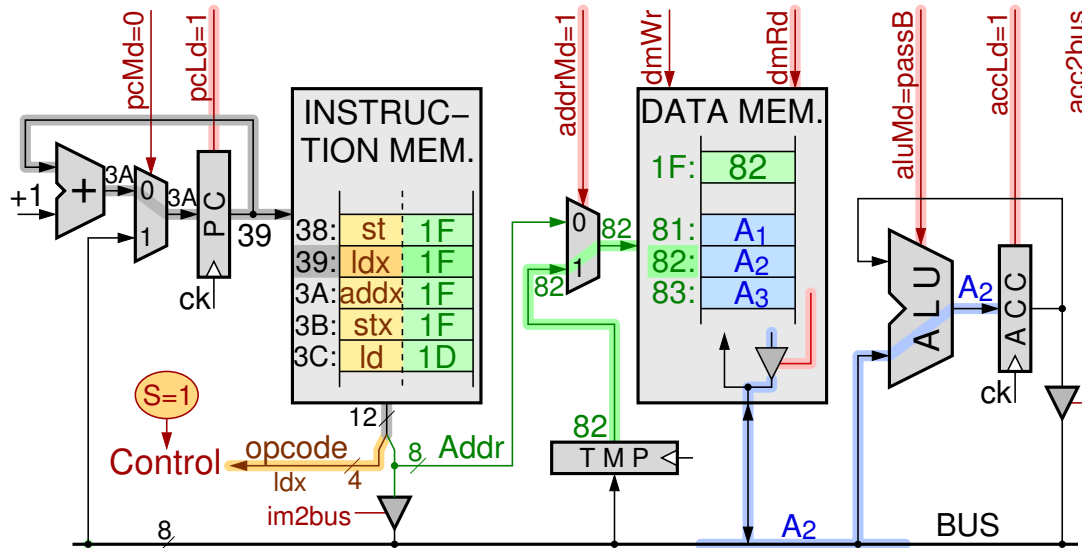
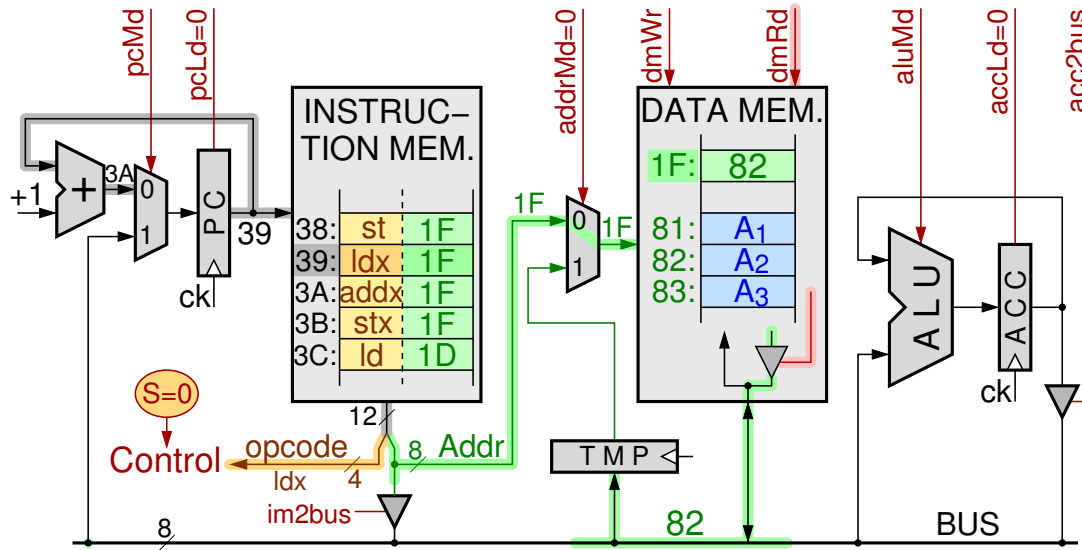
Υλοποίηση εντολής Indexed: Πρώτος Κύκλος



Υλοποίηση εντολής Indexed: Δεύτερος Κύκλος



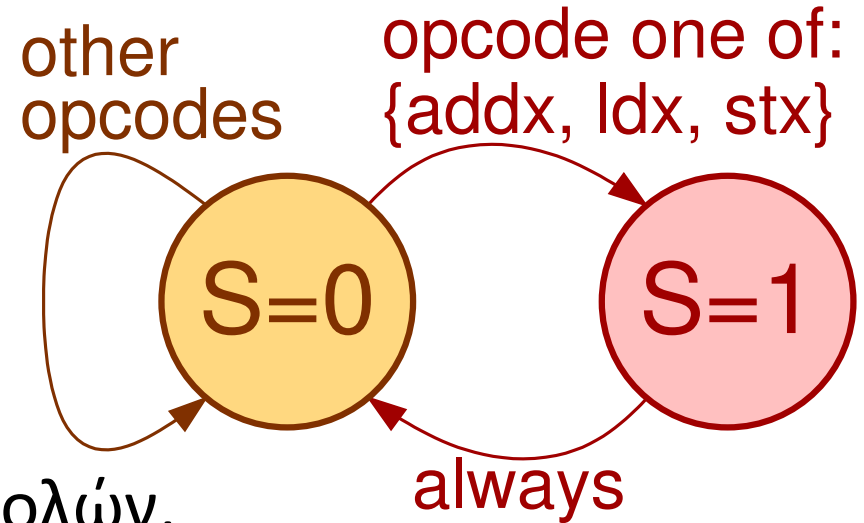
Οι δύο κύκλοι συγκριτικά



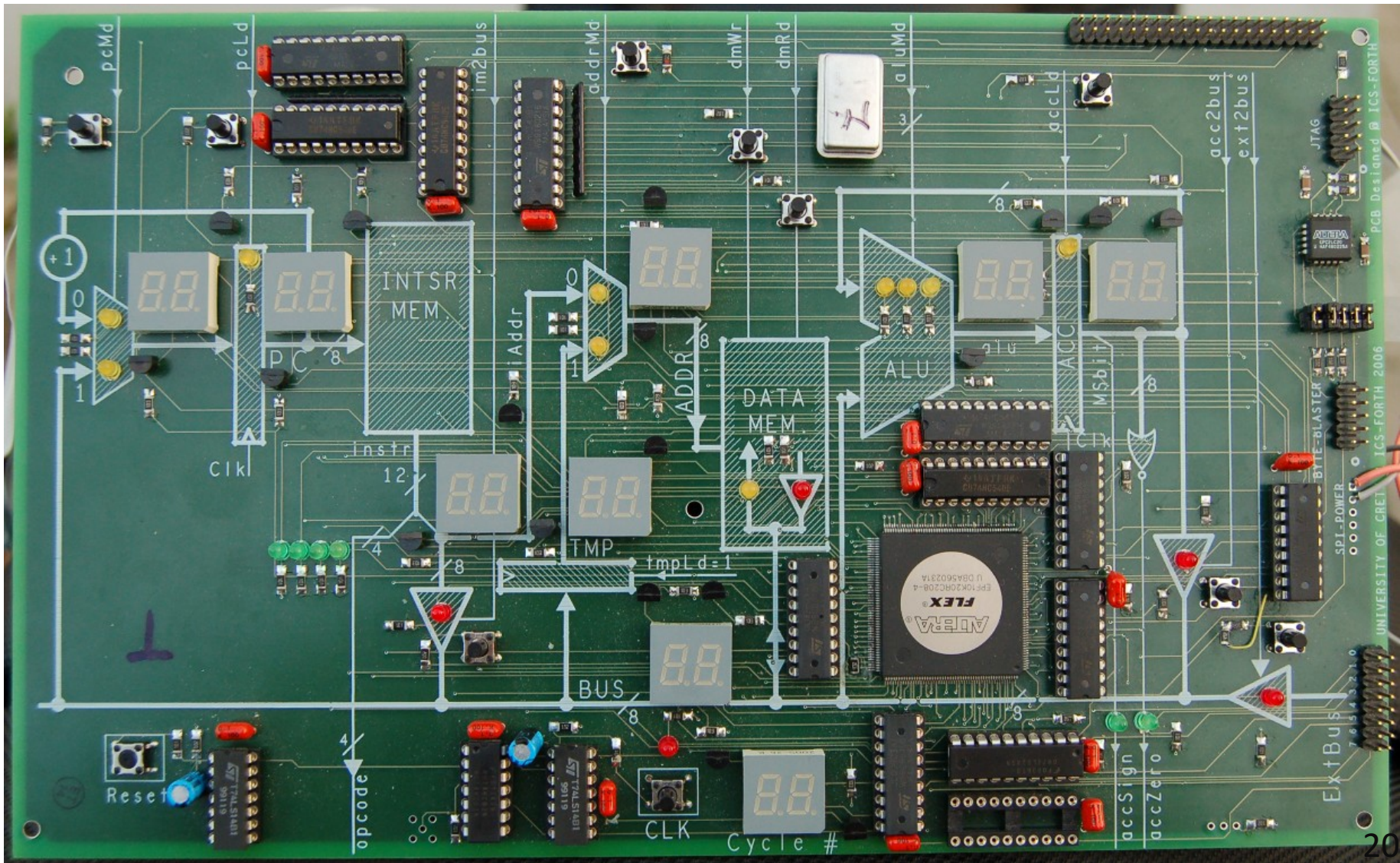
- Δύο προσπελάσεις στην Data Mem. \Rightarrow δύο κύκλοι
- Ακολουθ. Έλεγχος (FSM)
- Πρώτος κύκλος ($S=0$):
 - $pcLd=0$: εντολή δεν τελείωσε
 - $AccLd=0$: όχι ακόμα τα σωστά data, δεν χαλάμε τα παλαιά
 - $addrMd=0$: ανάγνωση του pointer από διεύθ. εντολής
- Πρώτος κύκλος ($S=0$):
 - $pcLd=1$: εντολή τελειώνει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

1^{ος} – 2^{ος} 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 προηγούμενων 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



<i>opcode:</i>		<i>Λειτουργία:</i>	<i>addrMd</i>	<i>aluMd</i>	<i>acc2bus</i>	<i>dmWr</i>	<i>pcMd</i>						
	<i>S:</i>		<i>nS:</i>	<i>dmRd</i>	<i>accLd</i>	<i>ext2bus</i>	<i>im2bus</i>	<i>pcLd</i>					
0000	(add)	x	ACC:=ACC+DM[A]	0	0	1	000	1	0	0	0	0	1
0001	(sub)	x	ACC:=ACC-DM[A]	0	0	1	001	1	0	0	0	0	1
0010	(and)	x	ACC:=ACCandDM[A]	0	0	1	010	1	0	0	0	0	1
0011	(nor)	x	ACC:=ACCnorDM[A]	0	0	1	011	1	0	0	0	0	1
0100	(inp)	x	DM[A]:=IO_BUS	0	0	0	xxx	0	0	1	1	0	1
0101	(ld)	x	ACC:=DM[A]	0	0	1	1xx	1	0	0	0	0	1
0110	(st)	x	DM[A]:=ACC	0	0	0	xxx	0	1	0	1	0	1
0111	(jmp)	x	PC := A	0	0	0	xxx	0	0	0	0	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



Το πλήρες Datapath και σήματα ελέγχου

