Άσκηση 12:
Ο Απλός Υπολογιστής στον Προσομοιωτή (μέρος Β')

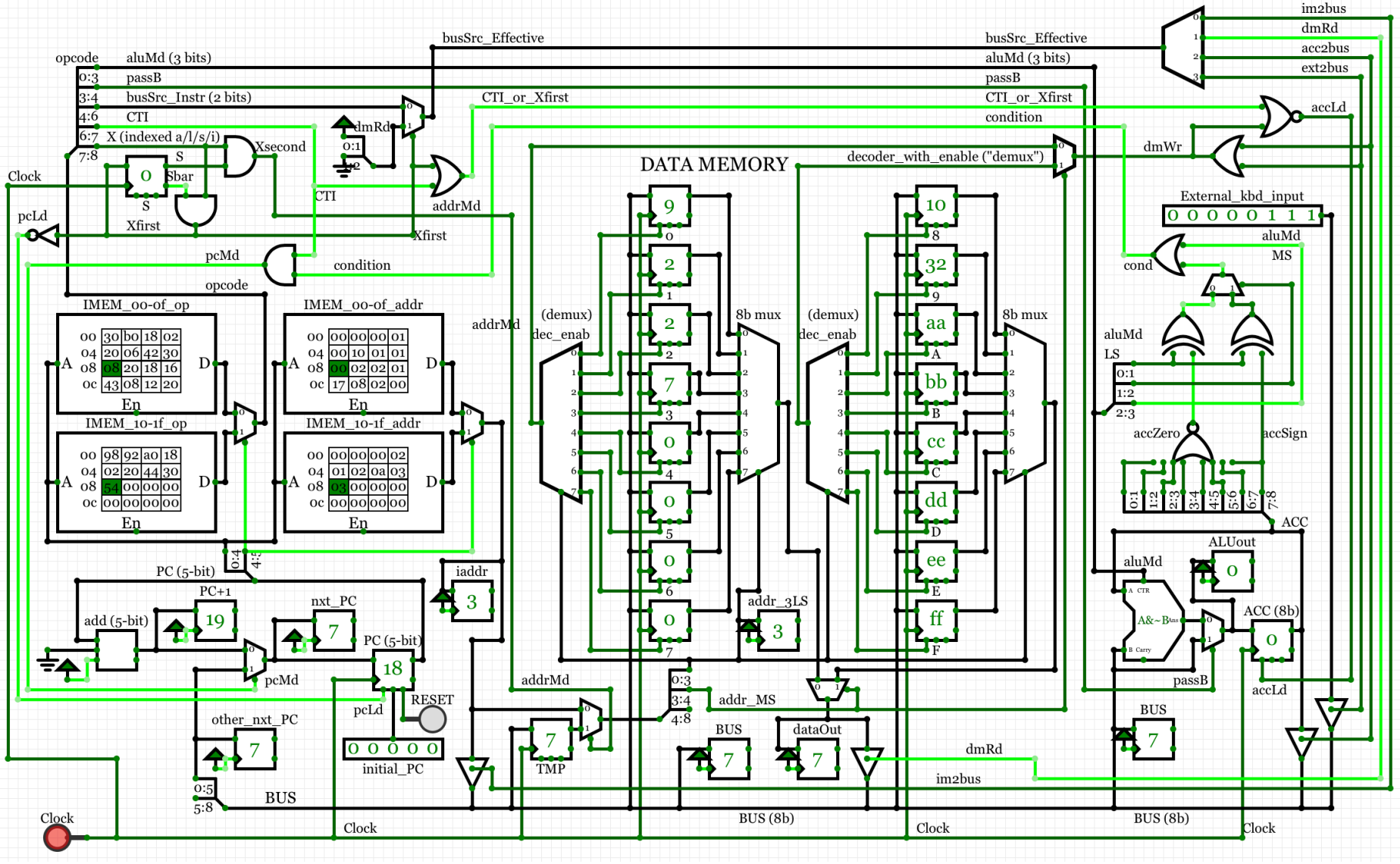
012c (Άσκηση 12) – 25 - 29 Ιαν. 2021 – Μανόλης Κατεβαίνης

Προοίμιο: Πρωτοβουλίες/παραλλαγές ευπρόσδεκτες

- Στην εκφώνηση αυτή παρουσιάζεται ένας δυνατός τρόπος να φτιαχτεί ένας απλός υπολογιστής σαν εκείνον του μαθήματος και του Εργαστηρίου στο CircuitVerse, όμως προφανώς υπάρχουν κι άλλοι
- Είστε ευπρόσδεκτοι να κάνετε τη δική σας παραλλαγή η πρόταση
- Καλές ιδέες από δικές σας παραλλαγές/προτάσεις θα εξεταστούν γιά πιθανή ενσωμάτωση στην εκφώνηση της επόμενης χρονιάς
- Καινοτομίες από πλευράς σας θα εκτιμηθούν βαθμολογικά θετικά (αρκεί η παραλλαγή σας να μην αφαιρεί από τον υπολογιστή κάποια ουσιαστική δυνατότητα, όπως δομές δεδομένων, επιστροφή από διαδικασίες, κλπ)
- Η εντολή Jump εδώ έχει διαφορετικόν Opcode απ' ό,τι είχε στην Άσκ. 11
- Η εντολή *jumpx* <u>*δεν*</u> ανήκει στην ομάδα "*Indexed* αριθμητικής/ld/st/inp"

Σημείωση γιά το νέο User Interface του CircuitVerse

- Το circuitverse.org/simulator απέκτησε νέο User Interface
- Μοιάζει να έχει (τουλάχιστον) ένα bug στο *Save Offline*
  - όταν έχει πολλά projects αποθηκευμένα και γεμίζει κατακόρυφα το μενού *Open Offline*, (α) χαλάει το UI, και (β) κάθε νέο save σβήνει και πανωγράφει το τελευταίο project. Επίσης, δεν δουλεύει το delete previously saved project (γιά να ξαλαφρώσει το σχετικό μενού)
- Το παλαιό User Interface, που μοιάζει να δουλεύει καλά, βρισκεται στο: https://circuitverse.org/simulator_old
  - οι Splitters στην εδώ εκφώνηση είναι από το παλαιό User Interf. και αριθμούν τα bits λανθασμένα (+1 στην θέση του τελικού bit)
  - και το font στα Properties είναι μαύρο πάνω σε μαύρο ☹

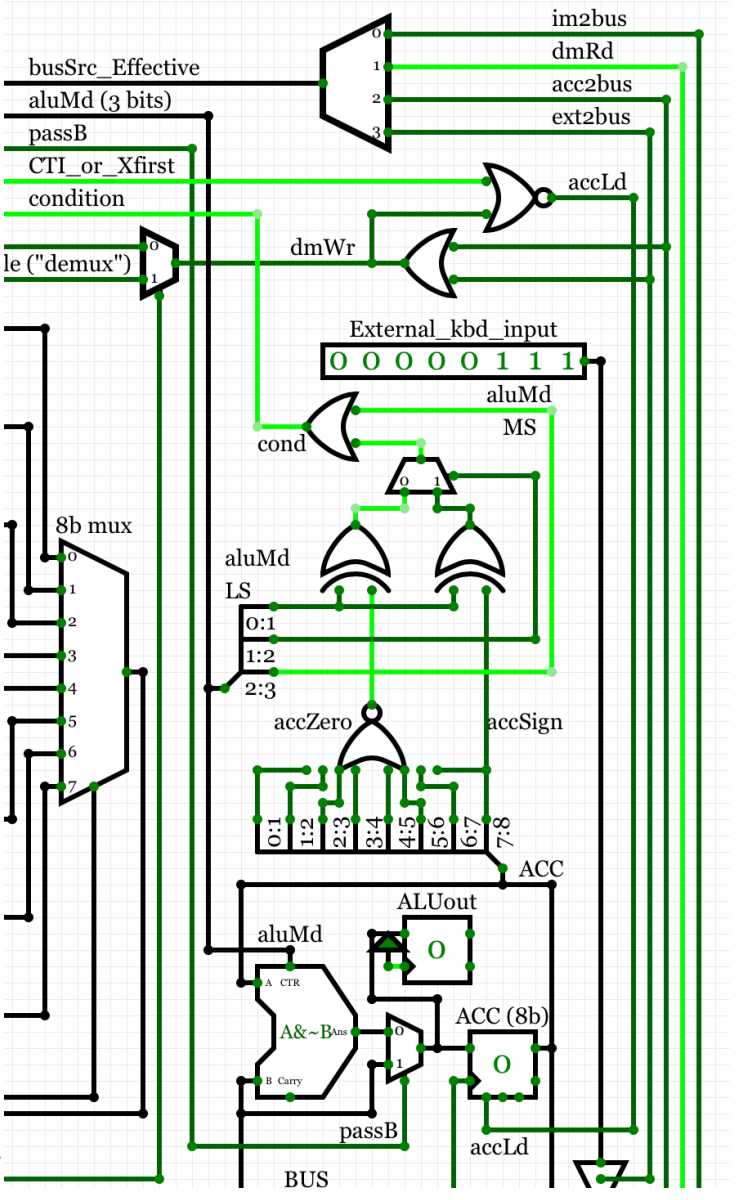

opcode
0:3
3:4
4:6
6:7
7:8
aluMd (3 bits)
passB
busSrc_Instr (2 bits)
CTI
X (indexed a/l/s/i)
Clock
pcLd
Xsecond
S
Sbar
Xfirst
CTI
dmRd
busSrc_Effective
CTI_or_Xfirst
addrMd
pcMd
condition
IMEM_00-of_op
IMEM_00-of_addr
IMEM_10-1f_op
IMEM_10-1f_addr
PC (5-bit)
PC+1
add (5-bit)
nxt_PC
other_nxt_PC
iaddr
RESET
initial_PC
TMP
BUS
DATA MEMORY
decoder_with_enable ("demux")
(demux)
dec_enab
8b mux
addr_3LS
addr_MS
dataOut
dmWr
busSrc_Effective
aluMd (3 bits)
passB
CTI_or_Xfirst
condition
im2bus
dmRd
acc2bus
ext2bus
accLd
External_kbd_input
cond
aluMd
MS
LS
accZero
accSign
ACC
ALUout
A&~B
ACC (8b)
passB
BUS (8b)

Αλλαγές από Άσκ. 11: μόνον επάνω, δεξιά & αριστερά

- Δεξιά πλευρά, πάνω από τον Συσσωρευτή (ACC):
  - *accLd*: γράφουμε στον ACC σε όλες τις άλλες περιπτώσεις *εκτός*: st/inp (*dmWr*), *CTI* (br/jmp), και τον πρώτο κύκλο των Indexed
  - *condition*: συνδυαστική συνάρτηση του περιεχομένου του ACC:
    - το aluMd επιλέγει: =accZero, ή =accSign, ή τα αντίθετά τους, ή =1
- Αριστερή πλευρά, πάνω από τη Μνήμη Εντολών (IMEM):
  - flip-flop κατάστασης S
    - *Xfirst*: ανάβει (μόνον) κατά τον 1ο κύκλο των Indexed ALU/ld/st/inp
    - *Xsecond*: ανάβει (μόνο) τον 2ο κύκλο των Indexed ALU/ld/st/inp
  - *busSrc*: όταν *Xfirst*, το αλλάζουμε σε 01 (δηλ. *dmRd*), μέσω mux
  - *pcMd*, *pcLd*, *addrMd*: νέες, ενημερωμένες συναρτήσεις

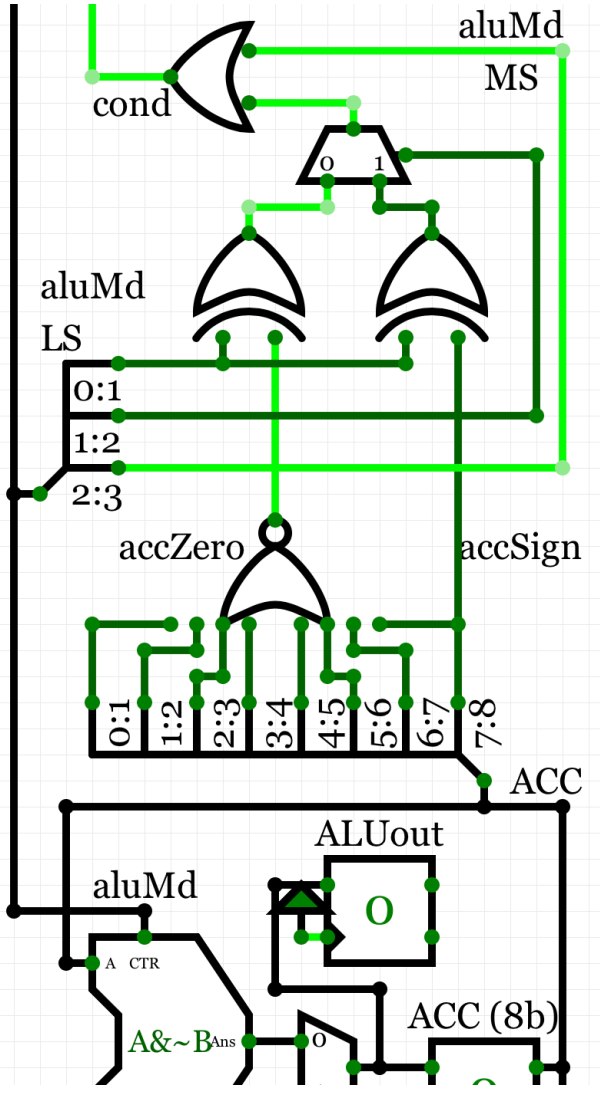
Σήματα ελέγχου ACC & BUS

- *busSrc_Effective*: είτε το *busSrc* όπως το δίνει η εντολή, είτε ο κωδικός 01 που τον χρειαζόμαστε τον 1ο κύκλο των εντολών Indexed
- *dmWr*: γράφουν στη μνήμη δεδομένων μόνον οι εντολές *store* και *input*, δηλαδή αυτές που ανάβουν το *acc2bus* ή το *ext2bus*
- *accLd* = **not** (*dmWr* **or** *CTI* **or** *Xfirst*)
  - γράφουμε στον *ACC* σε όλες τις άλλες περιπτώσεις εκτός *store/inp* (*dmWr*), εκτός *branch/jmp* (*CTI*), και εκτός τον πρώτο κύκλο των Indexed
- condition: συνδυαστική συνάρτηση του Συσσωρευτή (ACC) και του aluMd
  - βλ. επόμενη διαφάνεια

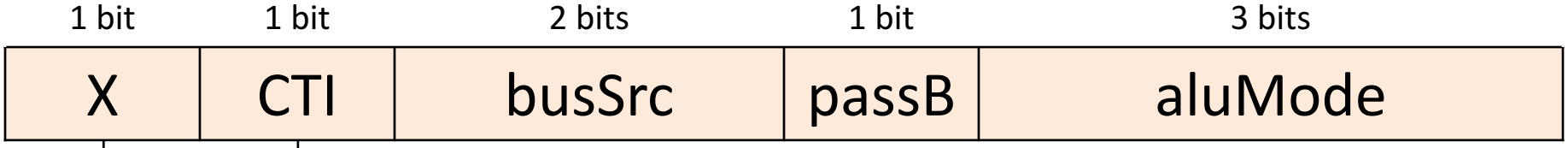


Condition: συνθήκη διακλαδώσεων

- Τόσο η ALU όσο και το Condition, ως συνδυαστικά κυκλώματα, πάντα κάτι υπολογίζουν καθένα, αλλά:
  - Το Condition χρησιμοποιείται μόνον όταν CTI==1, αλλά τότε accLd==0 και η έξοδος της ALU αγνοείται
  - Η έξοδος της ALU χρησιμοποιείται μόνον όταν accLd ==1, αλλά τότε CTI==0 και άρα το Condition αγνοείται
- Το MS bit του *aluMd*, όταν είναι 1, προκαλεί *cond*=1 ανεξαρτήτως τιμής του *ACC*, άρα χρησιμεύει γιά τα άλματα χωρίς συνθήκη (*jump* και *jumpx*)
  - Οι *jmp* & *jmpx* εκτελούνται με ακριβώς τον ίδιο τρόπο και οι δύο, εκτός *im2bus* γιά *jmp*, ενώ *dmRd* γιά *jmpx*
- Το μεσαίο bit του *aluMd* επιλέγει, μέσω πολυπλέ-κτη, αν θα κοιτάξουμε το *accZero* ή το *accSign*
- Το LS bit του *aluMd* επιλέγει αν θα θεωρηθεί επιτυ-χία η θετική ή η αρνητική πολικότητα των παραπάνω



Opcode



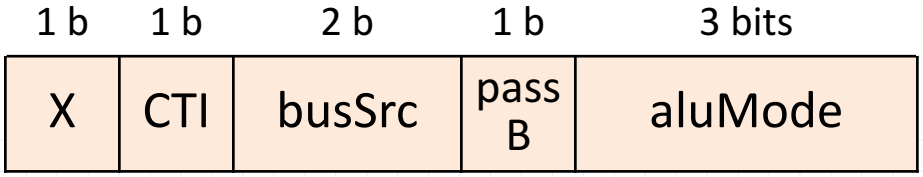
Indexed ALU/load/ store/inp (indirect DMEM access)

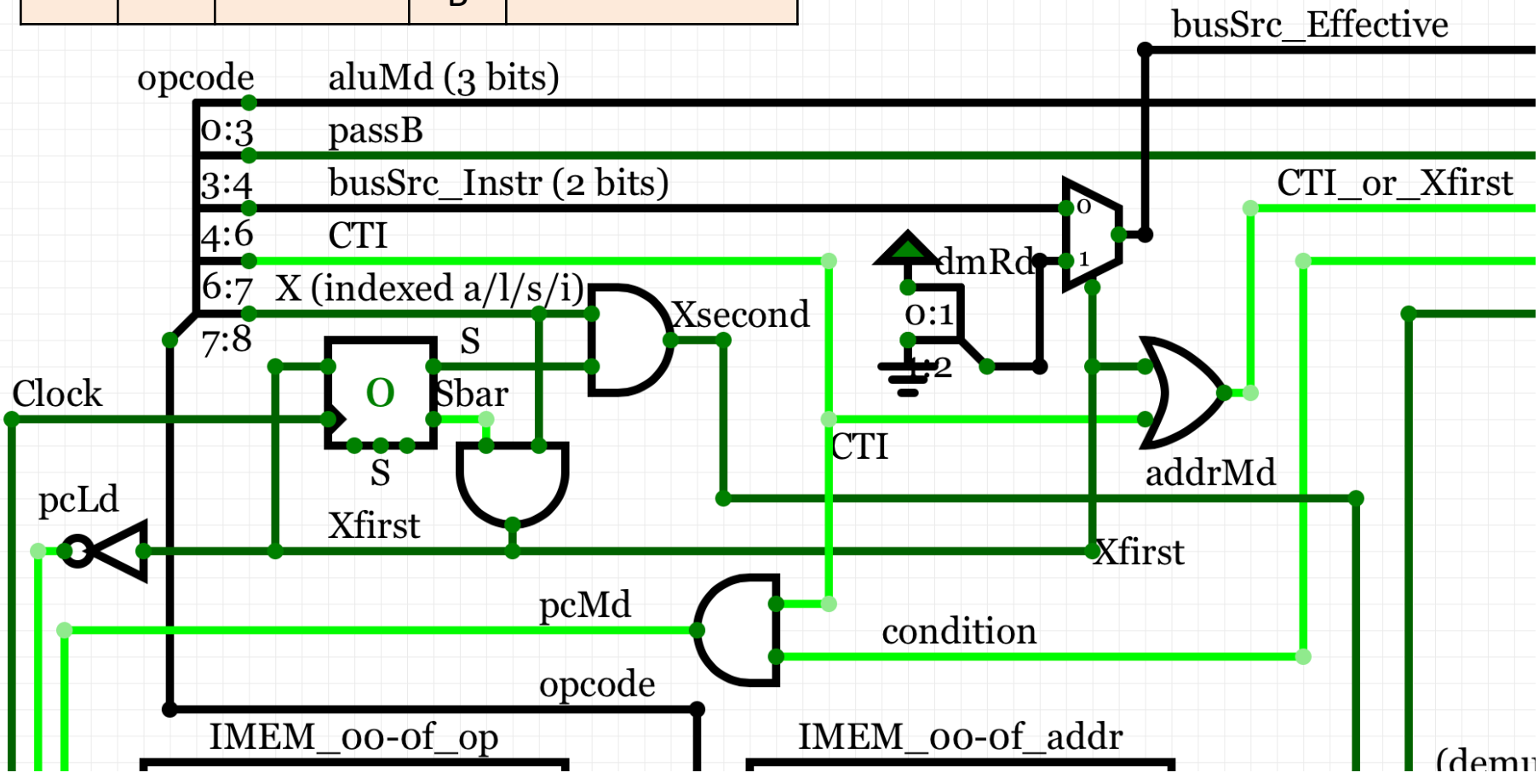
Control Transfer Instruction (branch /jump(x))

busSrc		enable
00	0	im2bus
01	1	dmRd
10	2	acc2bus
11	3	ext2bus

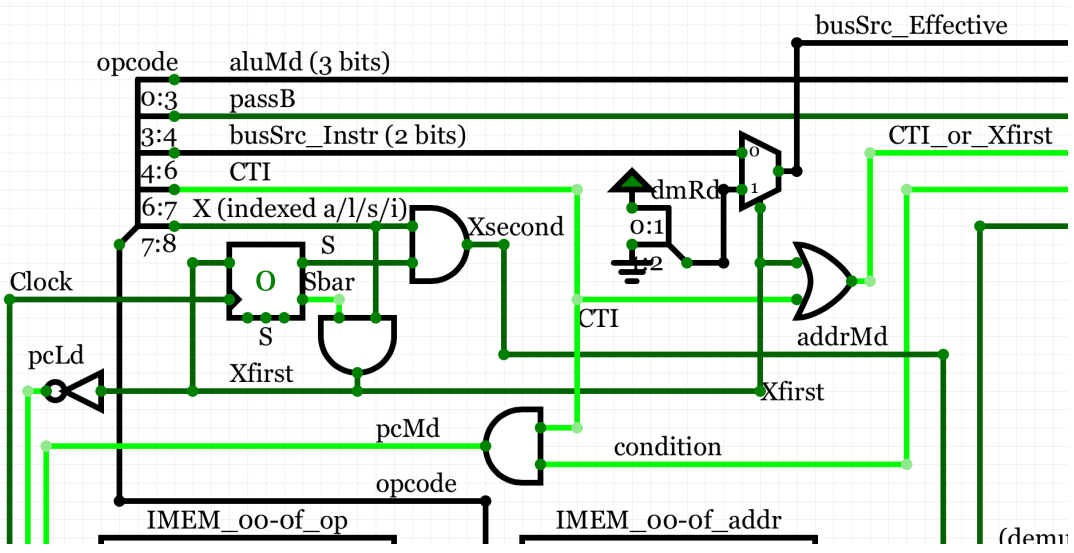
aluMd		ALUout	cond
000	0	A & B	= accZero
001	1	A | B	= ~accZero
010	2	A + B	= accSign
011	3	???	= ~accSign
100	4	A&(~B)	always true
101	5	A|(~B)	always true
110	6	A - B	always true
111	7	SLT	always true
(ίδια με CircuitVrs)			

Κυρίως Έλεγχος





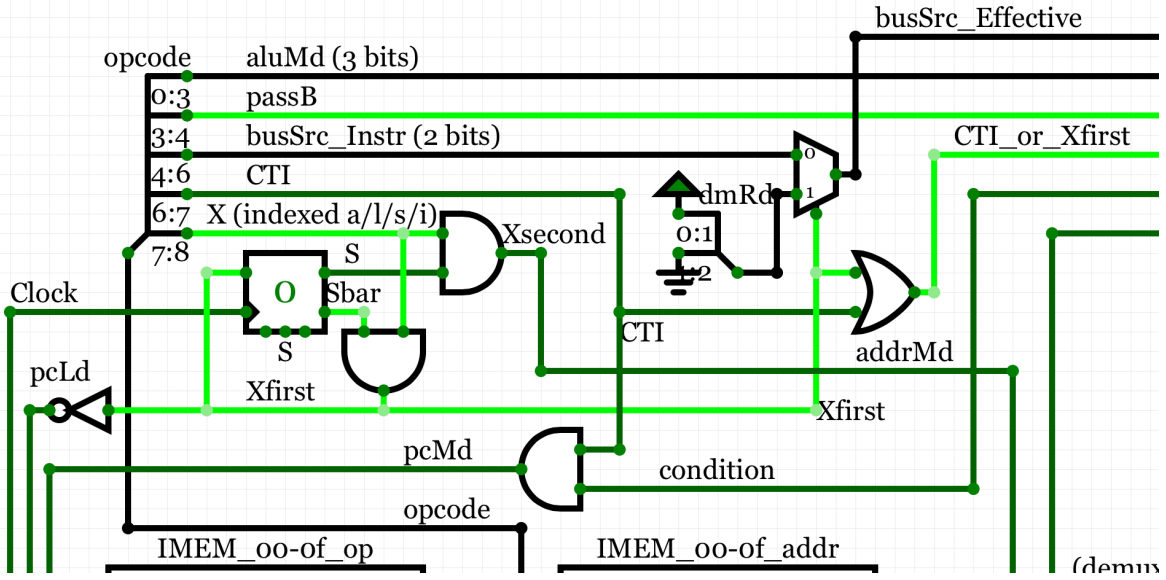
Εντολές CTI



- CTI = Control TransferInstruction (εντολή μτφ. ελέγχου), δηλ. διακλάδωση ή άλμα
- *pcMd* = *CTI* **and** *Condition*
- Υπόλοιπες εντολές εκτός CTI: *CTI*==0, άρα *pcMd* = 0, άρα ο πολυπλέκτης στην είσοδο του PC τροφοδοτείται από τον αυξητή *PC*+1
- Αποτυχημένες Διακλαδώσεις: *CTI*==1, αλλά *Condition*==0, οπότε *pcMd* = 0, επομένως και πάλι *PC* ← *PC*+1, αφού η συνθήκη της διακλάδωσης ήταν ψευδής
- Επιτυχημένες Διακλαδώσεις: *CTI*==1 και *Condition*==1, επομένως *pcMd* = 1, άρα ο πολυπλέκτης στην είσοδο του *PC* τροφοδοτείται από το *BUS*
  - *jmp* & *jmpx* είναι σαν πάντοτε επιτυχ. branch, επειδή *aluMd* επιλέγει πάντα *Cond* = 1
- *CTI*==1 προκαλεί *CTI_or_Xfirst* = 1 προς τα δεξιά, γιά να σβήσει το *accLd*

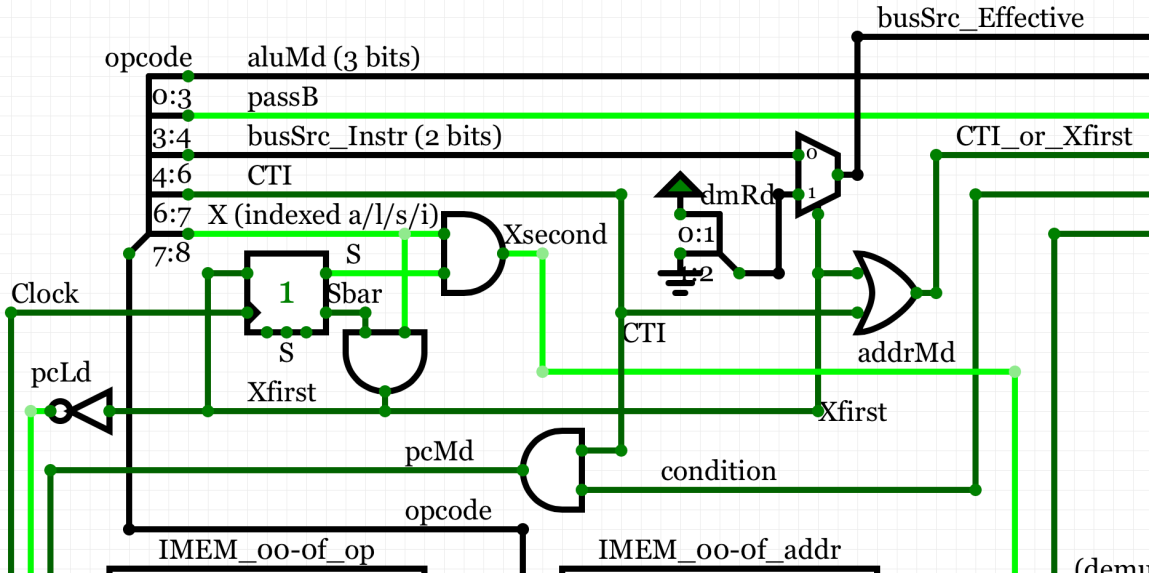
Indexed: 1ος κύκλος

- Το MS bit του Opcode, *X*, σηματοδοτεί τις εντολές *ALU*, *load*, store, ή *input* που είναι *Indexed* – η εντ. *jumpx* <u>*δεν*</u> είναι σε αυτές
- Γιά την αρχικοποίηση του state bit, *S*, βλ. επόμ. διαφ.



- Γιά τις εντολές αυτές όπου X==1, τον πρώτο κύκλο ρολογιού τους, S==0, άρα Sbar==1, άρα ανάβει το σήμα Xfirst
- Xfirst==1, μέσω του πολυπλέκτη δεξιά, δίνει 01 δηλ. dmRd, ανεξαρτήτως του πεδίου busSrc του Opcode, γιά ανάγνωση της διεύθυνσης του τελεστέου
- Xfirst==1, αριστερά, δίνει pcLd = 0 γιά να μείνει ο PC αμτβλ. στην ίδια εντολή
- Xfirst==1, αριστερά, δίνει επόμενη κατάσταση nxtS = 1 γιά τον 2ο κύκλο

Indexed: 2ος κύκλος



- Γιά τις εντολές αυτές με *X*==1, τον δεύτερο κύκλο τους, *S*==1, άρα ανάβει το σήμα *Xsecond*
- Το *Xsecond* επηρεάζει *μόνον* το σήμα *addrMd*, άρα στρίβει τον πολυπλέκτη της διεύθυνσης της *DMEM* προς τον *TMP*, ενώ σε όλες τις άλλες περιπτώσεις αυτός ήταν από το πεδίο *iaddr* της εντολής
- Από όλες τις άλλες απόψεις, η εντολή εκτελείται πανομοιότυπα με τις κανονικές εντολές *ALU/load/store/input*, και *nxtS* = 0 και *pcLd* = 1 όπως γιά τις υπόλοιπες εντ.
- Γιά όλες τις υπόλοιπες εντολές, *X*==0, επομένως *Xfirst* = *Xsecond* = 0, και το bit κατάστασης, *S*, αγνοείται. Έτσι, *nxtS* = 0 και *pcLd* = 1, άρα πάντα αλλάζουν τον *PC* πηγαίνοντας στην επόμενη εντολή, και πάντα *S*==0 όταν αρχίζει η επόμενη εντολή
  - Κανονικά, το Reset έπρεπε να πηγαίνει και στο S, αλλά εάν η πρώτη εντολή δεν είναι Indexed τότε αυτή θα αγνοήσει το S, και θα κάνει nxtS = 0, δηλ. σωστά γιά την επόμενη

Εντολές ALU/load/store/input Immediate & Κανονικές

Opcode		Opcode/control fields (binary)					Λειτουργία
Assembly	Hex	X	CTI	busSrc	passB	aluMode	
andi	00	0	0	00 (*im2bus*)	0	000	*ACC* ← *ACC & iaddr* (and immed.)
ori	01					001	*ACC* ← *ACC* | *iaddr* (or immediate)
addi	02					010	*ACC* ← *ACC + iaddr* (add immed.)
subi	06					110	*ACC* ← *ACC - iaddr* (sub immed.)
ldi	08				1	xxx	*ACC* ← *iaddr* (load immediate)
and	10	0	0	01 (*dmRd*)	0	000	*ACC* ← *ACC & DM[iaddr]* (and)
or	11					001	*ACC* ← *ACC* | *DM[iaddr]* (or)
add	12					010	*ACC* ← *ACC + DM[iaddr]* (add)
sub	16					110	*ACC* ← *ACC - DM[iaddr]* (subtract)
ld	18				1	xxx	*ACC* ← *DM[iaddr]* (load)
st	20	0	0	10 (acc2bus)	x	xxx	*DM[iaddr]* ← *ACC* (store)
inp	30			11 (ext2bus)	x	xxx	*DM[iaddr]* ← *Ext_kbd_in* (input)

Εντολές CTI και Indexed ALU/load/store/input

Opcode		Opcode/control fields (binary)					Λειτουργία
Assembly	Hex	X	CTI	busSrc	passB	aluMode	
beq	40	0	1	00 (*im2bus*)	x	000	**if** (*accZero*) **then** *PC* ← *iaddr*
bne	41					001	**if** (**not** *accZero*) **then** *PC* ← *iaddr*
blt	42					010	**if** (*accSign*) **then** *PC* ← *iaddr*
bge	43					011	**if** (**not** *accSign*) **then** *PC* ← *iaddr*
jmp	44				x	1xx	*PC* ← *iaddr* (unconditional jump)
jmpx	54			01 (dmRd)	x	1xx	*PC* ← *DM[iaddr]* (jump indexed)
andx	90	1	0	01 (*dmRd*)	0	000	*ACC* ← *ACC & DM[DM[iaddr]]* (and)
orx	91					001	*ACC* ← *ACC* | *DM[DM[iaddr]]* (or)
addx	92					010	*ACC* ← *ACC + DM[DM[[iaddr]]* (add)
subx	96					110	*ACC* ← *ACC – DM[DM[iaddr]]* (sub)
ldx	98				1	xxx	*ACC* ← *DM[DM[iaddr]]* (load indx)
stx	A0	1	0	10 (acc2bus)	x	xxx	*DM[DM[iaddr]]* ← *ACC* (store indx)
inpx	B0			11 (ext2bus)	x	xxx	*DM[DM[iaddr]]* ← *Ext_kbd_in*

Παράδειγμα 1: Αρχικοποίηση DMEM

Assembly:

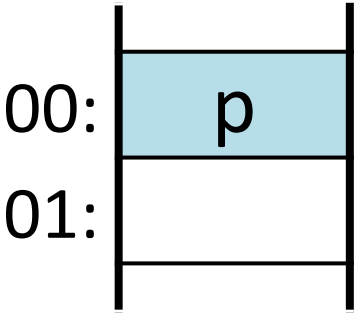
```
        00: inp   p (00)
loop → 01: inpx  *p  (00)
        02: ld    p (00)
        03: addi  1
        04: st    p (00)
        05: subi  10
        06: blt   01 (loop)
```

Instr. Mem.

00:	30	00
01:	B0	00
02:	18	00
03:	02	01
04:	20	00
05:	06	10
06:	42	01

```
p = input();
do {
    *p = input();
    p++;
} while (p<0x10);
```

Data Mem.



- Η inp στο 00 διαβάζει από το πληκτρολόγιο τη διεύθυνση της πρώτης θέσης της DMEM που θέλουμε να αρχικοποιήσουμε
- Στο 01 αρχικοποιούμε από πληκτρολόγιο εκεί που δείχνει ο p
- Στο 06 επαναλαμβάνουμε μέχρι και τη θέση 0F της DMEM

Παράδειγμα 2: Βρόχος διπλασιασμού στοιχείων πίνακα

Assembly:

```
        07: inp   n (01)
        08: ldi   0
        09: st    i (02)
loop →  0a: ld    i (02)
        0b: sub   n (01)
        0c: bge   exit (17)
        0d: ldi   a (08)
        0e: add   i (02)
        0f: st    p (00)
        10: ldx   *p (00)
        11: addx  *p (00)
        12: stx   *p (00)
        13: ld    i (02)
        14: addi  1
        15: st    i (02)
        16: jmp   loop (0a)
```

Instr. Mem.

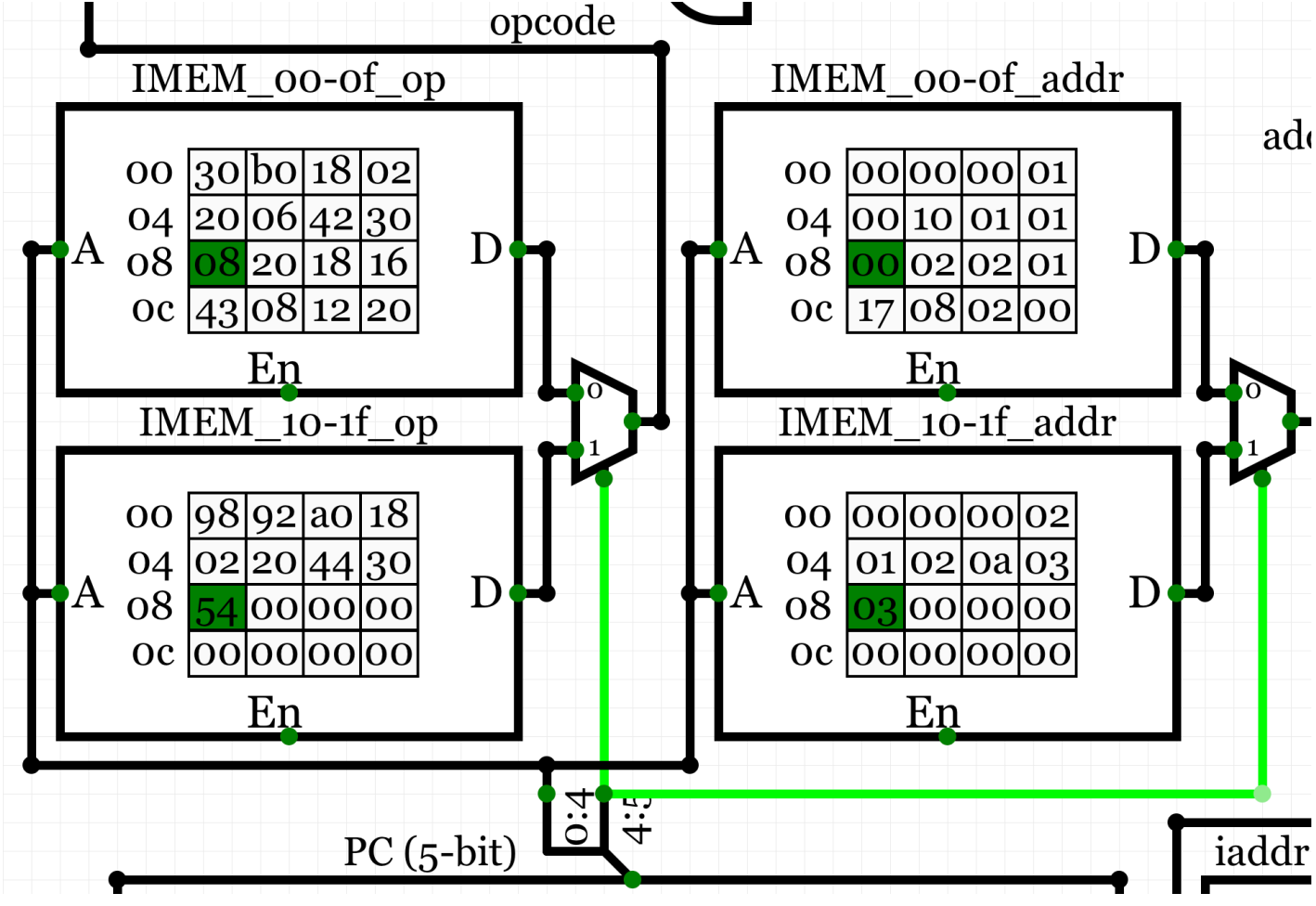
07:	30	01
08:	08	00
09:	20	02
0a:	18	02
0b:	16	01
0c:	43	17
0d:	08	08
0e:	12	02
0f:	20	00
10:	98	00
11:	92	00
12:	A0	00
13:	18	02
14:	02	01
15:	20	02
16:	44	0a

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

Data Mem.

00:	p
01:	n
02:	i
	…
08:	a[0]
09:	a[1]
0a:	a[2]
0b:	a[3]
0c:	a[4]

Η Μνήμη Εντολών με τα δύο αυτά παραδείγματα



Στις δύο επόμενες θέσεις, 17-18, είναι οι εντολές:

17: inp 03

18: jmpx 03

ώστε να διαβάζει έναν αριθμό – διεύθυνση εντολής από το πληκτρολόγιο και να πηδάει σε αυτή τη διεύθυνση (αυτό είναι και test της jmpx)

Τι έχετε να κάνετε

1. Μελετήστε και καταλάβετε την εκφώνηση
2. Τροποποιήστε τον απλό υπολογιστή σας της Άσκησης 11 ώστε αυτός να υλοποιεί και τις νέες εντολές αυτής εδώ της άσκησης
3. Κατανοήστε και ελέγξτε τη σωστή εκτέλεση των δύο παραδειγμάτων προγραμμάτων από τις διαφάνειες 15 και 16
4. Γράψτε και τρέξτε, και μερικές δικές σας εντολές στις θέσεις 17-1F ή 19-1F ή και 00-06, ή κάντε μιά δική σας παραλλαγή του βρόχου επεξεργασίας στοιχείων πίνακα («Παράδειγμα 2»)
5. Αναφορά (PDF):
   - φωτογρ. κυκλώματός σας & εξηγήσεις όπου διαφέρει από εκφώνηση,
   - Assembly, binary, & εξηγήσεις γιά τις εντολές/παραλλαγές στο 4 παραπ.