Άσκηση 11:
Ο Απλός Υπολογιστής στον Προσομοιωτή (μέρος Α')

011c (Άσκηση 11) – 4 - 7 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 Data Memory, εδώ, παραπιάνει πολύ χώρο – μακάρι να βρείτε κάποιον τρόπο να την μικρύνετε σε επιφάνεια που καταλαμβάνει στην οθόνη
- Εντολές διακλάδωσης & indexed δεν είναι εδώ - θα είναι στην Άσκηση 12

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
  - μόνο που το font στα Properties είναι μαύρο πάνω σε μαύρο ☹
  - οι Splitters στην εδώ εκφώνηση είναι από το παλαιό User Interf.

Μνήμη Εντολών: κατασκευή με ROM

- Η ROM έχει το πλεονέκτημα ότι βλέπουμε τα περιεχόμενά της (και ποιά λέξη διαβάζουμε τώρα), ότι μπορούμε με ποντίκι και πληκτρολόγιο να τα αρχικοποιούμε ή αλλάζουμε – δηλ. να γράφουμε εκεί τα προγράμματά μας – και αυτά διατηρούνται across save
- Η ROM είναι πάντα ενός μόνον μεγέθους: 16 λέξεις × 8 bits/λέξη
- Ίσως και να έφταναν 16 εντολές συνολικά (τσίμα-τσίμα το πρόγραμμα της §12.5), αλλά θα έπρεπε να σβήμουμε και γράφουμε τα διάφορα προγραμματα ελέγχου που θα έχουμε
- Γι' αυτό εγώ έβαλα χωρητικότητα 32 εντολές στην εδώ I_MEM, δηλαδή πεντάμπιτος PC και δύο ROM blocks καθ' ύψος
- Πλάτος 8 bits μόνον ανά εντολή θα ήταν σχεδόν αδύνατο (θα μας υποχρέωνε σε 4-μπιτες διευθύνσεις ⇒ 16 data words, 16 εντολές), άρα εγώ έβαλα 16-μπιτες εντολές ⇒ δύο ROM blocks κατά πλάτος

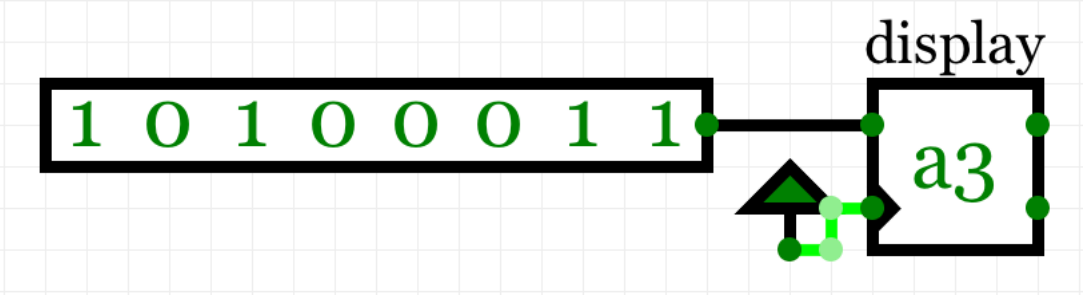
Οκτάμπιτος Opcode: Απλότητα και Ευελιξία Ελέγχου

- Κράτησα 8-μπιτο το δεξιό πεδίο των εντολών (συνήθως διεύθυνση)
  - έτσι, opcodes-διευθύνσεις χωρίζονται απλά σε αριστερές-δεξιές ROM
  - παρ' ότι 5 bits αρκούν γιά τις διευθύνσεις, όμως τώρα μπορούμε να βάζουμε και αριθμητικές σταθερές (8 bits) μέσα στις εντολές (χάρις στα πολλά opcodes)
- Με τα 8 bits που περισσεύουν γιά τον Opcode, μπορούμε τώρα να «σπαταλήσουμε» κάμποσους συνδυασμούς τους προκειμένου να απλοποιήσουμε το κύκλωμα ελέγχου
  - αφιερώνουμε opcode bits, ανά ένα ή λίγα, σε ένα ή μερικά σήματα ελέγχου
  - ακόμα και εάν μερικοί τέτοιοι συνδυασμοί εμφανίζονται σπανίως ή ποτέ
  - 3 bits γιά το ALU mode γιά τις 8 πράξεις της ALU του CircuitVerse
  - 1 bit («ολόκληρο»!) bit γιά "passB"
  - 2 bits γιά επιλογή ενός από τους 4 οδηγητές του BUS (imm, dm, acc, ext)
  - 1 bit γιά εντ. μεταφοράς ελέγχου (CTI – Control Transfer Instr. / jump-branch)
  - 1 bit γιά εντολές indexed (indirect) αριθμητικής και load/store

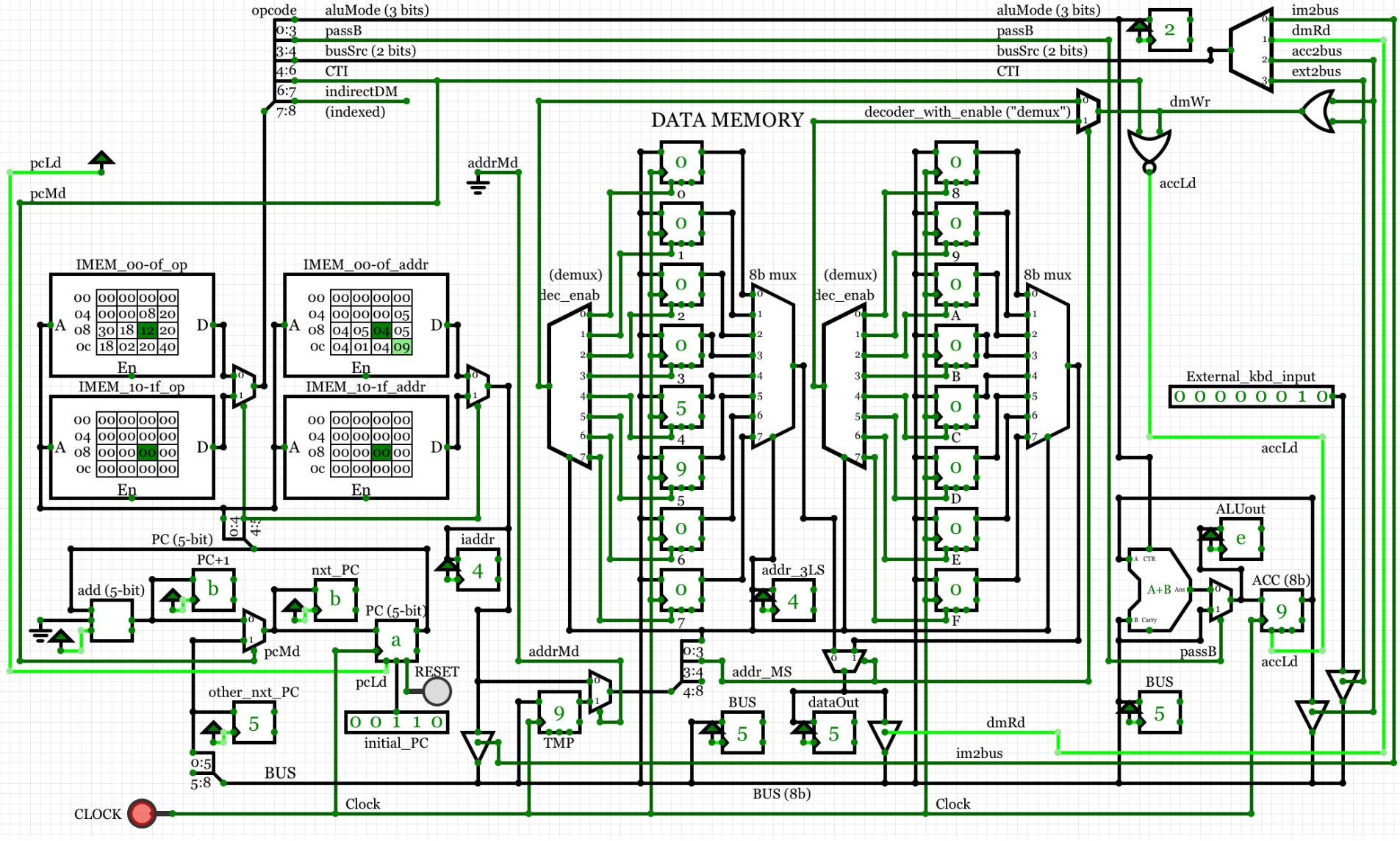
Μνήμη Δεδομένων με ορατά περιεχόμενα: Καταχωρητές

- Οι RAM του CircuitVerse υστερούν σε εποπτεία
  - δεν βλέπουμε τα περιεχόμενά τους (“core dump” δεν δουλεύει)
- Οι καταχωρητές (multi-bit DflipFlop’s) δείχνουν το περιεχόμενό τους
- Προτείνω την κατασκευή της Data Memory με καταχωρητές
  - Πλεονέκτημα: βλέπουμε συνεχώς όλα τα περιεχόμενα
  - Μειονέκτημα: πιάνει πολύ χώρο στην οθόνη
- Εγώ την έκανα με 16 λέξεις (τετράμπιτες διευθύνσεις)
  - σε δύο στήλες, γιά να μην είναι υπερβολικά ψηλή
  - 12 λέξεις θα αρκούσαν, αλλά δεν είναι δύναμη του 2
  - 8 λέξεις θα ζόριζαν στην άσκηση με τον πίνακα (§12.5)
- Ακμοπυροδότητοι καταχωρητές, επειδή στις εντολές εγγραφής το σήμα *dmWrite* μπορεί να ανάψει *πριν* η διεύθυνση να είναι έγκυρη

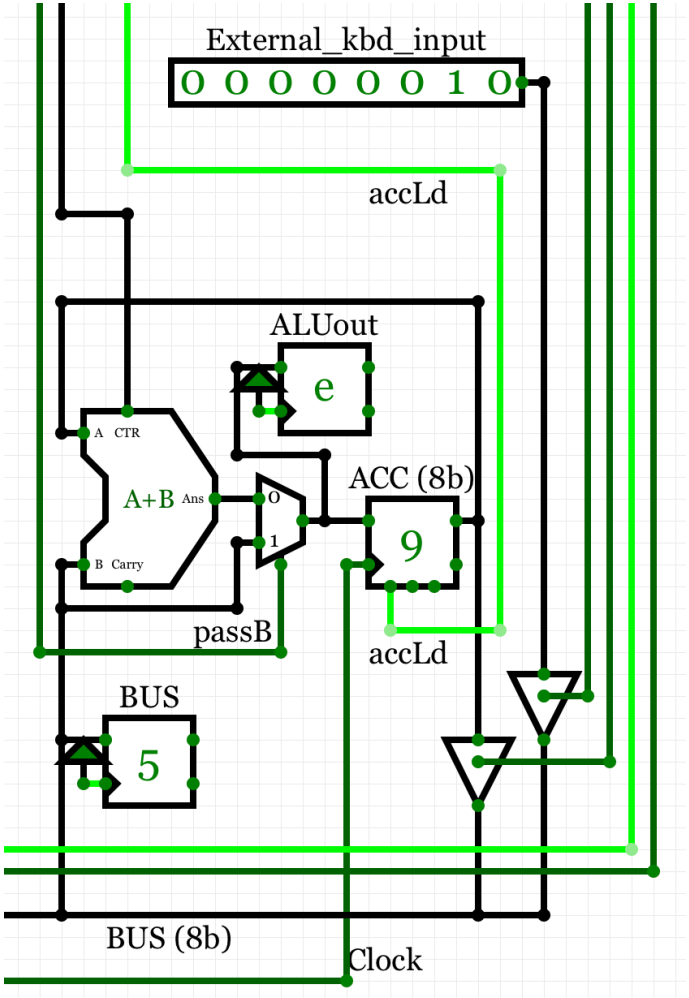
Παρακολούθηση τιμών: Μανταλωτές με *load=1*



- Dlatch (μανταλωτής)
  - από το “Sequential Elements” menu
- BitWidth = 8
- Label = επιθυμητό όνομα, να μας θυμίζει τι δείχνει
- Load enable input (την λέει “clock” – κακώς!) συνδεδεμένη σταθερά στο 1 ⇒ το περιεχόμενο αλλάζει αμέσως μόλις αλλάξει η είσοδος (1 = “Power”, από Input menu)
- Το περιεχόμενο εμφανίζεται στο Δεκαεξαδικό
  - πολύ πιό compact από τα HexDisplays

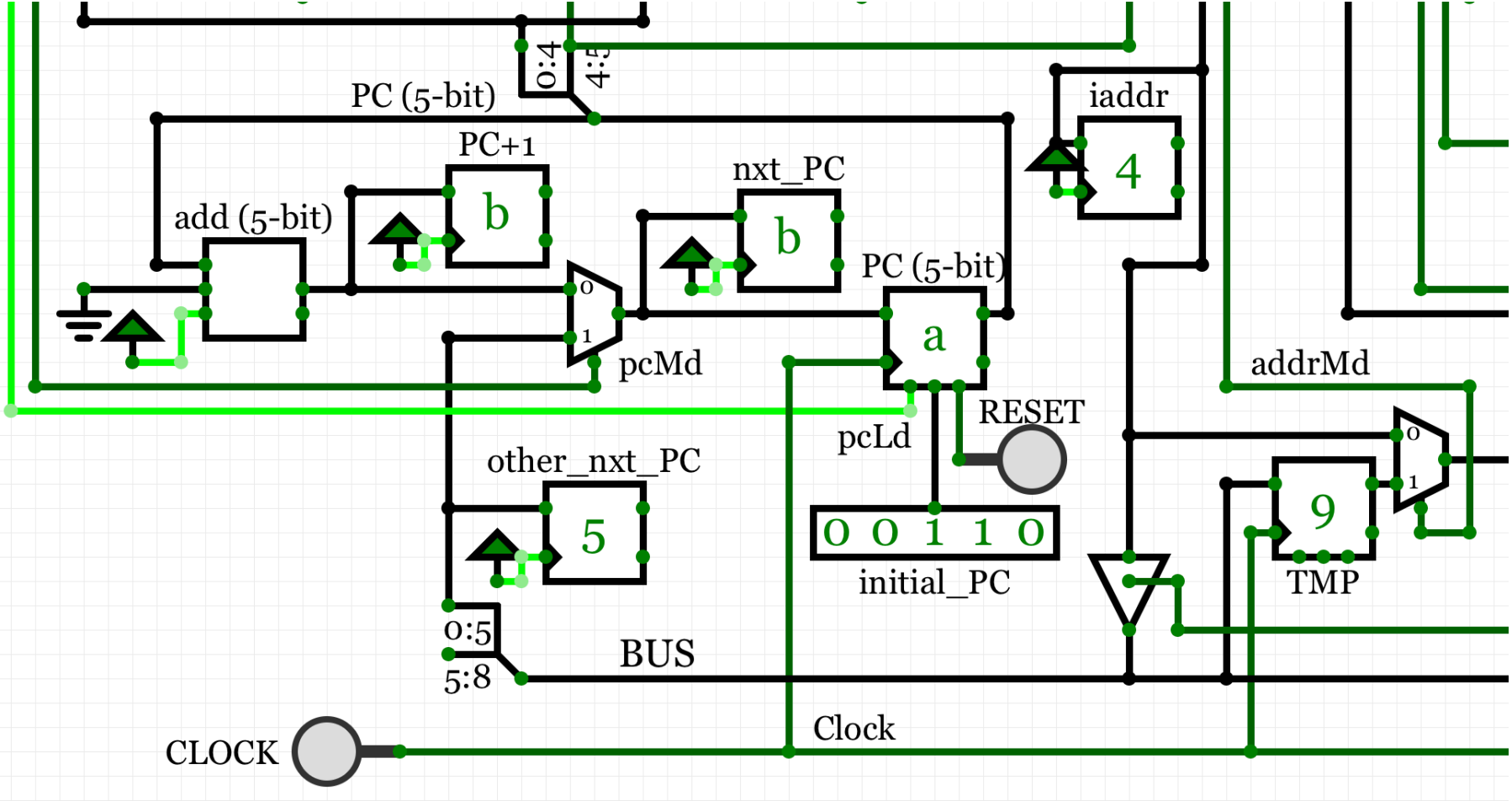

opcode
aluMode (3 bits)
0:3
passB
3:4
busSrc (2 bits)
4:6
CTI
6:7
indirectDM
7:8
(indexed)
aluMode (3 bits)
passB
busSrc (2 bits)
CTI
2
im2bus
dmRd
acc2bus
ext2bus
dmWr
accLd
DATA MEMORY
decoder_with_enable ("demux")
addrMd
pcLd
pcMd
IMEM_00-of_op
IMEM_00-of_addr
IMEM_10-1f_op
IMEM_10-1f_addr
(demux)
dec_enab
8b mux
External_kbd_input
ALUout
ACC (8b)
PC (5-bit)
PC+1
add (5-bit)
nxt_PC
iaddr
addr_3LS
addr_MS
other_nxt_PC
initial_PC
RESET
TMP
BUS
dataOut
BUS (8b)
CLOCK
Clock
passB
dmRd
im2bus
accLd

Αριθμητική-Λογική Μονάδα και συναφή



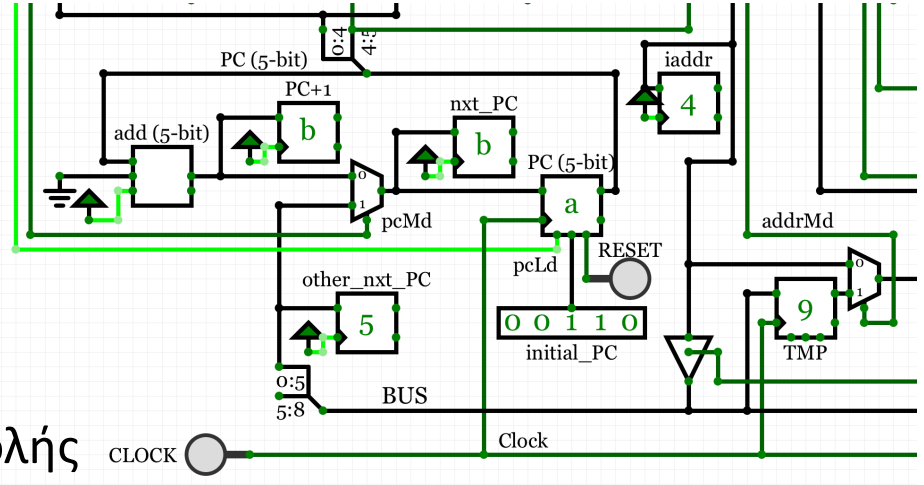
- BitWidth = 8 σε όλα τα στοιχεία εδώ
- **ALU** από το *Misc* menu του circuitverse
- Κρατάμε τις 8 πράξεις που αυτή έχει, όπως τις έχει, με το 3-μπιτο σήμα ελέγχου *aluMode*
- Δεν έχει «πράξη» *passB*, άρα χρειαζόμαστε εξωτερικά της τον πολυπλέκτη που φαίνεται, με το («τέταρτο») σήμα ελέγχου *passB*
- *DflipFlop* (8-bit) *ACC* ως Συσσωρευτής, με σήμα ελέγχου φόρτωσης *accLd*
- Δύο (8-μπιτα) *Dlatch*'es ως displays (*BUS*, *ALUout*)
- Εξωτερική είσοδος «πληκτρολογίου» (8-μπιτη)
- Δύο τρικατάστατοι οδηγητές ελεγχόμενοι από τα σήματα ελέγχου *acc2bus*, *ext2bus*

Program Counter (PC)



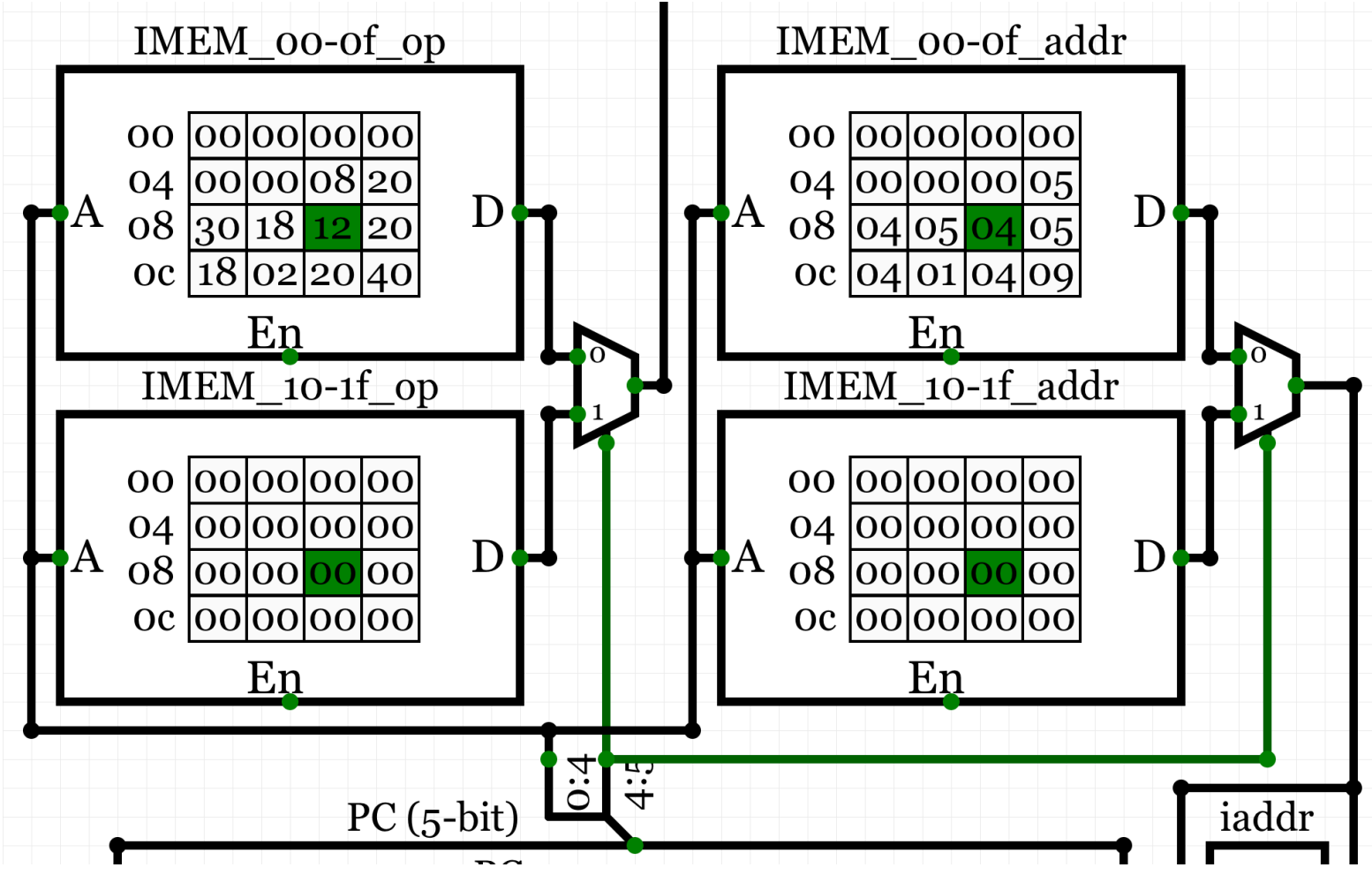
Program Counter (PC)

- Εάν κρατείστε το μέγεθος της μνήμης εντολών σε 32 όπως εδώ, τότε το BitWidth = 5 γιά PC και συναφή
- Το BUS όμως είναι 8-μπιτο, άρα χρειάζεται ένας Splitter κάτω αριστερά, εκεί που παίρνουμε τα 5 LS bits του BUS ως υποψήφια διεύθυνση επόμενης εντολής



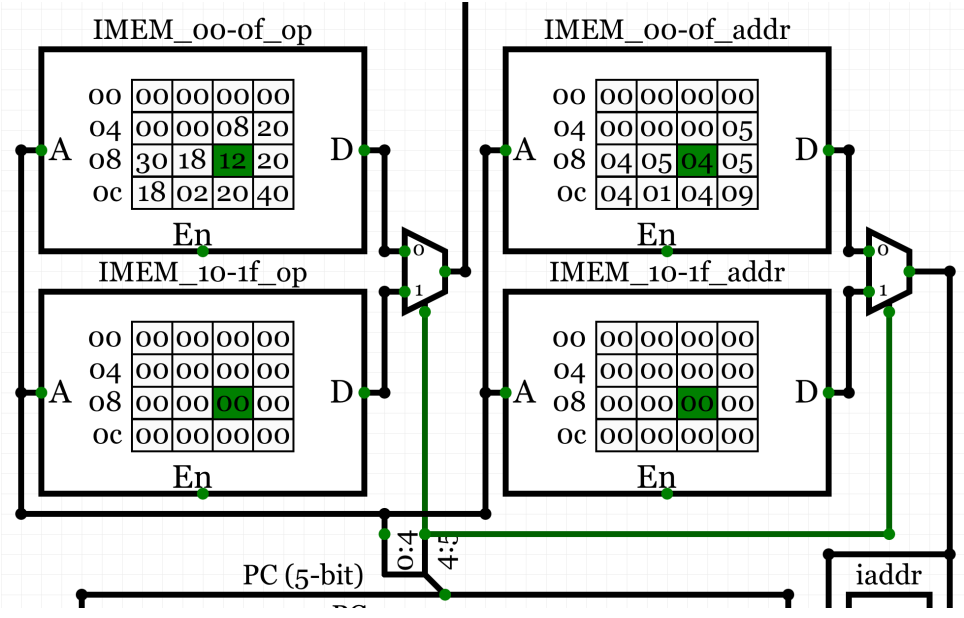
- *DflipFlop* (πεντάμπιτο) ως PC, δεξιά στο βρόχο, με έλεγχο load enable (*pcLd*)
- Σήμα *Reset* και διακόπτες αρχικής τιμής γιά διεύθ. εκκίνησης προγρ. (“*main*”)
- *Adder* (5-bit) από *Misc* menu, αριστερά, με σταθερές εισόδους γιά **+1** πάντα
- Πολυπλέκτης με έλεγχο *pcMd* γιά επόμενη εντολή: «από κάτω» ή «άλλη»
- Τρία (πεντάμπιτα) *Dlatch*’es ως displays (*PC+1*, *other_nxt_PC*, *nxt_PC*)
- Εδώ, pcLd είναι πάντα =1 (θα αλλάξει στην Άσκ. 12 γιά εντολές indexed)

Μνήμη Εντολών (*IMEM*)

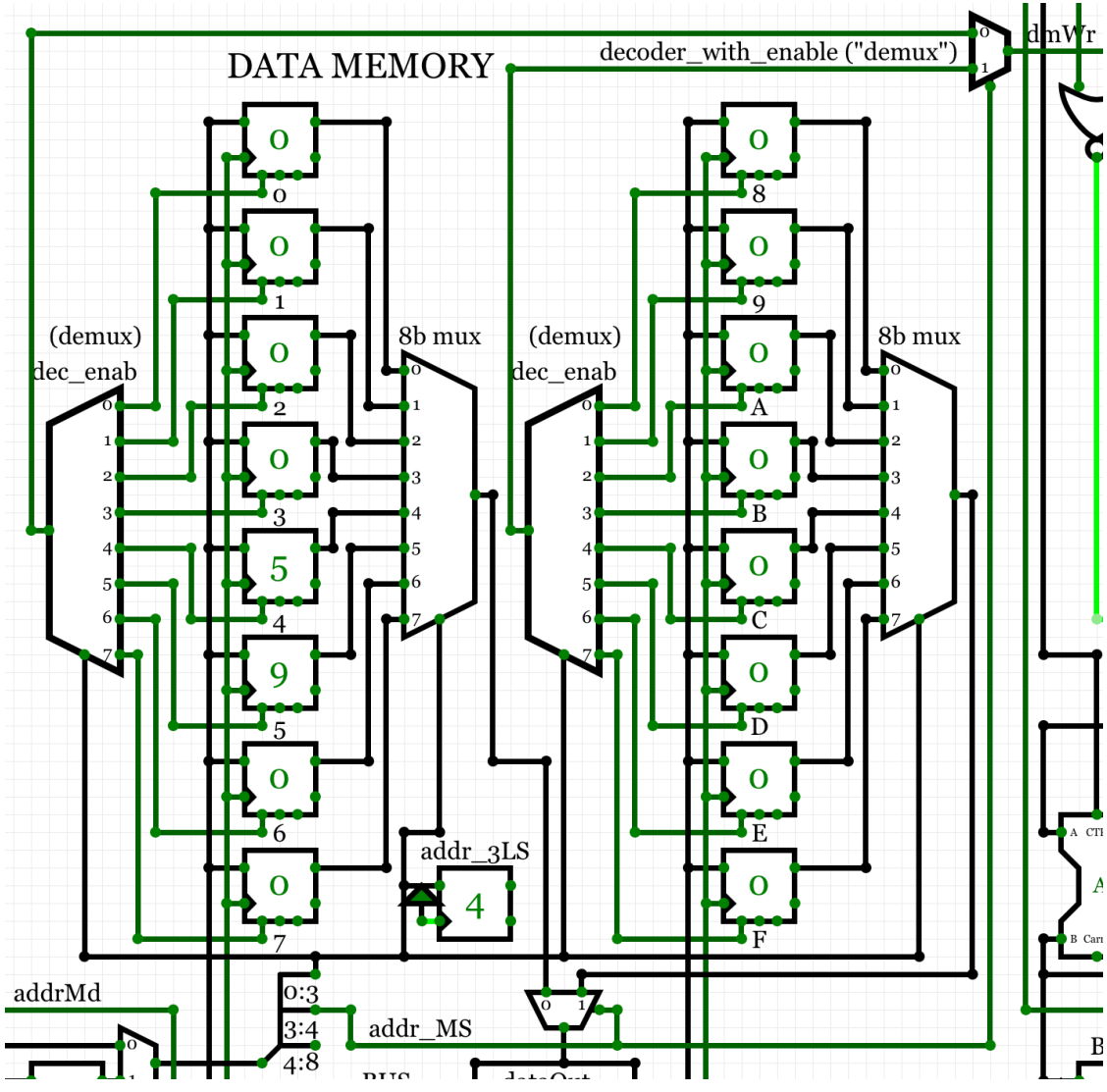


Μνήμη Εντολών (*IMEM*)

- Τέσσερα μπλόκ "*Rom*" από το *Memory Elements* menu
- Αριστερά οι Opcodes, με έξοδο προς τα επάνω (προς Έλεγχο)
- Δεξιά οι αριθμητικές σταθερές ή διευθύνσεις, iaddr ("address from instruction") με έξοδο δεξιά κάτω
- Επάνω οι διευθύνσεις 00 έως και 0F
- Κάτω οι διευθύνσεις 10 έως και 1F



- Διεύθυνση (πεντάμπιτη) από τον PC, κάτω, μέσω Splitter γιά 4 LS bits, 1 MS bit
- Τα 4 LS bits διεύθυνσης πάνε στο κάθε ROM block
- Το ένα MS bit διεύθυνσης επιλέγει, μέσω πολυπλεκτών, το πάνω ή κάτω μπλόκ

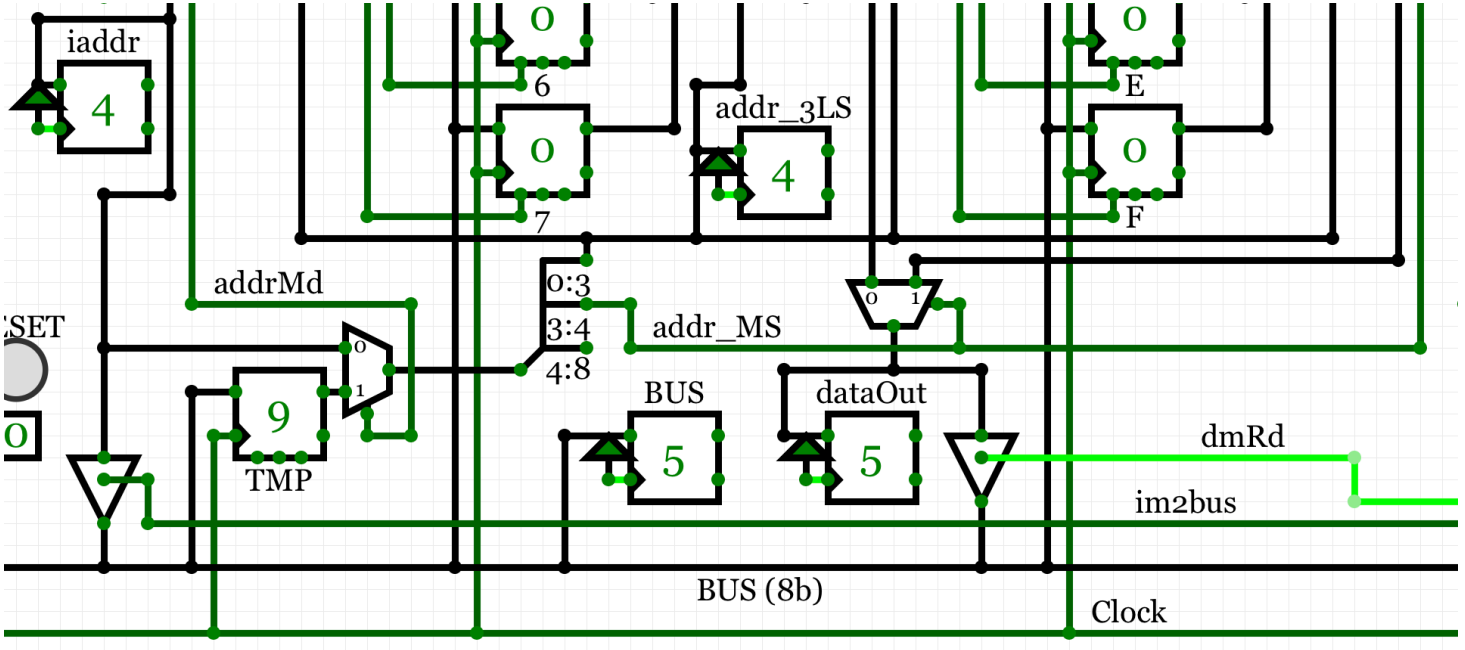


Μνήμη Δεδομένων

- DflipFlops (8-μπιτα) ως λέξεις
- Αριστερή στήλη οι διευθ. 0 - 7
- Δεξιά στήλη οι διευθ. 8 - F
- Τετράμπιτη Διεύθυνση από κάτω αριστερά μέσω Splitter:
  - 3 LS bits επιλογή πάνω-κάτω
  - 1 MS bit επιλ. αριστερά-δεξιά
- Ανάγνωση με πολυπλέκτες, δεξιά στην κάθε στήλη (3 LS addr. bits), και μετά πολυπλ. κάτω (*addr_MS*)
- Εγγραφή από αριστερά, με *Clock* και τα *DataIn* από το *BUS* (κάτω)
- Επιλογή λέξης γιά τυχόν εγγραφή μέσω των *loadEnable*, με χρήση αποκωδικοποιητών (αριστερά) που έχουν και σήμα Enable (“*demux*” τους λέει), ξεκινώντας από *dmWr* πάνω δεξιά, με MS addr. bit, κλπ.

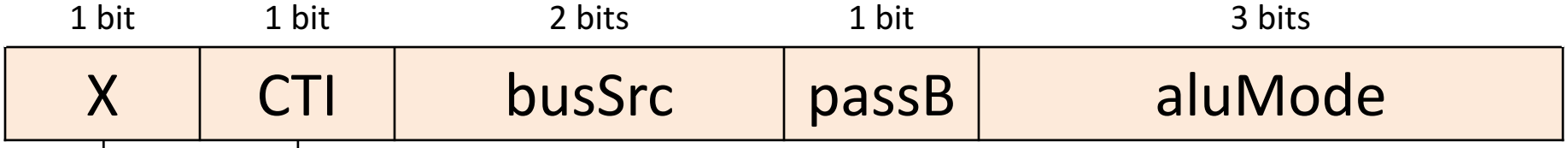
BUS, Οδήγηση από IMEM ή DMEM, Address Mode

- Τρικατάστατος προς BUS από δεξιό πεδίο της *IMEM* (“*iaddr*”) (στα αριστερά) ελεγχόμενο από *im2bus*
- Ομοίως από *DMEM* (μέσον κάτω, “*dataOut*”) ελεγχόμενο από *dmRd* (mem. Read)



- Τρία οκτάμπιτα *Dlatch*’es ως displays: *iaddr* (πάνω αριστερά), *BUS*, *dataOut* (κάτω μέσον)
- Οκτάμπιτος καταχωρητής *TMP* κρατά την τιμή του BUS από προηγ. κύκλο γιά εντ. indexed
- Διεύθυνση προς Data Memory μέσω οκτάμπιτου πολυπλέκτη ελεγχόμενου από addrMd
- Εδώ το *addrMd* πάντα =0 –η διεύθυνση από TMP μόνον γιά εντολές Indexed (Άσκ. 12)

Opcode



Indexed load/store /ALU (indirect DMEM access)

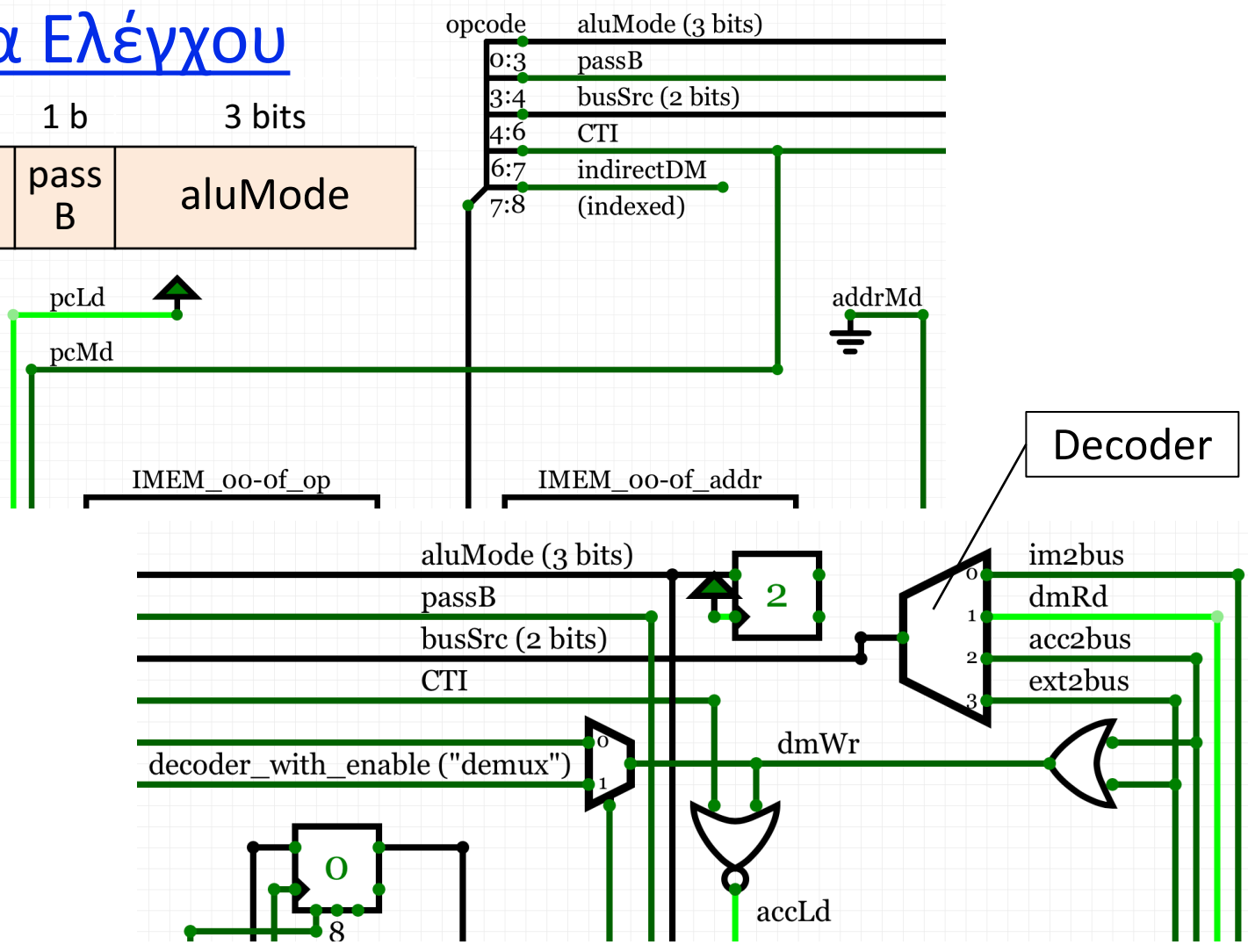
Control Transfer Instruction (branch /jump)

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

aluMode		Operation
000	0	A & B (and)
001	1	A | B (or)
010	2	A + B (add)
011	3	???
100	4	A & (~B)
101	5	A | (~B)
110	6	A - B (sub)
111	7	set-if-less-than
(ίδια όπως CircuitVerse ALU)		

Κύκλωμα Ελέγχου

1 b	1 b	2 b	1 b	3 bits
X	CTI	busSrc	pass B	aluMode



Άμεσες Σταθερές (Immediate Constants)

- Αφού δώσαμε περισσότερα bits στον Opcode, μπορούμε να έχουμε και εντολές αριθμητικής/load με σταθερό τελεστέο
- Σταθερά ως τελεστέος: οδήγηση του BUS από IMEM (*im2bus*)
- Μεταβλητή ως τελεστέος: οδήγηση του BUS από DMEM (*dmRd*)
- Έστω "***iaddr***" τα δεξιά 8 bits της εντολής (imm. constant / address):
  - *load immediate*: *ACC* ← *iaddr* (τα 8 bits από εντολή αυτά καθεαυτά)
  - *load* (παραδοσ.): *ACC* ← *DM[iaddr]* (η μτβλ. από εκεί που λένε τα 8 bits)
  - *add immediate*: *ACC* ← *ACC + iaddr* (πρόσθεση σταθεράς από εντολή)
  - *add* (παραδοσ.): *ACC* ← *ACC + DM[iaddr]* (πρόσθεση μεταβλητής)
- Έτσι απαλλασόμαστε και από το πρόβλημα του να πρέπει να βάζουμε τις σταθερές στην DMEM και να πρέπει και να τις αρχικοποιούμε

Οι πιό χρήσιμες Εντολές (γιά την Άσκηση 11)

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	0	0		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	0	0		1	xxx	*ACC* ← *DM[iaddr]* (load)
st	20	0	0	10 (acc2bus)	x	xxx	*DM[iaddr]* ← *ACC* (store)
inp	30	0	0	11 (ext2bus)	x	xxx	*DM[iaddr]* ← *Ext_kbd_in* (input)
jmp	40	0	1	00 (im2bus)	x	xxx	*PC* ← *iaddr* (jump)

Παράδειγμα: Άθροισμα ακεραίων –από δεδομένον & πάνω

Assembly:

```
        06: ldi  00
        07: st   s  (05)
        08: inp  n (04)
loop → 09: ld   s  (05)
        0a: add  n (04)
        0b: st   s  (05)
        0c: ld   n (04)
        0d: addi 01
        0e: st   n (04)
        0f: jmp  09 (loop)
```

Instr. Mem.

06:	08	00
07:	20	05
08:	30	04
09:	18	05
0a:	12	04
0b:	20	05
0c:	18	04
0d:	02	01
0e:	20	04
0f:	40	09

```
s = 0;
n = input();
while(1) {
    s = s+n;
    n = n+1;
}
```

Data Mem.

04:	n
05:	s

Τι έχετε να κάνετε

1. Μελετήστε και καταλάβετε την εκφώνηση
2. Σχεδιάστε τον απλό υπολογιστή σας στο CircuitVerse
   – είτε αυτόν που δίδεται σε αυτή την εκφώνηση, είτε την δική σας παραλλαγή
3. Γράψτε μερικές δικές σας απλές εντολές που να ελέγχουν το κύκλωμα, στις διευθύνσεις 00 έως και 05 (ή και πιό κάτω) της IMEM
4. Ορίστε *initial_PC* = 0, πατήστε *RESET*, μετά *CLOCK* επανειλημμένα, και κατανοήστε και ελέγξτε τη σωστή εκτέλεση των εντολών
5. Ομοίως γιά το πρόγραμμα της διαφάνειας 20 (με *initial_PC* = 06)
6. Γράψτε, τρέξτε, και ελέγξτε κι ένα άλλο δικό σας προγραμματάκι στις θέσεις 10 - 1F (ή άλλες) της μνήμης εντολών (*initial_PC* = 10)
7. Αναφορά (PDF): φωτογραφία του 2 & εξηγήσεις εάν διαφέρει από την εκφώνηση, Assembly, binary, & εξηγήσεις γιά 3 και 6 παραπάνω