ΗΥ-120: Ψηφιακή Σχεδίαση Τμ. Επ. Υπολογιστών
Φθινόπωρο 2007 © Πανεπιστήμιο Κρήτης

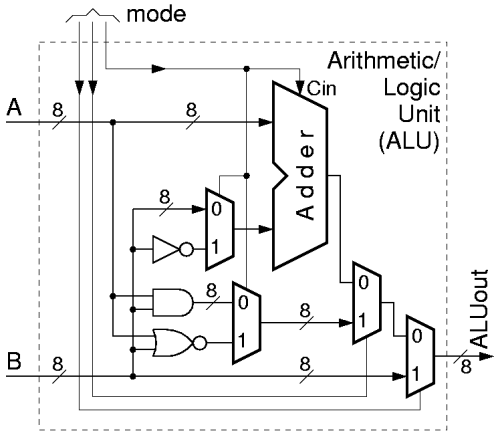
Εργαστήριο 11:
Ένας απλός Υπολογιστής: Datapath & Εντολές Πράξεων

17 - 20 Δεκεμβρίου 2007

Στο τελευταίο αυτό μέρος του μαθήματος θα δούμε πώς μπορούμε να φτιάξουμε έναν απλό υπολογιστή χρησιμοποιώντας μόνο βασικούς δομικούς λίθους αυτού του μαθήματος: καταχωρητές, μνήμες, αθροιστές, πολυπλέκτες, και ένα απλό συνδυαστικό κύκλωμα ή μιά απλή FSM γιά να ελέγχει τα παραπάνω. Στο εργαστήριο θα βρείτε έτοιμο το τμήμα δεδομένων (datapath) αυτού του υπολογιστή, όπου υπάρχουν ενδείκτες με τις τιμές όλων των διευθύνσεων, δεδομένων, και σημάτων, και θα φτιάξετε εσείς το κύκλωμα ελέγχου· την επόμενη βδομάδα, θα συνεχίσετε με πιό πολύπλοκες εντολές και κυκλώματα ελέγχου. Παρ' ότι ο υπολογιστής αυτός είναι πολύ απλός (απλοϊκός) --και πολύ αργός!-- είναι εντούτοις ένας κανονικός υπολογιστής, ικανός να εκτελεί (σχεδόν) το κάθε πρόγραμμα· τα μόνα που του λείπουν είναι σχετικά δευτερεύουσες λεπτομέρειες, και όχι κάτι εννοιολογικά κεντρικό. Ο υπολογιστής μας θα είναι οκτάμπιτος (8-bit), δηλαδή θα δουλεύει με λέξεις των 8 bits· μοναδική εξαίρεση θα είναι οι εντολές, που έχουν 12 bits καθεμία (οπότε και η μνήμη που τις περιέχει θα είναι δωδεκάμπιτη στο πλάτος).

11.1 Μία απλή Αριθμητική/Λογική Μονάδα (ALU):

Στην καρδιά ενός υπολογιστή είναι μιά μονάδα που εκτελεί αριθμητικές και λογικές πράξεις. Από αριθμητικές πράξεις, θα έχουμε μόνο πρόσθεση και αφαίρεση ακεραίων: γιά λόγους απλότητας δεν θα έχουμε πολλαπλασιασμό (μοιάζει με πολλές προσθέσεις), διαίρεση (μοιάζει με επανειλημμένες αφαιρέσεις), ή αριθμούς κινητής υποδιαστολής (οι σχετικές πράξεις ανάγονται τελικά σε πράξεις ακεραίων). Θα χρησιμοποιήσουμε σαν "Αριθμητική/Λογική Μονάδα" (Arithmetic/Logic Unit - **ALU**) το κύκλωμα του σχήματος, που στην καρδιά του έχει έναν αθροιστή (adder) σαν αυτούς που είδαμε στο εργαστήριο 5. Όπως είδαμε στην §6.7, η αφαίρεση A-B γίνεται μέσω της πρόσθεσης A+(B')+1, όπου A και B είναι προσημασμένοι ακέραιοι αριθμοί σε κωδικοποίηση συμπληρώματος ως προς 2, και (B') είναι ο αριθμός που προκύπτει από τον B αν αντιστρέψουμε το κάθε bit του (συμπλήρωμα ως προς 1, δηλ. λογικό-ΟΧΙ). Την ιδιότητα αυτή εκμεταλλευόμαστε, με τον αριστερό-πάνω πολυπλέκτη, γιά να κάνει η ALU μας και αφαίρεση A-B· το "+1" προκύπτει φροντίζοντας να είναι 1 το κρατούμενο εισόδου, Cin, του αθροιστή όποτε ο πολυπλέκτης επιλέγει το (B'), δηλαδή όποτε κάνουμε αφαίρεση.



Εκτός από τον οκτάμπιτο αθροιστή, η ALU έχει και 8 πύλες ΚΑΙ (AND) καθώς και 8 πύλες ΟΥΤΕ (NOR) γιά να μπορεί να κάνει τις αντίστοιχες λογικές πράξεις πάνω στις λέξεις εισόδου· πρόκειται γιά λογικές πράξεις bit-προς-bit (bitwise operations), δηλαδή το bit *i* της εξόδου θα είναι το λογικό ΚΑΙ ή το λογικό ΟΥΤΕ του bit *i* της εισόδου A και του bit *i* της εισόδου B. Κάθε τέτοια οκτάδα πυλών φαίνεται σχεδιασμένη σαν μία πύλη, κάτω αριστερά. Ο πολυπλέκτης αριστερά κάτω επιλέγει αν το αποτέλεσμα λογικής πράξης που θέλουμε να κρατήσουμε είναι το λογικό ΚΑΙ ή το λογικό ΟΥΤΕ. Η επιλογή αυτή γίνεται με το ίδιο σήμα ελέγχου που επιλέγει και εάν θα κάνουμε πρόσθεση ή αφαίρεση· αυτό σημαίνει ότι δεν μπορούμε να επιλέξουμε ταυτόχρονα συνδυασμούς όπως π.χ. πρόσθεση και ΟΥΤΕ --όμως, προφανώς, κάτι τέτοιο δεν μας ενδιαφέρει να το κάνουμε, αφού η ALU *μίας* μόνο πράξης το αποτέλεσμα μας ενδιαφέρει να βγάζει στην έξοδό της, και ποτέ δύο διαφορετικά αποτελέσματα ταυτόχρονα. Έτσι, ο πρώτος από τους πολυπλέκτες δεξιά επιλέγει αν επιθυμούμε να κρατήσουμε το αποτέλεσμα της αριθμητικής πράξης που κάνει ο αθροιστής, ή το αποτέλεσμα της λογικής πράξης που κάνουν οι πύλες. Και τελικά, ο τελευταίος δεξιά πολυπλέκτης επιλέγει αν θα δοθεί στην έξοδο (α) το αποτέλεσμα της αριθμητικής ή της λογικής πράξης, ή (β) η δεύτερη είσοδος της ALU, **B**, αυτή καθεαυτή χωρίς άλλην επεξεργασία. Την τελευταία αυτή δυνατότητα, που την λέμε "πέρασε το B" (passB), θα δούμε ότι θα την χρειαστούμε σε εντολές όπως η "load", παρακάτω. Έτσι, συνολικά, η ALU μπορεί να κάνει τις παρακάτω πράξεις, ανάλογα με την

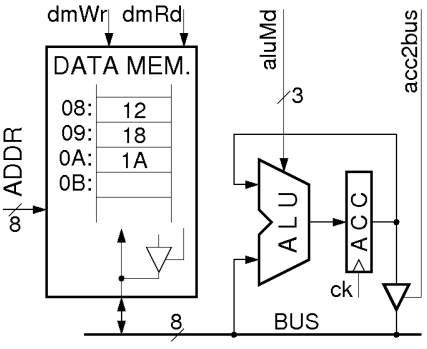
εκάστοτε τιμή των τριών σημάτων ελέγχου της, *mode*. Η ένδειξη "x" στο mode σημαίνει ότι η ALU κάνει την ίδια πράξη όποια τιμή και να έχει το αντίστοιχο bit του mode (συνθήκη αδιαφορίας, που συγγενεύει με όσα είδαμε στην §4.4).

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)

Συχνά, ένας υπολογιστής θέλουμε να έχει τουλάχιστο τρείς λογικές πράξεις: AND, OR, NOT. Ο δικός μας θα έχει μόνο δύο επειδή, όπως θα δούμε πιό κάτω, δέχεται μόνο 16 συνολικά εντολές, και δεν υπήρχε μεταξύ τους χώρος και γιά την OR και γιά την NOT. Όταν χρειαζόμαστε ένα απλό NOT, μπορούμε να κάνουμε NOR με το μηδέν, και όποτε χρειαζόμαστε ένα OR μπορούμε να κάνουμε πρώτα ένα NOR και μετά ένα NOT (δηλ. NOR με μηδέν). Αυτό, ταυτόχρονα, απλοποιεί και το κύκλωμα της ALU και την κωδικοποίηση των mode bits της.

11.2 Πράξεις ALU σ' έναν Επεξεργαστή Τύπου Συσσωρευτή (Accumulator):

Τώρα που έχουμε την ALU όπου θα εκτελούνται οι πράξεις, το επόμενο θέμα είναι πάνω σε ποιούς αριθμούς θα γίνονται αυτές οι πράξεις, πού θα βρίσκονται αυτοί οι αριθμοί, και πώς θα τους επιλέγουμε. Δεδομένου ότι οι υπολογιστές προορίζονται γιά την επεξεργασία μεγάλου όγκου δεδομένων, προφανώς θα χρειαστούμε μία (μεγάλη) μνήμη (§9.3) όπου θα κρατιούνται αυτά τα δεδομένα. Αυτή τη λέμε "**Μνήμη Δεδομένων**" (Data Memory), και φαίνεται στο παρακάτω σχήμα. Γιά να γίνει μία πράξη (π.χ. πρόσθεση), χρειαζόμαστε δύο αριθμούς πάνω στους οποίους θα γίνει η πράξη, και χρειάζεται να τοποθετηθεί κάπου και το αποτέλεσμα της πράξης. Υπάρχουν υπολογιστές που γιά να γίνει αυτό διαβάζουν δύο αριθμούς από τη μνήμη δεδομένων, και γράφουν το αποτέλεσμα επίσης σε κάποια θέση της μνήμης δεδομένων. Γιά μας όμως, κάτι τέτοιο θα ήταν πολύπλοκο, διότι θα απαιτούσε τρείς χωριστές προσπελάσεις στη μνήμη --δύο αναγνώσεις και μία εγγραφή. Εμείς θα ακολουθήσουμε μιάν άλλη "αρχιτεκτονική" υπολογιστή --την απλούστερη που έχει υπάρξει ιστορικά: την *αρχιτεκτονική συσσωρευτή* (accumulator architecture). Στην αρχιτεκτονική αυτή, υπάρχει ένας ειδικός καταχωρητής, έξω από τη μνήμη δεδομένων, ο οποίος κρατάει έναν αριθμό --το αποτέλεσμα της πιό πρόσφατης πράξης. Κάθε καινούργια πράξη γίνεται ανάμεσα σε αυτό το πιό πρόσφατο αποτέλεσμα και σ' ένα καινούργιο αριθμό που διαβάζουμε από τη μνήμη, και αφήνει το αποτελεσμά της πάλι σε αυτόν τον ειδικό καταχωρητή. Έτσι, π.χ., αν κάνουμε συνεχείς προσθέσεις, συσσωρεύεται σε αυτόν τον καταχωρητή το άθροισμα όλων των αριθμών που διαβάσαμε από τη μνήμη και προσθέσαμε, και γι' αυτό ο καταχωρητής αυτός ονομάστηκε "**συσσωρευτής**" (accumulator). Οι σημερινοί υπολογιστές έχουν αρκετούς τέτοιους καταχωρητές --συνήθως 32· τους ονομάζουμε "καταχωρητές γενικού σκοπού" (general-purpose registers), και όχι συσσωρευτές.



Στο σχήμα φαίνεται η απαιτούμενη συνδεσμολογιά γιά να γίνονται οι πράξεις όπως τις περιγράψαμε. Ο συσσωρευτής είναι ένας ακμοπυροδότητος καταχωρητής (§8.5), που σημειώνεται σαν "ACC". Ένα εξωτερικό κύκλωμα, που θα δούμε σε λίγο, τροφοδοτεί τη διεύθυνση ADDR στη μνήμη δεδομένων, καθώς και τα σήματα ελέγχου ανάγνωσης (*dmRd* - **d**ata **m**emory **rea**d enable) και εγγραφής (*dmWr* - **d**ata **m**emory **wr**ite enable), προκαλώντας την ανάγνωση μιάς λέξης (δηλ. ενός αριθμού) από τη θέση μνήμης ADDR. Η ALU παίρνει το περιεχόμενο του συσσωρευτή ACC στη μιά της είσοδο, και τον αριθμό που διαβάσαμε από τη μνήμη στην άλλη· το εξωτερικό κύκλωμα ελέγχου προσδιορίζει, μέσω του *aluMd* (ALU mode), το είδος της πράξης που πρέπει να γίνει, και το αποτέλεσμα της πράξης δίδεται σαν είσοδος στο συσσωρευτή. Όταν έλθει η ενεργή ακμή του ρολογιού, το αποτέλεσμα αυτής της πράξης αντικατθιστά το παλαιό περιεχόμενο του συσσωρευτή. Κατά καιρούς, πρέπει το αποτέλεσμα των πράξεων να αποθηκεύεται (γράφεται) σε μιά επιθυμητή θέση (λέξη) μνήμης, προκειμένου μετά να ξεκινήσει κάποια νέα σειρά πράξεων στο συσσωρευτή. Γιά να γίνεται η αποθήκευση αυτή προβλέφτηκε ένας τρικατάστατος οδηγητής, δεξιά κάτω, ο οποίος τοποθετεί το περιεχόμενο του ACC πάνω στη λεωφόρο (bus), απ' όπου και το παραλαμβάνει η μνήμη γιά να γίνει η εγγραφή. Τα σήματα ελέγχου που πρέπει να ενεργοποιηθούν είναι τα *acc2bus* (ACC προς bus - ACC to bus, όπου το "2" είναι ομόηχο με το "to") και *dmWr*.

11.3 Πρόγραμμα και Εντολές: οι Οδηγίες γιά τις Πράξεις

Γιά να λειτουργήσει το παραπάνω κύκλωμα και να γίνουν οι επιθυμητές πράξεις, πρέπει κάποιος να τροφοδοτεί τις διευθύνσεις, *ADDR*, και τα σήματα ελέγχου των πράξεων, *dmRd, aluMd, acc2bus,* και *dmWr*. Τη δουλειά αυτή αναλαμβάνει ένα άλλο κύκλωμα, που θα δούμε παρακάτω, το οποίο ακολουθεί πιστά τις σχετικές οδηγίες που έχει γράψει ένας άνθρωπος (με τη βοήθεια κάποιου υπολογιστή) και οι οποίες είναι αποθηκευμένες σε μιά μνήμη. Κάθε οδηγία γιά μιά συγκεκριμένη πράξη ή ενέργεια λέγεται **εντολή** (instruction), και το σύνολο των εντολών που έχουν δοθεί σ' έναν υπολογιστή (έχουν γραφτεί στη μνήμη του) και τις οποίες αυτός ακολουθεί σε δεδομένη στιγμή λέμε ότι αποτελούν το **πρόγραμμα** (program) που αυτός "εκτελεί" (executes) ή "τρέχει" (runs) τη δεδομένη στιγμή. Τα κυκλώματα αποτελούν το **υλικό** (hardware) του υπολογιστή, και τα προγράμματα που τρέχουν ή μπορούν να τρέξουν σε αυτόν αποτελούν το **λογισμικό** του (software).

Ο κάθε υπολογιστής "καταλαβαίνει", δηλαδή μπορεί να αναγνωρίσει και να εκτελέσει, ορισμένες μόνο, συγκεκριμένες εντολές ή τύπους εντολών· αυτές τις ονομάζουμε σύνολο ή **ρεπερτόριο εντολών** (instruction set) του υπολογιστή. Οι εντολές του δικού μας υπολογιστή αποτελούνται από δύο κομάτια καθεμία: έναν "κώδικα πράξης" (operation code, ή **opcode** εν συντομία), και μιά **διεύθυνση** ADDR. Κάθε εντολή αποτελείται από 12 bits, από τα οποία τα 4 MS bits είναι ο opcode και τα 8 LS bits είναι η διεύθυνση. Σ' αυτό το εργαστήριο, ο υπολογιστής μας θα έχει το ρεπερτόριο των 8 εντολών που φαίνεται στον παρακάτω πίνακα· μερικές εντολές θα εξηγηθούν σε επόμενες παραγράφους· τις υπόλοιπες 8 εντολές θα τις προσθέσουμε στο επόμενο εργαστήριο.

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 πληκτρολόγι
0101aaaaaaaa	*load ADDR*	ACC <- DM[ADDR]
0110aaaaaaaa	*store ADDR*	DM[ADDR] <- ACC
0111aaaaaaaa	*jump ADDR*	PC <- ADDR (επόμενη εντολή σε ADDR - §11.9)

Στην αριστερή στήλη του πίνακα φαίνεται η κάθε εντολή σε "**Γλώσσα Μηχανής** (machine language, ή object code, ή binary code), δηλαδή όπως αυτή υπάρχει μέσα στη μνήμη (εντολών) του υπολογιστή (άσσοι και μηδενικά). Τα 4 MS (αριστερά) bits της εντολής, όπως είπαμε, είναι ο opcode και υποδεικνύουν το είδος της πράξης. Τα 8 LS (δεξιά) bits της εντολής, που φαίνονται σαν "aaaaaaaa", είναι τα 8 bits της διεύθυνσης ADDR, όπως είπαμε παραπάνω.

Στο μεσαίο μέρος του πίνακα φαίνεται η συμβολική γραφή της εντολής: η πρώτη λέξη είναι το σύμβολο του opcode, ενώ το *ADDR* αντικαθίσταται κάθε φορά από τη συγκεκριμένη διεύθυνση που επιθυμούμε να χρησιμοποιήσουμε --έναν αριθμό από 0 μέχρι 255, αφού οι διευθύνσεις είναι οκτάμπιτες στον υπολογιστή μας. Ένα πρόγραμμα με τις εντολές του γραμμένες με αυτό το συμβολισμό λέμε ότι είναι γραμμένο σε "**Γλώσσα Assembly**". Στη μνήμη του υπολογιστή, φυσικά, το μόνο που υπάρχει είναι άσσοι και μηδενικά, άρα γιά να εκτελεστεί ένα πρόγραμμα Assembly πρέπει πρώτα να μετατραπεί στην δυαδική του αναπαράσταση, δηλαδή στη Γλώσσα Μηχανής. Η μετατροπή αυτή είναι πολύ απλή: κάθε συμβολικός opcode αντικαθίσταται με τον αντίστοιχο δυαδικό του κώδικα, και κάθε διεύθυνση μετατρέπεται στο δυαδικό. Τη μετατροπή αυτή (και μερικές άλλες σχετικές βοηθητικές εργασίες) την κάνει συνήθως ένα μικρό πρόγραμμα υπολογιστή που ονομάζεται **Assembler**.

Στις δεξιές στήλες του πίνακα φαίνεται μία "εξίσωση μεταφοράς καταχωρητών" (register transfer equation), η οποία περιγράφει τις ενέργειες που πρέπει να γίνουν προκειμένου ο υπολογιστής να εκτελέσει την εντολή. Σε αυτές τις εξισώσεις, το αριστερό βέλος υποδεικνύει μεταφορά και εγγραφή πληροφορίας (εκχώρηση - assignment). Ο συμβολισμός "DM[ADDR]" σημαίνει τη θέση (λέξη) της μνήμης δεδομένων (DM) στη διεύθυνση ADDR. Όταν ο συσσωρευτής, ACC, εμφανίζεται και δεξιά και αριστερά από το βέλος, τότε δεξιά μεν σημαίνει το παλαιό περιεχόμενό του (πριν την ακμή ρολογιού), αριστερά δε σημαίνει τη νέα τιμή του (μετά την ακμή ρολογιού) --όπως και στις εκχωρήσεις (assignments) των γλωσσών προγραμματισμού.

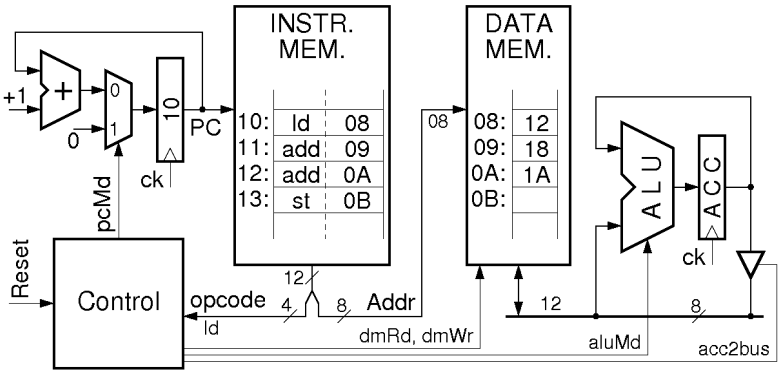
Έτσι, η εντολή *load ADDR* (φόρτωσε) διαβάζει τον αριθμό που περιέχεται στη διεύθυνση *ADDR* της μνήμης, δηλαδή διαβάζει το DM[ADDR], και το αντιγράφει στο συσσωρευτή. Η εντολή *add ADDR* προσθέτει το παλαιό περιεχόμενο του ACC με τον αριθμό που περιέχεται στη διεύθυνση *ADDR* της μνήμης, και γράφει το αποτέλεσμα στον ACC. Αντίστοιχα, οι εντολές *sub, and, nor* κάνουν τις

ανάλογες πράξεις. Τέλος, η εντολή `store ADDR` (αποθήκευσε) αντιγράφει το περιεχόμενο του ACC στη θέση (λέξη) μνήμης με διεύθυνση *ADDR*. Στην παράγραφο 11.8 θα μιλήσουμε γιά την εντολή `input ADDR`, η οποία γράφει ομοίως στη θέση (λέξη) μνήμης με διεύθυνση *ADDR*, αλλά γράφει δεδομένα από μιάν εξωτερική είσοδο (πληκτρολόγιο) και όχι από τον συσσωρευτή. Στην παράγραφο 11.9 θα μιλήσουμε γιά την εντολή `jump ADDR`, η οποία κάνει ώστε η επόμενη εντολή που θα εκτελεστεί να μην είναι η "από κάτω" γραμμένη στη μνήμη εντολών, αλλά μιά άλλη (στη διεύθυνση ADDR).

Γιά παράδειγμα, λοιπόν, αν η μνήμη δεδομένων περιέχει τους αριθμούς που φαίνονται στο παραπάνω σχήμα (διευθύνσεις και δεδομένα φαίνονται στο δεκαεξαδικό), τότε το πρόγραμμα: "`load 08; add 09; add 0A; store 0B;`" θα προκαλέσει τις εξής ενέργειες. Πρώτα θα διαβαστεί ο αριθμός 12 (δεκαεξαδικό) από τη θέση 08 και θα γραφτεί στο συσσωρευτή· μετά, θα διαβαστεί ο αριθμός $18_{(16)}$ και θα προστεθεί στον $12_{(16)}$ παράγοντας το αποτέλεσμα 2A (δεκαεξαδικό) το οποίο και θα γραφτεί στο συσσωρευτή· εν συνεχεία, θα διαβαστεί το 1A από τη θέση 0A, θα προστεθεί στο 2A, και το αποτέλεσμα 44 (δεκαεξαδικό) θα μείνει στο συσσωρευτή· και τέλος, το περιεχόμενο $44_{(16)}$ του συσσωρευτή θα γραφτεί στη θέση μνήμης 0B.

11.4 Ανάγνωση & Εκτέλεση Εντολών:

Γιά να μπορέσει ο υπολογιστής να εκτελέσει τις εντολές του προγράμματος, πρέπει αυτές να είναι κάπου γραμμένες, και από εκεί να τις διαβάζει μία-μία και να τις εκτελεί. Αυτός είναι ο ρόλος των στοιχείων που θα προσθέσουμε εδώ στο κύκλωμα της §11.2, και τα οποία φαίνονται στο επόμενο σχήμα - αριστερό ήμισυ. Το πρόγραμμα είναι αποθηκευμένο στη "**Μνήμη Εντολών**" (Instruction Memory). Κανονικά, οι υπολογιστές χρησιμοποιούν την ίδια μνήμη γιά να αποθηκεύουν τόσο τα δεδομένα όσο και το πρόγραμμα (σε χωριστές διευθύνσεις το καθένα)· με τον τρόπο αυτό, αν έχουμε ένα μικρό πρόγραμμα υπάρχει χώρος γιά πολλά δεδομένα, και αντιστρόφως, αν έχουμε λίγα δεδομένα μπορεί το πρόγραμμα να είναι μεγάλο. Εμείς εδώ χρησιμοποιούμε δύο χωριστές μνήμες, μιά γιά δεδομένα και μιά γιά εντολές, γιά να απλοποιηθεί το κύκλωμα και η λειτουργία του. Γιά να μπορέσει το κύκλωμά μας να εκτελέσει μιάν εντολή, πρέπει να την διαβάσει από τη μνήμη εντολών, και γιά το σκοπό αυτό χρειάζεται τη διεύθυνση της μνήμης αυτής όπου βρίσκεται η επιθυμητή εντολή. Αυτός είναι ο ρόλος του "**Μετρητή Προγράμματος**" (Program Counter - PC): ο καταχωρητής αυτός κρατά τη διεύθυνση της μνήμης εντολών όπου βρίσκεται η εντολή που θέλουμε να εκτελέσουμε.

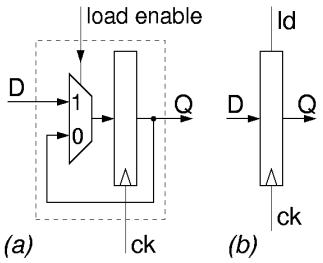


Στο παράδειγμα του σχήματος, ο PC περιέχει τον αριθμό 10 (δεκαεξαδικό), ο οποίος δίδεται σα διεύθυνση στη μνήμη εντολών· η μνήμη εντολών διαβάζει και μας δίνει το περιεχόμενο της θέσης 10, το οποίο εδώ τυχαίνει να είναι η εντολή `load 08` --κωδικοποιημένη σε γλώσσα μηχανής φυσικά, δηλαδή "010100001000" σύμφωνα με το παραπάνω ρεπερτόριο εντολών. Κάθε λέξη της μνήμης εντολών είναι 12 bits, και περιέχει μιάν εντολή. Τα 12 σύρματα ανάγνωσης που βγαίνουν από τη μνήμη αυτή, τα χωρίζουμε σε 4 αριστερά (MS) σύρματα που πηγαίνουν στο κύκλωμα ελέγχου, και 8 δεξιά (LS) σύρματα που πηγαίνουν σα διεύθυνση στη μνήμη δεδομένων. Αφού όλες οι εντολές του υπολογιστή μας αποτελούνται από έναν opcode στα 4 MS bits και μία διεύθυνση στα 8 LS bits, με τη συνδεσμολογία αυτή πηγαίνει ο opcode στο κύκλωμα ελέγχου και η διεύθυνση στη μνήμη δεδομένων. Στο παράδειγμά μας, ο opcode είναι 0101 (που σημαίνει `load`), και η διεύθυνση είναι 00001000 (δηλ. 08 δεκαεξαδικό). Το κύκλωμα ελέγχου, βλέποντας την εντολή `load`, θα ζητήσει ανάγνωση από τη μνήμη δεδομένων (*dmRd=1, dmWr=0, acc2bus=0*) και θα θέσει την ALU σε λειτουργία *passB* (*aluMd=1xx*). Η μνήμη δεδομένων, βλέποντας τη διεύθυνση 08 και ότι της ζητείται ανάγνωση, θα τοποθετήσει τον αριθμό 12 στη λεωφόρο· η ALU, εκτελώντας λειτουργία *passB*, θα περάσει τον αριθμό 12 στην έξοδό της· στην ενεργή ακμή του ρολογιού, ο αριθμός αυτός θα γραφτεί στο συσσωρευτή ACC, ολοκληρώνοντας έτσι την εκτέλεση της εντολής `load 08`.

Μετά την εκτέλεση της εντολής `load 08` από τη θέση 10 της μνήμης εντολών, πρέπει να εκτελεστεί η επόμενη εντολή. Κατά πάγια σύμβαση, εκτός ειδικών εξαιρέσεων που θα δούμε πιό κάτω (§11.9), η επόμενη εντολή βρίσκεται γραμμένη στην ακριβώς επόμενη θέση μνήμης --εδώ, στη διεύθυνση 11. Γιά

να προκύψει η επόμενη αυτή διεύθυνση γιά τη μνήμη εντολών, χρησιμοποιούμε τον αυξητή (incrementor) που φαίνεται αριστερά στο σχήμα, δηλαδή έναν αθροιστή με δεύτερη είσοδο το +1. Έτσι, στην ίδια παραπάνω ενεργή ακμή του ρολογιού που γράφεται ο αριθμός 12 στο συσσωρευτή ACC, γράφεται και το αποτέλεσμα της πρόσθεσης 10+1=11 στον καταχωρητή PC. Το αποτέλεσμα είναι ότι στον επόμενο κύκλο ρολογιού ο PC θα περιέχει 11· η μνήμη εντολών θα διαβάσει και θα μας δώσει το περιεχόμενο "000000001001" της θέσης 11, δηλαδή την εντολή `add 09`· το κύκλωμα ελέγχου, βλέποντας opcode=0000 (`add`), θα δώσει *dmRd=1, dmWr=0, acc2bus=0,* και *aluMd=000* (δηλ. add)· η μνήμη δεδομένων, βλέποντας διεύθυνση 09 και *dmRd=1*, θα διαβάσει και θα δώσει στη λεωφόρο τον αριθμό 18· η ALU, βλέποντας ACC=12, στη λεωφόρο το 18, και *aluMd=add*, θα προσθέσει 12+18 και θα δώσει στην έξοδό της 2A· και ο αθροιστής/αυξητής αριστερά, βλέποντας PC=11, θα δώσει στην έξοδό του 11+1=12. Μόλις έλθει η επόμενη ενεργή ακμή ρολογιού, το 2A θα μπεί στον ACC, και το 12 θα μπεί στον PC, ολοκληρώνοντας έτσι την εκτέλεση της εντολής `add 09`, και προετοιμάζοντάς μας γιά την επόμενη εντολή, `add 0A`, από τη θέση 12. Ο καταχωρητής PC ονομάζεται "Μετρητής Προγράμματος" ακριβώς επειδή είναι κατά βάση ένας μετρητής που αυξάνεται κατά 1 στο τέλος της εκτέλεσης κάθε εντολής γιά να μας δώσει τη διεύθυνση της επόμενης εντολής· ο πολυπλέκτης που υπάρχει στην είσοδό του προορίζεται γιά την αρχικοποίησή του, όταν δίδεται σήμα Reset.

11.5 Καταχωρητές με Επίτρεψη Φόρτωσης (Load Enable)

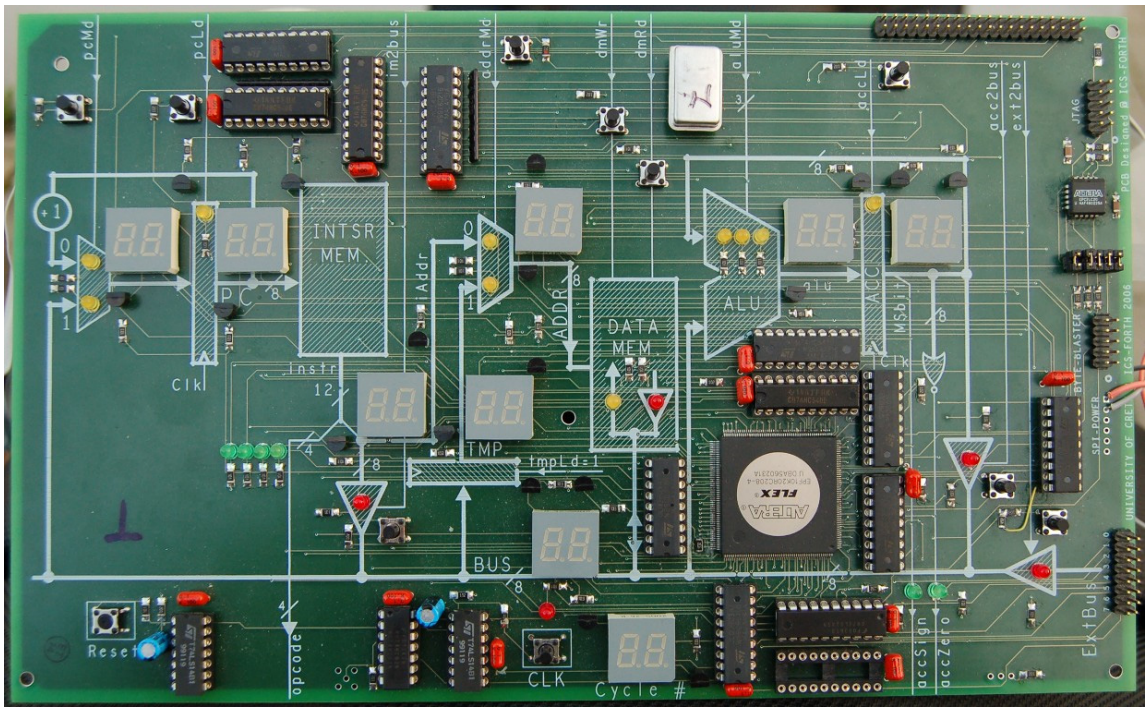


Είδαμε παραπάνω ότι ο απλός υπολογιστής μας χρησιμοποιεί δύο καταχωρητές (ακμοπυροδότητους), τους ACC και PC. Κάθε ακμοπυροδότητος καταχωρητής αποτελείται από πολλαπλά ακμοπυροδότητα flip-flops (εδώ 8 flip-flops, αφού οι καταχωρητές μας είναι οκτάμπιτοι), όπως ένας καταχωρητής μανταλωτών (§7.5) αποτελείται από πολλαπλούς μανταλωτές --έναν γιά κάθε bit. Το κύκλωμα της παραγράφου 8.5 γιά τα ακμοπυροδότητα flip-flops και καταχωρητές ήταν τέτοιο που ανεξαίρετα *σε κάθε* ενεργή ακμή ρολογιού η τιμή των συρμάτων εισόδου έμπαινε μέσα (εγγράφονταν) στον καταχωρητή. Υπάρχουν πολλές περιπτώσεις όπου δεν θέλουμε τέτοια εγγραφή να γίνεται σε κάθε (ενεργή) ακμή ρολογιού, αλλά μόνο στις ακμές εκείνες που εμείς επιλέγουμε --και αυτό θα το χρειαστούμε και στον δικό μας υπολογιστή, ιδιαίτερα στο επόμενο εργαστήριο. Στις περιπτώσεις αυτές, χρησιμοποιούμε το κύκλωμα που φαίνεται στο σχήμα δεξιά (a), και το οποίο συμβολίζουμε με το σύμβολο (b). Το κύκλωμα αυτό έχει ένα σήμα ελέγχου "**επίτρεψης φόρτωσης**" (load enable): όταν το σήμα αυτό είναι 0 (λίγο πρίν και κατά τη διάρκεια της ενεργής ακμής του ρολογιού), τότε η τιμή που είναι αποθηκευμένη στον καταχωρητή **δεν** αλλάζει μετά την ακμή, επειδή στην πραγματικότητα ξαναφορτώνεται η ίδια τιμή. Όταν όμως το σήμα επίτρεψης φόρτωσης είναι 1 (σ' ένα μικρό χρονικό παράθυρο γύρω από την ακμή του ρολογιού), τότε η τιμή των συρμάτων εισόδου, D, εγγράφεται σαν νέα τιμή του καταχωρητή στην ακμή του ρολογιού.

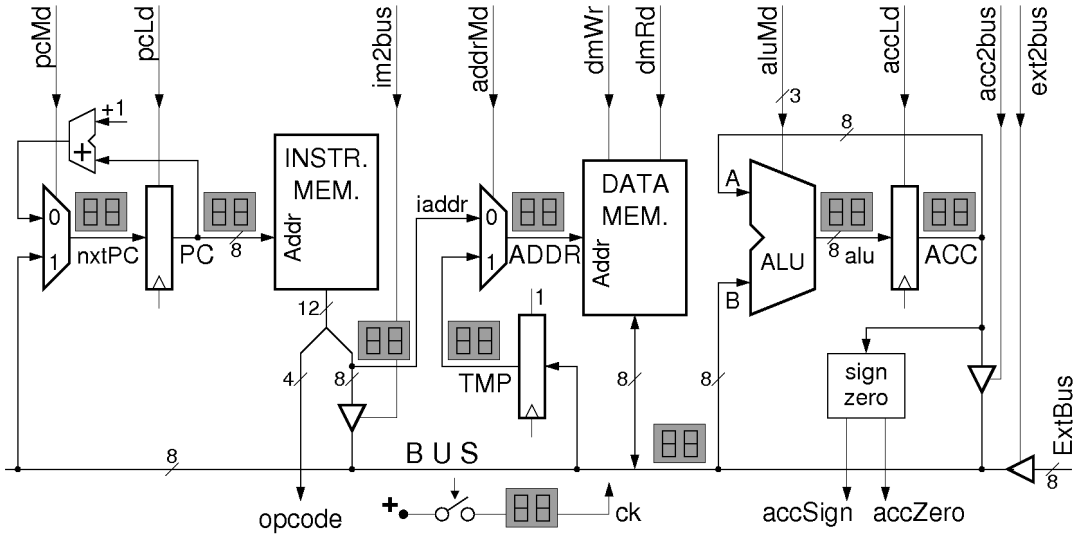
11.6 Ο απλός Υπολογιστής του Εργαστηρίου

Στη φωτογραφία παρακάτω φαίνεται η πλακέτα που υλοποιεί τον "δρόμο δεδομένων" (datapath) του απλού υπολογιστή, την οποία θα βρείτε στο εργαστήριο. Ευχαριστούμε γιά την σχεδίαση και κατασκευή της το Ινστιτούτο Πληροφορικής του ΙΤΕ και τους *Μιχάλη Λυγεράκη, Γιώργο Καλοκαιρινό*, και *Δημήτρη Τσαλιαγκό* (τεχνική περιγραφή του FPGA της πλακέτας (δεν αποτελεί μέρος του μαθήματος): Διπλωματική Εργασία (PDF - 1.1MB) του Δ. Τσαλιαγκού (Σεπ. 2007) [2ο αντίτυπο]). (Ενδέχεται μερικοί φοιτητές να χρειαστεί να κάνουν την άσκηση αυτή στε προηγούμενη έκδοση της πλακέτας --του 2005-- η οποία διαφέρει σε μερικά σημεία).

Αυτό που λείπει και πρέπει εσείς να φτιάξετε είναι το κύκλωμα ελέγχου, που "ενορχηστρώνει" όλες τις λειτουργίες που το datapath είναι σε θέση να κάνει. Το κύκλωμα ελέγχου παίρνει σαν εισόδους τα 4 bits του opcode (κάτω αριστερά) --και αργότερα και 2 bits σύγκρισης του συσσωρευτή με το μηδέν (κάτω δεξιά)-- και δίνει σαν εξόδους όλα τα σήματα ελέγχου (επάνω πλευρά της πλακέτας). Η πλακέτα του datapath έχει περισσότερα στοιχεία από αυτά που είδαμε στην §11.4, προκειμένου να προσφέρει και τις λειτουργίες που θα περιγράψουμε παρακάτω σε αυτό και στο επόμενο εργαστήριο. Το πλήρες κύκλωμα που υλοποιεί η πλακέτα φαίνεται στο σχηματικό διάγραμμα κάτω από τη φωτογραφία. Υπάρχουν εννέα διψήφιοι ενδείκτες 7 τμημάτων (7-segment displays) οι οποίοι σας δείχνουν ανά πάσα στιγμή την παρούσα τιμή σε εννέα ενδιαφέροντα (οκτάμπιτα) σημεία του κυκλώματος· ο κάτω-κάτω (μέση) ενδείκτης μετράει τους κύκλους ρολογιού. Οι ενδείκτες αυτοί λειτουργούν στο **δεκαεξαδικό** --τα ψηφία πάνω από το 9 μοιάζουν με A, b, c, d, E, και F· προσοχή: το 6 διαφέρει από το b κατά το ότι το μεν 6 έχει παύλα επάνω, ενώ το b δεν έχει.



Εκτός από τους διψήφιους ενδείκτες 7 τμημάτων γιά τους οκτάμπιτους δρόμους, υπάρχουν και μικρές LED που δείχνουν τις τιμές μεμονωμένων bits. Οι **κόκκινες LED** δείχνουν ποιός από τους τρικατάστατους οδηγητές που οδηγούν το BUS είναι ενεργοποιημένος. Όταν δεν είναι ενεργοποιημένος κανένας οδηγητής του BUS, ή όταν είναι ενεργοποιημένοι δύο ή περισσότεροι, οπότε η τιμή του BUS είναι απροσδιόριστη, η πλακέτα κατασκευάζει εσωτερικά και δείχει στους ενδείκτες **ψευδοτυχαίες** τιμές, οι οποίες εναλλάσονται με ρυθμό περίπου 2 Hz (η πλακέτα υλοποιεί εσωτερικά το BUS μέσω ενός πολυπλέκτη, οπότε δεν καίγεται όταν ανάβουν δύο ή περισσότεροι οδηγητές --αυτό βέβαια δεν είναι δικαιολογία γιά να οδηγήτε απρόσεκτα το BUS...). Οι **πράσινες LED** δείχνουν τις τιμές των εξόδων της πλακέτας: opcode και τα 2 bits σύγκρισης του συσσωρευτή με το μηδέν. Οι **κίτρινες LED** δείχνουν τις τιμές των υπολοίπων εισόδων (σημάτων ελέγχου) της πλακέτας. Στους δύο πολυπλέκτες, ανάβει πάντα μία από τις δύο LED, υποδεικνύοντας την επιλεγμένη είσοδο. Στους καταχωρητές PC και ACC η LED δείχνει την τιμή του σήματος επίτρεψης φόρτωσης (1 = αναμένη = φόρτωση στην επόμενη ακμή ρολογιού).



Οι είσοδοι των σημάτων ελέγχου (επάνω πλευρά της πλακέτας) έχουν ασθενείς **αντιστάσεις καθέλκυσης** πάνω στην πλακέτα, επομένως αν τις αφήσετε ανοικτοκυκλωμένες (ασύνδετες) παίρνουν την **default τιμή μηδέν (0)**. Αυτό είναι χρήσιμο γιά τους πολυπλέκτες (εκτελούν την συνηθισμένη λειτουργία τους που είδαμε στην §11.4), και γιά τους τρικατάστατους οδηγητές (όλοι σβηστοί --αρκεί να

ανάψετε έναν)· γιά τους καταχωρητές, η default τιμή 0 δεν είναι ιδιαίτερα χρήσιμη, δεδομένου ότι στο σημερινό εργαστήριο θέλουμε πάντα να φορτώνουμε τον PC, και συχνά τον ACC. Δίπλα ή πάνω σε κάθε σήμα ελέγχου (εκτός των aluMd) υπάρχει από ένας διακόπτης "bouton"· όταν πατηθεί ο διακόπτης αυτός **αντιστρέφει** την τιμή του αντίστοιχου σήματος ελέγχου --είτε αυτή είναι η default τιμή 0, είτε προέρχεται από το δικό σας, εξωτερικό κύκλωμα. Αυτό είναι χρήσιμο ιδιαίτερα στα πρώτα σημερινά πειράματα διότι σας επιτρέπει, χωρίς εξωτερικό κύκλωμα ή συνδέσεις, να ενεργοποιήτε χειροκίνητα --πατώντας το σχετικό διακόπτη-- οιοδήποτε σήμα ελέγχου επιθυμείτε (δηλαδή αντιστρέφοντας την default τιμή 0 του). Στα επόμενα πειράματα οι διακόπτες αυτοί μπορεί να σας φανούν χρήσιμοι εάν θελήσετε να "διορθώστε" χειροκίνητα κάποιο σήμα ελέγχου που σε ορισμένες περιπτώσεις το κύκλωμά σας δεν το οδηγεί σωστά.... (Οι παλαιές πλακέτες, έκδοσης 2005, δεν έχουν τέτοιους διακόπτες αντιστροφής πολικότητας).

Οι έξοδοι της πλακέτας (opcode και αποτελέσματα σύγκρισης --κάτω πλευρά) και οι είσοδοι ελέγχου της (επάνω πλευρά, καθώς και το ρολόϊ --βλ. αμέσως παρακάτω) συνδέονται με το breadboard μέσω μίας καλωδιοταινίας που ξεκινά από πάνω δεξιά και που στην κατάληξή της στο breadboard είναι όπως δείχνει η φωτογραφία δεξιά. Κατ' ανάλογο τρόπο προς τα παραπάνω σήματα ελέγχου με το δικόπτη-bouton, στο μέσον της κάτω πλευράς υπάρχει ένας διακόπτης που μπορεί να χρησιμοποιηθεί σαν είσοδος **ρολογιού**: όποτε πατιέται κάνει ck=1 (και ανάβει η κόκκινη LED από πάνω του), και όποτε είναι ελεύθερος δίνει ck=0· ο διακόπτης αυτός είναι debounced (§8.8). Η ακριβής λειτουργία είναι ότι η τιμή που δίνει αυτός ο διακόπτης γίνεται OR'ed με την τιμή που έρχεται από την εξωτερική είσοδο ρολογιού, μέσω της καλωδιοταινίας. Η είσοδος δεδομένων "ExtBus" (δεξιά) συνδέεται με το breadboard μέσω μιάς άλλης καλωδιοταινίας.

Σε σχέση με όσα είδαμε στην §11.4, οι επιπλέον δρόμοι που υπάρχουν στην πλακέτα είναι οι εξής, από αριστερά προς τα δεξιά: **(α)** Ο πολυπλέκτης στην είσοδο του PC, ελεγχόμενος από το pcMd, επιτρέπει η επόμενη εντολή να μην είναι πάντα η "από κάτω" (η "συν ένα") της τωρινής. **(β)** Η *Μνήμη Εντολών* βρίσκεται πάντα σε κατάσταση **ανάγνωσης**, δηλαδή λειτουργεί πάντα σαν συνδυαστικό κύκλωμα --**δεν** μπορείτε να εγγράψετε σε αυτήν (Read-Only Memory - ROM). **(γ)** Ο τρικατάστατος οδηγητής στα 8 LS bits της εντολής, που ελέγχεται από το im2bus, προορίζεται γιά τα άλματα (§11.9) και άλλες λειτουργίες του επόμενου εργαστηρίου. **(δ)** Ομοίως, ο καταχωρητής TMP και ο πολυπλέκτης στην έξοδό του είναι γιά το επόμενο εργαστήριο. **(ε)** Το στοιχείο "sign, zero" στην έξοδο του συσσωρευτή, που γεννά τις εξόδους accSign και accZero, συγκρίνει την τιμή του συσσωρευτή με το μηδέν (ίση, άνιση, αρνητική, θετική ή μηδέν), και θα το χρειαστούμε στις διακλαδώσεις υπό συνθήκη, στο επόμενο εργαστηριο. **(στ)** Τέλος, ο τρικατάστατος οδηγητής δεξιά, ελεγχόμενος από το σήμα ext2bus, προορίζεται γιά την παραλαβή δεδομένωναπό τον έξω κόσμο (ExtBus) --βλ. §11.8.

Πείραμα 11.7a: Γνωριμία με την Πλακέτα και την ALU του απλού Υπολογιστή

Στο πρώτο αυτό πείραμα θα ασχοληθούμε μόνο με την ALU --**δεν** θα βάλουμε την πλακέτα ακόμα να εκτελεί κανονικές εντολές σαν να είναι υπολογιστής. Όταν ανάβουμε την τροφοδοσία, οι μνήμες εντολών και δεδομένων *αρχικοποιούνται* αυτόματα με ορισμένες τιμές (από μία ROM πάνω στην πλακέτα), όπως αυτές δίδονται στον πίνακα, παρακάτω. Επίσης ο PC αρχικοποιείται στο μηδέν (0). Αυτά, καθώς και οι default τιμές των σημάτων ελέγχου διευκολύνουν την έναρξη των πειραμάτων μας.

(α) Συνδέστε με 3 σύρματα τα 3 δεξιά (LS) bits του opcode στα 3 bits του aluMd: έτσι, ο έλεγχος του τι πράξη κάνει η ALU θα γίνεται από τα περιεχόμενα των πρώτων θέσεων της μνήμης εντολών, που έχουν γραφτεί γιά το σκοπό αυτό (δεν πρόκειται γιά σύνδεση κανονικής λειτουργίας υπολογιστή --πρόκειται μόνο γιά σύνδεση πρώτης δοκιμής). **(β)** Κρατήστε τα 3 σήματα ελέγχου pcLd, dmRd, και accLd συνεχώς αναμένα· αυτό μπορείτε να το κάνετε είτε πατώντας συνεχώς τους 3 δικόπτες τους, είτε συνδέοντας στο breadboard τα pcLd, dmRd, και accLd στην θετική τάση τροφοδοσίας (λογικό 1). Έτσι, το BUS θα οδηγείται πάντα από τη μνήμη δεδομένων (dmRd=1, ενώ dmWr έχει την default τιμή 0), η ALU θα κάνει πάνω σε αυτή την τιμή του BUS την πράξη που που της υποδεικνύει ο opcode (α), και σε κάθε πάτημα του διακόπτη-ρολογιού το αποτέλεσμα αυτής της πράξης θα γράφεται στον ACC (accLd=1), ο δε PC θα αυξάνει κατά 1 (nxtPC=PC+1 αφού pcMd έχει την default τιμή 0, και η νέα τιμή PC+1 θα γράφεται πάντα στον PC επειδή pcLd=1). Έτσι, θα πρέπει να παρατηρήσετε τις τιμές στους πρώτους κύκλους

ρολογιού μετά το άναμα της τροφοδοσίας, βάσει των περιεχομένων των πρώτων θέσεων των μνημών εντολών και δεδομένων, όπως αυτά φαίνονται στον παρακάτω πίνακα. Τα περιεχόμενα της μνήμης είναι γραμμένα με τα 4 MS bits (opcode) στο δυαδικό, και τα 8 LS bits (ADDR) στο δεκαεξαδικό:

```
  ADDR   I_MEM:    aluMd:   Αποτέλεσμα μετά την ακμή ρολογιού:

   00:  0100.10    passB    ACC := DM[10] = FD
   01:  1111.10    passB    ACC := DM[10] = FD
   02:  0010.07    and      ACC := ACC and DM[07] = FD AND 0F = 0D
   03:  0011.00    nor      ACC := ACC nor DM[00] = 0D nor 00 = not 0D = F2
   04:  0011.04    nor      ACC := ACC nor DM[04] = F2 nor 08 = not FA = 05
   05:  0000.03    add      ACC := ACC + DM[03] = 05 + 04 = 09
   06:  0000.01    add      ACC := ACC + DM[01] = 09 + 01 = 0A
   07:  0000.0F    add      ACC := ACC + DM[0F] = 0A + FF = 09
   08:  0000.04    add      ACC := ACC + DM[04] = 09 + 08 = 11
   09:  0001.02    sub      ACC := ACC - DM[02] = 11 - 02 = 0F
   0A:  0001.05    sub      ACC := ACC - DM[05] = 0F - 10 = FF
   0B:  0001.01    sub      ACC := ACC - DM[01] = FF - 01 = FE
   0C:  0010.0A    and      ... < συμπληρώστε τον πίνακα εσείς >
   0D:  0011.06    ... < συμπληρώστε τον πίνακα πριν το εργαστήριο >
   0E:  0000.0B    ... < συμπληρώστε πριν το εργαστήριο >
   0F:  0000.06    ... < συμπληρώστε πριν το εργαστήριο >

Αρχικά Περιεχόμενα Μνήμης Δεδομένων (διευθύνσεις & δεδομένα στο δεκαεξαδικό):
 ADDR  D_MEM      ADDR  D_MEM  [ Μετέπειτα Χρήση: ]

  00:  00          08:  12
  01:  01          09:  18
  02:  02          0A:  1A
  03:  04          0B:  F0     [ αργότερα: tmp ]
  04:  08          0C:  F8     [ αργότερα: sum ]
  05:  10          0D:  FC (= -4 αν ερμηνευτεί σαν προσημασμένος)
  06:  80          0E:  FE (= -2 αν ερμηνευτεί σαν προσημασμένος)
  07:  0F          0F:  FF (= -1 αν ερμηνευτεί σαν προσημασμένος)
                   10:  FD     [ αργότερα: δεδομένα εισόδου από πληκτρολόγιο ]
```

Πείραμα 11.7b: Γνωριμία με τα Σήματα Ελέγχου του απλού Υπολογιστή

Αφήστε τα 3 LS bits του opcode να τροφοδοτούν τα 3 LS bits του aluMd, αλλά "ελευθερώστε" τα υπόλοιπα σήματα ελέγχου (δηλ. τα pcLd, dmRd, accLd). Παίξτε κάμποσο με τους διακόπτες των σημάτων ελέγχου, μέχρι να καταλάβετε πώς ελέγχετε εσείς ο ίδιος το τι λειτουργία λέτε στο datapath να κάνει! Φυσικά, δεν πρέπει να είναι αναμένοι ταυτόχρονα δύο ή περισσότεροι οδηγητές (κόκκινα LED) του BUS (αλλοιώς θα δείτε το BUS να παίρνει ψευδοτυχαίες τιμές): σβήστε το dmRd πριν αρχίσετε να ενεργοποιήτε εναλλάξ τα acc2bus, ext2bus, και im2bus. Γιά να καταλάβετε πλήρως τι κάνετε, θα πρέπει να έχετε διαβάσει και τις επόμενες δύο παραγράφους.

(Εάν δουλεύετε σε πλακέτα έκδοσης 2005, όπου δεν υπάρχουν διακόπτες στα σήματα ελέγχου, θα χρειαστεί να τροφοδοτήσετε αυτά τα σήματα από τους διακόπτες της κυρίως πλακέτας, με χρήση του breadboard και των καλωδιοταινιών: Αφήστε (αν θέλετε) το pcLd συνδεδεμένο μονίμως στο λογικό 1. Όμως, τροφοδοτήστε: **(α)** το dmRd από τον διακόπτη M της κυρίως πλακέτας, **(β)** το accLd από τον διακόπτη N της κυρίως πλακέτας, **(γ)** το acc2bus από τον διακόπτη A της κυρίως πλακέτας, **(δ)** το dmWr από τον διακόπτη B της κυρίως πλακέτας, **(ε)** το ext2bus από τον διακόπτη C της κυρίως πλακέτας, και **(στ)** το pcMd από τον διακόπτη D της κυρίως πλακέτας (ίσως να θέλετε να ελέγξετε και το im2bus από κάποιον διακόπτη --πιθανά από τον ίδιο τον D που ελέγχει και το pcMd, οπότε αυτά τα δύο im2bus = pcMd θα είναι μαζί σβηστά ή μαζί αναμένα)).

11.8 Η Εντολή Input: Εξωτερική Είσοδος

Θέλουμε ο υπολογιστής μας να μπορεί να επικοινωνεί με τον έξω κόσμο. Μιά στοιχειώδης έξοδος από τον υπολογιστή παρέχεται απο τους ενδείκτες 7 τμημάτων, μέσω των οποίων μπορούμε να βλέπουμε αριθμητικά αποτελέσματα υπολογισμών. Πρέπει όμως να φροντίσουμε και γιά είσοδο. Στην κάτω δεξιά άκρη της πλακέτας του υπολογιστή, υπάρχει σύνδεση γιά μιά οκτάμπιτη εξωτερική λεωφόρο, ExtBus· αυτή μπορεί να τροφοδοτήσει, μέσω τρικατάστατων οδηγητών, την εσωτερική λεωφόρο, BUS. Στο εργαστήριο, θα βρείτε έτοιμη μιά στενή καλωδιοταινία που συνδέει αυτούς τους 8 ακροδέκτες με τα 8 LS bits που έρχονται από το πληκτρολόγιο στο breadboard ("FROM KEYBOARD"). Όταν θέλουμε να έλθει μιά τέτοια εξωτερική πληροφορία (ένας οκτάμπιτος αριθμός) μέσα στον υπολογιστή μας, δεν έχουμε παρά να ανάψουμε τον οδηγητή που ελέγχεται από το σήμα ext2bus· τότε, η πληροφορία αυτή θα

τοποθετηθεί στο BUS.

Μέσα πλεόν στον υπολογιστή μας, τι θα κάνουμε την εξωτερική πληροφορία που έφτασε στο BUS; Η απάντηση είναι απλή: πρόκειται γιά κατάσταση αντίστοιχη αυτής όταν ο συσσωρευτής, ACC, βάζει την δική του πληροφορία στο BUS --την πληροφορία αυτή την αποθηκεύουμε στη θέση μνήμης με διεύθυνση ADDR, όπου ADDR είναι τα 8 LS bits της εντολής. Έτσι λοιπόν, η εντολή "input ADDR" θα κάνει: ExtBus --> BUS --> DM[ADDR], δηλαδή, πέρα από το άναμα του ext2bus, πρέπει να ανάψουμε και το dmWr γιά να γίνει εγγραφή στη μνήμη δεδομένων.

11.9 Άλμα (Jump): Συνέχιση Εκτέλεσης σε άλλο Σημείο

Εάν οι υπολογιστές το μόνο που έκαναν ήταν να εκτελούν την μία εντολή μετά την άλλη, στη σειρά, πολύ γρήγορα θα έφταναν στο τέλος της μνήμης και δεν θα είχαν άλλες εντολές να εκτελέσουν. Η μεγάλη δύναμη των υπολογιστών είναι η **επανάληψη** των ίδιων εντολών (του ίδιου αλγόριθμου!) δρώντας κάθε φορά πάνω σε διαφορετικά δεδομένα. Γιά να επιτευχθεί αυτή η επανάληψη, χρειάζεται ένας μηχανισμός που να επιτρέπει στον υπολογιστή η επόμενη εντολή που θα εκτελέσει να *μην* είναι η "από κάτω", αλλά μιά άλλη εντολή σε αυθαίρετο σημείο της μνήμης εντολών. Την δυνατότητα αυτή μας παρέχουν οι εντολές **άλματος** (jump) --που εκτελούνται πάντα, χωρίς συνθήκη, και γιά τις οποίες θα μιλήσουμε εδώ-- και οι εντολές **διακλάδωσης** (branch) --που εκτελούνται υπό ορισμένες συνθήκες μόνο, και οι οποίες θα μας απασχολήσουν στο επόμενο εργαστήριο.

Η εντολή "jump ADDR" έχει μιάν απλή αποστολή: η επόμενη εντολή που θα εκτελεστεί μετά από αυτήν **δεν** θα είναι η "από κάτω" εντολή της jump (η εντολή που είναι γραμμένη στην επόμενη διεύθυνση μνήμης), αλλά μιά άλλη εντολή, σε μιάν *αυθαίρετη* διεύθυνση ADDR της μνήμης εντολών. Δεδομένου ότι ο υπολογιστής πάντα διαβάζει και εκτελεί την εντολή που βρίσκεται στη διεύθυνση της μνήμης εντολών την οποία διεύθυνση περιέχει ο PC, προκύπτει ότι ο ρόλος της "jump ADDR" είναι να *μην* φορτώσει στον PC την παλαιά του τιμή συν 1, αλλά αντ' αυτής να φορτώσει εκεί την διεύθυνση ADDR. Γιά να επιτευχθεί αυτό, το datapath μας έχει τον οδηγητή που ελέγχεται από το σήμα im2bus, ο οποίος μπορεί να βάλει την ADDR πάνω στο BUS, και τον πολυπλέκτη στην είσοδο του PC, ο οποίος μπορεί να κάνει τον PC να φορτωθεί από το BUS και όχι από τον αθροιστή που δίνει την παλαιά του τιμή συν 1.

11.10 Αποκωδικοποίηση Εντολών - Κύκλωμα Ελέγχου

Το μόνο που λείπει πλέον γιά να δουλέψει ο υπολογιστής που φτιάξαμε (με τις 8 εντολές που είδαμε μέχρι στιγμής --στο επόμενο εργαστήριο θα προσθέσουμε και άλλες 8 που λείπουν) είναι το κύκλωμα **ελέγχου** (control). Αυτό είναι υπεύθυνο γιά τη δημιουργία όλων των σημάτων ελέγχου που λένε σε κάθε μονάδα τι να κάνει κάθε φορά. Όλες οι εντολές που είδαμε μέχρι στιγμής διαβάζονται και εκτελούνται σε έναν κύκλο ρολογιού η καθεμία, και γι' αυτό το κύκλωμα ελέγχου, μέχρι στιγμής, είναι ένα απλό συνδυαστικό κύκλωμα. Ο πίνακας αληθείας του προκύπτει αν σκεφτούμε τις εργασίες που πρέπει να γίνουν γιά την εκτέλεση κάθε εντολής:

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	1	0	0
0101 (load)	ACC:=DM[A]	1	1xx	1	0	0	0	0	0
0110 (store)	DM[A]:=ACC	0	xxx	0	1	0	1	0	0
0111 (jump)	PC := A	0	xxx	0	0	0	0	1	1

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

Οι εντολές `load` και `add` εκτελούνται όπως περιγράψαμε παραπάνω. Οι εντολές `sub, and, nor` εκτελούνται κατά εντελώς ανάλογο τρόπο --απλώς αλλάζει το mode της ALU. Η εντολή `store` διαφέρει λίγο: ανάβοντας το *acc2bus=1* (με *dmRd=0*, φυσικά), τοποθετεί την τιμή του συσσωρευτή στο bus· ενεργοποιώντας το *dmWr=1*, η τιμή αυτή από το bus εγγράφεται στη μνήμη δεδομένων· επίσης, σβήνοντας το *accLd=0*, ο ACC διατηρεί την τιμή του αμετάβλητη. Υπάρχει μιά λεπτομέρεια που δεν είναι σωστή σε αυτό το συνδυαστικό τρόπο γέννησης του σήματος *dmWr*: δεν υπάρχει εγγύηση ότι το σήμα αυτό θα ανάψει *μετά* τη σταθεροποίηση της διεύθυνσης της μνήμης δεδομένων και θα σβήσει *πριν* την επόμενη αλλαγή αυτής της διεύθυνσης, όπως πρέπει να γίνει. Το πρόβλημα αυτό δεν μπορεί να διορθωθεί παρά μόνο αν αλλάξει το κύκλωμα ελέγχου και γίνει ακολουθιακό (FSM), ή με άλλες πολύπλοκες μεθόδους· η πλακέτα του εργαστηρίου λύνει αυτό το πρόβλημα με μία τέτοια περίπλοκη

μέθοδο που δεν σας αφορά. Κανονικά, ένας υπολογιστής χρειάζεται και ένα σήμα Reset που να τον επαναφέρει στην αρχική κατάσταση εκκίνησης, ό,τι κι αν έκανε αυτός πριν (opcode=xxxx): να αρχικοποιεί τον PC στο 0, γιά να αρχίσει να εκτελεί εντολές από την αρχή της μνήμης εντολών. Στην πλακέτα του εργαστηρίου, γιά διευκόλυνσή σας, "Reset" γίνεται αυτόματα όποτε ανάβει η τροφοδοσία, καθώς και όποτε πατήσετε το σχετικό μικρό κουμπί, κάτω αριστερά στην πλακέτα.

Πείραμα 11.11: Σχεδίαση και Δοκιμή του Κυκλώματος Ελέγχου

Πριν φτάσετε στο εργαστήριο, σχεδιάστε το (συνδυαστικό) κύκλωμα του ελέγχου βάσει του παραπάνω πίνακα αληθείας, και την υλοποίησή του με πύλες από τα chips που έχετε. Παρ' ότι ο παραπάνω πίνακας αληθείας δείχνει ολους τους opcodes να αρχίζουν με 0 (και δεν προσδιορίζει τι πρέπει να συμβεί όταν ο opcode (είσοδος στο κύκλωμά σας) αρχίζει με 1), εσείς **αγνοήστε** (don't care) αυτό το ένα MS bit του opcode. Αυτό, πρώτον απλοποιεί το κύκλωμά σας, και δεύτερον είναι απαραίτητο διότι στη μνήμη εντολών της πλακέτας υπάρχει μιά εντολή που αρχίζει με 1 (η εντολή 1111 στη θέση 01), η οποία είναι εκεί διότι την χρειαζόμαστε στο επόμενο εργαστήριο (στην πραγματικότητα είναι η εντολή "jump indexed"), αλλά σε αυτό εδώ το εργαστηριο εσείς πρέπει να την θεωρήσετε ότι είναι μία κανονική jump, σαν ο opcode της να ήταν 0111, δηλαδή αγνοώντας το ένα MS bit του opcode.

Στο εργαστήριο, κατασκευάστε το κύκλωμα ελέγχου στο breadboard, συνδέστε το στην πλακέτα του υπολογιστή, και ελέγξτε το εκτελώντας το πρόγραμμα που υπάρχει στη μνήμη εντολών και φαίνεται στον παρακάτω πίνακα. Όταν ο PC είναι μηδέν (0), μην ξεχνάτε να δίνετε από το πληκτρολόγιο έναν "ενδιαφέροντα" οκτάμπιτο αριθμό.

```
  ADDR   I_MEM:    Assembly:    Αποτέλεσμα μετά την ακμή ρολογιού:

   00:  0100.10    input 10     DM[10] = input := δεδομένα εισόδου από πληκτρολόγιο
   01:  1111.10    jump  10     PC := 10
   ....
   10:  0101.08    load  08     ACC := DM[08] = 12
   11:  0000.09    add   09     ACC := ACC + DM[09] = 12 + 18 = 2A
   12:  0000.0A    add   0A     ACC := ACC + DM[0A] = 2A + 1A = 44
   13:  0110.0B    store 0B     DM[0B] = tmp := ACC = 44
   14:  0001.01    sub   01     ACC := ACC - DM[01] = 44 - 01 = 43
   15:  0010.07    and   07     ACC := ACC and DM[07] = 43 and 0F = 03
   16:  0011.00    nor   00     ACC := ACC nor DM[00] = 03 nor 00 = not 03 = FC
   17:  0011.01    nor   01     ACC := ACC nor DM[01] = FC nor 01 = not FD = 02
   18:  0000.10    add   10     ACC := ACC + DM[10] = 02 + input
   19:  0000.0C    add   0C     ACC := ACC + DM[0C] = 02 + input + sum
   1A:  0110.0C    store 0C     DM[0C] = sum := ACC = sum + input + 2
   1B:  0111.00    jump  00     PC := 00  [άπειρος βρόχος άθροισης]

Αρχικά Περιεχόμενα Μνήμης Δεδομένων (διευθ. & δεδομένα στο δεκαεξαδικό):
  ADDR  D_MEM      ADDR  D_MEM  [ Χρήση: ]

   00:  00          08:  12
   01:  01          09:  18
   02:  02          0A:  1A
   03:  04          0B:  F0     [ tmp ]
   04:  08          0C:  F8     [ sum ]
   05:  10          0D:  FC (= -4 αν ερμηνευτεί σαν προσημασμένος)
   06:  80          0E:  FE (= -2 αν ερμηνευτεί σαν προσημασμένος)
   07:  0F          0F:  FF (= -1 αν ερμηνευτεί σαν προσημασμένος)
                    10:  FD     [ input από πληκτρολόγιο ]
```

Up to the Home Page of CS-120

last updated: 9 Dec. 2007, by M. Katevenis.