ΧΑΡΙΔΗΜΟΣ Θ. ΒΕΡΓΟΣ
ΕΠΙΚΟΥΡΟΣ ΚΑΘΗΓΗΤΗΣ

Πανεπιστημιακές Παραδόσεις στην
ΕΙΣΑΓΩΓΗ ΣΤΑ ΣΥΣΤΗΜΑΤΑ ΥΠΟΛΟΓΙΣΤΩΝ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΕΡΓΑΣΤΗΡΙΟ ΤΕΧΝΟΛΟΓΙΑΣ & ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΚΔΟΣΗ 3.1
ΟΚΤΩΒΡΙΟΣ 2006

ΠΡΟΛΟΓΟΣ

Οι ηλεκτρονικοί υπολογιστές τα τελευταία χρόνια διαδραματίζουν έναν ολοένα και σημαντικότερο ρόλο στην καθημερινή μας ζωή. Τα αποτελέσματα των πανελληνίων εξετάσεων, τα εκκαθαριστικά σημειώματα της εφορίας, η μισθοδοσία χιλιάδων υπαλλήλων κ.α. αποτελούν παραδείγματα προβλημάτων που θα απαιτούσαν πολύ περισσότερο κόπο και χρόνο χωρίς τους ηλεκτρονικούς υπολογιστές. Σε αρκετές περιπτώσεις πλέον, οι υπολογιστές είναι συνυφασμένοι με αυτή την ίδια την ανθρώπινη ζωή, όπως για παράδειγμα στην τηλεϊατρική, τα χειρουργικά ρομπότ, τα συστήματα ελέγχου πυρηνικών κεφαλών, τα στρατιωτικά συστήματα, τις τηλεπικοινωνίες, τον έλεγχο αεροδιαδρόμων κλπ.

Μοιραία, ο ρόλος του Μηχανικού Η/Υ & Πληροφορικής αποκτά αυξανόμενη βαρύτητα μέσα στη σημερινή κοινωνία. Για να ανταποκριθεί σε αυτές τις ανάγκες ο Μηχανικός Η/Υ & Πληροφορικής θα πρέπει να έχει ικανό υπόβαθρο για να ανταποκρίνεται στα υπάρχοντα προβλήματα, μα κυρίως, για να οραματίζεται και να σχεδιάζει τις λύσεις για τα προβλήματα που ανακύπτουν.

Το μάθημα "Εισαγωγή στα Συστήματα Υπολογιστών" έχει σκοπό να εισάγει τις αρχές και την ορολογία της Επιστήμης των Υπολογιστών. Είναι ένα μάθημα δημιουργίας πρώτης ύλης, την οποία τα κατοπινά μαθήματα του Τμήματός μας θα εξειδικεύσουν για να δώσουν τις πραγματικές διαστάσεις της επιστήμης μας.

Οι πανεπιστημιακές παραδόσεις που κρατάτε στα χέρια σας ξεκίνησαν να γράφονται το Μάρτη του 2001˙ ανακαινίστηκαν τον επόμενο Φλεβάρη, βάσει των υποδείξεων των πρώτων αναγνωστών (φοιτητών και μεταπτυχιακών φοιτητών), τους οποίους και ευχαριστώ θερμά. Η έκδοση 3.0 ήταν η πρώτη που περιέλαβε ένα εκτενές σύνολο ενδεικτικών ασκήσεων. Η τρέχουσα έκδοση έχει μικρές επιπλέον διορθώσεις και προσθήκες.

Σκοπός των σημειώσεων είναι να αποτελέσουν ένα εργαλείο σύντομης αναδρομής από τους φοιτητές. Σαν τέτοιο, δε μπορούν –δε στοχεύουν άλλωστε– να υποκαταστήσουν τα όσα διδάσκονται στο αμφιθέατρο˙ κυρίως αδυνατούν να συμπεριλάβουν τις γόνιμες συζητήσεις που διεξάγονται εκεί. Εύχομαι να αποδειχθούν ένα χρήσιμο εργαλείο.

Χ. Βέργος

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή

Δοθέντων (α) μιας συνάρτησης f(x) και (β) μιας τιμής για το x, ένας *υπολογισμός* είναι η εύρεση της τιμής της συνάρτησης για τη δεδομένη τιμή x. Ο εκτελών τη διαδικασία του υπολογισμού ονομάζεται *υπολογιστής*. Τον τελευταίο αιώνα, κατασκευάζοντας μηχανές που στην αρχή αποτελούνταν από συνδυασμό ηλεκτρικών – ηλεκτρονικών στοιχείων και μηχανικών μερών, ενώ στις μέρες μας πλέον σχεδόν αποκλειστικά με ηλεκτρονικά στοιχεία, καταφέραμε πολλές από τις συναρτήσεις που συχνά μας απασχολούν στην καθημερινή μας ζωή να τις υπολογίζουμε πιο εύκολα, πιο αξιόπιστα και κυρίως πιο γρήγορα. Οι μηχανές αυτές ονομάστηκαν *ηλεκτρονικοί υπολογιστές* (*υπολογιστικά συστήματα*).

Η μεγάλη διαφορά μεταξύ ενός ηλεκτρονικού υπολογιστή και κάποιας άλλης από τις ηλεκτρονικές μηχανές που μας περιβάλλουν είναι η δυνατότητα που μας δίνει για την αλλαγή της προς υπολογισμό συνάρτησης (versatility). Αυτό επιτυγχάνεται με το να έχουμε δώσει στον υπολογιστή τη δυνατότητα πραγματικής υλοποίησης ελάχιστων πρωταρχικών συναρτήσεων (primitives) και την ανάγκη από μέρους μας περιγραφής κάθε άλλης πιο σύνθετης συνάρτησης σαν μια ακολουθία αυτών των πρωταρχικών συναρτήσεων. Οπως δηλαδή οι άνθρωποι έχουν ένα αλφάβητο πάνω από το οποίο χτίζουν τις λέξεις που κατοπινά τις χρησιμοποιούν για να χτίσουν εκφράσεις, έτσι και ο υπολογιστής έχει ένα πολύ συγκεκριμένο και περιορισμένο αλφάβητο συναρτήσεων, το οποίο χρησιμοποιούμε για να φτιάξουμε κάποια πιο σύνθετη συνάρτηση. Υποθέστε για παράδειγμα ότι έχετε ένα σύστημα που μπορεί να υλοποιήσει τις συναρτήσεις πρόσθεσης $f_1(A, B) = A + B$, αφαίρεσης $f_2(A, B) = A - B$ και σύγκρισης με το 0, $f_3(A) = 0$ αν $A \leq 0$ και 1 αλλιώς. Εστω τώρα ότι θέλουμε να υλοποιήσουμε τις $f_4(A, B, C) = A + B - C$ και $f_5(A, B) = A * B$. Τότε θα είχαμε ότι $f_4(A, B, C) = f_2(f_1(A, B), C)$ και $f_5(A, B) =$

$f_1(A, f_5(A, B-1))$ αν $f_3(f_2(B, 1)) = 1$ ή $f_1(A, 0)$ αλλιώς. Στα παραπάνω παραδείγματα περιγράψαμε νέες συναρτήσεις εφαρμόζοντας με συγκεκριμένη χρονική ακολουθία τις ήδη υπάρχουσες. Η διαδικασία αυτή ονομάζεται *προγραμματισμός* και η έκφραση των νέων συναρτήσεων σε μια γλώσσα η οποία είναι κατανοητή από τον υπολογιστή ονομάζεται *πρόγραμμα*.

1.1 Το μοντέλο του von Neumann

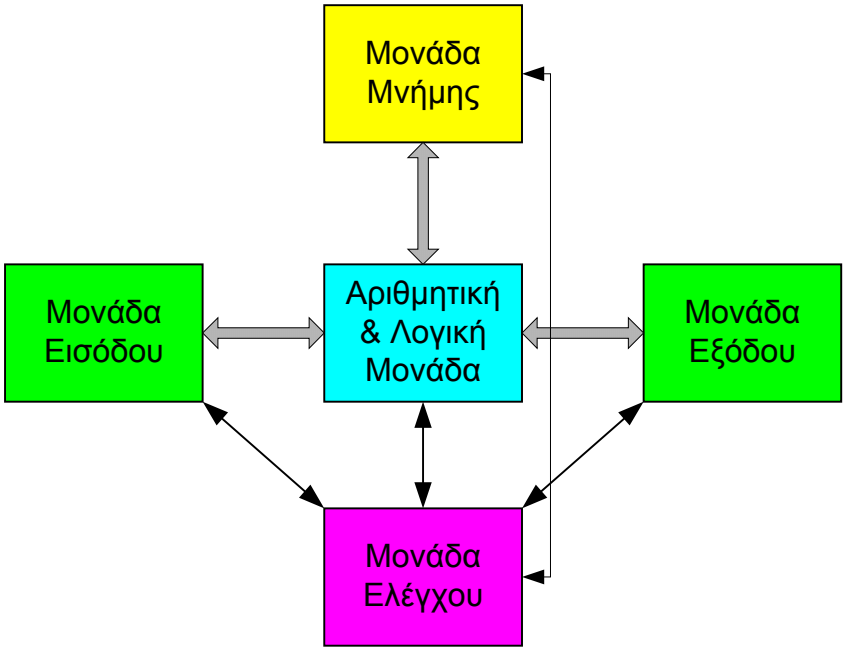
Στα πρώτα χρόνια παρουσίας των ηλεκτρονικών υπολογιστών ο προγραμματισμός γινόταν με την τοποθέτηση χιλιάδων διακοπτών σε συγκεκριμένες θέσεις. Κάθε διακόπτης είχε δύο θέσεις. Η τοποθέτηση στη μία ή την άλλη θέση σήμαινε και την ανάκληση και συνεπώς εκτέλεση διαφορετικής πρωταρχικής συνάρτησης από τη μηχανή, συνεπώς και την υλοποίηση διαφορετικού υπολογισμού. Αφού όλοι οι διακόπτες είχαν τεθεί στην επιθυμητή θέση, το ακόλουθο βήμα ήταν να δώσουμε στη μηχανή τις τιμές των μεταβλητών της συνάρτησης (τιμές εισόδου). Η μηχανή εκτελούσε τον υπολογισμό και παρήγαγε τα αποτελέσματα –στην ουσία καθόριζε τις θέσεις κάποιων διακοπτών-, μας έδινε δηλαδή τις τιμές εξόδου.

Υπήρχαν διάφορα προβλήματα σε αυτή τη πρώτη μορφή των υπολογιστών:

- Συνήθως ο ίδιος υπολογισμός χρειαζόταν να γίνει σε περισσότερα του ενός δεδομένα (π.χ. ο υπολογισμός του δώρου του Πάσχα χρειάζεται να γίνει πάνω σε πολλά μισθολογικά κλιμάκια). Συνεπώς θα έπρεπε με κάποιον τρόπο να μπορούμε τα δεδομένα εισόδου να τα δίνουμε όλα μαζί στη μηχανή, αντί να αλλάζουμε τη θέση διακοπτών για κάθε ένα. Αντίστοιχα θα θέλαμε τα αποτελέσματα να μπορούμε να τα έχουμε σε μια πιο μόνιμη μορφή, αντί να χρειάζεται να σταματάμε τη μηχανή, να διαβάζουμε τη θέση διακοπτών για να πάρουμε το αποτέλεσμα και να επανεκκινούμε τη μηχανή.
- Μεγαλύτερο πρόβλημα ήταν ότι ο προγραμματισμός που είχαμε κάνει στη μηχανή χανόταν κάθε φορά που την προγραμματίζαμε εκ νέου. Ως αποτέλεσμα κάθε πρόγραμμα που τυχόν είχαμε συντάξει για τη μηχανή θα έπρεπε να επανεισάγεται κάθε φορά πριν τη χρήση του. Η εισαγωγή ωστόσο ενός προγράμματος με την αλλαγή θέσης χιλιάδων διακοπτών ήταν μια διαδικασία επίπονη, χρονοβόρα και διόλου αξιόπιστη.

Τα δύο παραπάνω προβλήματα λύθηκαν με την εισαγωγή ενός νέου μοντέλου υπολογιστών από τον Ούγγρο μαθηματικό John Louis von Neumann (1903 – 1957). (Πολλοί ιστορικοί διαφωνούν σχετικά με τη πατρότητα του μοντέλου και το θεωρούν ως αποτέλεσμα συνολικής έρευνας μιας ομάδας στο Πανεπιστήμιο του Princeton στην οποία συμμετείχαν εκτός του von Neumann, ο J. Prespert Eckert

και ο John Mauchly. Αυτοί οι τρεις μαζί με τον Alan Turing ο οποίος εργαζόταν μόνος του την ίδια εποχή στην Αγγλία θεωρούνται οι πατέρες της επιστήμης των υπολογιστών και συνεχιστές της θεωρίας που είχε τον προπερασμένο αιώνα εισάγει ο Charles Babbage (1792 – 1871)). Το καινούργιο στοιχείο στο μοντέλο του von Neumann ήταν η πρόσθεση μιας μονάδας η οποία μπορούσε να αποθηκεύσει προγράμματα, τιμές εισόδου και εξόδου, η οποία ονομάστηκε *μονάδα μνήμης.* Το μοντέλο του von Neumann το οποίο φαίνεται στο παρακάτω σχήμα αποτελείται από 5 διακριτές υπομονάδες (με χοντρές γραμμές φαίνονται τα μονοπάτια ανταλλαγής δεδομένων, ενώ με λεπτές τα μονοπάτια ανταλλαγής σημάτων ελέγχου) :



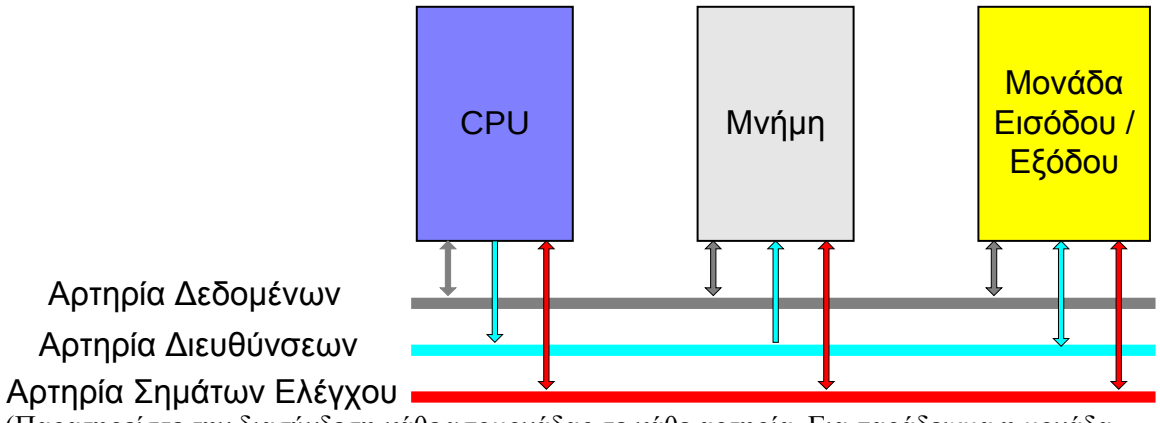
Ας δούμε αναλυτικά πως λειτουργεί το παραπάνω μοντέλο για να καταλάβουμε το ρόλο που καλείται να διαδραματίσει κάθε υπομονάδα. Χρησιμοποιώντας τη ***μονάδα εισόδου (Input Unit)***, εισάγουμε τα ***δεδομένα (data)*** στο υπολογιστικό σύστημα. Τα δεδομένα είναι πλέον τόσο οι ***εντολές (instructions)*** οι οποίες καθορίζουν την ακολουθία εκτέλεσης των πρωταρχικών συναρτήσεων της μηχανής αλλά και οι τιμές των μεταβλητών εισόδου. Τα δεδομένα αυτά αποθηκεύονται στη ***μονάδα μνήμης (Memory Unit)***. Το αποθηκευμένο πρόγραμμα όταν αρχίσει να εκτελείται, ταυτόχρονα περιγράφει και την διαδικασία ανάκλησης των μεταβλητών από τη μνήμη. Η εκτέλεση μιας εντολής του προγράμματος λαμβάνει χώρα στην ***Αριθμητική Λογική Μονάδα (Arithmetic Logic Unit – ALU)*** κάτω από την επίβλεψη της ***Μονάδας Ελέγχου (Control Unit)***. Κάθε αποτέλεσμα που παράγεται προωθείται στη ***Μονάδα Εξόδου (Output Unit)***. Πολλές φορές η ALU μαζί με την μονάδα ελέγχου

αναφέρονται συνολικά ως ***Κεντρική Μονάδα Επεξεργασίας - ΚΜΕ (Central Processing Unit – CPU).***

Το πιο ενδιαφέρον σημείο του μοντέλου του von Neumann είναι ο χειρισμός του προγράμματος κατά τον ίδιο τρόπο με τα υπόλοιπα δεδομένα. Δηλαδή ένα πρόγραμμα μπορεί να αποθηκεύεται στη μνήμη και να ανακαλείται όπως ακριβώς και μια μεταβλητή. Αν και σήμερα αυτό μας φαίνεται κάτι προφανές, υπήρξαν πολλά στάδια πριν γίνει κοινός τόπος. Τα προγράμματα σε παλιότερα συστήματα εισάγονταν κάθε φορά που απαιτείτο εκτέλεσή τους με μέσα όπως διάτρητες κάρτες, μαγνητικές ταινίες κλπ. Αυτή η ιδέα του von Neumann αποτέλεσε θεμέλιο για τη δημιουργία μεταγλωττιστών και λειτουργικών συστημάτων και είναι αυτή που κάνει τόσο ευέλικτους τους σημερινούς υπολογιστές.

1.2 Το μοντέλο αρτηριών συστήματος

Εκσυγχρονίζοντας και τυποποιώντας περισσότερο το μοντέλο του von Neumann μπορούμε να καταλήξουμε στο μοντέλο αρτηριών συστήματος το οποίο παρουσιάζεται στο παρακάτω σχήμα. Βάσει αυτού του μοντέλου ένα υπολογιστικό σύστημα χωρίζεται πλέον σε 3 υπομονάδες : την CPU που πλέον αντικαθιστά την ALU και την μονάδα ελέγχου, την μονάδα εισόδου / εξόδου (Input / Output Unit – I/O Unit) και την μονάδα μνήμης. Πληροφορίες (δεδομένα και πληροφορίες ελέγχου) ανταλλάσσονται βάσει αυτού του μοντέλου πάνω από 3 διαμοιραζόμενες αρτηρίες του συστήματος : την ***αρτηρία διευθύνσεων (Address Bus)***, την ***αρτηρία δεδομένων (Data Bus)*** και την ***αρτηρία ελέγχου (Control Bus).***



(Παρατηρείστε την διασύνδεση κάθε υπομονάδας σε κάθε αρτηρία. Για παράδειγμα η μονάδα μνήμης μπορεί μόνο να "διαβάσει" από την αρτηρία διευθύνσεων μα όχι και να την οδηγήσει. Υπάρχει επίσης και η αρτηρία παροχής τάσης η οποία υπονοείται).

Το παραπάνω μοντέλο βασίζεται στη θεώρηση ότι κάθε πληροφορία μέσα σε ένα υπολογιστικό σύστημα μπορεί να αναγνωριστεί μοναδικά βάσει της διευθύνσεώς της. Οπως για παράδειγμα καθένας μας έχει έναν αριθμό μητρώου ή έναν αριθμό αστυνομικής ταυτότητας που τον διαχωρίζει από όλους τους υπόλοιπους, έτσι και κάθε πληροφορία (εντολή ή δεδομένο) στο υπολογιστικό σύστημα έχει μια διεύθυνση. Ετσι στην αρτηρία δεδομένων διακινούνται οι πληροφορίες ενώ σε αυτήν των διευθύνσεων οι αντίστοιχες διευθύνσεις. Το παραπάνω μοντέλο περιγράφει ικανοποιητικά το σύνολο των σημερινών ηλεκτρονικών υπολογιστών. Στα κατοπινά κεφάλαια γίνεται αναλυτική παρουσίαση των λειτουργικών κομματιών που απαρτίζουν κάθε μία από τις 3 υπομονάδες καθώς και περιγράφονται οι τρόποι επικοινωνίας τους πάνω από τις διαμοιραζόμενες αρτηρίες του συστήματος.

1.3 Ο υπολογιστής σαν ιεραρχικό σύστημα

Οπως κάθε σύνθετο σύστημα έτσι και ο υπολογιστής είναι ένα ιεραρχικό σύστημα. Κάθε επίπεδο αυτής της ιεραρχίας στην ουσία εκμεταλλεύεται τις υπηρεσίες που του παρέχουν τα προηγούμενα (πιο κάτω στην ιεραρχία) επίπεδα και αναπτύσσει νέες υπηρεσίες που τις προσφέρει στο πιο πάνω και πιο κοντινό στον άνθρωπο επίπεδο. Σε ένα υπολογιστικό σύστημα μπορούμε να διακρίνουμε τα ακόλουθα επίπεδα :

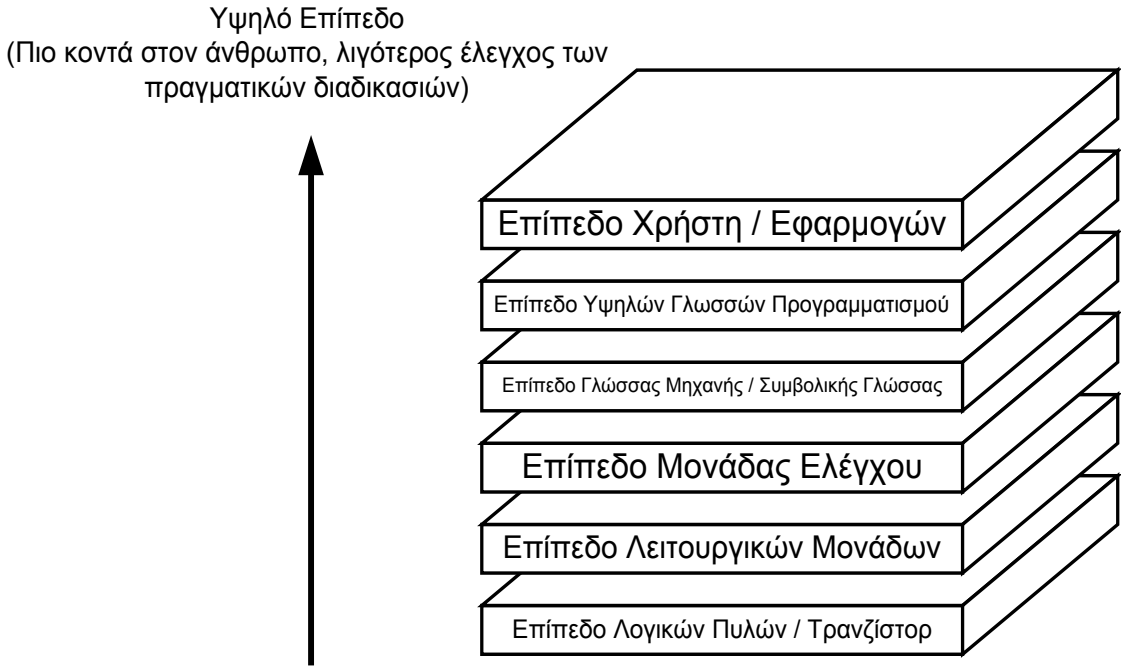
- Το επίπεδο λογικών πυλών, τρανζίστορ και αγωγών, που είναι και το κατώτατο επίπεδο ενός υπολογιστικού συστήματος. Στο επίπεδο αυτό η λειτουργικότητα του υπολογιστή εκφράζεται με δυναμικά, ρεύματα, χρόνους διάδοσης σημάτων και γενικά φαινόμενα πολύ χαμηλού επιπέδου. Τα τρανζίστορ χρησιμοποιούνται για την κατασκευή λογικών πυλών που είναι και η υπηρεσία που παρέχει το επίπεδο αυτό στο αμέσως υψηλότερο.
- Το επίπεδο λειτουργικών μονάδων. Στο επίπεδο αυτό ομάδες από λογικές πύλες οργανώνονται σε λειτουργικά κομμάτια όπως καταχωρητές, αριθμητικές μονάδες, μνήμες κλπ.
- Το επίπεδο μονάδας ελέγχου. Το επίπεδο αυτό ελέγχει τη μεταφορά πληροφορίας μεταξύ λειτουργικών μονάδων. Με τον τρόπο αυτό στην ουσία παρέχει ένα ρεπερτόριο διαφορετικών λειτουργιών, των οποίων διάφορες ακολουθίες αποτελούν *μικροπρογράμματα* για εντολές του υψηλότερου επιπέδου.
- Το επίπεδο γλώσσας μηχανής (Machine Code) / Συμβολικής Γλώσσας (Assembly Language). Είναι το υψηλότερο επίπεδο που είναι κατανοητό από τον υπολογιστή. Η εκτέλεση μιας εντολής αυτού του επιπέδου γίνεται στην ουσία με την εκτέλεση ενός μικροπρογράμματος από το αμέσως κατώτερο επίπεδο. Ο

προγραμματισμός σε αυτό το επίπεδο γίνεται είτε με συγγραφή τεραστίων σειρών από 0 και 1 είτε με συμβολική γλώσσα την οποία αναλαμβάνει ένα ειδικό πρόγραμμα (συμβολομεταφραστής – assembler- το οποίο είναι γραμμένο σε σειρές από 0 και 1) να τη μεταφράσει σε γλώσσα κατανοητή από το υλικό. Προσέξτε ότι αυτό είναι και το τελευταίο επίπεδο το οποίο δίνει στον προγραμματιστή πλήρη έλεγχο στο υλικό. Επίσης, είναι δυνατό τα κατώτερα επίπεδα δύο υπολογιστικών συστημάτων να έχουν διαφορετικές υλοποιήσεις, αλλά σε επίπεδο συμβολικής γλώσσας αυτά να προσφέρουν ακριβώς το ίδιο ρεπερτόριο. Τότε θα λέμε ότι αυτά τα υπολογιστικά συστήματα είναι ***συμβατά*** σε επίπεδο συμβολικής γλώσσας. Η συμβατότητα είναι πολύ σημαντική γιατί επιτρέπει προγράμματα που έχουν γραφτεί σε συμβολική γλώσσα ή σε γλώσσες πιο υψηλών επιπέδων να εκτελούνται σε άλλους υπολογιστές. Αυτός είναι και ο λόγος που οι υπολογιστές με CPU της εταιρείας Intel (πολύ γνωστοί ως προσωπικοί υπολογιστές – Personal Computers / PCs) έχουν βρει πολύ μεγάλη απήχηση τα τελευταία χρόνια, αν και σχεδιαστικά υπήρξαν πολύ καλύτερες λύσεις (όπως για παράδειγμα οι υπολογιστές της εταιρείας Apple που βασιζόταν σε CPUς της Motorola). Το επίπεδο αυτό θα αποτελέσει αντικείμενο του εργαστηρίου σας.

- Το επίπεδο των υψηλών γλωσσών προγραμματισμού. Στο επίπεδο αυτό ο προγραμματιστής συγγράφει το πρόγραμμά του χρησιμοποιώντας το ρεπερτόριο που του παρέχει μια γλώσσα η οποία είναι πολύ κοντά στον άνθρωπο και αρκετά μακριά από τη μηχανή. Οι πλέον διαδεδομένες υψηλές γλώσσες προγραμματισμού (high level languages) είναι η C, η Java, η Pascal και η Fortran. Ο προγραμματιστής αυτού του επιπέδου συγγράφει το πρόγραμμά του αγνοώντας και αδιαφορώντας για την υλοποίηση των συναρτήσεων που περιγράφει στη συγκεκριμένη μηχανή. Η ευθύνη μετατροπής του κώδικα υψηλής γλώσσας προγραμματισμού σε γλώσσα μηχανής επαφίεται στον μεταφραστή (compiler) ή τον διερμηνευτή (interpreter) που είναι προγράμματα ειδικά για το σκοπό αυτό. Οι μεν μεταφραστές (ο assembler είναι μιας ειδικής μορφής μεταφραστής) αφού αναγνώσουν όλο τον πηγαίο κώδικα (source code) που έγραψε ο προγραμματιστής τον μεταφράζουν σε κώδικα μηχανής και δημιουργούν ένα εκτελέσιμο αρχείο. Η διαδικασία μετάφρασης απαιτείται να πραγματοποιηθεί μόνο μια φορά. Οι διερμηνευτές, αφού μεταφράσουν μία γραμμή του πηγαίου κώδικα, την εκτελούν και η διαδικασία αυτή επαναλαμβάνεται μέχρις ότου ολοκληρωθεί η διερμήνευση όλου του πηγαίου κώδικα. Η διαδικασία αυτή επαναλαμβάνεται κάθε φορά που διερμηνεύεται ο κώδικας. Οι διερμηνευτές συχνά χρησιμοποιούνται στην αποσφαλμάτωση λαθών του πηγαίου κώδικα (debugging).

- Το επίπεδο χρήστη / επίπεδο εφαρμογών. Στο επίπεδο αυτό ο χρήστης επικοινωνεί με τον υπολογιστή εκτελώντας προγράμματα όπως επεξεργαστές κειμένου, φύλλα δεδομένων, παιχνίδια κλπ. Ο χρήστης στο επίπεδο αυτό δεν χρειάζεται να γνωρίζει τίποτα ούτε για την οργάνωση του υπολογιστή ούτε και για τη διαδικασία συγγραφής του προγράμματος που εκτελεί.

Η ιεραρχία των παραπάνω επιπέδων φαίνεται στο ακόλουθο σχήμα.



ΚΕΦΑΛΑΙΟ 2

Αναπαράσταση Δεδομένων

Πριν από οποιαδήποτε μορφή επεξεργασίας της πληροφορίας στον υπολογιστή, θα πρέπει πρώτα να έχουμε κάποιο τρόπο έκφρασής της σε μορφή κατανοητή από τον υπολογιστή. Η έκφραση της πληροφορίας (ισοδύναμα των δεδομένων) σε μορφή κατανοητή από τον υπολογιστή ονομάζεται *αναπαράσταση της πληροφορίας* ή εναλλακτικά *κωδικοποίησή* της ή *αναπαράσταση σε κάποιον κώδικα* (data encoding / data representation). Ενας κώδικας είναι ένα σύνολο κανόνων βάσει των οποίων μπορούμε να φτιάξουμε συστηματικές αναπαραστάσεις οι οποίες διέπονται από κάποιες ιδιότητες (αριθμητικές, ανίχνευσης / διόρθωσης σφαλμάτων κλπ). Πριν όμως μιλήσουμε για τους διάφορους κώδικες που είναι διαθέσιμοι, είναι καλό να δούμε τις μορφές της πληροφορίας που είναι κατανοητές από τον υπολογιστή.

2.1 Δυαδικό Σύστημα : Το σύστημα του υπολογιστή

Σήμερα (αρχές του 2001) τα τρέχοντα υπολογιστικά συστήματα λειτουργούν έχοντας σαν βάση δύο διακριτές καταστάσεις. Οι δύο αυτές διακριτές καταστάσεις εκφράζονται εσωτερικά στον υπολογιστή σαν δύο διακριτές στάθμες δυναμικού που ιδεατά είναι τα 0V και τα 5V. Βάσει αυτού θεωρούμε ότι κάθε στοιχειώδες κομμάτι πληροφορίας στον υπολογιστή μπορεί να είναι σε μία από δύο τιμές το 0 και το 1 με κάποια ένα προς ένα και επί συνάρτηση απεικόνισης μεταξύ αυτών των δύο τιμών και των σταθμών δυναμικού που αναφέρθηκαν πιο πάνω.

Για να το καταλάβουμε περισσότερο ας θεωρήσουμε ένα φως που μπορεί να είναι ανοικτό ή σβηστό. Αυτό που εμείς θεωρούμε σαν ανοικτό και σβηστό μπορεί να αναπαρασταθεί στον υπολογιστή το μεν ανοικτό για παράδειγμα με 0 και το δε

σβηστό με 1 ή εναλλακτικά το ανοικτό με 5V και το σβηστό με 0V. Προφανώς θα μπορούσαμε να αναπαραστήσουμε το ανοικτό με 0V και το σβηστό με 5V αλλά αυτό δεν αποτελεί παρά μια σύμβαση ή με άλλα λόγια έναν επιπλέον κανόνα στο σύνολο των κανόνων του κώδικα που χρησιμοποιούμε. Το στοιχειώδες κομμάτι πληροφορίας (που επίσης αποτελεί και το *στοιχειώδες κομμάτι μνήμης*) εσωτερικά του υπολογιστή που μπορεί να βρίσκεται σε μία από δύο διακριτές καταστάσεις το ονομάζουμε δυαδικό ψηφίο (BInary digiT – BIT) πληροφορίας. Μπορείτε να αντιπαραβάλλετε το δυαδικό ψηφίο που μπορεί να πάρει μόνο τις τιμές 0 και 1 με το δεκαδικό ψηφίο το οποίο μπορεί να πάρει τις τιμές 0, 1, 2, 3, 4, 5, 6, 7, 8 και 9.

Στα αμέσως επόμενα εξετάζουμε τα πολλαπλάσια της βασικής μονάδας πληροφορίας και φτιάχνουμε τον πρώτο μας κώδικα. Ας υποθέσουμε ότι θέλουμε να αναπαραστήσουμε στον υπολογιστή αυτοκόλλητα που μπορεί να έχουν ένα από τα εξής έξι χρώματα : άσπρο, μαύρο, κόκκινο, μπλε, κίτρινο και γαλάζιο. Χρησιμοποιώντας ένα δεκαδικό ψηφίο θα μπορούσαμε να κάνουμε την εξής αντιστοίχηση μεταξύ των καταστάσεων του δεκαδικού ψηφίου και των χρωμάτων (ισοδύναμα να φτιάξουμε τον εξής κώδικα) :

Κατάσταση Δεκαδικού Ψηφίου	**Χρώμα Αυτοκόλλητου**
0	Ασπρο
1	
2	Κίτρινο
3	
4	Γαλάζιο
5	
6	Κόκκινο
7	Μπλε
8	
9	Μαύρο

Στον υπολογιστή μας ωστόσο υπάρχουν μόνο δυαδικά ψηφία. Είναι λοιπόν προφανές ότι χρησιμοποιώντας ένα μόνο δυαδικό ψηφίο είναι αδύνατο να μπορέσουμε να κωδικοποιήσουμε τα έξι διαφορετικά χρώματα. Μπορείτε να αντιπαραβάλλετε αυτή την περίπτωση με την περίπτωση που έπρεπε να κωδικοποιήσουμε 20 χρώματα με 1 δεκαδικό ψηφίο. Συνεπώς στην περίπτωση αυτή θα χρειαστούμε περισσότερα του ενός δυαδικά ψηφία. Για να βρούμε το πόσα ακριβώς θα χρειαστούμε σκεφτόμαστε ως εξής :

1 δυαδικό ψηφίο μπορεί να κωδικοποιήσει 2 καταστάσεις.

2 δυαδικά ψηφία μπορούν να κωδικοποιήσουν 2^2 καταστάσεις.

3 δυαδικά ψηφία μπορούν να κωδικοποιήσουν 2^3 καταστάσεις.

...

ν δυαδικά ψηφία μπορούν να κωδικοποιήσουν $2^ν$ καταστάσεις.

Συνεπώς για το πρόβλημα των αυτοκόλλητων θα πρέπει να χρησιμοποιηθούν ν δυαδικά ψηφία ώστε $2^ν \geq 6$ ή ισοδύναμα $ν \geq \log_2 6$. Ο ελάχιστος αριθμός δυαδικών

ψηφίων δίδεται συνεπώς από τη σχέση $v = \lceil \log_2 \mu \rceil$ όπου μ οι διαφορετικές καταστάσεις που θέλω να κωδικοποιήσω. (Με $\lceil x \rceil$, συμβολίζουμε το μικρότερο ακέραιο y, για τον οποίο ισχύει $y \geq x$). Για το παράδειγμά μας v = 3 και έναν από τους δυνατούς κώδικες που μπορείτε να κατασκευάσετε βλέπετε στον επόμενο Πίνακα.

Κατάσταση Τριών Δυαδικών Ψηφίων $\beta_2\beta_1\beta_0$	**Χρώμα Αυτοκόλλητου**
000	Ασπρο
100	
010	Κίτρινο
001	Κόκκινο
110	Γαλάζιο
101	
011	Μπλε
111	Μαύρο

Είναι προφανές ότι πολλές φορές για πρακτικά προβλήματα θα χρειαστούμε ποσότητες πολλαπλάσιες του ενός δυαδικού ψηφίου. Υπάρχουν λοιπόν τα εξής πολλαπλάσια του ενός δυαδικού ψηφίου :

- Τετράδα (nibble) = ποσότητα τεσσάρων (2^2) δυαδικών ψηφίων
- Byte (οκτάδα) = ποσότητα οκτώ (2^3) δυαδικών ψηφίων
- Kilobyte - KByte (KB) = ποσότητα 1024 (2^{10}) Bytes
- Megabyte - MByte (MB) = ποσότητα 1024 (2^{10}) KBytes = 2^{20} Bytes
- Gigabyte - GByte (GB) = ποσότητα 1024 (2^{10}) MBytes = 2^{30} Bytes
- Terabyte - TByte (TB) = ποσότητα 1024 (2^{10}) GByte = 2^{40} Bytes

Για παράδειγμα 1GB είναι ποσότητα πληροφορίας (ισοδύναμα ποσότητα μνήμης) 2^{33} δυαδικών ψηφίων.

2.2 Αριθμητικά Συστήματα - Κώδικες με Βάρη - Αριθμητικοί Κώδικες

Στην υποενότητα αυτή εξετάζουμε κώδικες οι οποίοι θα μας επιτρέψουν να εκτελούμε αριθμητικές πράξεις στο υπολογιστικό μας σύστημα γρήγορα και απλά. Για την πλήρη ωστόσο κατανόηση των κωδίκων που χρησιμοποιούνται στα σημερινά συστήματα θα εξετάσουμε τους κώδικες που χρησιμοποιήθηκαν ιστορικά για το σκοπό αυτό, αλλά και θα εισάγουμε την απαραίτητη θεωρία πάνω στην οποία βασίζονται τα αριθμητικά συστήματα. Πολλές φορές θα χρησιμοποιούμε ως παράδειγμα το γνωστό μας δεκαδικό σύστημα.

Ενα σύστημα ονομάζεται ρ-δικό αν κάθε ψηφίο του μπορεί να πάρει τιμές στο διάστημα [0, ρ-1]. Ο αριθμός ρ ονομάζεται βάση ή ρίζα (radix) του συστήματος αυτού. Στο γνωστό μας δεκαδικό σύστημα ρ=10 και κάθε ψηφίο μπορεί να πάρει τιμή από 0 έως και 9. Αντίστοιχα στο οκταδικό σύστημα ρ=8 και κάθε ψηφίο μπορεί να πάρει τιμή από 0 ως 7. Για συστήματα με ρ>10

χρησιμοποιούνται τα γράμματα της αλφαβήτου για να υποδηλώσουν καταστάσεις των ψηφίων μεγαλύτερες του 10. Για παράδειγμα όταν ρ=16, κάθε ψηφίο μπορεί να πάρει τιμή από 0 έως και 15 και χρησιμοποιούνται τα γράμματα A, B, C, D, E και F για να υποδηλώσουν τις τιμές 10, 11, 12, 13, 14 και 15 αντίστοιχα, που μπορεί να πάρει κάθε ψηφίο.

Ενα αριθμητικό σύστημα είναι μια υποκατηγορία των κωδίκων με βάρη. Γενικά *κώδικας με βάρη είναι ένας κώδικας όπου ένα ψηφίο έχει μια βαρύτητα η οποία εξαρτάται από τη θέση την οποία καταλαμβάνει*. Ας υποθέσουμε την πληροφορία $\delta_{v-1}\delta_{v-2}...\delta_3\delta_2\delta_1\delta_0$, όπου τα δ_i, $0 \le i \le v-1$, είναι ψηφία του ρ-δικού συστήματος και η πληροφορία είναι κωδικοποιημένη με βάση τον κώδικα με βάρη $w_{v-1}w_{v-2} \dots w_2w_1w_0$. Αυτό σημαίνει ότι η θέση που έχει το ψηφίο δ_{v-1} έχει ένα βάρος w_{v-1}, η θέση που έχει το ψηφίο δ_{v-2} έχει ένα βάρος w_{v-2} κοκ. Για να καταλάβουμε ποια ποσότητα αντιπροσωπεύει η κωδικοποιημένη πληροφορία, αρκεί να εφαρμόσουμε τον *κανόνα του πολυωνύμου* :

$$\text{Ποσότητα} = \sum_{i=0}^{v-1} \left(w_i \delta_i \right)$$

Για παράδειγμα, έστω ότι χρησιμοποιούμε το επταδικό σύστημα και την αναπαράσταση 2 3 6 1 σε κώδικα με βάρη 6 4 3 1. Τότε η προηγούμενη είναι μια αναπαράσταση της ποσότητας 6*2+4*3+3*6+1*1 = 43. Προσέξτε ότι στον κώδικα αυτό, η ίδια ποσότητα έχει και άλλες πιθανές αναπαραστάσεις, όπως για παράδειγμα την 6 1 1 0 (αφού 6*6+4*1+3*1+1*0 = 43). Σε ένα αριθμητικό σύστημα το παραπάνω δεν είναι επιθυμητό. Αντιθέτως, είναι απόλυτα επιθυμητό κάθε ποσότητα να έχει μία και μόνο αναπαράσταση. Αυτό μπορεί να επιτευχθεί αν επιλέξουμε τα βάρη έτσι ώστε σε κάθε θέση i να αντιστοιχεί ένα βάρος ρ^i. Σε αυτή την περίπτωση ο κώδικας με βάρη γίνεται το ρ-δικό αριθμητικό σύστημα. Την αναπαράσταση μιας ποσότητας σε αυτό το σύστημα θα την συμβολίζουμε με $\delta_{v-1}\delta_{v-2}...\delta_3\delta_2\delta_1\delta_0{}_\rho$. Ακολουθούν μερικά παραδείγματα.

Συνηθίζουμε στο δεκαδικό σύστημα να γράφουμε την ποσότητα 827. Στην ουσία υπονοούμε την κωδικοποίηση 827_{10} δηλαδή μια ποσότητα ίση με $10^2*8 + 10^1*2+10^0*7$. Με ακριβώς την ίδια λογική η ποσότητα 1000110_2 είναι η κωδικοποίηση της ποσότητας $2^6*1+2^5*0+2^4*0+2^3*0+2^2*1+2^1*1+2^0*0 = 70_{10}$.

Ενα πρόβλημα που προκύπτει είναι πως δεδομένης μιας αναπαράστασης σε κάποιο σύστημα με βάση ρ μπορώ να βρω την αναπαράσταση της ίδιας ποσότητας σε ένα άλλο αριθμητικό σύστημα με βάση κ≠ρ. Προφανώς αν κ=10 αρκεί ο κανόνας του πολυωνύμου. Παρακάτω δίνουμε την διαδικασία μετατροπής όταν

ρ=10 και κ≠10. Προφανώς αν και το ρ≠10 θα πρέπει πρώτα να ακολουθήσουμε τον κανόνα του πολυωνύμου και μετά την παρακάτω περιγραφόμενη διαδικασία.

Εστω λ_{10} η ποσότητα που θέλουμε να μετατρέψουμε στο κ-δικό σύστημα. Αυτό σημαίνει ότι πρέπει να βρούμε μια αναπαράσταση της μορφής $\zeta_l\zeta_{l-1}...\zeta_1\zeta_{0\ \kappa} = \kappa^l\zeta_l+\kappa^{l-1}\zeta_{l-1}...+\kappa^1\zeta_1+\kappa^0\zeta_0 = \lambda_{10}$. Σκοπός προφανώς της διαδικασίας είναι ο καθορισμός των $\zeta_l,\zeta_{l-1},..., \zeta_1,\zeta_0$.

Βήμα 1. Διαιρούμε τα δύο μέλη της παραπάνω ισότητας με κ και έχουμε :

$$\kappa^{l-1}\zeta_l+\kappa^{l-2}\zeta_{l-1}...+\kappa^0\ \zeta_1+ (\zeta_0/\kappa) = (\lambda_{10} / \kappa)$$

Το παραπάνω σημαίνει ότι το ζ_0 μπορεί να καθοριστεί σαν το υπόλοιπο της διαίρεσης του λ με το κ. Συνεπώς έχουμε μετατρέψει ένα πρόβλημα καθορισμού ι αγνώστων σε ένα πρόβλημα καθορισμού ι-1 αγνώστων.

Βήμα 2. Θεώρησε σαν καινούριο στόχο τη μετατροπή του $\lambda = (\lambda_{10} / \kappa)$ και τον καθορισμό των $\zeta_l,\zeta_{l-1},..., \zeta_1$.

Βήμα 3. Αν λ = 0 η διαδικασία ολοκληρώθηκε, αλλιώς πήγαινε στο Βήμα 1.

(Τα παραπάνω απλά και κατανοητά βήματα που περιγράφουν μια διαδικασία που χρειάζεται να ακολουθηθεί ώστε να λυθεί κάποιο πρόβλημα, είναι στην ουσία η περιγραφή του πρώτου *Αλγόριθμου* τον οποίο διδάσκεστε).

Εστω ότι θέλουμε να μετατρέψουμε την ποσότητα 1145_8 στην αντίστοιχη αναπαράσταση στο δυαδικό. Πρώτα μετατρέπουμε αυτή τη ποσότητα στο δεκαδικό. Εχουμε λοιπόν $1145_8 = 8^3*1 + 8^2*1 + 8^1*4 + 8^0*5 = 613_{10}$. Ακολουθούμε τον παραπάνω αλγόριθμο για τη μετατροπή του 613_{10} σε δυαδικό και έχουμε :

Πράξη	**Πηλίκο**	**Υπόλοιπο**
613 / 2	306	1 (=β_0)
306 / 2	153	0 (=β_1)
153 / 2	76	1 (=β_2)
76 / 2	38	0 (=β_3)
38 / 2	19	0 (=β_4)
19 / 2	9	1 (=β_5)
9 / 2	4	1 (=β_6)
4 / 2	2	0 (=β_7)
2 / 2	1	0 (=β_8)
1 / 2	0	1 (=β_9)

Συνεπώς $1145_8 = 1001100101_2$.

Οταν η αρχική (ρ) και η στοχευόμενη (κ) βάση της αναπαράστασης συνδέονται με κάποια σχέση της μορφής $\rho=\kappa^a$ τότε η μετατροπή μπορεί να γίνει χωρίς το ενδιάμεσο βήμα της δεκαδικής αναπαράστασης, με αντικατάσταση ενός ψηφίου βάσης ρ με α ψηφία βάσης κ. Γενική απόδειξη για αυτό τον ισχυρισμό μπορεί κάποιος να βρει στα βιβλία αριθμητικής υπολογιστών. Ωστόσο εδώ θα προσπαθήσουμε να δώσουμε μια διαισθητική απόδειξη χρησιμοποιώντας ρ = 8 και

κ = 2. Επειδή αυτές οι δύο βάσεις συνδέονται με τη σχέση $\rho=\kappa^3$ μπορούμε να αντικαταστήσουμε ένα οκταδικό ψηφίο με 3 δυαδικά ψηφία. Εστω ο αριθμός 52_8 = $8^1*5 + 8^0*2 = 2^{3^1}*5+2^{3^0}*2=2^3*(2^2*1+2^1*0+2^0*1)+2^0*(2^2*0+2^1*1+2^0*0)=$ $2^5*1+2^4*0+2^3*1+2^2*0+2^1*1+2^0*0=$ 101010_2, όπου κάθε ψηφίο του οκταδικού συστήματος αντικαταστάθηκε από 3 δυαδικά ψηφία βάσει του κανόνα του πολυωνύμου και του παρακάτω πίνακα στον οποίο έχουμε συμπεριλάβει και τις αντικαταστάσεις για το δεκαεξαδικό σύστημα (προφανώς για το οκταδικό σύστημα πρέπει να χρησιμοποιούνται ΜΟΝΟ τα 3 λιγότερο σημαντικά ψηφία της δυαδικής αναπαράστασης) :

Ας προσπαθήσουμε να μετατρέψουμε τον αριθμό $A32D_{16}$ στον αντίστοιχο του στο οκταδικό σύστημα χρησιμοποιώντας αυτή τη φορά σαν ενδιάμεσο το δυαδικό σύστημα. Αφού κάθε ψηφίο του δεκαεξαδικού αντιστοιχεί σε τέσσερα δυαδικά ψηφία μπορούμε συμβουλευόμενοι και τον παρακάτω πίνακα να πάρουμε την εξής αναπαράσταση για το δυαδικό :

$$\underbrace{1010}_{A}\ \underbrace{0011}_{3}\ \underbrace{0010}_{2}\ \underbrace{1101}_{D}$$

Ψηφία του Οκταδικού	**Ψηφία του Δεκαεξαδικού**	**Ψηφία του Δυαδικού**
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
	8	1000
	9	1001
	A	1010
	B	1011
	C	1100
	D	1101
	E	1110
	F	1111

Για να βρούμε την ισοδύναμη αναπαράσταση στο οκταδικό, αφού κάθε οκταδικό ψηφίο αντικαθιστά τρία δυαδικά ψηφία θα πρέπει να χωρίσουμε τον δυαδικό σε τριάδες αρχίζοντας από τα δεξιά και συμπληρώνοντας με 0 την πλέον αριστερή τριάδα. Κατόπιν συμβουλευόμενοι τον παραπάνω πίνακα θα κάνουμε τις απαραίτητες αντικαταστάσεις :

$$\underbrace{001}_{1}\ \underbrace{010}_{2}\ \underbrace{001}_{1}\ \underbrace{100}_{4}\ \underbrace{101}_{5}\ \underbrace{101}_{5}$$

Συνεπώς $A32D_{16}$ = 1010001100101101_2 = 121455_8.

Χρησιμοποιώντας τον παραπάνω κώδικα καταφέραμε να αναπαραστήσουμε στον υπολογιστή μας όλες τις φυσικές ποσότητες. Ωστόσο στη καθημερινή ζωή μας αυτές είναι μόνο ένα μικρό υποσύνολο των αριθμητικών αναπαραστάσεων που χρειαζόμαστε. Στη συνέχεια θα αναπτύξουμε κώδικες με βάρη για την αναπαράσταση ακεραίων (προσημασμένων) αριθμών και κατόπιν κώδικες για την αναπαράσταση πραγματικών αριθμών.

2.3 Αναπαράσταση ακεραίων

Σε ότι αφορά τους προσημασμένους αριθμούς, υπάρχουν τέσσερις δημοφιλείς κώδικες στην κατηγορία κωδίκων με βάρη. Εξ αυτών σήμερα σε όλα τα υπολογιστικά συστήματα *χρησιμοποιείται κατά κόρο κώδικας συμπληρώματος ως προς 2.* Οι υπόλοιποι κώδικες χρησιμοποιούνται σε συγκεκριμένες εφαρμογές γιατί παρέχουν μοναδικές ιδιότητες ως προς ορισμένες αριθμητικές πράξεις. Ωστόσο πριν προχωρήσουμε παρακάτω είναι σκόπιμο να πούμε μερικά λόγια για τα προβλήματα της πεπερασμένης ακρίβειας αναπαράστασης.

Εχουμε συνηθίσει στην καθημερινή μας ζωή να χρησιμοποιούμε αναπαραστάσεις του δεκαδικού συστήματος χωρίς ποτέ να σκεφτόμαστε τον αριθμό των ψηφίων που μας είναι διαθέσιμα. Ωστόσο σε ένα υπολογιστικό σύστημα *θα πρέπει πάντοτε να λαμβάνουμε υπόψη μας τον αριθμό των δυαδικών ψηφίων που μας είναι διαθέσιμα, γιατί υπεισέρχεται η έννοια των πεπερασμένων ψηφίων αναπαράστασης.* Για να το καταλάβουμε αυτό καλύτερα ας υποθέσουμε ότι στο δεκαδικό σύστημα κάποιος μας περιόριζε τις νόμιμες αναπαραστάσεις μας στο ένα δεκαδικό ψηφίο. Τότε μπορούμε να διαπιστώσουμε ότι απλοί νόμοι, όπως αυτός της προσεταιριστικότητας της πρόσθεσης και της αφαίρεσης παύουν να ισχύουν, π.χ. :

$$7+(4-3) = 8$$

$$(7+4) -3 = ?$$

όπου η δεύτερη πράξη δεν μπορεί να εκτελεστεί, γιατί το ενδιάμεσο αποτέλεσμα οδηγεί σε ποσότητα η οποία δεν είναι νόμιμα αναπαραστήσιμη στον δεδομένο κώδικα. Αυτή η κατάσταση ονομάζεται *υπερχείλιση* και πολλές φορές μπορεί να μας οδηγήσει σε λανθασμένα αποτελέσματα. Συνεπώς θα πρέπει πάντοτε να λαμβάνουμε υπόψη μας το εύρος των δυνατών αναπαραστάσεων και να ελέγχουμε για τυχόν καταστάσεις υπερχείλισης.

2.3.1. Πρόσημο – Μέτρο (signed magnitude / sign and magnitude)

Ο κώδικας προσήμου-μέτρου θα μας φανεί ο πιο κατανοητός, μιας και παραλλαγή του χρησιμοποιούμε στο δεκαδικό σύστημα εισάγοντας την έννοια του προσήμου (+/-). Ετσι έχουμε συνηθίσει αναπαραστάσεις της μορφής -3245_{10} όπου πλέον το πρώτο ψηφίο της αναπαράστασης μας δείχνει αν ο αριθμός είναι θετικός ή αρνητικός και η υπόλοιπη αναπαράσταση μας δείχνει το μέτρο του αριθμού.

Με ακριβώς την ίδια λογική στον κώδικα προσήμου μέτρου χρησιμοποιούμε το πλέον αριστερότερο δυαδικό ψηφίο για να δείξουμε το πρόσημο του αριθμού (συνήθως η συμφωνία που γίνεται είναι ότι το 0 δείχνει θετικό αριθμό και το 1 αρνητικό) ενώ τα υπόλοιπα δυαδικά ψηφία δείχνουν το μέτρο του αριθμού σύμφωνα με τον κανόνα του πολυωνύμου. Ετσι για την αναπαράσταση β_{v-1} $\beta_{v-2}...\beta_1\beta_0$ ισχύει ότι αυτή αντιπροσωπεύει την ποσότητα :

$$(-1)^{\beta_{v-1}} \sum_{i=0}^{v-2} \left(2^i \beta_i\right)$$

Ο παραπάνω τύπος μας δίνει την διαδικασία αποκωδικοποίησης μιας αναπαράστασης σε πρόσημο-μέτρο. Προφανώς για την κωδικοποίηση απαιτείται η έκφραση του μέτρου βάσει του αλγορίθμου που δώσαμε προηγούμενα και η επισύναψη του δυαδικού ψηφίου προσήμου. Ας εξετάσουμε λίγο περισσότερο τις ικανότητες και τις αδυναμίες του συγκεκριμένου κώδικα. Δεδομένων κ δυαδικών ψηφίων αναπαράστασης, ο κώδικας αυτός μπορεί να μας προσφέρει αναπαραστάσεις για όλους τους ακεραίους που βρίσκονται στο διάστημα $[-(2^{κ-1}-1), 2^{κ-1}-1]$. Ο κώδικας αυτός σπαταλάει δύο αναπαραστάσεις του για το 0. Ειδικότερα υπάρχει το +0 με αναπαράσταση 000…00 και το –0 με αναπαράσταση 100…00. Σήμερα ο κώδικας αυτός χρησιμοποιείται σε ελάχιστες εφαρμογές.

2.3.2. Πόλωση - Πλεονασμός κατά K (excess / biased code)

Στον κώδικα αυτό οι αριθμοί μετατίθενται κατά μια ποσότητα ίση με K. Σκοπός της μετάθεσης είναι ο μικρότερος αριθμός (ο περισσότερο αρνητικός) να μπορεί να παρασταθεί με όλο 0 και οι υπόλοιποι αριθμοί κατά σειρά να αναπαρίστανται με αυξανόμενες σε ποσότητα αναπαραστάσεις. Για να το καταλάβουμε καλύτερα, ας υποθέσουμε ότι διαθέτουμε τρία δυαδικά ψηφία για τις αναπαραστάσεις μας. Αυτό μας οδηγεί σε οκτώ διαφορετικές αναπαραστάσεις. Συνεπώς μπορούμε να παραστήσουμε όλους τους ακεραίους του διαστήματος [-4, 3]. Σκοπός μας από εδώ και πέρα είναι να αναθέσουμε στο –4 την παράσταση 000 ή ισοδύναμα το μέτρο 0. Πρακτικά αυτό ισοδυναμεί με πρόσθεση του 4. Οι

υπόλοιπες παραστάσεις προκύπτουν με πρόσθεση του 4 και την εύρεση του μέτρου του αποτελέσματος. Ο παρακάτω πίνακας συνοψίζει τα παραπάνω.

Δεκαδική Παράσταση	**Πλεονασμός κατά 4**
+3	111 (=+3+4)
+2	110 (=+2+4)
+1	101 (=+1+4)
0	100 (= 0+4)
-1	011 (=-1+4)
-2	010 (=-2+4)
-3	001 (=-3+4)
-4	000 (=-4+4)

Δεδομένης ακρίβειας αναπαράστασης ρ δυαδικών ψηφίων συνήθως επιλέγεται $K = 2^{\rho-1}$. Οποιαδήποτε και αν είναι η επιλογή του K δεδομένης της αναπαράστασης $\rho_{v-1}\rho_{v-2}...\rho_1\rho_0$ η αποκωδικοποίηση μπορεί να γίνει βάσει του τύπου:

$$\sum_{i=0}^{v-1}\left(2^i \rho_i\right) - K$$

Εξετάζοντας πιο προσεκτικά τις ιδιότητες του κώδικα, μπορούμε να διαπιστώσουμε ότι αφενός δεν υπάρχει καμία σπατάλη αναπαραστάσεων, αφετέρου δε συγκρίσεις μεταξύ αριθμών μπορούν να γίνουν πολύ απλά, αδιαφορώντας εντελώς για το διαχωρισμό δυαδικών ψηφίων προσήμου.

2.3.3. Συμπλήρωμα ως προς 1

Για να αναπαραστήσουμε έναν αριθμό στον κώδικα συμπληρώματος ως προς 1 αρκεί να βρούμε το μέτρο του στο δυαδικό σύστημα και αν μεν ο αριθμός είναι θετικός τότε έχουμε την ζητούμενη αναπαράσταση, αν δε είναι αρνητικός η παράσταση προκύπτει συμπληρώνοντας (αντιστρέφοντας) κάθε δυαδικό ψηφίο του μέτρου του. Μπορείτε να θεωρείτε την εύρεση του συμπληρώματος σαν μια διαδικασία που απαιτεί την ύπαρξη ρ αντιστροφέων (inverters) που μαθαίνετε στο μάθημα του Λογικού Σχεδιασμού οι οποίοι ενεργοποιούνται αν και μόνο ο αριθμός που προσπαθούμε να παραστήσουμε είναι αρνητικός. Ας εξηγήσουμε τα παραπάνω με μερικά παραδείγματα θεωρώντας ακρίβεια παράστασης οκτώ δυαδικών ψηφίων:
Ο αριθμός $+12_{10}$ έχει μέτρο 00001100_2. Σε συμπλήρωμα ως προς 1 έχει παράσταση $00001100_{1's}$.
Ο αριθμός -12_{10} έχει μέτρο 00001100_2. Σε συμπλήρωμα ως προς 1 έχει παράσταση $11110011_{1's}$.

Ας δούμε τώρα μερικές από τις πιθανές διαδικασίες αποκωδικοποίησης μιας παράστασης σε συμπλήρωμα ως προς 1. Υποθέτουμε ακρίβεια κ δυαδικών ψηφίων και την παράσταση $\beta_{\kappa-1}\beta_{\kappa-2} \ldots \beta_1\beta_0$.

Διαδικασία 1. Εξετάζουμε το δυαδικό ψηφίο $\beta_{\kappa-1}$. Αν είναι 0 τότε ο αριθμός προκύπτει από τον κανόνα του πολυωνύμου πάνω στην δεδομένη παράσταση. Αν είναι 1 τότε αντιστρέφουμε όλα τα δυαδικά ψηφία της δεδομένης παράστασης. Το πρόσημο του αριθμού είναι αρνητικό ενώ το μέτρο του προκύπτει από τον κανόνα του πολυωνύμου της παράστασης $\overline{\beta_{\kappa-1}}\overline{\beta_{\kappa-2}}\ldots\overline{\beta_1}\overline{\beta_0}$, όπου το $\overline{\beta_i}$ συμβολίζει το αντίστροφο του δυαδικού ψηφίου β_i.

Διαδικασία 2. Εφαρμόζουμε τον κανόνα του πολυωνύμου θεωρώντας ως βάρος στη θέση κ-1 την ποσότητα $-(2^{\kappa-1}-1)$.

Ας δούμε πως μπορούμε να καταλάβουμε ποια ποσότητα αντιπροσωπεύει η παράσταση 11100010 σε συμπλήρωμα ως προς 1. Ακολουθώντας τη διαδικασία 1, διαπιστώνουμε ότι ο αριθμός είναι αρνητικός. Αντιστρέφοντας όλα τα δυαδικά ψηφία του παίρνουμε το μέτρο : 00011101_2 δηλαδή την ποσότητα 29_{10}. Συνεπώς πρόκειται για την παράσταση του -29_{10}. Εφαρμόζοντας τη διαδικασία 2 παίρνουμε: $-(2^7-1)*1 + 2^6*1 + 2^5*1 + 2^1*1 = -29_{10}$.

Αφού εξετάσαμε τις διαδικασίες κωδικοποίησης και αποκωδικοποίησης ας προσπαθήσουμε να καταλάβουμε λίγο περισσότερο τις ιδιότητες του κώδικα. Δεδομένης της ακρίβειας των κ δυαδικών ψηφίων, ο κώδικας μπορεί να παραστήσει όλους τους ακεραίους που βρίσκονται στο διάστημα $[-(2^{\kappa-1}-1), 2^{\kappa-1}-1]$, με την παράσταση 100…00 για τον μικρότερο αριθμό και την παράσταση 011..11 για τον μεγαλύτερο αριθμό. Ο κώδικας αυτός έχει το πρόβλημα της διπλής αναπαράστασης του 0 : 000…00 και 111…11.

Δεδομένης της αναπαράστασης μιας ποσότητας στον κώδικα συμπληρώματος ως προς 1 (ακριβώς το ίδιο ισχύει και για τον κώδικα συμπληρώματος ως προς 2 που ακολουθεί), μπορούμε πολύ εύκολα να βρούμε την αναπαράσταση αυτής της ποσότητας με λιγότερα ή περισσότερα ψηφία αναπαράστασης. Η διαδικασία που χρειάζεται να ακολουθήσουμε ονομάζεται *επέκταση προσήμου*. Εστω η παράσταση $\beta_{\kappa-1}\beta_{\kappa-2}\ldots\beta_1\beta_0$ 1's. Αφού αυτή αναπαριστά τη ποσότητα $[-(2^{\kappa-1}-1)]*\beta_{\kappa-1} + \sum_{i=0}^{\nu-2}\left(2^i\beta_i\right) = [-(2^{\kappa}-1)]*\beta_{\kappa-1} + 2^{\kappa-1}*\beta_{\kappa-1} + \sum_{i=0}^{\nu-2}\left(2^i\beta_i\right)$, ισχύει ότι $\beta_{\kappa-1}\beta_{\kappa-2}\ldots\beta_1\beta_0$ 1's $= \beta_{\kappa-1}\beta_{\kappa-1}\beta_{\kappa-2}\ldots\beta_1\beta_0$ 1's $= \beta_{\kappa-1}\,\beta_{\kappa-1}\,\beta_{\kappa-1}\beta_{\kappa-2}\ldots\beta_1\beta_0$ 1's $= \ldots$, δηλαδή μπορούμε να επαναλαμβάνουμε το αριστερότερο δυαδικό ψηφίο μιας αναπαράστασης στο συμπλήρωμα ως προς 1 όσες φορές θέλουμε στα αριστερά της. Επίσης μπορούμε να αποκόπτουμε ψηφία από τα αριστερά μιας παράστασης,

εφόσον το αριστερότερο ψηφίο παραμένει ίδιο με το αρχικό. Για παράδειγμα $1110101_{1's} = 11110101_{1's} = 110101_{1's} = 10101_{1's}$

Οπως θα μάθετε στο μάθημα του Λογικού Σχεδιασμού σας, ο κώδικας συμπληρώματος ως προς 1 ήταν ο πρώτος που χρησιμοποιήθηκε για τις συνήθεις αριθμητικές πράξεις. Ωστόσο στα σημερινά συστήματα που η ταχύτητα παίζει καθοριστικό ρόλο, ο κώδικας αυτός έχει εγκαταλειφθεί. Το γεγονός της ύπαρξης δύο παραστάσεων του 0 οδηγεί σε αυξημένη δυσκολία κατά τη σύγκριση αριθμών και συνεπώς σε καθυστέρηση των υπολογισμών.

Πριν προχωρήσουμε στον τελευταίο κώδικα με βάρη θα πρέπει να ρίξουμε μια πρώτη ματιά στην αριθμητική του δυαδικού συστήματος και συγκεκριμένα στην πρόσθεση. Η πρόσθεση στο δυαδικό σύστημα αρίθμησης γίνεται με τον ίδιο τρόπο που γίνεται στο δεκαδικό σύστημα. Δηλαδή ξεκινάμε από τα δεξιά προς τα αριστερά και σε κάθε θέση υπολογίζουμε ένα ψηφίο αθροίσματος και ένα ή περισσότερα ψηφία κρατουμένων. Ας δούμε πως γίνεται η πρόσθεση στο δεκαδικό:

$$\begin{array}{r} 29_{10} \\ 45_{10} + \\ \hline 74_{10} \end{array}$$

Ξεκινώντας από τα δεξιά προσθέτουμε πρώτα το 9 με το 5. Επειδή το άθροισμά τους υπερβαίνει τη βάση του συστήματος (10) έχουμε ένα ψηφίο αθροίσματος (4) και ένα κρατούμενο, δηλαδή μια μονάδα υψηλότερης βαθμίδας. Στην επόμενη βαθμίδα προσθέτουμε το 2 με το 4 και το εισερχόμενο κρατούμενο. Παρατηρούμε ότι δεν υπερβαίνουμε τη βάση και συνεπώς έχουμε μόνο ψηφίο αθροίσματος (7) και κανένα κρατούμενο προς την επόμενη βαθμίδα.

Αντίστοιχα μπορούμε να κάνουμε την πρόσθεση δύο αριθμών στο δυαδικό. Ο πίνακας που ακολουθεί υποδεικνύει τα παραγόμενα ψηφία αθροίσματος και κρατουμένου βάσει των διαφορετικών τιμών που μπορεί να πάρουν τα ψηφία των τελουμένων αλλά και το κρατούμενο από την προηγούμενη βαθμίδα. Προφανώς όταν ο αριθμός των αθροιζόμενων δυαδικών ψηφίων που είναι στο 1 ξεπερνάει τη ρίζα του συστήματος (2) τότε γεννιέται ένα κρατούμενο προς την επόμενη βαθμίδα.

Είσοδοι			Εξοδοι	
Δυαδικό Ψηφίο 1ου Προσθετέου	**Δυαδικό Ψηφίο 2ου Προσθετέου**	**Κρατούμενο από την προηγούμενη βαθμίδα**	**Υπολογιζόμενο δυαδικό ψηφίο αθροίσματος**	**Υπολογιζόμενο δυαδικό ψηφίο κρατουμένου**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Ενα παράδειγμα πρόσθεσης με βάση το δυαδικό σύστημα, παρουσιάζεται παρακάτω. Εχουμε αριθμήσει τις βαθμίδες πάνω από τα δύο τελούμενα για γρήγορη αναφορά. Ξεκινώντας από τη βαθμίδα 0 έχουμε δύο 0 στα ψηφία που πρόκειται να προστεθούν και 0 κρατούμενο από την προηγούμενη βαθμίδα (αφού δεν υπάρχει) και συνεπώς από την τρίτη γραμμή του παραπάνω πίνακα συμπεραίνουμε ότι θα πάρουμε ένα 0 στο άθροισμα και 0 κρατούμενο προς την επόμενη βαθμίδα. Συνεχίζοντας με τον ίδιο τρόπο, παρατηρούμε ότι στην βαθμίδα 3 δημιουργείται ένα κρατούμενο το οποίο διαδίδεται μέχρι και την τελευταία βαθμίδα. Αυτό αποτελεί το σοβαρότερο πρόβλημα του δυαδικού συστήματος με αρκετά περίπλοκες λύσεις.

$$\begin{array}{r} 76543210 \\ 01111100_2 \\ 01011010_2 + \\ \hline 11010110_2 \end{array}$$

Κάνοντας τις μετατροπές στο δεκαδικό σύστημα μπορείτε να διαπιστώσετε ότι το πρώτο τελούμενο είναι το 124_{10}, το δεύτερο το 90_{10} και το αποτέλεσμα το 214_{10}.

2.3.4. Συμπλήρωμα ως προς 2

Με τρόπο αντίστοιχο με το συμπλήρωμα ως προς 1 κατασκευάζεται το συμπλήρωμα ως προς 2 ενός ακεραίου. Συγκεκριμένα, για να αναπαραστήσουμε έναν ακέραιο στον κώδικα συμπληρώματος ως προς 2 αρκεί να βρούμε το μέτρο του στο δυαδικό σύστημα και αν μεν ο αριθμός είναι θετικός τότε έχουμε την ζητούμενη αναπαράσταση, αν δε είναι αρνητικός η παράσταση προκύπτει συμπληρώνοντας (αντιστρέφοντας) κάθε δυαδικό ψηφίο του μέτρου του και κατόπιν προσθέτοντας μία μονάδα. Μπορείτε να θεωρείτε την εύρεση του συμπληρώματος ως προς 2 σαν μια διαδικασία που απαιτεί την ύπαρξη ρ αντιστροφέων (inverters) και ενός αυξητή (incrementer) που μαθαίνετε στο μάθημα του Λογικού Σχεδιασμού οι οποίοι ενεργοποιούνται αν και μόνο ο αριθμός που προσπαθούμε να παραστήσουμε είναι αρνητικός. Μια άλλη διαδικασία που

μπορείτε να χρησιμοποιήσετε για την κωδικοποίηση ενός αρνητικού, είναι αφού βρείτε το μέτρο του, να αντιστρέψετε όλα τα δυαδικά ψηφία του μέτρου του εξαιρουμένων των ψηφίων που βρίσκονται δεξιά του δεξιότερου άσσου συμπεριλαμβανομένου και αυτού. Ας εξηγήσουμε τα παραπάνω με μερικά παραδείγματα θεωρώντας ακρίβεια παράστασης οκτώ δυαδικών ψηφίων :

- Ο αριθμός 43_{10} έχει παράσταση σε συμπλήρωμα ως προς 2 : 00101011
- Ο αριθμός –43 έχει μέτρο 00101011_2. Αντιστρέφουμε τα δυαδικά ψηφία του μέτρου του και παίρνουμε την παράσταση 11010100 (το συμπλήρωμα ως προς 1). Η πρόσθεση του 1 μας δίνει την παράσταση 11010101 που είναι το ζητούμενο συμπλήρωμα ως προς 2.
- Ο αριθμός –32 έχει μέτρο 00100000. Το συμπλήρωμα ως προς 2 είναι το 11100000, όπου αντιστρέψαμε τα δυαδικά ψηφία στα αριστερά του δεξιότερου 1.

Ας δούμε τώρα μερικές από τις πιθανές διαδικασίες αποκωδικοποίησης μιας παράστασης σε συμπλήρωμα ως προς 2. Υποθέτουμε ακρίβεια κ δυαδικών ψηφίων και την παράσταση $\beta_{\kappa-1}\beta_{\kappa-2} \ldots \beta_1\beta_0$.

Διαδικασία 1. Εξετάζουμε το δυαδικό ψηφίο $\beta_{\kappa-1}$. Αν είναι 0 τότε ο αριθμός προκύπτει από τον κανόνα του πολυωνύμου πάνω στην δεδομένη παράσταση. Αν είναι 1 τότε αντιστρέφουμε όλα τα δυαδικά ψηφία της δεδομένης παράστασης και προσθέτουμε μία μονάδα. Το πρόσημο του αριθμού είναι αρνητικό ενώ το μέτρο του προκύπτει από τον κανόνα του πολυωνύμου της παράστασης $(\overline{\beta_{\kappa-1}}\,\overline{\beta_{\kappa-2}} \ldots \overline{\beta_1}\,\overline{\beta_0} + 1)$, όπου το $\overline{\beta_i}$ συμβολίζει το αντίστροφο του δυαδικού ψηφίου β_i.

Διαδικασία 2. Εφαρμόζουμε τον κανόνα του πολυωνύμου θεωρώντας ως βάρος στη θέση κ-1 την ποσότητα $-2^{\kappa-1}$.

Ας δούμε πως μπορούμε να καταλάβουμε ποια ποσότητα αντιπροσωπεύει η παράσταση 11100010 σε συμπλήρωμα ως προς 2. Ακολουθώντας τη διαδικασία 1, διαπιστώνουμε ότι ο αριθμός είναι αρνητικός. Αντιστρέφοντας όλα τα δυαδικά ψηφία του και προσθέτοντας μία μονάδα παίρνουμε το μέτρο : 00011110_2 δηλαδή την ποσότητα 30_{10}. Συνεπώς πρόκειται για την παράσταση του -30_{10}. Εφαρμόζοντας τη διαδικασία 2 παίρνουμε : $-2^7*1 + 2^6*1 + 2^5*1 + 2^1*1 = -30_{10}$.

Ο κώδικας αυτός έχει μία μόνο παράσταση του 0, την 000…00. Επίσης επιτρέπει πρόσθεση και αφαίρεση ακεραίων να πραγματοποιούνται από μία κοινή μονάδα υλικού όπως θα μάθετε στο μάθημα του Λογικού Σχεδιασμού. Δεδομένων κ δυαδικών ψηφίων αναπαράστασης μπορούν να παρασταθούν όλοι οι ακέραιοι στο διάστημα $[-2^{\kappa-1}, 2^{\kappa-1}-1]$, όπου ο μικρότερος έχει την παράσταση 100…00 και ο

μεγαλύτερος την παράσταση 011…11. Υπενθυμίζεται τέλος, ότι και στον κώδικα αυτόν ισχύει η διαδικασία επέκτασης προσήμου.

2.4 Αναπαράσταση κλασματικών αριθμών

Στην καθημερινή μας ζωή οι ποσότητες που χρησιμοποιούμε συνήθως ανήκουν στο σύνολο των πραγματικών. Οι παραπάνω κώδικες όπως τους εξετάσαμε μέχρι ώρας δεν είναι ικανοί να προσφέρουν αναπαραστάσεις πραγματικών αριθμών. Στην πραγματικότητα όμως, ο κανόνας του πολυωνύμου μπορεί να χρησιμοποιηθεί για προσυμφωνημένες αναπαραστάσεις πραγματικών αριθμών. Ας το δούμε αυτό με ένα παράδειγμα. Ας υποθέσουμε μια μάλλον "μπερδεμένη" για τα δεδομένα του υπολογιστή αναπαράσταση (αφού ακόμη δεν έχουμε ξεκαθαρίσει το πως αναπαρίσταται η υποδιαστολή) την $101,0110_2$, όπου το κόμμα χρησιμοποιείται για να υποδηλώσει ένα δεκαδικό μέρος. Με την ίδια λογική που αναπτύχθηκε για τον κανόνα του πολυωνύμου, μπορούμε και εδώ να θεωρήσουμε ότι η παραπάνω αναπαράσταση υποδηλώνει την ποσότητα : $2^2*1+2^0*1+2^{-2}*1+2^{-3}*1 = 5,375_{10}$ και γενικά μπορούμε να θεωρούμε ότι η παράσταση του ρ-δικού συστήματος : $\beta_{\kappa-1}\beta_{\kappa-2}...\beta_1\beta_0,\beta_{-1}\beta_{-2}...\beta_{-(\lambda-1)}\beta_{-\lambda\ \rho}$ υποδηλώνει την ποσότητα : $\sum_{i=-\lambda}^{\kappa-1}\rho^i\beta_i$.

Επίσης μπορούμε με τρόπο ανάλογο με τον προηγούμενο να διατυπώσουμε και έναν αλγόριθμο για τη μετατροπή του δεκαδικού μέρους ενός πραγματικού στο κ-δικό αν παρατηρήσουμε ότι (προφανώς για έναν πραγματικό που έχει και ακέραιο και δεκαδικό μέρος οι δύο διαδικασίες μετατροπής απαιτείται να πραγματοποιηθούν ξεχωριστά) : αν συμβολίσουμε με $0,\lambda_{10}$ την ποσότητα που θέλουμε να μετατρέψουμε στο κ-δικό σύστημα, τότε πρέπει να βρούμε μια αναπαράσταση της μορφής $0,\zeta_{-1}\zeta_{-2}...\zeta_{-\varphi+1}\zeta_{-\varphi\ \kappa} = \kappa^{-1}\zeta_{-1} + \kappa^{-2}\zeta_{-2} \ldots + \kappa^{-\varphi+1}\zeta_{-\varphi+1} + \kappa^{-\varphi}\zeta_{-\varphi} = 0,\lambda_{10}$. Σκοπός προφανώς της διαδικασίας είναι ο καθορισμός των ζ_{-1}, ζ_{-2}, …, $\zeta_{-\varphi+1}$, $\zeta_{-\varphi}$.

Βήμα 1. Πολλαπλασιάζουμε τα δύο μέλη της παραπάνω ισότητας με κ και έχουμε :

$(0,\lambda_{10} * \kappa) = \kappa^0\zeta_{-1}+\kappa^{-1}\zeta_{-2}...+ \kappa^{-\varphi+2}\zeta_{-\varphi+1} + \kappa^{-\varphi+1}\zeta_{-\varphi} = \zeta_{-1},\zeta_{-2}...\zeta_{-\varphi+1}\zeta_{-\varphi\ \kappa}$

Το παραπάνω σημαίνει ότι το ζ_{-1} μπορεί να καθοριστεί σαν το ακέραιο μέρος του πολλαπλασιασμού του $0,\lambda_{10}$ με το κ. Συνεπώς έχουμε μετατρέψει ένα πρόβλημα καθορισμού φ αγνώστων σε ένα πρόβλημα καθορισμού φ-1 αγνώστων.

Βήμα 2. Θεώρησε σαν καινούριο στόχο τη μετατροπή του $\lambda' = (0,\lambda_{10} * \kappa) - \kappa^0\zeta_{-1}$ και τον καθορισμό των $\zeta_{-2},\zeta_{-3},..., \zeta_{-\varphi}$.

Βήμα 3. Αν λ' = 0 η διαδικασία ολοκληρώθηκε, αλλιώς πήγαινε στο Βήμα 1.
Ας δώσουμε ένα παράδειγμα για να γίνουν ακόμη πιο κατανοητά τα παραπάνω. Ας θεωρήσουμε σαν στόχο μας τη μετατροπή του $0,15_8$ στον αντίστοιχο του δυαδικού συστήματος. Σύμφωνα με τον κανόνα του πολυωνύμου μπορούμε να δούμε ότι $0,15_8 = 8^{-1}*1 + 8^{-2}*5 = 0,203125_{10}$. Εφαρμόζουμε τον παραπάνω αλγόριθμο των διαδοχικών πολλαπλασιασμών και παίρνουμε :

Πράξη	Ακέραιο Μέρος	Κλασματικό Μέρος
0,203125 * 2	0 (=β_{-1})	$0,40625_{10}$
0,40625 * 2	0 (=β_{-2})	$0,8125_{10}$
0,8125 * 2	1 (=β_{-3})	$0,625_{10}$
0,625 * 2	1 (=β_{-4})	$0,25_{10}$
0,25 * 2	0 (=β_{-5})	$0,5_{10}$
0,5 * 2	1 (=β_{-6})	0_{10}

Συνεπώς $0,15_8 = 0,203125_{10} = 0,001101_2$. Προφανώς μπορούσαμε να αποφύγουμε το ενδιάμεσο βήμα της παράστασης στο δεκαδικό σύστημα σε αυτήν την περίπτωση μιας και οι ρίζες των δυαδικό και οκταδικό συστημάτων αρίθμησης σχετίζονται βάσει της $8=2^3$ και συνεπώς αρκούσε η εφαρμογή του πίνακα αντικατάστασης ψηφίων. Μόνο που σε αυτή τη περίπτωση θα πρέπει να δουλεύουμε από τα αριστερά προς τα δεξιά συμπληρώνοντας με 0 όπου χρειάζονται. Για παράδειγμα η μετατροπή του $0,010111_2$ σε δεκαεξαδικό μπορεί να γίνει ως εξής : $0,\underbrace{0101}_{5}\underbrace{1100}_{C}{}_{2}$. Δηλαδή $0,010111_2 = 0,5C_{16}$.

Τα παραπάνω αν και μας δίνουν μια πολύ καλή θεωρητική εικόνα είναι πολύ ελλιπή στην εφαρμογή. Υπάρχουν διάφοροι λόγοι γι' αυτό. Ας δούμε μερικούς. Ας προσπαθήσουμε να μετατρέψουμε τον αριθμό $0,2_{10}$ στον αντίστοιχο του δυαδικού συστήματος. Τότε θα εφαρμόζαμε την εξής σειρά πολλαπλασιασμών:

Πράξη	Ακέραιο Μέρος	Κλασματικό Μέρος
0,2 * 2	0 (=β_{-1})	0,4
0,4 * 2	0 (=β_{-2})	0,8
0,8 * 2	1 (=β_{-3})	0,6
0,6 * 2	1 (=β_{-4})	0,2

Βλέπουμε δηλαδή ότι μετά από τους πρώτους τέσσερις πολλαπλασιασμούς φτάνουμε πάλι στο ίδιο πρόβλημα, την μετατροπή του 0,2. Συνεπώς συμπεραίνουμε ότι το $0,2_{10}$ δεν μπορεί να παρασταθεί με πεπερασμένο αριθμό δυαδικών ψηφίων. Στην ουσία αυτό συμβαίνει για την συντριπτική πλειοψηφία των αριθμών που έχουν πεπερασμένη παράσταση στο δεκαδικό. Προσέξτε επίσης ότι αριθμοί όπως το $0,333333\ldots_{10}$ μπορεί να έχουν πεπερασμένη αναπαράσταση σε κάποιο άλλο αριθμητικό σύστημα, π.χ. $0,1_3$.

2.4.1 Σταθερή υποδιαστολή

Ενα άλλο πρόβλημα που ακόμη δεν προσεγγίσαμε ικανοποιητικά είναι η αποθήκευση της θέσης της υποδιαστολής. Για να καταλάβουμε το πρόβλημα που δημιουργείται ας θεωρήσουμε έναν υπολογιστή που διαθέτει 16 δυαδικά ψηφία για την αναπαράσταση πραγματικών αριθμών. Ας θεωρήσουμε ότι το πλέον αριστερό ψηφίο διατίθεται για το πρόσημο και συνεπώς μένουν 15 δυαδικά ψηφία. Μια προφανής λύση για τη θέση της υποδιαστολής είναι να θεωρήσουμε ότι αυτή είναι πάντα στην ίδια θέση. Ας υποθέσουμε ότι συμφωνούμε να βρίσκεται δεξιά του $11^{ου}$ από τα εναπομείναντα δυαδικά ψηφία. Σε αυτή τη συμφωνία, ο αριθμός $0,2_{10}$ αναπαριστάται σαν 0000000000000011 δηλαδή σαν $0,1875_{10}$. Η απώλεια ακρίβειας για αυτή την αναπαράσταση είναι μεγαλύτερη του 6%. Αν θεωρούσαμε όμως την υποδιαστολή πάντοτε δεξιά του $7^{ου}$ δυαδικού ψηφίου (από τα 15 εναπομείναντα), η παράσταση που θα είχαμε θα ήταν 0000000000110011 δηλαδή $0,19921875_{10}$ με απώλεια ακρίβειας λιγότερη του 1%. Ωστόσο στην πρώτη περίπτωση μπορούμε να παραστήσουμε αριθμούς της τάξης του 2^{11} ενώ στη δεύτερη μόλις του 2^7. Συνεπώς αν και η δεύτερη θεώρηση μας δίνει καλύτερη ακρίβεια μας παρέχει πολύ μικρότερο εύρος αναπαράστασης.

Σύμφωνα με τα παραπάνω, η προσυμφωνημένη θέση της υποδιαστολής (fixed-point representations) μας οδηγεί μοιραία σε ένα παζάρι μεταξύ ακρίβειας και εύρους αναπαράστασης. Αν θέλαμε να τα καλύψουμε και τα δύο επαρκώς, τότε ο αριθμός των δυαδικών ψηφίων που θα απαιτείτο θα ήταν τεράστιος. Και αυτό γιατί στα επιστημονικά προβλήματα απαιτούνται πολλές φορές πολύ μεγάλοι (π.χ. η σταθερά του Avogadro $6,023 * 10^{23}$) ή πολύ μικροί (π.χ. φορτία ηλεκτρονίων, τιμές πυκνωτών) αριθμοί. Συνεπώς για το δυαδικό σύστημα θα θέλαμε να είχαμε περί τα 40 ψηφία εκατέρωθεν της υποδιαστολής. Ωστόσο κάτι τέτοιο οδηγεί σε τεράστιες ανάγκες μνήμης και μοιραία σε συστήματα πολύ υψηλού κόστους. Αρα στην πραγματικότητα δεν είναι συμφέρον να υπάρχουν αναπαραστάσεις με σταθερή θέση υποδιαστολής. Αυτό που θα θέλαμε είναι η υποδιαστολή να είναι κινούμενη ώστε να μπορούμε να καλύψουμε με μικρό αριθμό δυαδικών ψηφίων τόσο μεγάλο εύρος αριθμών όσο και να επιτύχουμε ικανοποιητική ακρίβεια ανάλογα με τις απαιτήσεις της εφαρμογής μας. Γι' αυτό και στα σημερινά συστήματα χρησιμοποιούνται αποκλειστικά και μόνο αναπαραστάσεις κινητής (floating – point) υποδιαστολής.

2.4.2 Κινητή υποδιαστολή

Η ιδέα της κινητής υποδιαστολής ξεκίνησε από την επιστημονική γραφή (scientific notation). Πολλές φορές όταν θέλουμε να γράψουμε έναν αριθμό με πολλά μηδενικά στα δεξιά του ή με πολλά μηδενικά αμέσως μετά την υποδιαστολή χρησιμοποιούμε τη γραφή $\Sigma*B^E$, όπου Σ ο συντελεστής (σημαντικό μέρος – significant – mantissa) του αριθμού, Β η βάση του αριθμητικού συστήματος που εργαζόμαστε και Ε ο εκθέτης (exponent) της παράστασης. Για παράδειγμα στον αριθμό του Avogadro $6,023*10^{23}$ ισχύουν Σ = 6,023, Β = 10 και Ε = 23. Προφανώς τα Σ και Ε είναι προσημασμένοι αριθμοί και πιο συγκεκριμένα ο Ε είναι ακέραιος. Αν εξ αρχής η βάση Β του αριθμητικού συστήματος προσυμφωνηθεί, τότε μπορεί να παραλείπεται κατά την παράσταση κάποιας ποσότητας.

Διαπιστώστε ότι ένας αριθμός μπορεί να έχει περισσότερες της μιας παραστάσεις στο ίδιο αριθμητικό σύστημα. Για παράδειγμα οι παραστάσεις $3,76*10^1$, $0,376*10^2$ και $376*10^{-2}$ παριστάνουν τον ίδιο αριθμό. Εάν θεωρήσουμε ότι ο συντελεστής είναι σε παράσταση πρόσημο-μέτρο τότε εξ αυτών των παραστάσεων θα καλούμε κανονικοποιημένη (normalized) παράσταση μιας ποσότητας αυτήν στην οποία η υποδιαστολή βρίσκεται στα αριστερά του αριστερότερου μη μηδενικού ψηφίου του συντελεστή. Στο παραπάνω παράδειγμα η δεύτερη είναι η κανονικοποιημένη παράσταση της ποσότητας.

Ερχόμενοι στον υπολογιστή, μπορούμε αυθαίρετα να ορίσουμε το πλήθος των δυαδικών ψηφίων που χρησιμοποιούνται για την παράσταση κινητής υποδιαστολής, πως αυτά κατανέμονται σε συντελεστή και εκθέτη, ποια είναι η θεωρούμενη βάση και εάν γίνεται κανονικοποίηση ή όχι. Ας δούμε ένα παράδειγμα στο οποίο επιλέγουμε :

- Ο συντελεστής είναι σε πρόσημο – μέτρο. Για το μέτρο χρησιμοποιούμε 3 δεκαεξαδικά ψηφία.
- Ο εκθέτης μας θα είναι τριών δυαδικών ψηφίων με αναπαράσταση πόλωσης κατά 4.
- Η βάση μας θα είναι το 16.
- Οι αποθηκευμένοι αριθμοί είναι σε κανονικοποιημένη μορφή.
- Η μορφή της αποθήκευσης είναι :

-	---	.	----	----	----
Πρόσημο του συντελεστή	Εκθέτης 3 δυαδικών ψηφίων	Υπονοούμενη υποδιαστολή	Τρία δεκαεξαδικά ψηφία		

Ο λόγος για αυτές τις επιλογές που μπορεί να σας φαίνονται λίγο παράξενες είναι ότι με τη μορφή αυτή δύο αριθμοί μπορούν άμεσα να συγκριθούν ως προς τη

ισότητα ή να βρεθεί ο μεγαλύτερος από αυτούς. Ας προσπαθήσουμε να παραστήσουμε τον αριθμό 143_{10} σε αυτό το σύστημα. Με τη μέθοδο των διαδοχικών διαιρέσεων μπορούμε να βρούμε ότι $143_{10}=8F_{16}$. Πριν την παράσταση του αριθμού θα πρέπει πρώτα να τον κανονικοποιήσουμε. Εχουμε λοιπόν ότι $8F_{16} = 8F*16^0 = 0,8F*16^2$. Τώρα μπορούμε πλέον να συμπληρώσουμε τα παραπάνω πεδία. Εχουμε ότι το πρόσημο του συντελεστή είναι 0 αφού ο αριθμός είναι θετικός. Ο εκθέτης (2) σε excess 4 παράσταση έχει τιμή 6 δηλαδή 110_2. Τα δεκαεξαδικά ψηφία δεξιά της υποδιαστολής είναι 8, F και συμπληρώνουμε 0 και συνεπώς η συνολική παράσταση είναι : 0110100011110000.

Η παραπάνω παράσταση βασίστηκε σε αυθαίρετες επιλογές. Αν όμως ο καθένας κάνει τις όποιες επιλογές του αυθαίρετα, αυτόματα δυσχεραίνεται η μεταφορά δεδομένων μεταξύ διαφορετικών υπολογιστικών συστημάτων αλλά και η εκτέλεση προγραμμάτων που έχουν συγγράψει διάφοροι υιοθετώντας ένα συγκεκριμένο μοντέλο παράστασης. Συνήθως όταν υπάρχουν τέτοιες δυνατότητες για επιλογές και αφού στο πραγματικό πεδίο διαπιστωθούν οι ικανότητες / αδυναμίες κάθε επιλογής, κάποια επιτροπή προτυποποίησης καλείται για να ορίσει κάποιο στάνταρτ, το οποίο αμέσως μετά υιοθετείται από τους διάφορους κατασκευαστές υπολογιστών και έτσι λύνονται αυτόματα τα παραπάνω προβλήματα. Στο πρόβλημα της αναπαράστασης κινητής υποδιαστολής λύση έδωσε το στάνταρτ 754 μιας επιτροπής της IEEE (Institute of Electrical and Electronic Engineers). Το στάνταρτ αυτό έχει υιοθετηθεί από όλα τα υπολογιστικά συστήματα της τελευταίας εικοσαετίας.

Σύμφωνα με αυτό το στάνταρτ για την παράσταση αριθμών κινητής υποδιαστολής διατίθενται είτε 32 (απλή ακρίβεια - single precision) είτε 64 (διπλή ακρίβεια - double precision) δυαδικά ψηφία, ανάλογα με την ακρίβεια και το εύρος που επιβάλει η εφαρμογή μας. Σε αυτό το σύστημα βάση επιλέγεται το 2, ο συντελεστής είναι σε παράσταση πρόσημο – μέτρο και ο εκθέτης σε κώδικα πόλωσης. Το εύρος κάθε πεδίου και την πόλωση για κάθε υποστηριζόμενη ακρίβεια παράστασης συνοψίζονται στον παρακάτω πίνακα :

	Συνολικά Δυαδικά Ψηφία	Δυαδικά ψηφία Εκθέτη	Κώδικας Εκθέτη	Δυαδικά Ψηφία Συντελεστή
Single Precision	32	8	Πόλωση κατά 127	1 + 23 (+1)
Double Precision	64	11	Πόλωση κατά 1023	1 + 52 (+1)

Τα παραπάνω πεδία ακολουθούν την εξής σειρά στην παράσταση : Πρόσημο συντελεστή, εκθέτης και συντελεστής. Προσέξτε ότι οι σχεδιαστές του στάνταρτ έχουν δώσει μια διαφορετική σκοπιά στην έννοια της κανονικοποίησης. Οι αριθμοί σε αυτό το στάνταρτ πριν την αποθήκευσή τους κανονικοποιούνται έτσι ώστε η υποδιαστολή να βρίσκεται στα δεξιά του πρώτου αριστερότερου δυαδικού ψηφίου. Δηλαδή η μορφή των αριθμών που υποστηρίζει το στάνταρτ είναι $1,XXX\ldots_2$ και όχι $0,1XX\ldots_2$ όπως η συνήθης μορφή κανονικοποίησης. Αυτό συν το γεγονός ότι χρησιμοποιείται σαν βάση το 2 οδηγεί στα εξής :

- Αφού πάντα αριστερά της υποδιαστολής υπάρχει ένα μη μηδενικό ψηφίο και στο δυαδικό σύστημα μη μηδενικό ψηφίο είναι μόνο το 1, συμπεραίνουμε ότι δε χρειάζεται να αποθηκεύεται αυτό το ψηφίο αλλά μπορεί να υπονοείται (hidden bit), αυξάνοντας τη προσφερόμενη ακρίβεια (αυτό υποδεικνύεται με το (+1) στον παραπάνω πίνακα).
- Για την παράσταση του 0 οι σχεδιαστές υποχρεώθηκαν να κρατήσουν το συνδυασμό όλο 0 του εκθέτη σαν ειδική περίπτωση. Για λόγους αναπαράστασης του +∞ και του -∞ ο συνδυασμός όλο 1 στον εκθέτη υποδηλώνει ειδικές περιπτώσεις.

Ο παρακάτω πίνακας υποδεικνύει μερικές εκ των ειδικών περιπτώσεων. Διαιρέσεις του τύπου 0/0 ή ο υπολογισμός της τετραγωνικής ρίζας του –1 οδηγούν στην τελευταία παράσταση που δηλώνει ότι το αποτέλεσμα δεν είναι πραγματικός αριθμός.

Συντελεστής		Εκθέτης	Σημασιολογία
0	000…00	000…00	+0
1	000…00	000…00	-0
0	000…00	111…11	+∞
1	000…00	111…11	-∞
0/1	≠ 000…00	111…11	NaN (Not a Number)

Σε κάθε άλλη περίπτωση η ποσότητα που αναπαρίσταται σε IEEE 754 δίνεται από τη σχέση : $(-1)^{κ} * (1+Σ) * 2^{(Ε\text{-πόλωση})}$ όπου κ το πρόσημο του συντελεστή. Ας δούμε μερικά παραδείγματα ώστε όλα τα παραπάνω να γίνουν πιο κατανοητά. Εστω ότι θέλουμε να παραστήσουμε τον $18,75_{10}$ σε single precision IEEE. Μετατρέπουμε το 18_{10} με τη μέθοδο των διαδοχικών διαιρέσεων στο δυαδικό και έχουμε $18_{10} = 10010_2$. Με τη μέθοδο των διαδοχικών πολλαπλασιασμών βρίσκουμε ότι $0,75_{10} = 0,11_2$. Αρα $18,75_{10} = 10010,11_2 = 10010,11 * 2^0 = 1,001011 * 2^4$. Μπορούμε λοιπόν τώρα να θέσουμε :

- Πρόσημο θετικό άρα το αριστερότερο δυαδικό ψηφίο στο 0.
- Εκθέτης = 4, άρα σε excess 127 θα έχει την παράσταση 10000011.
- Συντελεστής = 1,001011, άρα σε 23 δυαδικά ψηφία και αποκόπτοντας το πρώτο (hidden) θα έχει παράσταση 00101100000000000000000.

Η συνολική παράσταση συνεπώς του αριθμού είναι :

0 10000011 00101100000000000000000

Ας θεωρήσουμε ότι μας δίδεται η παράσταση

0 10000100 10100000000000000000000

Τότε μπορούμε να συμπεράνουμε ότι πρόκειται για θετικό αριθμό με εκθέτη 132-127=5 και συντελεστή (1).101. Συνεπώς πρόκειται για τον αριθμό $1,101*2^5$ = $110100 * 2^0 = 52_{10}$.

2.5 Κώδικες χωρίς Βάρη – Αναπαράσταση άλλων Δεδομένων

Πέρα από την αναπαράσταση αριθμητικών δεδομένων, στον υπολογιστή πρέπει να παραστήσουμε και χαρακτήρες, σημεία στίξης κλπ. Πάνω σε αυτά τα δεδομένα δεν πρόκειται να εκτελέσουμε αριθμητικές πράξεις, συνεπώς ένας κώδικας με βάρη δεν είναι απαραίτητος. (Συνεχίζει προφανώς να ισχύει ότι και αυτές οι πληροφορίες μπορεί σε ένα υπολογιστικό σύστημα να αναπαρασταθούν μόνο από σειρές από 0 και 1. Ο ελάχιστος αριθμός από δυαδικά ψηφία που θα χρειαστώ για την αναπαράστασή τους συνεπώς, είναι ανάλογος του μέγιστου αριθμού διαφορετικών συμβόλων που χρειάζομαι).

2.5.1. Αναπαράσταση αλφαριθμητικών χαρακτήρων

Για την αναπαράσταση αριθμών και χαρακτήρων, ο πλέον διαδεδομένος κώδικας είναι ο κώδικας ASCII (American Standard Code for Information Interchange), που φαίνεται στον επόμενο πίνακα.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Ο κώδικας αυτός χρησιμοποιεί 7 δυαδικά ψηφία για τις αναπαραστάσεις του και συνεπώς μπορεί να προσφέρει 128 διαφορετικές αναπαραστάσεις. Οι αναπαραστάσεις που βρίσκονται στο διάστημα 00_H – $1F_H$ και $7F_H$ είναι ειδικοί χαρακτήρες ελέγχου για την μετάδοση δεδομένων ή για την εκτύπωσή τους. Στους υπόλοιπους χαρακτήρες είναι ενδιαφέρον να παρατηρήσετε ότι ο κωδικός τους προκύπτει από το δυαδικό ισοδύναμο της στήλης και της γραμμής ενός κανονικού πληκτρολογίου. Προσέξτε επίσης ότι οι χαρακτήρες "a" και "A" είναι διαφορετικοί. Με τη διάταξη αυτή μπορούμε με απλούς υπολογισμούς να πάρουμε ορισμένες αντιστοιχίες. Για παράδειγμα από έναν χαρακτήρα-αριθμό, μπορούμε να πάρουμε την τιμή του αφαιρώντας το 48 (30_H). Για να μετατρέψουμε ένα κεφαλαίο χαρακτήρα στον αντίστοιχο μικρό του μπορούμε στην αναπαράσταση του πρώτου να προσθέσουμε το 32 (20_H).

Το μεγάλο πρόβλημα του κώδικα ASCII είναι το πολύ περιορισμένο σύνολο διαφορετικών χαρακτήρων το οποίο προσφέρει. Οι 128 αναπαραστάσεις είναι αδύνατο να επαρκέσουν για κάποιον που θέλει να μπορεί να χειρίζεται δύο ή περισσότερα αλφάβητα.

Μια πρώτη λύση στο παραπάνω πρόβλημα αποτέλεσε ο κώδικας EBCDIC (Extended Binary Coded Decimal Interchange Code), ο οποίος είναι ένας κώδικας 8 δυαδικών ψηφίων και συνεπώς μπορεί να παρέχει έως και 256 διαφορετικές αναπαραστάσεις. Μεταξύ αυτού του κώδικα και του ASCII υπάρχει μια αντιστοιχία, ώστε κάθε κείμενο γραμμένο στον ένα κώδικα να μπορεί να διαβαστεί από υπολογιστικό σύστημα που χρησιμοποιεί τον άλλον και αντίστροφα. Επιπλέον στον κώδικα αυτόν υπάρχουν οι ζητούμενες κενές θέσεις ώστε να μπορούν εκεί να εμφωλευτούν οι χαρακτήρες και ενός δεύτερου αλφαβήτου.

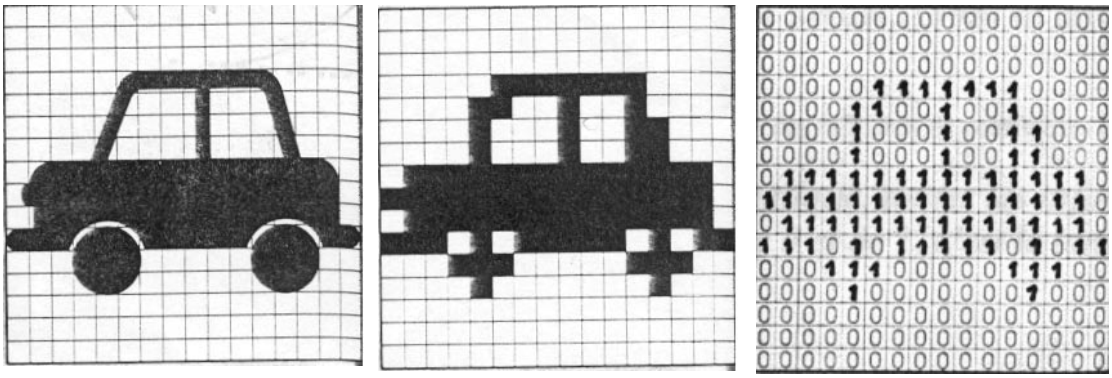
Ωστόσο ακόμα και οι επαυξημένες αναπαραστάσεις του EBCDIC είναι λίγες για τις σημερινές εφαρμογές των υπολογιστών. Η ανάγκη για ακόμη μεγαλύτερο αριθμό αναπαραστάσεων οδήγησε σε ένα παγκόσμιο στάνταρτ αναπαράστασης χαρακτήρων, που ονομάζεται Unicode. Το Unicode είναι ακόμη στο στάδιο της εξέλιξης. Στην έκδοση 2,0 αυτού του στάνταρτ έχουν καταχωρηθεί 38.885 διαφορετικοί χαρακτήρες που καλύπτουν αλφάβητα από τις πέντε ηπείρους του πλανήτη μας. Το Unicode χρησιμοποιεί 16 δυαδικά ψηφία για τις αναπαραστάσεις του. Ακόμα και αυτό το στάνταρτ όμως ίσως τελικά δεν επαρκέσει. Ευτυχώς, ήδη έχουμε προετοιμάσει το επόμενο το οποίο είναι υπερσύνολο του Unicode, χρησιμοποιεί αναπαραστάσεις των 32 δυαδικών ψηφίων και ονομάζεται 32-bit ISO 10646 Universal Character Set (UCS-4).

2.5.2. Αναπαράσταση εικόνας

Υπάρχουν διάφορα είδη εικόνας κατάλληλο το καθένα για συγκεκριμένα είδη εφαρμογών. Το πιο απλό σε σχέση με τη πολυπλοκότητα αναπαράστασής του, είναι οι ***διτονικές (duotone)*** εικόνες. Στις διτονικές εικόνες υπάρχουν μόνο δύο χρώματα (συνήθως μαύρο και άσπρο).

Μεγαλύτερη πολυπλοκότητα αναπαράστασης εμφανίζουν οι εικόνες ***συνεχούς τόνου (continuous tone)*** στις οποίες κάθε σημείο της εικόνας μπορεί να έχει σε αντίθεση με τις διτονικές, πολλές τονικές διαβαθμίσεις. Οι περισσότερες διαβαθμίσεις μας δίνουν και μια μεγαλύτερη ομαλότητα στην εικόνα. Στην κατηγορία των εικόνων συνεχούς τόνου έχουμε εικόνες ***κλίμακας του γκρίζου (gray scale)*** και ***έγχρωμες (color)***.

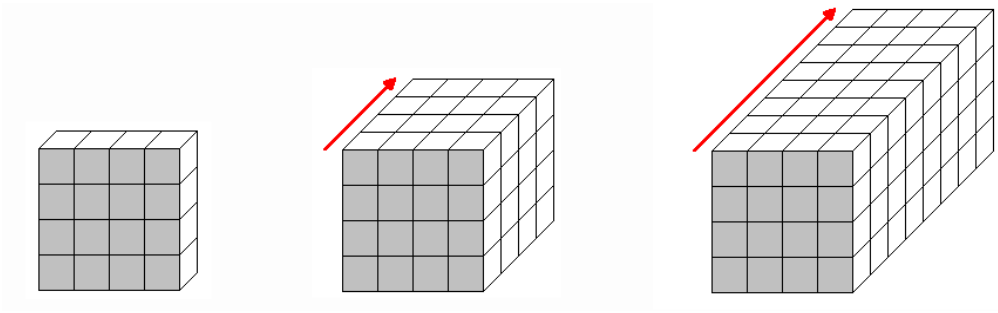
Η αναπαράσταση μιας εικόνας σε ένα υπολογιστικό σύστημα ξεκινά με την κατάτμησή της σε πολύ μικρές κουκίδες (εικονοστοιχεία, picture elements, pixels), με οριζόντιες και κάθετες νοητές γραμμές. Στις εικόνες (α) και (β) παρακάτω απεικονίζεται αυτή η διαδικασία για μια ασπρόμαυρη εικόνα ενός αυτοκινήτου. Παρατηρείστε ότι η διαδικασία αυτή επιφέρει παραμόρφωση της αρχικής εικόνας. Οσο μεγαλύτερος είναι ο αριθμός των κουκίδων στις οποίες διαιρείται μια εικόνα, τόσο πιο πιστή θα είναι η αναπαράστασή της στον υπολογιστή (τόσο μεγαλύτερη ανάλυση θα έχουμε).



(α) (β) (γ)

Ακολουθεί η αντιστοίχηση της πληροφορίας του κάθε εικονοστοιχείου σε κάποια από τις στάθμες τόνου. Για την περίπτωση μονόχρωμων εικόνων (μαύρο και άσπρο) απαιτείται ένα μόνο δυαδικό ψηφίο για αυτή την αντιστοίχηση, αφού κάθε εικονοστοιχείο μπορεί να είναι άσπρο ή μαύρο. Για τη προηγούμενη εικόνα, η διαδικασία αυτή φαίνεται στην εικόνα (γ) πιο πάνω.

Οσο περισσότεροι είναι οι τόνοι που επιθυμούμε να περιέχει η αναπαράσταση μιας εικόνα, τόσο περισσότερα είναι τα δυαδικά ψηφία που απαιτούνται για τον καθορισμό του τόνου κάθε κουκίδας με αντίστοιχες επιπτώσεις στο μέγεθος της αναπαράστασης. Για παράδειγμα, για 256 τόνους απαιτούνται 8 δυαδικά ψηφία.



Μονόχρωμη εικόνα	Εικόνα 16 χρωμάτων	Εικόνα 256 χρωμάτων
1 δυαδικό ψηφίο ανά pixel	4 δυαδικά ψηφία ανά pixel	8 δυαδικά ψηφία ανά pixel

Ως γνωστόν όλα τα χρώματα πλην του κόκκινου, του πράσινου και του μπλε, μπορούν να παραχθούν από τα παραπάνω. Για παράδειγμα το λευκό είναι η παρουσία στον ίδιο βαθμό των τριών βασικών χρωμάτων ενώ το μαύρο η απουσία και των τριών. Συνεπώς για μια έγχρωμη εικόνα, αρκεί να καθορίσουμε για κάθε εικονοστοιχείο της το ποσοστό συμβολής στο χρώμα του, κάθε ενός εκ των τριών βασικών χρωμάτων, ή με άλλα λόγια το τόνο κάθε βασικού χρώματος. Στην παρακάτω εικόνα φαίνεται η ανάλυση μιας έγχρωμης εικόνας στις βασικές συνιστώσες χρώματος.

Αρχική εικόνα

Μπλε Συνιστώσα

Πράσινη Συνιστώσα

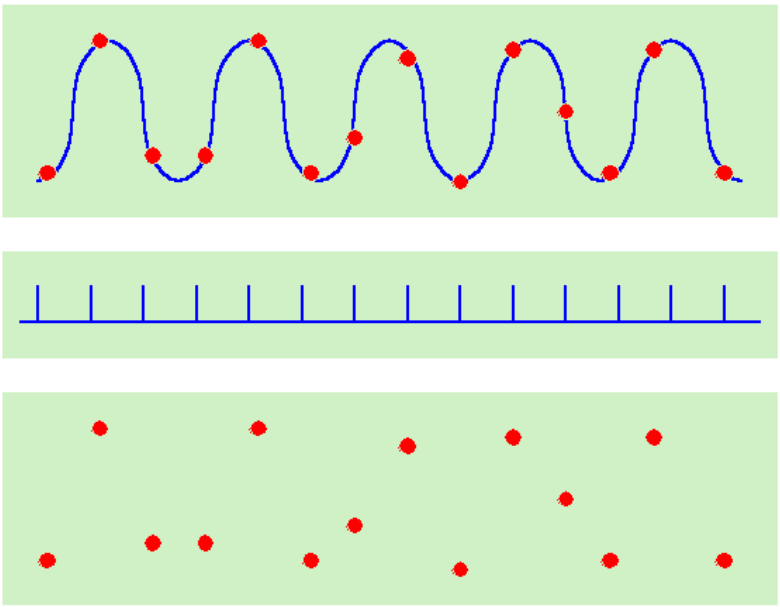
Κόκκινη Συνιστώσα

Ετσι, η αναπαράσταση μιας εικόνας που σε κάθε βασικό χρώμα επιτρέπει 256 τόνους, απαιτεί για κάθε εικονοστοιχείο της 24 δυαδικά ψηφία. Κάθε εικονοστοιχείο μπορεί να πάρει ένα από περίπου 16 εκατομμύρια χρώματα (256^3). Ο αριθμός των δυαδικών ψηφίων που αφιερώνεται για την αναπαράσταση μιας εικόνας ονομάζεται και βάθος (image depth - βλέπε και παραπάνω σχήμα).

2.5.3. Αναπαράσταση αναλογικού σήματος – Το παράδειγμα του ήχου

Για την αναπαράσταση ενός αναλογικού σήματος ακολουθούνται οι διαδικασίες της δειγματοληψίας (sampling) και της κβάντισης (quantification) των δειγμάτων.

Κατά τη διαδικασία της δειγματοληψίας καταγράφονται στιγμιαίες τιμές του πλάτους ενός σήματος ανά τακτά χρονικά διαστήματα. Ενα ερώτημα που προκύπτει είναι πόσος πρέπει να είναι ο μέγιστος χρόνος μεταξύ δύο δειγματοληψιών έτσι ώστε να μπορώ με μεγάλη ακρίβεια να ανασυνθέσω την αρχική μου μορφή ; Ο Nyquist έδειξε ότι προκειμένου για σήμα συχνότητας f, απαιτείται η δειγματοληψία να γίνεται με συχνότητα τουλάχιστον 2*f. Η διαδικασία της δειγματοληψίας φαίνεται στο παρακάτω σχήμα :



Αν υποθέσουμε ότι το αναλογικό μας σήμα είναι ο ήχος. Είναι γνωστό ότι ο άνθρωπος αντιλαμβάνεται ήχους εντός του διαστήματος συχνοτήτων 20Hz – 20KHz. Συνεπώς μια πιστή για τα ανθρώπινα δεδομένα αναπαράσταση του ήχου,

πρέπει να περιλαμβάνει διαδικασία δειγματοληψίας με συχνότητα τουλάχιστον 40KHz.

Με τη διαδικασία της κβάντισης των δειγμάτων στην ουσία διαιρούμε το συνεχές διάστημα τιμών ενός αναλογικού σήματος σε διακριτά διαστήματα και αναθέτουμε μια δυαδική κωδικοποίηση για κάθε διακριτό διάστημα. Εστω για παράδειγμα ότι έχουμε ένα σήμα δυναμικού το οποίο μπορεί να πάρει όλες τις δυνατές τιμές μεταξύ –2 και +6 V. Ας υποθέσουμε ότι κβαντίζουμε αυτό το διάστημα τιμών σε 8 διαστήματα του 1V, και αναπαριστούμε το καθένα από αυτά με τρία δυαδικά ψηφία ξεκινώντας από το 000 για το [-2, -1] έως το 111 για το (5, 6]. Τότε ένα δυναμικό της τάξης των 1.2V θα παρασταθεί με το δυαδικό 100. Προφανώς εδώ υπάρχει ένα "παζάρι" μεταξύ ακρίβειας αναπαράστασης και των απαιτούμενων δυαδικών ψηφίων. Η παραπάνω κβάντιση προσφέρει ακρίβεια της τάξης του 0,5V αλλά απαιτεί μόνο τρία δυαδικά ψηφία. Αν χωρίζαμε το διάστημα σε 256 διακριτά διαστήματα τότε θα είχαμε ακρίβεια 0,015625V αλλά θα χρειαζόμασταν και 8 δυαδικά ψηφία.

ΚΕΦΑΛΑΙΟ 3

Αριθμητικές πράξεις

Στο προηγούμενο κεφάλαιο αναπτύξαμε διάφορους τρόπους με τους οποίους μπορούμε να αναπαραστήσουμε αριθμούς σε ένα υπολογιστικό σύστημα. Σε αυτό το κεφάλαιο θα καλύψουμε τους τρόπους με τους οποίους υλοποιούνται οι βασικές αριθμητικές πράξεις. Πιο σύνθετες πράξεις μπορούν να αναχθούν σε συνδυασμούς των βασικών πράξεων.

Η αριθμητική υπολογιστών συνεχίζει ακόμη και σήμερα να προσφέρει ερευνητικές ευκαιρίες, αφού σύνθετα προβλήματα όπως οι προβλέψεις για τον καιρό, εξομοιώσεις τροχιών ηλεκτρονίων κλπ. καθιστούν ακόμη και τα σημερινά συστήματα, χαμηλής απόδοσης συστήματα για τα συγκεκριμένα προβλήματα παρά την ενσωμάτωση τεχνικών για αύξηση της απόδοσης.

Θα ξεκινήσουμε την αναφορά μας στην αριθμητική υπολογιστών εξετάζοντας πρώτα ακεραίους αριθμούς. Σήμερα το σύνολο των υπολογιστικών συστημάτων χρησιμοποιεί για τους ακεραίους αριθμούς την αναπαράσταση συμπληρώματος ως προς 2. Ιστορικά ωστόσο, η πρώτη παράσταση που χρησιμοποιήθηκε ήταν αυτή του συμπληρώματος ως προς 1 και γι' αυτό παρακάτω την εξετάζουμε πρώτη. Σε όλα μας τα παραδείγματα παρακάτω υιοθετούμε ακρίβεια αναπαράστασης 8 δυαδικών ψηφίων. Επίσης θα πρέπει να λάβετε υπόψη σας ότι η πρόσθεση και η αφαίρεση εξετάζονται συνολικά σαν μια πράξη, αφού η αφαίρεση Α-Β ισοδυναμεί με την πρόσθεση Α+(-Β) όπου το –Β εκφράζεται σύμφωνα με τους κανόνες κάθε συστήματος αναπαράστασης όπως αυτοί εισήχθησαν στο προηγούμενο κεφάλαιο.

3.1 Πρόσθεση / Αφαίρεση σε Συμπλήρωμα ως προς 1

Στην αριθμητική συμπληρώματος ως προς 1 εισάγεται η έννοια του επανεισαγόμενου κρατουμένου (end-around-carry). Συγκεκριμένα, εάν κατά μία πράξη πρόσθεσης ή αφαίρεσης με αριθμούς συμπληρώματος ως προς 1 προκύψει κρατούμενο εξόδου (κρατούμενο πέραν της τελευταίας βαθμίδας αναπαράστασης), τότε αυτό ***δεν*** θα πρέπει να αγνοηθεί, αλλά θα πρέπει να προστεθεί με δεύτερη πράξη στο αποτέλεσμα.

Η λογική του επανεισαγόμενου κρατουμένου βασίζεται πάνω στο γεγονός της διπλής αναπαράστασης του 0. Υποθέστε ότι έχω να προσθέσω το $A=-100_{10}$ με το $B=101_{10}$. Τότε σε παράσταση συμπληρώματος ως προς 1 έχω :

$$A = -100_{10} = 10011011_2$$

$$B = +101_{10} = 01100101_2$$

Το αποτέλεσμα της πρόσθεσης θα είναι 00000000_2 και 1 στο κρατούμενο εξόδου. Ας δούμε γιατί πρέπει να προσθέσουμε αυτό το κρατούμενο εξόδου στο αποτέλεσμα. Καθώς κάθε μονάδα του Β προστίθεται στο Α, το απόλυτο μέτρο της δυαδικής αναπαράστασης του Α μεγαλώνει. Με άλλα λόγια αν προσθέταμε το 1 θα είχαμε $A + 1 = 10011100_{1's} = -99_{10}$. Συνεχίζοντας με αυτόν τον τρόπο αν προσθέταμε το 99_{10} θα παίρναμε την παράσταση $11111110_{1's}$, δηλαδή σε συμπλήρωμα ως προς 1 την παράσταση του –1. Αν προσθέταμε το 100_{10} θα παίρναμε την παράσταση $11111111_{1's}$ δηλαδή το –0. Προσέξτε ότι σε καμία από αυτές τις προσθέσεις δε θα προέκυπτε κρατούμενο εξόδου. Οταν όμως προσθέσω το 101_{10} στην ουσία παίρνω λανθασμένα την δεύτερη παράσταση του 0, δηλαδή την $00000000_{1's}$. Στην ουσία δηλαδή έχω προσμετρήσει τον αριθμό 0 δύο φορές, όπως κινούμαι πάνω στον άξονα των ακεραίων αριθμών. Επιπλέον η υπέρβαση του μικρότερου αρνητικού αριθμού ($11111111_{1's}$) που είναι σε απόλυτη τιμή και ο μεγαλύτερος αριθμός που μπορώ να παραστήσω με 8 δυαδικά ψηφία μου υποδεικνύει αυτή την ανακολουθία μεταξύ ακεραίων και παραστάσεων με τη δημιουργία του κρατουμένου εξόδου. Για να την αποκαταστήσω χρειάζεται να προχωρήσω στην επόμενη παράσταση και αυτό επιτυγχάνεται με την πρόσθεση του κρατουμένου εξόδου.

Μερικά παραδείγματα θα αποσαφηνίσουν τα παραπάνω.

```
  01001011 (+75)                    11000101 (- 58)
  10001001 (-118) +                 01110010 (+114)+
0 11010100 (-43) Σωστό αποτέλεσμα 1 00110111 Λάθος αποτέλεσμα
                                             1
                                    00111000 (+56) Σωστό αποτέλεσμα
```

Προσέξτε ότι δεν θα πρέπει να συγχέετε τη λογική του επανεισαγόμενου κρατούμενου με την υπερχείλιση. Μια πράξη που οδηγεί σε υπερχείλιση είναι πάντα λανθασμένη ανεξάρτητα του αν επανεισάγετε ή όχι το κρατούμενο. Προσέξτε τις παρακάτω προσθέσεις :

```
    01001011 (+75)                          11000101 (- 58)
    01000000 (+64) +                        10001101 (-114)+
 0  10001011 (-116) Λάθος αποτέλεσμα    1  01010010 Λάθος αποτέλεσμα
                                                   1
                                            01010011 (+83) Λάθος αποτέλεσμα
```

Το πρόβλημα στις παραπάνω προσθέσεις είναι ότι ξεπεράσαμε τις μέγιστες τιμές αναπαράστασης ([–127, +127], δες προηγούμενο κεφάλαιο) και όχι το επανεισαγόμενο κρατούμενο. Θα δούμε στην αριθμητική συμπληρώματος ως προς 2 τρόπους διαπίστωσης της υπερχείλισης.

Η ανάγκη για τον διαχωρισμό μεταξύ των δύο παραστάσεων του 0 και η πιθανή ανάγκη για δεύτερη πρόσθεση στην περίπτωση που το κρατούμενο εξόδου είναι 1 είναι δύο λόγοι για να προτιμήσει κανείς την αριθμητική συμπληρώματος ως προς 2. Οι λόγοι αυτοί έχουν οδηγήσει στην σταδιακή εγκατάλειψη της αριθμητικής συμπληρώματος ως προς 1 στους εμπορικούς υπολογιστές. Ωστόσο η αριθμητική αυτή συνεχίζει να υπάρχει σε αρκετούς υπερυπολογιστές για άλλους λόγους.

3.2 Πρόσθεση / Αφαίρεση σε Συμπλήρωμα ως προς 2

Η αριθμητική συμπληρώματος ως προς 2 είναι απαλλαγμένη από την έννοια του επανεισαγόμενου κρατουμένου. Επιπλέον, προσφέρει τη δυνατότητα η πρόσθεση και η αφαίρεση δύο αριθμών να γίνονται από το ίδιο υλικό. Τη δυνατότητα αυτή θα την αναπτύξουμε πιο κάτω. Ας δούμε αρχικά μερικά παραδείγματα πρόσθεσης (ισοδύναμα αφαίρεσης) δύο αριθμών στο σύστημα αυτό.

```
    00001010  (+10)          00000101  (+5)          11111111  (-1)
    01010000  (+80) +        11111110  (-2) +        11111100  (-4)+
 0  01011010  (+90)       1  00000011  (+3)       1  11111011  (-5)
```

Οπως φαίνεται και από τα δύο δεξιότερα παραδείγματα το κρατούμενο εξόδου από την πλέον αριστερή βαθμίδα στο σύστημα αυτό μπορεί να αγνοηθεί μιας και πλέον υπάρχει μία προς μία αντιστοιχία μεταξύ των ακεραίων ενός διαστήματος και των αναπαραστάσεων που προσφέρει το σύστημα αναπαράστασης σε συμπλήρωμα ως προς 2.

Ωστόσο και σε αυτό το σύστημα υπάρχει η πιθανότητα υπερχείλισης. Είναι προφανές ότι δε μπορεί να συμβεί υπερχείλιση κατά την αφαίρεση δύο ομοσήμων αριθμών ή κατά την πρόσθεση δύο ετεροσήμων αριθμών, αφού το μέτρο του αποτελέσματος και στις δύο παραπάνω περιπτώσεις είναι μικρότερο από το μέτρο του μεγαλύτερου εκ των δύο τελουμένων (operands). Υπερχείλιση συνεπώς μπορεί να συμβεί μόνο κατά την πρόσθεση δύο ομοσήμων ή κατά την αφαίρεση δύο ετεροσήμων αριθμών. Ας δούμε μερικά παραδείγματα.

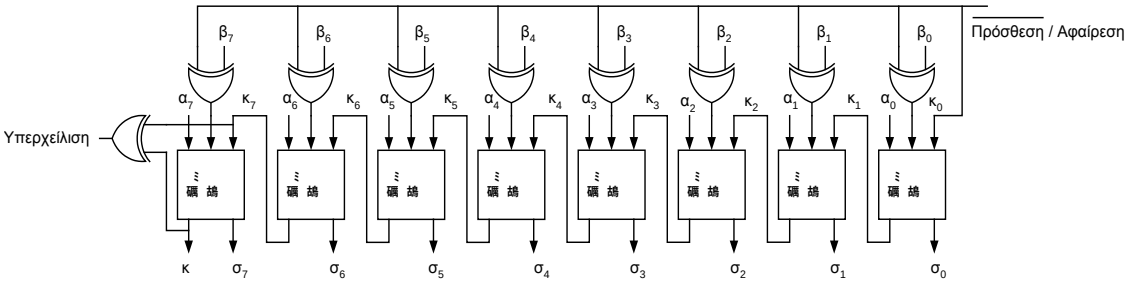
```
  00110010  (+50)                 11010001  (-47)
  01010000  (+80) +               10100110  (-90) +
0 10000010  (-126) !!!          1 01110111  (+119)!!!
```

Τα παραπάνω λανθασμένα αποτελέσματα δε θα πρέπει να μας εκπλήσσουν, αφού στο προηγούμενο κεφάλαιο είδαμε ότι με 8 δυαδικά ψηφία χρησιμοποιώντας αναπαράσταση συμπληρώματος ως προς 2 μπορούμε να αναπαριστούμε ακεραίους στο διάστημα [-128, +127]. Εφόσον τα αναμενόμενα αποτελέσματα (+130, -137) είναι εκτός του διαστήματος αυτού, μοιραία παίρνουμε λανθασμένα αποτελέσματα.

Μπορούμε να μαθηματικοποιήσουμε λίγο περισσότερο τις παραπάνω παρατηρήσεις μας ώστε να κατασκευάσουμε ένα κύκλωμα ανίχνευσης του φαινομένου της υπερχείλισης. Θεωρούμε μόνο τη πράξη της πρόσθεσης, αφού τελικά η πράξη της αφαίρεσης υλοποιείται και αυτή με πρόσθεση. Οπως είδαμε παραπάνω υπερχείλιση μπορεί να συμβεί μόνο κατά την πρόσθεση ομοσήμων και αυτή εκδηλώνεται με το να έχουμε αποτέλεσμα αντίθετου προσήμου. Ας θεωρήσουμε λοιπόν ότι τα έντελα της πρόσθεσης είναι $A = a_7a_6a_5a_4a_3a_2a_1a_0$ και $B = \beta_7\beta_6\beta_5\beta_4\beta_3\beta_2\beta_1\beta_0$. Εστω $\sigma_7\sigma_6\sigma_5\sigma_4\sigma_3\sigma_2\sigma_1\sigma_0$ το αποτέλεσμα της πρόσθεσης, κ το κρατούμενο εξόδου και $\kappa_7\kappa_6\kappa_5\kappa_4\kappa_3\kappa_2\kappa_1 0$ τα κρατούμενα εισόδου στις αντίστοιχες βαθμίδες. Υπερχείλιση συνεπώς έχουμε όταν $a_7 = \beta_7 = 1$ και $\sigma_7 = 0$ ή όταν $a_7 = \beta_7 = 0$ και $\sigma_7 = 1$. Η πρώτη περίπτωση μπορεί να συμβεί μόνον όταν $\kappa_7 = 0$ και στην περίπτωση αυτή είναι κ = 1, ενώ η δεύτερη περίπτωση μπορεί να συμβεί όταν $\kappa_7 = 1$ και μας δίνει κ = 0. Συνεπώς η κατάσταση υπερχείλισης μπορεί να ανιχνευτεί με τη διαπίστωση ανισότητας μεταξύ των σημάτων κ_7 και κ, ή με άλλα λόγια με μία απλή πύλη αποκλειστικής διάζευξης μεταξύ αυτών των δύο σημάτων.

Το παρακάτω σχήμα μας δίνει μια μονάδα πρόσθεσης / αφαίρεσης δύο αριθμών σε αριθμητική συμπληρώματος ως προς 2. Η υλοποίηση βασίζεται στο γνωστό σας από τον Λογικό Σχεδιασμό πλήρη αθροιστή ενός δυαδικού ψηφίου και στη λογική διάδοσης του κρατουμένου (ripple carry). Οταν η γραμμή επιλογής της επιθυμητής λειτουργίας είναι στο λογικό 0, τότε η μονάδα μας εκτελεί την πράξη

της πρόσθεσης. Στην περίπτωση που η γραμμή επιλογής είναι στο λογικό 1 τότε κάθε δυαδικό ψηφίο του εντέλου Β αντιστρέφεται με τη βοήθεια των πυλών αποκλειστικής διάζευξης και το κρατούμενο στη λιγότερο σημαντική βαθμίδα γίνεται 1 έτσι ώστε να σχηματιστεί το συμπλήρωμα ως προς 2 του Β (με άλλα λόγια η πράξη Α-Β υλοποιείται σαν $A+B_{2's}$). Στο σχήμα έχουμε επίσης προσθέσει μια πύλη αποκλειστικής διάζευξης ώστε να παίρνουμε την ένδειξη υπερχείλισης σύμφωνα με τα όσα αναφέρθηκαν πιο πάνω.



3.3 Πρόσθεση / Αφαίρεση στις υπόλοιπες αναπαραστάσεις

Για να προσθέσουμε ή να αφαιρέσουμε δύο αριθμούς οι οποίοι αναπαρίστανται σε πρόσημο – μέτρο, θα πρέπει να διακρίνουμε τις εξής περιπτώσεις προκειμένου για την πρόσθεση (για την αφαίρεση ισχύουν αντίστοιχα πράγματα) :

- Αν οι αριθμοί είναι ομόσημοι, τότε το πρόσημο του αποτελέσματος είναι το ίδιο με το πρόσημο των εντέλων. Το μέτρο του αποτελέσματος είναι το άθροισμα των μέτρων των δύο εντέλων.
- Αν οι αριθμοί είναι ετερόσημοι, τότε απαιτείται σύγκριση των μέτρων των δύο αριθμών. Η σύγκριση μπορεί να επιτευχθεί με τη χρήση ενός κυκλώματος συγκριτή όπως αυτά που μάθατε στο μάθημα του Λογικού Σχεδιασμού. Το πρόσημο του αποτελέσματος είναι το πρόσημο του αριθμού με το μεγαλύτερο μέτρο ενώ το μέτρο του αποτελέσματος είναι η διαφορά των μέτρων των δύο αριθμών.

Μερικά παραδείγματα εφαρμογής των πιο πάνω κανόνων φαίνονται παρακάτω.

00110010	(+50)		10101111	(-47)
00010000	(+16) +		00100000	(+32) +
01000010	(+66)		10001111	(-15)

όπου η αφαίρεση δύο δυαδικών ψηφίων ακολουθεί τον παρακάτω πίνακα αληθείας :

Είσοδοι			Εξοδοι	
Δυαδικό Ψηφίο Αφαιρετέου	**Δυαδικό Ψηφίο Αφαιρέτη**	**Δανεικό από την προηγούμενη βαθμίδα**	**Υπολογιζόμενο δυαδικό ψηφίο υπολοίπου**	**Υπολογιζόμενο δυαδικό ψηφίο δανεικού**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Η πρόσθεση / αφαίρεση αριθμών που βρίσκονται σε αριθμητική πλεονασμού κατά κ ακολουθεί τις εξής διαδικασίες :

- Για την πρόσθεση μπορούμε είτε να προσθέσουμε τους δύο αριθμούς ως μη προσημασμένους και κατόπιν να αφαιρέσουμε την πόλωση κ, είτε να μετατρέψουμε τους αριθμούς σε παράσταση προσήμου – μέτρου αφαιρώντας τους την πόλωση, μετά να ακολουθήσουμε την διαδικασία πρόσθεσης αριθμών σε πρόσημο – μέτρο όπως αναπτύχθηκε πιο πάνω και στο αποτέλεσμά της να προσθέσουμε την πόλωση.
- Για την αφαίρεση μπορούμε να θεωρήσουμε τους δύο αριθμούς σαν να ήταν σε παράσταση προσήμου – μέτρου με θετικό πρόσημο, να ακολουθήσουμε τη διαδικασία αφαίρεσης δύο αριθμών σε πρόσημο – μέτρο και να προσθέσουμε στο αποτέλεσμα την πόλωση. Εναλλακτικά θα μπορούσαμε να ακολουθήσουμε διαδικασία ανάλογη με τη δεύτερη περιγραφόμενη πιο πάνω.

Και στις δύο πιο πάνω περιπτώσεις και ανεξάρτητα από την ακολουθούμενη διαδικασία χρειάζεται προσεκτικός σχεδιασμός γιατί είναι πιθανόν η περιορισμένη ακρίβεια αναπαράστασης να μας οδηγήσει σε υπερχειλίσεις των ενδιάμεσων αποτελεσμάτων.

3.4 Μερικές χρήσιμες λογικές πράξεις

Στο μάθημα του Λογικού σας Σχεδιασμού έχετε διδαχθεί τις πιο πολλές από τις λογικές συναρτήσεις και τους πίνακες αληθείας τους. Πέρα από αυτές, μπορούμε να ορίσουμε και άλλους μοναδιαίους τελεστές που ισοδυναμούν με λογικές πράξεις οι οποίες βασίζονται σε ολισθήσεις (shifts) των δυαδικών ψηφίων μιας ψηφιολέξης. Οι λογικές αυτές πράξεις έχουν ιδιαίτερη σημασία στην αριθμητική υπολογιστών και είναι :

- Λογική ολίσθηση (logic shift) προς τα αριστερά ή τα δεξιά, κατά κ θέσεις. Κατά τη λειτουργία αυτή κάθε δυαδικό ψηφίο της αρχικής μας ψηφιολέξης ολισθαίνει προς τα αριστερά ή τα δεξιά κατά κ θέσεις. Οι κενές θέσεις που δημιουργούνται συμπληρώνονται με 0. Για παράδειγμα ας θεωρήσουμε τη ψηφιολέξη 011101.

Λογική ολίσθηση κατά δύο θέσεις προς τα δεξιά θα μας οδηγήσει στη ψηφιολέξη 000111 ενώ λογική ολίσθηση κατά μια θέση προς τα αριστερά θα μας οδηγήσει στη ψηφιολέξη 111010. Μπορεί να αποδειχθεί ότι αν θεωρήσουμε τις αρχικές μας ψηφιολέξεις σαν μη προσημασμένους δυαδικούς αριθμούς, η ολίσθηση προς τα αριστερά κατά κ θέσεις οδηγεί σε πολλαπλασιασμό μιας ψηφιολέξης κατά $2^κ$, ενώ η αντίστοιχη ολίσθηση προς τα δεξιά οδηγεί στο πηλίκο της διαίρεσης της αρχικής ψηφιολέξης μας με το $2^κ$.

- Αριθμητική ολίσθηση (arithmetic shift) προς τα δεξιά κατά κ θέσεις. Η ολίσθηση αυτή διαφέρει από την λογική ολίσθηση προς τα δεξιά στο ότι στα αριστερότερα κ ψηφία που απελευθερώνονται κατά την ολίσθηση δεν εισάγεται το 0 αλλά αντιγράφεται τα αριστερότερο ψηφίο της αρχικής μας ψηφιολέξης. Για παράδειγμα η αριθμητική ολίσθηση κατά 2 θέσεις της ψηφιολέξης 111000 οδηγεί στην ψηφιολέξη 111110. Αν θεωρήσετε ότι οι δύο παραπάνω ψηφιολέξεις είναι αριθμοί σε συμπλήρωμα ως προς 2, μπορείτε να διαπιστώσετε ότι με την αριθμητική ολίσθηση επεκτείνουμε την ιδιότητα πολλαπλασιασμού και διαίρεσης με δυνάμεις του 2 και σε προσημασμένους αριθμούς.

- Κυκλική ολίσθηση (rotation) προς τα δεξιά ή τα αριστερά, κατά κ θέσεις. Η κυκλική ολίσθηση διαφέρει από την λογική στο ότι οι θέσεις των δυαδικών ψηφίων που αδειάζουν γεμίζουν με τα δυαδικά ψηφία που φεύγουν από την ψηφιολέξη στο απέναντι άκρο της. Θεωρείστε την ψηφιολέξη 100101 και μια κυκλική ολίσθηση προς τα δεξιά. Το δυαδικό ψηφίο 1 που φεύγει από το δεξιό άκρο της ψηφιολέξης επανεισάγεται στο αριστερό της άκρο και συνεπώς η κυκλική ολίσθηση θα μας δώσει σαν αποτέλεσμα τη λέξη 110010.

Οι παραπάνω λειτουργίες ολίσθησης πραγματοποιούνται σε κάθε μοντέρνο υπολογιστικό σύστημα από ένα κύκλωμα που ονομάζεται ολισθητής (shifter) και υλοποιείται σύμφωνα με τα όσα έχετε μάθει στο Λογικό Σχεδιασμό.

3.5 Πολλαπλασιασμός μη προσημασμένων αριθμών

Αν προσπαθούσαμε στο χαρτί να πολλαπλασιάσουμε δύο μη προσημασμένους αριθμούς του δυαδικού συστήματος θα διαπιστώναμε ότι όσα έχουμε μάθει για το δεκαδικό σύστημα εφαρμόζονται και στο δυαδικό άμεσα. Το παρακάτω σχήμα δείχνει τον πολλαπλασιασμό του 11 με το 9 στο δυαδικό σύστημα.

			1	0	1	1	(11_{10})
		x	1	0	0	1	(9_{10})
			1	0	1	1	
		0	0	0	0		
	0	0	0	0			
1	0	1	1			+	
1	1	0	0	0	1	1	(99_{10})

Μπορούμε να κάνουμε ορισμένες παρατηρήσεις με βάση το παραπάνω παράδειγμα. Η πρώτη είναι ότι λόγω του δυαδικού συστήματος κάθε εξεταζόμενο ψηφίο του πολλαπλασιαστή μπορεί να έχει τις τιμές 0 και 1, με αποτέλεσμα τα αντίστοιχα μερικά γινόμενα που προκύπτουν για αυτές τις τιμές είναι είτε 0 είτε ο πολλαπλασιαστέος. Συνεπώς η προπαίδεια στο δυαδικό σύστημα δεν έχει καμία αξία. Η δεύτερη παρατήρηση είναι ότι ανάλογα με τη θέση του κάθε δυαδικού ψηφίου του πολλαπλασιαστή που είναι 1, το μερικό γινόμενο που προκύπτει έχει τιμή την τιμή του πολλαπλασιαστέου αριστερά ολισθημένου κατά τόσες θέσεις όση είναι και η θέση του ψηφίου. Για παράδειγμα το τελευταίο μερικό γινόμενο έχει αριθμητική τιμή 1011000, που προκύπτει από την αριστερή ολίσθηση του πολλαπλασιαστέου κατά 3 θέσεις. Η τελευταία παρατήρηση που μπορούμε να κάνουμε είναι ότι ο πολλαπλασιασμός δύο ποσοτήτων των κ και λ δυαδικών ψηφίων μας οδηγεί σε ένα αποτέλεσμα των κ + λ –1 δυαδικών ψηφίων.

Οι δύο πρώτες παρατηρήσεις είναι ικανές για να μας οδηγήσουν στο συμπέρασμα ότι ο πολλαπλασιασμός δύο δυαδικών αριθμών μπορεί να πραγματοποιηθεί με διαδοχικές προσθέσεις και ολισθήσεις. Παρακάτω περιγράφουμε σε απλά βήματα τη διαδικασία πολλαπλασιασμού του $A = a_{κ-1}a_{κ-2} \ldots a_0$ με το $B = \beta_{λ-1}\beta_{λ-2} \ldots \beta_0$. Η περιγραφή της λύσης ενός προβλήματος με απλά βήματα ονομάζεται ***αλγόριθμος***.

Βήμα 1. Θέσε δ=0 και Γ= Α και Αποτέλεσμα = 0.
Βήμα 2. Εξέτασε το δυαδικό ψηφίο $\beta_δ$.
Βήμα 3. Εάν είναι 0 πήγαινε στο βήμα 5.
Βήμα 4. Πρόσθεσε το Γ στο Αποτέλεσμα.
Βήμα 5. Θέσε Γ = Αριστερή Ολίσθηση κατά μια θέση του Γ.
Βήμα 6. Αύξησε κατά 1 το δ.
Βήμα 7. Είναι δ = λ ; Αν ναι τότε ο πολλαπλασιασμός ολοκληρώθηκε. Αν όχι πήγαινε στο Βήμα 2.

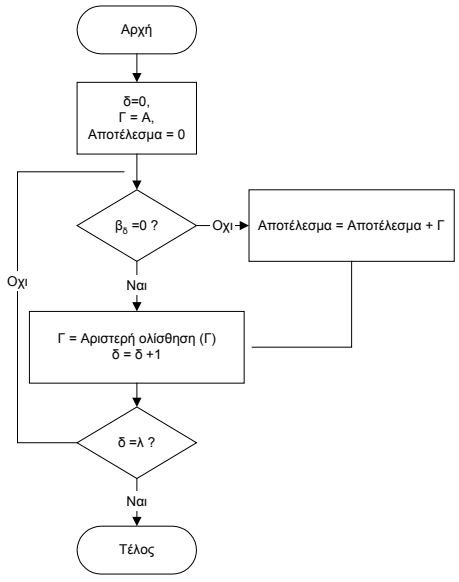
Ας παρακολουθήσουμε την εκτέλεση του παραπάνω αλγορίθμου για τον πολλαπλασιασμό Α=4=100 με το Β=5=101 βήμα προς βήμα.

Βήμα που εκτελείται	Τιμές των Μεταβλητών / Γεγονότα
1	δ=0, Γ=4, Αποτέλεσμα = 0
2	Εξέταση του β_0
3	Εκτέλεση του Βήματος 4
4	Αποτέλεσμα = 0 + 4 = 4
5	Γ = 8
6	δ = δ+1 = 1
7	Εκτέλεση του Βήματος 2
2	Εξέταση του β_1
3	Εκτέλεση του βήματος 5
5	Γ = 16
6	δ = δ+1 = 2
7	Εκτέλεση του Βήματος 2
2	Εξέταση του β_2
3	Εκτέλεση του Βήματος 4
4	Αποτέλεσμα = 4 + 16 = 20
5	Γ = 32
6	δ = δ+1 = 3
7	Πολλαπλασιασμός ολοκληρώθηκε

Πολλές φορές είναι έναν αλγόριθμο αντί να τον περιγράφουμε με λόγια να κατασκευάζουμε ένα διάγραμμα για την απεικόνισή του. Τα διαγράμματα αυτά ονομάζονται διαγράμματα ροής (flow charts). Στα διαγράμματα ροής συνήθως χρησιμοποιούνται τα εξής σχήματα :

- Το παραλληλόγραμμο για να ορίσει πράξεις που πρέπει να γίνουν
- Ο ρόμβος για να δείξει σημεία απόφασης
- Η έλλειψη για τη σηματοδότηση της αρχής και του τέλους ενός διαγράμματος

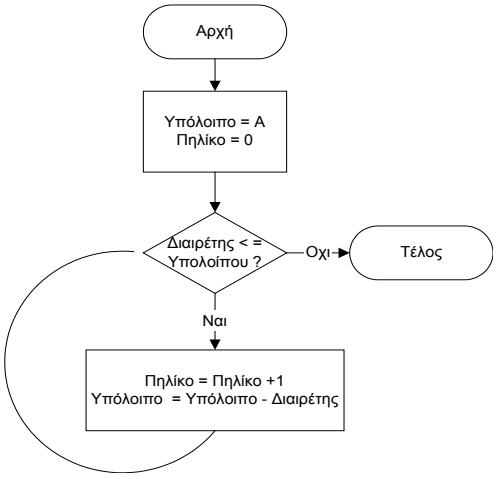
Τα παραπάνω σχήματα συνδέονται μεταξύ τους με βέλη που δείχνουν και τη σειρά εκτέλεσης (ροή) των βημάτων του αλγορίθμου. Ενα διάγραμμα ροής για τον πιο πάνω αλγόριθμο είναι το παρακάτω :



3.6 Διαίρεση μη προσημασμένων αριθμών

Με τρόπο αντίστοιχο με αυτόν του πολλαπλασιασμού, μπορούμε να εκφράσουμε τη διαίρεση δύο μη προσημασμένων αριθμών με έναν αλγόριθμο ή ένα διάγραμμα ροής που βασίζεται σε διαδοχικές επιτυχείς αφαιρέσεις του διαιρέτη από τον διαιρετέο. Υποθέτοντας ότι ο διαιρετέος έχει μήκος κ+λ δυαδικά ψηφία και ο διαιρέτης κ δυαδικά ψηφία, το πηλίκο της διαιρέσεως θα έχει μήκος λ ψηφία και το υπόλοιπο το πολύ κ δυαδικά ψηφία.

Το διάγραμμα ροής που περιγράφει τον αλγόριθμο των διαδοχικών αφαιρέσεων φαίνεται παρακάτω.



Από τα παραπάνω είναι προφανές ότι ένας πολλαπλασιασμός ή μια διαίρεση στο δυαδικό σύστημα είναι μια σημαντικά περισσότερο χρονοβόρα διαδικασία συγκρινόμενη με αυτήν της αφαίρεσης / πρόσθεσης, αφού ένας πολλαπλασιασμός για παράδειγμα μπορεί να απαιτήσει έως και λ προσθέσεις. Για το σκοπό αυτό στους προσημασμένους ακεραίους έχουν επινοηθεί τρόποι επιτάχυνσης των διαδικασιών αυτών όπως θα δούμε παρακάτω.

3.7 Πολλαπλασιασμός προσημασμένων αριθμών

Αν προσπαθήσουμε να εφαρμόσουμε την απλή μέθοδο του χαρτιού για τον πολλαπλασιασμό προσημασμένων αριθμών θα δούμε ότι καταλήγουμε σε λάθος αποτελέσματα. Αυτά οφείλονται στο γεγονός ότι δε λαμβάνουμε υπόψη μας το πρόσημο των μερικών γινομένων.

Εστω ότι πολλαπλασιάζουμε το Α = +5 με το Β=-3 σε αριθμητική συμπληρώματος ως προς 2. Τότε έχουμε :

			1	1	0	1		($-3_{2's}$)
		x	0	1	0	1		($+5_{2's}$)
			1	1	0	1		
		0	0	0	0			
	1	1	0	1				
0	0	0	0			+		
1	0	0	0	0	0	1		($-63_{2's}$) !!!

Είναι προφανές ότι το λάθος στον παραπάνω πολλαπλασιασμό προκαλείται από το γεγονός ότι ενώ το 1ο και το 3ο μερικό γινόμενο θα έπρεπε να είναι αρνητικοί αριθμοί εμείς τους παράγουμε ως θετικούς.

Λύση στο πρόβλημα αυτό μπορεί να επιτευχθεί με την επέκταση προσήμου των αρνητικών αριθμών. Αυτό σημαίνει ότι κάθε αρνητικό έντελο του πολλαπλασιασμού επεκτείνεται ώστε από την αρχή να καταλάβει τα αναμενόμενα δυαδικά ψηφία του αποτελέσματος. Για παράδειγμα στον παραπάνω πολλαπλασιασμό που έχουμε από 4 δυαδικά ψηφία σε κάθε έντελο, αναμένουμε ένα αποτέλεσμα των 8 δυαδικών ψηφίων. Για ορθότητα, επεκτείνουμε τον πολλαπλασιαστέο σε 8 δυαδικά ψηφία και παίρνουμε :

1	1	1	1	1	1	0	1	($-3_{2's}$)
x				0	1	0	1	($+5_{2's}$)
1	1	1	1	1	1	0	1	
0	0	0	0	0	0	0		
1	1	1	1	0	1			
0	0	0	0	0			+	
1	1	1	1	0	0	0	1	($-15_{2's}$)

Προσέξτε ότι τόσο στα μερικά γινόμενα όσο και στο τελικό αποτέλεσμα αγνοούμε τα δυαδικά ψηφία πέρα από το 8ο γιατί εξαρχής γνωρίζουμε την ακρίβεια αναπαράστασης που θα απαιτούσε το αποτέλεσμα. Παρατηρείστε επίσης στην περίπτωση που ο πολλαπλασιαστής είναι αρνητικός η επέκταση προσήμου θα προκαλούσε τη δημιουργία πολλών μερικών γινομένων διαφορετικών του 0, με αποτέλεσμα ο πολλαπλασιασμός να χρειάζεται περισσότερες προσθέσεις και συνεπώς να είναι ακόμα πιο αργός από την πρόσθεση.

3.8 Μείωση του χρόνου του πολλαπλασιασμού – Αλγόριθμοι Booth

Εστω Α ο πολλαπλασιαστέος και Β ο πολλαπλασιαστής σε μια πράξη πολλαπλασιασμού. Εστω $\beta_κ\ \beta_{κ-1}\ \beta_{κ-2}\ \ldots\ \beta_λ$ μια σειρά διαδοχικών δυαδικών ψηφίων του πολλαπλασιαστή που είναι όλα στο 1. Αυτά τα ψηφία δημιουργούν τα μερικά γινόμενα $A^* \beta_κ * 2^κ$, $A^* \beta_{κ-1} * 2^{κ-1}$, … , $A^* \beta_λ * 2^λ$, των οποίων η πρόσθεση μας

δίνει $A * 2^{\lambda} * (2^{\kappa-\lambda} + 2^{\kappa-1-\lambda} + \ldots + 2^{0})$. Παρατηρείστε ότι η ποσότητα εντός παρενθέσεως είναι ίση με $2^{\kappa-\lambda+1} - 1$. Η ποσότητα όμως εντός της παρενθέσεως απαιτεί την πρόσθεση κ-λ μερικών γινομένων, ενώ αντίθετα η ισοδύναμη ποσότητα απαιτεί μία και μόνη αφαίρεση δύο μερικών γινομένων. Εφόσον σε συμπλήρωμα ως προς 2 η αφαίρεση και η πρόσθεση απαιτούν ίδιο χρονικό διάστημα για την υλοποίησή τους συμπεραίνουμε ότι η αντικατάσταση δυαδικών σειρών από κ-λ άσσων με θετικά και αρνητικά μερικά γινόμενα είναι πολύ πιο προσοδοφόρα όταν το κ-λ είναι μεγαλύτερο του 2.

Η παραπάνω παρατήρηση πραγματοποιήθηκε για πρώτη φορά από τον Αγγλο μαθηματικό Booth, ο οποίος και πρότεινε την αντικατάσταση του πολλαπλασιαστή με μια σειρά ψηφίων των οποίων η τιμή μπορεί να είναι 0, (+1) και (–1). Η νέα κωδικοποιημένη μορφή ονομάζεται Booth recoded μορφή. Ας δούμε πως εφαρμόζεται ο αλγόριθμος του Booth στον πολλαπλασιασμό του A=5 με το B = 55. Αφού το B = 0110111, ο κανονικός πολλαπλασιασμός θα απαιτούσε την παραγωγή 5 μερικών γινομένων και την πρόσθεσή τους με 4 πράξεις πρόσθεσης για την παραγωγή του τελικού αποτελέσματος. Ο Booth αφού διαπίστωσε ότι $B = 55 = 64 - 16 + 8 - 1 = (100000_2) - (0010000_2) + (0001000) - (0000001)$ προτείνει την κωδικοποίηση B = (+1) 0 (-1) (+1) 0 0 (-1). Ο πολλαπλασιασμός της νέας κωδικοποίησης με το A χρειάζεται την παραγωγή μόνο 4 μερικών γινομένων και χρόνο ισοδύναμο με 3 προσθέσεις, συνεπώς είναι τουλάχιστον κατά 25% γρηγορότερος από την πρώτη περίπτωση. Το όφελος αυξάνει δραματικά ανάλογα με το μέγεθος των αλυσίδων από διαδοχικά δυαδικά ψηφία του πολλαπλασιαστή που είναι στο 1.

Η υλοποίηση σε υλικό του αλγορίθμου του Booth απαιτεί την εξέταση των ψηφίων του πολλαπλασιαστή από δεξιά προς τα αριστερά. Κάθε εξεταζόμενο ψηφίο που είναι 1 και το προηγούμενό του 0 κωδικοποιείται σαν (–1), ενώ κάθε ψηφίο που είναι 0 και το προηγούμενό του 1 κωδικοποιείται σαν (+1). Ολα τα υπόλοιπα ψηφία παραμένουν στο 0. Επίσης υπονοείται ένα έξτρα ψηφίο ίσο με 0 στα δεξιά του πολλαπλασιαστή για την αρχικοποίηση του αλγορίθμου.

Η πρώτη αυτή εργασία του Booth όμως έχει και σοβαρά μειονεκτήματα. Η recoded μορφή του $21_{10} = 010101_2$ είναι η (+1)(-1)(+1)(-1)(+1)(-1) που δυστυχώς απαιτεί τη παραγωγή και τη πρόσθεση περισσότερων μερικών γινομένων από ότι η αρχική μορφή. Για να λύσει αυτό το πρόβλημα, σε κατοπινή του εργασία ο Booth πρότεινε την αναπαράσταση του πολλαπλασιαστή με μια σειρά ψηφίων των οποίων η τιμή μπορεί να είναι 0, (+1), (+2), (-1) και (-2). Η τροποποιημένη

αυτή αναπαράσταση προκύπτει από την εξέταση της recoded μορφή του πολλαπλασιαστή και την αντικατάσταση ενός ζεύγους ψηφίων της με ένα μόνο ψηφίο της τελικής κωδικοποίησης. Αφού το αμέσως αριστερότερο ψηφίο έχει τη διπλάσια βαρύτητα, ζεύγη (+1)(-1) και (-1)(+1) της recoded μορφής μπορούν να αντικατασταθούν από (+1) και (-1) στην τελική κωδικοποίηση. Στο παραπάνω παράδειγμα η τελική μορφή του πολλαπλασιαστή θα είναι (+1)(+1)(+1) όπου κάθε ψηφίο έχει βαρύτητα $2^{2κ}$ και που είναι ισοδύναμη από πλευράς καθυστέρησης με την αρχική. Συνεπώς ο τελικός αλγόριθμος του Booth στη χειρότερη περίπτωση γεννά τα ίδια μερικά γινόμενα με τα αρχικά. Τα δυνατά ζεύγη των ψηφίων της recoded μορφής και οι αντικαταστάσεις τους στην τελική μορφή δίνονται στον παρακάτω πίνακα. Παρατηρείστε ότι καθώς κάθε ψηφίο στην recoded μορφή έχει προέλθει από την εξέταση δύο γειτονικών δυαδικών ψηφίων της αρχικής μορφής του πολλαπλασιαστή, μπορούμε να πάρουμε την τελική κωδικοποίηση με κατευθείαν αντικατάσταση τριάδων των αρχικών ψηφίων του πολλαπλασιαστή, όπου κάθε τριάδα έχει ένα κοινό ψηφίο με την επόμενη.

Ζεύγος	Τελική κωδικοποίηση	Αρχικά δυαδικά ψηφία πολλαπλασιαστή
0 0	0	000 ή 111
0 (+1)	+1	001
0 (–1)	-1	110
(+1) 0	+2	011
(+1) (–1)	+1	010
(-1) 0	-2	100
(-1) (+1)	-1	101

Εστω για παράδειγμα ο πολλαπλασιαστής B= 01010111110_2 =702_{10}. Η εφαρμογή της πρώτης κωδικοποίησης θα μας έδινε :

(+1)(-1)(+1)(-1)(+1)0000(-1)0

και η τελική κωδικοποίηση θα ήταν

(+1)(-1)(-1)00(-2)

όπου πλέον κάθε ψηφίο θέσης κ έχει βαρύτητα $2^{2κ}$ ή με άλλα λόγια η παραπάνω κωδικοποίηση είναι η γραφή του 702 σαν 1024-256-64-2.

3.9 Πράξεις σε αριθμούς κινητής υποδιαστολής

Η αριθμητική κινητής υποδιαστολής διαφέρει από αυτήν των αριθμών σταθερής υποδιαστολής στο ότι πέρα από τα σημαντικά μέρη θα πρέπει πλέον να μπορούμε να διαχειριστούμε και τους εκθέτες των αριθμών. Πριν λοιπόν από κάθε πρόσθεση / αφαίρεση, θα πρέπει στην αριθμητική κινητής υποδιαστολής να

εξασφαλίζουμε ότι οι εκθέτες είναι ίσοι. Ας εξετάσουμε πως μπορεί να γίνει αυτό χωρίς να απολέσουμε σημαντικό μέρος της ακρίβειας αναπαράστασης.

Υποθέστε ότι έχουμε να προσθέσουμε τους αριθμούς $0.11100 * 2^5$ και $0.01000 * 2^2$, όπου διατίθενται 6 δυαδικά ψηφία για το σημαντικό μέρος κάθε αριθμού. Ποιον από τους δύο εκθέτες πρέπει να υιοθετήσουμε ως εκθέτη του αποτελέσματος ; Ας υποθέσουμε ότι υιοθετούμε τον μικρότερο. Τότε μετατρέπουμε το $0.11100 * 2^5$ διαδοχικά σε $1.11000 * 2^4$, $1.10000 * 2^3$, $1.00000 * 2^2$. Το αποτέλεσμα της πράξης είναι $1.01000 * 2^2$. Στην αντίθετη περίπτωση, αν δηλαδή υιοθετούσαμε τον μεγαλύτερο ως εκθέτη του αποτελέσματος, τότε θα είχαμε ότι $0.01000 * 2^2 = 0.00100 * 2^3 = 0.00010 * 2^4 = 0.00001 * 2^5$ και το αποτέλεσμα θα ήταν $0.11101 * 2^5$. Είναι προφανές ότι η δεύτερη μέθοδος είναι η ενδεδειγμένη αφού μας δίνει το σωστό αποτέλεσμα, κάτι που ήταν άλλωστε και θεωρητικά αναμενόμενο, αφού το να χαθεί ένα ψηφίο από τον πρώτο αριθμό θα έχει κόστος ακρίβειας της τάξης του 2^5 ενώ κάτι αντίστοιχο από τον δεύτερο θα έχει σημαντικά μικρότερο κόστος ακρίβειας. Παρατηρείστε επίσης ότι μετά τη διαδικασία της πρόσθεσης μπορεί να απαιτηθεί νέα διαδικασία κανονικοποίησης η οποία και αυτή μπορεί να οδηγήσει σε απώλεια ακρίβειας.

Μετά τα παραπάνω είμαστε έτοιμοι να δώσουμε τα βήματα για τις διαδικασίες πρόσθεσης / αφαίρεσης δύο αριθμών κινητής υποδιαστολής.

- Συγκρίνουμε τους εκθέτες των δύο αριθμών ώστε να βρούμε τον αριθμό με τον μεγαλύτερο εκθέτη.
- Ο εκθέτης και το πρόσημο του αποτελέσματος θα είναι ο εκθέτης του μεγαλύτερου εκ των δύο αριθμών.
- Ο μικρότερος των δύο αριθμών ολισθαίνει δεξιά ώστε ο εκθέτης του να είναι ίσος με τον εκθέτη του μεγαλύτερου. Στην ουσία η διαφορά των δύο εκθετών μας δίνει τον αριθμό των θέσεων ολίσθησης.
- Γίνεται η πρόσθεση ή η αφαίρεση των σημαντικών μερών, αφού πρώτα από μορφή πρόσημο-μέτρο μετατραπούν σε μορφή συμπληρώματος ως προς 2.
- Ακολουθεί διαδικασία κανονικοποίησης.

Το πρώτο βήμα της παραπάνω διαδικασίας είναι αυτό που μας εξηγεί το λόγο για τον οποίο επιλέχθηκε η συγκεκριμένη μορφή κωδικοποίησης στο πρότυπο 754 της IEEE. Είναι προφανές ότι η σύγκριση δύο αριθμών μπορεί να ξεκινήσει αμέσως χωρίς να χρειάζεται να ερμηνευτεί η τιμή τους πριν. Η σύγκριση επίσης χρειάζεται να συνεχιστεί μόνο μέχρι να διαπιστωθεί ανισότητα σε ένα δυαδικό ψηφίο των αναπαραστάσεων των δύο αριθμών.

Οι πράξεις του πολλαπλασιασμού και της διαίρεσης στους αριθμούς κινητής υποδιαστολής διεξάγονται με αντίστοιχα βήματα. Αν και στις πράξεις αυτές δεν απαιτείται το βήμα της σύγκρισης των εκθετών, απαιτείται πράξη μεταξύ των εκθετών για τον καθορισμό του εκθέτη του αποτελέσματος. Τα βήματα που ακολουθούνται είναι :

- Το πρόσημο του αποτελέσματος είναι η λογική συνάρτηση αποκλειστικής διάζευξης μεταξύ των προσήμων των δύο εντέλων.
- Ο προσωρινός εκθέτης του αποτελέσματος προκύπτει για τον μεν πολλαπλασιασμό από την πρόσθεση, για τη δε διαίρεση από την αφαίρεση των εκθετών των δύο εντέλων.
- Το προσωρινό σημαντικό μέρος του αποτελέσματος προκύπτει από τον πολλαπλασιασμό ή την διαίρεση των σημαντικών μερών των δύο εντέλων.
- Η διαδικασία κανονικοποίησης θα μας δώσει τον τελικό εκθέτη και το τελικό σημαντικό μέρος του αποτελέσματος.

3.10 Σύνθετες Πράξεις

Οι υπόλοιπες μαθηματικές συναρτήσεις σε ένα υπολογιστικό σύστημα υλοποιούνται με αλγορίθμους που χρησιμοποιούν τις βασικές πράξεις. Προσέξτε ότι κάθε νέα συνάρτηση που φτιάχνουμε μπορεί μετέπειτα να χρησιμοποιηθεί για την υλοποίηση ακόμη πιο πολύπλοκων συναρτήσεων.

Για παράδειγμα βάσει ενός αλγορίθμου που ανήκει στον Ερατοσθένη, η ρίζα ενός αριθμού μπορεί να προσεγγιστεί από τον αριθμό επιτυχών αφαιρέσεων διαδοχικών περιττών αριθμών. Για παράδειγμα για το 25 θα έχω :

$$25 - 1 = 24,\ 24 - 3 = 21,\ 21 - 5 = 16,\ 16 - 7 = 9,\ 9 - 9 = 0$$

και για το 16 θα έχω :

$$16 - 1 = 15,\ 15 - 3 = 12,\ 12 - 5 = 7,\ 7 - 7 = 0$$

δηλαδή 5 επιτυχείς αφαιρέσεις για το 25 και 4 για το 16.

Πολλές από τις πολύπλοκες συναρτήσεις που θέλουμε να κατασκευάσουμε μπορούν να γίνουν πολύ γρήγορα αναδρομικά. Ας υποθέσουμε ότι θέλουμε να φτιάξουμε μια συνάρτηση που να μας υποδεικνύει αν ένας αριθμός είναι πρώτος ή όχι. Για την επίλυση αυτού αρκεί να εξετάσω τα υπόλοιπα της διαίρεσης του αριθμού με κάθε πρώτο αριθμό έως τη ρίζα του αριθμού και συνεπώς εδώ μπορώ να κάνω ένα βήμα αναδρομής. Για παράδειγμα για να βρω αν το 43623 είναι πρώτος αριθμός θα πρέπει να ελέγξω τα υπόλοιπα της διαίρεσής του με τους

πρώτους αριθμούς που φτάνουν μέχρι το 208. Η λοιπόν θα πρέπει να κάνω 208 διαιρέσεις που είναι μάλλον ασύμφορο, ή να εντοπίσω μόνο τους πρώτους αριθμούς μέχρι το 208. Για να εξετάσω αν ένας αριθμός από τους 208 είναι πρώτος μπορώ και πάλι να εφαρμόσω την ίδια λογική κοκ. Προφανώς για να βρω τη ρίζα του αριθμού μπορώ να χρησιμοποιήσω την μεθοδολογία της προηγούμενης παραγράφου.

ΚΕΦΑΛΑΙΟ 4

Βασικές Λειτουργικές Μονάδες

Σκοπός του κεφαλαίου αυτού είναι να κατανοήσουμε τον τρόπο με τον οποίο λειτουργούν οι βασικές λειτουργικές μονάδες του μοντέλου αρτηριών συστήματος, όπως το περιγράψαμε στο πρώτο κεφάλαιο. Θα ξεκινήσουμε μελετώντας πρώτα τη μονάδα μνήμης του συστήματός μας και κατόπιν την κεντρική μονάδα επεξεργασίας. Θα αφήσουμε την περιγραφή του συστήματος εισόδου / εξόδου για επόμενο κεφάλαιο.

4.1 Σύστημα μνήμης

Οπως είδαμε στο κεφάλαιο 2, όλες οι αναπαραστάσεις αριθμών, χαρακτήρων, εικόνας , ήχου κλπ στον υπολογιστή, καταλήγουν να είναι μια σειρά από δυαδικά ψηφία. Το σύστημα μνήμης είναι εκείνο το σύστημα που μπορεί να αποθηκεύει αυτά τα δυαδικά ψηφία και συνεπώς και την πληροφορία που αντιπροσωπεύουν. Για την αποθήκευση ενός δυαδικού ψηφίου υπεύθυνη είναι η κυψελίδα μνήμης (memory cell) η οποία είναι μια στοιχειώδης ποσότητα υλικού που μπορεί να βρίσκεται σε δύο διαφορετικές καταστάσεις. Για παράδειγμα ένας διακόπτης μπορεί να βρεθεί σε δύο καταστάσεις ανοικτός ή κλειστός. Συνεπώς μπορούμε να θεωρήσουμε ότι αυτός μπορεί να παίξει το ρόλο της κυψελίδας μνήμης.

Κατά τη διάρκεια ζωής των υπολογιστών διάφορα ήταν τα υλικά τα οποία χρησιμοποιήθηκαν σαν κυψελίδες μνήμης. Διακόπτες, πυρήνες σιδήρου με δύο φορές μαγνήτισης, φυσαλίδες με ρινίσματα σιδήρου είναι μερικά μόνο από τα είδη κυψελίδων μνήμης που έχουν χρησιμοποιηθεί. Στα σημερινά υπολογιστικά συστήματα χρησιμοποιούνται μόνο κυψελίδες μνήμης οι οποίες φτιάχνονται με ημιαγωγούς (semiconductors), για την κύρια μνήμη.

Χρησιμοποιώντας πολλά αντίγραφα της κυψελίδας μνήμης είναι προφανές ότι φτιάχνονται ποσότητες μνήμης ικανές να αποθηκεύσουν bytes ή πολλαπλάσιά του. Είναι προφανές επίσης ότι όσο μικρότερη είναι μια κυψελίδα μνήμης, τόσο και τα πολλαπλάσιά της που δύνανται να αποθηκεύσουν μια σταθερή ποσότητα πληροφορίας θα κατέχουν και μικρότερο χώρο. Ενας πρώτος σκοπός του συστήματος μνήμης είναι να δίνει στον χρήστη / προγραμματιστή επαρκή χωρητικότητα αποθήκευσης. Δεύτερος στόχος είναι αυτό να επιτυγχάνεται στον ελάχιστο δυνατό χώρο και με το ελάχιστο δυνατό κόστος.

Ας υποθέσουμε λοιπόν ότι αναπαριστούμε τον διαθέσιμο χώρο μνήμης ενός συστήματος σαν ένα δισδιάστατο πίνακα. Η μία διάσταση του πίνακα αντικατοπτρίζει το ποσό πληροφορίας που διαχειρίζεται ένα υπολογιστικό σύστημα ανά χρονικό κύκλο. Για παράδειγμα ένα υπολογιστικό σύστημα που διαθέτει ένα αθροιστή 8 δυαδικών ψηφίων, θα παράγει ένα αποτέλεσμα των 8 δυαδικών ψηφίων και συνεπώς αυτό θα πρέπει να αποθηκεύσει στη μνήμη του. Στη γενική περίπτωση κάθε υπολογιστικό σύστημα μπορεί να διαχειρίζεται ποσότητα πληροφορίας ενός byte ή πολλαπλασίων του (συνήθως 4 ή 8 byte για τα υπολογιστικά συστήματα του 2001). Την ποσότητα αυτή θα την ονομάζουμε ***λέξη (word)***. Η άλλη διάσταση του πίνακά μας θα είναι ο αριθμός των διαθέσιμων διαφορετικών λέξεων που μας δίνει το σύστημα μνήμης μας και που φυσικά καθορίζει και το μέγεθος της μνήμης του συστήματός μας. Για ένα σύστημα μνήμης των 32 δυαδικών ψηφίων και λέξεων του ενός byte ο πίνακας αυτός θα ήταν κάπως έτσι :

Λέξη 1								
Λέξη 2								
Λέξη 3								
Λέξη 4								

Εστω ότι θέλαμε να εκτελέσουμε την πράξη (A+B) * (C-D) όπου A=5, B=3, C=2 και D=1. Η πρόσθεση των A και B θα μας έδινε το ενδιάμεσο αποτέλεσμα 8, το οποίο προφανώς κάπου πρέπει να αποθηκευτεί για να χρησιμοποιηθεί αργότερα. Η αποθήκευση αυτού του αποτελέσματος προφανώς μπορεί να γίνει με τυχαίο τρόπο σε κάποια από τις θέσεις της μνήμης μας. Ας υποθέσουμε λοιπόν ότι τυχαία επιλέγεται η Λέξη 3 και συνεπώς η μνήμη μας αποκτά την εξής κατάσταση.

Λέξη 1								
Λέξη 2								
Λέξη 3	0	0	0	0	1	0	0	0
Λέξη 4								

Δυστυχώς το επόμενο ενδιάμεσο αποτέλεσμα (C – D = 1) μπορεί και αυτό λόγω του τυχαίου τρόπου να επιλέξει την αποθήκευσή του στην ίδια λέξη, με αποτέλεσμα να χαθεί η ήδη αποθηκευμένη εκεί πληροφορία. Συνεπώς, ο τυχαίος τρόπος δε μπορεί να ακολουθηθεί και θα πρέπει να υπάρχει ένας τρόπος διαχωρισμού των θέσεων μνήμης. Ο τρόπος αυτός είναι να δώσουμε σε κάθε λέξη της μνήμης μια ***διεύθυνση (address).*** Στο παράδειγμά μας υπάρχουν 4 διαφορετικές λέξεις και συνεπώς θα έχουμε και 4 διαφορετικές διευθύνσεις. Εστω ότι επιλέγουμε αυτές οι 4 διευθύνσεις να αναπαρασταθούν με δυαδικό τρόπο. Ετσι ο νέος ιδεατός πίνακας που προκύπτει είναι :

Διεύθυνση 00								
Διεύθυνση 01								
Διεύθυνση 10	0	0	0	0	1	0	0	0
Διεύθυνση 11								

Είναι πλέον υποχρέωση του κάθε ένα που έχει κάποια συναλλαγή με το σύστημα μνήμης να ορίζει και τη διεύθυνση της λέξης για αυτή τη συναλλαγή. Οι πιθανές συναλλαγές είναι :

- ***Εγγραφή στη μνήμη (Memory Write)***. Στη περίπτωση αυτή παρέχονται στη μνήμη τόσο η διεύθυνση στην οποία θα γραφτεί η πληροφορία όσο και η ίδια η πληροφορία.
- ***Ανάγνωση από τη μνήμη (Memory Read)***. Στη περίπτωση αυτή παρέχεται στη μνήμη η διεύθυνση της απαιτούμενης πληροφορίας και αυτή απαντά με την πληροφορία που είναι αποθηκευμένη εκεί.

Και οι δύο αυτές συναλλαγές ονομάζονται προσπελάσεις της μνήμης (memory accesses).

Μπορούμε πλέον να ολοκληρώσουμε το παραπάνω παράδειγμά μας. Μόλις ολοκληρωθεί η διαδικασία παραγωγής και του δεύτερου ενδιάμεσου αποτελέσματος, αυτό αποθηκεύεται σε κάποια άλλη θέση μνήμης. Είναι προφανώς ευθύνη αυτού που στέλνει την αίτηση εγγραφής στη μνήμη, να γνωρίζει ότι ήδη κάποια άλλη θέση μνήμης έχει ένα αποτέλεσμα που θα χρειαστεί στο μέλλον. Εστω ότι λοιπόν ότι το δεύτερο ενδιάμεσο αποτέλεσμα αποθηκεύεται στη διεύθυνση 1 και συνεπώς ο πίνακας της μνήμης μας έχει την εξής μορφή :

Διεύθυνση 00								
Διεύθυνση 01	0	0	0	0	0	0	0	1
Διεύθυνση 10	0	0	0	0	1	0	0	0
Διεύθυνση 11								

Πριν την επιτυχή εκτέλεση του πολλαπλασιασμού απαιτείται η ανάκληση των δύο εντέλων, ή με άλλα λόγια διαδικασίες ανάγνωσης των λέξεων στις διευθύνσεις 1 και 2. Προσέξτε ότι η διαδικασία ανάγνωσης δεν καταστρέφει τα δεδομένα που ήδη υπάρχουν στη μνήμη. Τα δεδομένα αυτά θα συνεχίσουν να υπάρχουν και μετά την ανάγνωσή τους. Το αποτέλεσμα του πολλαπλασιασμού, ενδεχόμενα να χρειαστεί να αποθηκευτεί και πάλι στη μνήμη. Αν τα προηγούμενα αποτελέσματα δε χρειάζονται μπορεί να επιλεγεί κάποια από τις ήδη χρησιμοποιηθείσες θέσεις μνήμης. Στην αντίθετη περίπτωση θα πρέπει να επιλεγεί είτε η θέση 0 είτε η θέση 3. Ας υποθέσουμε ότι επιλέγεται η θέση 3, οπότε μετά την εκτέλεση του πιο πάνω υπολογισμού η κατάσταση που θα επικρατεί στη μνήμη μας θα είναι :

Διεύθυνση 00								
Διεύθυνση 01	0	0	0	0	0	0	0	1
Διεύθυνση 10	0	0	0	0	1	0	0	0
Διεύθυνση 11	0	0	0	0	1	0	0	0

Στο παραπάνω παράδειγμα θεωρήσαμε ότι κάθε ποσότητα που αποθηκεύουμε είναι ίση με τη ποσότητα που μας παρέχεται σε κάθε θέση μνήμης. Αυτή είναι μια απλουστευμένη υπόθεση. Εφόσον χρειάζεται να διαχειριζόμαστε πολύ μεγάλους αριθμούς, η ποσότητα που μπορεί να χρειαστεί να αποθηκεύσουμε μπορεί να είναι πολλαπλάσια της χωρητικότητας της κάθε θέσης μνήμης. Για την αποθήκευση τέτοιων ποσοτήτων χρησιμοποιούμε περισσότερες της μιας ***συνεχόμενες*** θέσεις μνήμης. Είναι θεμιτό να μπορούμε να αυτοματοποιούμε την ανάκληση αυτών των ποσοτήτων από τη μνήμη παράγοντας μόνο μία διεύθυνση. Ωστόσο, ανακύπτουν διάφορα προβλήματα :

- Πως σχετίζεται η διεύθυνση αποθήκευσης με την αναπαριστάμενη ποσότητα ; Στο ερώτημα αυτό έχουν διεθνώς ακολουθηθεί δύο προσεγγίσεις. Η

προσέγγιση του ***big-endian*** θεωρεί ότι στη μικρότερη διεύθυνση από τις θέσεις που καταλαμβάνει μια ποσότητα αποθηκεύεται το πιο σημαντικό κομμάτι της. Τα υπόλοιπα λιγότερο σημαντικά κομμάτια αποθηκεύονται σε διαδοχικές υψηλότερες διευθύνσεις. Για παράδειγμα έστω ότι έχω να αποθηκεύσω την 32 bit ποσότητα $F2341088_H$ σε μία μνήμη που κάθε της θέση αποθηκεύει ένα byte. Αν θεωρήσουμε ότι η αποθήκευση ξεκίναγε από τη θέση μνήμης $33FE_H$ τότε σε μια big-endian μηχανή θα είχαμε τον εξής πίνακα αποθήκευσης :

Διεύθυνση	**Δεδομένο**
$33FE_H$	11110010 ($F2_H$)
$33FF_H$	00110100 (34_H)
3400_H	00010000 (10_H)
3401_H	10001000 (88_H)

Οι οπαδοί του little-endian προτείνουν την αποθήκευση των ποσοτήτων έτσι ώστε η μικρότερη διεύθυνση να αποθηκεύει το λιγότερο σημαντικό κομμάτι της ποσότητας και τα διαδοχικώς πιο σημαντικά κομμάτια σε αύξουσες διευθύνσεις. Η παραπάνω ποσότητα συνεπώς θα αποθηκευόταν σε μια little – endian μηχανή στις ίδιες θέσεις μνήμης σύμφωνα με τον εξής πίνακα :

Διεύθυνση	**Δεδομένο**
$33FE_H$	10001000 (88_H)
$33FF_H$	00010000 (10_H)
3400_H	00110100 (34_H)
3401_H	11110010 ($F2_H$)

Στην πράξη δεν υπάρχει σημαντικό πλεονέκτημα της μιας από την άλλη μέθοδο αποθήκευσης

- Επιτρέπεται δεδομένα διαφορετικού μήκους να αποθηκεύονται σε κάθε θέση μνήμης ή επιβάλλεται κάποιος κανόνας για τη παράταξή τους (alignment) ; Σε ένα σύστημα του οποίου κάθε θέση μνήμης αποθηκεύει 8 δυαδικά ψηφία μπορούμε να επιβάλλουμε διάφορους κανόνες στοίχισης των δεδομένων που απαιτούν περισσότερα από 8 δυαδικά ψηφία αποθήκευσης. Για παράδειγμα θα μπορούσαμε να επιβάλλουμε ότι δεδομένα των 16 δυαδικών ψηφίων αποθηκεύονται κατά little-endian πάντα έτσι ώστε το λιγότερο σημαντικό byte να αποθηκευτεί σε θέση μνήμης με άρτια διεύθυνση. Στην πράξη τα συστήματα που επιβάλλουν κανόνες alignment προσφέρουν καλύτερες δυνατότητες αποσφαλμάτωσης.

4.1.1 Κατηγοριοποίηση των μνημών

Οι διαθέσιμες στην αγορά μονάδες μνήμης μπορούν να χωριστούν ανάλογα με το υλικό από το οποίο είναι φτιαγμένες, με τα είδη προσπελάσεων που επιτρέπουν, με το τρόπο που γίνεται η προσπέλαση, με την ταχύτητα προσπέλασης, τον χρόνο προσπέλασης κλπ. Στα παρακάτω γίνεται μια γρήγορη αναφορά στις διάφορες κατηγοριοποιήσεις.

Ανάλογα με το υλικό που είναι φτιαγμένες οι μνήμες διαχωρίζονται σε ημιαγωγικές και μαγνητικές. Κύρια διαφορά αυτών των δύο κατηγοριών, είναι ότι οι ημιαγωγικές μνήμες προσφέρουν τον ίδιο ακριβώς χρόνο για την προσπέλαση οποιασδήποτε λέξης έχει αποθηκευτεί σε αυτές. Για τον λόγο αυτό οι μνήμες αυτές ονομάζονται και μνήμες τυχαίας προσπέλασης. Από την άλλη πλευρά στις μαγνητικές μνήμες ο χρόνος προσπέλασης εξαρτάται από την διεύθυνση της λέξης που προσπελάστηκε στον προηγούμενο κύκλο. Πολλές από αυτές τις μνήμες είναι σειριακές, δηλαδή για να προσπελαστεί η πληροφορία στη λέξη κ θα πρέπει πρώτα να προσπελαστούν όλες οι πληροφορίες στις θέσεις 0 έως κ-1. Κύριο πλεονέκτημα των ημιαγωγικών μνημών είναι η ταχύτητα προσπέλασης (ο χρόνος από τη στιγμή που δίνουμε μια διεύθυνση στη μνήμη μέχρι το χρόνο που αυτή μας έχει διαθέσιμη την πληροφορία αυτής της θέσης) που είναι της τάξης των ns. Κύριο πλεονέκτημα των μαγνητικών μνημών είναι το πολύ χαμηλό κόστος τους, που συνεπάγεται τη δυνατότητα κατασκευής και εμπορικής διάθεσης μνημών πολύ μεγάλης χωρητικότητας.

Οι ημιαγωγικές μνήμες διαχωρίζονται ανάλογα με το αν επιτρέπουν ή όχι προσπελάσεις εγγραφής σε Μνήμες Ανάγνωσης Μόνο (Read Only Memories – ROMs) και σε μνήμες Ανάγνωσης / Εγγραφής (RAMs). Οι μνήμες ανάγνωσης μόνο επίσης μπορούν να χωριστούν σε πολλές υποκατηγορίες :

- PROMs (Programmable ROMs) : ο προγραμματισμός των περιεχομένων της ROM γίνεται μόνο μία φορά στο εργοστάσιο κατασκευής τους.
- EPROMs (Erasable PPOMs) : Τα περιεχόμενά τους μπορούν να σβηστούν με την έκθεση των ολοκληρωμένων σε ακτινοβολία UV και κατόπιν να προγραμματιστούν.
- EEPROMs (Electrically Erasable PPOMs) : Τα περιεχόμενά τους σβήνονται με την εφαρμογή ηλεκτρικών τάσεων πέραν των συνηθισμένων λειτουργίας.
- Flash : Τα περιεχόμενά τους μπορούν να σβηστούν και να προγραμματιστούν ξανά με κανονικά δυναμικά λειτουργίας. Επίσης επιτρέπουν να σβηστούν επιλεγμένες διευθύνσεις μνήμης μόνο.

Οι μνήμες ανάγνωσης μόνο συνήθως προσφέρουν τη δυνατότητα να μην επηρεάζονται τα δεδομένα τους ακόμα και αν το ολοκληρωμένο μνήμης πάψει να τροφοδοτείται με ρεύμα (non volatile memories). Γι' αυτό και χρησιμοποιούνται στα σημερινά υπολογιστικά συστήματα για την αποθήκευση πληροφοριών που χρειάζονται κατά την εκκίνηση του υπολογιστικού συστήματος (bootstrap code). Αντίθετα οι RAMs χάνουν τα δεδομένα τους απουσία τροφοδοσίας (volatile memories). Οι RAMs επίσης διαχωρίζονται περαιτέρω σε στατικές (SRAM) και δυναμικές (DRAM). Ο διαχωρισμός αυτός έγκειται στο αν περιοδικά χρειάζεται μια ανανέωση των περιεχομένων της μνήμης ή όχι. Η κατασκευή των δυναμικών μνημών στηρίζεται σε μικροσκοπικούς πυκνωτές. Η παρουσία φορτίου ή όχι σε αυτούς καθορίζει και το αποθηκευμένο δυαδικό ψηφίο. Λόγω ρευμάτων διαρροής το φορτίο αυτό εξασθενεί με την πάροδο του χρόνου οι δυναμικές μνήμες απαιτούν ανά τακτά χρονικά διαστήματα την ανανέωση αυτού του φορτίου (memory refresh). Οι στατικές μνήμες είναι σήμερα (2001) περίπου 200% πιο γρήγορες από τις δυναμικές, αλλά ταυτόχρονα περίπου το ίδιο ακριβότερες.

Πέρα από τις μνήμες στις οποίες οι προσπελάσεις γίνονται βάσει των διευθύνσεων, υπάρχουν επίσης μνήμες στις οποίες η προσπέλαση γίνεται βάσει των δεδομένων τους. Οι μνήμες αυτές ονομάζονται μνήμες προσπελάσιμες βάσει του περιεχομένου τους (Content Addressable Memories – CAMs). Οι μνήμες αυτές βρίσκουν εφαρμογή μόνο σε πολύ ειδικού σκοπού υπολογιστικά συστήματα.

4.1.2 Ιεραρχία Μνήμης

Οπως είδαμε παραπάνω, στην αγορά διατίθενται μια πληθώρα διαφορετικών διατάξεων μνήμης. Κάθε μία από αυτές προσφέρει διαφορετικά χαρακτηριστικά μεγέθους, χρόνου προσπέλασης και τιμής ανά αποθηκευόμενο δυαδικό ψηφίο. Γιατί άραγε να υπάρχουν όλες αυτές οι διαφορετικές διατάξεις μνήμης ;

Η απάντηση στο ερώτημα αυτό προκύπτει από διάφορες παρατηρήσεις :

- Η προσπέλαση της μνήμης είναι μία από τις πλέον συνήθεις λειτουργίες σε ένα υπολογιστικό σύστημα. Παρατηρείστε στο παράδειγμα του υπολογισμού που δώσαμε πιο πάνω, ότι για 3 απλές πράξεις χρειάστηκαν 5 συνολικά προσπελάσεις της μνήμης. Συνεπώς αν θέλουμε ένα σύστημα που να εκτελεί γρήγορα τους υπολογισμούς μας πρέπει οι προσπελάσεις της μνήμης να γίνονται επίσης γρήγορα.
- Οι υπάρχουσες πιο γρήγορες μνήμες είναι ταυτόχρονα και οι πιο ακριβές.

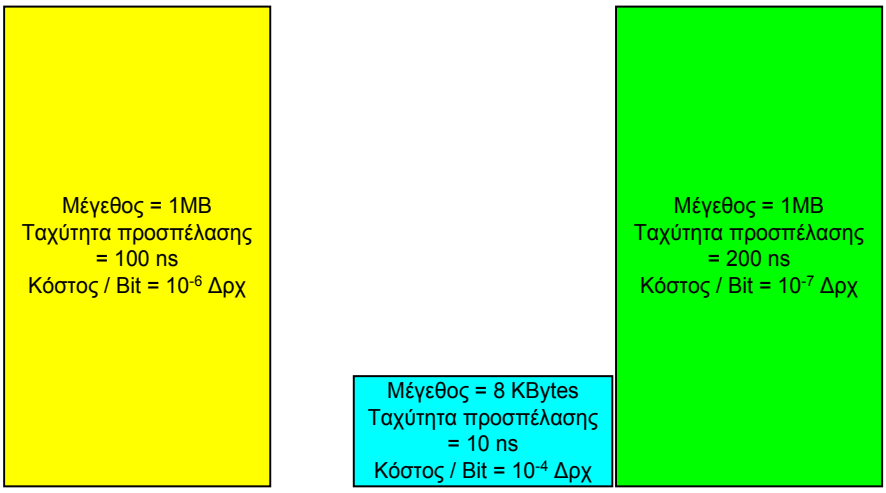
- Οσο μεγαλώνει το μέγεθος μιας μνήμης τόσο πιο αργή γίνεται. (Μια ελλιπής εξήγηση για αυτή τη παρατήρηση είναι ότι μια μεγαλύτερη μνήμη θα έχει περισσότερες γραμμές διευθύνσεων και συνεπώς θα χρειαστεί ένας μεγαλύτερος αποπλέκτης για την επιλογή της ζητούμενης λέξης μνήμης. Πολύ πιο πλήρη εξήγηση θα σας δώσουν τα μαθήματα τεχνολογίας VLSI).

Οι παραπάνω παρατηρήσεις κάνουν προφανές ότι είναι αδύνατο να πετύχω με μία μόνο διάταξη μνήμης, μεγάλο μέγεθος, μεγάλη ταχύτητα και μικρό κόστος ταυτόχρονα.

Το επόμενο ερώτημα που θα πρέπει να απαντήσουμε είναι πως θα πρέπει να εκμεταλλευτώ όλες αυτές τις διαφορετικές τεχνολογίες μνήμης ώστε να φτιάξω ένα σύστημα που να προσεγγίζει τους παραπάνω στόχους. Η απάντηση και σε αυτό το ερώτημα βασίζεται σε δύο παρατηρήσεις που έχουν γίνει για τις προσπελάσεις μνήμης :

- Οι προσπελάσεις μνήμης παρουσιάζουν τοπικότητα στο χρόνο. Με άλλα λόγια αν τη χρονική στιγμή τ προσπελαύνεται η λέξη με διεύθυνση δ είναι πολύ πιθανό τη χρονική στιγμή τ+κ να προσπελαστεί εκ νέου η ίδια διεύθυνση. Το γιατί είναι προφανές από το παράδειγμα υπολογισμού που δώσαμε στην αρχή του κεφαλαίου. Τα ενδιάμεσα αποτελέσματα που αποθηκεύτηκαν στη μνήμη χρειάστηκαν αμέσως μετά για την ολοκλήρωση του υπολογισμού.
- Οι προσπελάσεις μνήμης παρουσιάζουν τοπικότητα στο χώρο. Αν τη χρονική στιγμή τ προσπελαύνεται η λέξη με διεύθυνση δ είναι πολύ πιθανό τη χρονική στιγμή τ+κ να προσπελαστεί η λέξη με διεύθυνση δ ± κ. Η ιδιότητα αυτή ισχύει λόγω της σειριακής εκτέλεσης των προγραμμάτων αλλά και της ανάγκης διαμοίρασης ποσοτήτων μεγαλυτέρων της μιας λέξης σε διαδοχικές λέξεις.

Οι παραπάνω δύο ιδιότητες αποτέλεσαν το έναυσμα για να υλοποιήσουμε το σύστημα μνήμης σαν μια ιεραρχία διατάξεων μνήμης. Για να γίνει πιο κατανοητή η ιεραρχία μνήμης ας εισάγουμε δύο σενάρια για ένα σύστημα μνήμης που φαίνονται στο παρακάτω σχήμα :



Σενάριο Α Σενάριο Β

Στο σενάριο Α η μνήμη μας αποτελείται από μία μόνο διάταξη. Στο δεύτερο σενάριο η μνήμη μας αποτελείται από 2 διατάξεις, μια μικρή γρήγορη μνήμη και μια αργή μεγάλη μνήμη. Το κόστος ανά δυαδικό ψηφίο για κάθε διάταξη καθώς και οι χρόνοι προσπέλασης φαίνονται στο σχήμα. Προσέξτε ότι και οι δύο λύσεις οδηγούν σε περίπου ίδιο κόστος. Η εφαρμογή του δεύτερου σεναρίου επιβάλει ότι κάθε προσπέλαση μνήμης ανακοινώνεται και στις δύο μνήμες. Αν η μικρή μνήμη έχει τη ζητούμενη πληροφορία τότε απαντά σε 10 ns και πλέον προσπέλαση της μεγάλης μνήμης δεν απαιτείται. Στην περίπτωση που η μικρή μνήμη δεν περιέχει τη ζητούμενη πληροφορία, απαντά μόνο η μεγάλη μνήμη μετά από 200 ns. Ταυτόχρονα η μικρή μνήμη αποθηκεύει ένα αντίγραφο της πληροφορίας που της έλειπε αλλά και των γειτονικών της διευθύνσεων για να εκμεταλλευτεί στο άμεσο μέλλον προσπέλαση της ίδιας ή κοντινής διεύθυνσης. Υποθέστε ότι ο χρόνος που απαιτείται για την ενημέρωση της μικρής μνήμης είναι 40 ns και τέλος ότι οι δύο ιδιότητες τοπικότητας των αναφορών ισχύουν στο 80% των προσπελάσεων. Αν υποθέσουμε 1000 προσπελάσεις, τότε το σενάριο Α θα χρειαζόταν συνολικά για αυτές χρόνο 100 ns *1000 = 100 μs. Στο σενάριο Β ο συνολικός χρόνος θα ήταν (10 ns * 800) + (240 ns * 200) = 56 μs. Η χρησιμοποίηση της ιεραρχίας μνήμης συνεπώς μας πρόσφερε αύξηση της απόδοσης του συστήματος μνήμης κατά 44% χωρίς κανένα επιπλέον κόστος. (Προφανώς η χωρητικότητα της μνήμης μας δεν αυξήθηκε, αφού η μικρή μνήμη αποθηκεύει αντίγραφα της πληροφορίας της μεγάλης μνήμης και όχι διαφορετική πληροφορία).

Ο αριθμός των επιπέδων μιας ιεραρχικής μνήμης δεν είναι υποχρεωτικό να είναι μόνο 2. Στα σημερινά υπολογιστικά συστήματα ο αριθμός αυτός είναι σημαντικά μεγαλύτερος. Με άλλα λόγια η μνήμη, παρότι θεωρητικά μας συμφέρει να την σκεφτόμαστε σαν ένα δισδιάστατο πίνακα, στα σημερινά υπολογιστικά

συστήματα είναι διάσπαρτη και υλοποιείται με διαφορετικές διατάξεις που άλλες υπάρχουν για να προσφέρουν ταχύτητα και άλλες ικανοποιητική χωρητικότητα με πολύ μικρό κόστος. Σε ένα τυπικό σημερινό υπολογιστικό σύστημα υπάρχουν συνήθως τα εξής επίπεδα μνήμης :

Ονομα	**Μέγεθος**	**Προσπέλαση σε**	**Υλοποίηση με**
Καταχωρητές	64 – 16384 bits	0.25 – 1 ns	Static RAM
Κρυφή Μνήμη 1ου Επιπέδου (L1 Cache)	1KB – 1MB	2 – 10 ns	Static RAM
Κρυφή Μνήμη 2ου Επιπέδου (L2 Cache)	512 KB – 2 MB	5 – 20 ns	Static RAM
Κύρια Μνήμη	64 MB - 1GB	6 – 40 ns	Dynamic RAM
Μαγνητικοί Δίσκοι	4 – 60 GB	> 10 μs	Μαγνητικό Υλικό
Μονάδα Backup	> 1.2 GB	> 1 ms	Οπτικό / Μαγνητικό Υλικό

Παρατηρείστε στον παραπάνω πίνακα ότι παρότι τόσο οι καταχωρητές όσο και οι κρυφές μνήμες φτιάχνονται από στατική μνήμη, υπάρχουν σαφείς διαφορές στην ταχύτητά τους. Η κρυφή μνήμη 1^{ου} επιπέδου είναι γρηγορότερη από αυτήν του 2^{ου}, γιατί συνήθως βρίσκεται στο ίδιο ολοκληρωμένο με την κεντρική μονάδα επεξεργασίας, όπως και οι καταχωρητές. Ετσι αποφεύγονται χρονοβόρες προσπελάσεις εκτός του ολοκληρωμένου. Προφανώς οι καταχωρητές είναι πολύ ταχύτεροι καθώς είναι μια μνήμη ελάχιστων θέσεων.

Κλείνοντας την αναφορά μας στο σύστημα μνήμης θα πρέπει να σημειώσουμε ότι στα μικρά και γρήγορα επίπεδα της ιεραρχίας μνήμης, όλες οι διαδικασίες υλοποιούνται με υλικό έτσι ώστε να διατηρηθεί η απόδοση σε πολύ υψηλά επίπεδα. Ετσι για παράδειγμα η διαδικασία ενημέρωσης της cache 1^{ου} επιπέδου από αυτήν του 2^{ου} επιπέδου όταν μια πληροφορία που ζητηθεί δεν υπάρχει σε αυτήν του 1^{ου} αλλά βρεθεί σε αυτήν του δευτέρου, είναι μια διαδικασία πλήρως αδιαφανής για κάποιον προγραμματιστή ή για τον χρήστη του συστήματος. Επίσης, κάθε αλλαγή της ιεραρχίας δεν γίνεται αντιληπτή. Ενα σύστημα αν του αφαιρέσουμε την κρυφή μνήμη του 2^{ου} επιπέδου, θα συνεχίσει να μπορεί να εκτελεί τα ίδια με πριν προγράμματα με ελαφρά μειωμένη απόδοση.

4.2 Κεντρική Μονάδα Επεξεργασίας

Ανατρέχοντας στα προηγούμενα κεφάλαια, μπορεί κάποιος να διαπιστώσει ότι η ΚΜΕ είναι το πιο σύνθετο ολοκληρωμένο από όσα υπάρχουν σε ένα υπολογιστικό σύστημα. Μέσα εκεί εμφωλεύονται η Αριθμητική Λογική Μονάδα, η Μονάδα Ελέγχου, οι καταχωρητές και η κρυφή μνήμη $1^{ου}$ επιπέδου τουλάχιστον. Στη συνέχεια θα εξετάσουμε με μεγαλύτερη λεπτομέρεια το σκοπό ύπαρξης κάθε υπομονάδας.

Η Αριθμητική Λογική Μονάδα έχει ως σκοπό την εκτέλεση αριθμητικών και λογικών πράξεων. Σε ότι αφορά στις λογικές πράξεις η Αριθμητική Λογική Μονάδα πρέπει να είναι ικανή να πραγματοποιεί τουλάχιστον το AND, το OR, το XOR δύο λέξεων και να μπορεί επίσης να παράγει το συμπλήρωμα μιας λέξης εισόδου. Επίσης η Αριθμητική Λογική Μονάδα θα πρέπει να περιέχει ένα κύκλωμα ολισθητή για την υλοποίηση λογικών και αριθμητικών ολισθήσεων.

Το κομμάτι των αριθμητικών πράξεων, συνήθως υποδιαιρείται κι αυτό σε δύο ανεξάρτητα κομμάτια σχεδιασμού : τη μονάδα επεξεργασίας ακεραίων και τη μονάδα επεξεργασίας αριθμών κινητής υποδιαστολής. Στα πρώτα υπολογιστικά συστήματα η μονάδα επεξεργασίας δεδομένων κινητής υποδιαστολής υλοποιούνταν σαν ένα ξεχωριστό ολοκληρωμένο κύκλωμα, γνωστό σαν μαθηματικός συνεπεξεργαστής. Ετσι δινόταν η δυνατότητα σε κάποιον να επιλέξει μεταξύ ενός φθηνού συστήματος που δεν συμπεριλάμβανε τον συνεπεξεργαστή και ενός ακριβού και γρηγορότερου συστήματος με τον αριθμητικό συνεπεξεργαστή. Στην πρώτη περίπτωση η μονάδα ακεραίων ήταν αναγκασμένη να εκτελεί ένα ολόκληρο πρόγραμμα για την υλοποίηση κάθε συνάρτησης κινητής υποδιαστολής. Δηλαδή πρέπει να παρατηρήσετε ότι αν υπάρχει κάποια εντολή π.χ. πολλαπλασιασμού σε ένα σύστημα, αυτή δε συνεπάγεται ότι υπάρχει και ένας πολλαπλασιαστής υλοποιημένος με υλικό. Δεν αποκλείεται πάντως και να υπάρχει. Αν δεν υπάρχει όπως είδαμε και στο περασμένο κεφάλαιο, ο πολλαπλασιασμός θα υλοποιηθεί με διαδοχικές ολισθήσεις και προσθέσεις. Προφανώς υπάρχει ένα "παζάρι" μεταξύ κόστους και ταχύτητας. Οσο περισσότερες υπομονάδες υπάρχουν σε υλικό, τόσο πιο γρήγορο είναι ένα σύστημα· ταυτόχρονα όμως και τόσο πιο ακριβό. Στα σημερινά υπολογιστικά συστήματα συνήθως σε υλικό υπάρχουν υλοποιημένες :

- Ολες οι λογικές πράξεις
- Ολες οι ολισθήσεις
- Πρόσθεση / αφαίρεση ακεραίων
- Πολλαπλασιασμός ακεραίων

- Πρόσθεση / αφαίρεση αριθμών κινητής υποδιαστολής
- Πολλαπλασιασμός αριθμών κινητής υποδιαστολής

Σε όλα τα σημερινά υπολογιστικά συστήματα για τους ακεραίους υιοθετείται το συμπλήρωμα ως προς 2 και για τους αριθμούς κινητής υποδιαστολής το πρότυπο της IEEE.

Οι καταχωρητές που υπάρχουν στο ολοκληρωμένο της ΚΜΕ είναι η μικρότερη και ταυτόχρονα η ταχύτερη μνήμη που υπάρχει σε ένα υπολογιστικό σύστημα. Οι καταχωρητές χωρίζονται σε δύο κατηγορίες. Οι γενικού σκοπού καταχωρητές είναι μνήμη για την καταχώρηση ενδιάμεσων αποτελεσμάτων. Συνήθως είναι 8 έως 256 και ο καθένας τους έχει μήκος όσο μια λέξη του συστήματος. Οι ειδικού σκοπού καταχωρητές σχετίζονται περισσότερο με τη μονάδα ελέγχου και ποικίλουν ανάλογα με το κάθε υπολογιστικό σύστημα. Συνήθως σε ένα υπολογιστικό σύστημα υπάρχουν :

- Ο μετρητής προγράμματος (program counter). Η τιμή του καταχωρητή αυτού είναι πάντοτε η διεύθυνση της μνήμης που περιέχει την επόμενη προς εκτέλεση εντολή.
- Ο καταχωρητής καταστάσεως (status register). Κάθε δυαδικό ψηφίο του καταχωρητή αυτού μας δείχνει αν η προηγούμενη πράξη παρήγαγε ή όχι κάποια συγκεκριμένη συνθήκη. Για παράδειγμα η έξοδος της πύλης αποκλειστικής διάζευξης του προσθετή / αφαιρέτη που παράγει το σήμα υπερχείλισης, οδηγείται σαν είσοδος στον καταχωρητή καταστάσεως, έτσι ώστε ένα συγκεκριμένο δυαδικό του ψηφίο να μας δείχνει την υπερχείλιση.
- Ο καταχωρητής διευθύνσεων της μνήμης (Memory Address Register – MAR). Ο καταχωρητής αυτός όταν η ΚΜΕ προσπελαύνει τη μνήμη, περιέχει τη διεύθυνση προσπέλασης.
- Ο καταχωρητής δεδομένων της μνήμης (Memory Data Register – MDR). Ο καταχωρητής αυτός περιέχει τα δεδομένα που η ΚΜΕ σκοπεύει να γράψει στη μνήμη ή τα δεδομένα που διαβάστηκαν από τη μνήμη σε μια προσπέλαση ανάγνωσης.

Δεν θα πρέπει να λησμονούμε ότι οι καταχωρητές είναι κι αυτοί ένα επίπεδο της ιεραρχίας μνήμης. Συνεπώς όπως κάθε άλλο κομμάτι μνήμης και αυτοί είναι προσπελάσιμοι βάσει της διεύθυνσής τους που είναι ξεχωριστή μέσα στο χώρο διευθύνσεων.

Η τελευταία υπομονάδα του ολοκληρωμένου της ΚΜΕ είναι η μονάδα ελέγχου. Η μονάδα αυτή ονομάζεται έτσι γιατί παράγει ένα σύνολο από σήματα που ονομάζονται σήματα ελέγχου τις κατάλληλες χρονικές στιγμές ώστε να

μπορούν οι διάφορες υπομονάδες να επικοινωνούν απρόσκοπτα. Υποθέστε ότι εκτελείται μια πρόσθεση το αποτέλεσμα της οποίας θα πρέπει να αποθηκευτεί στη θέση μνήμης με διεύθυνση 60 (στα παρακάτω η φράση "με διεύθυνση" θα παραλείπεται). Η τιμή 60 υπάρχει στον καταχωρητή με διεύθυνση 32. Για να ολοκληρωθεί αυτή η λειτουργία θα πρέπει το αποτέλεσμα της πρόσθεσης από την αριθμητική λογική μονάδα να μεταφερθεί στον καταχωρητή δεδομένων μνήμης και επίσης η τιμή του καταχωρητή με διεύθυνση 32 θα πρέπει να μεταφερθεί στον καταχωρητή διευθύνσεων της μνήμης. Αυτές οι ***μικρολειτουργίες*** απαιτούν αυστηρό χρονισμό. Μόνο η μονάδα ελέγχου γνωρίζει πότε ολοκληρώθηκε η πρόσθεση και συνεπώς πότε θα πρέπει να ενεργοποιήσει τα σήματα μεταφοράς της πληροφορίας. Επίσης παρατηρείστε ότι η μονάδα ελέγχου είναι η μόνη που γνωρίζει ποια σήματα πρέπει να ενεργοποιηθούν σε κάθε χρονική στιγμή. Θα μπορούσαμε συνεπώς να πούμε ότι η μονάδα ελέγχου είναι υπεύθυνη για την εκτέλεση ακολουθίας μικρολειτουργιών. Από άλλη οπτική γωνία μια ακολουθία μικρολειτουργιών είναι και ένα πρόγραμμα (ισοδύναμα αλγόριθμος) υλοποίησης μιας λειτουργίας. Η μονάδα ελέγχου στα σημερινά υπολογιστικά συστήματα υλοποιείται εξ ολοκλήρου σε υλικό.

4.3 Η εκτέλεση μιας εντολής

Οπως έχουμε αναπτύξει προηγούμενα ο υπολογιστής έχει ένα συγκεκριμένο σύνολο εντολών που μπορεί να εκτελέσει. Αν σε κάθε μία από αυτές τις εντολές βάσει κάποιου κώδικα αντιστοιχίσουμε έναν κωδικό, τότε μπορούμε και αυτές τις εντολές να τις αποθηκεύουμε, να τις ανακαλούμε και γενικά να τις διαχειριζόμαστε όπως κάθε άλλη δυαδική πληροφορία. Ο κωδικός μιας εντολής ονομάζεται κωδικός λειτουργίας (operation code – opcode).

Πολλές φορές η ίδια λειτουργία μπορεί να μεταχειρίζεται αλλιώς τα δεδομένα της ή τα αποτελέσματά της. Για παράδειγμα η λειτουργία της πρόσθεσης μπορεί να διαχωριστεί σε πρόσθεση μιας λέξης, σε πρόσθεση ποσοτήτων μεγαλύτερων της μιας λέξης ή ποσοτήτων μικρότερων. Το αποτέλεσμα μιας πρόσθεσης μπορεί να αποθηκευτεί σε έναν καταχωρητή, στη μνήμη ή να αποτελέσει για παράδειγμα μια διεύθυνση. Είναι προφανές ότι οι παραπάνω διαφοροποιήσεις απαιτούν και διαφορετική ακολουθία μικρολειτουργιών από τη μονάδα ελέγχου για να εκτελεστούν σωστά. Συνεπώς θα πρέπει να υπάρχουν διαφορετικά opcodes για τις παραπάνω διαφοροποιήσεις και θα πρέπει αυτά να διερμηνεύονται διαφορετικά από την μονάδα ελέγχου.

Αν γνωρίζουμε τα opcodes ενός υπολογιστικού συστήματος, τότε είμαστε σε θέση να εκφράσουμε έναν υπολογισμό κατευθείαν σε γλώσσα κατανοητή από τον υπολογιστή. Σαν ένα υπέρ-απλουστευμένο παράδειγμα αν το opcode για την πρόσθεση δύο bytes είναι το 00010101 και θέλω να προσθέσω το 4 με το 18, αρκεί να γράψω :

00010101 00000100 00010010

Η συγγραφή ενός προγράμματος σε μια ακολουθία από 0 και 1 ονομάζεται προγραμματισμός σε γλώσσα μηχανής (machine code programming). Πέρα από την αμεσότητα με τη μηχανή που παρουσιάζει αυτός ο τρόπος προγραμματισμού, έχει πολλά μειονεκτήματα. Ο χρόνος που απαιτείται είναι σημαντικός. Ενα λάθος μπορεί εύκολα να παρεισφρήσει στις πολύ μεγάλες ακολουθίες 0 και 1 που χρειάζονται σε πραγματικά προγράμματα. Στο επόμενο κεφάλαιο θα γνωρίσουμε τη γλώσσα προγραμματισμού Assembly που αντιμετωπίζει σε μεγάλο βαθμό το μειονέκτημα της αξιοπιστίας του προγραμματισμού σε γλώσσα μηχανής.

Εχοντας εισάγει τα παραπάνω μπορούμε πλέον να περιγράψουμε το πως εκτελείται μια εντολή σε ένα υπολογιστικό σύστημα ή με άλλα λόγια τις διαδικασίες που λαμβάνουν χώρα σε έναν κύκλο εντολής. Ο κύκλος μιας εντολής αποτελείται από τη φάση προσκόμισης της εντολής, τη φάση εκτέλεσης της εντολής και τη φάση προετοιμασίας για την επόμενη εντολή.

Η φάση προσκόμισης αποτελείται από τα εξής βήματα (μέσα σε παρένθεση σε κάθε βήμα φαίνονται οι μικρολειτουργίες που συνεπάγεται το κάθε βήμα) :

1. Προσκόμιση του κωδικού λειτουργίας μιας εντολής. (Μεταφορά του περιεχομένου του μετρητή προγράμματος στον MAR, ανάγνωση από τη μνήμη).
2. Αποκωδικοποίηση της εντολής (Μεταφορά του περιεχομένου του MDR στην μονάδα ελέγχου και επιλογή της ακολουθίας μικρολειτουργιών που συνεπάγεται ο συγκεκριμένος κωδικός λειτουργίας).

Η φάση εκτέλεσης της εντολής αποτελείται από τα εξής βήματα :

1. Προσαγωγή των εντέλων της εντολής αν χρειάζεται. (Για κάθε τελούμενο που δε βρίσκεται σε κάποιον από τους καταχωρητές απαιτείται μεταφορά της διεύθυνσής του στον MAR, ανάγνωση από τη μνήμη και εφόσον δεν είναι το τελευταίο τελούμενο μεταφορά του από τον MDR σε κάποιον άλλο καταχωρητή).
2. Εκτέλεση της λειτουργίας της εντολής (Πρόσθεση / αφαίρεση / πολλαπλασιασμός / …)

3. Αποθήκευση του αποτελέσματος, αν απαιτείται (Μεταφορά του αποτελέσματος σε κάποιον καταχωρητή ή στον MDR με ταυτόχρονη μεταφορά της διεύθυνσης αποθήκευσης στον MAR, ακολουθούμενες από διαδικασία εγγραφής στη μνήμη).

Η φάση προετοιμασίας για την επόμενη προς εκτέλεση εντολή συνεπάγεται την ενημέρωση του περιεχομένου του μετρητή προγράμματος με τη διεύθυνση που βρίσκεται το επόμενο opcode. Συνήθως σε ένα υπολογιστικό σύστημα έχουμε ακολουθιακή εκτέλεση, δηλαδή μετά την εντολή στη διεύθυνση Α εκτελούνται οι εντολές στη διεύθυνση $A+\kappa_1$, $A+\kappa_1+\kappa_2$, $A+\kappa_1+\kappa_2+\kappa_3$, όπου κ_1, κ_2 και κ_3 είναι μετατοπίσεις στις διευθύνσεις μνήμης που καθορίζονται από το είδος των εντολών (πιο συγκεκριμένα από τον αριθμό και το είδος των εντέλων που θα απαιτήσει κάθε εντολή). Ωστόσο υπάρχουν περιπτώσεις που η επόμενη προς εκτέλεση εντολή δεν ακολουθεί άμεσα την προηγούμενη εντολή. Σε αυτές τις περιπτώσεις λέμε ότι έγινε κάποιο άλμα (jump) ή μια διακλάδωση (branch) στον εκτελούμενο κώδικα.

ΚΕΦΑΛΑΙΟ 5

Συμβολική Γλώσσα

Οπως είδαμε προηγούμενα η συγγραφή ενός προγράμματος σε γλώσσα μηχανής απαιτεί σημαντικό κόπο και χρόνο καθώς επίσης καθιστά δύσκολη την αποσφαλμάτωση ενός προγράμματος. Η συμβολική γλώσσα (Assembly) επινοήθηκε έτσι ώστε να δώσει λύση στα παραπάνω προβλήματα. Ωστόσο επειδή και αυτή η γλώσσα δίνει στον προγραμματιστή της τον πλήρη έλεγχο του υλικού, είναι μια γλώσσα πολύ κοντά στη μηχανή και συνεπώς πολύ μακριά από τον άνθρωπο. Σήμερα σε Assembly γράφονται κομμάτια κώδικα που είναι κρίσιμα από πλευράς χρόνου εκτέλεσης και μεγέθους.

Το ρεπερτόριο εντολών κάθε συμβολικής γλώσσας εξαρτάται από το ολοκληρωμένο ΚΜΕ και συνεπώς είναι διαφορετικό για διάφορα υπολογιστικά συστήματα. Θα μπορούσαμε λοιπόν να φανταστούμε ότι το ρεπερτόριο εντολών σε συμβολική γλώσσα μας δίνει και ένα επίπεδο αρχιτεκτονικής του συστήματος που συχνά ονομάζεται ***αρχιτεκτονική συνόλου εντολών (Instruction Set Architecture – ISA).*** Δύο υπολογιστικά συστήματα μπορεί με διαφορετικές υλοποιήσεις του υλικού να προσφέρουν το ίδιο σύνολο εντολών και συνεπώς να είναι συμβατά σε επίπεδο αρχιτεκτονικής εντολών. Ανεξάρτητα του ρεπερτορίου εντολών, ο προγραμματιστής σε συμβολική γλώσσα αφού γράψει το πρόγραμμά του βάσει των προσφερόμενων εντολών, θα καλέσει τον Assembler (τον μεταφραστή δηλαδή της συμβολικής γλώσσας), ο οποίος και θα μεταφράσει το πρόγραμμα σε γλώσσα μηχανής. Στη συνέχεια ο assembler θα καλέσει ένα άλλο πρόγραμμα, τον φορτωτή (loader), ο οποίος θα τοποθετήσει τον κώδικα μηχανής

σε κάποιες θέσεις μνήμης. Η διεύθυνση της πρώτης εντολής του κώδικα μηχανής ακολούθως θα φορτωθεί στον μετρητή προγράμματος και θα ξεκινήσει η διαδικασία εκτέλεσης του προγράμματος.

Η ποικιλία ρεπερτορίων που υπάρχει σε συμβολικές γλώσσες καθιστά απαραίτητο στα παρακάτω να επικεντρώσουμε σε μία και μόνο συμβολική γλώσσα, ώστε να την αναπτύξουμε σε βάθος. Αν και η μορφή των εντολών είναι ειδικευμένη, πολλά από τα αναφερόμενα παρακάτω αποτελούν κοινό τόπο για τις περισσότερες συμβολικές γλώσσες. Θα επικεντρώσουμε λοιπόν στη συμβολική γλώσσα του LAB196 του συστήματος δηλαδή στο οποίο θα διεξαχθούν τα εργαστήριά σας. Ας δούμε πρώτα κάποια χαρακτηριστικά αυτού του συστήματος.

Το σύστημα βασίζεται σε μια μονάδα επεξεργασίας που ανήκει στην οικογένεια επεξεργαστών / ελεγκτών MCS®-96 της εταιρείας Intel. Όλα τα μέλη της MCS-96 μοιράζονται κοινό σύνολο εντολών και κοινή αρχιτεκτονική. Η λέξη που χρησιμοποιούν οι επεξεργαστές της οικογένειας αυτής είναι των 16 δυαδικών ψηφίων. Η μνήμη που διατίθεται εντός του ολοκληρωμένου (on-chip RAM) είναι τουλάχιστον 230 bytes. με τη μορφή καταχωρητών γενικού και ειδικού σκοπού.

Ο 80C196KB έχει μια αρτηρία διευθύνσεων που αποτελείται από 16 δυαδικά ψηφία. Συνεπώς μπορεί να προσπελάσει το πολύ 2^{16} διαφορετικές διευθύνσεις μνήμης. Επειδή μπορεί να χειριστεί κατ' ελάχιστο ποσότητες των 8 δυαδικών ψηφίων κάθε θέση μνήμης είναι σκόπιμο να αποθηκεύει ένα byte. Συνεπώς η μέγιστη μνήμη του συστήματός μας αποτελείται από 64 Kbytes. Το μεγαλύτερο μέρος από αυτή τη μνήμη είναι διαθέσιμη στον χρήστη. Ωστόσο αυτός θα πρέπει να γνωρίζει ότι οι διευθύνσεις μνήμης 00H - FFH είναι οι διευθύνσεις των 256 καταχωρητών. Ο 80196 έχει την ιδιότητα να μπορεί να μεταχειρίζεται 2 bytes της μνήμης σαν μια αυτοτελή ποσότητα των 16 bits, δηλαδή σαν μια λέξη, ή δύο λέξεις μαζί σαν μια διπλή λέξη (double-word). Για να το επιτύχει αυτό απαιτεί οι ποσότητες μιας λέξης να είναι aligned σε άρτιες διευθύνσεις και οι ποσότητες μιας διπλής λέξης να είναι aligned σε διευθύνσεις που είναι πολλαπλάσιες του 4. Παρακάτω χρησιμοποιούμε τους εξής συμβολισμούς, για να περιγράψουμε ποσότητες που άλλοτε τις χρησιμοποιούμε σαν 8 και άλλοτε σαν 16 δυαδικών ψηφίων.

AX, BX, και CX είναι καταχωρητές των 16-bits

AL είναι το χαμηλό byte του AX, και AH είναι το υψηλό byte.

BL είναι το χαμηλό byte του BX.

CL είναι το χαμηλό byte του CX.

5.1 Τύποι Δεδομένων

Σε ένα υπολογιστικό σύστημα που βασίζεται στον 80196 υποστηρίζονται αρκετοί τύποι δεδομένων. Μπορεί κανείς να αντιστοιχήσει αυτούς τους τύπους δεδομένων με τους τύπους δεδομένων μιας υψηλής γλώσσας προγραμματισμού. Ο συμβολισμός που ακολουθείται παρακάτω είναι ότι το όνομα του τύπου του τελεστή γράφεται με κεφαλαία. Ένα ***"BYTE"*** είναι μία μη προσημασμένη μεταβλητή των 8 δυαδικών ψηφίων, ενώ ένα ***"byte"*** είναι δεδομένο 8 δυαδικών ψηφίων οποιουδήποτε τύπου. Οι υποστηριζόμενοι τύποι δεδομένων είναι :

- **BYTES:** Τα BYTES είναι μη προσημασμένες μεταβλητές 8 δυαδικών ψηφίων που μπορούν να πάρουν τις τιμές από 0 έως 255. Αριθμητικές ή πράξεις σύγκρισης μπορούν να πραγματοποιηθούν πάνω σε τέτοιου τύπου τελεστές μόνο που το αποτέλεσμα θα ερμηνευθεί σε modulo-255 αριθμητική. Λογικές πράξεις σε τέτοιους τελεστές υλοποιούνται με τις ίδιες λογικές πράξεις πάνω σε καθένα από τα 8 δυαδικά ψηφία του τελεστή. Δεν υπάρχει κάποιος περιορισμός για τη θέση μνήμης στην οποία μπορεί να τοποθετηθεί μία BYTE μεταβλητή, οπότε μπορεί να βρεθεί οπουδήποτε μέσα στο χώρο διευθύνσεων.
- **WORDS:** Οι μεταβλητές τύπου WORD είναι μη προσημασμένες μεταβλητές 16 δυαδικών ψηφίων που μπορούν να πάρουν τις τιμές από το 0 μέχρι το 65535. Αριθμητικές ή πράξεις σύγκρισης μπορούν να πραγματοποιηθούν πάνω σε τέτοιου τύπου τελεστές μόνο που το αποτέλεσμα θα ερμηνευθεί σε modulo-65535 αριθμητική. Λογικές πράξεις σε τέτοιους τελεστές υλοποιούνται με τις ίδιες λογικές πράξεις πάνω σε καθένα από τα 16 δυαδικά ψηφία του τελεστή. Η διεύθυνση μίας μεταβλητής τύπου WORD είναι η διεύθυνση μνήμης του λιγότερου σημαντικού byte.
- **SHORT-INTEGERS:** Οι SHORT-INTEGERS είναι 8-bit προσημασμένες μεταβλητές που μπορούν να πάρουν τις τιμές από το -128 έως το +127. Αριθμητικές πράξεις που παράγουν αποτελέσματα έξω από το πεδίο τιμών του τύπου SHORT-INTEGER θα θέσουν την τιμή 1 στο δυαδικό ψηφίο υπερχείλισης του καταχωρητή κατάστασης.
- **INTEGERS:** Οι INTEGERS είναι προσημασμένες μεταβλητές 16 δυαδικών ψηφίων που μπορούν να πάρουν τις τιμές από το -32768 μέχρι το +32767. Αριθμητικές πράξεις που παράγουν αποτελέσματα έξω από το πεδίο τιμών του τύπου INTEGER θα θέσουν την τιμή 1 στο δυαδικό ψηφίο ένδειξης υπερχείλισης στον καταχωρητή κατάστασης.
- **BITS:** Τα BITS είναι μεταβλητές του ενός δυαδικού ψηφίου που μπορούν να πάρουν την λογική τιμή αληθές ή ψευδές.

- **DOUBLE-WORDS:** Οι DOUBLE-WORDS είναι μη προσημασμένες μεταβλητές των 32 δυαδικών ψηφίων που μπορούν να πάρουν τιμές από το 0 μέχρι το 4294967295. Η αρχιτεκτονική του συστήματος παρέχει απ' ευθείας υποστήριξη γι' αυτόν τον τύπο δεδομένων μόνο για τις ολισθήσεις, καθώς και για τον διαιρετέο σε μία διαίρεση 32 δια 16 στο πλήθος δυαδικά ψηφία και για το γινόμενο σε έναν 16 επί 16 πολλαπλασιασμό. Η διεύθυνση μίας τέτοιας μεταβλητής είναι η διεύθυνση του λιγότερου σημαντικού byte. Πράξεις πάνω σε μεταβλητές τύπου DOUBLE-WORD που δεν υποστηρίζονται απ' ευθείας μπορούν εύκολα να υλοποιηθούν με δύο WORD πράξεις.
- **LONG-INTEGERS:** Οι LONG-INTEGERS είναι προσημασμένες μεταβλητές των 32 δυαδικών ψηφίων που μπορούν να πάρουν τιμές από το -2147483648 μέχρι το +2147483647. Η αρχιτεκτονική του συστήματος παρέχει απ' ευθείας υποστήριξη γι' αυτόν τον τύπο δεδομένων μόνο για τις ολισθήσεις, καθώς και για τον διαιρετέο σε μία διαίρεση 32 δια 16 στο πλήθος δυαδικά ψηφία και για το γινόμενο σε έναν 16 επί 16 πολλαπλασιασμό. Η διεύθυνση μίας τέτοιας μεταβλητής είναι η διεύθυνση του λιγότερου σημαντικού byte. Πράξεις πάνω σε μεταβλητές τύπου LONG-INTEGER που δεν υποστηρίζονται απ' ευθείας μπορούν εύκολα να υλοποιηθούν με δύο INTEGER πράξεις.

Θα μπορούσαμε εύκολα να δούμε αντιστοιχία των παραπάνω τύπων δεδομένων με τους τύπους δεδομένων της γλώσσας C. Για παράδειγμα μια μεταβλητή τύπου char στη C έχει τις ίδιες ανάγκες αποθήκευσης και χειρισμού όπως μια μεταβλητή τύπου BYTE, μια μεταβλητή τύπου short στη C έχει τις ίδιες ανάγκες αποθήκευσης και χειρισμού όπως μια μεταβλητή τύπου SHORT-INTEGER, ενώ τέλος μια μεταβλητή τύπου long στη C έχει τις ίδιες ανάγκες αποθήκευσης και χειρισμού όπως μια μεταβλητή τύπου LONG-INTEGER.

5.2 Ρεπερτόριο Εντολών

Ο κωδικός λειτουργίας κάθε εντολής μιας συμβολικής γλώσσας συνήθως περιγράφεται από μερικά γράμματα που αποτελούν συντόμευση για την εννοούμενη λειτουργία. Για παράδειγμα για την πρόσθεση συνήθως χρησιμοποιείται ο κωδικός ADD και για την αριστερή λογική ολίσθηση ο κωδικός SHL (Shift Left). Προσέξτε ότι σε κάθε υπολογιστικό σύστημα υπάρχει ένα προς ένα αντιστοιχία μεταξύ κωδικών συμβολικής γλώσσας και opcodes γλώσσας μηχανής. Επιπλέον, εφόσον υπάρχει δυνατότητα χειρισμού περισσότερων του ενός τύπου δεδομένων κάθε ένας από τους παραπάνω κωδικούς μεταλλάσσεται σε μια κατηγορία κωδικών. Δηλαδή η πρόσθεση δύο BYTES είναι μια διαφορετική

λειτουργία από την πρόσθεση δύο WORDS και συνεπώς θα έχουν δύο διαφορετικούς κωδικούς (στο σύστημά μας ADDB και ADD).

Στο ρεπερτόριο εντολών της γλώσσας Assembly συνήθως βρίσκουμε εντολές που ανήκουν στις εξής κατηγορίες :

- Αριθμητικές
- Λογικών Συναρτήσεων
- Μεταφοράς Δεδομένων
- Αλλαγής Ροής
- Μετατροπής τύπου Δεδομένων
- Χειρισμού Σωρού
- Αλλες

Στις αριθμητικές εντολές βρίσκουμε (σε παρένθεση υπάρχει ο γενικός κωδικός της λειτουργίας) :

- Πρόσθεση (ADD)
- Αφαίρεση (SUB)
- Πολλαπλασιασμό (MUL)
- Διαίρεση (DIV)
- Καθαρισμό (CLR)
- Σύγκριση (CMP)
- Αύξηση (INC)
- Μείωση (DEC)
- Αλλαγή προσήμου (NEG)

Στις εντολές λογικών συναρτήσεων βρίσκουμε :

- Συμπλήρωμα ως προς 1 (NOT)
- Λογικό ΚΑΙ (AND)
- Λογική Διάζευξη (OR)
- Λογική Αποκλειστική Διάζευξη (XOR)
- Ολίσθηση (SH)

Στις εντολές μεταφοράς δεδομένων βρίσκουμε :

- Φόρτωση (LD)
- Αποθήκευση (ST)
- Μεταφορά ενός κομματιού μνήμης σε κάποιες άλλες διευθύνσεις (BMOV – Block Move)

Η εντολή φόρτωσης χρησιμοποιείται κυρίως για τη δημιουργία ενός αντιγράφου μιας πληροφορίας που υπάρχει στην κύρια μνήμη σε κάποιον καταχωρητή του υπολογιστικού μας συστήματος. Οι πράξεις που θα χρησιμοποιήσουν κατοπινά αυτό το αντίγραφο θα εκτελεστούν έτσι πιο γρήγορα. Η εντολή αποθήκευσης χρησιμοποιείται κυρίως για τη δημιουργία ενός αντιγράφου μιας πληροφορίας που υπάρχει σε κάποιον καταχωρητή του υπολογιστικού μας συστήματος στην κύρια μνήμη. Σε μια σειρά υπολογισμών δηλαδή χρησιμοποιούμε τους καταχωρητές για την αποθήκευση των προσωρινών αποτελεσμάτων και στο τέλος της διαδικασίας, το τελικό αποτέλεσμα μεταφέρεται και στην κύρια μνήμη με μια εντολή αποθήκευσης.

Στις εντολές αλλαγής ροής βρίσκουμε :

- Διακλάδωση (BR – Branch) χωρίς συνθήκη
- Αλμα (Jump) με ή χωρίς συνθήκη
- Κλήση συνάρτησης / ρουτίνας (CALL)
- Επιστροφή από συνάρτηση / υπορουτίνα (RET)

Οπως ήδη έχει αναφερθεί, συνήθως η ροή σε ένα πρόγραμμα είναι ακολουθιακή. Για να αλλάξουμε τη ροή είτε υπό κάποια συνθήκη, είτε χωρίς θα πρέπει να δώσουμε μια καινούρια διεύθυνση στον μετρητή προγράμματος. Στην περίπτωση υπό συνθήκης άλματος / διακλάδωσης / κλήσης, η νέα διεύθυνση θα φορτωθεί στον μετρητή προγράμματος μόνον όταν ικανοποιείται κάποια συνθήκη. Οι συνθήκες που μπορεί να τεθούν έχουν σχέση με τη κατάσταση στην οποία βρίσκονται τα δυαδικά ψηφία του καταχωρητή κατάστασης. Η διαφορά του άλματος από τη κλήση, είναι ότι στο άλμα δε χρειάζεται ποτέ η ροή να γυρίσει προς τα πίσω. Αντίθετα αυτό είναι αναγκαίο στις άλλες δύο περιπτώσεις όταν συναντηθεί η εντολή επιστροφής. Για το λόγο αυτό χρειάζεται πριν την ανανέωση του μετρητή προγράμματος η τιμή που έχει να αποθηκευτεί στον σωρό. Ο σωρός είναι ένας ειδικός μηχανισμός αποθήκευσης που εξηγείται παρακάτω.

Στις εντολές μετατροπής τύπου δεδομένων βρίσκουμε :

- Επέκταση προσήμου SHORT / INTEGER για τη μετατροπή του σε INTEGER / LONG INTEGER αντίστοιχα (EXT)

Στις εντολές χειρισμού σωρού βρίσκουμε :

- Τοποθέτηση στο σωρό (PUSH)
- Ανάκληση από το σωρό (POP)

Ο σωρός (στοίβα) είναι ένας μηχανισμός αποθήκευσης που ακολουθεί τη φιλοσοφία LIFO (Last – In – First - Out). Για να καταλάβουμε αυτή τη φιλοσοφία

μπορούμε να φανταστούμε το σωρό σαν μια στοίβα από πιάτα. Κάθε φορά μπορούμε να προσθέτουμε πιάτα μόνο στη κορυφή της στοίβας και να απομακρύνουμε πιάτα μόνο από τη κορυφή. Η ίδια λειτουργία σε ένα υπολογιστικό σύστημα υλοποιείται με τη χρήση ενός καταχωρητή που ονομάζεται καταχωρητής σωρού (stack pointer). Εστω ότι ο σωρός μας υλοποιείται από τις θέσεις μνήμης $FFFF_H$ έως $FF00_H$. Αρχικά ο σωρός μας είναι άδειος και συνεπώς δείχνει στη θέση $FFFF_H$ (υποθέτουμε δηλαδή ότι ο σωρός μας μεγαλώνει καθώς κινούμαστε σε μικρότερες διευθύνσεις μνήμης). Η λειτουργία PUSH συνεπώς αντιστοιχεί με μια εγγραφή στη θέση μνήμης που δείχνει ο καταχωρητής σωρού και με κατοπινή μείωσή του ώστε να δείχνει στην επόμενη θέση αποθήκευσης. Η λειτουργία POP αντιστοιχεί με μια αύξηση της τιμής του καταχωρητή σωρού ακολουθούμενη από ανάγνωση της θέσης μνήμης που δείχνει ο καταχωρητής σωρού.

Στις υπόλοιπες εντολές βρίσκουμε :

- Καμία λειτουργία (NOP – No operation)
- Αρχικοποίηση του συστήματος (RST – Reset)
- Ενεργοποίηση σημάτων διακοπής (EI – Enable Interrupts)
- Απενεργοποίηση σημάτων διακοπής (DI – Disable Interrupts)
- Κατάσταση εξοικονόμησης ενέργειας (IDLPD)
- Χειρισμός ενδείξεων του καταχωρητή κατάστασης

Τα σήματα διακοπής θα αναλυθούν σε επόμενο κεφάλαιο.

Στη συμβολική γλώσσα μπορούμε πριν το κύριο μέρος του προγράμματος να κάνουμε ορισμένες αντιστοιχίσεις μεταξύ τιμών και συμβολικών ονομάτων έτσι ώστε να μη χρειάζεται να γράφουμε αριθμητικά την κάθε χρησιμοποιούμενη τιμή. Για παράδειγμα η εντολή :

```
# AX EQU 80
```

αντιστοιχεί στο σύμβολο AX τη τιμή 80. Ετσι στις υπόλοιπες γραμμές κώδικα μπορούμε να χρησιμοποιούμε το AX χωρίς να χρειάζεται να θυμόμαστε την πραγματική τιμή του. Ο λεκτικός αναλυτής που είναι το πρώτο κομμάτι του assembler που εκτελείται αναλαμβάνει να κάνει τις απαραίτητες αντικαταστάσεις μεταξύ συμβόλων και τιμών.

Επίσης σε κάθε εντολή της συμβολικής γλώσσας μπορούμε να επισυνάψουμε μια πινακίδα (label). Για παράδειγμα η πινακίδα START επισυνάπτεται στην πρόσθεση δύο λέξεων ως εξής :

```
[START] ADD AX, BX
```

Εντός του κώδικα σε συμβολική γλώσσα πολλές φορές χρειάζεται να αλλάξουμε τη ροή ενός προγράμματος με εντολές άλματος / διακλάδωσης / κλήσης. Την ώρα όμως που γράφουμε το πρόγραμμα, δε γνωρίζουμε τη θέση μνήμης στην οποία θα αποθηκευτεί ο κώδικας για την παραπάνω πρόσθεση. Αυτή θα προσδιοριστεί πολύ αργότερα, κατά τη διαδικασία μετάφρασης του συμβολικού κώδικα σε κώδικα μηχανής. Ωστόσο εμείς μπορούμε να χρησιμοποιήσουμε την πινακίδα η οποία θα αντικατασταθεί αργότερα στη διαδικασία συμβολομετάφρασης.

5.3 Αριθμός Εντέλων Μιας Εντολής

Η γλώσσα Assembly πολλών υπολογιστικών συστημάτων υποστηρίζει ένα και μόνο σταθερό αριθμό εντέλων (ισοδύναμα διευθύνσεων εντέλων) για όλους τους τελεστές που δεν είναι μοναδιαίοι. Αλλα πάλι υπολογιστικά συστήματα περικλείουν στο ρεπερτόριό τους τις ίδιες εντολές αλλά για διαφορετικό αριθμό εντέλων. Σε μερικά από αυτά μάλιστα η θέση κάθε εντέλου θέτει και περιορισμούς ως προς το τι διευθύνσεις μνήμης μπορεί να πάρει. Τέλος σε ορισμένα υπολογιστικά συστήματα υπάρχουν κάποια έντελα τα οποία δε δηλώνονται στη μορφή της εντολής, αλλά υπονοούνται. Ας δούμε μερικά παραδείγματα σε διάφορα τέτοια συστήματα χρησιμοποιώντας το παράδειγμα της πρόσθεσης C=A+B, όπου C, A, B είναι καταχωρητές.

Πρώτα υποθέτουμε ένα σύστημα στο οποίο το αποτέλεσμα κάθε πράξης αποθηκεύεται πάντοτε στον ίδιο καταχωρητή. Συνήθως αυτός ο καταχωρητής ονομάζεται accumulator (συσσωρευτής). Τότε για την επίλυση της πρόσθεσης θα χρειαζόταν ένα πρόγραμμα της μορφής :

```
LD A
ADD B
ST C
```

Η πρώτη εντολή αυτού του προγράμματος μεταφέρει τα δεδομένα που υπάρχουν στον καταχωρητή A στον συσσωρευτή, ο οποίος δε δηλώνεται αλλά υπονοείται. Η δεύτερη εντολή πραγματοποιεί την πρόσθεση του συσσωρευτή με το B και αποθηκεύει το αποτέλεσμα και πάλι στο συσσωρευτή. Με τη τρίτη εντολή το αποτέλεσμα της πράξης αποθηκεύεται στον καταχωρητή C.

Αν υποθέσουμε ότι έχουμε ένα σύστημα που επιτρέπει εντολές δύο εντέλων (ισοδύναμα 2 διευθύνσεων) το αντίστοιχο πρόγραμμα θα ήταν :

```
ADD A, B
ST A, C
```

Στις παραπάνω εντολές η σειρά αναγραφής των εντέλων έχει πολύ μεγάλη σημασία. Αν υποθέσουμε ότι το αποτέλεσμα της πράξης ADD αποθηκεύεται στο πρώτο έντελό της, τότε αν αρχικά τα A και B είχαν τιμές 3 και 5 αντίστοιχα, μετά την εκτέλεση της πρόσθεσης το αρχικό περιεχόμενο του A χάνεται !!! Οι τιμές των καταχωρητών μετά την εκτέλεση της πράξης θα είναι 8 και 5. Αν ήθελα να διατηρήσω την αρχική τιμή για το A και δε με ένοιαζε η αρχική τιμή του B, τότε θα έπρεπε να γράψω τον κώδικά μου σαν ADD B, A !!! Η δεύτερη εντολή απλά αποθηκεύει το αποτέλεσμα του A στο C. Αν σε ένα τέτοιο σύστημα ήθελα να διατηρήσω τις αρχικές τιμές στα A και B, τότε θα έπρεπε να χρησιμοποιήσω και έναν βοηθητικό καταχωρητή έστω D και να διαμορφώσω το πρόγραμμά μου ως εξής :

LD D, A
ADD D, B
ST D, C

Τέλος, σε ένα σύστημα 3 διευθύνσεων θα αρκούσε μόνο να γράψουμε :
ADD C, A, B

Ο πίνακας εντολών κάθε υπολογιστικού συστήματος είναι αυτός που μας καταγράφει όλους τους πιθανούς αριθμούς εντέλων μιας εντολής. Είναι σημαντικό σε έναν υπολογιστή που χρησιμοποιεί περισσότερα του ενός έντελα να προσέχουμε ποια χρησιμοποιούνται για ανάγνωση μόνο (sources) και ποιο χρησιμοποιείται για αποθήκευση (destination). Επίσης είναι σημαντικό να προσέχουμε αν κάθε source μπορεί να έχει όλες τους πιθανούς συνδυασμούς τρόπων διευθυνσιοδότησης. Αξίζει να τονιστεί για άλλη μια φορά ότι ένα σύστημα που προσφέρει εναλλακτικά εντολές 2 και 3 εντέλων (περισσότερα έντελα δε χρησιμοποιούνται στη πράξη) στην ουσία υποκρύπτει το γεγονός ότι αυτές είναι ξεχωριστές εντολές με διαφορετικούς κωδικούς λειτουργίας (opcodes).

5.4 Τρόποι Διευθυνσιοδότησης εντέλων

Οταν γράφουμε ένα πρόγραμμα σε συμβολική γλώσσα είναι αναγκαίο να προσδιορίζουμε τα έντελα κάθε εντολής βάσει των διευθύνσεων στις οποίες αυτά περιέχονται και όχι βάσει της τιμής τους. Ο λόγος είναι ότι σε συνεχείς υπολογισμούς τα έντελα είναι αποτελέσματα προηγούμενων υπολογισμών για τα οποία δε γνωρίζουμε εξ αρχής τις τιμές που θα έχουν.

Μερικές φορές όμως δε γνωρίζουμε ούτε καν τη διεύθυνση του εντέλου που θα λάβει μέρος στην εκτέλεση μιας εντολής, αλλά γνωρίζουμε πως να την

υπολογίσουμε. Θα θέλαμε συνεπώς το υπολογιστικό μας σύστημα να έχει τη δυνατότητα υπολογισμού της τελικής διεύθυνσης. Οι δυνατοί τρόποι υπολογισμού της διεύθυνσης ενός εντέλου που μας παρέχει ένα υπολογιστικό σύστημα ονομάζονται τρόποι διευθυνσιοδότησης (των εντέλων) ή τρόποι αναφοράς στη μνήμη. Να τονιστεί για μία ακόμη φορά ότι κάθε διαφορετικός τρόπος διευθυνσιοδότησης για την ίδια εντολή στην ουσία οδηγεί σε ξεχωριστές εντολές και συνεπώς απαιτεί διαφορετικό κωδικό λειτουργίας. Κάθε υπολογιστικό σύστημα μπορεί να προσφέρει μερικούς ή ακόμα περισσότερους από τους εξής τρόπους διευθυνσιοδότησης (υποθέστε ότι για τα παρακάτω ισχύουν :

```
# AX EQU 80
# BX EQU 82
# CX EQU 84
# AL EQU 80
# BL EQU 82
# CL EQU 84
```

Θεωρείστε ότι οι συναρτήσεις MEM_BYTE (X) και MEM_WORD (X) μας επιστρέφουν αντίστοιχα τις τιμές ενός BYTE και μιας WORD που βρίσκονται αποθηκευμένες στις θέσεις μνήμης X και X, X+1 αντίστοιχα. Υποθέτουμε ότι

```
MEM_BYTE (80) = A2
MEM_BYTE (81) = 03
MEM_BYTE (82) = 42
MEM_BYTE (83) = F2
MEM_BYTE (84) = 08
MEM_BYTE (85) = 0A        ):
```

- **Αμεσες αναφορές με καταχωρητή.**
  Τα έντελα σε αυτή τη περίπτωση είναι οι διευθύνσεις μνήμης καταχωρητών, π.χ.

  ```
  INCB CL
  ```

  Η εντολή αυτή ισοδυναμεί με MEM_BYTE (84) = MEM_BYTE (84) + 1

- **Εμμεσες Αναφορές.**
  Το έντελο στον έμμεσο τρόπο διευθυνσιοδότησης μας καθορίζει τη διεύθυνση του τελικού εντέλου που θα λάβει μέρος στην πράξη. Μία εντολή συνήθως μπορεί να περιέχει μόνο μία έμμεση αναφορά και οι υπόλοιποι τελεστές της εντολής, αν υπάρχουν, πρέπει να προσπελαύνονται με άμεση αναφορά. π.χ.

  ```
  LDB     AL,[AX]
  ```

Η εντολή αυτή ισοδυναμεί με
MEM_BYTE(80) = MEM_BYTE(MEM_WORD(80)) ή
MEM_BYTE(80) = MEM_BYTE(03A2)

- **Έμμεσες αναφορές με αυτόματη αύξηση.**
  Ο τρόπος διευθυνσιοδότησης αυτός είναι ίδιος με τον προηγούμενο με μόνη εξαίρεση ότι η μεταβλητή που διευθυνσιοδοτείται έμμεσα αυξάνεται μετά τη χρήση της. Η αύξηση μπορεί να είναι ανάλογη του τύπου δεδομένων που τελικά προσπελαύνεται. Ετσι, αν η εντολή ενεργεί πάνω σε BYTES ή SHORT-INTEGERS η μεταβλητή της έμμεσης διεύθυνσης θα αυξηθεί κατά ένα, ενώ αν ενεργεί πάνω σε WORDS ή INTEGERS θα αυξηθεί κατά δύο. π.χ.

  LD AX,[BX]+

  Η εντολή αυτή ισοδυναμεί με

  MEM_WORD(80) = MEM_WORD(F242)
  MEM_WORD(82) = MEM_WORD(82) + 2

  Αντίστοιχα η εντολή πρόσθεσης δύο BYTES

  ADDB AL,BL,[CX]+

  ισοδυναμεί με

  MEM_BYTE(80) = MEM_BYTE(82) + MEM_BYTE (0A08)
  MEM_BYTE(84) = MEM_BYTE(84) + 1

- **Άμεσες αναφορές.**
  Ο τελεστής και όχι η διεύθυνσή του δίνονται μαζί με τη εντολή. Μία εντολή μπορεί να περιέχει μόνο μία τέτοια αναφορά, ενώ οι υπόλοιποι τελεστές πρέπει να αναφέρονται χρησιμοποιώντας άμεση διευθυνσιοδότηση με καταχωρητή. π.χ.

  ADD AX,#340

  Η εντολή αυτή ισοδυναμεί με

  MEM_WORD(80) = MEM_WORD(80) + 340_H

- **Δεικτοδοτούμενες αναφορές**
  Σε αυτόν τον τρόπο διευθυνσιοδότησης ένα πεδίο το οποίο υπάρχει στην εντολή προστίθεται στη τιμή ενός εντέλου ώστε να μας δώσει τη διεύθυνση του ζητούμενου εντέλου. π.χ.

  LDB AL,12[BX]

  Η εντολή αυτή ισοδυναμεί με
  MEM_BYTE (80) = MEM_BYTE(12+MEM_WORD(82)), δηλαδή
  MEM_BYTE (80) = MEM_BYTE(F254),

Η αναπαράσταση που χρησιμοποιείται για το πεδίο είναι η δεκαεξαδική γραφή του αριθμού όταν αυτός έχει μορφή συμπληρώματος ως προς 2. Για παράδειγμα έστω ότι θέλουμε να προσπελάσουμε μια θέση μνήμης 61 θέσεις μετά από αυτήν που περιέχεται στον καταχωρητή 90. Το 61_{10} έχει αναπαράσταση 00111101 σε συμπλήρωμα ως προς 2 και συνεπώς το πεδίο που θα χρησιμοποιούσαμε είναι το 3D και η αναφορά στη μνήμη θα είχε τη μορφή 3D[90]. Ανάλογα με το πρόσημο του πεδίου που προστίθεται μπορούμε να έχουμε δεικτοδότηση είτε μεγαλύτερων είτε μικρότερων διευθύνσεων. Ετσι για τη θέση μνήμης που βρίσκεται 37 θέσεις πριν από αυτήν που περιέχεται στον καταχωρητή 90, η σωστή αναφορά θα ήταν DB[90], μιας και το -37_{10} έχει αναπαράσταση σε συμπλήρωμα ως προς 2 : 11011011. Τέλος, ανάλογα με το εύρος του πεδίου που προστίθεται μπορούμε να έχουμε δεικτοδοτημένες αναφορές μικρής ή μεγάλης απόστασης.

- **Διευθυνσιοδότηση βάσει του καταχωρητή σωρού.**
  Αν θεωρήσουμε ότι ο δείκτης σωρού συμβολίζεται με SP τότε μπορούμε με τη χρήση του εύκολα να προσπελάσουμε έντελα που ήδη βρίσκονται στη σωρό. Για παράδειγμα για να προσπελάσουμε τη προτελευταία λέξη του σωρού μπορούμε να γράψουμε :

```
LD      AX,2[SP]
```

- **Διευθυνσιοδότηση με μηδενικό καταχωρητή.**
  Ειδικά για ένα σύστημα της οικογένειας MCS-96 στις διευθύνσεις 0000_H και 0001_H υπάρχει αποθηκευμένη η τιμή 0. Αυτό μας διευκολύνει στην προσπέλαση μιας διεύθυνσης, χρησιμοποιώντας την ως μεταβλητή διευθυνσιοδότησης με δείκτη. π.χ. η εντολή

```
ADD   AX,1234[0]
```

  ισοδυναμεί με MEM_WORD(80) = MEM_WORD(1234).

5.4 Ο Καταχωρητής Κατάστασης

Ο καταχωρητής κατάστασης προγράμματος (PSW) είναι ένας από τους ειδικούς καταχωρητές που υπάρχον σε κάθε υπολογιστικό σύστημα. Κάθε δυαδικό του ψηφίο είναι μια Boolean μεταβλητή που παρέχει πληροφορία σχετικά με την κατάσταση του προγράμματος του χρήστη. Παρακάτω περιγράφουμε με περισσότερη λεπτομέρεια τα δυαδικά αυτά ψηφία, τα οποία και ονομάζονται σύμφωνα με το παρακάτω σχήμα :

7	6	5	4	3	2	1	0
Z	N	V	VT	C	X	I	ST

Z: Το δυαδικό ψηφίο Z *(Zero)* τίθεται στο 1 ώστε να δείχνει πως η λειτουργία που προηγήθηκε παρήγαγε ένα αποτέλεσμα ίσο με το μηδέν.

N: Το δυαδικό ψηφίο N *(Negative)* τίθεται στο 1 ώστε να υποδεικνύει πως η προηγούμενη λειτουργία παρήγαγε ένα αρνητικό αποτέλεσμα. Παρατηρήστε ότι ο ενδείκτης N θα βρίσκεται στην σωστή αλγεβρική τιμή ακόμα και αν παρουσιαστεί μία υπερχείλιση.

V: Το δυαδικό ψηφίο υπερχείλισης V (*oVerflow*) τίθεται στο 1 ώστε να υποδεικνύει πως η προηγούμενη λειτουργία παρήγαγε ένα αποτέλεσμα που είναι έξω από το μέγεθος του τύπου δεδομένων που αναμένουμε.

VT: Το δυαδικό ψηφίο VT *(oVerflow Trap)* τίθεται στο 1 όταν το δυαδικό ψηφίο V τίθεται στο 1, αλλά δεν μηδενίζεται από επόμενες εντολές. Η λογική του δυαδικού ψηφίου VT επιτρέπει τον έλεγχο για μία πιθανή κατάσταση υπερχείλισης στο τέλος μίας ακολουθίας σχετικών αριθμητικών λειτουργιών. Αυτό φυσιολογικά είναι πιο αποτελεσματικό από το να ελέγχουμε το V μετά από κάθε μία εντολή.

C: Το δυαδικό ψηφίο Κρατουμένου C *(Carry)* τίθεται στο 1 ώστε να υποδεικνύει την κατάσταση του αριθμητικού κρατουμένου από το περισσότερο σημαντικό bit της Αριθμητικής Λογικής Μονάδας για μία αριθμητική λειτουργία ή την κατάσταση του τελευταίου bit που ολίσθησε προς τα έξω κατά την ολίσθηση ενός τελεστή. Το Αριθμητικό Δανεικό μετά από μία πράξη αφαίρεσης είναι το συμπλήρωμα του δυαδικού ψηφίου κρατουμένου C (που σημαίνει πως αν η πράξη παρήγαγε ένα δανεικό τότε C=0).

I: Το δυαδικό ψηφίο αυτό επιτρέπει / αποτρέπει αιτήσεις διακοπών να εξυπηρετούνται. Για τις διακοπές θα μιλήσουμε σε επόμενο κεφάλαιο.

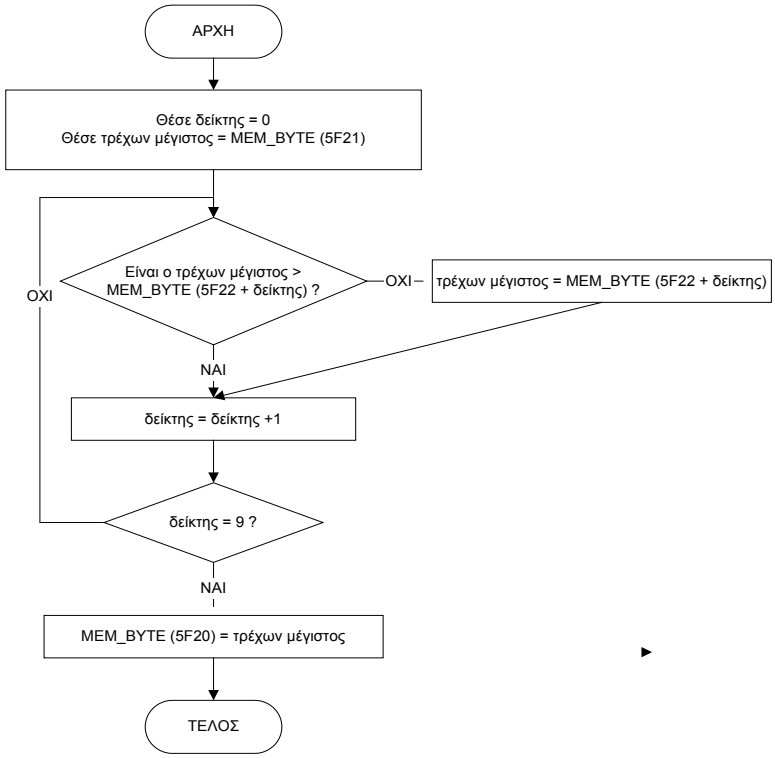
ST: Το δυαδικό ψηφίο ST *(STicky bit)* τίθεται στο 1 ώστε να υποδεικνύει ότι κατά τη διάρκεια μίας δεξιάς ολίσθησης ένας άσσος (1) έχει ολισθήσει πρώτα στον ενδείκτη κρατουμένου C και μετά προς τα έξω. Ο ενδείκτης ST μαζί με τον ενδείκτη κρατουμένου C μπορούν να χρησιμοποιηθούν για να αποφασίσουν για τον τρόπο στρογγύλευσης μετά από μία δεξιά ολίσθηση

Ο PSW μπορεί να διασωθεί στο σωρό του συστήματος με μία απλή πράξη και να ανακληθεί με παρόμοιο τρόπο.

5.5 Ενα παράδειγμα Προγραμματισμού σε Assembly

Στο υποκεφάλαιο αυτό παρουσιάζονται αναλυτικά τα βήματα που χρειάζονται να γίνουν για την επίλυση ενός προβλήματος χρησιμοποιώντας τη συμβολική γλώσσα. Υποθέστε ότι θέλουμε να λύσουμε το πρόβλημα της εύρεσης του μεγαλύτερου μη προσημασμένο byte που βρίσκεται στις θέσεις μνήμης 5F21 - 5F2A και της τοποθέτησής του στην θέση μνήμης 5F20.

Ο πιο ενδεδειγμένος τρόπος για την επίλυση κάθε προβλήματος είναι η κατασκευή ενός διαγράμματος ροής. Για το παραπάνω πρόβλημα ένα διάγραμμα ροής θα ήταν το ακόλουθο :



Το παραπάνω διαγράμματος ροής, χρησιμοποιεί δύο μεταβλητές, τις δείκτης και τρέχων μέγιστος. Ο δείκτης είναι μια μεταβλητή που την χρησιμοποιούμε για να σαρώσουμε διαδοχικά τις θέσεις μνήμης από την 5F22 έως και την 5F2A. Οπως είδαμε προηγούμενα αυτό μπορεί να επιτευχθεί μέσω της δεικτοδοτημένης προσπέλασης. Η μεταβλητή τρέχων μέγιστος ανά πάσα στιγμή έχει αποθηκευμένο το μέγιστο μη προσημασμένο byte που έχουμε συναντήσει. Αρχικοποιείται με τη τιμή που βρίσκεται στη θέση μνήμης 5F21 και συγκρίνεται διαδοχικά με τα περιεχόμενα των θέσεων μνήμης 5F22 έως και 5F2A. Κάθε φορά που συναντάται μεγαλύτερη ποσότητα, τότε αυτή γίνεται ο τρέχων μέγιστος.

Για τη μετάφραση του παραπάνω διαγράμματος ροής σε συμβολική γλώσσα χρειάζεται να ορίσουμε τους καταχωρητές που θα αποθηκεύουν τις δύο μεταβλητές. Εστω ότι χρησιμοποιούμε τον AX για τη μεταβλητή δείκτη και τον BL για τη μεταβλητή τρέχων μέγιστος, όπου AX και BL οι συμβολισμοί θέσεων μνήμης όπως παρουσιάστηκαν νωρίτερα. Το μόνο πλέον που απομένει είναι η συγγραφή των εντολών στη συμβολική γλώσσα. Ο συνολικός κώδικας που θα προέκυπτε θα ήταν :

Ετικέτα	**Εντολή**			**Επεξήγηση**
	# AX	EQU	80	Συμβολισμός της λέξης στις θέσεις μνήμης 81 και 80 με AX
	# AL	EQU	80	Συμβολισμός του byte στη θέση μνήμης 80 με AL
	# BL	EQU	82	Συμβολισμός του byte στη θέση μνήμης 82 με BL
	LD	AX,	#0000	Δείκτης = 0
	LDB	BL,	5F21[00]	Τρέχων μέγιστος = MEM_BYTE(5F21)
[LABEL 1]	CMPB	BL,	5F22[AX]	Τρέχων μέγιστος - MEM_BYTE(5F22 + Δείκτης)
	JH	[LABEL 2]		Αν το αποτέλεσμα της αφαίρεσης είναι θετικό παρέκαμψε την επόμενη εντολή
	LDB	BL,	5F22[AX]	Τρέχων μέγιστος = MEM_BYTE(5F22 + Δείκτης)
[LABEL 2]	INCB	AL		Δείκτης = Δείκτης + 1
	CMPB	AL,	#09	Δείκτης – 9
	JNE	[LABEL 1]		Αν το αποτέλεσμα της αφαίρεσης είναι διάφορο του 0 πήγαινε στην εντολή με πινακίδα [LABEL 1] αλλιώς συνέχισε την εκτέλεση ακολουθιακά
	STB	BL,	5F20[00]	MEM_BYTE (5F20) = μέγιστος

ΚΕΦΑΛΑΙΟ 6

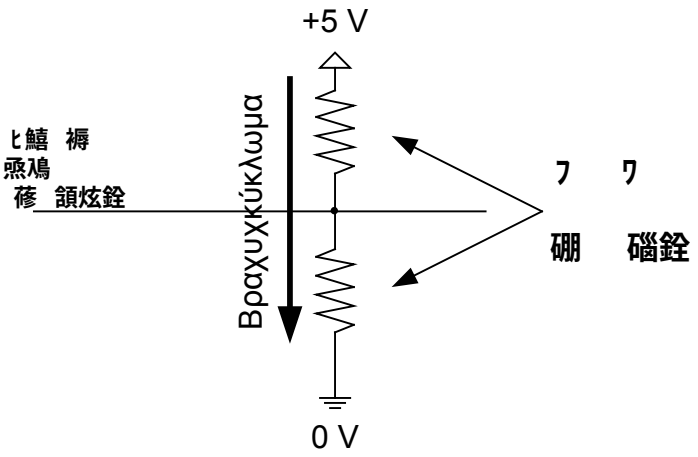
Είσοδος - Εξοδος

Οι μονάδες εισόδου – εξόδου είναι εκείνες οι μονάδες που μετατρέπουν την πληροφορία που διαχειρίζεται ο υπολογιστής σε πληροφορίες κατανοητές από τον άνθρωπο και αντίστροφα. Επειδή οι συσκευές αυτές βρίσκονται και τοπικά πιο κοντά στον άνθρωπο συχνά ονομάζονται και τερματικές ή περιφερειακές συσκευές. Σήμερα υπάρχουν δεκάδες διαφορετικές συσκευές που θα μπορούσαν να χαρακτηριστούν ως περιφερειακές συσκευές. Το πιο σύνηθες γνώρισμά τους είναι η σημαντικά (τάξεις μεγέθους) μικρότερη συχνότητα λειτουργίας τους σε σχέση με τη συχνότητα λειτουργίας της ΚΜΕ. Παρακάτω εξετάζονται οι διάφοροι τρόποι επικοινωνίας αυτών των μονάδων με το υπόλοιπο υπολογιστικό σύστημα και αναλύονται τα χαρακτηριστικά και οι αρχές λειτουργίας των πιο συνηθισμένων από αυτές. Πρώτα όμως αναλύονται οι τρόποι προσαρμογής των διαφόρων συσκευών σ' ένα υπολογιστικό σύστημα.

6.1 Κύριοι (masters) και σκλάβοι (slaves) σε μια αρτηρία

Εστω ότι πάνω σε μια αρτηρία υπάρχουν περισσότερες της μιας συσκευές. Θα ήταν καταστροφικό να επιτρέψουμε σε περισσότερες της μιας συσκευές να οδηγούν ***ταυτόχρονα*** τα σήματα της αρτηρίας, μιας που το μόνο που θα επιτυγχάναμε θα ήταν ένα βραχυκύκλωμα. Ας υποθέσουμε ότι η αρτηρία διευθύνσεων του υπολογιστικού μας συστήματος αποτελείται από 16 δυαδικά ψηφία και μπορεί ταυτόχρονα να οδηγηθεί εκτός από την ΚΜΕ και από κάποια άλλη συσκευή. Υποθέστε ότι η ΚΜΕ εκδίδει εντολή ανάγνωσης από τη διεύθυνση 0000_{HEX} και η άλλη συσκευή εντολή εγγραφής στη διεύθυνση 0001_{HEX}. Ας δούμε τι συμβαίνει πάνω στη γραμμή που αντιστοιχεί στο λιγότερο σημαντικό ψηφίο της αρτηρίας διευθύνσεων. Αφού η ΚΜΕ προσπαθεί να θέσει αυτή τη γραμμή στο λογικό 0, σημαίνει ότι αποκαθιστά ένα αγώγιμο μονοπάτι χαμηλής αντίστασης

μεταξύ αυτής της γραμμής και του δυναμικού του λογικού 0. Αντίθετα η άλλη συσκευή προσπαθεί να θέσει αυτή τη γραμμή στο λογικό 1 και συνεπώς αποκαθιστά ένα αγώγιμο μονοπάτι χαμηλής αντίστασης μεταξύ αυτής της γραμμής και του δυναμικού του λογικού 1. Αν υποθέσουμε ότι το λογικό 0 αντιστοιχεί στα 0V και το λογικό 1 στα 5V, τότε το ηλεκτρικό κύκλωμα που αντιστοιχεί θα είναι :

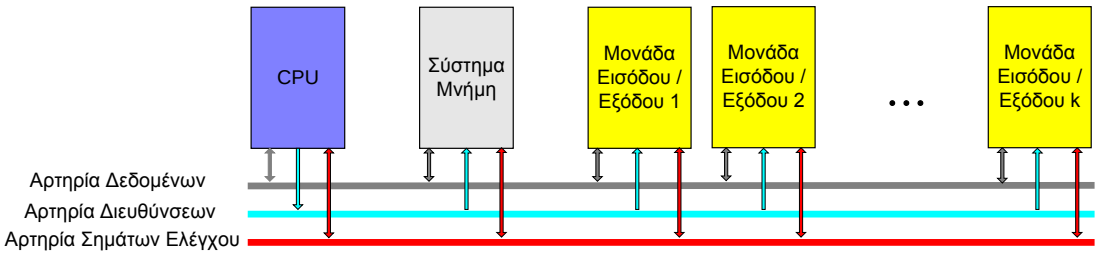


Βάσει των παραπάνω είναι προφανές ότι δε θα πρέπει να επιτρέπουμε ***ταυτόχρονα*** σε περισσότερες της μιας συσκευές να οδηγούν κάποια αρτηρία. Με άλλα λόγια σε κάθε χρονική στιγμή μόνο ένας μπορεί να οδηγήσει την αρτηρία, ενώ όλοι οι υπόλοιποι "ακούν" τις τιμές που υπάρχουν πάνω στην αρτηρία.

Κάθε συσκευή που έχει τη δυνατότητα να οδηγεί μια αρτηρία ονομάζεται κύριος (master) ενώ όλες οι υπόλοιπες συσκευές ονομάζονται σκλάβοι (slaves). Για παράδειγμα, αναλογιζόμενοι την αρτηρία διευθύνσεων, η ΚΜΕ είναι ένας κύριος, ενώ οι μονάδες μνήμης είναι σκλάβοι.

6.2 Ενας κύριος και πολλοί σκλάβοι (Single master – Many slaves)

Μια πολύ απλή λύση στο πρόβλημα της διαχείρισης μιας αρτηρίας είναι να επιτρέψουμε σε ένα σύστημα να έχει μόνο έναν κύριο και όλες οι υπόλοιπες συσκευές να είναι σκλάβοι. Για την επικοινωνία των περιφερειακών συσκευών με το υπόλοιπο υπολογιστικό σύστημα σε αυτή τη περίπτωση προκύπτει το σχήμα :



όπου μόνο η ΚΜΕ μπορεί να οδηγήσει την αρτηρία διευθύνσεων και συνεπώς μόνο αυτή μπορεί να κάνει μεταφορές δεδομένων προς τη μνήμη.

Παρακάτω δείχνουμε ότι το σχήμα αυτό δεν είναι καθόλου αποδοτικό. Υποθέστε ότι κάθε μονάδα εισόδου-εξόδου είναι εξοπλισμένη με μια τοπική μνήμη μεγέθους 256 bytes. Για ένα πληκτρολόγιο, αυτό μεταφράζεται σε γέμισμα της τοπικής μνήμης κάθε φορά που πληκτρολογούνται 256 χαρακτήρες και συνεπώς την ανάγκη αποδέσμευσης αυτής της μνήμης ώστε να συνεχιστεί απρόσκοπτα η διαδικασία πληκτρολόγησης. Τα δεδομένα συνεπώς που υπάρχουν στις τοπικές μνήμες θα πρέπει να μεταφερθούν στην κύρια μνήμη του συστήματός μας.

Ας υποθέσουμε αρχικά ότι οι μονάδες εισόδου-εξόδου δεν διαθέτουν κάποιον τρόπο ώστε να ειδοποιήσουν την ΚΜΕ ότι η τοπική τους μνήμη έχει γεμίσει. Απλά απορρίπτουν καινούργια δεδομένα μέχρι η ΚΜΕ να αδειάσει την τοπική μνήμη τους. Η ΚΜΕ πέραν των άλλων προγραμμάτων, περιοδικά αναλαμβάνει να τρέχει και ένα επιπλέον πρόγραμμα, το οποίο την οδηγεί στο να ελέγξει την κατάσταση των περιφερειακών συσκευών. Ετσι εξετάζει την κατάσταση της κάθε περιφερειακής συσκευής ξεχωριστά με κάποια προκαθορισμένη, βάσει ενός σχήματος προτεραιότητας, σειρά. Σε περίπτωση που διαπιστώσει ότι η περιφερειακή συσκευή απαιτεί εξυπηρέτηση, τότε εκτελεί ένα άλλο πρόγραμμα ειδικευμένο για κάθε συσκευή που περιγράφει το ποια δεδομένα και που θα μεταφερθούν. Το σενάριο αυτό εξυπηρέτησης των περιφερειακών συσκευών είναι γνωστό ως ***προγραμματισμένη διαδικασία εισόδου-εξόδου (programmed I/O).*** Η μη αποδοτικότητα αυτού του σχήματος προκύπτει από τα κάτωθι :

α) Η ΚΜΕ αναλώνει αρκετό χρόνο διερευνώντας ποιά περιφερειακά χρειάζονται εξυπηρέτηση. Για παράδειγμα αν μόνο το τελευταίο στην προτεραιότητα περιφερειακό χρειάζεται εξυπηρέτηση, αυτό θα διαπιστωθεί μετά από κ ελέγχους όπου οι κ-1 έλεγχοι ήταν άσκοποι.

β) Η ΚΜΕ χρειάζεται να εκτελεί τη μεταφορά των δεδομένων, ενώ στον ίδιο χρόνο θα μπορούσε να εκτελεί άλλα προγράμματα που δε σχετίζονται με τα μεταφερόμενα δεδομένα.

γ) Ο μέσος χρόνος αναμονής για εξυπηρέτηση σε μια περιφερειακή συσκευή χαμηλής προτεραιότητας μπορεί να γίνει πολύ μεγάλος.

Το σχήμα αυτό όμως έχει ένα πολύ μεγάλο πλεονέκτημα : την απλότητα των απαιτούμενων περιφερειακών συσκευών.

Σε μια δεύτερη εκδοχή, θυσιάζουμε ελαφρά περισσότερο υλικό ανά περιφερειακή συσκευή με σκοπό να απαλλαγούμε από το μειονέκτημα α) πιο

πάνω. Εφοδιάζουμε δηλαδή κάθε περιφερειακή συσκευή με τη δυνατότητα να παράγει ένα σήμα το οποίο δείχνει αν αυτή απαιτεί εξυπηρέτηση ή όχι. Μέσω της αρτηρίας ελέγχου τα σήματα αυτά κοινοποιούν την κατάσταση της κάθε περιφερειακής συσκευής στην ΚΜΕ και της υποδεικνύουν να διακόψει τη τρέχουσα λειτουργία της για να την εξυπηρετήσει. Τα σήματα αυτά ονομάζονται ***σήματα διακοπής (interrupt signals)***. Ετσι λοιπόν περνάμε στην ***διαδικασία εισόδου με χρήση σημάτων διακοπής (interrupt driven I/O).*** Σε αυτήν την εκδοχή η ΚΜΕ απαλλαγμένη από τον έλεγχο της κατάστασης των περιφερειακών, συνεχίζει κανονικά τους υπολογισμούς της μέχρι να λάβει ένα σήμα διακοπής. Τότε, αφού ολοκληρώσει την τρέχουσα εντολή και αποθηκεύσει το περιβάλλον εργασίας της (τιμές καταχωρητών, σημαίες κατάστασης κοκ.) στην σωρό του συστήματος, μεταπηδά στο πρόγραμμα εξυπηρέτησης της κάθε περιφερειακής συσκευής. Οταν ολοκληρώσει την εξυπηρέτηση της περιφερειακής συσκευής ανακαλεί την αποθηκευμένη της κατάσταση και επαναρχίζει την εκτέλεση από το σημείο που διακόπηκε. Οι περισσότερες ΚΜΕ, δίνουν στον προγραμματιστή τη δυνατότητα, να αποκλείσει τα σήματα διακοπής για κάποιο κομμάτι κώδικα που είναι κρίσιμου χρόνου. Πριν την εκτέλεση αυτού του κώδικα με ειδικές εντολές ο προγραμματιστής αποκλείει τις διακοπές (interrupt masking) και τις επιτρέπει αμέσως μετά το τέλος του κρίσιμου κώδικα.

Πέρα από τα προβλήματα β) και γ) η διαδικασία με χρήση σημάτων διακοπής επιφέρει το πρόβλημα των πόσων σημάτων διακοπής μπορούν να υπάρχουν. Αν και θα θέλαμε να έχουμε τόσα σήματα διακοπής όσες και οι περιφερειακές μας συσκευές, αυτό είναι αδύνατο στην τρέχουσα τεχνολογία. Τα σύγχρονα ολοκληρωμένα ΚΜΕ συνήθως διαθέτουν μόνο έναν ακροδέκτη για σήμα διακοπής. Ενα νέο πρόβλημα συνεπώς που προκύπτει είναι το πως γίνεται η διασύνδεση των πολλών σημάτων αίτησης εξυπηρέτησης στη μία και μοναδική γραμμή διακοπής.

Σε μια πρώτη απλοϊκή προσέγγιση θα μπορούσαμε να χρησιμοποιήσουμε μια πύλη λογικής διάζευξης (OR) η οποία θα συγκέντρωνε τα σήματα αίτησης εξυπηρέτησης σε ένα μόνο σήμα. Η λύση όμως αυτή αδυνατεί να υποδείξει το συγκεκριμένο περιφερειακό που ζητά εξυπηρέτηση. Ετσι λοιπόν το πρόγραμμα εξυπηρέτησης της διακοπής σε αυτή τη περίπτωση θα πρέπει να περιλαμβάνει την εξέταση κάθε περιφερειακού για να διαπιστωθεί αν αυτό είναι που ζήτησε την διακοπή ή όχι. Προσέξτε ότι το σενάριο αυτό είναι εντελώς διαφορετικό από το σενάριο της προγραμματισμένης διαδικασία εισόδου-εξόδου, στην οποία ο έλεγχος γίνεται ακόμη και όταν κανένα περιφερειακό δε θέλει εξυπηρέτηση.

Για να αποφευχθεί η χρονοβόρα αναζήτηση της συσκευής που ζήτησε τη διακοπή είναι προφανές ότι η απλοϊκή λύση δεν αρκεί. Χρειαζόμαστε με άλλα λόγια ένα ολοκληρωμένο κύκλωμα, το οποίο θα δέχεται όλες τις αιτήσεις διακοπής από τις περιφερειακές συσκευές, θα τις ταξινομεί βάσει ενός σχήματος προτεραιότητας και θα ανακοινώνει την ύπαρξη αίτησης εξυπηρέτησης στην ΚΜΕ. Το ολοκληρωμένο αυτό ονομάζεται ***διαχειριστής –ελεγκτής– διακοπών (interrupt controller)***. Σε ένα σύστημα εφοδιασμένο με διαχειριστή διακοπών, η ΚΜΕ δε χρειάζεται να ερευνά όλα τα περιφερειακά, αλλά ρωτά το συγκεκριμένο ολοκληρωμένο για το ποια συσκευή απαιτεί εξυπηρέτηση και αυτό της παρέχει κατευθείαν τη διεύθυνση του περιφερειακού που πρέπει να εξυπηρετηθεί. Η διεύθυνση αυτή μπορεί να χρησιμοποιηθεί σαν δείκτης σε ένα πίνακα που περιέχει τις διευθύνσεις αρχής των προγραμμάτων εξυπηρέτησης κάθε συσκευής ***(interrupt vector table).*** Οι σύγχρονοι διαχειριστές διακοπών δίνουν τη δυνατότητα να ρυθμίζεται η προτεραιότητα κάθε συσκευής. Επιπλέον, οι δυνατότητες επεκτασιμότητας αυτών των ολοκληρωμένων είναι πολύ μεγάλες. Περισσότερα του ενός τέτοια ολοκληρωμένα μπορούν να χρησιμοποιηθούν για την εξυπηρέτηση περισσότερων συσκευών ή για την επέκταση του σχήματος προτεραιότητας.

6.3 Διαιτησία μεταξύ πολλαπλών κυρίων (Multiple master arbitration)

Για την απελευθέρωση της ΚΜΕ από τη διαδικασία μεταφοράς των δεδομένων από και προς τις περιφερειακές συσκευές θα πρέπει να προσθέσουμε στο σύστημα άλλη μια μονάδα η οποία να είναι ικανή να οδηγήσει την αρτηρία διευθύνσεων, ή με άλλα λόγια έναν επιπλέον κύριο. Είδαμε όμως προηγούμενα ότι σε συστήματα με πολλούς κυρίους αναδύεται η ανάγκη συγχρονισμού τους, αφού δε θέλουμε αυτοί ταυτόχρονα να οδηγούν τις διαμοιραζόμενες αρτηρίες. Το πρόβλημα του συγχρονισμού λύνεται με δύο τρόπους : τη διαιτησία (arbitration) και την υποκλοπή κύκλων.

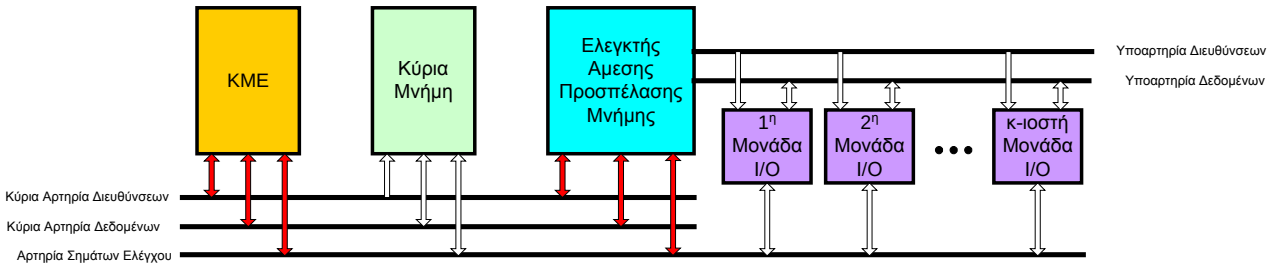
Σε ένα σχήμα με συγχρονισμό βάσει διαιτησίας μία από τις μονάδες κυρίους (συνήθως η ΚΜΕ) ή μία νέα μονάδα, ονομαζόμενη ***διαιτητής αρτηρίας (bus arbiter)*** αναλαμβάνει να συλλέγει τις αιτήσεις των διαφόρων μονάδων που διαμοιράζονται την αρτηρία, για αποκλειστική χρήση της. Βάσει ενός προκαθορισμένου ή ενός προγραμματιζόμενου σχήματος προτεραιότητας, ο διαιτητής αποφασίζει σε ποια μονάδα θα παραχωρηθεί η αρτηρία κατά τον επόμενο κύκλο. Ολες οι υπόλοιπες μονάδες, οφείλουν να σέβονται την απόφαση του διαιτητή και έτσι αμέσως μόλις ολοκληρωθεί ο τρέχων κύκλος, οφείλουν να οδηγήσουν τα κυκλώματα προσαρμογής με τη διαμοιραζόμενη αρτηρία στην

κατάσταση υψηλής εμπέδησης (high impendance), αποτρέποντας έτσι κάθε επηρεασμό του δυναμικού και του ρεύματος που θα επιβάλλει πάνω στην αρτηρία η μονάδα που θα αποκτήσει τον έλεγχο της αρτηρίας. Αυτό το σχήμα συγχρονισμού, συνήθως εφαρμόζεται όταν η ταχύτητα μεταφοράς δεδομένων πάνω από τη διαμοιραζόμενη αρτηρία είναι η ίδια για τις διάφορες συσκευές κυρίους, γιατί μόνο τότε ο αριθμός των κύκλων κατά τους οποίους η ΚΜΕ δε μπορεί να χρησιμοποιήσει τη διαμοιραζόμενη αρτηρία είναι συγκεκριμένος. Κάθε χαμένος κύκλος για τη ΚΜΕ ισοδυναμεί με υποβάθμιση της απόδοσης του συστήματος. Επίσης αυτό το σχήμα εφαρμόζεται σε συστήματα επεξεργασίας δεδομένων πραγματικού χρόνου (real-time systems).

Το σχήμα διαιτησίας με υποκλοπή κύκλων (cycle stealing) εφαρμόζεται στην περίπτωση ύπαρξης δύο κυρίων και βασίζεται στην παρατήρηση ότι μια μονάδα κύριος ακόμη και όταν έχει στην αποκλειστική χρήση της μια διαμοιραζόμενη αρτηρία δεν είναι υποχρεωτικό να την χρησιμοποιεί σε κάθε κύκλο. Ας υποθέσουμε για παράδειγμα την ΚΜΕ, στην περίπτωση που εκτελεί την ακόλουθη εντολή :

ADD R1, R2

όπου R1 και R2 είναι διευθύνσεις καταχωρητών και το αποτέλεσμα της πρόσθεσης αποθηκεύεται στον R1. Για την παραπάνω εντολή είναι προφανές ότι η ΚΜΕ δεν χρειάζεται να χρησιμοποιήσει την αρτηρία διευθύνσεων και δεδομένων, παρά μόνο στην αρχή της εντολής για την προσκόμιση της εντολής. Θυμηθείτε ότι οι καταχωρητές βρίσκονται στο ολοκληρωμένο της ΚΜΕ και συνεπώς δεν απαιτούνται προσπελάσεις της κύριας μνήμης για την προσαγωγή των δύο εντέλων. Συνεπώς, μια δεύτερη μονάδα κύριος μπορεί να χρησιμοποιήσει αυτές τις αρτηρίες την ώρα που η ΚΜΕ προσπελαύνει τους καταχωρητές της, εκτελεί την πρόσθεση και αποθηκεύει τα αποτελέσματά της. Για να είναι εφικτό αυτό το σχήμα θα πρέπει είτε η ΚΜΕ να δηλώνει στη δεύτερη μονάδα την πρόθεσή της να μην χρησιμοποιήσει και συνεπώς να αποδεσμεύσει τις αρτηρίες (bus grant) είτε η δεύτερη μονάδα κύριος να ακολουθεί μια διαδικασία αντίστοιχη της αποκωδικοποίησης της εντολής ώστε να γνωρίζει σε ποιούς υποκύκλους θα ελευθερωθούν οι αρτηρίες ώστε να τις χρησιμοποιήσει. Ας δούμε πως εφαρμόζεται αυτό το σχήμα στην περίπτωση της διασύνδεσης των περιφερειακών συσκευών με το υπόλοιπο υπολογιστικό σύστημα.



Στο παραπάνω σχήμα έχουμε προσθέσει στο υπολογιστικό μας σύστημα έναν ελεγκτή άμεσης προσπέλασης μνήμης (Direct Memory Access –DMA-controller), ένα ολοκληρωμένο το οποίο μπορεί να οδηγήσει τις κύριες αρτηρίες διευθύνσεων και δεδομένων. Για να καταλάβουμε τη λειτουργία αυτού του σχήματος ας υποθέσουμε ότι μια περιφερειακή συσκευή μεταφέρει σε κάθε εξυπηρέτησή της 1Kbyte πληροφορίας προς τη κύρια μνήμη. Υποθέτουμε επίσης την ύπαρξη ενός ελεγκτή σημάτων διακοπής (που δεν φαίνεται στο παραπάνω σχήμα). Η διαδικασία που θα ακολουθηθεί για τη μεταφορά αποτελείται από τα εξής βήματα :

1) Μέσω ενός σήματος διακοπής το περιφερειακό ειδοποιεί τον ελεγκτή διακοπών για το ότι χρειάζεται εξυπηρέτηση.
2) Ο ελεγκτής διακοπών ανακοινώνει την αίτηση διακοπής στην ΚΜΕ.
3) Η ΚΜΕ ολοκληρώνει τη τρέχουσα εντολή, αποθηκεύει τη κατάστασή της και ερωτά τον ελεγκτή διακοπών σχετικά με το ποια συσκευή ζητά εξυπηρέτηση.
4) Οταν ο ελεγκτής διακοπών την ενημερώσει, μεταπηδά στο πρόγραμμα εξυπηρέτησης της συγκεκριμένης συσκευής (πρόγραμμα οδηγός – device driver).
5) Το πρόγραμμα εξυπηρέτησης ενημερώνει την ΚΜΕ σχετικά με το από που θα διαβάσει τις πληροφορίες και που θα μεταφερθούν οι πληροφορίες. Η ΚΜΕ προγραμματίζει τον ελεγκτή άμεσης προσπέλασης για να κάνει τη μεταφορά (στην ουσία του ανακοινώνει τη διεύθυνση αρχής ανάγνωσης και εγγραφής και το μέγεθος μεταφοράς).
6) Η ΚΜΕ ανακαλεί την αποθηκευμένη της κατάσταση και συνεχίζει να εκτελεί κανονικά τους υπολογισμούς της.
7) Ο ελεγκτής άμεσης προσπέλασης μνήμης διαβάζει τις πληροφορίες από την περιφερειακή συσκευή χρησιμοποιώντας διαφορετικές υποαρτηρίες και τις αποθηκεύει σε δική του τοπική μνήμη, ενώ παράλληλα σε κύκλους που η ΚΜΕ δε χρησιμοποιεί τις κύριες αρτηρίες μεταφέρει τις πληροφορίες στην κύρια μνήμη.
8) Στο τέλος της μεταφοράς ο ελεγκτής άμεσης προσπέλασης μνήμης μέσω μιας αίτησης διακοπής, ανακοινώνει στην ΚΜΕ την ολοκλήρωση της μεταφοράς, η

οποία μέσω του ελεγκτή διακοπών ειδοποιεί την περιφερειακή συσκευή να αποσύρει την αίτηση διακοπής.

Τα παραπάνω βήματα είναι μια απλουστευμένη περιγραφή της όλης διαδικασίας, αφού αποκρύπτουν σημαντικά προβλήματα που μπορούν να προκύψουν, όπως για παράδειγμα τι γίνεται αν την ώρα που γίνεται μια μεταφορά έρθει αίτηση για μεταφορά υψηλότερης προτεραιότητας ή όταν μια μεταφορά δε μπορεί να ολοκληρωθεί ή η ολοκλήρωσή της απαιτεί πολύ μεγάλο χρόνο. Τα προβλήματα αυτά θα αναλυθούν διεξοδικά στα μαθήματα Αρχιτεκτονικής Υπολογιστών, Διασύνδεσης Μικροϋπολογιστικών Συστημάτων και Προχωρημένων Θεμάτων Αρχιτεκτονικής.

6.4 Σειριακή και παράλληλη ανταλλαγή δεδομένων

Μια βασική κατηγοριοποίηση των διάφορων αρτηριών που συνήθως διατίθενται σε ένα υπολογιστικό σύστημα είναι σε αρτηρίες που ανά χρονικό κύκλο μπορούν να μεταφέρουν μόνο ένα δυαδικό ψηφίο και σε αρτηρίες που σε κάθε χρονικό κύκλο μπορούν να μεταφέρουν μεγαλύτερα ποσά πληροφορίας. Οι αρτηρίες αυτές ονομάζονται σειριακές (serial) και παράλληλες (parallel) αντίστοιχα. Οι ονομασίες αυτές προκύπτουν από το ότι στη μεν πρώτη περίπτωση ένα ποσό πληροφορίας μεγαλύτερο από ένα δυαδικό ψηφίο θα πρέπει να μεταφερθεί σε διαδοχικούς κύκλους, ενώ στη δεύτερη περίπτωση χρησιμοποιούνται ταυτόχρονα όλοι οι διαθέσιμοι πόροι της αρτηρίας. Πολλές φορές οι καταλήξεις αυτών των αρτηριών σε κάποιον τυποποιημένο ακροδέκτη ονομάζονται και ***θύρες (ports)*** του υπολογιστικού συστήματος. Ετσι η φράση "σειριακό ποντίκι" δε σημαίνει τίποτα περισσότερο από μια περιφερειακή συσκευή που συνδέεται στον ακροδέκτη μιας σειριακής αρτηρίας.

Συγκρίνοντας τα δύο είδη αρτηριών μεταξύ τους κάποιος θα μπορούσε να σημειώσει ότι :

- Οι ρυθμοί μεταφοράς δεδομένων πάνω από τις παράλληλες αρτηρίες είναι σημαντικά μεγαλύτεροι από αυτές των σειριακών, όταν αυτές διαθέτουν τον ίδιο χρονικό κύκλο. Για παράδειγμα μια σειριακή αρτηρία με χρονικό κύκλο 30 ns (ισοδύναμα, μια αρτηρία που χρονίζεται με ρολόι 33,33... MHz) στη σειριακή περίπτωση προσφέρει μέγιστο ρυθμό μεταφοράς δεδομένων 33,33 Mbps (Megabits / sec), ενώ μια αντίστοιχη παράλληλη με 8 δυαδικά ψηφία θα μας έδινε μέγιστο ρυθμό μεταφοράς δεδομένων 266,64 Mbps. Γίνεται συνεπώς προφανές ότι οι παράλληλες αρτηρίες χρησιμοποιούνται για τα πιο γρήγορα περιφερειακά ενώ οι σειριακές για τα πιο αργά. (Προσέξτε ότι στα παραπάνω υποθέσαμε ότι οι αρτηρίες χρονίζονται με αντίστοιχα ρολόγια. Ωστόσο, αυτό

σπάνια ισχύει, μιας και είναι πολύ ευκολότερο να χρονίσουμε μια σειριακή αρτηρία με πολύ υψηλότερης συχνότητας ρολόι από ότι μια παράλληλη αρτηρία· σήμερα –2002- σειριακοί ρυθμοί μεταφοράς δεδομένων της τάξης των 100 Gbps είναι εμπορικά διαθέσιμοι).

- Το κόστος μιας παράλληλης αρτηρίας είναι πολλαπλάσιο αυτού μιας σειριακής. Αυτό οφείλεται πέρα από την ανάγκη ύπαρξης περισσότερων αγωγών στο ότι απαιτούνται περισσότεροι ακροδέκτες σε κάθε ολοκληρωμένο που χρησιμοποιεί μια παράλληλη αρτηρία, περισσότερα κυκλώματα οδήγησης κλπ. Σημαντικά μεγαλύτερο κόστος επίσης απαιτείται για τη διασφάλιση ποιότητας πάνω σε μια παράλληλη αρτηρία αφού η πιθανότητα λαθών είναι πολλαπλάσια.

Μια αρτηρία μπορεί επίσης να κατηγοριοποιηθεί ανάλογα με τις κατευθύνσεις επικοινωνίας που επιτρέπει. Υποθέτοντας την επικοινωνία μεταξύ των συσκευών Α και Β πάνω από μια αρτηρία, υπάρχουν οι εξής πιθανότητες :

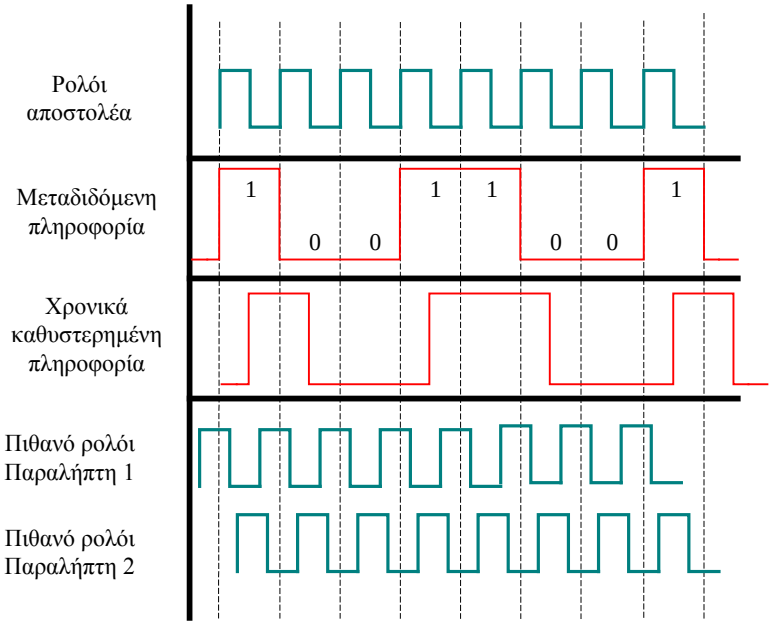
α) Η Α μπορεί να στείλει δεδομένα στην Β αλλά η Β δεν μπορεί να στείλει δεδομένα στην Α (ή το αντίστροφο). Σε αυτή την περίπτωση μιλάμε για μονόδρομη (simplex) αρτηρία.

β) Η Α μπορεί να στείλει δεδομένα στην Β αλλά ***ταυτόχρονα*** η Β δεν μπορεί να στείλει δεδομένα στην Α (ή το αντίστροφο). Σε αυτή την περίπτωση μιλάμε για ημι-αμφίδρομη (half-duplex) αρτηρία.

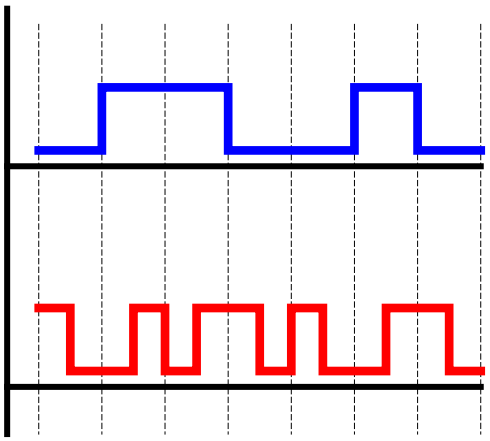
γ) Η Α μπορεί να στείλει δεδομένα στην Β και ***ταυτόχρονα*** η Β μπορεί να στείλει δεδομένα στην Α. Σε αυτή την περίπτωση μιλάμε για αμφίδρομη (full-duplex) αρτηρία.

Ενα μεγάλο πρόβλημα αποτελεί ο συγχρονισμός των συσκευών που χρησιμοποιούν την αρτηρία. Ο συγχρονισμός είναι απολύτως αναγκαίος για να αποφευχθούν συγκρούσεις πάνω στην αρτηρία, αλλά και για την αξιόπιστη αναπαραγωγή των δεδομένων από τον παραλήπτη μιας πληροφορίας. Για να γίνει πιο κατανοητό το πρόβλημα ας υποθέσουμε ότι ένας αποστολέας θέλει να αποστείλει πάνω από μια σειριακή αρτηρία την πληροφορία 10011001. Εστω ότι τοποθετεί κάθε ψηφίο της πληροφορίας βάσει της θετικής ακμής ενός τοπικού ρολογιού. Τότε θα είχαμε ένα σχήμα όπως το ακόλουθο που μας δείχνει την μεταδιδόμενη πληροφορία σύμφωνα με το ρολόι που χρησιμοποίησε ο αποστολέας. Ωστόσο η πληροφορία αυτή φτάνει ελαφρά καθυστερημένη στους διάφορους παραλήπτες οι οποίοι χρησιμοποιούν μεν ρολόι ίδιας συχνότητας αλλά ενδεχόμενα με διαφορά φάσης (phase shift) σε σχέση με το ρολόι του παραλήπτη. Ανάλογα λοιπόν με τη καθυστέρηση της πληροφορίας και τη διαφορά φάσης στα

ρολόγια των παραληπτών αυτοί μπορεί να αποκωδικοποιήσουν σωστά (π.χ. παραλήπτης 2) ή λάθος (παραλήπτης 1) τη μεταδιδόμενη πληροφορία.



Μια πρώτη προφανής λύση στο παραπάνω πρόβλημα είναι ο αποστολέας της πληροφορίας πέρα από αυτήν να μεταδίδει και το ρολόι το οποίο χρησιμοποίησε για την αποστολή της. Φυσικά ελπίζουμε ότι τα 2 σήματα θα υποστούν την ίδια καθυστέρηση μέχρι να φτάσουν στους παραλήπτες τους. Ωστόσο η λύση αυτή αυξάνει τον απαιτούμενο αριθμό αγωγών και αντενδείκνυται σε υψηλόσυχνα σήματα ρολογιού. Εναλλακτική λύση είναι η κωδικοποίηση κατά Manchester. Με την κωδικοποίηση αυτή στην ουσία εμφωλεύουμε το ρολόι εντός των δεδομένων και είναι ευθύνη του παραλήπτη να το αναπαράγει. Η κωδικοποίηση αυτή στο κέντρο του χρόνου μετάδοσης ενός δυαδικού ψηφίου έχει μια ανοδική ακμή αν το κωδικοποιούμενο ψηφίο είναι 1 και μια καθοδική ακμή για το 0. Το παρακάτω σχήμα δείχνει την κωδικοποίηση του 0110010



Οπωσδήποτε και αν αλλοιωθεί χρονικά η πληροφορία είναι σίγουρο ότι ο παραλήπτης της αμέσως μετά από τη λήψη δύο ίδιων δυαδικών ψηφίων μπορεί να ανασυνθέσει με μεγάλη ακρίβεια το ρολόι του παραλήπτη.

Ενα άλλο ζήτημα τέλος που απαιτεί συζήτηση είναι το πως αναγνωρίζει ο παραλήπτης την αρχή αποστολής δεδομένων. Στις σειριακές αρτηρίες αυτό επιτυγχάνεται με το να έχουμε την αρτηρία κάθε στιγμή στην οποία δε χρησιμοποιείται σε κάποιο προσυμφωνημένο δυναμικό. Η αλλαγή από αυτό το δυναμικό υποδεικνύει την αρχή μετάδοσης. Στις παράλληλες αρτηρίες αντίστοιχα υπάρχουν ορισμένοι συνδυασμοί κωδικών (πρόδρομοι – preambles) που υποδεικνύουν την αρχή μιας μετάδοσης. Βάσει των παραπάνω στις επόμενες παραγράφους παρουσιάζονται οι βασικές αρχές λειτουργίας μερικών περιφερειακών συσκευών.

6.5 Βοηθητική Μνήμη (Secondary Storage / Mass Storage)

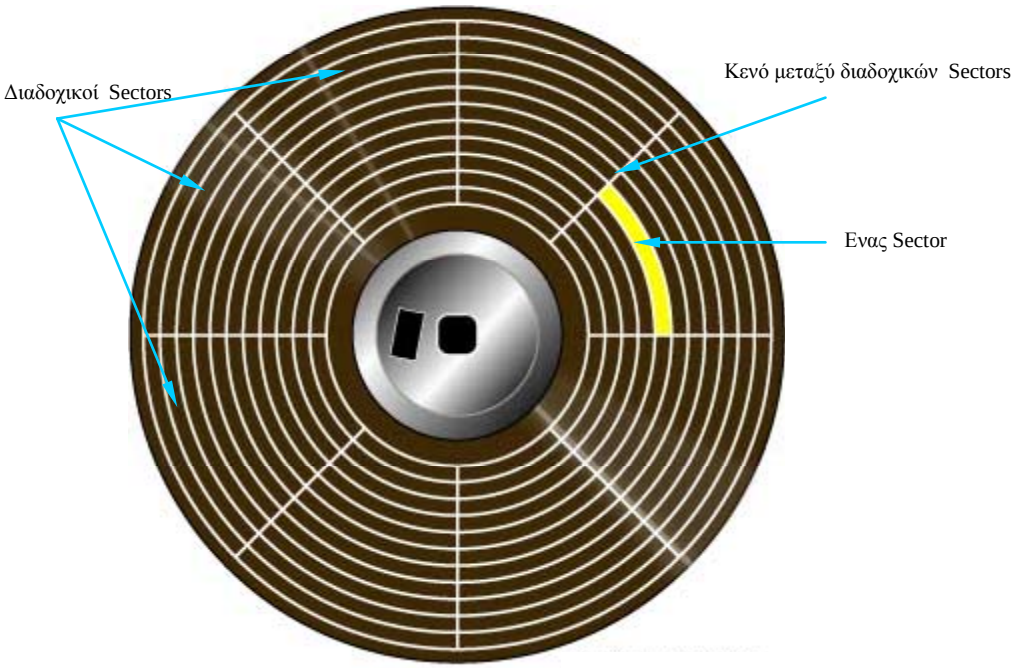
Το τελευταίο επίπεδο της ιεραρχίας μνήμης αποτελείται από περιφερειακές συσκευές όπως οι δισκέτες, οι μαγνητικοί δίσκοι, οι ταινίες κλπ. Συνήθως αναφερόμαστε σε αυτές τις συσκευές με τον όρο βοηθητική μνήμη, μιας και χρησιμοποιούνται σαν μνήμη υπερχείλισης της κύριας μνήμης του υπολογιστικού συστήματος. Πέρα από το ότι οι μνήμες αυτές προσφέρουν πολλαπλάσιες χωρητικότητες από ότι η κύρια μνήμη του συστήματος σε υποπολλαπλάσιο κόστος ανά δυαδικό ψηφίο, κύριο χαρακτηριστικό όλων των συσκευών βοηθητικής μνήμης είναι ότι διατηρούν τα δεδομένα τους, ακόμη και χωρίς την παροχή ηλεκτρικού ρεύματος (non volatile memory). Επιπλέον επιτρέπουν τη μεταφορά δεδομένων από το ένα σύστημα στο άλλο. Σήμερα ανάμεσα στις συσκευές αυτές οι πιο διαδεδομένες είναι οι δισκέτες, οι δίσκοι, οι ταινίες και οι οπτικοί δίσκοι που εξετάζονται παρακάτω.

6.5.1. Δισκέτες

Τα δυαδικά δεδομένα σε ένα εύκαμπτο δίσκο (flexible disk) αποθηκεύονται σαν την παρουσία ή την απουσία μαγνήτισης σε μικροσκοπικά κομμάτια μαγνητικού υλικού (συνήθως οξείδιο του σιδήρου). Τα σωματίδια αυτά σε μεγάλες ποσότητες χρησιμοποιούνται για την επίστρωση μιας εύκαμπτης πλαστικής επιφάνειας, με σχήμα ανάλογο των δίσκων βινυλίου. Η εύκαμπτη αυτή επιφάνεια τελικά περικλείεται από ένα ανθεκτικότερο πλαστικό. Το συνολικό τελικό προϊόν ονομάζεται δισκέτα (diskette). Η δισκέτα έχει ένα παράθυρο μέσω του οποίου γίνεται η ανάγνωση ή η εγγραφή δεδομένων πάνω στο μαγνητικό υλικό του εύκαμπτου δίσκου, από μια συσκευή γνωστή ως μονάδα δισκέτας (Floppy Disk

Drive – FDD). Οι πρώτες δισκέτες είχαν διάμετρο 8 ιντσών. Οι πρώτοι προσωπικοί υπολογιστές του 1981 χρησιμοποίησαν δισκέτες 5,25 ιντσών, οι οποίες σήμερα έχουν αντικατασταθεί από τις δισκέττες των 3,5 ιντσών με το πολύ πιο ανθεκτικό περίβλημα.

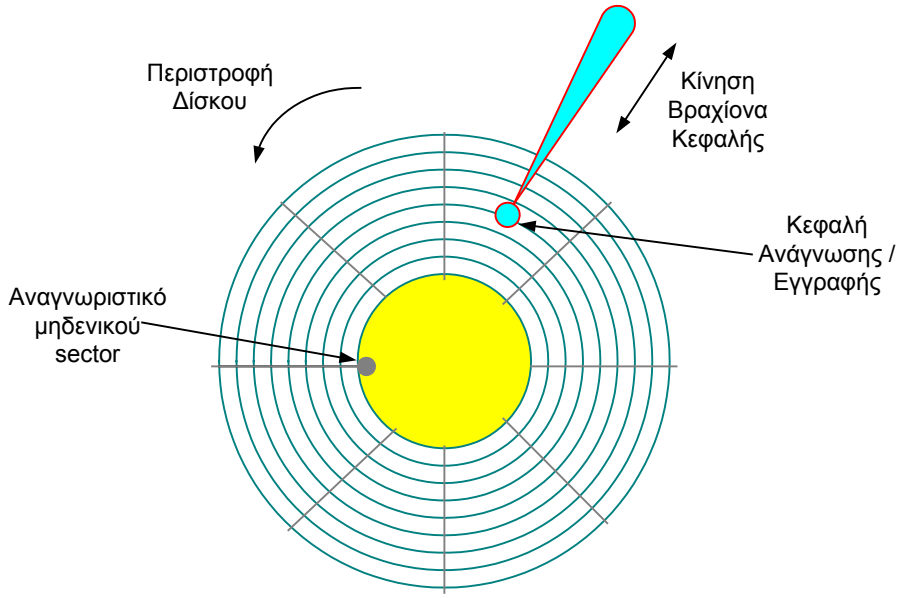
Αφού κάθε σωματίδιο μπορεί να αποθηκεύσει ένα δυαδικό ψηφίο, είναι προφανές ότι μεγαλύτερες ποσότητες πληροφορίας μπορούν να αποθηκευτούν σε μια σειρά από σωματίδια. Συνεπώς παρακάτω, εξετάζουμε τη φυσική οργάνωση των δεδομένων, πάνω σε μια δισκέτα. Κάθε επιφάνεια μιας δισκέτας διαιρείται σε μεγάλο αριθμό ομόκεντρων κύκλων, που ονομάζονται tracks. Κάθε track διαιρείται περαιτέρω σε έναν μεγάλο αριθμό από κυκλικούς τομείς, οι οποίοι ονομάζονται sectors. Σε κάθε sector αποθηκεύεται ένας σταθερός αριθμός πληροφορίας, με πιο συνηθισμένα μεγέθη τα 512, τα 1024 και τα 2048 bytes. Αυτό σημαίνει ότι οι εξωτερικοί sectors έχουν σημαντικά μικρότερη πυκνότητα εγγραφής από ότι οι κοντινοί ως προς το κέντρο sectors. Κενά, δηλαδή επιφάνειες χωρίς μαγνητικό υλικό μεσολαβούν τόσο μεταξύ των sectors όσο και μεταξύ των tracks, με σκοπό την καλύτερη συνεργασία μεταξύ ηλεκτρονικών και μηχανολογικών στοιχείων.



Η εγγραφή μιας πληροφορίας πάνω σε μια δισκέτα γίνεται πάντοτε σε ποσότητες πολλαπλάσιες του ενός sector. Αν δηλαδή κάθε sector μπορεί να αποθηκεύσει 512 bytes πληροφορίας, τότε ένα αρχείο μεγέθους 1243 bytes, θα καταλάβει 3 sectors, όπου ο τελευταίος θα είναι μερικώς γεμάτος. Συνεπώς σε μια δισκέτα συνολικού μεγέθους 1.2 Mbytes πολύ σπάνια μπορούμε να χρησιμοποιήσουμε όλο το διαθέσιμο αποθηκευτικό χώρο. Οι δισκέτες ανήκουν στις

συσκευές αποθήκευσης αμέσου προσπελάσεως. Σε αντιδιαστολή με τις συσκευές σειριακής προσπέλασης, ο χρόνος ανάκλησης μιας πληροφορίας είναι ανεξάρτητος από τη σειρά εγγραφής των διαφόρων πληροφοριών.

Η διαδικασία εγγραφής και ανάγνωσης πραγματοποιείται από μια κεφαλή ανά επιφάνεια. Για να αρχίσει η εγγραφή / ανάγνωση θα πρέπει να έχουμε τη δυνατότητα να τοποθετούμε τη κεφαλή πάνω από όποιον συγκεκριμένο sector θέλουμε. Αυτό πραγματοποιείται με τη κίνηση της κεφαλής κατά τον άξονα των tracks, και βάσει της ταχύτητας περιστροφής γνωρίζοντας έναν αρχικό sector.



Εστω για παράδειγμα, ότι η κεφαλή βρίσκεται στο track 12, (όπου τα tracks αριθμούνται από μέσα προς τα έξω) και θέλει να προσπελάσει τον sector 8 του track 3. Αυτό μεταφράζεται σε κίνηση του βραχίονα της κεφαλής προς το κέντρο της δισκέτας μέχρι το track 3 να βρεθεί κάτω από την κεφαλή. Δεδομένης της σταθερής ταχύτητας περιστροφής και κάποιου αναγνωριστικού του sector 0 σε κάθε track, η περιστροφή μπορεί να σταματήσει όταν ο sector 8 βρεθεί κάτω από την κεφαλή. Σαν αναγνωριστικά του μηδενικού sector έχουν χρησιμοποιηθεί οπτικές (μια μικρή οπή με μια φωτοδίοδο και έναν φωτοανιχνευτή εκατέρωθεν της) ή μηχανολογικές κατασκευές.

Ενα σύνολο πληροφορίας αποθηκεύεται σε μια δισκέτα σαν μια ενιαία οντότητα, ονομαζόμενη αρχείο (file). Ενα αρχείο μπορεί να καταλαμβάνει περισσότερους από έναν sectors, οι οποίοι μπορεί να βρίσκονται διάσπαρτοι σε διάφορα tracks. Είναι προφανές ότι για τη γρηγορότερη ανάκτηση ενός αρχείου από μια δισκέτα οι κινήσεις της κεφαλής χρειάζεται να ελαχιστοποιηθούν. Συνεπώς είναι επιθυμητό, οι sectors που αποτελούν ένα αρχείο να είναι διαδοχικοί (να

ανήκουν στο ίδιο track και να έπονται ο ένας του άλλου κατά τη διαδικασία περιστροφής), μιας και με μόνο μια κίνηση της κεφαλής μπορούμε σε μια περιστροφή να διαβάσουμε πολλούς από τους sectors του αρχείου. Δυστυχώς όμως, λόγω των διαρκών εγγραφών, διαγραφών και τροποποιήσεων των αρχείων που υπάρχουν σε μια δισκέτα, οι sectors που συνήθως είναι ελεύθεροι, είναι διάσπαρτοι πάνω στην επιφάνεια της δισκέτας. Ετσι κατά την εγγραφή του ένα αρχείο κερματίζεται σε sectors που τοπικά δεν είναι γειτονικοί. Το φαινόμενο αυτό ονομάζεται κερματισμός (fragmentation). Για την αποφυγή του φαινομένου αυτού συχνά χρησιμοποιούμε ρουτίνες του λειτουργικού συστήματος που αναλαμβάνουν την αναδιάρθρωση των αρχείων πάνω στη δισκέτα (defragmentation) και μειώνουν το φαινόμενο του κερματισμού, αυξάνοντας ταυτόχρονα την ταχύτητα προσπέλασης των δεδομένων.

Ομως και η αποθήκευση σε διαδοχικούς τομείς μπορεί να αποφέρει προβλήματα, λόγω της εμπλοκής των μηχανολογικών κατασκευών που απαρτίζουν μια μονάδα δισκέτας. Κατά την περιστροφή της δισκέτας, λόγω αδράνειας, μπορεί να προκληθούν : διάβασμα πέρα από το τέλος ενός αρχείου, αδυναμία ανάγνωσης επιτυχώς από διαδοχικούς sectors, αδυναμία να αντιληφθούμε το τέλος ενός sector κλπ. Διάφοροι τρόποι για την αποφυγή αυτών των προβλημάτων που έχουν προταθεί είναι :

- Η χρησιμοποίηση κώδικα Gray. Για τη δυνατόν γρηγορότερη και εγκυρότερη απόφαση σχετικά με τη θέση της κεφαλής, οι sectors δεν αριθμούνται βάσει του δυαδικού συστήματος, αλλά σε έναν κώδικα στον οποίο ο κάθε αριθμός διαφέρει από τον προηγούμενο και τον επόμενό του σε ένα μόνο δυαδικό ψηφίο. Η απεικόνιση μεταξύ δυαδικού και Gray κώδικα, γίνεται βάσει των εξής κανόνων :
  1. Αν $b_{n-1}b_{n-2} \dots b_0$ είναι η παράσταση ενός αριθμού στο δυαδικό σύστημα, τότε ο ίδιος αριθμός στον κώδικα Gray παρίσταται από τη ψηφιολέξη $g_{n-1}g_{n-2} \dots g_0$, όπου : $g_{n-1} = b_{n-1}$ και $g_k = b_{k+1} \oplus b_k$ για κ=0,1, ..., n-2 (το σύμβολο $\oplus$ υποδεικνύει τη λογική πράξη αποκλειστικής διάζευξης).
  2. Αν $g_{n-1}g_{n-2} \dots g_0$ είναι η παράσταση ενός αριθμού στον κώδικα Gray, τότε ο ίδιος αριθμός στο δυαδικό παρίσταται από τη ψηφιολέξη $b_{n-1}b_{n-2} \dots b_0$, όπου: $b_{n-1} = g_{n-1}$ και $b_k = b_{k+1} \oplus g_k$ για κ=0,1, ..., n-2.

  Ο κώδικας Gray ανήκει σε μια ευρύτερη κατηγορία κωδίκων οι οποίοι ονομάζονται κατοπτρικοί, γιατί μπορούν να κατασκευαστούν με κατοπτρισμό των ήδη κατασκευασμένων ψηφιολέξεων. Π.χ. για κώδικα Gray ενός δυαδικού ψηφίου έχουμε τον αντιστρεπτικό κατοπτρισμό 0 | 1. Για να φτιάξουμε τον κώδικα 2 δυαδικών ψηφίων παρεμβάλλουμε ένα αντιστρεπτικό κάτοπτρο και

προκύπτει το 0 | 1 | 1 0. Βάζουμε μπροστά από τις αρχικές λέξεις μας το 0 και τις κατοπτρικές το 1, οπότε παίρνουμε τις λέξεις 00 | 01 | 11 10, που είναι ο κώδικας Gray 2 ψηφίων. Με την ίδια διαδικασία παίρνουμε 00 | 01 | 11 10 | 10 11 01 00 και με την προσθήκη του αρχικού ψηφίου τις κωδικές λέξεις τριών δυαδικών ψηφίων : 000 | 001 | 011 010 |110 111 101 100, κοκ.

- Η αποθήκευση ενός αρχείου, χρησιμοποιώντας ένα ποσοστό μόνο διαδοχικών sectors, γνωστό ως interleaving factor. Για παράδειγμα, αν χρησιμοποιούμε κάποιον sector, τον μεθεπόμενό του, τον μεθεπόμενο του μεθεπομένου του κοκ, τότε έχουμε ένα interleaving factor ίσο με 2.

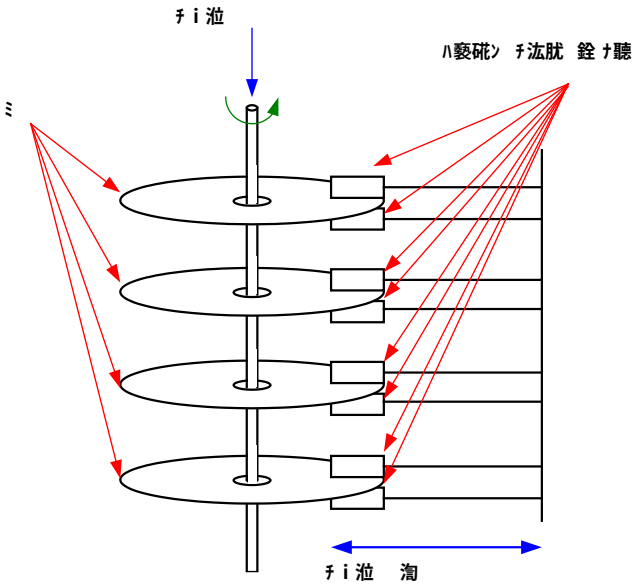
Η διεύθυνση του πρώτου sector κάθε αρχείου αποθηκεύεται στον sector 0 του track 0. Στον ίδιο sector αποθηκεύεται και η διεύθυνση του πρώτου άδειου sector. Ολες αυτές οι διευθύνσεις είναι δυάδες της μορφής <αριθμός track, sector του track>. Ο συγκεκριμένος sector ονομάζεται και sector εκκίνησης (boot sector) και για να αποφύγουμε να τον διαβάζουμε πριν από κάθε προσπέλαση, τα δεδομένα του μεταφέρονται στην κύρια μνήμη μετά την πρώτη προσπέλαση. Κάθε sector επίσης δείχνει είτε ότι είναι ο τελευταίος sector ενός αρχείου, είτε τη διεύθυνση του επομένου sector του αρχείου, είτε αν είναι κενός τη διεύθυνση του επομένου κενού sector.

Η μικρή χωρητικότητα (για τα σημερινά δεδομένα) των δισκετών περιόρισε τη χρήση τους τα τελευταία χρόνια μόνο σε περιπτώσεις μεταφοράς μικρών σε όγκο δεδομένων μεταξύ υπολογιστικών συστημάτων και για την εκκίνηση κάποιου υπολογιστικού συστήματος σε περιπτώσεις ανάγκης (σφάλμα στον σκληρό δίσκο, προσβολή από ιούς κλπ). Η ευρεία διάδοση των δικτύων υπολογιστών αναμένεται να περιορίσει ακόμη περισσότερο τη χρήση τους στο μέλλον.

6.5.2. Σκληροί Δίσκοι (Hard Disks)

Για την αποθήκευση μεγαλύτερων ποσοτήτων πληροφορίας στα σημερινά συστήματα χρησιμοποιούνται οι σκληροί δίσκοι (στο εξής θα αναφέρονται σαν δίσκοι). Οι δίσκοι αποτελούνται από περισσότερες της μιας μαγνητικές επιφάνειες (πλατό) που η κάθε μία τους έχει οργάνωση αντίστοιχη με αυτήν μιας δισκέτας και υπάρχουν τουλάχιστον τόσες κεφαλές όσες και οι μαγνητικές επιφάνειες. Ολες οι μαγνητικές επιφάνειες περιστρέφονται ταυτόχρονα γύρω από τον ίδιο άξονα. Συνήθης ταχύτητα περιστροφής είναι οι 7200 περιστροφές ανά λεπτό (revolutions per minute – rpm). Στο παρακάτω παράδειγμα όλες οι κεφαλές ανάγνωσης-

εγγραφής κινούνται ταυτόχρονα, αφού όλες τους στερεώνονται στον ίδιο αρμό (moving head disk). Στους σκληρούς δίσκους χρησιμοποιείται και η ορολογία του κυλίνδρου, που συμβολίζει τον νοητό κύλινδρο που σχηματίζεται θεωρώντας το ίδιο track στα διάφορα πλατό που απαρτίζουν το δίσκο. Στους δίσκους μπορούν να χρησιμοποιηθούν πολλές διαφορετικές οργανώσεις για τα αρχεία. Για παράδειγμα sectors του ίδιου αρχείου μπορούν παράλληλα να γράφονται και να διαβάζονται στα διαφορετικά tracks του ίδιου κυλίνδρου, ενώ σε άλλες περιπτώσεις σε κάθε πλατό μπορεί να προσπελαύνονται διαφορετικά bits της αποθηκευμένης πληροφορίας.



Ο χρόνος που απαιτείται για τη μεταφορά πληροφορίας μεγέθους ενός sector από και προς ένα δίσκο, αποτελείται από τις εξής συνιστώσες :

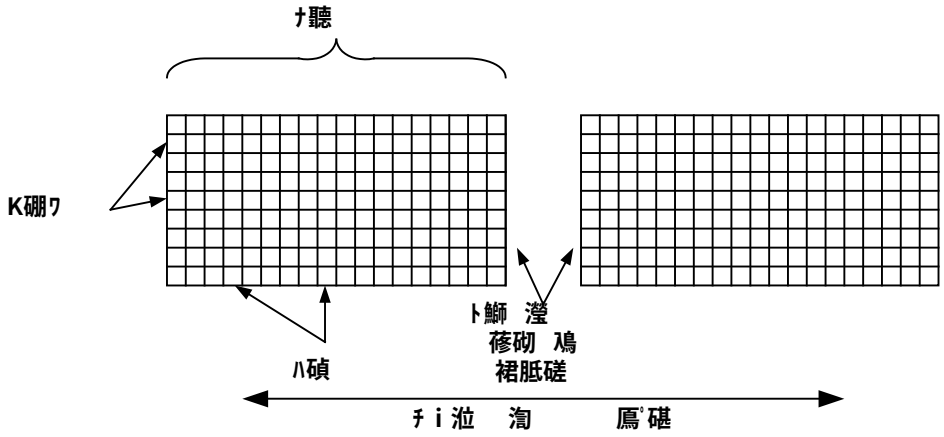
- Το χρόνο τοποθέτησης της κεφαλής στο σωστό track (χρόνος αναζήτησης – head seek time).
- Το χρόνο περιστροφής μέχρι ο σωστός sector να βρεθεί κάτω από την κεφαλή (χρόνος περιστροφής – rotational delay) και
- Το χρόνο ανάγνωσης του sector.

Η πρώτη από τις παραπάνω συνιστώσες είναι και η μεγαλύτερη. Για τη μετακίνηση μεταξύ γειτονικών tracks, μια κεφαλή απαιτεί χρόνο μερικών χιλιοστών του δευτερολέπτου. Η μείωση αυτής της συνιστώσας μπορεί να επιτευχθεί με την ύπαρξη περισσοτέρων της μίας μετακινούμενων κεφαλών ανά πλατό. Για παράδειγμα υποθέστε ότι διαθέτουμε 2 κεφαλές και 1024 tracks ανά επιφάνεια του δίσκου. Αναθέτοντας στην μία κεφαλή να εξυπηρετεί αιτήσεις των tracks 0 έως 511 και στη δεύτερη των υπολοίπων, μειώνουμε στο μισό το μέγιστο χρόνο αναζήτησης. Στην ακραία περίπτωση μπορούμε να έχουμε όσες κεφαλές όσα και τα

tracks κάθε επιφάνειας. Οι δίσκοι αυτοί ονομάζονται δίσκοι σταθερής κεφαλής (fixed heads disks) και φυσικά προσφέρουν μηδενικό χρόνο αναζήτησης. Ο αριθμός των κεφαλών ανά επιφάνεια ωστόσο καθορίζει και το κόστος ενός δίσκου. Οι δίσκοι σταθερών κεφαλών έχουν πολλαπλάσιο κόστος από τους αντίστοιχης χωρητικότητας δίσκους κινητής κεφαλής.

6.5.3. Μαγνητικές Ταινίες (Magnetic Tapes)

Σε αναλογία με τις ταινίες μαγνητοφώνου ή μπομπινοφώνου, οι μαγνητικές ταινίες (στο εξής θα αναφέρονται σαν ταινίες) αποτελούν ένα φτηνό μέσο σειριακής αποθήκευσης δεδομένων. Μια μονάδα μαγνητικής ταινίας συνήθως έχει μόνο μια κεφαλή, μπροστά από την οποία περνάει η ταινία που είναι επιστρωμένη με το μαγνητικό υλικό. Κατά την εγγραφή αυτό μαγνητίζεται επιλεκτικά, ενώ κατά την ανάγνωση η κεφαλή αισθάνεται ή όχι την παρουσία μαγνήτισης. Οι ταινίες χρησιμοποιούνται για την αποθήκευση μεγάλων ποσοτήτων πληροφοριών της τάξης των GB. Επίσης χρησιμοποιούνται πολύ συχνά για την δημιουργία αντιγράφων ασφαλείας. Η προσπέλαση κάποιας πληροφορίας πάνω στην ταινία είναι μια ιδιαίτερα χρονοβόρα διαδικασία, αφού όλες οι προηγούμενες πληροφορίες θα πρέπει να περάσουν μπρος από την κεφαλή μέχρι να βρεθεί η στοχευόμενη πληροφορία.



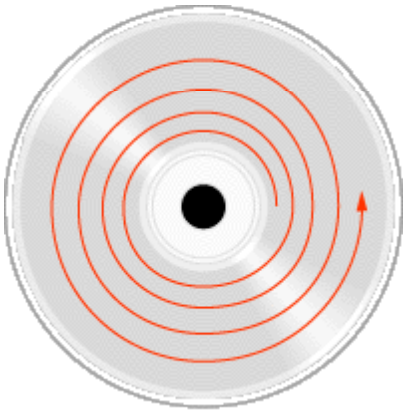
Η φυσική οργάνωση της πληροφορίας σε μια ταινία γίνεται σε δύο διαστάσεις. Η ταινία κατά πλάτος χωρίζεται σε διακριτές ζώνες, ονομαζόμενες κανάλια (tracks). Συνήθως υπάρχουν 9 κανάλια, τα οποία αποθηκεύουν ένα byte της πληροφορίας μαζί με ένα ψηφίο ισοτιμίας (parity – δες κεφάλαιο 8). Τα 9 αυτά ψηφία σχηματίζουν ένα καρέ (frame) πάνω στην ταινία. Μια ομάδα από συνεχή Καρρά συνιστούν μια εγγραφή (record) της ταινίας. Η εγγραφή είναι η μικρότερη ποσότητα πληροφορίας που μπορεί να διαβαστεί ή να γραφεί σε μια ταινία και αυτό συμβαίνει λόγω των μηχανολογικών κατασκευών που συνυπάρχουν σε ένα

μηχανισμό ταινίας και που δεν επιτρέπουν μικροσκοπικές κινήσεις της ταινίας. Για τον ίδιο λόγο ενδιάμεσα σε δύο εγγραφές πάντα υπάρχει ένα κενό κομμάτι της ταινίας που ονομάζεται διάκενο μεταξύ των εγγραφών (inter-record gap).

6.5.4. Οπτικοί Δίσκοι (Optical Disks)

Οι οπτικοί δίσκοι που χρησιμοποιούνται στα σημερινά υπολογιστικά συστήματα είναι τα CDs (compact disks) και τα DVDs (Digital Video Disks). Μια για τα τελευταία δεν υπάρχει ακόμη μια κοινή ενιαία αντιμετώπιση η παρακάτω συζήτηση επικεντρώνεται στα CD. Τα CD εισήχθησαν το 1980 για ψηφιακή αποθήκευση και αναπαραγωγή ήχου και έκτοτε έχουν γίνει πιο φθηνά, πολύ πιο αξιόπιστα και η πυκνότητα εγγραφής πληροφοριών σε αυτά έχει αυξηθεί σημαντικά. Οι λόγοι αυτοί οδήγησαν στην υιοθέτησή τους στα τρέχοντα υπολογιστικά συστήματα, στην αρχή με τη μορφή των CD ROMs.

Τα CD ROMs κατασκευάζονται μέσω πίεσης έναντι ενός πρότυπου καλουπιού. Η πίεση αυτή δημιουργεί κοιλάδες στην επιφάνεια του CD, οι οποίες ανακλούν διαφορετικά το φώς από ότι η υπόλοιπη επιφάνεια του δίσκου. Η αποθήκευση της πληροφορίας σε ένα CD γίνεται όμοια με αυτήν ενός δίσκου, με μόνη διαφορά ότι ενώ στο δίσκο τα κανάλια είναι οργανωμένα σαν ομόκεντροι κύκλοι, εδώ έχουμε μια σπειροειδή μορφή. Ο λόγος είναι ότι ένας δίσκος περιστρέφεται με σταθερή γωνιακή ταχύτητα και αλλάζουμε τη πυκνότητα εγγραφής, ενώ σε ένα οπτικό δίσκο η γραμμική ταχύτητα και η πυκνότητα εγγραφής είναι σταθερές.

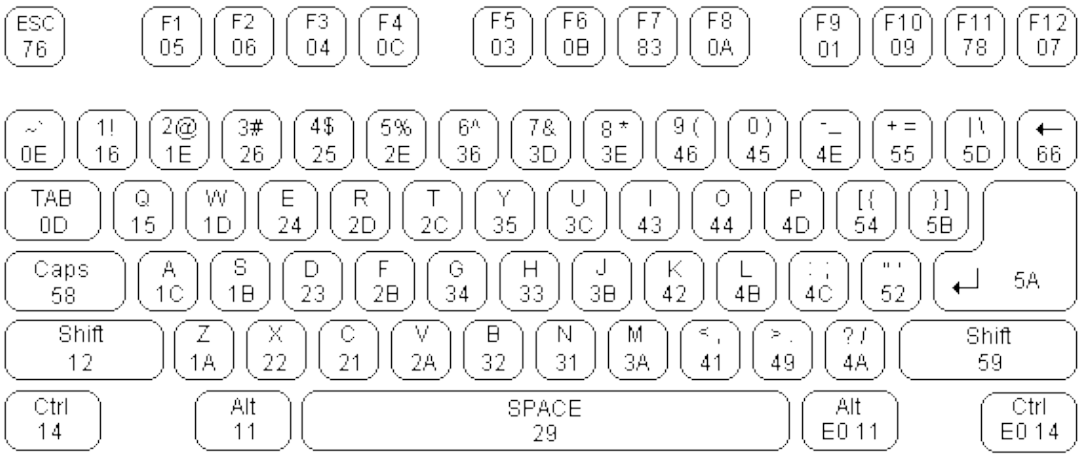


Πρόσφατα στην αγορά εμφανίστηκαν οι οπτικοί δίσκοι τεχνολογίας WORM (write once read many) και CD-R (read / write CDs). Για την εγγραφή τους απαιτείται ειδικευμένη συσκευή, η οποία χαράζει τον οπτικό δίσκο με ακτίνα laser χαμηλής εντάσεως. Η εγγραφή σε αυτές τις συσκευές διαρκεί σημαντικά περισσότερο από ότι η ανάγνωση ενός δίσκου.

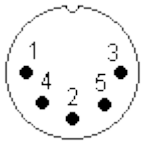
6.6 Συσκευές Εισόδου

6.6.1. Πληκτρολόγιο (Keyboard)

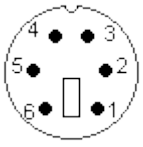
Το πληκτρολόγιο συγκαταλέγεται ανάμεσα στα πιο αργά περιφερειακά ενός υπολογιστικού συστήματος. Ως εκ τούτου η σύνδεσή του με το υπόλοιπο υπολογιστικό σύστημα γίνεται βάσει μιας σειριακής αρτηρίας. Στα παρακάτω αναλύεται η ανάγνωση δεδομένων από ένα πληκτρολόγιο τεχνολογίας AT. Τα περισσότερα πληκτρολόγια λειτουργούν με παρόμοιους τρόπους. Σε κάθε πλήκτρο αντιστοιχεί ένας κωδικός 8 δυαδικών ψηφίων που ονομάζεται κώδικας σάρωσης (scan code). Οι κώδικες σάρωσης για ένα πληκτρολόγιο AT φαίνονται στην παρακάτω εικόνα.



Το πάτημα ενός πλήκτρου, ισοδυναμεί με τη μεταφορά του αντίστοιχου κωδικού σάρωσης. Ο κωδικός σάρωσης θα συνεχίσει να παράγεται και να μεταδίδεται μέχρι να αφεθεί το πλήκτρο οπότε παράγεται ένας κωδικός αποτελούμενος από 2 bytes : το $F0_{16}$ και τον κωδικό σάρωσης. Για τη μεταφορά αυτών των δεδομένων πάνω από την αμφίδρομη σειριακή αρτηρία μεταδίδονται τόσο οι πληροφορίες (γραμμή KBD Data) όσο και το ρολόι του αποστολέα (KBD Clock), μιας και στην περίπτωση αυτή η συχνότητα των δεδομένων είναι μικρή· περίπου 20 – 30 KHz. Τα σήματα που χρησιμοποιούνται σε 5 Pin DIN και PS/2 προσαρμογείς είναι όπως παρακάτω :



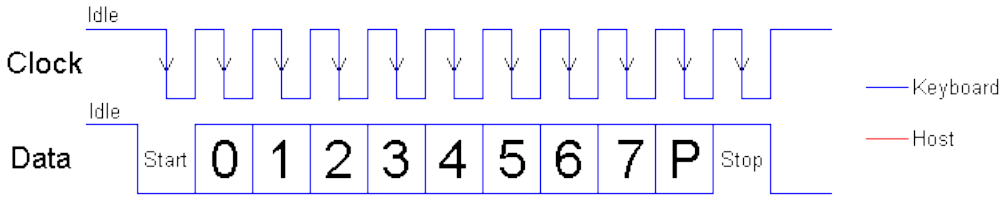
1. KBD Clock
2. KBD Data
3. N/C
4. GND
5. +5V (VCC)



1. KBD Clock
2. GND
3. KBD Data
4. N/C
5. +5V (VCC)
6. N/C

Το σειριακό πρωτόκολλο ανταλλαγής δεδομένων στην περίπτωση του πληκτρολογίου ακολουθεί τους εξής κανόνες :

- Οταν δεν ανταλλάσσονται δεδομένα η γραμμή δεδομένων (Data) είναι ανενεργή (στο λογικό 1).
- Για την αποστολή δεδομένων απαιτείται η αποστολή ενός πακέτου 11 bits με χρονισμό που φαίνεται στην παρακάτω εικόνα.



- Το πρώτο δυαδικό ψηφίο της γραμμής δεδομένων είναι ένα ψηφίο εκκίνησης (start bit), ακολουθούμενο από τα 8 δυαδικά ψηφία του κώδικα σάρωσης, ένα ψηφίο ***περιττής*** ισοτιμίας (δες κεφάλαιο 8) και ένα δυαδικό ψηφίο τερματισμού (stop bit).
- Το πρώτο δυαδικό ψηφίο πληροφορίας που αποστέλλεται είναι το λιγότερο σημαντικό (τόσο για τον κώδικα σάρωσης, όσο και για το $F0_{16}$).
- Ο παραλήπτης της πληροφορίας για τη δειγματοληψία θα πρέπει να χρησιμοποιήσει την ***αρνητική*** ακμή του Clock.

6.6.2. Λοιπές συσκευές εισόδου

Στις λοιπές συσκευές εισόδου μπορούμε να κατατάξουμε τα ποντίκια, τα trackballs, τις διατάξεις ψηφιοποίησης σχεδίου (digitizing tablet – bit pad), τα στυλό φωτός (lightpens) οι οθόνες αφής (touch screens) και τους μοχλούς – χειριστήρια (joystics).

Το ποντίκι είναι μια συσκευή εισόδου ενός χεριού. Υπάρχουν μηχανικά και οπτικά ποντίκια. Στα μηχανικά ποντίκια, μια μικρή λαστιχένια μπάλα στο κάτω μέρος του ποντικιού περιστρέφεται ανάλογα με τη κίνηση του ποντικιού. Η κίνηση μέσω μικρών αισθητήρων μεταφέρεται στο ηλεκτρονικό μέρος του ποντικιού, όπου μετατρέπεται σε δύο διαφορετικές πληροφορίες : την κατεύθυνση της κίνησης και την απόσταση που διανύθηκε. Οι δύο πληροφορίες αυτές μαζί με τη κατάσταση των πλήκτρων του ποντικιού μεταφέρονται στο υπολογιστικό σύστημα. Τα οπτικά ποντίκια χρησιμοποιούν μια ειδικά κατασκευασμένη πλατφόρμα κίνησης πάνω στην οποία με κάθετες και οριζόντιες γραμμές έχουν οριστεί περιοχές που ανακλούν ή

απορροφούν το φως. Το οπτικό ποντίκι αντί της λαστιχένιας μπάλας έχει μια φωτοδίοδο (light emitting diode – LED) και αισθητήρες φωτός προς τα τέσσερα σημεία του ορίζοντα. Καθώς μετακινούμε το ποντίκι, η κίνηση μεταφράζεται σε παλμοσειρές ύπαρξης ή μη ύπαρξης φωτός από τον κάθε αισθητήρα. Τα trackballs είναι αντίστοιχα με ένα αναποδογυρισμένο ποντίκι, όπου αντί να κουνάμε το σώμα του ποντικιού, κουνάμε τη μικρή μπάλα.

Οι διατάξεις ψηφιοποίησης σχεδίου χρησιμοποιούνται κυρίως από αρχιτέκτονες και γραφίστες για τη μεταφορά ενός σχεδίου από το χαρτί στον υπολογιστή. Οι διατάξεις αυτές αποτελούνται από μια ειδική επιφάνεια και ένα δείκτη στο σχήμα ενός ποντικιού. Στο κάτω μέρος της ειδικής επιφάνειας υπάρχει ένα πλέγμα από δεκάδες χιλιάδες καλώδια ικανά να αισθάνονται το ρεύμα που επάγεται πάνω τους από τον δείκτη όπως αυτός κινείται στα διάφορα μέρη της επιφάνειας. Ανάλογα με το ποιά καλώδια μας δίνουν ρεύμα μπορούμε να καθορίσουμε κάθε στιγμή τη θέση του δείκτη.

Τα στυλό φωτός στην ουσία δεν παράγουν φώς, αλλά αισθάνονται το φως που εκλύεται από μια οθόνη. Το φως κάθε εικονοστοιχείου της οθόνης ανανεώνεται 30 έως 60 φορές το δευτερόλεπτο. Αν ένα στυλό φωτός έχει τοποθετηθεί πάνω σε ένα τέτοιο εικονοστοιχείο, τότε συλλαμβάνει το επιπλέον αρχικό φως κατά τη σάρωση ενός εικονοστοιχείου και συνεπώς μας δίνει τη συντεταγμένη που βρίσκεται εκείνη τη στιγμή το στυλό. Στις οθόνες αφής, η επιφάνεια της οθόνης καλύπτεται από ένα πλέγμα φωτεινών δεσμών που εκπέμπονται κατά τον κάθετο και οριζόντιο άξονά της. Απέναντι από τους εκπομπούς των φωτεινών δεσμών υπάρχουν αισθητήρες. Οταν ο χρήστης με το χέρι του επιλέξει κάτι, ταυτόχρονα διακόπτει τις αντίστοιχες δέσμες κατά τον οριζόντιο και κάθετο άξονα. Η τομή αυτών των δεσμών μας δίνει τη συντεταγμένη της πληροφορίας που εισήχθηκε.

6.7 Συσκευές Εξόδου

6.7.1 Η οθόνη

Η οθόνη αποτελεί τη μονάδα απεικόνισης ενός υπολογιστικού συστήματος και όταν είναι ξεχωριστό κομμάτι του υπολογιστικού συστήματος αναφέρεται και ως μόνιτορ. Υπάρχουν διάφορες τεχνολογίες κατασκευής οθονών. Οι πιο διαδεδομένες ανάμεσά τους είναι :

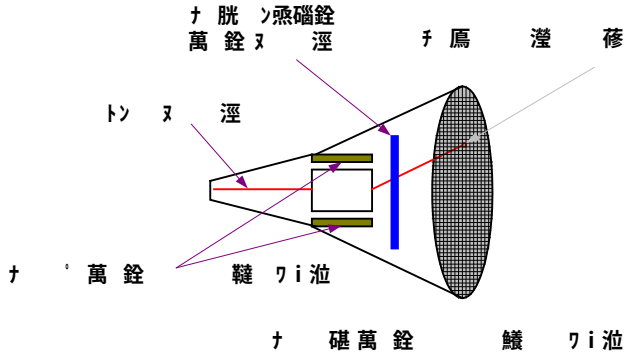
- Οι οθόνες καθοδικού σωλήνα (cathode ray tube – CRT) που είναι η κύρια τεχνολογία κατασκευής οθονών για υπολογιστές γραφείου

- Οι οθόνες υγρών κρυστάλλων (liquid crystal display – LCD) που είναι ανάμεσα στις δύο δημοφιλέστερες τεχνολογίες οθονών για φορητούς υπολογιστές
- Οι οθόνες φωτοδιόδων (light-emitting diode – LED)
- Οι οθόνες αερίων (neon and xenon gas plasma) και
- Οι οθόνες επίστρωσης φωτοτρανζίστορ (Thin – Film Transistor - TFT), η πιο δημοφιλής τεχνολογία οθονών για φορητούς υπολογιστές.

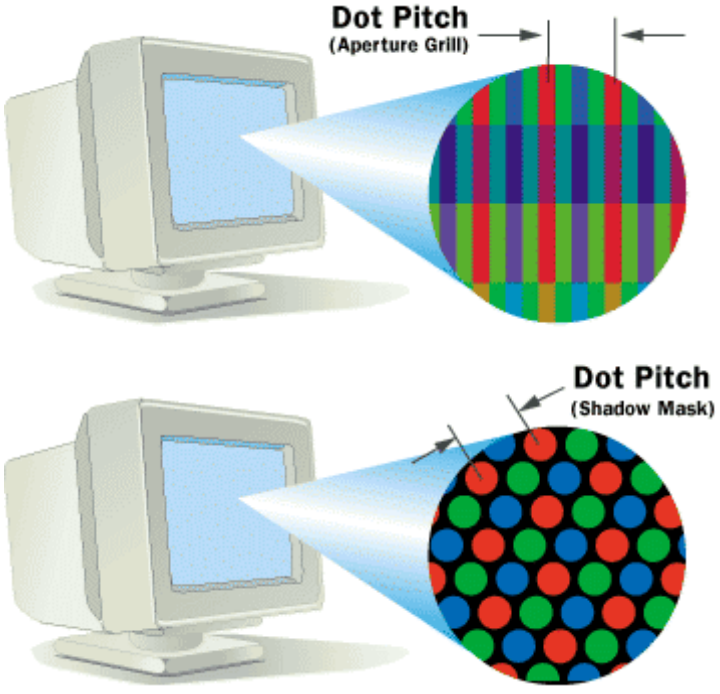
Ανεξάρτητα από την τεχνολογία που χρησιμοποιούμε για τη κατασκευή μιας οθόνης, η εικόνα που σχηματίζεται σε αυτές αποτελείται από μικροσκοπικά εικονοστοιχεία (pixels). Οσα περισσότερα είναι αυτά τα εικονοστοιχεία τόσο μεγαλύτερη είναι η ευκρίνεια της σχηματιζόμενης εικόνας. Ο αριθμός των δυαδικών ψηφίων που αφιερώνουμε για την παράσταση κάθε εικονοστοιχείου, στην ουσία μας δίνει τον αριθμό των διαφορετικών αποχρώσεων που αυτό μπορεί να λάβει. Ο αριθμός των εικονοστοιχείων (ισοδύναμα η ανάλυση της απεικόνισης – resolution) και τα δυαδικά ψηφία ανά εικονοστοιχείο δεν είναι αυθαίρετα, αλλά ακολουθούν κάποια πρότυπα τα οποία εμφανίστηκαν με την εξής ιστορική σειρά :

- Το 1981, η IBM εισήγαγε το πρότυπο CGA (Colour Graphics Adapter), με ανάλυση 320x200 εικονοστοιχεία (ο πρώτος αριθμός μας δίνει τα εικονοστοιχεία κατά τον οριζόντιο άξονα και ο δεύτερος κατά τον κάθετο) και 4 διαφορετικά χρώματα.
- Το 1984 η IBM εισήγαγε το πρότυπο EGA (Enhanced Graphics Adapter), με ανάλυση 640x350 και 16 διαφορετικά χρώματα.
- Το 1987 και το 1990 αντίστοιχα η IBM εισήγαγε τα πρότυπα VGA (Video Graphics Adapter) και SVGA (Super Video Graphics Adapter) – XGA (EXtended Graphics Array) με ανάλυση που μπορεί να φτάσει τα 800x600 εικονοστοιχεία με πραγματικό χρώμα (16,8 εκατομμύρια χρώματα – όσες διαφορετικές αποχρώσεις μπορεί να ξεχωρίσει ένα έμπειρο μάτι) ή 1024x768 και 2^{16} διαφορετικά χρώματα.
- Σήμερα, οι περισσότερες οθόνες ακολουθούν το πρότυπο Ultra Extended Graphics Array (UXGA), το οποίο υποστηρίζει αναλύσεις έως και 1600x1200 εικονοστοιχείων και μέχρι 16,8 εκατομμυρίων χρωμάτων.

Παρακάτω επικεντρώνουμε τη συζήτησή μας σε οθόνες τεχνολογίας CRT, αν και τα περισσότερα ισχύουν και για όλες τις υπόλοιπες τεχνολογίες. Η δημιουργία μιας εικόνας σε μια οθόνη CRT γίνεται βάσει του φωτισμού που παράγουν μικροσκοπικά κομμάτια φωσφόρου. Κάθε κομμάτι φωσφόρου παράγει φωτισμό ανάλογο με τη προσπίπτουσα ακτινοβολία που φτάνει σε αυτό, σύμφωνα με το παρακάτω σχήμα :

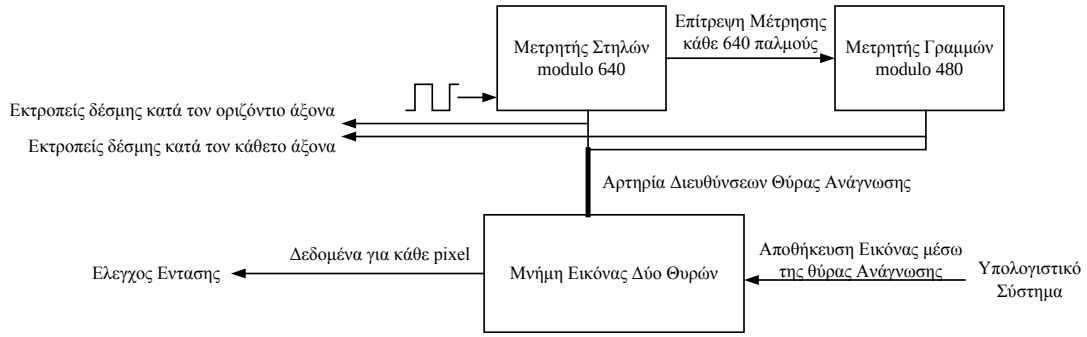


Η διάταξη στην περίπτωση των CRT οθονών αποτελείται από μια εκπεμπόμενη δέσμη ηλεκτρονίων, η οποία ανάλογα με το ποιο εικονοστοιχείο της εικόνας εξετάζεται εκτρέπεται κατά τον οριζόντιο και κάθετο άξονα βάσει ζευγών ηλεκτροδίων. Ανάλογα με τα δεδομένα που πρέπει να απεικονιστούν, περιορίζεται η ακτινοβολία της δέσμης που φτάνει στα μικροσκοπικά σωματίδια φωσφόρου, και συνεπώς προκαλεί διαφορετική λάμψη στο καθένα τους. Στην περίπτωση των έγχρωμων οθονών σε κάθε σημείο της οθόνης, υπάρχουν συνήθως 3 διαφορετικά είδη φωσφόρου διατεταγμένα με έναν κανονικό τρόπο, και τρεις δέσμες ηλεκτρονίων. Ανεξάρτητα από τη διάταξη των ειδών φωσφόρου που υιοθετείται η απόσταση μεταξύ δύο ίδιων ειδών φωσφόρου, ονομάζεται βήμα κουκίδας (dot pitch), όπως φαίνεται στο παρακάτω σχήμα.



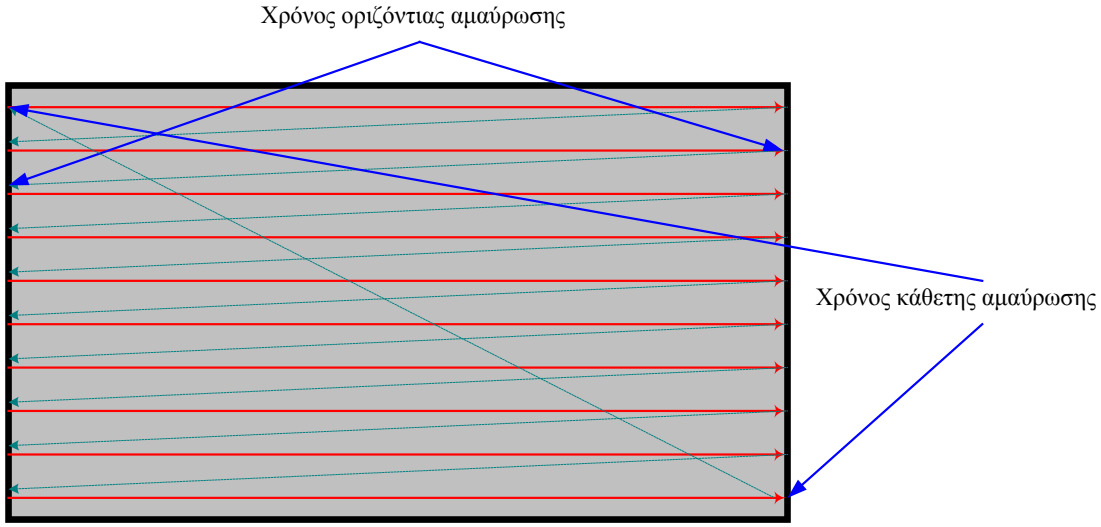
Η επικοινωνία του υπολογιστικού συστήματος με την οθόνη, συνήθως γίνεται μέσω μιας μνήμης δύο θυρών (two-port memory) που πολλές φορές ονομάζεται και μνήμη εικόνας (Video RAM – VRAM). Σε μια μνήμη δύο θυρών επιτρέπεται η ταυτόχρονη εγγραφή μιας θέσης μνήμης και η ανάγνωση από κάποια

άλλη θέση μνήμης. Στην περίπτωση της μνήμης εικόνας το υπολογιστικό σύστημα γράφει την επόμενη εικόνα που θέλει να απεικονίσει και ο ελεγκτής εικόνας διαβάζει και απεικονίζει την παρούσα. Στην περίπτωση μιας έγχρωμης οθόνης με ανάλυση 640 x 480 εικονοστοιχείων, το λογικό διάγραμμα του ελεγκτή θα μπορούσε να είναι το παρακάτω :



Το σχήμα αποτελείται από έναν μετρητή που μετράει από το 0 έως το 479 και κατευθύνει έτσι τους εκτροπείς της δέσμης κατά τον κάθετο άξονα να επιλέγουν μια από τις γραμμές της οθόνης. Για κάθε τιμή αυτού του μετρητή, ο μετρητής στηλών παίρνει όλες τις τιμές από το 0 έως το 639 και βάσει αυτών οι εκτροπείς της δέσμης κατά τον οριζόντιο άξονα εκτρέπουν τη δέσμη σε κάθε μια από τις στήλες της οθόνης. Παράλληλα οι δύο αυτές τιμές των μετρητών, αποτελούν τη διεύθυνση ανάγνωσης από τη μνήμη εικόνας. Τα δεδομένα που προκύπτουν για κάθε ζεύγος τιμών των δύο μετρητών ελέγχουν την ένταση της δέσμης που προσπίπτει σε κάθε εικονοστοιχείο.

Είναι προφανές από τα παραπάνω ότι μια εικόνα σχηματίζεται στην οθόνη με τη σάρωση των εικονοστοιχείων της, σύμφωνα με το παρακάτω σχήμα :



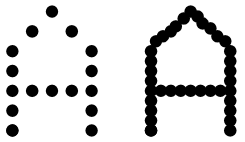
Για να δίνεται η εντύπωση της ανεάου κινήσεως στο ανθρώπινο μάτι, αρκεί η παραπάνω σάρωση να γίνεται τουλάχιστον 25 φορές κάθε δευτερόλεπτο. Ωστόσο, αφενός επειδή τα μικροσκοπικά σωματίδια φωσφόρου χάνουν εύκολα το φορτίο τους, αλλά και εξαιτίας ότι το ανθρώπινο μάτι έρχεται πολύ κοντύτερα στην οθόνη του υπολογιστή από ότι σε μια τηλεόραση, η παραπάνω συχνότητα δεν είναι αρκετή. Στις πρώτες οθόνες η σάρωση εκτελείτο περίπου 50 φορές το δευτερόλεπτο όπου όμως σε διαδοχικές σαρώσεις ανανεώνονταν διαδοχικά μόνο οι άρτιες και οι περιττές γραμμές της οθόνης (interlaced). Ενα τρεμόπαιγμα που δημιουργείτο από αυτήν την μη συνεχή σάρωση των γραμμών, λύθηκε στις σύγχρονες μη πολυπλεγμένες (non-interlaced) οθόνες με την ύπαρξη περισσότερων της μιας δεσμών που ταυτόχρονα σαρώνουν τις περιττές και τις άρτιες γραμμές της οθόνης. Ωστόσο λόγω αφενός της αυξημένης συχνότητας σάρωσης όσο και του διαρκώς συρρικνούμενου βήματος κουκίδας, πλέον απαιτείται μεταξύ της σάρωσης διαδοχικών γραμμών κάποιος χρόνος απομάκρυνσης του φορτίου που τυχόν ήδη έχει επαχθεί στα σωματίδια φωσφόρου (χρόνος οριζόντιας αμαύρωσης – horizontal blanking). Αντίστοιχος χρόνος απαιτείται πριν τη σάρωση ενός καινούργιου καρέ (χρόνος κάθετης αμαύρωσης – vertical blanking).

6.7.2 Ο εκτυπωτής

Μία από τις πλέον διαδεδομένες περιφερειακές συσκευές των σημερινών υπολογιστικών συστημάτων είναι οι εκτυπωτές (printers). Υπάρχουν δεκάδες δυνατές κατηγοριοποιήσεις των εκτυπωτών που είναι εμπορικά διαθέσιμοι. Παρακάτω επικεντρώνουμε στις πιο διαδεδομένες κατηγορίες εκτυπωτών.

Στους εκτυπωτές πρόσκρουσης, η εκτύπωση επιτυγχάνεται μέσω της πρόσκρουσης μιας κεφαλής, πάνω στην οποία υπάρχει ή σχηματίζεται ο προς εκτύπωση χαρακτήρας, με μια μελανοταινία, η οποία παρεμβάλλεται μεταξύ της κεφαλής εκτύπωσης και του χαρτιού. Στην περίπτωση που οι πιθανοί εκτυπώσιμοι χαρακτήρες είναι λίγοι, αυτοί μπορεί να προϋπάρχουν και απλά να επιλέγεται ένας από αυτούς κάθε φορά. Στην περίπτωση αυτή οι εκτυπωτές ονομάζονται εκτυπωτές αλυσίδας ή τροχού (daisy chain / daisy wheel printers). Οι εκτυπωτές αυτοί προσφέρουν άριστη ποιότητα εκτύπωσης. Το περιορισμένο σύνολο εκτυπώσιμων χαρακτήρων, η αδυναμία τους να εκτυπώσουν γραφικά και ο αρκετός θόρυβος κατά τη λειτουργία τους, έχει περιορίσει σημαντικά τη χρήση τους. Για την εκτύπωση γραφικών, αντί για προκατασκευασμένους χαρακτήρες, η κεφαλή αποτελείται από έναν πίνακα ακίδων (dot matrix). Ο πίνακας αυτός μπορεί

να αποτελείται από 7 έως 24 ακίδες. Κάθε ακίδα ενεργοποιείται με ηλεκτρομαγνητικά μέσα ανάλογα με το χαρακτήρα που τυπώνεται. Στην παρακάτω αριστερά εικόνα βλέπουμε την εκτύπωση του Α από έναν εκτυπωτή επτά ακίδων. Η ποιότητα της εκτύπωσης μπορεί να βελτιωθεί είτε με τη χρήση περισσότερων ακίδων είτε με την επικάλυψή τους (με κινήσεις δηλαδή είτε του χαρτιού είτε της κεφαλής σε βήματα μικρότερα της μιας κουκίδας – κάτω δεξιά εικόνα). Ωστόσο κάτι τέτοιο αυξάνει σημαντικά τον χρόνο εκτύπωσης. Οι εκτυπωτές πρόσκρουσης χρησιμοποιούνται σήμερα για την εκτύπωση σε πολύ μικρά ή πολύ μεγάλα μεγέθη χαρτιών και προτυπωμένες φόρμες. Είναι επίσης ιδανικοί για εκτύπωση σε πολλαπλές φόρμες με καρμπόν.



Αντίστοιχη είναι η φιλοσοφία που χρησιμοποιείται στους εκτυπωτές εκτόξευσης μελάνης. Στην περίπτωση αυτή οι ακίδες αντικαθίστανται από μικροσκοπικούς ψεκαστήρες και η μελανοταινία από μια δεξαμενή υγρής μελάνης. Η τεχνολογία αυτή μας δίνει εκτυπωτές με σημαντικά πλεονεκτήματα :

α) Είναι εντελώς αθόρυβοι

β) Με τη χρήση πολλαπλών μελανιών μπορούμε να πετύχουμε έγχρωμες εκτυπώσεις.

γ) Μπορούμε να εκτυπώσουμε πάνω σε διαφάνειες ή οποιαδήποτε άλλη επιφάνεια.

Η διακριτότητα αυτών των εκτυπωτών καθώς και αυτών του πίνακα ακίδων μετριέται στο πόσα σημεία μπορούν να εκτυπώσουν σε μια ίντσα χαρτιού (dots per inch – dpi). Σήμερα οι εκτυπωτές εκτόξευσης μελάνης μπορούν να προσεγγίσουν τη φωτογραφική ποιότητα εκτύπωσης (2400 dpi) και συνεπώς αποτελούν ιδανική λύση για εκτύπωση φωτογραφιών. Κύριο μειονέκτημα αυτών των εκτυπωτών είναι οι χαμηλές ταχύτητες εκτύπωσης στη μέγιστη ανάλυση και το υψηλό κόστος χρήσης.

Οι εκτυπωτές που χρησιμοποιούνται κατά κόρο στη σημερινή εποχή για τον αυτοματισμό γραφείου είναι οι τεχνολογίας λέιζερ. Οι εκτυπωτές αυτοί παράγουν μονομιάς μια σελίδα εκτύπωσης και συνεπώς μπορούν να χαρακτηριστούν και σαν εκτυπωτές σελίδας. Κάθε κύκλος εκτύπωσης σε αυτούς τους εκτυπωτές ακολουθεί τα εξής βήματα :

1. Το περιστρεφόμενο τύμπανο φορτίζεται και καλύπτεται με ένα φωτοευαίσθητο υλικό.

2. Μια δέσμη λέιζερ σαρώνει στη συνέχεια το τύμπανο γραμμή προς γραμμή (κατά τρόπο ανάλογο με την οθόνη) με τη βοήθεια ενός κατόπτρου και αποφορτίζει ορισμένα από τα σημεία του τυμπάνου.
3. Ενα στρώμα γραφίτη έρχεται σε επαφή με κάθε σαρωμένη γραμμή. Οι κουκίδες που συνεχίζουν να έχουν ηλεκτρικό φορτίο απορροφούν μερικά μόρια γραφίτη.
4. Κάθε γραμμή αφού περάσει πάνω από τον γραφίτη πιέζεται πάνω στο χαρτί όπου εναποτίθεται ο γραφίτης που είχε απορροφηθεί.
5. Ακολουθεί θέρμανση του χαρτιού ώστε η εναπόθεση του γραφίτη να πάρει μόνιμη μορφή.
6. Το τύμπανο τέλος αποφορτίζεται και απομακρύνεται τυχόν εναπομείνας γραφίτης ώστε να είναι έτοιμο για την επόμενη εκτύπωση.

Οι εκτυπωτές τεχνολογίας λέιζερ προσφέρουν άριστη ποιότητα εκτύπωσης (της τάξης των 1200 dpi) με σχετικά χαμηλό κόστος χρήσης. Στα μειονεκτήματά τους συγκαταλέγονται το υψηλό αρχικό κόστος κτήσης, η αδυναμία για έγχρωμη εκτύπωση (οι έγχρωμοι λέιζερ εκτυπωτές που είναι εμπορικά διαθέσιμοι κοστίζουν σήμερα περίπου 10 φορές περισσότερο από έναν αντίστοιχο ασπρόμαυρο) και η αδυναμία εκτύπωσης σε διαφορετικά μεγέθη χαρτιών.

ΚΕΦΑΛΑΙΟ 7

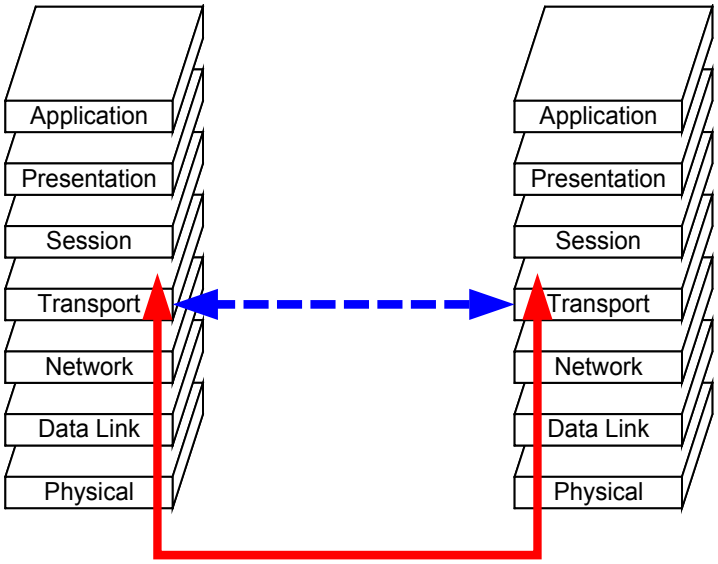
Δίκτυα Υπολογιστών

Οι τιμές πώλησης των μονάδων εισόδου / εξόδου έχουν υποστεί τον τελευταίο καιρό μια δραματική πτώση. Παλαιότερα όμως ήταν τόσο ακριβές που ήταν σύνηθες ένα υπολογιστικό κέντρο για παράδειγμα να έχει μόνο έναν εκτυπωτή τεχνολογίας λέιζερ. Η ανάγκη λοιπόν για αποκατάσταση επικοινωνίας όλων των τοπικών σταθμών με κάποιο συγκεκριμένο περιφερειακό ήταν αυτή που έδωσε το αρχικό έναυσμα για τη δημιουργία μέσων επικοινωνίας (υλικών και πρωτοκόλλων) που σήμερα χρησιμοποιούνται στα λεγόμενα δίκτυα υπολογιστών.

Ενα δίκτυο δεν είναι τίποτα περισσότερο από ένα μέσο διασύνδεσης υπολογιστικών συστημάτων, με σκοπό την διαμοίραση πληροφοριών, εφαρμογών και περιφερειακών συσκευών. Ενα δίκτυο αποτελείται από *υλικό, λογισμικό και πρωτόκολλα*. Το υλικό για παράδειγμα είναι καλώδια και κυκλώματα προσαρμογής. Το απαιτούμενο λογισμικό διασύνδεσης ενός χρήστη στο δίκτυο είναι σήμερα στις περισσότερες περιπτώσεις ρουτίνες εμφωλευμένες στο λειτουργικό σύστημα του υπολογιστή. Τα πρωτόκολλα είναι ένα σύνολο κανόνων που αφορούν το χρονισμό, τη δομή, την αλληλουχία και τον τρόπο ελέγχου ορθότητας της πληροφορίας που ανταλλάσσεται πάνω από το δίκτυο.

Στην αρχή παρουσίας των δικτύων υπολογιστών ο κάθε κατασκευαστής πρότεινε και ένα δικό του πρωτόκολλο επικοινωνίας, με αποτέλεσμα τα δίκτυα διαφορετικών κατασκευαστών να μη μπορούν να επικοινωνήσουν μεταξύ τους. Ο Διεθνής Οργανισμός Προτύπων (International Standards Organization – ISO) για

να γεφυρώσει αυτό το χάσμα πρότεινε ένα νέο καθολικό μοντέλο το οποίο καλείται OSI (Open System Interconnect). Για να δοθεί η δυνατότητα σε διάφορους κατασκευαστές να παράγουν συσκευές / λογισμικό το οποίο να ανταποκρίνεται σε ορισμένες μόνο ανάγκες επικοινωνίας, το OSI δεν είναι ένα μονολιθικό μοντέλο, αλλά μια ιεραρχία πρωτοκόλλων. Κατά το OSI η διαδικασία επικοινωνίας αποτελείται από επτά (7) επίπεδα που είναι (από το χαμηλότερο προς το υψηλότερο) : physical, data link, network, transport, session, presentation και application. Η ιεραρχία επιπέδων φαίνεται στο επόμενο σχήμα.



Το μοντέλο του OSI δε δίνει αυστηρούς ορισμούς για το πως θα γίνει η επικοινωνία. Αντίθετα, αποτελεί το σημείο αναφοράς για το πως η διαδικασία επικοινωνίας πρέπει να διαχωριστεί σε μικρότερες και απλούστερες διαδικασίες και ποια πρωτόκολλα πρέπει να αναπτυχθούν για την υλοποίηση αυτών των διαδικασιών στα διάφορα επίπεδα. Η πραγματική συμβολή του OSI είναι ότι ο τελικός χρήστης που επιθυμεί επικοινωνία με υπηρεσίες πολυπλοκότητας κάποιου επιπέδου αντιλαμβάνεται ότι αυτή συμβαίνει σαν να μην υπάρχουν κατώτερα επίπεδα (διακεκομμένη γραμμή του πιο πάνω σχήματος). Στην πραγματικότητα όμως τα πιο κάτω επίπεδα υλοποιούν τις απαραίτητες διαδικασίες για την απρόσκοπτη επικοινωνία του χρήστη και η πραγματική επικοινωνία συμβαίνει σύμφωνα με τη συμπαγή γραμμή. Με άλλα λόγια τα πιο κάτω επίπεδα αποκρύπτουν από τον τελικό χρήστη τις λεπτομέρειες της επικοινωνίας που δεν τον αφορούν και παρέχουν ένα νέο ιδεατό επίπεδο επικοινωνίας. Η διαστρωμάτωση αυτή προσφέρει επίσης ανεξαρτησία από κατασκευαστές και επιμέρους λεπτομέρειες.

Ας δούμε όμως τι υπηρεσίες καλείται να προσφέρει κάθε επίπεδο.

- Το επίπεδο εφαρμογών (Applications Layer) παρέχει επικοινωνία μεταξύ εφαρμογών. Τέτοιες εφαρμογές είναι κατανεμημένες βάσεις δεδομένων, ηλεκτρονικό ταχυδρομείο, μεταφορά αρχείων (ftp – file transfer protocol).
- Το επίπεδο παρουσίασης (Presentation Layer) ενσωματώνει διαδικασίες που διασφαλίζουν ότι οι επικοινωνούντες εφαρμογές λαμβάνουν πληροφορίες με σωστή δομή π.χ. κατά την επικοινωνία ενός little endian με ένα big endian σύστημα.
- Το επίπεδο συνεδρίας (Session Layer) είναι υπεύθυνο για την αρχή και τον τερματισμό συνεδριών μεταξύ επικοινωνούντων διεργασιών. Είναι επίσης υπεύθυνο για τον συγχρονισμό της επικοινωνίας αλλά και για να κρατάει σημεία αναφοράς έτσι ώστε διακοπείσες επικοινωνίες να μπορούν να συνεχίζουν μετά το σημείο διακοπής.
- Το επίπεδο μεταφοράς (Transport Layer) εξασφαλίζει την αξιοπιστία της επικοινωνίας. Είναι το επίπεδο που θα διασφαλίσει ότι υπάρχουν οι πόροι που απαιτούνται ώστε η μεταφορά της πληροφορίας να γίνει γρήγορα και σωστά. Το επίπεδο αυτό ιεραρχεί τις ανάγκες επικοινωνίας που του έρχονται από το επίπεδο συνεδρίας ώστε να επιτύχει το μικρότερο δυνατό χρόνο εξυπηρέτησης με το μικρότερο κόστος. Για παράδειγμα, το επίπεδο μεταφοράς μπορεί να αποφασίσει ότι μια πληροφορία θα πρέπει να διαχωριστεί σε πολλά πακέτα και κάθε πακέτο να μεταδοθεί μέσω διαφορετικών διαδρομών του δικτύου έτσι ώστε να διασφαλιστεί ένας καλός χρόνος μεταφοράς. Προσέξτε ότι το επίπεδο μεταφοράς του αποδέκτη της πληροφορίας είναι εκείνο που θα πρέπει να ανασυνθέσει τη πληροφορία ακόμα και αν τα πακέτα φτάσουν με διαφορετική σειρά.
- Το επίπεδο δικτύου (Network Layer) δρομολογεί τα δεδομένα στους διάφορους υποσταθμούς και υποδίκτυα. Το επίπεδο αυτό συνεπώς θα πρέπει να έχει σαφή γνώση της *τοπολογίας* του δικτύου, δηλαδή του δικτυώματος διασύνδεσης των σταθμών που βρίσκονται πάνω στο δίκτυο. Το επίπεδο αυτό κρατάει ενήμερο το επίπεδο μεταφοράς για την κατάσταση των διαφόρων συνδέσεων του δικτύου από πλευράς ταχύτητας, αξιοπιστίας και διαθεσιμότητας.
- Το επίπεδο ανασυγκρότησης της πληροφορίας (Data Link) διαχειρίζεται τις απευθείας συνδέσεις μεταξύ δύο συσκευών του δικτύου. Διαχειρίζεται κομμάτια πληροφορίας τα οποία ονομάζονται πλαίσια (frames) και τα οποία περιέχουν πέρα από την πληροφορία, τη διεύθυνση του παραλήπτη, πληροφορία δρομολόγησης κλπ.
- Το φυσικό επίπεδο (Physical Layer) διασφαλίζει την μετάδοση του 0 και του 1 πάνω από το φυσικό μέσο (καλώδιο, οπτική ίνα, κλπ).

Τα δίκτυα συνήθως χωρίζονται σε κατηγορίες ανάλογα με το εύρος της γεωγραφικής περιοχής που καλύπτουν, την τοπολογία τους, τον τρόπο βάσει του οποίου επιτυγχάνεται ο συγχρονισμός της μετάδοσης των δεδομένων, το φυσικό μέσο που χρησιμοποιούν, το ρυθμό μεταφοράς δεδομένων κλπ. Παρακάτω αποσαφηνίζονται ορισμένες μόνο από τις κατηγοριοποιήσεις. Στο μάθημα των Δικτύων Υπολογιστών θα έχετε την δυνατότητα να δείτε σε βάθος όλες τις παραμέτρους ενός δικτύου.

Ανάλογα με το εύρος της γεωγραφικής περιοχής που καλύπτουν τα δίκτυα κατατάσσονται σε :

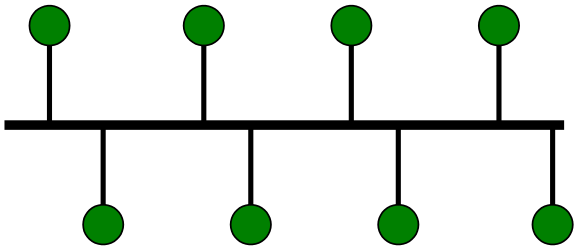
- Τοπικά (Local Area Networks – LANs) που καλύπτουν μια έκταση μερικών τετραγωνικών μέτρων ή το πολύ μερικών κοντινών κτιρίων.
- Ευρείας Περιοχής (Wide Area Networks / Long Haul Networks), που καλύπτουν πόλεις ή ολόκληρες χώρες.

Ενα από τα πλέον ευρέως χρησιμοποιούμενα μέτρα απόδοσης των δικτύων είναι ο ρυθμός μεταφοράς δεδομένων (bandwidth) που παρέχουν, ο οποίος ποικίλει σημαντικά ανάλογα με το εύρος της γεωγραφικής περιοχής κάλυψης, του μέσου που χρησιμοποιείται για τη φυσική διασύνδεση των σταθμών κλπ. Ο ρυθμός μεταφοράς δεδομένων αντικατοπτρίζει τη ποσότητα της πληροφορίας που μπορεί να μεταδοθεί πάνω από το δίκτυο στη μονάδα του χρόνου και μετριέται σε bits/sec (bps) και τα πολλαπλάσιά του, Kbps, Mbps. Προσέξτε ότι εδώ τα K, M εννοούν πολλαπλασιαστικούς παράγοντες 10^3 και 10^6 και όχι 2^{10} και 2^{20}. Πέρα όμως από την ωφέλιμη πληροφορία στο δίκτυο μεταφέρονται και άλλες πληροφορίες (συγχρονισμού, ψηφία ελέγχου της ορθότητας της πληροφορίας), οπότε μόνο ένα μέρος του bandwidth είναι πρακτικά ωφέλιμο. Διαφορετικά τμήματα του ίδιου δικτύου (segments) μπορεί να προσφέρουν διαφορετικούς ρυθμούς μεταφοράς δεδομένων. Ακόμη και στο ίδιο υπολογιστικό σύστημα μπορεί ο ρυθμός αποδοχής δεδομένων να είναι διαφορετικός αυτού της αποστολής τους (ασύμμετρη διασύνδεση - asymmetric connection).

Οι βασικές εναλλακτικές τοπολογίες που κατά καιρούς έχουν προταθεί για την δημιουργία ενός δικτύου είναι :

7.1. Τοπολογία Αρτηρίας

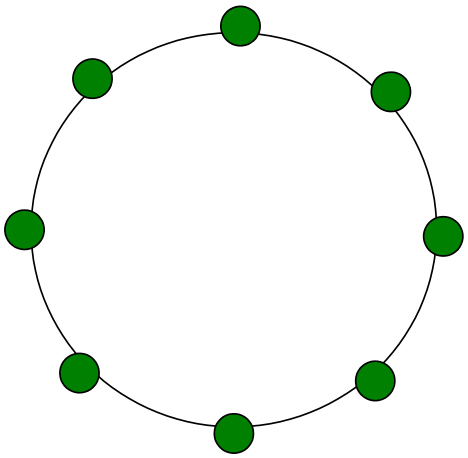
Η τοπολογία αρτηρίας φαίνεται στο παρακάτω σχήμα.



Η τοπολογία αυτή είναι η απλούστερη ανάμεσα στις προταθείσες και είναι παρόμοια με την τοπολογία διαμοιραζόμενης αρτηρίας που συνήθως υλοποιείται και εσωτερικά σε ένα υπολογιστικό σύστημα. Ενας καινούργιος σταθμός σε αυτή τη τοπολογία μπορεί εύκολα να προστεθεί με τη διασύνδεσή του πάνω στο φυσικό μέσο που αποτελεί την αρτηρία. Η τοπολογία αυτή παρουσιάζει το πλεονέκτημα ότι κάθε σταθμός του δικτύου μπορεί να επικοινωνήσει άμεσα με οποιονδήποτε άλλο σταθμό και νέοι σταθμοί μπορούν να προστίθενται εύκολα. Επιπλέον ο έλεγχος της αρτηρίας είναι σε αυτή τη τοπολογία κατανεμημένος με αποτέλεσμα το πολύ χαμηλό αρχικό κόστος για αυτή τη λύση. Η τοπολογία αυτή έχει το πρόβλημα ότι υπάρχει ένα μέγιστο όριο στο μήκος του φυσικού μέσου και ότι η πρόσθεση ενός καινούργιου σταθμού ή η βλάβη ενός σταθμού θέτει το δίκτυο εκτός λειτουργίας. Ενα παράδειγμα στο οποίο χρησιμοποιήθηκε μια τοπολογία αρτηρίας αποτελεί το δίκτυο Ethernet.

7.2. Τοπολογία Δακτυλίου

Η τοπολογία δακτυλίου φαίνεται στο παρακάτω σχήμα.

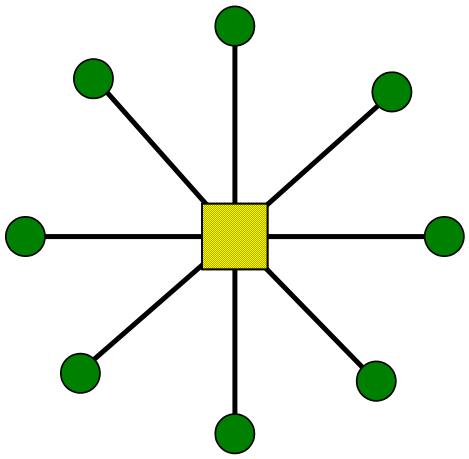


Η τοπολογία αυτή είναι αντίστοιχη με αυτήν της αρτηρίας με τη διαφορά ότι οι δύο άκρες είναι πλέον ενωμένες έτσι ώστε να σχηματίζεται ένας δακτύλιος. Τα πακέτα πληροφορίας τα οποία διακινούνται πάνω στο δίκτυο στην περίπτωση αυτή διακινούνται από τον ένα σταθμό στον διπλανό του μέχρις ότου καταλήξουν στον τελικό αποδέκτη τους. Ο τελευταίος αποσύρει το πακέτο από το δίκτυο και ο δακτύλιος αποδεσμεύεται για την επόμενη μετάδοση. Στην περίπτωση που ένα

πακέτο πληροφορίας ολοκληρώσει ένα κύκλο, τότε προφανώς ο παραλήπτης της πληροφορίας δεν μπορεί να τη δεχθεί και συνεπώς ο αποστολέας πρέπει είτε να την αποσύρει από τον δακτύλιο είτε να την τροποποιήσει σαν μια νέα μετάδοση. Η τοπολογία δακτυλίου χρησιμοποιείται ευρέως στα δίκτυα Token Ring που προτάθηκαν από την IBM.

7.3. Τοπολογία Αστέρα

Η τοπολογία αστέρα φαίνεται στο παρακάτω σχήμα.



Στην τοπολογία αυτή κάθε συσκευή του δικτύου συνδέεται σε αυτό μέσω ενός κεντρικού σταθμού (hub). Ο κεντρικός αυτός σταθμός είναι μεσάζων σε κάθε ανταλλαγή πληροφορίας μεταξύ δύο σταθμών του δικτύου. Σε μια απλουστευμένη υλοποίηση ο κεντρικός σταθμός αποστέλλει σε όλους κάθε πληροφορία που λαμβάνει και είναι πλέον ευθύνη του κάθε σταθμού να αποδεχτεί ή να απορρίψει τη εισερχόμενη πληροφορία. Σε πιο προηγμένες υλοποιήσεις όμως ο κεντρικός σταθμός αποστέλλει την πληροφορία μόνο στον πραγματικό αποδέκτη της. Το πλεονέκτημα της τοπολογίας αστέρα είναι ότι στις περισσότερες αναδιαρθρώσεις του συστήματος το βάρος πέφτει στην αναδιάρθρωση του κεντρικού σταθμού και μόνο. Προφανώς ένα πρόβλημα σε αυτό το σταθμό βγάζει εκτός λειτουργίας ολόκληρο το δίκτυο (single point of failure).

7.4. Συγχρονισμός των δικτύων

Για τον συγχρονισμό σε επίπεδο πακέτου πληροφορίας (για να αποφευχθούν ή να ελαττωθούν δηλαδή πιθανές συγκρούσεις λόγω ταυτόχρονης μετάδοσης που μπορεί να συμβούν στις τοπολογίες αρτηρίας και δακτυλίου) έχουν προταθεί διάφορα πρωτόκολλα :

7.4.1. CSMA / CD (Carrier Sense Multiple Access with Collision Detection)

Στο πρωτόκολλο αυτό οι συγκρούσεις επιτρέπονται. Οταν ένας σταθμός θελήσει να μεταδώσει δεδομένα, πρώτα ακούει το κανάλι. Οταν κάποιος άλλος ήδη μεταδίδει, τότε απαγορεύεται να μεταδώσει. Ο σταθμός περιμένει για κάποιο τυχαίο χρονικό διάστημα και ξανακούει το κανάλι. Η διαδικασία αυτή επαναλαμβάνεται μέχρις ότου ο σταθμός βρει το κανάλι ελεύθερο οπότε και αρχίζει τη δική του μετάδοση. Προσέξτε ότι και κάποιος άλλος σταθμός μπορεί ταυτόχρονα να αρχίσει μετάδοση. Στην περίπτωση αυτή έχουμε σύγκρουση. Οι σταθμοί που μεταδίδουν εκείνη τη στιγμή αντιλαμβάνονται τη σύγκρουση καθώς τα δεδομένα τους αλλοιώνονται και συνεπώς αποσύρονται και μπαίνουν εκ νέου σε κατάσταση αναμονής για μετάδοση. Το πρωτόκολλο αυτό είναι αρκετά αποδοτικό όσο οι σταθμοί του δικτύου μεταδίδουν πληροφορία λιγότερο από 35% του χρόνου. Στην αντίθετη περίπτωση η πιθανότητα για σύγκρουση αυξάνει εκθετικά. Στην τελευταία περίπτωση το πρωτόκολλο αυτό δεν μπορεί να εγγυηθεί το μέγιστο χρόνο που απαιτείται για την ορθή μετάδοση ενός κομματιού πληροφορίας.

7.4.2. Token Based

Με σκοπό την αποφυγή των συγκρούσεων η IBM εισήγαγε την έννοια του token. Το token είναι ένα ειδικό πακέτο πληροφορίας η κατοχή του οποίου δίνει τη δυνατότητα για μετάδοση. Υποθέτοντας μια τοπολογία δακτυλίου, ένας σταθμός μπορεί να μεταδώσει αν και μόνο αν κατέχει το token. Ειδεμή, απλά ακούει την πληροφορία που κυκλοφορεί στο δίκτυο και την αναμεταδίδει εφόσον πρόκειται για πληροφορία της οποίας δεν είναι αποδέκτης. Οταν ένας σταθμός θέλει να μεταδώσει, θα πρέπει να περιμένει ώστε να φτάσει σε αυτόν το token. Τότε θα το αποσύρει από το δίκτυο, θα βάλει τη πληροφορία που θέλει να μεταδώσει και μετά θα μεταδώσει και το token ώστε να επιτρέψει στον επόμενο να μεταδώσει και αυτός. Το πρωτόκολλο προβλέπει πως γεννάται το token όταν πρωτοδημιουργείται το δίκτυο και πως αναγεννιέται όταν ένας σταθμός που έχει αποσύρει το token πάθει βλάβη.

ΚΕΦΑΛΑΙΟ 8

Κώδικες για Ανίχνευση & Διόρθωση Λαθών

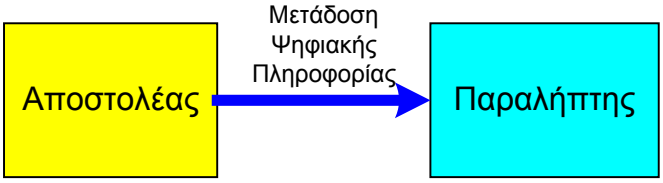
Η παρουσία λαθών στα ψηφιακά συστήματα επεξεργασίας πληροφοριών είναι κάτι που παρατηρήθηκε από τα πρώτα χρόνια εμφάνισής τους. Τα λάθη συμβαίνουν λόγω γήρανσης των μεταλλικών αγωγών, επηρεασμού από το περιβάλλον, ατελειών στην κατασκευή κλπ. Οπως θα μάθετε στα Ψηφιακά Ηλεκτρονικά στην ουσία το λάθος είναι απόρροια της ανικανότητας χειρισμού αλλαγών στις στάθμες δυναμικού πέρα από συγκεκριμένα όρια. Για να γίνει πιο κατανοητό αυτό, ας δώσουμε ένα πραγματικό παράδειγμα έχοντας κατά νου την παραδοσιακή τεχνολογία TTL.

Ενα ψηφιακό κύκλωμα της τεχνολογίας αυτής θεωρεί ως λογικό 0 οτιδήποτε βρίσκεται σε στάθμες δυναμικού 0 – 0,7 V και ως λογικό 1 οτιδήποτε βρίσκεται σε στάθμες δυναμικού 2,7 – 5 V. Υπάρχει συνεπώς μια περιοχή απαγορευμένων δυναμικών μεταξύ 0,7 και 2,7 V. Υπό κανονικές συνθήκες δεν περιμένουμε τα σήματα εισόδου του ψηφιακού μας κυκλώματος να βρεθούν εντός αυτής της περιοχής, διότι δεν ξέρουμε κατά πόσο οι αντιπροσωπευόμενες πληροφορίες θα ερμηνευτούν σωστά. Ας υποθέσουμε ωστόσο ότι μία εκ των εξόδων του ψηφιακού μας συστήματος διέρχεται μέσα από ένα πεδίο μέχρι να αποτελέσει είσοδο της επόμενης βαθμίδας επεξεργασίας. Υποθέστε ότι αυτή βρίσκεται στο λογικό 0 με δυναμικό 0,3 V. Ας υποθέσουμε ότι στιγμιαία το πεδίο μας λόγω παρεμβολών μετατρέπεται σε μαγνητικό. Η ψηφιακή γραμμή μας αυτόματα μετατρέπεται σε αγωγό εντός μαγνητικού πεδίου και συνεπώς επάγεται πάνω της κάποιο ρεύμα με αποτέλεσμα να αποκτήσει κάποιο νέο δυναμικό. Αν το

επαγόμενο ρεύμα έχει αντίθετη φορά με το αρχικό τότε το δυναμικό θα μειωθεί περαιτέρω από τα 0,3 V και συνεπώς το ψηφιακό μας σύστημα θα λειτουργήσει σωστά. Στην αντίθετη περίπτωση μπορεί είτε η επήρεια του επαγόμενου ρεύματος να είναι μικρή και το δυναμικό να μη ξεπεράσει τα 0,7 V είτε να συμβεί το αντίθετο. Στη δεύτερη περίπτωση αν το δυναμικό υπερβεί τα 2,7 V τότε το επόμενο ψηφιακό μας σύστημα θα ερμηνεύσει εντελώς ανάποδα την αρχική πληροφορία και ένα λάθος θα έχει συμβεί.

Στην παραπάνω απλουστευμένη περιγραφή θεωρήσαμε ότι ένα λάθος ισοδυναμεί με την αντιστροφή της τιμής ενός δυαδικού ψηφίου (single bit error). Αν και αυτή η θεώρηση θα μας διευκολύνει παρακάτω να διατυπώσουμε βασικές αρχές, δε συμβαδίζει πάντοτε με την πραγματικότητα. Μπορεί δηλαδή περισσότερα του ενός δυαδικά ψηφία να αλλοιωθούν (multiple bit error). Και οι δύο παραπάνω περιπτώσεις όμως απαιτούν την ανάπτυξη αποτελεσματικών μέτρων προστασίας. Το πλέον ευρύτερα διαδεδομένο όπλο κατά της δημιουργίας λαθών στα ψηφιακά συστήματα αποτελούν οι κώδικες ανίχνευσης ή / και διόρθωσης λαθών (error detection / correction codes).

Ας προσπαθήσουμε να εμβαθύνουμε λίγο περισσότερο στις ιδιότητες που πρέπει να διέπουν ένα σύστημα ώστε αυτό να έχει την ικανότητα για ανίχνευση / διόρθωση λαθών. Στο παρακάτω σχήμα φαίνεται ένα μοντέλο συστήματος που αποτελείται από έναν αποστολέα και ένα παραλήπτη κάποιας ψηφιακής πληροφορίας.



Το μοντέλο αυτό είναι αρκετά γενικό, έτσι ώστε να μπορούν να περιληφθούν κάτω από αυτό δίκτυα υπολογιστών, ΚΜΕ και μνήμη, μεταφορά πάνω από αρτηρίες, συστήματα εισόδου – εξόδου κλπ. Μπορούμε να παρατηρήσουμε ότι εάν αποστολέας – παραλήπτης ανταλλάσσουν πληροφορίες μεγέθους n δυαδικών ψηφίων και χρησιμοποιούν όλους τους 2^n συνδυασμούς, τότε δεν υπάρχει καμία δυνατότητα ανίχνευσης / διόρθωσης λαθών. Αυτό γιατί οποιαδήποτε πληροφορία και αν παράγει ο αποστολέας, όπως και αυτή να αλλοιωθεί ο παραλήπτης θα λάβει μια πληροφορία η οποία είναι καθ' όλα νόμιμη και αναμενόμενη. (Προφανώς ο παραλήπτης αναμένει κάθε δυνατή λέξη από τις 2^n, γιατί αν ο παραλήπτης ήξερε τι να αναμένει δεν υπήρχε λόγος για την αποστολή της πληροφορίας).

Η παραπάνω ανάλυση μας οδηγεί στο συμπέρασμα ότι για να υπάρχει δυνατότητα ανίχνευσης / διόρθωσης λαθών, πρέπει ο αποστολέας και ο παραλήπτης να συνεννοηθούν εξ αρχής, έτσι ώστε ένα μόνο υποσύνολο των δυνατών συνδυασμών να θεωρείται έγκυρο (σύνολο κωδικών λέξεων – set of codewords) και οι λοιποί συνδυασμοί να θεωρούνται άκυροι (non-codewords). Η ιδέα πίσω από αυτή τη συμφωνία είναι ότι ο αποστολέας παράγει μόνο κωδικές λέξεις που απουσία λαθών φτάνουν και γίνονται αποδεκτές από τον παραλήπτη, ενώ παρουσία λαθών μετατρέπονται σε μη κωδικές λέξεις που απαιτούν ειδικές ενέργειες από τον παραλήπτη. Η υλοποίηση της παραπάνω σύμβασης μπορεί να πραγματοποιηθεί επισυνάπτοντας κ επιπλέον δυαδικά ψηφία στην αρχική πληροφορία (ψηφία ελέγχου – check bits) και χαρακτηρίζοντας ορισμένους μόνο από τους 2^{n+k} συνδυασμούς ως έγκυρους. Στην ουσία αυτό οδηγεί σε μια κωδικοποίηση της αρχικής πληροφορίας αφού κάθε αρχική λέξη των n δυαδικών ψηφίων αντιστοιχίζεται σε μια νέα λέξη των n+k δυαδικών ψηφίων. Η ανάλυση αυτή καθιστά προφανές ότι η ανάγκη για ανίχνευση / διόρθωση λαθών οδηγεί το αρχικό μας σύστημα σε διάφορες επιβαρύνσεις. Ο ρυθμός μεταφοράς δεδομένων είναι πλέον χαμηλότερος ενώ χρειάζεται χρόνος στον αποστολέα για την κωδικοποίηση της πληροφορίας και στον παραλήπτη για τον έλεγχο της ορθότητας.

Υποθέστε ότι το αρχικό μας σύστημα έχει n=8 και ρυθμό μεταφοράς δεδομένων 2 Mbit/s. Αν θέσουμε k=3, τότε σε 1 sec μεταφέρονται πάλι 2Mbit, με τη διαφορά όμως ότι μόνο τα 8/11 είναι χρήσιμη πληροφορία και τα 3/11 πληροφορία ελέγχου. Συνεπώς ο ωφέλιμος ρυθμός μεταφοράς δεδομένων είναι πλέον (8/11) * 2 Mbit = 1,45 Mbit και η επιβάρυνση του συστήματός μας είναι 27%.

Τα μεγάλα ερωτήματα που καλείται η επιστήμη μας να απαντήσει είναι :

α) Πόσα είναι τα ελάχιστα ψηφία ελέγχου που απαιτούνται για την επίτευξη διαφορετικών στόχων ανίχνευσης / διόρθωσης λαθών ;

β) Μεταξύ των διαφορετικών κωδικοποιήσεων που μπορώ να πάρω με k δυαδικά ψηφία ελέγχου, ποια προσφέρει τις περισσότερες δυνατότητες ανίχνευσης / διόρθωσης λαθών, ποια οδηγεί στα πιο γρήγορα κυκλώματα κωδικοποίησης / αποκωδικοποίησης κλπ. ;

Απαντήσεις σε πολλά από αυτά τα ερωτήματα βρίσκουμε από τις εργασίες του μαθηματικού Richard Wesley Hamming (1915 – 1998), ο οποίος θεωρείται ο θεμελιωτής αυτού του κομματιού της επιστήμης μας μιας και ήταν ο πρώτος που

εισήγαγε τυπικούς τρόπους κατασκευής ορισμένων κωδίκων. Στη συνέχεια αναπτύσσουμε ορισμένες από τις έννοιες που εισήγαγε ο Hamming. Πρωταρχική είναι η έννοια της *απόστασης μεταξύ δύο ψηφιολέξεων (Hamming distance)*, ο αριθμός δηλαδή των αντίστοιχων δυαδικών ψηφίων στις οποίες αυτές διαφέρουν. Για παράδειγμα οι ψηφιολέξεις 01001 και 11101 έχουν απόσταση κατά Hamming 2, αφού διαφέρουν στο $1^{ο}$ και το $3^{ο}$ από αριστερά ψηφία. Ομοια οι ψηφιολέξεις 111 και 011 έχουν απόσταση 1. Ας υποθέσουμε τώρα ότι κατασκευάζουμε ένα κώδικα του οποίου οι κωδικές λέξεις έχουν όλες μήκος 2^{n+k} δυαδικά ψηφία. Ο Hamming ονόμασε απόσταση του κώδικα (code distance) τον ελάχιστο αριθμό των ψηφίων στα οποία διαφέρει κάθε δυνατό ζεύγος κωδικών λέξεων. Για παράδειγμα ας υποθέσουμε τον κώδικα που έχει τις εξής κωδικές λέξεις : 1111, 0011, 0000, 1101. Η κατά Hamming απόσταση μεταξύ 2 κωδικών λέξεων φαίνεται στον πιο κάτω πίνακα. Ο κώδικάς συνεπώς έχει απόσταση την ελάχιστη τιμή που εμφανίζεται στη δεξιά στήλη, δηλαδή 1.

Μεταξύ	Απόσταση
1ης και 2ης	2
1ης και 3ης	4
1ης και 4ης	1
2ης και 3ης	2
2ης και 4ης	3
3ης και 4ης	3

Ας εξετάσουμε τώρα τη φυσική έννοια αυτών των μεγεθών και ας μαθηματικοποιήσουμε ορισμένες παρατηρήσεις. Παρατηρείστε ότι *ένας κώδικας με απόσταση 1 δε μας προσφέρει καμία δυνατότητα για ανίχνευση και διόρθωση λαθών.* Ο λόγος είναι προφανής, αφού ακόμη και ένα λάθος μπορεί να οδηγήσει σε άλλη κωδική λέξη. Στον παραπάνω κώδικα εάν ο αποστολέας έστελνε την κωδική λέξη 1111 και αυτή αλλοιωνόταν στο $1^{ο}$ δυαδικό ψηφίο, τότε ο παραλήπτης θα έπαιρνε την λέξη 0111, η οποία δεν είναι λέξη του κώδικα και συνεπώς θα αντιλαμβανόταν ότι κάτι πήγε λάθος. Ωστόσο αν αλλοιωνόταν το $3^{ο}$ δυαδικό ψηφίο, ο παραλήπτης θα λάμβανε λανθασμένα την λέξη 1101, που καθόσον είναι κωδική θα γινόταν αποδεκτή.

Αν περιορίζαμε τον κώδικά μας στις λέξεις 1111, 0011, 0000, τότε αυτός θα αποκτούσε απόσταση 2. Είναι πλέον προφανές ότι ο νέος κώδικας έχει την ικανότητα ανίχνευσης απλού λάθους. Ενα απλό λάθος εφαρμοζόμενο σε οποιαδήποτε κωδική λέξη δε μπορεί να οδηγήσει σε κάποια άλλη αφού κάθε άλλη

διαφέρει σε τουλάχιστον 2 δυαδικά ψηφία. Γενικεύοντας αυτή τη λογική μπορούμε να πούμε ότι ένας κώδικας με απόσταση d+1 προσφέρει ικανότητα ανίχνευσης έως και d λαθών.

Υποθέστε τώρα ότι μεταξύ του αποστολέα και του παραλήπτη ανταλλάσσονται πλέον κωδικές λέξεις από κάποιον κώδικα με απόσταση 2. Οταν λοιπόν συμβεί ένα απλό λάθος ο παραλήπτης αγνοεί αυτή τη πληροφορία και ζητάει από τον αποστολέα να επαναλάβει την αποστολή της ίδιας πληροφορίας. Το παραπάνω σχήμα είναι αποδοτικό μόνον :
α) Ο ρυθμός εμφάνισης λαθών είναι μικρός
β) Η πληροφορία δεν είναι κρίσιμου χρόνου.
Προφανώς όταν ο ρυθμός των λαθών είναι αυξημένος στο παραπάνω σχήμα χάνεται πολύτιμος χρόνος για επαναμετάδοση πληροφοριών. Επίσης ο μέγιστος χρόνος για μια ορθή μετάδοση δεν είναι φραγμένος άνω. Σε αυτές τις περιπτώσεις είναι ίσως καλύτερο να δώσουμε στον παραλήπτη την ικανότητα να διορθώνει τυχόν λάθη. Δεδομένου του δυαδικού συστήματος, η διόρθωση των λαθών στην πραγματικότητα έγκειται στον εντοπισμό των δυαδικών ψηφίων που έχουν υποστεί αλλοίωση και στην αντιστροφή της αλλοιωμένης τιμής τους. Πως όμως μπορεί να γίνει ο εντοπισμός αυτός ;

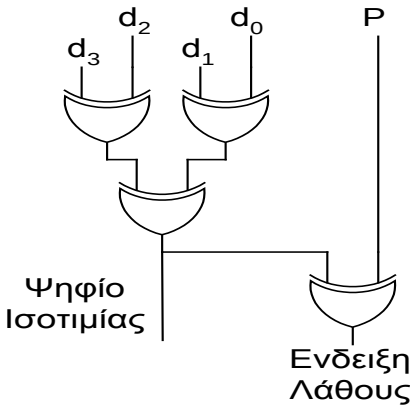
Υποθέστε και πάλι το περιορισμένο κώδικα με απόσταση 2 που χρησιμοποιήσαμε πιο πάνω. Ας υποθέσουμε ότι ο αποστολέας στέλνει την κωδική λέξη 1111 και ένα απλό λάθος την αλλοιώνει στην μη κωδική λέξη 1011. Ο παραλήπτης προφανώς αντιλαμβάνεται ότι συνέβη λάθος όμως δε μπορεί να το διορθώσει γιατί η λέξη 1011 που έφτασε σε αυτόν "μοιάζει" με δύο κωδικές λέξεις την 1111 και την 0011. Αν περιορίζαμε ακόμα περισσότερο τον κώδικά μας σε δύο μόνο κωδικές λέξεις 1111 και 0000 τότε είναι προφανές ότι το πιο πάνω λάθος σε καθεστώς απλών λαθών είναι σαφέστατα διαχωρίσιμο και συνεπώς ο παραλήπτης μπορεί πλέον να το διορθώσει. Οι παραπάνω δύο κωδικές λέξεις έχουν απόσταση 4 αλλά το ίδιο θα συνέβαινε και αν είχαν απόσταση 3. Κάθε απλό λάθος σε έναν κώδικα με απόσταση 3 οδηγεί σε μία μη κωδική λέξη που έχει απόσταση 1 από μία μόνο κωδική λέξη και 2 από κάθε άλλη. Συνεπώς ο παραλήπτης μπορεί να αναγνωρίζει αυτή τη μη κωδική λέξη σαν την κωδική λέξη εκ της οποίας απέχει το λιγότερο. Γενικεύοντας αυτή τη λογική μπορούμε να πούμε ότι ένας κώδικας με απόσταση 2c+1 προσφέρει ικανότητα διόρθωσης έως και c λαθών.

Για την εφαρμογή των παραπάνω στην πράξη, κάποιος χρειάζεται να ορίσει τον αριθμό των ψηφίων ελέγχου καθώς και τις λογικές συναρτήσεις που θα πρέπει

να υλοποιηθούν για την αύξηση της απόστασης μεταξύ των πληροφοριών που θα κωδικοποιηθούν. Προφανώς ένας κώδικας είναι εφαρμόσιμος αν οδηγεί σε απλά κυκλώματα κωδικοποίησης / αποκωδικοποίησης.

Στην πλέον συνήθη περίπτωση της ανίχνευσης απλού λάθους έχει δειχθεί ότι απαιτείται μόνο ένα επιπλέον δυαδικό ψηφίο ισοτιμίας για την επίτευξη κώδικα απόστασης 2. Το ψηφίο αυτό ονομάζεται ψηφίο ισοτιμίας (parity bit) και υπολογίζεται έτσι ώστε ο τελικός αριθμός των άσσων στην κωδική λέξη να είναι άρτιος ή περιττός ανάλογα με την επιλογή άρτιας ή περιττής ισοτιμίας. Αν η αρχική μας πληροφορία είναι η 100111011 και θελήσουμε να κατασκευάσουμε την κωδική λέξη για αυτή την πληροφορία σε έναν κώδικα άρτιας ισοτιμίας θα πρέπει να προσθέσουμε ένα δυαδικό ψηφίο ώστε η τελική λέξη να έχει άρτιο αριθμό από άσους. Αρα στο παράδειγμά μας η προκύπτουσα κωδική λέξη θα είναι : 0 | 100111011. (Στα παρακάτω θεωρούμε ότι ακολουθείται πάντα η άρτια ισοτιμία).

Η αρχική μας πληροφορία αποτελείται από n δυαδικά ψηφία και συνεπώς μπορεί να πάρει οποιαδήποτε τιμή ανάμεσα στους 2^n συνδυασμούς. Η προσθήκη ενός ψηφίου ισοτιμίας οδηγεί σε n+1 δυαδικά ψηφία. Ομως εκ των 2^{n+1} συνδυασμών μόνο οι μισοί έχουν άρτιο αριθμό από άσους, ενώ οι υπόλοιποι έχουν περιττό. Επιλέγοντας συνεπώς άρτια ισοτιμία μόνο τις μισές από τις πιθανές ψηφιολέξεις των n+1 δυαδικών ψηφίων θεωρούμε ως κωδικές. Η άρτια ισοτιμία οδηγεί σε κώδικα απόστασης 2 αφού ένα απλό λάθος μπορεί να αλλάξει είτε ένα 0 σε 1 είτε ένα 1 σε 0 και συνεπώς να οδηγήσει σε μία παράσταση με περιττό αριθμό από 1, δηλαδή σε μη κωδική λέξη. Το ψηφίο ισοτιμίας καθώς και ο έλεγχος ορθότητας μιας λέξης για τον κώδικα ισοτιμίας μπορούν να γίνουν με ένα απλό δέντρο από EX-OR πύλες. Το κύκλωμα για n=4, για αρχική πληροφορία που αποτελείται από τα ψηφία d_3, d_2, d_1 και d_0 φαίνεται παρακάτω (P το ψηφίο άρτιας ισοτιμίας). Προσέξτε ότι στο ίδιο κύκλωμα έχει ενσωματωθεί τόσο η λειτουργία κωδικοποίησης όσο και αυτή του ελέγχου της εισερχόμενης κωδικής λέξης.



Μπορούμε να προεκτείνουμε την έννοια της ισοτιμίας για την διόρθωση λαθών που συμβαίνουν σε ομάδα πληροφορίας. Υποθέστε ότι έχουμε μια ομάδα 3 πληροφοριών που κάθε μια τους είναι των 4 δυαδικών ψηφίων, σύμφωνα με το παρακάτω σχήμα :

1 0 1 0
0 1 1 1
1 1 0 1

Σε αυτήν την αρχική πληροφορία επισυνάπτουμε ένα ψηφίο (ας θεωρήσουμε άρτιας) ισοτιμίας σε κάθε στήλη και κάθε γραμμή της, οπότε παίρνουμε μια νέα κωδικοποιημένη ομάδα πληροφορίας ως εξής (έχουμε επίσης επισυνάψει ένα ψηφίο για τον έλεγχο της ισοτιμίας των ψηφίων ελέγχου):

1	0	1	0	0
0	1	1	1	1
1	1	0	1	1
0	0	0	0	0

Οριζόντια Ισοτιμία

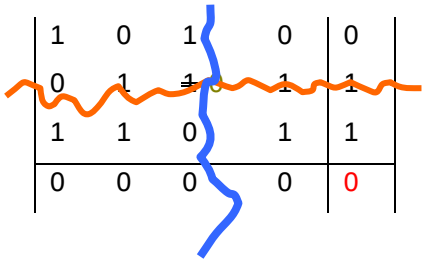
Ψηφίο Ελέγχου των Ψηφίων Ελέγχου

Κάθετη Ισοτιμία

Ας υποθέσουμε ότι στην παραπάνω κωδικοποιημένη πληροφορία συμβαίνει ένα απλό σφάλμα σύμφωνα με το παρακάτω σχήμα :

1	0	1	0	0
0	1	~~1~~ 0	1	1
1	1	0	1	1
0	0	0	0	0

Είναι προφανές ότι κάθε λάθος επηρεάζει ένα ψηφίο ισοτιμίας τόσο κατά τον κάθετο όσο και κατά τον οριζόντιο άξονα. Συνεπώς το λάθος δυαδικό ψηφίο βρίσκεται στην τομή της γραμμής και της στήλης με λάθος ισοτιμία. Το παρακάτω σχήμα δείχνει τη διαδικασία εντοπισμού του λάθους :



Από πλευράς υλοποίησης ωστόσο, η τεχνική της κάθετης και οριζόντιας ισοτιμίας απαιτεί n+k ψηφία ισοτιμίας (+1 αν υιοθετήσουμε και το ψηφίο ελέγχου των ψηφίων ελέγχου) όπου n το μήκος κάθε λέξης και k το πλήθος των λέξεων. Για κάθε ένα από αυτά απαιτείται ένα κύκλωμα σαν αυτό της προηγούμενης σελίδας, ενώ ο εντοπισμός του συγκεκριμένου δυαδικού ψηφίου απαιτεί επιπλέον υλικό.

Ο Hamming πρώτος πρότεινε βέλτιστους κώδικες για διόρθωση απλών λαθών. Η θεωρία πάνω στην οποία βασίστηκε αναπτύσσεται εν συντομία παρακάτω. Εστω ότι η αρχική πληροφορία αποτελείται από n δυαδικά ψηφία και μπορεί να πάρει όλους τους 2^n συνδυασμούς τιμών. Σε αυτήν την πληροφορία θέλω να επισυνάψω τα ελάχιστα k δυαδικά ψηφία ώστε ο παραλήπτης να μπορεί να διορθώνει απλό, διπλό, τριπλό κλπ. λάθος. Παρακάτω δίνουμε το παράδειγμα για απλό λάθος το οποίο εύκολα μπορεί να επαυξηθεί για τα περισσότερα λάθη. Η πληροφορία που φτάνει στον αποδέκτη έχει μήκος n+k δυαδικά ψηφία. Κάθε κωδική λέξη παρουσία απλών λαθών μπορεί να μετασχηματιστεί :

α) Στην αρχική κωδική λέξη αν δε συμβεί κανένα λάθος.

β) Σε n διαφορετικές λέξεις αν το λάθος συμβεί σε κάποιο δυαδικό ψηφίο της πληροφορίας

γ) Σε k διαφορετικές λέξεις αν το λάθος συμβεί σε κάποιο δυαδικό ψηφίο των ψηφίων ελέγχου.

Εστω για παράδειγμα ότι η αρχική πληροφορία είναι η $b_3\ b_2\ b_1\ b_0$ και σε αυτήν επισυνάπτονται τα ψηφία ελέγχου $c_{k-1} \dots c_0$. Η κωδική λέξη συνεπώς που σχηματίζεται είναι η $b_3\ b_2\ b_1\ b_0\ c_{k-1} \dots c_0$. Παρουσία απλών λαθών η λέξη αυτή μπορεί να αλλοιωθεί ή να μην αλλοιωθεί ως εξής :

$b_3\ b_2\ b_1\ b_0\ c_{k-1} \dots c_0$	Να μην αλλοιωθεί διόλου
$\overline{b_3}\ b_2\ b_1\ b_0\ c_{k-1} \dots c_0$	
$b_3\ \overline{b_2}\ b_1\ b_0\ c_{k-1} \dots c_0$	Να αλλοιωθεί κατά 4 τρόπους, όσους
$b_3\ b_2\ \overline{b_1}\ b_0\ c_{k-1} \dots c_0$	δηλαδή και τα ψηφία πληροφορίας
$b_3\ b_2\ b_1\ \overline{b_0}\ c_{k-1} \dots c_0$	
$b_3\ b_2\ b_1\ b_0\ \overline{c_{k-1}} \dots c_0$	Να αλλοιωθεί κατά κ τρόπους, όσους
$b_3\ b_2\ b_1\ b_0\ c_{k-1} \dots \overline{c_0}$	δηλαδή και τα ψηφία ελέγχου

Σύμφωνα με τα παραπάνω, η αρχική μορφή όσο και οι πιθανές αλλοιώσεις μιας πληροφορίας συνιστούν μια ***ομάδα*** με πληθάριθμο (1+n+k). Ο παραλήπτης της πληροφορίας για να έχει ικανότητα διόρθωσης του απλού λάθους θα πρέπει κάθε στοιχείο μιας τέτοιας ομάδας να μπορεί να το αντιστοιχεί στην αρχική πληροφορία, π.χ. η πληροφορία $b_3b_2b_1b_0c_{k-1}…c_0$ θα πρέπει να αντιστοιχεί μοναδικά στην μη αλλοιωμένη πληροφορία $b_3b_2b_1b_0c_{k-1}…c_0$. Αυτό σημαίνει ότι κάθε ομάδα θα πρέπει να είναι ένα υποσύνολο ξένο ως προς κάθε άλλη πιθανή ομάδα. Εφόσον ο αριθμός των ομάδων είναι 2^n, τα διαφορετικά στοιχεία που πρέπει να παρασταθούν είναι $2^n * (1+n+k)$. Ομως με n+k δυαδικά ψηφία μπορώ να έχω το πολύ 2^{n+k} διαφορετικές παραστάσεις. Συνεπώς :

$$2^n * (1+n+k) \leq 2^{n+k} \Leftrightarrow (1+n+k) \leq 2^k.$$

Η παραπάνω σχέση δίνει τον ελάχιστο αριθμό δυαδικών ψηφίων ελέγχου που πρέπει να προστεθούν σε n δυαδικά ψηφία πληροφορίας ώστε να υπάρχει η δυνατότητα διόρθωσης απλού λάθους. Εστω για παράδειγμα ότι n=12. Τότε η παραπάνω ανισότητα ισχύει για κ ≥ 5. Προφανώς επιλέγεται κ=5 ώστε η επιβάρυνση να είναι η μικρότερη δυνατή. Μπορείτε να συγκρίνετε τον ελάχιστο αυτό αριθμό δυαδικών ψηφίων με τα 7 (ή 8) ψηφία ελέγχου που απαιτεί η ισοτιμία στήλης γραμμής και να διαπιστώσετε πόσο αποτελεσματικός είναι ο κώδικας Hamming.

Για να γίνει όμως αποδεκτός ο παραπάνω κώδικας, απαιτείται και η εφαρμογή του να είναι εύκολη. Η εφαρμογή βασίζεται στην ιδέα των επικαλυπτόμενων ομάδων ισοτιμίας. Δηλαδή, τα ψηφία της πληροφορίας χωρίζονται σε ομάδες, που όμως δεν είναι ξένες μεταξύ τους. Κάθε ομάδα έχει ένα ψηφίο ισοτιμίας και προφανώς ένα ψηφίο πληροφορίας που ανήκει σε περισσότερες από μία ομάδες ελέγχεται από περισσότερα του ενός ψηφία ελέγχου. Ο αριθμός και η θέση των ψηφίων ελέγχου που παρουσιάζουν λάθος ισοτιμία μας εντοπίζουν το ψηφίο που είναι λανθασμένο. Ας δούμε ένα παράδειγμα. Εστω ότι έχω 4 δυαδικά ψηφία πληροφορίας $b_3b_2b_1b_0$ και συνεπώς και 3 ψηφία ελέγχου $c_3c_2c_1$. Σχηματίζω τις εξής ομάδες : (b_3, b_1, b_0, c_1), (b_3, b_2, b_0, c_2) και (b_3, b_2, b_1, c_3). Κάθε ψηφίο ελέγχου ορίζεται σαν το ψηφίο ισοτιμίας για την ομάδα που ανήκει. Εστω λοιπόν τώρα ότι αλλοιώνεται η τιμή του b_0. Ο παραλήπτης θα διαπιστώσει λάθος ισοτιμία τόσο στο ψηφίο ελέγχου c_1 όσο και στο c_2. Οι δύο ομάδες ισοτιμίας που ελέγχονται από τα ψηφία ελέγχου c_1 και c_2 έχουν κοινά στοιχεία τα b_0 και b_3. Το b_3 όμως δε μπορεί να είναι λάθος, γιατί τότε θα υπήρχε λάθος ισοτιμίας και στην ομάδα του c_3, κάτι που δεν ισχύει. Συνεπώς το λάθος εντοπίζεται στο b_0 και μπορεί να διορθωθεί. Ο παρακάτω πίνακας δείχνει ποια

ψηφία ελέγχου θα επηρεαστούν ανάλογα με τις αλλοιώσεις που μπορεί να πάθει η πληροφορία μας. Παρατηρείστε ότι το αποτέλεσμα κάθε πιθανού λάθους είναι μοναδικό, δηλαδή για κάθε περίπτωση προκύπτει ένας μοναδικός συνδυασμός λαθών στα ψηφία ελέγχου, ώστε να επιτρέπει τη διόρθωση του λάθους.

Λάθος Δυαδικό Ψηφίο	Λάθος Ψηφίο(α) Ελέγχου
b_0	C_1, c_2
b_1	C_1, c_3
b_2	C_2, c_3
b_3	c_1, c_2, c_3
c_1	c_1
c_2	c_2
c_3	c_3

Ας δούμε τώρα πως μπορούμε να δώσουμε έναν μηχανιστικό κανόνα για την κατασκευή των ομάδων ισοτιμίας και των επικαλύψεών τους. Θα χρησιμοποιήσουμε για παράδειγμα την αρχική πληροφορία M = $M_9M_8M_7M_6M_5M_4M_3M_2M_1M_0$ = 1000111011. Εφόσον n = 10, σύμφωνα με την πιο πάνω ανάλυση για την κωδικοποίηση θα χρειαστούμε 4 δυαδικά ψηφία ελέγχου, έστω τα $C_8C_4C_2C_1$. Κατασκευάζουμε τη κωδική μας λέξη, έτσι ώστε στις θέσεις που είναι δυνάμεις του 2, να είναι τα ψηφία ελέγχου. Για το παράδειγμά μας η κωδική μας λέξη θα είναι :

14	13	12	11	10	9	8	7	6	5	4	3	2	1
M_9	M_8	M_7	M_6	M_5	M_4	C_8	M_3	M_2	M_1	C_4	M_0	C_2	C_1
1	0	0	0	1	1		1	0	1		1		

Ορίζω τα ψηφία ελέγχου σαν τα ψηφία ισοτιμίας (υποθέστε άρτιας) των ψηφίων του κωδικοποιημένου μηνύματος που βρίσκονται στις θέσεις :

- Για το C_1 : 1, 3, 5, 7, 9, 11, 13, …
- Για το C_2 : 2, 3, 6, 7, 10, 11, 14, 15
- Για το C_4 : 4, 5, 6, 7, 12, 13, 14, 15, …
- Για το C_8 : 8, 9, 10, 11, 12, 13, 14, 15, …

(Εναλλακτικά, για κάθε ψηφίο πληροφορίας γράφω τη θέση που βρίσκεται σαν άθροισμα δυνάμεων του 2. Κάθε τέτοια δύναμη μου δηλώνει και τη συμμετοχή του ψηφίου αυτού στην ομάδα ισοτιμίας του ψηφίου ελέγχου που βρίσκεται στη θέση που είναι δύναμη του 2. Π.χ. το ψηφίο πληροφορίας M_6, βρίσκεται στη θέση 11.

Επειδή 11 = 8 + 2 + 1, το ψηφίο αυτό ελέγχεται από τα ψηφία ελέγχου που βρίσκονται στις θέσεις 8, 2 και 1, δηλαδή από τα C_8, C_2, C_1).
Εφαρμόζοντας την ισοτιμία βρίσκω :

- C_1 : 0
- C_2 : 0
- C_4 : 1
- C_8 : 1

Οπότε το κωδικό μήνυμα που προκύπτει είναι το εξής :

14	13	12	11	10	9	8	7	6	5	4	3	2	1
M_9	M_8	M_7	M_6	M_5	M_4	C_8	M_3	M_2	M_1	C_4	M_0	C_2	C_1
1	0	0	0	1	1	1	1	0	1	1	1	0	0

Το υλικό που χρειάζεται για την διαδικασία κωδικοποίησης του παραδείγματός μας είναι προφανώς τέσσερα αντίγραφα του υλικού που παρουσιάστηκε για τον κώδικα ισοτιμίας.

Ας υποθέσουμε ότι συμβαίνει κάποιο λάθος στο κωδικό μας μήνυμα που έχει ως αποτέλεσμα την αντιστροφή του M_4. Τότε ο παραλήπτης παίρνει το κάτωθι μήνυμα :

14	13	12	11	10	9	8	7	6	5	4	3	2	1
M_9	M_8	M_7	M_6	M_5	M_4	C_8	M_3	M_2	M_1	C_4	M_0	C_2	C_1
1	0	0	0	1	~~1~~ **0**	1	1	0	1	1	1	0	0

Ελέγχοντας την ισοτιμία των ομάδων όπως ορίστηκαν παραπάνω ο παραλήπτης παίρνει ένδειξη λάθους στην ισοτιμία των C_1 και C_8. Τα κοινά στοιχεία στις ομάδες ισοτιμίας των C_1 και C_8 είναι τα ψηφία στις θέσεις 9, 11 και 13. Λάθος στο ψηφίο της θέσης 11 ωστόσο θα προκαλούσε λάθος και στην ισοτιμία του C_2. Ομοια, λάθος στο ψηφίο της θέσης 13 θα προκαλούσε λάθος C_4. Μιας τόσο το C_2 όσο και το C_4 είναι σωστά, συμπεραίνουμε ότι λανθασμένο είναι το ψηφίο στη θέση 9 και συνεπώς το αντιστρέφουμε.

Το υλικό που χρειάζεται για την διαδικασία αποκωδικοποίησης προφανώς αποτελείται από XOR δένδρα για την δημιουργία των ενδείξεων λάθους, έναν αποπλέκτη που παίρνει τις ενδείξεις λάθους και παράγει μία μάσκα της οποίας μόνο ένα ψηφίο είναι στο 1 σε περίπτωση που κάποια από τις ενδείξεις λάθους είναι στο

λογικό 1 και μία πύλη XOR ανά δυαδικό ψηφίο που παίρνει την κωδική λέξη και τη μάσκα και παράγει τη διορθωμένη πληροφορία.

Η βασική μορφή του κώδικα Hamming που περιγράψαμε πιο πάνω είναι η πλέον ενδεδειγμένη για την διόρθωση απλών λαθών. Δυστυχώς, αν χρησιμοποιηθεί σε περιβάλλον διπλών λαθών, τότε τα διπλά λάθη διορθώνονται επίσης λανθασμένα. Με ελάχιστο επιπλέον υλικό μπορούμε να κατασκευάσουμε έναν κώδικα που αφενός διορθώνει απλά λάθη αφετέρου ανιχνεύει τα διπλά. Ο κώδικας αυτός ονομάζεται τροποποιημένος κώδικας Hamming και σχηματίζεται με την προσθήκη ενός ψηφίου ισοτιμίας για όλη την κωδική λέξη. Προκειμένου για άρτια ισοτιμία η κωδική λέξη του παραδείγματός μας σε αυτήν την περίπτωση θα ήταν :

14	13	12	11	10	9	8	7	6	5	4	3	2	1	Parity
M_9	M_8	M_7	M_6	M_5	M_4	C_8	M_3	M_2	M_1	C_4	M_0	C_2	C_1	P
1	0	0	0	1	1	1	1	0	1	1	1	0	0	0

Ο παραλήπτης της πληροφορίας μπορεί πλέον να ανιχνεύσει διπλά σφάλματα και να διορθώσει τα απλά έχοντας υπόψη του τα παρακάτω :

α) Υπάρχει ένδειξη λάθους και το επιπλέον δυαδικό ψηφίο ισοτιμίας είναι λάθος. Τότε υπάρχει απλό σφάλμα και προχωράμε στη διόρθωσή του.

β) Υπάρχει ένδειξη λάθους και το επιπλέον δυαδικό ψηφίο ισοτιμίας είναι σωστό. Τότε υπάρχει διπλό σφάλμα και πρέπει να απορρίψουμε την πληροφορία.

γ) Δεν υπάρχει ένδειξη λάθους και το επιπλέον δυαδικό ψηφίο ισοτιμίας είναι σωστό. Τότε η πληροφορία είναι σωστή.

Στα παραπάνω θεωρήσαμε την ύπαρξη μόνο απλών ή διπλών λαθών. Αν και αυτό το είδος λαθών είναι το πλέον σύνηθες στα ψηφιακά κυκλώματα τα πολλαπλά λάθη παίρνουν τη σκυτάλη στον τομέα των δικτύων ή γενικότερα των ψηφιακών τηλεπικοινωνιών. Οι πλέον διαδεδομένοι κώδικες στους τομείς αυτούς είναι οι κυκλικοί κώδικες (Cyclic Redundancy – CRC Codes). Οι κώδικες αυτοί βασίζονται στην ιδέα ότι μια δυαδική πληροφορία μπορεί να παριστάνει ένα πολυώνυμο αν εφαρμόσουμε τον κανόνα του πολυωνύμου και θέσουμε σαν ρίζα του αριθμητικού μας συστήματος το x. Ετσι η δυαδική πληροφορία 100111 μπορεί να θεωρηθεί ότι παριστάνει το πολυώνυμο :

$$1*x^5 + 0*x^4 + 0*x^3 + 1*x^2 + 1*x^1 + 1*x^0 = x^5 + x^2 + x + 1$$

Οι Lin και Costello έδειξαν ότι μπορούν να πετύχουν ανίχνευση πολλαπλών λαθών αν επισυνάψουν στην αρχική πληροφορία το υπόλοιπο της διαίρεσής της θεωρούμενης ως πολυώνυμο με ένα άλλο πολυώνυμο το οποίο ονομάζεται γεννήτορας πολυώνυμο. Οι ικανότητες ανίχνευσης που παρέχονται από αυτούς τους κώδικες περιορίζονται μόνο από το βαθμό και την "ποιότητα" του γεννήτορα πολυωνύμου. Πολλά από αυτά τα πολυώνυμα προστατεύονται από πατέντες και αποτελούν πλέον στάνταρτ τα οποία υποχρεωτικά πρέπει να υλοποιούν όλες οι συσκευές ενός δικτύου. Τους κυκλικούς κώδικες θα έχετε την ευκαιρία να γνωρίσετε αναλυτικότερα στα μαθήματα των Δικτύων Υπολογιστών και Σχεδιασμού Συστημάτων Ειδικού Σκοπού.

ΚΕΦΑΛΑΙΟ 9

Επιλεγμένες Ασκήσεις

Άσκηση 1

Πόσα δυαδικά ψηφία απαιτούνται για την αναπαράσταση ενός φυσικού αριθμού που αναπαρίσταται με κ ψηφία στο δεκαδικό σύστημα ;

Λύση

Με κ ψηφία στο δεκαδικό σύστημα μπορώ να αναπαραστήσω όλους τους αριθμούς από 0 έως $10^κ$-1, δηλαδή να έχω $10^κ$ αναπαραστάσεις. Για αρκετά μεγάλο κ, ο μεγαλύτερος αναπαριστώμενος αριθμός θα έχει τιμή περίπου $10^κ$. Η αναπαράσταση αυτού του αριθμού στο δυαδικό χρειάζεται $\log_2 10^κ$ δυαδικά ψηφία, δηλαδή περίπου 3,32*κ ψηφία.

Άσκηση 2

Αποδείξτε ότι ένας μη προσημασμένος δυαδικός Χ είναι δύναμη του 2 αν και μόνο αν το λογικό AND των ψηφίων του Χ και του Χ-1 είναι 0.

Λύση

Ορθό : κάθε αριθμός που είναι δύναμη του 2 μπορεί να γραφεί σα 0…0**1**0…0, όπου εκατέρωθεν της μονάδος υπάρχουν κ και λ ψηφία αντίστοιχα, με 0 ≤ κ, λ. Ο Χ-1 έχει τη μορφή 0…0**0**1…1, δηλαδή έχει 0 στη θέση που ο Χ έχει το μοναδικό του 1 και τα πιθανά 1 του βρίσκονται σε θέσεις που ο Χ έχει 0. Το λογικό AND αυτών των δύο ψηφιολέξεων προφανώς δίνει το 0.

Αντίστροφο : Για να υφίσταται ο Χ-1 και να είναι μη προσημασμένος δυαδικός αριθμός θα πρέπει Χ>0. Κάθε Χ μπορεί να γραφεί με τη μορφή b…b**1**0…0 όπου b∈{0,1} και ο αριθμός των 0 είναι κ, με κ ≥ 0. Ο Χ-1 τότε έχει τη μορφή b…b**0**1…1. Αφού το λογικό AND αυτών είναι 0, συνεπάγεται ότι b…b = 0…0 και συνεπώς ο Χ έχει μορφή 0…010…0. Αρα είναι δύναμη του 2.

Ασκηση 3

- Ποιο είναι το βάρος του 1 στους αριθμούς : 1000_2, 1000_8, 1000_{10}, 1000_{16} ?
- Ποιο δεκαδικό αριθμό αναπαριστά ο δεκαεξαδικός $A01B_{16}$?
- Μετατρέψτε τον δεκαεξαδικό Α5 σε δυαδικό και τον δυαδικό 11101100 σε δεκαεξαδικό.

Λύση

- Είναι αντίστοιχα 2^3, 8^3, 10^3 και 16^3.
- Τον $A*16^3 + 1*16^1 + B*16^0 = 10 * 4096 + 16 + 11 = 40987_{10}$.
- $A5_{16} = (1010)(0101)_2 = 10100101_2$.
  $11101100_2 = (1110)(1100)_2 = EC_{16}$.

Ασκηση 4

Κάθε μία από τις ακόλουθες αριθμητικές πράξεις είναι σωστή σε ένα τουλάχιστον αριθμητικό σύστημα, που έχει βάση r. Για την κάθε περίπτωση, να βρεθεί το r, ώστε η πράξη να είναι σωστή.

- 1234 + 5432 = 6666
- 41/3 = 13
- 33/3 = 11
- 23 + 44 + 14 + 32 = 223

Λύση

- $(1+5)r^3 + (2+4)r^2 + (3+3)r^1 + (4+2)r^0 = 6r^3 + 6r^2 + 6r^1 + 6r^0 = 6666_r$. Αρα ισχύει $\forall$ $r \geq 7$.
- $(4r+1) = 3(r+3) \Leftrightarrow r=8$.
- $3r+3 = 3(r+1)$. Η ισότητα αυτή ισχύει για κάθε $r \geq 4$.
- $2r + 3 + 4r + 4 + r + 4 + 3r + 2 = 2r^2 + 2r + 3 \Leftrightarrow 2r^2 – 8r – 10 = 0$. Η ισότητα αυτή ισχύει για r = 5 και r= -1. Η δεύτερη λύση απορρίπτεται.

Ασκηση 5

α) Αναπαραστήστε τους κάτωθι αριθμούς του δεκαδικού συστήματος στο δυαδικό, οκταδικό και δεκαεξαδικό σύστημα : 12345 123,1875 και 0,125

β) Αναπαραστήστε τους κάτωθι αριθμούς του δεκαδικού συστήματος στο δυαδικό σύστημα χρησιμοποιώντας αναπαράσταση πρόσημο και μέτρο, συμπλήρωμα ως προς 1 και συμπλήρωμα ως προς 2 : Α= -23, Β=45, Γ= -64

Λύση

α) Με τη μέθοδο των διαδοχικών διαιρέσεων είναι 12345 = 11000000111001_2, και 123 = 1111011_2. Με τη μέθοδο των διαδοχικών πολλαπλασιασμών είναι 0,1875 = $0,0011_2$ και 0,125 = $0,001_2$. Χωρίζοντας σε τετράδες για το δεκαεξαδικό και τριάδες για το οκταδικό τη δυαδική αναπαράσταση ξεκινώντας από την υποδιαστολή πάντα και κινούμενοι προς τα άκρα της δυαδικής αναπαράστασης παίρνουμε :

Δεκαδικό	Δυαδικό	Οκταδικό	Δεκαεξαδικό
12345	11000000111001	30071	3039
123,1875	1111011,0011	173,14	7B,3
0,125	0,001	0,1	0,2

β) Το υποερώτημα δε μας δίνει τον απαιτούμενο αριθμό ψηφίων αναπαράστασης. Υποθέτουμε παρακάτω ότι χρησιμοποιούμε 8 δυαδικά ψηφία, τον ελάχιστο δυνατό αριθμό ψηφίων δηλαδή που να καλύπτει και τους τρεις αριθμούς.

Εχουμε ότι Α = -23_{10}. Επειδή 23_{10} = 00010111_2, ο –23 είναι σε πρόσημο μέτρο 10010111, σε συμπλήρωμα ως προς 1 11101000 και σε συμπλήρωμα ως προς 2 11101001.

Ο 45_{10} = 00101101_2 και αυτή είναι η παράστασή του και στους τρεις ζητούμενους κώδικες.

Ο 64_{10} = 01000000_2. Αρα ο –64 σε πρόσημο-μέτρο είναι 11000000, σε συμπλήρωμα ως προς 1 10111111 και σε συμπλήρωμα ως προς 2 11000000.

Ασκηση 6

α) Δίδονται οι αριθμοί : Α= 11001000, Β = 10000000, Γ = 01111111, σε αναπαράσταση συμπληρώματος ως προς 2. Εκτελέστε χρησιμοποιώντας 8-bit ακρίβεια τις πράξεις Α+Β, Α-Γ, Β+Γ. Σχολιάστε θεωρητικά και πρακτικά την ορθότητα των αποτελεσμάτων.

β) Αναπαραστήστε τους Α, Β, Γ με τα ελάχιστα δυαδικά ψηφία που απαιτούνται για την ορθή εκτέλεση των παραπάνω πράξεων σε μορφή συμπληρώματος ως προς 1.

Λύση

- Είναι Α= 11001000 = -128+64+8 = -56_{10}, Β=-128_{10} και Γ=127_{10}. Επίσης –Γ = -127_{10} = 10000001 σε συμπλήρωμα ως προς 2. Α+Β = 11001000 + 10000000 = 01001000 και κρατούμενο εξόδου. Πρακτικά το αποτέλεσμα είναι λάθος γιατί -56_{10} -128_{10} = -184_{10} και όχι 72_{10} που βρήκαμε εμείς. Θεωρητικά το αποτέλεσμα είναι λάθος, γιατί υπάρχει κρατούμενο εξόδου αλλά όχι κρατούμενο προς την υψηλότερη βαθμίδα και συνεπώς υπάρχει υπερχείλιση.
- Α-Γ = Α+(-Γ) = 11001000 + 10000001 = 01001001 και κρατούμενο εξόδου. Θεωρητικά έχουμε ίδια με τη προηγούμενη περίπτωση. Πρακτικά Α-Γ = -183_{10} ενώ εμείς υπολογίσαμε ότι Α-Γ = 73_{10}.
- Β+Γ = 10000000 + 01111111 = 11111111 χωρίς κρατούμενο εξόδου. Πρακτικά το αποτέλεσμα είναι σωστό, αφού σε συμπλήρωμα ως προς 2 το 11111111 =

-128+64+32+16+8+4+2+1 = -1_{10} και το αναμενόμενο αποτέλεσμα ήταν πάλι -128+127 = -1. Θεωρητικά το αποτέλεσμα είναι σωστό αφού κατά τη πρόσθεση ετεροσήμων δε μπορεί να προκύψει υπερχείλιση.

- Με κ δυαδικά ψηφία σε συμπλήρωμα ως προς 1 αναπαριστούνται οι ακέραιοι από το $-(2^{κ-1}-1)$ έως το $2^{κ-1}-1$. Από τη παραπάνω συζήτηση φαίνεται ότι ο κατά μέτρο μεγαλύτερος αριθμός που θα πρέπει να αναπαρασταθεί είναι ο 184_{10}. Συνεπώς θα πρέπει να επιλέξω έτσι το κ ώστε $2^{κ-1}-1 \geq 184 \Leftrightarrow 2^{κ-1} \geq 185 \Leftrightarrow 2^{κ} \geq 370 \Leftrightarrow κ \geq \log_2 370 \Leftrightarrow κ \geq 8{,}53$. Αρα επιλέγω κ =9.

Ασκηση 7

Δίδονται οι αριθμοί Α=-42_{10}, Β=400_5 και Γ=52_{16}.

α) Εκφράστε τον Α σε συμπλήρωμα ως προς 1, τον Β σε κώδικα πλεονασμού κατά 64 και τον Γ σε συμπλήρωμα ως προς 2 χρησιμοποιώντας παντού ακρίβεια 8 δυαδικών ψηφίων.

β) Εκτελέστε τις πράξεις Β+Γ σε συμπλήρωμα ως προς 1 και Α-Β σε συμπλήρωμα ως προς 2, χρησιμοποιώντας 8 ή 9 δυαδικά ψηφία. Σχολιάστε την ορθότητα των αποτελεσμάτων.

γ) Εκτελέστε την παρακάτω πρόσθεση, γράφοντας τα δυαδικά ψηφία του αποτελέσματος και πάνω από κάθε τόξο σε ΔΕΚΑΔΙΚΗ ΜΟΡΦΗ τα κρατούμενα προς τις επόμενες βαθμίδες,

1	0	1	1	0
1	0	1	0	1
0	0	1	1	0
0	1	1	1	1 +

↙ ↙ ↙ ↙ ↙ ↙

Λύση

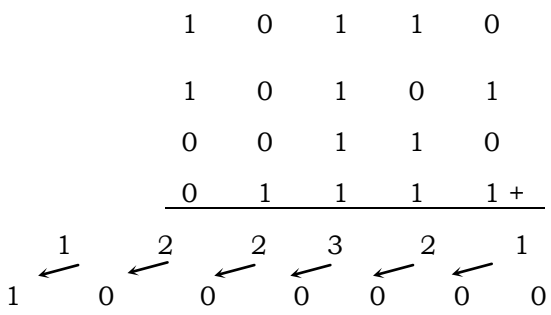
α) Αφού $42_{10} = 00101010_2$, $-42_{10} = 11010101_{1's}$. $Β=400_5 = 4*5^2 = 100_{10} = 01100100_2$. Σε κώδικα πλεονασμού κατά 64 ο Β θα έχει τη δυαδική αναπαράσταση του 164_{10} δηλαδή την 10100100_2. Τέλος $Γ = 52_{16} = 01010010_2$. Αφού ο αριθμός είναι θετικός, θα έχει και την ίδια αναπαράσταση σε κώδικα συμπληρώματος ως προς 2.

β) Είναι $Β = 01100100_{1's}$ και $Γ = 01010010_{1's}$. Κάνοντας τη πράξη Β+Γ βρίσκουμε $10110110_{1's} = -127 + 32 + 16 +4 +2 = -73_{10}$. Προφανώς λόγω υπερχείλισης το αποτέλεσμα είναι λανθασμένο αφού η πρόσθεση δύο θετικών μας έδωσε αρνητικό αποτέλεσμα. Αν κάναμε τη πράξη Β+Γ χρησιμοποιώντας 9 δυαδικά ψηφία θα είχαμε : $Β+Γ=001100100_{1's} + 001010010_{1's} = 010110110_{1's} = 128 + 32 + 16 + 4 + 2 = 182_{10}$, που είναι και το σωστό αποτέλεσμα.

Το Α-Β είναι ίσο με Α+(-Β). $A=-42_{10} = 11010110_{2's}$ και $–B = -100_{10} = 10011100_{2's}$. Αρα έχουμε ότι $A-B = 11010110_{2's} + 10011100_{2's} = 01110010_{2's} = 114_{10}$. Και πάλι το αποτέλεσμα είναι λανθασμένο καθώς προσθέτουμε δύο αρνητικούς αριθμούς και το αποτέλεσμά μας είναι θετικός. (Αυτό επίσης μπορεί να διαπιστωθεί από την ανισότητα του κρατουμένου εξόδου που είναι στο 1 με το κρατούμενο προς την υψηλότερη βαθμίδα που είναι 0). Με 9 δυαδικά ψηφία θα είχαμε $A-B = 111010110_{2's} + 110011100_{2's} = 101110010_{2's} = -256 + 64 + 32 + 16 + 2 = -142_{10}$, που είναι και το σωστό αποτέλεσμα.

Αντίστοιχα συμπεράσματα μπορούμε να εξάγουμε αναλογιζόμενοι ότι με n δυαδικά ψηφία μπορούμε να αναπαραστήσουμε στα συμπληρώματα ως προς 1 και ως προς 2 τους ακεραίους των διαστημάτων $(-2^{n-1},2^{n-1})$ και $[-2^{n-1},2^{n-1})$ αντίστοιχα.

γ) Για την άθροιση, πρέπει σε κάθε στήλη ουσιαστικά να μετράμε τις μονάδες. Από τη δυαδική αναπαράσταση του αριθμού που βρίσκουμε, το τελευταίο ψηφίο είναι αυτό του αθροίσματος, ενώ τα υπόλοιπα είναι η δυαδική αναπαράσταση των κρατουμένων. Τη δεκαδική τιμή αυτών χρειάζεται να γράψουμε πάνω από τα τόξα.



Ασκηση 8

Εστω οι αριθμοί :

- $A = -56_{10}$
- $B = -114_{10}$
- $Γ = 3F_{16}$

Χρησιμοποιώντας αναπαράσταση συμπληρώματος ως προς 2 και ακρίβεια 8 δυαδικών ψηφίων εκτελέστε τις πράξεις Α+Β, Α-Γ, Β+Γ.

1. Σχολιάστε και δικαιολογείστε την ορθότητα ή μη των αποτελεσμάτων.
2. Ποια είναι τα ελάχιστα δυαδικά ψηφία που απαιτούνται για την ορθή εκτέλεση των παραπάνω πράξεων ?

Λύση

- Εχουμε ότι $A=-56_{10}$. Σε 8 δυαδικά ψηφία ο 56_{10} έχει αναπαράσταση 00111000_2 και συνεπώς η αναπαράσταση του -56_{10} σε συμπλήρωμα ως προς 2 είναι $11000111_2 + 1 =$

$11001000_{2's}$.

- Ομοια βρίσκουμε ότι Β = $10001110_{2's}$.
- Τέλος Γ = $3F_{16}$ = 00111111_2 = $00111111_{2's}$ = 63_{10}. Επίσης –Γ = $11000001_{2's}$

➢ Για την πράξη Α+Β παίρνουμε : 11001000 + 10001110 = 01010110 και κρατούμενο εξόδου. Το αποτέλεσμα είναι προφανώς λανθασμένο αφού με την πρόσθεση δύο αρνητικών αριθμών (το πιο σημαντικό ψηφίο των προσθετέων είναι 1) πήραμε ως αποτέλεσμα θετικό αριθμό (το πιο σημαντικό ψηφίο του αποτελέσματος είναι 0). Ενώ δηλαδή αναμέναμε το Α+Β = -56_{10} - 114_{10} = -170_{10}, βρήκαμε 01010110_2 = 86_{10}. Το λάθος οφείλεται σε υπερχείλιση. Πιο συγκεκριμένα, το κρατούμενο προς τη τελευταία βαθμίδα (κ_6) είναι 0 ενώ το κρατούμενο εξόδου (κ_7) είναι 1.

Για τη σωστή εκτέλεση της παραπάνω πράξης απαιτούνται τόσα ψηφία ώστε να μπορεί να αναπαρασταθεί ασφαλώς το αποτέλεσμα. Αφού με n ψηφία μπορώ στο συμπλήρωμα ως προς 2 να αναπαραστήσω το διάστημα των ακεραίων $[-2^{n-1}, 2^{n-1})$, για την αναπαράσταση του -170_{10} θα χρειαζόταν ακρίβεια 9 δυαδικών ψηφίων.

➢ Α – Γ = $11001000_{2's}$ + $11000001_{2's}$ = $10001001_{2's}$ = -119_{10} και κ_7 =1. Το αποτέλεσμα είναι σωστό αφού Α – Γ = -56_{10} - 63_{10} = - 119_{10}. Επίσης $\kappa_6 \oplus \kappa_7$=1 ⊕ 1 = 0 και συνεπώς δεν υπάρχει υπερχείλιση.

➢ Β + Γ = $10001110_{2's}$ + $00111111_{2's}$ = $11001101_{2's}$ = -51_{10} και κ_7=0. Το αποτέλεσμα είναι σωστό αφού Β + Γ = -114_{10} + 63_{10} = -51_{10}. Επίσης $\kappa_6 \oplus \kappa_7$=0 ⊕ 0 = 0 και δεν υπάρχει υπερχείλιση. Επίσης μπορούμε να πούμε ότι στην περίπτωση αυτή δεν μπορεί να υπάρξει πρόβλημα υπερχείλισης αφού προστίθενται ετερόσημοι αριθμοί.

Ασκηση 9

Δώστε την αναπαράσταση του string :

Lucky 52

στο κώδικα ASCII χρησιμοποιώντας τις δεκαεξαδικές αντιστοιχίσεις.

Λύση

Η άσκηση αυτή έχει δύο ενδιαφέροντα σημεία. Το πρώτο είναι ότι το κενό είναι ένας χαρακτήρας και συνεπώς χρειάζεται και η δική του αναπαράσταση. Το δεύτερο είναι ότι το 52 θα αναπαρασταθεί σα δύο χαρακτήρες, το 5 και το 2. Προσέξτε ότι η αναπαράσταση του 5 και του 2 σα χαρακτήρες δεν έχει καμία σχέση με την αριθμητική δυαδική αναπαράστασή τους. Αρα η ζητούμενη αναπαράσταση ανατρέχοντας σε οποιοδήποτε πίνακα του ASCII είναι :

4C 75 63 6B 79 20 35 32

Ασκηση 10

- Δίδονται οι A = -11_{10} και B = 23_{10}. Εκφράστε τους A, B σε συμπλήρωμα ως προς 2, χρησιμοποιώντας 6 δυαδικά ψηφία. Εκτελέστε το πολλαπλασιασμό A*B. Πόσα είναι τα μη μηδενικά γινόμενα και πόσες προσθέσεις απαιτούνται ?
- Εκφράστε το B σε αρχική και τελική κωδικοποίηση κατά Booth.
- Με βάση τη τελική κωδικοποίηση, γράψτε τα μερικά γινόμενα που γεννιούνται κατά το πολλαπλασιασμό A* $B_{recoded}$, και εξάγετε το τελικό αποτέλεσμα. Πόσα είναι τώρα τα μη μηδενικά μερικά γινόμενα, πόσες προσθέσεις απαιτούνται και ποιο το όφελος ?

Λύση

- Είναι A = -11_{10}. Αφού ο 11_{10} σε 6 δυαδικά ψηφία είναι ο 001011, βρίσκουμε ότι A = $110101_{2's}$. B = $010111_{2's}$. Για τον ορθό πολλαπλασιασμό του A με το B, αφού είναι προσημασμένοι, θα πρέπει να κάνουμε επέκταση προσήμου κάθε μερικού γινομένου μέχρι του μήκους του αποτελέσματος δηλαδή των 11 δυαδικών ψηφίων. Αρα έχουμε :

					1	1	0	1	0	1	
					0	1	0	1	1	1	x
1	1	1	1	1	1	1	0	1	0	1	
1	1	1	1	1	1	0	1	0	1		
1	1	1	1	1	0	1	0	1			
1	1	1	0	1	0	1					+
1	1	1	0	0	0	0	0	0	1	1	

  Τα μη μηδενικά μερικά γινόμενα είναι 4 και οι προσθέσεις που απαιτούνται είναι 3.
- Στην αρχική κωδικοποίηση κατά Booth το B εκφράζεται σαν (+1)(-1)(+1)00(-1). Η τελική κωδικοποίηση είναι (+1)(+2)(-1), όπου κάθε ψηφίο έχει βαρύτητα 2^{2i} ή με άλλα λόγια B=(+1)(+2)(-1) = $(+1)2^4+(+2)*2^2+(-1)2^0 = 2^4+2^3-2^0$.
- Τα γινόμενα που πλέον γεννιούνται είναι τα A*B = A * $(2^4+2^3-2^0)$= $A*2^4+A*2^3$-A, δηλαδή τα :

1	1	1	0	1	0	1	0	0	0	0	$A*2^4$
1	1	1	1	0	1	0	1	0	0	0	$A*2^3$
0	0	0	0	0	0	0	1	0	1	1+	-A
1	1	1	0	0	0	0	0	0	1	1	

  Τα μη μηδενικά μερικά γινόμενα είναι σε αυτή τη περίπτωση 3. Απαιτούνται 2 προσθέσεις και το όφελος είναι 33% γρηγορότερος πολλαπλασιασμός.

Ασκηση 11

Ο πολλαπλασιαστής Β σε μια πράξη πολλαπλασιασμού είναι ο 131_6.

α) Εκφράστε τον αριθμό στο δυαδικό. Πόσα μη μηδενικά μερικά γινόμενα θα γεννηθούν κατά τη πράξη Α*Β και πόσες αθροίσεις θα χρειαστούν ?

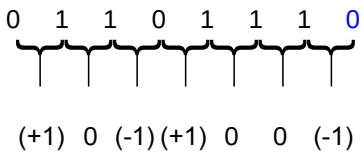
β) Κωδικοποιείστε τον Β κατά Booth. Πόσα γινόμενα θα παραχθούν σε αυτή τη περίπτωση ? Πόση η % βελτίωση ?

γ) Επαναλάβετε το (β) για τον τροποποιημένο αλγόριθμο Booth.

Λύση

α) Είναι $131_6 = 1*36 + 3*6 + 1 = 55_{10} = 0110111_2$. Για κάθε δυαδικό ψηφίο του πολλαπλασιαστή που είναι 1, θα γεννηθεί κι ένα μη μηδενικό μερικό γινόμενο. Αρα θα έχουμε 5 μερικά γινόμενα. Αυτά θα χρειαστούν 4 προσθέσεις για να πάρουμε το τελικό αποτέλεσμα.

β) Υποθέτουμε ένα 0 δεξιότερα της προηγούμενης αναπαράστασης και εξετάζουμε διαδοχικά επικαλυπτόμενα ζεύγη ψηφίων από δεξιά προς αριστερά. Κάθε μετάβαση από το 0 στο 1 μας δίνει (-1), κάθε μετάβαση από 1 σε 0 μας δίνει (+1) ενώ κάθε άλλος συνδυασμός μας δίνει 0. Αρα η κωδικοποίηση που παίρνουμε είναι :



Ο πολλαπλασιαστής δηλαδή κωδικοποιείται σαν

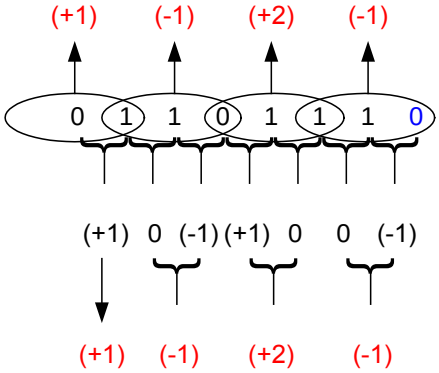
$$(+1)*2^6 + (-1)*2^4 + (+1)*2^3 + (-1)*2^0 = 64 - 16 + 8 - 1 = 55_{10}.$$

Τα μη μηδενικά μερικά γινόμενα είναι 4 και θα χρειαστούν 3 προσθέσεις. Εχουμε συνεπώς μια βελτίωση στη πράξη του πολλαπλασιασμού κατά 25%.

(Σημείωση : σε συμπλήρωμα ως προς 2 η πρόσθεση και η αφαίρεση έχουν ακριβώς την ίδια πολυπλοκότητα. Για παράδειγμα ο όρος $(-1)*2^4$ του πολλαπλασιαστή θα υλοποιηθεί με 4 αριστερές ολισθήσεις του πολλαπλασιαστέου και τη πρόσθεση του συμπληρώματος ως προς 2 της παράστασης που θα προκύψει).

γ) Για να βρούμε τη κωδικοποίηση του πολλαπλασιαστή στον τροποποιημένο αλγόριθμο Booth, μπορούμε είτε να ξεκινήσουμε από την αρχική αναπαράσταση είτε από την αναπαράσταση που βρήκαμε στο προηγούμενο ερώτημα. Ξεκινώντας από την αρχική, υποθέτουμε ένα επιπλέον 0 δεξιότερα της αναπαράστασης και χωρίζουμε τον αριθμό σε επικαλυπτόμενες τριάδες ψηφίων από δεξιά προς τα αριστερά. Βασιζόμενοι στον πίνακα αντικατάστασης, αντικαθιστούμε τέλος κάθε τριάδα με την ισοδύναμη αναπαράστασή της. Ξεκινώντας από την αναπαράσταση του απλού αλγορίθμου Booth αντικαθιστούμε κάθε δυάδα ψηφίων που συναντάμε διατρέχοντας την αναπαράσταση από δεξιά προς τα

αριστερά με ένα ψηφίο της καινούργιας αναπαράστασης. Ανεξάρτητα του ποιον τρόπο επιλέξουμε θα καταλήξουμε στην εξής τελική κωδικοποίηση :



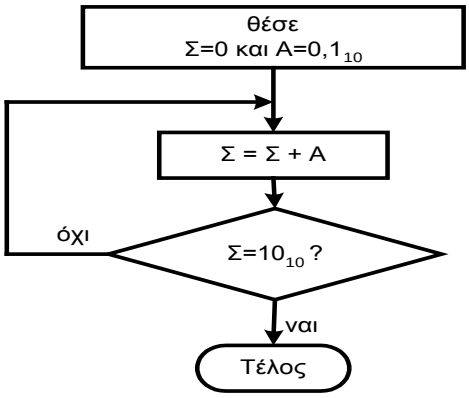
όπου ο πολλαπλασιαστής έχει εκφραστεί με ψηφία διπλάσιας βαρύτητας ως :

$$(+1)*2^{2*3} + (-1)*2^{2*2} + (+2)*2^{2*1} + (-1)*2^{2*0} = 64 - 16 + 8 - 1 = 55_{10}.$$

Και πάλι υπάρχουν 4 μη μηδενικοί όροι και συνεπώς θα γεννηθούν αντίστοιχα μερικά γινόμενα και θα χρειαστούν 3 προσθέσεις.

Ασκηση 12

Να περιγράψετε τι θα γίνει κατά την εκτέλεση κάποιου προγράμματος που υλοποιεί το κάτωθι λογικό διάγραμμα. Θεωρείστε ότι ο υπολογιστής στον οποίο θα εκτελεστεί το πρόγραμμα υποστηρίζει μόνο το δυαδικό σύστημα αναπαράστασης αριθμών. Δικαιολογείστε την απάντησή σας.



Λύση

Αφού ο υπολογιστής μας υποστηρίζει μόνο το δυαδικό σύστημα αναπαράστασης, η αναπαράσταση του $A=0,1_{10}$ θα γίνει χρησιμοποιώντας το δυαδικό σύστημα. Επομένως πρέπει να βρούμε την δυαδική παράσταση του δεκαδικού αριθμού 0,1. Αυτό θα γίνει με διαδοχικούς πολλαπλασιασμούς με την βάση του δυαδικού συστήματος που είναι το 2. Για να πάρουμε μία ακριβή παράσταση του δεκαδικού αριθμού 0,1 στο δυαδικό θα πρέπει να κάνουμε τόσους πολλαπλασιασμούς μέχρι το κλασματικό μέρος που θα προκύψει να είναι μηδέν.

Βήμα 1. α_{-1} = ακέραιο μέρος του 2x0,1 = 0 και κλασματικό =0,2
Βήμα 2. α_{-2} = ακέραιο μέρος του 2x0,2 = 0 και κλασματικό =0,4
Βήμα 3. α_{-3} = ακέραιο μέρος του 2x0,4 = 0 και κλασματικό =0,8
Βήμα 4. α_{-4} = ακέραιο μέρος του 2x0,8 = 1 και κλασματικό =0,6
Βήμα 5. α_{-5} = ακέραιο μέρος του 2x0,6 = 1 και κλασματικό =0,2
Βήμα 6. α_{-6} = ακέραιο μέρος του 2x0,2 = 0 και κλασματικό =0,4
Βήμα 7. α_{-7} = ακέραιο μέρος του 2x0,4 = 0 και κλασματικό =0,8

Παρατηρούμε μία περιοδικότητα στα διαδοχικά κλασματικά μέρη που προκύπτουν: 0,2, 0,4, 0,8, 0,6 και πάλι 0,2, 0,4, 0,8 κλπ. Επομένως δεν θα πάρουμε ποτέ κλασματικό μέρος ίσο με μηδέν, άρα δεν είναι δυνατόν να παραστήσουμε τον δεκαδικό αριθμό 0,1 ακριβώς στο δυαδικό σύστημα με πεπερασμένο αριθμό δυαδικών ψηφίων. Αυτό θα έχει σαν αποτέλεσμα το άθροισμα Σ που προκύπτει από τις διαδοχικές προσθέσεις του Α ποτέ να μην πάρει ακριβώς την τιμή 10_{10}. Αρχικά θα είναι μικρότερο και στη συνέχεια θα γίνει μεγαλύτερο του 10_{10}, οπότε δεν θα σταματήσει η εκτέλεση του προγράμματος.

Η άσκηση αυτή ας προβληματίσει σοβαρά όσους ισχυρίζονται ότι μπορούν να γίνουν καλοί προγραμματιστές αν αγνοούν τη δομή και οργάνωση του υπολογιστή ή τον τρόπο αναπαράστασης της πληροφορίας στον υπολογιστή.

Ασκηση 13

Εξηγείστε τις έννοιες compiler, interpreter, assembler.

Λύση

Και οι τρεις έννοιες αναφέρονται σε λογισμικό συστήματος που έχει ως σκοπό τη μετάφραση κώδικα από κάποια γλώσσα προγραμματισμού σε εκτελέσιμο κώδικα. Οι compilers και interpreters παίρνουν σαν είσοδο κώδικα υψηλών γλωσσών προγραμματισμού, ενώ ο assembler κώδικα γλώσσας Assembly, της υψηλότερης γλώσσας που μας επιτρέπει να έχουμε απόλυτο έλεγχο στο υλικό του συστήματός μας. Η διαφορά μεταξύ των compilers (μεταφραστών) και interpreters (διερμηνευτών) έγκειται στο ότι οι μεν πρώτοι μεταφράζουν μονομιάς όλο τον πηγαίο κώδικα και παράγουν ένα εκτελέσιμο αρχείο, ενώ οι δεύτεροι μεταφράζουν μία-μία τις γραμμές του πηγαίου κώδικα εκτελώντας τις παράλληλα. Τα σύγχρονα εργαλεία ανάπτυξης λογισμικού συνενώνουν αυτές τις δύο φιλοσοφίες. Εχοντας γράψει ο προγραμματιστής την πρώτη έκδοση του προγράμματός του, τον εκτελεί γραμμή - γραμμή για να διαπιστώσει τυχόν λάθη. Οταν πλέον έχει αποσφαλματώσει πλήρως τον κώδικά του καλεί τον μεταφραστή για να πάρει το αρχείο του εκτελέσιμου κώδικα. Ο assembler (συμβολομεταφραστής) είναι ένας μεταφραστής της συμβολικής γλώσσας, που όντας πολύ πιο κοντά στον κώδικα μηχανής είναι σχετικά απλό πρόγραμμα να κατασκευαστεί, αφού χρειάζεται μόνο να αντικαθιστά τις συμβολικές εντολές με τα πραγματικά opcodes, τα ονόματα καταχωρητών με τις πραγματικές τους διευθύνσεις και τις σταθερές με τα δυαδικά τους ισοδύναμα.

Ασκηση 14

Εξηγείστε το γιατί χρειάζεται η ιεραρχία μνήμης.

Λύση

Από τη μνήμη του συστήματός μας θέλουμε :

1. Να είναι γρήγορη.
2. Να είναι πολύ μεγάλη.
3. Να είναι φτηνή.

Οι διατάξεις μνήμης που κυκλοφορούν στην αγορά όμως διέπονται από τα εξής :

4. Οι γρήγορες μνήμες είναι πολύ ακριβές.
5. Οι γρήγορες μνήμες είναι πολύ μικρές.

Η ιεραρχία μνήμης έρχεται να γεφυρώσει τις αντιθέσεις που υπάρχουν μεταξύ των όσων ισχύουν στον πραγματικό κόσμο και των όσων θα θέλαμε να ισχύουν.

Ασκηση 15

Ενας υπολογιστής έχει 8192 διαφορετικές διευθυνσιοδοτούμενες θέσεις μνήμης. Πόσα δυαδικά ψηφία έχει η αρτηρία διευθύνσεών του ; Πόση είναι η μέγιστη μνήμη του υπολογιστή (εκφρασμένη κατά περίπτωση σε KB, MB ή GB) όταν :

- Κάθε θέση μνήμης είναι ένα byte ;
- Κάθε θέση μνήμης είναι μία λέξη των 32 δυαδικών ψηφίων ;

Λύση

Για την παραγωγή 8192 διαφορετικών διευθύνσεων, απαιτούνται $\lceil \log_2 8192 \rceil$ δυαδικά ψηφία. Αρα η αρτηρία διευθύνσεών του έχει 13 δυαδικά ψηφία. Οταν σε κάθε θέση αποθηκεύεται ένα byte η συνολική μνήμη είναι 8192 bytes = 8 *1024 bytes = 8 $*2^{10}$ bytes = 8 Kbytes. Οταν αποθηκεύονται σε κάθε θέση 32 δυαδικά ψηφία δηλαδή 4 $=2^2$ bytes, η μνήμη του συστήματός μας είναι $2^2 * 2^3 * 2^{10}$ bytes = 32Kbytes.

Ασκηση 16

Σχεδιάζετε ένα υπολογιστικό σύστημα που θέλετε να διαθέτει κ διαφορετικές εντολές (π.χ. ADD, SUB, ...), που κάθε μία τους μπορεί να έχει 2 ή 3 operands, και κάθε operand μπορεί να έχει λ διαφορετικούς τρόπους διευθυνσιοδότησης. Δώστε τον κλειστό τύπο των ελάχιστων δυαδικών ψηφίων που θα πρέπει να χρησιμοποιήσετε για τους κωδικούς λειτουργίας (opcodes) του συστήματός σας.

Λύση

Η άσκηση αυτή έχει σα στόχο να σας προβληματίσει για το πότε χρειάζεται ένα καινούργιο opcode. Η απάντηση είναι ότι κάτι τέτοιο χρειάζεται αν απαιτείται η εκτέλεση διαφορετικού

μικροπρογράμματος για την εκτέλεση αυτών των εντολών. Οι εντολές με 2 και με 3 operands προφανώς απαιτούν διαφορετικούς κωδικούς λειτουργίας, αφού στη δεύτερη περίπτωση χρειάζεται μια επιπλέον προσπέλαση μνήμης. Ας δούμε ένα παράδειγμα για να καταλάβουμε αν απαιτείται διαφορετικό μικροπρόγραμμα ανάλογα με το τρόπο διευθυνσιοδότησης. Έστω οι εντολές :

LDB 80, [90]

LDB 80, #90

Ενώ η δεύτερη εντολή θα κάνει μόνο μια προσπέλαση στη μνήμη για το καθορισμό του τελικού τελουμένου που θα λάβει χώρα στην εντολή, η πρώτη θα πρέπει να κάνει δύο. Αυτό οδηγεί σε διαφορετικά μικροπρογράμματα και συνεπώς διαφορετικούς opcodes.

Σε κάθε εντολή δύο διευθύνσεων υπάρχουν λ τρόποι διευθυνσιοδότησης του πρώτου operand και για καθέναν από αυτούς λ διαφορετικοί για το δεύτερο. Αρα συνολικά για κάθε εντολή δύο διευθύνσεων υπάρχουν λ^2 διαφορετικοί τρόποι διευθυνσιοδότησης των εντέλων και τελικά κ λ^2 διαφορετικά μικροπρογράμματα. Με ακριβώς το ίδιο σκεπτικό μπορούμε να βρούμε ότι θα χρειαστούμε κ λ^3 διαφορετικούς κωδικούς λειτουργίας για τις εντολές τριών διευθύνσεων. Οπότε οι διαφορετικοί opcodes είναι κ ($\lambda^2 + \lambda^3$) και τα ελάχιστα δυαδικά ψηφία που θα πρέπει να αφιερώσουμε για κωδικό λειτουργίας στην εντολή μας είναι :

$$z = \lceil \log_2 (\kappa\lambda^2 + \kappa\lambda^3) \rceil$$

Ασκηση 17

- Η αρτηρία διευθύνσεων ενός υπολογιστικού συστήματος έχει εύρος 32 δυαδικών ψηφίων. Αν σε κάθε θέση μνήμης μπορούμε να αποθηκεύουμε πληροφορία 16 δυαδικών ψηφίων, ποια είναι η μέγιστη φυσική μνήμη που μπορεί να έχει αυτό το σύστημα ?
- Ενα υπολογιστικό σύστημα διαθέτει κ διαφορετικές εντολές (π.χ. ADD, LOAD, STORE, …). Σε κάθε εντολή, ένα τελούμενο μπορεί να διευθυνσιοδοτείται με λ διαφορετικούς τρόπους.
  1. Ποιος είναι ο ελάχιστος αριθμός δυαδικών ψηφίων της εντολής που θα πρέπει να αφιερωθεί για κωδικό λειτουργίας και ποιος για το τρόπο διευθυνσιοδότησης ?
  2. Προτείνετε εναλλακτική κωδικοποίηση αυτών των δύο πεδίων που να ελαχιστοποιεί τα χρησιμοποιούμενα δυαδικά ψηφία της εντολής.

Λύση

- Σε κάθε θέση μνήμης μπορούμε να αποθηκεύουμε 16 δυαδικά ψηφία ή αλλιώς 2 bytes. Οι διαφορετικές θέσεις φυσικής μνήμης που μπορούμε να έχουμε είναι όλοι οι πιθανοί συνδυασμοί των ψηφίων της αρτηρίας διευθύνσεων δηλαδή 2^{32}. Αρα η μέγιστη φυσική μνήμη στο σύστημά μας είναι 2×2^{32} bytes = $2^3 \times 2^{30}$ bytes = 8 GB.
- 1. Αν διαθέταμε ξεχωριστά πεδία στην εντολή μας για κωδικό λειτουργίας και τρόπο

διευθυνσιοδότησης θα χρειαζόμασταν $x = \lceil \log_2 \kappa \rceil$ και $y = \lceil \log_2 \lambda \rceil$ δυαδικά ψηφία αντίστοιχα. Ο συνολικός αριθμός αυτών είναι $A = x + y = \lceil \log_2 \kappa \rceil + \lceil \log_2 \lambda \rceil$. Για παράδειγμα, αν υποθέσουμε ότι κ=25 και λ=5, θα χρειαζόμασταν 5+3=8 δυαδικά ψηφία.

2. Αφού ισχύει ότι $\kappa \le 2^x$ και $\lambda \le 2^y$, πολλαπλασιάζοντας κατά μέλη παίρνουμε ότι $\kappa\lambda \le 2^{x+y}$ και συνεπώς $\lceil \log_2 (\kappa\lambda) \rceil \le x+y \Leftrightarrow \lceil \log_2 (\kappa\lambda) \rceil \le \lceil \log_2 \kappa \rceil + \lceil \log_2 \lambda \rceil$. Συνεπώς ένας τρόπος ελαχιστοποίησης των δυαδικών ψηφίων που χρησιμοποιούνται είναι η συνένωση αυτών των δύο πεδίων σε ένα, το οποίο καθορίζει τόσο τον κωδικό λειτουργίας όσο και το τρόπο διευθυνσιοδότησης. Οι διαφορετικές καταστάσεις αυτού του πεδίου είναι κλ και συνεπώς ο ελάχιστος αριθμός δυαδικών ψηφίων που θα πρέπει να αφιερωθεί γι' αυτό το πεδίο είναι $B = \lceil \log_2(\kappa\lambda) \rceil$. Για το παράδειγμά μας είναι B = 7.

Άσκηση 18

Ποιες είναι οι συναρτήσεις που υλοποιούν οι παρακάτω κώδικες γλώσσας μηχανής;

α) Αρχιτεκτονική μηχανισμού σωρού:

```
PUSH  A
PUSH  D
ADD
PUSH  B
PUSH  D
MUL
POP   E
PUSH  A
MUL
PUSH  C
ADD
PUSH  E
ADD
POP   E
```

β) Αρχιτεκτονική συσσωρευτή :

```
LOAD   A
MUL    C
ADD    B
STORE  E
LOAD   D
ADD    E
STORE  E
```

γ) Αρχιτεκτονική καταχωρητή-μνήμης:

```
LOAD   R1, A
ADD    R1, D
LOAD   R2, B
ADD    R2, C
STORE  R2, E
ADD    R1, E
MUL    R1, A
STORE  R1, E
```

δ) Αρχιτεκτονική καταχωρητή-καταχωρητή :

```
LOAD   R1, C
LOAD   R2, A
ADD    R2, R2, R1
LOAD   R1, B
ADD    R1, R2, R1
LOAD   R2, D
MUL    R2, R2, R1
ADD    R1, R2, R1
STORE  R1, E
```

Λύση :

α)

Εντολή		Σωρός μετά την εκτέλεση (Κορυφή στα αριστερά)
PUSH	A	A
PUSH	D	D, A
ADD		D+A
PUSH	B	B, D+A
PUSH	D	D, B, D+A
MUL		D*B, D+A
POP	E	D+A
	Σημείωση :	E=D*B
PUSH	A	A, D+A
MUL		A*(D+A)
PUSH	C	C, A*(D+A)
ADD		C+A*(D+A)
PUSH	E	D*B, C+A*(D+A)
ADD		D*B+C+A*(D+A)
POP	E	ΚΕΝΟ

E = D*B+C+A*(A+D)

β)

Εντολή		Αποτέλεσμα μετά την εκτέλεση (Σ = συσσωρευτής)
LOAD	A	Σ = A
MUL	C	Σ = A*C
ADD	B	Σ = B+A*C
STORE	E	E = B+A*C
LOAD	D	Σ = D
ADD	E	Σ = D+B+A*C
STORE	E	E = D+B+A*C

E = D+B+A*C

γ)

Εντολή		Αποτέλεσμα μετά την εκτέλεση
LOAD	R1, A	R1 = A
ADD	R1, D	R1 = A + D
LOAD	R2, B	R2 = B
ADD	R2, C	R2 = B + C
STORE	R2, E	E = B + C
ADD	R1, E	R1 = A+D+B+C
MUL	R1, A	R1 = A*(A+D+B+C)
STORE	R1, E	E = A*(A+D+B+C)

E = A * (A + D + B + C)

δ) Αρχιτεκτονική καταχωρητή-καταχωρητή :

Εντολή		Αποτέλεσμα μετά την εκτέλεση
LOAD	R1, C	R1 = C
LOAD	R2, A	R2 = A
ADD	R2, R2, R1	R2 = A+C
LOAD	R1, B	R1 = B
ADD	R1, R2, R1	R1 = A+C+B
LOAD	R2, D	R2 = D
MUL	R2, R2, R1	R2 = D*(A+C+B)
ADD	R1, R2, R1	R1= D*(A+C+B)+ (A+C+B)
STORE	R1, E	E= D*(A+C+B)+ (A+C+B)

E = D * (A + C + B) + (A + C + B)

Ασκηση 19

Αναφέρετε πλεονεκτήματα / μειονεκτήματα του προγραμματισμού σε γλώσσα Assembly.

Λύση

Πλεονεκτήματα :

- Λιγότερος κώδικας από τη γλώσσα μηχανής.
- Λιγότερος κίνδυνος λαθών από τον προγραμματισμό σε γλώσσα μηχανής.
- Πλήρης έλεγχος του υλικού.
- Πλήρης έλεγχος της ταχύτητας εκτέλεσης, σε επίπεδο κύκλων μηχανής.

Μειονεκτήματα :

- Ελάχιστη ως καθόλου μεταφερσιμότητα μεταξύ διαφορετικών συστημάτων.
- Κάθε διαφορετικό ολοκληρωμένο επεξεργαστή έχει τη δική του γλώσσα assembly.
- Χαμηλή γλώσσα προγραμματισμού => πολύ μακριά από το τρόπο σκέψης του προγραμματιστή.
- Ο προγραμματισμός και η αποσφαλμάτωση σε γλώσσα assembly είναι μια εξαιρετικά επίπονη διαδικασία.

Άσκηση 20

Η εντολή LDB AL, [80] (φόρτωσης ενός byte σε έναν τυχαίο καταχωρητή AL με έμμεση αναφορά στη μνήμη) ποια λειτουργία εκτελεί ; Ποιο θα είναι το αποτέλεσμα εκτέλεσης αυτής της εντολής όταν αρχικά ισχύουν : [AL] = F3, [80] = F3A2, [F3A0] = AA, [F3A1] = AB, [F3A2]=BA, [F3A3]=77 (το σύμβολο [X] δείχνει τα περιεχόμενα του X) ;

Λύση

Η εντολή αυτή προσπελαύνει τις θέσεις μνήμης 80 και 81, ώστε να ανακαλέσει μια ποσότητα των 16 δυαδικών ψηφίων. Ακόλουθα προσπελαύνει τη θέση μνήμης που υποδεικνύουν αυτά τα 16 δυαδικά ψηφία ανακαλώντας από εκεί τη τελική ποσότητα των 8 δυαδικών ψηφίων. Αντίγραφο αυτής της ποσότητας αποθηκεύεται στον καταχωρητή AL.

Για τα δεδομένα της άσκησης παίρνουμε ότι η προσπέλαση στις θέσεις μνήμης 80 και 81 θα μας επιστρέψει το F3A2. Προσπελαύνουμε στη συνέχεια τη θέση μνήμης F3A2, η οποία μας επιστρέφει τη τιμή BA. Η τιμή αυτή θα αποθηκευτεί στον καταχωρητή AL, δηλαδή στη θέση μνήμης F3. Συνεπώς η εκτέλεση αυτής της εντολής θα έχει ως αποτέλεσμα [F3]=BA.

Άσκηση 21

Εξηγείστε την αναγκαιότητα ύπαρξης ετικετών (labels) στη Γλώσσα Assembly.

Λύση

Κατά τη συγγραφή του κώδικα σε συμβολική γλώσσα πολλές φορές χρειάζεται να αλλάξουμε τη ροή ενός προγράμματος με εντολές άλματος / διακλάδωσης / κλήσης. Την ώρα όμως που γράφουμε το πρόγραμμα, δε γνωρίζουμε τη θέση μνήμης στην οποία θα αποθηκευτεί ο κώδικας. Ως αποτέλεσμα δε μπορούμε να προσδιορίσουμε τη διεύθυνση – στόχο του άλματος. Αυτή θα προσδιοριστεί πολύ αργότερα, κατά τη διαδικασία μετάφρασης του συμβολικού κώδικα σε κώδικα μηχανής. Ωστόσο εμείς μπορούμε να χρησιμοποιήσουμε την πινακίδα η οποία θα αντικατασταθεί αργότερα στη διαδικασία συμβολομετάφρασης.

Κάποιος θα μπορούσε να επιχειρηματολογήσει ότι εφόσον η Assembly είναι μια χαμηλή γλώσσα που δίνει πλήρη έλεγχο στο υλικό, θα μπορούσε ο προγραμματιστής να επιβάλλει στον Assembler την αρχική διεύθυνση στην οποία θα παράγει τον εκτελέσιμο κώδικα και μετρώντας τις εντολές της Assembly και τα τελούμενά τους, να βρει τις απόλυτες διευθύνσεις – στόχους. Το επιχείρημα αυτό είναι σαθρό καθώς :

- Η δουλειά του προγραμματιστή είναι μόνο η συγγραφή του προγράμματος και όχι το να ψάχνει να βρει ποια περιοχή της μνήμης είναι διαθέσιμη και χωρά το πρόγραμμά του.
- Με το τρόπο αυτό ο κώδικας χάνει τη δυνατότητα μετακίνησής του (relocability). Με τις πινακίδες αντίθετα, το πρόγραμμα μπορεί εύκολα να φορτωθεί σε οποιαδήποτε θέση μνήμης είναι διαθέσιμη.

Άσκηση 22

Γράψτε ένα πρόγραμμα Assembly που να αντιγράφει το byte που βρίσκεται στη θέση μνήμης $5F40_{16}$ και στη θέση μνήμης $5F80_{16}$.

Λύση

```
LDB     80,     5F40[00]
STB     80,     5F80[00]
```

Άσκηση 23

Γράψτε ένα πρόγραμμα Assembly που να προσθέτει δύο τελούμενα των 16 δυαδικών ψηφίων χωρίς να κάνετε χρήση της εντολής πρόσθεσης λέξεων. Τα δυαδικά ψηφία του πρώτου τελουμένου βρίσκονται στις θέσεις $5F20_{16}$ και $5F21_{16}$, του δευτέρου στις θέσεις $5F40_{16}$ και $5F41_{16}$, ενώ το αποτέλεσμα της πρόσθεσης θέλουμε να αποθηκεύεται στις θέσεις μνήμης $5F80_{16}$ και $5F81_{16}$,

Λύση

```
LDB     80,     5F20[00]
LDB     81,     5F40[00]
ADDB    80,     81
STB     80,     5F80[00]
LDB     80,     5F21[00]
LDB     81,     5F41[00]
ADDCB   80,     81
STB     80,     5F81[00]
```

Υπάρχουν 2 λεπτά σημεία στην παραπάνω άσκηση. Το πρώτο είναι ότι εφόσον δεν μας διατίθεται πρόσθεση λέξεων θα πρέπει να χρησιμοποιήσουμε τις διατιθέμενες εντολές για πρόσθεση bytes. Θα πρέπει να φροντίσουμε για τη διάδοση τυχόν κρατουμένου που παράγεται κατά την πρόσθεση των bytes χαμηλής σημαντικότητας κατά την πρόσθεση αυτών της υψηλής σημαντικότητας. Ετσι στη δεύτερη εντολή πρόσθεσης θα πρέπει να λάβουμε υπ' όψιν μας και το κρατούμενο. Το δεύτερο λεπτό σημείο είναι το πότε παρήχθηκε το κρατούμενο το οποίο λαμβάνουμε υπ' όψιν μας στη δεύτερη άθροιση. Είναι όντως αυτό που τυχόν παρήχθη από την πρώτη ή μήπως έχει αλλοιωθεί από τις ενδιάμεσες εντολές ? Ανατρέχοντας λοιπόν στο manual των εντολών, θα πρέπει να διαπιστώσουμε ότι οι ενδιάμεσες εντολές LDB δεν έχουν τη δυνατότητα να επηρεάσουν την κατάσταση του κρατουμένου.

Ασκηση 24

Γράψτε ένα πρόγραμμα Assembly που εξετάζει τη τιμή του byte της θέσης μνήμης $5F20_{16}$ και αν είναι άρτιος αριθμός αποθηκεύει 0 στη θέση μνήμης $5F21_{16}$. Αλλιώς αποθηκεύει 1.

Λύση

Για το πρόβλημα αυτό μπορούμε να επινοήσουμε σειρά από διαφορετικές λύσεις. Παρακάτω παρουσιάζονται μερικές από αυτές για να δείξουμε ότι η Assembly παρέχει πραγματικά τη δυνατότητα σε έναν έμπειρο προγραμματιστή να ξεχωρίσει από έναν άπειρο. Κάθε άρτιος αριθμός έχει το δεξιότερο δυαδικό ψηφίο 0. Το αντίστοιχο ψηφίο σε έναν περιττό είναι 1. Η πρώτη εκδοχή εξετάζει αυτό το ψηφίο μέσω δεξιάς ολίσθησης με κατοπινή εξέταση του carry flag.

```
        LDB   80,   5F20[00]
        LDB   81,   #01
        SHRB  80,   81
        JC    ODD
        LDB   90,   #00
        STB   90,   5F21[00]
        SJMP  END
[ODD]   LDB   90,   #01
        STB   90,   5F21[00]
[END]   BRK
```

Ο παραπάνω κώδικας είναι επιεικώς απαράδεκτος. Κι αυτό γιατί κατά πρώτον χρησιμοποιείται ο 90 για να φορτωθεί το #01, κάτι που ήδη υπάρχει στον 81 ! Επίσης είτε ο αριθμός είναι άρτιος είτε περιττός η αποθήκευση θα πρέπει να γίνει. Συνεπώς η εντολή αποθήκευσης πρέπει να είναι ανεξάρτητη του ποιο κομμάτι κώδικα θα ακολουθήσουμε. Μια πρώτη βελτιωμένη έκδοση του παραπάνω κώδικα θα μπορούσε να είναι η ακόλουθη :

```
        LDB   80,   5F20[00]
        LDB   81,   #01
        SHRB  80,   81
        JC    ODD
        LDB   81,   #00
[ODD]   STB   81,   5F21[00]
```

Η βελτίωση έγκειται αφενός στη χρήση λιγότερων καταχωρητών και αφετέρου στο ότι ο δεύτερος κώδικας είναι μικρότερος και συνεπώς ταχύτερος σε εκτέλεση.

Η εξέταση ψηφίων ενός αριθμού με τη μέθοδο των ολισθήσεων και κατοπινού ελέγχου του κρατουμένου θα πρέπει να αποφεύγεται. Κι αυτό γιατί αν θέλουμε να εξετάσουμε μήτε το αριστερότερο μήτε το δεξιότερο ψηφίο ενός αριθμού, ο χρόνος της ολίσθησης αυξάνει σημαντικά. Επιπλέον πολλές φορές χρειάζεται να εξετάσουμε όχι μόνο ένα αλλά περισσότερα ψηφία ενός αριθμού. Μέσω ολισθήσεων τότε παίρνουμε έναν εκθετικό αριθμό από διαφορετικές

περιπτώσεις που πρέπει να εξεταστούν καταλήγοντας σε εξαιρετικά πολύπλοκο κώδικα με πολλά άλματα (γνωστός και ως κώδικας spaghetti). Ο παρακάτω κώδικας είναι αντίστοιχος με τον πρώτο, μόνο που χρησιμοποιεί τη μέθοδο της μάσκας, απομονώνει δηλαδή τα ψηφία που μας ενδιαφέρουν.

```
            LDB   80,   5F20[00]
            ANDB  80,   #01
            JNE   ODD
            LDB   80,   #00
            STB   80,   5F21[00]
            SJMP  END
[ODD]       LDB   80,   #01
            STB   80,   5F21[00]
[END]       BRK
```

Ο κώδικας αυτός είναι σαφώς καλύτερος από αυτόν της πρώτης περίπτωσης. Είναι μικρότερος και χρησιμοποιεί μόνο έναν καταχωρητή. Μια επιπλέον παρατήρηση όμως μπορεί να μας οδηγήσει σε ακόμη καλύτερα αποτελέσματα. Μετά το λογικό AND του αριθμού με τη μάσκα, στον καταχωρητή 80 μένει το 0 αν ο αριθμός ήταν άρτιος και το 1 αν ο αριθμός ήταν περιττός ! Συνεπώς δε χρειάζεται να εξεταστούν περαιτέρω οι δύο περιπτώσεις και μπορούμε να καταλήξουμε στην αποθήκευση του 80 στη θέση $5F21_{16}$:

```
            LDB   80,   5F20[00]
            ANDB  80,   #01
            STB   80,   5F21[00]
            BRK
```

Άσκηση 25

Γράψτε ένα πρόγραμμα Assembly που να βρίσκει το μεγαλύτερο μη προσημασμένο byte μεταξύ των αποθηκευμένων στις θέσεις μνήμης 5F21 – $5F2A_{16}$ και να αποθηκεύει ένα αντίγραφό του στη θέση μνήμης $5F20_{16}$.

Λύση

```
            CLR   80
            LDB   82,   5F21[80]
[LOOP]      CMPB  82,   5F22[80]
            JH    SKIP
            LDB   82,   5F22[80]
[SKIP]      INC   80
            CMP   80,   #0009
            JNE   LOOP
            STB   82,   5F20[00]
            BRK
```

Άσκηση 26

Δώστε τον κώδικα σε γλώσσα Assembly για το πρόβλημα της αντιμετάθεσης των τιμών δύο καταχωρητών υποθέτοντας ότι το σύνολο εντολών σας ΔΕΝ περιλαμβάνει καμία εντολή LOAD ή STORE.

Λύση

Ενας δόκιμος τρόπος για την αντιμετάθεση είναι η χρησιμοποίηση της σωρού. Υποθέστε ότι οι δύο καταχωρητές των οποίων επιθυμούμε να αντιμεταθέσουμε τα περιεχόμενα είναι αυτοί με διευθύνσεις 80 και 90. Τότε το ζητούμενο πρόγραμμα είναι :

```
PUSH 80
PUSH 90
POP 80
POP 90
```

Ασκηση 27

Δώστε τον κώδικα σε γλώσσα Assembly για το πρόβλημα της εύρεσης του μέσου όρου 4 αριθμών που είναι αποθηκευμένοι σε διαδοχικές θέσεις μνήμης υποθέτοντας ότι το σύνολο εντολών σας ΔΕΝ περιλαμβάνει εντολή διαίρεσης.

Λύση

Δύο είναι τα λεπτά σημεία αυτής της άσκησης :

1. Η εύρεση του μέσου όρου 4 αριθμών χωρίς εντολή διαίρεσης. Γνωρίζοντας ότι διαίρεση δια 2 μπορεί να επιτευχθεί με δεξιά ολίσθηση κατά 1 θέση, είναι προφανές ότι η ζητούμενη διαίρεση μπορεί να γίνει με δεξιά ολίσθηση κατά 2 θέσεις.
2. Ενα άλλο πρόβλημα είναι αυτό της ακρίβειας που χρειάζεται. Αν υποθέσουμε ότι οι 4 αριθμοί είναι μεγέθους ενός byte, τότε το άθροισμά τους προφανώς δε μπορεί να αποθηκευτεί σε ένα byte λόγω πιθανότητας υπερχείλισης. Αρα θα πρέπει η άθροιση να γίνει με διπλάσια ακρίβεια.

Υποθέτουμε παρακάτω ότι οι 4 αριθμοί βρίσκονται στις θέσεις μνήμης 6000_H έως 6003_H και ο ζητούμενος μέσος όρος μένει στον καταχωρητή με διεύθυνση 80.

```
LDBSE 80, 6000[00]
LDBSE 82, 6001[00]
LDBSE 84, 6002[00]
LDBSE 86, 6003[00]
ADD 80, 82
ADD 84, 86
ADD 80, 84
LDB 82, #02
SHRA 80, 82
```

Άσκηση 28

Δίδεται ένα μη προσημασμένο byte στη θέση μνήμης 5F20. Δώστε τον κώδικα Assembly για τον υπολογισμό του ακεραίου μέρους της τετραγωνικής ρίζας του δοθέντος αριθμού και την αποθήκευσή του στη θέση μνήμης 5F21.

Λύση

Το ακέραιο μέρος της ρίζας ενός αριθμού μπορεί να προσεγγιστεί από τον αριθμό των επιτυχών αφαιρέσεων διαδοχικών περιττών αριθμών. Ο αλγόριθμος μπορεί να εκφραστεί από τον παρακάτω κώδικα, όπου ο καταχωρητής 81 διατρέχει τους περιττούς αριθμούς και ο καταχωρητής 82 είναι ο μετρητής των επιτυχών αφαιρέσεων :

```
            LDB     81,     #01
            LDB     80,     5F20[00]
            CLRB    82
[AGAIN]     CMPB    80,     81
            JLT     DONE
            SUBB    80,     81
            INCB    82
            ADDB    81,     #02
            SJMP    AGAIN
[DONE]      STB     82,     5F21[00]
```

Άσκηση 29

Εξηγείστε πως μπορεί να προσεγγιστεί η διαίρεση δύο μη προσημασμένων αριθμών με τη μέθοδο των διαδοχικών επιτυχών αφαιρέσεων. Δώστε τον κώδικα Assembly.

Λύση

Προφανώς η διαίρεση μπορεί να προσεγγιστεί με τη μέθοδο των διαδοχικών αφαιρέσεων του διαιρέτη από τον διαιρετέο. Κάθε επιτυχής αφαίρεση θα προσθέτει 1 στο πηλίκο. Το υπόλοιπο είναι ότι μένει στο διαιρετέο μετά και την τελευταία επιτυχή αφαίρεση. Στο παρακάτω υποθέτουμε ότι διαιρετέος και διαιρέτης είναι μήκους 1 byte και βρίσκονται στις θέσεις μνήμης 5F20 και 5F21 αντίστοιχα. Το πηλίκο της διαίρεσης αποθηκεύεται στον καταχωρητή με διεύθυνση 82, ενώ το υπόλοιπο παραμένει στον 80.

```
            LDB     80,     5F20[00]
            LDB     81,     5F21[00]
            CLRB    82
[LOOP]      CMPB    80, 81
            JLT     END
            INCB    82
            SUBB    80, 81
            SJMP    LOOP
[END]       BRK
```

Άσκηση 30

Δώστε το τον κώδικα Assembly για την εύρεση του μέσου όρου ΜΟΝΟ των περιττών αριθμών που βρίσκονται στις θέσεις μνήμης 5B20 – 5B29. Ο μέσος όρος αποθηκεύεται στη θέση μνήμης 5C00.

Λύση

Στην άσκηση αυτή υπάρχουν μερικά λεπτά σημεία. Το πρώτο είναι η διαπίστωση αν ένας αριθμός είναι περιττός ή όχι. Στη παρακάτω λύση η εξέταση γίνεται απομονώνοντας το λιγότερο σημαντικό ψηφίο και συγκρίνοντας αυτό το byte με το 0. Το δεύτερο είναι ότι το άθροισμα των περιττών αριθμών μπορεί να ξεπεράσει ως ενδιάμεσο αποτέλεσμα την ακρίβεια του ενός byte. Αρα θα πρέπει από πριν να μεριμνήσουμε ώστε κάθε άθροιση να γίνει με διπλή ακρίβεια. Η επέκταση κάθε byte σε λέξη διπλάσιας ακρίβειας προφανώς πρέπει να γίνει με επέκταση προσήμου, αφού η εκφώνηση δεν κάνει νύξη για μη προσημασμένους αριθμούς.

```
            CLR     90
            CLRB    86
            LD      80,     #5B20
[LOOP]      LDB     82,     [80]
            ANDB    83,     82,     #01
            JE      NEXT
            LDBSE   84,     82
            ADD     90,     84
            INCB    86
[NEXT]      INC     80
            CMP     80,     #5B2A
            JNE     LOOP
            DIV     90,     86
            STB     90,     5C00[00]
```

Ο απαιτούμενος κώδικας για τη λύση της άσκησης δίδεται παραπάνω. Στον καταχωρητή λέξης 90 αποθηκεύεται το ενδιάμεσο άθροισμα των περιττών αριθμών. Ο καταχωρητής 86 μετράει το πλήθος τους. Ο καταχωρητής 80 χρησιμοποιείται για την έμμεση προσπέλαση των αριθμών της λίστας, που προσωρινά αποθηκεύονται στον καταχωρητή 82. Ο καταχωρητής 83 αποθηκεύει το αποτέλεσμα της λογικής σύζευξης του 82 με τη μάσκα που απομονώνει το τελευταίο δυαδικό ψηφίο. Ο επεκταμένος κατά πρόσημο περιττός αριθμός αποθηκεύεται στον καταχωρητή 84. Τέλος προσέξτε ότι κατά τη διαίρεση του αθροίσματος με το πλήθος των περιττών αριθμών, το πηλίκο (ο ζητούμενος μέσος όρος δηλαδή) μένει στη χαμηλή διεύθυνση μνήμης.

Άσκηση 31

Στη θέση μνήμης 5F20 υπάρχει το byte $b_7b_6b_5b_4b_3b_2b_1b_0$. Ζητείται να γράψετε ένα πρόγραμμα σε γλώσσα Assembly που να αντιμεταθέτει τα nibbles αυτού του byte, που δηλαδή μετά την εκτέλεσή του, στη θέση μνήμης 5F20 θα υπάρχει το byte $b_3b_2b_1b_0b_7b_6b_5b_4$.

Λύση

Θα δώσουμε δύο διαφορετικές λύσεις σε αυτή την άσκηση. Η πρώτη λύση απομονώνει αυτές τις δύο τετράδες μέσω των απαραίτητων ολισθήσεων και μετά συνενώνει τα δύο αποτελέσματα μέσω της λογικής διάζευξης.

```
        LDB     80,     5F20[00]
        LDB     81,     80
        LDB     83,     #04
        SHRB    81,     83
        SHLB    80,     83
        ORB     80,     81
        STB     80,     5F20[00]
```

Η δεύτερη είναι αρκετά πιο εξεζητημένη και βασίζεται στην ολίσθηση μιας ολόκληρης λέξης (εντολή SHR) όπου τα περιεχόμενα της υψηλής διεύθυνσης ολισθαίνουν στα υψηλής τάξης ψηφία της διεύθυνσης με τη χαμηλή διεύθυνση.

```
        LDB     80,     5F20[00]
        LDB     81,     80
        LDB     82,     #04
        SHR     80,     82
        STB     80,     5F20[00]
```

Ασκηση 32

Γράψτε ένα πρόγραμμα Assembly που διατρέχει μια λίστα προσημασμένων αριθμών (short integers) και μετατρέπει όλα τα bit προσήμου σε 0, όταν ισχύουν τα εξής :

- Ο 16-bit καταχωρητής 82, θα περιέχει την αρχική διεύθυνση της λίστας
- Ο 8-bit καταχωρητής 85, θα περιέχει το μήκος της λίστας (σε bytes).

Λύση

```
        LDBZE   90,     85
        ADD     90,     82
        DEC     90
[LOOP]  CMP     90,     82
        JLT     END
        LDB     84,     [82]
        ANDB    84,     #7F
        STB     84,     [82]+
        SJMP    LOOP
[END]   BRK
```

Άσκηση 33

Στη θέση μνήμης $5F20_{16}$ υπάρχει αποθηκευμένο ένα byte. Γράψτε ένα πρόγραμμα σε γλώσσα Assembly που να ανιχνεύει πόσες φορές εμφανίζεται ο συνδυασμός "110" στα δυαδικά ψηφία αυτού του byte και θα αποθηκεύει τη τιμή αυτή στη θέση μνήμης $5F21_{16}$.

Λύση

```
        LDB     91,     #01
        LDB     90,     #00
        LDB     80,     5F20[00]
        LDB     81,     #07
[LOOP]  ANDB    82,     80,     81
        CMPB    82,     #06
        JNE     SKIP
        INCB    90
[SKIP]  SHRB    80,     91
        CMPB    80,     #06
        JLT     END
        SJMP    LOOP
[END]   BRK
```

Άσκηση 34

Γράψτε ένα πρόγραμμα που αντιστρέφει τη σειρά των στοιχείων μιας λίστας, δηλαδή το τελευταίο στοιχείο γίνεται πρώτο, το προτελευταίο δεύτερο κοκ. Ισχύουν :

- Ο 16-bit καταχωρητής 82, θα περιέχει την αρχική διεύθυνση της λίστας
- Ο 8-bit καταχωρητής 85, θα περιέχει το μήκος της λίστας (σε bytes).

Λύση

```
        LDBZE   90,     85
        ADD     90,     82
        DEC     90
[LOOP]  CMP     90,     82
        JLE     END
        PUSH    [82]
        PUSH    [90]
        POP     [82]+
        POP     [90]
        DEC     90
        SJMP    LOOP
[END]   BRK
```

Άσκηση 35

Γράψτε ένα πρόγραμμα που αναλόγως των δυαδικών ψηφίων μιας μάσκας μηδενίζει ή αφήνει αναλλοίωτα τα στοιχεία μιας λίστας. Για παράδειγμα αν η μάσκα είναι 11001011, το 3ο, το 5ο και το 6ο στοιχείο της λίστας θα μηδενιστούν ενώ όλα τα υπόλοιπα θα μείνουν ως έχουν. Υποθέστε ότι :

- Η λίστα ξεκινά από τη διεύθυνση $5F00_{16}$ και έχει μήκος 16 bytes.
- Η μάσκα (16-bit) βρίσκεται στη θέση μνήμης $5F40_{16}$.

Λύση

```
        LD      7A,     #0001
        LD      80,     5F40[00]
        CLR     90
[LOOP]  SHR     80,     7A
        JC      SKIP
        STB     7B,     5F00[90]
[SKIP]  INC     90
        CMPB    90,     #10
        JNE     LOOP
        BRK
```

Άσκηση 36

Γράψτε ένα πρόγραμμα σε Assembly που υπολογίζει το μέγιστο κοινό διαιρέτη δύο bytes, που είναι αποθηκευμένα στις θέσεις μνήμης $5F20_{16}$ και $5F21_{16}$. Ο μέγιστος κοινός διαιρέτης αποθηκεύεται στη θέση μνήμης $5F24_{16}$.

Λύση

Ο μέγιστος κοινός διαιρέτης δεν μπορεί να είναι μεγαλύτερος του μικρότερου εκ των δύο αριθμών. Στο παρακάτω κομμάτι κώδικα εντοπίζουμε τον μικρότερο από τους δύο αριθμούς και τον αποθηκεύουμε στον καταχωρητή 82. Ο μεγαλύτερος αποθηκεύεται zero extended στον καταχωρητή 84.

```
        LDB     80,     5F20[00]
        CMPB    80,     5F21[00]
        JLT     SWITCH
        LDB     82,     5F21[00]
        LDBZE   90,     82
        LDBZE   84,     80
        SJMP    LOOP
[SWITCH] LDB    82,     80
        LDBZE   84,     5F21[00]
        LDBZE   90,     80
```

Για να βρούμε τον μέγιστο κοινό διαιρέτη, ψάχνουμε διαδοχικά όλους τους αριθμούς από το περιεχόμενο του καταχωρητή 82 προς τα κάτω για το αν διαιρούν και τους δύο αριθμούς. Προσέξτε ότι η πράξη της διαίρεσης πειράζει τα περιεχόμενα του διαιρετέου και γι αυτό θα πρέπει να κρατάμε αντίγραφα πριν από κάθε διαίρεση στους καταχωρητές 86 και 88.

```
[LOOP]   LD     86,   84
         DIVUB  86,   82
         CMPB   87,   #00
         JNE    NEXT
         LD     88,   90
         DIVUB  88,   82
         CMPB   89,   #00
         JE     FOUND
[NEXT]   DECB   82
         JE     STOP
         SJMP   LOOP
[FOUND]  STB    82,   5F24[00]
[STOP]   BRK
```

Άσκηση 37

Γράψτε ένα πρόγραμμα σε Assembly που να βρίσκει μετά από πόσες αριστερές ολισθήσεις του byte που βρίσκεται στη θέση μνήμης $5F20_{16}$ θα προκύψει ποσότητα μεγαλύτερη του 39_{10}. Το πλήθος των ολισθήσεων αυτών να αποθηκευτεί στη θέση μνήμης $5F22_{16}$.

Λύση

```
         LDB    80,   5F20[00]
         CMPB   80,   #00
         JE     EXIT
         LDB    81,   #00
         LDB    82,   #27          (39(10)=00100111(2)=27(16))
         LDB    83,   #01
[LOOP]   CMPB   80,   82
         JGT    DONE
         SHLB   80,   83
         INCB   81
         SJMP   LOOP
[DONE]   STB    81,   5F22[00]
[EXIT]   BRK
```

Άσκηση 38

Γράψτε ένα πρόγραμμα σε Assembly που να εντοπίζει τον πρώτο άρτιο αριθμό στη λίστα που υπάρχει αποθηκευμένη στις θέσεις μνήμης $5F20_{16}$ έως $5F29_{16}$. Η διεύθυνση που εντοπίστηκε θα πρέπει να αποθηκευτεί στις θέσεις μνήμης $5F2A_{16}$ και $5F2B_{16}$.

Λύση

```
            LD      80,     #5F20
[LOOP]      LDB     82,     [80]
            ANDB    82,     #01
            JE      FOUND
            INC     80
            CMP     80,     #5F2A
            JNE     LOOP
            SJMP    SKIP
[FOUND]     ST      80, 5F2A[00]
[SKIP]      BRK
```

Άσκηση 39

Δείξτε πως μπορείτε να υλοποιήσετε σε γλώσσα Assembly τις ακόλουθες δύο δομές της γλώσσας C :

- if συνθήκη then { code segment A} else {code segment B}
  όπου συνθήκη μία από τις ακόλουθες : α!=β, α==β, α>β, α<β, α<=β, α>=β
- for {i=0; i<MAXVALUE; i++} {code segment C}

Λύση

- Εστω ότι οι καταχωρητές 80 και 82 περιέχουν αντίστοιχα τις τιμές α και β. Τότε για την υλοποίηση της δομής if, ο κώδικας Assembly που απαιτείται είναι ο ακόλουθος :

```
            CMP     80,     82
            JXX     SEGMENTA
[SEGMENTB]  …
            Εδώ τοποθετείται ο κώδικας Assembly που υλοποιεί το code segment B
            …
            SJMP NEXT
[SEGMENTA] …
            Εδώ τοποθετείται ο κώδικας Assembly που υλοποιεί το code segment B
            …
    [NEXT]
```

όπου JXX είναι ένα υπό συνθήκη άλμα. Ανάλογα με τη συνθήκη που θέλουμε να εξετάσουμε στο if statement, το άλμα που χρειαζόμαστε διαμορφώνεται σύμφωνα με τον παρακάτω πίνακα :

α!=β	JNE
a==b	JE
a>b	JGT
a<b	JLT
a>=b	JGE
a<=b	JLE

- Εστω ότι ο καταχωρητής 80 θα παίξει το ρόλο της μεταβλητής i του loop. Εστω ότι η τιμή του καταχωρητή 82 είναι ίση με MAXVALUE. Επίσης θεωρούμε ότι MAXVALUE ≤ 255_{10}. Τότε ο κώδικας Assembly που απαιτείται έχει τη μορφή :

```
            LDB     80,     #00
[LOOP]      …
            Εδώ τοποθετείται ο κώδικας Assembly που υλοποιεί το code segment C
            …
            INCB    80
            CMPB    80, 82
            JNE     LOOP
```

Ασκηση 40

Περιγράψτε συνοπτικά τις διαδικασίες που λαμβάνουν χώρα όταν συμβεί μια αίτηση διακοπής.

Λύση

Υπάρχουν δύο περιπτώσεις :

1. Ο επεξεργαστής να μη βρίσκεται σε κατάσταση εκτέλεσης κάποιου κώδικα κρίσιμου χρόνου. Στην περίπτωση αυτή η αίτηση διακοπής που ξεκινά από μια περιφερειακή συσκευή, μέσω του ελεγκτή διακοπών ανακοινώνεται στον επεξεργαστή. Αυτός ολοκληρώνει την εκτέλεση της τρέχουσας εντολής, αποθηκεύει τη κατάσταση του προγράμματος που έτρεχε στη σωρό συμπεριλαμβανομένης της διεύθυνσης της επόμενης προς εκτέλεση εντολής και ακολούθως μέσω του ελεγκτή διακοπών ενημερώνεται για το ποιο είναι το περιφερειακό που ζητά εξυπηρέτηση. Μεταπηδά στη διεύθυνση αρχής του προγράμματος εξυπηρέτησης για το συγκεκριμένο περιφερειακό και εκτελεί το πρόγραμμα εξυπηρέτησης. Οταν τελειώσει αυτό το πρόγραμμα, ανακαλεί από τη σωρό το περιβάλλον της διακοπείσας διαδικασίας και συνεχίζει κανονικά την εκτέλεσή της.

2. Ο επεξεργαστής να βρίσκεται στην εκτέλεση κάποιου κώδικα κρίσιμου χρόνου. Στην περίπτωση αυτή πολύ πιθανόν να έχει απενεργοποιήσει τη δυνατότητα διακοπών, οπότε η αίτηση διακοπής θα αγνοηθεί μέχρις ότου ολοκληρωθεί η εκτέλεση του κρίσιμου κώδικα.

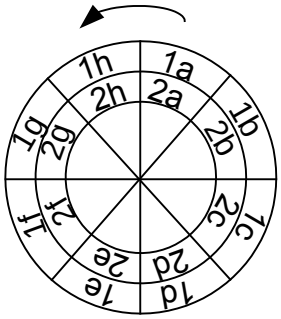
Ασκηση 41

Υποστηρίξτε την αλήθεια ή το ψέμα της πρότασης : "Αν με τη χρήση DMA η μεταφορά ενός byte μεταξύ περιφερειακού - κύριας μνήμης διαρκεί 1 κύκλο ρολογιού, τότε μπορώ να μεταφέρω ΠΑΝΤΟΤΕ 100 bytes σε 100 διαδοχικούς κύκλους".

Λύση

Η πρόταση αυτή είναι προφανώς ψευδής. Η ταχύτητα μεταφοράς δεν έχει τίποτε να κάνει με το εάν ο ελεγκτής του DMA θα ελέγχει για 100 κύκλους τις αρτηρίες του συστήματος. Με άλλα λόγια για να μπορεί να μεταφέρει διαρκώς 1 byte / κύκλο θα πρέπει να είναι κύριος των αρτηριών για 100 κύκλους, πράγμα εξαιρετικά απίθανο.

Ασκηση 42



α) Στην εικόνα φαίνονται 2 tracks ενός δίσκου με κινούμενη κεφαλή με 8 sectors το καθένα. Αν ο κάθε sector είναι των 512 bytes πόσους sectors, θα καταλάβει η αποθήκευση των αρχείων Α, Β και Γ όταν :

- το Α έχει μέγεθος 2123 bytes ?
- το Β έχει μέγεθος 2848 bytes ?
- το Γ έχει μέγεθος 640 bytes ?

β) Αριθμώντας τους sectors κάθε αρχείου, υποδείξτε τη φυσική οργάνωση πάνω στο δίσκο που επιτρέπει την ταχύτερη ανάγνωση και των τριών αρχείων Α, Β και Γ σειριακά. (π.χ. γράφοντας "Α1 στο 1c", υποδεικνύετε ότι ο πρώτος sector του αρχείου Α θα τοποθετηθεί στο sector 1c της εικόνας).

Λύση

α) Το Α θα καταλάβει $\lceil 2123/(512) \rceil = 5$ sectors, το Β θα καταλάβει $\lceil 2848/(512) \rceil = 6$ sectors και το Γ $\lceil 640/(512) \rceil = 2$ sectors. Με άλλα λόγια μόνο ακέραια πολλαπλάσια του

sector μπορούν να ανατεθούν στα διάφορα αρχεία, ακόμη κι αν μέρη των sectors αυτών είναι μερικά γεμάτα.

β) Για την ταχύτερη σειριακή ανάγνωση αυτών των αρχείων, θα πρέπει να ελαχιστοποιήσουμε τις κινήσεις της κεφαλής. Συνεπώς θα πρέπει να προσπαθήσουμε να αποθηκεύσουμε όσο το δυνατόν περισσότερη πληροφορία σε γειτονικούς sectors. Βάζουμε λοιπόν τους sectors του Α αρχείου ως εξής : Α1, Α2, Α3, Α4, Α5 στους sectors 1a, 1b, 1c, 1d, 1e αντίστοιχα. Κατόπιν συνεχίζουμε σειριακά με το αρχείο Β και βάζουμε Β1, Β2 και Β3 στους sectors 1f, 1g και 1h αντίστοιχα. Αφού τώρα έχει γεμίσει αυτό το track θα πρέπει να συνεχίσουμε στο $2^{ο}$. Η κεφαλή όμως έχει διαβάσει τον 1h και συνεπώς η μικρότερη κίνηση είναι απλά να μετακινηθεί στο εσωτερικό track. Αρα η αποθήκευση θα πρέπει να συνεχίσει Β4, Β5 και Β6, Γ1 και Γ2 στους sectors 2a, 2b, 2c και 2d και 2e αντίστοιχα.

(Σημείωση : Λόγω αδράνειας η κεφαλή θα έχει κατά πάσα πιθανότητα μετακινηθεί πέρα από το τέλος του 1h. Οπότε το απόλυτο σωστό πρακτικά είναι η αποθήκευση στο $2^{ο}$ track να αρχίσει από το 2b).

Ασκηση 43

Υποστηρίξτε την αλήθεια ή το ψέμα των παρακάτω :

- Προσθέτοντας ένα ψηφίο ισοτιμίας σε μια τυχαία ομάδα ψηφιολέξεων που έχουν απόσταση κ, φτιάχνω μια νέα ομάδα λέξεων με απόσταση κ+1.
- Κάθε ακέραιος σε αναπαράσταση συμπληρώματος ως προς 2 των n δυαδικών ψηφίων, έχει τον αντίθετό του.

Λύση

- Η πρόταση είναι ψευδής. Εστω για παράδειγμα ένας κώδικας με τις δύο πιο κάτω κωδικές λέξεις :

  0000

  1111

  με απόσταση 4. Αν προσθέσω ένα ψηφίο άρτιας ισοτιμίας θα πάρω τις ακόλουθες κωδικές λέξεις

  00000

  11110

  που συνεχίζουν να έχουν απόσταση 4.
- Η πρόταση είναι λάθος. Ο κώδικας συμπληρώματος ως προς 2 με n δυαδικά ψηφία μπορεί να απεικονίσει το διάστημα ακεραίων $[-2^{n-1}, 2^{n-1})$, το οποίο δεν είναι πλήρως συμμετρικό, αφού ο αντίθετος του -2^{n-1}, δεν ανήκει στο διάστημα των αναπαριστώμενων ποσοτήτων.

Ασκηση 44

Δώστε τον κώδικα Assembly ώστε στη θέση μνήμης 5F21 να αποθηκεύεται το δυαδικό ψηφίο περιττής ισοτιμίας του byte που βρίσκεται στη θέση μνήμης 5F20.

Λύση

Το πρόβλημα λύνεται απλά εξετάζοντας διαδοχικά τα δυαδικά ψηφία του byte και αλλάζοντας ένα αρχικό ψηφίο ισοτιμίας μετά από κάθε εξέταση. Στον καταχωρητή 80 φορτώνεται το byte του οποίου θέλουμε να φτιάξουμε το ψηφίο περιττής ισοτιμίας. Στον 81 υπάρχει το τρέχων υπολογισθέν ψηφίο. Στον 82 αποθηκεύεται η ποσότητα ολίσθησης που γίνεται σε κάθε κύκλο, δηλαδή ολίσθηση κατά 1 δυαδικό ψηφίο. Τέλος αντιστροφή του ψηφίου ισοτιμίας έχουμε κάθε φορά που κατά την ολίσθηση προκύψει κρατούμενο. Η αντιστροφή γίνεται με μια εντολή αποκλειστικής διάζευξης που αντιστρέφει μόνο το τελευταίο ψηφίο του 81.

```
            LDB   80,    5F20[00]
            LDB   81,    #01
            LDB   82,    #01
[LOOP]      CMPB  80,    #00
            JE    END
            SHRB  80,    82
            JNC   LOOP
            XORB  81,    #01
            SJMP  LOOP
[END]       STB   81,    5F21[00]
```

Ασκηση 45

Από τα πλέον συνηθισμένα λάθη στα ψηφιακά κυκλώματα είναι αυτά που προκαλούνται από παροδικές βυθίσεις ή αυξήσεις στη τάση τροφοδοσίας. Αυτές έχουν σαν αποτέλεσμα μια ψηφιολέξη n δυαδικών ψηφίων να γίνεται όλο 0 ή όλο 1.

α) Μπορεί ο κώδικας ισοτιμίας να μας προστατέψει παράλληλα και από τα δύο παραπάνω είδη λαθών ? (Υπόδειξη : Προφανώς επηρεάζεται αναλόγως και το ψηφίο ισοτιμίας. Εξετάστε δύο περιπτώσεις : το n να είναι άρτιος ή περιττός).

β) Χρησιμοποιώντας τις ιδιότητες που παρατηρήσατε στο (α), προτείνετε έναν αποδοτικό κώδικα προστασίας από τέτοια λάθη.

Λύση

Ο παρακάτω πίνακας υποδεικνύει το τι συμβαίνει για άρτιο / περιττό n, άρτια / περιττή ισοτιμία και για λάθη που παράγουν όλο 0 ή όλο 1.

n	***Ισοτιμία***	***Λάθος***	***Προστασία***
Αρτιος	Αρτια	Ολα 0	Οχι
		Ολα 1	Ναι
Αρτιος	Περιττή	Ολα 0	Ναι
		Ολα 1	Οχι
Περιττός	Αρτια	Ολα 0	Ναι
		Ολα 1	Οχι
Περιττός	Περιττή	Ολα 0	Ναι
		Ολα 1	Ναι

Βλέπουμε συνεπώς ότι ένας κώδικας ισοτιμίας δε μπορεί να μας προστατέψει από αυτά τα λάθη στη περίπτωση που το n είναι άρτιος, ενώ επίσης δε μπορεί να μας προστατέψει όταν το n είναι περιττός και χρησιμοποιήσουμε άρτια ισοτιμία.

Μια λύση σε αυτό το πρόβλημα είναι να διαιρέσω την αρχική μου πληροφορία σε δύο το δυνατόν ίσα μέρη. Στο πρώτο μέρος επισυνάπτω ψηφίο άρτιας ισοτιμίας και στο δεύτερο ψηφίο περιττής ισοτιμίας, φροντίζοντας ώστε αν υπάρχει μέρος με περιττό αριθμό ψηφίων σε αυτό να εφαρμοστεί η περιττή ισοτιμία.

Ασκηση 46

Πόση είναι η απόσταση του κώδικα που πρέπει να χρησιμοποιήσω για να μπορώ να επιτύχω ανίχνευση έως και τετραπλών λαθών ? Αν ο κώδικας φτιάχνεται από λέξεις των 5 δυαδικών ψηφίων πόσες κωδικές λέξεις υπάρχουν ? Αποδείξτε τον ισχυρισμό σας.

Λύση

Για να μπορώ να ανιχνεύσω την ύπαρξη έως και d λαθών, χρειάζομαι έναν κώδικα με απόσταση d+1. Συνεπώς για την ανίχνευση τετραπλών λαθών θα πρέπει να χρησιμοποιήσω έναν κώδικα απόστασης 5. Κώδικας απόστασης 5 σημαίνει ότι κάθε ζεύγος δυαδικών ψηφιολέξεων του κώδικα έχουν απόσταση κατά Hamming τουλάχιστον 5, δηλαδή διαφέρουν τουλάχιστον σε 5 δυαδικά ψηφία. Αν οι κωδικές λέξεις έχουν όλα κι όλα 5 δυαδικά ψηφία, αυτό σημαίνει ότι οι λέξεις που υπάρχουν στον κώδικα είναι μόνο δύο, οι :

$$\beta_4\ \beta_3\ \beta_2\ \beta_1\ \beta_0$$

$$!\beta_4\ !\beta_3\ !\beta_2\ !\beta_1\ !\beta_0$$

(Σημείωση : ανάλογα με τις τιμές που θα υιοθετηθούν για τα β_4, β_3, β_2, β_1 και β_0 προκύπτει κι ένας διαφορετικός κώδικας, όχι όμως άλλες κωδικές λέξεις στον ίδιο κώδικα. Συνολικά υπάρχουν 16 διαφορετικοί κώδικες με δύο κωδικές λέξεις ο κάθε ένας).

Ασκηση 47

α) Πόση είναι η απόσταση ενός κώδικα που επιτρέπει την ανίχνευση διπλών λαθών ?

β) Πόση είναι η απόσταση ενός κώδικα που επιτρέπει τη διόρθωση απλών λαθών ?

γ) Πόση είναι η απόσταση ενός κώδικα που θα επιτρέπει τόσο το (α) όσο και το (β) παράλληλα? Εξηγείστε αναλυτικά το γιατί.

δ) Αναπαραστήστε τον -78_{10} σε συμπλήρωμα ως προς 1, χρησιμοποιώντας 8 δυαδικά ψηφία. Κωδικοποιείστε την προκύπτουσα αναπαράσταση σε κώδικα με τις ιδιότητες του υποερωτήματος (γ), χρησιμοποιώντας τα ελάχιστα δυαδικά ψηφία.

Λύση

α) Για να μπορώ να ανιχνεύσω την ύπαρξη έως και d λαθών, χρειάζομαι έναν κώδικα με απόσταση d+1. Συνεπώς για την ανίχνευση διπλών λαθών θα πρέπει να χρησιμοποιήσω έναν κώδικα απόστασης 3.

β) Για να μπορώ να διορθώσω έως και c λάθη, χρειάζομαι έναν κώδικα με απόσταση 2c+1. Συνεπώς για την διόρθωση απλών λαθών θα πρέπει να χρησιμοποιήσω έναν κώδικα απόστασης 3.

γ) Ισχύουν τα ακόλουθα :

- Ο απλός κώδικας Hamming διορθώνει απλά λάθη, χωρίς να μπορεί να προσφέρει άλλες ιδιότητες ανίχνευσης / διόρθωσης. Αρα έχει απόσταση τουλάχιστον 3.
- Ο απλός κώδικας Hamming είναι βέλτιστος. Συνεπώς η απόστασή του είναι ακριβώς 3.
- Με τη πρόσθεση ενός δυαδικού ψηφίου ισοτιμίας παίρνουμε τον κώδικα modified Hamming που παρέχει ικανότητες ανίχνευσης διπλών και διόρθωσης απλών λαθών και επίσης είναι βέλτιστος.

Αρα το ερώτημα της άσκησης μπορεί να τροποποιηθεί στο πόση είναι η απόσταση ενός modified Hamming κώδικα. Θα αποδείξουμε ότι είναι εκ κατασκευής 4. Εστω οι λέξεις του απλού κώδικα Hamming A και B. Αν οι A και B έχουν απόσταση 4 ή μεγαλύτερη ότι και ψηφίο ισοτιμίας να προσθέσω για τη κατασκευή των αντίστοιχων λέξεων του modified Hamming κώδικα, οι προκύπτουσες κωδικές λέξεις θα έχουν απόσταση τουλάχιστον 4. Εστω τώρα ότι οι A και B έχουν απόσταση ακριβώς 3, δηλαδή διαφέρουν στα δυαδικά ψηφία x, y και z και ότι ο αριθμός των άσσων στα υπόλοιπα ψηφία των A και B είναι φ. Τότε μπορούμε να κατασκευάσουμε τον παρακάτω πίνακα που δείχνει ότι στον modified κώδικα Hamming οι λέξεις που προκύπτουν από τα A και B θα διαφέρουν και στο επισυναπτόμενο ψηφίο καθολικής ισοτιμίας (P_A και P_B).

φ	Αριθμός 1 στα x, y και z	Αριθμός 1 στα x, y και z	P_A	P_B
Αρτιος	0	3	0	1
	1	2	1	0
	2	1	0	1
	3	0	1	0
Περιττός	0	3	1	0
	1	2	0	1
	2	1	1	0
	3	0	0	1

Αρα η απόσταση και αυτών των ζευγών θα είναι 4 στον κώδικα modified Hamming.

(Σημείωση : με την ίδια αποδεικτική διαδικασία μπορούμε να δείξουμε ότι η πρόσθεση ενός ψηφίου ισοτιμίας σε κάθε κώδικα με περιττό αριθμό απόστασης, αυξάνει την αρχική απόσταση κατά 1).

δ) Ο -78_{10} σε συμπλήρωμα ως προς 1 έχει ως αναπαράσταση το συμπλήρωμα της δυαδικής αναπαράστασης του 78_{10}, που με τη μέθοδο των διαδοχικών διαιρέσεων μπορούμε να βρούμε ότι είναι 01001110_2. Αρα $-78_{10} = 10110001_{1's}$. Η ζητούμενη κωδικοποίηση έχει συνεπώς τη μορφή :

13	12	11	10	9	8	7	6	5	4	3	2	1
P	M_7	M_6	M_5	M_4	C_8	M_3	M_2	M_1	C_4	M_0	C_2	C_1
	1	0	1	1		0	0	0		1		

$C_1=M_0 \oplus M_1 \oplus M_3 \oplus M_4 \oplus M_6 = 0$

$C_2=M_0 \oplus M_2 \oplus M_3 \oplus M_5 \oplus M_6 = 0$

$C_4=M_1 \oplus M_2 \oplus M_3 \oplus M_7 = 1$

$C_8=M_4 \oplus M_5 \oplus M_6 \oplus M_7 = 1$

$P=M_0 \oplus M_1 \oplus M_2 \oplus M_3 \oplus M_4 \oplus M_5 \oplus M_6 \oplus M_7 \oplus C_1 \oplus C_2 \oplus C_4 \oplus C_8 = 0$

και συνεπώς το κωδικοποιημένο μήνυμα είναι :

13	12	11	10	9	8	7	6	5	4	3	2	1
P	M_7	M_6	M_5	M_4	C_8	M_3	M_2	M_1	C_4	M_0	C_2	C_1
0	1	0	1	1	1	0	0	0	1	1	0	0

Ασκηση 48

α) Δίδεται το ακόλουθο δυαδικό μήνυμα :

πιο σημαντικό ψηφίο -> **0 0 0 1 1 0 1 0 1 1 0 1** <- λιγότερο σημαντικό ψηφίο

Ο αποστολέας του μηνύματος το έχει κωδικοποιήσει κατά Hamming έχοντας χρησιμοποιήσει τα ελάχιστα απαιτούμενα δυαδικά ψηφία ελέγχου. Ελέγξτε την ορθότητα του μηνύματος, διορθώστε τυχόν λάθη του και εξάγετε την αρχική πληροφορία.

β) Απαντήστε στον αποστολέα του πιο πάνω μηνύματος χρησιμοποιώντας κώδικα ισοτιμίας στήλης γραμμής (με όποια οργάνωση επιθυμείτε) και την πληροφορία :

πιο σημαντικό ψηφίο -> **0 1 1 1 0 1 1** <- λιγότερο σημαντικό ψηφίο

γ) Υποθέτοντας ένα ρυθμό μετάδοσης δεδομένων 1bit/ns πόσο χρόνο διήρκεσε η ανταλλαγή αυτών των πληροφοριών ? Σε τι ποσοστό του χρόνου ανταλλάχθηκε χρήσιμη πληροφορία ? Ποιος από τους δύο χρησιμοποιηθέντες κώδικες ευθύνεται περισσότερο για την μικρή ωφέλιμη χρήση του διαθέσιμου ρυθμού μετάδοσης ?

Λύση

α) Αφού το μήνυμα είναι κωδικοποιημένο κατά Hamming συμπεραίνουμε ότι στις θέσεις που είναι δυνάμεις του 2 βρίσκονται ψηφία ελέγχου και στις υπόλοιπες ψηφία πληροφορίας. Με άλλα λόγια το μήνυμα έχει τη μορφή :

12	11	10	9	8	7	6	5	4	3	2	1
M_7	M_6	M_5	M_4	C_8	M_3	M_2	M_1	C_4	M_0	C_2	C_1
0	0	0	1	1	0	1	0	1	1	0	1

Ελέγχοντας εκ νέου την ισοτιμία των ομάδων παίρνουμε :

$P_1 = M_6 \oplus M_4 \oplus M_3 \oplus M_1 \oplus M_0 \oplus C_1 = 1$

$P_2 = M_6 \oplus M_5 \oplus M_3 \oplus M_2 \oplus M_0 \oplus C_2 = 0$

$P_4 = M_7 \oplus M_3 \oplus M_2 \oplus M_1 \oplus C_4 = 0$

$P_8 = M_7 \oplus M_6 \oplus M_5 \oplus M_4 \oplus C_8 = 0$

Συνεπώς έχει συμβεί λάθος στο $P_8P_4P_2P_1 = 0001 = 1^{ο}$ ψηφίο του μηνύματος, το οποίο και αντιστρέφεται ώστε να πάρουμε τη σωστή πληροφορία :

12	11	10	9	8	7	6	5	4	3	2	1
M_7	M_6	M_5	M_4	C_8	M_3	M_2	M_1	C_4	M_0	C_2	C_1
0	0	0	1	1	0	1	0	1	1	0	0

Απορρίπτοντας τα ψηφία ελέγχου, βρίσκουμε ότι ο αποστολέας έστειλε τη πληροφορία $M_7M_6M_5M_4M_3M_2M_1M_0$=00010101.

β) Εστω ότι επιλέγω την οργάνωση της πληροφορίας μου σε τέσσερις γραμμές και δύο στήλες. Υποθέτοντας άρτια ισοτιμία παίρνω :

0	1	1
1	1	0
0	1	1
1	0	1
0	1	1

Το έξτρα δυαδικό ψηφίο δε χρειάζεται να μεταδοθεί, αλλά μπορεί να υπονοείται από αποστολέα και παραλήπτη. Επίσης αν δε μας ενδιαφέρει η ισοτιμία των ψηφίων ισοτιμίας δε χρειάζεται να παραχθεί και να μεταδοθεί το κάτω δεξιά ψηφίο.

γ) Τα συνολικά ψηφία που μεταφέρονται είναι 12 στην (α) περίπτωση και 13 (14) στη (β). Για τα 25 (26) αυτά ψηφία θα χρειαστεί χρόνος 25 (26) ns. Η χρήσιμη πληροφορία είναι 8 δυαδικά ψηφία στην (α) περίπτωση και 7 δυαδικά ψηφία στη (β). Αρα το ποσοστό ωφέλιμης χρήσης είναι 15 / 25 = 60% (15/26 = 57,7%). Στην (α) έχουμε ωφέλιμη χρήση κατά 8/12 = 75% ενώ στη (β) κατά 7/13 = 53,8% (7/14 = 50%). Συνεπώς ο δεύτερος κώδικας επιβαρύνει περισσότερο το ρυθμό μετάδοσης.

Ασκηση 49

- Υπολογίστε πόσα δυαδικά ψηφία ελέγχου απαιτούνται για την διόρθωση απλού λάθους μηνύματος μήκους 11 δυαδικών ψηφίων.
- Δείξτε την κωδικοποίηση που προκύπτει για το μήνυμα : 0 1 1 1 1 0 0 1 0 1 0.
- Δείξτε πως μπορείτε να κάνετε διόρθωση του κωδικοποιημένου μηνύματος, όταν συμβεί λάθος στο 3ο δυαδικό ψηφίο του αρχικού μηνύματος.
- Επαναλάβετε τα παραπάνω όταν σπάσουμε το αρχικό μήνυμα σε τετράδες και εφαρμόσουμε κώδικα ισοτιμίας στήλης / γραμμής. Ποια είναι προτιμότερη λύση και γιατί ?

Λύση

- Από τον τύπο $2^c \geq c+d+1$, για d=11, βρίσκουμε ότι ο ελάχιστος αριθμός ψηφίων ελέγχου που ικανοποιούν την ανισότητα είναι c=4 και συνεπώς θα πρέπει να προσθέσουμε 4 δυαδικά ψηφία ελέγχου.
- Η ζητούμενη κωδικοποίηση έχει τη μορφή :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
M_{10}	M_9	M_8	M_7	M_6	M_5	M_4	C_8	M_3	M_2	M_1	C_4	M_0	C_2	C_1
0	1	1	1	1	0	0		1	0	1		0		

$$C_1=M_0 \oplus M_1 \oplus M_3 \oplus M_4 \oplus M_6 \oplus M_8 \oplus M_{10} = 0$$

$$C_2=M_0 \oplus M_2 \oplus M_5 \oplus M_6 \oplus M_9 \oplus M_{10} = 1$$

$$C_4=M_1 \oplus M_2 \oplus M_3 \oplus M_7 \oplus M_8 \oplus M_9 \oplus M_{10} = 1$$

$$C_8=M_4 \oplus M_5 \oplus M_6 \oplus M_7 \oplus M_8 \oplus M_9 \oplus M_{10} = 0$$

και συνεπώς το κωδικοποιημένο μήνυμα είναι :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
M_{10}	M_9	M_8	M_7	M_6	M_5	M_4	C_8	M_3	M_2	M_1	C_4	M_0	C_2	C_1
0	1	1	1	1	0	0	0	1	0	1	1	0	1	0

- Υποθέτουμε ότι συμβαίνει λάθος στο M_8 και συνεπώς λαμβάνουμε το ακόλουθο λανθασμένο μήνυμα :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
M_{10}	M_9	M_8	M_7	M_6	M_5	M_4	C_8	M_3	M_2	M_1	C_4	M_0	C_2	C_1
0	1	0	1	1	0	0	0	1	0	1	1	0	1	0

Επαναϋπολογίζουμε την ισοτιμία των ομάδων και παίρνουμε :

$$P_1 = C_1 \oplus M_0 \oplus M_1 \oplus M_3 \oplus M_4 \oplus M_6 \oplus M_8 \oplus M_{10} = 1$$

$$P_2 = C_2 \oplus M_0 \oplus M_2 \oplus M_5 \oplus M_6 \oplus M_9 \oplus M_{10} = 0$$

$$P_4 = C_4 \oplus M_1 \oplus M_2 \oplus M_3 \oplus M_7 \oplus M_8 \oplus M_9 \oplus M_{10} = 1$$

$$P_8 = C_8 \oplus M_4 \oplus M_5 \oplus M_6 \oplus M_7 \oplus M_8 \oplus M_9 \oplus M_{10} = 1$$

Συνεπώς το λάθος εντοπίζεται στη θέση $P_8P_4P_2P_1 = 1101 = 13_{10}$ που βρίσκεται το M_8.

- Σπάζοντας το μήνυμα σε 4άδες και εφαρμόζοντας οριζόντια και κάθετη ισοτιμία παίρνουμε την εξής κωδικοποίηση :

0	1	1	1	1
1	0	0	1	0
0	1	0	0	1
1	0	1	0	0

όπου υπονοούμε ένα επιπλέον ψηφίο ίσο με 0. Εστω ότι πάλι αλλοιώνεται το M_8, δηλαδή ότι ο παραλήπτης παίρνει την εξής πληροφορία :

0	1	0	1	1
1	0	0	1	0
0	1	0	0	1
1	0	1	0	0

Από τον έλεγχο ισοτιμίας διαπιστώνει πρόβλημα στη πρώτη γραμμή και τη πρώτη στήλη. Συνεπώς το λανθασμένο δυαδικό ψηφίο εντοπίζεται στην τομή τους και αντιστρέφεται.

- Μεταξύ των δύο κωδίκων προτιμότερος είναι ο Hamming, αφού για μήνυμα ίδιου μεγέθους απαιτεί σημαντικά μικρότερο αριθμό ψηφίων ελέγχου.

Άσκηση 50

Το ακόλουθο μήνυμα που έφτασε σε σας είναι κωδικοποιημένο σε κώδικα modified Hamming. Ελέγξτε την ορθότητά του και εφόσον σας το επιτρέπουν οι δυνατότητες του κώδικα, διορθώστε τυχόν λάθη του. Υποθέστε άρτια ισοτιμία κωδικοποίησης.

(Πιο σημαντικό ψηφίο) → 0 0 1 1 1 1 0 1 1 0 ← (Ψηφίο Ισοτιμίας)

Λύση

Αφού το ψηφίο ισοτιμίας έχει υπολογιστεί πάνω σε όλη τη κωδική λέξη, μπορούμε να συμπεράνουμε ότι η ισοτιμία της λέξης είναι $0 \oplus 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 0$, που συμπίπτει με το μεταδοθέν ψηφίο ισοτιμίας. Αρα ή η κωδική λέξη είναι σωστή ή έχει συμβεί άρτιος αριθμός λαθών και πρέπει να την απορρίψουμε. Η κωδική λέξη έχει τη μορφή :

9	8	7	6	5	4	3	2	1
M_4	C_8	M_3	M_2	M_1	C_4	M_0	C_2	C_1
0	0	1	1	1	1	0	1	1

και μπορούμε να δούμε ότι η ισοτιμία για το C_1, δηλαδή η $C_1 \oplus M_0 \oplus M_1 \oplus M_3 \oplus M_4$ είναι λανθασμένη. Συνεπώς έχει συμβεί άρτιος αριθμός λαθών και πρέπει να απορρίψουμε τη πληροφορία.

Άσκηση 51

Θεωρείστε μια half-duplex σειριακή επικοινωνία μεταξύ δύο σταθμών, έστω Α και Β ενός δικτύου. Οταν ο Α στέλνει πληροφορίες στον Β, τις στέλνει σε πακέτα των 64 δυαδικών ψηφίων στα οποία επισυνάπτει Hamming κωδικοποίηση. Οταν ο Β στέλνει πληροφορίες στον Α, τις στέλνει σε πακέτα των 16 δυαδικών ψηφίων στα οποία επισυνάπτει κώδικα ισοτιμίας στήλης – γραμμής. Υποθέτοντας ότι στο 60% του χρόνου στο κανάλι εκπέμπει ο σταθμός Α και στο υπόλοιπο 40% ο σταθμός Β, υπολογίστε το ποσοστό ωφέλιμης χρήσης του καναλιού, δηλαδή πόσο τοις εκατό χρησιμοποιείται το κανάλι για την ανταλλαγή καθαρής πληροφορίας μεταξύ των δύο σταθμών.

Λύση

Στα 64 δυαδικά ψηφία πληροφορίας ο Α επισυνάπτει τα ελάχιστα ψηφία ελέγχου c έτσι ώστε να ικανοποιείται η σχέση $2^c \geq 64+c+1$. Αρα ο Α επισυνάπτει 7 ψηφία ελέγχου. Συνεπώς όταν μεταδίδει ο Α, μεταδίδει πακέτα των 64+7=71 ψηφίων εκ των οποίων καθαρά ωφέλιμη πληροφορία είναι τα 64.

Τα 16 δυαδικά ψηφία πληροφορίας ο Β τα διατάσσει σαν ένα πίνακα 4x4 και επισυνάπτει 8 (9) ψηφία ελέγχου (4 για κάθετη ισοτιμία και 4 για οριζόντια) (και 1 ακόμη για την ισοτιμία των ψηφίων ισοτιμίας). Συνεπώς όταν μεταδίδει ο Β, μεταδίδει πακέτα των 16+8(9)=24(25) ψηφίων εκ των οποίων καθαρά ωφέλιμη πληροφορία είναι τα 16.

Το ποσοστό ωφέλιμης χρήσης του καναλιού όταν μεταδίδει ο Α κατά 60% και ο Β κατά το υπόλοιπο είναι 0,6*64/71 + 0,4*16/24(25) = 0,8075 (0,7968) δηλαδή 80,75%(79,68%).