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

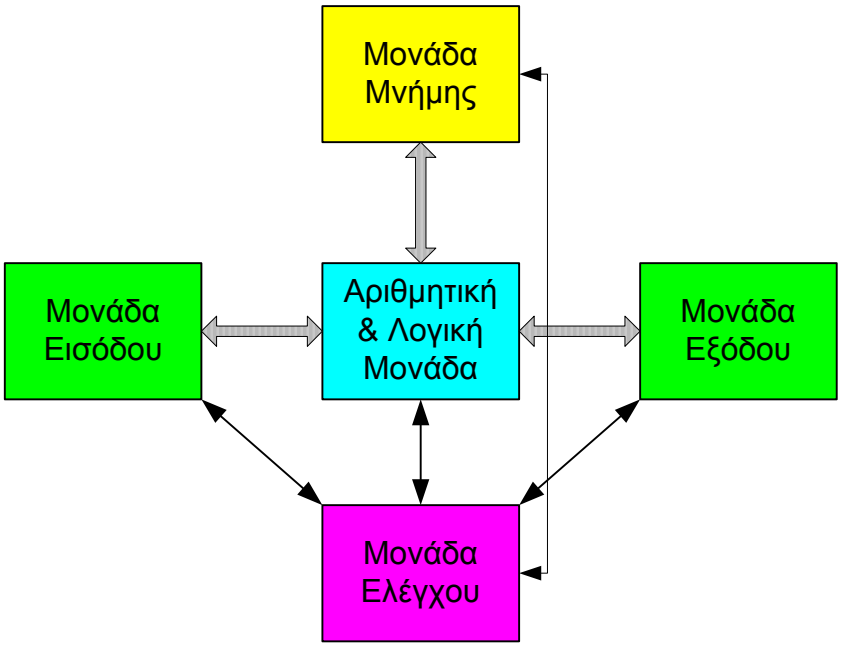
Στα πρώτα χρόνια παρουσίας των ηλεκτρονικών υπολογιστών ο προγραμματισμός γινόταν με την τοποθέτηση χιλιάδων διακοπτών σε συγκεκριμένες θέσεις. Κάθε διακόπτης είχε δύο θέσεις. Η τοποθέτηση στη μία ή την άλλη θέση σήμαινε και την ανάκληση και συνεπώς εκτέλεση διαφορετικής πρωταρχικής συνάρτησης από τη 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) :



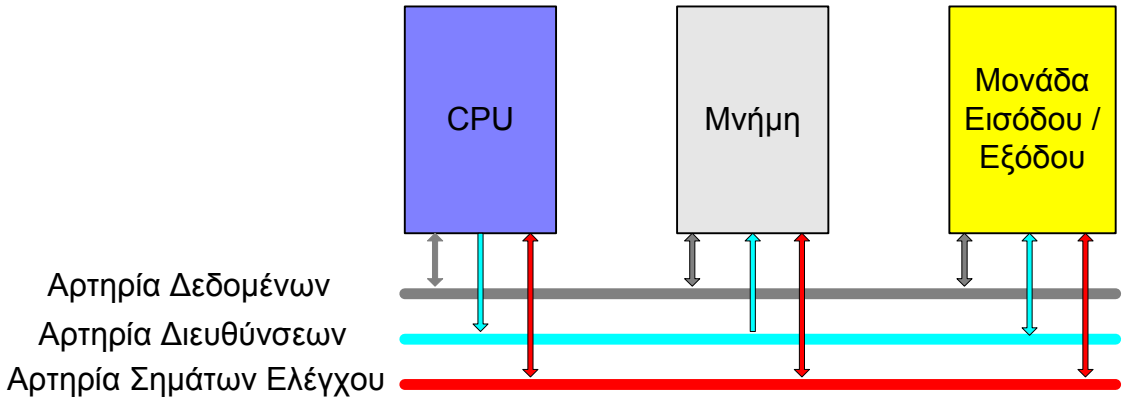
Ας δούμε αναλυτικά πως λειτουργεί το παραπάνω μοντέλο για να καταλάβουμε το ρόλο που καλείται να διαδραματίσει κάθε υπομονάδα. Χρησιμοποιώντας τη ***μονάδα εισόδου (Input Unit)***, εισάγουμε τα ***δεδομένα (data)*** στο υπολογιστικό σύστημα. Τα δεδομένα είναι πλέον τόσο οι ***εντολές (instructions)*** 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 :

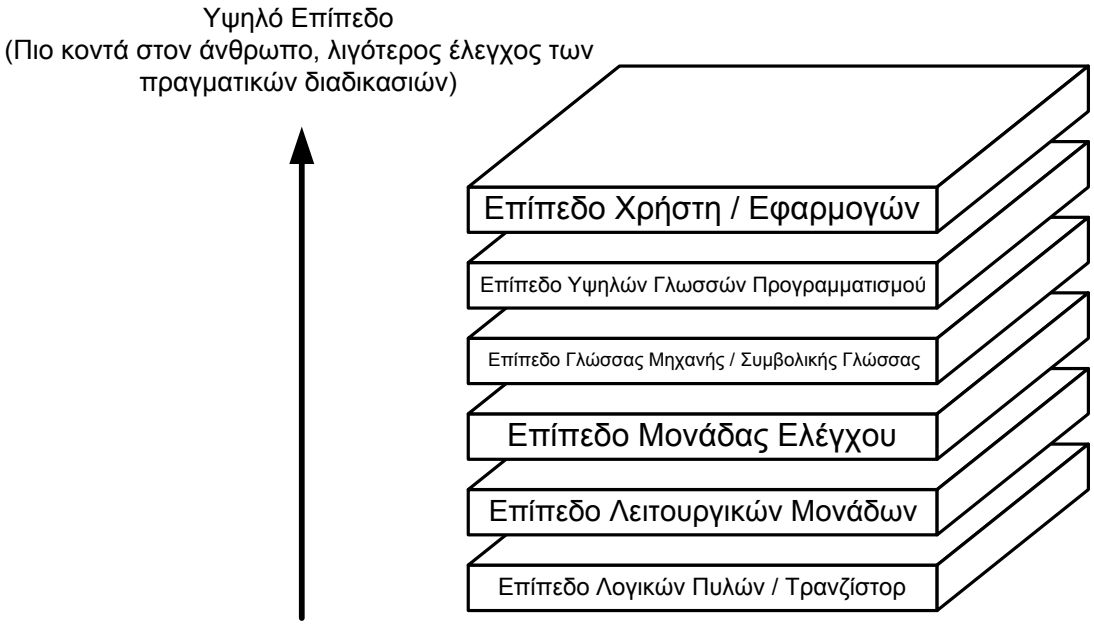
- Το επίπεδο λογικών πυλών, τρανζίστορ και αγωγών, που είναι και το κατώτατο επίπεδο ενός υπολογιστικού συστήματος. Στο επίπεδο αυτό η λειτουργικότητα του υπολογιστή εκφράζεται με δυναμικά, ρεύματα, χρόνους διάδοσης σημάτων και γενικά φαινόμενα πολύ χαμηλού επιπέδου. Τα τρανζίστορ χρησιμοποιούνται για 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 $\delta_{v-1}\delta_{v-2}\ldots\delta_3\delta_2\delta_1\delta_0$, 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 $w_{v-1}w_{v-2} \ldots w_2w_1w_0$. 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 $\delta_{v-1}\delta_{v-2}\ldots\delta_3\delta_2\delta_1\delta_0{}_\rho$. 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 $[-(2^{\kappa-1}-1), 2^{\kappa-1}-1]$. 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 $\rho_{v-1}\rho_{v-2}\ldots\rho_1\rho_0$ 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 = …, 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 $\beta_{\kappa-1}\beta_{\kappa-2} \ldots \beta_1\beta_0$.

<u>Διαδικασία 1.</u> 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.

<u>Διαδικασία 2.</u> 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 $Σ*B^E$, 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.

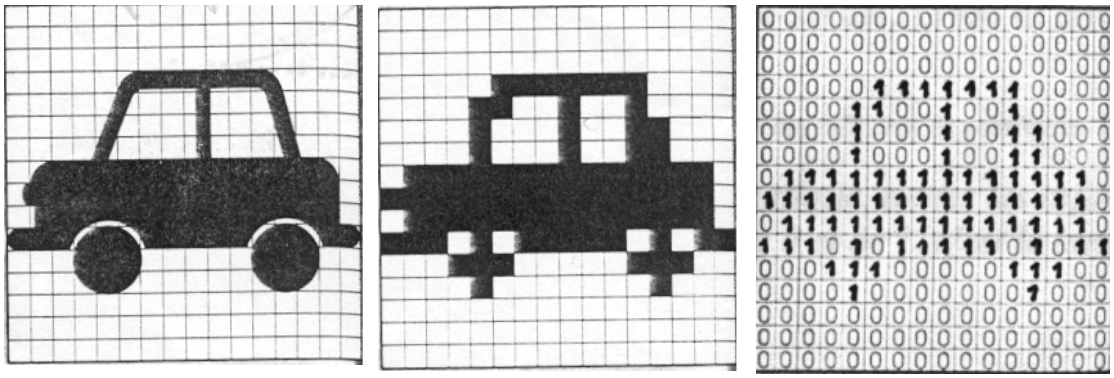
Ωστόσο ακόμα και οι επαυξημένες αναπαραστάσεις του EBCDIC είναι λίγες για τις σημερινές εφαρμογές των υπολογιστών. Η ανάγκη για ακόμη μεγαλύτερο αριθμό αναπαραστάσεων οδήγησε σε ένα παγκόσμιο στάνταρτ αναπαράστασης χαρακτήρων, που ονομάζεται Unicode. Το Unicode είναι ακόμη στο στάδιο της εξέλιξης. Στην 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)***.

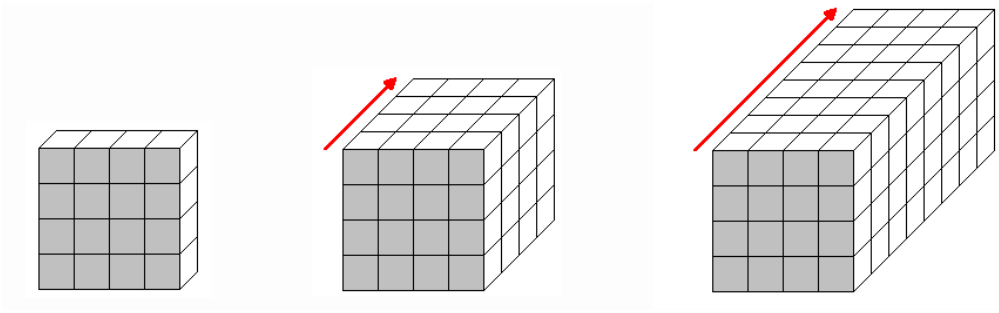
Η αναπαράσταση μιας εικόνας σε ένα υπολογιστικό σύστημα ξεκινά με την κατάτμησή της σε πολύ μικρές κουκίδες (εικονοστοιχεία, picture elements, pixels), με οριζόντιες και κάθετες νοητές γραμμές. Στις εικόνες (α) και (β) παρακάτω απεικονίζεται αυτή η διαδικασία για μια ασπρόμαυρη εικόνα ενός αυτοκινήτου. 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

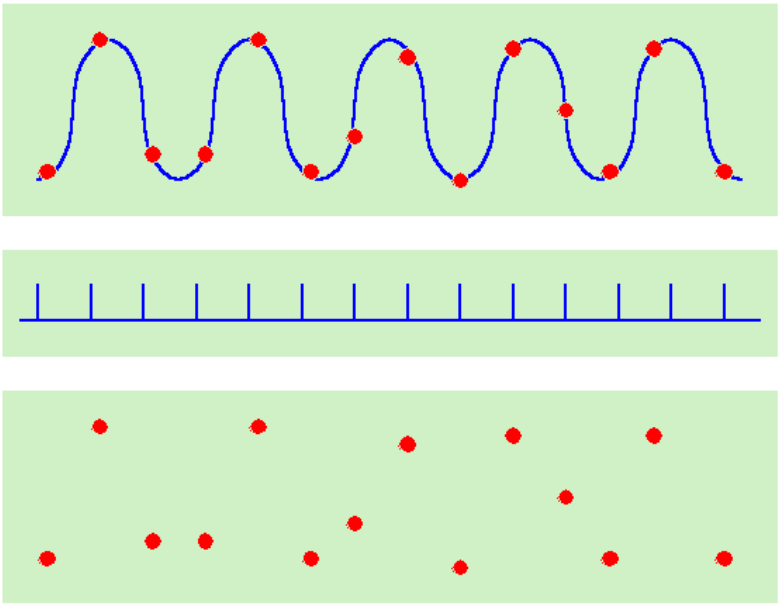
Κόκκινη Συνιστώσα

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.

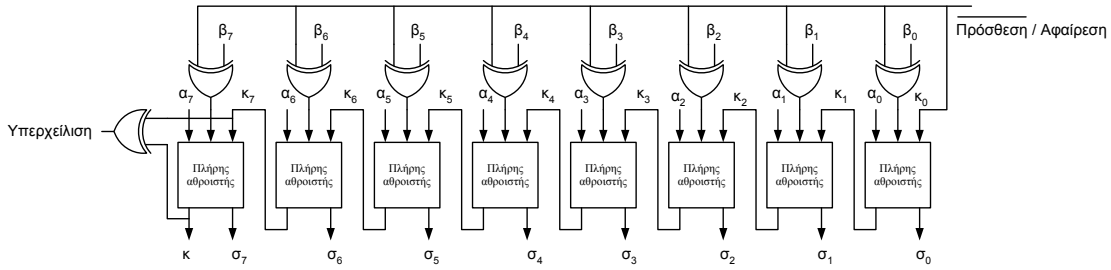
	00110010	(+50)		11010001	(-47)
	01010000	(+80) +		10100110	(-90) +
0	10000010	(-126) !!!	1	01110111	(+119)!!!

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.

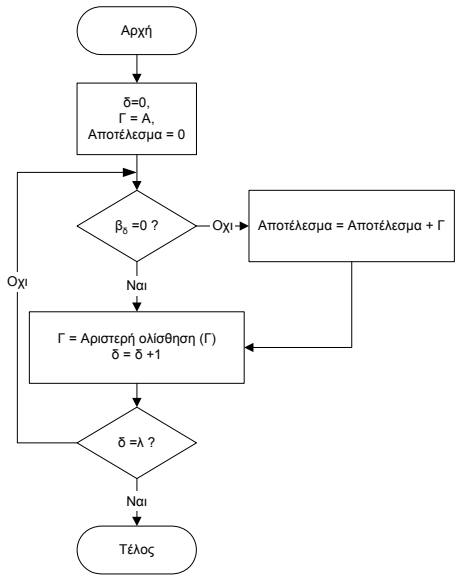
Ας παρακολουθήσουμε την εκτέλεση του παραπάνω αλγορίθμου για τον πολλαπλασιασμό Α=4=100 με το Β=5=101 βήμα προς βήμα.

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

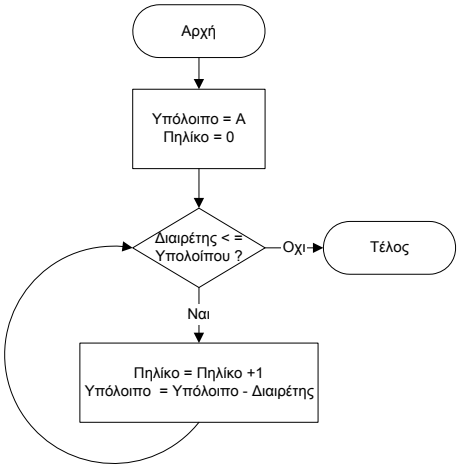
Τα παραπάνω σχήματα συνδέονται μεταξύ τους με βέλη που δείχνουν και τη σειρά εκτέλεσης (ροή) των βημάτων του αλγορίθμου. Ενα διάγραμμα ροής για τον πιο πάνω αλγόριθμο είναι το παρακάτω :



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.

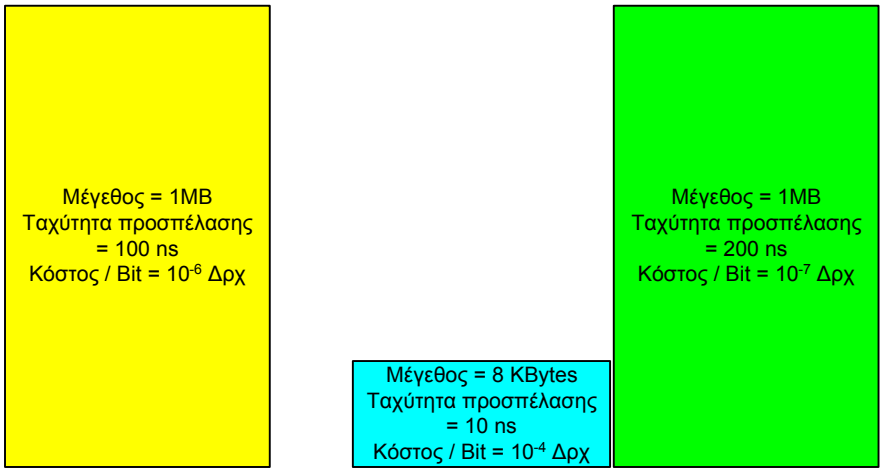
- Οσο μεγαλώνει το μέγεθος μιας μνήμης τόσο πιο αργή γίνεται. (Μια ελλιπής εξήγηση για αυτή τη παρατήρηση είναι ότι μια μεγαλύτερη μνήμη θα έχει περισσότερες γραμμές διευθύνσεων και συνεπώς θα χρειαστεί ένας μεγαλύτερος αποπλέκτης για την επιλογή της ζητούμενης λέξης μνήμης. Πολύ πιο πλήρη εξήγηση θα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

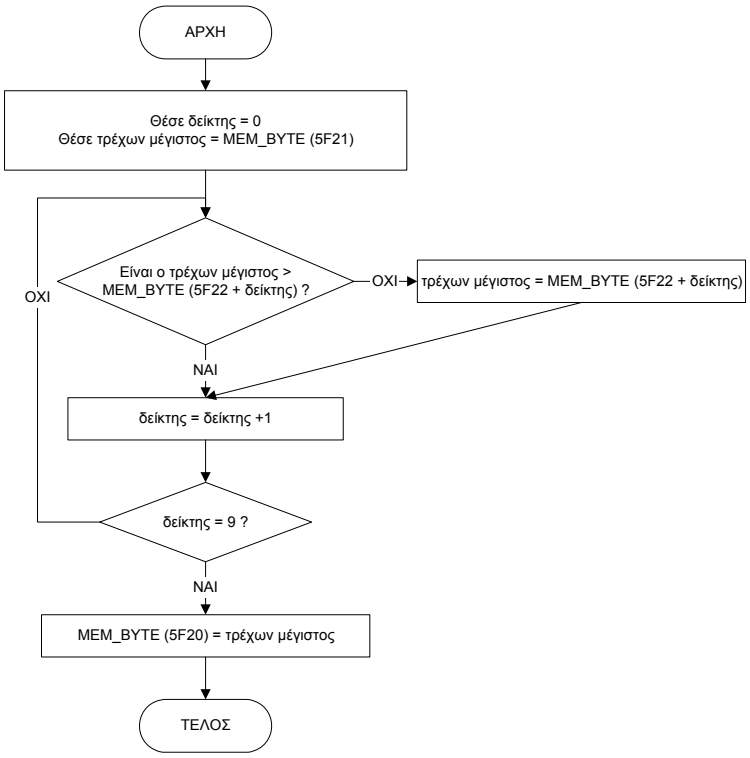
ST: Το δυαδικό ψηφίο ST *(STicky bit)* τίθεται στο 1 ώστε να υποδεικνύει ότι κατά τη διάρκεια μίας δεξιάς ολίσθησης ένας άσσος (1) έχει ολισθήσει πρώτα στον ενδείκτη κρατουμένου C και μετά προς τα έξω. Ο ενδείκτης ST μαζί με τον ενδείκτη κρατουμένου C μπορούν να χρησιμοποιηθούν για να αποφασίσουν για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