Τίτλος Μαθήματος

ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΤΕ

ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΥΠΟΛΟΓΙΣΤΩΝ

Καθηγητής Σπύρος Καζαρλής

ΣΕΡΡΕΣ, ΣΕΠΤΕΜΒΡΙΟΣ 2015

Άδειες Χρήσης

Χρηματοδότηση

Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.

Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο ΤΕΙ Κεντρικής Μακεδονίας**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.

Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

ΤΕΙ ΣΕΡΡΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΕΠΙΚΟΙΝΩΝΙΩΝ

ΣΗΜΕΙΩΣΕΙΣ ΘΕΩΡΙΑΣ
για το μάθημα
ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΥΠΟΛΟΓΙΣΤΩΝ

Δρ. Σπύρος Α. Καζαρλής
Αναπληρωτής Καθηγητής

Με την πολύτιμη συμβολή των συνεργατών
Ιωάννη Μαδεμλή , και
Κωνσταντίνου Δομουχτσή

Σέρρες, Ιούλιος 2008

Κατάλογος περιεχομένων

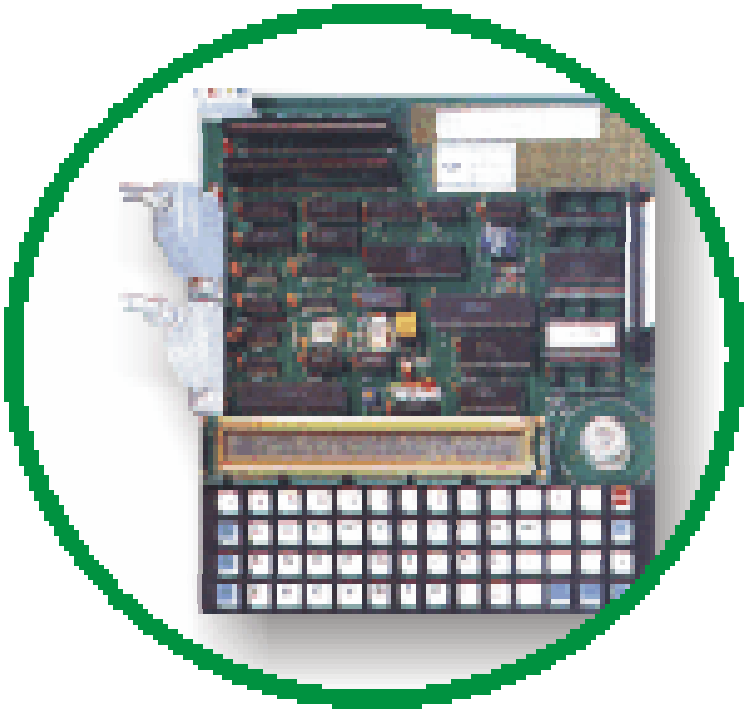
ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΥΠΟΛΟΓΙΣΤΩΝ

Κεφάλαια του Μαθήματος

1. Εισαγωγή στον Η/Υ BGC-8088 και τον Μ/Ε Intel 8088
2. Ιστορία και εξέλιξη των Υπολογιστών
3. Βασικά και σύνθετα ψηφιακά κυκλώματα (Πύλες, ολοκληρωμένα κυκλώματα, συνδυαστικά κυκλώματα, αθροιστές, ALU, κυκλώματα ρολογιού, μνήμες)
4. Συστήματα με Μικροεπεξεργαστές (ιστορική εξέλιξη, εσωτερική αρχιτεκτονική, μέθοδοι κατασκευής, είδη υπολογιστών) εσωτερική δομή Μικροεπεξεργαστών (καταχωρητές, ALU, μονάδα ελέγχου, δίαυλος, μνήμη cache, τεχνολογίες)
5. Εντολές γλώσσας μηχανής (κύκλοι εντολών, φάσεις εκτέλεσης, ανάλυση εντολών σε βήματα μικροκώδικα, κατηγορίες εντολών, παραλληλία εντολών, CISC και RISK)
6. Σήματα ελέγχου (μνήμης, περιφερειακών συσκευών, DMA, σήματα διακοπής, σήματα κατάστασης, λοιπά σήματα ελέγχου), Διασύνδεση CPU και περιφερειακών συσκευών (DMA, INT)
7. Μνήμη (Ιεραρχία, οργάνωση μνήμης, ανάγνωση/εγγραφή, είδη μνήμης RAM-ROM, τεχνολογίες, κώδικες διόρθωσης λαθών)
8. Μνήμες ROM και RAM (τεχνολογίες κατασκευής, είδη μνήμης). Κώδικες ανίχνευσης και διόρθωσης σφαλμάτων μνήμης.
9. Η μνήμη cache (Αρχές λειτουργίας, μέθοδοι σχεδίασης-υλοποίησης)
10. Δίαυλοι επικοινωνίας (ISA, PCI, PCMCIA, USB, FireWire,AGP)

Κεφάλαιο 1. Εισαγωγή στον Η/Υ BGC-8088 και τον Μ/Ε Intel 8088

1.1. Ο Εκπαιδευτικός Η/Υ BGC-8088



Ο Εκπαιδευτικός Αναπτυξιακός Υπολογιστής BGC-8088 MicroEngineer V.3 είναι κατασκευή της εταιρείας MICROPORT. Βασίζεται στον μικροεπεξεργαστή INTEL 8088-2, που αποτελεί την έκδοση διπλού χρονισμού (λειτουργεί στα 4.77 MHz και τα 8 MHz) του απλούστερου INTEL 8088 (4.77 MHz) που ήταν ο επεξεργαστής του πρώτου IBM PC, στον οποίο βασίζονται οι σύγχρονοι υπολογιστές με επεξεργαστές της σειράς x86 και Pentium της INTEL

Ο μικροεπεξεργαστής (Μ/Ε) 8088 είναι ο πρόγονος όλων των Μ/Ε 80x86 και Pentium της Intel, οι οποίοι είναι όλοι προς τα πίσω συμβατοί με αυτόν. αυτό σημαίνει ότι ένα πρόγραμμα γραμμένο σε γλώσσα μηχανής του 8088 μπορεί να εκτελεστεί ο χωρίς καμία τροποποίηση σε όλους τους επόμενους επεξεργαστές της να παίρνουν. Έτσι τα προγράμματα που φτιάχνουμε για τον 8088 μπορούν κάλλιστα να εκτελεστούν και στα desktop PC ή laptop που έχουμε και στο σπίτι.

Ο BGC-8088 έχει 32K RAM και 16K ROM που μπορούν να επεκταθούν με άλλα 16 K ROM με προγράμματα του χρήστη.

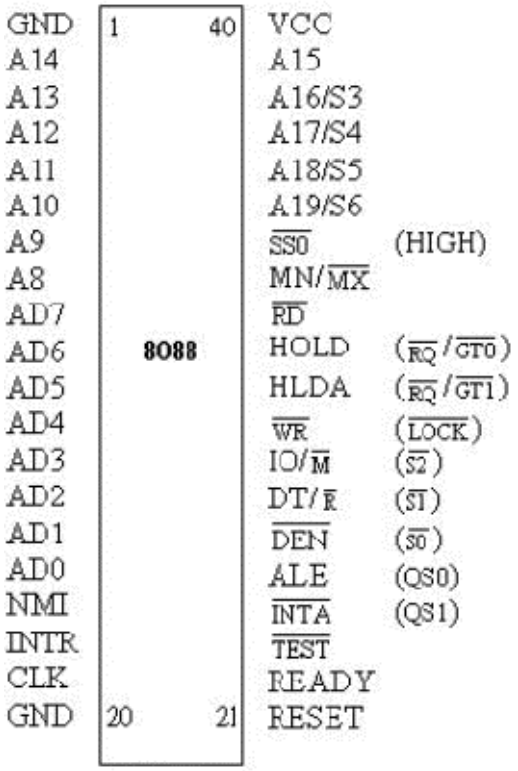
Η επικοινωνία με τον χρήστη γίνεται μέσω ενός πληκτρολογίου 56 πλήκτρων σε διάταξη QWERTY που περιλαμβάνει όλους τους εκτυπώσιμους χαρακτήρες και σύμβολα, καθώς και πλήκτρα λειτουργιών και ελέγχου, και μίας LCD οθόνης 2 γραμμών και 40 χαρακτήρων.

Προαιρετικά μπορεί να δεχθεί κάρτα HERCULES καθώς διαθέτει υποδοχές προτύπου ISA όπως και ο IBM-PC. Η κάρτα HERCULES είναι μια κάρτα γραφικών που προσφέρει ανάλυση 80 χαρακτήρων και 24 γραμμών με μονόχρωμη απεικόνιση. Με τη χρήση της κάρτας αυτής και με τη χρήση έτοιμων ρουτινών βιβλιοθήκης που είναι αποθηκευμένες στη μνήμη ROM του υπολογιστή, η εμφάνιση στην οθόνη μπορεί να προσομοιάσει αυτή των

κανονικών υπολογιστών, καθιστώντας την εργασία με τον ηλεκτρονικό υπολογιστή πολύ πιο άνετη.
Διαθέτει 2 υποδοχές ISA των 62 pins για σύνδεση καρτών επέκτασης ISA των 8 bit.
Διαθέτει μία σειριακή θύρα επικοινωνίας RS232-C, και μία παράλληλη θύρα επικοινωνίας (πχ. για σύνδεση εκτυπωτή), καθώς και ειδική θύρα σύνδεσης εκπαιδευτικών πλακετών των 50 pin (address, data, control). Σε αυτή την θύρα εξάγονται όλες οι γραμμές του διαύλου διευθύνσεων, του διαύλου δεδομένων αλλά και του διαύλου ελέγχου, και χρησιμοποιείται για σύνδεση επιπλέον περιφερειακών συσκευών, όπως καρτών Multi-I/O.
Διαθέτει επίσης έναν Προγραμματιζόμενο Χρονιστή (Prorammable Interval Timer – PIT) 8254 της INTEL που χρησιμοποιείται για παραγωγή σημάτων χρονισμού και ως μετρητής χρόνου (counter), ένα chip Προγραμματιζόμενης Διεπαφής Περιφερειακών (Programmable Peripheral Interface – PPI) 8255 της INTEL που παρέχει 3 θύρες Εισόδου / Εξόδου των 8-bit πλήρως προγραμματιζόμενες, και έναν Προγραμματιζόμενο Ελεγκτή Διακοπών (Programmable Interrupt Controller – PIC) 8259A της INTEL που παρέχει 8 γραμμές διακοπών interrupts.
Τέλος διαθέτει ειδική θύρα σύνδεσης εκπαιδευτικών πλακετών των 50 pin όπου συνδέονται όλες οι γραμμές του data bus, του address bus και οι γραμμές ελέγχου (control bus) του BGC-8088.

1.2. Ο μικροεπεξεργαστής Intel 8088

Ο 8088 είναι ένας Μ/Ε τεχνολογίας HMOS, με εσωτερική αρχιτεκτονική των 16 bit (χρησιμοποιεί καταχωρητές των 16 bit) αλλά έχει Δίαυλο Δεδομένων (Data Bus) των 8 bit για επικοινωνία με την μνήμη και τις Π.Σ., κάτι που κάνει πιο φθηνή την κατασκευή μητρικών (motherboards) και περιφερειακών σε σχέση με τον 8086 που έχει Data Bus των 16 bit. Με την επιλογή τους αυτή οι τεχνικοί της εταιρείας INTEL θέλησαν να δώσουν μεγαλύτερο βάρος στην ελαχιστοποίηση του κόστους κατασκευής μητρικών πακέτων αλλά και περιφερειακών συσκευών για τον επεξεργαστή 8088 θυσιάζοντας την ταχύτητα του διαύλου δεδομένων, ο οποίος έχει το μισό εύρος από ό,τι οι καταχωρητές του επεξεργαστή.
Επίσης έχει Δίαυλο Διευθύνσεων (Address Bus) των 20 bit, οπότε και μπορεί να απευθυνθεί σε 2^{20} = 1048576 = 1MB μνήμη (RAM και ROM).
Διαθέτει 14 καταχωρητές των 16 bit για γενικές και ειδικές λειτουργίες, μερικοί από τους οποίους μπορούν να χωρισθούν σε δύο καταχωρητές των 8-bit.
Διαθέτει 90 συνολικά εντολές γλώσσας μηχανής που έχουν 24 συνολικά διαφορετικούς τρόπους σύνταξης (addressing modes). Περιλαμβάνουν αριθμητικές πράξεις στα 8 ή 16 bit, με πρόσημο ή χωρίς, (περιλαμβ. πολ/σμος και διαίρεση). Στο σύνολο εντολών του επεξεργαστή περιλαμβάνονται επίσης και πολύπλοκες αριθμητικές εντολές όπως ο πολλαπλασιασμός ακεραίων αριθμών και η διαίρεση ακεραίων αριθμών, με πρόσημο και χωρίς πρόσημο.



Ο Μικροεπεξεργαστής INTEL 8088 και τα pin του.

1.3. Οι καταχωρητές του 8088

Διαθέτει 14 καταχωρητές των 16 bit για γενικές και ειδικές λειτουργίες, μερικοί από τους οποίους μπορούν να χωρισθούν σε δύο καταχωρητές των 8-bit. Οι καταχωρητές του 8088 είναι οι ακόλουθοι :

Ονομα Καταχωρητή	Δομή		Χρήση
AX	**AH**	**AL**	Accumulator (Συσσωρευτής)
BX	**BH**	**BL**	Base (Καταχωρητής Βάσης)
CX	**CH**	**CL**	Counter (Μετρητής)
DX	**DH**	**DL**	Data (Δεδομένα)
BP			Base Pointer (Δείκτης Βάσης)
SI			Source Index (Δείκτης Πηγής)
DI			Destination Index (Δείκτης Κατεύθυνσης)
SP			Stack Pointer (Δείκτης Στοίβας)
IP			Instruction Pointer (Δείκτης Εντολής)
CS			Code Segment (Τμήμα Κώδικα)
DS			Data Segment (Τμήμα Δεδομένων)
SS			Stack Segment (Τμήμα Στοίβας)
ES			Extra Segment (Εξτρα Τμήμα)
FG			FlaG Register (Καταχωρητής Σημαιών)

1.4. Καταχωρητές γενικής και ειδικής χρήσης

Οι **καταχωρητές AX, BX, CX, DX** μπορούν να θεωρηθούν και ως διπλοί καταχωρητές των 2x8 bit. Για παράδειγμα ο χρήστης μπορεί να προσπελάσει τον καταχωρητή AX και μέσω των δύο 8-bit καταχωρητών AH και AL. Ο AH αντιστοιχεί στο υψηλότερης τάξης τμήμα του AX (δηλαδή τα bits 8-15) ενώ ο AL αντιστοιχεί στο χαμηλότερης τάξης τμήμα του AX (δηλαδή τα bits 0-7). Το ίδιο συμβαίνει αντίστοιχα και για τους καταχωρητές BX, CX και DX. Οι καταχωρητές αυτοί είναι γενικής χρήσης, δηλαδή μπορούν να μεταφέρουν δεδομένα προς και από τη μνήμη, να εκτελέσουν αριθμητικές πράξεις κλ.π. Ο Συσσωρευτής όμως (AX) είναι πιο σημαντικός από τους υπόλοιπους καθώς συγκεκριμένες εντολές και τρόποι προσπέλασης μνήμης εκτελούνται μόνο με αυτόν, όπως ο πολλαπλασιασμός και η διαίρεση.
Οι καταχωρητές **CS, DS, SS, ES είναι καταχωρητές τμημάτων** (segment registers) και χρησιμοποιούνται για την προσπέλαση μνήμης του 8088 η οποία γίνεται με τμήματα (segments) και μετατοπίσεις (offsets).

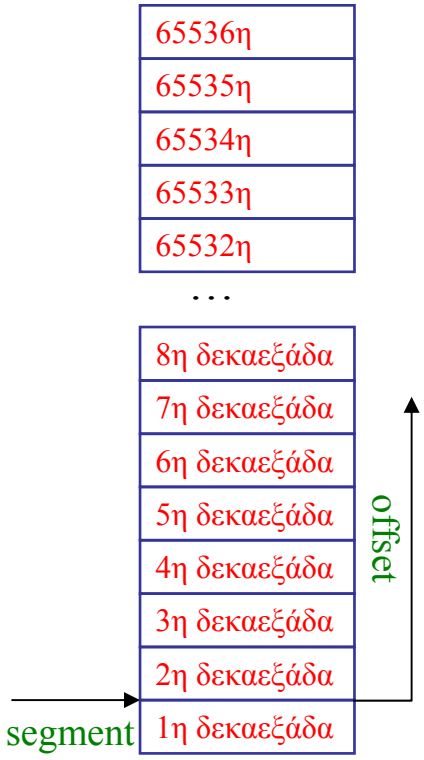
1.5. Η προσπέλαση μνήμης στον 8088

Ο επεξεργαστής 8088 μπορεί να προσπελάσει 1MB μνήμης έχοντας address bus με 20 γραμμές διευθύνσεων (2^{20}=1048576). Για συμβατότητα όμως με τον προηγούμενο επεξεργαστή της INTEL, τον 8080, κατασκευάστηκε με καταχωρητές διευθύνσεων των 16 bits (216=65536). Έτσι επινοήθηκε η μέθοδος των παραγράφων (paragraphs), σύμφωνα με την οποία, κάθε διεύθυνση των 16 bits δεν αναφέρεται σε απόλυτη διεύθυνση αλλά στην αρχή μίας δεκαεξάδας από bytes (π.χ. η διεύθυνση 000116 αναφέρεται στην αρχή της πρώτης δεκαεξάδας μνήμης : 0001016=16, η 000216 στην αρχή της δεύτερης δεκαεξάδας μνήμης : 0002016=32 κ.λ.π.). Η τελευταία δεκαεξάδα είναι η FFFF16 που αναφέρεται στην διεύθυνση FFFF016 = 1048560.
Για την προσπέλαση οποιασδήποτε διεύθυνσης μνήμης εκτός από την διεύθυνση παραγράφου (segment) χρειάζεται και μία διεύθυνση μετατόπισης (offset). Έτσι οι διευθύνσεις σχηματίζονται ως segment:offset. Η μετατόπιση (offset) θα αρκούσε να είναι των 4 bits (24=16) έτσι ώστε να προσδιορίζει το byte της δεκαεξάδας. Για λόγους ομοιογένειας όμως το offset είναι επίσης των 16 bits και μπορεί να δώσει 216=65536 διαφορετικές διευθύνσεις πάνω από την αρχή της παραγράφου, και όχι μόνο 16.

Έτσι έχοντας το segment σταθερό και αλλάζοντας το offset, προφανώς «μπαίνουμε» και στον χώρο των επόμενων δεκαεξάδων της μνήμης. Από τα παραπάνω είναι κατανοητό ότι δεν είναι μονοσήμαντη η διευθυνσιοδότηση μνήμης με την μέθοδο segment:offset, δηλαδή για κάθε απόλυτη διεύθυνση μνήμης υπάρχουν πολλοί συνδυασμοί segment:offset που την προσδιορίζουν (π.χ. το 17ο byte μνήμης έχει διεύθυνση 0001:0001 αλλά και 0000:0011).
Για να βρούμε την απόλυτη διεύθυνση μνήμης ακολουθούμε τον απλό τύπο :

Απόλυτη διεύθυνση = segment X 16 + offset.

Έτσι οι καταχωρητες τμημάτων (CS,DS,SS,ES) φτιάχτηκαν για να καταχωρούν το κομμάτι ‘segment’ από την έκφραση «segment:offset» των διευθύνσεων μνήμης που σχηματίζονται στον επεξεργαστή κατά την διάρκεια της λειτουργίας του και της εκτέλεσης εντολών.



Η σημασία του segment και του offset

1.6. Τμήματα προγράμματος και καταχωρητές

Οι τεχνικοί της εταιρείας INTEL, μαζί με τις προδιαγραφές του επεξεργαστή, καθόρισαν και τις προδιαγραφές των προγραμμάτων που θα μπορούσε αυτός να εκτελέσει. Έτσι, όρισαν ότι κάθε εκτελούμενο πρόγραμμα θα έπρεπε να αποτελείται από τρία διαφορετικά τμήματα, τα οποία είναι :

1. Το τμήμα κώδικα (code segment) όπου βρίσκεται το ίδιο το πρόγραμμα,
2. Το τμήμα δεδομένων (data segment) όπου βρίσκονται οι μεταβλητές του προγράμματος (data) και,
3. Το τμήμα στοίβας (stack segment) που χρησιμοποιείται για την στοίβα του προγράμματος (stack) που είναι ένα τμήμα μνήμης το οποίο προσπελαύνεται από τον 8088 με δομή LIFO (Last in First out) και χρησιμοποιείται για προσωρινή αποθήκευση δεδομένων, όπως προσωρινή αποθήκευση καταχωρητών, πέρασμα παραμέτρων σε υπορουτίνες, κ.λ.π).

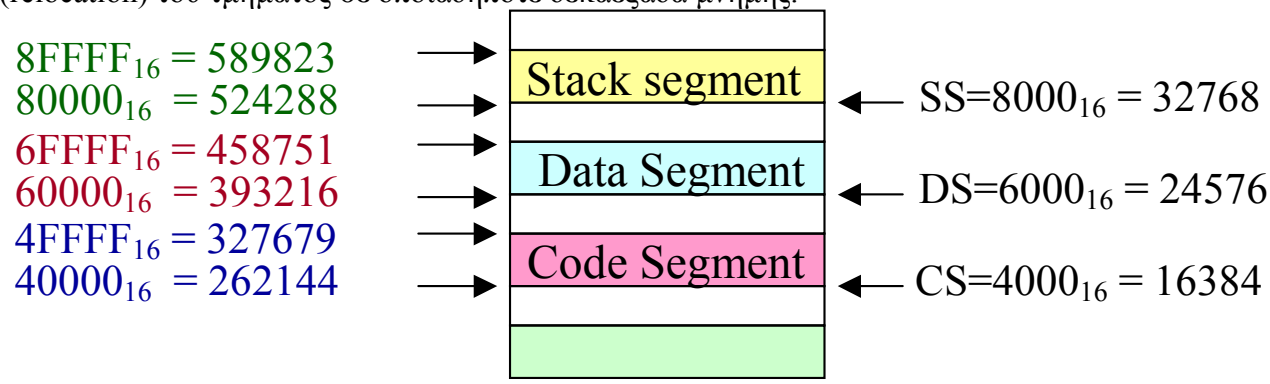
Έτσι ο 8088 έχει έναν καταχωρητή τμήματος για κάθε ένα από τα 3 αυτά τμήματα, που «δείχνει» την δεκαεξάδα στην οποία ξεκινάει το κάθε τμήμα. Οι καταχωρητές αυτοί είναι :

1. Ο CS (code segment register) που δείχνει στο τμήμα κώδικα,
2. Ο DS (data segment register) που δείχνει στο τμήμα δεδομένων, και
3. Ο SS (stack segment register) που δείχνει στο τμήμα στοίβας.

Ο τέταρτος καταχωρητής τμήματος που είναι ο ES (Extra segment register) χρησιμοποιείται ως «γενικής χρήσης» καταχωρητής τμήματος που μπορεί να «δείξει» σε οποιοδήποτε άλλο τμήμα (δεκαεξάδα) της μνήμης ώστε να προσπελάσει οποιαδήποτε διεύθυνση χωρίς να «χαλάσει» τις τιμές των CS,DS,SS που είναι αφιερωμένοι για άλλο σκοπό.

Το κάθε τμήμα (κώδικα, δεδομένων, στοίβας, έξτρα) έχοντας σταθερή διεύθυνση segment, μπορεί μέσω του μεταβλητού 16μπιτου offset να έχει μέγεθος μέχρι 64K (0000..FFFF).

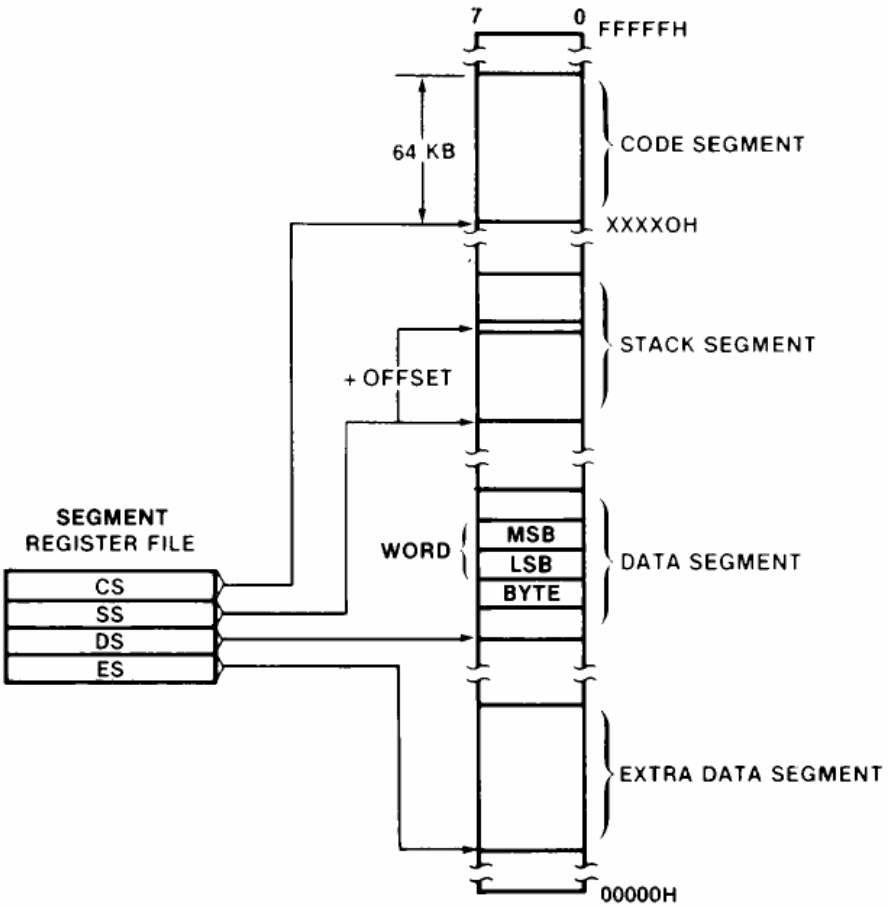
Αλλάζοντας τιμή στον segment register κάποιου τμήματος είναι δυνατή η επανατοποθέτηση (relocation) του τμήματος σε οποιαδήποτε δεκαεξάδα μνήμης.



Παράδειγμα τοποθέτησης των 3 τμημάτων ενός προγράμματος στη μνήμη και αντίστοιχες τιμές των καταχωρητών τμημάτων.

Στην επόμενη εικόνα που συμπεριλαμβάνεται στο εγχειρίδιο της Intel για τον 8088 φαίνονται χαρακτηριστικά τα 3 τμήματα προγράμματος καθώς και το «Εξτρα Τμήμα» στο ποίο μπορεί να δείχνει ο ES register Επίσης φαίνονται σχηματικά:

1. Το μέγεθος των τμημάτων που δεν ξεπερνά τα 64K
2. Η έννοια της μετατόπισης (offset)
3. Η σειρά με την οποία τοποθετούνται στην μνήμη τα bytes των αριθμών πολλαπλών byte, δηλαδή πρώτα το byte χαμηλής τάξης (LSB) και μετά το byte υψηλής τάξης (MSB).



Τμήματα προγράμματος και αντίστοιχοι καταχωρητές τμημάτων

1.7. Καταχωρητές Offset

Για την αποθήκευση του "offset" κομματιού της διεύθυνσης μνήμης που είναι σε μορφή «segment:offset» χρησιμοποιούνται άλλοι καταχωρητές του 8088. Για παράδειγμα, ο καταχωρητής IP (Instruction Pointer) κρατάει το 'offset' κομμάτι της διεύθυνσης που δείχνει πάντα την επόμενη εντολή που θα εκτελεστεί. Το 'segment' κομμάτι της διεύθυνσης το κρατάει ο καταχωρητής CS (code segment register). Έτσι η διεύθυνση μνήμης που σχηματίζεται από τους καταχωρητές CS:IP είναι πάντα η διεύθυνση της επόμενης εντολής κώδικα μηχανής που θα εκτελεστεί.
Σημείωση: στον BGC-8088 οι εξ'ορισμού τιμές για τους καταχωρητές τμημάτων είναι :

Α/Α	Καταχωρητής	Αρχική Τιμή
1	CS	0100
2	DS	0100
3	SS	0040
4	ES	0100

Λαμβάνοντας υπ' όψιν ότι η αρχική τιμή του καταχωρητή IP είναι 0000, συμπεραίνουμε ότι τα προγράμματα που γράφουμε στον BGC-8088 ξεκινούν εξ' ορισμού από την διεύθυνση : 0100:0000 που μεταφράζεται σε απόλυτη διεύθυνση : $01000_{16}=4096_{10}$.
Παράδειγμα 2: Η εντολή CMPSB συγκρίνει ένα byte ενός string από την διεύθυνση DS:SI με ένα byte ενός άλλου string στην διεύθυνση ES:DI, επηρεάζοντας τις σημαίες (flags). Εδώ οι SI (Source Index) και DI (Destination Index) έχουν καταχωρημένα τα offset.

Το ποιός καταχωρητής segment χρησιμοποιείται σε κάθε προσπέλαση μνήμης είναι πολύ εύκολο να το ξεχωρίσουμε:

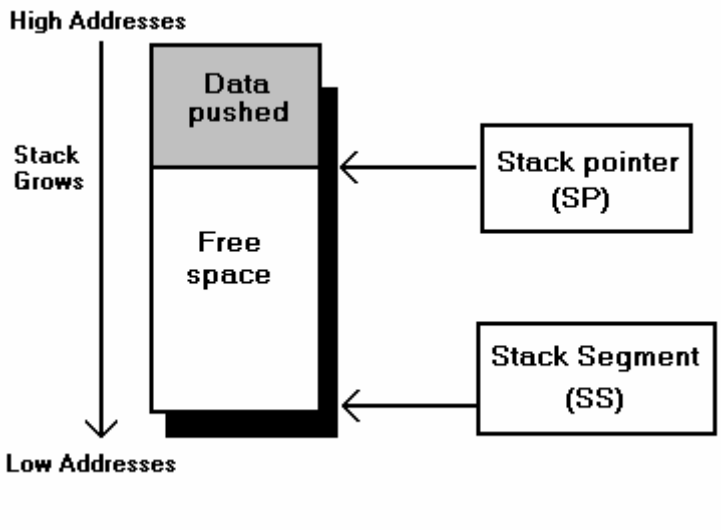
- Όταν προσπελαύνεται διεύθυνση μνήμης που αντιστοιχεί σε κώδικα (πρόγραμμα), τότε χρησιμοποιείται ο CS register. Παράδειγμα στην εντολή JMP 0300, επειδή η διεύθυνση 0300 αντιστοιχεί σε κάποιο κομμάτι του προγράμματος γλώσσας μηχανής, η πραγματική διεύθυνση διακλάδωσης του προγράμματος είναι CS:0300, δηλαδή ότι αριθμό έχει ο CS επί 16 συν το 0300_{16}.
- Όταν προσπελαύνεται διεύθυνση μνήμης που αντιστοιχεί σε δεδομένα, τότε χρησιμοποιείται ο DS register. Για παράδειγμα στην εντολή ADD AX,[1234] η οποία θα προσθέσει στο περιεχόμενο του καταχωρητή AX το περιεχόμενο της θέσης μνήμης [1234], η πραγματική διεύθυνση μνήμης που διαβάζεται είναι η DS:1234.
- Τέλος, όταν προσπελαύνεται διεύθυνση μνήμης που αντιστοιχεί στην στοίβα (stack), τότε χρησιμοποιείται ο καταχωρητής SS. Παράδειγμα στην εντολή PUSH BX το περιεχόμενο του καταχωρητή BX θα καταχωρηθεί στην πραγματική διεύθυνση SS:SP όπου ο SP (Stack Pointer) είναι ο καταχωρητής που δείχνει πάντοτε την «κορυφή» του stack.

1.8. Ειδικοί Καταχωρητές

Στο σετ των καταχωρητών του 8088 υπάρχουν και ειδικοί καταχωρητές όπως :

- Ο καταχωρητής IP (Instruction Pointer) που δείχνει την επόμενη εντολή που θα εκτελεστεί (μέσα στο Τμήμα Κώδικα – Code Segment)
- Ο καταχωρητής SP (Stack Pointer) ο οποίος δείχνει το σημείο μέχρι το οποίο έχει γεμίσει η στοίβα (stack) μέσα στο Τμήμα Στοίβας (Stack Segment). Ο καταχωρητής ξεκινά με

μεγάλη τιμή (0B3F) και όσο γεμίζει το stack αυτός μειώνεται (το stack γεμίζει από πάνω προς τα κάτω, δηλαδή από μεγάλες διευθύνσεις προς μικρότερες).



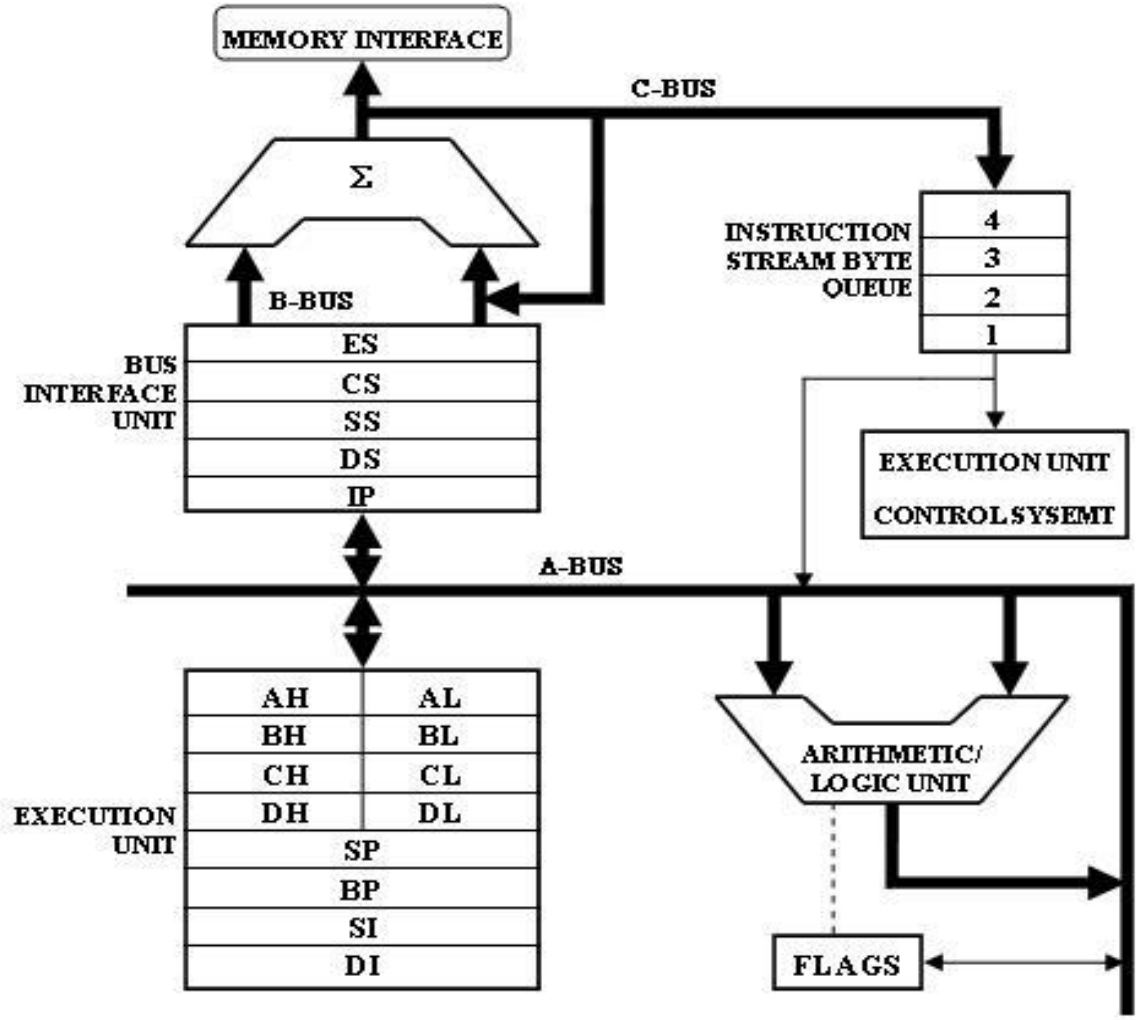
1.9. Ο Καταχωρητής Σημαιών

Ο καταχωρητής FG (Flag Register) περιέχει σημαίες (flags) που δείχνουν ή καθορίζουν την κατάσταση του επεξεργαστή. Κάθε σημαία (flag) έχει μέγεθος 1 bit καθώς αναπαριστά μία διακοπτική ένδειξη ή λειτουργία που μπορεί να είναι ON ή OFF. Στον καταχωρητή FG δεν έχει σημασία η τιμή του ως 16-bit δυαδικό νούμερο, αλλά το κάθε bit ξεχωριστά και ανεξάρτητα από τα υπόλοιπα, που έχει και την δική του ξεχωριστή σημασία. Η σημασία των bits του Καταχωρητή Σημαιών του 8088 είναι η εξής :
Ως γνωστόν από τα 16 bits του FG χρησιμοποιούνται μόνο τα 9 που είναι :

Bit	Ονομασία	Συντ	Εξήγηση
0	Carry Flag	CY	Κρατούμενο, γίνεται 1 όταν αποτελέσματα πράξεων ξεπερνούν το όριο των 16 bits.
1	*Δεν χρησιμοποιείται*		
2	Parity Flag	PF	Σημαία Ισοτιμίας, γίνεται 1 όταν το αποτέλεσμα μίας πράξης έχει ζυγό αριθμό μονάδων.
3	*Δεν χρησιμοποιείται*		
4	Auxiliary Carry	AF	Βοηθητικό Κρατούμενο, γίνεται 1 όταν μεταφέρεται κρατούμενο από το byte χαμηλής τάξης στο byte υψηλής τάξης (low nibble carry)
5	*Δεν χρησιμοποιείται*		
6	Zero Flag	ZF	Σημαία Μηδενός, γίνεται 1 όταν το αποτέλεσμα μίας πράξης π.χ. ADD σε οποιοδήποτε καταχωρητή, είναι ίσο με 0.
7	Sign Flag	SF	Σημαία Προσήμου, γίνεται 1 όταν το αποτέλεσμα μίας πράξης είναι αρνητικός αριθμός.
8	Trap Flag	TF	Σημαία Βηματικής Εκτέλεσης, όταν είναι 1 εκτελεί τις εντολές βήμα-βήμα για debugging.

9	Interrupt Flag	IF	Σημαία Αποδοχής Διακοπών, όταν είναι 1 επιτρέπεται η διακοπή του προγ/τος από interrupts
10	Direction Flag	DF	Σημαία Κατεύθυνσης, 1 = οι εντολές string εκτελούνται από υψηλές δ/νσεις προς χαμηλές.
11	Overflow Flag	OF	Σημαία Υπερχείλισης, γίνεται 1 όταν το αποτέλε-σμα μίας πράξης ξεπερνά το όριο των προσημασμένων αριθμών δηλαδή -32768…+32767
12	Δεν χρησιμοποιείται		
13	Δεν χρησιμοποιείται		
14	Δεν χρησιμοποιείται		
15	Δεν χρησιμοποιείται		

1.10. Λειτουργικό Διάγραμμα Καταχωρητών



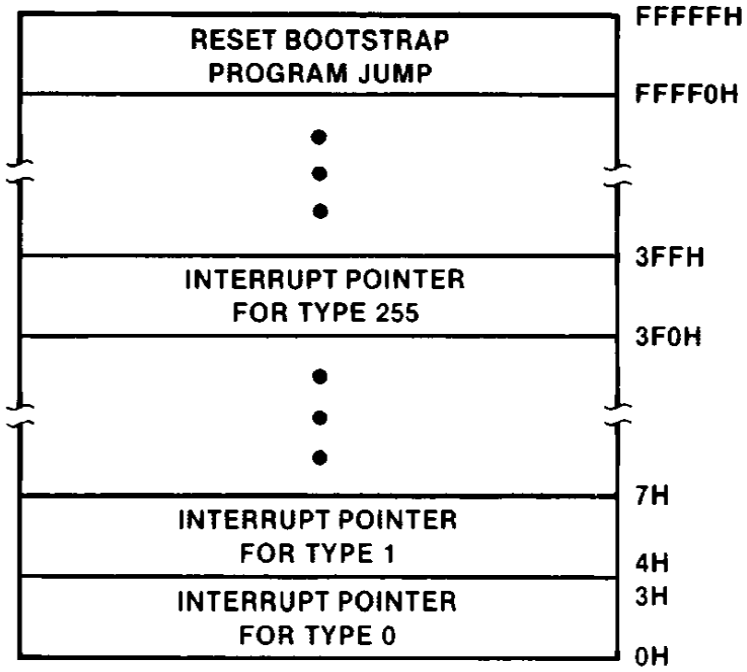
Στο παραπάνω διάγραμμα φαίνεται καθαρά ότι : Οι καταχωρητές γενικής χρήσεως καθώς και οι καταχωρητές δείκτη βρίσκονται τοπολογικά κοντά και συνδέονται μεταξύ τους με έναν δίαυλο που ονομάζεται A-Bus. Στον ίδιο δίαυλο συνδέεται η αριθμητική και λογική μονάδα ALU η οποία κάνει τις πράξεις των ακεραίων. Επίσης βλέπουμε ότι οι

καταχωρητές τμημάτων που σχετίζονται με τις διευθύνσεις είναι επίσης τοπολογικά κοντά και συνδέονται μεταξύ τους με έναν άλλο δίαυλο που ονομάζεται B-Bus. Στον ίδιο δίαυλο βλέπουμε να συνδέεται μία μονάδα πρόσθεσης η οποία στην ουσία βρίσκεται μέσα στην αριθμητική και λογική μονάδα και η οποία χρησιμοποιείται για να προσθέτει τις διευθύνσεις βάσης και τους δείκτες ώστε να προκύπτουν οι τελικές διευθύνσεις όταν χρησιμοποιούνται δεικτοδοτούμενες διευθυνσιοδοτήσεις π.χ. MOV CX,[0200+SI].

1.11. Η μνήμη του BGC-8088

Ο BGC-8088 έχει 32K RAM και 16K ROM επεκτάσιμα με άλλα 16 K ROM με προγράμματα του χρήστη. Θεωρητικά θα μπορούσε να φιλοξενήσει έως και 1MB μνήμη (RAM + ROM).
Η RAM μνήμη είναι χαρτογραφημένη στις διευθύνσεις 00000_{16} … $07FFF_{16}$ (0…32767). Οι πρώτες 4096 διευθύνσεις είναι δεσμευμένες και χρησιμοποιούνται από το πρόγραμμα MONITOR (00000_{16} … $00FFF_{16}$). Από αυτές οι πρώτες 1024 διευθύνσεις (00000_{16} … $003FF_{16}$) περιέχουν τις διευθύνσεις των ρουτινών εξυπηρέτησης των διακοπών (vectors of interrupt handler routines – interrupt vectors).
Οι διακοπές μπορεί να είναι μέχρι και 256. Για κάθε ρουτίνα η διεύθυνσή της αποτελείται από 4 byte (2 byte segment + 2 byte offset). Για παράδειγμα η διεύθυνση της ρουτίνας εξυπηρέτησης της διακοπής 85 (INT 85), που όταν εκτελεστεί επανεκκινεί το πρόγραμμα MONITOR, βρίσκεται στην θέση $00214_{16} = 85_{16}$ x 4. Εκεί υπάρχουν διαδοχικά τα bytes 9F 01 00 FC. Τα πρώτα 2 είναι low και high bytes του offset, και τα 2 επόμενα τα low και high bytes του segment. Έτσι η πραγματική διεύθυνση της ρουτίνας εξυπηρέτησης της διακοπής 85 είναι η FC00:019F. Η διεύθυνση αυτή είναι στην ROM του μηχανήματος.
Οι διευθύνσεις των ρουτινών εξυπηρέτησης των διακοπών βρίσκονται στη μνήμη RAM. Επειδή όμως κατά την έναρξη λειτουργίας του ηλεκτρονικού υπολογιστή η μνήμη RAM είναι κενή ή περιέχει σκουπίδια, θα πρέπει προφανώς με κάποιον τρόπον οι διευθύνσεις των ρουτινών να τοποθετούνται στις σωστές διευθύνσεις της μνήμης RAM. Αυτός που τις τοποθετεί στις σωστές διευθύνσεις κατά την εκκίνηση του ηλεκτρονικού υπολογιστή δεν είναι άλλος από το πρόγραμμα εκκίνησης που βρίσκεται σε μνήμη ROM και που στα PC ονομάζεται BIOS ενώ στον BGC-8088 ονομάζεται πρόγραμμα MONITOR. Οι διευθύνσεις των ρουτινών είναι τοποθετημένες στη RAM και όχι σε μόνιμη μνήμη ROM για έναν πολύ καλό λόγο: όταν μετά το πρόγραμμα εκκίνησης, φορτωθεί κάποιο λειτουργικό σύστημα, όπως τα Windows η το Linux, τότε αυτό έχει την δυνατότητα να αλλάξει τις διευθύνσεις των ρουτινών εξυπηρέτησης των διακοπών, εφόσον αυτές είναι στη RAM, και να βάλει δικές τους διευθύνσεις ρουτινών οι οποίες θα δείχνουν σε δικές του υπορουτίνες διαχείρισης των διακοπών. Οι ρουτίνες διαχείρισης των διακοπών των λειτουργικών συστημάτων πρώτα καλούν τις αντίστοιχες ρουτίνες της ROM και στη συνέχεια εκτελούν επιπλέον κώδικα που είναι απαραίτητος για την ομαλή λειτουργία του εκάστοτε λειτουργικού συστήματος.

Τα interrupt vectors και το jump εκκίνησης



Οι υπόλοιπες 28672 θέσεις μνήμης είναι διαθέσιμες για προγράμματα του χρήστη (01000_{16} … $07FFF_{16}$).
Για τον λόγο αυτό ο καταχωρητής CS έχει την εξ' ορισμού τιμή 0100 (0100:0000 = απόλυτη διεύθυνση $01000_{16} = 4096_{10}$).

1.12. Η μνήμη ROM του BGC-8088

Τα 16 KB της μνήμης ROM που περιλαμβάνουν το πρόγραμμα MONITOR και τους drivers είναι χαρτογραφημένα στις διευθύνσεις $FC000_{16}$ … $FFFFF_{16}$, δηλαδή «ψηλά» στην μνήμη στο όριο του 1 MB.
Τα 16K ROM που μπορεί να προσθέσει ο χρήστης (ελεύθερο slot μνήμης) χαρτογραφούνται στις διευθύνσεις $F8000_{16}$ … $FBFFF_{16}$, δηλαδή ακριβώς κάτω από την υπάρχουσα μνήμη ROM.
Κατά την εκκίνηση (εφαρμογή τροφοδοσίας) και κατά την επανεκκίνηση (reset) του μηχανήματος, ο 8088 αλλά και όλοι οι απόγονοί του στην σειρά x86 και Pentium της INTEL, ξεκινούν με την εκτέλεση της εντολής που βρίσκεται στην διεύθυνση $FFFF0_{16}$, που είναι στην ROM.
Η εντολή αυτή συνήθως είναι ένα JMP στην ρουτίνα εκκίνησης του συστήματος που ανήκει στο BIOS (του προγράμματος MONITOR στην περίπτωση του BGC-8088) που είναι σε μόνιμη μνήμη ROM.

1.13. Ο χάρτης μνήμης του BGC-8088

Στο παρακάτω σχήμα φαίνεται η πλήρης χαρτογράφηση της μνήμης του BGC-8088

Περιοχή Μνήμης	**Διευθύνσεις**	**Χώρος (bytes)**
Εντολή εκκίνησης του συστήματος	FFFFF FFFF0	00010_{16} 16_{10}
ROM που περιέχει το πρόγραμμα MONITOR και τους drivers	FFFEF FC000	$03FF0_{16}$ 16368_{10}
Χώρος για ROM με προγράμματα του χρήστη (ελεύθερο slot μνήμης)	FBFFF F8000	04000_{16} 16384_{10}
Χώρος διευθύνσεων που δεν αντιστοιχεί σε chip μνήμης	F7FFF …. 08000	$F0000_{16}$ 983040_{10}
Ελεύθερος χώρος για προγράμματα του χρήστη	07FFF … 01000	07000_{16} 28672_{10}
Περιοχή Μεταβλητών του προγράμματος MONITOR	00FFF 00400	$00C00_{16}$ 3072_{10}
Διευθύνσεις των ρουτινών εξυπηρέτησης των διακοπών – Interrupt Vectors	003FF 00000	00400_{16} 1024_{10}

1.14. Οι Εντολές Γλώσσας Μηχανής του 8088

Ο Intel 8088 έχει 90 συνολικά εντολές Γλώσσας Μηχανής. Κάθε τέτοια εντολή συντάσσεται με έως και 30 διαφορετικούς τρόπους. Κάθε Συνδυασμός Εντολής (Τρόπος Σύνταξης) έχει τον δικό του κωδικό εντολής (operation code ή opcode) και παραμέτρους που είναι συνήθως, σταθερά νούμερα, καταχωρητές ή θέσεις μνήμης.

Οι εντολές του 8088 χωρίζονται σε 6 κατηγορίες που είναι :

1. Εντολές Μεταφοράς δεδομένων (Data Transfer Commands)
2. Εντολές Αριθμητικών Πράξεων (Arithmetic Commands)
3. Εντολές Λογικών Πράξεων (Logic Commands)
4. Εντολές χειρισμού αλφαριθμητικών (String Manipulation Commands)
5. Εντολές Ελέγχου Ροής Προγράμματος (Program Flow Control Commands)
6. Εντολές Ελέγχου του Επεξεργαστή (Processor Control Commands)

1.14.1. Εντολές Μεταφοράς Δεδομένων

α/α	**Εντολή**	**Εξήγηση**	**Λειτουργία**
1	MOV	Move (Μετακίνηση)	Μεταφέρει δεδομένα μεταξύ καταχωρητών και μνήμης
2	PUSH	Push (Ώθησε στη στοίβα)	Ωθεί τους καταχωρητές στη στοίβα
3	POP	Pop (Ανέκτησε από στοίβα)	Ανακτά τιμές καταχωρητών από τη στοίβα

4	XCHG	Exchange (Ανταλλαγή)	Ανταλλάσσει τιμές καταχωρητών και μνήμης
5	IN	Input (Είσοδος από θύρα)	Διαβάζει δεδομένα από θύρα I/O
6	OUT	Output (Έξοδος σε θύρα)	Εξάγει δεδομένα σε θύρα I/O
7	XLAT	Translate (μετατροπή βάσει πίνακα αντιστοίχισης)	Ψάχνει σε look-up table στη θέση DS:BX+AL και αντικαθιστά τον AL
8	LAHF	Load AH with Flags	Βάζει τις σημαίες στον AH
9	SAHF	Store AH into Flags	Αντιγράφει τον AH στις σημαίες

1.14.2. Εντολές Αριθμητικών Πράξεων

α/α	**Εντολή**	**Εξήγηση**	**Λειτουργία**
1	ADD	Add (Πρόσθεση χωρίς κρ.)	Προσθέτει χωρίς κρατούμενο
2	ADC	Add with Carry (Πρόσθεση με κρατούμενο)	Προσθέτει με κρατούμενο
3	INC	Increment (Αύξηση)	Αυξάνει καταχωρητές κατά 1
4	SUB	Subtract (Αφαίρεση χωρίς δανεικό)	Αφαιρεί χωρίς να χρησιμοποιεί δανεικό
5	SBB	Subtract with Borrow	Αφαιρεί με χρήση δανεικού
6	DEC	Decrement (Μείωση)	Μειώνει καταχωρητές κατά 1
7	NEG	Negative (Αρνητικό)	Αλλάζει το πρόσημο του αριθμού
8	CMP	Compare (Σύγκριση)	Συγκρίνει τα περιεχόμενα καταχωρητών και μνήμης
9	MUL	Multiply (Πολλαπλασιασμός)	Πολλαπλασιάζει θετικούς αριθμούς, χωρίς πρόσημο
10	IMUL	Integer Multiply (Πολ/μος αριθμών με πρόσημο)	Πολλαπλασιάζει ακέραιους προσημασμένους αριθμούς
11	DIV	Division (Διαίρεση)	Διαιρεί ακέραιους θετικούς αριθμούς χωρίς πρόσημο
12	IDIV	Integer Division (Διαίρεση αριθμών με πρόσημο)	Διαιρεί ακέραιους προσημασμένους αριθμούς
13	CBW	Convert Byte to Word	Μετατρέπει έναν αριθμό από 1 byte σε 2
14	CWD	Convert Word to Double Word	Μετατρέπει έναν αριθμό από 2 byte σε 4 byte

1.14.3. Εντολές Λογικών Πράξεων

α/α	**Εντολή**	**Εξήγηση**	**Λειτουργία**
1	AND	And (Λογικό ΚΑΙ)	Εκτελεί λογικό ΚΑΙ μεταξύ bits

2	OR	Or (Λογικό Ή)	Εκτελεί λογικό Ή μεταξύ bits
3	XOR	Xor (Αποκλειστικό Ή)	Εκτελεί αποκλειστικό Ή μεταξύ bits
4	TEST	Test (Έλεγχος bits)	Εκτελεί λογικό ΚΑΙ μεταξύ bits χωρίς να καταχωρεί το αποτέλεσμα
5	NOT	Not (Λογική Άρνηση)	Αντιστρέφει τα bits του αριθμού
6	SHL	Shift Logical Left (μετακίνηση bits αριστερά)	Μετακινεί τα bits αριστερά συμπληρώνοντας μηδενικά
7	SHR	Shift Logical Right (μετακίνηση bits δεξιά)	Μετακινεί τα bits δεξιά συμπληρώνοντας μηδενικά
8	SAR	Shift Arithmetic Right (αριθμ.μετακ. bits δεξιά)	Μετακινεί τα bits δεξιά κρατώντας το πρόσημο
9	ROL	Rotate Left (Περιστροφή αριστερά)	Περιστρέφει τα bits του αριθμού 1 θέση αριστερά (το τελευταίο πάει πρώτο)
10	ROR	Rotate Right (Περιστροφή δεξιά)	Περιστρέφει τα bits του αριθμού 1 θέση δεξιά (το πρώτο πάει τελευταίο)
11	RCL	Rotate with Carry Left (Περ/φή αριστερά με κρατ.)	Περιστρέφει τα bits του αριθμού αρι-στερά με συμμετοχή του κρατούμενου
12	RCR	Rotate with Carry Right (Περ/φή δεξιά με κρατούμ.)	Περιστρέφει τα bits του αριθμού δεξιά με συμμετοχή του κρατούμενου

1.14.4. Εντολές Χειρισμού Αλφαριθμητικών

α/α	**Εντολή**	**Εξήγηση**	**Λειτουργία**
1	REP	Repeat (Επανέλαβε)	Επαναλαμβάνει την επόμενη εντολή
2	MOVS	Move String (Αντιγραφή String)	Αντιγράφει ένα String σε ένα άλλο byte προς byte
3	CMPS	Compare String (Σύγκριση String)	Συγκρίνει δύο String byte προς byte
4	SCAS	Scan String (Ανίχνευση String)	Ψάχνει ένα χαρακτήρα μέσα σε ένα string
5	LODS	Load String (Φόρτωση String)	Φορτώνει τα byte ενός string σε καταχωρητή
6	STOS	Store String (Αποθήκευση String)	Αποθηκεύει την τιμή ενός καταχωρητή σε ένα String.

1.14.5. Εντολές Ελέγχου Ροής Προγράμματος

α/α	**Εντολή**	**Εξήγηση**	**Λειτουργία**
1	CALL	Call (Κλήση υπορουτίνας)	Μεταφέρει την εκτέλεση σε υπορ/να

2	RET	Return (Επιστροφή από υπορουτίνα)	Επιστρέφει στο κυρίως πρόγραμμα μετά από την υπορουτίνα
3	JMP	Jump (Διακλάδωση)	Μεταφέρει την εκτέλεση σε άλλη εντολή
4	JE/JZ	Jump on Equal/Zero	Διακλάδωση σε ισότητα /μηδέν
5	JNE/JNZ	Jump on Not Equal/Non Zero	Διακλάδωση σε ανισότητα / διάφορο του μηδενός
6	JL/JNGE	Jump on Less/Not Greater or Equal	Διακλάδωση σε περίπτωση που ο 1ος είναι μικρότερος του 2ου (προσημασμ.)
7	JLE/JNG	Jump on Less or Equal/Not Greater	Διακλάδωση σε περίπτωση που ο 1ος είναι μικρότερος ή ίσος του 2ου (προσ.)
8	JNL/JGE	Jump on not Less/ Greater or Equal	Διακλάδωση σε περίπτωση που ο 1ος δεν είναι μικρότερος του 2ου (προσημασμ.)
9	JNLE/JG	Jump on not Less or Equal/ Greater	Διακλάδωση σε περίπτωση που ο 1ος είναι μεγαλύτερος του 2ου (προσημασμ.)
10	JB/JNAE	Jump on Below/Not Above or Equal	Διακλάδωση σε περίπτωση που ο 1ος είναι μικρότερος του 2ου (θετικοί)
11	JBE/JNA	Jump on Below or Equal/Not Above	Διακλάδωση σε περίπτωση που ο 1ος είναι μικρότερος ή ίσος του 2ου (θετικοί)
12	JNB/JAE	Jump on not Below / Above or Equal	Διακλάδωση σε περίπτωση που ο 1ος δεν είναι μικρότερος του 2ου (θετικοί)
13	JNBE/JA	Jump on not Below or Equal / Above	Διακλάδωση σε περίπτωση που ο 1ος είναι μεγαλύτερος του 2ου (θετικοί)
14	JP/JPE	Jump on Parity Even	Διακλάδωση σε Ζυγή Ισοτιμία
15	JNP/JPO	Jump on Parity Odd	Διακλάδωση σε Μονή Ισοτιμία
16	JS	Jump on Sign	Διακλάδωση σε αρνητικό αριθμό
17	JNS	Jump on not Sign	Διακλάδωση σε θετικό αριθμό
18	JCXZ	Jump on CX Zero	Διακλάδωση όταν ο CX γίνει 0
19	LOOP	Loop (Βρόχος CX)	Εκτελεί επαναληπτικά εντολές με απαριθμητή τον CX
20	LOOPZ	Loop Zero (Βρόχος CX με συνθήκη ισότητας)	Εκτελεί επαναληπτικά εντολές με απαριθμητή τον CX και όσο ισχύει σχέση ισότητας
21	LOOPNZ	Loop Zero (Βρόχος CX με συνθήκη ανισότητας)	Εκτελεί επαναληπτικά εντολές με απαριθμητή τον CX και όσο ισχύει σχέση ανισότητας
22	INT	Interrupt (Διακοπή)	Διακοπή Λογισμικού – Μεταφέρει την εκτέλεση σε ρουτίνα ROM
23	IRET	Interrupt Return (Επιστροφή από Διακοπή)	Επιστρέφει στο κυρίως πρόγραμμα μετά από ρουτίνα διακοπής.

1.14.6. Εντολές Ελέγχου του Επεξεργαστή

α/α	Εντολή	Εξήγηση	Λειτουργία
1	CLC	Clear Carry	Καθαρίζει το κρατούμενο (=0)
2	STC	Set Carry	Θέτει το κρατούμενο (=1)
3	CMC	Complement Carry	Αντιστρέφει το κρατούμενο
4	CLD	Clear Direction	Καθαρίζει τη σημαία κατεύθυνσης (=0)
5	STD	Set Direction	Θέτει τη σημαία κατεύθυνσης (=1)
6	CLI	Clear Interrupt	Καθαρίζει τη σημαία διακοπών (=0)
7	STI	Set Interrupt	Θέτει τη σημαία διακοπών (=1)
8	HALT	Halt (Σταμάτα)	Σταματά τον επεξεργαστή (στάση)
9	WAIT	Wait (Περίμενε)	Περιμένει για διαχείριση σφαλμάτων κινητής υποδιαστολής
10	ESC	Escape	Χορήγηση διαύλων σε άλλους Μ/Ε
11	LOCK	Lock	Κλείδωμα Διαύλων
12	NOP	No Operation	Καμία Ενέργεια

1.15. Οι κωδικοί των εντολών

Στις επόμενες σελίδες παραθέτονται οι εντολές του 8088 σε συνοπτική μορφή μαζί με τους κωδικούς των εντολών (opcodes).

8086/8088 Instruction Set Summary

Mnemonic and Description	Instruction Code			
DATA TRANSFER				
MOV = Move:	76543210	76543210	76543210	76543210
Register/Memory to/from Register	100010dw	mod reg r/m		
Immediate to Register/Memory	1100011w	mod 000 r/m	data	data if w = 1
Immediate to Register	1011 w reg	data	data if w = 1	
Memory to Accumulator	1010000w	addr-low	addr-high	
Accumulator to Memory	1010001w	addr-low	addr-high	
Register/Memory to Segment Register	10001110	mod 0 reg r/m		
Segment Register to Register/Memory	10001100	mod 0 reg r/m		
PUSH = Push:				
Register/Memory	11111111	mod 1 1 0 r/m		
Register	01010 reg			
Segment Register	000 reg 110			
POP = Pop:				
Register/Memory	10001111	mod 000 r/m		
Register	01011 reg			
Segment Register	000 reg 111			
XCHG = Exchange:				
Register/Memory with Register	1000011w	mod reg r/m		
Register with Accumulator	10010 reg			
IN = Input from:				
Fixed Port	1110010w	port		
Variable Port	1110110w			
OUT = Output to:				
Fixed Port	1110011w	port		
Variable Port	1110111w			
XLAT = Translate Byte to AL	11010111			
LEA = Load EA to Register	10001101	mod reg r/m		
LDS = Load Pointer to DS	11000101	mod reg r/m		
LES = Load Pointer to ES	11000100	mod reg r/m		
LAHF = Load AH with Flags	10011111			
SAHF = Store AH into Flags	10011110			
PUSHF = Push Flags	10011100			
POPF = Pop Flags	10011101			

8086/8088 Instruction Set Summary (Continued)

Mnemonic and Description	Instruction Code			
ARITHMETIC	76543210	76543210	76543210	76543210
ADD = Add:				
Reg./Memory with Register to Either	000000dw	mod reg r/m		
Immediate to Register/Memory	100000sw	mod 000 r/m	data	data if s:w = 01
Immediate to Accumulator	0000010w	data	data if w = 1	
ADC = Add with Carry:				
Reg./Memory with Register to Either	000100dw	mod reg r/m		
Immediate to Register/Memory	100000sw	mod 010 r/m	data	data if s:w = 01
Immediate to Accumulator	0001010w	data	data if w = 1	
INC = Increment:				
Register/Memory	1111111w	mod 000 r/m		
Register	01000reg			
AAA = ASCII Adjust for Add	00110111			
BAA = Decimal Adjust for Add	00100111			
SUB = Subtract:				
Reg./Memory and Register to Either	001010dw	mod reg r/m		
Immediate from Register/Memory	100000sw	mod 101 r/m	data	data if s:w = 01
Immediate from Accumulator	0010110w	data	data if w = 1	
SSB = Subtract with Borrow				
Reg./Memory and Register to Either	000110dw	mod reg r/m		
Immediate from Register/Memory	100000sw	mod 011 r/m	data	data if s:w = 01
Immediate from Accumulator	0001111w	data	data if w = 1	
DEC = Decrement:				
Register/memory	1111111w	mod 001 r/m		
Register	01001 reg			
NEG = Change sign	1111011w	mod 011 r/m		
CMP = Compare:				
Register/Memory and Register	001110dw	mod reg r/m		
Immediate with Register/Memory	100000sw	mod 111 r/m	data	data if s:w = 01
Immediate with Accumulator	0011110w	data	data if w = 1	
AAS = ASCII Adjust for Subtract	00111111			
DAS = Decimal Adjust for Subtract	00101111			
MUL = Multiply (Unsigned)	1111011w	mod 100 r/m		
IMUL = Integer Multiply (Signed)	1111011w	mod 101 r/m		
AAM = ASCII Adjust for Multiply	11010100	00001010		
DIV = Divide (Unsigned)	1111011w	mod 110 r/m		
IDIV = Integer Divide (Signed)	1111011w	mod 111 r/m		
AAD = ASCII Adjust for Divide	11010101	00001010		
CBW = Convert Byte to Word	10011000			
CWD = Convert Word to Double Word	10011001			

8086/8088 Instruction Set Summary (Continued)

Mnemonic and Description	Instruction Code			
LOGIC	76543210	76543210	76543210	76543210
NOT = Invert	1111011w	mod 010 r/m		
SHL/SAL = Shift Logical/Arithmetic Left	110100vw	mod 100 r/m		
SHR = Shift Logical Right	110100vw	mod 101 r/m		
SAR = Shift Arithmetic Right	110100vw	mod 111 r/m		
ROL = Rotate Left	110100vw	mod 000 r/m		
ROR = Rotate Right	110100vw	mod 001 r/m		
RCL = Rotate Through Carry Flag Left	110100vw	mod 010 r/m		
RCR = Rotate Through Carry Right	110100vw	mod 011 r/m		
AND = **And:**				
Reg./Memory and Register to Either	001000dw	mod reg r/m		
Immediate to Register/Memory	1000000w	mod 100 r/m	data	data if w = 1
Immediate to Accumulator	0010010w	data	data if w = 1	
TEST = **And Function to Flags. No Result:**				
Register/Memory and Register	1000010w	mod reg r/m		
Immediate Data and Register/Memory	1111011w	mod 000 r/m	data	data if w = 1
Immediate Data and Accumulator	1010100w	data	data if w = 1	
OR = **Or:**				
Reg./Memory and Register to Either	000010dw	mod reg r/m		
Immediate to Register/Memory	1000000w	mod 001 r/m	data	data if w = 1
Immediate to Accumulator	0000110w	data	data if w = 1	
XOR = **Exclusive or:**				
Reg./Memory and Register to Either	001100dw	mod reg r/m		
Immediate to Register/Memory	1000000w	mod 110 r/m	data	data if w = 1
Immediate to Accumulator	0011010w	data	data if w = 1	
STRING MANIPULATION				
REP = Repeat	1111001z			
MOVS = Move Byte/Word	1010010w			
CMPS = Compare Byte/Word	1010011w			
SCAS = Scan Byte/Word	1010111w			
LODS = Load Byte/Wd to AL/AX	1010110w			
STOS = Stor Byte/Wd from AL/A	1010101w			
CONTROL TRANSFER				
CALL = **Call:**				
Direct Within Segment	11101000	disp-low	disp-high	
Indirect Within Segment	11111111	mod 010 r/m		
Direct Intersegment	10011010	offset-low	offset-high	
		seg-low	seg-high	
Indirect Intersegment	11111111	mod 011 r/m		

8086/8088 Instruction Set Summary (Continued)

Mnemonic and Description	Instruction Code		
JMP = Unconditional Jump:	76543210	76543210	76543210
Direct Within Segment	11101001	disp-low	disp-high
Direct Within Segment-Short	11101011	disp	
Indirect Within Segment	11111111	mod 100 r/m	
Direct Intersegment	11101010	offset-low	offset-high
		seg-low	seg-high
Indirect Intersegment	11111111	mod 101 r/m	
RET = Return from CALL:			
Within Segment	11000011		
Within Seg Adding Immed to SP	11000010	data-low	data-high
Intersegment	11001011		
Intersegment Adding Immediate to SP	11001010	data-low	data-high
JE/JZ = Jump on Equal/Zero	01110100	disp	
JL/JNGE = Jump on Less/Not Greater or Equal	01111100	disp	
JLE/JNG = Jump on Less or Equal/ Not Greater	01111110	disp	
JB/JNAE = Jump on Below/Not Above or Equal	01110010	disp	
JBE/JNA = Jump on Below or Equal/ Not Above	01110110	disp	
JP/JPE = Jump on Parity/Parity Even	01111010	disp	
JO = Jump on Overflow	01110000	disp	
JS = Jump on Sign	01111000	disp	
JNE/JNZ = Jump on Not Equal/Not Zero	01110101	disp	
JNL/JGE = Jump on Not Less/Greater or Equal	01111101	disp	
JNLE/JG = Jump on Not Less or Equal/ Greater	01111111	disp	
JNB/JAE = Jump on Not Below/Above or Equal	01110011	disp	
JNBE/JA = Jump on Not Below or Equal/Above	01110111	disp	
JNP/JPO = Jump on Not Par/Par Odd	01111011	disp	
JNO = Jump on Not Overflow	01110001	disp	
JNS = Jump on Not Sign	01111001	disp	
LOOP = Loop CX Times	11100010	disp	
LOOPZ/LOOPE = Loop While Zero/Equal	11100001	disp	
LOOPNZ/LOOPNE = Loop While Not Zero/Equal	11100000	disp	
JCXZ = Jump on CX Zero	11100011	disp	
INT = Interrupt			
Type Specified	11001101	type	
Type 3	11001100		
INTO = Interrupt on Overflow	11001110		
IRET = Interrupt Return	11001111		

8086/8088 Instruction Set Summary (Continued)

Mnemonic and Description	Instruction Code	
	76543210	76543210
PROCESSOR CONTROL		
CLC = Clear Carry	11111000	
CMC = Complement Carry	11110101	
STC = Set Carry	11111001	
CLD = Clear Direction	11111100	
STD = Set Direction	11111101	
CLI = Clear Interrupt	11111010	
STI = Set Interrupt	11111011	
HLT = Halt	11110100	
WAIT = Wait	10011011	
ESC = Escape (to External Device)	11011xxx	mod x x x r/m
LOCK = Bus Lock Prefix	11110000	

NOTES:
AL = 8-bit accumulator
AX = 16-bit accumulator
CX = Count register
DS = Data segment
ES = Extra segment
Above/below refers to unsigned value
Greater = more positive:
Less = less positive (more negative) signed values
if d = 1 then "to" reg; if d = 0 then "from" reg
if w = 1 then word instruction; if w = 0 then byte instruction
if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16 bits, disp-high is absent
if mod = 10 then DISP = disp-high; disp-low
if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP
DISP follows 2nd byte of instruction (before data if required)
*except if mod = 00 and r/m = then EA = disp-high: disp-low.
if s:w = 01 then 16 bits of immediate data form the operand
if s:w = 11 then an immediate data byte is sign extended to form the 16-bit operand
if v = 0 then "count" = 1; if v = 1 then "count" in (CL) register
x = don't care
z is used for string primitives for comparison with ZF FLAG
SEGMENT OVERRIDE PREFIX

0 0 1 reg 1 1 0

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)	Segment
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

Instructions which reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file:
FLAGS =
X:X:X:X:(OF):(DF):(IF):(TF):(SF):(ZF):X:(AF):X:(PF):X:(CF)

DATA SHEET REVISION REVIEW

The following list represents key differences between this and the -005 data sheet. Please review this summary carefully.

1. The Intel 8088 implementation technology (HMOS) has been changed to (HMOS-II).

1.16. Τρόποι Σύνταξης των Εντολών

Οι εντολές του 8088 συντάσσονται με αρκετούς διαφορετικούς τρόπους, και δέχονται διαφορετικές παραμέτρους, αλλά δεν συντάσσονται όλες οι εντολές με όλους τους τρόπους :

1. Υπονοούμενος (Implied) : Η σύνταξη της εντολής δεν περιέχει παράμετρο ή συντάσσεται με συγκεκριμένο τρόπο ο οποίος υπονοεί το ποιες είναι οι παράμετροι.
   Παράδειγμα : STC (Set Carry),
   LODSB ([DS:SI]→ AL)

2. Παράμετρος Καταχωρητής (Register Operand) : Η εντολή επενεργεί σε ένα καταχωρητή.
   Παράδειγμα : PUSH AX (AX→Stack),
   DEC BX (BX=BX-1)

3. Παράμετρος Θέση Μνήμης (Memory Operand) : Η εντολή επενεργεί σε δεδομένα που βρίσκονται στην μνήμη και μάλιστα στο τμήμα δεδομένων (data segment).
   Παράδειγμα : POP [0200] (Stack→ DS:0200)

4. Παράμετρος Σχετική Μετατόπιση (Relative Offset Operand) : Η εντολή μεταφέρει την εκτέλεση (εντολές Jxx, ...) σε διεύθυνση του Code Segment που απέχει κάποια bytes, (+/-), από την αρχή της επόμενης εντολής.
   Παράδειγμα : JNE 0020 (IF (ZF=0) IP=0020)

5. Παράμετρος Αριθμός (Numerical Operand) : Η εντολή δέχεται ως παράμετρο ένα σταθερό νούμερο που έχει θέση δεδομένου.
   Παράδειγμα : INT 3 (Display Registers & Return to MONITOR)

6. Έμμεση Προσπέλαση (Indirect Addressing) : Η εντολή δέχεται ως παράμετρο μία θέση μνήμης από την οποία διαβάζει κάποια bytes (2 ή 4) τα οποία σχηματίζουν την τελική διεύθυνση στην οποία θα επενεργήσει η εντολή.
   Παράδειγμα : CALL FAR [0500] (θα πάει στην διεύθυνση 0500 και θα διαβάσει από εκεί 4 bytes, 2 για το segment και 2 για το offset, και στην διεύθυνση που σχηματίζεται ως segment:offset θα γίνει τελικά η κλήση υπορουτίνας)

1.17. Συνδυασμοί Σύνταξης των Εντολών

Οι παραπάνω τρόποι σύνταξης μπορούν να συνδυαστούν ώστε να μας δώσουν περισσότερους σύνθετους τρόπους σύνταξης των εντολών που είναι :

1. Καταχωρητής σε Καταχωρητή (Register to Register) : Η εντολή διαβάζει δεδομένα από ένα καταχωρητή και μετά από επεξεργασία τα αποθηκεύει σε άλλο καταχωρητή.
   Παράδειγμα : XCHG AX,BX (AX=BX, BX=AX)

2. Καταχωρητής και Μνήμη (Register to/from Memory) : Η εντολή διαβάζει δεδομένα από ένα καταχωρητή ή τη μνήμη και μετά από επεξεργασία τα σώζει στη μνήμη ή σε καταχωρητή αντίστοιχα.
   Παράδειγμα : SUB AX,[0100] (AX=AX-[DS:0100])

3. Καταχωρητής και Αριθμός (Register and Numerical Value): Η εντολή παίρνει ένα νούμερο και ένα καταχωρητή και μετά από κάποια επεξεργασία το αποθηκεύει σε ένα καταχωρητή.
   Παράδειγμα : TEST DX, FFFF (DX~FFFF ?)

4. Μνήμη και Αριθμός (Memory and Numerical Value): Η εντολή επενεργεί πάνω σε ένα νούμερο και μία διεύθυνση μνήμης.
   Παράδειγμα : ADC BY[0300], 33 ([DS:0300] += 33 + CF)

1.18. Τρόποι Διευθυνσιοδότησης Μνήμης

Στις εντολές, που συντάσσονται με διεύθυνση μνήμης (π.χ. POP [0200]) η διεύθυνση μνήμης μπορεί να είναι μία σταθερή διεύθυνση ή συνδυασμός διεύθυνσης και καταχωρητών-δεικτών (BX, SI, DI, BP) που προστίθενται στην διεύθυνση μνήμης που δώσαμε, δίνοντας 24 διαφορετικούς τρόπους σύνταξης διευθύνσεων μνήμης :

1. [Addr16] Παράδειγμα : MOV AX, [0200]
2. [BP+Addr8] Παράδειγμα : MOV AX, [BP+27]
3. [BP+Addr16] Παράδειγμα : MOV AX, [BP+8765]
4. [BP+SI] Παράδειγμα : MOV AX, [BP+SI]
5. [BP+SI+Addr8] Παράδειγμα : MOV AX, [BP+SI
6. [BP+SI+Addr16] Παράδειγμα : MOV AX, [BP+SI+900A]
7. [BP+DI] Παράδειγμα : MOV AX, [BP+DI]
8. [BP+DI+Addr8] Παράδειγμα : MOV AX, [BP+DI+77]
9. [BP+DI+Addr16] Παράδειγμα : MOV AX, [BP+DI+11EE]
10. [BX] Παράδειγμα : MOV AX, [BX]
11. [BX+Addr8] Παράδειγμα : MOV AX, [BX+2B]
12. [BX+Addr16] Παράδειγμα : MOV AX, [BX+1234]
13. [BX+SI] Παράδειγμα : MOV AX, [BX+SI]
14. [BX+DI] Παράδειγμα : MOV AX, [BX+DI]
15. [BX+SI+Addr8] Παράδειγμα : MOV AX, [BX+SI+A8]
16. [BX+DI+Addr8] Παράδειγμα : MOV AX, [BX+DI+59]
17. [BX+SI+Addr16] Παράδειγμα : MOV AX, [BX+SI+4C7E]
18. [BX+DI+Addr16] Παράδειγμα : MOV AX, [BX+DI+91DE]
19. [SI] Παράδειγμα : MOV AX, [SI]
20. [SI+Addr8] Παράδειγμα : MOV AX, [SI+23]
21. [SI+Addr16] Παράδειγμα : MOV AX, [SI+FEDC]
22. [DI] Παράδειγμα : MOV AX, [DI]
23. [DI+Addr8] Παράδειγμα : MOV AX, [DI+23]
24. [DI+Addr16] Παράδειγμα : MOV AX, [DI+2552]

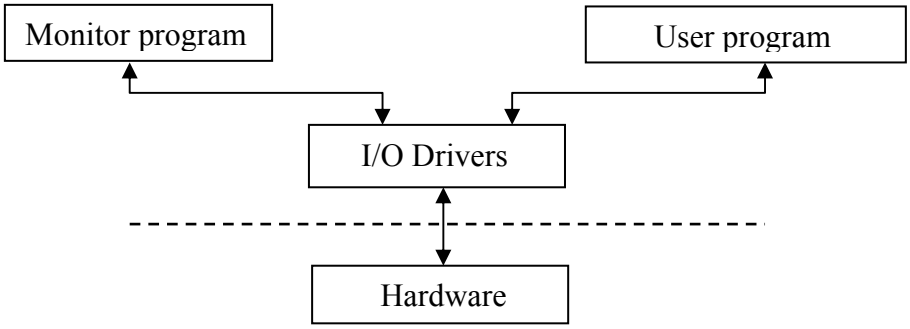
1.19. Οι drivers και ο τρόπος χειρισμού του hardware

Εκτός από το πρόγραμμα MONITOR που επιτρέπει την διαχείριση του μηχανήματος και την συγγραφή και εκτέλεση προγραμμάτων, στην ROM υπάρχουν και οι οδηγοί των συσκευών Εισόδου/Εξόδου του μηχανήματος (I/O drivers). Οι οδηγοί αυτοί είναι στην ουσία υπορουτίνες που τις καλούμε με μία εντολή δακοπής (INT xx) και όχι με την εντολή CALL που είναι η κατ'εξοχήν εντολή για κλήση υπορουτινών. Καλώντας αυτές τις υπορουτίνες

μπορούμε να πραγματοποιήσουμε διαδικασίες Εισόδου/Εξόδου με την αντίστοιχη συσκευή. Η κάθε υπορουτίνα δέχεται και παραμέτρους που καθορίζουν το ποιά ενέργεια Εισόδου/Εξοδου θα πραγματοποιήσουν και πώς. Οι παράμετροι δίνονται ως τιμές σε συγκεκριμένους καταχωρητές πριν από την εκτέλεση της ρουτίνας του οδηγού.
Για παράδειγμα, η εντολή INT 84 εκτελεί την υπορουτίνα του οδηγού για την οθόνη LCD. Η ρουτίνα αυτή το μόνο που μπορεί να κάνει είναι να εμφανίσει στην οθόνη (στην τρέχουσα θέση του κέρσορα) ένα χαρακτήρα και να προχωρήσει τον κέρσορα στην επόμενη θέση. Το ποιος χαρακτήρας θα εμφανιστεί καθορίζεται από την τιμή που θα βάλουμε στον καταχωρητή AL πριν την εκτέλεση της εντολής INT 84. Η τιμή αυτή αντιστοιχεί στον ASCII κωδικό του χαρακτήρα που θέλουμε να εμφανιστεί (π.χ. ‘A’=41, ‘B’=42, …, ‘a’=61,…). Δηλαδή ο κώδκας :

```
MOV AL,41
INT 84
```

εμφανίζει στην οθόνη LCD το γράμμα ‘A’.
Στο Σχήμα 6 φαίνεται ο τρόπος με τον οποίο το MONITOR και τα προγράμματα του χρήστη μπορούν να απευθυνθούν στο hardware.



Σχήμα 6. Η προσπέλαση του hardware από τα προγράμματα (Monitor και χρήστη) μέσω των οδηγών συσκευών Εισόδου/Εξόδου (I/O Drivers)

Πρέπει εδώ να σημειωθεί ότι ο χρήστης μπορεί να συντάξει προγράμματα γλώσσας μηχανής τα οποία θα χρησιμοποιούν το hardware χωρίς την χρήση των οδηγών συσκευών που παρέχονται στην μνήμη ROM. Αυτό όμως απαιτεί τόσο τη γνώση για το πώς ακριβώς μπορεί να προγραμματιστεί η κάθε συσκευή Εισόδου / Εξόδου σε χαμηλό επίπεδο, όσο και αρκετό προγραμματισμό (και debugging) από την πλευρά του χρήστη.
Οι οδηγοί συσκευών που βρίσκονται στην ROM και οι αντίστοιχοι κωδικοί διακοπών είναι οι εξής :

Οδηγός Σειριακής Θύρας RS-232 INT 80

Μέσω του οδηγού αυτού μπορούμε να κάνουμε τα ακόλουθα :

1. Να αρχικοποιήσουμε την Σειριακή Θύρα (initialize).
2. Να μεταδώσουμε δεδομένα (transmit).
3. Να λάβουμε δεδομένα (receive).
4. Να διαβάσουμε την κατάσταση της σειριακής πόρτας (read status).

Οδηγός Πληκτρολογίου INT 81

Μέσω του οδηγού αυτού μπορούμε να κάνουμε τα ακόλουθα :

1. Να διαβάσουμε ένα χαρακτήρα από το πληκτρολόγιο (read caracter).
2. Να διαβάσουμε μία ολόκληρη γραμμή από το πληκτρολόγιο (read command line)
3. Να διαβάσουμε την κατάσταση του πληκτρολογίου (read status).

Οδηγός Εκτυπωτή (Παράλληλης) INT 82

Μέσω του οδηγού αυτού μπορούμε να κάνουμε τα ακόλουθα :

1. Να αρχικοποιήσουμε τον εκτυπωτή (initialize).

2. Να τυπώσουμε δεδομένα (print data).
3. Να διαβάσουμε την κατάσταση του εκτυπωτή (read status).
4. Να προγραμματίσουμε την θύρα ελέγχου του εκτυπωτή (write control port)

Οδηγός Κάρτας Γραφικών Hercules INT 83

Μέσω του οδηγού αυτού μπορούμε να κάνουμε τα ακόλουθα :

1. Να καθορίσουμε το σχήμα του κέρσορα (set cursor type).
2. Να εμφανίσουμε έναν χαρακτήρα στην τρέχουσα θέση του κέρσορα (display character)
3. Να καθορίσουμε τα χαρακτηριστικά (attribute) ενός χαρακτήρα (set attribute)

Οδηγός Οθόνης LCD INT 84

Μέσω του οδηγού αυτού μπορούμε μόνο να εμφανίσουμε έναν οποιοδήποτε χαρακτήρα στην θέση του κέρσορα.

Επιστροφή στο MONITOR INT 85

Με την διακοπή αυτή επανεκκινείται το πρόγραμμα MONITOR. Μπορούμε να την θέτουμε ως τελευταία εντολή ενός προγράμματος. Μηδενίζει τις τιμές των καταχωρητών, γι' αυτό δεν ενδείκνυται για DEBUGGING ενός προγράμματος.

Διακοπή Debugging INT 3

Η διακοπή αυτή χρησιμοποιείται έξυπνα από το πρόγραμμα MONITOR, κατά την εκτέλεση ενός προγράμματος, με καθορισμό διευθύνσεων παύσης (breakpoints). Μπορεί να χρησιμοποιηθεί ως η τελευταία εντολή ενός προγράμματος, καθώς μόλις εκτελείται, εμφανίζει τις τιμές των καταχωρητών και επιστρέφει στο MONITOR.

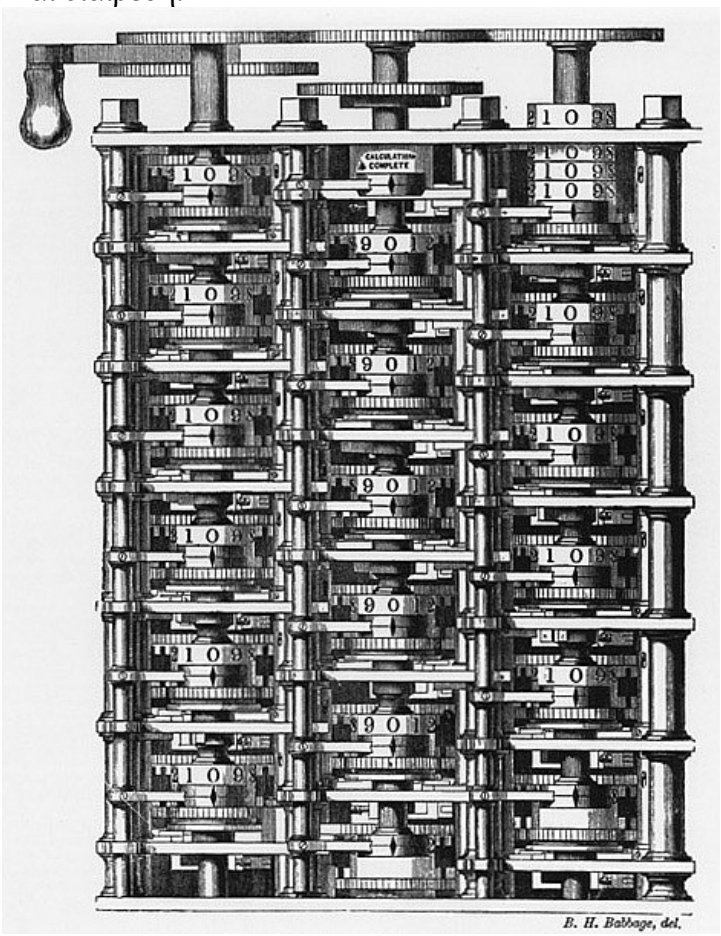
Κεφάλαιο 2: Ιστορία και Εξέλιξη των Υπολογιστών

2.1. Μηχανικοί Υπολογιστές (1642-1945)

Επηρεαζόμενες από την επικρατούσα μηχανιστική θεωρία της εποχής, η οποία θεωρούσε τα πάντα ως απλά ή σύνθετα μηχανικά συστήματα, οι πρώτες υπολογιστικές μηχανές, που κατασκευάστηκαν τον 17ο αιώνα, ήταν καθαρά μηχανικά συστήματα. Οι μηχανές αυτές ήταν ένα σημαντικό βήμα προόδου καθώς μπορούσαν να πραγματοποιούν, σχετικά γρήγορα, χρονοβόρους υπολογισμούς. Σαν μηχανικά συστήματα, όμως, είχαν ανοχές και ατέλειες, ενώ η τεχνολογία της εποχής δεν μπορούσε να κατασκευάσει τα μέρη τους με την απαιτούμενη υψηλή ακρίβεια.

Ο πρώτος που κατασκεύασε μια υπολογιστική μηχανή που λειτουργούσε ήταν ο Γάλλος επιστήμονας **Blaise Pascal** (1623-1662), προς τιμήν του οποίου ονομάστηκε η γλώσσα προγραμματισμού Pascal. Η μηχανή του Pascal χρησιμοποιούσε γρανάζια και τροφοδοτούνταν χειροκίνητα με μανιβέλα, ενώ μπορούσε να εκτελέσει μόνο αφαίρεση και πρόσθεση.

Μερικά χρόνια αργότερα, ο Γερμανός μαθηματικός **Gottfried Wilhelm von Leibniz** (1646-1716) κατασκεύασε μια βελτιωμένη μηχανή που μπορούσε να εκτελεί πρόσθεση, αφαίρεση, πολ/σμό και διαίρεση.



Η Μηχανή Διαφορών του Charles Babbage

Επόμενος σταθμός στους μηχανικούς υπολογιστές ήταν η **Μηχανή Διαφορών** (Difference Engine) του μαθηματικού **Charles Babbage** (1792-1871). Η μηχανή εκτελούσε

μόνο πρόσθεση και αφαίρεση, χρησιμοποιούνταν για τον υπολογισμό πινάκων στην ναυσιπλοΐα και εκτελούσε μόνον έναν αλγόριθμο, τη μέθοδο των πεπερασμένων διαφορών με χρήση πολυωνύμων. Σημαντική ήταν και η ικανότητα της μηχανής να εξάγει τα αποτελέσματα, αποτυπώνοντας τα σε μια πλάκα χαλκογραφίας.

Στη συνέχεια ο Babbage εξέλιξε μια τελειότερη συσκευή, την **Αναλυτική Μηχανή** (Analytical Engine). Η συσκευή αυτή εκτελούσε τις 4 βασικές πράξεις, δέχονταν είσοδο σε μορφή διάτρητων καρτών και έδινε έξοδο σε διάτρητες κάρτες και σε εκτύπωση. Η μεγάλη καινοτομία που εισήγαγε η μηχανή αυτή ήταν ότι μπορούσε να χρησιμοποιηθεί για διάφορες χρήσεις, τροφοδοτώντας την με διαφορετικές κάρτες στην είσοδο.

Ο Babbage κατανόησε ότι, εφ' όσον η Αναλυτική Μηχανή μπορούσε να προγραμματιστεί σε μια απλή συμβολική γλώσσα, χρειαζόταν να δημιουργηθεί λογισμικό. Για το σκοπό αυτό προσέλαβε σαν προγραμματίστρια την **Ada Augusta Lovelace**, κόρη του Λόρδου Byron. Η Ada Lovelace θεωρείται η πρώτη προγραμματίστρια στον κόσμο και προς τιμήν της ονομάστηκε η σημερινή γλώσσα προγραμματισμού **Ada**.

Παρά τις προχωρημένες ιδέες του, ο Babbage ποτέ δεν κατάφερε να αποσφαλματώσει τελείως το υλικό, προδομένος από την τεχνολογία της εποχής του, η οποία αδυνατούσε να κατασκευάσει οδοντώσεις, τροχούς και γρανάζια με τον απαιτούμενο βαθμό ακρίβειας.

Στα τέλη της δεκαετίας του '30, ο Γερμανός μηχανικός **Konrad Zuse** κατασκεύασε μια σειρά από αυτόματες υπολογιστικές μηχανές χρησιμοποιώντας ηλεκτρονόμους (ρελέ), οι οποίες όμως καταστράφηκαν στον πόλεμο. Λίγο αργότερα, στις ΗΠΑ, οι **John Atanasoff** και **George Stibbitz** σχεδίασαν αριθμομηχανές. Η συσκευή του Atanasoff χρησιμοποιούσε δυαδική αριθμητική και μνήμη από πυκνωτές αλλά δεν μπόρεσε να λειτουργήσει, προδομένη από την ανεπαρκή τεχνολογία της εποχής. Η μηχανή του Stibbitz, αν και πιο απλή, λειτούργησε και έγιναν αρκετές δημόσιες επιδείξεις της.

Ένας άλλος μηχανικός που προσπάθησε να κατασκευάσει υπολογιστή γενικής χρήσης με ηλεκτρονόμους ήταν ο **Howard Aiken**. Το πρώτο του μοντέλο ήταν ο **Mark I**, ο οποίος χρησιμοποιούσε διάτρητη ταινία για είσοδο και έξοδο και είχε χρόνο εντολής 6 sec. Ο Aiken κατασκεύασε και ένα πιο βελτιωμένο μοντέλο, τον **Mark II** αλλά μέχρι τότε οι μηχανικοί υπολογιστές ήταν ήδη ξεπερασμένοι και όλοι είχαν στραφεί στους ηλεκτρονικούς.

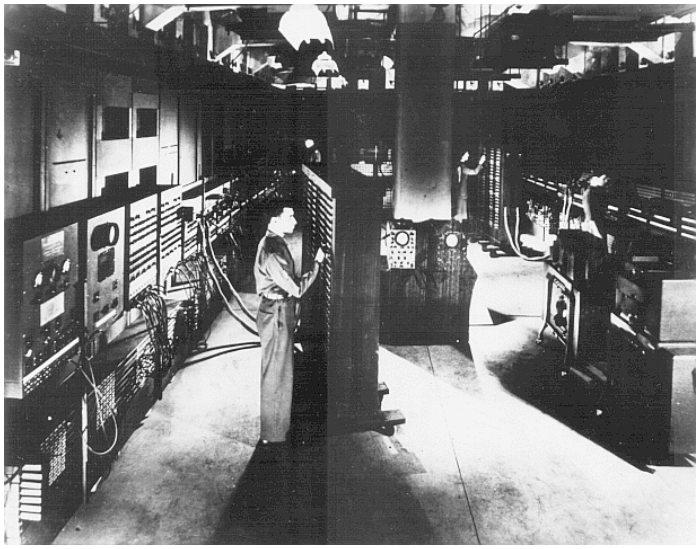
2.2. Πρώτη Γενιά Υπολογιστών - Λυχνίες Κενού (1945-1955)

Η ανακάλυψη της ηλεκτρονικής λυχνίας κενού από τον **Lee de Forest** στις αρχές του 20ου αιώνα έδωσε μια τεράστια ώθηση στην εξέλιξη των υπολογιστών, καθώς από μηχανικά ή ηλεκτρομηχανικά συστήματα οι υπολογιστές έγιναν καθαρά ηλεκτρονικά συστήματα, επιτυγχάνοντας υψηλές αποδόσεις και μεγαλύτερη αξιοπιστία.

Το βασικό ίσως ερέθισμα για την δημιουργία του ηλεκτρονικού υπολογιστή ήταν ο Β' Παγκόσμιος Πόλεμος και οι αυξημένες υπολογιστικές ανάγκες που προέκυψαν. Το 1943 τέθηκε σε λειτουργία ο πρώτος ηλεκτρονικός υπολογιστής στον κόσμο ο **COLOSSUS**. Σχεδιάστηκε από τον μαθηματικό **Alan Turing**, ανήκε στην Βρετανική Κυβέρνηση και χρησιμοποιήθηκε με επιτυχία για την αποκρυπτογράφηση μηνυμάτων των Γερμανών που είχαν κρυπτογραφηθεί με την συσκευή ENIGMA. Ελάχιστα στοιχεία έγιναν γνωστά για τον υπολογιστή αυτό λόγω του στρατιωτικού απορρήτου που ίσχυε γι' αυτόν.

Στις Η.Π.Α. υπήρχε η ανάγκη δημιουργίας πινάκων βολών για το πυροβολικό, διαδικασία χρονοβόρα που απαιτούσε πολλά άτομα ενώ τα λάθη ήταν συχνά. Μετά από έγκριση του Στρατού ο **John Mauchley**, με τον **J. Presper Eckert**, ξεκίνησαν το 1943 την κατασκευή ενός ηλεκτρονικού υπολογιστή για τον υπολογισμό των πινάκων αυτών. Το αποτέλεσμα ήταν ο **ENIAC** (Electronic Numerical Integrator and Computer), μια μηχανή βάρους 30 τόνων και ισχύος 140 KW, με 18.000 λυχνίες και 1.500 ηλεκτρονόμους. Ο προγραμματισμός

γινόταν με τη ρύθμιση 6.000 διακοπτών και ενός πλήθους καλωδίων. Ο ENIAC ολοκληρώθηκε το 1946, πολύ αργά για να επιτελέσει το σκοπό για τον οποίο κατασκευάστηκε. Ωστόσο, επιδείχθηκε σε πολλούς ερευνητές και μηχανικούς και έδωσε το έναυσμα για την κατασκευή και άλλων παρόμοιων υπολογιστών.



Ο Υπολογιστής ENIAC (1943)

Στα βήματα του ENIAC, ακολούθησαν οι EDSAC (1949), JOHNIAC, ILLIAC, MANIAC και WEIZAC. Οι δημιουργοί του ENIAC Eckert και Mauchley άρχισαν την κατασκευή του EDVAC (Electronic Discrete Variable Automatic Computer), αλλά η κατασκευή διακόπηκε και οι δύο κατασκευαστές ίδρυσαν μια εταιρία η οποία εξελίχθηκε στην σημερινή Unisys Corporation.

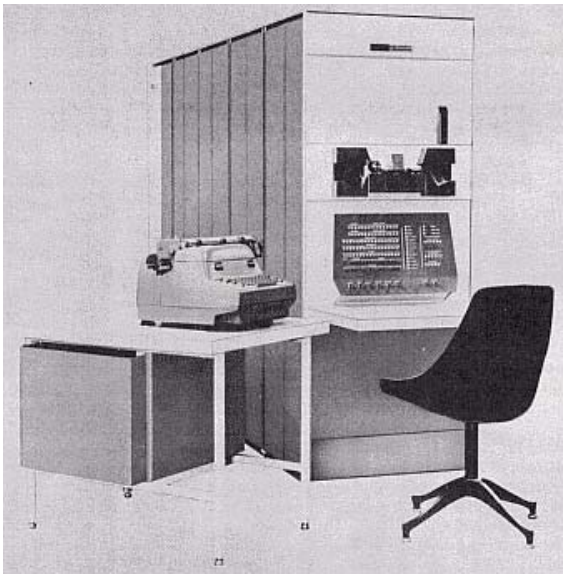
Ενας από τους κατασκευαστές του ENIAC, ο **John von Neumann**, ένας ιδιοφυής μηχανικός, ξεκίνησε την κατασκευή της δικής του εκδοχής του EDVAC, της μηχανής **IAS**. Ο von Neumann διαπίστωσε ότι ο προγραμματισμός των υπολογιστών με έναν τεράστιο αριθμό διακοπτών και καλωδίων ήταν αργός, κοπιαστικός και άκαμπτος και κατέληξε στο συμπέρασμα ότι τα προγράμματα μπορούσαν να παρασταθούν σε δυαδική μορφή στη μνήμη του υπολογιστή, μαζί με τα δεδομένα. Ο βασικός σχεδιασμός που ο ίδιος παρουσίασε είναι γνωστός σήμερα ως **Μηχανή von Neumann**, πρωτοχρησιμοποιήθηκε στον EDSAC και στον IAS και εξακολουθεί να είναι η βάση όλων σχεδόν των ψηφιακών υπολογιστών, μέχρι και σήμερα.

Ο John Von Neumann

Την ίδια εποχή που κατασκευάζονταν οι IAS, ENIAC και οι άλλες μηχανές αυτού του τύπου, που προορίζονταν για αριθμητικούς υπολογισμούς, στο M.I.T. κατασκευάστηκε ο **Whirlwind I**, ο οποίος ήταν σχεδιασμένος για έλεγχο πραγματικού χρόνου, και οδήγησε στην εφεύρεση της μνήμης μαγνητικών πυρήνων και τελικά στον πρώτο εμπορικό mini υπολογιστή.
Το 1952, ο **UNIVAC I**, ο πρώτος εμπορικός υπολογιστής της Remington Rand που παρουσιάστηκε το 1951 και σχεδιάστηκε από τους Mauchley και Eckert, χρησιμοποιείται από το τηλεοπτικό δίκτυο CBS για να προβλέψει τον νικητή των εκλογών της χρονιάς εκείνης στις Η.Π.Α..
Η **IBM** κατασκεύασε τον πρώτο υπολογιστή της με λυχνίες, τον **IBM 701**, το 1953. Το 1956 κατασκεύασε τον IBM 704 και το 1958 τον IBM 709, που ήταν ο τελευταίος υπολογιστής της με λυχνίες.

2.3. Δεύτερη Γενιά - Τρανζίστορ (1955-1965)

Το τρανζίστορ εφευρέθηκε στα εργαστήρια Bell Labs το 1948 από τους **J. Bardeen, W. Brattain και W. Shockley**, στους οποίους απονεμήθηκε το βραβείο Nobel Φυσικής το 1956 για την εφεύρεσή τους αυτή. Το τρανζίστορ έφερε επανάσταση στους υπολογιστές και μέχρι το τέλος της δεκαετίας του '50 οι υπολογιστές με λυχνίες είχαν καταργηθεί.
Ο πρώτος υπολογιστής με τρανζίστορ κατασκευάστηκε στο M.I.T. και ακολουθούσε τη φιλοσοφία του Whirlwind. Ονομάστηκε **TX-0** και προορίζονταν για τη δοκιμή του **TX-2**, ο οποίος όμως δεν απέδωσε τα αναμενόμενα.
Ενας από τους μηχανικούς που δούλεψαν στους TX, ο **Kenneth Olsen**, δημιούργησε το 1957 την **Digital Equipment Corporation (DEC)** με σκοπό την κατασκευή ενός εμπορικού υπολογιστή. Σε 4 χρόνια ο **PDP-1** ήταν γεγονός. Είχε 4K λέξεις των 18 bit και χρόνο κύκλου 5 μsec. Είχε τη μισή απόδοση του **IBM 7090**, διαδόχου με τρανζίστορ του 709 και ταχύτερου υπολογιστή στον κόσμο την εποχή εκείνη, αλλά κόστιζε ένα κλάσμα της αξίας του IBM. Πουλήθηκαν μερικές δεκάδες PDP-1 και έτσι γεννήθηκε η βιομηχανία των mini υπολογιστών. Μερικά χρόνια αργότερα παρουσιάστηκε ο **PDP-8** ο οποίος ήταν υπολογιστής των 12 bit αλλά πολύ φθηνότερος του PDP-1. Σημαντική καινοτομία ήταν η χρήση ενός μόνο διαύλου, του **Omnibus**, για την σύνδεση όλων των κυκλωμάτων. Πουλήθηκαν μερικές χιλιάδες PDP-8 και η DEC κατέκτησε την πρωτοκαθεδρία στο χώρο των mini υπολογιστών.



Ο mini Ηλεκτρονικός Υπολογιστής DEC PDP-1

Η IBM κατασκεύασε σαν απάντηση τους **7090** και **7094**. Οι δύο αυτοί υπολογιστές ήταν οι τελευταίοι τύπου ENIAC που κατασκευάστηκαν και κυριάρχησαν στο χώρο των επιστημονικών υπολογιστών την δεκαετία του '60. Παράλληλα, η IBM κατασκεύασε μια μικρή μηχανή, τον **1401**, ο οποίος μπορούσε να γράφει και να διαβάζει σε μαγνητικές ταινίες, να διαβάζει και να τρυπά κάρτες και να δίνει εκτυπώσεις σαν έξοδο. Ηταν απαράδεκτος για επιστημονικούς σκοπούς αλλά πολύ καλός για τήρηση αρχείων μιας επιχείρησης και γνώρισε επιτυχία στο χώρο αυτό.
Το 1964 μια νέα εταιρία, η **Control Data Corporation (CDC)** παρουσίασε τον **CDC-6600**, μια μηχανή μία τάξη μεγέθους ταχύτερη από τον IBM 7094. Χρησιμοποιούσε παραλληλία και άλλες προχωρημένες τεχνικές για την εποχή του για να επιτύχει αυτή την υψηλή απόδοση. Σχεδιαστής του 6600 ήταν ο ιδιοφυής **Seymour Cray**, ο οποίος αφιέρωσε τη ζωή του στην κατασκευή όλο και πιο γρήγορων υπολογιστών που σήμερα λέγονται υπερυπολογιστές (supercomputers) όπως ο 6600, ο 7600 και ο **Cray-1**.

Ο Ηλεκτρονικός Υπολογιστής CDC-6000

Υπήρξαν και άλλοι πολλοί υπολογιστές την εποχή εκείνη αλλά ένας ξεχωρίζει για εντελώς διαφορετικό λόγο και αξίζει να αναφερθεί, ο **Burroughs B5000**. Οι σχεδιαστές υπολογιστών ήταν ολοκληρωτικά απασχολημένοι με το υλικό ώστε να παράγουν υπολογιστές ταχύτερους ή φθηνότερους, αδιαφορώντας για το λογισμικό. Οι σχεδιαστές του B5000 ακολούθησαν διαφορετική τακτική. Κατασκεύασαν μια μηχανή ειδικά για προγραμματισμό σε Algol 60,

έναν πρόδρομο της Pascal, και περιέλαβαν πολλές δυνατότητες στο υλικό για να διευκολύνουν τη δουλειά του μεταγλωττιστή. Έτσι, γεννήθηκε η ιδέα ότι και το λογισμικό παίζει ρόλο.

2.4. Τρίτη Γενιά - Ολοκληρωμένα Κυκλώματα (1965-1980)

Η εφεύρεση του ολοκληρωμένου κυκλώματος πυριτίου από τον Robert Noyce το 1958 επέτρεψε να τοποθετηθούν δεκάδες τρανζίστορ σε ένα μόνο chip. Ετσι έγινε δυνατή η κατασκευή υπολογιστών μικρότερων, γρηγορότερων και φθηνότερων από τους υπολογιστές με τρανζίστορ της προηγούμενης γενιάς. Μερικοί από τους σημαντικότερους υπολογιστές αυτής της γενιάς περιγράφονται πιο κάτω.
Το 1964 η IBM ήταν ήδη κυρίαρχη στην αγορά των υπολογιστών, όμως είχε ένα μεγάλο πρόβλημα με τις δύο πιο επιτυχημένες μηχανές της, τον 7094 και τον 1401, ήταν τελείως ασύμβατοι μεταξύ τους. Πολλές εταιρίες είχαν και τους 2 υπολογιστές και δυσανασχετούσαν καθώς έπρεπε να διατηρούν δύο ξεχωριστά τμήματα προγραμματισμού. Ετσι, ο αντικαταστάτης των δύο αυτών υπολογιστών, ο **System/360**, έκανε ένα ριζοσπαστικό βήμα. Αποτελούσε μια οικογένεια από 5-6 υπολογιστές, με διαφορετικές αποδόσεις και τιμή, οι οποίοι όμως μπορούσαν να τρέχουν τα ίδια προγράμματα. Το Model 30 προορίζονταν να αντικαταστήσει τους 1401, το Model 75 τους 7094 ενώ μεταξύ τους υπήρχαν τα μοντέλα Model 40, Model 50 και Model 65, με κλιμακούμενη απόδοση και τιμή.

Ο Ηλεκτρονικός Υπολογιστής IBM System/360

Η ιδέα της οικογένειας υπολογιστών ήταν επιτυχημένη και σύντομα ακολουθήθηκε και από τους άλλους κατασκευαστές. Άλλη μια καινοτομία του System/360 ήταν ο πολυπρογραμματισμός (multiprogramming), δηλαδή η ταυτόχρονη φόρτωση και εκτέλεση πολλών προγραμμάτων. Επίσης, οι υπολογιστές της σειράς είχαν την ικανότητα να εξομοιώνουν τους 1401 και 7094, ώστε να μπορούν να εκτελούνται και παλαιότερα προγράμματα για τους υπολογιστές αυτούς. Τέλος ο χώρος διευθύνσεων ήταν τεράστιος για την εποχή του, φθάνοντας τα 16 MB. Την σειρά 360 διαδέχθηκαν οι σειρές 370, 4300, 3080 και 3090.
Στο χώρο των mini υπολογιστών, η DEC κυκλοφόρησε τον PDP-11, ακολουθώντας τη φιλοσοφία του 360. Ο PDP-11 ήταν εξαιρετικά επιτυχημένος, ιδίως στα Πανεπιστήμια, και διατήρησε την πρωτοκαθεδρία της DEC στους mini υπολογιστές.

2.5. Τέταρη Γενιά-Ολοκλήρωση μεγάλης κλίμακας (1980-σήμερα)

- Τεχνολογία LSI και VLSI, χιλιάδες και εκατομμύρια τρανζίστορ σε ένα chip. Pentium 4, 55 εκατομμύρια τρανζίστορ. Αυγή των προσωπικών υπολογιστών (PC – Personal Computer).
- Εμφάνιση του Intel 8080. Πρώτοι Η/Υ σε κίτ, χωρίς λειτουργικό. Λειτουργικό CP/M (Gary Kildall).
- Αλλες CPU, Z80 Zilog (1976), 6502 MOS Tech Corp. (1976).
- Steve Jobs και Steve Wozniak, Apple Computers, Apple και Apple II με επεξεργαστή 6502, BBC, Electron.
- IBM PC (1981) με 8088 και 16 Kb RAM και δισκέτα. Ανοικτή αρχιτεκτονική, επέκταση με κάρτες. Αντιγραφή σε PC-compatibles. Microsoft MS-DOS.
- Home Computers, Sinclair Spectrum (Z80), Commodore 64 (6502), Amstrad CPC-464 (Z80), Atari (6502) με ενσωματωμένη BASIC.
- Ανάπτυξη επεξεργαστών 8086, 80286, 80386, 80486, Pentium, Pentium Pro, Pentium II, Pentium III, Pentium 4.
- Ανάπτυξη επεξεργαστών RISC για Σταθμούς Εργασίας, HP, SUN.

Στη δεκαετία του '80, η τεχνολογία **VLSI** (Very Large Scale Integration) επέτρεψε να τοποθετηθούν αρχικά δεκάδες και εκατοντάδες χιλιάδες και αργότερα εκατομμύρια τρανζίστορ σε ένα μόνο chip. Η εξέλιξη αυτή οδήγησε σε μικρότερους και σε φθηνότερους υπολογιστές και τελικά στους **προσωπικούς υπολογιστές (PC, Personal Computer)**. Οι προσωπικοί υπολογιστές χρησιμοποιούνται με πολύ διαφορετικό τρόπο από τους μεγάλους υπολογιστές. Χρησιμεύουν για επεξεργασία κειμένου, για λογιστικά φύλλα και για πολλές έντονα αλληλεπιδραστικές εφαρμογές, τις οποίες οι μεγαλύτεροι υπολογιστές δεν μπορούν να χειριστούν με ευχέρεια.
Οι πρώτοι προσωπικοί υπολογιστές είχαν συνήθως μορφή kit. Το kit περιελάμβανε ένα τυπωμένο κύκλωμα, έναν μικροεπεξεργαστή **Intel 8080**, συνήθως, και μερικά ακόμη chip, μερικά καλώδια, ένα τροφοδοτικό και ίσως μια μονάδα δισκέτας. Η συναρμολόγηση του υπολογιστή γίνονταν από τον αγοραστή και επίσης το λογισμικό γράφονταν από αυτόν. Αργότερα, το λειτουργικό σύστημα **CP/M**, που γράφτηκε από τον Gary Kildall, έγινε δημοφιλές στους υπολογιστές με 8080.
Σταδιακά οι προσωπικοί υπολογιστές άρχισαν να πωλούνται συναρμολογημένοι. Ενας από τους πρώτους ήταν ο Apple, και αργότερα ο **Apple II**, των Steve Jobs και Steve Wozniak ο οποίος είχε τον μικροεπεξεργαστή 6502 και γνώρισε μεγάλη επιτυχία.

Ο Προσωπικός Ηλεκτρονικός Υπολογιστής APPLE II (1977)

Η IBM μπήκε στο χώρο των PC σχετικά αργά, στις αρχές του '80. Ακολουθώντας μια τακτική η οποία επηρέασε βαθύτατα την εξέλιξη των PC στις επόμενες δεκαετίες, η IBM κατασκεύασε τον **IBM PC** το 1981 χρησιμοποιώντας υλικά του εμπορίου και δημοσιεύοντας ελεύθερα τα σχέδια του υπολογιστή. Η κίνηση αυτή έγινε από την IBM με σκοπό να μπορέσουν τρίτοι κατασκευαστές να παράγουν κάρτες επέκτασης για τον IBM PC αλλά τελικά πολλές εταιρίες αντέγραψαν ολόκληρο τον υπολογιστή, δημιουργώντας κλώνους (clones) του IBM PC (ή IBM-συμβατά, IBM-compatibles) και ξεκινώντας μια ολόκληρη βιομηχανία προσωπικών υπολογιστών η οποία κυριάρχησε τελικά στο χώρο αυτό.

Ο προσωπικός ηλεκτρονικός υπολογιστής IBM-PC (1981)

Τη δεκαετία του '80 παρουσιάστηκαν και άλλα PC με CPU που δεν ήταν της Intel. Ετσι, για κάποιο διάστημα κυριάρχησαν στην αγορά οι υπολογιστές της Sinclair, της Amstrad, της Atari, της Commodore κ.α. Οι υπολογιστές αυτοί βασίζονταν σε μικροεπεξεργασές όπως ο Z80 της Zilog (1976), ο 6502 της MOS Tech Corp. (1976) και ο 68000 της Motorolla, και ονομάστηκαν Home Computers. Στην κατηγορία αυτή αξίζει να αναφέρουμε τον Sinclair Spectrum (Z80), τον Commodore 64 (6502), τον Amstrad CPC-464 (Z80) και CPC-6128 (Z80) με τη πρώτη μονάδα δισκέτας (για Home Computer) με μέγεθος 200 KB και εγγραφή και στις δύο όψεις της δισκέτας με drive μονής κεφαλής που απαιτούσε αλλαγή πλευράς !. Επίσης μπορούμε να αναφέρουμε τον Atari (6502) με ενσωματωμένη BASIC, τον BBC και Electron (6502) τον Oric Atmos αλλά και την Amiga που ήταν υπολογιστής ορόσημο για τα γραφικά του.

Φωτογραφιες

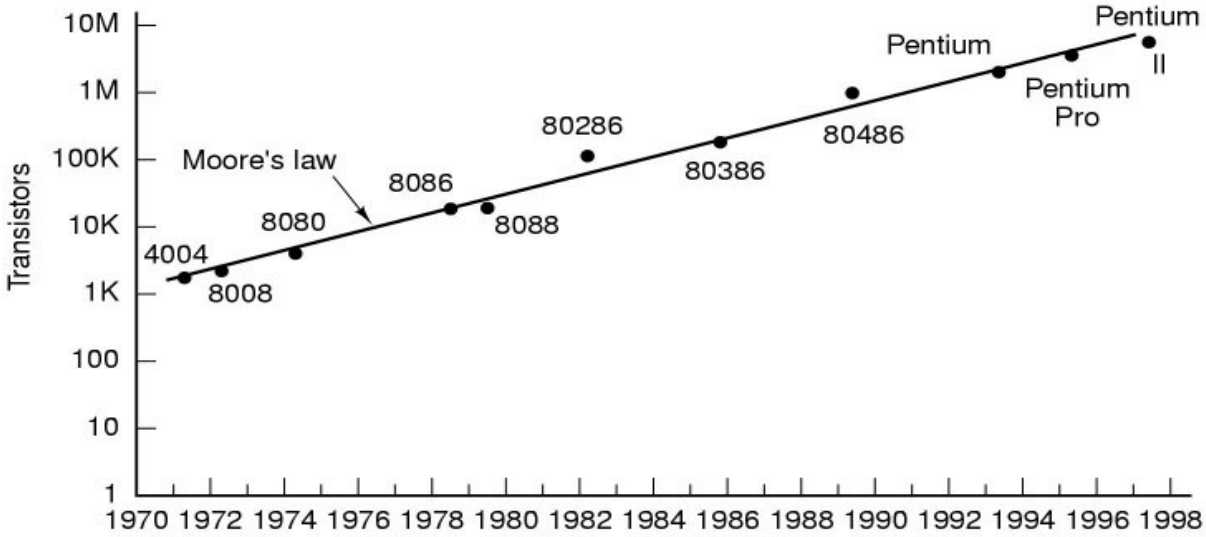
Όμως μέχρι το τέλος της δεκαετίας του ΄80 η επικράτηση των IBM-συμβατών PC έναντι των Home Computers ήταν ολοκληρωτική καθώς οι πρώτοι είχαν επαγγελματικές προδιαγραφές και δυνατότητες και ανοιχτή αρχιτεκτονική, ενώ οι τιμές τους έπεφταν διαρκώς.

Η πρώτη έκδοση του IBM PC ήταν εξοπλισμένη με το λειτουργικό σύστημα MS-DOS που προερχόταν από την μικρή, τότε, Microsoft Corporation. Η Microsoft μαζί με την IBM κατασκεύασαν ένα λειτουργικό σύστημα που εκμεταλλεύονταν τις δυνατότητες των νέων CPU της Intel, το OS/2. Το OS/2 απέτυχε εμπορικά και κυριάρχησε ένα άλλο λειτουργικό που ετοίμαζε η Microsoft παράλληλα με το OS/2, τα Windows. Το πώς τελικά δυο μικρές τότε εταιρίες, η Intel και η Microsoft, κατόρθωσαν να εκθρονίσουν την IBM, μια εταιρία με τεράστια ισχύ και πόρους, αποτελεί Case-study που διδάσκεται στις οικονομικές σχολές σε όλο τον κόσμο.

Στα μέσα της δεκαετίας του '80 κάνει την εμφάνισή της μια νέα αρχιτεκτονική, η RISC, η οποία υλοποιούσε CPU με μικρό και απλό σύνολο εντολών, με σκοπό την αύξηση της απόδοσης. Άλλη μια εξέλιξη προς την κατεύθυνση αυτή ήταν και η ανάπτυξη των Υπερβαθμωτών (superscalar) CPU τη δεκαετία του '90.

2.6. Εξέλιξη και Κατηγορίες Υπολογιστών

Η βιομηχανία των υπολογιστών εξελίσσεται με ταχείς ρυθμούς. Η βασική κινητήρια δύναμη είναι η ικανότητα των κατασκευαστών chip να ενσωματώνουν σε ένα chip όλο και περισσότερα τρανζίστορ, γεγονός που οδηγεί σε μεγαλύτερες μνήμες και ισχυρότερους επεξεργαστές.



Η επιβεβαίωση του νόμου του Moore από το 1970 έως το 1998

Ο ρυθμός της τεχνολογικής προόδου μπορεί να συνοψιστεί σε μια εμπειρική παρατήρηση που λέγεται Νόμος του Moore. Ο νόμος αυτός πήρε το όνομα του από τον Gordon Moore, συνιδρυτή της Intel και προβλέπει ότι ο αριθμός των τρανζίστορ που μπορούν να ενσωματωθούν σε ένα chip (και συνεπώς και η υπολογιστική ισχύς) διπλασιάζεται κάθε 18 μήνες, γεγονός που σημαίνει ετήσια αύξηση περίπου 60%. Στην Εικ. 1.9 φαίνεται η επαλήθευση του Νόμου του Moore για chip CPU. Ο Νόμος αυτός αναμένεται να ισχύει για αρκετά χρόνια ακόμη και το τέλος της συνεχούς αυτής εξέλιξης φαίνεται να είναι οι Κβαντικοί υπολογιστές, όπου θα μπορεί να αποθηκεύεται 1 bit πληροφορίας στο spin κάθε ηλεκτρονίου.

Εκτός από τους κβαντικούς υπολογιστές, άλλοι εναλλακτικοί υπολογιστές πάνω στους οποίους γίνεται έρευνα είναι οι Φωτονικοί Υπολογιστές (Photonics), οι οποίοι χρησιμοποιούν οπτικούς κρυστάλλους και φως αντί για τρανζίστορ και ρεύμα, και οι Βιολογικοί υπολογιστές, οι οποίοι θα αποτελούνται από οργανικά μέρη. Οι τεχνολογίες αυτές αναμένεται να ξεπεράσουν τους περιορισμούς που ισχύουν για τη σημερινή μορφή των υπολογιστών αλλά απέχουν πολλά χρόνια ακόμη από την μαζική τους παραγωγή.

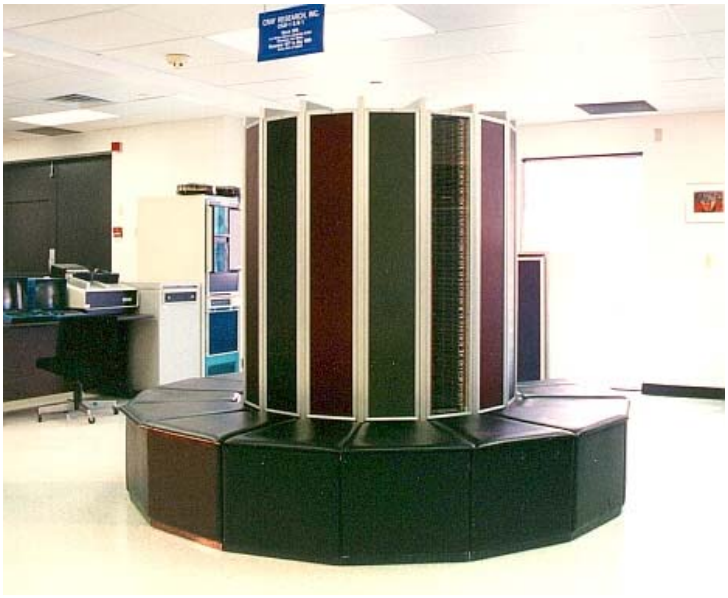
Τελειώνοντας, θα αναφέρουμε το σημερινό φάσμα των διαθέσιμων υπολογιστών

2.7. Κατηγορίες Υπολογιστών και κλίμακα κόστους

Τύπος	Τιμή σε $	Παράδειγμα εφαρμογής
Αναλώσιμος Υπολογιστής	1	Ευχετήριες μουσικές κάρτες
Ενσωματωμένος Υπολογιστής	10	Ρολόγια, αυτοκίνητα, οικιακές συσκευές
Υπολογιστής παιχνιδιών	100	Οικιακά video games
Προσωπικός Υπολογιστής	1.000	Επιτραπέζιοι ή φορητοί υπολογιστές
Διακομιστής (Server)	10.000	Σταθμός εργασίας ή διακομιστής δικτύου
Συστοιχία σταθμών εργασίας	100.000	Mini υπολογιστής τμήματος
Κεντρικό σύστημα (Mainframe)	1.000.000	Κεντρικός υπολογιστής τράπεζας
Υπερυπολογιστής	10.000.000	Επιστημονικές εφαρμογές

Στο κατώτερο σημείο του φάσματος υπάρχουν μεμονωμένα chip τα οποία βρίσκονται στο εσωτερικό ευχετήριων καρτών, έξυπνων καρτών κ.α. Η αξία τους είναι ελάχιστη και γι' αυτό είναι αναλώσιμοι.
Στο επόμενο επίπεδο έχουμε τους υπολογιστές που είναι ενσωματωμένοι σε διάφορες καταναλωτικές ηλεκτρονικές συσκευές (Embedded computers) όπως τηλέφωνα, τηλεοράσεις, παιγνίδια, συσκευές CD κ.α. Οι υπολογιστές αυτοί περιέχουν έναν μικροεπεξεργαστή, λιγότερο από 1 MB μνήμης και ελάχιστες δυνατότητες εισόδου/εξόδου.
Ένα σκαλοπάτι πιο πάνω βρίσκονται οι μηχανές για video games. Πρόκειται για κανονικούς υπολογιστές με ειδικές δυνατότητες γραφικών αλλά με περιορισμένο λογισμικό και μηδενική σχεδόν επεκτασιμότητα. Στην ίδια κατηγορία βρίσκονται οι ηλεκτρονικές ατζέντες, οι φορητοί ψηφιακοί βοηθοί (personal digital assistants, PDA) και τα τερματικά για το Web. Περιέχουν έναν μικροεπεξεργαστή, μερικά MB μνήμης και κάποιο είδος οθόνης.
Οι προσωπικοί υπολογιστές, στο επόμενο επίπεδο, έχουν ουσιαστικά ταυτιστεί με τη λέξη 'υπολογιστής'. Περιέχουν μικροεπεξεργαστή, πολλά MB μνήμης, σκληρό δίσκο πολλών GB, διάφορες περιφερειακές συσκευές, προηγμένο λειτουργικό σύστημα, πολλές δυνατότητες επέκτασης και διαθέτουν μεγάλο φάσμα διαθέσιμου λογισμικού. Οι CPU της Intel με τα λειτουργικά συστήματα της Microsoft κυριαρχούν στο χώρο αυτό.
Οι διακομιστές δικτύου, είτε για τοπικά δίκτυα είτε για το Internet, είναι ενισχυμένοι προσωπικοί υπολογιστές, με έναν ή περισσότερες CPU, μνήμη μέχρι GB, συστοιχίες δίσκων πολλών GB και δικτυακές δυνατότητες υψηλής ταχύτητας.
Η Συστοιχία σταθμών εργασίας (Cluster of Workstations, COW) αποτελείται από προσωπικούς υπολογιστές συνδεδεμένους μεταξύ τους σε γρήγορο δίκτυο, οι οποίοι εκτελούν ειδικό λογισμικό που επιτρέπει σε όλες τις μηχανές να συνεργάζονται για να επιλύσουν ένα πρόβλημα, επιστημονικό ή τεχνικό. Κλιμακώνονται εύκολα και μπορεί να περιλαμβάνουν από λίγους σταθμούς εργασίας μέχρι μερικές χιλιάδες. Η συστοιχία έχει την απόδοση ενός mini υπερυπολογιστή.
Τα mainframes, με μέγεθος δωματίου, είναι συνήθως απόγονοι του IBM 360. Οι περισσότεροι δεν είναι πιο γρήγοροι από τους πιο γρήγορους διακομιστές, αλλά έχουν αυξημένες δυνατότητες εισόδου/εξόδου και είναι εξοπλισμένοι με συστοιχίες δίσκων που περιέχουν ακόμη και TB ($=10^{12}$ byte) πληροφοριών. Είναι εξαιρετικά δαπανηροί και σε πολλές περιπτώσεις χρησιμοποιούνται κυρίως εξαιτίας της μεγάλης επένδυσης σε λογισμικό, δεδομένα, διαδικασίες λειτουργίας και προσωπικό που αντιπροσωπεύουν.
Στο ανώτερο επίπεδο βρίσκονται οι υπερυπολογιστές (supercomputers), οι οποίοι έχουν ταχύτατες CPU, πολλά GB κύριας μνήμης και πολύ γρήγορους δίσκους και δίκτυα.

Χρησιμοποιούνται για την επίλυση σύνθετων επιστημονικών προβλημάτων όπως η προσομοίωση συγκρουόμενων γαλαξιών, η σύνθεση νέων φαρμάκων, η πρόγνωση του καιρού κ.α.

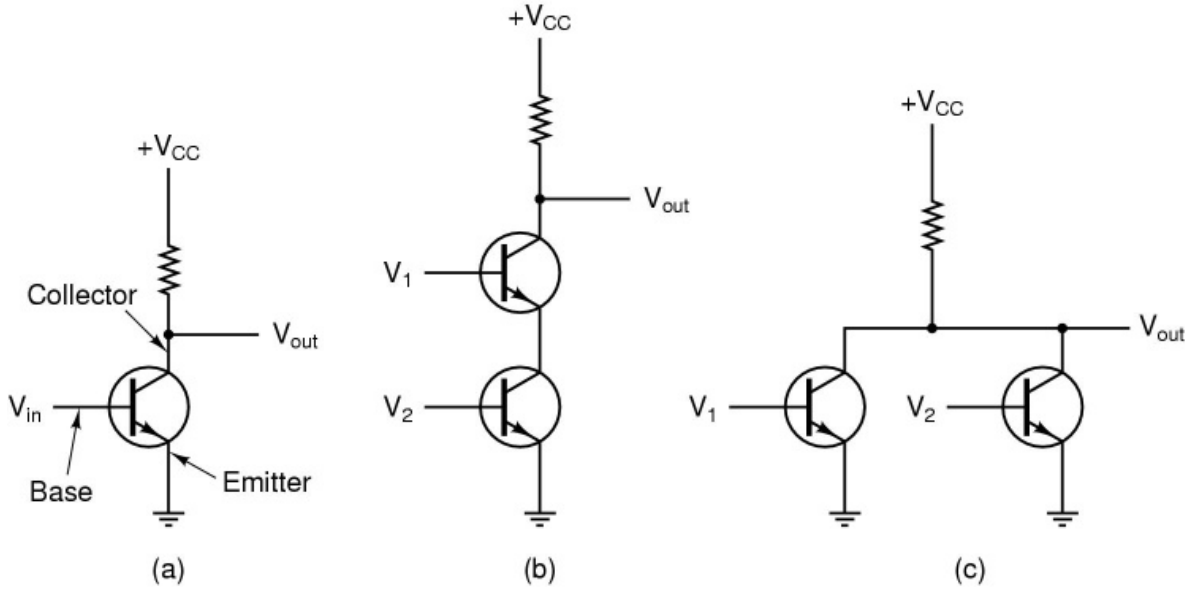


Ο Υπερ-υπολογιστής Cray-1 (1976)

Κεφάλαιο 3. Βασικά και σύνθετα ψηφιακά κυκλώματα

3.1. Πύλες

Τα βασικά στοιχεία από τα οποία αποτελούνται οι υπολογιστές είναι αρκετά απλά. Τα περισσότερα κυκλώματα δημιουργούνται από κατάλληλους συνδυασμούς πυλών (gates), οι οποίες είναι η βάση υλικού πάνω στην οποία οικοδομούνται όλοι οι ψηφιακοί υπολογιστές. Οι συνδυασμοί των πυλών δημιουργούν σύνθετα κυκλώματα που μπορούν να εκτελούν αριθμητικές και λογικές πράξεις, συνδυαστικές πράξεις κ.α. Για την ανάλυση των κυκλωμάτων αυτών χρησιμοποιείται η δίτιμη Άλγεβρα Boole.



Υλοποίηση πυλών με τρανζίστορ a) πύλη NOT, b) πύλη NAND, c) πύλη NOR

Σε ένα ψηφιακό κύκλωμα υπάρχουν μόνο 2 λογικές τιμές. Συνήθως ένα σήμα μεταξύ 0 και 1 Volt αντιπροσωπεύει τη μία τιμή (π.χ. το λογικό 0) και ένα σήμα μεταξύ 2 και 5 Volt αντιπροσωπεύει την άλλη τιμή (π.χ. το λογικό 1). Τιμές έξω από τις περιοχές αυτές δεν επιτρέπονται. Οι πύλες είναι μικροσκοπικές ηλεκτρονικές συσκευές που μπορούν να υπολογίζουν διάφορες συναρτήσεις αυτών των δίτιμων σημάτων.

Το βασικό ηλεκτρονικό εξάρτημα των πυλών είναι τα τρανζίστορ. Στην παραπάνω εικόνα φαίνεται ο τρόπος υλοποίησης των πυλών NOT, NAND, NOR.

Στην υλοποίηση της πύλης NOT, που υλοποιείται με ένα μόνο τρανζίστορ (περίπτωση (a) του σχήματος), παρατηρούμε ότι :

1. Η είσοδος στην πύλη οδηγείται στη βάση του τρανζίστορ (base),
2. Η έξοδος της πύλης λαμβάνεται από το συλλέκτη (collector), ο οποίος είναι συνδεδεμένος στην τάση τροφοδοσίας (πολωμένος) μέσω μιας αντίστασης, ενώ
3. Ο εκπομπός (emitter) της πύλης είναι γειωμένος.

Σύμφωνα με τη λειτουργία του τρανζίστορ όταν εφαρμόζουμε τάση στην βάση του τότε η γραμμή εκπομπού - συλλέκτη «άγει», δηλαδή έχει μεγάλη αγωγιμότητα και παρουσιάζει εξαιρετικά μικρή αντίσταση. Αντίθετα όταν δεν υπάρχει τάση στην βάση του τρανζίστορ, τότε η γραμμή εκπομπού - συλλέκτη παρουσιάζει μεγάλη αντίσταση και πρακτικά θεωρείται ανοικτό κύκλωμα

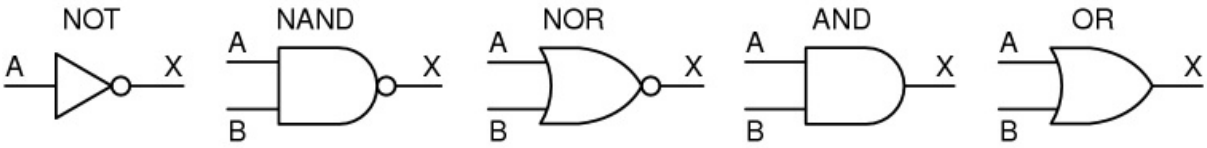
Σύμφωνα λοιπόν με τα παραπάνω όταν στην είσοδο της πύλης εφαρμόζεται λογικό 0, δηλαδή «όχι τάση», τότε η γραμμή εκπομπού - συλλέκτη είναι ανοιχτό κύκλωμα, και επομένως ο

συλλέκτης ταυτίζεται με την τάση τροφοδοσίας, οπότε στην έξοδο της πύλης εμφανίζονται τα 5 Volt της τροφοδοσίας. Αντίθετα, όταν στην είσοδο της πύλης εφαρμόζεται το λογικό 1, δηλαδή 5 Volt τότε η γραμμή εκπομπού - συλλέκτη άγει και αποτελεί σχεδόν βραχυκύκλωμα, οπότε η τάση εξόδου Vout συνδέεται με την γη και επομένως είναι σε δυναμικό 0 Volt. Άρα η πύλη βγάζει στην έξοδό της το λογικό 0.
Επομένως η πύλη αυτή με είσοδο 0 βγάζει έξοδο 1, και με είσοδο 1 βγάζει έξοδο 0, δηλαδή λειτουργεί σωστά ως μία πύλη NOT.

Στην υλοποίηση της πύλης NAND (σχήμα (b)), έχουμε δύο τρανζίστορ που είναι συνδεδεμένα έτσι ώστε οι γραμμές εκπομπού - συλλέκτη να είναι σε σειρά. Έτσι λοιπόν αν μια από τις εισόδους V1 ή V2 είναι στα 0 volt, τότε ένα από τα δύο τρανζίστορ δεν θα άγει (θα είναι ανοικτό κύκλωμα), και επομένως η έξοδος της πύλης θα ταυτίζεται με την τάση τροφοδοσίας οπότε θα είναι στο λογικό 1. Αν τώρα και οι δύο είσοδοι είναι στο λογικό 1, τότε και τα δύο τρανζίστορ θα άγουν, και επομένως η έξοδος της πύλης θα είναι γειωμένη και άρα στο λογικό 0. Έτσι επαληθεύεται ο πίνακας αλήθειας της πύλης NAND.

Στην υλοποίηση της πύλης NOR (σχήμα (c)), έχουμε δύο τρανζίστορ που είναι συνδεδεμένα έτσι ώστε οι γραμμές εκπομπού - συλλέκτη να είναι παράλληλα. Έτσι λοιπόν αν μια από τις εισόδους V1 ή V2 είναι στο λογικό 1 (5 volt), τότε υπάρχει σύνδεση της εξόδου με την γη και επομένως η έξοδος της πύλης είναι στο λογικό 0. Αν τώρα και οι δύο είσοδοι είναι στο λογικό 0 (0 volt), τότε κανένα από τα δύο τρανζίστορ δεν άγει, και επομένως η έξοδος της πύλης θα ταυτίζεται με την Vcc (5 volt) και άρα θα είναι στο λογικό 1. Έτσι επαληθεύεται ο πίνακας αλήθειας της πύλης NOR.

Επίσης, στην επόμενη εικόνα φαίνονται τα σύμβολα και οι Πίνακες Αληθείας των 5 βασικών πυλών NOT, NAND, NOR, AND και OR.



A	X
0	1
1	0

(a)

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

(b)

A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

(c)

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

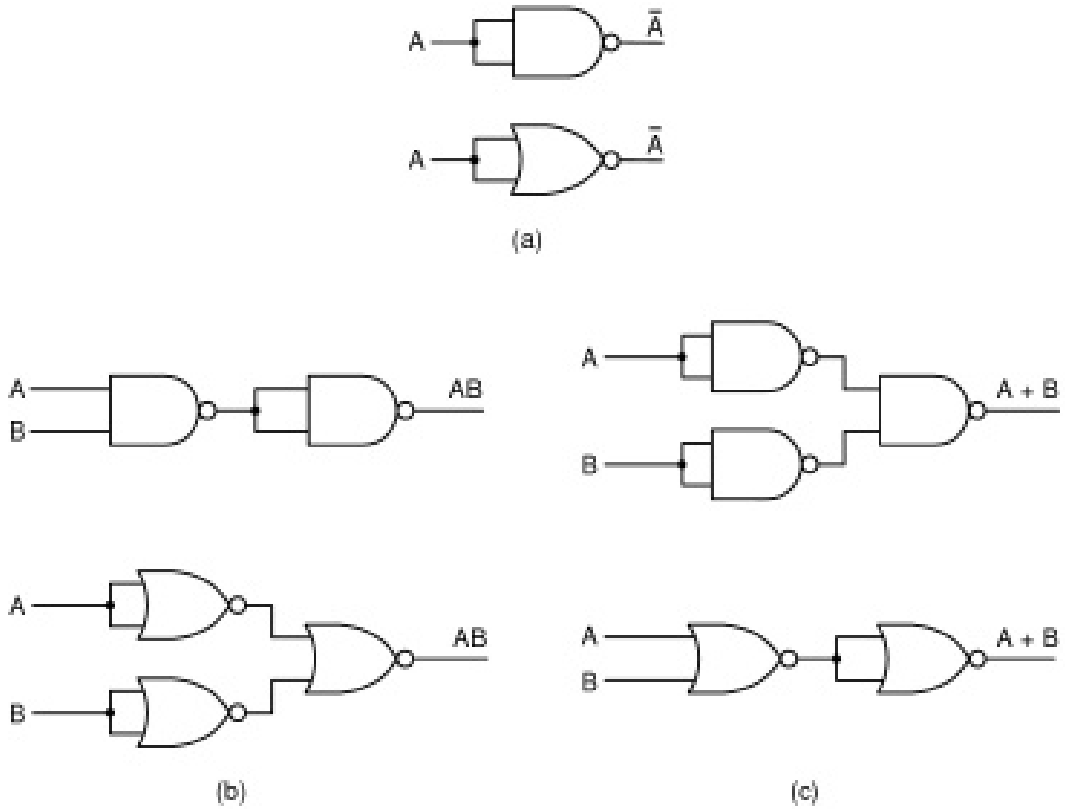
(d)

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

(e)

Σύμβολα και πίνακες αληθείας για 5 βασικές πύλες

Αν και οι πύλες NAND και NOR προκύπτουν με συνδυασμό των απλών πυλών NOT, AND και OR, πολλοί σχεδιαστές προτιμούν να χρησιμοποιούν αυτές τις πύλες στο σχεδιασμό κυκλωμάτων διότι οι NAND και NOR υλοποιούνται με 2 τρανζίστορ έναντι 3 των απλών πυλών. Με βάση δε την Αρχή της Ισοδυναμίας Κυκλωμάτων, και οι πύλες NOT, AND, OR μπορούν να κατασκευαστούν με συνδυασμό των NAND και NOR, όπως φαίνεται στην επόμενη εικόνα:



Υλοποίηση πυλών a) NOT, b) AND, c) OR, μόνο με πύλες NAND ή NOR

Σε επίπεδο ολοκληρωμένων κυκλωμάτων χρησιμοποιούνται οι ακόλουθες τεχνολογίες κατασκευής πυλών :

- Διπολική Τεχνολογία. Υπάρχουν οι ακόλουθοι τύποι διπολικών πυλών :
  - TTL (Transistor-Transistor Logic). Ο βασικός τύπος πυλών που λειτουργούν στα 5 Volt και έχουν χρόνο απόκρισης t=10nsec ($10x10^{-9}$ sec), που σημαίνει ότι μπορούν να λειτουργήσουν σε συχνότητες χρονισμού έως και $f = 1 / t = 1/(10x10^{-9}) = 1 / 10^{-8} = 10^8 = 100MHz$.
  - IIL (Integrated Injection Logic) Λειτουργούν με μικρή τάση (~0.8 V) αλλά έχουν χρόνο απόκρισης μεγαλύτερο από αυτόν της κατηγορίας TTL. Ο χρόνος απόκρισής τους είναι της τάξης των 20-50 nsec. Επομένως μπορούν να λειτουργήσουν σε συχνότητες χρονισμού στην περιοχή 20-50 MHz.
  - ECL (Emitter-Coupled Logic). Χρησιμοποιείται για κυκλώματα υψηλής ταχύτητας. Έχουν χρόνο απόκρισης της τάξης των 0.3 – 2 nsec οπότε μπορούν να χρονιστούν σε συχνότητες 500MHz – 3 GHz.
- Τεχνολογία MOS (Metal Oxide Semiconductor). Υλοποιούνται με τρανζίστορ FET (Field Efect Transistor). Οι πύλες MOS είναι πολύ πιο αργές από τις διπολικές (10-100 φορές) αλλά απαιτούν ελάχιστη ισχύ και καταλαμβάνουν ελάχιστο χώρο. Οι πύλες αυτές εμφανίζονται σε πολλές παραλλαγές όπως PMOS, NMOS, HMOS και CMOS. Ο M/E Intel 8088 είναι τεχνολογίας HMOS.

Για πολλά χρόνια το όριο μεγέθους των τρανζίστορ στα ολοκληρωμένα κυκλώματα μεγάλης κλίμακας ολοκλήρωσης (VLSI), ήταν στα 0,6 μ (micron) ή 600 nm (nanometers). Όμως 3 μεγάλες κατασκευάστριες εταιρείες, η Intel, η AMD και η Motorolla ένωσαν τις δυνάμεις τους και το XXXXXXXXXX εφηύραν την λιθογραφική μέθοδο ακτίνων Χ που ξεπέρασε το αρχικό όριο και έφτασε άνετα στα 0,09 μ (micron) ή 90 nm (nanometers). Η τεχνολογία αυτή

σήμερα έχει φτάσει μέχρι και στο όριο των 45 nm όπως στους Μικροεπεξεργαστές Pentium 4 Quad Q6600, AMD Phenom XXXXX.
Για τις συνδέσεις μεταξύ των τρανζίστορ χρησιμοποιούνταν κατά κόρον το αλουμίνιο, καθώς είναι ένα φτηνό μέταλλο με καλή αγωγιμότητα και ελατότητα. Από το 1997 όμως, αντί για το αλουμίνιο χρησιμοποιείται χαλκός, που αν και ακριβότερος, διαθέτει ακόμα καλύτερη αγωγιμότητα αλλά και ελατότητα, δηλαδή μπορεί να δώσει νήματα εξαιρετικά μικρής διατομής (μικρότερης από 0,1 μ = 100 nm) χωρίς να σπάει.

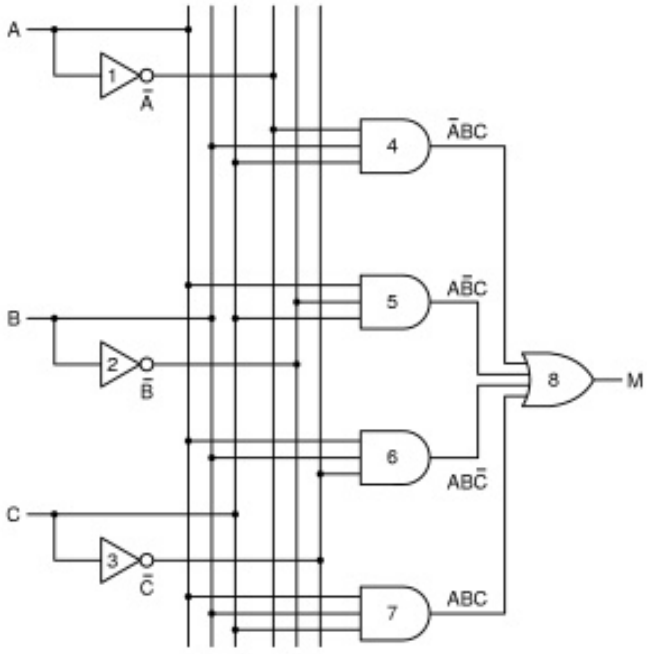
3.2. Υλοποίηση Συναρτήσεων Boole

Στον σχεδιασμό ψηφιακών ηλεκτρονικών κυκλωμάτων, αρχικά προσδιορίζεται η συνάρτηση της Αλγεβρας Boole που θέλουμε να υλοποιήσουμε και, κατόπιν, προσπαθούμε να σχεδιάσουμε ένα κύκλωμα με πύλες το οποίο να υλοποιεί την συνάρτηση αυτή. Ο σχεδιασμός απλοποιείται αν, χρησιμοποιώντας τους κανόνες της άλγεβρας Boole, μετασχηματίσουμε την συνάρτηση σε άθροισμα (OR) 2^n γινομένων (AND).
Στη συνέχεια θα δώσουμε ένα παράδειγμα σχεδιασμού κυκλώματος με βάση δοθείσα συνάρτηση, χωρίς να μπούμε σε πολλές λεπτομέρειες, ώστε να αντιληφθεί ο σπουδαστής τις βασικές αρχές της σχεδίασης υπολογιστικών κυκλωμάτων.
Έστω ότι θέλουμε να κατασκευάσουμε ένα ψηφιακό κύκλωμα που να υλοποιεί την συνάρτηση :

$$M = \overline{A}\,BC + A\,\overline{B}\,C + AB\,\overline{C} + ABC$$

Τα βήματα που ακολουθούμε στον σχεδιασμό του κυκλώματος είναι τα ακόλουθα :
1. Γράφουμε τον Πίνακα Αληθείας της συνάρτησης
2. Προσθέτουμε στις εισόδους πύλες NOT για την παραγωγή του συμπληρώματος κάθε εισόδου
3. Σχεδιάζουμε μια πύλη AND για κάθε όρο που έχει bit με τιμή 1 στη στήλη των αποτελεσμάτων
4. Συνδέουμε τις πύλες AND με τις κατάλληλες εισόδους
5. Στέλνουμε την έξοδο όλων των πυλών AND σε μια πύλη OR

Στην επόμενη εικόνα φαίνεται το ψηφιακό κύκλωμα που προκύπτει, το οποίο χρησιμοποιεί συνολικά 8 πύλες. Το κύκλωμα που προέκυψε είναι λειτουργικό αλλά δεν είναι ακόμη βελτιστοποιημένο. Κατασκευαστικά, είναι προτιμότερο να χρησιμοποιείται ο ίδιος τύπος πυλών. Έτσι, στο κύκλωμα αυτό είναι προτιμότερο να αντικατασταθούν οι πύλες NOT, AND και OR με τους ισοδύναμους συνδυασμούς NAND και NOR. Οι πύλες NAND και NOR λέγονται και «Πλήρεις Πύλες» γιατί είναι οι μόνες με τις οποίες μπορεί να υλοποιηθεί οποιαδήποτε συνάρτηση.



A	B	C	M
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

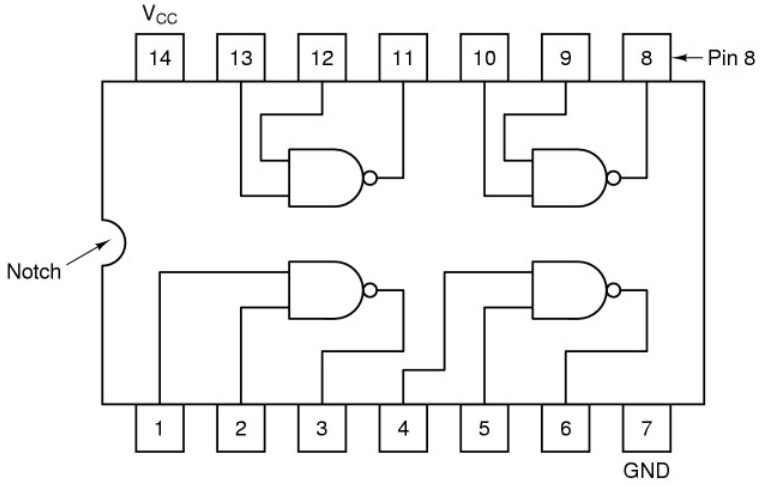
Κύκλωμα υλοποίησης της συνάρτησης M(A,B,C) και πίνακας αληθείας

3.3. Ολοκληρωμένα Κυκλώματα

Οι πύλες δεν κατασκευάζονται ούτε πωλούνται μεμονωμένα, αλλά σε μονάδες που λέγονται Ολοκληρωμένα Κυκλώματα (Integrated Circuits, IC) ή απλά chips. Ένα chip είναι ένα τετράγωνο κομμάτι πυριτίου, πάνω στο οποίο τοποθετείται ένας αριθμός πυλών. Τα μικρά ολοκληρωμένα κυκλώματα είναι συνήθως συσκευασμένα σε ένα ορθογώνιο πλαστικό ή κεραμικό περίβλημα που τους παρέχει μηχανική στήριξη και προστασία από το περιβάλλον. Κατά μήκος των πλευρών τους υπάρχουν 2 παράλληλες σειρές ακροδεκτών, οι οποίοι κολλούνται απ' ευθείας πάνω στην πλακέτα (Printed Circuit Board, PCB) ή εισάγονται σε ειδικές βάσεις. Κάθε ακροδέκτης συνδέεται με την είσοδο ή έξοδο μιας πύλης του chip ή με την τροφοδοσία ή γείωση του ολοκληρωμένου. Οι συσκευασίες αυτές με τις 2 παράλληλες σειρές ακροδεκτών λέγονται DIP (Dual Inline Package) και έχουν συνήθως 14,16,18,20,22,24,28,40,64 ή 68 ακροδέκτες. Τα μεγαλύτερα σε μέγεθος chip χρησιμοποιούν συνήθως τετραγωνικές συσκευασίες με ακροδέκτες είτε από κάτω είτε και στις 4 πλευρές.
Ανάλογα με τον αριθμό των πυλών που περιέχουν, τα chip χωρίζονται στις παρακάτω κατηγορίες.

- Κυκλώματα **SSI** (Small Scale Integration). Ολοκλήρωση μικρής κλίμακας. Περιέχουν 1-10 πύλες
- Κυκλώματα **MSI** (Medium Scale Integration). Ολοκλήρωση μεσαίας κλίμακας. Περιέχουν 10-100 πύλες
- Κυκλώματα **LSI** (Large Scale Integration). Ολοκλήρωση μεγάλης κλίμακας. Περιέχουν 100-100.000 πύλες
- Κυκλώματα **VLSI** (Very Large Scale Integration). Ολοκλήρωση πολύ μεγάλης κλίμακας. Περιέχουν πάνω από 100.000 πύλες

Στην επόμενη εικόνα φαίνεται το σχηματικό διάγραμμα ενός κοινού chip SSI που περιέχει 4 πύλες NAND.



Ολοκληρωμένο κύκλωμα SSI με 4 πύλες NAND.

Κατά τον σχεδιασμό κυκλωμάτων, θεωρούμε τις πύλες ιδανικές με την έννοια ότι η έξοδος εμφανίζεται αμέσως μόλις εφαρμοστεί η είσοδος. Στην πραγματικότητα, όλα τα chip έχουν μια Καθυστέρηση Πύλης (Gate Delay), η οποία περιλαμβάνει το χρόνο διάδοσης του σήματος μέσω του chip και το χρόνο μεταγωγής, και κυμαίνεται από 0,3 έως 50 nsec.
Εκτός από την καθυστέρηση πύλης, υπάρχει και άλλο ένα μέγεθος που είναι σημαντικό στα ολοκληρωμένα κυκλώματα και αυτό είναι το λεγόμενο fan-out. Το fan-out είναι στην ουσία ένας αριθμός που εκφράζει το πόσες πύλες ίδιας τεχνολογίας μπορούμε να συνδέσουμε στην έξοδο μιας πύλης, έτσι ώστε στην έξοδο της τελευταίας πύλης να μπορεί να γίνεται σαφής διάκριση μεταξύ των τιμών 0 και 1. Επομένως το fan-out εκφράζει το πόσα στάδια (από πύλη σε πύλη) μπορεί να έχει ένα ψηφιακό κύκλωμα που σκοπεύουμε να υλοποιήσουμε με ολοκληρωμένα κυκλώματα της συγκεκριμένης τεχνολογίας.
Ενώ στα chip SSI κάθε είσοδος ή έξοδος μιας πύλης αντιστοιχεί σε έναν ακροδέκτη, στα chip MSI και πάνω είναι τεχνικά δύσκολο έως αδύνατον να έχει κάθε πύλη ακροδέκτες, καθώς θα απαιτούνταν μεγάλος αριθμός ακροδεκτών, που σε ένα chip VLSI θα ήταν μερικά εκατομμύρια! Γι' αυτό, στα chip αυτά ένας αριθμός πυλών συνδυάζονται εσωτερικά για να προσφέρουν μια χρήσιμη λειτουργία ενώ εξωτερικά απαιτούν έναν μικρό αριθμό ακροδεκτών. Στη συνέχεια θα εξετάσουμε μερικά τέτοια κυκλώματα που χρησιμοποιούνται σε υπολογιστές.

3.4. Συνδυαστικά Κυκλώματα

Πολλές εφαρμογές ψηφιακής λογικής απαιτούν ένα κύκλωμα με πολλές εισόδους και πολλές εξόδους, στο οποίο οι έξοδοι καθορίζονται μονοσήμαντα από τις τρέχουσες εισόδους. Τα κυκλώματα αυτά λέγονται Συνδυαστικά (Combinational). Θα εξετάσουμε εν συντομία μερικά βασικά συνδυαστικά κυκλώματα.

3.4.1. Αποκωδικοποιητές (Decoders)

Οι Αποκωδικοποιητές (Decoders) είναι κυκλώματα με n εισόδους και 2^n εξόδους. Ανάλογα με την τιμή στις γραμμές εισόδου επιλέγεται μία από τις εξόδους.

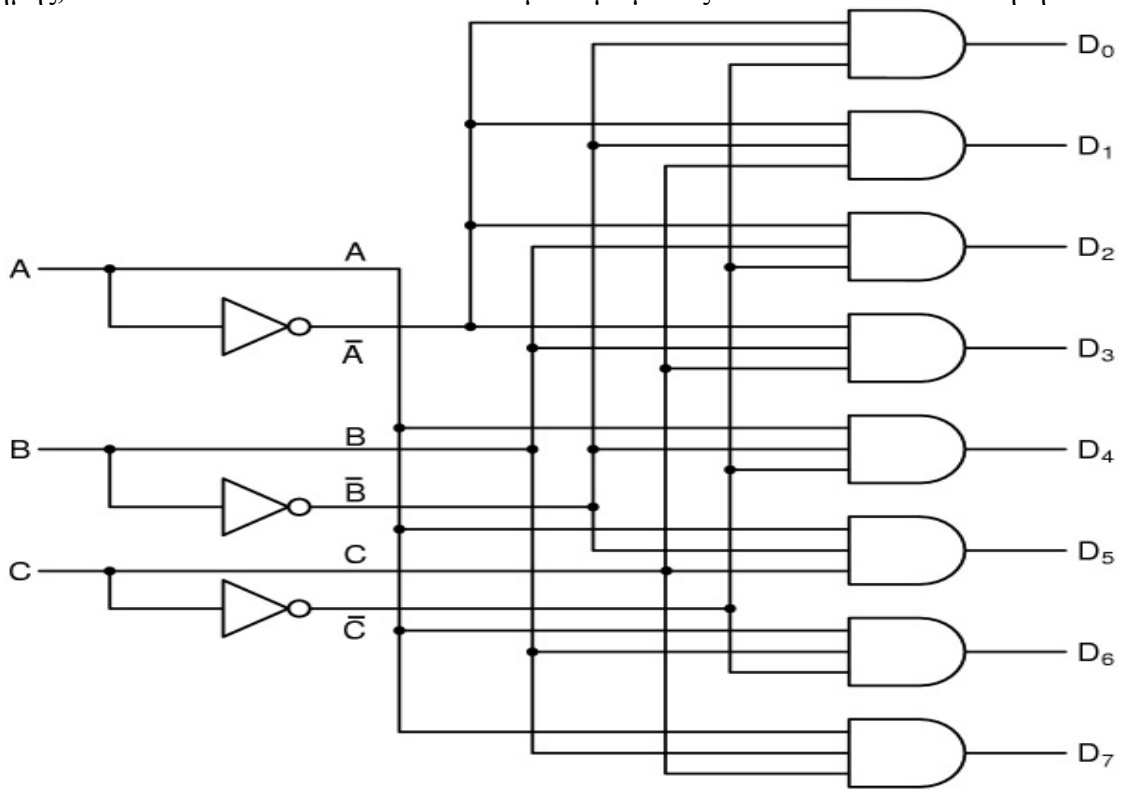
Για την υλοποίηση του κυκλώματος, για καθεμία από τις εισόδους λαμβάνουμε και την αντιστροφή της. Στην έξοδο του κυκλώματος έχουμε τόσες πύλες AND όσες και οι έξοδοι.

Σε κάθε πύλη AND που είναι πολλαπλών εισόδων, οδηγούμε τις κατάλληλες γραμμές των εισόδων η των αντίστροφών τους.

Για παράδειγμα η έξοδος D0 θέλουμε να ενεργοποιείται (να είναι σε λογικό 1) στην περίπτωση που και οι τρεις είσοδοι θα είναι 0, δηλαδή (A,B,C)=(0,0,0). Για να ενεργοποιηθεί όμως η πύλη και να δώσει στην έξοδο λογικό 1, θα πρέπει όλες οι είσοδοι της να είναι μονάδα. Έτσι οδηγούμε στην είσοδο της πύλης αυτής τα αντίστροφα των εισόδων, δηλαδή τα ($\overline{A}$, $\overline{B}$, $\overline{C}$).

Ομοίως, η έξοδος D1 θα πρέπει να ενεργοποιείται όταν οι είσοδοι θα είναι : (A,B,C)=(0,0,1). Επομένως θα οδηγήσουμε στην $2^{η}$ πύλη AND τα ($\overline{A}$, $\overline{B}$,C). Με παρόμοιο τρόπο καθορίζουμε τις εισόδους και στις υπόλοιπες πύλες.

Οι αποκωδικοποιητές χρησιμοποιούνται στους υπολογιστές κυρίως για επιλογή chip ή κυκλωμάτων. Στην επόμενη εικόνα φαίνεται ένας αποκωδικοποιητής 3 σε 8. Μια εφαρμογή του κυκλώματος αυτού μπορεί να είναι η επιλογή chip μνήμης. Εστω ένας υπολογιστής με μνήμη που αποτελείται από 8 chip, καθένα από τα οποία περιέχει 1MB. Το chip 0 περιέχει τις διευθύνσεις 0-1MB, το chip 1 τις διευθύνσεις 1-2MB κ.ο.κ. Όταν ζητείται μια διεύθυνση μνήμης πως θα γνωρίζει το κύκλωμα ελέγχου της μνήμης ποιο chip να προσπελάσει; Η απάντηση είναι χρησιμοποιώντας τα 3 bit υψηλής τάξης των διευθύνσεων για την επιλογή ενός από τα 8 chip. Τα 3 αυτά bit οδηγούνται στην είσοδο του αποκωδικοποιητή 3 σε 8 και ανάλογα με την τιμή που έχουν ενεργοποιούν ένα από τα 8 chip μνήμης, καθένα από τα οποία είναι συνδεδεμένο με μία έξοδο του αποκωδικοποιητή.



Αποκωδικοποιητής 3 σε 8

3.4.2. Πολυπλέκτες (Multiplexers)

Οι Πολυπλέκτες (Multiplexers) είναι κυκλώματα με 2^{n} εισόδους, μία έξοδο και n εισόδους ελέγχου που επιλέγουν μία από τις εισόδους. Η επιλεγμένη είσοδος συνδέεται τότε με την έξοδο. Στην επόμενη εικόνα φαίνεται ένας πολυπλέκτης 8 εισόδων, 1 εξόδου και 3

γραμμών ελέγχου. Οι γραμμές D0-D7 αποτελούν τις εισόδους του ενώ οι γραμμές A,B,C είναι οι γραμμές ελέγχου. Οι 3 γραμμές ελέγχου μπορούν να πάρουν 2^3=8 δυνατούς συνδυασμούς τιμών, κάθε ένας εκ των οποίων επιλέγει μια από τις 8 εισόδους.

Για την υλοποίηση του κυκλώματος, για καθεμία από τις γραμμές ελέγχου λαμβάνουμε και την αντιστροφή της. Στην έξοδο του κυκλώματος έχουμε τόσες πύλες AND όσες και οι είσοδοι οι οποίες οδηγούνται όλες σε μία πύλη OR πολλαπλών εισόδων.

Σε κάθε πύλη AND που είναι επίσης πολλαπλών εισόδων, οδηγούμε την αντίστοιχη είσοδο και τις κατάλληλες γραμμές ελέγχου ή τα αντίστροφά τους.

Για παράδειγμα, η είσοδος D0 θα οδηγηθεί στην έξοδο όταν οι γραμμές ελέγχου θα είναι (A,B,C)=(0,0,0). Για να γίνει αυτό θα πρέπει να ενεργοποιηθεί η $1^η$ πύλη AND. Αυτό σημαίνει ότι θα πρέπει όλες οι είσοδοι της $1^{ης}$ πύλης AND που προέρχονται από τις γραμμές ελέγχου να είναι όλες 1. Έτσι το τι θα βγει στην έξοδο εξαρτάται αποκλειστικά από την είσοδο D0. Αν η είσοδος D0 είναι 0 τότε η έξοδος της πύλης AND θα είναι επίσης 0. Αν όμως η είσοδος D0 είναι 1 τότε και η έξοδος της πύλης θα είναι 1, αφού και οι άλλες είσοδοι που προέρχονται από τις γραμμές ελέγχου είναι 1. Έτσι οδηγούμε στην είσοδο της πύλης αυτής τα αντίστροφα των γραμμών ελέγχου, δηλαδή τα ($\overline{A}$, $\overline{B}$, $\overline{C}$).

Πρέπει εδώ να σημειωθεί ότι όταν οι γραμμές ελέγχου είναι (A,B,C)=(0,0,0) τότε μόνο η $1^η$ πύλη AND ενεργοποιείται, ενώ όλες οι άλλες παραμένουν απενεργοποιημένες, δηλαδή βγάζουν στην εξοδό τους 0, ανεξάρτητα από την αντίστοιχη είσοδο, καθώς από τις εισόδους τους που προέρχονται από τις γραμμές ελέγχου υπάρχει τουλάχιστον ένα 0.

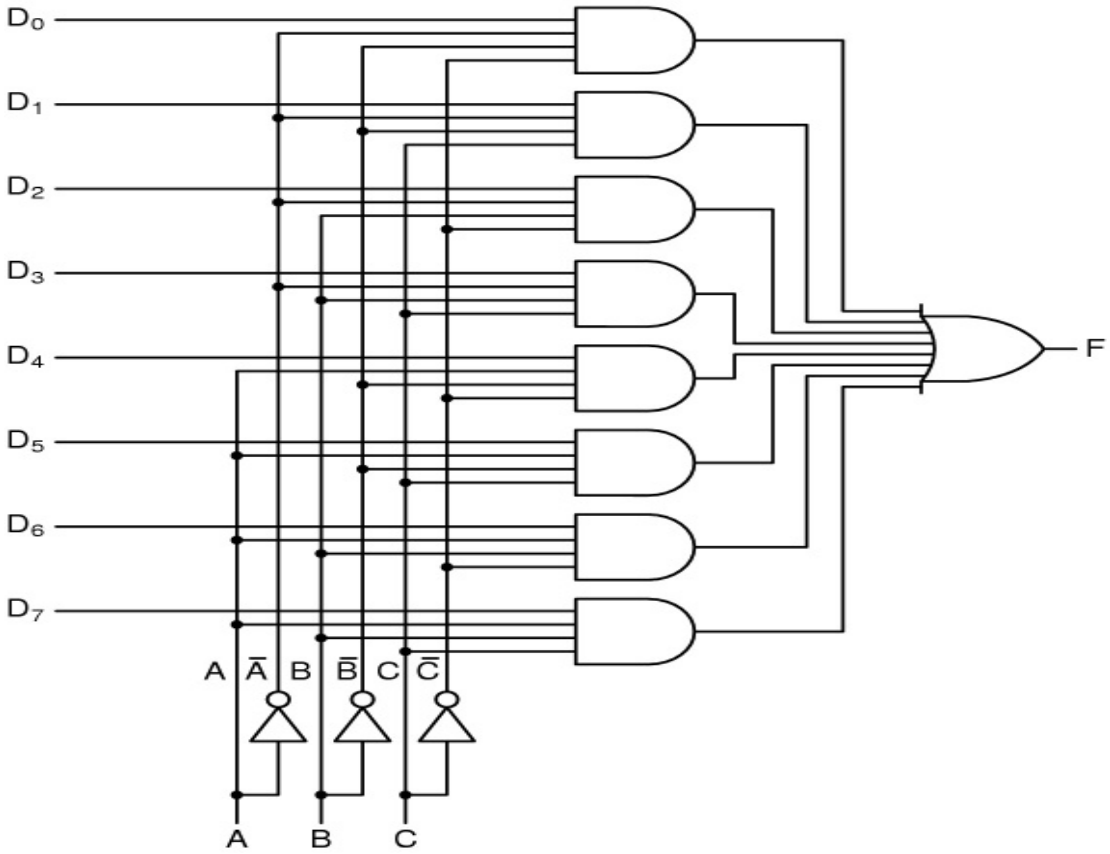
Ομοίως η είσοδος D1 θα οδηγείται στην έξοδο όταν οι γραμμές ελέγχου θα είναι : (A,B,C)=(0,0,1). Επομένως θα οδηγήσουμε στην $2^η$ πύλη AND την δεύτερη είσοδο (D1) και τα ($\overline{A}$, $\overline{B}$,C). Με παρόμοιο τρόπο καθορίζουμε τις εισόδους και στις υπόλοιπες πύλες.

Η συγκεντρωτική πύλη OR στην έξοδο του κυκλώματος κάνει την εξής δουλειά: Αν εμφανιστεί μία μονάδα από οποιαδήποτε πύλη AND τότε αυτή οδηγείται στην έξοδο. Αν όλες οι πύλες AND δίνουν 0 (ακόμα και η ενεργοποιημένη) τότε η έξοδος θα είναι 0.

Συνήθης εφαρμογή των πολυπλεκτών στους υπολογιστές είναι ο διαμοιρασμός κάποιου κυκλώματος σε πολλά άλλα. Για παράδειγμα, μια ομάδα αγωγών στην μητρική πλακέτα μπορεί να χρησιμοποιείται άλλοτε σαν μέρος του data bus και άλλοτε σαν μέρος του address bus. Ο πολυπλέκτης τοποθετείται στο σημείο όπου πολλοί δίαυλοι καταλήγουν σε έναν.

Άλλη χρήση είναι η μετατροπή παράλληλων δεδομένων σε σειριακά. Στο κύκλωμα της εικόνας για παράδειγμα, από τη στιγμή που στην είσοδο του πολυπλέκτη εμφανιστούν τα παράλληλα δεδομένα, δίνοντας στις γραμμές ελέγχου διαδοχικά τις τιμές 000 έως 111 με ένα κύκλωμα μετρητή (counter), στην έξοδο θα εμφανιστούν διαδοχικά τα δεδομένα εισόδου σε σειριακή μορφή. Ο ρυθμός σειριακής μετάδοσης των δεδομένων στην έξοδο καθορίζεται από τον ρυθμό με τον οποίο αλλάζει τιμές ο μετρητής (counter). Επίσης θα πρέπει κατά την επαναφορά των τιμών στις γραμμές ελέγχου από το 111 ξανά στο 000 θα πρέπει να ειδοποιείται η συσκευή που εξάγει παράλληλα δεδομένα ώστε να εμφανίσει την επόμενη λέξη.

Επίσης ο πολυπλέκτης χρησιμοποιείται για ανάγνωση μνήμης. Υποθέστε ότι πριν από κάθε μια από τις εισόδους βρίσκεται και ένα κελί μνήμης. Στο παράδειγμα του σχήματος θα έχουμε μια μνήμη με οκτώ συνολικά κελιά. Για να διαβάσουμε ένα οποιοδήποτε κελί αρκεί να σχηματίσουμε στις γραμμές ελέγχου την κατάλληλη «διεύθυνση μνήμης», δηλαδή τον κατάλληλο ψηφιακό αριθμό από 000 έως 111. Δίνοντας την κατάλληλη διεύθυνση στις γραμμές ελέγχου το αντίστοιχο κελί (γραμμή εισόδου) οδηγείται στην έξοδο, η οποία βέβαια αποτελεί τον κοινό δίαυλο δεδομένων από τον οποίο γίνεται η ανάγνωση της μνήμης.



Πολυπλέκτης 8 εισόδων, 1 εξόδου και 3 γραμμών ελέγχου

3.4.3. Αποπολυπλέκτες (Demultiplexers)

Το αντίστροφο κύκλωμα ονομάζεται Αποπολυπλέκτης (Demultiplexer) και δρομολογεί το σήμα της μοναδικής εισόδου του σε μία από τις 2^n εξόδους του, ανάλογα με τις τιμές στις n γραμμές ελέγχου. Οι Αποπολυπλέκτες (Demultiplexers) είναι κυκλώματα με 1 είσοδο, n γραμμές ελέγχου και 2^n εξόδους. Ανάλογα με τον συνδυασμό των τιμών στις γραμμές ελέγχου, η είσοδος οδηγείται σε μία από τις εξόδους. Στην επόμενη εικόνα φαίνεται ένας αποπολυπλέκτης 4 εξόδων, 1 εισόδου και 2 γραμμών ελέγχου. Οι 2 γραμμές ελέγχου μπορούν να πάρουν 2^2=4 δυνατούς συνδυασμούς τιμών, κάθε ένας εκ των οποίων συνδέει την μοναδική είσοδο στην αντίστοιχη έξοδο.

Για την υλοποίηση του κυκλώματος, για καθεμία από τις γραμμές ελέγχου λαμβάνουμε και την αντιστροφή της. Στην έξοδο του κυκλώματος έχουμε τόσες πύλες AND όσες και οι έξοδοι.

Σε κάθε πύλη AND που είναι επίσης πολλαπλών εισόδων, οδηγούμε την μοναδική είσοδο και τις κατάλληλες γραμμές ελέγχου ή τα αντίστροφά τους.

Για παράδειγμα, η είσοδος θα οδηγηθεί στην $1^η$ έξοδο όταν οι γραμμές ελέγχου θα είναι (A1,A0)=(0,0). Για να γίνει αυτό θα πρέπει να ενεργοποιηθεί η $1^η$ πύλη AND. Αυτό σημαίνει ότι θα πρέπει όλες οι είσοδοι της $1^{ης}$ πύλης AND που προέρχονται από τις γραμμές ελέγχου να είναι όλες 1. Έτσι το τι θα βγει στην έξοδο της πύλης AND εξαρτάται αποκλειστικά από την είσοδο D. Αν η είσοδος D είναι 0 τότε η έξοδος της πύλης AND θα είναι επίσης 0. Αν όμως η είσοδος D είναι 1 τότε και η έξοδος της πύλης θα είναι 1, αφού και οι άλλες είσοδοι που προέρχονται από τις γραμμές ελέγχου είναι 1. Έτσι οδηγούμε στην είσοδο της πύλης αυτής τα αντίστροφα των γραμμών ελέγχου, δηλαδή τα ($\overline{A1}$, $\overline{A0}$).

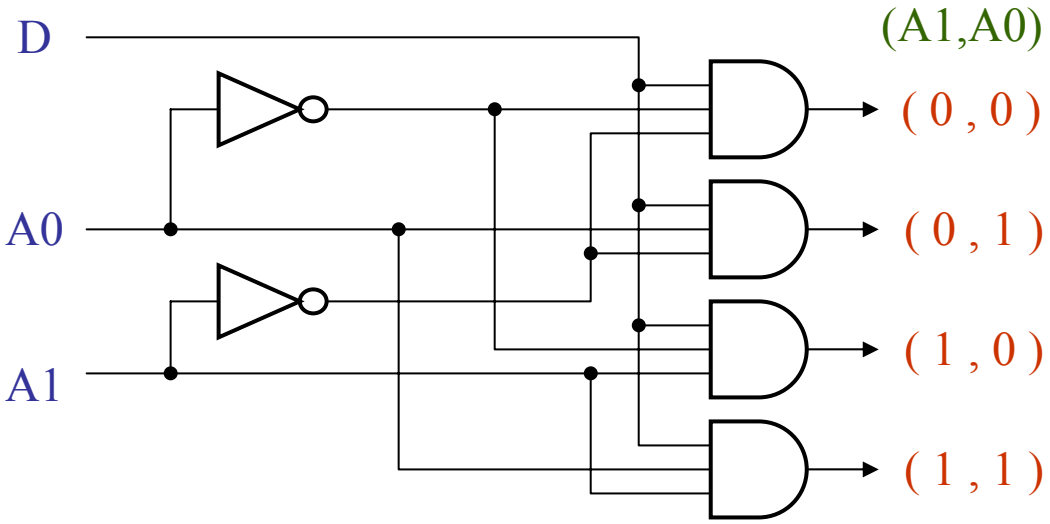
Πρέπει εδώ να σημειωθεί ότι όταν οι γραμμές ελέγχου είναι (A1,A0)=(0,0) τότε μόνο η $1^{η}$ πύλη AND ενεργοποιείται, ενώ όλες οι άλλες παραμένουν απενεργοποιημένες, δηλαδή βγάζουν στην έξοδό τους 0, ανεξάρτητα από την είσοδο, καθώς από τις εισόδους τους που προέρχονται από τις γραμμές ελέγχου υπάρχει τουλάχιστον ένα 0.

Ομοίως η είσοδος D θα οδηγείται στην $2^{η}$ έξοδο όταν οι γραμμές ελέγχου θα είναι : (A1,A0)=(0,1). Επομένως θα οδηγήσουμε στην $2^{η}$ πύλη AND την μοναδική είσοδο D και τα ($\overline{A1}$,A0). Με παρόμοιο τρόπο καθορίζουμε τις εισόδους και στις υπόλοιπες πύλες.

Συνήθης εφαρμογή των αποπολυπλεκτών στους υπολογιστές είναι ο διαμοιρασμός κάποιου κυκλώματος σε πολλά άλλα. Για παράδειγμα, μια ομάδα αγωγών στην μητρική πλακέτα μπορεί να χρησιμοποιείται άλλοτε σαν μέρος του data bus και άλλοτε σαν μέρος του address bus. Ο αποπολυπλέκτης τοποθετείται στο σημείο όπου ένας δίαυλος διαμοιράζεται σε πολλούς.

Άλλη χρήση είναι η μετατροπή σειριακών δεδομένων σε παράλληλα. Στο κύκλωμα της εικόνας για παράδειγμα, μπορούμε να εφαρμόσουμε το σειριακό σήμα στην μοναδική είσοδο D. Εν συνεχεία δίνοντας στις γραμμές ελέγχου διαδοχικά τις τιμές 00 έως 11 με ένα κύκλωμα μετρητή (counter), τα σειριακά bit θα τοποθετούνται στην κατάλληλη έξοδο, σχηματίζοντας μία «παράλληλη» λέξη. Ο ρυθμός με τον οποίο αλλάζει τιμές ο μετρητής (counter) θα πρέπει να ταυτίζεται με τον ρυθμό σειριακής μετάδοσης των δεδομένων. Επίσης θα πρέπει κατά την επαναφορά των τιμών στις γραμμές ελέγχου από το 11 ξανά στο 00 θα πρέπει να ειδοποιείται η συσκευή που διαβάζει τα παράλληλα δεδομένα ώστε να διαβάσει την ολοκληρωμένη λέξη και να μπει σε αναμονή για την επόμενη.

Επίσης ο αποπολυπλέκτης χρησιμοποιείται για εγγραφή μνήμης. Υποθέστε ότι μετά από κάθε μια από τις εξόδους βρίσκεται και ένα κελί μνήμης. Στο παράδειγμα του σχήματος θα έχουμε μια μνήμη με τέσσερα συνολικά κελιά. Για να γράψουμε σε ένα οποιοδήποτε κελί αρκεί να σχηματίσουμε στις γραμμές ελέγχου την κατάλληλη «διεύθυνση μνήμης», δηλαδή τον κατάλληλο ψηφιακό αριθμό από 00 έως 11. Δίνοντας την κατάλληλη διεύθυνση στις γραμμές ελέγχου, η μοναδική είσοδος που θα αποτελεί τον κοινό δίαυλο δεδομένων οδηγείται στο αντίστοιχο κελί μνήμης (γραμμή εξόδου).



Αποπολυπλέκτης 1 εισόδου, 2 γραμμών ελέγχου και 4 εξόδων

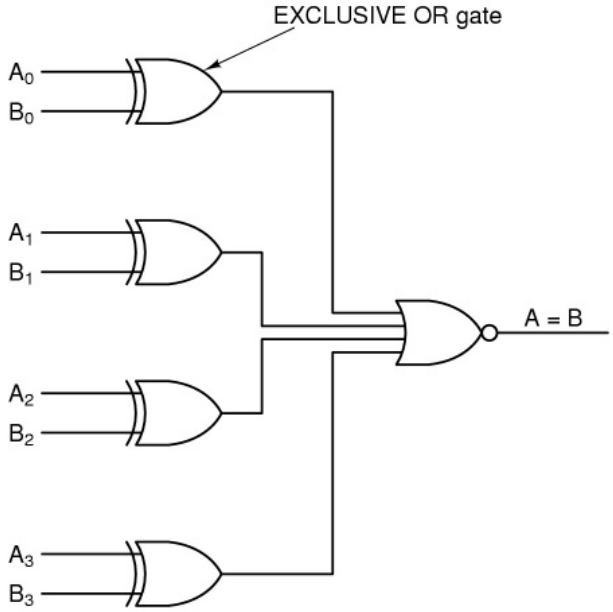
3.4.4. Συγκριτές (Comparators)

Οι Συγκριτές (Comparators) είναι κυκλώματα που συγκρίνουν τα δεδομένα στην είσοδό τους και δίνουν ανάλογη τιμή στην έξοδό τους. Στην επόμενη εικόνα φαίνεται ένας απλός συγκριτής ο οποίος δέχεται σαν είσοδο τις 4-bit λέξεις A και B και στην έξοδο δίνει τιμή 1 αν είναι ίσες και 0 αν είναι άνισες. Το κύκλωμα βασίζεται στην πύλη XOR η οποία δίνει έξοδο 0 όταν οι είσοδοι της είναι ίσες και 1 όταν είναι διαφορετικές.

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Ο πίνακας αληθείας της XOR που μπορεί να χρησιμοποιηθεί για σύγκριση αριθμών

Αν οι 2 λέξεις εισόδου είναι ίσες, και οι 4 XOR θα δώσουν έξοδο 0. Οι έξοδοι των XOR αποτελούν την είσοδο μιας NOR. Αν η έξοδος της NOR είναι 1 οι λέξεις εισόδου είναι ίδιες αλλιώς είναι διαφορετικές.

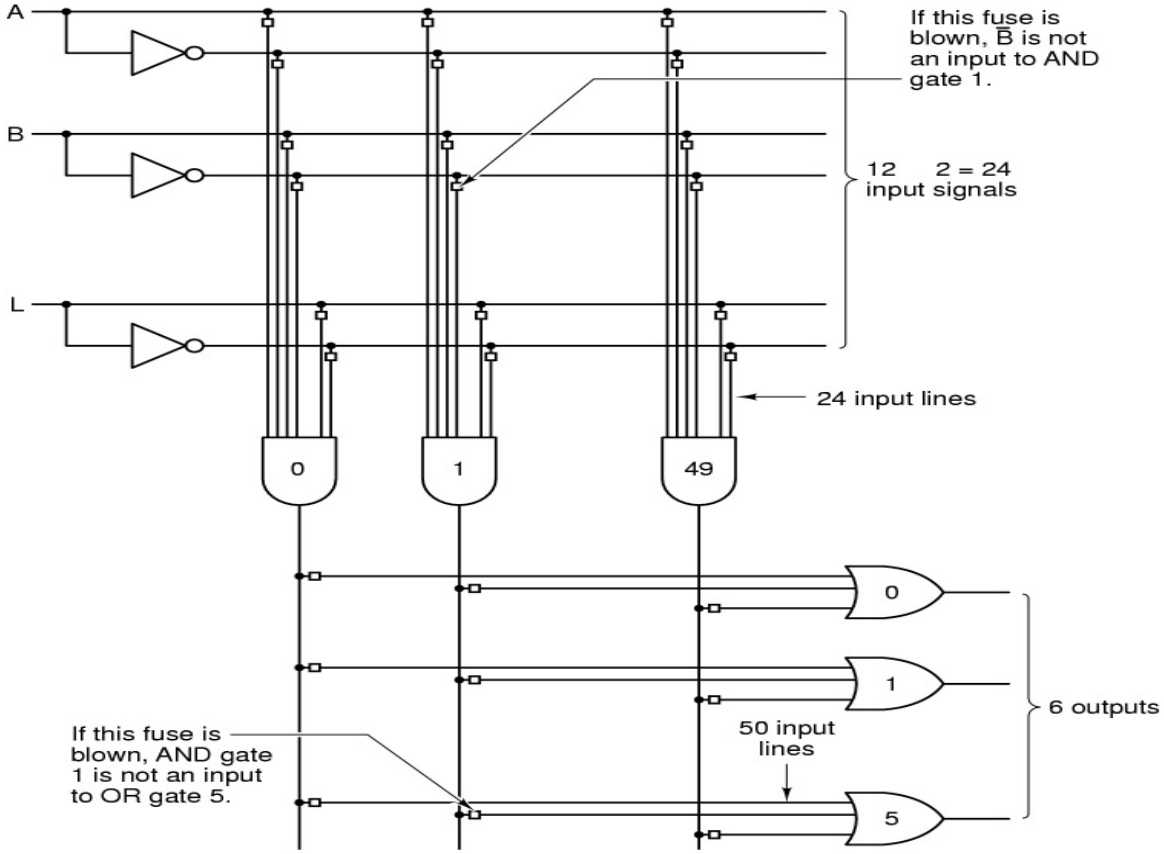


Συγκριτής δύο λέξεων των 4 bit

3.4.5. Προγραμματιζόμενοι Λογικοί Πίνακες (Programmable Logic Arrays – PLAs)

Όπως είδαμε πιο πάνω, κάθε συνάρτηση μπορεί να κατασκευαστεί αν μετατραπεί σε άθροισμα 2^n γινομένων, όπου τα γινόμενα υλοποιούνται με πύλες AND και τα αθροίσματα με πύλες OR. Οι Προγραμματιζόμενοι Λογικοί Πίνακες (Programmable Logic Array - PLAs) είναι κυκλώματα γενικής χρήσης τα οποία μπορούν να προγραμματιστούν από τον χρήστη για την υλοποίηση μιας λογικής συνάρτησης αθροίσματος γινομένων. Στην επόμενη εικόνα φαίνεται ένα PLA 12 εισόδων και 6 εξόδων. Καρδιά του κυκλώματος αυτού είναι ένας πίνακας πυλών AND στην είσοδο και ένας πίνακας πυλών OR στην έξοδο. Όταν το ολοκληρωμένο κύκλωμα κατασκευάζεται στο εργοστάσιο, όλες οι πύλες επικοινωνούν μεταξύ τους μέσω εύτηκτων ασφαλειών, δηλαδή ασφαλειών που καίγονται εύκολα αν περάσει από μέσα τους ρεύμα κατάλληλης έντασης. Ο χρήστης, μπορεί να προγραμματίσει

το PLA χρησιμοποιώντας ένα ειδικό κύκλωμα PLA-Programmer, το οποίο καίει επιλεκτικά κάποιες ασφάλειες, εφαρμόζοντας υψηλή τάση στο chip, αφήνοντας έτσι μόνο τις επιθυμητές συνδέσεις για την υλοποίηση της εκάστοτε συνάρτησης.



Προγραμματιζόμενος Λογικός Πίνακας 12 εισόδων και 6 εξόδων με 50 πύλες AND

3.5. Αριθμητικά Κυκλώματα

Στην κατηγορία αυτή ανήκουν τα κυκλώματα που πραγματοποιούν αριθμητικές πράξεις.

3.5.1. Ολισθητές (Shifters)

Οι Ολισθητές (Shifters) είναι κυκλώματα n εισόδων, n εξόδων και 1 γραμμής ελέγχου. Η έξοδος του κυκλώματος είναι τα bit εισόδου μετατοπισμένα κατά 1 bit. Την φορά της ολίσθησης, προς τα δεξιά ή αριστερά, καθορίζει το bit στη γραμμή ελέγχου. Σε μία ψηφιακή λέξη η ολίσθηση προς τα αριστερά σημαίνει πολλαπλασιασμός επί 2, ενώ ολίσθηση προς τα δεξιά σημαίνει διαίρεση δια 2.

Στην επόμενη εικόνα φαίνεται ένας ολισθητής 8 bit. Τα bit εισόδου D0-D7 εμφανίζονται στην έξοδο S0-S7 μετατοπισμένα κατά 1 bit δεξιά ή αριστερά, ανάλογα με την τιμή στην γραμμή ελέγχου C.

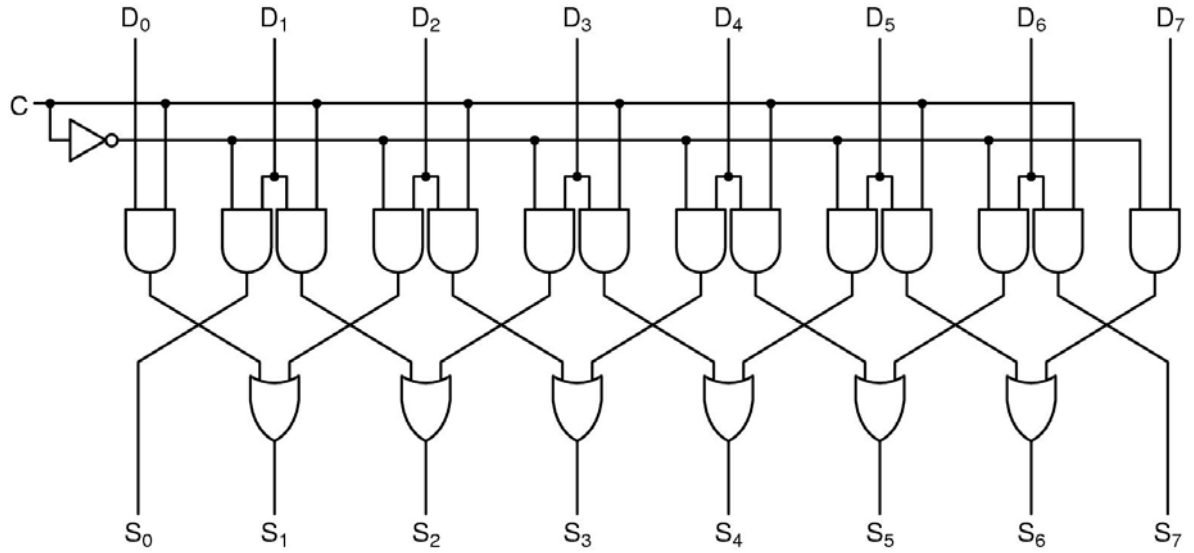
Στον ολισθητή όλα τα bit του αριθμού εκτός από τα δύο ακραία (D0 και D7) οδηγούνται σε ένα ζευγάρι πυλών AND. Σε κάθε τέτοιο ζευγάρι στην δεξιά AND συνδέεται η γραμμή C ενώ στην αριστερή AND η αντίστροφη της.

Έτσι αν η γραμμή ελέγχου έχει την τιμή 0 και προφανώς η αντίστροφή της έχει τιμή 1, «ενεργοποιούνται» οι «αριστερές» πύλες AND οδηγώντας τα bits προς τα αριστερά (shift

left). Προσέξτε ότι σε αυτή την περίπτωση το bit D0 δεν οδηγείται πουθενά και χάνεται. Προσέξτε επίσης ότι το bit S7 σχηματίζεται από την έξοδο της «δεξιάς» πύλης που αντιστοιχεί στο bit εισόδου D6, η οποία λόγω του ότι έχει στην είσοδο την γραμμή C που είναι 0, δίνει και στην έξοδό της 0, και έτσι πάντα το bit S7 παίρνει την τιμή 0 σε περίπτωση αριστερής ολίσθησης. Δηλαδή η λέξη «γεμίζει» από δεξιά με μηδενικά.

Αν αντίθετα η γραμμή ελέγχου έχει την τιμή 1 και προφανώς η αντίστροφή της έχει τιμή 0, «ενεργοποιούνται» οι «δεξιές» πύλες AND οδηγώντας τα bits προς τα δεξιά (shift right). Προσέξτε ότι σε αυτή την περίπτωση το bit D7 δεν οδηγείται πουθενά και χάνεται. Προσέξτε επίσης ότι το bit S0 σχηματίζεται από την έξοδο της «αριστερής» πύλης που αντιστοιχεί στο bit εισόδου D1, η οποία λόγω του ότι έχει στην είσοδο την γραμμή $\overline{C}$ (το αντίστροφο της C) που είναι 0, δίνει και στην έξοδό της 0, και έτσι πάντα το bit S0 παίρνει την τιμή 0 σε περίπτωση δεξιάς ολίσθησης. Δηλαδή η λέξη «γεμίζει» από αριστερά με μηδενικά.

Σημειώστε επίσης ότι τα παραπάνω ισχύουν για τον ολισθητή του σχήματος στον οποίο τα bit της λέξης εμφανίζονται με ανάποδη σειρά από την συνηθισμένη.



Ολισθητής λέξης των 8 bit ε μία γραμμή ελέγχου για ολίσθηση δεξιά ή αριστερά

Οι ολισθητές χρησιμοποιούνται στις Αριθμητικές και Λογικές Μονάδες (ALUs) για να υλοποιούν τις εντολές ολίσθησης και περιστροφής του σετ εντολών γλώσσας μηχανής του επεξεργαστή (π.χ. SHL, SHR, SAR, ROL, ROR, RCL, RCR), αλλά όπως προαναφέρθηκε μπορεί να χρησιμοποιηθούν και για πολλαπλασιασμό ή διαίρεση με το 2 ή με δυνάμεις του 2. Επίσης μπορούν να χρησιμοποιηθούν για μετατροπή λέξεων από παράλληλες σε σειριακές. Αυτό μπορεί να γίνει ως εξής:

1. Η λέξη εμφανίζεται ως είσοδος στον ολισθητή.
2. Εκτελούνται διαδοχικές π.χ. ολισθήσεις δεξιά ,
3. Λαμβάνονται τα bit της λέξης με σειριακό τρόπο (ένα-ένα) από το bit S7

Τα βήματα 2 και 3 μπορούν ισοδύναμα να γίνουν :

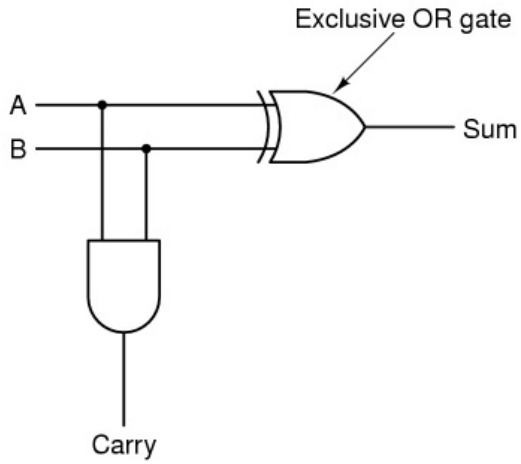
2. Εκτελούνται διαδοχικές ολισθήσεις αριστερά,
3. Λαμβάνονται τα bit της λέξης με σειριακό τρόπο (ένα-ένα) από το bit S0

Αυτό που διαφέρει στις δύο εναλλακτικές υλοποιήσεις είναι η σειρά μετατροπής των bit. Στην πρώτη περίπτωση τα bits λαμβάνονται σειριακά με την ακολουθία D7→ D0, ενώ στην δεύτερη περίπτωση με την ακολουθία D0→ D7.

3.5.2. Αθροιστές (Adders)

Οι Αθροιστές (Adders) είναι κυκλώματα που χρησιμοποιούνται για την πρόσθεση ακεραίων. Η απλούστερη μορφή αθροιστή είναι ο Ημιαθροιστής (Half-Adder). Στην επόμενη εικόνα φαίνεται ένας ημιαθροιστής ο οποίος προσθέτει 2 αριθμούς του 1 bit. Στην έξοδο εμφανίζεται το άθροισμα των 2 εισόδων και το κρατούμενο. Ο ημιαθροιστής είναι κατάλληλος για την πρόσθεση των bit χαμηλής τάξης δύο λέξεων εισόδου με πολλά bit αλλά είναι ακατάλληλος για πρόσθεση bit στο μέσον των λέξεων, καθώς δεν υπολογίζει στην πράξη το ενδεχόμενο κρατούμενο που προέρχεται από την προηγούμενη πρόσθεση χαμηλότερης τάξης bit.

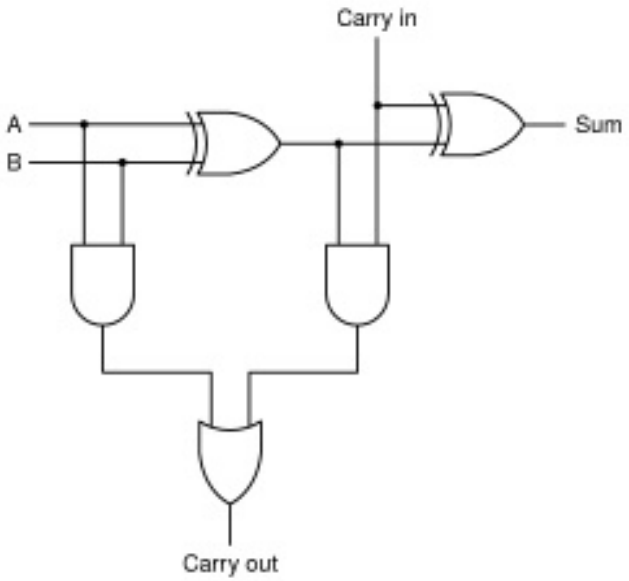
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Ημιαθροιστής 2 λέξεων του ενός bit και ο πίνακας αληθείας του

Συνδυάζοντας 2 ημιαθροιστές μπορούμε να υλοποιήσουμε τον Πλήρη Αθροιστή (Full Adder), ο οποίος δέχεται σαν είσοδο εκτός από τα 2 bit που πρόκειται να προστεθούν και το κρατούμενο που προέκυψε από προηγούμενη πράξη. Ένα τέτοιο κύκλωμα φαίνεται στην επόμενη εικόνα.

A	B	Carry in	Sum	Carry out
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

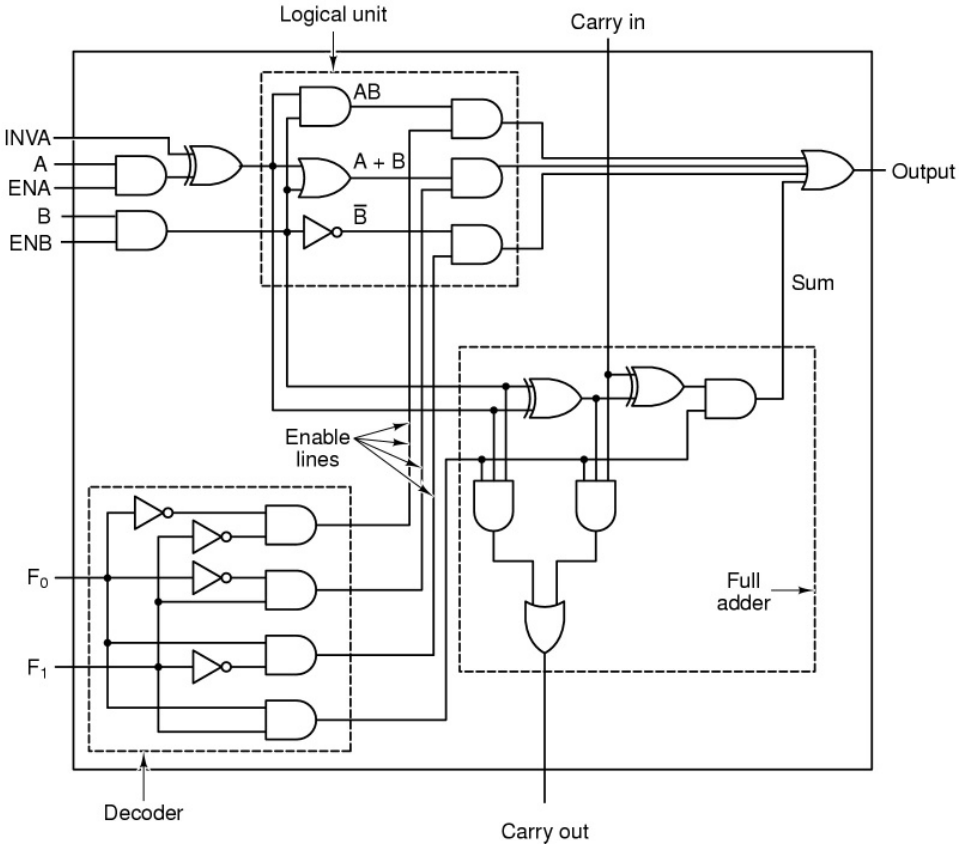


Πλήρης Αθροιστής 2 λέξεων του ενός bit και ο πίνακας αληθείας του

Για την πρόσθεση δύο λέξεων των 16, 32 ή 64 bit, χρησιμοποιούμε πολλούς πλήρεις αθροιστές παράλληλα, με τρόπο ώστε η έξοδος και τα κρατούμενο του προηγούμενου να αποτελεί είσοδο του επόμενου. Ανάλογα με τον τρόπο που συνδέονται οι επιμέρους πλήρεις αθροιστές προκύπτει είτε ο Αθροιστής Κυματοειδούς Διάδοσης Κρατουμένου (Ripple Carry Adder) είτε ο ταχύτερος Αθροιστής Επιλογής Κρατουμένου (Carry Select Adder).

3.5.3. Αριθμητικές και Λογικές Μονάδες (Arithmetic and Logic Units – ALUs)

Οι Αριθμητικές & Λογικές Μονάδες (Arithmentic and Logic Unit, ALU) είναι υπεύθυνες για την εκτέλεση αριθμητικών πράξεων σε ακεραίους, καθώς και λογικών πράξεων (AND, OR, XOR, …) και συγκρίσεων, όπως και εκτέλεση πράξεων ολίσθησης και περιστροφής. Περιέχουν τα κατάλληλα ψηφιακά κυκλώματα για την εκτέλεση των πράξεων αυτών, δύο εισόδους δεδομένων A και B για την προσκόμιση των αριθμών που θα συμμετέχουν στην πράξη, καθώς και μία είσοδο εντολής στην οποία η CPU τοποθετεί ένα κατάλληλο κωδικό αριθμό για να καθορίσει το τι πράξη θα γίνει. Στην απλούστερη περίπτωση, όπως αυτή του παρακάτω σχήματος, μπορούν να περιέχουν ένα μόνο σύνθετο κύκλωμα που εκτελεί τις πράξεις AND, OR, NOT και την αριθμητική πρόσθεση, μεταξύ 2 bit εισόδου. Για πράξεις σε λέξεις των n bits συνδέονται n τέτοια κυκλώματα.
Στην επόμενη εικόνα φαίνεται ένα τέτοιο κύκλωμα.



Αριθμητική και Λογική Μονάδα του 1 bit με πράξεις AND, OR, NOT και πρόσθεση.

Τα bit εισόδου εφαρμόζονται στις εισόδους A και B. Στην κάτω αριστερή γωνία υπάρχει ένας αποκωδικοποιητής δύο εισόδων (F0 & F1) ο οποίος επιλέγει μία από τις 4 ανωτέρω πράξεις. Δηλαδή στις εισόδους F0, F1 εφαρμόζεται ένας διψήφιος δυαδικός αριθμός από 00 έως και 11 ο οποίος καθορίζει το ποια πράξη θα εκτελεστεί. Επομένως ο αριθμός αυτός

(F0,F1) είναι στην ουσία ο κωδικός εντολής που δέχεται η ALU. Ο αποκωδικοποιητής (decoder) στη συνέχεια ενεργοποιεί (φέρνει σε λογικό 1) μία από τις 4 εξόδους του, ανάλογα με τον συνδυασμό των εισόδων. Η αντιστοιχία των αριθμών με τις πράξεις στο συγκεκριμένο παράδειγμα ALU είναι η ακόλουθη:

Αριθμός κωδικού εντολής της ALU		**Αντίστοιχη Πράξη που θα εκτελεστεί**
F0	**F1**	
0	0	A AND B
0	1	A OR B
1	0	NOT B
1	1	A + B + Carry_In (πρόσθεση)

Στην επάνω αριστερή γωνία υπάρχει η Λογική μονάδα η οποία εκτελεί τις πράξεις AND, OR και NOT. Στην κάτω δεξιά γωνία υπάρχει ένας πλήρης αθροιστής ο οποίος πραγματοποιεί την αριθμητική πρόσθεση. Όπως μπορούμε να δούμε, και οι 4 πράξεις πραγματοποιούνται μέσα στο κύκλωμα ταυτόχρονα. Ωστόσο, το ποια θα εμφανιστεί στην έξοδο εξαρτάται από το ποια από τις γραμμές εξόδου του αποκωδικοποιητή έχει τιμή 1. Αυτό υλοποιείται οδηγώντας την έξοδο της κάθε πράξης σε μια πύλη AND μαζί με την αντίστοιχη γραμμή εξόδου του αποκωδικοποιητή. Επειδή για κάθε εντολή που δέχεται η ALU μόνο μια έξοδος του αποκωδικοποιητή ενεργοποιείται (έρχεται σε λογικό 1), ενώ όλες οι υπόλοιπες παραμένουν σε λογικό 0, από τις τέσσερις πύλες AND οι τρεις θα είναι απενεργοποιημένες (θα βγάζουν στην έξοδο λογικό 0) καθώς στην είσοδό τους θα λαμβάνουν ένα τουλάχιστον μηδενικό, από την έξοδο του αποκωδικοποιητή. Μία μόνο πύλη θα λαμβάνει λογικό ένα από την έξοδο του αποκωδικοποιητή, και έτσι το αποτέλεσμα που θα βγάζει στην έξοδό της θα εξαρτάται αποκλειστικά από το αποτέλεσμα της αντίστοιχης πράξης.
Τέλος, τα σήματα ελέγχου ENA και ENB μπορούν να ενεργοποιήσουν ή να απενεργοποιήσουν τις εισόδους A και B αντίστοιχα ενώ το INVA όταν έχει τιμή 1 δίνει στον κύκλωμα σαν είσοδο το A αντεστραμμένο.
Η συγκεντρωτική πύλη OR στην έξοδο του κυκλώματος κάνει την εξής δουλειά: Αν εμφανιστεί μία μονάδα από οποιαδήποτε πύλη AND τότε αυτή οδηγείται στην έξοδο. Αν όλες οι πύλες AND δίνουν 0 (ακόμα και η ενεργοποιημένη) τότε η έξοδος θα είναι 0.
Κυκλώματα όπως αυτό που περιγράφηκε κυκλοφορούν στο εμπόριο και ονομάζονται Δυαδικές Μονάδες (Bit Slices). Συνδέοντας παράλληλα πολλές τέτοιες μονάδες μπορούν να εκτελεστούν πράξεις σε λέξεις οποιουδήποτε εύρους σε bit.

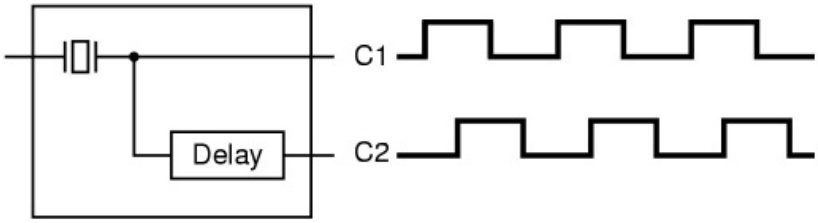
3.6. Κυκλώματα Ρολογιού (Clocks)

Σε κάθε υπολογιστή, η σειρά με την οποία πραγματοποιούνται οι διάφορες ενέργειες είναι κρίσιμη, ενώ κάποιες ενέργειες πρέπει να πραγματοποιούνται πριν από κάποιες άλλες ή ταυτόχρονα με αυτές. Για την επίτευξη του απαιτούμενου συγχρονισμού χρησιμοποιούνται κυκλώματα ταλαντωτών, τα οποία ονομάζονται απλά Ρολόγια (clocks) και παράγουν τετραγωνικούς παλμούς με ακριβές πλάτος και συχνότητα. Για την επίτευξη της απαιτούμενης υψηλής ακρίβειας στη συχνότητα, τα ρολόγια χρησιμοποιούν συνήθως κρυστάλλους χαλαζία (quartz), ενώ οι συχνότητες συνεχώς αυξάνουν σε τιμή, έχοντας ξεπεράσει τα 3 GHz. Το χρονικό διάστημα μεταξύ των αντίστοιχων ακμών δύο διαδοχικών παλμών του ρολογιού λέγεται Χρόνος Κύκλου Ρολογιού (Clock Cycle Time) και είναι της τάξης των nsec (10^{-9} sec).

Συνήθως, ένα μόνο σήμα από το ρολόι δεν επαρκεί για τον συγχρονισμό όλων των ενεργειών που πρέπει να πραγματοποιηθούν σε έναν κύκλο ρολογιού. Γι' αυτό, ο κύκλος ρολογιού υποδιαιρείται σε επιμέρους τμήματα. Ένας κοινός τρόπος για να γίνει αυτό είναι η υποκλοπή του σήματος της πρωτεύουσας γραμμής του ρολογιού και η εφαρμογή της σε ένα κύκλωμα γνωστής καθυστέρησης. Έτσι, παράγεται ένα δευτερεύον σήμα ρολογιού που βρίσκεται σε διαφορά φάσης με το πρωτεύον. Αν χρειάζονται επιπλέον σήματα, ακολουθείται η ίδια μέθοδος με πρόσθετα κυκλώματα καθυστέρησης.
Στην επόμενη εικόνα φαίνεται ένα ρολόι με 2 εξόδους. Στην συγκεκριμένη περίπτωση υπάρχουν 4 χρονικές αναφορές για διακριτά συμβάντα :

- Ακμή ανόδου σήματος C1
- Ακμή καθόδου σήματος C1
- Ακμή ανόδου σήματος C2
- Ακμή καθόδου σήματος C2

Με τη σύνδεση διαφορετικών συμβάντων στις διάφορες ακμές του παλμού μπορεί να επιτευχθεί η απαιτούμενη χρονική ακολουθία.

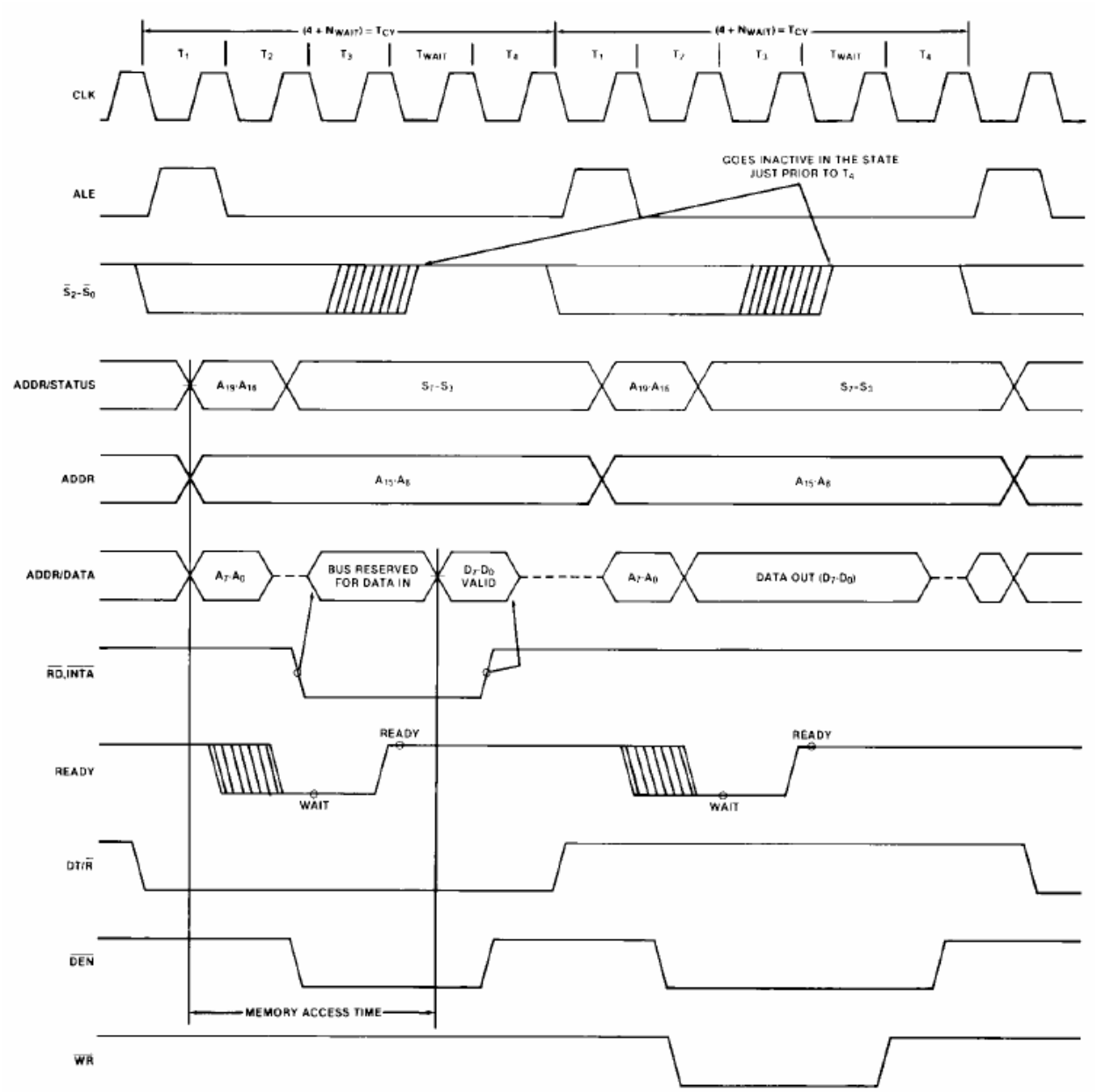


Κύκλωμα Ρολογιού και παραγόμενες παλμοσειρές

Ορισμένα συμβάντα δεν σχετίζονται με διακριτές χρονικές στιγμές αλλά με χρονικά διαστήματα. Κάποιο συμβάν, για παράδειγμα, μπορεί να συμβεί μόνον όταν το σήμα C1 είναι στην υψηλή κατάσταση και όχι ακριβώς στην ακμή ανόδου του. Στην προηγούμενη περίπτωση, τα χρονικά διαστήματα που δημιουργούνται από την επικάλυψη των C1 και C2 είναι :

1. C1=1, C2=1
2. C1=0, C2=0
3. C1=1, C2=0
4. C1=0, C2=1

Στην επόμενη εικόνα (πηγή Intel) φαίνεται ο χρονισμός των σημάτων του 8088. Βασική παλμοσειρά αναφοράς είναι αυτή που παράγει το ρολόι του συστήματος (CLK) που για τον 8088 λειτουργεί σε συχνότητα 4,77 MHz.
Κάθε διαδικασία μεταφοράς δεδομένων δια μέσω των διαύλων του επεξεργαστή (CPU bus cycle) αποτελείται από τουλάχιστον 4 κύκλους του ρολογιού CLK. Αυτοί φαίνονται στην εικόνα ως T1, T2, T3 και T4. Η διεύθυνση αποστέλλεται από τον επεξεργαστή προς την μνήμη (μέσω του MAR) κατά τον κύκλο T1. Τα δεδομένα μεταφέρονται μέσω του διαύλου δεδομένων κατά την διάρκεια των T3 και T4. Ο κύκλος T2 χρησιμοποιείται κυρίως για την αλλαγή της κατεύθυνσης της ροής των δεδομένων στον δίαυλο δεδομένων (read/write).



Βασικός χρονισμός των σημάτων του 8088

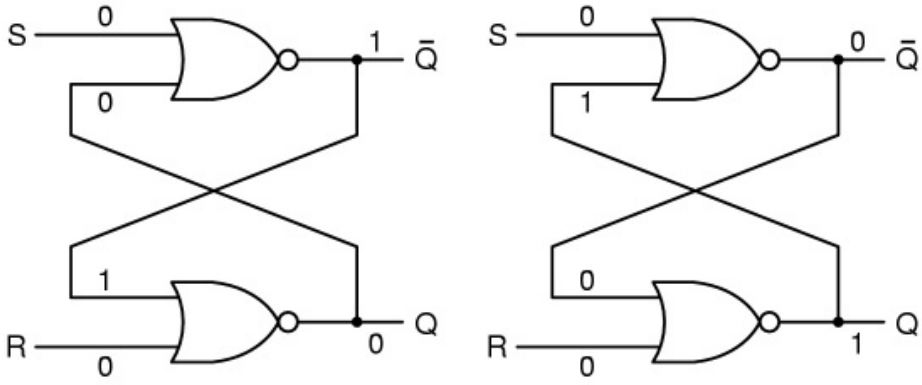
3.7. Κυκλώματα Μνήμης (Memory Circuits)

Ένα βασικό συστατικό κάθε υπολογιστή είναι η μνήμη, η οποία χρησιμοποιείται για την αποθήκευση δεδομένων και εντολών. Βέβαια σε έναν Η/Υ υπάρχουν πολλά συστήματα μνήμης όπως οι μνήμες μαγνητικής αποθήκευσης (σκληροί δίσκοι) ή οπτικής αποθήκευσης (CD, DVD, κ.λ.π.). Εδώ αναφερόμαστε σε ηλεκτρονικές μνήμες (solid state) RAM που υλοποιούνται με πύλες. Στις επόμενες παραγράφους θα εξετάσουμε τα βασικά συστατικά ενός συστήματος μνήμης, σε επίπεδο πυλών.

3.7.1. Κυκλώματα Μανδάλωσης (Latch Circuits)

Για να δημιουργηθεί μια μνήμη του 1 bit χρειάζεται ένα κύκλωμα που να "θυμάται" με κάποιον τρόπο τις προηγούμενες τιμές εισόδου. Τέτοια κυκλώματα μπορούν να κατασκευαστούν από δύο πύλες NAND ή NOR και ανατροφοδότηση και ονομάζονται Κυκλώματα Μανδάλωσης (Latch Circuits). Σε αντίθεση με τα συνδυαστικά κυκλώματα, οι έξοδοι ενός τέτοιου κυκλώματος δεν καθορίζονται μονοσήμαντα από τις τρέχουσες εισόδους. Διακρίνουμε τις παρακάτω κατηγορίες κυκλωμάτων μανδάλωσης :

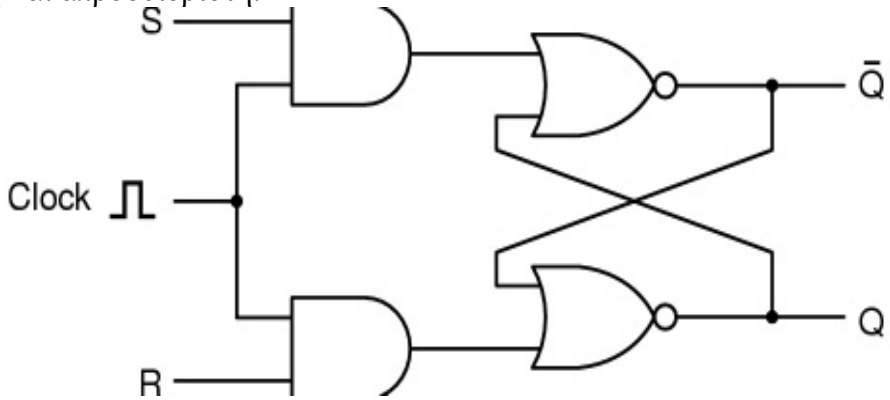
• Κύκλωμα Μανδάλωσης SR (SR Latch). Ένα τέτοιο κύκλωμα έχει 2 εισόδους (S,R) και 2 συμπληρωματικές εξόδους (Q, $\overline{Q}$). Στο κύκλωμα αυτό, όταν δίνεται στιγμιαία η τιμή 1 στο S το κύκλωμα καταλήγει στην κατάσταση Q=1, ενώ για S=0 καταλήγει στην κατάσταση Q=0, ανεξάρτητα από την κατάσταση που βρισκόταν προηγουμένως. Το κύκλωμα από δω και πέρα παραμένει στην κατάσταση αυτή, ανεξάρτητα από τις τιμές της εισόδου S. Για να αλλάξει κατάσταση πρέπει να δοθεί στιγμιαία στο R η τιμή 1, οπότε το κύκλωμα τίθεται στην κατάσταση Q=0. Η είσοδος R, επομένως, εκτελεί reset στο κύκλωμα και το επαναφέρει στην κατάσταση Q=0. Το κύκλωμα δηλαδή "θυμάται" και συγκρατεί την τιμή της εισόδου S μέχρι να γίνει reset, ανεξάρτητα από τις τιμές στην είσοδο. Εκμεταλλευόμενοι αυτή την ιδιότητα μπορούμε να κατασκευάζουμε μνήμες υπολογιστών.



A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

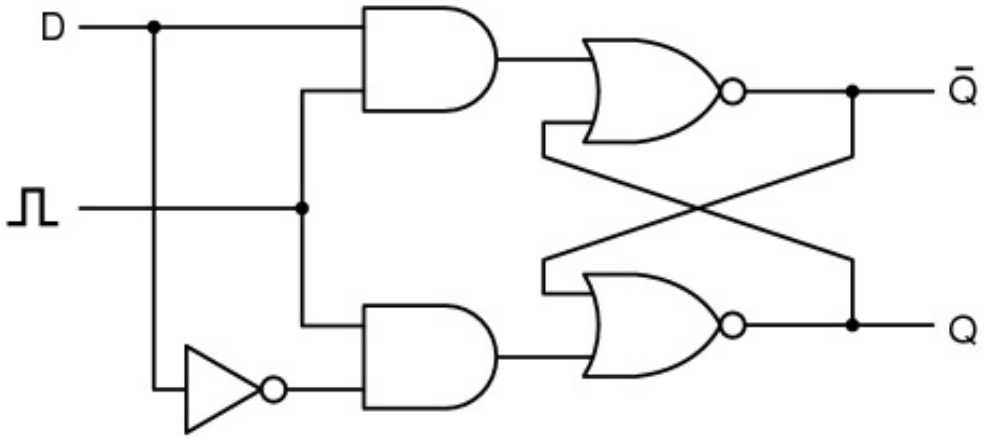
Κύκλωμα μανδαλωσης SR

• Χρονισμένο Κύκλωμα Μανδάλωσης SR (Clocked SR Latch). Στο κύκλωμα αυτό έχει προστεθεί άλλη μια είσοδος που συνήθως είναι το σήμα του ρολογιού, με σκοπό να καθορίζεται από το σήμα αυτό το χρονικό διάστημα κατά το οποίο το κύκλωμα μανδάλωσης αντιλαμβάνεται τις μεταβολές στα S και R. Έτσι, όταν το σήμα του ρολογιού παίρνει τιμή 1 το κύκλωμα εκτελεί τη λειτουργία μνήμης που είδαμε πιο πάνω, ενώ όταν είναι 0 το κύκλωμα είναι αδιάφορο στις τιμές στις εισόδους S,R και η κατάσταση του παραμένει αμετάβλητη και απροσδιόριστη.



Χρονισμένο Κύκλωμα Μανδάλωσης SR

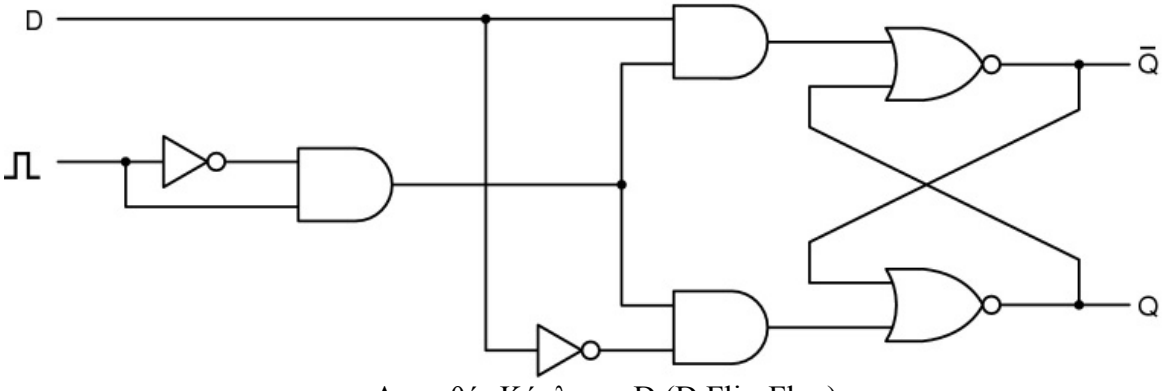
• Χρονισμένο Κύκλωμα Μανδάλωσης D (Clocked D Latch). Το κύκλωμα αυτό αποτελεί βελτίωση του προηγουμένου και διορθώνει την ασαφή συμπεριφορά του όταν S=R=1. Εχει 2 εισόδους, την D, που περιέχει το προς αποθήκευση bit, και το σήμα του ρολογιού. Όταν το ρολόϊ έχει τιμή 1, διαβάζεται και αποθηκεύεται η τιμή της εισόδου D, η οποία είναι διαθέσιμη στην έξοδο Q. Το κύκλωμα αυτό χρησιμοποιείται κυρίως στην κατασκευή μνημών για υπολογιστές.



Χρονισμένο Κύκλωμα Μανδάλωσης D

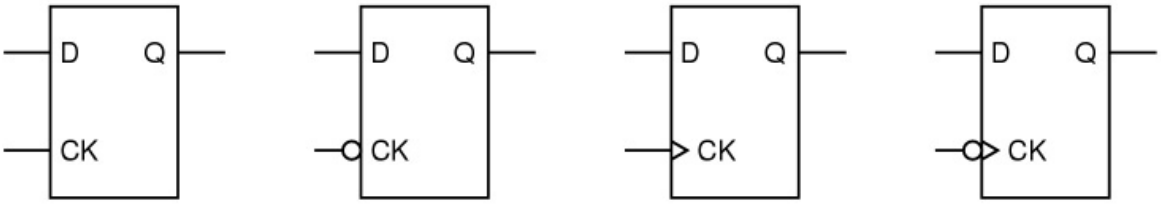
3.7.2. Δισταθή Κυκλώματα (FLIP-FLOP)

Σε πολλά κυκλώματα είναι απαραίτητο να γίνεται δειγματοληψία της τιμής κάποιας γραμμής σε μια συγκεκριμένη χρονική στιγμή και να αποθηκεύεται η τιμή αυτή. Σε αυτό το είδος του κυκλώματος, που λέγεται Δισταθές Κύκλωμα ή flip-flop, η αλλαγή κατάστασης δεν γίνεται όταν το ρολόι έχει τιμή 1 αλλά κατά τη μετάβαση του ρολογιού από το 0 στο 1 (ακμή ανόδου) και από το 1 στο 0 (ακμή καθόδου). Έτσι το μήκος του παλμού του ρολογιού δεν έχει σημασία, εφ' όσον η αλλαγή κατάστασης γίνεται γρήγορα.
Συνεπώς, ενώ τα κυκλώματα μανδάλωσης είναι ενεργοποιούμενα με επίπεδο (level triggered), τα δισταθή κυκλώματα είναι ενεργοποιούμενα με ακμή (edge triggered). Η διαφορά αυτή μεταξύ τους επιβάλλει διαφορετικό λογικό σχεδιασμό αλλά η λειτουργία που επιτελούν είναι η ίδια. Στην επόμενη εικόνα φαίνεται ένα D Flip-Flop.



Δισταθές Κύκλωμα D (D Flip-Flop)

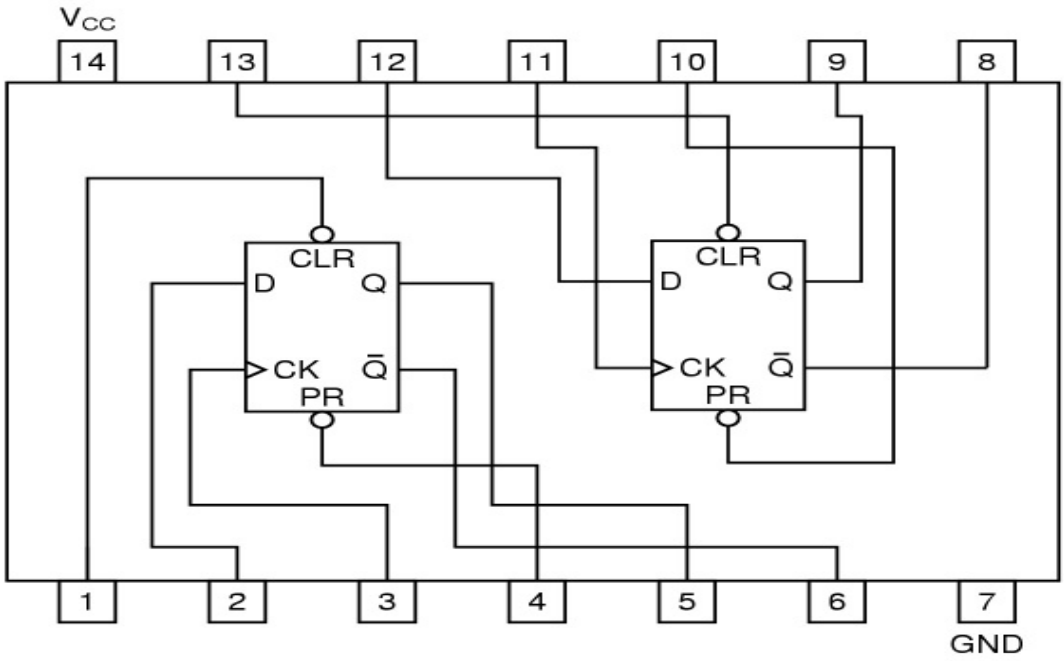
Και στην επόμενη εικόνα φαίνονται τα καθιερωμένα σύμβολα για τα κυκλώματα μανδάλωσης και τα δισταθή κυκλώματα.



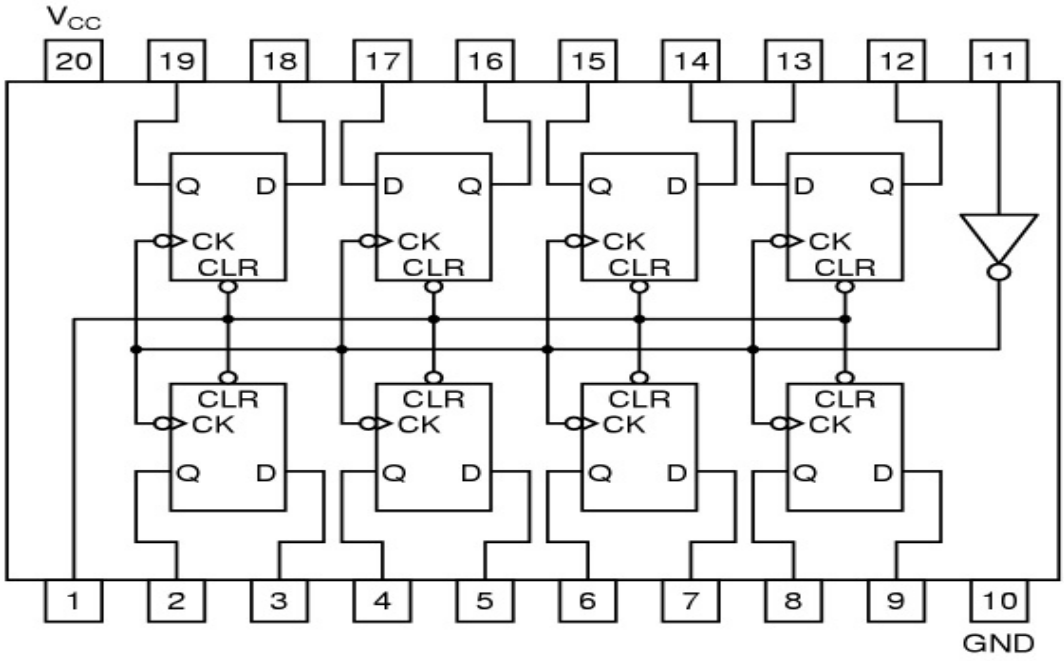
(a) (b) (c) (d)

Τα σύμβολα για (a), (b) κυκλώματα μανδάλωσης D και (c),(d) δισταθή κυκλώματα D

Τα δισταθή κυκλώματα μπορούν να λειτουργούν είτε ανεξάρτητα είτε πολλά μαζί σε συνδυασμό ώστε να σχηματίζουν μονάδες μνήμης των n bit. Στην επόμενη εικόνα φαίνεται ένα chip 14 ακροδεκτών με 2 ανεξάρτητα D Flip-Flop και ένα chip των 20 ακροδεκτών με 8 συνδεδεμένα D Flip-Flop που σχηματίζουν έναν καταχωρητή (register) των 8 bit. Συνδέοντας 2 τέτοια chip παράλληλα μπορούμε να δημιουργήσουμε έναν καταχωρητή των 16 bit κ.ο.κ.



(a)



(b)

Ολοκληρωμένο κύκλωμα (a) με δύο ανεξάρτητα δισταθή κυκλώματα D και (b) καταχωρτής 8 bit με συνδυασμό 8 δισταθών κυκλωμάτων D

Κεφάλαιο 4. Μικροεπεξεργαστές

4.1. Γενικά για τους Μικροεπεξεργαστές

Οι Μικροεπεξεργαστές (Microprocessors) είναι κυκλώματα LSI/VLSI που περιέχουν μεγάλο αριθμό ψηφιακών κυκλωμάτων ομαδοποιημένα σε υπομονάδες.
Εκτελούν βασικές αριθμητικές και λογικές λειτουργίες καθώς και λειτουργίες ελέγχου και μεταφορά δεδομένων από/προς τη μνήμη και τις περιφερειακές συσκευές.
Καθοδηγούνται από σειρές εντολών γλώσσας μηχανής (πρόγραμμα) που συντάσσονται με βάση ένα πεπερασμένο σετ εντολών, ειδικό για κάθε μικροεπεξεργαστή.
Χαρακτηρίζονται από το εύρος του διαύλου δεδομένων (μήκος λέξης) που ταυτίζεται με το μέγεθος των βασικών καταχωρητών και είναι 4, 8, 16, 32, 64, 128 bit, …
Χαρακτηρίζονται από την μέγιστη συχνότητα λειτουργίας τους (100KHz .. 10GHz) που καθορίζεται από ένα κύκλωμα χρονισμού (clock).
Στη συσκευασία τους έχουν μεγάλο αριθμό pin (16...478) για σύνδεση με την τροφοδοσία, τους διαύλους, τις γραμμές ελέγχου και διακοπών και τις υπόλοιπες υπομονάδες ενός υπολογιστικού συστήματος.

Οι μικροεπεξεργαστές (Μ/Ε) εκτοπίζουν τα παραδοσιακά ηλεκτρονικά από σχεδόν κάθε πεδίο που περιλαμβάνει προγραμματισμό ή αυτόματο έλεγχο. Έτσι, εκτός από την κατασκευή Η/Υ χρησιμοποιούνται σε οικιακές συσκευές, συσκευές γραφείου, ηλεκτρονικά παιγνίδια, στην αυτοκινητοβιομηχανία κ.α. Τα πλεονεκτήματα που παρουσιάζουν οι Μ/Ε έναντι άλλων λύσεων είναι :

- ✓ Λιγότερα εξαρτήματα, γεγονός που σημαίνει μικρότερος όγκος, μειωμένη κατανάλωση ενέργειας και μεγαλύτερη αξιοπιστία
- ✓ Μικρότερο κόστος
- ✓ Δυνατότητα προγραμματισμού, γεγονός που σημαίνει απλοποίηση του σχεδιασμού, μικρότερος χρόνος ανάπτυξης και δυνατότητα μετατροπών

Η ιστορία των Μ/Ε ξεκινά με τον 4-bit Μ/Ε 4004 της Intel το 1971. Το chip αποτελούνταν από 2.300 τρανζίστορ και εκτελούσε περίπου 60.000 πράξεις/sec. Προορίζονταν για την υλοποίηση υπολογιστών τσέπης (calculator) αλλά αποτέλεσε μεγάλη εμπορική επιτυχία και άνοιξε το δρόμο για την δημιουργία των μικροϋπολογιστών. Η Intel εξέλιξε τη σειρά των Μ/Ε της και κυριαρχεί στο χώρο αυτό μέχρι σήμερα με τη σειρά 80X86 και αργότερα τη σειρά των Pentium.

Άλλη μια σειρά που γνώρισε εμπορική επιτυχία είναι η σειρά 68X00 της Motorola. Διάφορες άλλες σχεδιάσεις εμφανίστηκαν με επιτυχία για κάποιες περιόδους στην αγορά αλλά δεν συνέχισαν να εξελίσσονται, όπως ο Z-80 της Zilog, ο 6502 της Rockwell κ.α.

Για να έχει ο αναγνώστης ένα μέτρο σύγκρισης, θα πρέπει να αναφέρουμε ότι οι σύγχρονοι Μ/Ε έχουν περισσότερα από 500.000.000 (500 εκατομμύρια) τρανζίστορ και εκτελούν μερικές δεκάδες εκατομμύρια πράξεις/sec.

Στον επόμενο πίνακα παρατίθενται συγκριτικά τα στοιχεία των πρώτων ιστορικών Μ/Ε της Intel.

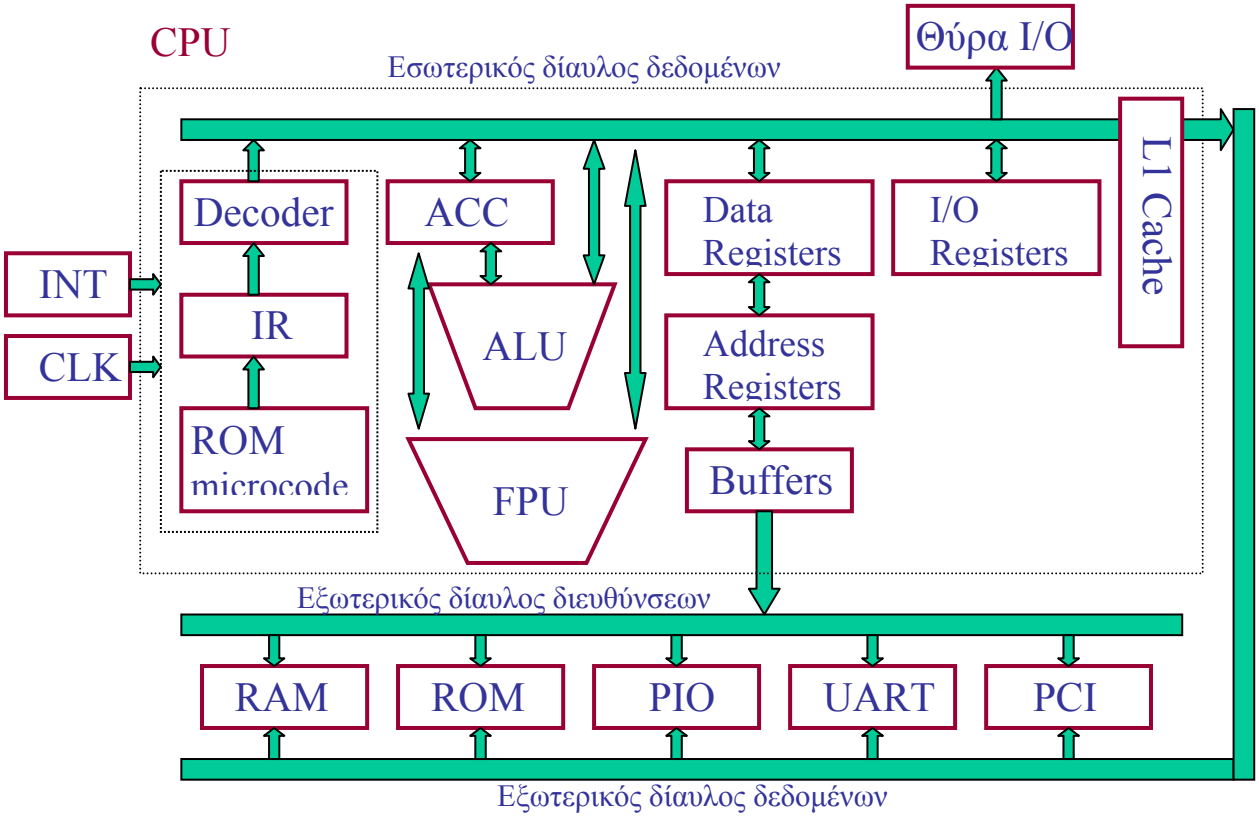
Ονομα	Ετος	MHz	Τρανζίστορ	Μνήμη	Σχόλια
4004	1971	0.108	2.300	640	Πρώτος μ/ε σε chip

8008	1972	0.108	3.500	16K	Πρώτος μ/ε 8 bit
8080	1974	2	6.000	64K	Πρώτος μ/ε γενικής χρήσης
8086	1978	5-10	29.000	1M	Πρώτος μ/ε 16 bit
8088	1979	5-8	29.000	1M	Ο μ/ε του IBM PC
80286	1982	8-12	134.000	16M	Αύξηση χώρου διευθύνσεων σε 16M και προστασία μνήμης
80386	1985	16-33	275.000	4GB	Πρώτος μ/ε 32 bit
80486	1989	25-100	1.2 M	4GB	Ενσωματ. Cache 8K
Pentium	1993	60-233	3.1 M	4GB	Δύο γραμμές διοχέτευσης
Pentium Pro	1995	150-200	5.5 M	4GB	Δύο επίπεδα ενσωματωμένης μνήμης cache
Pentium II	1997	233-400	7.5 M	4GB	Pentium Pro με MMX

Στον επόμενο πίνακα παρατίθενται συγκριτικά τα στοιχεία των πρώτων ιστορικών Μ/Ε της Motorolla.

Ονομα	**Ετος**	**Πλάτος Καταχωρητών σε bit**	**Πλάτος Διαύλου Δεδομένων σε bit**	**Χώρος Διευθύνσεων**	**Σχόλια**
68000	1979	32	4	16M	Πρώτο μέλος της οικογένειας
68008	1982	32	8	4M	μ/ε με δίαυλο 8 bit
68010	1983	32	16	16M	Υποστήριξη εικονικής μνήμης
68012	1983	32	16	2G	Βελτιωμένος 68010
68020	1984	32	32	4G	Πραγματικός μ/ε 32 bit
68030	1987	32	32	4G	On-Chip διαχείριση μνήμης
68040	1989	32	32	4G	Ταχύτερος 68030

4.2. Εσωτερική δομή μικροεπεξεργαστων



Μπλόκ διάγραμμα του εσωτερικού ενός μικροεπεξεργαστή με τα βασικά του μέρη

Αν και οι Μ/Ε αποτελούν πολύπλοκα ψηφιακά κυκλώματα στην σχεδίαση αλλά και στην υλοποίηση τους, και κάθε Μ/Ε είναι σίγουρα διαφορετικός από τους άλλους, ωστόσο όλοι οι Μ/Ε αποτελούνται από κάποια στάνταρ μέρη, που είναι τα ακόλουθα :

- Καταχωρητές (Registers) : αποτελούν μικρές μνήμες μίας λέξης με μέγεθος ίδιο με αυτό της κλάσης του μικροεπεξεργαστή. Οι καταχωρητές χρησιμοποιούνται για την προσκόμιση δεδομένων μέσα στον επεξεργαστή από τη μνήμη και τις περιφερειακές συσκευές, για την τροφοδοσία των αριθμητικών και λογικών μονάδων με δεδομένα για την εκτέλεση πράξεων, καθώς επίσης και για την αποθήκευση ενδιάμεσων και τελικών αποτελεσμάτων. Πρακτικά σε ένα πρόγραμμα γλώσσας μηχανής οι καταχωρητές είναι για τον προγραμματιστή οι μόνιμες μεταβλητές του προγράμματος τις οποίες μπορεί να χρησιμοποιήσει.
- Αριθμητική και Λογική Μονάδα (Arithmetic and Logic Unit – ALU) : μονάδα που εκτελεί αριθμητικές και λογικές πράξεις (+, -, AND, OR, ..)
- Μονάδα Κινητής Υποδιαστολής (Floating Point Unit – FPU) : μονάδα που εκτελεί πράξεις κινητής υποδιαστολής (με δεκαδικά).
- Μονάδα Ελέγχου (Control Unit) : Η μονάδα αυτή, είναι η πιο σημαντική μονάδα μέσα σε έναν μικροεπεξεργαστή καθώς αποτελεί τον «εγκέφαλο» του συστήματος. Είναι υπεύθυνη για την ακολουθιακή εκτέλεση των βημάτων που απαιτούνται για την ολοκλήρωση των εντολών γλώσσας μηχανής, τη λήψη και αποστολή σημάτων ελέγχου στον εξωτερικό κόσμο (μητρική κάρτα – μνήμη – περιφερειακές συσκευές), και τον συντονισμό όλων των επιμέρους τμημάτων ενός μικροεπεξεργαστή.

Περιλαμβάνει τον καταχωρητή εντολής (Instruction Register – IR), τον αποκωδικοποιητή εντολής (Decoder) και ROM μικροκώδικα για την εκτέλεση των εντολών. Δέχεται είσοδο από το ρολόι και τις γραμμές διακοπών (Interrupts).

- Εσωτερικός δίαυλος δεδομένων (Internal Data Bus) : γραμμή μίας λέξης που ενώνει εσωτερικά τους καταχωρητές και τις υπομονάδες. Ενώνεται με τον εξωτερικό δίαυλο δεδομένων (External Data Bus) προς τη μνήμη και τις περιφερειακές συσκευές.
- Ενδιάμεση Μνήμη επιπέδου 1 (Level 1 Cache memory) : μικρή μνήμη που λειτουργεί ως buffer ανάμεσα στην RAM και την CPU.

4.3. Τα βασικά μέρη ενός μικροεπεξεργαστή

4.3.1. Καταχωρητές

Οι καταχωρητές είναι μνήμες μίας λέξης (data) ή εύρους διευθύνσεων (address). Υλοποιούνται με Flip-Flop μεγάλης ταχύτητας. Μεταφέρουν δεδομένα από και προς το Data Bus. Υπάρχουν δύο κατηγορίες καταχωρητών :

1. **Καταχωρητές γενικής χρήσης** : εκτελούν όλες τις δυνατές λειτουργίες (πράξεις, μεταφορά δεδομένων). Συνδέονται άμεσα με την ALU, την FPU και το Data Bus. Ενώ όλοι οι καταχωρητές γενικής χρήσης είναι ισοδύναμοικ, συνήθως ένας από τους καταχωρητές γενικής χρήσης είναι «πιο ίσος» από τους άλλους. Αυτός συνήθως ονομάζεται «Συσσωρευτής» (Accumulator), και έχει μεγαλύτερα προνόμια από τους υπόλοιπους. Αυτό σημαίνει ότι υπάρχουν πράξεις και εντολές που εκτελούνται μόνο με τον συσσωρευτή αλλά και το ότι οι εντολές που χρησιμοποιούν τον συσσωρευτή συνήθως είναι μικρότερες σε αριθμό byte. Στον επεξεργαστή 8088 τον ρόλο του συσσωρευτή έχει ο AX. Οι καταχωρητές γενικής χρήσης συνήθως χρησιμοποιούνται για όλες της εργασίες όπως : αριθμητικές πράξεις, λογικές πράξεις, συγκρίσεις, μεταφορά δεδομένων, κ.λ.π.
2. **Καταχωρητές ειδικής χρήσης :** Οι Καταχωρητές ειδικής χρήσης έχουν συγκεκριμένο ρόλο μέσα στον επεξεργαστή. Υπάρχουν όμως πολλά είδη καταχωρητών ειδικής χρήσης που είναι:
   a. **Καταχωρητές Διευθύνσεων** (Address Registers): Καταχωρούν διευθύνσεις που σχετίζονται με την εκτέλεση των εντολών. Συνδέονται με το δίαυλο διευθύνσεων για σχηματισμό διεύθυνσης. Συνδέονται και με το Data Bus για φόρτωση διευθύνσεων από την μνήμη.
      i. **Δείκτης Εντολής** (Instruction Pointer – IP) : ειδικός καταχωρητής διεύθυνσης που φυλάσσει την διεύθυνση εκτέλεσης της επόμενης εντολής. Αυξάνεται αυτόματα από τη μονάδα ελέγχου. Εξάγεται στο Address Bus για να προσκομιστεί η επόμενη εντολή.
      ii. **Καταχωρητές Δείκτη** (Index Register): χρησιμοποιούνται για προσπέλαση της μνήμης υπό τη μορφή πίνακα. Περιέχουν είτε τη διεύθυνση βάσης (προστίθεται η μετατόπιση) ή την μετατόπιση (προστίθεται η διεύθυνση βάσης).
      iii. **Καταχωρητές Τμημάτων** (Segment Registers): κρατούν διευθύνσεις τμημάτων κώδικα, δεδομένων, στοίβας και έξτρα τμήμαστος. Σχηματίζουν διευθύνσεις μαζί με τους καταχωρητές δεικτών που έχουν το offset κομμάτι της διεύθυνσης.
      iv. **Δείκτης Στοίβας** (Stack Pointer – SP) : περιέχει την διεύθυνση κορυφής της στοίβας (Stack). Η στοίβα είναι συγκεκριμένη περιοχή μνήμης με δομή LIFO που γεμίζει από πάνω προς τα κάτω. Χρησιμοποιείται για προσωρινή αποθήκευση της

κατάστασης της CPU πριν από την εκτέλεση υπορουτινών. Την διαχειριζόμαστε με εντολές PUSH και POP.

b. **Καταχωρητής Εντολής** (Instruction Register-IR): περιέχει τον κώδικα της εντολής που θα εκτελεστεί. Είναι μέρος της μονάδας ελέγχου και συνδέεται με το Data Bus.
c. **Καταχωρητής Σημαιών** (Flag Register-FG): καταχωρητής που αποτελείται από μεμονωμένα bits που λειτουργούν ως σημαίες κατάστασης (flags). Οι σημαίες διαμορφώνονται μετά από την εκτέλεση κάθε εντολής και εξαρτώνται από το αποτέλεσμα της εντολής. Ορισμένα bits λειτουργούν και ως διακόπτες που ενεργοποιούν / απενεργοποιούν λειτουργίες.
Ελέγχεται από εντολές διακλάδωσης JE, JNE, JL, JLE, JNL,JNLE...

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	OF	DF	IF	TF	SF	ZF	-	AF	-	PF	-	CY

Τα bit του καταχωρητή σημαιών

Τα bit του καταχωρητή σημαιών στον 8088 είναι τα ακόλουθα:

i. (CY) Carry – Κρατούμενο : αποθηκεύει το επιπλέον bit που μπορεί να προκύψει από αριθμητικές πράξεις (κρατούμενο – δανεικό). Μπορεί να χρησιμοποιηθεί για πράξεις αριθμών πολλαπλών λέξεων (Add with Carry – Carry Propagation). Επίσης αποθηκεύει το bit υπερχείλισης σε εντολές ολίσθησης και περιστροφής. Μπορεί να τεθεί 1 ή 0 με ειδικές εντολές : STC, CLC, CMC. Ελέγχεται από εντολές διακλάδωσης JB/JNAE, JBE/JNA, JNB, JAE, JNBE/JA.
ii. (PF) Parity – Ισοτιμία : γίνεται 1 όταν το αποτέλεσμα μίας πράξης είναι δυαδικός αριθμός που έχει ζυγό αριθμό μονάδων. Ελέγχεται από εντολές διακλάδωσης JP/JPE, JNP/JPO
iii. (AF) Auxiliary Carry–Βοηθητικό Κρατούμενο : γίνεται 1 όταν σε μία πράξη μεταφέρεται κρατούμενο από το byte χαμηλής τάξης στο byte υψηλής τάξης (low nibble carry).
iv. (ZF) Zero – Μηδενικό : γίνεται 1 όταν το αποτέλεσμα οποιασδήποτε εντολής δώσει αποτέλεσμα 0. Ελέγχεται με εντολές διακλάδωσης JE/JZ, JNE/JNZ, JLE/JNG, JNLE/JG, JBE/JNA, JNBE/JA.
v. (SF) Sign – Αρνητικό : γίνεται 1 όταν το αποτέλεσμα οποιασδήποτε πράξης δώσει αποτέλεσμα αρνητικό με την σύμβαση συμπληρώματος ως προς 2 (MSB=1). Ελέγχεται από εντολές διακλάδωσης JS, JNS, JL/JNGE, JLE/JNG, JNL/JGE, JNLE/JG
vi. (TF) Trap – Παγίδευση : Σημαία Βηματικής Εκτέλεσης, όταν είναι 1 εκτελεί τις εντολές βήμα-βήμα για debugging. Μπορεί να αλλάξει με τη βοήθεια των εντολών PUSHF και POPF ως εξής:

```
PUSHF           Σώζει τον FG στο Stack
POP AX          Τον αντιγράφει στον AX
OR AX,0100h     Κάνει «1» το 8ο bit (0100h = 0000000100000000)
PUSH AX         Σώζει την νέα τιμή στο stack
POPF            Την αντιγράφει στον FG
```

Όταν η σημαία είναι «1» τότε στην αρχή της εκτέλεσης κάθε εντολής γλώσσας μηχανής ο 8088 παράγει το Interrupt 01h. Έτσι αν εμείς έχουμε αλλάξει την διεύθυνση του Interrupt 01h και έχουμε βάλει διεύθυνση δικής μας ρουτίνας, μπορούμε πριν από κάθε εκτέλεση εντολής να παρεμβάλλεται δικός μας κώδικας για διαδικασίες debugging. Η παραπάνω τεχνική μας επιτρέπει να εκτελούμε ένα πρόγραμμα βήμα-βήμα, και ενδιάμεσα να μπορούμε να βλέπουμε π.χ. τις τιμές των καταχωρητών, της μνήμης κ.λ.π.

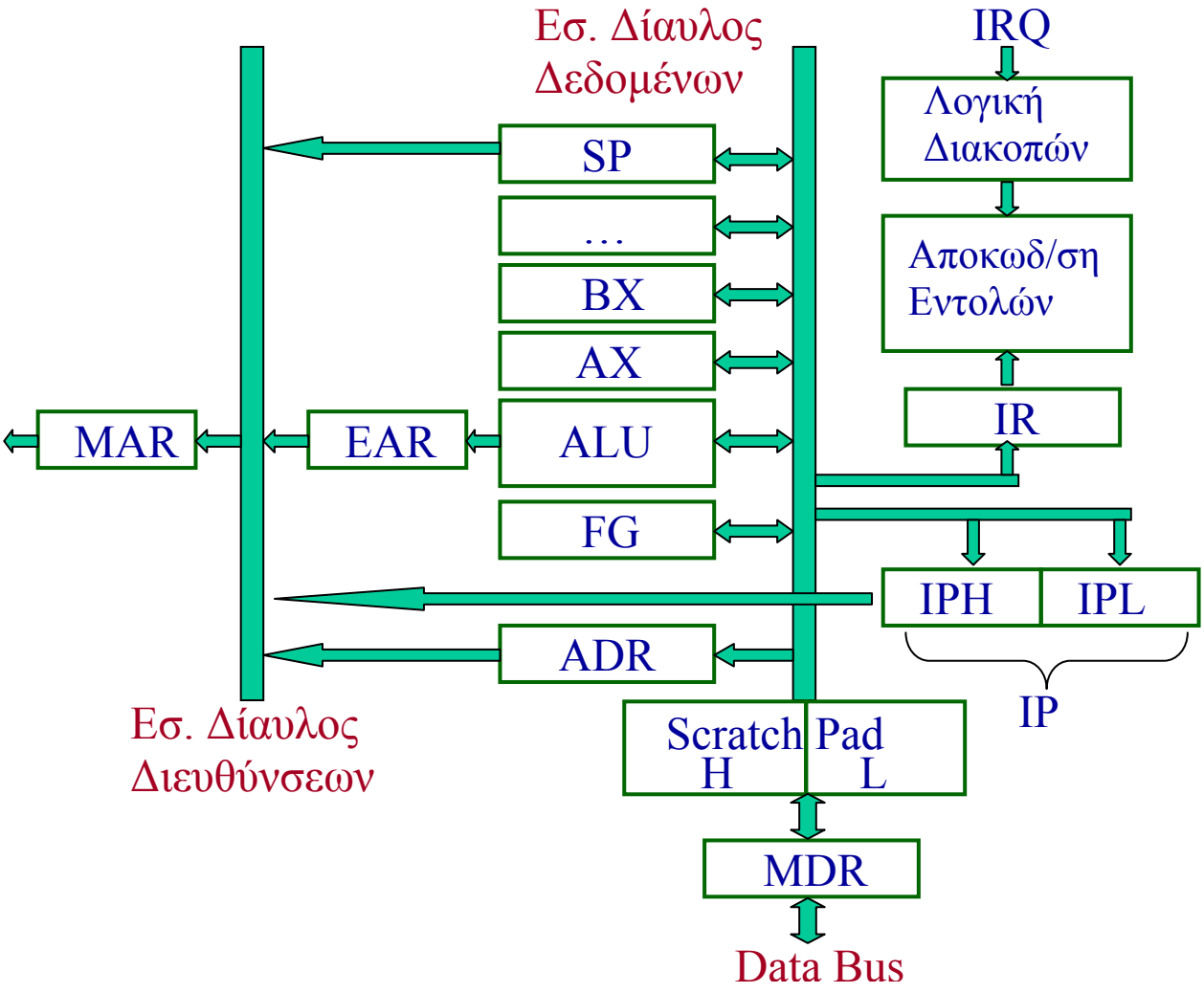
Όταν η σημαία είναι «0» τότε ΔΕΝ παράγεται το Interrupt 01h στην αρχή εκτέλεσης κάθε εντολής.

vii. (IF) Interrupt – Διακοπή : όταν είναι 1 επιτρέπει την διακοπή του προγράμματος από Interrupt (Interrupt Request) για την εκτέλεση συγκεκριμένης ρουτίνας εξυπηρέτησης της διακοπής. Όταν είναι 0 απαγορεύει τις διακοπές. Αλλάζει με εντολές CLI, STI.

viii. (DF) Direction – Κατεύθυνση : όταν είναι 1, οι εντολές των string (π.χ. LODSB, STOSB, CMPSB, κ.λ.π.) εκτελούνται από υψηλές διευθύνσεις προς χαμηλές, δηλαδή ανάποδα από το κανονικό που ισχύει για DF=0. Αλλάζει με εντολές CLD, STD.

ix. (OF) Overflow – Υπερχείλιση : Σημαία Υπερχείλισης, γίνεται 1 όταν το αποτέλεσμα μίας πράξης ξεπερνά το όριο των προσημασμένων αριθμών δηλαδή -32768…+32767. Ελέγχεται με εντολές διακλάδωσης JO, JNO, JL/JNGE, JLE/JNG, JNL/JGE, JNLE/JG.

4.3.2. Οι εσωτερικοί καταχωρητές του μικροεπεξεργαστή



Οι εσωτερικοί καταχωρητές δεν είναι προσπελάσιμοι από εντολές γλώσσας μηχανής, αλλά εξυπηρετούν την εσωτερική λειτουργία του μικροεπεξεργαστή. Οι πιο κοινοί εσωτερικοί καταχωρητές σε έναν μικροεπεξεργαστή είναι :

- **Memory Data Register** (MDR): καταχωρητής μίας λέξης που αποθηκεύει την πληροφορία που εισέρχεται στον M/E από το Data Bus. Είναι στην ουσία ο «πορτιέρης» του διαύλου δεδομένων. Οτιδήποτε εισέρχεται στον μικροεπεξεργαστή από τον δίαυλο δεδομένων αποθηκεύεται πρώτα σε αυτόν. Επίσης οτιδήποτε εξέρχεται από τον μικροεπεξεργαστή προς τον δίαυλο δεδομένων αποθηκεύεται πρώτα σε αυτόν. Αποτελεί στην ουσία buffer διασύνδεσης του M/E με τον δίαυλο δεδομένων. Στον 8088 έχει μέγεθος 8 bit, όσα και ο δίαυλος δεδομένων.
- **Memory Address Register** (MAR): καταχωρητής εύρους διεύθυνσης στον οποίο καταχωρείται η σχηματιζόμενη διεύθυνση μνήμης για να προωθηθεί στο address bus. Είναι στην ουσία ο «πορτιέρης» του διαύλου διευθύνσεων. Οποιαδήποτε διεύθυνση σχηματίζει ο M/E στον δίαυλο διευθύνσεων αποθηκεύεται πρώτα σε αυτόν. Αποτελεί στην ουσία buffer διασύνδεσης του M/E με τον δίαυλο διευθύνσεων. Στον 8088 έχει μέγεθος 20 bit όσα και ο δίαυλος διευθύνσεων.
- **Address Data Register** (ADR): καταχωρητής εύρους διεύθυνσης στον οποίο καταχωρείται μέσω του Data Bus και του MDR η διεύθυνση που είναι παράμετρος μίας εντολής (π.χ. η 0200 στην εντολή ADD AX, [0200]). Έχει μέγεθος 16 bit.
- **Effective Address Register** (EAR): καταχωρητής εύρους διεύθυνσης στον οποίο υπολογίζονται οι τελικές διευθύνσεις μνήμης με δεικτοδοτούμενες και έμμεσες διευθυνσιοδοτήσεις π.χ. στην εντολή ADD AX, [1234+SI] αποθηκεύει το αποτέλεσμα της πρόσθεσης 1234+SI που εκτελείται στην ALU. Έχει μέγεθος 16 bit.
- **Scratch Pad Register (SCR)** : καταχωρητής προχείρου με εύρος ίδιο με αυτό των υπόλοιπων καταχωρητών γενικής χρήσης (16 bit στον 8088). Χρησιμοποιείται για καταχώρηση παραμέτρων των 16 bit που δεν χωράνε εξ' ολολκήρου στον MDR. Συνήθως στους M/E υπάρχουν περισσότεροι του ενός καταχωρητές προχείρου που αποθηκεύουν ενδιάμεσα αποτελέσματα και έτσι βοηθάνε στην εκτέλεση των σύνθετων εντολών, χωρίς να χρειαστεί να χρησιμοποιηθεί η μνήμη RAM για τον σκοπό αυτό, η οποία είναι πολύ πιο αργή στην απόκριση απ' ότι οι καταχωρητές του M/E.

Στο μπλοκ διάγραμμα φαίνεται ότι οι καταχωρητές διευθύνσεων όπως είναι ο ADR ο EAR, ο SP και ο IP συνδέονται απευθείας στον καταχωρητή MAR. Βέβαια σύμφωνα με τα παραπάνω οι καταχωρητές ADR, EAR, SP και IP έχουν μέγεθος 16 bit, ενώ ο καταχωρητής MAR έχει μέγεθος 20 bits. Εδώ λοιπόν υπάρχει μια ανακολουθία. Πώς δηλαδή οι καταχωρητές μεγέθους 16 bits αποθηκεύονται στον καταχωρητή MAR των 20 bits που σχηματίζει μέσα του την τελική διεύθυνση μνήμης που θα προσπελαστεί από 0 έως 1 MB ; Η απάντηση είναι απλή αν σκεφτούμε τη μέθοδο σχηματισμού διευθύνσεων που ισχύει στον επεξεργαστή 8088 και που δεν είναι άλλη από την μέθοδο segment:offset.
Στην ουσία το περιεχόμενο των καταχωρητών ADR, EAR, SP και IP είναι το offset κομμάτι της διεύθυνσης. Αυτό πρέπει να συνδυαστεί με το κατάλληλο segment κομμάτι ώστε να μας δώσει την τελική διεύθυνση.

Όταν οι καταχωρητές ADR και EAR περιέχουν διεύθυνση δεδομένων και εξάγονται στον MAR για προσπέλαση των δεδομένων αυτών, για παράδειγμα στην εντολή:

ADD AX, [1234], όπου ο ADR θα αποθηκεύσει μέσω του MDR την διεύθυνση 1234, ή

SUB BX, [5678+SI] , όπου ο EAR θα αποθηκεύσει το αποτέλεσμα της πρόσθεσης 5678+SI,

τότε, οι καταχωρητές αυτοί (ADR ή EAR) θα συνδυαστούν με τον καταχωρητή τμήματος DS, δηλαδή η τελική διεύθυνση των 20 bit που θα προωθηθεί στον MAR θα προκύψει ως εξής:

MAR = DS : ADR = DS * 16 + ADR, ή

MAR = DS : EAR = DS * 16 + EAR

Στην περίπτωση του καταχωρητή IP τα πράγματα είναι πιο απλά. Ο καταχωρητής αυτός πάντα περιέχει την διεύθυνση της επόμενης εντολής που θα εκτελεστεί, και επομένως αφού αφορά το τμήμα κώδικα, συνδυάζεται πάντα με τον CS register, δηλαδή :

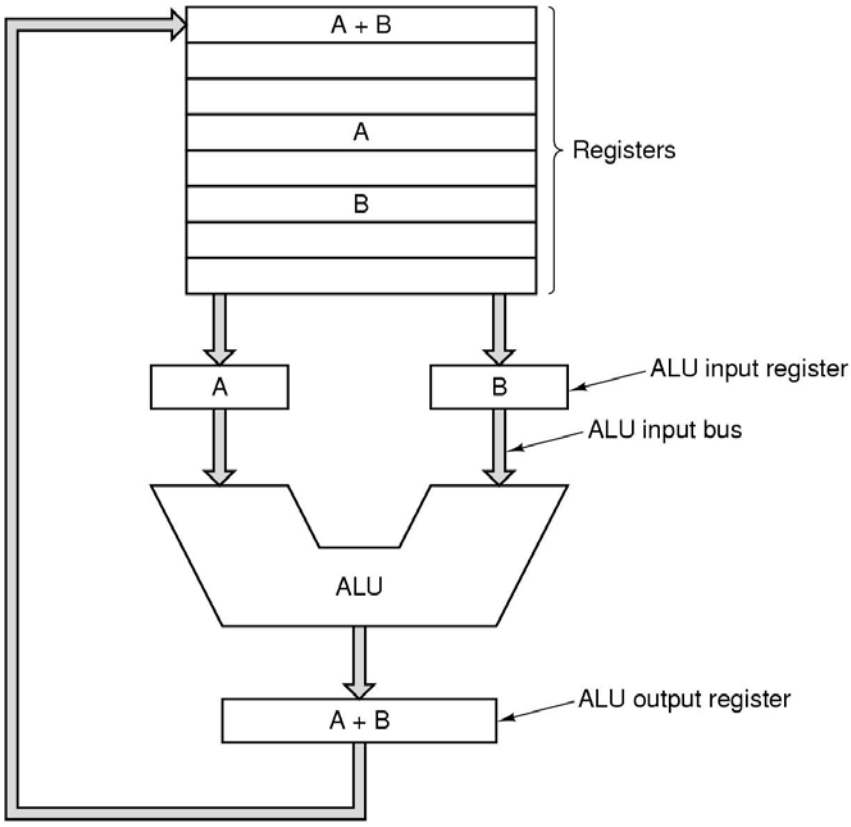
MAR = CS : IP = CS * 16 + IP

Επίσης στην περίπτωση του καταχωρητή SP, που «δείχνει» πάντα την κορυφή της στοίβας (stack), αυτός πάντα συνδυάζεται με τον καταχωρητή τμήματος SS (Stack Segment register) :

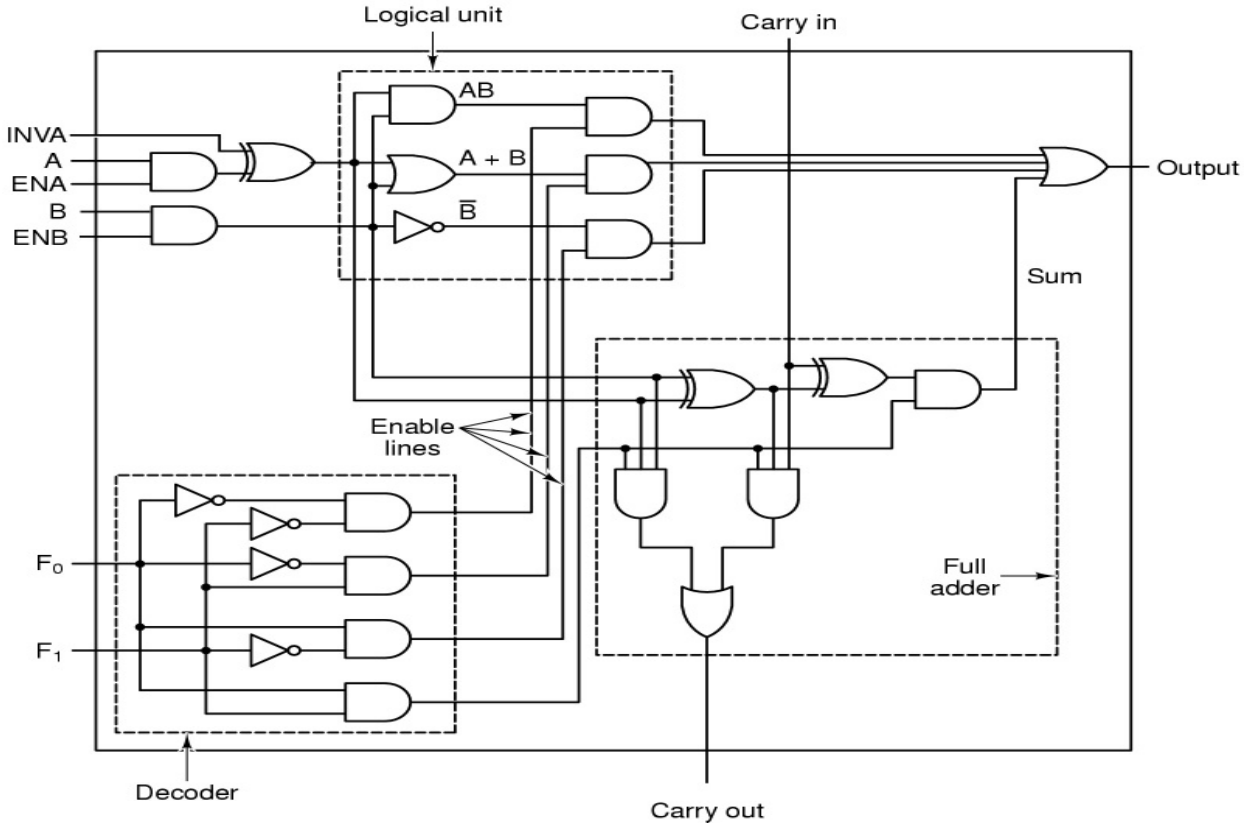
MAR = SS : SP = SS * 16 + SP

4.3.3. Η Αριθμητική και Λογική Μονάδα

Η Αριθμητική και Λογική Μονάδα (Arithmetic and Logic Unit – ALU) είναι υπεύθυνη για την εκτέλεση αριθμητικών πράξεων σε ακεραίους, καθώς και λογικών πράξεων (AND, OR, XOR, …) και συγκρίσεων, όπως και εκτέλεση πράξεων ολίσθησης και περιστροφής. Εκτελεί πρόσθεση, αφαίρεση, ενδεχομένως και πολλαπλασιασμό και διαίρεση, πράξεις της άλγεβρας Boole, όπως AND, OR, XOR, NOT, ολίσθηση (shift) ή περιστροφή (rotation) δεξιά ή αριστερά. Περιέχει τα κατάλληλα ψηφιακά κυκλώματα για την εκτέλεση των πράξεων αυτών, όπως συστοιχία πλήρων αθροιστών (full-adders), Ολισθητές (shifters), και συστοιχίες πυλών για εκτέλεση λογικών πράξεων. Επίσης περιέχει δύο εισόδους δεδομένων A και B για την προσκόμιση των αριθμών που θα συμμετέχουν στην πράξη, καθώς και μία είσοδο εντολής στην οποία η CPU τοποθετεί ένα κατάλληλο κωδικό αριθμό για να καθορίσει το τι πράξη θα γίνει. Στο παρακάτω σχήμα φαίνεται χονδρικά η δομή μίας ALU με τις δύο εισόδους δεδομένων A και B και την έξοδο του αποτελέσματος.



Στην απλούστερη περίπτωση, όπως αυτή του παρακάτω σχήματος, μπορούν να περιέχουν ένα μόνο σύνθετο κύκλωμα που εκτελεί τις πράξεις AND, OR, NOT και την αριθμητική πρόσθεση, μεταξύ 2 bit εισόδου. Για πράξεις σε λέξεις των n bits συνδέονται n τέτοια κυκλώματα.



Παράδειγμα απλής ALU που μπορεί να κάνει 4 πράξεις σε δεδομένα του 1 bit.

Από το σετ εντολών ου 8088 ένα μεγάλο μέρος αυτών, οι Εντολές Αριθμητικών και Λογικών Πράξεων καθώς και οι Εντολές Ολίσθησης και Περιστροφής, εκτελούνται στην ALU.
Τέτοιες εντολές είναι για παράδειγμα οι ακόλουθες:

ADD
ADC
SUB
SBB
MUL
DIV
AND
OR
SHL …

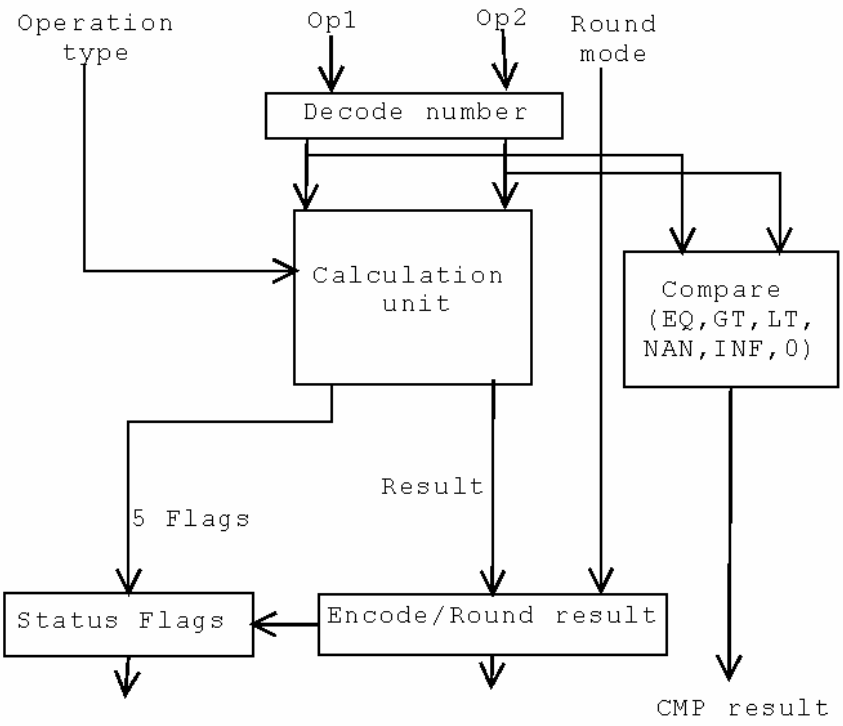
4.3.4. Μονάδα Κινητής Υποδιαστολής (Floating Point Unit – FPU)

Η Μονάδα Κινητής Υποδιαστολής (Floating Point Unit – FPU) εκτελεί πράξεις με δεκαδικούς αριθμούς.
Οι δεκαδικοί αριθμοί που χειρίζεται η FPU ακολουθούν συγκεκριμένη κωδικοποίηση που αναλύεται σε επόμενο κεφάλαιο, και στην ουσία αποτελείται από έναν αριθμό «βάσης» που είναι δυαδικός προσημασμένος ακέραιος αλλά θεωρείται κατά σύμβαση ότι κατά απόλυτη

τιμή είναι στην περιοχή 0..1, και από έναν προσημασμένο ακέραιο «εκθέτη» που καθορίζει το πραγματικό μέγεθος του αριθμού. Δηλαδή η κωδικοποίηση είναι ως εξής :

$$\text{Δεκαδικός Αριθμός} = \text{Βάση} \times 10^{\text{Εκθέτης}}$$

Ο προσημασμένος εκθέτης μπορεί να μας δώσει πολύ μικρούς αριθμούς (αρνητικοί εκθέτες) ή πολύ μεγάλους αριθμούς (θετικοί εκθέτες).
Η FPU μπορεί να εκτελεί αλγεβρικές πράξεις με δεκαδικούς αριθμούς όπως πρόσθεση, αφαίρεση πολλαπλασιασμό και διαίρεση. Επίσης μπορεί να εκτελεί πράξεις σύγκρισης, όπως ==, >=, <=, !=, όπως επίσης και έλεγχο για άπειρο (INF) ή για η αποδεκτούς αριθμούς (NAN – Not A Number). Έχει δικές της εντολές που αναγνωρίζει και τις οποίες οδηγεί η μονάδα ελέγχου στην FPU για την εκτέλεση της σωστής πράξης. Επίσης έχει και δικούς της καταχωρητές για καταχώρηση των αριθμών που θα συμμετέχουν στην πράξη ή θα αποθηκεύσουν ενδιάμεσα αποτελέσματα (data registers), όπως επίσης και καταχωρητές που αντανακλούν την κατάσταση της FPU ή του αποτελέσματος της πράξης (status registers).
Στο παρακάτω σχήμα φαίνεται η εικόνα ενός μπλόκ διαγράμματος μιας απλής FPU.



Μπλοκ διάγραμμα απλής FPU

Στο διάγραμμα αυτό φαίνονται καθαρά οι γραμμές προσκόμισης των δύο αριθμών που θα συμμετέχουν στην πράξη και που φαίνονται στο διάγραμμα με τα ονόματα op1 και op2. Επίσης φαίνεται η είσοδος της εντολής για την πράξη που θα εκτελέσει η μονάδα κινητής υποδιαστολής (operation type). Επίσης φαίνεται η μονάδα εκτέλεσης των πράξεων (calculation unit) όπως και η μονάδα που κάνει τις συγκρίσεις (Compare). Τα αποτελέσματα των μονάδων αυτών καταχωρούνται σε ειδικούς καταχωρητές αποτελεσμάτων όπως και καταχωρητές κατάστασης (status).
Στην επόμενη εικόνα φαίνεται το μπλοκ διάγραμμα μιας πιο σύνθετης μονάδας κινητής υποδιαστολής. Στη μονάδα αυτή είναι εμφανές ότι υπάρχουν δύο ανεξάρτητοι αγωγοί προσκόμισης εντολών με τις ονομασίες Instruction 1 και Instruction 2. Οι δύο αυτοί αγωγοί επιτρέπουν την ταυτόχρονη προσκόμιση δύο εντολών στη μονάδα. Εφόσον οι εντολές αυτές

είναι ανεξάρτητες μεταξύ τους μπορεί και να εκτελεστούν παράλληλα. Έτσι αυξάνεται η αποδοτικότητα της μονάδας στην ταχύτητα εκτέλεσης εντολών.

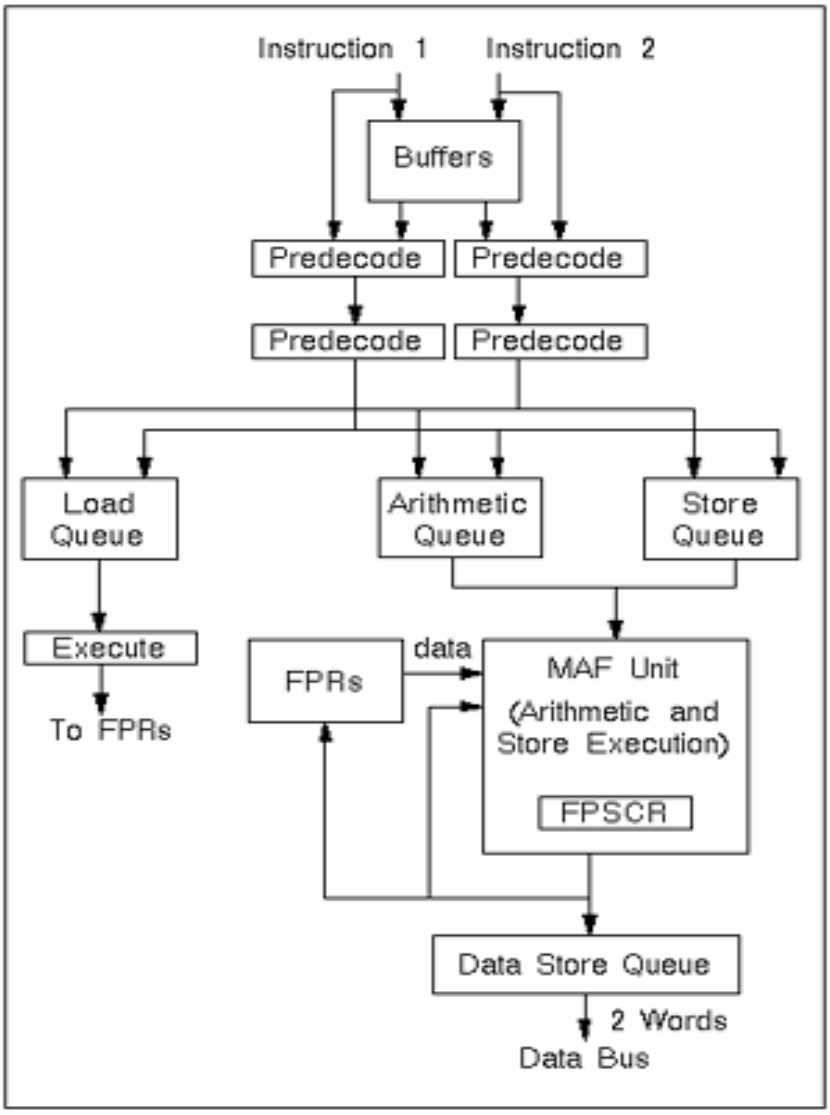


Figure 1 Block Diagram of POWER FPU

4.3.5. Η κωδικοποίηση των δεκαδικών αριθμών

Οι δεκαδικοί αριθμοί στους Η/Υ ακολουθούν συγκεκριμένη κωδικοποίηση που καθιερώθηκε το 1985 με το στάνταρ IEEE 754, το οποίο και υποστηρίζεται από την μεγάλη πλειοψηφία των μικροεπεξεργαστών παγκοσμίως. Πριν εμβαθύνουμε στην κωδικοποίηση των δεκαδικών, θα αναλύσουμε την δυαδική κωδικοποίηση των κλασματικών αριθμών.
Όπως ένας ακέραιος μπορεί να παρασταθεί ως δυαδικός αριθμός, π.χ. ο αριθμός 179_{10} αντιστοιχεί στον δυαδικό αριθμό $1\ 0\ 1\ 1\ 0\ 0\ 1\ 1_{\ 2}$, το οποίο φαίνεται παραστατικά σύμφωνα με την παρακάτω αναλυση :

Θέση bit	7	6	5	4	3	2	1	0
Δύναμη του 2	128	64	32	16	8	4	2	1
Δυαδικός Αριθμός	1	0	1	1	0	0	1	1
Συμβολή των bits	$1x2^7$	$0x2^6$	$1x2^5$	$1x2^4$	$0x2^3$	$0x2^2$	$1x2^1$	$1x2^0$

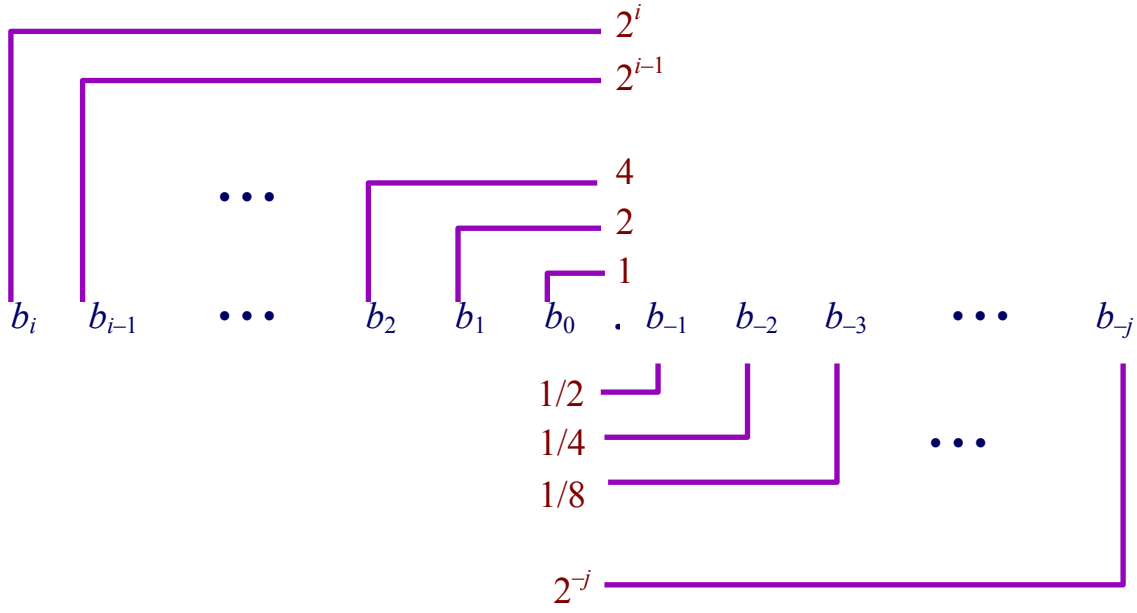
Αποτέλεσμα	128	0	32	16	0	0	2	1
Αριθμός	128+32+16+2+1 = 179							

έτσι κατ' επέκταση μπορούμε να έχουμε και δυαδικούς αριθμούς που να περιλαμβάνουν δεκαδική τελεία. Σε ένα τέτοιο αριθμό, τα ψηφία στα δεξιά της τελείας πολλαπλασιάζονται με αρνητικές δυνάμεις του 2. Για παράδειγμα ο δυαδικός κλασματικός αριθμός :

1 0 1 1 . 1 0 1 0 1 αναπαριστά τον αριθμό :

$1x2^3+0x2^2+1x2^1+1x2^0+1x2^{-1}+0x2^{-2}+1x2^{-3}+0x2^{-4}+1x2^{-5} =$
$1x8+0x4+1x2+1x1+1x1/2+0x1/4+1x1/8+0x1/16+1x1/32 =$
$8 + 0 + 2 + 1 + 1/2 + 0 + 1/8 + 0 + 1/32 = 11.65625$

Γενικά ένας κλασματικός δυαδικός αριθμός της μορφής :



αναπαριστά τον αριθμό:

$$\sum_{k=-j}^{i} b_k \cdot 2^k$$

Παραδείγματα δυαδικών κλασματικών αριθμών :

$5\frac{3}{4} = 5.75 = 101.11$

$13\frac{7}{8} = 13.875 = 1101.111$

$51\frac{63}{64} = 51.984375 = 110011.111111$

Προφανώς η μετατόπιση (shift) ενός κλασματικού δυαδικού αριθμού προς τα αριστερά πολλαπλασιάζει τον αριθμό επί 2, ενώ η μετατόπιση προς τα δεξιά τον διαιρεί δια 2.

Ο αριθμός 0.11111111111111…….$_2$ προφανώς τείνει στην μονάδα χωρίς ποτέ βέβαια να την φτάνει, δηλαδή ταυτίζεται με τον αριθμό 0.99999999999999……$_{10}$. Για τον λόγο αυτό συμβολίζεται ως «1.0-ε» όπου «ε» ένας πολύ μικρός αριθμός.

Η αναπαράσταση αυτή έχει βέβαια και κάποιους περιορισμούς. Ο βασικός περιορισμός είναι ότι μπορεί να αναπαραστήσει με πεπερασμένο πλήθος ψηφίων μόνο αριθμούς της μορφής $X/2^k$, δηλαδή αριθμούς οι οποίοι μας δίνουν ακέραιο αριθμό χωρίς κλασματικό μέρος όταν πολλαπλασιαστούν με κάποια δύναμη του 2 (2^k).
Έτσι οι αριθμοί όπως 1/3 = 0.3333333, ή 1/5 = 0.2, κ.λ.π. αναπαρίστανται με άπειρα επαναλαμβανόμενα ψηφία στο δεκαδικό μέρος του αριθμού :
1/3 = 0.333333….. = 0.0101010101010101[01]…..$_2$
1/5 = 0.2 = 0.0011001100110011[0011]…..$_2$
1/10 = 0.1 = 0.000110011001100110011[0011]….$_2$

Αναπαράσταση αριθμών κινητής υποδιαστολής
Με βάση τους κλασματικούς αριθμούς που αναλύθηκαν παραπάνω κτίζεται και η αναπαράσταση των αριθμών κινητής υποδιαστολής που χρησιμοποιείται στους Η/Υ. η αναπαράσταση αυτή αποτελείται από 3 μέρη όπως φαίνεται παρακάτω:

όπου :
S : είναι το bit προσήμου (sign bit) που παίρνει τιμές 0 για θετικό πρόσημο (+) και 1 για αρνητικό πρόσημο (-)
E : είναι ένας ακέραιος εκθέτης στον οποίο υψώνεται το 2 για να δημιουργήσει μία δύναμη του 2 που πολλαπλασιάζει τον αριθμό. Εκφράζει το μέγεθος του αριθμού δηλαδή το πόσο μεγάλος ή μικρός είναι ο αριθμός. Μεγάλα θετικά νούμερα δημιουργούν πολύ μεγάλους αριθμούς, ενώ μεγάλα αρνητικά νούμερα δημιουργούν πολύ μικρούς αριθμούς. Ονομάζεται **Exponent** δηλαδή εκθέτης.
M : είναι ένας κλασματικός αριθμός στην περιοχή $1.0 \leq M \leq 2.0$ που καθορίζει τα σημαντικά ψηφία του αριθμού (την ακρίβεια του αριθμού) και ονομάζεται **Mantissa** ή **Significand**

Τυπικά ο αριθμός που κωδικοποιείται με τα 3 αυτά πεδία είναι ο αριθμός

$$-1^S \times M \times 2^E$$

Για να κωδικοποιήσουμε έναν αριθμό κινητής υποδιαστολής με αυτό τον τρόπο κάνουμε τα εξής βήματα:
1. Μετατρέπουμε τον αριθμό σε κλασματικό δυαδικό
2. Τον μετατοπίζουμε δεξιά (διαίρεση δια 2) τόσες φορές όσες χρειάζεται ώστε να έρθει στην μορφή 1.XXXXXX…, δηλαδή να υπάρχει μόνο μία μονάδα στα αριστερά της δεκαδικής τελείας και όλο το υπόλοιπο νούμερο να είναι στα αριστερά της (κανονικοποίηση).
3. Υπολογίζουμε πόσες μετατοπίσεις κάναμε και το νούμερο αυτό αποτελεί τον εκθέτη.

Παράδειγμα ο αριθμός 51.984375 θα κωδικοποιηθεί με τα ακόλουθα βήματα:
1. Μετατρέπεται σε κλασματικό δυαδικό και γίνεται 110011.111111
2. Μετατοπίζεται 5 φορές στα δεξιά και γίνεται 1.10011111111
3. Κωδικοποιείται ως : S=0 E=101_2 (5_{10}) M= 1.10011111111

Επειδή η Mantissa πάντα είναι της μορφής 1.XXXXXXX…, η μονάδα στα αριστερά της δεκαδικής τελείας δεν αποθηκεύεται στον αριθμό αλλά εννοείται, καθώς δεν αποτελεί πληροφορία που πρέπει να αποθηκευθεί. Έτσι στην κωδικοποίηση αυτή κερδίζουμε 1 bit στην ακρίβεια του αριθμού. Δηλαδή αντί για τον αριθμό M= 1.10011111111 αποθηκεύεται ο αριθμός M= 0.10011111111, δηλαδή τελικά με την αφαίρεση του «0.» το οποίο δεν χρειάζεται, ο αριθμός M που αποθηκεύεται θα είναι M= 10011111111. Στον αριθμό αυτό εννοείται ότι προηγείται η μονάδα και η δεκαδική τελεία.

Επίσης για να μπορούμε να καταχωρούμε και αρνητικούς εκθέτες οι εκθέτες αποθηκεύονται με ένα offset ή bias το οποίο τυπικά είναι offset=+127. Δηλαδή οι αριθμοί εκθετών από E=0 έως E=126 αντιστοιχούν στους πραγματικούς εκθέτες -127 έως -1, ο αριθμός εκθέτη E=127 αντιστοιχεί σε μηδενικό εκθέτη, και οι αριθμοί εκθετών E=128 έως E=254 αντιστοιχούν στους πραγματικούς εκθέτες +1 έως +127. Ο αριθμός εκθέτη E=255 είναι δεσμευμένος για να κωδικοποιήσει την τιμή του άπειρου (infinite ή ∞).
Έτσι ο αριθμός του παραπάνω παραδείγματος θα κωδικοποιηθεί τελικά ως εξής :

0	10000100	10011111111

που έχει :

S=0	δηλαδή ο αριθμός είναι θετικός
$E=10000100_2 = 132_{10}$=(127+5)	δηλαδή ο εκθέτης είναι 5 (127 είναι το offset ή bias)
M=10011111111	δηλαδή 1.10011111111

Και ο αριθμός που σχηματίζεται είναι ο :

$$+ 1.10011111111_2 \times 2^5 = 110011.111111_2 = 51.984375_{10}$$

Τύποι και μεγέθη δεκαδικών αναπαραστάσεων

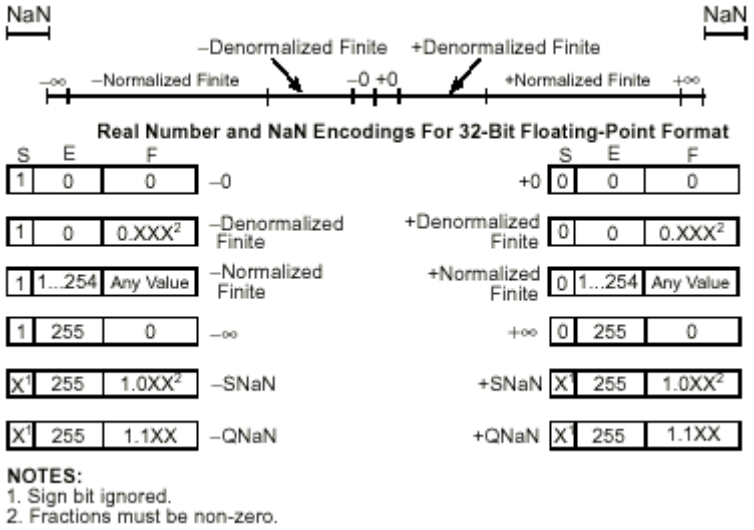
Υπάρχουν τρεις τύποι δεκαδικών αναπαραστάσεων που διαφέρουν στο πλήθος των bits που προβλέπουν για κάθε κομμάτι του κωδικοποιημένου αριθμού. Έτσι έχουμε τους τύπους :

Τύπος	**Bits Εκθέτη**	**Bits Mantissa**	**Σύνολο Bits (μαζί με το bit προσήμου)**
Απλής ακρίβειας - Single precision (τύπος float στην γλώσσα C)	8 (-127..0..127)	23	32
Διπλής ακρίβειας - Double precision (τύπος double στην γλώσσα C)	11 (-1023..0..1023)	52	64
Εκτεταμένης ακρίβειας - Extended precision (χρησιμοποιείται μόνο από την Intel)	15 (-16383..0..16383)	63	80 (ένα bit δεν χρησιμοποιείται)

Μερικές χαρακτηριστικές τιμές στην κωδικοποίηση των αριθμών είναι οι ακόλουθες:

S	**E**	**M**	**Σημασία**
0	0	0	+0
1	0	0	-0
0	255	0	+∞
1	255	0	-∞
0 ή 1	255	Οποιαδήποτε τιμή	NAN (Not A Number)

Παρακάτω φαίνεται ένας πίνακας της Intel που αναφέρει πιο αναλυτικά τις τιμές αυτές στην κωδικοποίηση των αριθμών κινητής υποδιαστολής:



Πίνακας της Intel για την κωδικοποίηση των αριθμών κινητής υποδιαστολής

Πράξεις μεταξύ δεκαδικών αριθμών

Οι πράξεις που γίνονται με τους κωδικοποιημένους δεκαδικούς αριθμούς γίνονται ακολουθώντας το στάνταρ IEEE 754. Στην γενική περίπτωση οι δύο αριθμοί που θα συμμετέχουν στην πράξη αποκωδικοποιούνται σε κλασματικούς δυαδικούς αριθμούς. Στη συνέχεια πραγματοποιείται η πράξη επάνω σε αυτούς με δυαδική αριθμητική. Το αποτέλεσμα που προκύπτει κανονικοποιείται και μετατρέπεται πάλι σε κωδικοποιημένο αριθμό κινητής υποδιαστολής.

Σε μερικές περιπτώσεις πράξεων μπορεί όμως η διαδικασία να απλοποιηθεί και να επιταχυνθεί όπως στην περίπτωση της προσθέσεις και της αφαίρεσης. Η πρόσθεση και ή αφαίρεση δέχονται δύο τελεσταίους αριθμούς που θα συμμετέχουν στην πράξη. Αν και οι δύο αριθμοί είναι αποδεκτοί δεκαδικοί αριθμοί και δεν ανήκουν σε κάποια από τις ειδικές περιπτώσεις (όπως NAN ή ∞) τότε ακολουθούνται οι παρακάτω κανόνες για την πραγματοποίηση της πράξης.

1. Αρχικά συγκρίνονται οι εκθέτες των δύο αριθμών.
2. Αν οι εκθέτες είναι οι ίδιοι τότε μπορούν άμεσα να προστεθούν τα κομμάτια των δύο αριθμών που περιέχουν τα σημαντικά τους ψηφία (mantissa)
3. Αν το αποτέλεσμα της πράξης είναι συμβατό με την κωδικοποίηση, δηλαδή είναι $1.0 \leq M \leq 2.0$, τότε η πράξη έχει ολοκληρωθεί και ο αριθμός του αποτελέσματος έχει τον ίδιο εκθέτη με τους αρχικούς τελεσταίους.
4. Αν το αποτέλεσμα της πράξης δεν είναι συμβατό, δηλαδή $M<1.0$ ή $M>2.0$ τότε θα πρέπει να γίνει μια διαδικασία κανονικοποίησης η οποία θα φέρει την ακρίβεια του αριθμού (mantissa) στα σωστά όρια επηρεάζοντας τον εκθέτη. Για παράδειγμα αν το αποτέλεσμα είναι:
   10.1010_2 και ο εκθέτης ήταν 101_2 (5_{10}) τότε η mantissa θα γίνει
   1.01010_2 και ο εκθέτης θα αυξηθεί κατά ένα και θα γίνει 110_2 (6_{10})
5. Εάν όμως οι εκθέτες των δύο αριθμών διαφέρουν τότε ο αριθμός με τον μικρότερο εκθέτη παραμένει ο ίδιος, ενώ αριθμός με τον μεγαλύτερο εκθέτη πολλαπλασιάζεται επί δύο (αριστερή μετατόπιση) και ο εκθέτης του μειώνεται κατά ένα, τόσες φορές όσες χρειάζεται ώστε να εξισωθούν οι δύο εκθέτες.

6. Στη συνέχεια η πράξη εξελίσσεται όπως και προηγουμένως

Το πρόσημο του αριθμού καθορίζεται ως εξής.

- Αν το αποτέλεσμα είναι μη μηδενικό, τότε το πρόσημο του αποτελέσματος ταυτίζεται με το πρόσημο του αρχικού αριθμού (τελεσταίου) που είχε τη μεγαλύτερη απόλυτη τιμή.
- Διαφορετικά εάν το αποτέλεσμα είναι μηδενικό, τότε το πρόσημο του είναι 0 (θετικό) και το αποτέλεσμα είναι «+0», εκτός εάν και οι δύο αρχικοί αριθμοί ήταν αρνητικοί οπότε γίνεται 1 (αρνητικό) και το αποτέλεσμα είναι «-0».

Η αφαίρεση είναι η ίδια με την πρόσθεση αρκεί να αλλαχθεί το πρόσημο του δεύτερου τελεσταίου αριθμού.

Παράδειγμα 1 :

Πρόσθεση ($+1.0101 \times 2^{101}$, $+1.1010 \times 2^{101}$)

Οι εκθέτες είναι ίδιοι άρα προστίθενται τα +1.0101 και +1.1010, με αποτέλεσμα 10.1111 και μαζί με τον εκθέτη και το πρόσημο θα είναι $+10.1111 \times 2^{101}$. Το αποτέλεσμα της mantissa (significand) δεν είναι στα όρια [1.0,2.0) και επομένως πρέπει να κανονικοποιηθεί. Έτσι η mantissa θα διαιρεθεί δια 2 (μετατόπιση δεξιά) και ο εκθέτης θα αυξηθεί κατά ένα, δηλαδή

- ✓ Η mantissa θα γίνει 10.1111 → 1.01111 , και
- ✓ εκθέτης θα γίνει 101 → 110

Το τελικό αποτέλεσμα της πρόσθεσης που είναι επίσης και συμβατό με την κωδικοποίηση των δεκαδικών αριθμών θα είναι : $+1.01111 \times 2^{110}$

Παράδειγμα 2 :

Αφαίρεση ($+1.0011 \times 2^{101}$, -1.011×2^{110})

Αρχικά η αφαίρεση θα γίνει πρόσθεση αντιστρέφοντας το πρόσημο του δεύτερου αριθμού, δηλαδή θα γίνει :

Πρόσθεση ($+1.0011 \times 2^{101}$, $+1.011 \times 2^{110}$)

Οι εκθέτες δεν είναι ίδιοι άρα αυτός με τον μικρότερο εκθέτη (ο πρώτος) θα παραμείνει αναλλοίωτος. Ο δεύτερος από τους δύο θα πρέπει να αλλαχθεί ώστε να αποκτήσει τον ίδιο εκθέτη με τον πρώτο.

Έτσι στον αριθμό $+1.011 \times 2^{110}$ θα μειωθεί ο εκθέτης κατά ένα (διαίρεση δια δύο) και θα μετατοπιστεί η mantissa μία θέση αριστερά (πολλαπλασιασμός επί 2). Άρα

- ✓ Ο εκθέτης θα γίνει 110 →101 , και
- ✓ Η mantissa θα γίνει 1.011 → 10.11

Έτσι η πράξη ανάγεται στην παρακάτω :

Πρόσθεση ($+1.0011 \times 2^{101}$, $+10.11 \times 2^{101}$)

Επομένως τώρα προστίθενται τα +1.0011 και +10.11, με αποτέλεσμα 11.1111 και μαζί με τον εκθέτη και το πρόσημο θα είναι $+11.1111 \times 2^{101}$. Το αποτέλεσμα της mantissa (significand) δεν είναι στα όρια [1.0,2.0) και επομένως πρέπει να κανονικοποιηθεί. Έτσι η mantissa θα διαιρεθεί δια 2 (μετατόπιση δεξιά) και ο εκθέτης θα αυξηθεί κατά ένα, δηλαδή

- ✓ Η mantissa θα γίνει 11.1111 → 1.11111 , και
- ✓ εκθέτης θα γίνει 101 → 110

Το τελικό αποτέλεσμα της πρόσθεσης που είναι επίσης και συμβατό με την κωδικοποίηση των δεκαδικών αριθμών θα είναι : $+1.11111 \times 2^{110}$

Οι ειδικές περιπτώσεις συνοψίζονται στα παρακάτω:

- add('Infinity', '1') ==> 'Infinity'
- add('NaN', '1') ==> 'NaN'
- add('NaN', 'Infinity') ==> 'NaN'
- subtract('1', 'Infinity') ==> '–Infinity'

- multiply('–1', 'Infinity') ==> '–Infinity'
- subtract('–0', '0') ==> '–0'
- multiply('–1', '0') ==> '–0'
- divide('1', '0') ==> 'Infinity'
- divide('1', '–0') ==> '–Infinity'
- divide('–1', '0') ==> '–Infinity'

4.3.6. Η Μονάδα Ελέγχου

Η μονάδα ελέγχου είναι το πιο σημαντικό κομμάτι σε ένα μικροεπεξεργαστή. Αποτελεί στην ουσία τον «εγκέφαλο» της CPU ο οποίος είναι υπεύθυνος για τον συντονισμό και την λειτουργία όλων των υπόλοιπων τμημάτων του μικροεπεξεργαστή. Η μονάδα ελέγχου αναλαμβάνει την σειριακή φόρτωση των εντολών γλώσσας μηχανής του προγράμματος που εκτελείται κάθε φορά στο εσωτερικό του επεξεργαστή, την αναγνώριση της κάθε εντολής και των παραμέτρων της, και την αποστολή των κατάλληλων σημάτων ελέγχου προς τα υπόλοιπα τμήματα τα CPU (όπως καταχωρητές, εσωτερικοί δίαυλοι, ALU, FPU, level 1 cache, κ.λ.π.) με την κατάλληλη σειρά και τον κατάλληλο χρονισμό, ώστε να εκτελεστεί με επιτυχία η κάθε εντολή. Είναι προφανές ότι το σετ εντολών που αναγνωρίζει μία CPU , όπως οι 90 εντολές που αναγνωρίζει ο 8088, είναι κωδικοποιημένες μέσα στην κατασκευή της μονάδας ελέγχου. Δηλαδή η μονάδα ελέγχου είναι αυτή που είναι υπεύθυνη για το γεγονός ότι ένας επεξεργαστής όπως ο 8088 αναγνωρίζει το συγκεκριμένο σετ εντολών (π.χ. 90 εντολές στον 8088). Έτσι αν θέλουμε να ξανασχεδιάσουμε έναν επεξεργαστή ώστε να του προσθέσουμε την δυνατότητα να αναγνωρίζει 10 επιπλέον εντολές, το τμήμα της CPU που θα πρέπει να ξανασχεδιάσουμε είναι η μονάδα ελέγχου.
Το έργο οιπόν της μονάδας ελέγχου συνοψίζεται στα παραάτω:

1. Παίρνει είσοδο από το ρολόι του συστήματος και με βάση αυτό συγχρονίζει όλες τις ενέργειές της..
2. Παράγει σήματα συγχρονισμού προς όλες τις υπομονάδες.
3. Προσκομίζει κάθε εντολή και την αποθηκεύει στον IR.
4. Αποκωδικοποιεί την εντολή ενεργοποιώντας τις κατάλληλες υπομονάδες του M/E για την εκτέλεσή της.
5. Χειρίζεται τις εντολές μεταφοράς δεδομένων μεταξύ καταχωρητών, ALU, FPU, μνήμης και I/O.
6. Συνδέεται και ελέγχει το Control Bus, που χρησιμοποιείται για έλεγχο όλου του υπολογιστικού συστήματος.
7. Δέχεται τις διακοπές Interrupts και τις προωθεί στον M/E και παράγει σήματα διακοπής προς περιφερειακές συσκευές.
8. Δέχεται όλα τα υπόλοιπα σήματα ελέγχου, όπως τα σήματα ελέγχου του καναλιού DMA και αποκρίνεται σε αυτά.

Ο κύκλος εκτέλεσης κάθε εντολής υλοποιείται από τη μονάδα ελέγχου και έχει τρεις φάσεις :

1. Προσκόμιση Εντολής (fetch),
2. Αποκωδικοποίηση της εντολής και Υπολογισμός τελικής διεύθυνσης ή προσκόμιση τελικών δεδομένων (decoding & effective address calculation / final data fetching),
3. Εκτέλεση εντολής (execution).

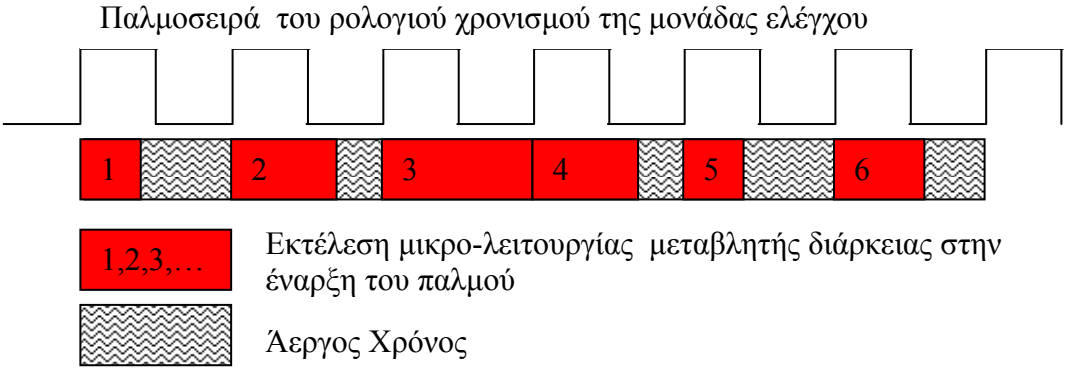
Οι μονάδες ελέγχου στους μικροεπεξεργαστές υλοποιούνται με δύο διαφορετικές τεχνικές υλοποίησης.

1. Η πρώτη τεχνική είναι η κατασκευή μονάδων ελέγχου με συνδυαστικά **λογικά κυκλώματα**.
2. Η δεύτερη τεχνική συνιστάται στην κατασκευή **μικρο-προγραμματιζόμενων μονάδων ελέγχου** οι οποίες είναι κατασκευασμένες να εκτελούν εντολές γραμμένες σε μια γλώσσα μικρο-προγραμματισμού (μικρο-εντολές). Περιέχουν μια μόνιμη μνήμη ROM η οποία περιέχει ρουτίνες μικροκώδικα για όλες τις εντολές που αναγνωρίζει ο επεξεργαστής.

1. Στην υλοποίηση μονάδων ελέγχου με **λογικά κυκλώματα** (Hard-wired Control Unit) η μονάδα ελέγχου σχεδιάζεται εξολοκλήρου ως ένα πάρα πολύ σύνθετο ψηφιακό ακολουθιακό κύκλωμα. Αυτό έχει την ικανότητα να αναγνωρίζει στην είσοδό του ένα μεγάλο σετ διαφορετικών αριθμών-εντολών και να αποκρίνεται σε κάθε τέτοια εντολή με μια ακολουθία τιμών εξόδου οι οποίες οδηγούνται στα διάφορα τμήματα του επεξεργαστή, όπως οι καταχωρητές, η εσωτερικοί δίαυλοι, η ALU, η FPU κ.λ.π., και τα ενεργοποιούν ακολουθιακά ώστε να εκτελεστεί με επιτυχία η κάθε εντολή.
   Όπως γίνεται προφανές η κατασκευή τέτοιων μονάδων ελέγχου είναι εξαιρετικά δύσκολη και τόσο πιο δύσκολη όσο μεγαλύτερο είναι το σετ εντολών που πρέπει να αναγνωρίσει η μονάδα ελέγχου.
   Βέβαια, επειδή η λειτουργία της μονάδας ελέγχου υλοποιείται εξολοκλήρου από υικό (hardware), οι μονάδες αυτές είναι ταχύτατες στην απόκρισή τους και στη λειτουργία τους. Αυτό συμβάλλει στην ταχύτατη εκτέλεση εντολών γλώσσας μηχανής από τον μικροεπεξεργαστή. Και αυτό βέβαια είναι το πλεονέκτημα αυτής της τεχνολογίας κατασκευής μονάδων ελέγχου
   Οι εντολές γλώσσας μηχανής εκτελούνται πρακτικά σε ένα κύκλο εκτέλεσης
   Το μειονέκτημα όμως σε αυτές τις μονάδες είναι ότι εάν θελήσουμε να αυξήσουμε το σετ εντολών του μικροεπεξεργαστή τότε θα πρέπει να ξανασχεδιάσουμε την μονάδα ελέγχου από την αρχή. Επομένως στις μονάδες αυτές δεν είναι εύκολο να επεκταθεί το σετ εντολών του μικροεπεξεργαστή, δηλαδή οι εντολές γλώσσας μηχανής που αναγνωρίζει και εκτελεί.
   Οι μονάδες αυτές που κατασκευάζονται με ψηφιακά κυκλώματα χωρίζονται σε δύο υποκατηγορίες :
   A. Στις **σύγχρονες μονάδες ελέγχου** και
   B. Στις **ασύγχρονες μονάδες ελέγχου**.

   A. Στις **σύγχρονες μονάδες ελέγχου** η λειτουργία τους βασίζεται σε ένα ρολόι δηλαδή σε ένα ψηφιακό κύκλωμα που παράγει μια τετραγωνική παλμοσειρά αυστηρά καθορισμένης συχνότητας.
      Όλες οι ενέργειες της μονάδας ελέγχου γίνονται με βάση τους παλμούς αυτού του ρολογιού.
      Σε κάθε παλμό του ρολογιού εκτελείται και μια μικρό-λειτουργία μέσα στον επεξεργαστή καθοδηγούμενη από τη μονάδα ελέγχου
      Για παράδειγμα, μπορεί μέσα σε έναν παλμό του ρολογιού να εκτελείται μία πρόσθεση στην αριθμητική και λογική μονάδα ή να μεταφέρονται δεδομένα από τον ένα καταχωρητή στον άλλο.
      Επειδή όμως όλες αυτές οι ενέργειες δεν έχουν την ίδια διάρκεια, δηλαδή άλλες είναι γρήγορες και άλλες αργές, η περίοδος του ρολογιού είναι κατάλληλα ρυθμισμένη ώστε η μονάδα ελέγχου να προλαβαίνει να εκτελέσει ακόμα και την πιο αργή μικρό-λειτουργία
      Αυτό έχει όμως σαν συνέπεια, τόσο οι αργές λειτουργίες όσο και οι γρήγορες λειτουργίες να καταλαμβάνουν στην ουσία έναν παλμό του ρολογιού, με αποτέλεσμα

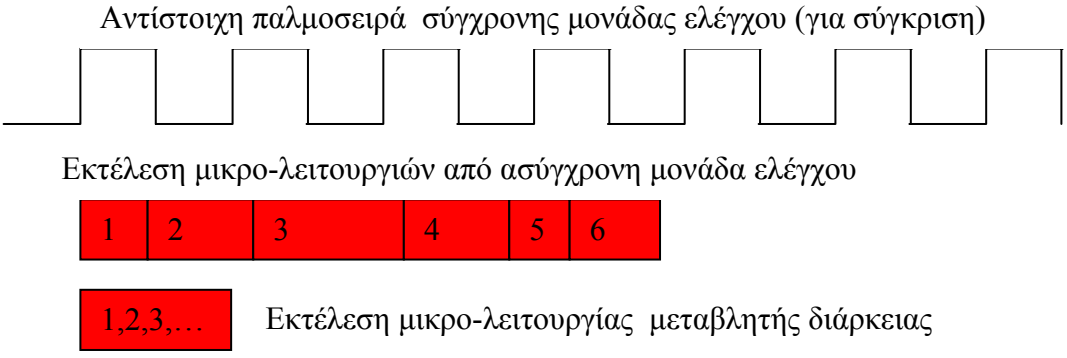
να υπάρχει αρκετός άεργος χρόνος στην εκτέλεση των λειτουργιών στη μονάδα ελέγχου, όπως φαίνεται στο παρακάτω σχήμα.



Εκτέλεση βημάτων από σύγχρονη μονάδα ελέγχου

Αυτό είναι και το μειονέκτημα των σύγχρονων μονάδων ελέγχου. Αντίθετα το πλεονέκτημά τους είναι ότι είναι ευκολότερες στην κατασκευή σε σχέση με την επόμενη τεχνική τις ασύγχρονες μονάδες ελέγχου.

Β. Η δεύτερη κατηγορία μονάδων ελέγχου που κατασκευάζονται αποκλειστικά με ψηφιακά κυκλώματα είναι οι **ασύγχρονες μονάδες ελέγχου**. Σε αυτές τις μονάδες δεν υπάρχει ρολόι πού να συγχρονίζει την εκτέλεση των μικρών λειτουργιών από τη μονάδα ελέγχου. Αντίθετα μετά το πέρας κάθε μικρο-λειτουργίας διεγείρεται η έναρξη του επόμενου βήματος. Αυτή η τεχνική φαίνεται στο παρακάτω σχήμα:



Εκτέλεση βημάτων από ασύγχρονη μονάδα ελέγχου

Με την τεχνική αυτή δεν υπάρχει πλέον άεργος χρόνος στη λειτουργία της μονάδας ελέγχου και έτσι αυξάνεται συνολικά η απόδοση του μικροεπεξεργαστή σχετικά με την ταχύτητα εκτέλεσης εντολών. Η ταχύτητα αυτών των μονάδων ελέγχου είναι και το πλεονέκτημά τους. Το μειονέκτημά τους είναι ότι είναι δυσκολότερες στην σχεδίαση και την κατασκευή σε σχέση με τις σύγχρονες μονάδες ελέγχου.

2. Η δεύτερη τεχνική κατασκευής μονάδων ελέγχου είναι οι μονάδες ελέγχου που λειτουργούν **με μικρό-προγραμματισμό** (Microprogrammed Control Unit).
Ξεκινώντας από τη δεκαετία του '50, με τη σειρά System/360 της IBM, και μέχρι το τέλος της δεκαετίας του '70 κυριάρχησε η τάση οι εντολές να διερμηνεύονται μέσω μικροπρογράμματος και να μην εκτελούνται απευθείας από το υλικό, ενώ βαθμιαία οι

εντολές γίνονταν όλο και πιο σύνθετες. Αποκορύφωμα αυτής της τάσης ήταν ο VAX της Digital Equipment Corporation, ο οποίος είχε εκατοντάδες διερμηνευόμενες εντολές και περισσότερους από 200 τρόπους για τον προσδιορισμό των τελεστέων των εντολών. Σταδιακά, και με την έλευση των RISC μ/ε (δες σχετικό κεφάλαιο) φάνηκε ότι η διερμηνευόμενες εντολές μείωναν την απόδοση. Σήμερα η τάση είναι ένα μέρος των εντολών να είναι διερμηνευόμενες και ένα μέρος άμεσα εκτελέσιμες από το υλικό.

Σε αυτή την τεχνολογία οι μονάδες ελέγχου κατασκευάζονται ως «μικροί επεξεργαστές» καθώς έχουν την ικανότητα εκτέλεσης εντολών όπως και ένας μικροεπεξεργαστής. Οι εντολές αυτές που ονομάζονται μικρο-εντολές αφορούν στην εκτέλεση μικρών λειτουργιών μέσα στον επεξεργαστή. Οι μικρο-εντολές ελέγχουν όλα τα εσωτερικά μέρη του Μ/Ε (διαύλους, εσωτερικούς καταχωρητές, κανονικούς καταχωρητές, ALU, FPU κ.λ.π.). Οι μικρο-εντολές εκτελούνται σειριακά, δηλαδή η μία μετά την άλλη. Όμως αρκετές μικρο-εντολές μπορούν να εκτελούνται και παράλληλα. Ο παραλληλισμός στην εκτέλεση εντολών μικρο-κώδικα είναι θέμα έξυπνης σχεδίασης της μονάδας ελέγχου, και μπορεί να συμβάλλει σημαντικά στην αύξηση της ταχύτητας του μικροεπεξεργαστή.

Κάθε εντολή γλώσσας μηχανής απαιτεί πολλούς κύκλους ρολογιού, καθώς υλοποιείται με πολλές μικρολειτουργίες, δηλαδή με πολλές μικρο-εντολές. Δηλαδή, κάθε εντολή γλώσσας μηχανής υλοποιείται ως μία ακολουθία εντολών μικροκώδικα.

Οι εντολές αυτές σχηματίζουν μια γλώσσα προγραμματισμού της μονάδας ελέγχου που ονομάζεται γλώσσα μικρό-προγράμματος η γλώσσα μικρο-κώδικα. Είναι προφανές ότι η γλώσσα αυτή είναι κατά ένα επίπεδο κατώτερη από τη γλώσσα μηχανής. Σε αυτή τη γλώσσα προγραμματισμού δεν έχει πρόσβαση ο χρήστης αλλά μόνο ο κατασκευαστής του επεξεργαστή.

Για να μπορέσουν οι μονάδες ελέγχου αυτές να υλοποιήσουν ένα μεγάλο σετ εντολών γλώσσας μηχανής θα πρέπει να περιλαμβάνουν και μια μόνιμη μνήμη ROM μέσα στην μονάδα ελέγχου η οποία να περιλαμβάνει ένα μικρό-πρόγραμμα (ρουτίνα) με μικρο-εντολές για κάθε εντολή γλώσσας μηχανής που αναγνωρίζει ο μικρό επεξεργαστής

Έτσι με την προσκόμιση μιας εντολής γλώσσας μηχανής από τη μνήμη του υπολογιστή, η εντολή αναγνωρίζεται, και στη συνέχεια η μονάδα ελέγχου αρχίζει την εκτέλεση της κατάλληλης ρουτίνας από την μνήμη ROM που αντιστοιχεί σε αυτή την εντολή

Η ρουτίνα αυτή θα περιέχει όλα τα μικρό-βήματα που είναι απαραίτητα για την επιτυχή εκτέλεση της εντολής μέσα στον επεξεργαστή

Η μόνιμη μνήμη ROM περιέχει τόσες υπορουτίνες μικρο-κώδικα όσες είναι και οι εντολές που αναγνωρίζει ο επεξεργαστής.

Είναι προφανές ότι έτσι η εκτέλεση κάθε εντολής γλώσσας μηχανής ανάγεται στην εκτέλεση ενός μεγάλου αριθμού βημάτων μικρο-εντολών από μια μηχανή που τις διαβάζει και τις εκτελεί και που στην περίπτωση αυτή είναι η μονάδα ελέγχου. Αυτό κάνει τις μικρο-προγραμματιζόμενες μονάδες ελέγχουν να είναι αργές στην εκτέλεση των εντολών γλώσσας μηχανής. Και αυτό είναι το σημαντικό μειονέκτημα των μονάδων αυτών.

Το πλεονέκτημά τους όμως είναι ότι είναι πολύ εύκολο ο σχεδιαστής τους να επεκτείνει το σετ εντολών τους. Για να προστεθεί μια εντολή στο σετ εντολών που αναγνωρίζει ο επεξεργαστής, αρκεί να προσθέσει μία ακόμα υπορουτίνα μικρο-κώδικα μέσα στην ROM της μονάδας ελέγχου που να αντιστοιχεί στην επιπλέον εντολή.

Θα πρέπει να σημειωθεί ότι υπάρχει πλήρης ισοδυναμία ανάμεσα σε εντολές και υλικό, καθώς η κάθε εντολή μπορεί να εκτελεστεί και από υλικό και αντιστρόφως. Οι σχεδιαστές κάθε CPU αποφασίζουν κατά τα πρώτα στάδια της σχεδίασης ποιες λειτουργίες θα ενσωματώσουν σε εντολές και ποιες σε υλικό.

Για παράδειγμα ακολούθως παρατίθενται μερικές μικροεντολές σε συμβολική μορφή με την εξήγησή τους.

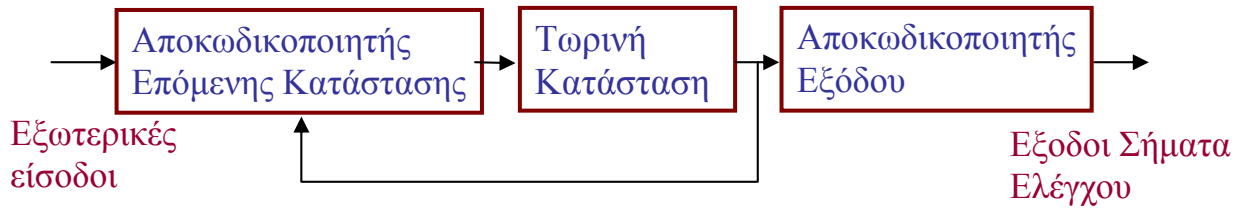
Παραδείγματα εντολών μικροκώδικα :

- MAR←IP αντέγραψε τον καταχωρητή IP στον MAR (σε συνδυασμό με τον CS). Η διεύθυνση που επεριείχαν οι CS:IP εξάγεται στο Address Bus
- IP=IP+1 αύξησε τον IP κατά ένα
- Wait περίοδος παύσης μέχρι να αποκριθεί η μνήμη.
- MDR ← DB σύνδεσε τον MDR με το Data Bus έτσι ώστε η λέξη (8 bit) που θα έρθει από το Data Bus να αποθηκευθεί στον MDR
- IR←MDR αποθήκευσε το περιεχόμενο του MDR στον καταχωρητή IR

Η ακολουθία των παραπάνω μικρο-εντολών ολοκληρώνει ένα κύκλο επικοινωνίας με την μνήμη RAM που ονομάζεται και **κύκλος μηχανής** (machine cycle). Κατά τον κύκλο αυτό η CPU αποστέλλει στην μνήμη την διεύθυνση του byte που θέλει να διαβάσει και αφού η μνήμη αποκριθεί, το byte διαβάζεται από τον δίαυλο δεδομένων και αποθηκεύεται πρώτα στον MDR και στην συγκεκριμένη περίπτωση προωθείται και στον καταχωρητή IR. Ο συγκεκριμένος κύκλος είναι στην ουσία ένας κύκλος προσκόμισης του opcode μίας εντολής γλώσσας μηχανής το οποίο διαβάζεται από την διεύθυνση νήμης που «δείχνει» ο IP (CS:IP) και αποθηκεύεται τελικά στον IR (Instruction Register), που βρίσκεται μέσα στην μονάδα ελέγχου, ώστε η εντολή να αποκωδικοποιηθεί , δηλαδή να καταλάβει η μονάδα ελέγχου για ποια εντολή πρόκειται. Ο κύκλος αυτός επαναλαμβάνεται **ο ίδιος**, κάθε φορά που διαβάζεται μία νέα εντολή γλώσσας μηχανής από το τμήμα κώδικα στην μνήμη.

Μπλόκ διαγράμματα

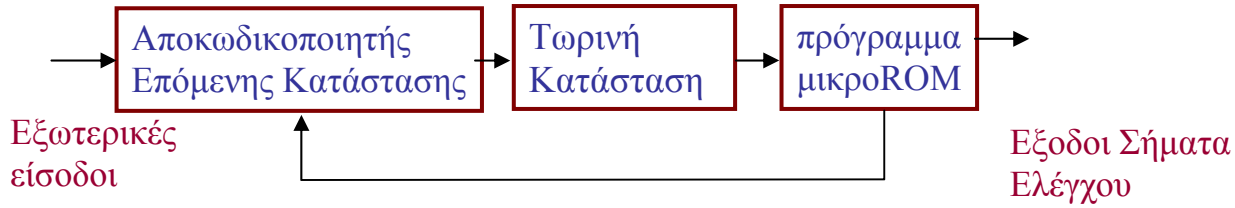
Στη συνέχεια παρατίθενται δύο μπλοκ διαγράμματα που εξηγούν με σχηματικό τρόπο την λειτουργία των δύο τεχνικών υλοποίησης μονάδων ελέγχου: 1. με Ψηφιακά Κυκλώματα, και 2. Με Μικροπρογραμματισμό :



Μπλοκ διάγραμμα μονάδα ελέγχου υλοποιημένης με υλικό (hard-wired) :

Στο διάγραμμα της μονάδας ελέγχου που λειτουργεί με Ψηφιακά Κυκλώματα φαίνεται ο τρόπος με τον οποίο λειτουργεί η μονάδα αυτή. Ανά πάσα στιγμή η μονάδα ελέγχου βρίσκεται σε μια κατάσταση που αναφέρεται στο διάγραμμα ως «τωρινή κατάσταση». Στην κατάσταση αυτή η μονάδα ελέγχου εκτελεί την τρέχουσα μικρό λειτουργία. Η τωρινή κατάσταση περιλαμβάνει επίσης την κατάσταση των καταχωρητών και του flag register.
Η τωρινή κατάσταση ανά τροφοδοτείται σε ένα άλλο κομμάτι του κυκλώματος το οποίο δέχεται και τις εξωτερικές εισόδους. Το κομμάτι αυτό ονομάζεται «αποκωδικοποιητής επόμενης κατάστασης». Είναι υπεύθυνο για την παραγωγή της επόμενης κατάστασης της μονάδας ελέγχου, δηλαδή της μετάβασης στην επόμενη μικρο-λειτουργία που απαιτείται να εκτελεστεί με βάση την προηγούμενη κατάσταση αλλά και τις εξωτερικές εισόδους.
Είσοδοι είναι ο κώδικας της εντολής γλώσσας μηχανής που εκτελείται κάθε στιγμή ή και άλλες εξωτερικές πληροφορίες, όπως για παράδειγμα τα σήματα διακοπών (interrupts).

Ένα τρίτο τμήμα της μονάδας ελέγχου που ονομάζεται «αποκωδικοποιητής εξόδου» είναι υπεύθυνο να παράγει τα κατάλληλα σήματα ελέγχου προς τις διάφορες υπό-μονάδες του επεξεργαστή αποκωδικοποιώντας έτσι την τωρινή κατάσταση σε πολλαπλά σήματα.



Μονάδα ελέγχου υλοποιημένη με μικροκώδικα (microprogrammed) :

Στο διάγραμμα της μονάδας ελέγχου που λειτουργεί με μικροπρογραμματισμό φαίνεται ότι και εδώ υπάρχει η έννοια της «τωρινής κατάστασης» όπου η μονάδα ελέγχου εκτελεί την τρέχουσα μικρό-λειτουργία.
Η τωρινή κατάσταση επίσης ανατροφοδοτείται στον «αποκωδικοποιητή επόμενης κατάστασης» που είναι υπεύθυνος για την παραγωγή της επόμενης «τωρινής κατάστασης» της μονάδας ελέγχου. Ο «αποκωδικοποιητής επόμενης κατάστασης» δέχεται επίσης και τις εξωτερικές εισόδους που είναι ο κώδικας της εντολής γλώσσας μηχανής που εκτελείται κάθε στιγμή αλλά και άλλες εξωτερικές πληροφορίες, όπως για παράδειγμα τα σήματα διακοπών (interrupts).
Η «τωρινή κατάσταση» οδηγεί στην εκτέλεση ενός μικρο-προγράμματος από την μνήμη ROM που περιέχει τις ρουτίνες μικρο-κώδικα. Δεν υπάρχει δηλαδή εδώ το τμήμα του «αποκωδικοποιητή εξόδου». Στις μικροπρογραμματιζόμενες μονάδες ελέγχου το εκάστοτε μικρο-πρόγραμμα είναι υπεύθυνο να πραγματοποιήσει τις κατάλληλες ενέργειες που απαιτούνται για την υλοποίηση της «τωρινής κατάστασης», ενεργοποιώντας τις κατάλληλες υπό-μονάδες του επεξεργαστή και με την κατάλληλη σειρά.

4.3.7.Οι εσωτερικοί δίαυλοι

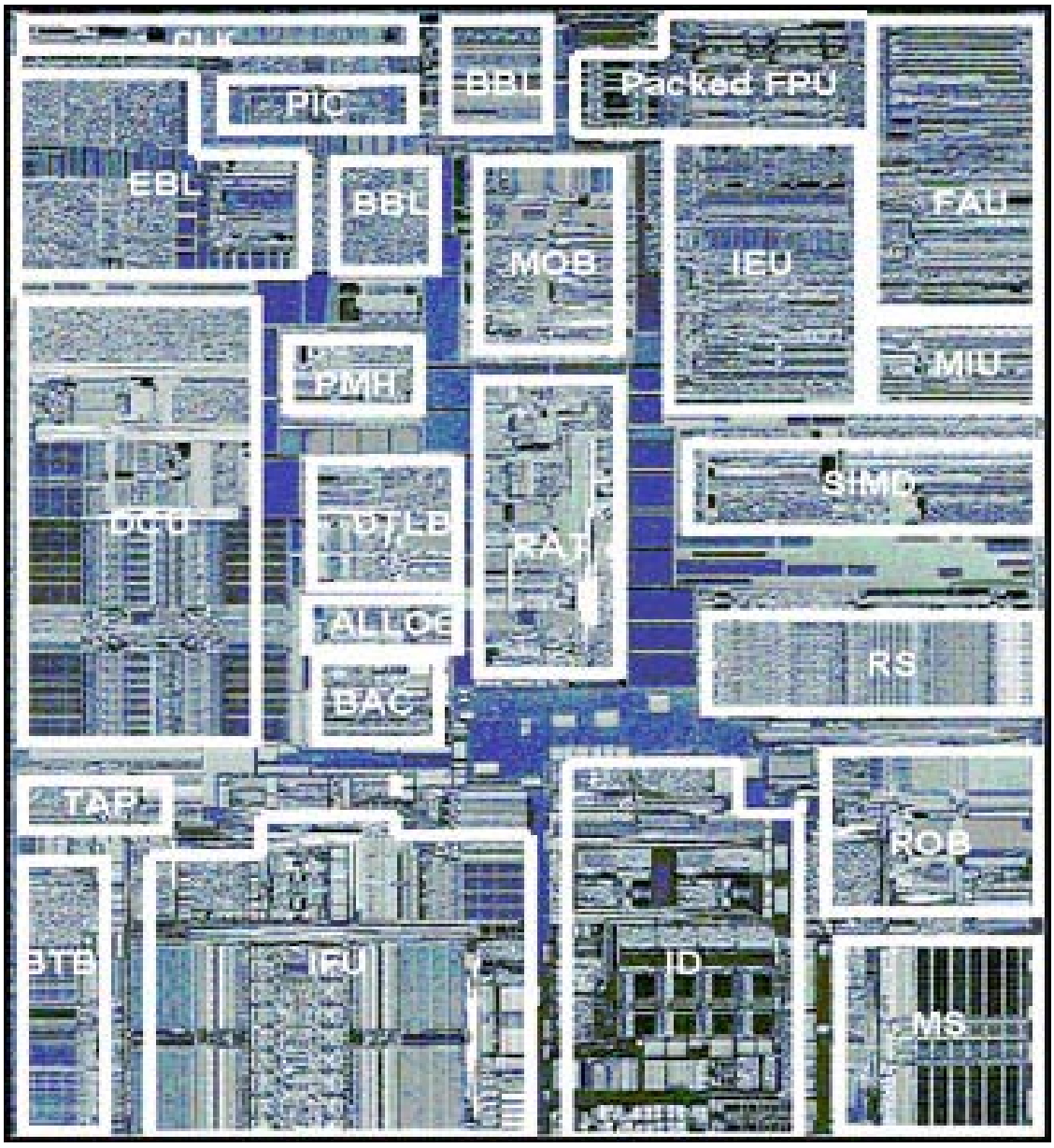
Ένας τυπικός Μ/Ε έχει δύο εσωτερικούς διαύλους :

1. Τον Εσωτερικό Δίαυλο Δεδομένων – Internal Data Bus : που είναι αμφίδρομος και συνδέει τους καταχωρητές μεταξύ τους. Συνδέεται με τον εξωτερικό δίαυλο δεδομένων (external Data Bus) μέσω του εσωτερικού καταχωρητή MDR και εν συνεχεία των ακροδεκτών D0..D7 του chip, ώστε να μεταφέρει δεδομένα προς και από τη μνήμη και τις περιφερειακές συσκευές. Έχει πλάτος συνήθως μία λέξη (όση και η χωρητικότητα των καταχωρητών). Στον 8088 το data bus έχει εύρος 8 bit ενώ οι καταχωρητές είναι των 16 bit. Η σύνδεση με τον εξωτερικό δίαυλο γίνεται μέσω απομονωτή (buffer) τον ρόλο του οποίου παίζει ο εσωτερικός καταχωρητής MDR.
2. Τον Εσωτερικό Δίαυλος Διευθύνσεων – Internal Address Bus : συνδέεται με τους καταχωρητές διευθύνσεων ώστε να σχηματίζεται σε αυτόν η διεύθυνση μνήμης που πρέπει να προσπελαστεί. Συνδέεται στον εξωτερικό δίαυλο διευθύνσεων μέσω του εσωτερικού καταχωρητή MAR και εν συνεχεία των ακροδεκτών A0..A19, ώστε να επιλέγει διευθύνσεις από τη μνήμη RAM, τη μνήμη ROM, ή τις περιφερειακές συσκευές. Στον 8088 το Address Bus έχει εύρος 20 bit. Η σύνδεση με τον εξωτερικό δίαυλο γίνεται μέσω απομονωτή (buffer), τον ρόλο του οποίου παίζει ο εσωτερικός καταχωρητής MAR.

4.4. Τυπικοί μικροεπεξεργαστές

Στη συνέχεια παρατίθενται δύο τυπικοί μικροεπεξεργαστές και οι φωτογραφίες τους με μικροσκόπιο.

Pentium III Katmai

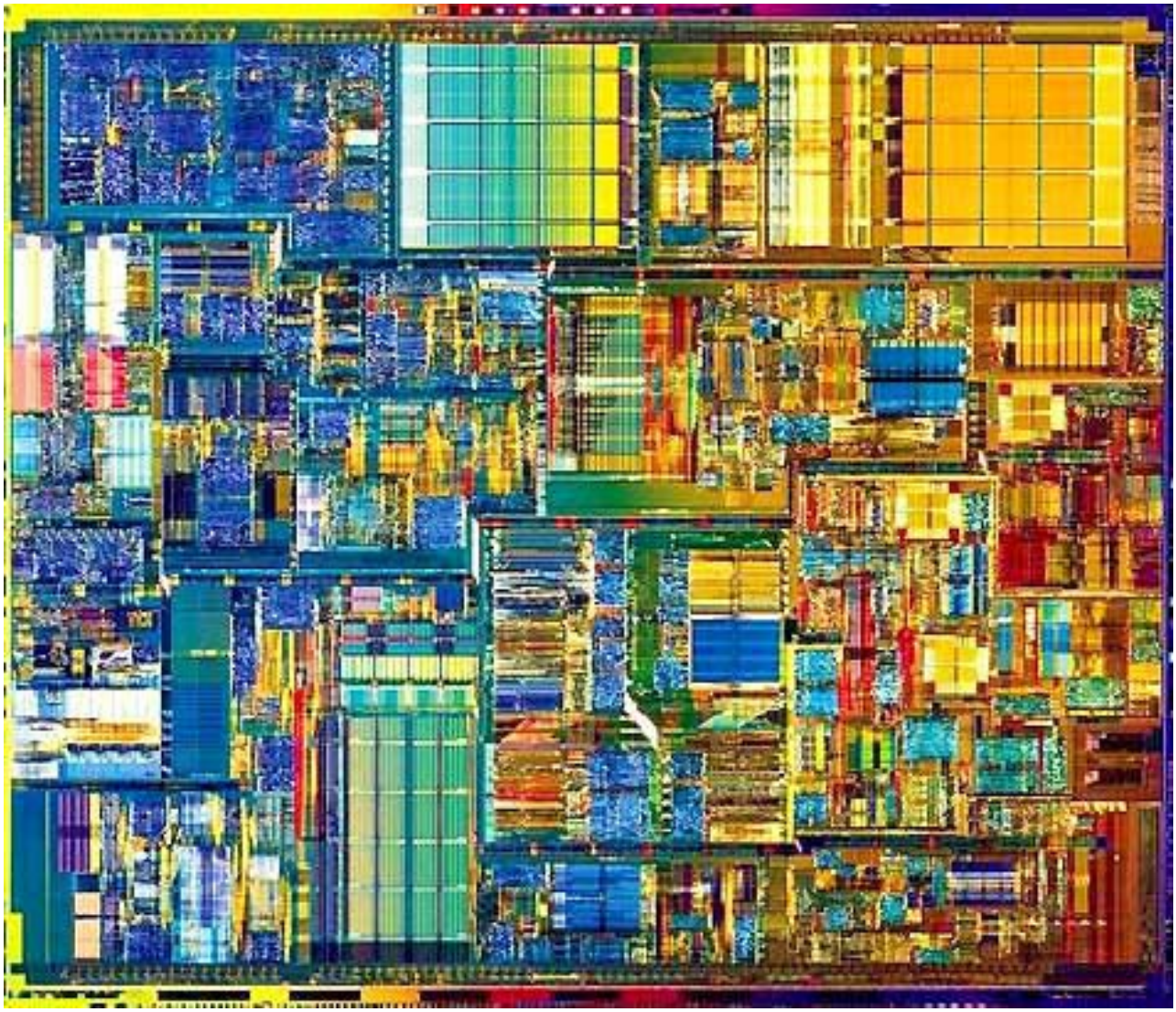


Pentium III, Katmai: 9.5 M transistors, 12.3 x 10.4 mm, 250 nm CMOS with 5 layers of Al

Ο Pentium III, Katmai, έχει 9,5 εκατομμύρια τρανζίστορ. Το μέγεθος του ολοκληρωμένου κυκλώματος είναι 12,3 επί 10,4 χιλιοστά (mm). Βασίζεται σε τεχνολογία CMOS των 250 nm. Οι συνδέσεις μεταξύ των τρανζίστορ αναπτύσσονται σε πέντε επίπεδα και είναι κατασκευασμένες από αλουμίνιο. Η χρήση του αλουμινίου είναι αυτή που δίνει και το ασημο-γαλάζιο χρώμα στη φωτογραφία του μικροεπεξεργαστή.
Στη συνέχεια επεξηγούνται οι συντομογραφίες των τμημάτων του επεξεργαστή που φαίνονται στην εικόνα.

- EBL/BBL - Bus logic, Front, Back
- MOB - Memory Order Buffer
- Packed FPU - MMX Fl. Pt. (SSE)
- IEU - Integer Execution Unit
- FAU - Fl. Pt. Arithmetic Unit
- MIU - Memory Interface Unit
- DCU - Data Cache Unit
- PMH - Page Miss Handler
- DTLB - Data TLB
- BAC - Branch Address Calculator
- RAT - Register Alias Table
- SIMD - Packed Fl. Pt.
- RS - Reservation Station
- BTB - Branch Target Buffer
- IFU - Instruction Fetch Unit (+I$)
- ID - Instruction Decode
- ROB - Reorder Buffer
- MS - Micro-instruction Sequencer

Pentium 4



- 55M xistors
  - PIII: 26M
- 217 mm2
  - PIII: 106 mm2
- L1 Execution Cache
  - Buffer 12,000 Micro-Ops
- 8KB data cache
- 256KB L2$

Ο Pentium 4 είναι ένα ολοκληρωμένο κύκλωμα κατασκευασμένο με 55 εκατομμύρια τρανζίστορ. Η επιφάνεια του ολοκληρωμένου κυκλώματος ανέρχεται σε 217 τετραγωνικά χιλιοστά (mm^2). Η μνήμη cache επιπέδου ένα (level 1 cache) είναι χωρισμένη σε δύο κομμάτια (split cache) : στη μνήμη cache εκτέλεσης εντολών (execution cache) που έχει χωρητικότητα 12.000 μικρό-εντολές και στη μνήμη cache δεδομένων (data cache) η οποία ανέρχεται σε 8 KByte. Η μνήμη cache επιπέδου δύο είναι κοινή και για εντολές και για δεδομένα και ανέρχεται σε 256 KByte.

4.4. Κριτήρια Επιλογής Μκροεπεξεργαστή

Τα κριτήρια που θα πρέπει να λαμβάνονται υπ όψιν κατά την επιλογή ενός μικροεπεξεργαστή για την κατασκευή ενός συστήματος μπορούν να αναλυθούν ως εξής :

1. Απόδοση. Ο πιο ουσιαστικός παράγοντας κατά την επιλογή ενός μ/ε είναι το εάν ο υποψήφιος μ/ε παρέχει ικανοποιητική απόδοση στην εφαρμογή που θα χρησιμοποιηθεί. Για την μέτρηση της απόδοσης μ/ε χρησιμοποιούνται τυποποιημένα Μετροπρογράμματα (benchmarks) τα οποία εκτελούν συνήθως επαναληπτικές λειτουργίες και παρέχουν ένα μέτρο σύγκρισης της απόδοσης διαφορετικών μ/ε.
2. Αριθμός μονάδων που θα παραχθούν. Αποτελεί ένα ουσιαστικό κριτήριο που έχει να κάνει με το κόστος. Γενικά, για παραγωγή μικρών ή μεσαίων ποσοτήτων είναι οικονομικότερη η χρήση ενός τυποποιημένου μ/ε. Για παραγωγή μεγάλης κλίμακας είναι πιθανόν οικονομικότερη η χρήση ενός μ/ε κατά-παραγγελία, ο οποίος θα είναι βελτιστοποιημένος για την συγκεκριμένη εφαρμογή.
3. Διαθεσιμότητα. Σημαντικό κριτήριο κατά την επιλογή μ/ε είναι η διαθεσιμότητα όλων των υποσυστημάτων που απαιτούνται για τη λειτουργία του συστήματος, όπως chipset υποστήριξης, chip μνήμης και I/O κ.α.
4. Λογισμικό. Το υλικό μέρος του συστήματος δεν μπορεί να λειτουργήσει χωρίς το κατάλληλο λογισμικό. Κατά την επιλογή του μ/ε θα πρέπει να εξεταστεί εάν υπάρχουν διαθέσιμες έτοιμες εφαρμογές ή assemblers / compilers για την ανάπτυξη προγραμμάτων.
5. Εργαλεία Ανάπτυξης. Η σχεδίαση του συστήματος και η ανάπτυξη του λογισμικού διευκολύνεται και επιταχύνεται εάν είναι διαθέσιμα διάφορα εργαλεία ανάπτυξης όπως Αναπτυξιακά Συστήματα, emulators του μ/ε κ.α.
6. Ειδικά Κριτήρια. Η επιλογή μ/ε είναι πιθανόν να περιορίζεται από ειδικά κριτήρια. Ετσι, η χαμηλή κατανάλωση παίζει σημαντικό ρόλο σε φορητά συστήματα, ενώ για στρατιωτική χρήση ο μ/ε θα πρέπει να πληροί συγκεκριμένες και αυστηρές προδιαγραφές (MIL SPEC)

4.5. Μέτρηση της απόδοσης ενός Μικροεπεξεργαστή

Για την μέτρηση της απόδοσης και τη σύγκριση διαφορετικών μ/ε υπάρχουν ορισμένα διεθνώς αποδεκτά Μετροπρογράμματα (Benchmarks) και μεγέθη, τα οποία αναλύονται πιο κάτω:

- **MIPS** (Million Instructions Per Second). Είναι η παλιότερη μονάδα μέτρησης ταχύτητας μιας CPU και ισοδυναμεί περίπου με τον αριθμό εντολών μηχανής που εκτελούνται σε ένα δευτερόλεπτο (second). Τα MIPS αναφέρονται καθαρά στην απόδοση της CPU και έτσι δεν είναι αντιπροσωπευτικά της απόδοσης ενός πλήρους συστήματος, καθώς εκεί παράγοντες όπως η μνήμη και οι περιφερειακές μονάδες επηρεάζουν σημαντικά την τελική απόδοση του. Χονδρικά, οι Intel 8088/86 απέδιδαν 0.25 MIPS ενώ οι Pentium ξεπερνούν τα 100 MIPS.
- **FLOPS** (Floating-Point Operations per Second). Υπολογίζει τον αριθμό πράξεων κινητής υποδιαστολής ανά sec. Οι πράξεις αυτές είναι πιο αργές από τις αντίστοιχες πράξεις ακεραίων. Οι σύγχρονες CPU έχουν μια ειδική μονάδα για την εκτέλεση τέτοιων πράξεων (FPU, Floating Point Unit) και τα FLOPS δίνουν την απόδοση αυτής της μονάδας. Από πολλούς τα FLOPS δεν θεωρούνται πλήρως αντιπροσωπευτικά της ταχύτητας μιας CPU καθώς δεν λαμβάνονται υπόψη παράγοντες όπως ο φόρτος της CPU, το είδος της πράξης κ.α. Πολλαπλάσια του FLOP είναι το MFLOP και το GFLOP.

Ένα γνωστό μετροπρόγραμμα που υπολογίζει τα FLOPS είναι το Linpack. Η απόδοση ενός υπερυπολογιστή CRAY-1 είναι περίπου 100 MFLOPS.

- **Dhrystone**. Είναι ένα μετροπρόγραμμα που αναπτύχθηκε το 1984, είναι γραμμένο σε μια γλώσσα ανώτερου επιπέδου και υπολογίζει την απόδοση της CPU σε πράξεις ακεραίων, χωρίς να εκτελεί λειτουργίες I/O ή κλήσεις στο λειτουργικό σύστημα. Τα Dhrystones per second μετρούν το πόσες φορές μπορεί να εκτελεστεί ένα πρόγραμμα σε ένα sec.
- **SPEC** (Standard Performance Evaluation Corporation). Είναι ένας μη κερδοσκοπικός οργανισμός που συγκροτήθηκε από εταιρίες κατασκευής υπολογιστών και μ/ε με σκοπό την ανάπτυξη ενός τυποποιημένου συνόλου μετροπρογραμμάτων για τη βιομηχανία υπολογιστών. Τα Μετροπρογράμματα αυτά υπολογίζουν την απόδοση της CPU σε πράξεις ακεραίων και κινητής υποδιαστολής, καθώς και την απόδοση της μνήμης, των μεταγλωττιστών κ.α.
- Τέλος, έχει αναπτυχθεί ένας μεγάλος αριθμός μετροπρογραμμάτων από διάφορες εταιρίες και περιοδικά πληροφορικής ανά τον κόσμο. Είναι προφανές ότι δεν μπορεί να γίνει σύγκριση αποτελεσμάτων μεταξύ διαφορετικών μετροπρογραμμάτων.

Κεφάλαιο 5. Η δομή των εντολών

Στο κεφάλαιο αυτό θα μελετήσουμε τις εντολές γλώσσας μηχανής. Θα αναλύσουμε τη δομή των εντολών γλώσσας μηχανής και τα τμήματα από τα οποία αποτελούνται. Θα εμβαθύνουμε στον τρόπο με τον οποίον διαβάζονται οι εντολές γλώσσας μηχανής από τον επεξεργαστή, και στον τρόπο με τον οποίον αποκωδικοποιούνται, ώστε να γίνει αντιληπτό από τον επεξεργαστή για ποια εντολή πρόκειται, ώστε να προβεί στην εκτέλεση των σωστών βημάτων για την ολοκλήρωση της εντολής.
Επίσης θα αναλύσουμε τις εντολές στο επίπεδο μικρό-κώδικα που εκτελούν οι μικρο-προγραμματιζόμενες μονάδες ελέγχου για την ολοκλήρωση των εντολών. Θα αναλύσουμε γνωστές εντολές γλώσσας μηχανής του 8088 σε εντολές μικρό-κώδικα. Επίσης θα μιλήσουμε για τους διάφορους τρόπους σύνταξης των εντολών γλώσσας μηχανής που αναφέρονται και ως τρόποι διευθυνσιοδότησης μνήμης, και το πώς αυτοί λειτουργούν. Τέλος θα μιλήσουμε για δύο αρχιτεκτονικές κατασκευής επεξεργαστών την τεχνολογία CISC και την τεχνολογία RISC.

5.1. Οι εντολές γλώσσας μηχανής

Κάθε μικρο-επεξεργαστής διαθέτει ένα σετ εντολών που μπορεί να αναγνωρίζει και να εκτελεί. Με βάση αυτό το σέτ, μπορεί ο προγραμματιστής να αναπτύξει οποιοδήποτε πρόγραμμα οσοδήποτε πολύπλοκο και να είναι. Το σετ εντολών του επεξεργαστή απαρτίζει στην ουσία μια γλώσσα προγραμματισμού πολύ χαμηλού επιπέδου, στην οποία μεταφράζονται τα προγράμματα που γράφονται σε όλες τις ανωτέρου επιπέδου γλώσσες, όπως η C η PASCAL η BASIC η JAVA κ.λ.π.

Η γλώσσα αυτή εμφανίζεται σε δύο μορφές:

1. την **Συμβολική Γλώσσα Μηχανής (Assembly Language)**, που είναι μια μορφή των εντολών εύκολα αντιληπτή από τον άνθρωπο και που μπορεί ο προγραμματιστής να απομνημόνευσει εύκολα (π.χ. MOV AX,[1234]), και
2. τον **Κώδικα Μηχανής (Machine Code)** που αποτελεί την αντιστοίχιση των συμβολικών εντολών σε αριθμούς (bytes) τους οποίους και μπορεί να αναγνωρίσει και να εκτελέσει τελικά ο επεξεργαστής (π.χ. MOV AX,[1234] αντιστοιχεί σε «B8 34 12»).

Τα προγράμματα χαμηλού επιπέδου αναπτύσσονται σε συμβολική γλώσσα γιατί είναι ευκολότερη στην ανάπτυξη προγραμμάτων. Η μετατροπή των συμβολικών εντολών που γράφει ο προγραμματιστής στους αντίστοιχους αριθμούς του κώδικα μηχανής γίνεται με ένα βοηθητικό πρόγραμμα που πρέπει να υπάρχει στον υπολογιστή και που ονομάζεται **Συμβολομεταφραστής** ή **Assembler**. Η ύπαρξη Assembler μας διευκολύνει στο ότι μπορούμε να προγραμματίζουμε σε συμβολική γλώσσα και όχι απευθείας με αριθμούς του κώδικα μηχανής. Ο σύμβολο-μεταφραστής μεταφράζει κάθε συμβολική εντολή στον κατάλληλο αριθμό bytes που αντιστοιχεί, και τοποθετεί τα bytes αυτά με τη σειρά στη μνήμη. Στη συνέχεια παίρνει την επομένη εντολή, την μεταφράζει στα αντίστοιχα bytes της, και τα τοποθετεί αμέσως μετά από τα πρώτα στη μνήμη. Έτσι κατά την διαδικασία της συμβολομετάφρασης, στη μνήμη χτίζεται μια ακολουθία από bytes που αντιστοιχούν σε εντολές εκτελέσιμες από τον επεξεργαστή. Όταν οι εντολές αυτές διαβαστούν και εκτελεστούν με τη σειρά, θα αποτελέσουν ένα ολοκληρωμένο πρόγραμμα με συγκεκριμένη λογική και αποτελέσματα.

Κάθε πρόγραμμα που εκτελείται σε έναν υπολογιστή, είτε αυτός έχει λειτουργικό σύστημα Windows, είτε αυτός έχει λειτουργικό σύστημα Linux, ή οποιοδήποτε άλλο λειτουργικό σύστημα, και ανεξάρτητα από τη γλώσσα προγραμματισμού στην οποία αναπτύχθηκε η εφαρμογή, είτε αυτή ήταν απλή διαδικαστική γλώσσα, είτε ήταν αντικειμενοστραφής, είτε ήταν γλώσσα οπτικού προγραμματισμού, αποτελείται από εντολές κώδικα μηχανής που αναγνωρίζει ο επεξεργαστής.

Οι γλώσσες προγραμματισμού ανώτερου επιπέδου είναι μια εφεύρεση που μας επιτρέπει να προγραμματίζουμε τους υπολογιστές με πιο ανθρώπινο τρόπο από ότι ο «μηχανιστικός» και «κωδικοποιημένος» κώδικας μηχανής, να φτιάχνουμε ευκολότερα μεγαλύτερα προγράμματα και σε συντομότερο χρόνο, και να μπορούμε να οργανώνουμε καλύτερα και να ξανα-χρησιμοποιούμε τα προγράμματα που φτιάχνουμε. Οποιοδήποτε όμως πρόγραμμα γραμμένο σε οποιαδήποτε γλώσσα, οποιασδήποτε τεχνοτροπίας, για να εκτελεστεί σε έναν υπολογιστή θα πρέπει πρώτα να μεταφραστεί σε κώδικα μηχανής, του επεξεργαστή που φιλοξενεί ο υπολογιστής. Αυτό που εκτελείται στην ουσία είναι οι εντολές κώδικα μηχανής.

Κάθε εντολή γλώσσας μηχανής της Κεντρικής Μονάδας Επεξεργασίας (ΚΜΕ) αποτελείται από 1 έως N bytes ανάλογα με τον επεξεργαστή. Στον 8088 οι εντολές γλώσσας μηχανής αποτελούνται από 1 έως 6 bytes.

Το πρώτο ή τα 2 πρώτα bytes της κάθε εντολής περιέχουν τον κωδικό της εντολής (Opcode). Το opcode είναι ο χαρακτηριστικός κωδικός αριθμός κάθε εντολής που αναγνωρίζει ο επεξεργαστής. Κάθε εντολή του κώδικα μηχανής διακρίνεται απόλυτα από το opcode της. Όταν δηλαδή ο επεξεργαστής διαβάσει το opcode μίας εντολής, τότε καταλαβαίνει απόλυτα για ποια εντολή πρόκειται. Επίσης καταλαβαίνει αν αυτή η εντολή ακολουθείται από παραμέτρους, πόσα byte είναι οι παράμετροι και ποια είναι η σημασία κάθε παραμέτρου (αριθμός, διεύθυνση μνήμης, κ.λ.π.).

Κάθε εντολή, στην συμβολική της μορφή, όπως για παράδειγμα η εντολή MOV, μπορεί να έχει ένα πλήθος από διαφορετικά opcodes ανάλογα με τα είδη διευθυνσιοδότησης που συντάσσεται. Για να γίνει αυτό αντιληπτό παραθέτουμε 5 διαφορετικούς τρόπους σύνταξης της εντολής MOV καθώς και τα opcodes και τις παραμέτρους της καθεμιάς.

Εντολή	**Κώδικας μηχανής**	**Πλήθος byte**	**Opcode**	**Παράμετρος**	**Λειτουργία**
MOV AL,FF	B0 FF	2	B0	FF	Βάζει στον AL τον αριθμό FF_{16} (255_{10})
MOV BL,FF	B3 FC	2	B3	FC	Βάζει στον BL τον αριθμό FC_{16} (252_{10})
MOV AL,[1234]	A0 34 12	3	A0	34 12 → 1234	Βάζει στον AL τον αριθμό (1 byte) που υπάρχει στην διεύθυνση μνήμης DS:1234_{16}
MOV AX,1234	B8 34 12	3	B8	34 12 → 1234	Βάζει στον AX τον αριθμό 1234_{16} (4660_{10})
MOV DX,5678	BA 78 56	3	BA	78 56 → 5678	Βάζει στον DX τον αριθμό 5678_{16} (22136_{10})
MOV AX,[2345]	A1 45 34	3	A1	45 23 → 2345	Βάζει στον AX τον αριθμό (2 byte) που

					υπάρχει στην διεύθυνση μνήμης $DS:2345_{16}$
MOV CX,[2345]	8B 0E 45 23	4	8B 0E	45 23 → 2345	Βάζει στον CX τον αριθμό (2 byte) που υπάρχει στην διεύθυνση μνήμης $DS:2345_{16}$
MOV BX,[2345+SI]	8B 9C 45 23	4	8B 9C	45 23 → 2345	Βάζει στον BX τον αριθμό (2 byte) που υπάρχει στην διεύθυνση μνήμης DS: (2345_{16}+SI)
MOV [3456],ABCD	C7 06 56 34 CD AB	6	C7 06	56 34 → 3456 CDAB→ABCD	Βάζει στην διεύθυνση μνήμης [3456] τον αριθμό ABCD

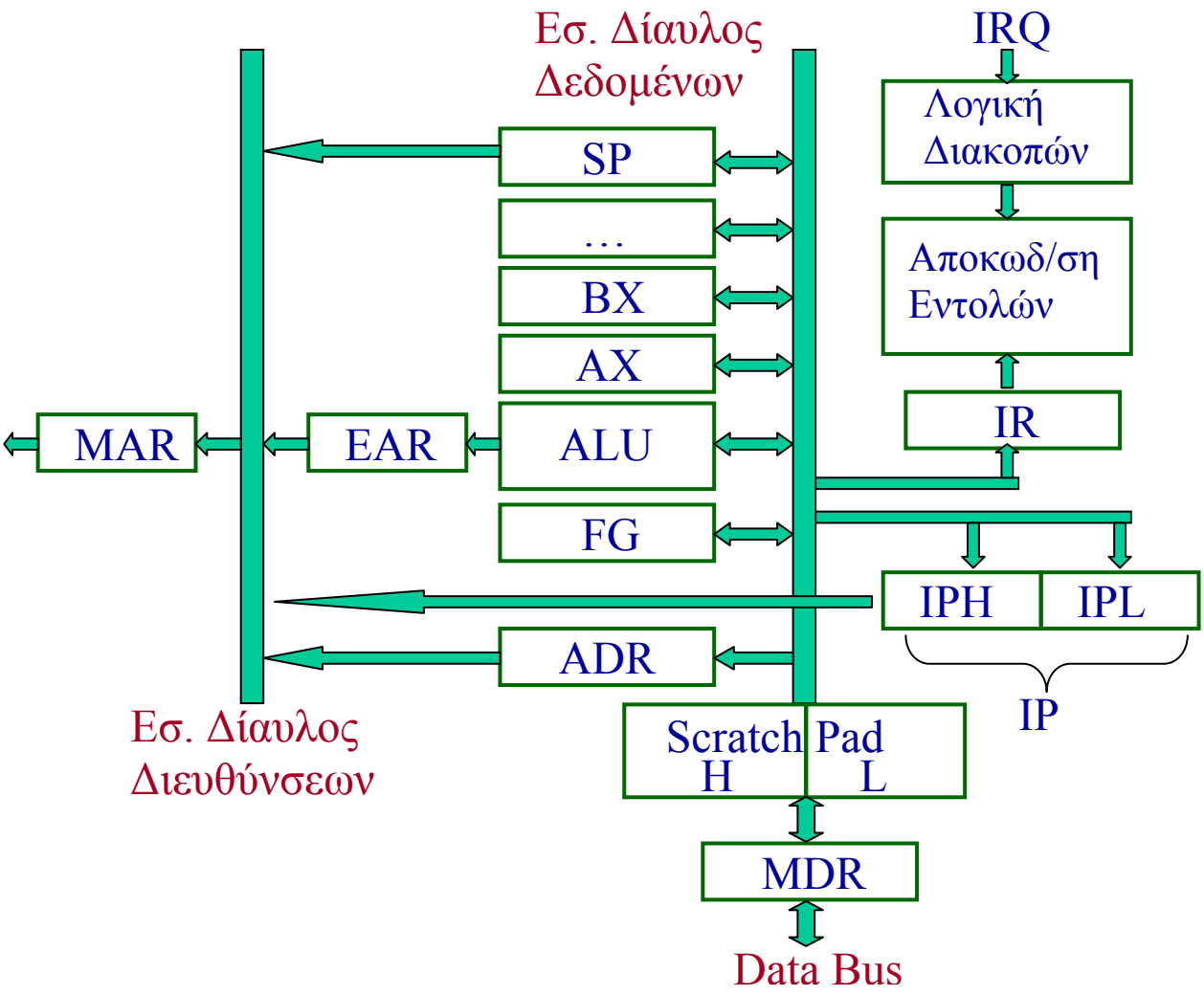
Τα επόμενα bytes μετά το opcode (αν υπάρχουν) περιέχουν παραμέτρους της εντολής (αριθμούς ή διευθύνσεις) και ονομάζονται τελεσταίοι (operands). Ακολουθεί ένας πίνακας με τρία παραδείγματα εντολών γλώσσας μηχανής που αποτελούνται από διαφορετικό αριθμό Bytes η κάθε μία.

Συμβολική Εντολή	**Κώδικας Μηχανής**	**Συμβολική Εντολή**	**Κώδικας Μηχανής**	**Συμβολική Εντολή**	**Κώδικας Μηχανής**
CLC	F8	INT 85	CD	ADD AX,1234	05
		(αριθμός)	85	(low byte)	34
				(high byte)	12

Όσα περισσότερα byte έχει μία εντολή τόσο περισσότερο χρόνο χρειάζεται για να εκτελεστεί. Αυτό γίνεται κατανοητό γιατί όταν μια εντολή αποτελείται από μεγάλο πλήθος bytes, τότε θα απαιτηθεί πολύς χρόνος μόνο και μόνο για να μπορέσει να προσκομιστεί η εντολή αυτή μέσα στον επεξεργαστή. Είναι προφανές ότι μια εντολή των 4 bytes χρειάζεται διπλάσιο χρόνο για την προσκόμισή της στον επεξεργαστή από ότι μία εντολή των 2 bytes. Επίσης, συνήθως οι εντολές των πολλών bytes είναι σύνθετες εντολές οι οποίες ακόμα και στο στάδιο της εκτέλεσής τους απαιτούν παραπάνω χρόνο από ότι οι απλούστερες εντολές.

5.2. Η εκτέλεση των εντολών γλώσσας μηχανής

Στο κεφάλαιο αυτό θα αναπτύξουμε τον τρόπο με τον οποίον γίνεται η εκτέλεση των εντολών γλώσσας μηχανής μέσα στον επεξεργαστή. Για την καλύτερη κατανόηση των εννοιών και για να έχουμε καλύτερη εποπτική εικόνα του επεξεργαστή, παραθέτουμε ακριβώς από κάτω ένα μπλοκ διάγραμμα του εσωτερικού του επεξεργαστή όπου φαίνονται οι, πολύ σημαντικοί για τη διαδικασία αυτή, εσωτερικοί καταχωρητές.



Κάθε εντολή της ΚΜΕ εκτελείται σε 3 φάσεις :

1. **Προσκόμιση** (fetch)
2. **Αποκωδικοποίηση** (Decode)
3. **Εκτέλεση** (Execute)

Τα 3 αυτά βήματα αναλύονται στις ακόλουθες ενέργειες:

1. Προσκόμιση της επόμενης εντολής από την μνήμη (από την διεύθυνση που «δείχνει» ο δείκτης IP). Αποθήκευση του opcode της εντολής (1-2 byte) στον καταχωρητή εντολών (IR).
2. Αλλαγή του IP ώστε να δείχνει την επόμενη εντολή.
3. Προσδιορισμός του τύπου της εντολής από τη Μονάδα Ελέγχου.
4. Διάβασμα των τυχόν παραμέτρων της εντολής (αριθμός ή διεύθυνση) και αποθήκευση στους καταλληλους καταχωρητές. Στην περίπτωση διεύθυνσης (παράμετρος εντολής) αυτή καταχωρείται στον καταχωρητή ADR.
5. Υπολογισμός της τελικής διεύθυνσης δεδομένων (έμμεση ή δεικτοδοτούμενη διευθυνσιοδότηση).
6. Ανάκτηση των τελικών δεδομένων από την μνήμη.
7. Εκτέλεση της εντολής.
8. Αποθήκευση των αποτελεσμάτων στην κατάλληλη θέση (καταχωρητής ή διεύθυνση μνήμης.

Τα παραπάνω βήματα αναλύονται περισσότερο ακολούθως:

5.2.1. Προσκόμιση της εντολής

- **Βήμα 1 : προσκόμιση της εντολής (fetch).**
  1. Η επόμενη εντολή βρίσκεται στην διεύθυνση που δείχνει ο IP.
  2. Το περιεχόμενο του IP μεταφέρεται στον καταχωρητή διεύθυνσης μνήμης (MAR) συνδυαζόμενο με τον καταχωρητή τμήματος CS (MAR=CS:IP=CSx16+IP), και από εκεί στον εξωτερικό δίαυλο διευθύνσεων (Address Bus).
  3. Η διεύθυνση του IP αυξάνεται κατά 1 ώστε να δείχνει στο επόμενο byte (operands ή επόμενο opcode). Η ενημέρωση του IP γίνεται μετά από κάθε ανάγνωση μνήμης μέσω του IP (CS:IP).
  4. Μετά από μία χρονική καθυστέρηση (χρόνος προσπέλασης μνήμης) το πρώτο byte της εντολής (opcode) εμφανίζεται στον εξωτερικό δίαυλο δεδομένων (Data Bus)
  5. Τα δεδομένα του Data Bus αποθηκεύονται στον καταχωρητή δεδομένων (MDR).
  6. Το περιεχόμενο του MDR μεταφέρεται στον καταχωρητή εντολής (IR).
  7. Αν η εντολή περιέχει 2 opcodes τότε τα βήματα 2-6 επαναλαμβάνονται άλλη μία φορά.

Στο παρακάτω σχήμα φαίνεται η «εικόνα» που έχει στην μνήμη μία εντολή γλώσσας μηχανής με 2 opcode και 2 byte παραμέτρων, τοποθετημένη στην διεύθυνση μνήμης 0100:0000. Ο IP ήδη έχει την τιμή 0000 ενώ ο CS την τιμή 0100, δηλαδή «δείχνει» σε αυτή την εντολή και είναι έτοιμη να αναγνωστεί και να εκτελεστεί :

Διευθύνσεις	**Κώδικας Μηχανής**	
0100:0005	...	
0100:0004	*(επόμενο opcode)*	
0100:0003	*(parameter 2)*	
0100:0002	*(parameter 1)*	
0100:0001	*(opcode 2)*	
0100:0000	*(opcode 1)*	← IP (IP=0000, CS=0100)

Εικόνα στην μνήμη μίας εντολής γλώσσας μηχανής με 2 opcode και 2 byte παραμέτρων, τοποθετημένη στην διεύθυνση μνήμης 0100:0000.

Τα παραπάνω βήματα αναλύονται παρακάτω με συμβολικές εντολές μικροπρογράμματος:

Α/Α	**Συμβολική Μικρο-εντολή**	**Σημασία**
1	MAR ← IP	Το περιεχόμενο του IP αντιγράφεται στον MAR συνδυαζόμενο με τον καταχωρητή τμήματος CS (MAR=CS:IP=CSx16+IP
2	IP ← IP + 1	Ο IP αυξάνεται κατά 1 ώστε να δείχνει στο επόμενο byte που ανάλογα με την εντολή μπορεί να είναι το δεύτερο byte του opcode, ή το πρώτο byte των παραμέτρων ή , αν δεν υπάρχουν παράμετροι, το opcode της επόμενης εντολής.
3	Wait	Αναμονή για να απαντήσει η μνήμη. Ο χρόνος αυτός

		ταυτίζεται ή είναι μεγαλύτερος από τον χρόνο απόκρισης της μνήμης.
4	MDR ← DB	Η λέξη (1 byte) που εμφανίζεται στον δίαυλο δεδομένων (Data Bus) αντιγράφεται στον MDR
5	IR ← MDR	Το περιεχόμενο του MDR αντιγράφεται στον IR (Instruction Register) καθώς το byte που διαβάστηκε αποτελεί ένα opcode που πρέπει να προωθηθεί στην μονάδα ελέγχου για αναγνώριση και αποκωδικοποίηση.

5.2.2. Αποκωδικοποίηση της εντολής

- **Βήμα 2 : αποκωδικοποίηση της εντολής (opcode decoding).**
  1. Η μονάδα ελέγχου αποκωδικοποιεί την εντολή, είτε με συνδυαστικά ψηφιακά κυκλώματα (hard-wired control unit) είτε με μικροπρόγραμμα (microprogrammed control unit), αναγνωρίζει για ποια εντολή πρόκειται, και αποφασίζει ποιά τμήματα του Μ/Ε θα ενεργοποιηθούν και με ποια σειρά.
  2. Σε αυτό το στάδιο η μονάδα ελέγχου καταλαβαίνει εάν στην εντολή περιλαμβάνεται και δεύτερο opcode ή όχι, εάν υπάρχουν παράμετροι ή όχι, από πόσα byte αποτελούνται οι παράμετροι και ποιος είναι ο ρόλος της κάθε παραμέτρου που ακολουθεί το opcode.
  3. Αν κατά την αποκωδικοποίηση της εντολής (δηλαδή από το πρώτο opcode) διαπιστωθεί ότι υπάρχει και δεύτερο, τότε ακολουθεί η ανάγνωση του και η αποθήκευσή του στο καταχωρητή IR. Αυτό θα γίνει με επανάληψη των εντολών που αναφέρονται στο βήμα 1, με την διαφορά ότι το δεύτερο opcode πηγαίνει στον IR_H , δηλαδή στο υψηλής τάξης byte του IR. Πρέπει εδώ να σημειωθεί ότι το αν μία εντολή έχει και δεύτερο opcode ή όχι καθορίζεται απόλυτα από το πρώτο opcode. Για παράδειγμα το opcode A1 (MOV AX,[####]) ΔΕΝ ακολουθείται από δεύτερο. Ενώ το opcode 8B (MOV ##,[####]) ακολουθείται πάντα και από δεύτερο. Έτσι αν η Μονάδα Ελέγχου διαβάσει το opcode A1, καταλαβαίνει ότι δεν υπάρχει 2^o opcode αλλά ότι ακολουθεί παράμετρος διεύθυνσης των 2 bytes. Ενώ αν διαβάσει το opcode 8B καταλαβαίνει ότι υπάρχει και 2^o opcode που προσδιορίζει επακριβώς τον καταχωρητή που συμμετέχει στην εντολή, καθώς και τον τρόπο διευθυνσιοδότησης, και ότι μετέπειτα ακολουθεί και παράμετρος διεύθυνσης των 2 bytes (σύνολο 4 bytes).

5.2.3. Ανάγνωση παραμέτρων - υπολογισμός και ανάγνωση τελικής διεύθυνσης ή και δεδομένων

- **Βήμα 3 : Ανάγνωση παραμέτρων - Υπολογισμός και ανάγνωση τελικής διεύθυνσης ή και δεδομένων.**

Εάν διαπιστωθεί ότι υπάρχουν παράμετροι τότε ακολουθούν ένας ή περισσότεροι κύκλοι ανάγνωσης μνήμης, κατά τους οποίους διαβάζονται από την μνήμη και προσκομίζονται στον επεξεργαστή όλες οι παράμετροι. Οι παράμετροι καταχωρούνται σε κατάλληλους καταχωρητές.

Βήμα 3.1. Αν η **παράμετρος είναι αριθμός του 1 byte** τότε θα παραμείνει στον MDR. Για παράδειγμα στην εντολή MOV AL,45 που αντιστοιχεί στον κώδικα μηχανής B0 45,

μετά το διάβασμα του Opcode (B0) που θα γίνει με τις εντολές του Βήματος 1, θα ακολουθήσει άλλος ένας κύκλος ανάγνωσης μνήμης για το διάβασμα της παραμέτρου.
Στο παρακάτω σχήμα φαίνεται η «εικόνα» που έχει στην μνήμη η εντολή MOV AL,45, τοποθετημένη στην διεύθυνση μνήμης 0100:0000. Ο IP ήδη έχει την τιμή 0000 ενώ ο CS την τιμή 0100, δηλαδή «δείχνει» σε αυτή την εντολή και είναι έτοιμη να αναγνωστεί και να εκτελεστεί :

Διευθύνσεις	Κώδικας Μηχανής	
0100:0002	…	
0100:0001	45	
0100:0000	B0	← IP (IP=0000, CS=0100)

Εικόνα στην μνήμη της εντολής MOV AL,45

Δηλαδή μετά το διάβασμα του opcode (Βήμα 1) θα έχουμε τα εξής βήματα :

1. Το περιεχόμενο του IP μεταφέρεται στον καταχωρητή διεύθυνσης μνήμης (MAR) συνδυαζόμενο με τον καταχωρητή τμήματος CS (MAR=CS:IP=CSx16+IP), και από εκεί στον εξωτερικό δίαυλο διευθύνσεων (Address Bus).
2. Η διεύθυνση του IP αυξάνεται κατά 1 ώστε να δείχνει στο επόμενο byte (operands ή επόμενο opcode). Η ενημέρωση του IP γίνεται μετά από κάθε ανάγνωση μνήμης μέσω του IP (CS:IP).
3. Μετά από μία χρονική καθυστέρηση (χρόνος προσπέλασης μνήμης) το byte της παραμέτρου εμφανίζεται στον εξωτερικό δίαυλο δεδομένων (Data Bus)
4. Τα δεδομένα του Data Bus αποθηκεύονται στον καταχωρητή δεδομένων (MDR). Αναφορικά με το συγκεκριμένο παράδειγμα έχει διαβαστεί ο αριθμός 45.

Τα παραπάνω βήματα αναλύονται παρακάτω με συμβολικές εντολές μικροπρογράμματος:

Α/Α	**Συμβολική Μικρο-εντολή**	**Σημασία**
1	MAR ← IP	Το περιεχόμενο του IP αντιγράφεται στον MAR συνδυαζόμενο με τον καταχωρητή τμήματος CS (MAR=CS:IP=CSx16+IP
2	IP ← IP + 1	Ο IP αυξάνεται κατά 1 ώστε να δείχνει στο επόμενο byte που ανάλογα με την εντολή μπορεί να είναι το δεύτερο byte των παραμέτρων ή το opcode της επόμενης εντολής.
3	Wait	Αναμονή για να απαντήσει η μνήμη. Ο χρόνος αυτός ταυτίζεται ή είναι μεγαλύτερος από τον χρόνο απόκρισης της μνήμης.
4	MDR ← DB	Η λέξη (1 byte) που εμφανίζεται στον δίαυλο δεδομένων (Data Bus) αντιγράφεται στον MDR

Βήμα 3.2. Αν **η παράμετρος είναι αριθμός των 2 bytes** όπως για παράδειγμα στην εντολή SBB AX,1234, που αντιστοιχεί στον κώδικα μηχανής 1D 34 12, τότε μετά το διάβασμα του Opcode (1D) που θα γίνει με τις εντολές του Βήματος 1, θα ακολουθήσουν άλλοι δύο κύκλοι ανάγνωσης μνήμης για το διάβασμα της παραμέτρου, η οποία μέσω του MDR θα καταχωρηθεί στον 16 bit Καταχωρητή Προχείρου (Scratch Pad).

Στο παρακάτω σχήμα φαίνεται η «εικόνα» που έχει στην μνήμη η εντολή SBB AX,1234, τοποθετημένη στην διεύθυνση μνήμης 0100:0000. Ο IP ήδη έχει την τιμή 0000 ενώ ο CS την τιμή 0100, δηλαδή «δείχνει» σε αυτή την εντολή και είναι έτοιμη να αναγνωστεί και να εκτελεστεί :

Διευθύνσεις	Κώδικας Μηχανής	
0100:0003	…	
0100:0002	12	
0100:0001	34	
0100:0000	1D	← IP (IP=0000, CS=0100)

Εικόνα στην μνήμη της εντολής SBB AX,1234

Δηλαδή μετά το διάβασμα του opcode (Βήμα 1) θα έχουμε τα εξής βήματα :

1. Το περιεχόμενο του IP μεταφέρεται στον καταχωρητή διεύθυνσης μνήμης (MAR) συνδυαζόμενο με τον καταχωρητή τμήματος CS (MAR=CS:IP=CSx16+IP), και από εκεί στον εξωτερικό δίαυλο διευθύνσεων (Address Bus).
2. Η διεύθυνση του IP αυξάνεται κατά 1 ώστε να δείχνει στο επόμενο byte (δεύτερο byte της παραμέτρου). Η ενημέρωση του IP γίνεται μετά από κάθε ανάγνωση μνήμης μέσω του IP (CS:IP).
3. Μετά από μία χρονική καθυστέρηση (χρόνος προσπέλασης μνήμης) το byte της παραμέτρου εμφανίζεται στον εξωτερικό δίαυλο δεδομένων (Data Bus)
4. Τα δεδομένα του Data Bus αποθηκεύονται στον καταχωρητή δεδομένων (MDR). Αναφορικά με το συγκεκριμένο παράδειγμα έχει διαβαστεί ο αριθμός 34.
5. Τα δεδομένα του MDR αντιγράφονται στον SCR_L, δηλαδή στο χαμηλής τάξης byte του καταχωρητή προχείρου ScrathPad Register.

Τα παραπάνω βήματα θα επαναληφθούν άλλη μία φορά για να διαβαστεί και το 2^o byte της παραμέτρου:

6. Το περιεχόμενο του IP μεταφέρεται στον καταχωρητή διεύθυνσης μνήμης (MAR) συνδυαζόμενο με τον καταχωρητή τμήματος CS (MAR=CS:IP=CSx16+IP), και από εκεί στον εξωτερικό δίαυλο διευθύνσεων (Address Bus).
7. Η διεύθυνση του IP αυξάνεται κατά 1 ώστε να δείχνει στο επόμενο byte (επόμενο opcode). Η ενημέρωση του IP γίνεται μετά από κάθε ανάγνωση μνήμης μέσω του IP (CS:IP).
8. Μετά από μία χρονική καθυστέρηση (χρόνος προσπέλασης μνήμης) το byte της παραμέτρου εμφανίζεται στον εξωτερικό δίαυλο δεδομένων (Data Bus).
9. Τα δεδομένα του Data Bus αποθηκεύονται στον καταχωρητή δεδομένων (MDR). Αναφορικά με το συγκεκριμένο παράδειγμα έχει διαβαστεί ο αριθμός 12.
10. Τα δεδομένα του MDR αντιγράφονται στον SCR_H, δηλαδή στο υψηλής τάξης byte του καταχωρητή προχείρου ScrathPad Register.

Τα παραπάνω βήματα αναλύονται παρακάτω με συμβολικές εντολές μικροπρογράμματος:

Α/Α	**Συμβολική Μικρο-εντολή**	**Σημασία**
1	MAR ← IP	Το περιεχόμενο του IP αντιγράφεται στον MAR συνδυαζόμενο με τον καταχωρητή τμήματος CS

		(MAR=CS:IP=CSx16+IP
2	IP ← IP + 1	Ο IP αυξάνεται κατά 1 ώστε να δείχνει στο επόμενο byte που είναι το δεύτερο byte των παραμέτρων.
3	Wait	Αναμονή για να απαντήσει η μνήμη. Ο χρόνος αυτός ταυτίζεται ή είναι μεγαλύτερος από τον χρόνο απόκρισης της μνήμης.
4	MDR ← DB	Η λέξη (1 byte) που εμφανίζεται στον δίαυλο δεδομένων (Data Bus) αντιγράφεται στον MDR
5	SCR_L ← MDR	Τα δεδομένα του MDR αντιγράφονται στο χαμηλής τάξης byte του ScratchPad Register

6	MAR ← IP	Το περιεχόμενο του IP αντιγράφεται στον MAR συνδυαζόμενο με τον καταχωρητή τμήματος CS (MAR=CS:IP=CSx16+IP
7	IP ← IP + 1	Ο IP αυξάνεται κατά 1 ώστε να δείχνει στο επόμενο byte που είναι το opcode της επόμενης εντολής.
8	Wait	Αναμονή για να απαντήσει η μνήμη. Ο χρόνος αυτός ταυτίζεται ή είναι μεγαλύτερος από τον χρόνο απόκρισης της μνήμης.
9	MDR ← DB	Η λέξη (1 byte) που εμφανίζεται στον δίαυλο δεδομένων (Data Bus) αντιγράφεται στον MDR
10	SCR_H ← MDR	Τα δεδομένα του MDR αντιγράφονται στο υψηλής τάξης byte του ScratchPad Register

Βήμα 3.3. Αν **η παράμετρος είναι διεύθυνση μνήμης (2 bytes)** όπως για παράδειγμα στην εντολή ADC AX,[4567], που αντιστοιχεί στον κώδικα μηχανής 15 67 45, τότε μετά το διάβασμα του Opcode (15) που θα γίνει με τις εντολές του Βήματος 1, θα ακολουθήσουν άλλοι δύο κύκλοι ανάγνωσης μνήμης για το διάβασμα της παραμέτρου-διεύθυνσης (4567_{16}), η οποία μέσω του MDR θα καταχωρηθεί στον 16 bit Καταχωρητή ADR, που πάντα αποθηκεύει τις διευθύνσεις που είναι παράμετροι των εντολών. Πρώτα θα διαβαστεί το byte χαμηλής τάξης (low byte) και θα πάει στον ADR_L και στη συνέχεια το byte υψηλής τάξης (high byte) και θα τοποθετηθεί στον ADR_H.
Στο παρακάτω σχήμα φαίνεται η «εικόνα» που έχει στην μνήμη η εντολή ADC AX,[4567], τοποθετημένη στην διεύθυνση μνήμης 0100:0000. Ο IP ήδη έχει την τιμή 0000 ενώ ο CS την τιμή 0100, δηλαδή «δείχνει» σε αυτή την εντολή και είναι έτοιμη να αναγνωστεί και να εκτελεστεί :

Διευθύνσεις	Κώδικας Μηχανής	
0100:0003	…	
0100:0002	45	
0100:0001	67	
0100:0000	15	← IP (IP=0000, CS=0100)

Εικόνα στην μνήμη της εντολής ADC AX,[4567]

Δηλαδή μετά το διάβασμα του opcode (Βήμα 1) θα έχουμε τα εξής βήματα :

1. Το περιεχόμενο του IP μεταφέρεται στον καταχωρητή διεύθυνσης μνήμης (MAR) συνδυαζόμενο με τον καταχωρητή τμήματος CS (MAR=CS:IP=CSx16+IP), και από εκεί στον εξωτερικό δίαυλο διευθύνσεων (Address Bus).
2. Η διεύθυνση του IP αυξάνεται κατά 1 ώστε να δείχνει στο επόμενο byte (δεύτερο byte της παραμέτρου). Η ενημέρωση του IP γίνεται μετά από κάθε ανάγνωση μνήμης μέσω του IP (CS:IP).
3. Μετά από μία χρονική καθυστέρηση (χρόνος προσπέλασης μνήμης) το byte της παραμέτρου εμφανίζεται στον εξωτερικό δίαυλο δεδομένων (Data Bus)
4. Τα δεδομένα του Data Bus αποθηκεύονται στον καταχωρητή δεδομένων (MDR). Αναφορικά με το συγκεκριμένο παράδειγμα έχει διαβαστεί ο αριθμός 67.
5. Τα δεδομένα του MDR αντιγράφονται στον ADR_L, δηλαδή στο χαμηλής τάξης byte του καταχωρητή ADR.

Τα παραπάνω βήματα θα επαναληφθούν άλλη μία φορά για να διαβαστεί και το 2^o byte της παραμέτρου:

6. Το περιεχόμενο του IP μεταφέρεται στον καταχωρητή διεύθυνσης μνήμης (MAR) συνδυαζόμενο με τον καταχωρητή τμήματος CS (MAR=CS:IP=CSx16+IP), και από εκεί στον εξωτερικό δίαυλο διευθύνσεων (Address Bus).
7. Η διεύθυνση του IP αυξάνεται κατά 1 ώστε να δείχνει στο επόμενο byte (επόμενο opcode). Η ενημέρωση του IP γίνεται μετά από κάθε ανάγνωση μνήμης μέσω του IP (CS:IP).
8. Μετά από μία χρονική καθυστέρηση (χρόνος προσπέλασης μνήμης) το byte της παραμέτρου εμφανίζεται στον εξωτερικό δίαυλο δεδομένων (Data Bus).
9. Τα δεδομένα του Data Bus αποθηκεύονται στον καταχωρητή δεδομένων (MDR). Αναφορικά με το συγκεκριμένο παράδειγμα έχει διαβαστεί ο αριθμός 45.
10. Τα δεδομένα του MDR αντιγράφονται στον ADR_H, δηλαδή στο υψηλής τάξης byte του καταχωρητή ADR.

Τα παραπάνω βήματα αναλύονται παρακάτω με συμβολικές εντολές μικροπρογράμματος:

Α/Α	**Συμβολική Μικρο-εντολή**	**Σημασία**
1	MAR ← IP	Το περιεχόμενο του IP αντιγράφεται στον MAR συνδυαζόμενο με τον καταχωρητή τμήματος CS (MAR=CS:IP=CSx16+IP
2	IP ← IP + 1	Ο IP αυξάνεται κατά 1 ώστε να δείχνει στο επόμενο byte που είναι το δεύτερο byte των παραμέτρων.
3	Wait	Αναμονή για να απαντήσει η μνήμη. Ο χρόνος αυτός ταυτίζεται ή είναι μεγαλύτερος από τον χρόνο απόκρισης της μνήμης.
4	MDR ← DB	Η λέξη (1 byte) που εμφανίζεται στον δίαυλο δεδομένων (Data Bus) αντιγράφεται στον MDR
5	ADR_L ← MDR	Τα δεδομένα του MDR αντιγράφονται στο χαμηλής τάξης byte του ADR
6	MAR ← IP	Το περιεχόμενο του IP αντιγράφεται στον MAR συνδυαζόμενο με τον καταχωρητή τμήματος CS (MAR=CS:IP=CSx16+IP
7	IP ← IP + 1	Ο IP αυξάνεται κατά 1 ώστε να δείχνει στο επόμενο

		byte που είναι το opcode της επόμενης εντολής.
8	Wait	Αναμονή για να απαντήσει η μνήμη. Ο χρόνος αυτός ταυτίζεται ή είναι μεγαλύτερος από τον χρόνο απόκρισης της μνήμης.
9	MDR ← DB	Η λέξη (1 byte) που εμφανίζεται στον δίαυλο δεδομένων (Data Bus) αντιγράφεται στον MDR
10	ADR_H ← MDR	Τα δεδομένα του MDR αντιγράφονται στο υψηλής τάξης byte του ADR

Βήμα 3.4. Αν απαιτείται και **υπολογισμός τελικής διεύθυνσης** σε περίπτωση **δεικτοδοτούμενης διευθυνσιοδότησης**, για παράδειγμα στην περίπτωση της εντολής MOV AX,[1234+SI], που αντιστοιχεί στον κώδικα μηχανής : 8B 84 34 12, τότε, μετά το διάβασμα των 2 Opcodes (8B 84) που θα γίνει με τις εντολές του Βήματος 1 (2 επαναλήψεις), θα ακολουθήσουν άλλοι δύο κύκλοι ανάγνωσης μνήμης για το διάβασμα της παραμέτρου-διεύθυνσης (1234_{16}) με τις εντολές του Βήματος 3.2. Η διεύθυνση μέσω του MDR θα καταχωρηθεί στον 16 bit Καταχωρητή ADR, που πάντα αποθηκεύει τις διευθύνσεις που είναι παράμετροι των εντολών. Πρώτα θα διαβαστεί το byte χαμηλής τάξης (low byte) και θα πάει στον ADR_L και στη συνέχεια το byte υψηλής τάξης (high byte) και θα τοποθετηθεί στον ADR_H. Η διεύθυνση 1234_{16} είναι η διεύθυνση «βάσης» του πίνακα που προσπελαύνει η εντολή. Ο καταχωρητής SI περιέχει την μετατόπιση (σε bytes) πάνω από την «βάση», που βρίσκεται το στοιχείο που θα προσπελάσουμε.
Επομένως στην συνέχεια θα πρέπει να υπολογιστεί η τελική διεύθυνση μνήμης, προσθέτοντας την διεύθυνση «βάσης» με την «μετατόπιση» δηλαδή, όπως υποδηλώνει και η εντολή, θα πρέπει να γίνει η πράξη :

Τελική Διεύθυνση = «βάση» + «μετατόπιση» = 1234 + SI

Στο παρακάτω σχήμα φαίνεται η «εικόνα» που έχει στην μνήμη η εντολή MOV AX,[1234+SI], τοποθετημένη στην διεύθυνση μνήμης 0100:0000. Ο IP ήδη έχει την τιμή 0000 ενώ ο CS την τιμή 0100, δηλαδή «δείχνει» σε αυτή την εντολή και είναι έτοιμη να αναγνωστεί και να εκτελεστεί :

Διευθύνσεις	Κώδικας Μηχανής	
0100:0004	…	
0100:0003	12	
0100:0002	34	
0100:0001	84	
0100:0000	8B	← IP (IP=0000, CS=0100)

Εικόνα στην μνήμη της εντολής MOV AX,[1234+SI]

Δηλαδή μετά το διάβασμα των opcode (Βήμα 1) και μετά το διάβασμα της 16 bit διεύθυνσης (Βήμα 3.3) θα έχουμε τα εξής βήματα :

1. Πρόσθεση στην ALU του αριθμού που βρίσκεται στον ADR με το περιεχόμενο του καταχωρητή δείκτη που συμμετέχει στην εντολή. Στο συγκεκριμένο παράδειγμα θα προστεθεί ο αριθμός 1234_{16} με τον καταχωρητή SI. Το αποτέλεσμα της πρόσθεσης αποθηκεύεται από την ALU στον καταχωρητή EAR (Effective Address Register), που βρίσκεται στην έξοδο της ALU και αποθηκεύει τις τελικές διευθύνσεις που προκύπτουν μετά από πράξεις.

Τα παραπάνω βήματα αναλύονται παρακάτω με συμβολικές εντολές μικροπρογράμματος:

Α/Α	**Συμβολική Μικρο-εντολή**	**Σημασία**
1	EAR ← ADR+SI	Το περιεχόμενο του ADR προστίθεται με το περιεχόμενο του SI. Η πρόσθεση γίνεται στην ALU. Το αποτέλεσμα καταχωρείται στον EAR.

Βήμα 3.5. Αν απαιτείται και **προσκόμιση τελικής διεύθυνσης** σε περίπτωση **έμμεσης διευθυνσιοδότησης**, για παράδειγμα στην περίπτωση της εντολής CALL [1234], που αντιστοιχεί στον κώδικα μηχανής : FF 16 34 12. Η εντολή αυτή αποτελεί κλήση υπορουτίνας. Πρώτα θα πάει στην διεύθυνση μνήμης 1234, θα διαβάσει από εκεί 2 bytes, τα οποία θα τα εκλάβει ως διεύθυνση και σε αυτή την διεύθυνση θα μεταβεί η εκτέλεση του προγράμματος (έμμεση απόλυτη διευθυνσιοδότηση – Indirect Absolute). Για την εκτέλεση της εντολής, μετά το διάβασμα των 2 Opcodes (FF 16) που θα γίνει με τις εντολές του Βήματος 1 (2 επαναλήψεις), θα ακολουθήσουν άλλοι δύο κύκλοι ανάγνωσης μνήμης για το διάβασμα της παραμέτρου-διεύθυνσης (1234_{16}) με τις εντολές του Βήματος 3.2. Η διεύθυνση μέσω του MDR θα καταχωρηθεί στον 16 bit καταχωρητή ADR, που πάντα αποθηκεύει τις διευθύνσεις που είναι παράμετροι των εντολών. Πρώτα θα διαβαστεί το byte χαμηλής τάξης (low byte) και θα πάει στον ADR_L και στη συνέχεια το byte υψηλής τάξης (high byte) και θα τοποθετηθεί στον ADR_H.
Επομένως στην συνέχεια θα πρέπει να διαβαστεί η τελική διεύθυνση της υπορουτίνας, από την διεύθυνση μνήμης που περιέχει ο ADR. Επομένως θα πρέπει ο ADR να εξαχθεί στον MAR και να διαβαστεί το πρώτο byte ης διεύθυνσης, και ση συνέχεια ο ADR να αυξηθεί κατά 1 και να εξαχθεί και πάλι στον MAR, ώστε να διαβαστεί και το 2^o byte της διεύθυνσης. Η διεύθυνση θα καταχωρηθεί και πάλι στον ADR.

Στο παρακάτω σχήμα φαίνεται η «εικόνα» που έχει στην μνήμη η εντολή CALL [1234], τοποθετημένη στην διεύθυνση μνήμης 0100:0000. Ο IP ήδη έχει την τιμή 0000 ενώ ο CS την τιμή 0100, δηλαδή «δείχνει» σε αυτή την εντολή και είναι έτοιμη να αναγνωστεί και να εκτελεστεί :

Διευθύνσεις	Δεδομένα	
0100:1236	…	**DATA SEGMENT**
0100:1235	56	
0100:1234	78	Τελική διεύθυνση 5678

Διευθύνσεις	Κώδικας Μηχανής	
0100:0004	…	
0100:0003	12	**CODE SEGMENT**
0100:0002	34	
0100:0001	16	
0100:0000	FF	← IP (IP=0000, CS=0100)

Εικόνα στην μνήμη της εντολής CALL [1234]

Δηλαδή μετά το διάβασμα των opcode, δηλαδή των bytes FF 16 που θα γίνει με τις εντολές του Βήματος 1, και μετά το διάβασμα της 16 bit διεύθυνσης, που είναι η 1234, που θα γίνει με τις εντολές του Βήματος 3.3, θα έχουμε τα εξής βήματα :

1. Το περιεχόμενο του ADR μεταφέρεται στον καταχωρητή διεύθυνσης μνήμης (MAR) συνδυαζόμενο με τον καταχωρητή τμήματος DS (MAR=DS:ADR=DSx16+ADR), και από εκεί στον εξωτερικό δίαυλο διευθύνσεων (Address Bus).
2. Μετά από μία χρονική καθυστέρηση (χρόνος προσπέλασης μνήμης) το 1ο byte της διεύθυνσης εμφανίζεται στον εξωτερικό δίαυλο δεδομένων (Data Bus)
3. Τα δεδομένα του Data Bus αποθηκεύονται στον καταχωρητή δεδομένων (MDR). Αναφορικά με το συγκεκριμένο παράδειγμα έχει διαβαστεί το byte που περιέχεται στην διεύθυνση 1234_{16}, και ας υποθέσουμε ότι εκεί υπάρχει ο αριθμός 78.
4. Τα δεδομένα του MDR αντιγράφονται στον $ScratchPad_L$, δηλαδή στο χαμηλής τάξης byte του καταχωρητή προχείρου, καθώς δεν πρέπει να αλλοιωθεί ακόμα η τιμή του ADR, επειδή εκκρεμεί η ανάγνωση και του $2^{ου}$ byte της διεύθυνσης, στην θέση $1234_{16}+1 = 1235_{16}$. Και ο μόνος καταχωρητής που περιέχει την διεύθυνση 1234 είναι ο ADR, οπότε δεν πρέπει να αλλοιωθεί!

Τα παραπάνω βήματα θα επαναληφθούν άλλη μία φορά για να διαβαστεί και το $2^{ο}$ byte της τελικής διεύθυνσης. Το περιεχόμενο του ADR ΔΕΝ θα αυξηθεί κατά 1. Απλά θα γίνει η πράξη ADR+1 στην ALU και το αποτέλεσμα (1235) θα καταχωρηθεί στον EAR:

5. Το περιεχόμενο του ADR προστίθεται με τον αριθμό 1 ώστε να προκύψει η επόμενη διεύθυνση μνήμης, και να διαβαστεί και το $2^{ο}$ byte της τελικής διεύθυνσης. Η πρόσθεση θα γίνει στην ALU. Το αποτέλεσμα θα εξαχθεί στον καταχωρητή EAR.
6. Το περιεχόμενο του EAR μεταφέρεται στον καταχωρητή διεύθυνσης μνήμης (MAR) συνδυαζόμενο με τον καταχωρητή τμήματος DS (MAR=DS:EAR=DSx16+EAR), και από εκεί στον εξωτερικό δίαυλο διευθύνσεων (Address Bus).
7. Μετά από μία χρονική καθυστέρηση (χρόνος προσπέλασης μνήμης) το byte της παραμέτρου εμφανίζεται στον εξωτερικό δίαυλο δεδομένων (Data Bus).
8. Τα δεδομένα του Data Bus αποθηκεύονται στον καταχωρητή δεδομένων (MDR). Αναφορικά με το συγκεκριμένο παράδειγμα έχει διαβαστεί το byte που περιέχεται στην διεύθυνση 1235_{16}, και ας υποθέσουμε ότι εκεί υπάρχει ο αριθμός 56.
9. Τα δεδομένα του MDR αντιγράφονται στον $ScratchPad_H$, δηλαδή στο υψηλής τάξης byte του καταχωρητή προχείρου,
10. Τέλος το περιεχόμενο του ScratchPad αντιγράφεται στον ADR που τώρα μπορεί να αλλοιωθεί, καθώς δεν μας χρειάζεται πλέον η διεύθυνση μνήμης 1234. Τώρα ο ADR περιέχει την τελική διεύθυνση της υπορουτίνας. Αναφορικά με το παράδειγμά μας, ο ADR θα έχει την τιμή 5678_{16}.

Τα παραπάνω βήματα αναλύονται παρακάτω με συμβολικές εντολές μικροπρογράμματος:

Α/Α	**Συμβολική Μικρο-εντολή**	**Σημασία**
1	MAR ← ADR	Το περιεχόμενο του ADR αντιγράφεται στον MAR συνδυαζόμενο με τον καταχωρητή τμήματος DS (MAR=DS:ADR=DSx16+ADR
2	Wait	Αναμονή για να απαντήσει η μνήμη. Ο χρόνος αυτός ταυτίζεται ή είναι μεγαλύτερος από τον χρόνο απόκρισης της μνήμης.
3	MDR ← DB	Η λέξη (1 byte) που εμφανίζεται στον δίαυλο

		δεδομένων (Data Bus) αντιγράφεται στον MDR
4	SCR_L ← MDR	Τα δεδομένα του MDR αντιγράφονται στο χαμηλής τάξης byte του ScratchPad

5	EAR ← ADR+1	Το περιεχόμενο του ADR προστίθεται με τη μονάδα και το αποτέλεσμα εξάγεται στον EAR
6	MAR ← EAR	Το περιεχόμενο του EAR αντιγράφεται στον MAR συνδυαζόμενο με τον καταχωρητή τμήματος DS (MAR=DS:EAR=DSx16+EAR
7	Wait	Αναμονή για να απαντήσει η μνήμη. Ο χρόνος αυτός ταυτίζεται ή είναι μεγαλύτερος από τον χρόνο απόκρισης της μνήμης.
8	MDR ← DB	Η λέξη (1 byte) που εμφανίζεται στον δίαυλο δεδομένων (Data Bus) αντιγράφεται στον MDR
9	SCR_H ← MDR	Τα δεδομένα του MDR αντιγράφονται στο υψηλής τάξης byte του ScratchPad
10	ADR ← SCR	Τα περιεχόμενα του ScratchPad (τελική διεύθυνση) αντιγράφονται στον ADR

5.2.4. Ανάκτηση των τελικών δεδομένων

- **Βήμα 4 : Ανάκτηση των τελικών δεδομένων.**
  Εάν τα τελικά δεδομένα δεν περιλαμβάνονται στην εντολή, τότε θα πρέπει να προσκομιστούν στην CPU για να πραγματοποιηθεί η εκτέλεση της εντολής. Παρακάτω παρατίθενται δύο παραδείγματα εντολών για να γίνει κατανοητή αυτή η διαφορά:
  Εντολή MOV AL,12 Τα τελικά δεδομένα (ο αριθμός 12) περιλαμβάνονται στην εντολή.
  Εντολή MOV AL,[1234] Τα τελικά δεδομένα δεν περιλαμβάνονται στην εντολή, αλλά πρέπει να διαβαστούν από την διεύθυνση [1234].

1. Η διεύθυνση των δεδομένων βρίσκεται είτε στον ADR (απόλυτη ή έμμεση διευθυνσιοδότηση) είτε στον EAR (δεικτοδοτούμενες διευθυνσοδοτήσεις) είτε στον SP (δεικτοδότηση σωρού – stack). Παρακάτω αναφέρονται μερικά παραδείγματα εντολών, ο τρόπος διευθυνσιοδότησής τους και ο καταχωρητής που έχει την διεύθυνση των δεδομένων:

Α/Α	**Εντολή**	**Διευθυνσιοδότηση**	**Καταχωρητής που έχει την διεύθυνση των δεδομένων**
1	SBB DX,[1234]	Απόλυτη (Absolute)	ADR
2	ADD CX,[2345+BP+DI]	Δεικτοδοτούμενη (Indexed)	EAR
3	PUSH AX	Σωρού (Stack)	SP
4	CALL [0500]	Έμμεση Απόλυτη (Indirect Absolute)	ADR

2. Η διεύθυνση του κατάλληλου καταχωρητή μεταφέρεται στον MAR συνδυαζόμενη με τον καταχωρητή τμήματος DS (MAR = DS : <καταχωρητής> = DS x 16 + <καταχωρητής>) για τις περιπτώσεις των ADR και EAR, ενώ συνδυάζεται με τον SS (MAR = SS : <καταχωρητής> = SS x 16 + <καταχωρητής>) αν πρόκειται για τον SP.

3. Μετά από μία χρονική καθυστέρηση (χρόνος προσπέλασης μνήμης) το byte των δεδομένων εμφανίζεται στον εξωτερικό δίαυλο δεδομένων (Data Bus).
4. Τα δεδομένα του Data Bus αποθηκεύονται στον καταχωρητή δεδομένων (MDR).
5. Αν η εντολή είναι των 16 bit τότε θα πρέπει να διαβαστεί και 2^o byte δεδομένων από την επόμενη διεύθυνση. Για τον λόγο αυτό και επειδή θα ξαναχρησιμοποιηθεί μοιραία ο MDR, θα πρέπει αυτός να αποθηκευθεί στον ScratchPad. Αν η εντολή είναι των 8 bit τότε ο ScratchPad ΔΕΝ χρησιμοποιείται, και τα δεδομένα απλά παραμένουν στον MDR.
   Παρακάτω αναφέρονται παραδείγματα εντολών των 8 και 16 bit:

Εντολή	ADD DH,23	8 bit	(λόγω του DH, DH=8bit)
Εντολή	ADC DX,23	16bit	(λόγω του DX, DX=16bit)
Εντολή	SBB BY[1234],FF	8bit	(λόγω του BY=BYTE)
Εντολή	SUB WO[1234],FF	16bit	(λόγω του WO=WORD)

Για 16 bit εντολές λοιπόν τα παραπάνω βήματα θα επαναληφθούν άλλη μία φορά για να διαβαστεί και το 2^o byte των δεδομένων. Για να σχηματιστεί η επόμενη διεύθυνση μνήμης, θα γίνει η πράξη <καταχωρητής> + 1 στην ALU και το αποτέλεσμα θα καταχωρηθεί στον EAR. Όπου <καταχωρητής> εννοείται ο καταχωρητής διεύθυνσης που έχει την διεύθυνση των τελικών δεδομένων και που κατά περίπτωση μπορεί να είναι ο ADR, ο EAR ή ο SP .
Στην περίπτωση προσπέλασης της στοίβας, η επαυξημένη διεύθυνση που βρίσκεται στον EAR αντιγράφεται πάλι στον SP ώστε ο SP να είναι πάντα ενημερωμένος για την κορυφή της στοίβας.

6. Το περιεχόμενο του <καταχωρητή> (ADR ή EAR ή SP) προστίθεται με τον αριθμό 1 ώστε να προκύψει η επόμενη διεύθυνση μνήμης, και να διαβαστεί και το 2^o byte των τελικών δεδομενών. Η πρόσθεση θα γίνει στην ALU. Το αποτέλεσμα θα εξαχθεί στον καταχωρητή EAR. Αν πρόκειται για προσπέλαση στοίβας ο EAR αντιγράφεται στον SP
7. Το περιεχόμενο του EAR μεταφέρεται στον καταχωρητή διεύθυνσης μνήμης (MAR) συνδυαζόμενο με τον καταχωρητή τμήματος DS (MAR=DS:EAR=DSx16+EAR), και από εκεί στον εξωτερικό δίαυλο διευθύνσεων (Address Bus). Στην περίπτωση προσπέλασης στοίβας μεταφέρεται στον καταχωρητή διεύθυνσης μνήμης (MAR) το περιεχόμενο του SP, συνδυαζόμενο με τον καταχωρητή τμήματος SS (MAR=SS:SP=SSx16+SP).
8. Μετά από μία χρονική καθυστέρηση (χρόνος προσπέλασης μνήμης) το byte της παραμέτρου εμφανίζεται στον εξωτερικό δίαυλο δεδομένων (Data Bus).
9. Τα δεδομένα του Data Bus αποθηκεύονται στον καταχωρητή δεδομένων (MDR).
10. Τα δεδομένα του MDR αντιγράφονται στον $ScratchPad_H$, δηλαδή στο υψηλής τάξης byte του καταχωρητή προχείρου,

Έτσι με την ολοκλήρωση αυτού του βήματος τα τελικά δεδομένα βρίσκονται είτε στον MDR, αν πρόκειται για εντολή 8bit, είτε στον ScratchPad, αν πρόκειται για εντολή 16bit.

Τα παραπάνω βήματα αναλύονται παρακάτω με συμβολικές εντολές μικροπρογράμματος:

Α/Α	**Συμβολική Μικρο-εντολή**	**Σημασία**
1	MAR ← <καταχωρητής>	Το περιεχόμενο του <καταχωρητή> (ADR ή EAR ή SP) αντιγράφεται στον MAR συνδυαζόμενο με τον καταχωρητή τμήματος DS (MAR =

		DS:<καταχωρητής> = DS x 16 + <καταχωρητής>) για τις περιπτώσεις των ADR και EAR, ενώ συνδυάζεται με τον SS (MAR = SS : <καταχωρητής> = SS x 16 + <καταχωρητής>) αν πρόκειται για τον SP.
2	Wait	Αναμονή για να απαντήσει η μνήμη. Ο χρόνος αυτός ταυτίζεται ή είναι μεγαλύτερος από τον χρόνο απόκρισης της μνήμης.
3	MDR ← DB	Η λέξη (1 byte) που εμφανίζεται στον δίαυλο δεδομένων (Data Bus) αντιγράφεται στον MDR. **Αν η εντολή είναι των 8 bit τότε τελειώνει εδώ**
4	SCR_L ← MDR	Τα δεδομένα του MDR αντιγράφονται στο χαμηλής τάξης byte του ScratchPad

5	EAR ← <καταχωρητής>+1 (Επιπρόσθετα SP←EAR για προσπέλαση στοίβας)	Το περιεχόμενο του <καταχωρητή> προστίθεται με τη μονάδα και το αποτέλεσμα εξάγεται στον EAR. Σε περίπτωση προσπέλασης στοίβας, ο EAR αντιγράφεται στον SP
6	MAR ← EAR ή MAR←SP	Το περιεχόμενο του EAR αντιγράφεται στον MAR συνδυαζόμενο με τον καταχωρητή τμήματος DS. Σε περίπτωση προσπέλασης στοίβας, ο SP αντιγράφεται στον MAR συνδυαζόμενος με τον SS.
7	Wait	Αναμονή για να απαντήσει η μνήμη. Ο χρόνος αυτός ταυτίζεται ή είναι μεγαλύτερος από τον χρόνο απόκρισης της μνήμης.
8	MDR ← DB	Η λέξη (1 byte) που εμφανίζεται στον δίαυλο δεδομένων (Data Bus) αντιγράφεται στον MDR
9	SCR_H ← MDR	Τα δεδομένα του MDR αντιγράφονται στο υψηλής τάξης byte του ScratchPad

5.2.5. Εκτέλεση της εντολής

- **Βήμα 5 : εκτέλεση της εντολής** .
  Ανάλογα με το είδος της εντολής, σε αυτό το στάδιο απαιτείται μία ακόμα σειρά βημάτων για την ολοκλήρωση της εντολής και την αποθήκευση των αποτελεσμάτων σε καταχωρητές ή στη μνήμη.
  Παραδείγματα:

Α/Α	Εντολή	Καταχωρητής που έχει τα τελικά δεδομένα	Ενέργεια	Εξηγηση
1	MOV AL,12	MDR	AL←MDR	Αντέγραψε τον MDR στον AL
2	ADD AL,[1234]	MDR, AL	AL←AL+MDR	Πρόσθεσε στον AL τον MDR
3	MOV BX,[1234+SI]	SCR	BX←SCR	Αντέγραψε τον SCR στον BX

4	SBB CX,[5678]	SCR, CX	CX←CX-SCR-CY	Αφαίρεσε από τον CX τον SCR και το Carry bit
5	XCHG BX,DX	BX,DX	BX↔DX	Ανταλλάσσονται οι τιμές των BX,DX
6	POP AX	SCR	AX←SCR	Αντέγραψε τον SCR στον AX
7	INC SI	SI	SI←SI+1	Αύξησε τον SI κατά ένα
8	CMP DX,ABCD	SCR, DX	DX~SCR	Σύγκρινε τον DX και τον SCR επηρεάζοντας τις σημαίες – flags
9	MUL BL	AX,BL	AX←AL*BL	Πολλαπλασίασε τον AL και τον BL και βάλε το αποτέλεσμα στον AX
10	DIV CX	DX:AX, CX	AX←DX:AX/CX, υπόλοιπο στον DX	Διαίρεσε τον 32 bit αριθμό που βρίσκεται στους καταχωρητές DX:AX με τον CX και βάλε το πηλίκο στον AX και το υπόλοιπο στον DX
11	AND DL,FF	MDR,DL	DL←DL & MDR	Κάνε την λογική πράξη AND μεταξύ των DL και MDR και βάλε το αποτέλεσμα στον DL
12	JMP 0300	ADR	IP←ADR	Άντέγραψε στον IP την διεύθυνση που έχει ο ADR (0300) που σημαίνει ότι το πρόγραμμα θα συνεχιστεί από την 0300

5.2.6. Αποθήκευση των αποτελεσμάτων

- **Βήμα 6 : Αποθήκευση των αποτελεσμάτων**.
  Αν απαιτείται **τα τελικά αποτελέσματα να αποθηκευθούν στην μνήμη**, για παράδειγμα στην περίπτωση της εντολής MOV [1234],AL που αντιστοιχεί στον κώδικα μηχανής : A2

34 12, τότε θα απαιτηθούν ένα ή δύο επιπλέον προσπελάσεις στην μνήμη για αποθήκευση των δεδομένων που μπορεί να είναι των 8 ή των 16 bit. Σε αυτές τις εντολές η πρώτη παράμετρος είναι πάντα διεύθυνση μνήμης.
Η διεύθυνση αποθήκευσης των τελικών δεδομένων θα βρίσκεται είτε στον ADR, σε περιπτώσεις απόλυτης διευθυνσιοδότησης, είτε στον EAR, σε περιπτώσεις δεικτοδοτούμενων διευθυνσιοδοτήσεων, είτε στον SP σε περιπτώσεις προσπέλασης της στοίβας (εντολές PUSH). Οπότε ο κατάλληλος κάθε φορά καταχωρητής διεύθυνσης θα πρέπει να εξαχθεί στον MAR συνδυαζόμενος με τον κατάλληλο καταχωρητή τμήματος, ώστε να προσπελαστεί η διεύθυνση μνήμης στην οποία θα αποθηκευτεί το αποτέλεσμα. Αυτή τη φορά όμως θα γίνει ΕΓΓΡΑΦΗ στην μνήμη (write) και όχι ανάγνωση (read).
Αν τα δεδομένα προς αποθήκευση είναι 16 bit τότε τα παραπάνω θα πρέπει να επαναληφθούν άλλη μία φορά αλλά για την επόμενη διεύθυνση μνήμης, ώστε να αποθηκευθεί και το 2^o byte των αποτελεσμάτων.
Παρακάτω αναφέρονται μερικά παραδείγματα εντολών, ο τρόπος διευθυνσιοδότησής τους και ο καταχωρητής που έχει την διεύθυνση αποθήκευσης των τελικών αποτελεσμάτων:

Α/Α	**Εντολή**	**Μέγεθος αποτελεσμάτων**	**Διευθυνσιοδότηση**	**Καταχωρητής που έχει την διεύθυνση αποθήκευσης των αποτελεσμάτων**
1	MOV [1234],AL	8bit	Απόλυτη (Absolute)	ADR
2	ADD [2345+DI],BX	16bit	Δεικτοδοτούμενη (Indexed)	EAR
3	PUSH DX	16bit	Σωρού (Stack)	SP
4	MOV BY[5678],FF	8bit	Απόλυτη (Absolute)	ADR

Δηλαδή για την αποθήκευση του αποτελέσματος θα έχουμε τα εξής βήματα :

1. Η διεύθυνση του κατάλληλου καταχωρητή μεταφέρεται στον MAR συνδυαζόμενη με τον καταχωρητή τμήματος DS (MAR = DS : <καταχωρητής> = DS x 16 + <καταχωρητής>) για τις περιπτώσεις των ADR και EAR, ενώ συνδυάζεται με τον SS (MAR = SS : <καταχωρητής> = SS x 16 + <καταχωρητής>) αν πρόκειται για τον SP.
2. Τα δεδομένα προς εγγραφή αντιγράφονται στον MDR και από εκεί εξάγονται στον εξωτερικό δίαυλο δεδομένων, με κατεύθυνση την μνήμη RAM.
   Ανάλογα με την εντολή τα δεδομένα προς εγγραφή μπορεί να βρίσκονται σε διαφορετικό καταχωρητή δεδομένων κάθε φορά. Παρακάτω παρατίθεται μία λίστα με παραδείγματα εντολών και αναφορά του καταχωρητή που έχει το αποτέλεσμα προς εγγραφή.

Α/Α	**Εντολή**	**Μέγεθος αποτελε-σμάτων**	**Καταχωρητής που έχει τα δεδομένα προς εγγραφή στην μνήμη**	**Σχόλια**
1	MOV [1234],AL	8bit	AL	Ο AL εγγράφεται στην μνήμη [1234]
2	ADD [2345+DI],BX	16bit	SCR	Γίνεται πρόσθεση μεταξύ του περιεχομένου της μνήμης [2345+DI] και του BX (στην ALU) και το αποτέλεσμα

				καταχωρείται στον SCR, για να μην επηρεάσει άλλο καταχωρητή.
3	PUSH DX	16bit	DX	Ο DX εγγράφεται στην μνήμη
4	MOV BY[5678],FF	8bit	MDR	Εδώ τα προς εγγραφή δεδομένα είναι των 8 bit και γι'αυτό έχουν παραμείνει στον MDR, κατά την ανάγνωση της εντολής

3. Μετά από μία χρονική καθυστέρηση (χρόνος εγγραφής μνήμης) το byte των δεδομένων εγγράφεται στην μνήμη, στην διεύθυνση που είχε ήδη καθοριστεί. Εδώ ολοκληρώνεται η διαδικασία αν τα δεδομένα που θα αποθηκευτούν είναι των 8 bit.
4. Αν τα δεδομένα που θα αποθηκευτούν είναι των 16 bit τότε θα πρέπει να εγγραφεί και το 2^o byte των δεδομένων στην μνήμη, αλλά στην επόμενη διεύθυνση.
   Παρακάτω αναφέρονται παραδείγματα εντολών των 8 και 16 bit:

Εντολή	ADD [5678],DH	8 bit	(λόγω του DH, DH=8bit)
Εντολή	ADC [7890],DX	16bit	(λόγω του DX, DX=16bit)
Εντολή	SBB BY[1234],FF	8bit	(λόγω του BY=BYTE)
Εντολή	SUB WO[1234],FF	16bit	(λόγω του WO=WORD)

Για 16 bit εντολές λοιπόν τα παραπάνω βήματα θα επαναληφθούν άλλη μία φορά για να εγγραφεί στην μνήμη και το 2^o byte των αποτελεσμάτων. Για να σχηματιστεί η επόμενη διεύθυνση μνήμης, θα γίνει η πράξη <καταχωρητής> + 1 στην ALU και το αποτέλεσμα θα καταχωρηθεί στον EAR. Όπου <καταχωρητής> εννοείται ο καταχωρητής διεύθυνσης που έχει την διεύθυνση που θα αποθηκευτούν τα αποτελέσματα, και που κατά περίπτωση μπορεί να είναι ο ADR, ο EAR ή ο SP.
Στην περίπτωση προσπέλασης της στοίβας, η επαυξημένη διεύθυνση που βρίσκεται στον EAR αντιγράφεται πάλι στον SP ώστε ο SP να είναι πάντα ενημερωμένος για την κορυφή της στοίβας.

5. Το περιεχόμενο του <καταχωρητή> (ADR ή EAR ή SP) προστίθεται με τον αριθμό 1 ώστε να προκύψει η επόμενη διεύθυνση μνήμης, και να εγγραφεί και το 2^o byte των αποτελεσμάτων. Η πρόσθεση θα γίνει στην ALU. Το αποτέλεσμα θα εξαχθεί στον καταχωρητή EAR. Αν πρόκειται για προσπέλαση στοίβας ο EAR αντιγράφεται στον SP.
6. Το περιεχόμενο του EAR μεταφέρεται στον καταχωρητή διεύθυνσης μνήμης (MAR) συνδυαζόμενο με τον καταχωρητή τμήματος DS (MAR=DS:EAR=DSx16+EAR), και από εκεί στον εξωτερικό δίαυλο διευθύνσεων (Address Bus). Στην περίπτωση προσπέλασης στοίβας μεταφέρεται στον καταχωρητή διεύθυνσης μνήμης (MAR) το περιεχόμενο του SP, συνδυαζόμενο με τον καταχωρητή τμήματος SS (MAR=SS:SP=SSx16+SP).
7. Μετά από μία χρονική καθυστέρηση (χρόνος εγγραφής μνήμης) το 2^o byte του αποτελέσματος εγγράφεται στην μνήμη. Εδώ ολοκληρώνεται η διαδικασία αν το αποτέλεσμα είναι των 16 bit.

Τα παραπάνω βήματα αναλύονται παρακάτω με συμβολικές εντολές μικροπρογράμματος

A/A	**Συμβολική Μικρο-εντολή**	**Σημασία**
1	MAR ← <καταχωρητής>	Το περιεχόμενο του <καταχωρητή> (ADR ή EAR ή SP) αντιγράφεται στον MAR

		συνδυαζόμενο με τον καταχωρητή τμήματος DS (MAR = DS:<καταχωρητής> = DS x 16 + <καταχωρητής>) για τις περιπτώσεις των ADR και EAR, ενώ συνδυάζεται με τον SS (MAR = SS : <καταχωρητής> = SS x 16 + <καταχωρητής>) αν πρόκειται για τον SP.
2	MDR←<καταχωρητής αποτελέσματος>	Το περιεχόμενο του καταχωρητή που έχει το αποτέλεσμα της εντολής αντιγράφεται στον MDR. Αν το αποτέλεσμα είναι των 2 byte τότε μόνο το 1^{ο} byte (low byte) αντιγράφεται.
3	Wait	Αναμονή για την εγγραφή στην μνήμη. Ο χρόνος αυτός ταυτίζεται ή είναι μεγαλύτερος από τον χρόνο εγγραφής μνήμης. **Αν τα αποτελέσματα είναι των 8 bit τότε τελειώνει εδώ.**

4	EAR ← <καταχωρητής>+1 (Επιπρόσθετα SP←EAR για προσπέλαση στοίβας)	Το περιεχόμενο του <καταχωρητή> προστίθεται με τη μονάδα και το αποτέλεσμα εξάγεται στον EAR. Σε περίπτωση προσπέλασης στοίβας, ο EAR αντιγράφεται στον SP
5	MAR ← EAR ή MAR←SP	Το περιεχόμενο του EAR αντιγράφεται στον MAR συνδυαζόμενο με τον καταχωρητή τμήματος DS. Σε περίπτωση προσπέλασης στοίβας, ο SP αντιγράφεται στον MAR συνδυαζόμενος με τον SS.
6	MDR←<2^{ο} byte καταχωρητή αποτελέσματος>	Το 2^{ο} byte (high byte) του καταχωρητή που έχει το αποτέλεσμα της εντολής αντιγράφεται στον MDR.
7	Wait	Αναμονή για την εγγραφή στην μνήμη. Ο χρόνος αυτός ταυτίζεται ή είναι μεγαλύτερος από τον χρόνο εγγραφής μνήμης. **Αν τα αποτελέσματα είναι των 16 bit τότε η διαδικασία τελειώνει εδώ.**

5.3. Παραδείγματα εκτέλεσης εντολών

Στη συνέχεια ακολουθούν μερικά ολοκληρωμένα παραδείγματα εντολών γλώσσας μηχανής του 8088 και των αντίστοιχων μικροεντολών που εκτελούνται για την ολοκλήρωσή της.

5.3.1. Παράδειγμα εκτέλεσης της εντολής CLC

Παράδειγμα εκτέλεσης εντολής **CLC** που αντιστοιχεί στον κώδικα μηχανής F8, από τη θέση 0100:0000. :

Ταυτότητα Εντολής	
Εντολή	: CLC
Κώδικας Μηχανής	: F8
Μέγεθος Εντολής	: 1 bytes

Είδος Εντολής	: -
Διευθυνσιοδότηση	: Υπονοούμενη
Αποθήκευση	: Σε bit καταχωρητή

Φάση	Εξωτερική Λειτουργία	AB	DB	Εσωτερική Μικρο-λειτουργία
1	Διάβασμα opcode	0100:0000	F8	MAR←IP; IP←IP+1(0001); wait; MDR←DB; IR←MDR
2	Εκτέλεση εντολής	-	-	CY=0
3	Επόμενη εντολή	0100:0001	(0001)	MAR←IP; IP←IP+1(0002); wait; MDR←DB; IR←MDR

5.3.2. Παράδειγμα εκτέλεσης της εντολής MOV AL,FF

Παράδειγμα εκτέλεσης εντολής **MOV AL,FF** που αντιστοιχεί στον κώδικα μηχανής B0 FF, από τη θέση 0100:0000. Όταν εμφανίζεται διεύθυνση μέσα σε παρένθεση, π.χ. (1234) εννοείται το byte που περιέχεται στην διεύθυνση αυτή :

Ταυτότητα Εντολής	
Εντολή	: MOV AL,FF
Κώδικας Μηχανής	: B0 FF
Μέγεθος Εντολής	: 2 bytes
Είδος Εντολής	: 8bit
Διευθυνσιοδότηση	: Άμεση
Αποθήκευση	: Σε καταχωρητή

Φάση	Εξωτερική Λειτουργία	AB	DB	Εσωτερική Μικρο-λειτουργία
1	Διάβασμα opcode	0100:0000	B0	MAR←IP; IP←IP+1(0001); wait; MDR←DB; IR←MDR
2	Διάβασμα byte παραμέτρου	0100:0001	FF	MAR←IP; IP←IP+1(0002); wait; MDR←DB;
3	Εκτέλεση εντολής - τοποθέτηση στον AL	-	-	AL ← MDR
4	Επόμενη εντολή	0100:0002	(0002)	MAR←IP; IP←IP+1(0003); wait; MDR←DB; IR←MDR

5.3.3. Παράδειγμα εκτέλεσης της εντολής MOV AL,[1234]

Παράδειγμα εκτέλεσης εντολής **MOV AL,[1234]** που αντιστοιχεί στον κώδικα μηχανής A0 34 12, από τη θέση 0100:0000. Όταν εμφανίζεται διεύθυνση μέσα σε παρένθεση, π.χ. (1234) εννοείται το byte που περιέχεται στην διεύθυνση αυτή :

Ταυτότητα Εντολής	
Εντολή	: MOV AL,[1234]
Κώδικας Μηχανής	: A0 34 12
Μέγεθος Εντολής	: 3 bytes

Είδος Εντολής	: 8bit
Διευθυνσιοδότηση	: Απόλυτη
Αποθήκευση	: Σε καταχωρητή

Φάση	Εξωτερική Λειτουργία	AB	DB	Εσωτερική Μικρο-λειτουργία
1	Διάβασμα opcode	0100:0000	A0	MAR←IP; IP←IP+1(0001); wait; MDR←DB; IR←MDR
2	Διάβασμα low byte παραμέτρου διεύθυνσης	0100:0001	34	MAR←IP; IP←IP+1(0002); wait; MDR←DB; ADR_L←MDR
3	Διάβασμα high byte	0100:0002	12	MAR←IP; IP←IP+1(0003); wait; MDR←DB; ADR_H←MDR
4	Διάβασμα τελικών δεδομένων	0100:1234	(1234)	MAR←ADR; wait; MDR ←DB
5	Εκτέλεση εντολής - τοποθέτηση στον AL	-	-	AL ← MDR
6	Επόμενη εντολή	0100:0003	(0003)	MAR←IP; IP←IP+1(0004); wait; MDR←DB; IR←MDR

5.3.4. Παράδειγμα εκτέλεσης της εντολής MOV AX,[1234]

Παράδειγμα εκτέλεσης εντολής **MOV AX,[1234]** που αντιστοιχεί στον κώδικα μηχανής A1 34 12, από τη θέση 0100:0000 και είναι μία εντολή των 16 bit με απόλυτη διευθυνσιοδότηση. Όταν εμφανίζεται διεύθυνση μέσα σε παρένθεση, π.χ. (1234) εννοείται το byte που περιέχεται στην διεύθυνση αυτή :

Ταυτότητα Εντολής	
Εντολή	: MOV AX,[1234]
Κώδικας Μηχανής	: A1 34 12
Μέγεθος Εντολής	: 3 bytes
Είδος Εντολής	: 16bit
Διευθυνσιοδότηση	: Απόλυτη
Αποθήκευση	: Σε καταχωρητή

Φάση	Εξωτερική Λειτουργία	AB	DB	Εσωτερική Μικρο-λειτουργία
1	Διάβασμα opcode	0100:0000	A1	MAR←IP; IP←IP+1; wait; MDR←DB; IR←MDR
2	Διάβασμα low byte παραμέτρου διεύθυνσης	0100:0001	34	MAR←IP; IP←IP+1; wait; MDR←DB; ADR_L←MDR
3	Διάβασμα high byte	0100:0002	12	MAR←IP; IP←IP+1; wait; MDR←DB; ADR_H←MDR
4	Διάβασμα τελικών δεδομένων 1ο byte	0100:1234	(1234)	MAR←ADR; wait; MDR ←DB; SCR_L←MDR
5	Διάβασμα τελικών δεδομένων 2ο byte	0100:1235	(1235)	EAR←ADR+1; MAR←EAR; wait; MDR ←DB; SCR_H←MDR

5	Εκτέλεση εντολής - τοποθέτηση στον AX	-	-	AX ← SCR
6	Επόμενη εντολή	0100:0003	(0003)	MAR←IP; IP←IP+1; wait; MDR←DB; IR←MDR

5.3.5. Παράδειγμα εκτέλεσης της εντολής ADD BX,[3456]

Παράδειγμα εκτέλεσης εντολής **ADD BX,[3456]** που αντιστοιχεί στον κώδικα μηχανής 03 1E 56 34, από τη θέση 0100:0000 και είναι μία εντολή των 16 bit με απόλυτη διευθυνσιοδότηση. Όταν εμφανίζεται διεύθυνση μέσα σε παρένθεση, π.χ. (1234) εννοείται το byte που περιέχεται στην διεύθυνση αυτή :

Ταυτότητα Εντολής
Εντολή : ADD BX,[3456]
Κώδικας Μηχανής : 03 1E 56 34
Μέγεθος Εντολής : 4 bytes
Είδος Εντολής : 16bit
Διευθυνσιοδότηση : Απόλυτη
Αποθήκευση : Σε καταχωρητή

Φάση	Εξωτερική Λειτουργία	AB	DB	Εσωτερική Μικρο-λειτουργία
1	Διάβασμα opcode	0100:0000	03	MAR←IP; IP←IP+1; wait; MDR←DB; IR_L←MDR
2	Διάβασμα 2ου opcode	0100:0001	1E	MAR←IP; IP←IP+1; wait; MDR←DB; IR_H←MDR
3	Διάβασμα low byte παραμέτρου διεύθυνσης	0100:0002	56	MAR←IP; IP←IP+1; wait; MDR←DB; ADR_L←MDR
4	Διάβασμα high byte	0100:0003	34	MAR←IP; IP←IP+1; wait; MDR←DB; ADR_H←MDR
5	Διάβασμα τελικών δεδομένων 1ο byte	0100:3456	(3456)	MAR←ADR; wait; MDR ←DB; SCR_L←MDR
6	Διάβασμα τελικών δεδομένων 2ο byte	0100:3457	(3457)	EAR←ADR+1; MAR←EAR; wait; MDR ←DB; SCR_H←MDR
7	Εκτέλεση εντολής - τοποθέτηση στον BX	-	-	BX ← BX+SCR
8	Επόμενη εντολή	0100:0004	(0004)	MAR←IP; IP←IP+1; wait; MDR←DB; IR←MDR

5.3.6. Παράδειγμα εκτέλεσης της εντολής ADC DX,[5678+SI]

Παράδειγμα εκτέλεσης εντολής **ADC DX,[5678+SI]** που αντιστοιχεί στον κώδικα μηχανής 13 94 78 56, από τη θέση 0100:0000 και είναι μία εντολή των 16 bit με δεικτοδοτούμενη διευθυνσιοδότηση. Όταν εμφανίζεται διεύθυνση μέσα σε παρένθεση, π.χ. (1234) εννοείται το byte που περιέχεται στην διεύθυνση αυτή :

Ταυτότητα Εντολής

Εντολή	: ADC DX,[5678+SI]
Κώδικας Μηχανής	: 13 94 78 56
Μέγεθος Εντολής	: 4 bytes
Είδος Εντολής	: 16bit
Διευθυνσιοδότηση	: Δεικτοδοτούμενη
Αποθήκευση	: Σε καταχωρητή

Φάση	**Εξωτερική Λειτουργία**	**AB**	**DB**	**Εσωτερική Μικρο-λειτουργία**
1	Διάβασμα opcode	0100:0000	13	MAR←IP; IP←IP+1; wait; MDR←DB; IR_L←MDR
2	Διάβασμα 2ου opcode	0100:0001	94	MAR←IP; IP←IP+1; wait; MDR←DB; IR_H←MDR
3	Διάβασμα low byte παραμέτρου διεύθυνσης	0100:0002	78	MAR←IP; IP←IP+1; wait; MDR←DB; ADR_L←MDR
4	Διάβασμα high byte	0100:0003	56	MAR←IP; IP←IP+1; wait; MDR←DB; ADR_H←MDR
5	Διάβασμα τελικών δεδομένων 1ο byte	0100:5678+SI	(5678+SI)	EAR←ADR+SI; MAR←EAR; wait; MDR ←DB; SCR_L←MDR
6	Διάβασμα τελικών δεδομένων 2ο byte	0100:5679+SI	(5679+SI)	EAR←EAR+1; MAR←EAR; wait; MDR ←DB; SCR_H←MDR
7	Εκτέλεση εντολής - τοποθέτηση στον DX	-	-	DX ← DX+SCR+CY (Carry)
8	Επόμενη εντολή	0100:0004	(0004)	MAR←IP; IP←IP+1; wait; MDR←DB; IR←MDR

5.3.7. Παράδειγμα εκτέλεσης της εντολής JMP [6789]

Παράδειγμα εκτέλεσης εντολής **JMP [6789]** που αντιστοιχεί στον κώδικα μηχανής FF 26 89 67, από τη θέση 0100:0000 και είναι μία εντολή των 16 bit με έμμεση διευθυνσιοδότηση. Στην εντολή αυτή υπάρχει εμμεσότητα, δηλαδή πρέπει να διαβαστούν 2 bytes από την διεύθυνση [6789] και αυτά τα bytes να εκληφθούν ως η τελική διεύθυνση στην οποία θα διακλαδωθεί η εκτέλεση του προγράμματος. Η διακλάδωση του προγράμματος σε κάποια άλλη διεύθυνση, που γίνεται με τις εντολές JMP, Jxx, CALL, RET, INT, IRET, γίνεται πολύ απλά με αλλαγή της τιμής της διεύθυνσης που έχει μέσα του ο καταχωρητής IP. Η Μονάδα Ελέγχου πάντα εκτελεί την επόμενη εντολή από την διεύθυνση που «δείχνει» ο IP. Όταν εμφανίζεται διεύθυνση μέσα σε παρένθεση, π.χ. (1234) εννοείται το byte που περιέχεται στην διεύθυνση αυτή :

Ταυτότητα Εντολής	
Εντολή	: JMP [6789]
Κώδικας Μηχανής	: FF 26 89 67
Μέγεθος Εντολής	: 4 bytes
Είδος Εντολής	: 16bit
Διευθυνσιοδότηση	: Έμμεση
Αποθήκευση	: Σε καταχωρητή (αλλαγή του IP)

Φάση	Εξωτερική Λειτουργία	AB	DB	Εσωτερική Μικρο-λειτουργία
1	Διάβασμα opcode	0100:0000	FF	MAR←IP; IP←IP+1; wait; MDR←DB; IR_L←MDR
2	Διάβασμα 2ου opcode	0100:0001	26	MAR←IP; IP←IP+1; wait; MDR←DB; IR_H←MDR
3	Διάβασμα low byte παραμέτρου διεύθυνσης	0100:0002	89	MAR←IP; IP←IP+1; wait; MDR←DB; ADR_L←MDR
4	Διάβασμα high byte	0100:0003	67	MAR←IP; IP←IP+1; wait; MDR←DB; ADR_H←MDR
5	Διάβασμα τελικής διεύθυνσης 1ο byte	0100:6789	(6789)	MAR←ADR; wait; MDR ←DB; SCR_L←MDR
6	Διάβασμα τελικής διεύθυνσης 2ο byte	0100:678A	(678A)	EAR←ADR+1; MAR←EAR; wait; MDR ←DB; SCR_H←MDR
7	Εκτέλεση εντολής - τοποθέτηση της τελικής διεύθυνσης στον IP	-	-	IP ← SCR
8	Επόμενη εντολή	0100:0004	(0004)	MAR←IP; IP←IP+1; wait; MDR←DB; IR←MDR

5.3.8. Παράδειγμα εκτέλεσης της εντολής POP CX

Παράδειγμα εκτέλεσης εντολής **POP CX** που αντιστοιχεί στον κώδικα μηχανής 59, από τη θέση 0100:0000 και είναι μία εντολή των 16 bit με διευθυνσιοδότηση στοίβας (stack). Η εντολή αυτή θα διαβάσει 2 bytes από την κορυφή της στοίβας στην οποία «δείχνει» ο SP, και θα ενημερώσει τον SP αυξάνοντάς τον δύο φορές κατά 1. Να θυμίσουμε εδώ ότι η στοίβα γεμίζει από πάνω προς τα κάτω με δομή LIFO (Last In First Out). Όταν εμφανίζεται διεύθυνση μέσα σε παρένθεση, π.χ. (1234) εννοείται το byte που περιέχεται στην διεύθυνση αυτή. Υποθέτουμε ότι ο SS έχει την τιμή 0800 και ότι ο SP έχει την τιμή 1234:

Ταυτότητα Εντολής
Εντολή : POP CX
Κώδικας Μηχανής : 59
Μέγεθος Εντολής : 1 byte
Είδος Εντολής : 16bit
Διευθυνσιοδότηση : Στοίβας
Αποθήκευση : Σε καταχωρητή

Φάση	Εξωτερική Λειτουργία	AB	DB	Εσωτερική Μικρο-λειτουργία
1	Διάβασμα opcode	0100:0000	59	MAR←IP; IP←IP+1; wait; MDR←DB; IR←MDR
2	Διάβασμα τελικών δεδομένων 1ο byte	0800:1234	(1234)	MAR←SP; SP←SP+1; wait; MDR ←DB; SCR_L←MDR
5	Διάβασμα τελικής διεύθυνσης 2ο byte	0800:1235	(1235)	MAR←SP; SP←SP+1; wait; MDR ←DB; SCR_H←MDR

5	Εκτέλεση εντολής	-	-	CX ← SCR
6	Επόμενη εντολή	0100:0001	(0001)	MAR←IP; IP←IP+1; wait; MDR←DB; IR←MDR

5.3.9. Παράδειγμα εκτέλεσης της εντολής MOV [1234],AL

Παράδειγμα εκτέλεσης εντολής **MOV [1234],AL** που αντιστοιχεί στον κώδικα μηχανής A2 34 12, από τη θέση 0100:0000. Η εντολή είναι 8 bit αλλά η αποθήκευση των αποτελεσμάτων (AL) γίνεται στην μνήμη (διεύθυνση 1234). Όταν εμφανίζεται διεύθυνση μέσα σε παρένθεση, π.χ. (1234) εννοείται το byte που περιέχεται στην διεύθυνση αυτή :

Ταυτότητα Εντολής
Εντολή : MOV [1234],AL
Κώδικας Μηχανής : A2 34 12
Μέγεθος Εντολής : 3 bytes
Είδος Εντολής : 8bit
Διευθυνσιοδότηση : Απόλυτη
Αποθήκευση : Στην μνήμη

Φάση	Εξωτερική Λειτουργία	AB	DB	Εσωτερική Μικρο-λειτουργία
1	Διάβασμα opcode	0100:0000	A2	MAR←IP; IP←IP+1(0001); wait; MDR←DB; IR←MDR
2	Διάβασμα low byte παραμέτρου διεύθυνσης	0100:0001	34	MAR←IP; IP←IP+1(0002); wait; MDR←DB; ADR_L←MDR
3	Διάβασμα high byte	0100:0002	12	MAR←IP; IP←IP+1(0003); wait; MDR←DB; ADR_H←MDR
4	Εκτέλεση εντολής – αποθήκευση δεδομένων στην μνήμη	0100:1234	(AL)	MAR←ADR;MDR←AL; wait; (εγγραφή στην μνήμη και τέλος διαδικασίας)
5	Επόμενη εντολή	0100:0003	(0003)	MAR←IP; IP←IP+1(0004); wait; MDR←DB; IR←MDR

5.3.10. Παράδειγμα εκτέλεσης της εντολής MOV [1234],AX

Παράδειγμα εκτέλεσης εντολής **MOV [2345],AX** που αντιστοιχεί στον κώδικα μηχανής A3 34 12, από τη θέση 0100:0000. Η εντολή είναι 16 bit και η αποθήκευση των αποτελεσμάτων (AX) γίνεται στην μνήμη (διεύθυνση 2345). Όταν εμφανίζεται διεύθυνση μέσα σε παρένθεση, π.χ. (1234) εννοείται το byte που περιέχεται στην διεύθυνση αυτή :

Ταυτότητα Εντολής
Εντολή : MOV [2345],AX
Κώδικας Μηχανής : A3 34 12
Μέγεθος Εντολής : 3 bytes
Είδος Εντολής : 16bit
Διευθυνσιοδότηση : Απόλυτη
Αποθήκευση : Στην μνήμη

Φάση	Εξωτερική Λειτουργία	AB	DB	Εσωτερική Μικρο-λειτουργία
1	Διάβασμα opcode	0100:0000	A3	MAR←IP; IP←IP+1(0001); wait; MDR←DB; IR←MDR
2	Διάβασμα low byte παραμέτρου διεύθυνσης	0100:0001	34	MAR←IP; IP←IP+1(0002); wait; MDR←DB; ADR_L←MDR
3	Διάβασμα high byte	0100:0002	12	MAR←IP; IP←IP+1(0003); wait; MDR←DB; ADR_H←MDR
4	Εκτέλεση εντολής – αποθήκευση 1ου byte δεδομένων στην μνήμη	0100:2345	(AL)	MAR←ADR;MDR←AL; wait; (εγγραφή στην μνήμη)
5	Εκτέλεση εντολής – αποθήκευση 2ου byte δεδομένων στην μνήμη	0100:2346	(AH)	EAR←ADR+1;MAR←EAR;MDR←AH; wait; (εγγραφή στην μνήμη και τέλος διαδικασίας)
6	Επόμενη εντολή	0100:0003	(0003)	MAR←IP; IP←IP+1(0004); wait; MDR←DB; IR←MDR

5.3.11. Παράδειγμα εκτέλεσης της εντολής ADD [1234],ABCD

Παράδειγμα εκτέλεσης εντολής **ADD [1234],ABCD** που αντιστοιχεί στον κώδικα μηχανής 81 06 34 12 CD AB, από τη θέση 0100:0000 και είναι μία εντολή των 16 bit με απόλυτη διευθυνσιοδότηση και αποθήκευση αποτελεσμάτων στην μνήμη. Στην εντολή αυτή που ίσως είναι από τις πλέον σύνθετες θα πρέπει:
α) να διαβαστούν τα 2 opcodes,
β) να διαβαστούν τα 2 bytes της διεύθυνσης 1234,
γ) να διαβαστούν τα 2 bytes του αριθμού ABCD,
δ) να διαβαστούν τα τελικά δεδομένα (2 bytes) που βρίσκονται στην διεύθυνση [1234],
ε) να προστεθούν αυτά με τον αριθμό ABCD,
στ) να αποθηκευτούν τα 2 bytes του αποτελέσματος ξανά στην μνήμη.
Κατά την εκτέλεση της εντολής αυτής, και για να μην αλλοιωθεί κανένας κανονικός καταχωρητής του 8088, απαιτείται η χρησιμοποίηση και δεύτερου ScratchPad register, που αναφέρεται στις εντολές ως SCR2. Ο 1ος ScratchPad απαιτείται για την αποθήκευση του αριθμού ABCD που είναι παράμετρος, και ο 2ος για την αποθήκευση του περιεχομένου της διεύθυνσης [1234], το οποίο και θα προστεθεί με τον 1ο ScrathPad, δηλαδή με τον αριθμό ABCD.
Όταν εμφανίζεται διεύθυνση μέσα σε παρένθεση, π.χ. (1234) εννοείται το byte που περιέχεται στην διεύθυνση αυτή :

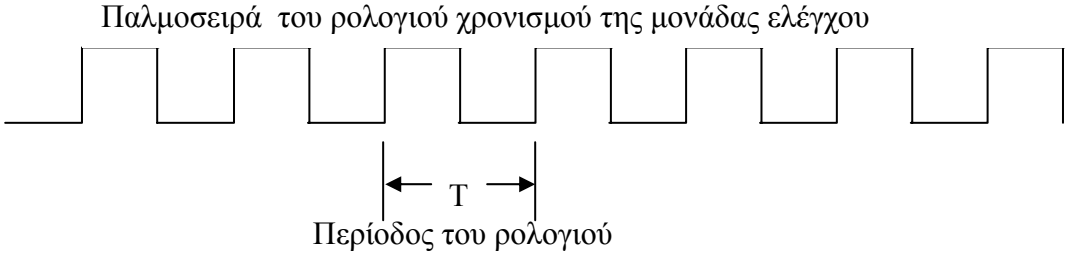
Ταυτότητα Εντολής
Εντολή : ADD [1234],ABCD
Κώδικας Μηχανής : 81 06 34 12 CD AB
Μέγεθος Εντολής : 6 bytes
Είδος Εντολής : 16bit
Διευθυνσιοδότηση : Απόλυτη
Αποθήκευση : Στην μνήμη

Φάση	Εξωτερική	AB	DB	Εσωτερική Μικρο-λειτουργία

	Λειτουργία			
1	Διάβασμα opcode	0100:0000	81	MAR←IP; IP←IP+1; wait; MDR←DB; IR_L←MDR
2	Διάβασμα 2ου opcode	0100:0001	06	MAR←IP; IP←IP+1; wait; MDR←DB; IR_H←MDR
3	Διάβασμα low byte παραμέτρου διεύθυνσης	0100:0002	34	MAR←IP; IP←IP+1; wait; MDR←DB; ADR_L←MDR
4	Διάβασμα high byte	0100:0003	12	MAR←IP; IP←IP+1; wait; MDR←DB; ADR_H←MDR
5	Διάβασμα low byte παραμέτρου αριθμού	0100:0004	CD	MAR←IP; IP←IP+1; wait; MDR←DB; SCR_L←MDR
6	Διάβασμα high byte	0100:0005	AB	MAR←IP; IP←IP+1; wait; MDR←DB; SCR_H←MDR
7	Διάβασμα τελικών δεδομένων 1ο byte	0100:1234	(1234)	MAR←ADR; wait; MDR ←DB; $SCR2_L$←MDR
8	Διάβασμα τελικών δεδομένων 2ο byte	0100:1235	(1235)	EAR←ADR+1; MAR←EAR; wait; MDR ←DB; $SCR2_H$←MDR
9	Εκτέλεση εντολής	-	-	SCR ← SCR+SCR2
10	Αποθήκευση 1ου byte αποτελέσματος στην μνήμη	0100:1234	(SCR_L)	MAR←ADR;MDR←SCR_L; wait; (εγγραφή στην μνήμη)
11	Αποθήκευση 2ου byte αποτελέσματος στην μνήμη	0100:1235	(SCR_H)	EAR←ADR+1;MAR←EAR;MDR←SCR_H; wait; (εγγραφή στην μνήμη και τέλος διαδικασίας)
12	Επόμενη εντολή	0100:0006	(0006)	MAR←IP; IP←IP+1; wait; MDR←DB; IR←MDR

5.4. Κύκλοι Εκτέλεσης Εντολών

Σε κάθε περίοδο του ρολογιού που ταυτίζεται με (1/συχνότητα), εκτελείται μία ή περισσότερες εντολές μικροκώδικα. Για παράδειγμα στον 8088 που λειτουργεί στα 4,77 MHz η περίοδος του ρολογιού έχει διάρκεια T=1/4,77MHz = 0,20964 $x10^{-6}$sec = 0,20964μsec.



Η περίοδος του ρολογιού χρονισμού του επεξεργαστή

Κάθε τέτοια περίοδος ονομάζεται «**Μικρός Κύκλος**» (minor cycle) ή «**Κύκλος Εσωτερικής Κατάστασης**» (state cycle). Είναι η μικρότερη χρονική μονάδα στην οποία μπορεί να εκτελεστεί μία μικρο-λειτουργία του επεξεργαστή, και αποτελεί την μονάδα χρόνου για οποιαδήποτε ενέργεια του επεξεργαστή. Σε ορισμένες περιπτώσεις είναι δυνατόν δύο τέτοιες μικρο-λειτουργίες να εκτελεστούν παράλληλα. Αυτό είναι δυνατόν όταν οι λειτουργίες αυτές είναι ανεξάρτητες η μία από την άλλη.
Για παράδειγμα, στην φάση προσκόμισης του Opcode, οι λειτουργίες θα μπορούσαν να υλοποιηθούν σε 3 μικρούς κύκλους :

T1: MAR ← IP
T2: IP ← IP+1 ; Αναμονή
T3: MDR ← DB ; IR ← MDR

Παρατηρούμε στον κύκλο T2 την εκτέλεση 2 λειτουργιών που είναι η αύξηση του IP κατά 1 και η αναμονή να απαντήσει η μνήμη, που επειδή είναι τελείως ανεξάρτητες, μπορούν να εκτελεστούν και παράλληλα. Επίσης στον κύκλο T3 η σύνδεση του DB με τον MDR και η σύνδεση του MDR με τον IR μπορούν να γίνουν επίσης ταυτόχρονα, ώστε η λέξη που θα «έρθει» από τον δίαυλο δεδομένων, δια μέσω του MDR να αποθηκευτεί κατ' ευθείαν στον IR.
Επίσης στο παραπάνω παράδειγμα ενδέχεται η φάση της «αναμονής» να διαρκέσει περισσότερους από 1 «μικρούς κύκλους» δηλαδή περιόδους του ρολογιού.
Γνωρίζοντας το πόσους «μικρούς κύκλους» απαιτεί μία διαδικασία, και τν συχνότητα του ρολογιού, μπορούμε να υπολογίσουμε ακριβώς πόσος χρόνος απαιτείται για την περάτωση αυτής της διαδικασίας στην CPU. Στο παραπάνω παράδειγμα που το opcode απαιτεί 3 κύκλους και η συχνότητα είναι 4,77 MHz (T=0,20964 $x10^{-6}$sec), η διάρκεια της διαδικασίας είναι :
Διάρκεια Ανάγνωσης Opcode = 3 x T = 3 x 0,20964$x10^{-6}$ = 0,62893 $x10^{-6}$ sec.

Κύκλος Μηχανής (machine cycle) ή **Μεγάλος Κύκλος** (major cycle) είναι ο χρόνος που απαιτείται για την προσπέλαση μίας θέσης μνήμης και απαιτεί έναν αριθμό από μικρούς κύκλους. Στο παραπάνω παράδειγμα ανάγνωσης του opcode πραγματοποιείται μία ανάγνωση μνήμης, για το διάβασμα του Opcode. Επομένως η διάρκεια ολόκληρης της διαδικασίας ταυτίζεται με ένα Κύκλο Μηχανής.
Η έννοια του κύκλου μηχανής είναι πολύ σημαντική στην εκτέλεση των εντολών. Κάθε κύκλος μηχανής αντιστοιχεί σε μία προσπέλαση μνήμης. Η μνήμη όμως είναι πολύ πιο αργή στην απόκρισή της απ' ότι οι καταχωρητές του επεξεργαστή. Για τον λόγο αυτό το πλήθος των κύκλων μηχανής που απαιτεί μία εντολή για να ολοκληρωθεί, είναι χαρακτηριστικό του πόσο αργή ή γρήγορη είναι η εντολή αυτή στην εκτέλεση της. Όσο περισσότερους κύκλους μηχανής χρειάζεται μία εντολή για να εκτελεστεί, τόσο πιο χρονοβόρα είναι η εντολή στην εκτέλεσή της.

Κύκλος Εντολής (instruction cycle) είναι ο χρόνος ολοκλήρωσης μία εντολής και διαρκεί έναν ή περισσότερους κύκλους μηχανής. Ο Κύκλος Εντολής είναι διαφορετικός για κάθε εντολή του επεξεργαστή.

Στη συνέχεια δίνεται ένα παράδειγμα εκτέλεσης εντολής και ο χωρισμός της σε κύκλους μηχανής και μικρούς κύκλους

Παράδειγμα 1: Κύκλοι Εκτέλεσης της εντολής CLC

ΚΥΚΛΟΣ ΕΝΤΟΛΗΣ (CLC)			
Κύκλος Μηχανής 1			
T1	T2	T3	T4
MAR←IP	IP←IP+1; Wait	MDR←DB; IR ←MDR;	CY=0

Ο κύκλος T4 στον οποίο γίνεται μηδενισμός του Carry είναι τόσο μικρός που είναι αμελητέος σε σχέση με τον κύκλο μηχανής που απαιτεί η προσκόμιση του opcode.

Παράδειγμα 2: Κύκλοι Εκτέλεσης της εντολής MOV AX,1234

ΚΥΚΛΟΣ ΕΝΤΟΛΗΣ (MOV AX,1234)									
Κύκλος Μηχανής 1			Κύκλος Μηχανής 2			Κύκλος Μηχανής 3			
T1	T2	T3	T1	T2	T3	T1	T2	T3	T4
MAR←IP	IP←IP+1; wait	MDR←DB; IR ←MDR;	MAR←IP	IP←IP+1; wait	MDR←DB; SCR_L←MDR	MAR←IP	IP←IP+1; wait	MDR←DB; SCR_H←MDR	AX←SCR

Στον κύκλο μηχανής 3 ο χρόνος T4 κατά τον οποίο αντιγράφεται ο SCR στον AX είναι αμελητέος σε σχέση με την προσπέλαση μνήμης. Γι' αυτό και ο κύκλος T4 δεν θεωρείται ξεχωριστός Κύκλος Μηχανής.
Γενικά είναι εύκολο να υπολογίσουμε πόσους κύκλους μηχανής απαιτεί μία εντολή για την εκτέλεσή της, μετρώντας το πόσες φορές χρειάζεται να προσπελάσει την μνήμη, είτε για ανάγνωση είτε για εγγραφή σε αυτήν.
Για παράδειγμα η εντολή MOV [1234],AX στο παράδειγμα του κεφαλαίου 5.3.10 απαιτεί 5 κύκλους μηχανής, ενώ η εντολή ADD [1234],ABCD στο παράδειγμα του κεφαλαίου 5.3.11 απαιτεί 10 κύκλους μηχανής.

5.5. Εντολές, δεδομένα και διευθύνσεις

Σύμφωνα με την δομή Von Neumann, που επικρατεί πλέον σε όλες τις αρχιτεκτονικές υπολογιστών, η αναπαράσταση στην μνήμη των εντολών και των δεδομένων γίνεται με τον ίδιο τρόπο δηλαδή με αριθμούς. Έτσι οι εντολές γλώσσας μηχανής αντιστοιχούν σε αριθμούς όπως και τα δεδομένα ενός προγράμματος.
Επόμενο είναι αν δοθεί μία διεύθυνση στην μνήμη, ο επεξεργαστής να μην μπορεί να καταλάβει αν πρόκειται για δεδομένα ή για εντολή. Βέβαια ούτε και ο άνθρωπος μπορεί να το καταλάβει αυτό, εκτός και αν ίδιος έχει βάλει το νούμερο εκεί, οπότε και θα ξέρει τι αναπαριστά.
Έτσι όταν η CPU ξεκινά την εκτέλεση ενός προγράμματος από κάποια διεύθυνση, με εντολή βέβαια του χειριστή του Η/Υ, τότε **ΕΚΛΑΜΒΑΝΕΙ τα bytes που βρίσκει εκεί ως ΕΝΤΟΛΕΣ** γλώσσας μηχανής.
Αν τα bytes αυτά δεν αποτελούν εντολές αλλά δεδομένα, ο επεξεργαστής δεν μπορεί να το καταλάβει. Έτσι θα τα εκλάβει ως εντολές του κώδικα μηχανής και θα τις εκτελέσει, με απρόβλεπτες βέβαια συνέπειες, για την έκβαση του προγράμματος.
Για παράδειγμα, αν στην θέση 0100:0000 έχει τοποθετηθεί η εντολή MOV AX,[59F8+SI], με αντίστοιχο κώδικα μηχανής 8B 84 F8 59 τότε η εικόνα της στην μνήμη θα πρέπει να είναι αυτή του παρακάτω σχήματος :

Διευθύνσεις	Κώδικας Μηχανής	
0100:0004	…	
0100:0003	59	
0100:0002	F8	
0100:0001	84	
0100:0000	8B	← IP (IP=0000, CS=0100)

Εικόνα στην μνήμη της εντολής MOV AX,[59F8+SI]

Αν λοιπόν η εκτέλεση του προγράμματος ξεκινήσει από την διεύθυνση 0100:0000 τότε «όλα θα πάνε καλά» και θα διαβαστούν πρώτα τα δύο opcodes 8B και 84 και στη συνέχεια τα 2 bytes της διεύθυνσης [59F8], και μετά την προσκόμιση η εντολή θα εκτελεστεί κανονικά. Στη συνέχει αθα διαβαστεί η επόμενη εντολή κ.ο.κ.
Αν όμως κατά λάθος ξεκινήσει η εκτέλεση του προγράμματος από την διεύθυνση 0100:0002, τότε η CPU ΔΕΝ ΘΑ ΚΑΤΑΛΑΒΕΙ ότι το byte που υπάρχει εκεί (F8) αποτελεί το πρώτο byte της διεύθυνσης ΜΙΑΣ ΕΝΤΟΛΗΣ ΠΟΥ ΞΕΚΙΝΑΕΙ 2 BYTES ΝΩΡΙΤΕΡΑ.
Η ενέργεια που θα κάνει η CPU είναι να εκλάβει το byte αυτό (F8) ως ΤΟ OPCODE ΜΙΑΣ ΕΝΤΟΛΗΣ ΓΛΩΣΣΑΣ ΜΗΧΑΝΗΣ και να το εκτελέσει.
Συγκεκριμένα το Opcode F8 αντιστοιχεί στην εντολή CLC, και επομένως η CPU θα εκτελέσει τα κατάλληλα βήματα για να μηδενίσει το Carry bit του καταχωρητή σημαιών (FG – Flag Register).
Στη συνέχεια και εφόσον η πρώτη εντολή δεν είχε παραμέτρους θα διαβάσει το επόμενο byte (59) και θα το εκλάβει ως το opcode ΤΗΣ ΕΠΟΜΕΝΗΣ ΕΝΤΟΛΗΣ.
Συγκεκριμένα το Opcode 59 αντιστοιχεί στην εντολή POP CX, και επομένως η CPU θα εκτελέσει τα κατάλληλα βήματα για να ανακαλέσει από την στοίβα την τιμή του καταχωρητή CX, ενημερώνοντας και τον SP.
Όπως γίνεται κατανοητό, λόγω της εκκίνησης του προγράμματος από λάθος διεύθυνση, τα byte που αποτελούν παραμέτρους εντολών, εκλαμβάνονται ως opcodes και εκτελούνται, με απρόβλεπτες βέβαια συνέπειες, καθώς οι αντίστοιχες εντολές γλώσσας μηχανής είναι «τυχαίες».
Επομένως γίνεται κατανοητό ότι ο προγραμματιστής και χειριστής του Η/Υ είναι ο μόνος υπεύθυνος για την σωστή εκτέλεση των προγραμμάτων και την εκκίνησή τους από τις σωστές διευθύνσεις.

5.6. Κατηγορίες Εντολών

Οι εντολές ενός μικρο-επεξεργαστή χωρίζονται σε αρκετές κατηγορίες που συνήθως είναι :
1. Εντολές **μεταφοράς δεδομένων** μεταξύ καταχωρητών και μνήμης (MOV, PUSH, POP, XCHG, IN, OUT, XLAT, LAHF, SAHF,).
2. Εντολές **αριθμητικών πράξεων** (ADD, ADC, INC, SUB, SBB, DEC, NEG, MUL, IMUL, DIV, IDIV, CBW, CWD).
3. Εντολές **λογικών πράξεων** (AND, OR, XOR, TEST, NOT).
4. Εντολές **σύγκρισης** καταχωρητών και μνήμης (CMP)
5. Εντολές **ολίσθησης και περιστροφής** (SHL, SHR, SAR, ROL, ROR, RCL, RCR)
6. Εντολές χειρισμού **αλφαριθμητικών** (REP, MOVS, CMPS, SCAS, LODS, STOS)
7. Εντολές **διακλάδωσης** υπό συνθήκη ή άνευ συνθήκης (CALL, JMP, LOOP, INT, JE/NZ, JNE/JNZ, JL/JNGE, JLE/JNG, JNL/JGE, …).

8. **Ειδικές εντολές** που δεν υπάγονται στις παραπάνω κατηγορίες (NOP, CLC, STC, CMC, CLD, STD, CLI, STI, HALT, WAIT, ESC, LOCK).

5.7. Τρόποι διευθυνσιοδότησης μνήμης

Οι εντολές του 8088 συντάσσονται με αρκετούς διαφορετικούς τρόπους, και δέχονται διαφορετικές παραμέτρους, αλλά δεν συντάσσονται όλες οι εντολές με όλους τους τρόπους. Οι διαφορετικοί αυτοί τρόποι σύνταξης ονομάζονται και τρόποι διευθυνσιοδότησης μνήμης : Στη συνέχεια επεξηγούνται με παραδείγματα και μπλόκ διαγράμματα οι διάφοροι τρόποι διευθυνσιοδότησης μνήμης που χρησιμοποιούνται στην σύνταξη των εντολών γλώσσας μηχανής:

1. **Υπονοούμενος** (Implied Addressing Mode) : Η σύνταξη της εντολής δεν περιέχει παράμετρο ή συντάσσεται με συγκεκριμένο τρόπο ο οποίος υπονοεί το ποιες είναι οι παράμετροι.
   Παράδειγμα : STC (Set Carry),
   LODSB ([DS:SI]→ AL)

   Υπονοούμενος (Implied). Παράδειγμα : CMC

   Σε αυτό τον τρόπο διευθυνσιοδότησης μετά το διάβασμα του opcode (F5) που καταλήγει στον καταχωρητή IR, ακολουθεί αμέσως η εκτέλεση της εντολής

2. **Άμεση προσπέλαση** (Immediate/Direct Addressing Mode): σε αυτό τον τρόπο διευθυνσιοδότησης η παράμετρος της εντολής είναι σταθερός αριθμός.
   Παράδειγμα : ADD AL,78 (AL+=78),
   SUB AX,1234 (AX-=1234)

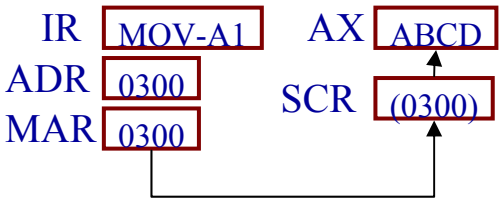
   Άμεση Προσπέλαση (Immediate). Παράδειγμα : SUB AX, 1234

   Στην μέθοδο αυτή μετά το διάβασμα του opcode που καταλήγει στον IR , διαβάζεται και ο σταθερός αριθμός που πηγαίνει στον SCR (16bit) ή MDR (8bit). Στη συνέχεια ακολουθεί η εκτέλεση της εντολής.

3. **Απόλυτη Προσπέλαση** (Absolute Addressing Mode) : σε αυτό τον τρόπο διευθυνσιοδότησης η παράμετρος της εντολής είναι σταθερή διεύθυνση.
   Παράδειγμα : MOV AX,[2345] (AX← (DS:2345))
   XOR CX,[0800]

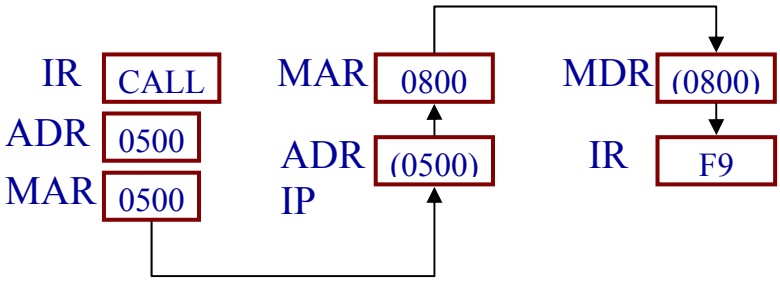
Απόλυτη Προσπέλαση (Absolute). Π.χ. : MOV AX,[0300]
Περιεχόμενο (0300,0301)=ABCD



Στην μέθοδο αυτή μετά το διάβασμα του opcode που καταλήγει στον IR , διαβάζεται και η διεύθυνση που είναι παράμετρος της εντολής και καταλήγει στον ADR. Στη συνέχεια η διεύθυνση αυτή εξάγεται στον MAR για προσκόμιση των τελικών δεδομένων. Σε εντολές 16bit θα απαιτηθούν 2 προσπελάσεις στη μνήμη. Τα τελικά δεδομένα διαβάζονται και καταλήγουν είτε στον SCR (16bit) είτε παραμένουν στον MDR (8bit). Στη συνέχεια ακολουθεί η εκτέλεση της εντολής.

4. **Έμμεση Προσπέλαση** (Indirect Addressing Mode) : Η εντολή δέχεται ως παράμετρο μία διεύθυνση μνήμης από την οποία διαβάζει κάποια bytes (2 ή 4) τα οποία σχηματίζουν την τελική διεύθυνση στην οποία θα επενεργήσει η εντολή.
Παράδειγμα : CALL FAR [0500] (θα πάει στην διεύθυνση 0500 και θα διαβάσει από εκεί 4 bytes, 2 για το segment και 2 για το offset, και στην διεύθυνση που σχηματίζεται ως segment:offset θα γίνει τελικά η κλήση υπορουτίνας)

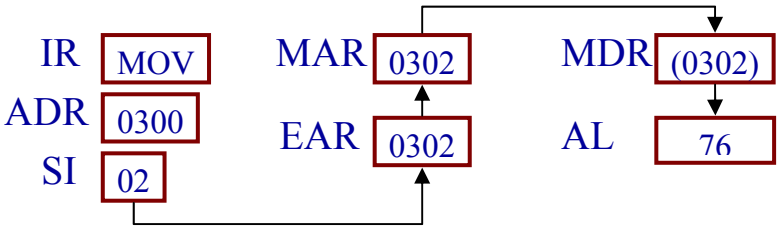
Έμμεσος απόλυτος (Indirect Absolute). Π.χ. : CALL [0500],
(0500)=0800, (0800)=F9 (STC)



Στην μέθοδο αυτή μετά το διάβασμα του opcode που καταλήγει στον IR , διαβάζεται και η διεύθυνση που είναι παράμετρος της εντολής και καταλήγει στον ADR (0500). Στη συνέχεια η διεύθυνση αυτή εξάγεται στον MAR για προσκόμιση της διεύθυνσης που υπάρχει εκεί. Η προσπέλαση μνήμης θα γίνει 2 φορές (0500, 0501) για να διαβαστούν τα 2 bytes της διεύθυνσης. Αυτά θα αποθηκευτούν στον ADR και θα αντιγραφούν στον IP για αλλαγή της ροής του προγράμματος. Ο IP θα προωθηθεί στον MAR και μέσω του MDR θα διαβαστεί το opcode που υπάρχει εκεί.

5. **Απόλυτος Δεικτοδοτούμενος** (Absolute Indexed Addressing Mode). Η εντολή δέχεται ως παράμετρο μία διεύθυνση μνήμης που αποτελεί την «βάση» ενός πίνακα. Στην εντολή συμμετέχει και ένας καταχωρητής δείκτη (index) ο οποίος θα προστεθεί στην διεύθυνση βάσης για να δώσει την τελική διεύθυνση των δεδομένων.

Απόλυτος Δεικτοδοτούμενος (Absolute Indexed). Π.χ. :
MOV AL,[0300+SI] (SI)=02, (0302)=76



Στην μέθοδο αυτή μετά το διάβασμα του opcode που καταλήγει στον IR , διαβάζεται και η διεύθυνση που είναι παράμετρος της εντολής και καταλήγει στον ADR (0300). Στη συνέχεια στη διεύθυνση αυτή προστίθεται ο καταχωρητής δείκτη που στο παράδειγμα είναι ο SI. Το αποτέλεσμα αποθηκεύεται στον EAR που εξάγεται στον MAR για να προσπελαστεί η διεύθυνση. Τα τελικά δεδομένα διαβάζονται και εκτελείται η εντολή.

5.8. Αρχιτεκτονικές CISC - RISC

Μέχρι τα τέλη της δεκαετίας του '70 κυρίαρχη τάση στον σχεδιασμό CPU ήταν η δημιουργία σύνθετων εντολών μηχανής, οι οποίες διερμηνεύονταν μέσω των κατάλληλων μικροπρογραμμάτων μέσα στις CPU. Σκοπός της τάσης αυτής ήταν η γεφύρωση του χάσματος ανάμεσα στις εντολές της γλώσσας μηχανής και αυτές των γλωσσών προγραμματισμού υψηλού επιπέδου. Οι εντολές αυτές απαιτούν αρκετούς κύκλους μηχανής, καθώς κάθε εντολή γλώσσας μηχανής αναλύεται σε έναν αριθμό από μικρο-εντολές. Οι υπολογιστές που βασίζονται σε αυτή την αρχή ονομάζονται **CISC** (Complex Instruction Set Computer). Η μεγάλη πλειοψηφία των μικροεπεξεργαστών που έχουν κατασκευαστεί ποτέ και κατασκευάζονται, ακολουθεί αυτή την τεχνολογία. Στους επεξεργαστές CISC που διαθέτουν πλούσιο σετ εντολών, μία εντολή υψηλού επιπέδου μίας γλώσσας όπως η C (π.χ. A=B/2+C*D), υλοποιείται με λίγες εντολές γλώσσας μηχανής. Στο συγκεκριμένο παράδειγμα η πράξη θα αναλυθεί σε μία διαίρεση, ένα πολλαπλασιασμό και μία πρόσθεση, καθώς μία CISC CPU συνήθως διαθέτει έτοιμες εντολές γλώσσας μηχανής για πολλαπλασιασμό και διαίρεση.

Στις αρχές της δεκαετίας του '80 παρουσιάστηκε στο Berkeley και στο Stanford μια διαφορετική αντίληψη στον σχεδιασμό των Συνόλων Εντολών, η οποία ονομάστηκε **RISC** (Reduced Instruction Set Computers). Οι πρώτες CPU που σχεδιάστηκαν με βάση την νέα αντίληψη ήταν ο RISC I και ο MIPS που εξελίχθηκαν στις μηχανές SPARC και MIPS αντίστοιχα.

Στην αρχιτεκτονική RISC η έμφαση δόθηκε, αρχικά, στην δημιουργία Συνόλου Εντολών που θα περιείχαν απλές μόνο εντολές, οι οποίες θα εκτελούνταν ταχύτατα. Αργότερα, διαπιστώθηκε ότι το κλειδί για την καλή απόδοση ήταν οι εντολές να μπορούν να υποβάλλονται, δηλαδή να ξεκινούν, γρήγορα. Ο χρόνος που χρειαζόταν η εντολή για να εκτελεστεί είχε μικρότερη σημασία από το πόσες εντολές μπορούσαν να ξεκινήσουν ανά δευτερόλεπτο. Συνοψίζοντας, σήμερα υπάρχει ένα σύνολο αρχών σχεδιασμού, οι **Αρχές Σχεδιασμού RISC**, τις οποίες προσπαθούν να ακολουθήσουν όλοι οι κατασκευαστές CPU γενικής χρήσης. Οι Αρχές αυτές προβλέπουν τα ακόλουθα :

1. Εκτέλεση εντολών απευθείας από το υλικό. Στις RISC CPU δεν υπάρχει μικροπρόγραμμα και διερμηνεία των εντολών. Αντίθετα, κάθε εντολή εκτελείται άμεσα από κάποιο τμήμα υλικού μέσα στη CPU. Έτσι οι RISC επεξεργαστές χρησιμοποιούν Μονάδες Ελέγχου κατασκευασμένες αποκλειστικά με ψηφιακά κυκλώματα (Hard-Wired Control Units)
2. Μέγιστος ρυθμός υποβολής εντολών. Όπως είδαμε κρίσιμος παράγοντας για την μεγιστοποίηση της απόδοσης μιας CPU είναι να υποβάλλονται ταυτόχρονα όσο το δυνατόν περισσότερες εντολές. Αυτό μπορεί να επιτευχθεί χρησιμοποιώντας παραλληλία και άλλους τρόπους.
3. Εύκολη αποκωδικοποίηση των εντολών. Ένα κρίσιμο όριο για τον ρυθμό υποβολής των εντολών οφείλεται στην ανάγκη να αποκωδικοποιούνται οι μεμονωμένες εντολές για να προσδιοριστεί τι πόρους χρειάζονται. Για εύκολη αποκωδικοποίηση οι εντολές θα πρέπει να έχουν κανονική μορφή (opcode+operands), σταθερό μήκος και μικρό αριθμό τελεστέων. Όσο δε λιγότερες μορφές των εντολών υπάρχουν τόσο το καλύτερο.
4. Περιορισμός εντολών που προσπελαύνουν την μνήμη. Λόγω του ότι η προσπέλαση στη μνήμη είναι μια αργή και απρόβλεπτη, χρονικά, διαδικασία, θα πρέπει όλες οι εντολές, εκτός αυτών της μορφής LOAD και STORE (MOV), να χρησιμοποιούν τους καταχωρητές και όχι τη μνήμη.
5. Αφθονία καταχωρητών γενικής χρήσης. Για να ικανοποιηθεί η προηγούμενη αρχή θα πρέπει να υπάρχουν πολλοί καταχωρητές στη CPU, ώστε να μπορούν να αποθηκευτούν πολλά δεδομένα σε κάθε στιγμή. Η εξάντληση των καταχωρητών και η ανάγκη μεταφοράς των δεδομένων τους στη μνήμη και αργότερα πάλι στη CPU είναι ανεπιθύμητη κατάσταση και μπορεί να αποφευχθεί μόνο με την αφθονία καταχωρητών στην CPU.

Παραδείγματα RISC επεξεγαστών : SPARC (SUN), MIPS (MIPS Technologies Inc.), Alpha (DEC), POWER4 (IBM). Χρησιμοποιούνται σε Workstations υψηλών δυνατοτήτων.
Παρόλα τα πλεονεκτήματα της αρχιτεκτονικής RISC, οι SPARC, MIPS, Alpha και οι άλλες RISC CPU, δεν κατόρθωσαν να υποσκελίσουν τους CISC μ/ε της Intel. Ενας λόγος γι' αυτό ήταν η συμβατότητα προς τα πίσω που προσέφερε η Intel. Ένας άλλος λόγος ήταν ότι η Intel ενσωμάτωσε στοιχεία RISC στους CISC μ/ε της, δημιουργώντας υβριδικές CPU που συνδύαζαν τα πλεονεκτήματα και των δύο αρχιτεκτονικών. Έτσι, από τον 80486 και μετά, όλες οι CPU της Intel έχουν έναν πυρήνα RISC, ο οποίος εκτελεί τις απλούστερες εντολές, που είναι και οι πιο συνηθισμένες, ενώ οι σύνθετες εντολές εκτελούνται μέσω μικροπρογραμμάτων και διερμηνείας, όπως σε κάθε CISC μ/ε. Το τελικό αποτέλεσμα είναι οι πιο συνηθισμένες εντολές να είναι γρήγορες και οι λιγότερο συνηθισμένες πιο αργές. Έτσι, η υβριδική αυτή αρχιτεκτονική προσφέρει ανταγωνιστική απόδοση, σε σχέση με μια καθαρά RISC CPU, και ταυτόχρονα επιτρέπει την εκτέλεση παλαιότερου λογισμικού χωρίς καμιά τροποποίηση.