Μάθημα 3.1 Τμήματα του Υπολογιστή

Σκοπός του μαθήματος

Σκοπός του μαθήματος αυτού είναι να περιγράψει τη βασική εσωτερική δομή ενός υπολογιστή και ιδιαίτερα της Κεντρικής Μονάδας Επεξεργασίας· να δείξει τη χρησιμότητα και τη λειτουργία κάθε μονάδας, και τις αλληλεπιδράσεις μεταξύ τους.

Τι θα μάθεις;

Όταν ολοκληρώσεις το μάθημα αυτό, θα μπορείς:

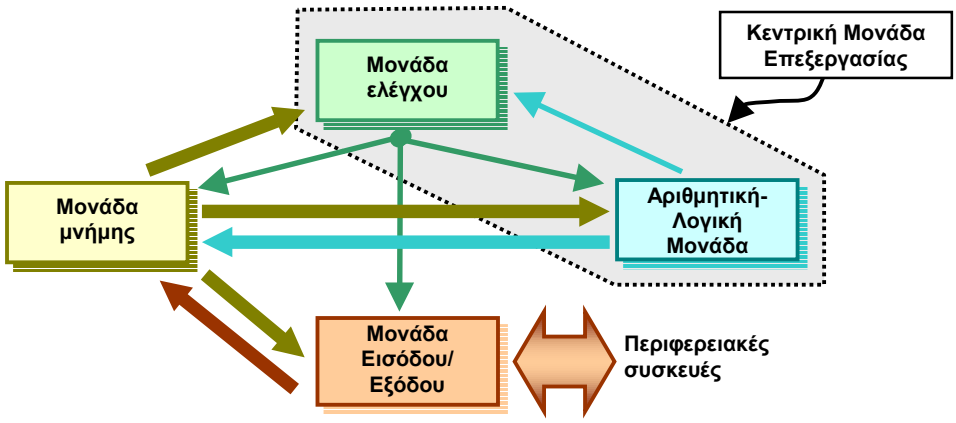
- *Να απαριθμείς τα βασικά τμήματα ενός υπολογιστή*
- *Να περιγράφεις τη λειτουργία κάθε μίας από τις βασικές μονάδες του υπολογιστή*
- *Να εξηγείς τον τρόπο που οι μονάδες αλληλεπιδρούν μεταξύ τους*

Κάθε ηλεκτρονικός υπολογιστής αποτελείται από τουλάχιστον τέσσερα κύρια τμήματα:

- ⇨ την *αριθμητική-λογική μονάδα* (arithmetic logical unit - ALU)
- ⇨ τη *μονάδα μνήμης* (memory unit)
- ⇨ τη *μονάδα εισόδου-εξόδου* (Input-Output ή I/O unit)
- ⇨ τη *μονάδα ελέγχου* (control unit).

Η μονάδα ελέγχου μαζί με την αριθμητική-λογική μονάδα αποτελούν την *κεντρική μονάδα επεξεργασίας* ή *ΚΜΕ* (Central Processing Unit - CPU) ή *επεξεργαστή* (processor).

Μολονότι κάθε μία από αυτές τις μονάδες διαφέρει από υπολογιστή σε υπολογιστή ως προς την πολυπλοκότητα, το μέγεθος και το σχεδιασμό της, ωστόσο επιτελεί την ίδια εργασία.



Καταχωρητές

Η ΚΜΕ χρειάζεται κάποιο τρόπο να αποθηκεύει προσωρινά κάποια δεδομένα, τα οποία χρησιμοποιεί κατά τη λειτουργία της.

> Για να εκτελέσει μια πρόσθεση, η ΚΜΕ πρέπει να έχει αποθηκεύσει κάπου τους προσθετέους και μετά να καταχωρήσει το αποτέλεσμα.

Έτσι στο εσωτερικό της η ΚΜΕ διαθέτει μικρές μονάδες αποθήκευσης δεδομένων, που έχουν μέγεθος ένα byte ή όσο το μήκος λέξης του υπολογιστή και ονομάζονται *καταχωρητές* (registers).

Οργάνωση του υπολογιστή

Η **αριθμητική-λογική μονάδα** έχει τις ακόλουθες δυνατότητες:

- εκτελεί τις βασικές αριθμητικές πράξεις, δηλαδή την πρόσθεση, αφαίρεση, πολλαπλασιασμό και διαίρεση
- εκτελεί λογικές πράξεις, π.χ. *λογικό αθροισμα* (OR), *λογικό γινόμενο* (AND).
- εκτελεί διάφορες βοηθητικές εργασίες, π.χ. *ολίσθηση* (shift) μίας τιμής.

Οι πράξεις αυτές μπορούν να γίνουν μέσα σε κλάσματα του δευτερολέπτου. Οι μεγάλες όμως ταχύτητες θα ήταν άχρηστες, αν τα δεδομένα για την εκτέλεση των πράξεων έπρεπε να διοχετευθούν στον υπολογιστή από τον άνθρωπο αμέσως πριν από την εκτέλεση κάθε πράξης.

Έτσι οι πληροφορίες για τους υπολογισμούς που πρόκειται να εκτελέσει ο υπολογιστής, δίνονται όλες μαζί στην αρχή και στη συνέχεια αυτός λειτουργεί αυτόματα, χωρίς τη μεσολάβηση του ανθρώπου. Αναγκαία προϋπόθεση για να γίνει αυτό δυνατό είναι η ύπαρξη της μονάδας μνήμης.

Η **μονάδα μνήμης** χρησιμεύει για την αποθήκευση των αρχικών δεδομένων που είναι απαραίτητα για τους υπολογισμούς, των αποτελεσμάτων που προκύπτουν ενδιάμεσα, των τελικών αποτελεσμάτων καθώς και ενός συνόλου εντολών (οδηγιών), τις οποίες ο υπολογιστής πρέπει να εκτελέσει για να πραγματοποιήσει τον υπολογισμό.

Τα αρχικά δεδομένα ονομάζονται *δεδομένα εισόδου* (input data), τα τελικά αποτελέσματα *δεδομένα εξόδου* (output data) και το σύνολο των εντολών *πρόγραμμα* (program).

Η **μονάδα εισόδου/εξόδου** επιτρέπει την επικοινωνία του εξωτερικού περιβάλλοντος με τον υπολογιστή. Στη μονάδα αυτή συνδέονται οι διάφορες περιφερειακές μονάδες, όπως π.χ. ο εκτυπωτής και το πληκτρολόγιο.

Η **μονάδα ελέγχου** έχει σκοπό τον έλεγχο και το συντονισμό όλων των λειτουργιών του υπολογιστή. Παίρνει από τη μνήμη μία προς μία τις εντολές του προγράμματος, τις αναλύει σε στοιχειώδεις εργασίες και στέλνει στις διάφορες μονάδες λεπτομερείς οδηγίες για το ποια λειτουργία πρέπει να εκτελέσουν και πότε.

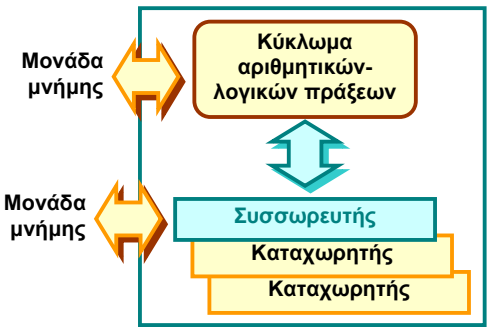
Το μοντέλο αυτό του υπολογιστή δόθηκε για πρώτη φορά από τον Von Neumann το 1947. Σήμερα οι περισσότεροι υπολογιστές ακολουθούν βασικά αυτό το μοντέλο.

Αριθμητική-λογική μονάδα

Η αριθμητική-λογική μονάδα, όπως είδαμε, εκτελεί αριθμητικές και λογικές πράξεις καθώς και διάφορες βοηθητικές εργασίες. Αποτελείται από δύο τμήματα:

- ☑ καταχωρητές που χρησιμεύουν για την προσωρινή αποθήκευση αριθμών
- ☑ κυκλώματα για την εκτέλεση πράξεων και άλλων βοηθητικών εργασιών

Συνήθως σε μία αριθμητική-λογική μονάδα υπάρχει ένας καταχωρητής που χρησιμοποιείται για την αποθήκευση του αποτελέσματος από κάθε πράξη που εκτελεί η μονάδα. Ο καταχωρητής αυτός ονομάζεται *συσσωρευτής* (accumulator), γιατί «συσσωρεύει» τα διάφορα αποτελέσματα των υπολογισμών.



Αριθμητική-Λογική Μονάδα

> Κατά τον υπολογισμό της αριθμητικής έκφρασης ((x+1)/5)*3 + y ο συσσωρευτής θα περιέχει κατά σειρά τις τιμές x, x+1, (x+1)/5, ((x+1)/5)*3, ((x+1)/5)*3 + y.

Για να εκτελεστούν οι πράξεις από την αριθμητική-λογική μονάδα πρέπει πρώτα να μεταφερθούν τα δεδομένα των πράξεων στους καταχωρητές της από τη μνήμη και μετά να μεταφερθούν τα αποτελέσματα από την αριθμητική-λογική μονάδα προς τη μνήμη.

Μονάδα μνήμης

Η μονάδα μνήμης του υπολογιστή αποτελείται από ολοκληρωμένα κυκλώματα που αποθηκεύουν δυαδικά ψηφία, δηλαδή πληροφορίες με τη μορφή 0 ή 1.

Διαιρείται σε *θέσεις μνήμης* (memory positions) και κάθε μια θέση μνήμης μπορεί να αποθηκεύσει μία *λέξη* (word), δηλαδή ένα σύνολο δυαδικών ψηφίων. Το πλήθος των δυαδικών ψηφίων της λέξης ονομάζεται *μήκος λέξης* (word length) και είναι σταθερό για κάθε υπολογιστή· συνήθως είναι μία δύναμη του 2.

Το πλήθος των λέξεων που αποτελούν τη μνήμη είναι το *μέγεθός* της. Όπως το μήκος λέξης, έτσι και το μέγεθος της μνήμης συνήθως είναι μία δύναμη του 2.

Για να μπορούμε να αναφερθούμε στις θέσεις της μνήμης, τις αριθμούμε. Κάθε θέση μνήμης έτσι έχει ένα αριθμό που αντιστοιχεί σε αυτή και ονομάζεται *διεύθυνση* (address). Αν η διεύθυνση παριστάνεται με m δυαδικά ψηφία, οι διευθύνσεις της μνήμης ξεκινούν από το 0 και φθάνουν έως 2^m-1.

> Σε ένα υπολογιστή που η μνήμη έχει μέγεθος 2^{10} λέξεις, οι διευθύνσεις της μνήμης ξεκινούν από 0 και φθάνουν μέχρι τη 2^{10}-1 = 1023. Οι αριθμοί αυτοί μπορούν να παρασταθούν στο δυαδικό σύστημα με 10 bits, έτσι οι διευθύνσεις στον υπολογιστή αυτό παριστάνονται σαν ένας 10-ψήφιος δυαδικός αριθμός.

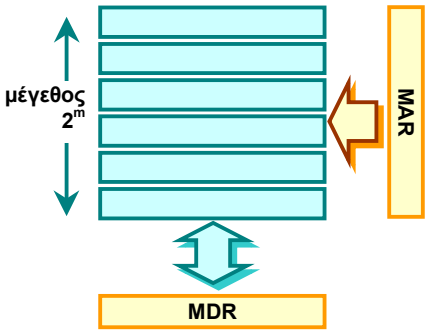
Εκτός από τα κυκλώματα που αποθηκεύουν δεδομένα, η μονάδα μνήμης περιλαμβάνει επίσης τουλάχιστον δύο καταχωρητές:

Ο *καταχωρητής δεδομένων της μνήμης* (memory data register - MDR) χρησιμοποιείται για την προσωρινή αποθήκευση των δεδομένων που διαβάζονται ή γράφονται στη μνήμη. Επειδή χρησιμοποιείται για τη μεταφορά δεδομένων, ο MDR έχει ίδιο μέγεθος με τη λέξη του υπολογιστή.

MDR

Ο *καταχωρητής διευθύνσεων της μνήμης* (memory address register - MAR) κρατά τη διεύθυνση της λέξης που πρόκειται να διαβαστεί από τη μνήμη ή να γραφεί σε αυτή.
Επειδή χρησιμοποιείται για την αποθήκευση διευθύνσεων, το μέγεθος του MAR είναι τέτοιο ώστε να κωδικοποιήσει οποιαδήποτε διεύθυνση.

MAR



Για να αποθηκευθεί μία λέξη στη μνήμη, η τιμή της γράφεται στον MDR και η διεύθυνση όπου θα αποθηκευθεί η λέξη γράφεται στον MAR. Μετά ενεργοποιείται η

μονάδα της μνήμης και γίνεται η μεταφορά των δεδομένων στη θέση που πρέπει. Η διαδικασία αυτή ονομάζεται *εγγραφή* (write) στη μνήμη. Μετά από μία εγγραφή, το περιεχόμενο μίας θέσης της μνήμης διατηρείται το ίδιο έως ότου γίνει νέα εγγραφή στην ίδια θέση. Τότε το παλιό της περιεχόμενο χάνεται και αντικαθίσταται με το νέο.

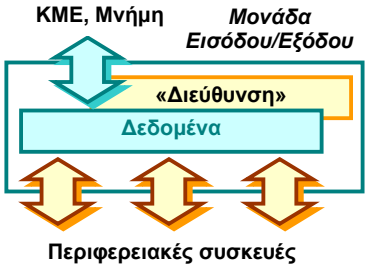
Για να διαβαστεί το περιεχόμενο μίας θέσης της μνήμης, πρέπει πρώτα να γραφεί στον MAR η διεύθυνσή της. Μόλις ενεργοποιηθεί η μονάδα της μνήμης, τα περιεχόμενα της θέσης αυτής μεταφέρονται στον MDR, για να χρησιμοποιηθούν μετά από κάποια άλλη μονάδα του υπολογιστή. Αυτή η διαδικασία ονομάζεται *ανάγνωση* (read). Η ανάγνωση δεν επηρεάζει το περιεχόμενο της θέσης μνήμης που διαβάζεται.

Η μονάδα μνήμης περιέχει επιπλέον κυκλώματα για τον έλεγχο της λειτουργίας της και την επικοινωνία με τις υπόλοιπες μονάδες του υπολογιστή.

Μονάδα Εισόδου - Εξόδου

Η μονάδα εισόδου - εξόδου δέχεται πληροφορίες από το εξωτερικό περιβάλλον του υπολογιστή, μέσω των περιφερειακών μονάδων και δίνει πάλι μέσω των μονάδων τα αποτελέσματα σε αυτό.

Όπως και η μονάδα μνήμης, η μονάδα εισόδου / εξόδου διαθέτει τουλάχιστον δύο καταχωρητές, έναν για τα δεδομένα που μεταδίδονται από και προς τις περιφερειακές συσκευές, και έναν για τη «διεύθυνση» της περιφερειακής συσκευής με την οποία ανταλλάσσονται τα δεδομένα. Τα δεδομένα αποθηκεύονται προσωρινά σε μικρές «μνήμες» της μονάδας που ονομάζονται *απομονωτές* (buffers).

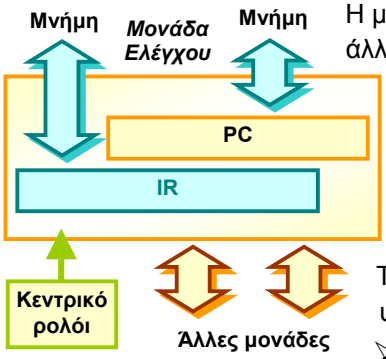


Το μέγεθος του καταχωρητή των δεδομένων της μονάδας Ε/Ε είναι συνήθως ίδιο με το μήκος λέξης του υπολογιστή, ενώ το μέγεθος του καταχωρητή των «διευθύνσεων» εξαρτάται από το πλήθος των περιφερειακών που μπορεί να χειριστεί η μονάδα.

Οι περιφερειακές μονάδες έχουν γενικά μικρή ταχύτητα, πολύ μικρότερη από την ΚΜΕ. Για να γίνει συγχρονισμός της διαφοράς ταχύτητας μεταξύ ΚΜΕ και περιφερειακών μονάδων, υπάρχουν μέσα στη μονάδα εισόδου-εξόδου ειδικά κυκλώματα για κάθε περιφερειακό, τα οποία ελέγχουν εάν έχει τελειώσει μια μεταφορά δεδομένων από τη μονάδα προς το περιφερειακό (ή αντίστροφα).

Επίσης, η μονάδα εισόδου-εξόδου περιέχει διάφορα κυκλώματα για τον έλεγχο της λειτουργίας της και την επικοινωνία της με τις υπόλοιπες μονάδες του υπολογιστή.

Μονάδα ελέγχου



Η μονάδα ελέγχου αποτελείται από δύο κύριους καταχωρητές, καθώς και από άλλα απαραίτητα κυκλώματα για τον έλεγχο και συντονισμό της λειτουργίας του υπολογιστή.

- *Καταχωρητής εντολών* (instruction register - IR). Ο καταχωρητής αυτός που δέχεται μία προς μία τις εντολές του προγράμματος από τη μνήμη, για να αναγνωρισθούν, να αναλυθούν σε επί μέρους εργασίες και τέλος να εκτελεσθούν.

Το μήκος του καταχωρητή εντολών IR είναι ίσο με το μήκος λέξεως του υπολογιστή.

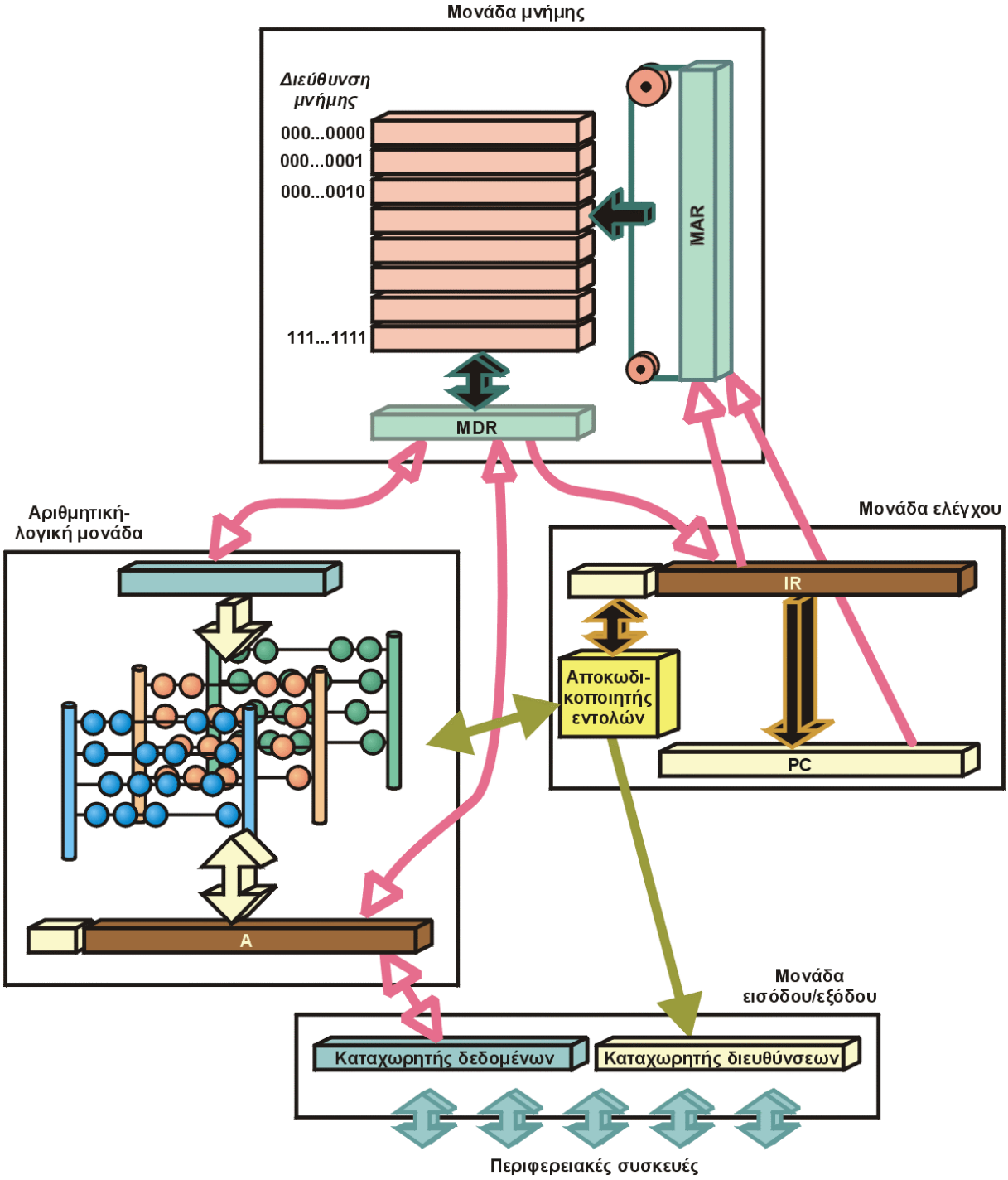
- *Μετρητής προγράμματος* ή *μετρητής εντολών* (program counter – PC, instruction counter). Το περιεχόμενό του δίνει κάθε φορά τη διεύθυνση κάποιας θέσης της μνήμης στην οποία υπάρχει η επόμενη εντολή προγράμματος που θα εκτελεστεί.

Η εντολή αυτή θα μεταφερθεί πρώτα από τη μνήμη στον καταχωρητή IR, και μετά

ο μετρητής προγράμματος θα αυξηθεί αυτομάτως κατά 1 για να αναφέρεται στην επόμενη εντολή.
Αν 2^k είναι το πλήθος των θέσεων της μνήμης που μπορεί να έχει ο υπολογιστής, το μήκος του μετρητή προγράμματος PC είναι k.

Η μονάδα ελέγχου δέχεται σε τακτικά χρονικά διαστήματα, της τάξης των λίγων ns, ένα σήμα από το κεντρικό ρολόι του υπολογιστή (ένα κρύσταλλο). Στο «ρυθμό» αυτού του σήματος γίνονται όλες οι λειτουργίες στον υπολογιστή.

Η γενική οργάνωση του υπολογιστή φαίνεται και στο σχήμα που ακολουθεί.



Ανακεφαλαίωση

Οι βασικές μονάδες ενός υπολογιστή είναι τέσσερις:

- *Η αριθμητική-λογική μονάδα που εκτελεί όλες τις αριθμητικές και λογικές πράξεις που περιέχονται σε ένα πρόγραμμα.*
- *Η μονάδα μνήμης, η οποία αποθηκεύει τα προγράμματα και τα δεδομένα τους.*
- *Η μονάδα εισόδου/εξόδου, η οποία συντονίζει την επικοινωνία του υπολογιστή με τις περιφερειακές συσκευές, και, μέσω αυτών, με το χρήστη.*
- *Η μονάδα ελέγχου η οποία αποκωδικοποιεί τα προγράμματα που εκτελούνται από τον υπολογιστή και αποφασίζει ποιες λειτουργίες πρέπει να εκτελέσουν οι υπόλοιπες μονάδες.*

Η αριθμητική-λογική μονάδα και η μονάδα ελέγχου αποτελούν την Κεντρική Μονάδα Επεξεργασίας του Υπολογιστή.

Γλωσσάριο όρων

Ανάγνωση	*Read*
Απομονωτής	*Buffer*
Αριθμητική-Λογική Μονάδα	*Arithmetic Logical Unit - ALU*
Δεδομένα Εισόδου	*Input Data*
Δεδομένα Εξόδου	*Output Data*
Διεύθυνση	*Address*
Εγγραφή	*Write*
Επεξεργαστής	*Processor*
Θέση Μνήμης	*Memory Position*
Καταχωρητής	*Register*
Καταχωρητής Δεδομένων Της Μνήμης	*Memory Data Register - MDR*
Καταχωρητής Διευθύνσεων Της Μνήμης	*Memory Address Register - MAR*
Καταχωρητής Εντολών	*Instruction Register - IR*
Κεντρική Μονάδα Επεξεργασίας - ΚΜΕ	*Central Processing Unit - CPU*
Λέξη	*Word*
Λογικό Αθροισμα	*OR*
Λογικό Γινόμενο	*AND*
Μετρητής Προγράμματος ή Μετρητής Εντολών	*Program Counter - PC ή Instruction Counter*
Μονάδα Εισόδου-Εξόδου	*Input/Output ή I/O Unit*
Μονάδα Ελέγχου	*Control Unit*
Μονάδα Μνήμης	*Memory Unit*
Ολίσθηση	*Shift*
Πρόγραμμα	*Program*
Συσσωρευτής	*Accumulator*

Ερωτήσεις

? Ποια είναι τα κύρια μέρη ενός ηλεκτρονικού υπολογιστή;

? Ποια είναι η οργάνωση και η λειτουργία της αριθμητικής-λογικής μονάδας;

? Ποια είναι η οργάνωση και η λειτουργία της μονάδας μνήμης;

? Ποια είναι η οργάνωση και η λειτουργία της μονάδας εισόδου/εξόδου;

? Ποια είναι η οργάνωση και η λειτουργία της μονάδας ελέγχου;

? Τι ονομάζουμε μήκος λέξης;

? Ποια είναι η λειτουργία των καταχωρητών MAR και MDR;

? Ποια είναι η λειτουργία του καταχωρητή εντολών και του μετρητή προγράμματος;

? Περίγραψε τη διαδικασία της εγγραφής και της ανάγνωσης στη μνήμη.

Μάθημα 3.2 Γλώσσα Μηχανής

Σκοπός του μαθήματος

Σκοπός του μαθήματος αυτού είναι να παρουσιάσει τις εντολές που μπορεί να εκτελέσει η ΚΜΕ ενός εκπαιδευτικού υπολογιστή και τον τρόπο που μπορούμε να γράψουμε προγράμματα με αυτές.

Τι θα μάθεις;

Όταν ολοκληρώσεις το μάθημα αυτό, θα μπορείς:

- *Να εξηγείς την αρχή των δύο φάσεων*
- *Να απαριθμείς τις εντολές ενός εκπαιδευτικού υπολογιστή και να εξηγείς τη λειτουργία τους*
- *Να γράφεις απλά προγράμματα για τον υπολογιστή αυτόν και να εξηγείς τα αποτελέσματα της εκτέλεσής τους*

Άβακας: ο αρχαιότερος αριθμητικός υπολογιστής

Στο μάθημα αυτό και στα επόμενα, θα περιγράψουμε τον τρόπο λειτουργίας ενός υπολογιστή με τη βοήθεια ενός εκπαιδευτικού υπολογιστή που ονομάζεται Άβακας. Ο Άβακας είναι σκόπιμα μικρός και απλός για εκπαιδευτικούς λόγους, ωστόσο οι αρχές λειτουργίας του είναι οι ίδιες με αυτές των πραγματικών υπολογιστών.

Το μήκος λέξης του Άβακα είναι 16 bits. Στην Αριθμητική-Λογική του μονάδα περιέχει το συσσωρευτή Α. Οι αρνητικοί αριθμοί στον Άβακα κωδικοποιούνται με την παράσταση συμπληρώματος του 2.

Η αρχή των δύο φάσεων

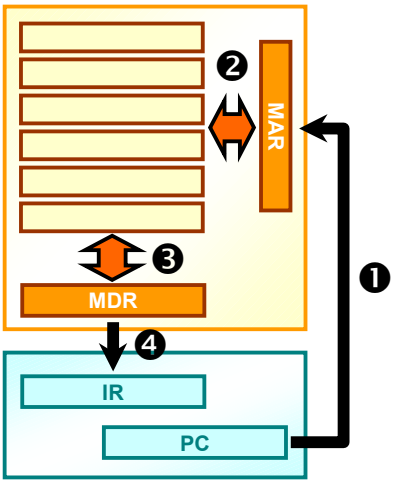
Όπως έχουμε ήδη μάθει, στη μνήμη αποθηκεύονται πληροφορίες που μπορούν να ερμηνευθούν σαν δεδομένα ή σαν εντολές. Για να εκτελεστεί ένα πρόγραμμα από τον υπολογιστή, πρέπει οι εντολές του προγράμματος να είναι αποθηκευμένες σε κάποιες από τις θέσεις της μνήμης. Μάλιστα, συνήθως οι εντολές αποθηκεύονται σε διαδοχικές θέσεις της μνήμης σύμφωνα με την σειρά που πρέπει να εκτελεστούν.

Υπάρχει λοιπόν μια ομάδα διευθύνσεων μνήμης οι οποίες αντιστοιχούν σε θέσεις μνήμης που περιέχουν εντολές. Θα αποκαλούμε «διεύθυνση της εντολής» τη διεύθυνση της θέσης μνήμης που περιέχει την εντολή.

Για να εκτελέσει κάθε εντολή του προγράμματος, ο υπολογιστής περνά από δύο *φάσεις* (phases) ή *κύκλους* (cycles):

- Τη *φάση* (ή *κύκλο*) *ανάκλησης* (fetch phase ή cycle). Στη φάση αυτή μεταφέρεται από τη μνήμη στην ΚΜΕ η εντολή που θα εκτελεστεί.
- Τη *φάση* (ή *κύκλο*) *εκτέλεσης* (execution phase ή cycle), όπου γίνεται η εκτέλεση των λειτουργιών που υποδηλώνει η συγκεκριμένη εντολή.

Φάση ανάκλησης



Στη φάση ανάκλησης, η διεύθυνση εντολής που περιέχεται στο μετρητή προγράμματος PC μεταφέρεται στον καταχωρητή διευθύνσεων μνήμης MAR της μονάδας μνήμης (βήμα ❶). Στη συνέχεια ενεργοποιείται η λειτουργία ανάγνωσης της μονάδας μνήμης (βήμα ❷) και το περιεχόμενο της θέσης που υποδεικνύει ο MAR, δηλαδή η εντολή, μεταφέρεται στον καταχωρητή δεδομένων μνήμης MDR (βήμα ❸). Από εκεί, η εντολή μεταφέρεται στον καταχωρητή εντολών IR της μονάδας ελέγχου (βήμα ❹).

Όταν τελειώσει η μεταφορά της εντολής στον IR, ο μετρητής προγράμματος PC αυξάνεται αυτομάτως κατά 1, και υποδεικνύει την επόμενη εντολή στη μνήμη. Έτσι είναι έτοιμος για την ανάκληση της επόμενης εντολής.

Ο καταχωρητής εντολών IR περιέχει πλέον την τρέχουσα εντολή, η οποία είναι έτοιμη να εκτελεστεί από την ΚΜΕ.

Φάση εκτέλεσης

Στη συνέχεια ακολουθεί η φάση εκτέλεσης. Σε αυτή η εντολή που βρίσκεται στον καταχωρητή εντολών IR αποκωδικοποιείται από ένα ειδικό κύκλωμα που ονομάζεται *αποκωδικοποιητής* (decoder) και αναλύεται σε επί μέρους στοιχειώδεις λειτουργίες.

Η μονάδα ελέγχου στέλνει τότε στις υπόλοιπες μονάδες τα κατάλληλα σήματα για την εκτέλεση των επί μέρους λειτουργιών με την κατάλληλη σειρά. Η ολοκλήρωση των επί μέρους λειτουργιών σημαίνει και εκτέλεση της ίδιας της εντολής.

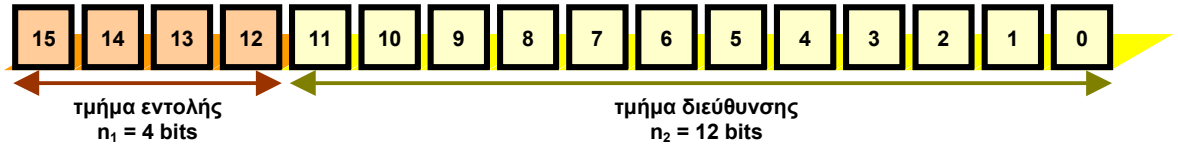
Τη φάση εκτέλεσης ακολουθεί ένα νέο ζεύγος φάσης ανάκλησης - φάσης εκτέλεσης για την επόμενη εντολή.

Τα ζεύγη των δύο φάσεων θα επαναλαμβάνονται για όλες τις εντολές του προγράμματος με τη σειρά, ώσπου να ανακληθεί και εκτελεστεί μια συγκεκριμένη εντολή του υπολογιστή, η οποία υποδεικνύει τη διακοπή της λειτουργίας του (εντολή STOP).

Στην πράξη, στους πιο πολλούς υπολογιστές υπάρχουν και άλλες δευτερεύουσες φάσεις μέσα στον κύκλο κάθε εντολής.

Μορφή της εντολής

Κάθε εντολή παριστάνεται με μία ή περισσότερες λέξεις του υπολογιστή, ενώ μερικοί υπολογιστές αποθηκεύουν πολλές εντολές σε μία λέξη. Στον Άβακα κάθε εντολή έχει μέγεθος μίας λέξης, δηλαδή 16 bits.



Η εντολή χωρίζεται σε δύο τμήματα, όπως βλέπουμε και στο σχήμα. Το πρώτο ονομάζεται *κώδικας εντολής* (instruction code) και έχει μήκος n_1 bits και το δεύτερο *τμήμα διευθύνσεως* (address part) και έχει μήκος n_2 bits· το άθροισμα $n_1 + n_2$ δίνει το μήκος της εντολής. Ο κώδικας εντολής ορίζει τη λειτουργία που πρέπει να εκτελεστεί από την ΚΜΕ, ενώ το τμήμα διευθύνσεως περιέχει τα δεδομένα της.

> Στον Άβακα, που το μήκος της εντολής είναι 16 bits, τα $n_1 = 4$ πρώτα bits περιέχουν τον κώδικα της εντολής και τα υπόλοιπα $n_2 = 12$ bits αποτελούν το τμήμα διευθύνσεως.

Ο σχεδιαστής κάθε υπολογιστή αποφασίζει και καθορίζει τις εντολές που θα μπορεί να εκτελέσει ο υπολογιστής, δηλαδή το *ρεπερτόριο εντολών* του (instruction set). Στη συνέχεια αντιστοιχίζει κάθε μία από τις εντολές με ένα δυαδικό αριθμό των n_1 bits που θα τη συμβολίζει.

Αφού ο κώδικας εντολής έχει μήκος n_1 bits και κάθε κώδικας πρέπει βέβαια να παριστάνει μόνο μία εντολή, κάθε υπολογιστής μπορεί να διαθέτει έως 2^{n_1} διαφορετικές εντολές.

> Στον Άβακα όπου n_1=4 bits, ο μέγιστος αριθμός των διαφορετικών εντολών είναι 2^4=16.

Ο πίνακας που περιέχει τις εντολές ενός υπολογιστή και τον αντίστοιχο δυαδικό κώδικα για κάθε εντολή είναι ο *πίνακας εντολών* του υπολογιστή αυτού.

Οι εντολές χωρίζονται σε ομάδες, ανάλογα με την εργασία την οποία επιτελούν. Υπάρχουν λοιπόν εντολές εκτέλεσης αριθμητικών πράξεων, εντολές μεταφοράς πληροφοριών, εντολές άλματος, εντολές ολίσθησης, εντολές εισόδου-εξόδου κλπ.

Το τμήμα διευθύνσεως της εντολής, που έχει μήκος n_2, είναι ένας δυαδικός αριθμός, που συνήθως παριστάνει μια διεύθυνση μνήμης.

> Η εντολή 0011000000001101 του Άβακα, έχει κώδικα εντολής 0011 και τμήμα διευθύνσεως $000000001101_{(2)}$. Η τιμή αυτή, αν ερμηνευθεί σαν διεύθυνση μνήμης, δηλώνει τη θέση μνήμης με διεύθυνση $1101_{(2)} = 13_{(10)}$.

Οι έννοιες «διεύθυνση της εντολής» και «τμήμα διευθύνσεως της εντολής» είναι διαφορετικές και δεν πρέπει να τις συγχέουμε.

Διεύθυνση της εντολής είναι η διεύθυνση μιας θέσεως της μνήμης στην οποία είναι *αποθηκευμένη* η εντολή.

Το **τμήμα διευθύνσεως της εντολής** είναι ένα μέρος της ίδιας της εντολής, το οποίο συνήθως υποδεικνύει τη διεύθυνση κάποιας *άλλης* θέσεως της μνήμης, το περιεχόμενο της οποίας θα χρησιμοποιηθεί κατά την εκτέλεση της εντολής.

Εντολές αναφοράς στη μνήμη

Θα ξεκινήσουμε με το ρεπερτόριο εντολών του Άβακα από μία ομάδα εντολών που ονομάζονται *εντολές αναφοράς στη μνήμη* (memory reference instructions). Στις εντολές αυτές, το τμήμα διεύθυνσης ερμηνεύεται πάντα σαν μία διεύθυνση της μνήμης.

Στον πίνακα που ακολουθεί θα συμβολίζουμε:

- την τιμή του τμήματος διεύθυνσης της εντολής με **N**
- το περιεχόμενο του συσσωρευτή A με **(A)**
- το περιεχόμενο της διεύθυνσης που υποδεικνύει το τμήμα διεύθυνσης N της εντολής με **(N)**

- την αποθήκευση μίας τιμής σε μία θέση μνήμης ή σε ένα καταχωρητή με το σύμβολο ←

Κώδικας εντολής	Σημασία της εντολής	
0001	(A) ← (N)	**Το περιεχόμενο της διεύθυνσης N της μνήμης αντιγράφεται στο συσσωρευτή Α.**
0010	(N) ← (A)	**Το περιεχόμενο του συσσωρευτή Α αντιγράφεται στη διεύθυνση N της μνήμης.**
0011	(A) ← (A)+(N)	**Το περιεχόμενο της διεύθυνσης N της μνήμης προστίθεται στο περιεχόμενο του συσσωρευτή Α. Το αποτέλεσμα μένει στο συσσωρευτή και αποτελεί το νέο περιεχόμενό του.**
0100	(A) ← (A)-(N)	**Το περιεχόμενο της διεύθυνσης N της μνήμης αφαιρείται από το περιεχόμενο του συσσωρευτή Α. Το αποτέλεσμα μένει στο συσσωρευτή.**
0101	(A) ← (A)×(N)	**Το περιεχόμενο του συσσωρευτή Α πολλαπλασιάζεται με το περιεχόμενο της διεύθυνσης N της μνήμης. Το αποτέλεσμα μένει στο συσσωρευτή.**
0110	(A) ← (A):(N)	**Το περιεχόμενο του συσσωρευτή Α διαιρείται με το περιεχόμενο της διεύθυνσης N της μνήμης. Το ακέραιο πηλίκο της διαίρεσης μένει στο συσσωρευτή, ενώ το υπόλοιπο χάνεται.**

Η εντολή 0010000000000010 έχει κώδικα εντολής 0001 και τμήμα διεύθυνσης $000000000010_{(2)} = 2_{(10)}$. Έτσι η εντολή σημαίνει «αντίγραψε το περιεχόμενο του συσσωρευτή Α στη θέση 2 της μνήμης».

Η εντολή 0100000000000110 έχει κώδικα εντολής 0100 και τμήμα διεύθυνσης $000000000110_{(2)} = 6_{(10)}$. Η εντολή σημαίνει «αφαίρεσε το περιεχόμενο της θέσης 6 της μνήμης από το συσσωρευτή Α και αποθήκευσε το αποτέλεσμα πάλι στον Α». Το περιεχόμενο της θέσης 6 της μνήμης δεν αλλάζει.

Ας δούμε ένα μικρό πρόγραμμα που χρησιμοποιεί εντολές αναφοράς στη μνήμη.

Διεύθυνση μνήμης	Περιεχόμενο: Κώδικας εντολής	Περιεχόμενο: Τμήμα διεύθυνσης	
000000000000	0001	000000001010	Πρόγραμμα
000000000001	0011	000000001011	
000000000010	0010	000000001100	
000000000011	0000	000000001000	
000000000100	**Το περιεχόμενο αυτών των θέσεων μνήμης δε μας ενδιαφέρει.**		
000000000101			
000000000110			
000000000111			
000000001000			
000000001001			
000000001010	0000000010000000		Δεδομένα προγράμματος
000000001011	0000000010000101		
000000001100	0000000000000000		

Το πρόγραμμα αυτό αποτελείται από τέσσερις εντολές, αποθηκευμένες στις διευθύνσεις της μνήμης $0_{(10)}$ - $3_{(10)}$.

Εντολή 0: Ο κώδικας εντολής είναι 0001 και το τμήμα διευθύνσεως έχει την τιμή $1010_{(2)} = 10_{(10)}$. Η εντολή σημαίνει «φόρτωσε το συσσωρευτή Α με το περιεχόμενο της θέσης μνήμης $10_{(10)}$».

Εντολή 1: Ο κώδικας εντολής είναι 0011 και το τμήμα διευθύνσεως έχει την τιμή $1011_{(2)} = 11_{(10)}$. Η εντολή σημαίνει «πρόσθεσε στο συσσωρευτή Α με το περιεχόμενο της θέσης μνήμης $11_{(10)}$».

Εντολή 2: Ο κώδικας εντολής είναι 0010 και το τμήμα διευθύνσεως έχει την τιμή $1100_{(2)} = 12_{(10)}$. Η εντολή σημαίνει «αποθήκευσε το περιεχόμενο του συσσωρευτή Α στη θέση μνήμης $12_{(10)}$».

Εντολή 3: Ο κώδικας εντολής είναι 0000, που όπως θα δούμε παρακάτω σημαίνει «τερμάτισε τη λειτουργία του υπολογιστή», είναι δηλαδή η εντολή STOP.

Θα παρακολουθήσουμε βήμα-βήμα την εκτέλεση του προγράμματος αυτού. Αρχικά, οι τρεις καταχωρητές της ΚΜΕ που χρησιμοποιούμε, δηλαδή ο PC, ο IR και ο A έχουν την τιμή 0.

Εντολή	Λειτουργία της εντολής	Καταχωρητές της ΚΜΕ
0 ανά-κληση	Το περιεχόμενο της θέσης μνήμης που δείχνει ο PC μεταφέρεται στον IR. Ο PC αυξάνεται κατά 1.	(PC) = 000000000001 (IR) = 0001000000001010 (A) = 0000000000000000
0 εκτέλεση	Ο κώδικας εντολής ($0001_{(2)}$) του IR αποκωδικοποιείται και βρίσκεται ότι είναι η εντολή (A) ← (N), δηλαδή (A) ← ($1010_{(2)}$). Ο συσσωρευτής Α φορτώνεται με το περιεχόμενο της θέσης μνήμης $1010_{(2)}$ με τα εξής βήματα: 1) Η διεύθυνση μνήμης N = 000000001010 μεταφέρεται στον καταχωρητή MAR της μονάδας μνήμης. 2) Ενεργοποιείται η μονάδα μνήμης για ανάγνωση και τα περιεχόμενα της θέσης μνήμης N γράφονται στον καταχωρητή δεδομένων MDR. 3) Από τον MDR τα δεδομένα που έχουν την τιμή $0000000010000000_{(2)}$ αντιγράφονται στο συσσωρευτή Α.	(PC) = 000000000001 (IR) = 0001000000001010 (A) = 0000000010000000
1 ανά-κληση	Το περιεχόμενο της θέσης μνήμης που δείχνει ο PC μεταφέρεται στον IR. Ο PC αυξάνεται κατά 1.	(PC) = 000000000010 (IR) = 0011000000001011 (A) = 0000000010000000
1 εκτέλεση	Ο κώδικας εντολής ($0011_{(2)}$) του IR αποκωδικοποιείται και βρίσκεται ότι είναι η εντολή (A) ← (A)+(N), δηλαδή (A) ← (A)+($1011_{(2)}$). Ο υπολογισμός γίνεται με τα εξής στάδια: 1) Η διεύθυνση μνήμης N = 000000001011 μεταφέρεται στον καταχωρητή MAR της μονάδας μνήμης. 2) Ενεργοποιείται η μονάδα μνήμης για ανάγνωση και τα περιεχόμενα της θέσης μνήμης N γράφονται στον καταχωρητή δεδομένων MDR. 3) Το περιεχόμενο του MDR που έχει την τιμή $0000000010000101_{(2)}$ προστίθεται μέσα στην αριθμητική-λογική μονάδα του Άβακα με τα περιεχόμενα του Α. Το αποτέλεσμα έχει την τιμή $0000000100000101_{(2)}$ και είναι αποθηκευμένο μέσα στην αριθμητική-λογική μονάδα. 4) Το αποτέλεσμα της πρόσθεσης καταχωρείται στο συσσωρευτή Α.	(PC) = 000000000010 (IR) = 0011000000001011 (A) = 0000000100000101
2 ανά-κληση	Το περιεχόμενο της θέσης μνήμης που δείχνει ο PC μεταφέρεται στον IR. Ο PC αυξάνεται κατά 1.	(PC) = 000000000011 (IR) = 0010000000001100 (A) = 0000000100000101
2 εκτέλεση	Ο κώδικας εντολής ($0010_{(2)}$) του IR αποκωδικοποιείται και βρίσκεται ότι είναι η εντολή (N) ← (A), δηλαδή ($1100_{(2)}$) ← (A). Το περιεχόμενο του συσσωρευτή καταχωρείται στη θέση μνήμης $1100_{(2)}$ με τα εξής βήματα: 1) Η διεύθυνση μνήμης N = 000000001100 μεταφέρεται στον καταχωρητή MAR της μονάδας μνήμης. 2) Το περιεχόμενο του συσσωρευτή Α που είναι $0000000100000101_{(2)}$ αντιγράφεται στον MDR. 3) Ενεργοποιείται η μονάδα μνήμης για εγγραφή και τα περιεχόμενα του καταχωρητή δεδομένων MDR γράφονται στη θέση μνήμης N.	(PC) = 000000000011 (IR) = 0010000000001100 (A) = 0000000100000101
3 ανά-κληση	Το περιεχόμενο της θέσης μνήμης που δείχνει ο PC μεταφέρεται στον IR. Ο PC αυξάνεται κατά 1.	(PC) = 000000000100 (IR) = 0000000000001000 (A) = 0000000100000101
3 εκτέλεση	Ο κώδικας εντολής ($0000_{(2)}$) του IR αποκωδικοποιείται και βρίσκεται ότι είναι η εντολή STOP. Η λειτουργία του υπολογιστή σταματά. Το τμήμα διευθύνσεως της εντολής αγνοείται.	

Μετά την ολοκλήρωση του προγράμματος το περιεχόμενο της θέσης μνήμης $1100_{(2)}$ = $12_{(10)}$ έχει αλλάξει και έχει πάρει την τιμή $0000000100000101_{(2)}$. Οι υπόλοιπες θέσεις μνήμης διατηρούν τα περιεχόμενά τους.

Στο πρόγραμμα αυτό οι θέσεις μνήμης με διευθύνσεις από 000000000000 έως και 000000000011 περιέχουν εντολές, ενώ οι θέσεις μνήμης με διευθύνσεις από 000000001010 έως και 000000001100 περιέχουν δεδομένα. Ο υπολογιστής όμως δεν κάνει διάκριση μεταξύ δεδομένων και εντολών. Αν κάποια στιγμή ο μετρητής

προγράμματος PC είχε την τιμή 000000001011, το περιεχόμενο της αντίστοιχης θέσης μνήμης θα είχε εκτελεστεί σαν να ήταν εντολή, παρότι στην πραγματικότητα αποτελεί δεδομένα.

> Είναι ευθύνη του προγραμματιστή να γράψει το πρόγραμμα κατά τέτοιο τρόπο, ώστε ο υπολογιστής να εκτελεί τις εντολές και όχι κατά λάθος δεδομένα σαν εντολές.

Η εντολή STOP

Η εντολή STOP έχει κωδικό 0000 και η εκτέλεσή της προκαλεί τον τερματισμό της λειτουργίας του Άβακα.

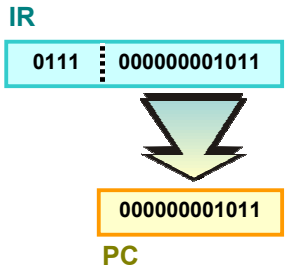
Εντολές άλματος

> Ας φανταστούμε ένα πρόγραμμα αρκετά μεγάλο, ώστε να καταλαμβάνει όλες τις θέσεις της μνήμης του Άβακα, που είναι $2^{12} = 4096$. Αν κάθε εντολή του Άβακα απαιτεί κατά μέσο όρο 2ms[1] για να εκτελεστεί, τότε όλο το πρόγραμμα θα χρειαζόταν $4096 \cdot 2\text{ms} = 8{,}192\text{s}$ για να εκτελεστεί. Ξέρουμε όμως πάρα πολλά προγράμματα που απαιτούν πολύ περισσότερο χρόνο για να εκτελεστούν σε πολύ πιο γρήγορους υπολογιστές. Τι συμβαίνει;

Στο ρεπερτόριο εντολών όλων των υπολογιστών υπάρχουν εντολές οι οποίες αλλάζουν τη σειρά εκτέλεσης των εντολών. Όταν εκτελεστεί μια τέτοια εντολή, αλλάζει η τιμή του μετρητή προγράμματος και η εκτέλεση του προγράμματος συνεχίζεται σε κάποιο άλλο σημείο του, σε μεγαλύτερη ή μικρότερη διεύθυνση μνήμης. Αυτές είναι οι *εντολές άλματος* (jump instructions) και είναι απαραίτητες για τη δημιουργία προγραμμάτων, όπου κάποιες εντολές εκτελούνται κατ' επανάληψη.

Ο Άβακας διαθέτει δύο εντολές άλματος:

Α) Την *εντολή άλματος χωρίς συνθήκη* (unconditional jump instruction) με κώδικα εντολής 0111.



Κατά την εκτέλεση της εντολής αυτής το περιεχόμενο του τμήματος διευθύνσεως του καταχωρητή εντολών IR γράφεται στο μετρητή προγράμματος PC.

Στην επόμενη φάση ανακλήσεως θα διαβαστεί από τη μνήμη η επόμενη εντολή, της οποίας η διεύθυνση περιέχεται στο μετρητή προγράμματος. Το περιεχόμενο του μετρητή προγράμματος όμως τώρα έχει μεταβληθεί από την εντολή άλματος και είναι το ίδιο με το τμήμα διευθύνσεως της εντολής άλματος χωρίς συνθήκη.

Διεύθυνση μνήμης	Περιεχόμενο: Κώδικας εντολής	Περιεχόμενο: Τμήμα διεύθυνσης
000000000000	0111	000000000010
000000000001	0000	000000001011
000000000010	0111	000000000000
000000000011	0000	000000001000

Ας δούμε τι θα γίνει, όταν εκτελεστεί το πρόγραμμα του σχήματος. Αρχικά ο μετρητής προγράμματος έχει την τιμή 0. Έτσι κατά τη φάση ανάκλησης

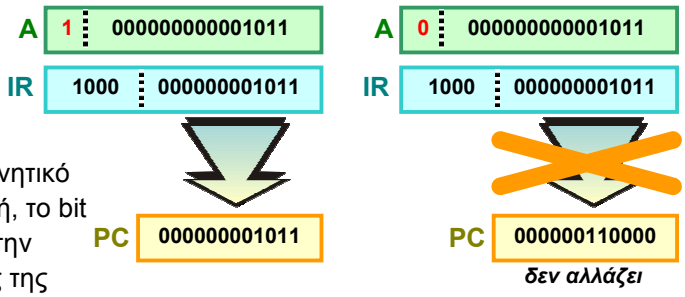
[1] Τα 2ms στην πραγματικότητα είναι πολύ μεγάλος χρόνος για μια εντολή. Οι σύγχρονοι επεξεργαστές εκτελούν χιλιάδες εντολές σ' αυτό το χρονικό διάστημα.

διαβάζεται από τη μνήμη η εντολή που βρίσκεται στη θέση μνήμης 000000000000, και ο μετρητής προγράμματος αυξάνεται κατά 1, παίρνοντας την τιμή 000000000001. Η εντολή που θα εκτελεστεί είναι η 0111 000000000010. Πρόκειται για μια εντολή άλματος χωρίς συνθήκη και η εκτέλεσή της έχει σαν αποτέλεσμα την εγγραφή στον PC του τμήματος διευθύνσεως του IR, το οποίο είναι το 000000000010. Η τιμή του PC λοιπόν αλλάζει και γίνεται 000000000010.

Η εκτέλεση της εντολής ολοκληρώθηκε και ήρθε η ώρα για την επόμενη φάση ανάκλησης. Τώρα το περιεχόμενο του PC «δείχνει» στην τρίτη εντολή του προγράμματος, και όχι στη δεύτερη όπως θα γινόταν αν δεν είχε εκτελεστεί η εντολή άλματος.

Β) Την *εντολή άλματος υπό συνθήκη αρνητικού περιεχομένου του συσσωρευτή* (conditional jump instruction) με κώδικα εντολής 1000.

Όταν εκτελείται η εντολή άλματος υπό συνθήκη αρνητικού περιεχομένου του συσσωρευτή, το αποτέλεσμα εξαρτάται από την τιμή του συσσωρευτή Α.



- Αν το περιεχόμενο του συσσωρευτή είναι αρνητικό (δηλαδή το αριστερότερο bit του συσσωρευτή, το bit του προσήμου, είναι 1), εκτελείται το άλμα στην εντολή που υποδεκνύει το τμήμα διεύθυνσης της εντολής, γράφοντας στο μετρητή προγράμματος το περιεχόμενο του τμήματος διεύθυνσης.
- Αν το περιεχόμενο του συσσωρευτή είναι 0 ή θετικός αριθμός, δηλαδή το αριστερότερό του bit έχει την τιμή 0, τότε η εντολή δεν προκαλεί την εκτέλεση καμιάς λειτουργίας.

Ας δούμε την εκτέλεση ενός προγράμματος, που υπολογίζει την απόλυτη τιμή του αριθμού που βρίσκεται στη θέση $20_{(10)} = 10100_{(2)}$ της μνήμης και την αποθηκεύει στη θέση $50_{(10)} = 110010_{(2)}$ της μνήμης.

Στο σχήμα βλέπουμε ότι ο αριθμός που είναι αποθηκευμένος στη θέση 20 είναι θετικός, αφού το πιο σημαντικό του ψηφίο έχει την τιμή 0. Έτσι, όταν εκτελεστεί το πρόγραμμα, η θέση 50 της μνήμης πρέπει να περιέχει τον αριθμό αυτό.

Διεύθυνση μνήμης	Περιεχόμενο: Κώδικας εντολής	Περιεχόμενο: Τμήμα διεύθυνσης
000000000000	0001	000000010100
000000000001	1000	000000000100
000000000010	0010	000000110010
000000000011	0000	000000001000
000000000100	0001	000000000111
000000000101	0100	000000010100
000000000110	0111	000000000010
000000000111	0000	000000000000
000000010100	0000000010000000	
000000110010	0000000000000000	

Οι εντολές του προγράμματος είναι οι εξής:

Εντολή 0: Ο κώδικας εντολής είναι 0001 και το τμήμα διευθύνσεως έχει την τιμή $10100_{(2)} = 20_{(10)}$. Η εντολή σημαίνει «φόρτωσε το συσσωρευτή Α με το περιεχόμενο της θέσης μνήμης 20».

Εντολή 1: Ο κώδικας εντολής είναι 1000 και το τμήμα διευθύνσεως έχει την τιμή $100_{(2)} = 4_{(10)}$. Αυτή είναι μια εντολή άλματος υπό συνθήκη, και σημαίνει «αν ο Α είναι αρνητικός, πήγαινε στην εντολή 4, αλλιώς μην κάνεις τίποτα».

Εντολή 2: Ο κώδικας εντολής είναι 0010 και το τμήμα διευθύνσεως έχει την τιμή $110010_{(2)} = 50_{(10)}$, πρόκειται δηλαδή για την εντολή «αποθήκευσε το περιεχόμενο του συσσωρευτή Α στη θέση μνήμης 50».

Εντολή 3: Η εντολή αυτή έχει κώδικα 0000, είναι δηλαδή μία εντολή STOP.

Εντολή 4: Έχει κώδικα 0001 και τμήμα διευθύνσεως $111_{(2)} = 7_{(10)}$. Η εντολή αυτή θα φέρει στον Α το περιεχόμενο της θέσης 7 της μνήμης, η οποία όπως βλέπουμε έχει την τιμή 0. Έτσι πρακτικά η εντολή αυτή θέτει τον Α στην τιμή 0.

Εντολή 5: Η εντολή αυτή με κωδικό 0100 αφαιρεί από το συσσωρευτή (που έχει την τιμή 0) το περιεχόμενο της θέσης μνήμης 20, υπολογίζοντας έτσι την αντίθετη τιμή του.

Εντολή 6: Ο κώδικας εντολής είναι 0111 0010 και το τμήμα διευθύνσεως έχει την τιμή $10_{(2)} = 2_{(10)}$, πρόκειται δηλαδή για άλμα χωρίς συνθήκη στην εντολή 2.

Εντολή		Λειτουργία της εντολής	Καταχωρητές της ΚΜΕ
0	ανάκληση	Το περιεχόμενο της θέσης μνήμης που δείχνει ο PC μεταφέρεται στον IR. Ο PC αυξάνεται κατά 1.	(PC) = 000000000001 (IR) = 0001000000010100 (A) = 0000000000000000
	εκτέλεση	Ο κώδικας εντολής ($0001_{(2)}$) του IR αποκωδικοποιείται και βρίσκεται ότι είναι η εντολή (A) ← (N), δηλαδή (A) ← ($10100_{(2)}$). Ο συσσωρευτής Α φορτώνεται με το περιεχόμενο της θέσης μνήμης $10100_{(2)}$.	(PC) = 000000000001 (IR) = 0001000000010100 (A) = 0000000010000000
1	ανάκληση	Το περιεχόμενο της θέσης μνήμης που δείχνει ο PC μεταφέρεται στον IR. Ο PC αυξάνεται κατά 1.	(PC) = 000000000010 (IR) = 1000000000000100 (A) = 0000000010000000
	εκτέλεση	Ο κώδικας εντολής ($1000_{(2)}$) του IR αποκωδικοποιείται και βρίσκεται ότι είναι η εντολή άλματος υπό συνθήκη. Το αριστερότερο bit του συσσωρευτή είναι 0, έτσι δε γίνεται καμία άλλη λειτουργία.	(PC) = 000000000010 (IR) = 1000000000000100 (A) = 0000000010000000
2	ανάκληση	Το περιεχόμενο της θέσης μνήμης που δείχνει ο PC μεταφέρεται στον IR. Ο PC αυξάνεται κατά 1.	(PC) = 000000000011 (IR) = 0010000000110010 (A) = 0000000010000000
	εκτέλεση	Ο κώδικας εντολής ($0010_{(2)}$) του IR αποκωδικοποιείται και βρίσκεται ότι είναι η εντολή (N) ← (A), δηλαδή ($110010_{(2)}$) ← (A). Το περιεχόμενο του συσσωρευτή καταχωρείται στη θέση μνήμης $110010_{(2)}$.	(PC) = 000000000011 (IR) = 0010000000110010 (A) = 0000000010000000
3	ανάκληση	Το περιεχόμενο της θέσης μνήμης που δείχνει ο PC μεταφέρεται στον IR. Ο PC αυξάνεται κατά 1.	(PC) = 000000000100 (IR) = 0000000000001000 (A) = 0000000010000000
	εκτέλεση	Ο κώδικας εντολής ($0000_{(2)}$) του IR αποκωδικοποιείται και βρίσκεται ότι είναι η εντολή STOP. Η λειτουργία του υπολογιστή σταματά.	

Μετά την εκτέλεση της τελευταίας εντολής η θέση μνήμης 50 περιέχει την τιμή 0000000010000000, η οποία είναι η απόλυτη τιμή του αριθμού που περιέχεται στη θέση 20 της μνήμης.

Ας δούμε τι γίνεται αν στη θέση μνήμης 20 περιέχεται ένας αρνητικός αριθμός, π.χ. ο $-14_{(10)} = 1111111111110010_{(2)}$.

Εντολή	Λειτουργία της εντολής	Καταχωρητές της ΚΜΕ
0 ανάκληση	Το περιεχόμενο της θέσης μνήμης που δείχνει ο PC μεταφέρεται στον IR. Ο PC αυξάνεται κατά 1.	(PC) = 000000000001 (IR) = 0001000000010100 (A) = 0000000000000000
0 εκτέλεση	Ο κώδικας εντολής ($0001_{(2)}$) του IR αποκωδικοποιείται και βρίσκεται ότι είναι η εντολή (A) ← (N), δηλαδή (A) ← ($10100_{(2)}$). Ο συσσωρευτής Α φορτώνεται με το περιεχόμενο της θέσης μνήμης $10100_{(2)}$.	(PC) = 000000000001 (IR) = 0001000000010100 (A) = 1111111111110010
1 ανάκληση	Το περιεχόμενο της θέσης μνήμης που δείχνει ο PC μεταφέρεται στον IR. Ο PC αυξάνεται κατά 1.	(PC) = 000000000010 (IR) = 1000000000000100 (A) = 1111111111110010
1 εκτέλεση	Ο κώδικας εντολής ($1000_{(2)}$) του IR αποκωδικοποιείται και βρίσκεται ότι είναι η εντολή άλματος υπό συνθήκη. Το αριστερότερο bit του συσσωρευτή είναι 1, έτσι εκτελείται η εντολή άλματος. Το τμήμα διεύθυνσης του IR μεταφέρεται στον PC, που τώρα δείχνει στην εντολή 4 και όχι στη 2 που έδειχνε μετά τη φάση ανάκλησης.	(PC) = 000000000100 (IR) = 1000000000000100 (A) = 1111111111110010
2 ανάκληση	Το περιεχόμενο της θέσης μνήμης που δείχνει ο PC μεταφέρεται στον IR. Ο PC αυξάνεται κατά 1.	(PC) = 000000000101 (IR) = 0001000000000111 (A) = 1111111111110010
2 εκτέλεση	Ο κώδικας εντολής ($0001_{(2)}$) του IR αποκωδικοποιείται και βρίσκεται ότι είναι η εντολή (A) ← (N), δηλαδή (A_0) ← ($111_{(2)}$). Το περιεχόμενο της θέσης μνήμης $111_{(2)}$ μεταφέρεται στον Α.	(PC) = 000000000101 (IR) = 0001000000000111 (A) = 0000000000000000
3 ανάκληση	Το περιεχόμενο της θέσης μνήμης που δείχνει ο PC μεταφέρεται στον IR. Ο PC αυξάνεται κατά 1.	(PC) = 000000000110 (IR) = 0100000000010100 (A) = 0000000000000000
3 εκτέλεση	Ο κώδικας εντολής ($0100_{(2)}$) του IR αποκωδικοποιείται και βρίσκεται ότι είναι η εντολή (A) ← (A) - (N), δηλαδή (A) ← (A) -($10100_{(2)}$). Το περιεχόμενο της θέσης μνήμης $10100_{(2)}$ αφαιρείται από τον Α που έχει την τιμή 0, δηλαδή υπολογίζεται ο αντίθετός της. Το αποτέλεσμα καταχωρείται στον Α.	(PC) = 000000000110 (IR) = 0100000000010100 (A) = 0000000000001110
4 ανάκληση	Το περιεχόμενο της θέσης μνήμης που δείχνει ο PC μεταφέρεται στον IR. Ο PC αυξάνεται κατά 1.	(PC) = 000000000111 (IR) = 0111000000000010 (A) = 0000000000001110
4 εκτέλεση	Ο κώδικας εντολής ($0111_{(2)}$) του IR αποκωδικοποιείται και βρίσκεται ότι πρόκειται για εντολή άλματος χωρίς συνθήκη. Το τμήμα διεύθυνσης του IR αντιγράφεται στον PC.	(PC) = 000000000010 (IR) = 0111000000000010 (A) = 0000000000001110
5 ανάκληση	Το περιεχόμενο της θέσης μνήμης που δείχνει ο PC μεταφέρεται στον IR. Ο PC αυξάνεται κατά 1.	(PC) = 000000000011 (IR) = 0010000000110010 (A) = 0000000000001110
5 εκτέλεση	Ο κώδικας εντολής ($0010_{(2)}$) του IR αποκωδικοποιείται και βρίσκεται ότι είναι η εντολή (N) ← (A), δηλαδή ($110010_{(2)}$) ← (A). Το περιεχόμενο του συσσωρευτή καταχωρείται στη θέση μνήμης $110010_{(2)}$.	(PC) = 000000000011 (IR) = 0010000000110010 (A) = 0000000000001110

6 ανά-κληση εκτέλεση	**Το περιεχόμενο της θέσης μνήμης που δείχνει ο PC μεταφέρεται στον IR. Ο PC αυξάνεται κατά 1.** **Ο κώδικας εντολής ($0000_{(2)}$) του IR αποκωδικοποιείται και βρίσκεται ότι είναι η εντολή STOP. Η λειτουργία του υπολογιστή σταματά.**	(PC) = 000000000100 (IR) = 0000000000001000 (A) = 0000000000001110

Αυτή τη φορά η θέση μνήμης 50 περιέχει την τιμή $0000000000001110_{(2)} = 14_{(2)}$, η οποία είναι και πάλι η απόλυτη τιμή του αριθμού που περιέχεται στη θέση 20 της μνήμης.

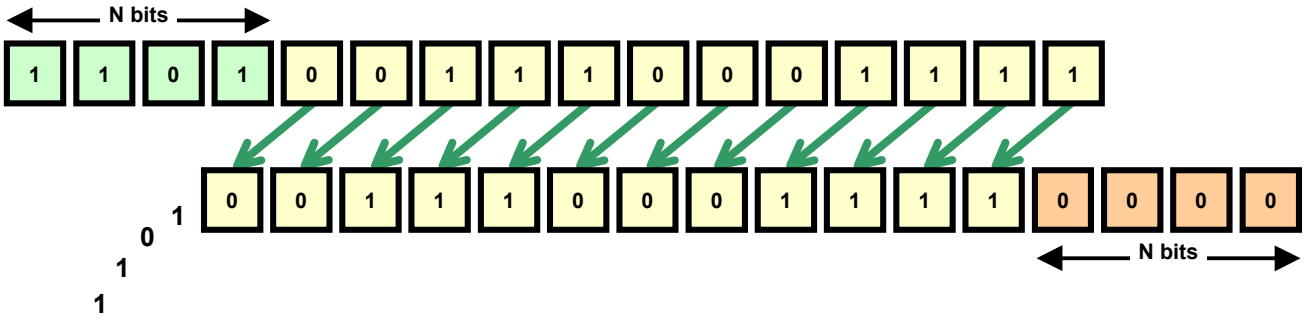
Εντολές ολίσθησης

Ο Άβακας διαθέτει δύο *εντολές ολίσθησης* (shift instructions).

Η πρώτη από αυτές ονομάζεται *εντολή ολίσθησης προς τα αριστερά* (shift left instruction) και έχει κώδικα εντολής 1011.

Η εντολή προκαλεί ολίσθηση του περιεχομένου του συσσωρευτή Α κατά Ν θέσεις προς τα αριστερά, όπου Ν είναι ο αριθμός που περιέχει το τμήμα διευθύνσεως της εντολής. Από τα δεξιά ο αριθμός συμπληρώνεται με μηδενικά bits.

> Αν ο Α περιέχει την τιμή 1101001110001111 και εκτελεστεί η εντολή 1011 000000000100 (ολίσθηση προς τα αριστερά κατά 4 θέσεις), το νέο του περιεχόμενο θα είναι το 0011100011110000.



Για να μηδενίσουμε λοιπόν τον Α μπορούμε να τον ολισθήσουμε προς τα αριστερά κατά 16 θέσεις, οπότε όλα του τα ψηφία θα χαθούν και θα αντικατασταθούν από μηδενικά.

Η δεύτερη εντολή ολίσθησης είναι η *εντολή ολίσθησης προς τα δεξιά* (shift right instruction) με κώδικα εντολής 1100. Η εντολή προκαλεί ολίσθηση του περιεχομένου του συσσωρευτή Α κατά Ν θέσεις προς τα δεξιά, όπου Ν είναι ο αριθμός που περιέχει το τμήμα διευθύνσεως της εντολής. Από τα αριστερά ο αριθμός συμπληρώνεται με μηδενικά bits.

> Αν ο Α περιέχει την τιμή **1101001110001**111 και εκτελεστεί η εντολή 1100 000000000100 (ολίσθηση προς τα δεξιά κατά 4 θέσεις), το νέο του περιεχόμενο θα είναι το 0000**1101001110001**.

Στις εντολές ολίσθησης το τμήμα διευθύνσεως δεν περιέχει μια διεύθυνση αλλά τον αριθμό των ολισθήσεων που θα πραγματοποιηθούν.

Εντολές εισόδου-εξόδου

Η επικοινωνία του ανθρώπου με τον υπολογιστή πραγματοποιείται με τις περιφερειακές μονάδες, μέσω της μονάδας εισόδου-εξόδου (ή *μονάδας E/E*). Υπάρχουν πολλών ειδών περιφερειακές μονάδες στους υπολογιστές: πληκτρολόγιο, οθόνη, ποντίκι, εκτυπωτής, κλπ.

Ο Άβακας διαθέτει ένα πληκτρολόγιο και μια οθόνη ως περιφερειακές μονάδες που είναι συνδεδεμένα με την μονάδα E/E. Για να επικοινωνεί με τις περιφερειακές μονάδες αυτές, και μέσω αυτών με το χρήστη, ο Άβακας περιλαμβάνει στο ρεπερτόριο εντολών του δύο *εντολές E/E* (input /output instructions ή I/O instructions).

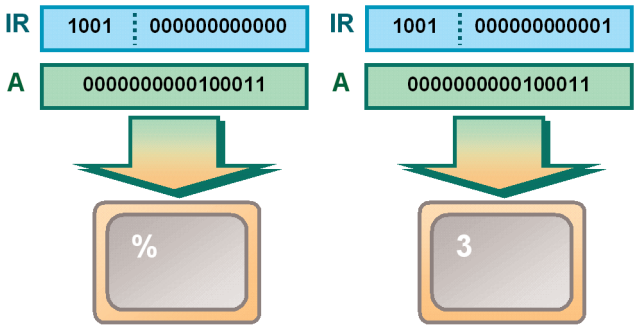
Η πρώτη είναι η *εντολή εξόδου* (output instruction) με κώδικα εντολής 1001.

– Αν το δεξιότερο bit της εντολής, δηλαδή το λιγότερο σημαντικό ψηφίο της, έχει την τιμή 0, τότε τα 7 δεξιότερα bits του συσσωρευτή A (bits 0-6) στέλνονται στην οθόνη όπου εμφανίζεται ο αντίστοιχος χαρακτήρας της κωδικοποίησης ASCII.

– Αν το δεξιότερο bit της εντολής έχει την τιμή 1, τότε τα 3 δεξιότερα bits του συσσωρευτή A (bits 0-2) στέλνονται στην οθόνη όπου εμφανίζεται το αντίστοιχο οκταδικό ψηφίο.

Αν ο συσσωρευτής έχει την τιμή 0000000000**0100011**, η εντολή 1001000000000000 προκαλεί την εμφάνιση στην οθόνη του χαρακτήρα που αντιστοιχεί στον κωδικό 0100011, δηλαδή του **%**.

Η εντολή 1001000000000001 θα έχει σαν αποτέλεσμα την εμφάνιση του οκταδικού αριθμού που αντιστοιχεί στα bits 011, δηλαδή του **3**.



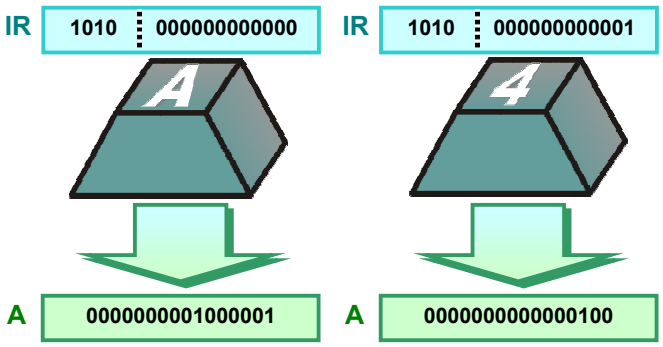
Η δεύτερη εντολή I/O είναι η *εντολή εισόδου* (input instruction) με κώδικα εντολής 1010.

Με την εντολή αυτή διαβάζεται από το πληκτρολόγιο μια πληροφορία και καταχωρείται στο συσσωρευτή. Ο υπολογιστής αναμένει το χρήστη να πατήσει ένα πλήκτρο, οπότε αποθηκεύεται στον καταχωρητή δεδομένων της μονάδας E/E ένας χαρακτήρας ASCII. Η πληροφορία που θα αποθηκευθεί στον A εξαρτάται από το δεξιότερο bit της εντολής:

– Αν το δεξιότερο bit της εντολής, δηλαδή το λιγότερο σημαντικό ψηφίο της, έχει την τιμή 0, τα 7 δεξιότερα bits του καταχωρητή δεδομένων της μονάδας E/E αποθηκεύονται στα 7 δεξιότερα bits του A.

– Αν το δεξιότερο bit της εντολής, δηλαδή το λιγότερο σημαντικό ψηφίο της, έχει την τιμή 1, τα 3 δεξιότερα bits του καταχωρητή δεδομένων της μονάδας E/E αποθηκεύονται στα 3 δεξιότερα bits του A και παριστάνουν έναν οκταδικό αριθμό. Αυτό γίνεται ακόμα και αν το πλήκτρο που πάτησε ο χρήστης δεν είναι ένα οκταδικό ψηφίο.

Και στις εντολές E/E το τμήμα διευθύνσεως της εντολής δεν δείχνει κάποια διεύθυνση· χρησιμοποιείται μόνο το δεξιότερο bit του για τον προσδιορισμό του τύπου των δεδομένων που μεταφέρονται, αριθμού ή χαρακτήρα.



Όταν εκτελεστεί η εντολή 1010000000000000, ο υπολογιστής περιμένει το χρήστη να πατήσει κάποιο πλήκτρο. Αν ο χρήστης πατήσει το πλήκτρο Α, που έχει ASCII κώδικα $65_{(10)}$, στα 7 τελευταία bits του Α θα καταχωρισθεί η τιμή $1000001_{(2)}$.

Αν εκτελεστεί η εντολή 1010000000000001, και ο χρήστης πατήσει το πλήκτρο 4, στα 3 τελευταία bits του Α θα καταχωρισθεί η τιμή $100_{(2)}$ = 4.

Ανακεφαλαίωση

Η εκτέλεση μίας εντολής μηχανής από ένα υπολογιστή περιλαμβάνει τη φάση ανάκλησης της εντολής στη μνήμη και τη φάση εκτέλεσης της εντολής.

Ο εκπαιδευτικός υπολογιστής Άβακας περιέχει ένα καταχωρητή στην αριθμητική-λογική μονάδα του, το συσσωρευτή Α. Διαθέτει 13 εντολές μηχανής, που αποτελούνται από εντολές αναφοράς στη μνήμη (περιλαμβάνουν και τις τέσσερις αριθμητικές πράξεις), εντολές άλματος χωρίς και υπό συνθήκη, εντολές ολίσθησης και εντολές εισόδου-εξόδου.

Γλωσσάριο όρων

Αποκωδικοποιητής	*Decoder*
Εντολή Άλματος	*Jump Instruction*
Εντολή Άλματος Υπό Συνθήκη	*Conditional Jump Instruction*
Εντολή Άλματος Χωρίς Συνθήκη	*Unconditional Jump Instruction*
Εντολή Αναφοράς Στη Μνήμη	*Memory Reference Instruction*
Εντολή Ε/Ε	*Input /Output Instruction ή I/O Instruction*
Εντολή Εισόδου	*Input Instruction*
Εντολή Εξόδου	*Output Instruction*
Εντολή Ολίσθησης	*Shift Instruction*
Εντολή Ολίσθησης Προς Τα Αριστερά	*Shift Left Instruction*
Εντολή Ολίσθησης Προς Τα Δεξιά	*Shift Right Instruction*
Κύκλος	*Cycle*
Κώδικας Εντολής	*Instruction Code*
Τμήμα Διευθύνσεως	*Address Part*
Φάση	*Phase*
Φάση ή Κύκλος Ανάκλησης	*Fetch Phase ή Cycle*
Φάση ή Κύκλος Εκτέλεσης	*Execution Phase ή Cycle*

Ερωτήσεις

? Ποιες είναι οι φάσεις εκτέλεσης μίας εντολής και τα τέσσερα βήματα της φάσης ανάκλησης;

? Τι εννοούμε με τον όρο «διεύθυνση εντολής» και με τον όρο «τμήμα διεύθυνσης εντολής»;

? Ποια είναι η λειτουργία των εντολών άλματος;

? Ποια είναι η λειτουργία των εντολών ολίσθησης;

Μάθημα 3.3 Συμβολική Γλώσσα

Σκοπός του μαθήματος

Σκοπός του μαθήματος αυτού είναι να περιγράψει τον προγραμματισμό ενός υπολογιστή σε συμβολική γλώσσα.

Τι θα μάθεις;

Όταν ολοκληρώσεις το μάθημα αυτό, θα μπορείς:

- *Να εξηγείς τις διαφορές της συμβολικής γλώσσας από τη γλώσσα μηχανής.*
- *Να περιγράφεις το ρόλο των ψευδοεντολών σε ένα πρόγραμμα.*
- *Να γράφεις απλά προγράμματα στη συμβολική γλώσσα του Άβακα.*

Μέχρι τώρα είδαμε παραδείγματα προγραμματισμού του Άβακα όπου τόσο οι εντολές όσο και οι διευθύνσεις μνήμης ήταν γραμμένες σε αριθμητική μορφή και μάλιστα στο δυαδικό σύστημα. Τα προγράμματα αυτά είναι γραμμένα σε *γλώσσα μηχανής* (machine language).

Είναι φανερό ότι η σύνταξη έστω και πολύ απλών προγραμμάτων στη γλώσσα μηχανής είναι εργασία κουραστική και απαιτεί χρόνο. Τα πράγματα θα ήταν πολύ πιο απλά αν γράφαμε τις εντολές μας με τρόπο «συμβολικό», που να μας θυμίζει αμέσως το νόημά τους.

> Η εντολή 0001 φορτώνει (**L**oa**D**s) το συσσωρευτή (**A**ccumulator) με μία τιμή. Ένα καλό μνημονικό όνομα για την εντολή αυτή είναι το **LDA**.

Μπορούμε επίσης να δίνουμε συμβολικά ονόματα στις διευθύνσεις της μνήμης, ώστε να δουλεύουμε με σύμβολα αντί με αριθμούς.

> Αν συμβολίσουμε τη διεύθυνση $000000100101_{(2)}$ με το όνομα «IN_NUM», τότε η εντολή 0001 000000100101 θα μπορούσε να γραφτεί συμβολικά σαν «**LDA IN_NUM**».

Με τον τρόπο αυτό μπορούμε να γράφουμε τις εντολές του προγράμματος μας σε μια άλλη γλώσσα, που ονομάζουμε *συμβολική γλώσσα* (assembly language).

Ένα πρόγραμμα γραμμένο σε συμβολική γλώσσα, για να μπορεί να εκτελεστεί από τον υπολογιστή, πρέπει να μεταφραστεί πρώτα σε γλώσσα μηχανής. Τη μετάφραση αυτή των προγραμμάτων από συμβολική γλώσσα σε γλώσσα μηχανής την πραγματοποιεί ο ίδιος ο υπολογιστής, με ένα ειδικό πρόγραμμα που είναι γραμμένο σε γλώσσα μηχανής και ονομάζεται *συμβολομεταφραστής* (assembler).

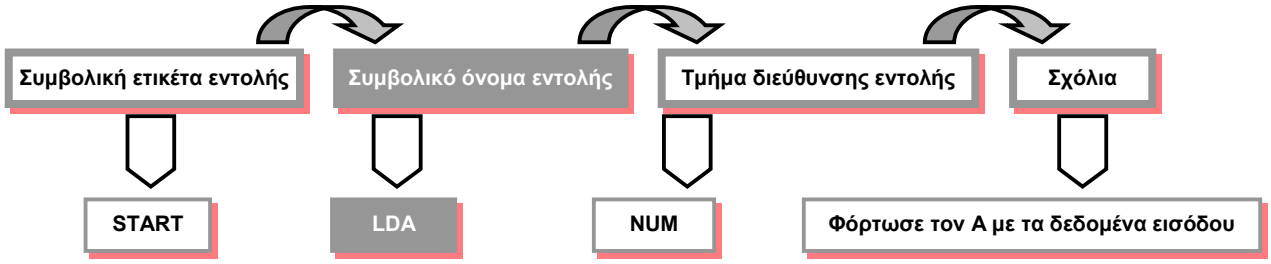
Ο συμβολομεταφραστής, εκτός από το να μετατρέπει το πρόγραμμα από τη συμβολική γλώσσα σε γλώσσα μηχανής, αναγνωρίζει τα λάθη («συντακτικά» ή «ορθογραφικά»), που μπορεί να υπάρχουν στις εντολές του προγράμματός μας.

Για κάθε εντολή από το ρεπερτόριο εντολών του υπολογιστή υπάρχει και η αντίστοιχη εντολή σε συμβολική γλώσσα. Επιπλέον όμως η συμβολική γλώσσα διαθέτει μερικές ακόμα εντολές, τις *ψευδοεντολές* (pseudo instructions). Οι ψευδοεντολές δεν εκτελούνται από τον υπολογιστή, αλλά χρησιμοποιούνται από το συμβολομεταφραστή κατά το μεταφραστικό στάδιο και διευκολύνουν τον προγραμματιστή όταν γράφει το πρόγραμμα.

Ας δούμε ποιες είναι οι συμβολικές μορφές των εντολών του Άβακα:

Συμβολικό όνομα	Κώδικας	Λειτουργία
HLT = **HaLT**	0000	STOP
LDA = **LoaD A**	0001	Φόρτωση του Α (A) ← (N)
STA = **STore to A**	0010	Αποθήκευση του Α (N) ← (A)
ADA = **ADd to A**	0011	Πρόσθεση στον Α (A) ← (A) + (N)
SBA = **SuBtract from A**	0100	Αφαίρεση από τον Α (A) ← (A) - (N)
MLA = **MuLtiply A**	0101	Πολλαπλασιασμός με τον Α (A) ← (A) * (N)
DVA = **DiVide A**	0110	Διαίρεση του Α (A) ← (A) / (N)
JMP = **JuMP**	0111	Άλμα χωρίς συνθήκη (PC) ← N
JAN = **Jump if A is Negative**	1000	Άλμα υπό συνθήκη (PC) ← N αν (A) < 0
OUT = **OUTput**	1001	Έξοδος δεδομένων στην οθόνη
INP = **INPut**	1010	Είσοδος δεδομένων από το πληκτρολόγιο
SAL = **Shift A Left**	1011	Ολίσθηση του Α αριστερά
SAR = **Shift A Right**	1100	Ολίσθηση του Α δεξιά

Κάθε εντολή σε ένα πρόγραμμα συμβολικής γλώσσας έχει την εξής μορφή:



Συμβολική ετικέτα εντολής: Το τμήμα αυτό της εντολής είναι προαιρετικό και αποτελεί το «όνομα» που δίνουμε στην εντολή για να μπορούμε να αναφερθούμε σε αυτή σε μία άλλη εντολή άλματος.

Συμβολικό όνομα της εντολής: Περιέχει μία από τις συμβολικές εντολές που είδαμε παραπάνω. Αυτό είναι το μόνο τμήμα της εντολής που υπάρχει υποχρεωτικά.

Τμήμα διεύθυνσης της εντολής: Στο τμήμα αυτό γράφουμε τα δεδομένα που αντιστοιχούν στο τμήμα διεύθυνσης της εντολής γλώσσας μηχανής.

Αν πρόκειται για μία εντολή αναφοράς στη μνήμη, αυτό το πεδίο πρέπει να περιέχει μία διεύθυνση στη μνήμη. Η διεύθυνση μπορεί να γραφτεί σαν αριθμός, σαν ετικέτα ή σαν μία αριθμητική παράσταση που συνδυάζει αριθμούς και ετικέτες.

> Η εντολή «STA 10» αποθηκεύει το περιεχόμενο του A στη θέση μνήμης με διεύθυνση 10. Αν αυτή η θέση μνήμης χαρακτηρίζεται από την ετικέτα SUM, τότε μπορούμε να γράψουμε την εντολή σαν «STA SUM» η οποία αποθηκεύει το περιεχόμενο του A στη θέση μνήμης που συμβολίζεται με την ετικέτα SUM. Παρακάτω θα δούμε πώς αντιστοιχίζουμε στο πρόγραμμά μας θέσεις μνήμης για δεδομένα με ετικέτες.
>
> Αν στη συνέχεια θέλουμε να μεταφέρουμε στον A τα περιεχόμενα της επόμενης θέσης μνήμης από τη SUM, θα γράψουμε «LDA SUM+1».

Αν πρόκειται για εντολή άλματος, το τμήμα διεύθυνσης θα πρέπει να αναφέρεται σε μια άλλη εντολή του προγράμματος. Έτσι μπορεί να περιέχει έναν αριθμό, μία ετικέτα εντολής, ή μία αριθμητική παράσταση με αριθμούς και ετικέτες εντολής.

> Η εντολή «JMP START» προκαλεί άλμα στην εντολή με ετικέτα START, ενώ η εντολή «JMP START+1» προκαλεί άλμα στην επόμενη από την εντολή με ετικέτα START.

Η ειδική ετικέτα εντολής «*» συμβολίζει την ίδια την εντολή.

> Ας δούμε το πρόγραμμα:
>
> ```
> MLA   10
> LDA   SUM
> JMP   *-2
> ```
>
> Όταν φθάσουμε στην τελευταία εντολή, θα εκτελεστεί ένα άλμα στην εντολή που βρίσκεται 2 θέσεις πιο πριν απ' αυτή, δηλαδή στην πρώτη εντολή «MLA 10».

Σύγκρινε το πρόγραμμα αυτό με το παράδειγμα της σελίδας 56.

Στις εντολές εισόδου-εξόδου, όπως έχουμε δει, το δεξιότερο bit του τμήματος διεύθυνσης προσδιορίζει, αν τα δεδομένα που θα μεταφερθούν από ή προς την περιφερειακή συσκευή θα ερμηνευθούν σαν αριθμός ή σαν χαρακτήρας. Στη συμβολική γλώσσα το τμήμα διεύθυνσης έχει αντίστοιχα το περιεχόμενο 0 ή 1.

> Με την εντολή:
>
> ```
> INP   1
> ```
>
> διαβάζουμε από το πληκτρολόγιο έναν οκταδικό αριθμό που αποθηκεύεται στο συσσωρευτή A.

Στις εντολές ολίσθησης το τμήμα διεύθυνσης περιέχει τον αριθμό των θέσεων κατά τις οποίες θέλουμε να ολισθήσουμε το συσσωρευτή.

> Η εντολή «SAR 12» ολισθαίνει το συσσωρευτή Α κατά 12 θέσεις προς τα δεξιά.

Σχόλια: Στο τελευταίο τμήμα, που είναι προαιρετικό, γράφουμε ένα κείμενο της επιλογής μας, το οποίο αποτελεί επεξήγηση για τη λειτουργία της εντολής. Ο συμβολομεταφραστής αγνοεί το τμήμα αυτό.

> Στην παρακάτω εντολή, το κείμενο «Αποθήκευση του Α» αναγνωρίζεται από το συμβολομεταφραστή σαν σχόλιο, και έτσι αγνοείται.
>
> ```
> STA   17    Αποθήκευση του Α
> ```

Ψευδοεντολές

Οι ψευδοεντολές δε μεταφράζονται από το συμβολομεταφραστή σε εντολές μηχανής, αλλά χρησιμοποιούνται κατά τη μετάφραση του προγράμματος. Έχουν και αυτές ένα συμβολικό όνομα που τοποθετείται στο αντίστοιχο τμήμα της εντολής αλλά δεν αντιστοιχεί με κάποιο κώδικα εντολής. Οι ψευδοεντολές μπορούν και αυτές να έχουν συμπληρωμένα και τα υπόλοιπα τμήματά τους.

Στην αρχή του προγράμματος γράφουμε μία ψευδοεντολή NAM (από τη λέξη name = όνομα). Στο τμήμα διεύθυνσής της γράφουμε μία λέξη που αποτελεί το όνομα του προγράμματος.

> Το πρόγραμμα που υπολογίζει την απόλυτη τιμή ενός αριθμού θα ξεκινά με την ψευδοεντολή «NAM ABS» (από τη λέξη absolute = απόλυτος).

Για να ορίσουμε θέσεις μνήμης που περιέχουν κάποιους αριθμούς και έχουν ένα συμβολικό όνομα, χρησιμοποιούμε την ψευδοεντολή CON (από τη λέξη constant = σταθερά). Στην ψευδοεντολή αυτή πρέπει να δώσουμε μία ετικέτα, για να μπορούμε να αναφερόμαστε στις θέσεις μνήμης αυτές, και αντί για πεδίο διεύθυνσης γράφουμε μία σειρά από αριθμούς.

Η πραγματική διεύθυνση μνήμης που αντιστοιχεί στην ετικέτα δε μας ενδιαφέρει· την επιλέγει ο συμβολομεταφραστής.

> Με την ψευδοεντολή «DATA CON 0,8,27» ορίζουμε τρεις θέσεις μνήμης, που περιέχουν τους αριθμούς 0, 8 και 27 αντίστοιχα. Στην πρώτη θέση αναφερόμαστε ως «DATA», στη δεύτερη ως «DATA+1» και στην τρίτη ως «DATA+3».
>
> Αν μέσα στο πρόγραμμά μας υπάρχει η εντολή «LDA DATA+1», αυτή θα μεταφέρει στον Α το περιεχόμενο της θέσης μνήμης DATA+1, δηλαδή την τιμή 8.

Όπως συμβολίζουμε θέσεις μνήμης που περιέχουν δεδομένα, μπορούμε να συμβολίσουμε θέσεις μνήμης όπου θα αποθηκευθούν τα αποτελέσματα που υπολογίζει το πρόγραμμα. Αυτό γίνεται με μία ψευδοεντολή RES (από το αγγλικό ρήμα reserve = κρατώ, κάνω κράτηση).

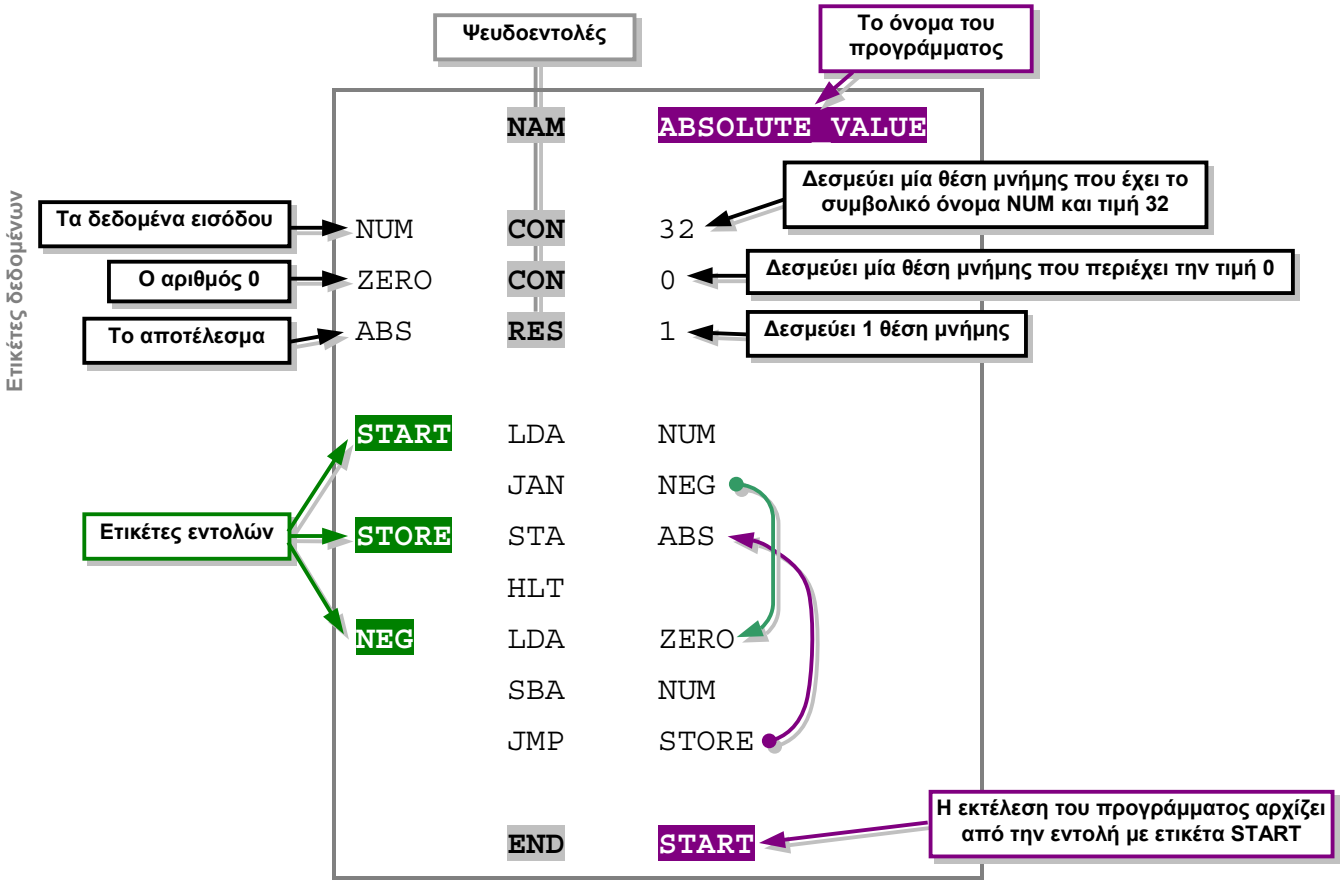
Στην ψευδοεντολή RES δίνουμε μία ετικέτα, για να αναφερόμαστε στις θέσεις μνήμης που κρατεί. Το πεδίο διεύθυνσής της περιέχει έναν αριθμό, το πλήθος των θέσεων μνήμης που θέλουμε να κρατήσουμε.

> Με την ψευδοεντολή «SUM RES 2» κρατάμε δύο θέσεις μνήμης για να αποθηκεύσουμε αποτελέσματα. Για να καταχωρήσουμε το περιεχόμενο του A στην πρώτη από αυτές τις θέσεις, θα γράψουμε την εντολή «STA SUM», ενώ για να το αποθηκεύσουμε στη δεύτερη θέση θα γράψουμε «STA SUM+1».

Το τέλος του προγράμματος συμβολίζεται από μία ψευδοεντολή END.Στο τμήμα διεύθυνσής της γράφουμε την ετικέτα της πρώτης εντολής του προγράμματος. Έτσι μπορούμε στην αρχή του προγράμματος σε συμβολική γλώσσα να τοποθετήσουμε δεδομένα, αλλά η εκτέλεσή του να αρχίσει από το σημείο που θα υποδείξουμε με την ψευδοεντολή END.

Δεν πρέπει να μπερδέψουμε την ψευδοεντολή END με την εντολή HLT. Η πρώτη απευθύνεται μόνο στο συμβολομεταφραστή και υποδεικνύει την ολοκλήρωση της μετάφρασης. Η δεύτερη απευθύνεται και στον υπολογιστή και προκαλεί παύση της λειτουργίας του.

Ας δούμε ένα πλήρες πρόγραμμα γραμμένο σε συμβολική γλώσσα, που υπολογίζει την απόλυτη τιμή ενός αριθμού. Ο αριθμός βρίσκεται στη θέση μνήμης που έχει οριστεί με την ψευδοεντολή CON και έχει ετικέτα NUM, και το αποτέλεσμα αποθηκεύεται στη θέση μνήμης που έχει οριστεί με την ψευδοεντολή RES και έχει ετικέτα ABS. Στο πρόγραμμα αυτό οι εντολές άλματος χρησιμοποιούνται για τη διακλάδωση της ροής εκτέλεσής του.



Πρόγραμμα με βρόχο

Οι εντολές άλματος έχουν μία πολύ σημαντική χρησιμότητα: βοηθούν να κατασκευάζουμε προγράμματα που εκτελούν επαναληπτικά κάποιες λειτουργίες.

> Η επαναληπτική εκτέλεση μίας ομάδας εντολών σε ένα πρόγραμμα ονομάζεται *βρόχος* (loop).

Αν μάλιστα η επαναληπτική εκτέλεση του βρόχου δε σταματά, τότε λέμε ότι το πρόγραμμα έχει πέσει σε *ατέρμονα βρόχο* (infinite loop).

Μία διαδικασία που μπορεί να υλοποιηθεί με ένα βρόχο, είναι η ανάγνωση από το πληκτρολόγιο ενός αριθμού, bit προς bit. Η διαδικασία ανάγνωσης ενός bit είναι η ίδια, ανεξάρτητα από τη θέση του μέσα στη λέξη. Έτσι μπορούμε να κατασκευάσουμε ένα βρόχο που θα επαναληφθεί 16 φορές και θα διαβάζει ένα bit από το πληκτρολόγιο.

```
INP  1
SAL  15
SAR  15
```

Επειδή από το πληκτρολόγιο μπορούμε να διαβάσουμε ή χαρακτήρες ή οκταδικούς αριθμούς, και όχι μόνο 1 bit, θα επιλέξουμε την ανάγνωση οκταδικών αριθμών, αλλά θα χρησιμοποιούμε μόνο το τελευταίο δυαδικό ψηφίο τους. Για να το κάνουμε αυτό, διαβάζουμε τον οκταδικό αριθμό στο συσσωρευτή Α, τον ολισθαίνουμε 15 θέσεις προς τα αριστερά (οπότε όλα τα ψηφία του εκτός από το τελευταίο χάνονται) και μετά τον ολισθαίνουμε 15 θέσεις προς τα δεξιά, οπότε επανέρχεται το τελευταίο bit στη σωστή θέση και συμπληρώνεται από τα αριστερά με μηδενικά.

> Αν ο συσσωρευτής Α περιέχει αρχικά τον αριθμό 0010000001010001 και διαβαστεί ο οκταδικός αριθμός $3_{(8)}=011_{(2)}$ από το πληκτρολόγιο, η νέα τιμή του Α θα είναι 0010000001010011. Ολισθαίνοντας τον Α κατά 15 θέσεις προς τα αριστερά, θα πάρει τη νέα τιμή 1000000000000000. Όταν τον ολισθήσουμε ξανά 15 θέσεις προς τα δεξιά, η τιμή του θα γίνει 0000000000000001, έχει διατηρηθεί δηλαδή μόνο το δεξιότερο bit.

Ο αριθμός που διαβάζουμε θα αποθηκευθεί σε μία θέση μνήμης στην οποία θα δώσουμε το όνομα NUM. Αρχικά η θέση μνήμης NUM θα έχει την τιμή 0. Επίσης θα χρειαστούμε άλλη μία θέση μνήμης, με το όνομα COUNTER, στην οποία θα μετράμε το πλήθος των επαναλήψεων, ώστε να σταματήσουμε μετά από 16 επαναλήψεις, όσες και τα bits μίας λέξης. Και η μεταβλητή COUNTER αρχικά θα έχει την τιμή 0. Σε γενικές γραμμές, η λειτουργία του προγράμματος θα είναι η εξής:

Βήμα 1: Φόρτωσε την τρέχουσα τιμή του αριθμού από τη θέση μνήμης NUM.

Βήμα 2: Ολίσθησε προς τα αριστερά την τρέχουσα τιμή του αριθμού κατά 1, για να κάνεις «χώρο» για το νέο bit.

Βήμα 3: Αποθήκευσε την τρέχουσα τιμή του αριθμού στη θέση μνήμης NUM.

Βήμα 4: Διάβασε ένα bit από το πληκτρολόγιο.

Βήμα 5: Πρόσθεσε την τρέχουσα τιμή του αριθμού NUM στο bit αυτό, για να πάρεις το νέο αριθμό.

Βήμα 6: Αποθήκευσε τη νέα τιμή του αριθμού στη θέση μνήμης NUM.

Βήμα 7: Φόρτωσε την τιμή του μετρητή COUNTER στο συσσωρευτή.

Βήμα 8: Πρόσθεσε 1 στην τιμή του μετρητή.

Βήμα 9: Αποθήκευσε τη νέα τιμή του συσσωρευτή στη θέση μνήμης COUNTER.

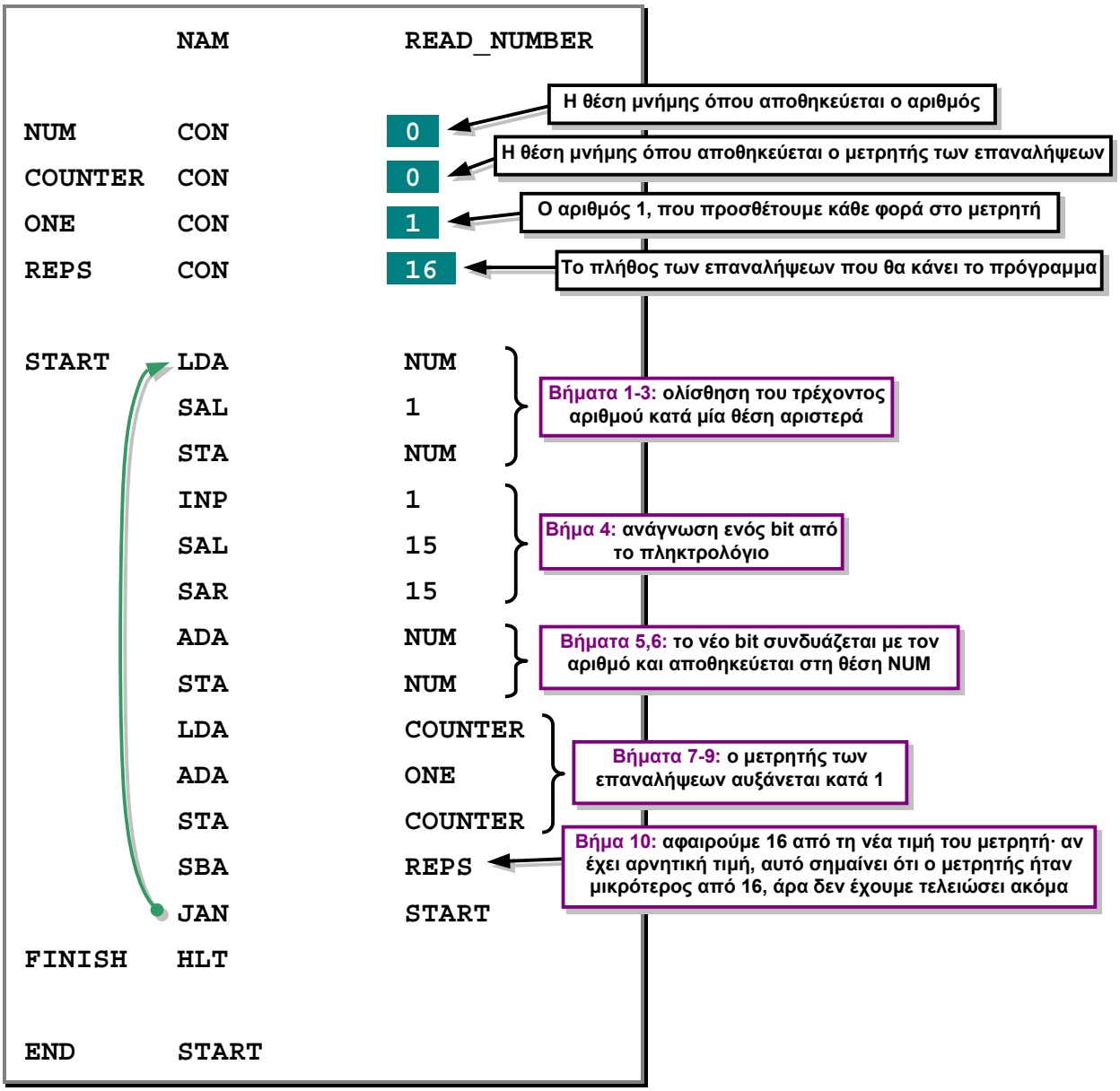
Βήμα 10: Αφαίρεσε 16 από την τιμή του συσσωρευτή.

Βήμα 11: Αν η τιμή του Α είναι αρνητική, τότε το πρόγραμμα πρέπει να τελειώσει, οπότε γίνεται άλμα στην τελευταία εντολή του, το βήμα 13.

Βήμα 12: Αν φθάσουμε στο βήμα αυτό, το πρόγραμμα δεν έχει τελειώσει ακόμα, έτσι πάμε ξανά στο βήμα 1.

Βήμα 13: STOP.

Ας δούμε και ολόκληρο το πρόγραμμα σε συμβολική γλώσσα:



Επειδή ο προγραμματισμός σε γλώσσα μηχανής είναι δύσκολος, γράφουμε τις εντολές του προγράμματος με συμβολικά ονόματα, δηλαδή σε συμβολική γλώσσα. Ένα ειδικό πρόγραμμα, ο συμβολομεταφραστής αναλαμβάνει να μετατρέψει το πρόγραμμα από τη συμβολική γλώσσα σε γλώσσα μηχανής. Ο συμβολομεταφραστής επίσης παρέχει στον προγραμματιστή τη δυνατότητα να ορίζει θέσεις μνήμης με συμβολικά ονόματα, οι οποίες είτε περιέχουν αρχικά κάποιες τιμές, είτε χρησιμοποιούνται για την αποθήκευση αποτελεσμάτων.

Ατέρμων Βρόχος	*Infinite Loop*
Γλώσσα Μηχανής	*Machine Language*
Συμβολική Γλώσσα	*Assembly Language*
Συμβολομεταφραστής	*Assembler*
Ψευδοεντολή	*Pseudo Instruction*

Ερωτήσεις

- ? Σε τι μας βοηθά ο προγραμματισμός σε συμβολική γλώσσα;
- ? Ποια εντολή παριστάνεται με το συμβολικό όνομα LDA, STA, DVA;
- ? Ποιες εντολές παριστάνονται με τα ονόματα JMP, JAN;
- ? Τι περιέχει το πρώτο πεδίο της εντολής συμβολικής γλώσσας;
- ? Ποια είναι η λειτουργία της ψευδοεντολής NAM;
- ? Σε τι διαφέρει η ψευδοεντολή CON από την ψευδοεντολή RES;
- ? Τι είναι ο βρόχος;

Μάθημα 3.4 Από τη Γλώσσα Μηχανής στα Προγράμματα Εφαρμογής

Σκοπός του μαθήματος

Σκοπός του μαθήματος αυτού είναι να περιγράψει τις τεχνικές και τα εργαλεία που διαθέτουν οι προγραμματιστές για να γράφουν γρήγορα και εύκολα τα προγράμματά τους.

Τι θα μάθεις;

Όταν ολοκληρώσεις το μάθημα αυτό, θα μπορείς:

- *Να ορίζεις τις γλώσσες προγραμματισμού*
- *Να περιγράφεις τη λειτουργία του διερμηνέα, του μεταφραστή και του συμβολομεταφραστή*
- *Να εξηγείς πώς λειτουργεί ένας μεταφραστής*

Γλώσσες προγραμματισμού

Ο προγραμματισμός σε γλώσσα μηχανής είναι ένα αρκετά χρονοβόρο έργο, στο οποίο υποβόσκουν κίνδυνοι για πολλά προγραμματιστικά λάθη:

- Είναι πολύ εύκολο να γράψει κανείς ένα λανθασμένο κώδικα εντολής, όταν μάλιστα αυτός αποτελείται από αρκετά bits. Μπορεί να γράψει π.χ. «100101» αντί για «101001» αλλοιώνοντας τελείως το νόημα της εντολής και, βέβαια, το αποτέλεσμα του προγράμματος.
- Επειδή οι εντολές αναφέρονται σε συγκεκριμένες διευθύνσεις μνήμης τόσο για τα δεδομένα τους όσο και για τις εντολές άλματος, ο προγραμματιστής μπορεί να γράψει εύκολα κάποια λανθασμένη διεύθυνση. Τα λάθη αυτά, όπως και τα λάθη στον κώδικα εντολής, είναι δύσκολο και κουραστικό να εντοπιστούν μέσα σε ένα πρόγραμμα.
- Σε όλες τις εντολές άλματος η διεύθυνση προορισμού είναι καθορισμένη με ακρίβεια. Αν όμως το πρόγραμμα μεταβληθεί με την εισαγωγή μίας επιπλέον εντολής ή την αφαίρεση άλλης, όλες οι εντολές άλματος που επηρεάζονται πρέπει να μετατραπούν.

Επίσης όλα τα προγράμματα σε γλώσσα μηχανής περιέχουν ένα πολύ μεγάλο αριθμό εντολών, διότι ακόμα και για τα απλούστερα έργα, όπως μία απλή αριθμητική παράσταση απαιτούνται αρκετές εντολές. Έτσι ο προγραμματιστής σπαταλά πολύ χρόνο γράφοντάς τα και ακόμα περισσότερο ελέγχοντάς τα.

Τα προβλήματα που αναφέρονται λύνονται εν μέρει με τη συμβολική γλώσσα, αν και σε αυτή δεν είναι πολύ δύσκολο να κάνει ένας προγραμματιστής ένα λάθος. Παραμένει όμως το γεγονός ότι τα προγράμματα είναι πολύ ογκώδη, και βέβαια οι προγραμματιστές πρέπει να θυμούνται πάντα τα συμβολικά ονόματα για όλες τις εντολές του υπολογιστή.

Ένα πρόγραμμα σε συμβολική γλώσσα μπορεί να εκτελεστεί μόνο στον τύπο υπολογιστή για τον οποίο γράφτηκε. Για να εκτελεστεί σε οποιοδήποτε άλλο υπολογιστή πρέπει να ξαναγραφεί από την αρχή.

Για να αντιμετωπιστούν όλα αυτά τα προβλήματα, δημιουργήθηκαν οι *γλώσσες προγραμματισμού υψηλού επιπέδου* (high level programming languages). Στις γλώσσες αυτές, που είναι πιο κοντά στην ανθρώπινη γλώσσα και διευκολύνουν τον άνθρωπο-προγραμματιστή, κάθε εντολή αποτελεί μία πολύπλοκη λειτουργία και αντιστοιχεί σε πολλές εντολές γλώσσας μηχανής. Επίσης οι γλώσσες προγραμματισμού δεν εξαρτώνται από κάποια συγκεκριμένη αρχιτεκτονική υπολογιστή.

Μεταφραστές

Οι γλώσσες προγραμματισμού κάνουν το έργο των προγραμματιστών πιο εύκολο, αλλά δεν μπορούν να εκτελεστούν από τον υπολογιστή, που «καταλαβαίνει» μόνο εντολές μηχανής. Το ίδιο ισχύει βέβαια και για τα προγράμματα σε συμβολική γλώσσα. Το χάσμα αυτό γεφυρώνουν οι *μεταφραστές* (translators), ειδικά προγράμματα που μετατρέπουν τα προγράμματα από μία μορφή κατανοητή στον άνθρωπο στη μορφή που μπορεί να εκτελεστεί από τον υπολογιστή.

Οι μεταφραστές διακρίνονται σε τρεις μεγάλες κατηγορίες:

Α. Οι *συμβολομεταφραστές* (assemblers) μετατρέπουν προγράμματα από συμβολική γλώσσα σε γλώσσα μηχανής. Επειδή η αντιστοιχία μεταξύ των συμβολικών εντολών και των εντολών μηχανής είναι ένα προς ένα, το έργο του συμβολομεταφραστή είναι:

(1) Να αντικαθιστά κάθε συμβολική εντολή με την αντίστοιχη εντολή μηχανής **(2)** να υπολογίζει τις διευθύνσεις κάθε εντολής για να θέτει σωστά τις διευθύνσεις προορισμού στις εντολές άλματος και **(3)** να υπολογίζει τις διευθύνσεις των δεδομένων του προγράμματος και να αντικαθιστά τις ετικέτες που αναφέρονται σε δεδομένα με τις πραγματικές διευθύνσεις.

Επειδή υπάρχει αυτή η στενή σχέση μεταξύ συμβολικής γλώσσας και γλώσσας μηχανής, ο συμβολομεταφραστής λειτουργεί μόνο στον υπολογιστή για τον οποίο γράφτηκε.

Β. Οι *μεταγλωττιστές* (compilers) διαβάζουν ένα πρόγραμμα σε μία γλώσσα προγραμματισμού και το μετατρέπουν στην αντίστοιχη ακολουθία από εντολές μηχανής. Κάθε εντολή της γλώσσας προγραμματισμού αντικαθίσταται από πολλές εντολές μηχανής.

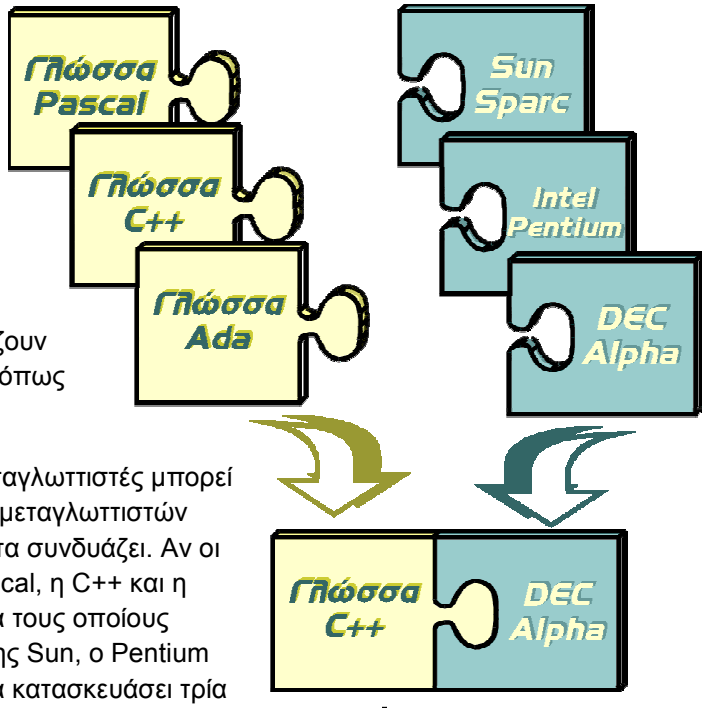
Ο μεταγλωττιστής είναι πιο πολύπλοκο πρόγραμμα από το συμβολομεταφραστή, γιατί αναγνωρίζει και μετατρέπει πιο πολύπλοκες γλώσσες. Μερικές φορές μάλιστα οι μεταγλωττιστές παράγουν ένα πρόγραμμα σε συμβολική γλώσσα, το οποίο μετά αναλαμβάνει ο συμβολομεταφραστής.

Γ. Οι *διερμηνείς* (interpreters) συγχρόνως μεταφράζουν και εκτελούν ένα πρόγραμμα εντολή προς εντολή. Αυτό σημαίνει ότι κάθε εντολή του προγράμματος μεταφράζεται μόνο όταν έρθει η ώρα να εκτελεστεί, αλλά και ότι μπορεί να μεταφραστεί πολλές φορές, μία για κάθε φορά που εκτελείται.

Δομή του μεταγλωττιστή

Ένας μεταγλωττιστής γενικά αποτελείται από δύο μέρη: το ένα τμήμα είναι αφιερωμένο στην αναγνώριση και ανάλυση του προγράμματος στη γλώσσα υψηλού επιπέδου και το άλλο αναλαμβάνει την παραγωγή της συμβολικής γλώσσας και της

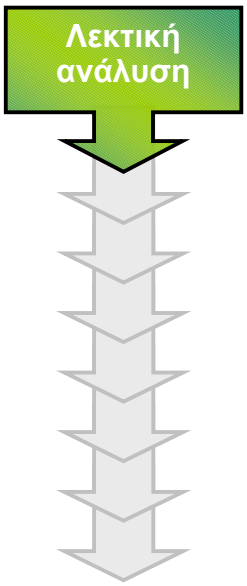
γλώσσας μηχανής. Το τμήμα που κάνει την αναγνώριση της γλώσσας προγραμματισμού δεν εξαρτάται από τον υπολογιστή στον οποίο θα εκτελεστεί αλλά μόνο από τη γλώσσα προγραμματισμού. Αντίστοιχα, το τμήμα που κάνει την παραγωγή της γλώσσας μηχανής δεν εξαρτάται από τη γλώσσα προγραμματισμού αλλά μόνο από το ρεπερτόριο εντολών μηχανής που παράγει. Αυτό δίνει τη δυνατότητα στους κατασκευαστές των μεταγλωττιστών να κατασκευάζουν ανεξάρτητα τα δύο τμήματα, και να τα συνδυάζουν όπως χρειάζεται.



Μία εταιρία λογισμικού που κατασκευάζει μεταγλωττιστές μπορεί να σχεδιάζει και να υλοποιεί τα τμήματα των μεταγλωττιστών ανεξάρτητα το ένα από το άλλο, και μετά να τα συνδυάζει. Αν οι γλώσσες με τις οποίες ασχολείται είναι η Pascal, η C++ και η Ada, ενώ οι αρχιτεκτονικές επεξεργαστών για τους οποίους παράγονται τα προγράμματα είναι ο Sparc της Sun, ο Pentium της Intel και ο Alpha της DEC, τότε μπορεί να κατασκευάσει τρία τμήματα για τις γλώσσες προγραμματισμού και άλλα τρία για τις αρχιτεκτονικές. Συνδυάζοντας τα έξι τμήματα αυτά μεταξύ τους έχει έτοιμους εννέα μεταγλωττιστές.

Το πρώτο τμήμα του μεταγλωττιστή εκτελεί τρεις λειτουργίες: τη λεκτική, τη συντακτική και τη σημασιολογική ανάλυση του προγράμματος. Από τις λειτουργίες αυτές προκύπτει μία αναπαράσταση του προγράμματος η οποία δεν εξαρτάται από τη γλώσσα προγραμματισμού. Την αναπαράσταση αυτή χρησιμοποιεί το δεύτερο μέρος του μεταγλωττιστή, που επιτελεί την παραγωγή κώδικα μηχανής (ή συμβολικής γλώσσας) και τη βελτιστοποίησή του.

Στη *λεκτική ανάλυση* (lexical analysis) του προγράμματος, ο μεταγλωττιστής το χωρίζει στις βασικές του μονάδες: τις λέξεις, τα σύμβολα, τους αριθμούς κλπ. Τα διαστήματα που διαχωρίζουν τις μονάδες του προγράμματος αφαιρούνται, και το αποτέλεσμα είναι μία σειρά από *μονάδες του προγράμματος* (tokens).

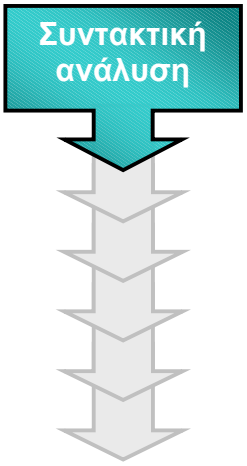


Σε ένα πρόγραμμα Pascal εμφανίζονται οι εντολές

```
y := 5;
x := y+15;
```

Ο λεκτικός αναλυτής θα χωρίσει αυτό το τμήμα του προγράμματος στην ακολουθία:

```
«αναγνωριστικό όνομα y», «τελεστής ανάθεσης»,
«αριθμητική σταθερά 5», «ερωτηματικό»,
«αναγνωριστικό όνομα x», «τελεστής ανάθεσης»,
«αναγνωριστικό όνομα y», «τελεστής συν»,
«αριθμητική σταθερά 15», «ερωτηματικό».
```

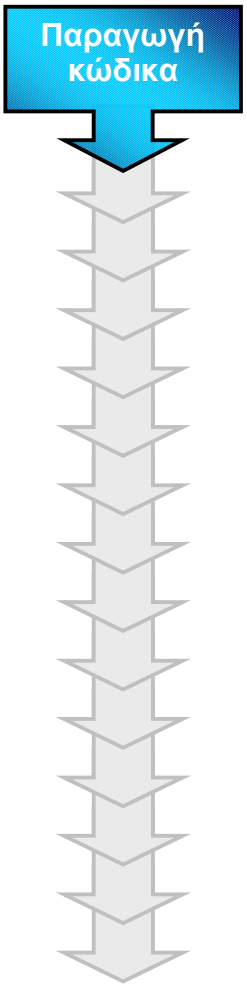


Στη *συντακτική ανάλυση* (syntax analysis) του προγράμματος, ελέγχεται κατά πόσο αυτό ακολουθεί τους συντακτικούς κανόνες της γλώσσας προγραμματισμού, και μετασχηματίζεται σε κάποια ενδιάμεση αναπαράσταση, την *ενδιάμεση γλώσσα* (intermediate language), η οποία δεν εξαρτάται από τη γλώσσα προγραμματισμού.

> Στη γλώσσα Pascal οι εντολές χωρίζονται μεταξύ τους με ερωτηματικό («;»). Ο συντακτικός αναλυτής θα διαπιστώσει το λάθος και θα ενημερώσει τον προγραμματιστή αν έχει κατά λάθος γράψει τις παραπάνω εντολές σαν:
>
> ```
> y := 5
> x := y+15;
> ```
>
> όπου λείπει ένα ερωτηματικό μεταξύ των εντολών.

Παράλληλα με τη συντακτική, γίνεται και η *σημασιολογική ανάλυση* (semantic analysis).

> Αν ο προγραμματιστής γράψει σε ένα πρόγραμμα Pascal:
>
> ```
> y := 5 + "abcde";
> ```
>
> ο σημασιολογικός αναλυτής θα διαπιστώσει την εσφαλμένη πρόσθεση μεταξύ ενός αριθμού και ενός κειμένου, και θα ενημερώσει τον προγραμματιστή για αυτό.



Στη συνέχεια ο μεταγλωττιστής παράγει τις εντολές σε γλώσσα μηχανής ή σε συμβολική γλώσσα οι οποίες αντιστοιχούν στο πρόγραμμα το οποίο μεταφράζει. Αυτό το στάδιο της μεταγλώττισης ονομάζεται *παραγωγή κώδικα* (code generation). Με βάση την ενδιάμεση μορφή του προγράμματος που έχει κατασκευάσει κατά τη συντακτική ανάλυση, ο μεταγλωττιστής αντικαθιστά κάθε εντολή του προγράμματος με τις κατάλληλες εντολές μηχανής. Στο στάδιο αυτό αποφασίζει για τις θέσεις της μνήμης όπου θα αποθηκευθούν τα δεδομένα του προγράμματος.

> Για τις δύο εντολές Pascal που είδαμε, ένας μεταγλωττιστής για τον Άβακα πρέπει να αποφασίσει πού στη μνήμη θα κρατούνται οι δύο σταθερές 5 και 15 και οι δύο μεταβλητές x και y. Αν οι σταθερές τοποθετούνται σε θέσεις μνήμης με συμβολικά ονόματα C5 και C15 αντίστοιχα, και οι δύο μεταβλητές τοποθετούνται σε θέσεις μνήμης με τα συμβολικά ονόματα X και Y, τότε το πρόγραμμα σε συμβολική γλώσσα που θα παραγάγει ο μεταγλωττιστής είναι:
>
> ```
> C5      CON     5
> C15     CON     15
> ...
> X       RES     1
> Y       RES     1
> ...
>         LDA     C5
>         STA     Y
>
>         LDA     Y
>         ADA     C15
>         STA     X
> ```
>
> σταθερές
>
> μεταβλητές
>
> y := 5
>
> x :=y+1 5

Οι πληροφορίες για τη θέση των μεταβλητών και των σταθερών κρατούνται από το μεταγλωττιστή στον *πίνακα συμβόλων* (symbol table).

Ο πίνακας συμβόλων για το παράδειγμά μας είναι:

Συμβολική διεύθυνση	Σχετική διεύθυνση	Αρχικό περιεχόμενο
C5	0	0000000000000101
C15	1	0000000000001111
Y	2	0000000000000000
X	3	0000000000000000

Βελτιστοποίηση κώδικα

Το τελευταίο στάδιο στη μεταγλώττιση ενός προγράμματος είναι η *βελτιστοποίηση του κώδικα* (code optimization). Στη φάση αυτή, γίνονται μικρές μετατροπές στον κώδικα μηχανής, με τέτοιο τρόπο ώστε να επιτελεί τις ίδιες λειτουργίες αλλά να είναι πιο γρήγορος ή πιο μικρός σε μέγεθος.

Το πρόγραμμα του παραδείγματός μας, με τη δεύτερη εντολή του αποθηκεύει το περιεχόμενο του συσσωρευτή στη θέση μνήμης με ετικέτα Y και αμέσως μετά αντιγράφει πάλι το περιεχόμενο της θέσης μνήμης Y στο συσσωρευτή. Η δεύτερη αντιγραφή είναι περιττή, αφού η θέση μνήμης Y και ο συσσωρευτής έχουν το ίδιο περιεχόμενο. Κατά τη βελτιστοποίηση του κώδικα, η δεύτερη εντολή θα παραλειφθεί, και το πρόγραμμα θα γίνει τελικά:

```
C5      CON     5
C15     CON     15
X       RES     1
Y       RES     1
START   LDA     C5
        STA     Y
        ADA     C15
        STA     X
        END     START
```

Κώδικας μηχανής

Αν ο μεταγλωττιστής παράγει συμβολική γλώσσα, τότε ο συμβολομεταφραστής θα επεξεργαστεί το πρόγραμμα και θα παραγάγει τις εντολές μηχανής που είναι έτοιμες για εκτέλεση από τον υπολογιστή.

Τελικά, το πρόγραμμα σε γλώσσα μηχανής είναι:

Διεύθυνση	Περιεχόμενο		
000000000000	0001000000000101	LDA	C5
000000000001	0010000000001000	STA	Y
000000000010	0011000000000110	ADA	C15
000000000011	0010000000000111	STA	X
000000000100	0000000000000000	HLT	
000000000101	0000000000000101	C5	
000000000110	0000000000001111	C15	
000000000111	0000000000000000	X	
000000001000	0000000000000000	Y	

Οι γλώσσες προγραμματισμού υψηλού επιπέδου έχουν δημιουργηθεί για να κάνουν το έργο των προγραμματιστών πιο απλό και γρήγορο. Τα προγράμματα μετατρέπονται από τη γλώσσα προγραμματισμού σε γλώσσα μηχανής από ειδικά προγράμματα: τους μεταφραστές και τους διερμηνείς. Ένας μεταφραστής ακολουθεί συνήθως πέντε στάδια για να μεταφράσει ένα πρόγραμμα: τη λεκτική ανάλυση, τη συντακτική ανάλυση, τη σημασιολογική ανάλυση, την παραγωγή κώδικα και τη βελτιστοποίηση κώδικα.

Βελτιστοποίηση Κώδικα	*Code Optimization*
Γλώσσα Προγραμματισμού Υψηλού Επιπέδου	*High Level Programming Language*
Διερμηνέας	*Interpreter*
Ενδιάμεση Γλώσσα	*Intermediate Language*
Λεκτική Ανάλυση	*Lexical Analysis*
Μεταγλωττιστής	*Compiler*
Μεταφραστής	*Translator*
Μονάδες Προγράμματος	*Tokens*
Παραγωγή Κώδικα	*Code Generation*
Πίνακας Συμβόλων	*Symbol Table*
Σημασιολογική Ανάλυση	*Semantic Analysis*
Συμβολομεταφραστής	*Assembler*
Συντακτική Ανάλυση	*Syntax Analysis*

Ερωτήσεις

? Ποιες είναι οι δυσκολίες του προγραμματισμού σε γλώσσα μηχανής;

? Τι είναι οι συμβολομεταφραστές και οι μεταφραστές;

? Ποια είναι η διαφορά του μεταγλωττιστή από το διερμηνέα;

? Ποια είναι τα στάδια της μεταγλώττισης;

? Σε τι τμήματα χωρίζει ένα πρόγραμμα η λεκτική ανάλυση;

? Ποιο είναι το έργο της συντακτικής ανάλυσης;

? Ποιο είναι το έργο της σημασιολογικής ανάλυσης;

? Τι γίνεται κατά την παραγωγή κώδικα;

? Ποιος είναι ο σκοπός της βελτιστοποίησης κώδικα;

? Τι περιέχει ο πίνακας συμβόλων;

Μάθημα 3.5 Σύγχρονοι Επεξεργαστές

Σκοπός του μαθήματος

Σκοπός του μαθήματος αυτού είναι να παρουσιάσει τα ιδιαίτερα χαρακτηριστικά των αρχιτεκτονικών που χρησιμοποιούνται στους σύγχρονους επεξεργαστές.

Τι θα μάθεις;

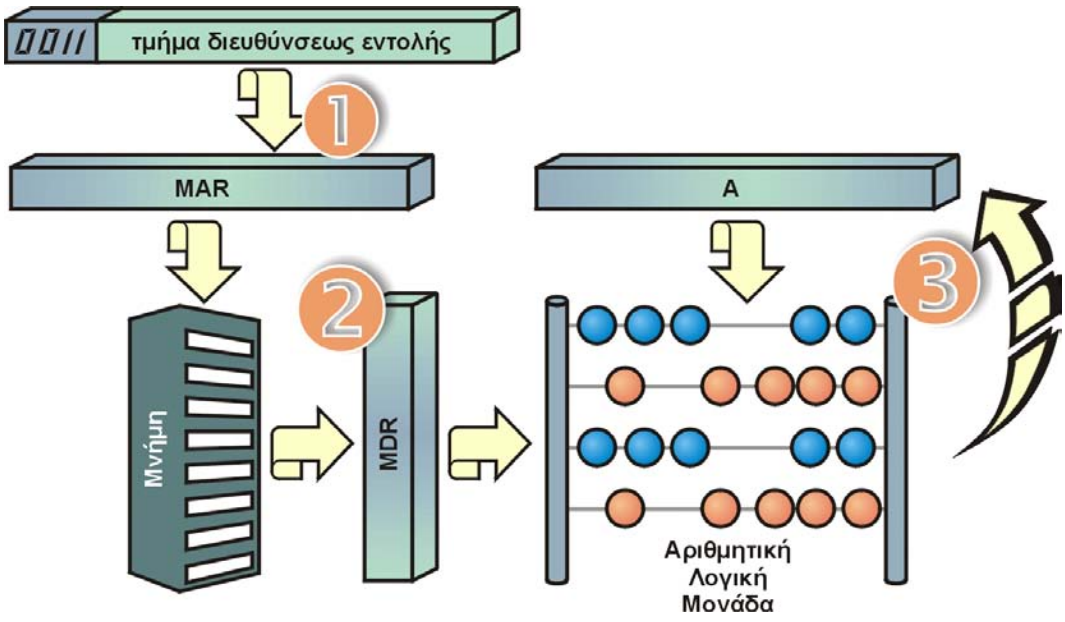
Όταν ολοκληρώσεις το μάθημα αυτό, θα μπορείς:

- *Να εξηγείς τη λειτουργικότητα του μικροπρογραμματισμού*
- *Να περιγράφεις την αρχιτεκτονική RISC και την αρχιτεκτονική αγωγού*
- *Να απαριθμείς τις διάφορες εναλλακτικές αρχιτεκτονικές επεξεργαστών που χρησιμοποιούνται σήμερα*

Μικρολειτουργίες

Γνωρίζουμε ότι κάθε επεξεργαστής μπορεί να εκτελέσει ένα σύνολο από εντολές, το ρεπερτόριο εντολών του. Κάθε εντολή εκτελείται στο εσωτερικό του επεξεργαστή σαν μία σειρά από πολύ απλά και στοιχειώδη βήματα.

Η εντολή του Άβακα με κωδικό 0011 και συμβολικό όνομα ADA, προσθέτει το περιεχόμενο του συσσωρευτή A και το περιεχόμενο της θέσης μνήμης που προσδιορίζεται από το τμήμα διευθύνσεως της εντολής.



Στο σχήμα βλέπουμε τα τρία στάδια εκτέλεσης της εντολής αυτής. Κάθε στάδιο είναι μία στοιχειώδης λειτουργία ή *μικρολειτουργία* (microoperation) του επεξεργαστή.

Το τμήμα διευθύνσεως της εντολής περιέχει τη διεύθυνση μίας θέσης στη μνήμη. Για να διαβαστεί το περιεχόμενο της θέσης αυτής, πρέπει το τμήμα διευθύνσεως του IR (δηλαδή τα 12 λιγότερο σημαντικά bits, από το bit 0 έως το bit 11) να αντιγραφεί στον καταχωρητή MAR της μονάδας μνήμης. Αυτό είναι η πρώτη μικρολειτουργία στην εκτέλεση της εντολής, που θα συμβολίζεται με:

MAR ← IR(0,11)

Στη δεύτερη μικρολειτουργία που αποτελεί την εντολή, γίνεται μία λειτουργία ανάγνωσης στη μνήμη. Με τη λειτουργία αυτή, το περιεχόμενο της διεύθυνσης μνήμης που υποδεικνύει ο MAR μεταφέρεται στον MDR, τον καταχωρητή δεδομένων της μονάδας μνήμης. Η μικρολειτουργία αυτή θα συμβολίζεται σαν:

MDR ← M[MAR] ή απλά **READ**

Στην τρίτη μικρολειτουργία γίνεται τελικά η πρόσθεση μεταξύ των δύο αριθμών. Ο ένας αριθμός λαμβάνεται από τον καταχωρητή δεδομένων MDR και ο άλλος από το συσσωρευτή A. Τα δεδομένα περνούν από τα κυκλώματα της αριθμητικής-λογικής μονάδας που εκτελούν την πρόσθεση και το αποτέλεσμα διοχετεύεται πάλι στο συσσωρευτή A. Η αποθήκευση γίνεται κατευθείαν στον A, μια που αυτός βρίσκεται στο εσωτερικό της αριθμητικής-λογικής μονάδας. Η μικρολειτουργία αυτή θα συμβολίζεται ως:

A ← A + MDR

Μικροπρογραμματισμός

Όπως η εντολή ADA, έτσι και όλες οι υπόλοιπες εντολές του Άβακα και κάθε υπολογιστή μπορούν να αναλυθούν σε μία ακολουθία από μικρολειτουργίες. Οι μικρολειτουργίες που χρειάζονται για να περιγραφούν όλες οι εντολές ενός υπολογιστή δεν είναι πολλές· το πλήθος τους είναι παρόμοιο ή μικρότερο από το πλήθος των εντολών μηχανής του υπολογιστή.

Ένας πιο συστηματικός τρόπος οργάνωσης ενός επεξεργαστή είναι η *μικροπρογραμματιζόμενη αρχιτεκτονική* (microprogrammed architecture). Η βασική της ιδέα είναι να αναλύονται οι εντολές μηχανής σε μικρολειτουργίες, και οι ακολουθίες μικρολειτουργιών για κάθε εντολή να είναι καταχωρημένες σαν προγράμματα σε μία ειδική μνήμη, τη *μνήμη ελέγχου* (control memory). Η μνήμη αυτή βρίσκεται στο εσωτερικό του επεξεργαστή και όχι στη μονάδα μνήμης του υπολογιστή.

Σε κάθε μικρολειτουργία αντιστοιχίζεται μία *μικροεντολή* (microinstruction), και η μνήμη ελέγχου περιέχει μία σειρά μικροεντολών, δηλαδή ένα *μικροπρόγραμμα* (microprogram). Για να εκτελεστεί το μικροπρόγραμμα μίας εντολής από την ΚΜΕ, και να εκτελεστεί η εντολή, είναι απαραίτητοι οι αντίστοχοι καταχωρητές ελέγχου μέσα στην ΚΜΕ, ο *καταχωρητής μικροεντολών* (MIR) και ο *μετρητής μικροπρογράμματος*

(MPC). Οι καταχωρητές αυτοί παίζουν για την ΚΜΕ τον ίδιο ρόλο που παίζουν ο καταχωρητής εντολών IR και ο μετρητής προγράμματος PC για τον υπολογιστή.

Μπορούμε να φανταστούμε έτσι ότι ο επεξεργαστής είναι στην πραγματικότητα ένας «μικρός υπολογιστής» μέσα στον υπολογιστή, που εκτελεί προγράμματα στη δική του γλώσσα μηχανής, τη γλώσσα των μικρολειτουργιών. Κάθε πρόγραμμα στη γλώσσα αυτή είναι μία εντολή μηχανής για τον υπολογιστή.

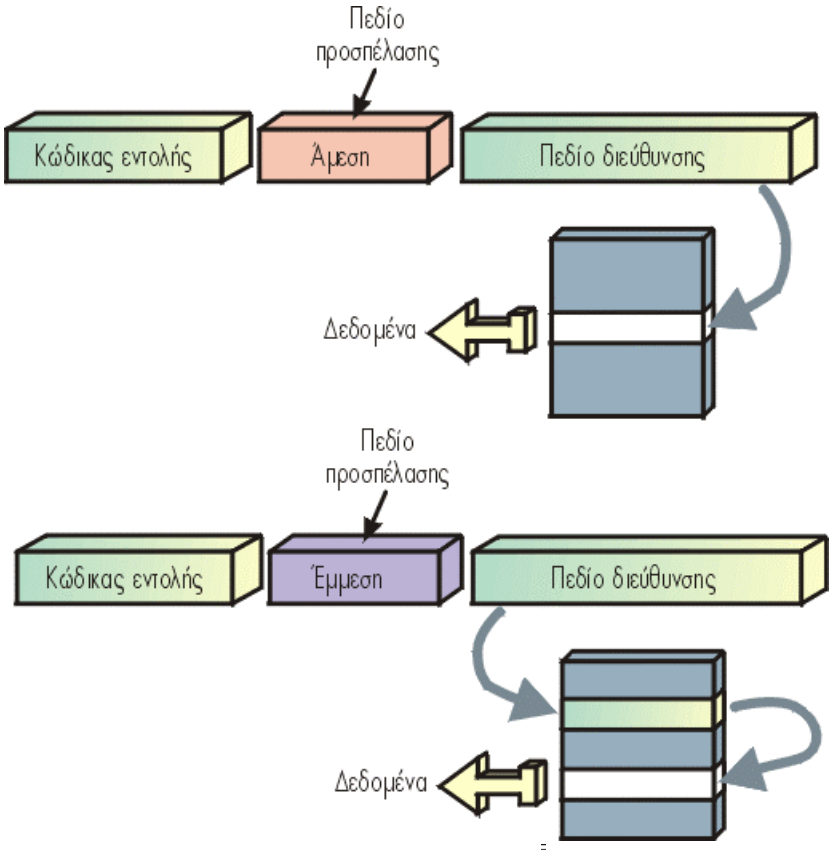
Σε πολλούς υπολογιστές ο κατασκευαστής δίνει τη δυνατότητα στους χρήστες να επέμβουν στο επίπεδο των μικρολειτουργιών και να κατασκευάσουν το δικό τους ρεπερτόριο εντολών μηχανής. Με έμμεσο τρόπο έτσι οι χρήστες μπορούν να επέμβουν «προγραμματιστικά» στην οργάνωση του υλικού. Αυτό το συνδυασμό υλικού και λογισμικού τον ονομάζουμε *υλικολογισμικό* (firmware).

Άλλες αρχιτεκτονικές επεξεργαστών

Ο Άβακας, ο υπολογιστής που είδαμε μέχρι τώρα, είναι πολύ απλός γιατί σκοπός του είναι να καταδείξει τις βασικές αρχές λειτουργίας των επεξεργαστών. Στην πράξη τα πράγματα είναι αρκετά πιο πολύπλοκα.

Σε όλους τους σύγχρονους επεξεργαστές κάθε εντολή αποτελείται από πολλά πεδία, και όχι μόνο από δύο. Εκτός από το πεδίο εντολής και το πεδίο διεύθυνσης, μπορούν να υπάρχουν:

- *Πεδία που προσδιορίζουν τον τρόπο προσπέλασης στη μνήμη*.



Συνήθως το πεδίο διεύθυνσης της εντολής προσδιορίζει τη θέση της μνήμης όπου βρίσκονται κάποια δεδομένα. Ο τρόπος αυτός αναφοράς στη μνήμη ονομάζεται *άμεση αναφορά* (direct access). Το πεδίο διεύθυνσης μπορεί επίσης να προσδιορίζει τη θέση στη μνήμη όπου βρίσκεται η διεύθυνση η οποία τελικά περιέχει τα δεδομένα και πρέπει να χρησιμοποιήσει η εντολή. Αυτή είναι η *έμμεση αναφορά* (indirect access) στη μνήμη και υπάρχει ειδικό πεδίο μέσα στην εντολή που προσδιορίζει αν θα εφαρμοστεί ή όχι. Το πεδίο αυτό μπορεί επίσης να προσδιορίζει ότι τα δεδομένα περιέχονται απευθείας στο τμήμα διεύθυνσης της εντολής, δηλαδή ότι γίνεται *απευθείας αναφορά* (immediate access).

- *Πεδία που προσδιορίζουν τη συνθήκη για τις εντολές άλματος*. Στον Άβακα, η εντολή άλματος υπό συνθήκη ελέγχει αν ο συσσωρευτής Α έχει αρνητική τιμή. Ένα άλμα υπό συνθήκη όμως μπορεί να γίνει ανάλογα με τις τιμές διαφόρων καταχωρητών, με το αποτέλεσμα μίας αριθμητικής ή λογικής πράξης που έγινε προηγουμένως, κλπ. Οι συνθήκες αυτές κωδικοποιούνται μέσα σε ένα πεδίο της εντολής.

- *Πεδία που αναφέρονται σε καταχωρητές της ΚΜΕ*. Όταν η ΚΜΕ περιέχει πολλούς καταχωρητές, οι εντολές μπορούν να αναφέρονται σε οποιονδήποτε από αυτούς.

Μία εντολή της μορφής LDA, μπορεί να μεταφέρει δεδομένα από τη μνήμη σε οποιοδήποτε καταχωρητή και όχι μόνο τον Α. Έτσι υπάρχει ένας κώδικας εντολής, που σημαίνει «μεταφορά από τη μνήμη σε καταχωρητή» και ένα πεδίο εντολής που περιέχει τον «κωδικό» του καταχωρητή στον οποίο θα γραφούν τα δεδομένα.

Υπολογιστές σύνθετου ρεπερτορίου εντολών

Οι *υπολογιστές σύνθετου ρεπερτορίου εντολών* (Complex Instruction Set Computers - CISC) διαθέτουν ένα μεγάλο ρεπερτόριο εντολών (120 έως 350 περίπου), χρησιμοποιούν μικρό αριθμό καταχωρητών (8 έως 24) και επίσης χρησιμοποιούν πολλούς διαφορετικούς τρόπους προσπέλασης στη μνήμη (περισσότερους από 12).

Οι υπολογιστές CISC χρησιμοποιούν συνήθως ένα συνδυασμό μικροπρογραμματιζόμενης και καλωδιωμένης αρχιτεκτονικής. Διαθέτουν λανθάνουσα μνήμη και διαδρόμους μεγάλου πλάτους, κοινούς για δεδομένα και διευθύνσεις.

Το βασικό τους πλεονέκτημα είναι ότι διευκολύνουν ιδιαίτερα τον προγραμματισμό σε γλώσσα μηχανής και οι μεταφραστές για τις γλώσσες προγραμματισμού είναι σχετικά απλοί. Μειονέκτημά τους είναι ότι κάθε εντολή τους απαιτεί αρκετό χρόνο να εκτελεστεί και βέβαια η σχεδίασή τους είναι αρκετά πολύπλοκη.

Οι περισσότεροι προσωπικοί υπολογιστές χρησιμοποιούν επεξεργαστές CISC.

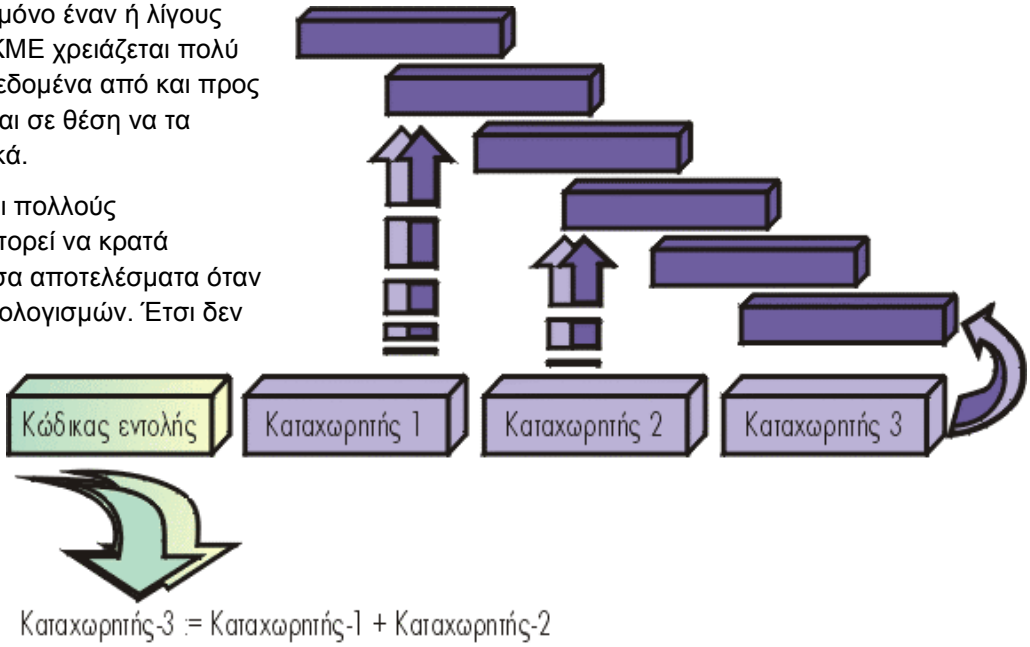
Υπολογιστές μειωμένου ρεπερτορίου εντολών

Με την εξέλιξη των επεξεργαστών CISC, το ρεπερτόριο εντολών τους γινόταν όλο και μεγαλύτερο. Οι σχεδιαστές υπολογιστών παρατήρησαν όμως στατιστικά ότι μόνο το 25% των εντολών ενός υπολογιστή χρησιμοποιούνται πολύ (περίπου το 95% του χρόνου). Το ρεπερτόριο εντολών, απ' ό,τι φαινόταν, θα μπορούσε να μειωθεί δραστικά, απλουστεύοντας έτσι και το υλικό. Οι επιπλέον εντολές θα μπορούσαν να υλοποιηθούν με λογισμικό.

Αυτή είναι η κεντρική ιδέα των *υπολογιστών μειωμένου ρεπερτορίου εντολών* (Reduced Instruction Set Computers, RISC). Το ρεπερτόριο εντολών των υπολογιστών αυτών είναι μικρό, με λιγότερες από 100 εντολές, οπότε το πεδίο κώδικα εντολής δεν καταλαμβάνει μεγάλο χώρο στην εντολή. Έτσι οι εντολές εκτελούνται γρήγορα.

Όταν η ΚΜΕ περιέχει μόνο έναν ή λίγους καταχωρητές, τότε η ΚΜΕ χρειάζεται πολύ συχνά να μεταφέρει δεδομένα από και προς τη μνήμη, γιατί δεν είναι σε θέση να τα αποθηκεύσει εσωτερικά.

Μία ΚΜΕ που περιέχει πολλούς καταχωρητές όμως μπορεί να κρατά εσωτερικά τα ενδιάμεσα αποτελέσματα όταν εκτελεί ακολουθίες υπολογισμών. Έτσι δεν έχει τόσο μεγάλη ανάγκη για ανταλλαγή δεδομένων και προσπέλαση της μνήμης. Αυτό έχει σαν άμεσο αποτέλεσμα την αύξηση της ταχύτητας του υπολογιστή σε σχέση με κάποιον με λίγους καταχωρητές.



> Μία εντολή ενός υπολογιστή RISC είναι αυτή που προσθέτει δύο αριθμούς που είναι αποθηκευμένοι σε δύο καταχωρητές και αποθηκεύει το αποτέλεσμα σε έναν τρίτο καταχωρητή. Για την εντολή αυτή υπάρχουν πολλοί συνδυασμοί εκτέλεσης, αλλά ο κώδικας της εντολής είναι ένας. Σε συμβολική γλώσσα, η μορφή της εντολής θα ήταν:
>
> $$\text{ADD } R_1, R_2, R_3$$
>
> και θα είχε την έννοια $R_3 := R_1 + R_2$.
>
> Αν η ΚΜΕ περιέχει 10 καταχωρητές, τότε για κάθε έναν από τους R_1, R_2, R_3 θα υπάρχουν 10 διαφορετικές επιλογές και οι τελικοί συνδυασμοί θα είναι $10 \times 10 \times 10 = 1000$. Με έναν κώδικα εντολής λοιπόν μπορούν να κωδικοποιηθούν 1000 διαφορετικές εντολές.

Πρακτικά, οι πολλοί καταχωρητές μέσα στην ΚΜΕ συνθέτουν μία πολύ μικρή και γρήγορη μνήμη. Σήμερα οι περισσότεροι επεξεργαστές, κυρίως σε σταθμούς εργασίας, είναι τύπου RISC.

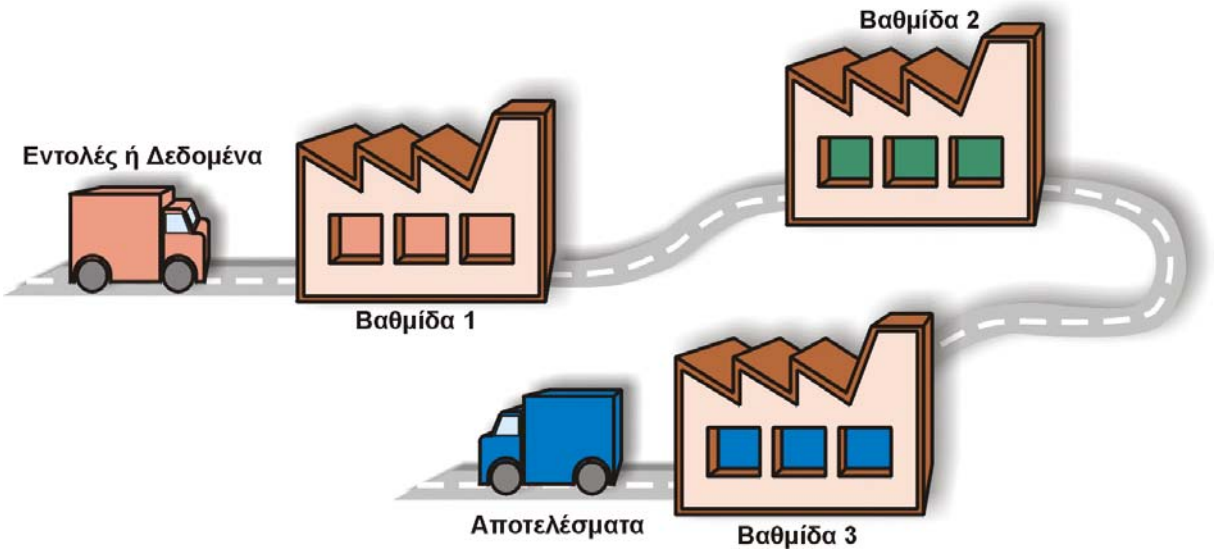
Οι υπολογιστές RISC δε χρησιμοποιούν μικροπρογραμματιζόμενη αρχιτεκτονική εξαιτίας της απλότητας των εντολών τους. Επίσης διαθέτουν ξεχωριστούς διαδρόμους για τα δεδομένα και τις διευθύνσεις τους.

Το μειονέκτημα των υπολογιστών RISC είναι ότι ο προγραμματισμός σε γλώσσα μηχανής γίνεται πιο δύσκολος και απαιτούνται πιο «έξυπνοι» μεταφραστές για τις γλώσσες προγραμματισμού.

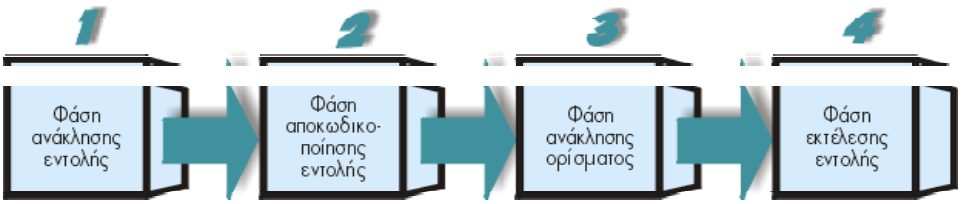
Αρχιτεκτονική αγωγού

Μία αρχιτεκτονική που χρησιμοποιείται σε πολλούς σύγχρονους επεξεργαστές είναι η *αρχιτεκτονική αγωγού* (pipeline architecture).

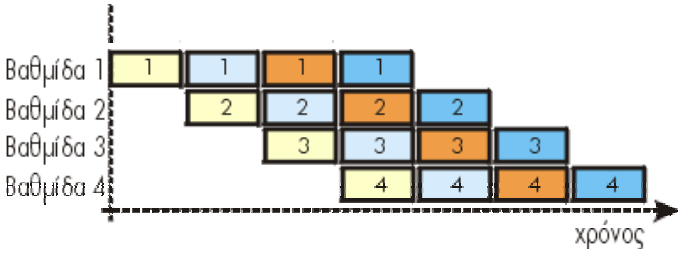
Στους επεξεργαστές αυτούς, ο κύκλος εκτέλεσης μίας εντολής διαιρείται σε πολλές διαδοχικές φάσεις, που ονομάζονται *κύκλοι αγωγού* (pipeline cycles). Μέσα στην ΚΜΕ υπάρχει ένα διακεκριμένο τμήμα που αντιστοιχεί σε κάθε ένα κύκλο και ονομάζεται *βαθμίδα* (stage). Κάθε εντολή ξεκινά από την πρώτη βαθμίδα του αγωγού και περνά από όλες με τη σειρά μέχρι να φθάσει και στην τελευταία. Μόλις μία βαθμίδα τελειώσει την επεξεργασία μίας εντολής, μπορεί αμέσως να προχωρήσει στην επόμενη χωρίς καθυστέρηση.



Ένας υπολογιστής είναι οργανωμένος σε αγωγό τεσσάρων βαθμίδων. Κάθε βαθμίδα εκτελεί μία φάση στην εκτέλεση μίας εντολής, όπως φαίνεται στο σχήμα.



Αν κάθε βαθμίδα της εντολής ολοκληρώνει τη λειτουργία της σε 1 χρονική μονάδα, τότε η εκτέλεση των εντολών γίνεται όπως φαίνεται στο πρώτο χρονικό διάγραμμα εκτέλεσης. Παρατηρούμε ότι η πρώτη εντολή απαιτεί τέσσερις χρονικές μονάδες να ολοκληρωθεί, αλλά από εκεί και στο εξής σε κάθε χρονική μονάδα ολοκληρώνεται μία εντολή. Έτσι μακροπρόθεσμα μπορούμε να πούμε ότι ο υπολογιστής αυτός εκτελεί μία εντολή ανά χρονική μονάδα.



Εκτέλεση εντολών στον υπολογιστή με αγωγό: κατά μέσο όρο ολοκληρώνεται μία εντολή σε κάθε χρονική μονάδα

Αν αντιπαραβάλουμε τον υπολογιστή αυτόν με κάποιον που δεν έχει αγωγό αλλά εκτελεί τις εντολές μία μία βλέπουμε ότι είναι τέσσερις φορές πιο γρήγορος, αφού ο υπολογιστής χωρίς αγωγό χρειάζεται τέσσερις χρονικές μονάδες ανά εντολή, μία ανά φάση εκτέλεσης.

Εκτέλεση εντολών στον υπολογιστή χωρίς αγωγό: κατά μέσο όρο ολοκληρώνεται μία εντολή κάθε τέσσερις χρονικές μονάδες

Μία ιδιαιτερότητα της αρχιτεκτονικής αγωγού είναι ο τρόπος που εκτελούνται οι εντολές άλματος υπό συνθήκη. Το αν θα εκτελεστεί ένα τέτοιο άλμα ή όχι, συνήθως εξαρτάται από το αποτέλεσμα μίας αριθμητικής ή λογικής πράξης, η οποία μπορεί να μην έχει ακόμα ολοκληρωθεί όταν η εντολή άλματος «εισαχθεί» στον αγωγό. Έτσι λοιπόν ο επεξεργαστής πρέπει να λάβει υπόψη του συγχρόνως και τα δύο «σενάρια» εκτέλεσης του προγράμματος. Όταν ολοκληρωθεί η λειτουργία που επηρεάζει την εκτέλεση του άλματος, ο επεξεργαστής θα επιλέξει το κατάλληλο «σενάριο» και θα το συνεχίσει.

Εκτός από τους αγωγούς που αναλαμβάνουν ολόκληρη την εκτέλεση μίας εντολής, υπάρχουν και οι *αριθμητικοί αγωγοί*, οι οποίοι χρησιμοποιούνται για το γρηγορότερο υπολογισμό αριθμητικών εκφράσεων.

Μία αρχιτεκτονική παρόμοια με την αρχιτεκτονική αγωγού είναι η *υπερβαθμωτή* (superscalar) αρχιτεκτονική. Σε αυτήν, ο επεξεργαστής μπορεί να εκτελέσει παράλληλα πολλές, ανεξάρτητες μεταξύ τους εντολές μηχανής, ενώ εντολές που επηρεάζουν η μία την άλλη εκτελούνται ακολουθιακά.

Υπάρχουν ακόμα διάφορες άλλες αρχιτεκτονικές υπολογιστών, όπως:

- **Οι παράλληλοι υπολογιστές**. Στο μοντέλο αυτό, πολλοί επεξεργαστές συνεργάζονται και συγχρονίζονται για να εκτελέσουν ένα πρόγραμμα γραμμένο έτσι ώστε να μοιράζει το υπολογιστικό φορτίο ανάμεσα στις διάφορες ΚΜΕ. Οι επεξεργαστές αυτοί μπορούν να έχουν κοινή μνήμη, οπότε ονομάζονται *πολυεπεξεργαστές* (multiprocessors) ή να έχει καθένας τη δική του μνήμη, οπότε μιλάμε για *πολυυπολογιστές* (multicomputers).
- **Οι μη συμβατικοί υπολογιστές**. Αυτοί δεν ακολουθούν το κλασικό μοντέλο του Von Neumann όπως ο Άβακας, όπου το πρόγραμμα είναι αποθηκευμένο στη μνήμη και υπάρχει ο μετρητής προγράμματος. Τέτοιοι υπολογιστές είναι οι *μηχανές ροής δεδομένων* (data flow machines), και οι *μηχανές αναγωγής* (reduction machines).
- **Οι υπολογιστές ειδικού σκοπού**, όπως οι *συστολικές διατάξεις* (systolic arrays), οι *κυτταρικές διατάξεις* (cellular arrays), οι ειδικοί επεξεργαστές για επεξεργασία σήματος (DSP) κλπ.
- **Οι *υπολογιστές μεγάλου μήκους λέξης*** ή ***πολύ μεγάλου μήκους λέξης*** (long instruction word computers, LIW ή very long instruction word computers, VLIW). Σε αυτούς το μήκος λέξης είναι πολύ μεγαλύτερο από ό,τι στους περισσότερους υπολογιστές, της τάξης των εκατοντάδων bits. Κάθε εντολή περιλαμβάνει την ταυτόχρονη εκτέλεση πολλών λειτουργιών.

Ανακεφαλαίωση

Σε πολλούς σύγχρονους επεξεργαστές οι εντολές σε γλώσσα μηχανής αποτελούν με τη σειρά τους προγράμματα που αποτελούνται από στοιχειώδεις λειτουργίες του επεξεργαστή, τις μικροεντολές.
Στους σύγχρονους επεξεργαστές συναντάμε μία ποικιλία αρχιτεκτονικών, όπως είναι η αρχιτεκτονική μειωμένου ρεπερτορίου εντολών, η αρχιτεκτονική αγωγού, η υπερβαθμωτή αρχιτεκτονική κλπ.

Γλωσσάριο όρων

Άμεση Αναφορά	*Direct Access*
Απευθείας Αναφορά	*Immediate Access*
Αρχιτεκτονική Αγωγού	*Pipeline Architecture*
Βαθμίδα	*Stage*
Έμμεση Αναφορά	*Indirect Access*
Καταχωρητής Μικροεντολών	*MIR*
Κύκλοι Αγωγού	*Pipeline Cycles*
Κυτταρικές Διατάξεις	*Cellular Arrays*
Μετρητής Μικροπρογράμματος	*MPC*
Μηχανές Αναγωγής	*Reduction Machines*
Μηχανές Ροής Δεδομένων	*Data Flow Machines*
Μικροεντολή	*Microinstruction*
Μικρολειτουργία	*Microoperation*
Μικροπρόγραμμα	*Microprogram*
Μικροπρογραμματιζόμενη Αρχιτεκτονική	*Microprogrammed Architecture*
Μνήμη Ελέγχου	*Control Memory*
Πολυεπεξεργαστές	*Multiprocessors*
Πολυυπολογιστές	*Multicomputers*
Συστολικές Διατάξεις	*Systolic Arrays*
Υλικολογισμικό	*Firmware*
Υπερβαθμωτή Αρχιτεκτονική	*Superscalar Architecture*
Υπολογιστής Μεγάλου Μήκους Λέξης	*Long Instruction Word - LIW*
Υπολογιστής Μειωμένου Ρεπερτορίου Εντολών	*Reduced Instruction Set Computer - RISC*
Υπολογιστής Σύνθετου Ρεπερτορίου Εντολών	*Complex Instruction Set Computer - CISC*

Ερωτήσεις

? Τι είναι η μικρολειτουργία και το μικροπρόγραμμα;

? Ποιο είναι το χαρακτηριστικό των υπολογιστών μειωμένου ρεπερτορίου εντολών;

? Πώς λειτουργούν οι υπολογιστές με αρχιτεκτονική αγωγού;