ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
www.cslab.ece.ntua.gr

31/5/2005

Εισαγωγή στην Επιστήμη των Υπολογιστών

Εξάμηνο 4ο-ΣΗΜΜΥ

Σημειώσεις στη Συμβολική Γλώσσα του Υπολογιστή ΕΚΥ (ASSEMBLY)

(διδάσκων καθηγητής: Νεκτάριος Κοζύρης)
(επιμέλεια σημειώσεων: Μαρία Αθανασάκη)

I. ΔΟΜΗ ΚΑΙ ΛΕΙΤΟΥΡΓΙΑ ΤΟΥ ΥΠΟΛΟΓΙΣΤΗ

1. Εισαγωγή

Κάθε Ηλεκτρονικός Υπολογιστής αποτελείται από τέσσερα βασικά τμήματα:

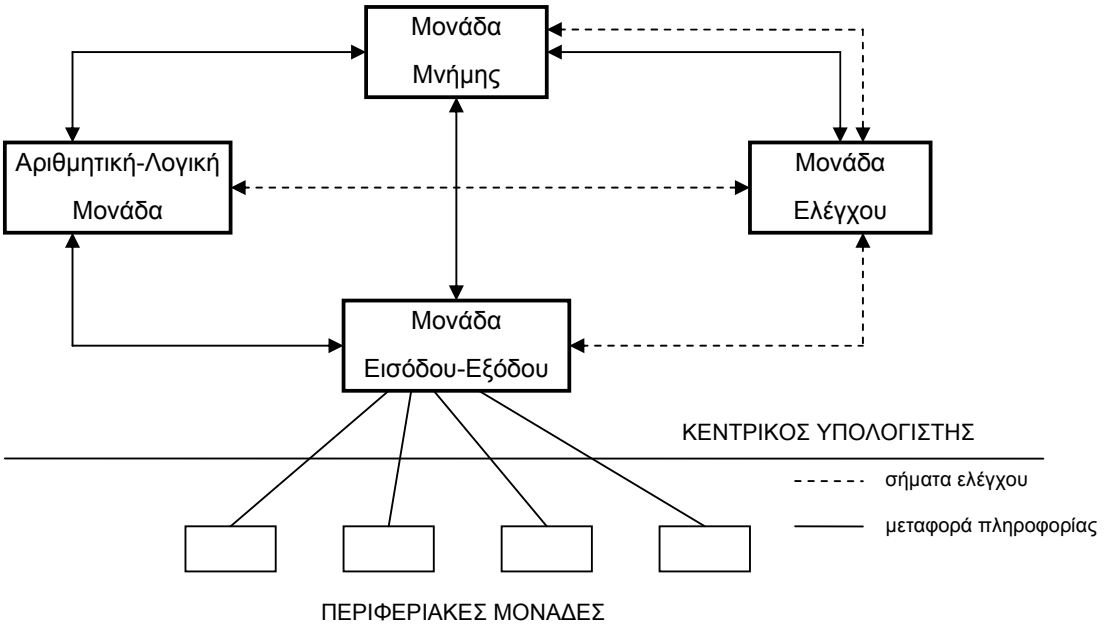
- την αριθμητική – λογική μονάδα (Arithmetic – Logic Unit ή ALU)
- τη μονάδα μνήμης (memory unit)
- τη μονάδα εισόδου – εξόδου (Input – Output ή I/O unit)
- τη μονάδα ελέγχου (control unit)

Η μονάδα ελέγχου, μαζί με την αριθμητική – λογική μονάδα αναφέρονται και σαν *κεντρική μονάδα επεξεργασίας (Central Processing Unit, CPU)*. Παρόλο που κάθε μία από τις μονάδες αυτές διαφέρει από υπολογιστή σε υπολογιστή ως προς την πολυπλοκότητα, το μέγεθος και τον τρόπο της υλοποίησής της, η εργασία που επιτελεί είναι σε γενικές γραμμές πάντα η ίδια.

Η αριθμητική – λογική μονάδα εκτελεί τις εξής λειτουργίες:

- εκτελεί τις βασικές αριθμητικές πράξεις (πρόσθεση, αφαίρεση, πολλαπλασιασμό και διαίρεση)
- εκτελεί τις λογικές πράξεις, π.χ. εύρεση του λογικού αθροίσματος OR, του λογικού γινομένου AND, κλπ)

- εκτελεί άλλες βοηθητικές εργασίες, π.χ. ολίσθηση (shift) του περιεχομένου ενός καταχωρητή.



Η μονάδα μνήμης χρησιμεύει για την αποθήκευση των αρχικών δεδομένων, των αποτελεσμάτων που προκύπτουν ενδιάμεσα, των τελικών αποτελεσμάτων και του συνόλου εντολών που πρέπει να εκτελέσει ο υπολογιστής.

Τα αρχικά δεδομένα ονομάζονται *δεδομένα εισόδου*, τα τελικά αποτελέσματα, *δεδομένα εξόδου* και το σύνολο των εντολών, *πρόγραμμα*.

Η μονάδα I/O επιτρέπει την επικοινωνία του ανθρώπου με τον υπολογιστή. Σε αυτήν συνδέονται οι περιφερειακές μονάδες, όπως το πληκτρολόγιο, το ποντίκι (για την είσοδο), η οθόνη, ο εκτυπωτής (για την έξοδο).

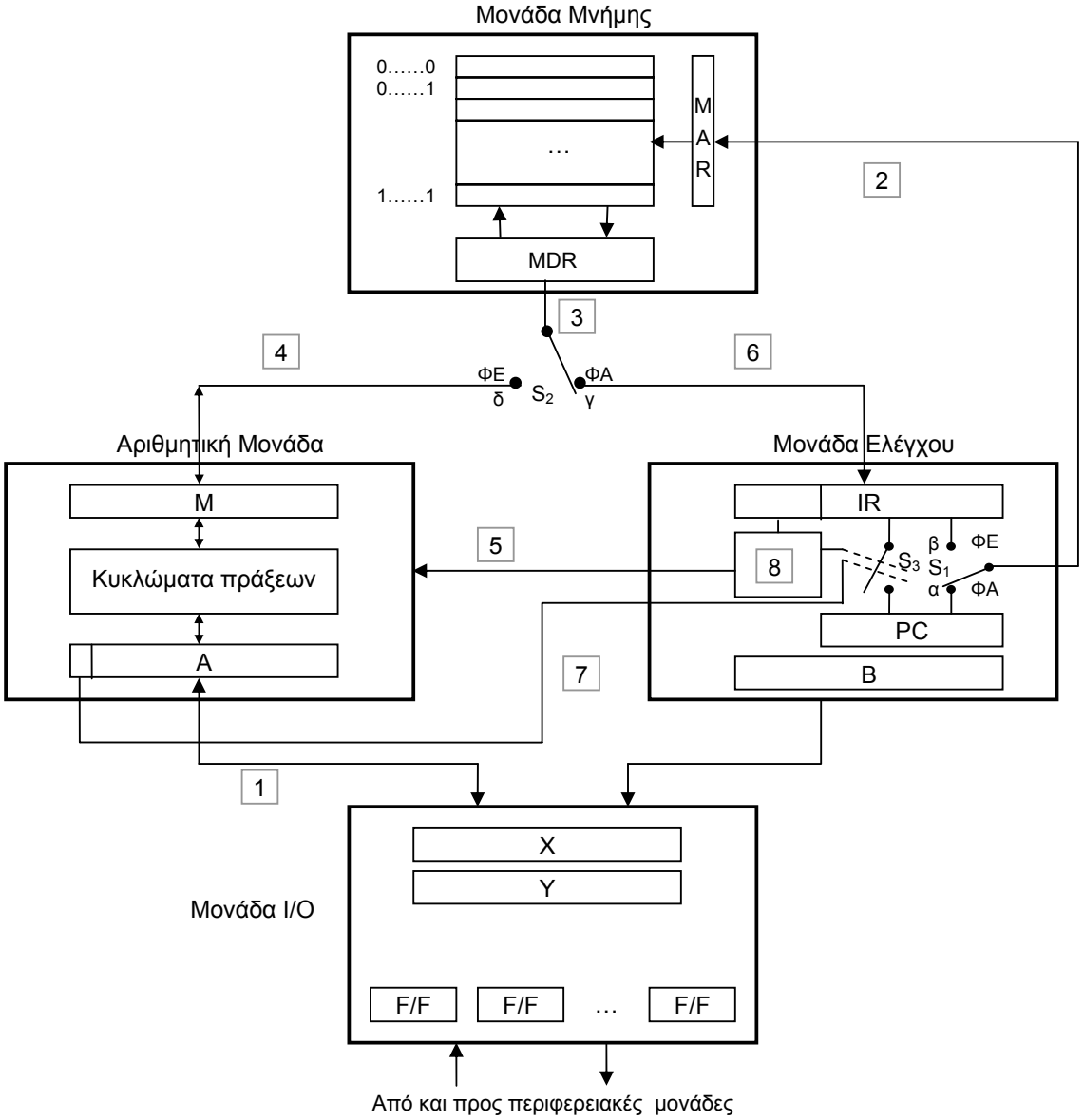
Η μονάδα ελέγχου είναι η πιο βασική μονάδα του υπολογιστή. Αποσκοπεί στον έλεγχο και συντονισμό όλης της λειτουργίας του. Παίρνει από τη μνήμη μία προς μία τις εντολές του προγράμματος, τις αναλύει σε στοιχειώδεις λειτουργίες και στέλνει στις διάφορες μονάδες λεπτομερείς οδηγίες για το τι και πότε πρέπει να εκτελέσουν.

Το υπολογιστικό αυτό μοντέλο περιγράφηκε για πρώτη φορά από τον Von Neumann το 1947. Σήμερα το ακολουθούν ακόμη, σε γενικές γραμμές, αρκετοί υπολογιστές, παρόλο που η αρχιτεκτονική τους έχει υποστεί πολλές αλλαγές.

2. Δομικά στοιχεία του υπολογιστή

Για την καλύτερη κατανόησης της δομής και της λειτουργίας των δομικών μονάδων που απαριθμήθηκαν προηγουμένως, έχουμε απεικονίσει στο παρακάτω σχήμα τα βασικά κυκλώματα ενός απλού ηλεκτρονικού υπολογιστή και τη ροή πληροφοριών μεταξύ τους. Πρόκειται για έναν υποθετικό υπολογιστή, που ονομάζουμε ΕΚΥ (ΕΚπαιδευτικός

Υπολογιστής). Ο ΕΚΥ είναι σκόπιμα απλός και μικρός, χωρίς αυτό να σημαίνει ότι λειτουργικά διαφέρει πολύ από τους πραγματικούς υπολογιστές.



2.1. Αριθμητική – Λογική μονάδα

Η ALU, όπως ήδη είπαμε, εκτελεί τις αριθμητικές και λογικές πράξεις, καθώς και διάφορες άλλες βοηθητικές λειτουργίες. Γενικά, αποτελείται από έναν αριθμό καταχωρητών και από τα απαραίτητα λογικά κυκλώματα για την εκτέλεση των παραπάνω πράξεων.

Συγκεκριμένα, ο υπολογιστής ΕΚΥ, διαθέτει έναν μόνο καταχωρητή γενικού σκοπού (accumulator), τον οποίο συχνά συμβολίζουμε ως A, και έναν καταχωρητή δείκτη, τον οποίο συνήθως συμβολίζουμε ως B. Επίσης, υπάρχουν και αρκετοί ακόμη καταχωρητές, που

χρησιμοποιούνται για τη μεταφορά δεδομένων και την αποθήκευση ενδιάμεσων αποτελεσμάτων, οι οποίοι δεν είναι άμεσα προσπελάσιμοι από τον προγραμματιστή. Χρησιμοποιούνται μόνο για την εκτέλεση προκαθορισμένων λειτουργιών από το υλικό.

Για την εκτέλεση μιας πράξης απαιτούνται δύο αριθμοί – ορίσματα. Ο ένας από αυτούς είναι το περιεχόμενο του καταχωρητή Α και ο άλλος του προσωρινού καταχωρητή Μ, που εικονίζεται στο παραπάνω σχήμα. Στον ΕΚΥ, το αποτέλεσμα μένει πάντα στον καταχωρητή Α. Για παράδειγμα, αν έχουμε να εκτελέσουμε διαδοχικές προσθέσεις, ο καταχωρητής Α συσσωρεύει όλους τους αριθμούς που προστίθενται, γι' αυτό ονομάζεται συσσωρευτής (accumulator).

Για την εκτέλεση των πράξεων πρέπει να μεταφέρονται δεδομένα προς και από τη μνήμη. Για το λόγο αυτό, όπως απεικονίζεται στο προηγούμενο σχήμα, υπάρχουν οι κατάλληλες συνδέσεις μεταξύ της μνήμης και της αριθμητικής μονάδας.

2.2. Μνήμη (Memory)

Η μνήμη διαιρείται σε θέσεις μνήμης. Κάθε *θέση* αποτελείται από συγκεκριμένο αριθμό από bytes (1 byte = 8 bits). Η *λέξη* του υπολογιστή αποτελείται από συγκεκριμένο αριθμό bytes (σταθερό για κάθε υπολογιστή). Όταν αποθηκεύεται στη μνήμη, καταλαμβάνει τον ίδιο αριθμό από συνεχόμενες θέσεις μνήμης (μπορεί να καταλαμβάνει 1 ή περισσότερες θέσεις μνήμης). Ο αριθμός των bits μιας λέξης (=8 × αριθμό bytes) ονομάζεται *μήκος λέξης (word length).*

Για να προσπελαύνουμε τις διάφορες θέσεις μνήμης, τις αριθμούμε και ο αριθμός που αντιστοιχεί σε κάθε μία από αυτές ονομάζεται *διεύθυνση (address)*.

Συγκεκριμένα, στον υπολογιστή ΕΚΥ, η λέξη έχει μήκος 16 bits. Επομένως, αποτελείται από 2 bytes. Επίσης, κάθε θέση μνήμης αποτελείται από 2 bytes. Επομένως, κάθε λέξη καταλαμβάνει ακριβώς μία θέση της μνήμης. Επίσης, κάθε εντολή της συμβολικής γλώσσας του αποθηκεύεται σε μία ακριβώς θέση μνήμης. Αυτό σημαίνει ότι, για να προσπελάσουμε την επόμενη εντολή, θα πρέπει κάθε φορά να προχωράμε 1 διεύθυνση παρακάτω.

Εκτός από τα flip-flops ή πυκνωτές, που αποσκοπούν καθαρά στην αποθήκευση των δεδομένων, η μνήμη χρειάζεται και κάποιον μηχανισμό προσπέλασης αυτών. Για το σκοπό αυτό περιλαμβάνει και δύο καταχωρητές, τους MDR και MAR, που ονομάζονται αντίστοιχα *καταχωρητής δεδομένων της μνήμης (memory data register)* και *καταχωρητής διευθύνσεων της μνήμης (memory address register)*.

Όταν μια λέξη πρόκειται να αποθηκευτεί στη μνήμη, πρέπει πρώτα να αποθηκευτεί στον καταχωρητή δεδομένων της μνήμης MDR και η διεύθυνση, στην οποία πρόκειται να γίνει η αποθήκευση, πρέπει να γραφεί στον καταχωρητή διευθύνσεων της μνήμης MAR. Στη συνέχεια, τα δεδομένα μεταφέρονται από τον MDR στην αντίστοιχη λέξη της μνήμης, η διεύθυνση της οποίας είναι γραμμένη στον MAR. Η διαδικασία αυτή ονομάζεται *εγγραφή* στη μνήμη. Η μνήμη διατηρεί το περιεχόμενό της αναλλοίωτο, μέχρι να συμβεί *επανεγγραφή*. Όταν γίνει επανεγγραφή σε μια θέση της μνήμης, το προηγούμενο περιεχόμενό της εξαφανίζεται. Η αντίστροφη διαδικασία της εγγραφής είναι η *ανάγνωση* μιας λέξης από τη μνήμη. Κατ' αυτήν, το περιεχόμενο μιας λέξης, της οποίας η διεύθυνση είναι γραμμένη στον

MAR, μεταφέρεται στον MDR, για να μεταφερθεί στη συνέχεια στις άλλες μονάδες του υπολογιστή.

Το μήκος τους MDR είναι όσο το μήκος της λέξης του υπολογιστή, ενώ το μήκος του MAR είναι k bits, όπου 2^k είναι το μέγιστο πλήθος θέσεων της μνήμης. (Επειδή 2^k το πλήθος αριθμοί μπορούν να παρασταθούν με k bits.)

Τέλος, η μνήμη περιέχει και αρκετά άλλα λογικά και ηλεκτρονικά κυκλώματα για τον έλεγχο της λειτουργίας της και για την επικοινωνία με τις υπόλοιπες μονάδες του υπολογιστή.

2.3. Μονάδα Εισόδου-Εξόδου (Input-Output, I/O Unit)

Η μονάδα εισόδου-εξόδου δέχεται τις πληροφορίες, που δίνονται από τον άνθρωπο, μέσω των περιφερειακών μονάδων υπό μορφή λέξεων και δίνει μέσω αυτών τα αποτελέσματα. Περιέχει έναν καταχωρητή δεδομένων Χ, που δέχεται από τις περιφερειακές μονάδες τις λέξεις που προορίζονται για την κεντρική μονάδα επεξεργασίας και το αντίστροφο. Επίσης, περιέχει έναν καταχωρητή Υ, που περιέχει τη διεύθυνση του περιφερειακού από ή προς το οποίο θα γίνει η μεταφορά της λέξης. Το μήκος του καταχωρητή Χ είναι μικρότερο ή ίσο με το μήκος λέξης του υπολογιστή και το μήκος του καταχωρητή Υ εξαρτάται από τον αριθμό των περιφερειακών μονάδων που μπορεί να έχει ο υπολογιστής.

Οι περιφερειακές μονάδες γενικά έχουν μικρή ταχύτητα, αρκετά μικρότερη από την ταχύτητα με την οποία μπορεί να γίνεται η επεξεργασία των αντίστοιχων δεδομένων από τη CPU. Επομένως, για να γίνεται συγχρονισμός της διαφοράς ταχύτητας μεταξύ CPU και περιφερειακών μονάδων, υπάρχουν μέσα στη μονάδα εισόδου-εξόδου flip-flops για κάθε περιφερειακό, που καθορίζουν αν έχει τελειώσει ή όχι μια μεταφορά λέξης από ή προς τον καταχωρητή Χ.

Τέλος, η μονάδα εισόδου-εξόδου περιέχει και αρκετά άλλα λογικά και ηλεκτρονικά κυκλώματα για τον έλεγχο της λειτουργίας της και για την επικοινωνία με τις υπόλοιπες μονάδες του υπολογιστή.

2.4. Μονάδα ελέγχου (Control Unit)

Η μονάδα ελέγχου αποτελείται από δύο κύριους καταχωρητές, τους IR και PC και από τα απαραίτητα κυκλώματα για τον έλεγχο και συντονισμό της λειτουργίας του υπολογιστή.

Ο καταχωρητής IR ονομάζεται *καταχωρητής εντολών (instruction register)* και είναι ο καταχωρητής εκείνος που δέχεται μία προς μία τις εντολές του προγράμματος από τη μνήμη για να αναγνωρισθούν, να αναλυθούν σε επιμέρους εργασίες και, τέλος, να εκτελεσθούν.

Ο καταχωρητής PC ονομάζεται *μετρητής προγράμματος*, ή *μετρητής εντολών (program counter, instruction counter)*. Το περιεχόμενό του δίνει κάθε φορά τη διεύθυνση της θέσης της μνήμης στην οποία είναι αποθηκευμένη η επόμενη εντολή. Η εντολή αυτή θα μεταφερθεί από τη μνήμη στον καταχωρητή IR της μονάδας ελέγχου. Όταν γίνει η μεταφορά, ο μετρητής προγράμματος θα αυξηθεί αυτόματα, ώστε να δείχνει την επόμενη εντολή που πρέπει να μεταφερθεί.

Το μήκος του καταχωρητή IR είναι ίσο με το μήκος λέξης του υπολογιστή, ενώ το μήκος του μετρητή προγράμματος ισούται με k, όπου 2^k είναι το πλήθος των λέξεων ή των διευθύνσεων της μνήμης του υπολογιστή.

3. Λειτουργία του Υπολογιστή

Όπως αναφέραμε και παραπάνω, στη μνήμη αποθηκεύονται δεδομένα και εντολές. Υπάρχει, λοιπόν, μια ομάδα διευθύνσεων, που αντιστοιχούν σε θέσεις της μνήμης, που περιέχουν εντολές. Συνήθως, οι εντολές αποθηκεύονται σε διαδοχικές λέξεις της μνήμης, σύμφωνα με την επιθυμητή σειρά εκτέλεσής τους.

Για κάθε εντολή, ο υπολογιστής συμπληρώνει δύο φάσεις, τις οποίες ονομάζουμε *φάση ανάκλησης (fetch phase)* και *φάση εκτέλεσης (execution phase)*. Οι φάσεις μπορεί να αναφέρονται και σαν κύκλοι, δηλαδή *κύκλος ανάκλησης (fetch cycle)* και *κύκλος εκτέλεσης (execution cycle)*.

Κατά τη φάση ανάκλησης, η εντολή, της οποίας η διεύθυνση βρίσκεται στο μετρητή προγράμματος PC, μεταφέρεται από τη μνήμη στον MDR και ύστερα, μέσω της διαδρομής 3→6, στον καταχωρητή εντολών IR της μονάδας ελέγχου. Οι διακόπτες S_1 και S_2 βρίσκονται κατά τη φάση ανάκλησης στις θέσεις α και γ αντίστοιχα. Στην πραγματικότητα, τα κυκλώματα που στο σχήμα συμβολίζονται ως διακόπτες είναι κυκλώματα λογικών πυλών (π.χ. αποπλέκτες), τα οποία δέχονται σήματα από τη μονάδα ελέγχου. Όταν ολοκληρωθεί η μεταφορά της εντολής στον IR, ο μετρητής προγράμματος PC αυξάνεται αυτόματα, ώστε να προετοιμαστεί η μεταφορά της επόμενης εντολής. Συγκεκριμένα, στον ΕΚΥ, που οι εντολές καταλαμβάνουν μία θέση μνήμης η κάθε μία, ο μετρητής προγράμματος αυξάνεται κάθε φορά κατά 1. Στο σημείο αυτό τελειώνει η φάση ανάκλησης.

Έπειτα ακολουθεί η φάση εκτέλεσης. Κατά τη φάση εκτέλεσης, η εντολή που βρίσκεται στον καταχωρητή εντολών, αποκωδικοποιείται και αναλύεται σε επιμέρους εργασίες που πρέπει να εκτελεστούν. Στη συνέχεια, η μονάδα ελέγχου στέλνει στις υπόλοιπες μονάδες τα κατάλληλα σήματα ελέγχου για την εκτέλεση των επιμέρους εργασιών με την κατάλληλη σειρά. Η εκτέλεση των επιμέρους εργασιών σημαίνει και την εκτέλεση της εντολής, οπότε τελειώνει η φάση εκτέλεσης και κλείνει ο κύκλος φάση ανάκλησης – φάση εκτέλεσης για τη συγκεκριμένη εντολή.

Στη συνέχεια, έχουμε νέο κύκλο φάσεως ανάκλησης - φάσεως εκτέλεσης για νέα εντολή. Κατά το νέο κύκλο θα γίνει πρώτα η μεταφορά από τη μνήμη της επόμενης εντολής, δεδομένου ότι κατά τον προηγούμενο κύκλο είχε αυξηθεί κατάλληλα το περιεχόμενο του μετρητή προγράμματος. Στη συνέχεια, θα ακολουθήσει η εκτέλεση της νέας εντολής, οπότε συμπληρώνεται ο νέος κύκλος.

Οι κύκλοι των δύο φάσεων (ανάκλησης – εκτέλεσης) θα επαναλαμβάνονται συνέχεια, μέχρι να εμφανισθεί και να εκτελεσθεί μία συγκεκριμένη εντολή, με την οποία διακόπτεται η λειτουργία του υπολογιστή.

II. ΣΥΜΒΟΛΙΚΗ ΓΛΩΣΣΑ (ASSEMBLY) ΤΟΥ ΕΚΥ

Για να ζητήσουμε από το υλικό του υπολογιστή να εκτελέσει μια λειτουργία, πρέπει να «μιλάμε τη γλώσσα του». Οι λέξεις στη γλώσσα του υπολογιστή ονομάζονται *εντολές* και το λεξιλόγιο είναι το *σύνολο εντολών*. Στο κεφάλαιο αυτό θα δούμε το σύνολο εντολών του ΕΚΥ στη μορφή που είναι εύχρηστη για τους ανθρώπους και στη μορφή που είναι κατανοητή από τους υπολογιστές.

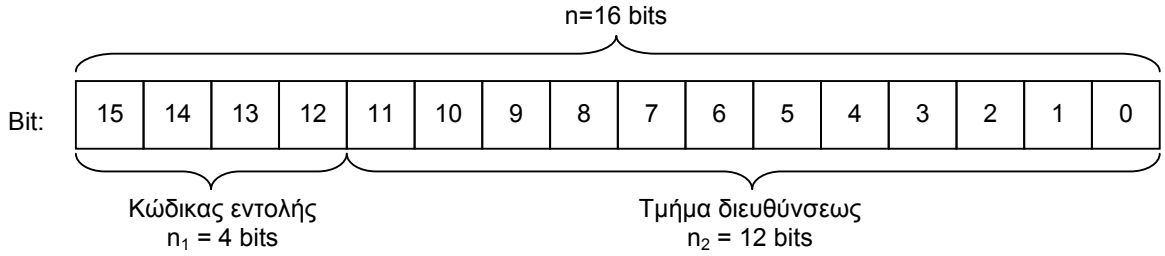
Μπορεί κανείς να σκεφτεί ότι οι γλώσσες των υπολογιστών ποικίλουν όσο οι γλώσσες των ανθρώπων. Στην πραγματικότητα, όμως, οι γλώσσες των υπολογιστών μοιάζουν πολύ μεταξύ τους, όπως οι τοπικές διάλεκτοι της ίδιας γλώσσας. Επομένως, αν κανείς μάθει μία από αυτές μπορεί πολύ εύκολα να εξοικειωθεί και με τις υπόλοιπες. Η ομοιότητα αυτή μεταξύ τους οφείλεται στο ότι όλοι οι υπολογιστές κατασκευάζονται από τεχνολογίες υλικού που ακολουθούν τις ίδιες βασικές αρχές και στο ότι υπάρχουν κάποιες βασικές λειτουργίες που όλοι οι υπολογιστές πρέπει να εξυπηρετούν. Επίσης, οι αρχιτέκτονες υπολογιστών έχουν πάντα τον ίδιο σκοπό: να βρουν μια γλώσσα που κάνει εύκολη τόσο την κατασκευή του υλικού, όσο και του compiler, και μεγιστοποιεί την απόδοση, ενώ ταυτόχρονα ελαχιστοποιεί το κόστος.

1. Εντολές γλώσσας ASSEMBLY του ΕΚΥ

1.1. Μορφή εντολής

Στον ΕΚΥ, *κάθε εντολή παριστάνεται με μία λέξη του υπολογιστή*. Υπάρχουν και υπολογιστές, στους οποίους κάθε εντολή παριστάνεται με δύο ή περισσότερες λέξεις, ή αντίστροφα, μία λέξη παριστάνει πολλές εντολές.

Στον ΕΚΥ, η εντολή χωρίζεται νοερά σε δύο τμήματα, από τα οποία το πρώτο ονομάζεται κώδικας εντολής (instruction code) και το δεύτερο τμήμα διεύθυνσης (address part), όπως φαίνεται και στο παρακάτω σχήμα. Το πρώτο τμήμα είναι αυτό που χαρακτηρίζει την εντολή.



Κάθε υπολογιστής μπορεί να εκτελέσει ένα συγκεκριμένο ρεπερτόριο εντολών, το οποίο καθορίζεται κατά το σχεδιασμό του. Ο κώδικας εντολής είναι ένας δυαδικός αριθμός μήκους n_1, διαφορετικός για κάθε εντολή. Επομένως, ο μέγιστος αριθμός εντολών που μπορεί να έχει ένας υπολογιστής με μήκος του κώδικα εντολής n_1, είναι 2^{n_1}. Στον ΕΚΥ, n_1=4, οπότε ο αριθμός των δυνατών διαφορετικών εντολών είναι 2^4 = 16. Ο πίνακας που περιγράφει τις εντολές ενός υπολογιστή και τους αντίστοιχους δυαδικούς κώδικές τους ονομάζεται *πίνακας εντολών* του συγκεκριμένου υπολογιστή και περιέχει το *ρεπερτόριο εντολών* αυτού Οι εντολές χωρίζονται σε ομάδες, ανάλογα με τη λειτουργία που επιτελούν π.χ. εντολές εκτέλεσης αριθμητικών πράξεων, εντολές μεταφοράς δεδομένων, εντολές άλματος, εντολές ολίσθησης, κ.λ.π.

Το τμήμα διεύθυνσης της εντολής στον ΕΚΥ έχει μήκος 12 bits. Αυτό παριστάνει τη διεύθυνση μιας λέξης της μνήμης, η οποία αποτελεί το όρισμα της εντολής, που περιγράφεται από τον κώδικα της εντολής.

Για παράδειγμα, η εντολή 0011 000000001101 του ΕΚΥ, έχει κώδικα εντολής 0011 και τμήμα διεύθυνσης 000000001101. Η εντολής με κώδικα 0011 σημαίνει «Πρόσθεση στο περιεχόμενο του συσσωρευτή του περιεχομένου της διεύθυνσης της μνήμης, την οποία διεύθυνση δείχνει το τμήμα διεύθυνσης της εντολής. Το αποτέλεσμα της πρόσθεσης μένει στο συσσωρευτή.» Επομένως, η εντολή 0011 000000001101 σημαίνει «Πρόσθεση στο συσσωρευτή του περιεχομένου της διεύθυνσης 000000001101. Το αποτέλεσμα μένει στο συσσωρευτή.»

Στο σημείο αυτό πρέπει να επισημάνουμε τη διαφορά μεταξύ των όρων «διεύθυνση εντολής» και «τμήμα διευθύνσεως εντολής». Διεύθυνση εντολής είναι η διεύθυνση της θέσης της μνήμης, στην οποία είναι αποθηκευμένη η εντολή. Τμήμα διεύθυνσης της εντολής είναι τα n_2 δεξιότερα bits αυτής. Το τμήμα διεύθυνσης της εντολής μπορεί να δείχνει τη διεύθυνση κάποιας θέσης της μνήμης, της οποίας το περιεχόμενο θα χρησιμοποιηθεί κατά την εκτέλεση της εντολής.

1.2. Εντολές αναφοράς στη μνήμη (Memory Reference Instructions)

Οι γλώσσες προγραμματισμού έχουν απλές μεταβλητές, που περιέχουν απλά δεδομένα, αλλά έχουν και πιο πολύπλοκες δομές: πίνακες και δομές δεδομένων. Οι πολύπλοκες αυτές δομές δεδομένων μπορεί να περιέχουν πολύ περισσότερα δεδομένα από αυτά που χωρούν στους καταχωρητές του υπολογιστή, όσους καταχωρητές και αν διαθέτει ένας υπολογιστής

(Αυτό, λοιπόν ισχύει πολύ περισσότερο για τον ΕΚΥ, ο οποίος έχει μόνο έναν καταχωρητή-συσσωρευτή, τον Α). Πώς θα γίνει, λοιπόν, η αναπαράσταση και η προσπέλαση αυτών των μεγάλων δομών δεδομένων; Η απάντηση είναι ότι πολύ λίγα δεδομένα χωρούν στους καταχωρητές, αλλά η μνήμη του υπολογιστή μπορεί να αποθηκεύσει πολλές τάξεις μεγέθους περισσότερα. Επομένως, οι δομές δεδομένων αποθηκεύονται στη μνήμη.

Θα πρέπει, επομένως, στο ρεπερτόριο των εντολών του να συμπεριλαμβάνονται πάντα και εντολές που μεταφέρουν δεδομένα μεταξύ της μνήμης και των καταχωρητών. Οι εντολές αυτές ονομάζονται *εντολές μεταφοράς δεδομένων*. Για να προσπελαστεί μία λέξη της μνήμης, η εντολή πρέπει να παρέχει τη διεύθυνσή της.

Η εντολή μεταφοράς δεδομένων που αντιγράφει δεδομένα από τη μνήμη σε έναν καταχωρητή λέγεται *εντολή φόρτωσης (load)* και συμβολίζεται ως

```
LDA         XXX
```

Αυτό σημαίνει:

```
Αντέγραψε στον καταχωρητή Α τα περιεχόμενα της λέξης της μνήμης με
διεύθυνση XXX – LoaD A
```

Η αντίστροφη εντολή της φόρτωσης είναι η *εντολή αποθήκευσης (store)*. Αυτή αντιγράφει δεδομένα από έναν καταχωρητή στη μνήμη και συμβολίζεται ως

```
STA         XXX
```

Αυτό σημαίνει:

```
Αντέγραψε στη λέξη της μνήμης με διεύθυνση XXX τα περιεχόμενα του
καταχωρητή Α – STore A
```

Εκτός από τις δύο αυτές εντολές που μόλις περιγράψαμε, στον ΕΚΥ, στην ίδια κατηγορία (εντολές αναφοράς στη μνήμη) εντάσσονται και οι εντολές που φορτώνουν δεδομένα από τη μνήμη και εκτελούν κάποια αριθμητική πράξη με αυτά. Για το λόγο αυτό, στις εντολές αυτές το τμήμα διεύθυνσης δείχνει τη διεύθυνση ενός αριθμού. Στον παρακάτω πίνακα περιγράφονται οι εντολές αυτές, μαζί με τους αντίστοιχους δυαδικούς κώδικές τους.

α/α	Κώδικας εντολής (περιεχόμενο των 4 πρώτων bits της εντολής)	Μνημονικό όνομα εντολής	Σημασία εντολής	Επεξήγηση
1	0001	LDA	(A) := (N)	Μεταφορά στο συσσωρευτή Α του περιεχομένου της διεύθυνσης Ν της μνήμης
2	0010	STA	(N) := (A)	Μεταφορά στη διεύθυνση Ν της μνήμης του περιεχομένου του συσσωρευτή Α
3	0011	ADA	(A) := (A) + (N)	Πρόσθεση του περιεχομένου του συσσωρευτή Α με το περιεχόμενο της διεύθυνσης Ν της μνήμης. Το αποτέλεσμα μένει στο συσσωρευτή και αποτελεί το νέο περιεχόμενό του.
4	0100	SBA	(A) := (A) - (N)	Αφαίρεση από το περιεχόμενο του συσσωρευτή Α του περιεχομένου της διεύθυνσης Ν της μνήμης. Το αποτέλεσμα μένει στο συσσωρευτή Α.

5	0101	MLA	(A) := (A) * (N)	Πολλαπλασιασμός του περιεχομένου του συσσωρευτή Α με το περιεχόμενο της διεύθυνσης Ν της μνήμης. Το αποτέλεσμα μένει στο συσσωρευτή Α.
6	0110	DVA	(A) := (A) / (N)	Πολλαπλασιασμός του περιεχομένου του συσσωρευτή Α με το περιεχόμενο της διεύθυνσης Ν της μνήμης. Το πηλίκο της διαίρεσης μένει στο συσσωρευτή Α, το υπόλοιπο χάνεται.

Στον παραπάνω πίνακα χρησιμοποιούμε τον εξής συμβολισμό:

:= → ανάθεση τιμής

(X**)** → το περιεχόμενο του καταχωρητή X, αν το X είναι καταχωρητής, ή το

Για παράδειγμα, (A) := (00000010001) σημαίνει ότι το περιεχόμενο του καταχωρητή Α γίνεται ίσο με το περιεχόμενο της διεύθυνσης 00000010001. Στη συνέχεια, θα παρακολουθήσουμε, με τη βοήθεια του παραπάνω πίνακα, τη λειτουργία του ΕΚΥ κατά την εκτέλεση ενός στοιχειώδους προγράμματος.

Παράδειγμα:

Ζητείται να γραφεί πρόγραμμα που να προσθέτει τα προσθέτει τα περιεχόμενα των διευθύνσεων 0000001010 και 0000001011 της μνήμης και να αποθηκεύει το αποτέλεσμα της πρόσθεσης στη διεύθυνση 0000001100.

Υποθέτουμε ότι οι αριθμοί που είναι για πρόσθεση, καθώς και οι εντολές του προγράμματος έχουν ήδη τοποθετηθεί στη μνήμη στις κατάλληλες θέσεις, πριν την έναρξή του. Επομένως, πριν την έναρξη του προγράμματος, το περιεχόμενο της μνήμης θα είναι το εξής:

Διεύθυνση μνήμης	Περιεχόμενο		
	Κώδικας εντολής	Τμήμα διεύθυνσης	
000000000000	0001	000000001010	Εντολές προγράμματος
000000000001	0011	000000001011	
000000000010	0010	000000001100	
000000000011	0000	000000000000	
…	…	…	
000000001010	0000	000100000000	Δεδομένα
000000001011	0000	000100000001	
000000001100	0000	000000000000	Διεύθυνση αποθήκευσης αποτελέσματος
…	…	…	

Ο πίνακας αυτός αποτελεί ταυτόχρονα και το ζητούμενο πρόγραμμα σε *γλώσσα μηχανής* του ΕΚΥ. Στο σημείο αυτό, λοιπόν, αξίζει να επισημάνουμε τη διαφορά μεταξύ γλώσσας μηχανής και συμβολικής γλώσσας (ASSEMLY) ενός υπολογιστή (βλέπε και σελίδα 18). Η γλώσσα μηχανής είναι η γλώσσα που γίνεται απ' ευθείας αντιληπτή από τον υπολογιστή. Η συμβολική γλώσσα είναι αντιληπτή από τον άνθρωπο. Η διαφορά της συμβολικής γλώσσας από τις γνωστές γλώσσες προγραμματισμού υψηλού επιπέδου έγκειται στο ότι οι εντολές της συμβολικής γλώσσας αντιστοιχίζονται σε μία ακριβώς εντολή της γλώσσας μηχανής. Επομένως, κάθε υπολογιστής, έχει τη δική του συμβολική γλώσσα, της οποίας οι εντολές αντιστοιχούν μία προς μία με τις εντολές μηχανής αυτού.

Για να γράψει ο προγραμματιστής ένα πρόγραμμα σε γλώσσα μηχανής ενός υπολογιστή, πρέπει να γράψει τις εντολές μία προς μία, τα δεδομένα, και τις διευθύνσεις των θέσεων μνήμης, οι οποίες θα περιέχουν τις εντολές και τα δεδομένα, όπως στον παραπάνω πίνακα.

Στο παράδειγμα αυτό, υποθέτουμε, επίσης, ότι το περιεχόμενο του μετρητή προγράμματος PC είναι αρχικά ίσο με 0. Πιέζοντας το πλήκτρο START του υπολογιστή, αρχίζει η λειτουργία του. Αυτό σημαίνει μία συνεχή επανάληψη κύκλων των δύο φάσεων ανάκλησης ($ΦA_i$) και εκτέλεσης ($ΦE_i$) για κάθε εντολή i. Στο παράδειγμά μας το i παίρνει τις τιμές 1, 2, 3, 4. Στη συνέχεια, περιγράφουμε αναλυτικά τι συμβαίνει κατά τις φάσεις ανάκλησης και εκτέλεσης του παραπάνω προγράμματος.

$ΦA_1$: Αφού το περιεχόμενο του μετρητή προγράμματος είναι 0, η εντολή που βρίσκεται στη διεύθυνση 0 της μνήμης, μεταφέρεται στον καταχωρητή εντολών IR μέσω της διαδρομής (3)→(6), στο σχήμα της σελίδας 3. Για να γίνει αυτό, οι διακόπτες S_1 και S_2, που αλλάζουν θέση ταυτόχρονα, βρίσκονται στις θέσεις α και γ, αντίστοιχα. Το αποτέλεσμα της μεταφοράς αυτής είναι `(IR) := 0001000000001010`. Στη συνέχεια, το περιεχόμενο του μετρητή προγράμματος PC αυξάνεται αυτόματα κατά μία μονάδα, οπότε γίνεται: `(PC) := 000000000001`.

$ΦE_1$: Οι διακόπτες S_1 και S_2 αλλάζουν θέση και έρχονται αντίστοιχα στις θέσεις β και δ. Η εντολή, που είναι γραμμένη στον καταχωρητή εντολών IR, αναγνωρίζεται με τη βοήθεια ενός αποκωδικοποιητή από τη μονάδα ελέγχου, με εξέταση του τμήματος κώδικα εντολής, που είναι 0001. Αναγνωρίζεται, επομένως η εντολή `LDA`, (A):=(N), όπου, στη συγκεκριμένη περίπτωση N=000000001010. Η μονάδα ελέγχου, αναλύει, στη συνέχεια, την εντολή σε στοιχειώδεις εργασίες, που πρέπει να εκτελεσθούν. Στη συγκεκριμένη περίπτωση, πρέπει

α) Η διεύθυνση N=000000001010 να σταλεί στον καταχωρητή διευθύνσεων MAR της μνήμης, μέσω του διακόπτη S_1 και της διαδρομής (2).

β) Το περιεχόμενο της διεύθυνσης N να μεταφερθεί από τη μνήμη στον καταχωρητή δεδομένων της μνήμης MDR.

γ) Το περιεχόμενο του καταχωρητή δεδομένων της μνήμης να μεταφερθεί από τη διαδρομή (3)→(4), μέσω του διακόπτη S_2, στο συσσωρευτή A της αριθμητικής μονάδας.

Το αποτέλεσμα της εκτέλεσης της εντολής αυτής είναι: (A) := 0000000100000000.

$ΦA_2$: Οι διακόπτες S_1 και S_2 πηγαίνουν στις θέσεις α και γ αντίστοιχα. Επομένως, γίνεται το εξής: `(IR) := ((PC)) = (000000000001)`, δηλαδή `(IR) :=`

0011000000001011. Τέλος, (PC) := (PC) +1, δηλαδή, (PC) := 000000000010.

ΦΕ$_2$: Οι διακόπτες S_1 και S_2 μεταφέρονται πάλι στις θέσεις β και δ, αντίστοιχα. Αναγνωρίζεται η εντολή ADA, (A) := (A) + (N) (κώδικας 0011), όπου N=000000001011. Το περιεχόμενο της διεύθυνσης 000000001011 μεταφέρεται στον καταχωρητή Μ της αριθμητικής μονάδας και προστίθεται στο περιεχόμενο του συσσωρευτή Α. (A) := (A) + (000000001011) = 0000000100000000 + 0000000100000001 = 0000001000000001.

ΦΑ$_3$: Οι διακόπτες S_1 και S_2 πηγαίνουν στις θέσεις α και γ αντίστοιχα. Επομένως, γίνεται το εξής: (IR) := ((PC)) = (000000000010), δηλαδή (IR) := 0010000000001100. Τέλος, (PC) := (PC) +1, δηλαδή, (PC) := 000000000011.

ΦΕ$_3$: Οι διακόπτες S_1 και S_2 μεταφέρονται πάλι στις θέσεις β και δ, αντίστοιχα. Αναγνωρίζεται η εντολή STA, (N) := (A), όπου N=000000001100. Επομένως, το περιεχόμενο του Α μεταφέρεται στη διεύθυνση 000000001100. Αποτέλεσμα της εκτέλεσης της εντολής αυτής είναι (000000001100) := 0000001000000001.

ΦΑ$_4$: Οι διακόπτες S_1 και S_2 πηγαίνουν στις θέσεις α και γ αντίστοιχα. Επομένως, γίνεται το εξής: (IR) := ((PC)) = (000000000011), δηλαδή (IR) := 0000000000000000. Τέλος, (PC) := (PC) +1, δηλαδή, (PC) := 000000000100.

ΦΕ$_4$: Οι διακόπτες S_1 και S_2 μεταφέρονται πάλι στις θέσεις β και δ, αντίστοιχα. Αναγνωρίζεται η εντολή με κώδικα 0000. Πρόκειται για την εντολή διακοπής της λειτουργίας του υπολογιστή. Το τμήμα διεύθυνσης της εντολής αυτής δε χρησιμοποιείται και μπορεί να περιέχει ο,τιδήποτε.

Μετά τον τερματισμό του παραπάνω προγράμματος, τα περιεχόμενα της μνήμης είναι σχεδόν τα ίδια με προηγουμένως, με τη μόνη διαφορά ότι το περιεχόμενο της διεύθυνσης 000000001100 δεν είναι 0 όπως πριν, αλλά 0000001000000001, ίσο δηλαδή με το άθροισμα των περιεχομένων των διευθύνσεων 000000001010 και 000000001011 της μνήμης.

Παρατηρούμε ότι στις διευθύνσεις 000000000000 έως και 000000000011, το περιεχόμενο των αντίστοιχων θέσεων μνήμης είναι εντολές, ενώ στις διευθύνσεις 000000001010 έως και 000000001100 έχουμε δεδομένα. Ο υπολογιστής δεν κάνει διάκριση μεταξύ εντολών και δεδομένων. Επομένως, είναι ευθύνη του προγραμματιστή να γράψει το πρόγραμμα κατά τέτοιο τρόπο, ώστε να υπολογιστής να εκτελεί εντολές και όχι δεδομένα ως εντολές.

Για συντομία, τα προγράμματα συνήθως δεν γράφονται σε δυαδική μορφή, αλλά σε οκταδική ή δεκαεξαδική.. Σε δεκαεξαδική μορφή, χρησιμοποιούμε 3 ψηφία για τις διευθύνσεις, ενώ για την παράσταση λέξεων 4 ψηφία. Το πρόγραμμα του παραπάνω πίνακα, γράφεται, επομένως, σε δεκαεξαδική μορφή ως εξής:

Διεύθυνση μνήμης	Περιεχόμενο	
	Κώδικας εντολής	Τμήμα διεύθυνσης
000	1	00A
001	3	00B
002	2	00C
003	0	000
…	…	…
00A	0	100
00B	0	101
00C	0	000
…	…	…

1.3. Εντολές άλματος (Jump Instructions)

Η κυριότερη διαφορά μεταξύ ενός υπολογιστή και μίας αριθμομηχανής είναι ότι ο υπολογιστής μπορεί να παίρνει αποφάσεις και να τροποποιεί τη ροή του προγράμματος, ανάλογα με το αποτέλεσμά τους. Στις γλώσσες προγραμματισμού υψηλού επιπέδου, η λήψη αποφάσεων υλοποιείται με την εντολής `if`, σε συνδυασμό, κάποιες φορές, με την εντολή `goto` και τη χρήση ετικετών. Εξάλλου, ακόμη και οι δομές των βρόχων (`while, repeat`, κλπ) μπορούν να παρασταθούν με χρήση μόνο των `if` και `goto`.

Η διαφοροποίηση της ροής εκτέλεσης του προγράμματος πραγματοποιείται με τις εντολές άλματος (jump instructions). Ο ΕΚΥ διαθέτει δύο εντολές άλματος, την εντολή άλματος χωρίς συνθήκη (unconditional jump instruction) και την εντολή άλματος υπό συνθήκη (conditional jump instruction).

Η *εντολή άλματος χωρίς συνθήκη* συμβολίζεται ως

```
JMP          XXX
```

Αυτό σημαίνει:

```
Η επόμενη εντολή που πρέπει να εκτελεστεί βρίσκεται στη λέξη της
μνήμης με διεύθυνση XXX - JuMP
```

Κατά την εκτέλεση της εντολής αυτής, το περιεχόμενο του τμήματος διευθύνσεων του καταχωρητή εντολών (Instruction Register – IR), αντιγράφεται στο μετρητή προγράμματος.

Η *εντολή άλματος υπό συνθήκη* συμβολίζεται ως

```
JAN          XXX
```

Αυτό σημαίνει:

Αν το περιεχόμενο του καταχωρητή Α είναι αρνητικός αριθμός, η επόμενη εντολή που πρέπει να εκτελεστεί βρίσκεται στη λέξη της μνήμης με διεύθυνση XXX – Jump if A Negative

Κατά την εκτέλεση της εντολής αυτής, το περιεχόμενο του τμήματος διευθύνσεων του καταχωρητή εντολών (IR), αντιγράφεται στο μετρητή προγράμματος μόνο αν το περιεχόμενο του καταχωρητή Α είναι αρνητικός αριθμός, δηλ. αν το πρώτο bit του είναι ίσο με 1. Διαφορετικά, η εντολή αυτή δεν έχει καμία επίδραση στην εκτέλεση του προγράμματος.

1.4. Εντολές ολίσθησης (Shift Instruction)

Ο ΕΚΥ διαθέτει δύο εντολές ολίσθησης. Η πρώτη ονομάζεται *εντολή ολίσθησης προς τα αριστερά* και συμβολίζεται ως

```
SAL         XXX
```

Αυτό σημαίνει:

Ολίσθησε το περιεχόμενο του καταχωρητή Α κατά XXX θέσεις αριστερά – Shift A Left

Επισημαίνουμε ότι το τμήμα διεύθυνσης της εντολής αυτής περιέχει τον αριθμό των ολισθήσεων που πρέπει να πραγματοποιηθούν και όχι κάποια διεύθυνση. Τα XXX δεξιά bits του αριθμού αντικαθίστανται με μηδενικά bits καθώς μετατοπίζονται. Επομένως, αν τα XXX+1 αριστερότερα bits ενός θετικού προσημασμένου ακεραίου αριθμού, ή τα XXX αριστερότερα bits ενός μη προσημασμένου ακεραίου αριθμού είναι ίσα με 0, τότε η εντολή αυτή ισοδυναμεί με πολλαπλασιασμό του αριθμού με 2^{XXX}.

Η δεύτερη εντολή ολίσθησης ονομάζεται *εντολή ολίσθησης προς τα δεξιά* και συμβολίζεται ως

```
SAR         XXX
```

Αυτό σημαίνει:

Ολίσθησε το περιεχόμενο του καταχωρητή Α κατά XXX θέσεις δεξιά – Shift A Right

Τα XXX αριστερά bits του αριθμού αντικαθίστανται με μηδενικά bits καθώς μετατοπίζονται. Επομένως, αν πρόκειται για μη προσημασμένο ή θετικό προσημασμένο ακέραιο αριθμό, η εντολή αυτή ισοδυναμεί με ακέραια διαίρεση του αριθμού με 2^{XXX}.

1.5. Εντολές Εισόδου – Εξόδου (Input – Output Instructions)

Όπως αναφέραμε και στην παράγραφο 2.3 του μέρους Ι. του κειμένου αυτού, η επικοινωνία ανθρώπου – υπολογιστή πραγματοποιείται με τις περιφερειακές μονάδες, μέσω της μονάδας εισόδου-εξόδου (η οποία συμβολίζεται ως μονάδα I/O). Υπάρχουν πολλών ειδών περιφερειακές μονάδες στους Η/Υ. Για απλούστευση εδώ θα θεωρήσουμε ότι ο ΕΚΥ διαθέτει μόνο ένα τηλέτυπο, συνδεδεμένο με τη μονάδα I/O. Αυτό μπορεί να είναι πολύ απαρχαιωμένο σαν περιφερειακή συσκευή, αλλά η αρχή λειτουργίας του είναι σχεδόν ίδια με όλες τις σύγχρονες περιφερειακές συσκευές. Με το τηλέτυπο μπορούν να εισαχθούν

δεδομένα και εντολές στον ΕΚΥ, αλλά και να ληφθούν αποτελέσματα που προκύπτουν από την εκτέλεση των προγραμμάτων.

Ο ΕΚΥ διαθέτει δύο εντολές I/O. Η πρώτη είναι η *εντολή εξόδου*. Το αποτέλεσμα της εντολής αυτής είναι να στέλνονται προς το τηλέτυπο τα τρία δεξιά bits του καταχωρητή A (bits 2, 1, 0), τα οποία εκτυπώνονται σαν ένας οκταδικός αριθμός. Με την ίδια εντολή μπορούν να αποσταλούν προς το τηλέτυπο τα 7 δεξιά bits του καταχωρητή A (bits 6, …, 0), τα οποία εκτυπώνονται σαν ένας χαρακτήρας ASCII (βλέπε Παράρτημα). Το εάν θα αποσταλεί οκταδικός αριθμός ή ASCII χαρακτήρας εξαρτάται από το bit 0 της εντολής. Αν το bit αυτό ισούται με 1, θα αποσταλεί οκταδικό ψηφίο, αν ισούται με 0, θα αποσταλεί χαρακτήρας ASCII.

Η εντολή αυτή συμβολίζεται ως εξής:

```
OUT, 1
```

προκειμένου για εκτύπωση οκταδικού αριθμού, ή

```
OUT, 0
```

προκειμένου για εκτύπωση χαρακτήρα ASCII.

Η δεύτερη εντολή I/O είναι η *εντολή εισόδου*. Τα αποτελέσματα της εκτέλεσής της είναι ακριβώς τα αντίθετα των αποτελεσμάτων της εντολής εξόδου.

Η εντολή αυτή συμβολίζεται ως εξής:

```
INP, 1
```

προκειμένου για ανάγνωση οκταδικού αριθμού, ο οποίος καταλαμβάνει τα τρία δεξιά bits του καταχωρητή A, ή

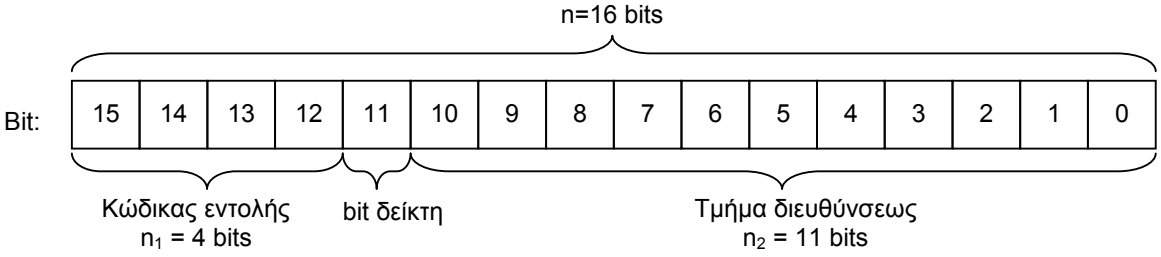
```
INP, 0
```

προκειμένου για ανάγνωση χαρακτήρα ASCII, ο οποίος καταλαμβάνει τα 7 δεξιά bits του καταχωρητή A.

Στο σημείο αυτό επισημαίνουμε ότι και στις εντολές I/O, το τμήμα διευθύνσεως δεν περιέχει κάποια διεύθυνση. Το bit 0 του τμήματος διεύθυνσης καθορίζει αν θα γίνει μεταφορά οκταδικού αριθμού ή χαρακτήρα ASCII. Τα υπόλοιπα bits μπορούν να χρησιμοποιηθούν σε μία πιθανή επέκταση του ΕΚΥ, για την επικοινωνία του με άλλες περιφερειακές συσκευές. Για παράδειγμα, αν το bit k (k≠0) ισούται με 1, τότε η είσοδος ή η έξοδος θα πραγματοποιηθεί από ή προς την περιφερειακή συσκευή k.

1.6. Εντολές καταχωρητή δείκτη (Index register instructions)

Για τη διευκόλυνση της υλοποίησης προγραμμάτων που περιέχουν βρόχους, οι υπολογιστές διαθέτουν έναν αριθμό καταχωρητών, που ονομάζονται *καταχωρητές δείκτη (index registers)* και βρίσκονται στη μονάδα ελέγχου. Ο ΕΚΥ διαθέτει έναν καταχωρητή δείκτη B, μήκους 16 bits. Συνήθως, σε κάθε καταχωρητή δείκτη αντιστοιχεί ένα bit στην εντολή, που καθορίζει εάν η εντολή αναφέρεται στο συγκεκριμένο καταχωρητή δείκτη, ανάλογα αν το bit αυτό ισούται με 0 ή 1. Στον ΕΚΥ, το bit δείκτη, όπως ονομάζεται, είναι το bit 11. Επομένως, η μορφή της εντολής διαμορφώνεται όπως φαίνεται στο παρακάτω σχήμα.



Αν μία εντολή αναφοράς στη μνήμη χρησιμοποιεί τον καταχωρητή δείκτη, δηλαδή αν το bit δείκτη ισούται με 1, τότε η διεύθυνση Ν της θέσης της μνήμης, στην οποία αναφέρεται η εντολή αυτή, δεν είναι πλέον αυτή που είναι γραμμένη στο τμήμα διεύθυνσης της εντολής. Η διεύθυνση Ν προκύπτει από το άθροισμα του περιεχομένου του καταχωρητή Β και του περιεχομένου του τμήματος διεύθυνσης της εντολής.

Επίσης, στις εντολές άλματος, αν το bit δείκτη είναι 1, το άλμα δεν πραγματοποιείται στη διεύθυνση που δείχνει το τμήμα διεύθυνσης της εντολής, αλλά στη διεύθυνση που δίδεται από το άθροισμα του περιεχομένου του καταχωρητή Β και του περιεχομένου του τμήματος διεύθυνσης της εντολής.

Οι υπολογιστές διαθέτουν, επίσης, μια ομάδα εντολών, που ονομάζονται εντολές καταχωρητών δείκτη, μέσω των οποίων τροποποιούμε και χρησιμοποιούμε τους καταχωρητές δείκτη. Ο ΕΚΥ έχει 3 τέτοιες εντολές. Η πρώτη είναι η *εντολή ανάγνωσης του καταχωρητή δείκτη Β*, που συμβολίζεται ως

```
LDI         XXX
```

Αυτό σημαίνει:

```
Αντέγραψε το περιεχόμενο της διεύθυνσης XXX της μνήμης στον
καταχωρητή Β – LoaD Index
```

Η δεύτερη εντολή καταχωρητή δείκτη είναι η *εντολή αποθήκευσης του καταχωρητή δείκτη Β*, που συμβολίζεται ως

```
STI         XXX
```

Αυτό σημαίνει:

```
Αντέγραψε το περιεχόμενο του καταχωρητή Β στη διεύθυνση XXX της
μνήμης– STore Index
```

Η τρίτη εντολή καταχωρητή δείκτη είναι η *εντολή δείκτη άλματος*, που συμβολίζεται ως

```
INJ         XXX
```

Αυτό σημαίνει:

```
Αν το περιεχόμενο του Β είναι διάφορο του 0, τότε (α) το περιεχόμενο
του Β μειώνεται κατά 1 και (β) η επόμενη εντολή που θα εκτελεστεί
βρίσκεται στη διεύθυνση XXX. Διαφορετικά, δεν συμβαίνει καμία αλλαγή
στη ροή εκτέλεσης του προγράμματος, ούτε στο περιεχόμενο του Β. –
INdex Jump
```

Στο σημείο αυτό επισημαίνουμε ότι οι εντολές καταχωρητή δείκτη, οι εντολές ολίσθησης και οι εντολές εισόδου-εξόδου *δεν χρησιμοποιούν το bit δείκτη*.

1.7. Ανακεφαλαίωση

Ο παρακάτω πίνακας ανακεφαλαιώνει όλες τις εντολές του ΕΚΥ, που περιγράψαμε στις προηγούμενες παραγράφους:

Γλώσσα ASSEMBLY του ΕΚΥ

Κατηγορία	Κώδικας (δεκαεξαδικό)	Μνημονικό όνομα εντολής	Σημασία
	0	HLT	STOP
Αναφοράς στη μνήμη	1	LDA	(A) := (N), LoaD A
	2	STA	(N) := (A), Store A
	3	ADA	(A) := (A) + (N), Add to A
	4	SBA	(A) := (A) - (N), SuBtract from A
	5	MLA	(A) := (A) * (N), MuLtiply with A
	6	DVA	(A) := (A) / (N), DiVide with A
Άλματος	7	JMP	(PC):=N, JuMP
	8	JAN	(PC):=N, αν (A)<0, Jump if A Negative
Εισόδου-Εξόδου	9	OUT	OUTput
	A	INP	INPut
Ολίσθησης	B	SAL	Ολίσθηση του Α προς τα αριστερά Ν θέσεις, Shift A Left
	C	SAR	Ολίσθηση του Α προς τα δεξιά Ν θέσεις, Shift A Right
Καταχωρητή δείκτη	D	LDI	(B) := (N), LoaD Index
	E	STI	(N) := (B), STore Index
	F	INJ	Αν (B)≠0, τότε (B):=(B)-1 και (PC):=N, INdex Jump

Στον πίνακα αυτό ακολουθούμε τους εξής συμβολισμούς:

:= → ανάθεση τιμής

(X**)** → περιεχόμενο του X (καταχωρητή ή θέσης μνήμης)

PC → Μετρητής προγράμματος

A → Καταχωρητής γενικού σκοπού (συσσωρευτής)

B → Καταχωρητής δείκτη

N → ***(α)*** Για τις εντολές ολίσθησης είναι ο αριθμός που περιέχεται στο τμήμα διεύθυνσης της εντολής

(β) Για τις εντολές καταχωρητή δείκτη είναι η διεύθυνση που περιέχεται στο τμήμα διεύθυνσης της εντολής

(γ) Για τις εντολές αναφοράς στη μνήμη και τις εντολές άλματος,

(i) Αν bit 11=0, είναι η διεύθυνση που περιέχεται στο τμήμα διεύθυνσης της εντολής

(ii) Αν bit 11=1, είναι η διεύθυνση που δίδεται από το άθροισμα του περιεχομένου του τμήματος διεύθυνσης της εντολής και του περιεχομένου του καταχωρητή δείκτη

2. Προγραμματισμός Η/Υ

Συνήθως ο προγραμματιστής γράφει το πρόγραμμά του σε κάποια γλώσσα υψηλού επιπέδου (C, C++, Java, Pacal, κλπ). Στην περίπτωση αυτή, ο ***compiler*** μετατρέπει το πρόγραμμα σε γλώσσα μηχανής. Ένα πρόγραμμα σε γλώσσα μηχανής αποτελείται από ένα σύνολο λέξεων υπολογιστή, όπως στο σχήμα της Σελ.16, τοποθετημένες σε συγκεκριμένη διεύθυνση της μνήμης του υπολογιστή. Ένα πρόγραμμα σε γλώσσα μηχανής βρίσκεται ακριβώς στη μορφή που είναι κατανοητή για τον υπολογιστή, οπότε είναι έτοιμο για εκτέλεση. Πολλές φορές, όμως, είναι απαραίτητο ο προγραμματιστής να έχει μεγαλύτερο έλεγχο στην τελική μορφή του προγράμματος, από αυτόν που του παρέχει ο compiler. Για παράδειγμα, αν στόχος του είναι η επίτευξή ιδιαίτερα υψηλής απόδοσης, μπορεί να χρειαστεί να ελέγχει ακριβώς ποια είναι η ακολουθία εντολών μηχανής ή σε ποια θέση αποθηκεύονται τα δεδομένα του προγράμματος. Στην περίπτωση αυτή, είναι απαραίτητο να γραφεί το πρόγραμμα σε γλώσσα assembly, η οποία αποτελείται από εντολές όμοιες με αυτές που εξετάσαμε στις παραγράφους 1.2 έως 1.7. (Ή, τουλάχιστον, μπορεί να χρειαστεί ο προγραμματιστής να επέμβει στο πρόγραμμα assembly που παράγεται αυτόματα από τον compiler ως ενδιάμεσο στάδιο μεταξύ του προγράμματος σε γλώσσα υψηλού επιπέδου και του προγράμματος σε γλώσσα μηχανής.) Στη συνέχεια, ο ***assembler*** μετατρέπει το πρόγραμμα από γλώσσα assembly σε γλώσσα μηχανής. Ο assembler δίδεται από τον κατασκευαστή κάθε υπολογιστή. Η βασική διαφορά του από τον compiler έγκειται στο ότι, ***ο assembler μεταφράζει μία εντολή γλώσσας assembly σε μία ακριβώς εντολή γλώσσας μηχανής***, ενώ ο compiler μεταφράζει μία εντολή γλώσσας assembly σε πολλές εντολές γλώσσας μηχανής.

2.1. Περιεχόμενο πεδίου διεύθυνσης

Στο πεδίο διεύθυνσης των εντολών assembly μπορεί να περιέχονται:

(1) ένα συμβολικό όνομα. Τα συμβολικά ονόματα που χρησιμοποιούνται στο πεδίο διεύθυνσης των εντολών πρέπει απαραίτητα να ορίζονται ως labels μπροστά από κάποια εντολή ή ψευδοεντολή (βλ. παράγραφο 2.2).

(2) ένας αριθμός. Οι αριθμοί μπορεί να είναι αριθμητικές διευθύνσεις σε οκταδική μορφή, ή αριθμητικές σταθερές. Οι αριθμητικές σταθερές μπορεί να είναι ακέραιοι δεκαδικοί προσημασμένοι ή οκταδικοί αριθμοί. Στην τελευταία περίπτωση χρησιμοποιείται το σύμβολο `$` για να γίνεται διάκριση από τους δεκαδικούς αριθμούς.

(3) το σύμβολο `*`. Αυτό είναι ένα ειδικό σύμβολο, του οποίου η τιμή είναι η απόλυτη διεύθυνση της εντολής που το περιέχει στο πεδίο διεύθυνσής της. Δηλαδή, παριστάνει το περιεχόμενο του μετρητή προγράμματος κατά την ανάκληση της εντολής.

(4) συνδυασμό των παραπάνω, μαζί με αριθμητικούς τελεστές (`+`, `-`, `*`, `/`) για το σχηματισμό παραστάσεων διεύθυνσης. Π.χ. `TAG+3`, `*-2`. Αυτό έχει σαν αποτέλεσμα να υπολογίζεται η παράσταση σαν αριθμητική έκφραση, όπου κάθε συμβολικό όνομα αντικαθίσταται με την αντίστοιχη απόλυτη διεύθυνση.

2.2. Ψευδοεντολές

Η συμβολική γλώσσα assembly, εκτός των εντολών που είδαμε στις προηγούμενες παραγράφους, διαθέτει και τις ονομαζόμενες ψευδοεντολές (pseudo instructions), οι οποίες διευκολύνουν τον προγραμματισμό και ταυτόχρονα καθοδηγούν τον assembler κατά τη μετάφραση σε γλώσσα μηχανής. Οι ψευδοεντολές δεν εκτελούνται από τον υπολογιστή κατά την εκτέλεση του προγράμματος, αλλά χρησιμοποιούνται από τον assembler κατά το στάδιο της μετάφρασης. Τοποθετούνται ή στην αρχή του προγράμματος, πριν την πρώτη εκτελέσιμη εντολή, ή στο τέλος του προγράμματος. Στη συνέχεια, περιγράφονται οι ψευδοεντολές του ΕΚΥ.

2.2.1. Ψευδοεντολή CON

Η ψευδοεντολή `CON` χρησιμοποιείται για τη δήλωση σταθερών. Η μορφή της είναι:

```
label       CON      e1, e2, …
```

όπου `label` είναι το συμβολικό όνομα που αντιστοιχεί στη διεύθυνση της εντολής και `e1, e2, …` είναι αριθμητικές σταθερές (οκταδικές ή δεκαδικές). Κατά τη μετάφραση αντιστοιχίζεται μία διεύθυνση στη συμβολική διεύθυνση `label` και το περιεχόμενο αυτής της διεύθυνσης γίνεται ίσο με η σταθερά `e1`. Το περιεχόμενο της επόμενης διεύθυνσης της μνήμης γίνεται ίσο με `e2`, κοκ.

2.2.2. Ψευδοεντολή RES

Η ψευδοεντολή `RES` χρησιμοποιείται για τη δήλωση μεταβλητών και πινάκων. Η μορφή της είναι:

```
label       RES     n
```

όπου `label` είναι το συμβολικό όνομα της διεύθυνσης της πρώτης θέσης μιας περιοχής `n` διαδοχικών θέσεων της μνήμης, η οποία θα χρησιμοποιηθεί από το πρόγραμμα για βοηθητικούς λόγους, π.χ. για αποθήκευση ενδιάμεσων και τελικών αποτελεσμάτων.

2.3. Υποπρογράμματα

Όταν σε διαφορετικές θέσεις ενός προγράμματος επαναλαμβάνεται η ίδια εργασία, μπορούμε να γράψουμε ένα άλλο πρόγραμμα που να εκτελεί την εργασία αυτή και να το καλούμε όποτε χρειάζεται. Το δεύτερο αυτό πρόγραμμα, όπως και στις γλώσσες προγραμματισμού υψηλού επιπέδου, ονομάζεται υποπρόγραμμα και μπορεί να είναι μέρος του κυρίως προγράμματος. Η επικοινωνία του κυρίως προγράμματος με το υποπρόγραμμα γίνεται ως εξής:

```
Κυρίως πρόγραμμα:
            … …
ADR1        LDA     X1
            STA     TAG
            JMP     TAG+1
            … …
ADR2        LDA     X2
            STA     TAG
            JMP     TAG+1
            … …
X1          JMP     ADR1+3
X2          JMP     ADR2+3

Υποπρόγραμμα:
TAG         JMP     *
            … …
            Εντολές υποπρογράμματος
            … …
            JMP     TAG
```

Στο παραπάνω τμήμα κώδικα υποθέτουμε ότι στις διευθύνσεις `ADR1` και `ADR2` του κυρίως προγράμματος, θέλουμε να χρησιμοποιήσουμε το υποπρόγραμμα που αρχίζει στη

διεύθυνση TAG. Η κλήση του υποπρογράμματος γίνεται με τρεις εντολές. Η πρώτη από αυτές φορτώνει στο συσσωρευτή (την πρώτη φορά που εκτελείται το υποπρόγραμμα) το περιεχόμενο της διεύθυνσης X1 και η δεύτερη το αντιγράφει στη διεύθυνση TAG του υποπρογράμματος. Δηλαδή, η διεύθυνση TAG αλλάζει περιεχόμενο και τώρα περιέχει μία εντολή άλματος στη διεύθυνση στην οποία πρέπει να επιστρέψει ο έλεγχος του προγράμματος μετά την εκτέλεση του προγράμματος. Με την τρίτη εντολή κλήσεως, JMP TAG+1, καλείται και εκτελείται το υποπρόγραμμα. Σημειώνουμε ότι η εκτέλεση δεν ξεκινά από τη διεύθυνση TAG, που περιέχει τη διεύθυνση επιστροφής, αλλά από την αμέσως επόμενη εντολή. Η τελευταία εντολή του υποπρογράμματος είναι μία εντολή άλματος στη διεύθυνση TAG, στην οποία έχει αποθηκευτεί η διεύθυνση επιστροφής. Έτσι, επανερχόμαστε στο κυρίως πρόγραμμα, στο σημείο από το οποίο έγινε η κλήση του υποπρογράμματος.

2.4. Μακροεντολές

Ένας άλλος τρόπος να αποφύγουμε την επανάληψη ενός τμήματος του προγράμματος που εκτελεί την ίδια εργασία είναι η χρησιμοποίηση των μακροεντολών. Στη συνέχεια, θα περιγράψουμε τις μακροεντολές με ένα παράδειγμα:

Έστω ότι σε πολλά σημεία του κυρίως προγράμματος απαιτείται η εύρεση του αθροίσματος των περιεχομένων δύο θέσεων μνήμης και η αποθήκευση του αποτελέσματος στην πρώτη από αυτές:

```
            … …
            LDA     X1
            ADA     X2
            STA     X1
            … …
            LDA     Y1
            ADA     Y2
            STA     Y1
            … …
```

Αντί να επαναλαμβάνουμε το ίδιο κομμάτι κώδικα, με διαφορετικές συμβολικές διευθύνσεις κάθε φορά, μπορούμε να χρησιμοποιήσουμε μία μακροεντολή, ως εξής:

```
            … …
            EX      X1, X2
            … …
            EX      Y1, Y2
            … …
EX          MACRO   A, B
            LDA     A
```

```
ADA     B
STA     A
EMC
```

Κατά τη μετάφραση του προγράμματος αυτού, ο assembler αντικαθιστά τη μακροεντολή `EX` με τις εντολές `LDA A, ADA B, STA A`. Σαν διευθύνσεις `A` και `B` βάζει τις διευθύνσεις που υπάρχουν στο πεδίο διεύθυνσης της μακροεντολής, δηλαδή τις `X1` και `X2` κατά την πρώτη κλήση της, τις `Y1` και `Y2` κατά τη δεύτερη. Στο τέλος της περιγραφής της μακροεντολής υπάρχει πάντα η ψευδοεντολή `EMC`, που οριοθετεί το τέλος της. Σε κάποιους υπολογιστές έχουν ορισθεί από πριν κάποιες απλές μακροεντολές, οι οποίες μπορούν να χρησιμοποιούνται από τον προγραμματιστή χωρίς να χρειάζεται να ορίζονται κάθε φορά.

Η βασική διαφορά μεταξύ υποπρογραμμάτων και μακροεντολών έγκειται στο εξής: Στα υποπρογράμματα έχουμε ενέργεια στο στάδιο της εκτέλεσης του προγράμματος (δηλ. από το κυρίως πρόγραμμα γίνεται άλμα στο υποπρόγραμμα, εκτέλεση αυτού και άλμα πάλι πίσω στο κυρίως πρόγραμμα. Στις μακροεντολές έχουμε ενέργεια στο στάδιο της μετάφρασης, δηλαδή ο assembler αντικαθιστά τις μακροεντολές με μια ομάδα εντολών που αναφέρονται κάθε φορά στις κατάλληλες διευθύνσεις.

3. Ασκήσεις

3.1. Σε 100 θέσεις μνήμης που αρχίζουν από την διεύθυνση LIST, βρίσκονται 100 αριθμοί. Να γραφεί πρόγραμμα σε γλώσσα assembly που να θέτει στη θέση DIV3 το άθροισμα αυτών που διαιρούνται δια του 3 και στη θέση MOROS το μέσο όρο των υπολοίπων.

Το παρακάτω πρόγραμμα σαρώνει τους 100 αριθμούς που βρίσκονται αποθηκευμένοι στις θέσεις LIST έως LIST+99, από κάτω προς τα πάνω. Για κάθε έναν από αυτούς ελέγχει αν διαιρείται με το 3. Αν ναι, τότε τον προσθέτει στο άθροισμα που βρίσκεται ήδη στη θέση DIV3. Αν όχι, τον προσθέτει στο άθροισμα που βρίσκεται ήδη στη θέση MOROS.

Ταυτόχρονα, υπάρχει αποθηκευμένος στη θέση CNT ένας μετρητής που δείχνει πόσοι αριθμοί έχουν βρεθεί μέχρι στιγμής μη διαιρετοί με το 3. Κάθε φορά που βρίσκεται άλλος ένας τέτοιος αριθμός, αμέσως μετά την ενημέρωση του αθροίσματος MOROS, ενημερώνεται και ο μετρητής CNT, αυξάνοντάς τον κατά 1.

‘Ολες οι θέσεις μνήμης που χρησιμοποιούνται για υπολογισμούς: DIV3, MOROS, CNT, έχουν αρχικοποιηθεί στην τιμή 0. Επίσης, σημειώστε ότι στη συγκεκριμένη άσκηση ο μέσος όρος έχει υπολογιστεί με αποκοπή των δεκαδικών ψηφίων και όχι στρογγυλοποίηση.

```
ONE         CON     1
THREE       CON     3
HUNDRED     CON     99
LIST        RES     100
DIV3        RES     1
MOROS       RES     1
CNT         RES     1
HELP        RES     1

            SAL     16          ; Μηδενισμός του καταχωρητή A
            STA     DIV3        ; και στη συνέχεια και των
            STA     MOROS       ; θέσων που χρησιμοποιύνται
            STA     CNT         ; για υπολογισμούς

            LDI     HUNDRED
```

```
LOOP    LDA,I   LIST
        STA     HELP
        DVA     THREE     ; Έλεγχος για διαιρετότητα με
        MLA     THREE     ; το 3 βασιζόμενοι στην
        SBA     HELP      ; ιδιότητα: (x/3)*3<x ανν x
                          ; δεν είναι πολλαπλάσιο του 3.
        JAN     N_DIV

        LDA     DIV3      ; Το πρόγραμμα φτάνει εδώ μόνο
        ADA     HELP      ; σε περίπτωση που ο αριθμός
        STA     DIV3      ; διαιρείται με 3.
        INJ     LOOP      ; Ενημερώνεται επομένως το
        JMP     END       ; αντίστοιχο άθροισμα

N_DIV   LDA     MOROS     ; Το πρόγραμμα φτάνει εδώ μόνο
        ADA     HELP      ; σε περίπτωση που ο αριθμός
        STA     MOROS     ; δε διαιρείται με 3.
        LDA     CNT
        ADA     ONE       ; Ενημερώνεται το αντίστοιχο
        STA     CNT       ; άθροισμα και ο μετρητής των
        INJ     LOOP      ; μη διαιρετών με 3 αριθμών.

END     LDA     MOROS     ; Πριν τον τερματισμό του
        DVA     CNT       ; προγράμματος, το άθροισμα
        STA     MOROS     ; των μη διαιρετών με 3
                          ; αριθμών διαιρείται με το
                          ; πλήθος τους, ώστε να
                          ; προκύψει ο μέσος όρος αυτών
        HLT
```

3.2. Στη θέση μνήμης ενός Η/Υ START βρίσκεται ο αριθμός Ν. Να γραφεί πρόγραμμα ASSEMBLY που να υπολογίζει το Ν!

Υπόδειξη: N! = N (N-1)!

1η λύση:

```
INDEX       RES     1           ; Ο τρέχων αριθμός (παράγοντας) που
                                ; θα πολλαπλασιαστεί. Αυξάνεται σε
                                ; κάθε βρόχο κατά 1 μέχρι να γίνει
                                ; INDEX=N
FACTOR      RES     1           ; Το τρέχων παραγοντικό, δηλαδή
                                ; INDEX!
ONE         CON     1
START       CON     N+1

            LDA     ONE         ; Αρχικοποίηση των INDEX &
            STA     FACTOR      ; FACTOR στην τιμή 1
            STA     INDEX

LOOP        LDA     FACTOR
            MLA     INDEX
            STA     FACTOR      ; FACTOR *= INDEX
            LDA     INDEX
            ADA     ONE
            STA     INDEX       ; INDEX ++
            SBA     START
            JAN     LOOP        ; Αν INDEX > N, τέλος
END         HLT
```

2η λύση:

Στο παρακάτω πρόγραμμα χρησιμοποιούμε index register. Προκειμένου να αποφύγουμε το μηδενισμό του τελικού αποτελέσματος όταν ο index register πάρει την τιμή 0, πολλαπλασιάζουμε κάθε φορά την τιμή του τρέχοντος αποτελέσματος με την τιμή του index register προσαυξημένη κατά ένα. Για το λόγο αυτό όμως προσέχουμε ώστε η αρχική τιμή του index register να μην είναι N, αλλά N-1.

```
INDEX       RES     1           ; Βοηθητική θέση
                                ; μνήμης για την αποθήκευση
                                ; του index register.
FACTOR      RES     1           ; Εδώ αποθηκεύεται το αποτέλεσμα
```

```
ONE         CON     1
START       CON     N-1

            LDA     ONE
            STA     FACTOR
            LDI     START

LOOP        STI     INDEX
            LDA     INDEX       ; Φόρτωση και αύξηση
            ADA     ONE         ; της τιμής του index
            MLA     FACTOR      ; register κατά 1 πριν
            STA     FACTOR      ; πολλαπλασιαστεί στο
            INJ     LOOP        ; μέχρι στιγμής αποτέλεσμα

            HLT
```

3.3. Στις θέσεις μνήμης N έως και N+K βρίσκονται K+1 ακέραιοι αριθμοί. Να τοποθετηθούν στις θέσεις MIN και MAX ο μικρότερος και ο μεγαλύτερος από αυτούς αντίστοιχα.

Το παρακάτω πρόγραμμα αρχικοποιεί τις θέσεις MIN και MAX με την τιμή του τελευταίου αποθηκευμένου αριθμού στη θέση N+K. Στη συνέχεια σαρώνει τις K υπόλοιπες θέσεις μνήμης από το τέλος προς την αρχή. Συγκρίνει κάθε φορά τον υπό εξέταση αριθμό με αυτούς που βρίσκονται ήδη στις θέσεις MIN και MAX και, αν χρειάζεται, ενημερώνει τις θέσεις MIN και MAX.

```
MIN         RES     1
MAX         RES     1
N           RES     K+1
IND         CON     K-1

            LDA     N+K         ; Αρχικοποίηση θέσεων MIN
            STA     MIN         ; και MAX
            STA     MAX
```

```
            LDI     IND
LOOP        LDA,I   N           ; Σύγκριση περιεχομένου
            SBA     MAX         ; (N+IND) με MAX
            JAN     CONT

UPDATE_MAX  LDA,I   N
            STA     MAX         ; Ενημέρωση MAX
            INJ     LOOP        ; Σε περίπτωση που
            HLT                 ; χρειάστηκε ενημέρωση του
                                ; MAX αποκλείεται να
                                ; χρειαστεί ενημέρωση και
                                ; του MIN με τον ίδιο
                                ; αριθμό. Άρα αποφεύγεται
                                ; και η σύγκριση.

CONT        LDA,I   N           ; Σύγκριση περιεχομένου
            SBA     MIN         ; (N+IND) με MIN
            JAN     UPDATE_MIN
            INJ     LOOP
            HLT

UPDATE_MIN  LDA,I   N
            STA     MIN         ; Ενημέρωση MIN
            INJ     LOOP
            HLT
```

3.4. Σε μια περιοχή της μνήμης που ξεκινά από τη συμβολική διεύθυνση BEGIN έχουν καταχωρηθεί N ακέραιοι αριθμοί. Να γραφεί πρόγραμμα που να τους ταξινομεί κατά σειρά φθίνοντος μεγέθους.

Χρησιμοποιείται η μέθοδος ταξινόμησης Bubblesort.

```
TEMP        RES     1
```

```
FLAG        RES     1           ; Θα χρησιμοποιηθεί για έλεγχο
                                ; αν κατά την τελευταία σάρωση
                                ; υπήρξε ανάγκη εναλλαγής δύο
                                ; γειτονικών θέσεων
POS         CON     1
NEG         CON     -1
START       CON     N-2
BEGIN       RES     N

LOOP_EXT    LDI     START       ; Αρχίζει μια σάρωση από κάτω
            LDA     POS         ; προς τα πάνω.
            STA     FLAG        ;

LOOP        LDA,I   BEGIN       ; Σύγκριση δύο γειτονικών
            SBA,I   BEGIN+1     ; θέσεων
            JAN     SWAP        ; και εναλλαγή αν χρειάζεται
TAG         INJ     LOOP

            LDA     FLAG
            JAN     LOOP_EXT
            HLT

SWAP        LDA,I   BEGIN
            STA     TEMP
            LDA,I   BEGIN+1
            STA     BEGIN
            LDA     TEMP
            STA,I   BEGIN+1
            LDA     NEG
            STA     FLAG
            JMP     TAG
```

Επεξηγηματικά, δίδεται και το αντίστοιχο πρόγραμμα σε ψευδοκώδικα:

```
REPEAT
            FLAG:=1
            FOR I=N-2 DOWNTO 0
            {
                    IF (A[I]-A[I+1]<0)
                    {
                              SWAP(A[I],A[I+1])
                              FLAG:=-1
                    }
            }
UNTIL FLAG>=0
```

3.5. Να γραφεί πρόγραμμα σε ASSEMBLY που να τυπώνει το αλφάβητο.

Υπόδειξη: Ο κώδικας ASCII έχει τα γράμματα του αλφαβήτου σε διαδοχικές θέσεις.

```
AA          CON     $101        ; ASCII A σε οκταδικό
Z_NEXT      CON     $133        ; ASCII Z +1
ONE         CON     1
CUR         RES     1           ; Βοηθητική θέση μνήμης για την
                                ; αποθήκευση του τρέχοντος
                                ; ψηφίου

            LDA     AA
            STA     CUR

LOOP        LDA     CUR
            OUT,0               ; Τυπώνεται ο ASCII του
            ADA     ONE         ; τρέχοντος ψηφίου,
            STA     CUR         ; προσαυξάνεται κατά 1 και αν
            SBA     Z_NEXT      ; ξεπέρασε το Z, τότε το
            JAN     LOOP        ; πρόγραμμα τελειώνει
```

```
END         HLT
```

3.6. Σε μια περιοχή της μνήμης, που ξεκινά από τη συμβολική διεύθυνση AREA1, έχουν τοποθετηθεί N αριθμοί. Να γραφεί πρόγραμμα σε γλώσσα ASSEMBLY, μέσω του οποίου αυτοί να τοποθετούνται σε μια άλλη περιοχή της μνήμης, που ξεκινά από τη συμβολική διεύθυνση AREA2, έτσι ώστε πρώτοι να τοποθετούνται οι μη αρνητικοί αριθμοί και στη συνέχεια οι αρνητικοί αριθμοί με την ίδια σειρά, με την οποία εμφανίζονται στην αρχική περιοχή. Δεν υπάρχει δυνατότητα δημιουργίας άλλων νέων περιοχών στη μνήμη του υπολογιστή.

1η λύση:

Κατά την εκτέλεση του προγράμματος η περιοχή AREA1 σαρώνεται δύο φορές. Την πρώτη φορά αντιγράφονται οι μη αρνητικοί αριθμοί στην περιοχή AREA2 και τη δεύτερη φορά αντιγράφονται οι αρνητικοί. Σε κάθε περίπτωση διατηρούνται δύο δείκτες IND1, IND2. Ο IND1 δείχνει από ποιο σημείο της AREA1 διαβάζουμε και ο IND2 σε ποιο σημείο της AREA2 γράφουμε.

```
AREA1       RES     N
AREA2       RES     N
IND1        RES     1
IND2        RES     1
NUM         CON     N
ONE         CON     1

            SAL     16          ; Αρχικά μηδενίζονται οι
            STA     IND1        ; δείκτες των δύο περιοχών
            STA     IND2

LOOP_POS    LDI     IND1        ; Σαρώνεται για πρώτη φορά
            LDA,I   AREA1       ; η AREA1
            JAN     NEXT

            LDI     IND2        ; Αν βρούμε θετικό
            STA,I   AREA2       ; τον αντιγράφουμε
            LDA     IND2        ; και ενημερώνουμε τον IND2
            ADA     ONE         ; αυξάνοντάς τον κατά 1
```

```
            STA     IND2

NEXT        LDA     IND1        ; Αυξάνεται ο IND1 ώστε να
            ADA     ONE         ; προσπελαστεί το επόμενο
            STA     IND1        ; στοιχείο της AREA1
            SBA     NUM         ; Έλεγχος αν υπάρχει άλλο
            JAN     LOOP_POS    ; στοιχείο στην AREA1

            SAL     16          ; Μηδενισμός του IND1 ώστε να
            STA     IND1        ; αρχίσει η σάρωση της AREA1
                                ; από την αρχή. Ο IND2 δεν
                                ; αλλάζει, αφού οι αρνητικοί
                                ; θα αποθηκευθούν αμέσων μετά
                                ; τους θετικούς
LOOP_NEG    LDI     IND1        ; Σαρώνεται για δεύτερη φορά
            LDA,I   AREA1       ; η AREA1
            JAN     STORE

CONT_NEG    LDA     IND1        ; Αυξάνεται ο IND1 ώστε να
            ADA     ONE         ; προσπελαστεί το επόμενο
            STA     IND1        ; στοιχείο της AREA1
            SBA     NUM         ; Έλεγχος αν υπάρχει άλλο
            JAN     LOOP_NEG    ; στοιχείο στην AREA1

END         HLT                 ; Αν δεν υπάρχει, τέλος.

STORE       LDI     IND2        ; Ομοίως, αν βρούμε αρνητικό
            STA,I   AREA2       ; τον αντιγράφουμε
            LDA     IND2        ; και ενημερώνουμε τον IND2
            ADA     ONE         ; αυξάνοντάς τον κατά 1
            STA     IND2
            JMP     CONT_NEG
```

2η λύση:

Διαφορετικά, η περιοχή AREA1 μπορεί σαρώνεται μία μόνο φορά. Στην περίπτωση αυτή οι θετικοί αριθμοί αντιγράφονται στην αρχή της περιοχής AREA2, ενώ οι αρνητικοί στο τέλος με αντίστροφη σειρά από ότι είναι αποθηκευμένοι στην AREA1. Για το σκοπό αυτό τώρα χρησιμοποιούμε 3 δείκτες. Ο IND δείχνει από ποιο σημείο της AREA1 διαβάζουμε, ο IND_POS δείχνει σε ποιο σημείο της AREA2 γράφουμε σους θετικούς, ο IND_NEG δείχνει σε ποιο σημείο της AREA2 γράφουμε τους αρνητικούς. Οι IND και IND_POS αυξάνονται, ενώ ο IND_NEG μειώνεται κάθε φορά που γράφουμε ένα στοιχείο στην περιοχή του. Όταν τελειώσει αυτή η διαδικασία, οι αρνητικοί αριθμοί που είναι αποθηκευμένοι με αντίστροφη σειρά από την επιθυμητή, πρέπει να τοποθετηθούν στη σωστή σειρά. Για το λόγο αυτό εναλλάσσουμε τον πρώτο με τον τελευταίο, το δεύτερο με τον προτελευταίο, κ.ο.κ. (βλέπε ασκ. 8)

```
AREA1       RES     N
AREA2       RES     N
IND         RES     1
IND_POS     RES     1
IND_NEG     RES     1
TEMP        RES     1
NUM         CON     N
NUM_1       CON     N-1
ONE         CON     1

            SAL     16          ; Αρχικά μηδενίζονται οι
            STA     IND         ; δείκτες των δύο περιοχών
            STA     IND_POS
            LDA     NUM_1
            STA     IND_NEG

LOOP_POS    LDI     IND         ; Βρόχος: Σαρώνεται η AREA1
            LDA,I   AREA1       ;
            JAN     ST_NEG      ; 'Ελέγχουμε αν διαβάστηκε
                                ; θετικός ή αρνητικός
            LDI     IND_POS     ; Αν είναι θετικός,
            STA,I   AREA2       ; αποθηκεύεται εκεί που
            LDA     IND_POS     ; δείχνει ο IND_POS και ο
```

```
            ADA     ONE         ; IND_POS αυξάνεται κατά 1
            STA     IND_POS
            JMP     CONT

ST_NEG      LDI     IND_NEG     ; Αν είναι αρνητικός,
            STA,I   AREA2       ; αποθηκεύεται εκεί που
            LDA     IND_NEG     ; δείχνει ο IND_NEG και ο
            SBA     ONE         ; IND_NEG μειώνεται κατά 1
            STA     IND_NEG

CONT        LDA     IND         ; Προσαρμογή του δείκτη IND
            ADA     ONE         ; για την επόμενη ανάγνωση
            STA     IND
            SBA     NUM         ; Έλεγχος αν φθάσαμε στο
            JAN     LOOP        ; τέλος της AREA1. Αν όχι,
                                ; επαναλαμβάνεται ο βρόχος.
            LDA     IND_NEG     ; Αρχικοποίηση δεικτών IND_NEG
            ADA     ONE         ; και IND ώστε να αρχίσει το
            STA     IND_NEG     ; αναποδογύρισμα της περιοχής
            LDA     NUM_1       ; των αρνητικών αριθμών. Ο
            STA     IND         ; IND_NEG τοποθετείται στην
                                ; αρχή της περιοχής και ο IND
                                ; στο τέλος της.
LOOP_SWP    LDI     IND         ; Βρόχος: Εναλλαγή θέσεων
            LDA,I   AREA2       ; μνήμης στις οποίες δείχνουν
            STA     TEMP        ; οι IND και IND_NEG
            LDI     IND_NEG
            LDA,I   AREA2
            LDI     IND
            STA,I   AREA2
            LDI     IND_NEG
            LDA     TEMP
            STA,I   AREA2
```

```
        LDA     IND         ; Τροποποίηση δεικτών:
        SBA     ONE         ; Αύξηση IND_NEG και
        STA     IND         ; μείωση IND
        LDA     IND_NEG
        SBA     ONE
        STA     IND_NEG
        SBA     IND         ; Έλεγχος αν διασταυρώθηκαν,
        JAN     LOOP_SWP    ; ώστε να τερματίσει.
        HLT
```

3.7. Σε μια περιοχή της μνήμης του υπολογιστή έχουν καταχωρηθεί 80 αριθμοί. Να γραφεί πρόγραμμα σε γλώσσα ASSEMBLY μέσω του οποίου τυπώνονται οι αριθμοί της περιοχής στο δεκαδικό σύστημα και στη μορφή $+\Delta_4\Delta_3\Delta_2\Delta_1\Delta_0$ (θετικός) ή $-\Delta_4\Delta_3\Delta_2\Delta_1\Delta_0$ (αρνητικός) όπου Δ_i, i=0, 1, 2, 3, 4 τα δεκαδικά ψηφία του αριθμού.

```
MASK10_4    CON     10.000
MASK        RES     1
INDEX       RES     1

MINUS_ONE   CON     -1
ONE         CON     1
TEN         CON     10
N           CON     80

ASCII_0     CON     $60         ; ASCII 0
PLUS        CON     $53         ; ASCII +
MINUS       CON     $55         ; ASCII -
ENTER       CON     $12         ; ASCII <ENTER>

DIGIT       RES     1
CURRENT     RES     1
MATRIX      RES     80
```

```
            SAL     16
            STA     INDEX       ; Αρχικοποίηση μετρητή

LOOP1       LDI     INDEX       ; Κύριος βρόχος: Σαρώνει τον
                                ; πίνακα MATRIX
            LDA,I   MATRIX      ; Φορτώνουμε τον τρέχοντα
            STA     CURRENT     ; αριθμό και τον αποθηκεύουμε.
            JAN     NEG         ; Ελέγχουμε το πρόσημο

            LDA     PLUS        ; Αν είναι θετικός,
            OUT,0               ; τυπώνουμε '+'
            JMP     CONT

NEG         LDA     MINUS       ; Αν είναι αρνητικός,
            OUT,0               ; τυπώνουμε '-'
            LDA     CURRENT     ; και αποθηκεύουμε
            MLA     MINUS_ONE   ; το μέτρο του.
            STA     CURRENT

CONT        LDA     MASK10_4
            STA     MASK        ; Εδώ αποθηκεύεται ο αριθμός
                                ; με τον οποίο πρέπει να
                                ; διαιρέσουμε για να βρούμε το
                                ; επόμενο ψηφίο
LOOP2       LDA     CURRENT     ; Δευτερεύων βρόχος:
            DVA     MASK        ; Διαιρούμε τον αριθμό με την
            STA     DIGIT       ; τρέχουσα μάσκα, για να
            ADA     ASCII_0     ; βρούμε το επόμενο ψηφίο που
            OUT,0               ; θα τυπωθεί, και τυπώνουμε.
            LDA     DIGIT
            MLA     MASK
            MLA     MINUS_ONE   ; Βρίσκουμε το υπόλοιπο και το
            ADA     CURRENT     ; αποθηκεύουμε για την επόμενη
```

```
            STA     CURRENT   ; διαίρεση
            LDA     MASK      ; Βρίσκουμε το διαιρέτη που
            DVA     TEN       ; χρειαζόμαστε για να
            STA     MASK      ; απομονώσουμε το επόμενο
                              ; ψηφίο.
            SBA     ONE       ; Έλεγχος αν υπάρχει άλλο
            JAN     RESET     ; ψηφίο να τυπωθεί.
            JMP     LOOP2

RESET       LDA     ENTER     ; Αλλάζουμε γραμμή
            OUT,0
            LDA     INDEX     ; Αυξάνουμε το δείκτη
            ADA     ONE
            STA     INDEX
            SBA     N         ; και ελέγχουμε αν υπάρχει
            JAN     LOOP1     ; επόμενο στοιχείο
                              ;αποθηκευμένο. Αν ναι, γίνεται
                              ;επανάληψη του LOOP1
            HLT
```

3.8. Σε μια περιοχή της μνήμης που ξεκινά από τη θέση START έχουμε καταχωρήσει N αριθμούς. Να γραφεί πρόγραμμα ώστε ο πρώτος αριθμός να γίνει τελευταίος, ο τελευταίος πρώτος, κ.ο.κ.

Στο παρακάτω πρόγραμμα χρησιμοποιούνται δύο δείκτες IND1, IND2. Ο IND1 αρχικά δείχνει στην πρώτη θέση της περιοχής, ο IND2 στην τελευταία. Σε κάθε επανάληψη ο IND1 αυξάνεται κατά 1, ενώ ο IND2 μειώνεται κατά 1. Επομένως, ανά πάσα στιγμή οι IND1, IND2 δείχνουν στις δύο θέσεις που πρέπει να εναλλαχθούν μεταξύ τους. Το πρόγραμμα σταματά όταν οι δύο δείκτες διασταυρωθούν.

```
ONE         CON     1
NUM         CON     N-1
IND1        RES     1
IND2        RES     1
TEMP        RES     1
```

```
START       RES     N

            LDA     NUM         ; Αρχικοποίηση δεικτών
            STA     IND2
            SAL     16
            STA     IND1

LOOP        LDI     IND2        ; Κύριος βρόχος: Εναλλαγή
            LDA,I   START       ; δύο θέσεων μνήμης
            STA     TEMP
            LDI     IND1
            LDA,I   START
            LDI     IND2
            STA,I   START
            LDI     IND1
            LDA     TEMP
            STA,I   START

            LDA     IND2        ; Τροποποίηση δεικτών:
            SBA     ONE         ; Αύξηση IND1 και μείωση IND2
            STA     IND2
            LDA     IND1
            SBA     ONE
            STA     IND1
            SBA     IND2        ; Έλεγχος αν διασταυρώθηκαν,
            JAN     LOOP        ; ώστε να τερματίσει.
            HLT
```

3.9. Να γαφεί πρόγραμμα σε γλώσσα ASSEMBLY που επεξεργάζεται τους αριθμούς που περιέχονται σε μια περιοχή της μνήμης που ξεκινάει από τη συμβολική διεύθυνση START και τελειώνει σε διεύθυνση με περιεχόμενο 33330<8>. Η επεξεργασία των αριθμών που πρέπει να γίνει είναι ο προσδιορισμός του μέσου όρου των τετραγώνων των αριθμών και η

διαπίστωση αν αυτός είναι διαιρέτης του 33330<8>, που δεν περιλαμβάνεται στους υπό επεξεργασία αριθμούς.

Στο παρακάτω πρόγραμμα διατηρείται ένας δείκτης INDEX, ο οποίος ανά πάσα στιγμή δείχνει από ποια θέση της μνήμης πρέπει να φορτώσουμε τον επόμενο αριθμό. Ταυτόχρονα, λειτουργεί σα μετρητής ώστε στο τέλος του προγράμματος να ξέρουμε πόσοι αριθμοί έχουν υποστεί επεξεργασία και συνεπώς να μπορούμε να υπολογίσουμε το ζητούμενο μέσο όρο. Στη θέση RESULT αποθηκεύεται η τιμή YES αν ο τελικός μέσος όρος είναι πολλαπλάσιο του $33330_{<8>}$, αλλιώς αποθηκεύεται η τιμή NO.

```
INDEX       RES     1
START       RES     N
CONST       CON     $33330
ONE         CON     1
AVERG       RES     1
RESULT      RES     1
YES         CON     1
NO          CON     0

            SAL     16          ; Αρχικοποίηση
            STA     AVERG       ; μετρητή και
            STA     INDEX       ; αθροίσματος

LOOP        LDI     INDEX       ; Βρόχος: Σαρώνει τη μνήμη
            LDA,I   START       ; Έλεγχος αν φτάσαμε στο τέλος,
            SBA     CONST       ; συγκρίνοντας με τη δοσμένη
            JAN     CONT        ; σταθερά CONST
            LDA     CONST
            SBA,I   START
            JAN     CONT

END         LDA     MOROS       ; Υπολογισμός μέσου όρου,
            DVA     INDEX       ; διαιρώντας με το πλήθος των
            STA     MOROS       ; αριθμών που επεξεργάστηκαν
            DVA     CONST       ; Έλεγχος αν είναι πολλαπλάσιο
            MLA     CONST       ; του CONST.
```

```
            SBA     MOROS
            JAN     NO_MULT
            LDA     YES
            STA     RESULT
            HLT

NO_MULT     LDA     NO
            STA     RESULT
            HLT

CONT        LDA,I   START       ; Υπολογισμός τετραγώνου και
            MLA,I   START
            ADA     AVERG       ; προσθήκη σε
            STA     AVERG       ; προηγούμενο άθροισμα

            LDA     INDEX       ; Αύξηση μετρητή ώστε
            ADA     ONE         ; να προσπελαστεί το
            STA     INDEX       ; επόμενο στοιχείο
            JMP     LOOP
```

3.10. Στις θέσεις N μέχρι και N+K της μνήμης βρίσκονται οι συντελεστές a_0 μέχρι και a_K του πολυωνύμου $f(w) = a_0 + a_1w + a_2w^2 + \ldots + a_Kw^K$. Στη θέση X βρίσκεται η ανεξάρτητη μεταβλητή. Να τοποθετηθεί στη θέση Y της μνήμης η τιμή y=f(x).

Για τη λύση του προβλήματος αυτού εκμεταλλευόμαστε την ισότητα:

$$f(w) = a_0 + a_1w + a_2w^2 + \ldots + a_Kw^K = (\ldots((\ a_Kw + a_{K-1}\ )w + a_{K-2}\ )w + \ldots + a_1\ )w + a_0$$

```
X           CON     x
Y           RES     1
NUM         CON     K
NUM_1       CON     K-1

            LDI     NUM
            LDA,I   N
```

```
            LDI     NUM_1
LOOP        MLA     X
            ADA,I   N
            INJ     LOOP
            STA     Y
            HLT
```

Επομένως, σε κάθε επανάληψη του βρόχου το μέχρι στιγμής αποτέλεσμα, το οποίο βρίσκεται στον καταχωρητή Α, πολλαπλασιάζεται με w και προστίθεται σε αυτό ο επόμενος συντελεστής.

Παράρτημα: Αναπαράσταση χαρακτήρων κατά ASCII

Τιμή ASCII (hex)	Χαρακτήρας
00	NULL
01	SOM (start of message)
02	EOA (end of address)
03	EOM (end of message)
04	EOT (end of transmission)
05	WRU ("Who are you?")
06	RU ("Are you …")
07	BELL (audible signal)
08	backspace
09	HT (horizontal tabulation - tab)
0A	LF (line feed)
0B	VT (vertical tabulation)
0C	FF (form feed)
0D	CR (carriage return)
0E	SO (shift out)
0F	SI (shift in)
10	DC_0 (device control reserved for data link escape)
11	DC_1 (device control)
12	DC_2 (device control)
13	DC_3 (device control)
14	DC_4 (device control – stop)
15	ERR (error)
16	SYNC (synchronous idle)
17	LEM (logical end of media)
18	S_0 (Separator)

Τιμή ASCII (hex)	Χαρακτήρας
19	S_1 (Separator)
1A	S_3 (Separator)
1B	S_3 (Separator)
1C	S_4 (Separator)
1D	S_5 (Separator)
1E	S_6 (Separator)
1F	S_7 (Separator)
20	space
21	!
22	"
23	#
24	$
25	%
26	&
27	' (apostrophe)
28	(
29	)
2A	*
2B	+
2C	, (comma)
2D	-
2E	.
2F	/
30	0
31	1
32	2
33	3
34	4
35	5
36	6
37	7
38	8
39	9
3A	:
3B	;

Τιμή ASCII (hex)	Χαρακτήρας
3C	<
3D	=
3E	>
3F	?
40	@
41	A
42	B
43	C
44	D
5	E
46	F
47	G
48	H
49	I
4A	J
4B	K
4C	L
4D	M
4E	N
4F	O
50	P
51	Q
52	R
53	S
54	T
55	U
56	V
57	W
58	X
59	Y
5A	Z
5B	[
5C	\
5D	]
5E	^

Τιμή ASCII (hex)	Χαρακτήρας
5F	_
60	`
61	a
62	b
63	c
64	d
65	e
66	f
67	g
68	h
69	i
6A	j
6B	k
6C	l
6D	m
6E	n
6F	o
70	p
71	q
72	r
73	s
74	t
75	u
76	v
77	w
78	x
79	y
7A	z
7B	{
7C	|
7D	}
7E	~
7F	DEL (delete)