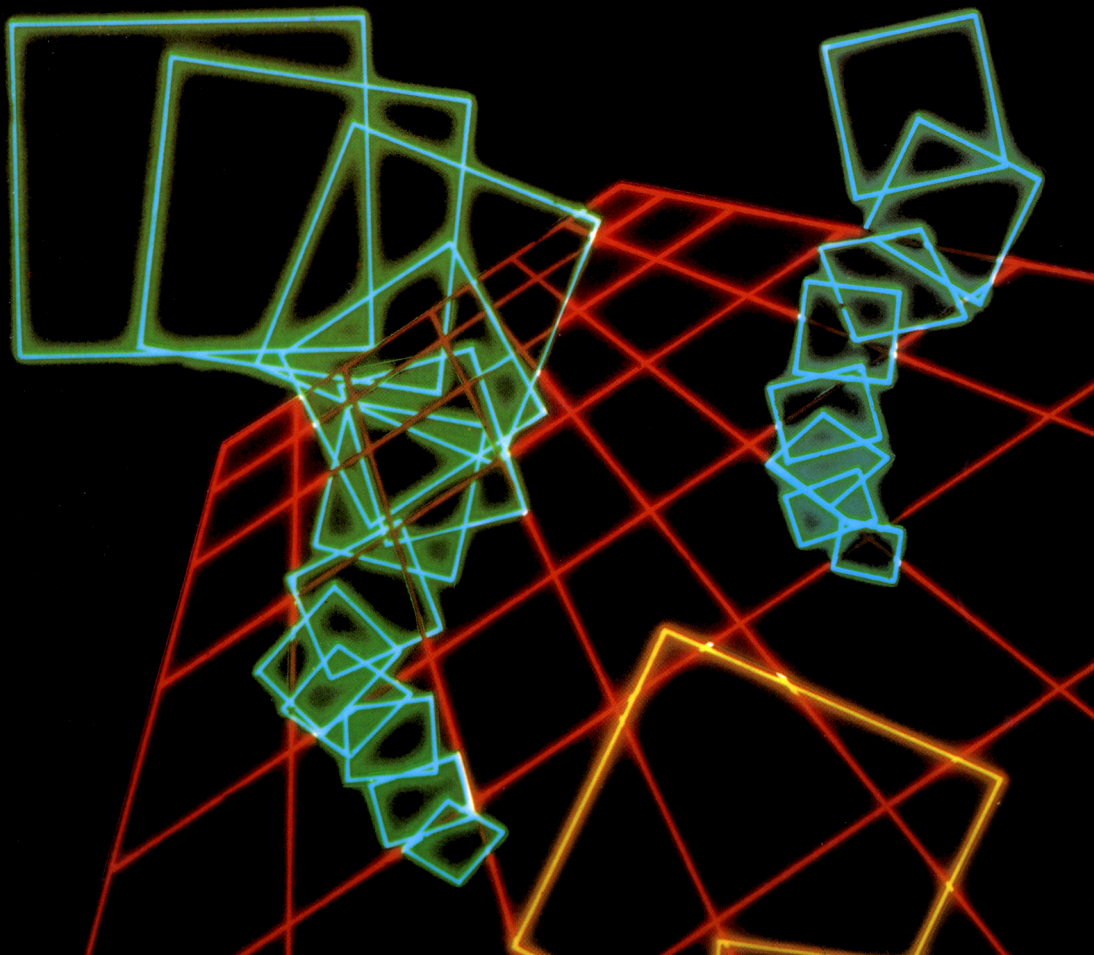


Γλώσσα μηχανής για αρχάριους στον **Amstrad**

Για τους τύπους CPC-464, 664, 6128

Steve Kramer



ΚΑΕΙΛΑΡΙΘΜΟΣ

Γλώσσα μηχανής για αρχάριους στον **Amstrad**

Για τους τύπους CPC-464, 664, 6128



ΕΚΔΟΣΕΙΣ ΚΛΕΙΔΑΡΙΘΜΟΣ
ΣΤΟΥΡΝΑΡΑ 27 Β
ΑΘΗΝΑ 106 82
ΤΗΛ: 3632044

Τίτλος πρωτοτύπου:

MACHINE CODE FOR BEGINNERS ON THE AMSTRAD

Αποκλειστικότητα για την ελληνική γλώσσα
Εκδόσεις ΚΛΕΙΔΑΡΙΘΜΟΣ
Στουρνάρα 27β
ΑΘΗΝΑ 106 82
ΤΗΛ.: 3632044

First published in 1984 in the United Kingdom by
Micro Press
Castle House, 27 London Road
Tunbridge Wells, Kent

Reprinted 1985 (twice)

© Steve Kramer 1984

Amstrad and CPC 464 are trademarks of
Amstrad Consumer Electronics PLC

ΜΕΤΑΦΡΑΣΗ: ΤΑΣΟΣ ΠΑΝΟΠΟΥΛΟΣ Μηχανολογος-Ηλεκτρολογος

ΠΕΡΙΕΧΟΜΕΝΑ

Κεφάλαιο 1	ΕΙΣΑΓΩΓΗ	5
Κεφάλαιο 2	ΤΙ ΕΙΝΑΙ Η ΓΛΩΣΣΑ ΜΗΧΑΝΗΣ ΚΑΙ ΓΙΑΤΙ ΤΗ ΧΡΗΣΙΜΟΠΟΙΟΥΜΕ	7
Κεφάλαιο 3	ΠΡΩΤΕΣ ΕΝΝΟΙΕΣ Hex Δυαδικό Assemblers και Συμβολικές εντολές	11
Κεφάλαιο 4	ΔΙΑΓΡΑΜΜΑΤΑ ΡΟΗΣ (FLOW CHARTS)	17
Κεφάλαιο 5	ΑΠΛΕΣ ΕΝΤΟΛΕΣ ΣΕ ΓΛΩΣΣΑ ΜΗΧΑΝΗΣ LD, CALL, RET, JP, JR	20
Κεφάλαιο 6	ΑΠΛΑ ΜΑΘΗΜΑΤΙΚΑ ADD, ADC, SUB, SBC, DEC, INC	44
Κεφάλαιο 7	FLAGS, ΚΑΤΑΣΤΑΣΕΙΣ ΚΑΙ ΛΗΨΕΙΣ ΑΠΟΦΑΣΕΩΝ CP, Z, NZ, C, NC, M, P, PE, PO, CCF, SCF, DJNZ	66
Κεφάλαιο 8	ΛΟΓΙΚΕΣ ΠΡΑΞΕΙΣ AND, OR, XOR, CPL, NEG	79
Κεφάλαιο 9	ΧΡΗΣΙΜΟΠΟΙΩΝΤΑΣ ΤΗ ΣΤΟΙΒΑ ΜΗΧΑΝΗΣ PUSH, POP και εντολές με SP	87
Κεφάλαιο 10	ΧΡΗΣΙΜΟΠΟΙΩΝΤΑΣ ΕΝΤΟΛΕΣ ΠΟΥ ΑΣΧΟΛΟΥΝΤΑΙ ΜΕ ΕΝΑ BIT SET, RES, BIT	96

Κεφάλαιο 11	ΠΕΡΙΣΤΡΟΦΕΣ ΚΑΙ ΜΕΤΑΤΟΠΗΣΕΙΣ, Πολλαπλασιασμός και Διαιρέση RL, RLA, RLC, RLCA, RLD, SLA, SLL RR, RRA, RRC, RRCA,RRD, SRA, SRL	102
Κεφάλαιο 12	ΑΥΤΟΜΑΤΕΣ ΜΕΤΑΚΙΝΗΣΕΙΣ ΚΑΙ ΕΡΕΥΝΕΣ LDD, LDDR, CPD, CPDR, LDI, LDIR, CPI, CPIR	121
Κεφάλαιο 13	ΕΠΙΚΟΙΝΩΝΩΝΤΑΣ ΜΕ ΤΟΝ ΕΞΩΤΕΡΙΚΟ ΚΟΣΜΟ IN και OUT	132
Κεφάλαιο 14	ΑΛΛΕΣ ΕΝΤΟΛΕΣ, Indexed Addressing ΜΕ ΤΟΥΣ ΚΑΤΑΧΩΡΗΤΕΣ IX ΚΑΙ IY	136
Κεφάλαιο 15	ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΕΣ ΙΔΕΕΣ ΚΑΙ ΧΡΗΣΙΜΟΠΟΙΗΣΗ ΤΟΥ FIRMWARE (Πώς να χρησιμοποιείτε μερικές από τις χρησιμότερες λειτουργίες του Λειτουργικού Συστήματος)	149
ΠΑΡΑΡΤΗΜΑΤΑ	A Η ΟΜΑΔΑ ΕΝΤΟΛΩΝ ΤΟΥ Z80 ΠΡΟΣΦΟ- ΡΑ ΤΗΣ ZILOG INC	153
	B HEX LOADER	162
	C ΜΕΤΑΤΡΟΠΗ HEX ΣΤΟ ΔΕΚΑΔΙΚΟ ΤΟΥ MSB	164
	D ΜΕΤΑΤΡΟΠΗ HEX ΣΤΟ ΔΕΚΑΔΙΚΟ ΤΟΥ LSB ΚΑΙ BINARY NIBBLES	165
	E ΜΕΤΑΤΡΟΠΕΣ ΣΥΜΠΛΗΡΩΜΑΤΩΝ ΤΟΥ 2	166
	F ΧΑΡΤΗΣ ΟΘΟΝΗΣ ΚΑΙ ΧΑΡΤΗΣ ΤΩΝ BITS ΓΙΑ PIXELS	168
	G ΧΡΗΣΙΜΕΣ ΚΑΗΣΕΙΣ ΔΙΕΥΘΥΝΣΕΩΝ	170

ΕΙΣΑΓΩΓΗ

Ο Amstrad είναι ίσως ο συναρπαστικότερος από τους νέους υπολογιστές που εμφανίστηκαν μετά τον Sinclair Spectrum.

Προσφέρει μια Basic με πολλά εξελιγμένα χαρακτηριστικά που παλαιότερα τα διέθεταν απείρως ακριβότερα μηχανήματα, καθώς και τη δυνατότητα επέκτασης με λογικό κόστος που δεν ξεπερνά το αντίστοιχο κόστος αποιουδήποτε άλλου οικιακού υπολογιστή.

Αυτό όμως που έχει ουσιαστική σημασία για τον προγραμματιστή, είναι η απόφαση της Amstrad να τεκμηριώσει και να θέσει σε κυκλοφορία λεπτομέρειες του λειτουργικού της συστήματος. Πρόκειται για μια χωρίς προηγούμενο αύξηση της τεκμηρίωσης οικιακού υπολογιστή από τον κατασκευαστή, και προσφέρει στο χρήστη μια πραγματική ευκαιρία να μάθει εύκολα προγραμματισμό σε γλώσσα μηχανής με σχεδόν άμεσα αποτελέσματα, χρησιμοποιώντας τις κλήσεις προς το λειτουργικό σύστημα.

Δεν υπάρχει πλέον ο φαύλος κύκλος που τον δημιούργησε η πεποίθηση ότι αν δεν γνωρίζετε γλώσσα μηχανής δεν μπορείτε να τη χρησιμοποιήσετε, και αν δεν μπορείτε να τη χρησιμοποιήσετε δεν μπορείτε να βρείτε τρόπο να την μάθετε με τον υπολογιστή σας, γιατί δεν γνωρίζετε πως να υποχρεώσετε τον υπολογιστή να αποκρίνεται.

Το βιβλίο αυτό προορίζεται για τον αρχάριο που θέλει να μάθει πως να χρησιμοποιεί τη Γλώσσα Μηχανής στον υπολογιστή Amstrad. Αρχίζει από τις βασικές έννοιες του προγραμματισμού σε γλώσσα μηχανής, εξηγεί τις εντολές που καταλαβαίνει η Κεντρική Μονάδα Επεξεργασίας (Central Processing Unit ή CPU για συντομία - το τσιπ από πυρίτιο που κάνει όλη τη δουλειά μέσα στον υπολογιστή) και πως να τις χρησιμοποιείτε, και σας γνωρίζει ορισμένες από τις ρουτίνες του λειτουργικού συστήματος σε διάφορα στάδια του βιβλίου.

Δύο τελειώς αρχάριοι στη γλώσσα μηχανής βοήθησαν για να γραφτεί αυτό το βιβλίο, και τα προβλήματα και οι ερωτήσεις τους αποτέλεσαν τη βάση για το σχεδιασμό του. Βοήθησαν, ακόμη, στο να εξασφαλιστεί ότι δεν παραλείφθηκε καμμία πληροφορία ή λεπτομέρεια η οποία είναι τόσο προφανής στους γνώστες ώστε να τους έχει γίνει σχεδόν δευτέρα φύση και θεμελιώδης όσον αφορά τη δυνατότητα εκτέλεσης κάποιας προγραμματιστικής λειτουργίας. Αυτή η παράλειψη είναι συχνά η αιτία της περιπλάνησης των αρχαρίων, όπως όταν λέμε σε κάποιον να πάει στη γωνία των οδών Καποδιστρίου και Καραϊσκάκη. Αν δεν γνωρίζει για ποιάν ακριβώς οδό Καποδιστρίου πρόκειται ή την περιοχή που βρίσκεται, δεν τον βοηθάμε και πολύ.

Στο βιβλίο δίνονται σύντομα listings που θα σας βοηθήσουν κατά την εισαγωγή προγραμμάτων σε γλώσσα μηχανής και κατά τον έλεγχο και την αλλαγή ή μετακίνηση των περιεχομένων ενός τμήματος της μνήμης. Σας συνιστούμε, πάντως να

αγοράσετε το πρόγραμμα *Assembler/Disassembler* για τον *Amstrad*. Αυτό θα σας επιτρέψει να γράφετε το πρόγραμμα με συμβολικές εντολές (*mnemonics*, ένα είδος στενογραφίας των ονομάτων των εντολών που καταλαβαίνει η *CPU*) και όχι με αριθμούς. Θα σας επιτρέψει επίσης να κάνετε διορθώσεις και μοιάζει πολύ με τη *BASIC* όσον αφορά τον τρόπο εισαγωγής των προγραμμάτων.

Θα μπορούσατε να καθίσετε και να διαβάσετε αυτό το βιβλίο από την αρχή μέχρι το τέλος χωρίς διακοπή. Επειδή, όμως, η γλώσσα μηχανής είναι ένα θέμα που θα μπορούσε εύκολα να σας μπερδέψει και είναι πολύ πιθανό να έρθετε σε επαφή με πάρα πολλές νέες έννοιες, σας συνιστούμε να καθίσετε μπροστά στον υπολογιστή σας για να πληκτρολογείτε και να εκτελείτε (ελπίζουμε όχι δι' απαγχονισμού) τα προγράμματα, μόλις εμφανίζονται στα σχετικά κεφάλαια. Μόνον αφού βεβαιωθείτε ότι καταλάβατε αυτό που συμβαίνει θα πρέπει να προχωράτε.

Για να μπορείτε να βλέπετε αμέσως τα αποτελέσματα των προγραμμάτων, γίνεται εκτεταμένη χρήση του λειτουργικού συστήματος. Αυτό επιτυγχάνεται χάρη στο πρόγραμμα *Amstrad Firmware Specification (Soft 158)* το οποίο, παρ' όλο που τώρα σας φαίνεται τελείως ακατανόητο, θα μπορούσε να αποτελέσει μια πολύτιμη προσθήκη στη βιβλιοθήκη σας, αφού τελειώσετε αυτό το βιβλίο και κατανοήσετε τις έννοιες που εξηγεί.

Η *Z80 CPU* είναι μια από τις πλατύτερα χρησιμοποιούμενες *CPUs* στην αγορά των οικιακών υπολογιστών και, μέχρι πρόσφατα, ήταν συχνά η κύρια *CPU* σε πολλούς *business* υπολογιστές. Δίνει τη δυνατότητα προσπέλασης στη μεγαλύτερη ποικιλία *software* του κόσμου μέσω του *CP/M* (συντομογραφία του *Control Program for Microcomputers*), και η *Amstrad* προσφέρει το *CP/M* μαζί με τα *disk drives*. Ο *Z80*, επίσης, αρχίζει να εμφανίζεται σαν δεύτερος επεξεργαστής στους *business* υπολογιστές, ενώ διατίθεται σαν πρόσθετος στον *BBC model B*, στον *Commodore 64* και στον *Apple* και τους ομοίους του. Επομένως, αυτά που θα μάθετε σε τούτο το βιβλίο είναι πιθανό να τα χρησιμοποιήσετε, αν σας δοθεί η ευκαιρία για να προγραμματίσετε και άλλους υπολογιστές σε γλώσσα μηχανής.

ΤΙ ΕΙΝΑΙ Η ΓΛΩΣΣΑ ΜΗΧΑΝΗΣ ΚΑΙ ΓΙΑΤΙ ΤΗ ΧΡΗΣΙΜΟΠΟΙΟΥΜΕ

Η CPU του υπολογιστή σας είναι στην ουσία ένα πολύ ηλίθιο δημιούργημα. Τρέχει πολύ καλά όλα τα προγράμματα σας σε BASIC και κάνει εξαιρετικά τη δουλειά της αλλά, παρ' όλα αυτά, εξακολουθεί να είναι ηλίθια. Αυτό που την κάνει να φαίνεται τόσο έξυπνη είναι το *firmware*, τα προγράμματα που τρέχουν σε όλη την διάρκεια λειτουργίας του υπολογιστή. Ο Amstrad, στην ατροποποίητη έκδοσή του, τρέχει ένα Λειτουργικό Σύστημα (Operating System) και το μεταφραστικό πρόγραμμα της BASIC (BASIC interpreter).

Το λειτουργικό σύστημα είναι επιφορτισμένο με τέτοιου είδους καθήκοντα, όπως είναι τα να ελέγχει αν πατήθηκε κάποιο πλήκτρο, το «φόρτωμα» από κασέτα ή η τοποθέτηση ενός χαρακτήρα στην οθόνη. Θα μπορούσατε να το θεωρήσετε σαν υπεύθυνο για όλες τις επικοινωνίες, και αν δεν ήταν παρόν δεν θα είχατε κανέναν άλλο τρόπο να μάθετε αν ο υπολογιστής σας είναι νεκρός ή ζωντανός, γιατί δεν θα ήσασταν σε θέση να του δώσετε καμιά πληροφορία και εκείνος δεν θα μπορούσε να σας πει τίποτα.

Το μεταφραστικό πρόγραμμα της BASIC κάνει αυτό που λέει και το όνομά του, δηλαδή μεταφράζει τη BASIC στη γλώσσα που καταλαβαίνει η CPU. Φανταστείτε για μια στιγμή ότι κάποιος σας λέει να πάτε στη σελίδα 35. Χωρίς κανένα πρόβλημα, πηγαίνετε στη σελίδα 35. Τι θα κάνατε, όμως, αν σας έλεγαν 𐀀; Τώρα αντιμετωπίζετε κάποιο πρόβλημα, γιατί όχι μόνο δεν ξέρετε τι να κάνετε, αλλά ίσως και να μην αναγνωρίζετε καν τον τύπο της εντολής.

Είναι δηλαδή σαν να βάζετε τον εαυτό σας στη θέση της CPU και να του δίνετε να εκτελέσει μια εντολή σε BASIC. Η CPU δεν γνωρίζει καθόλου BASIC, και το πρόβλημα δεν σταματάει εδώ. Προηγουμένως, ο Κινέζος χρησιμοποίησε ένα σύμβολο για να πει κάτι που όταν μεταφερθεί στα Ελληνικά θα χρειαστεί αρκετά σύμβολα: Τσανγκ. Και πάλι, όμως, δεν καταλάβατε και πολλά πράγματα, παρ' όλο που μπορείτε τώρα να το διαβάσετε. Η μετάφρασή του είναι: «σπέρνω χωρίς προηγουμένως να οργώσω το έδαφος». Η δυσκολία τώρα είναι όμοια με εκείνη που θα αντιμετώπιζε η CPU αν της ζητούσαμε να ασχοληθεί άμεσα με τη BASIC. Μια εντολή σε BASIC αντιπροσωπεύει συχνά ένα πλήθος εντολών σε γλώσσα μηχανής και το χειρότερο είναι πως οι χαρακτήρες που χρησιμοποιούνται από τη BASIC δεν μπορούν να γίνουν κατανοητοί από τη CPU, η οποία καταλαβαίνει μόνο δύο καταστάσεις, "on" και "off".

Ευτυχώς, τα "on" και "off" είναι συγκεντρωμένα σε ομάδες των οκτώ, πράγμα που μας δίνει 256 διαφορετικούς συνδυασμούς. Αυτοί είναι οι συνδυασμοί που χρησιμοποιούνται στη γλώσσα μηχανής. Θα μπορούσατε να τους παρουσιάσετε με τους κινέζικους χαρακτήρες, που ένα τους είδατε προηγουμένως.

Τα προβλήματα, εν τούτοις, δεν τελειώνουν εδώ. Εφ' όσον ένας χαρακτήρας αντιστοιχεί σε μια πλήρη λέξη και υπάρχουν μόνο 256 δυνατοί συνδυασμοί, η CPU φαίνεται να περιορίζεται σε ένα λεξιλόγιο μόλις 256 λέξεων. Αυτό είναι περίπου σωστό, αλλά, όπως και στα Ελληνικά, ορισμένες λέξεις δημιουργούνται από τη συνένωση δύο ή περισσότερων λέξεων μικρότερου μήκους.

Για παράδειγμα, οι λέξεις πόδι και σφαίρα, ενώνονται και σχηματίζουν την εντελώς νέα λέξη «ποδόσφαιρο». Σε άλλες περιπτώσεις, η έννοια της λέξης μπορεί να αλλάξει με την τοποθέτηση ενός προθέματος, π.χ. συνέπεια - ασυνέπεια, τρωτός - άτρωτος. Η Z80 CPU διαθέτει ορισμένες δομές λέξεων που χρησιμοποιούν αυτούς τους κατασκευαστικούς τύπους. Παρ' όλα αυτά, το πρόβλημα του περιορισμένου λεξιλογίου παραμένει.

Ο περιορισμός αυτός δεν εμποδίζει την έκφραση όλων των δυνατών εννοιών, απλώς σε ορισμένες περιπτώσεις χρειάζεστε περισσότερες λέξεις για να πείτε αυτό που θέλετε. Για παράδειγμα, η αρχή αυτής της παραγράφου θα μπορούσε να είναι: Αυτή η μη ύπαρξη πολλών λέξεων δεν τοποθετεί εμπόδια στο να αναφερθούν όλες οι έννοιες που είναι δυνατόν να αναφερθούν όταν υπάρχουν πολλές λέξεις... ίδιο νόημα αλλά περισσότερες λέξεις, σε όχι και τόσο καλά ελληνικά και με πολλές επαναλαμβανόμενες λέξεις.

Επομένως, η γλώσσα μηχανής χρειάζεται ένα πλήθος απλών λέξεων, ή εντολών για να δημιουργήσει το ανάλογο μιας εντολής BASIC, ενώ δεν υπάρχει όριο στους τρόπους με τους οποίους μπορούν να συνδυασθούν οι εντολές σε γλώσσα μηχανής, και μερικές φορές αυτό σημαίνει ότι η γλώσσα μηχανής χρειάζεται λιγότερες εντολές από τη BASIC.

Κάθε φορά που τρέχετε ένα πρόγραμμα σε BASIC, το μεταφραστικό πρόγραμμα (interpreter) ελέγχει τις εντολές μια προς μια ώστε να βεβαιωθεί για την ορθότητά τους, μετά τις μεταφράζει σε μια σειρά εντολών σε γλώσσα μηχανής τις οποίες κατόπιν εκτελεί η CPU, ενώ τα αποτελέσματα ελέγχονται για να εξασφαλισθεί ότι είναι αυτά που αναμενόταν και αποθηκεύονται για περαιτέρω χρήση. Όλα αυτά χρειάζονται χρόνο.

Στη γλώσσα μηχανής, εν τούτοις, δεν υπάρχει έλεγχος λαθών, ούτε απαιτείται χρόνος για μετάφραση, και τίποτα δεν αποθηκεύεται εκτός εάν πείτε εσείς στη CPU να το φυλάξει.

Για να δείτε το χρόνο που εξοικονομείτε, πληκτρολογήστε το παρακάτω πρόγραμμα BASIC. Προηγουμένως, όμως, αν έχετε ήδη ανοίξει τον υπολογιστή σας, κλείστε τον και ανοίξτε τον πάλι για να βεβαιωθείτε ότι όλα είναι «παρθένα».

```
10 MM=42619
20 MEMORY 42499
30 FOR N=42500 TO 42509:READ D:POKE N,D:A=A+D:N
EXT
40 IF A<>133B THEN CLS:PEN 3:PRINT"DATA ERROR":
PEN 1:EDIT 90
50 INPUT"PRESS ENTER TO START";A:B=255
```

```

60 PRINT "A"; : B=B-1 : IF B <> 0 THEN 60
70 PRINT
80 CALL 42500
90 DATA 6, 255, 62, 65, 205, 90, 187, 16, 251, 201
100 END

```

Παρατηρήστε πως το ? χρησιμοποιείται στη θέση του PRINT για εξοικονόμηση χρόνου.

Αφού τελειώσετε τη πληκτρολόγηση του προγράμματος γράψτε RUN και πατήστε το πλήκτρο ENTER. Αν όλα είναι εντάξει θα δείτε το μήνυμα «PRESS ENTER TO START» («ΠΑΤΕΙΣΤΕ ENTER ΓΙΑ ΝΑ ΑΡΧΙΣΕΤΕ»), αλλιώς θα εμφανιστεί η γραμμή 90 σε «edit mode» γιατί θα έχετε κάνει κάποιο λάθος κατά τη πληκτρολόγηση των DATA.

Όταν πατηθεί το πλήκτρο ENTER θα τυπωθούν 255 «A» από τη BASIC της γραμμής 60, ενώ αμέσως μετά θα ακολουθήσουν και άλλα 255 «A» τυπωμένα από την υπορουτίνα σε γλώσσα μηχανής την οποία κάνατε «POKE» στη μνήμη με τη γραμμή 30, και την καλέσατε με τη γραμμή 80.

Παρ' όλο που το πρόγραμμα δεν είναι πολύ συναρπαστικό, σας δείχνει τη ταχύτητα της γλώσσας μηχανής.

Αν μετρήσετε τους χαρακτήρες που χρησιμοποιήθηκαν από την υπορουτίνα σε γλώσσα μηχανής (έχοντας υπόψη ότι ο κάθε αριθμός στην εντολή DATA είναι ένας χαρακτήρας, σε γλώσσα μηχανής) θα βρείτε 10, από τους οποίους ο τελευταίος, το 201, υπάρχει μόνο για να πει στο πρόγραμμα να επιστρέψει στη BASIC. Από την άλλη μεριά, το πρόγραμμα σε BASIC χρησιμοποιεί 37 χαρακτήρες αν συμπεριλάβετε τα κενά διαστήματα και δεν μετρήσετε τους αριθμούς γραμμής. Ακόμα κι αν είχε γραφτεί χωρίς παραπανήσια κενά διαστήματα, θα απαιτούσε χώρο που αναλογεί σε 25 χαρακτήρες γλώσσας μηχανής.

Μπορείτε, αν θέλετε, να τα ελέγξετε και μόνοι σας, προσθέτοντας στο πρόγραμμά σας τις παρακάτω γραμμές :

```

110 PEN 3: FOR N = 520 TO 630 : A = PEEK (N)
120 ? A; : IF A = 32 THEN 150
130 IF A > 32 AND A < 129 AND B <> 1 THEN PEN 2: ? : ?N
140 IF A > 32 AND A < 129 THEN PEN 1: ? CHR$ (A); : PEN 3: B = 0
150 NEXT
160 PEN 1
170 END

```

Όταν τρέξει το πρόγραμμα θα παρουσιάσει με κόκκινο χρώμα τις τιμές που έχουν κρατηθεί στις θέσεις της μνήμης όπου βρίσκεται το πρόγραμμα σε BASIC. Εάν υπάρχει ένας συγκεκριμένος χαρακτήρας που αντιπροσωπεύεται από τον αριθμό, θα παρουσιαστεί με κίτρινο. Θα μπορέσετε να ξεχωρίσετε την αρχή της γραμμής 60 ψάχνοντας για το «PRESS ENTER TO START»; στη γραμμή 50 και συνεχίζοντας μέχρι να φτάσετε στο 0 60 σε κόκκινο < σε κίτρινο και 0 σε κόκκινο. Τα 60 0 είναι ο αριθμός γραμμής και ο αριθμός που προηγείται του πρώτου 0 είναι ο αριθμός χαρακτήρων της γραμμής. Ο αριθμός σε χρώμα ανοιχτό γαλάζιο που βρίσκεται στην αρχή της κάθε γραμμής της οθόνης, είναι ο αριθμός της πρώτης θέσης μνήμης στη συγκεκριμένη γραμμή της οθόνης.

Το πρώτο πράγμα που θα παρατηρήσετε είναι ότι οι μόνοι χαρακτήρες που έχουν αποθηκευτεί κατά τον ίδιο τρόπο που τους γράψατε, είναι τα «A». Όλοι οι άλλοι έχουν μεταρραπεί σε ένα είδος κώδικα τον οποίο χειρίζεται ο interpreter με μεγαλύτερη ευκολία. Κάθε φορά που πληκτρολογείτε LIST, ο interpreter ξαναμεταφράζει αυτούς τους αριθμούς σε ό,τι είχατε πληκτρολογήσει.

Από τα παραπάνω βγαίνει το συμπέρασμα ότι το πρόγραμμα σε γλώσσα μηχανής δεν ήταν μονάχα ταχύτερο, αλλά και οικονομικότερο όσον αφορά τη χρησιμοποιούμενη μνήμη. Αυτά είναι τα δύο βασικά πλεονεκτήματα των προγραμμάτων σε γλώσσα μηχανής. Στην πραγματικότητα, ένα πρόγραμμα σε BASIC μπορεί να είναι μέχρι 100 φορές βραδύτερο από το ανάλογο του σε γλώσσα μηχανής.

Τα κύρια μειονεκτήματα της γλώσσας μηχανής είναι ότι τα προγράμματα γίνονται σχεδόν τελείως ακατανόητα και επομένως τα λάθη τους εντοπίζονται δύσκολα, και ότι συνήθως, αν συγκριθούν με τη BASIC ή κάποια άλλη γλώσσα υψηλού επιπέδου ως προς τον αριθμό απαιτούμενων εντολών, τα προγράμματα σε γλώσσα μηχανής είναι μεγαλύτερου μήκους.

Η καταληπτότητα της γλώσσας μηχανής διευκολύνεται πάρα πολύ με τη χρήση των προγραμμάτων assembler και disassembler για τα οποία μιλάμε στο επόμενο κεφάλαιο, και ενώ κανονικά δεν υπάρχει τρόπος αντιμετώπισης του προβλήματος του μεγάλου αριθμού απαιτούμενων εντολών, με την Amstrad, μπορείτε να χρησιμοποιήσετε τις υπορουτίνες του λειτουργικού του συστήματος. Χάρη στην πρόνοια της Amstrad να θέσει στη διάθεσή σας τις λεπτομέρειές του, μπορέσατε ήδη να το κάνετε, εφ' όσον γράψατε το παραπάνω πρόγραμμα. Το μεγαλύτερο τμήμα των εντολών που υπάρχουν στα περισσότερα προγράμματα, έχουν ήδη γραφτεί για σας, από τη Locomotive Software, όταν έγραψαν το λειτουργικό σύστημα.

ΠΡΩΤΕΣ ΕΝΝΟΙΕΣ

Πριν εισέλθουμε στον κόσμο της γλώσσας μηχανής, υπάρχουν οροσμένες έννοιες που ίσως είναι καινούριες για σας και, επειδή είναι σημαντικό το να έχετε μια στοιχειώδη τουλάχιστον κατανόησή τους, θα εξηγηθούν εδώ σε συντομία.

Δεκαεξαδικό (Hex) και δυαδικό (Binary) σύστημα

Τα συστήματα Hex και Binary είναι διαφορετικοί τρόποι μέτρησης, το Binary με βάση το 2 και το Hex με βάση το 16. Ίσως να έχετε συναντήσει το δυαδικό σύστημα στο σχολείο, οπότε χωρίς αμφιβολία θα σκεφτήκατε ότι ήταν ένας πολύ χαζός τρόπος μέτρησης. Για υπολογιστή, εν τούτοις, είναι ο μοναδικός τρόπος, όπως χωρίς αμφιβολία θα έχετε καταλάβει. Επειδή η CPU αναγνωρίζει μόνο τις δύο καταστάσεις OFF και ON, ο μόνος τρόπος που μπορεί να μετρήσει είναι με το δυαδικό σύστημα, οπότε το ON αντιστοιχεί στο 1 και το OFF στο 0.

Το καθένα δυαδικό ψηφίο (Binary Digit ή bit για συντομία), έχει μια συγκεκριμένη τιμή ανάλογα με τη θέση του. Το δεκαδικό σύστημα χρησιμοποιεί την ίδια σύμβαση. Το δεξιότερο ψηφίο είναι ο αριθμός των μονάδων, το επόμενο προς τα αριστερά είναι ο αριθμός των δεκάδων, το επόμενο είναι οι εκατοντάδες κ.ο.κ. Στο δυαδικό, επειδή υπάρχουν μόνο μονάδες και μηδενικά, οι τιμές των θέσεων θα πρέπει να προσαρμοστούν ώστε να μπορεί να γραφτεί οποιοσδήποτε αριθμός. Αν χρησιμοποιούνται στις διάφορες θέσεις ίδιες τιμές με εκείνες του δεκαδικού, τότε θα μετρούσατε ένα, δέκα, έντεκα, εκατό, εκατόν ένα, κ.ο.κ.

Ο υπολογιστής σας αποθηκεύει τις πληροφορίες του σε ομάδες των 8 bits, τα λεγόμενα bytes, και μπορεί να χειριστεί ζεύγη από bytes (16 bits, που είναι τα γνωστά σας Words) σαν να αντιπροσωπεύουν ένα μόνο αριθμό. Οι τιμές που αντιστοιχούν σε κάθε μια από τις 16 θέσεις των bits φαίνονται παρακάτω.

BIT NUMBER															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
VALUE															

Με το συνδυασμό αυτό μπορεί να παρασταθεί οποιοσδήποτε αριθμός από 0 έως 65535 με κάποιο συνδυασμό μονάδων και μηδενικών. Παρατηρήστε ότι το bit με τη μικρότερη τιμή είναι γνωστό σαν bit 0.

Ορισμένες φορές επιθυμούμε να παραστήσουμε μια αρνητική τιμή, και υπάρχει

μια σύμβαση γι' αυτό το σκοπό. Αν αρχίσετε με το 0, που παραμένει το ίδιο σε όλα τα συστήματα αρίθμησης, και αφαιρέσετε 1 θα έχετε -1. Εφαρμόστε τα σε ένα δυαδικό αριθμό και θα αλλάξετε όλα τα bits από 0 σε 1, όσον κι αν προχωρήσετε. Δείτε το παρακάτω παράδειγμα που χρησιμοποιεί έναν αριθμό με 4 bits.

```

0000

-1  0 - 1 = 1 borrow 1

-1  0 - 1 = 1 borrow 1

-1  0 - 1 = 1 borrow 1

-1  0 - 1 = 1 borrow 1

1111

```

Και, επειδή η αφαίρεση περιορίστηκε στα 4 bits, το αποτέλεσμα είναι 1111 στο δυαδικό, ή 15 στο δεκαδικό σύστημα. Το ίδιο θα συμβεί αν η αφαίρεση γίνει με 8 bits, ή 16bits. Το αποτέλεσμα θα είναι 255 και 32767 αντίστοιχα.

Το ίδιο θα συμβεί αν αφαιρέσετε κάποιον αριθμό αντί για το 1, και το αριστερό (με την μεγαλύτερη τιμή) bits θα γίνεται πάντα 1 όταν το αποτέλεσμα είναι αρνητικός αριθμός. Αυτό το τελευταίο μας οδήγησε στον τρόπο παράστασης των δεκαδικών αριθμών.

Αν πρόκειται να χρησιμοποιηθούν αρνητικοί αριθμοί, ή προκύψουν από κάποια αφαίρεση, υπάρχει η σύμβαση να χρησιμοποιείται το αριστερότερο bit για να δηλώνει το πρόσημο του αριθμού: 1 όταν ο αριθμός είναι αρνητικός και 0 για θετικό. Αυτό αλλάζει το εύρος των αριθμών που μπορούν να παρασταθούν από ένα δεδομένο αριθμό bits. Τα 16 bits μπορούν τώρα να παραστήσουν αριθμούς από -32768 έως 32767, και τα 8 bits από -128 έως +127. Η τεχνική αυτή της παράστασης προσημασμένων αριθμών στο δυαδικό σύστημα ονομάζεται «συμπλήρωμα του 2». Αν αλλάξετε όλα τα ψηφία ενός δυαδικού αριθμού (γίνουν οι μονάδες μηδενικά και αντίστροφα), και προσθέσετε 1, του αλλάζετε το πρόσημο.

Από σας θα εξαρτηθεί αν θα χρησιμοποιήσετε «συμπληρώματα του 2» στα προγράμματα, ή κανονικούς δυαδικούς χωρίς πρόσημο. Μπορείτε ακόμη να χρησιμοποιήσετε κάποιο συνδυασμό αυτών των δύο. Η προτεινόμενη για χρήση παράσταση σε μια συγκεκριμένη εντολή, θα φανεί να χρησιμοποιείται στο υπόλοιπο βιβλίο. Όπου επίσης θα μπορούν να χρησιμοποιηθούν και οι δύο μέθοδοι και να μας δώσουν το ανάλογο αποτελέσματα, αυτό θα δηλώνεται.

Ο assembler GENS σας επιτρέπει να χρησιμοποιείτε δυαδικούς αριθμούς σε ένα πρόγραμμα, εφ' όσον μπροστά από το δυαδικό αριθμό υπάρχει το σύμβολο %.

Τι γίνεται όμως με το δεκαεξαδικό σύστημα, το HEX; Ο υπολογιστής δεν αντιμετωπίζει κανένα πρόβλημα όταν σκέφτεται με ONs και OFFs ή 0 και 1, αλλά προσπαθήστε εσείς να μετρήσετε στο δυαδικό σύστημα ή να γράψετε και να κάνετε ελέγχους. Θα το βρείτε απίστευτα αδέξιο! Τις περισσότερες φορές το ευκολότερο προς χρήση αριθμητικό σύστημα θα είναι το δεκαδικό, αλλά θα υπάρξουν ορισμένες περιπτώσεις που θα σας είναι ευκολότερο να σκέπτεστε με όρους του δυαδικού. Για παράδειγμα, όταν θέλετε να βάλετε κάποιον αριθμό σε ένα byte με ειδικό τρόπο. Αν θέλετε να έχετε τον αριθμό 9 (δεκαδικό) στα δύο μισά ενός byte, θα

πρέπει να επιστρέψετε στο δυαδικό για να βρείτε τον τρόπο. Ο 1001 στο δυαδικό είναι ο 9 του δεκαδικού, ($1*8+0*4+0*2+1*1=9$), έτσι θα πρέπει να έχετε τον 1001 1001 ώστε το κάθε μισό του byte να περιέχει τον 9 του δεκαδικού. Η τιμή του αριθμού στο δεκαδικό σύστημα είναι :

$1*128+0*64+0*32+1*16+1*8+0*4+0*2+1*1$

που ισούται με 153 στο δεκαδικό. Μπερδεμένο, έτσι;

Μπορείτε να έχετε οποιαδήποτε τιμή μεταξύ 0 και 15 στο κάθε μισό ενός byte, δημιουργώντας έτσι 16 διαφορετικές τιμές για κάθε ομάδα των 4 bits. Έτσι, για να μπορείτε να εργάζεστε εύκολα με τους δυαδικούς αριθμούς στα bytes, χρειάζεστε ένα νέο σύστημα αρίθμησης με βάση το 16. Αν ήταν δυνατό κάτι τέτοιο, θα μπορούσατε να έχετε γράψει απλώς 99 χωρίς να έχετε κουραστεί ψάχνοντας για το δεκαδικό ισοδύναμο του αριθμού που θέλατε. Το σύστημα αρίθμησης με βάση το 16 ονομάζεται δεκαεξαδικό (HEXadecimal), αλλά επειδή το όνομα είναι αρκετά μεγάλο, όλοι το λένε HEX.

Το πρώτο πρόβλημα που θα συναντήσετε είναι ότι, ενώ υπάρχουν ήδη αριθμοί για να μετράτε από το 0 μέχρι το 9, δεν ξέρετε τι να κάνετε με τους 10 έως 15. Αντί να μάθετε νέα σύμβολα γι' αυτούς τους αριθμούς, θεωρήθηκε προτιμότερο να χρησιμοποιηθούν τα 6 πρώτα κεφαλαία γράμματα του αλφαβήτου. Το 10 του δεκαδικού συστήματος γίνεται επομένως A, το 11 γίνεται B κ.ο.κ. μέχρι το 15 που είναι το F. Το άλλο πρόβλημα είναι ότι οι άλλοι θα νομίζουν ότι χρησιμοποιείτε το δεκαδικό σύστημα, γι' αυτό είναι βασικό να υπάρχει κάποια ένδειξη που να πληροφορεί ότι ο συγκεκριμένος αριθμός έχει σαν βάση το 16 (αριθμός HEX).

Επειδή ατυχώς δεν έχει καθιερωθεί γι' αυτό κάποια σύμβαση, ο Amstrad χρησιμοποιεί το σύμβολο & για να δείξει ότι ο αριθμός που ακολουθεί είναι HEX, το Firmware Specification Manual χρησιμοποιεί το £ και ο assembler GENS χρησιμοποιεί το #, ενώ πολλοί άλλοι assemblers (συμπεριλαμβανομένης πιθανώς και της προσφοράς Picturesque) χρησιμοποιούν το μικρό ή το κεφαλαίο h. Όλα αυτά είναι αρκετά μπερδεμένα, γι' αυτό ας αρκεστούμε στο να πούμε ότι αν κάποιος αριθμός περιέχει και κάτι άλλο εκτός από νούμερα, τότε είναι πιθανώς HEX.

Σ' αυτό το βιβλίο, όλοι οι δεκαεξαδικοί αριθμοί (HEX) ακολουθούνται από το μικρό h, εκτός από τα listings του assembler GENS, όπου υπάρχει το πρόθεμα #.

ASCII

ASCII είναι τα αρχικά των λέξεων American Standard Code for Information Interchange. Αποτελείται από αριθμούς που αντιπροσωπεύουν γράμματα και πράξεις. Στο παράρτημα III (Appendix III) του βιβλίου Amstrad User Instructions περιέχεται ο πλήρης κατάλογος των κωδικών ASCII.

Address (Διεύθυνση)

Ο όρος διεύθυνση χρησιμοποιείται για να περιγράψει μια θέση μνήμης. Η κάθε θέση μνήμης έχει μια μόνο διεύθυνση, αρχίζοντας από το 0 για την πρώτη θέση και καταλήγοντας στο 65535 (FFFFh). Γράφεται συχνά σαν δεκαεξαδικός αριθμός αντί για δεκαδικός, και οι περισσότεροι assemblers δίνουν τη διεύθυνση μιας εντολής στις πρώτες στήλες του κειμένου που εκτυπώνουν.

Assembler

Αναφέρθηκε παραπάνω το όνομα assembler, αλλά τι είναι ένας assembler; Ο assembler είναι ένα πρόγραμμα που σας επιτρέπει να προγραμματίζετε σε γλώσσα μηχανής σε μια μορφή που αναγνωρίζεται ευκολότερα από τους αριθμούς, χρησιμοποιώντας συμβολικές εντολές (mnemonics). Οι συμβολικές εντολές είναι ένα είδος στενογραφίας, με τις οποίες περιγράφουμε τις εργασίες που εκτελούνται από τη γλώσσα μηχανής. Προσφέρουν σημαντική βοήθεια στη μνήμη μας, γιατί χωρίς τη βοήθειά τους δεν θα είμαστε σε θέση να θυμόμαστε όλες τις εντολές αριθμητικής μορφής: Ο assembler σας επιτρέπει να γράψετε το πρόγραμμά σας σ' αυτή τη στενογραφημένη μορφή και, αφού τελειώσετε, το μεταφράζει σε μονάδες και μηδενικά για να γίνουν κατανοητά από τον υπολογιστή.

Οι περισσότεροι assemblers διαθέτουν επίσης ένα ολοκληρωμένο σύστημα διάρθρωσης λαθών (integral Editor), για να μπορείτε να γράφετε και να τροποποιείτε με ευκολία τα προγράμματά σας. Χωρίς αυτή την ευκολία, αν είχατε γράψει ένα μεγάλο πρόγραμμα και κατόπιν ανακαλύπτατε ότι είχατε κάνει λάθος στη ν-οστή εντολή, θα έπρεπε να τα ξαναγράψετε από εκείνο το σημείο μέχρι το τέλος.

Το πρόγραμμα που γράφετε χρησιμοποιώντας τον assembler ονομάζεται πηγαίος κώδικας (source code) και μπορεί να αποθηκευτεί σε κασέτα ώστε να το διορθώσετε αργότερα εφ' όσον χρειάζεται, αλλά δεν είναι ανάγκη να τρέξετε άμεσα ένα πρόγραμμα που γράψατε με τον assembler. Το πραγματικό πρόγραμμα που μπορεί να τρέξει, ή να εκτελεσθεί όπως θα έπρεπε να λέμε, είναι ο αντικειμενικός κώδικας (object code).

Ο αντικειμενικός κώδικας μπορεί επίσης να αποθηκευτεί σε κασέτα, χρησιμοποιώντας την εντολή O από τον assembler GENS ή από τη BASIC. Όταν αποθηκεύεται από τη BASIC, ο τύπος της εντολής είναι: SAVE «όνομα», B, αρχική διεύθυνση μήκος, σημείο εισόδου. Το σημείο εισόδου είναι η διεύθυνση από την οποία αρχίζει η εκτέλεση εφ' όσον το πρόγραμμα φορτώνεται με την εντολή RUN, ενώ αν δεν έχει καθοριστεί, ο Amstrad θα κάνει reset όταν το πρόγραμμα φορτώνεται με RUN.

Ένας assembler θα σας επιτρέψει να χρησιμοποιήσετε τις λεγόμενες «ετικέτες» ('labels') αντί για διευθύνσεις όταν γράφετε ένα πρόγραμμα γλώσσας μηχανής. Αυτό είναι εξαιρετικά χρήσιμο και μοιάζει πολύ με την ανάλογη ευκολία της PASCAL. (Η PASCAL είναι μια γλώσσα υψηλού επιπέδου σαν τη BASIC αλλά έχει σχεδιαστεί να μεταφράζεται σαν τη γλώσσα assembly. Η γλώσσα μηχανής που δημιουργεί δεν τρέχει τόσο γρήγορα όσο εκείνη που δημιουργείται από την assembly, και χρειάζεται περισσότερο χώρο, αλλά πάντως είναι πολύ ταχύτερη από τη BASIC).

Στην PASCAL αντί να χρησιμοποιείτε το GOSUB ακολουθούμενο από έναν αριθμό γραμμής, δίνετε κάποιο όνομα σε μια υπορουτίνα και γράφετε απλώς το όνομά της μέσα στο πρόγραμμα. Μόλις συναντηθεί αυτό το όνομα, εκτελείται η σχετική υπορουτίνα. Ο assembler επιτρέπει να τοποθετούνται οι labels (σύντομα ονόματα που τελειώνουν σε δύο τελείες:) στο listing δίπλα σε μία εντολή και όταν αυτή η label αναφέρεται χρησιμοποιείται η διεύθυνση στην οποία αποθηκεύτηκε η εντολή που βρίσκεται δίπλα της. Αυτό είναι σαν να μπορείτε να δίνετε όνομα σε υπορουτίνα, οπότε δεν χρειάζεται πλέον να γνωρίζετε τον αριθμό γραμμής της αρχικής της εντολής γιατί μπορείτε απλώς να γράφετε GOSUB και το όνομα της υ-

πορουτίνας.

Ο assembler επιτρέπει επίσης τη χρήση ψευδολειτουργιών (Pseudo Operations) ή ψευδοσυμβολικών εντολών (Pseudo Mnemonics) όπως λέγονται σε ορισμένες περιπτώσεις. (Το γιατί δεν μπορεί να το μαντέψει κανείς, γιατί πρόκειται για πραγματικές συμβολικές εντολές που απλώς δεν μεταφράζονται σε γλώσσα μηχανής). Χρησιμοποιούνται για να λένε στον assembler να κάνει κάτι με τον επόμενο αριθμό, και οι κυριότερες είναι:

;	Που πληροφορεί τον assembler πως ό,τι ακολουθεί είναι σχόλιο και πρέπει να αγνοηθεί. Είναι όμοια με τη REM της BASIC.
EQU	Από το EQUate, ή EQUals (είναι ίσο). Σας επιτρέπει να χρησιμοποιήσετε μια label που θα αντιπροσωπεύει οποιονδήποτε αριθμό διαλέξετε. Αυτή η label πρέπει να τοποθετηθεί στα αριστερά της ψευδολειτουργίας EQU και να τελειώνει με τις συνηθισμένες δύο τελείες, ενώ ο αριθμός με τον οποίο θέλετε να εξισώσετε τη label πρέπει να είναι στα δεξιά. Για παράδειγμα, η LABEL:EQU#1234 θα υποχρεώσει την ετικέτα (label) με το όνομα LABEL να παίρνει την τιμή 1234 h (4660 στο δεκαδικό) κάθε φορά που χρησιμοποιείται.
DEFB	DEFine Byte. Το byte αυτής της διεύθυνσης θα περιέχει την τιμή που ακολουθεί. Για παράδειγμα, η DEFB #20 θα υποχρεώσει το byte στη διεύθυνση της συμβολικής εντολής DEFB να πάρει την τιμή 20 h (στο HEX) όταν το πρόγραμμα μεταφραστεί.
DEFW	DEFine Word. Ίδια με την προηγούμενη συμβολική εντολή, με τη διαφορά ότι θα τοποθετηθεί ένας αριθμός των 16 bits σε δύο bytes μνήμης. Στο byte της εντολής και στο αμέσως επόμενο.
DEFM	DEFine Message. Επιτρέπει να τοποθετηθούν γράμματα μετά τη συμβολική εντολή, μεταξύ εισαγωγικών, με τους ASCII κωδικούς τους τοποθετημένους σε διαδοχικές θέσεις στο μεταφρασμένο πρόγραμμα.
DEFS	DEFine Space. Κατά τη μετάφραση του προγράμματος σε γλώσσα μηχανής, θα αγνοηθούν θέσεις μνήμης ίσες με τον αριθμό που ακολουθεί τη συμβολική εντολή DEFS.
ORG	ORiGinate. Ο αριθμός που ακολουθεί τη συμβολική εντολή ORG θα είναι η διεύθυνση της επόμενης εντολής όταν μεταφραστεί το πρόγραμμα.
ENT	ENTry. Η διεύθυνση από την οποία θα αρχίσει η εκτέλεση, σε απάντηση της εντολής J του assembler.

Η συμβολική εντολή ORG σε ένα listing θα δώσει την αρχική διεύθυνση (start address) ενός τμήματος του προγράμματος, την οποία χρειάζεται το πρόγραμμα HEX Loader, και η συμβολική εντολή ENT θα ακολουθείται από τη διεύθυνση που θα πρέπει να κληθεί με την εντολή CALL από την BASIC για να τρέξει ένα πρόγραμμα γλώσσας μηχανής.

Μια εντολή γλώσσας μηχανής αποτελείται από τον κωδικό λειτουργίας (OP-CODE) που λέει στον υπολογιστή τι να κάνει και σε ορισμένες περιπτώσεις ακολουθείται από το «τελούμενο» (OPERAND) που δίνει τις πληροφορίες, σύμφωνα με τις οποίες θα εκτελεστεί η εντολή.

Assembler listings

Τα listings των προγραμμάτων γλώσσας μηχανής συνήθως έχουν πέντε στήλες, ή έξι στη περίπτωση που χρησιμοποιούνται σχόλια (που έχουν, αν θυμόμαστε, ένα ; μπροστά τους). Η πρώτη στήλη δίνει την αρχική διεύθυνση της εντολής, συνήθως στο HEX.

Η δεύτερη στήλη παρουσιάζει τη δεκαεξαδική μορφή της εντολής σε γλώσσα μηχανής, και αυτή είναι που θα πρέπει να εισαχθεί αν χρησιμοποιείτε ένα HEX Loader πρόγραμμα, σαν κι αυτό που βρίσκεται στο Παράρτημα, σε ζεύγη αριθμών.

Η τρίτη στήλη είναι ένας αριθμός γραμμής, και είναι χρήσιμη μόνον όταν γράφετε το πρόγραμμα.

Στην τέταρτη βρίσκονται οι διάφορες labels, δίπλα στην εντολή που καταλαμβάνει τη διεύθυνση στην οποία αναφέρεται η label, αλλά η στήλη στην οποία πρέπει να τελειώνει η label δεν φαίνεται. Πρέπει να τη γράψετε αν αντιγράψετε κάποιο πρόγραμμα από listing.

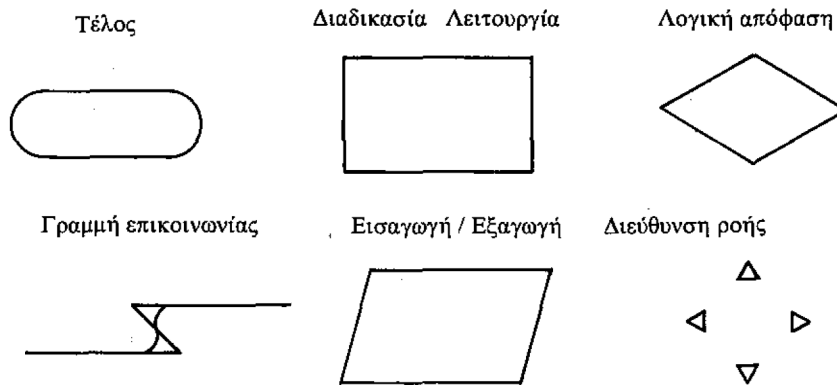
Η πέμπτη στήλη είναι η συμβολική μορφή της εντολής, όπως εισάγεται από τον προγραμματιστή, και αυτό είναι που γράφετε αν χρησιμοποιείτε assembler, μετά από τις labels που πιθανόν να υπάρχουν στην τέταρτη στήλη της ίδιας γραμμής.

Η έκτη στήλη μπορεί να περιέχει ένα σχόλιο.

Τώρα που εφοδιαστήκατε μ' αυτές τις πληροφορίες πρέπει να είστε έτοιμοι να προχωρήσετε!

ΔΙΑΓΡΑΜΜΑΤΑ ΡΟΗΣ (FLOW CHARTS)

Το διάγραμμα ροής χρησιμοποιείται συχνά κατά το σχεδιασμό και το στάδιο ανάπτυξης ενός προγράμματος. Πρόκειται απλώς για μια συμβολική παράσταση της ροής του υπό ανάπτυξη προγράμματος. Υπάρχει μια τυποποιημένη ομάδα συμβόλων που χρησιμοποιείται στο σχεδιασμό διαγραμμάτων ροής και αυτά που είναι πολύ πιθανό να χρησιμοποιήσετε φαίνονται στο Σχήμα 4.1 με τις λειτουργίες τους.

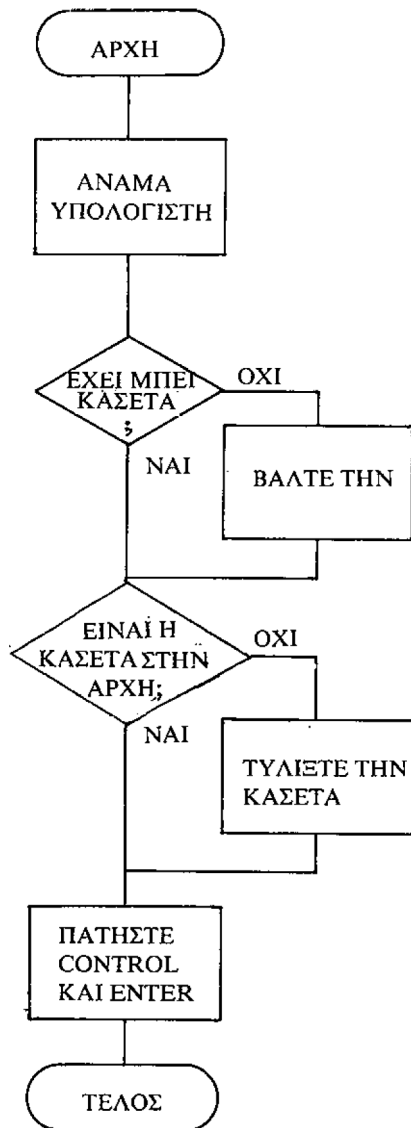


Σχήμα 4.1

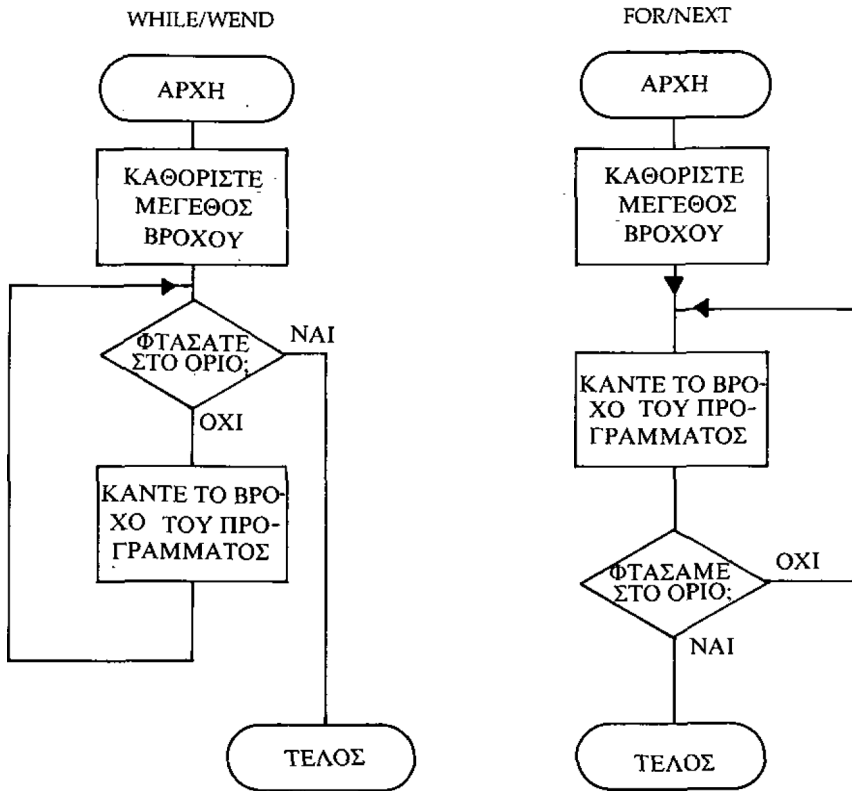
Υπάρχουν και αρκετά άλλα σύμβολα, αλλά δεν χρησιμοποιούνται συχνά. Ο σκοπός του διαγράμματος ροής είναι να καθορίσει με σαφήνεια τις εργασίες που εκτελεί ένα πρόγραμμα. Ας θεωρήσουμε ένα πολύ απλό παράδειγμα, ένα διάγραμμα ροής για να φορτωθεί ένα πρόγραμμα από την κασέτα στον Amstrad σας.

Το διάγραμμα ροής περιλαμβάνει τις βασικές εργασίες, χωρίς να εισέρχεται σε πάρα πολλές λεπτομέρειες, όπως θα έπρεπε να κάνει ένα διάγραμμα ροής. Υπάρχουν πολλές εφαρμογές όπου το διάγραμμα ροής παίζει ουσιώδη ρόλο στην ανάλυση των ενεργειών που εκτελούνται, ή πρέπει να εκτελούνται, από ένα πρόγραμμα. Μπορείτε συχνά να εντοπίσετε λάθη πριν συμβούν, γιατί με μια ματιά στο διάγραμμα ροής μπορείτε να καταλάβετε τις γενικές αρχές του προγράμματος που εξετάζετε.

Δείτε το παράδειγμα στο Σχήμα 4.3, όπου φαίνεται η διαφορά μεταξύ ενός βρόχου WHILE και ενός βρόχου FOR NEXT στη BASIC. Θα πρέπει να καταλάβετε αμέσως, τη βασική διαφορά τους.



Σχήμα 4.2



Σχήμα 4.3

ΑΠΛΕΣ ΕΝΤΟΛΕΣ ΣΕ ΓΛΩΣΣΑ ΜΗΧΑΝΗΣ

LD CALL RET JP JR

Η CPU διαθέτει 14 καταχωρητές (registers), που ο καθένας μπορεί να θεωρηθεί ότι μοιάζει με μια ακέραια μεταβλητή της BASIC. Παρουσιάζονται παρακάτω μαζί με τις λειτουργίες τους. Μην ανησυχείτε αν δεν βγάξετε νόημα, θα τα καταλάβετε όλα σε λίγο.

ΚΑΤΑΧΩΡΗΤΕΣ CPU Z 80

	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1
Συσσωρευτής	A							F							Flag	
ΚΑΤΑΧΩΡΗΤΕΣ	B							C								
ΓΕΝΙΚΗΣ	D							E								
ΧΡΗΣΗΣ	H							L								
ΔΙΑΚΟΠΕΣ	I							R							REFRESH	
ΚΑΤΑΧΩΡΗΤΕΣ ΠΕΡΙΕΧΟΜΕΝΩΝ	XH			IX			XL									
	YH			IY			YL									
ΜΕΤΡΗΤΗΣ ΣΤΟΙΒΑΣ								SP								
ΜΕΤΡΗΤΗΣ ΠΡΟΓΡΑΜΜΑΤΟΣ								PC								

Σχήμα 5.1

Υπάρχουν 6 καταχωρήτες γενικής χρήσης (general - purpose registers), και μ' αυτούς πρόκειται να ασχοληθούμε σε τούτο το κεφάλαιο, μαζί με τον καταχωρητή ειδικής χρήσης «A» ή συσσωρευτή (accumulator) και τον καταχωρητή PC ή μετρητή προγράμματος (program counter register).

Ο καθένας από τους καταχωρητές γενικής χρήσης B, C, D, E, H και L μπορεί να συγκρατεί κάποιον αριθμό από 0 έως 255, που δημιουργείται από 8 bits, και μπορεί να φορτωθεί κατά 3 βασικούς τρόπους. Για να συνεχίσουμε την παρομοίωση με την μεταβλητή της BASIC και για να σας εξηγήσουμε τους τρόπους με τους οποίους μπορεί να φορτωθεί ένας καταχωρητής, πληκτρολογήστε το μικρό πρόγραμμα BASIC του Σχ. 5.2. Δεν χρειάζεται να «σθήσετε» το προηγούμενο πρόγραμμα, εφ' όσον το έχετε κρατήσει.

```
180 CLS
190 WINDOW#1, 1, 40, 1, 10
200 WINDOW#2, 1, 40, 13, 23
210 WINDOW#3, 1, 40, 12, 12
220 PEN#3, 2: PRINT#3, " DECIMAL BINA
RY    HEX"
230 INPUT#1, "ENTER A NUMBER ";A
240 IF A > 255 THEN PRINT#1, "INVALID IN
PUT, IT MUST BE BELOW 256": GOTO 230
250 A = INT (A)
260 PRINT#2,USING "#####"; A; : PRINT#2
, "      "; BIN$ (A,B); "      "; HEX$ (A,2)
270 PRINT#1 : PRINT#2
280 GOTO 230
```

Σχήμα 5.2

Αφού τελειώσετε τη πληκτρολόγηση του προγράμματος, «τρέξτε» το γράφοντας RUN 180, και θα σας ζητηθεί να δώσετε έναν αριθμό. Η BASIC μεταβλητή «A» στο πρόγραμμα αντιπροσωπεύει τον καταχωρητή «A» στη CPU. Αν ο αριθμός που θα δώσετε είναι μεταξύ 0 και 255, θα τυπωθεί στο δεκαδικό, όπως τον γράψατε, στο δυαδικό, όπως τον χειρίζεται ο υπολογιστής, και στο δεκαεξαδικό σύστημα.

Ο αριθμός που δίνετε, φορτώνεται στη BASIC μεταβλητή A η οποία χρησιμοποιείται κατόπιν για να προμηθεύει τον αριθμό όταν τον χρειάζεται το πρόγραμμα. Αν σκοπεύατε να δώσετε το 77, τότε θα «φορτώνατε στην A το 77» (στα αγγλικά "load A, 77"), πατώντας το ENTER.

Στη γλώσσα μηχανής, η παραπάνω εντολή είναι: Load A, 77. Εύκολο, έτσι;

Δυστυχώς όμως, η CPU δεν τα καταλαβαίνει, γιατί αυτό που χρειάζεται είναι το 001111100 ακολουθούμενο από το 01001101, ή το 3Eh ακολουθούμενο από το 4Dh, ή τους δεκαδικούς 62 και 77. Τώρα δυσκολεύουν πάλι τα πράγματα και εδώ είναι που μπαίνει στο προσκήνιο ο assembler. Μπορείτε να πείτε στον assembler ότι η εντολή είναι LD A, 77. Ο assembler θα κάνει τη μετάφραση της εντολής αντί για

σας. Παρατηρήστε πως το load έγινε LD για συντομία, εξοικονομώντας έτσι κόπο όταν πληκτρολογείτε προγράμματα σε γλώσσα μηχανής, και πρόκειται για τυπική σύμβαση του Z80 assembler.

Αν επιστρέψετε τώρα στο πρόγραμμα του δεύτερου Κεφαλαίου και κοιτάξετε τη γραμμή 90, θα δείτε ότι το τρίτο δεδομένο της εντολής DATA ήταν το 62. Αυτό θα σας δώσει μια ιδέα για κάποια από τις εντολές του προγράμματος, και αν βρίσκατε τον κωδικό ASCII του A (από τα οποία τύπωσε πάρα πολλά το πρόγραμμα), θα βλέπατε ότι είναι το 65. Τα πράγματα πρέπει να αρχίζουν να καθαρίζουν τώρα, και ίσως γίνουν ακόμη καθαρότερα όταν ανακαλύψετε ότι ο κωδικός της εντολής που φορτώνει τον καταχωρητή B με ένα αριθμό είναι ο δυαδικός 00000110 ή ο δεκαδικός 6 (και hex σ' αυτή την περίπτωση), και θυμηθείτε ότι τυπώθηκαν 255 A.

Ποιός είπε ότι η γλώσσα μηχανής είναι δύσκολη; Οι δύο αρχικές εντολές του προγράμματος, στον assembler, είναι επομένως:

LD B, 255

LD A, 65

Γνωρίζοντας τα παραπάνω, μπορείτε τώρα να αλλάξετε τον αριθμό των χαρακτήρων που τυπώνονται, καθώς και το είδος του χαρακτήρα. Πρώτα πηγαίνετε στο Παράρτημα III του βιβλίου Amstrad User Instruction που συνοδεύει τον υπολογιστή σας. Στη σελίδα 1 θα δείτε τους κωδικούς όλων των ASCII χαρακτήρων και στις επόμενες σελίδες τους χαρακτήρες που δημιουργούνται από όλους τους κωδικούς από 32 έως 255.

Αφαιρέστε τώρα τη γραμμή 40, που ελέγχει αν γράφετε σωστά τα DATA κατά την πληκτρολόγηση του προγράμματος. Ο έλεγχος θα έβρισκε λάθη, εφ' όσον πρόκειται να αλλάξετε δεδομένα στη γραμμή DATA.

Το μόνο που χρειάζεται να κάνετε τώρα είναι να αλλάξετε το 255 με τον αριθμό των χαρακτήρων που θέλετε να τυπωθούν, το 65 με τον κωδικό του χαρακτήρα που θα τυπωθεί, και κατόπιν να «τρέξετε» το πρόγραμμα από την αρχή για να δείτε το αποτέλεσμα. Μην χρησιμοποιήσετε κωδικούς μικρότερους του 32, γιατί μπορεί να έχετε περίεργα αποτελέσματα.

Αν αντικαταστήσετε το 255 με το 0, θα δείτε ότι ο χαρακτήρας του καταχωρητή A τυπώνεται 256 φορές. Έχοντας υπόψη ότι ο κάθε καταχωρητής μπορεί να κρατήσει μόνο 8 bits και κοιτάζοντας στη BASIC της γραμμής 40, μπορείτε να καταλάβετε γιατί συμβαίνει αυτό;

Σκεφθείτε τη διαδοχή των πράξεων, $A=0$ ή 00000000b. $0-1=-1$ αλλά 00000000b - 00000001b = 11111111b που αντιστοιχεί στο δεκαδικό αριθμό 255.

Αυτό μπορείτε να το ελέγξετε ρωτώντας τον υπολογιστή σας. Πληκτρολογήστε ? BINS (-1) και θα πάρετε την απάντηση 11111111111111. Επειδή ένας καταχωρητής ή θέση μνήμης μπορεί να κρατήσει μόνο τα 8 αριστερότερα ψηφία, ο δεκαδικός -1 ισούται με το 255 όταν μεταφέρεται μέσω ενός καταχωρητή των 8 bits. Μπερδευτήκατε; Επιστρέψτε τότε στο Κεφάλαιο 3 αυτού του βιβλίου και στη σελίδα 2 του Παραρτήματος II του βιβλίου Amstrad Computer User's Instructions.

Ο καθένας από τους καταχωρητές γενικής χρήσης μπορεί να φορτωθεί με τον ίδιο τρόπο που φορτώνονται οι καταχωρητές A και B. Παρακάτω δίνεται ο κωδικός του καθενός:

ASSEMBLER	DECIMAL	HEX	BINARY
LD B,n	06 n	06 n	00 000 110 n
LD C,n	14 n	0E n	00 001 110 n
LD D,n	22 n	16 n	00 010 110 n
LD E,n	30 n	1E n	00 011 110 n
LD H,n	38 n	26 n	00 100 110 n
LD L,n	46 n	2E n	00 101 110 n
LD A,n	62 n	3E n	00 111 110 n

Σχήμα 5.3

Σε κάθε περίπτωση το n αντιπροσωπεύει οποιονδήποτε αριθμό μεταξύ 0 και 225 στο δεκαδικό (FFh ή 1111111b) που πρόκειται να φορτωθεί στο σχετικό καταχωρητή.

Αν κοιτάξετε προσεκτικότερα τους δυαδικούς κώδικες της κάθε εντολής, θα πρέπει να παρατηρήσετε δύο πράγματα.

Το πρώτο είναι ότι τα δύο άκρα όλων των εντολών είναι πάντοτε ίδια. Αυτά τα δύο τμήματα πληροφορούν τη CPU ότι πρόκειται για εντολή φορτώματος (load instruction) που περιλαμβάνει έναν αριθμό ο οποίος θα φορτωθεί σε κάποιον καταχωρητή.

Το δεύτερο είναι ότι ο καταχωρητής ορίζεται από τα bits 5, 4 και 3 και ότι λείπει ένας από τους δυνατούς συνδυασμούς. Τα τρία bits που αποφασίζουν για τον καταχωρητή που θα χρησιμοποιηθεί είναι πάντα ίδια για όλους τους καταχωρητές. Κάθε φορά που πρόκειται να εκτελεστεί μια εντολή σε κάποιον από τους καταχωρητές γενικής χρήσης, χρησιμοποιούνται τρία bits για να πληροφορήσουν τη CPU ποιος καταχωρητής πρόκειται να χρησιμοποιηθεί.

- B είναι πάντοτε 000
- C είναι πάντοτε 001
- D είναι πάντοτε 010
- E είναι πάντοτε 011
- H είναι πάντοτε 100
- L είναι πάντοτε 101
- A είναι πάντοτε 111

Σχήμα 5.4

Ο συνδυασμός που λείπει, ο 110, χρησιμοποιείται σε ειδικές χρήσεις και θα εξηγηθεί αργότερα σ' αυτό το κεφάλαιο.

Εκτός από τη δυνατότητα άμεσης φόρτωσης ενός καταχωρητή με έναν αριθμό από την επόμενη θέση μνήμης, είναι δυνατό να φορτωθεί ένας καταχωρητής είτε με τα περιεχόμενα ενός άλλου καταχωρητή είτε από τη μνήμη.

Σκεφθείτε την εντολή της BASIC A=B. Αυτό που κάνει είναι να πληροφορεί τον υπολογιστή ότι θέλετε η μεταβλητή «A» να γίνει ίση με τη μεταβλητή «B».

Αν πληκτρολογήσετε στον υπολογιστή σας τις επόμενες εντολές και τις εκτελέσετε γράφοντας RUN 300 θα δείτε ότι μετά τη γραμμή 320 η A πήρε την τιμή της B, ενώ η B δεν άλλαξε.

```
300 B = 10
310 ? "BEFORE :A=";A;" B=";B
320 A = B
330 ? "AFTER :A=";A;" B=";B
```

Σχήμα 5.5

Γνωρίζοντας ότι η συμβολική εντολή σε γλώσσα μηχανής που αντιστοιχεί στη γραμμή 300 είναι LD B,10 ποιά νομίζετε ότι είναι η αντίστοιχη της γραμμής 320;

Είναι φανερό, δεν είν' έτσι; LD A,B και το ίδιο ισχύει για όλους τους καταχωρητές.

Οι πραγματικές εντολές κατασκευάζονται με τον ίδιο τρόπο που φορτώνεται ένας αριθμός σε καταχωρητή, με τη διαφορά ότι τα bits 7 και 6 αλλάζουν από 00 σε 01, ενώ τα τρία τελευταία bits, αντί να είναι 110 χρησιμοποιούνται για να δείχνουν τον καταχωρητή από τον οποίο θα ληφθεί η τιμή. Η εντολή επομένως είναι:

```
ASSEMBLER DECIMAL HEX BINARY
LD A,B      120      7B 01 111 000
```

Αν θυμάστε ποιά είναι τα τρία bits που αντιπροσωπεύουν τον καθένα καταχωρητή γενικής χρήσης ή αφού τα ξαναδείτε, θα πρέπει να μπορείτε να δώσετε σε δυαδική μορφή την εντολή φόρτωσης ενός καταχωρητή σε οποιονδήποτε άλλο.

Τα bits 7 και 6 θα είναι πάντοτε 01, τα bits 5,4 και 3 θα είναι ο καταχωρητής που θα φορτωθεί και τα bits 0,1 και 2 ο καταχωρητής από τον οποίο θα γίνει το φόρτωμα.

Επομένως, η LD H, A θα είναι 01 100 111. Πως γράφεται η LD A, H; H LD B, D;

Τώρα διαθέτετε δύο τρόπους με τους οποίους μπορείτε να τοποθετήσετε αριθμούς σε καταχωρητές και χωρίς αμφιβολία αντιλαμβάνεστε ότι ο τρόπος σχηματισμού των εντολών είναι πολύ λογικός. Συνεπώς, μπορείτε να τις παρακολουθήσετε χωρίς μεγάλη δυσκολία.

Όλοι οι καταχωρητές γενικής χρήσης μπορούν να εκτελέσουν και ορισμένες ειδικές λειτουργίες, τις οποίες θα γνωρίσετε σε διάφορα στάδια αυτού του βιβλίου.

Δυστυχώς όμως, κι αυτό δεν συμβαίνει με τις BASIC μεταβλητές, οι περιορισμοί στη χρήση τους είναι σταθεροί, δεν αποφασίζονται από το χρήστη, και είναι αυστηρότεροι, ή μάλλον αποτελεσματικότεροι.

Μην ανησυχείτε, η ομοιότητα μεταξύ των καταχωρητών στη CPU και στις μεταβλητές της BASIC η οποία έχει τονιστεί, εξακολουθεί να ισχύει, αλλά, ενώ όταν θέσετε σε λειτουργία τον Amstrad, οποιαδήποτε μεταβλητή μπορεί να χρησιμοποιηθεί για οποιονδήποτε σκοπό ανεξάρτητα από το αν είναι αλφαριθμητική, ακέραια ή πραγματική και κάθε αριθμητική μεταβλητή μπορεί να αντικαταστήσει μια άλλη, στη γλώσσα μηχανής του Z80 υπάρχουν πράγματα που μπορούν να γίνουν μόνο από ειδικούς καταχωρητές.

Το παραπάνω μοιάζει με το αποτέλεσμα που θα είχε η πρόσθεση της επόμενης γραμμής στο πρόγραμμα του Κεφαλαίου 2: 21 DEFSTR A. Όταν τώρα τρέξετε το πρόγραμμα θα εμφανιστεί το μήνυμα λάθους type mismatch όταν η γραμμή 30 προσπαθήσει να προσθέσει αριθμητικά δεδομένα σε μια αλφαριθμητική μεταβλητή. Έχει ήδη εξηγηθεί ότι ένας καταχωρητής γενικής χρήσης μπορεί να κρατήσει μια τιμή των 8 bits μόνο, εκτός όμως από αυτό το περιορισμό, οποιοσδήποτε καταχωρητής μπορεί να αντιπροσωπεύσει έναν αριθμό, σαν τον καταχωρητή B στο πρόγραμμα του Κεφαλαίου 2, ή ένα γράμμα, σαν τον καταχωρητή A στο ίδιο πρόγραμμα.

Αν γράφατε την πρόταση 8=9 θα ήταν ανόητο, μιας και όλοι γνωρίζουμε ότι 8=8 και 9=9. Αν γράψετε στον υπολογιστή 8=9 και πατήσετε το ENTER δεν θα εμφανιστεί μήνυμα λάθους, κι αυτό γιατί ο υπολογιστής θεωρεί το πρώτο 8 σαν αριθμό γραμμής κι αν κάνετε list στο πρόγραμμα θα δείτε τη γραμμή 8. Αν το αλλάξετε γράφοντας 8 8=9 και προσπαθήσετε να το εκτελέσετε, ο υπολογιστής θα παρουσιάσει syntax error. Το ίδιο θα συμβεί αν προσπαθήσετε να εξισώσετε έναν αριθμό με μια μεταβλητή, για παράδειγμα 8=HL.

Ανεξάρτητα από το τι κάνετε (εκτός αν ξαναπρογραμματίσετε το πλήκτρο), αν γράψετε ? 8 θα σας δώσει σαν απάντηση το 8. Αλλάξτε το όμως με το ? PEEK(8) και θα πάρετε σαν απάντηση το 195. Αυτό γίνεται γιατί προσθέτοντας τη συνάρτηση PEEK (PEEK function) ρωτάτε τον υπολογιστή «ποιά είναι τα περιεχόμενα στη θέση μνήμης με διεύθυνση 8;» αντί για «τι είναι το 8;».

Όταν γράφατε σε γλώσσα μηχανής δεν χρειάζεστε τη συνάρτηση PEEK, ή την ανάλογη εντολή POKE γιατί βρίσκεστε ήδη σε επίπεδο μηχανής. Εξακολουθείτε όμως να έχετε ανάγκη προσπέλασης στις θέσεις μνήμης.

Ο καταχωρητής A εξηγείται στο επόμενο κεφάλαιο ως προς το ρόλο του σαν συσσωρευτής (accumulator), αλλά διαθέτει και ορισμένες ειδικές εντολές που έχουν σχέση με τούτο το κεφάλαιο. Αυτό οφείλεται στο ότι είναι ο μόνος καταχωρητής των 8 bits που μπορεί να φορτωθεί άμεσα από μια θέση μνήμης, έχοντας σαν αντίστοιχο στη BASIC το:

```
A=PEEK(nn)
```

όπου nn είναι οποιοσδήποτε αριθμός των 16 bits.

Η πρώτη σκέψη για την εντολή γλώσσας μηχανής που θα φορτώνει τον καταχωρητή A με τα περιεχόμενα της θέσης μνήμης με διεύθυνση 8 θα ήταν να είναι LD A, 8 αλλά αμέσως θα παρατηρήσετε ότι έτσι θα φορτωνόταν η τιμή 8 στον A.

Χρειάζεται, επομένως, μια μέθοδος διαφοροποίησης.

Η κάθε μία θέση μνήμης μπορεί να θεωρηθεί σαν ένα κουτί, χωρισμένο σε 8 μικρότερα κουτάκια, και η συνάρτηση PEEK της BASIC μας ενισχύει αυτή την αντίληψη, χρησιμοποιώντας παρενθέσεις — που μοιάζουν λίγο με μικρό κουτί — γύρω από τον αριθμό του κουτιού που πρόκειται να επιθεωρηθεί.

Ίσως να το έχετε ήδη καταλάβει, οπότε δεν χρειάζεται να σας το πούμε. Για επιβεβαίωση, πάντως σας λέμε ότι η συμβολική εντολή για το «με τα περιεχόμενα του» χρειάζεται παρενθέσεις γύρω από αυτό που αντιπροσωπεύει τη διεύθυνση της συγκεκριμένη θέσης μνήμης. Επομένως, η εντολή φόρτωσης του καταχωρητή A με τα περιεχόμενα της μνήμης που έχει διεύθυνση 8 είναι LD A, (8), ενώ για να φορτώσετε στη διεύθυνση 4000 της μνήμης τα περιεχόμενα του καταχωρητή A η εντολή είναι LD (4000), A.

Αν δεν έχετε assembler τα πράγματα μπερδεύονται κάπως, αλλά όχι πολύ.

ASSEMBLER	DECIMAL	HEX	BINARY
LD A, (nn)	5B n n	3A n n	00 111 010 n n
LD (nn), A	50 n n	32 n n	00 110 010 n n

Είναι σημαντικό να θυμάστε ότι τα 2 “n” στις παραπάνω εντολές καταλαμβάνουν μια θέση μνήμης και υπολογίζονται από τον τύπο:

$n1 = \text{αριθμός MOD (256)}$ και $n2 = \text{INT(αριθμός/256)}$

Αυτό γίνεται εξαιτίας της εσωτερικής λειτουργίας της CPU και δεν υπάρχει άλλος τρόπος παρά να έχετε το λιγότερο σημαντικό byte μπροστά από το σημαντικότερο byte. Το αντίθετο, δηλ., από αυτό που θα περιμένατε. (Σ.τ.Μ.: Ένα ψηφίο θεωρείται σημαντικότερο από ένα άλλο, αν κατέχει ψηλότερη αριθμητική τάξη. Π.χ. στο δεκαδικό σύστημα το ψηφίο των δεκάδων είναι σημαντικότερο από το ψηφίο των μονάδων). Όλοι οι αριθμοί των 16 bits αποθηκεύονται στη μνήμη κατ’ αυτόν τον τρόπο, είτε είναι τμήματα κάποιας εντολής είτε είναι απλώς δεδομένα στη μνήμη τα οποία τοποθετήθηκαν εκεί από τη CPU.

Θα έπρεπε να μπορείτε να χρησιμοποιείτε την παραπάνω εξίσωση κατευθείαν στον υπολογιστή σας ώστε να μην χρειάζεται αν υπολογίζετε τα $n1$ και $n2$ του κάθε νέου αριθμού, αλλά εξαιτίας σοβαρών ατελειών στην BASIC του Amstrad, η οποία σε ορισμένες περιπτώσεις χρησιμοποιεί το «συμπλήρωμα του 2» και σε άλλες την κανονική ακέραια απεικόνιση, η συνάρτηση MOD είναι άχρηστη για τιμές μεγαλύτερες από 32767.

Πάντως, η παραπάνω γραμμή σε BASIC θα κάνει τη δουλειά σας, και μπορείτε να την προσθέσετε στο πρόγραμμα που ήδη υπάρχει στη μνήμη.

```
1010 N2= INT (NUMBER/256): N1 = NUMBER - N2*256: ? "N1 =";N1;"  
N2 =";N2
```

Αν πληκτρολογήσετε τώρα:

NUMBER=40000:GOTO 1010 και πατήσετε το [ENTER]

θα πρέπει να πάρετε την απάντηση N1=64 N2=156. Συνεπώς ολόκληρη η εντολή του καθένα από τους τελευταίους κώδικες λειτουργίας (opcodes), μαζί με τη διεύθυνση 40000 από και προς την οποία γίνεται η φόρτωση, θα είναι:

	ASSEMBLER	DECIMAL	HEX
	LD A, (40000)	58 64 156	3A 40 9C
	LD (40000), A	50 64 156	32 40 9C
		BINARY	
	LD A, (40000)	00 111 010 0100 0000	1001 1100
	LD (40000), A	00 110 010 0100 0000	1001 1100

or for address 8;

	ASSEMBLER	DECIMAL	HEX
	LD A, (8)	58 8 0	3A 08 00
	LD (8), A	50 8 0	32 08 00
		BINARY	
	LD A, (8)	00 111 010 0000 1000	0000 0000
	LD (8), A	00 110 010 0000 1000	0000 0000

Μπορείτε, αν θέλετε, να το ελέγξετε και μόνοι σας αλλάζοντας το πρόγραμμα που τύπωνε τα «Α».

Η γραμμή 60 πρέπει να γίνει:

```
40 FOR N=42580 TO 42590:READ D:POKE N,D:A=A+D:NEXT
```

Και η δεύτερη εντολή γλώσσας μηχανής στη DATA της γραμμής 90 πρέπει να αλλάξει και από LD A, 65 να γίνει LD A, (8).

Η γραμμή συνεπώς θα γίνει:

```
90 DATA 6,255,58,8,0,205,90,187,16,251,201
```

Ο έλεγχος της γραμμής 40, εφ' όσον εξακολουθεί να υπάρχει, πρέπει να γίνει 1277.

Η σημαντικότερη γραμμή 30 πρέπει να γίνει:

```
40 FOR N=42580 TO 42590:READ D:POKE N,D:A=A+D:NEXT
```

κι αυτό γιατί τώρα υπάρχει ένα ακόμα byte του κώδικα που πρέπει να γίνει POKE στη μνήμη.

Αν προσθέσατε προηγουμένως τη γραμμή 21 (DEFSTR A) θυμηθείτε να την αφαιρέσετε!

Όταν τρέξετε τώρα το πρόγραμμα, αντί για «A» θα εμφανιστούν τα σύμβολα / (backslashes).

Ίσως το επόμενο χρησιμότερο πράγμα που αφορά τους καταχωρητές γενικής χρήσης, είναι ότι μπορούν να χρησιμοποιηθούν κατά ζεύγη, δηλαδή σαν BC, DE και HL. Όταν χρησιμοποιούνται κατ' αυτόν τον τρόπο μπορούν να θεωρηθούν σαν καταχωρητές των 16 bits.

Ενώ με ένα μόνο καταχωρητή περιορίζεστε σε αριθμούς που μπορούν να παρασταθούν από 8 bits, μ' άλλα λόγια μεταξύ 0 και 255, με ένα ζεύγος καταχωρητών μπορείτε να χρησιμοποιήσετε οποιονδήποτε αριθμό από το 0 μέχρι το 65535, γιατί τώρα έχετε στη διάθεσή σας 16 bits. Υπάρχει όμως κάποιο τμήμα που πρέπει να πληρώσετε για να μπορείτε να χρησιμοποιείτε ζεύγη καταχωρητών σαν να ήταν καταχωρητές των 16 bits.

Γνωρίζετε πως, όταν χρησιμοποιούνται μόνοι τους, ΟΛΟΙ οι καταχωρητές γενικής χρήσης, συμπεριλαμβανομένου του «A» ή οι συσσωρευτές, μπορούν να:

- 1) φορτωθούν από οποιονδήποτε άλλο καταχωρητή γενικής χρήσης
- 2) κρατήσουν έναν αριθμό απευθείας,

αλλά μόνο ο καταχωρητής «A» μπορεί να φορτώσει ή να φορτωθεί από μια αριθμημένη θέση μνήμης, χρησιμοποιώντας τις εντολές LD A, (nn) ή LD (nn), A.

Με τα ζεύγη καταχωρητών, όμως, δεν υπάρχει εντολή γλώσσας μηχανής τύπου LD rr, rr' (φόρτωσε ένα ζεύγος καταχωρητών με τα περιεχόμενα ενός άλλου ζεύγους). Πάντως, μπορείτε να φορτώσετε απευθείας έναν αριθμό σε οποιοδήποτε ζεύγος καταχωρητών.

Όσοι από σας διαθέτουν assembler, μπορούν να το κάνουν πολύ εύκολα. Ίσως να μην χρειάζονται καν να τους πούμε ποιά είναι η εντολή! Είναι η LD rr, nn. Το rr είναι ένα από τα ζεύγη καταχωρητών BC, DE ή HL και το nn είναι ένας ακέραιος των 16 bits.

Επομένως, η εντολή τοποθέτησης της τιμής 40000 στο ζεύγος καταχωρητών BC, θα είναι:

```
LD BC, 40000
```

ενώ για να φορτώσετε την τιμή 8 στο ζεύγος καταχωρητών, γράφεται:

LD HL, 8

Αν θυμάστε τη δομή της δυαδικής εντολής φόρτωσης ενός αριθμού σε καταχωρητή, τότε είναι σχεδόν σίγουρο ότι μπορείτε να μαντέψετε τα δύο πρώτα bits της δυαδικής εντολής φόρτωσης ενός ζεύγους καταχωρητών.

Αν δεν θυμάστε, τότε ρίξτε της μια ματιά.

Τα δύο πρώτα bits είναι 00.

Η υπόλοιπη εντολή κατασκευάζεται με ανάλογο τρόπο. Μετά από το 00 της αρχής, τα επόμενα δύο bits χρησιμοποιούνται για να καθορίσουν το ζεύγος καταχωρητών που θα φορτωθεί.

00 για το ζεύγος καταχωρητών BC.

01 για το ζεύγος καταχωρητών DE.

10 για το ζεύγος καταχωρητών HL.

(Αυτοί οι κώδικες των δύο bits είναι πάντοτε ίδιοι για το κάθε ζεύγος καταχωρητών; και χρησιμοποιούνται κάθε φορά που παρουσιάζεται κάποια εντολή για ζεύγος καταχωρητών).

Το επόμενο bit είναι 0 και τα τρία τελευταία bits είναι 001.

Επομένως, η πλήρης εντολή για το κάθε ζεύγος καταχωρητών είναι:

ASSEMBLER	DECIMAL	HEX	BINARY
LD BC, nn	1 n n	01 n n	00 000 001 n n
LD DE, nn	17 n n	11 n n	00 010 001 n n
LD HL, nn	33 n n	21 n n	00 100 001 n n

Ο υπολογισμός των n1 και n2 γίνεται με τον ίδιο τρόπο που υπολογίζονται στις εντολές LD A, (nn). Επομένως:

ASSEMBLER	DECIMAL	HEX	BINARY
LD BC, 40000	1 64 156	01 40 9C	00 000 001 0100 0000 1001 1100
LD HL, 8	33 8 0	21 08 00	00 100 001 0000 1000 0000 0000

Γνωρίζοντας πως να φορτώνετε ένα ζεύγος καταχωρητών με κάποιον αριθμό των 16 bits, αντιλαμβάνεστε πως χάνετε τον καιρό σας αν δεν μπορείτε να χρησιμοποιήσετε ένα φορτωμένο ζεύγος καταχωρητών. Μια από τις συνηθισμένες χρήσεις ενός ζεύγους καταχωρητών είναι η χρήση σαν μεταβλητής που δείχνει μια θέση μνήμης.

Νωρίτερα εξηγήθηκε πως το PEEK(8) στη BASIC χρησιμοποιείται για να βρούμε τα περιεχόμενα της θέσης μνήμης 8, καθώς και η ανάλογη εντολή γλώσσας μηχανής που χρησιμοποιεί τον καταχωρητή «A». Αυτός ο τύπος εντολής είναι πολύ περιοριστικός, ιδίως όταν πρόκειται να διαβαστεί ή να γραφτεί μια σειρά θέσεων μνήμης. Στη BASIC το πρόβλημα αντιμετωπίζετε με τη χρήση μιας με-

ταβλητής. Για παράδειγμα αν η μεταβλητή HL γινόταν ίση με 8 θα μπορούσατε να χρησιμοποιήσετε την εντολή PEEK (HL).

Στη γλώσσα μηχανής ισχύει το ίδιο, αλλά εδώ είναι που αρχίζει να υπερισχύει η ιδιοσυγκρασία της Z80 CPU.

Με ένα καταχωρητή γενικής χρήσης, αφ' ενός είναι αδύνατο να χρησιμοποιήσετε τους τύπους εντολών LD r, (nn), ή LD(nn),r, και αφ' ετέρου μόνο ο καταχωρητής HL μπορεί να χρησιμοποιηθεί σαν δείκτης (pointer). Γνωρίζετε ότι ο κωδικός 110b λείπει από τη σειρά των τριών bits που αντιπροσωπεύουν τους καταχωρητές γενικής χρήσης. Χρησιμοποιείται στο τέλος μιας εντολής LD r, n που αρχίζει με 00b και σημαίνει ότι το επόμενο byte (αυτό που βρίσκεται αμέσως μετά την εντολή) πρόκειται να χρησιμοποιηθεί σαν αριθμός.

Όταν χρησιμοποιείται στη μέση της εντολής LD r,n ή στις εντολές LD r, r', που αρχίζουν με 01b, έχει διαφορετική ερμηνεία. Θα ήταν αδύνατο για τον 110b να έχει το ίδιο νόημα αφ' όσον, όπως δείξαμενωρίτερα, ένας αριθμός έχει πάντοτε την ίδια τιμή. Όταν προσπαθήσαμε να αλλάξουμε την τιμή από τη BASIC εμφανίστηκε ένα syntax error. Υπάρχει όμως τρόπος για να αλλάξει ένας αριθμός, κι αυτός είναι όταν χρησιμοποιείται σαν διεύθυνση.

Όταν ο κωδικός 110b χρησιμοποιείται σε εντολή load στη θέση ενός κωδικού που αντιπροσωπεύει ένα καταχωρητή, η CPU θεωρεί ότι αναφέρεται στη θέση μνήμης της οποίας η διεύθυνση υπάρχει στο ζεύγος καταχωρητών HL.

Επομένως, για να φορτώσετε τον καταχωρητή D με ό,τι υπάρχει στη θέση μνήμης 4000 θα γράψετε:

ASSEMBLY	DECIMAL	HEX	BINARY
LD HL,40000	33 64 156	21 40 9C	00 100 001 0100 0000 1001 1100
LD D, (HL)	86	56	01 010 110

ή για να φορτώσετε τα περιεχόμενα της μνήμης στη διεύθυνση 8 στον καταχωρητή B:

ASSEMBLY	DECIMAL	HEX	BINARY
LD HL,B	33 8 0	21 08 00	00 100 001 0000 1000 0000 0000
LD B, (HL)	70	46	01 000 110

Παρατηρείστε τις παρενθέσεις γύρω από την HL στην εντολή της γλώσσας assembly, που σημαίνει «με τα περιεχόμενα της διεύθυνσης στο».

Είναι απόλυτα λογικό να αντιστρέψετε αυτή τη διαδικασία και αντί να φορτώσετε ένα καταχωρητή με τα περιεχόμενα μιας θέσης μνήμης, να φορτώσετε μια θέση μνήμης με τα περιεχόμενα ενός καταχωρητή. Η εντολή τότε γίνεται:

LD (HL),r

Όπως και με την LD r, (HL) μπορεί να χρησιμοποιηθεί οποιοσδήποτε καταχωρητής γενικής χρήσης ή ο «A» και ο δυαδικός κωδικός λειτουργίας είναι εύκολο να βρεθεί.

Για να φορτώσετε ένα καταχωρητή με τα περιεχόμενα της θέσης μνήμης με

διεύθυνση HL, ο κωδικός λειτουργίας (opcode) είναι:

[01] [ο κωδικός τριών bits του καταχωρητή] [110]

οπότε, για να φορτώσετε στη θέση μνήμης με διεύθυνση HL τα περιεχόμενα ενός καταχωρητή, γίνεται:

[01] [110] [ο κωδικός τριών bits του καταχωρητή]

Δεν ήταν ανάγκη να σας το πούμε, έτσι δεν είναι:

Αν αλλάξετε την εντολή DATA στη γραμμή 90 του BASIC προγράμματος σε:

90 DATA 33,8,0,70,58,8,0,205,90,187,16,251,201

αλλάξετε τον έλεγχο της γραμμής 40 σε 1127 και τον αριθμό μετά το TO της γραμμής 30 σε 42592 θα μπορέσετε να δείτε σε δράση τις εντολές LD B, (HL) και LD HL, nn. Το ανάλογο σε BASIC θα ήταν να αλλάξετε το τέλος της γραμμής 50 από B=255 σε HL=8:B=PEEK (HL).

Όταν τρέξει η ρουτίνα γλώσσας μηχανής θα φορτώσει το ζεύγος καταχωρητών HL με το 8 και κατόπιν θα φορτώσει το B με οτιδήποτε υπάρχει στη θέση HL. Η αρχή της ρουτίνας είναι τώρα:

ASSEMBLY	DECIMAL
LD HL,8	33 80
LD B, (HL)	70
LD A, (8)	58 80

Όταν χρησιμοποιείται ο κωδικός 110b στο τμήμα r του κωδικού λειτουργίας LD r,n, δίνοντας το δυαδικό κωδικό λειτουργίας 01 110 110, τότε δημιουργείται η εντολή σε γλώσσα assembly: LD (HL), η που θα φορτώσει στη διεύθυνση HL της μνήμης τον αριθμό που ακολουθεί την εντολή.

Με τον καταχωρητή «A» μπορείτε να χρησιμοποιήσετε οποιοδήποτε ζεύγος καταχωρητών σαν μεταβλητή, ή δείκτη. Οι εντολές γλώσσας assembly είναι προφανείς. Είναι LD A, (rr) ή LD (rr, A), όπου rr είναι ένα ζεύγος καταχωρητών. Για παράδειγμα, είναι παραδεκτό να γράψετε χρησιμοποιώντας τον καταχωρητή «A»:

LD DE,8
LD A, (DE)

Ο κωδικός λειτουργίας της LD A, (HL) έχει ήδη εξηγηθεί και θα παρατηρήσατε χωρίς αμφιβολία ότι έχουν χρησιμοποιηθεί όλοι οι δυνατοί συνδυασμοί των εντολών που αρχίζουν με 01. Πρέπει να χρησιμοποιηθεί μια διαφορετική κατασκευή για τις εντολές LD A, (BC) LD A, (DE) LD (BC), A και LD (DE), A.

Μια ένδειξη για τον τρόπο κατασκευής αυτών των εντολών μπορεί να βρεθεί στους κωδικούς λειτουργίας των LD A, (nn) και LD (nn), A, και μια άλλη υπάρχει στους κωδικούς των ζευγών καταχωρητών.

Έχετε προσέξει πως οι κωδικοί των τριών bits στους καταχωρητές γενικής χρήσης μοιράζονται τα δύο σημαντικότερα bits με τους κωδικούς των 2 bits των ζευγών καταχωρητών που τους χρησιμοποιούν. Ο B έχει κωδικό 000 και ο C 001, ο BC έχει κωδικό 00. Ο D έχει κωδικό 010 και ο E 011, ο DE είναι 01. Ο H είναι 100 και ο L 101, δίνοντας HL10.

Ο κωδικός λειτουργίας της LD A,(nn) στο δυαδικό είναι 00111010 και έχει τονιστεί ότι, στις εντολές φόρτωσης ενός αριθμού σε ζεύγος καταχωρητών, τα bits 5 και 4 λένε στη CPU ποιό ζεύγος καταχωρητών να χρησιμοποιήσει. Ο μοναδικός κωδικός λειτουργίας που λείπει είναι ο 11. Προσέξτε τώρα! Τι έχουμε στα bits 5 και 4 της LD A,(nn); και ποιά υποθέσετε ότι θα είναι η δυαδική μορφή της εντολής LD A, (BC);

Συγχαρητήρια σε όσους το βρήκαν, παρ' όλο που δεν ήταν και τόσο δύσκολο. Ο δυαδικός κωδικός λειτουργίας της LD A, (BC) είναι 00 001 010, ενώ της LD A, (DE) είναι 00 011 010.

Ο κωδικός λειτουργίας της LD A, (HL) δεν είναι 00 101 010, είναι 01 111 110 και εξηγήθηκε παραπάνω, οπότε τι νομίζετε ότι κάνει η 00 101 010; Περιμένετε λίγο και θα τα καταλάβετε όλα. Πρώτα όμως πρέπει να δοθεί απάντηση στο θέμα των κωδικών λειτουργίας της LD (rr), A, και προσέξτε! Είναι το ίδιο πολύπλοκοι με τους κωδικούς λειτουργίας της LD A, (rr).

Η LD (nn), A στο δυαδικό είναι 00 110 010. Πάλι ο 11b που λείπει από την ομάδα των κωδικών για τα ζεύγη καταχωρητών, υπάρχει στα bits 5 και 4. Εδώ, όπως και στις προηγούμενες εντολές, για να αλλάξετε το (nn) σε (BC) ή (DE), το μόνο που πρέπει να κάνετε είναι να αλλάξετε το 11b σε 00b ή 01b. Ο κωδικός λειτουργίας της LD (BC),A είναι επομένως 00 000 010, ενώ της LD (DE),A 00 010 010. Αλλά, όπως και στις προηγούμενες εντολές, ο κωδικός 10b του HL που δημιουργεί τον κωδικό λειτουργίας 00 100 010 δεν σημαίνει LD (HL),A.

Όλοι οι παραπάνω κωδικοί λειτουργίας, που φορτώνουν ή φορτώνονται από μια θέση μνήμης η οποία προσδιορίζεται από κάποιο ζεύγος καταχωρητών, είναι εντολές του ενός byte (single byte instructions). Οι κωδικοί 00 100 010 και 00 101 010, που χρησιμοποιούν τον κωδικό 10 του HL, κατασκευάζονται από τρία bytes. Το πρώτο είναι, βέβαια, ο κωδικός λειτουργίας. Το 00 100 010 ή το 00 101 010, καθώς και τα επόμενα δύο bytes αποτελούν το operand. (Θυμηθείτε, το operand είναι οι πληροφορίες που χρειάζονται για να μπορεί να εκτελεστεί η εντολή του κωδικού λειτουργίας) Αυτοί οι δύο κωδικοί λειτουργίας χρησιμοποιούνται είτε για να φορτώσει ο καταχωρητής HL είτε για να φορτωθεί με τη θέση μνήμης η οποία βρίσκεται στα επόμενα δύο bytes. Ο τρόπος που λειτουργούν είναι ίδιος με της εντολής LD (nn), A που μεταβιβάζει τα περιεχόμενα του καταχωρητή A, και της εντολής LD A, (nn) που φορτώνει τον καταχωρητή A με μια θέση μνήμης που καθορίζεται από το (nn).

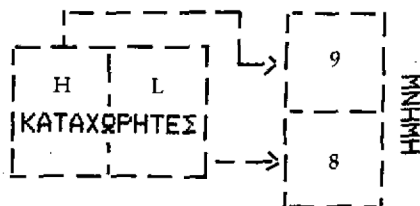
Ακολουθούν οι πλήρεις εντολές σε γλώσσα assembly και σε δυαδική μορφή.

ASSEMBLER	BINARY
LD HL, (nn)	00 101 010 n n
LD (nn), HL	00 100 010 n n

Υποθέστε ότι η διεύθυνση (nn) είναι 8, όπως έγινε και με τις προηγούμενες εντολές φόρτωσης. Αυτό θα κάνει τον n1 0000 1000 και τον n2 0000 0000 και στα δύο παραπάνω παραδείγματα δυαδικής μορφής, ενώ οι assembly εντολές θα γίνουν: LD HL, (8) και LD (8), HL

Στην πρώτη περίπτωση το ζεύγος καταχωρητών HL θα φορτωθεί με έναν αριθμό των 16 bits από την καθοριζόμενη διεύθυνση μνήμης, και στη δεύτερη περίπτωση ο αριθμός των 16 bits που βρίσκεται στον HL θα φορτωθεί στην καθοριζόμενη διεύθυνση μνήμης.

Υπάρχει όμως το εξής πρόβλημα: μόνο μια διεύθυνση μνήμης καθορίστηκε από το operand, και μια θέση μνήμης περιέχει μόνο 8 bits, οπότε πώς μπορεί ένας αριθμός των 16 bits να συμπιεστεί σε 8 bits; Η σύντομη απάντηση είναι πως δεν μπορεί. Η CPU αντιμετωπίζει αυτό το πρόβλημα χρησιμοποιώντας τη θέση μνή-



Σχήμα 5.6

μης που καθορίζεται για το λιγότερο σημαντικό byte, και την αμέσως από πάνω θέση για το σημαντικότερο byte. Λογικά, γιατί υπολογιστές και λογική ταυτίζονται, το σημαντικότερο byte προέρχεται από τον καταχωρητή H και το λιγότερο σημαντικό byte από τον καταχωρητή L, και το λιγότερο σημαντικό byte βρίσκεται στη χαμηλότερη θέση μνήμης.

Η CPU αρχίζει πάντοτε από κάτω και πηγαίνει προς τα πάνω και σ' αυτό οφείλεται το ότι, όταν φορτώνεται στη μνήμη γλώσσα μηχανής, όλοι οι αριθμοί των 16 bits αναστρέφονται. Αν θυμάστε αυτά που είπαμε παραπάνω, θα μπορούσατε να αποφύγετε τα λάθη όταν γράφετε σε γλώσσα μηχανής. Όταν χρησιμοποιείτε assembler, τότε όλες οι αναστροφές αριθμών γίνονται από αυτόν οπότε, δεν χρειάζεται να αλλάξετε τους αριθμούς από την κανονική μορφή τους.

Οι τελευταίες εντολές φόρτωσης που θα εξηγηθούν σ' αυτό το κεφάλαιο εργάζονται όπως και οι προηγούμενες, αλλά αναφέρονται στα ζεύγη καταχωρητών BC και DE. Χρησιμοποιούνται λιγότερο από τις εντολές που χρησιμοποιούν το ζεύγος καταχωρητών HL γιατί χρειάζονται δύο bytes μνήμης για να αποθηκεύσουν την εντολή. Σε γλώσσα assembly, οι εντολές αυτές είναι όπως ακριβώς θα περιμένατε:

LD BC, (nn) LD DE, (nn) LD (nn), BC LD (nn), DE

Αν επιστρέψετε νοερά στο Κεφάλαιο 2, μπορεί να θυμηθείτε ότι, στην αναλογία των Ελληνικών και Κινέζικων λέξεων, αποδείχτηκε πως η πρόσθεση μιας ακόμη λέξης μπορεί να αλλάξει το νόημα ολόκληρης της λέξης. Λοιπόν, οι κωδικοί λει-

τουργίας των παραπάνω τεσσάρων εντολών έχουν το πρόθεμα (στο HEX) ED. (11101101b ή 237 στο δεκαδικό, αλλά το ED είναι ευκολότερο να το θυμάστε). Όλα τα σχόλια που έχουν σχέση με τις εντολές που μόλις εξηγήθηκαν, χρησιμοποιώντας το ζεύγος καταχωρητών HL, ισχύουν και γι' αυτές τις εντολές. Καλό θα ήταν να χρησιμοποιείτε, όπου μπορείτε, εντολές που χρησιμοποιούν το ζεύγος καταχωρητών HL, γιατί ο κωδικός λειτουργίας τους απαιτεί τη μισή ποσότητα μνήμης. Οι πραγματικοί κωδικοί λειτουργίας είναι οι ακόλουθοι:

ASSEMBLER	DECIMAL	HEX	BINARY
LD BC, (nn)	237 75, n n	ED 4B n n	1110 1101 01 001 011 n n
LD DE, (nn)	237 91 n n	ED 5B n n	1110 1101 01 011 011 n n
LD (nn), BC	237 67 n n	ED 43 n n	1110 1101 01 000 011 n n
LD (nn), DE	237 83 n n	ED 53 n n	1110 1101 01 010 011 n n

Σχήμα 5.7

Στο τέλος αυτού του κεφαλαίου υπάρχει μια περίληψη όλων των εντολών που αρχίζουν με LD, ενώ στο Παράρτημα υπάρχει μια γραφική αναπαράσταση της περίληψης της Zilog.

Ο καταχωρητής Program Counter

Κάθε φορά που μπαίνει σε λειτουργία ο υπολογιστής σας, εκτός κι αν σταματήσετε τη CPU για κάποιο λόγο, ο καταχωρητής program counter (μετρητής προγράμματος, PC) αρχίζει να δουλεύει, έχοντας σαν μοναδικό σκοπό του να φτάσει στην κορυφή και να ξαναρχίσει. Ο σκοπός του είναι να παρακολουθεί το πρόγραμμα που τρέχει, και θα περιέχει πάντοτε τη διεύθυνση μνήμης της εντολής του προγράμματος που εκτελείται εκείνη τη στιγμή. Όταν μπαίνει σε λειτουργία ο υπολογιστής, ο PC περιέχει υποχρεωτικά τη διεύθυνση 0, ώστε η πρώτη εντολή να έρθει από εκεί. Αυτό σημαίνει ότι με το άνοιγμα του υπολογιστή μπορεί να εκτελεστεί αυτόματα ένα πρόγραμμα που θα φέρει τον υπολογιστή σε μια γνωστή κατάσταση. Αυτό είναι γνωστό σαν Cold Start (Ψυχρό Ξεκίνημα) ή Early Morning (Χάραμα) ή Wake Up (Ξύπνημα), και στον Amstrad χωρίς επεκτάσεις (unexpanded) έχει σαν αποτέλεσμα να τον θάλει στο mode προγραμματισμού σε BASIC και να προυσιάσει το copyright της Amstrad και της Locomotive.

Όπως αναφέρθηκε ήδη, κάθε φορά που μπαίνει σε λειτουργία ο υπολογιστής τρέχει ένα πρόγραμμα αλλά, εφ' όσον δεν γνωρίζει τι πρόκειται να του ζητηθεί να κάνει κατόπιν, είναι ουσιαστικά να μπορούμε να ελέγχουμε το μετρητή προγράμματος. Φανταστείτε μια κατάσταση όπου η μνήμη του υπολογιστή θα αντιστοιχούσε στα πλήκτρα του πιάνου, και το μόνο που θα μπορούσατε να κάνετε θα ήταν να παίζετε διαδοχικά τις νότες από τη μια άκρη μέχρι την άλλη, και μετά να ξαναρχίζετε. Θα μπορούσατε να το κάνετε πιο ενδιαφέρον ρυθμίζοντας τις χορδές του πιάνου ώστε να παίζουν κάποια σύντομη μελωδία, που όμως σύντομα θα γινόταν

βαρετή, ενώ θα έπρεπε να ξαναρυθμιστούν όλες οι χορδές κάθε φορά που θα χρειάζοσαστε μια νέα μελωδία. Το παραπάνω μοιάζει με αυτό που θα συνέβαινε αν ο μετρητής προγράμματος του υπολογιστή δεν μπορούσε να αλλάξει. Η ρύθμιση των χορδών αντιστοιχεί στο πρόγραμμα, και τα πλήκτρα του πιάνου στις θέσεις μνήμης.

Ευτυχώς μπορείτε να αλλάξετε τον PC, και στη BASIC το κατορθώνεται με τις εντολές GOTO, GOSUB και RETURN. Η GOTO υποχρεώνει την εκτέλεση του προγράμματος να πηδήξει στη γραμμή που δίνετε, και GOSUB καλεί μια υπορουτίνα στη γραμμή που διαλέξατε. Όταν η υπορουτίνα τελειώσει, ο έλεγχος επιστρέφεται (με την εντολή RETURN) στο κυρίως πρόγραμμα, που συνεχίζει με την εντολή που βρίσκεται μετά το GOSUB.

Οι εντολές γλώσσας μηχανής που είναι ανάλογες προς εκείνες της BASIC κάνουν ακριβώς την ίδια δουλειά αλλά με διαφορετικά ονόματα, όπως γίνεται με τη συμβολική εντολή LD, που περιγράφει την εργασία που κάνει και η ανάλογη εντολή BASIC (LD σημαίνει load-φορτώνω).

Μπορείτε να μαντέψετε ποιές είναι οι εντολές σε γλώσσα μηχανής; Η GOTO γίνεται JUMP (πηδω), η GOSUB γίνεται CALL (καλώ) και η RETURN δεν αλλάζει. Η εντολή RETURN συντομεύεται σε RET όταν χρησιμοποιούμε τον assembler, ενώ η CALL γράφεται ολόκληρη. Η JUMP είναι περισσότερο πολύπλοκη γι' αυτό θα εξηγηθεί αργότερα σε τούτο το κεφάλαιο, αφού ορισθούν οι χρήσεις των CALL και RET.

Οι εντολές CALL και RET αναπαράγουν με ακρίβεια τις ανάλογές τους στη BASIC αλλά, επειδή η γλώσσα μηχανής δεν διαθέτει αριθμούς γραμμής, η CALL αναφέρεται στη διεύθυνση που περιέχει την αρχή της πρώτης εντολής της υπορουτίνας.

Η εντολή CALL κατασκευάζεται από 3 bytes, το πρώτο είναι ο κωδικός λειτουργίας (opcode), και τα επόμενα δύο είναι η διεύθυνση της υπορουτίνας που καλείται. Τα δύο bytes που περιέχουν τη διεύθυνση γίνονται με τη συνηθισμένη μέθοδο του Z80, το λιγότερο σημαντικό byte πρώτο. Αν χρησιμοποιείτε assembler, θα κάνει αυτός τον υπολογισμό στη θέση σας, όπως τον έκανε και για τις εντολές LD.

Οι κωδικοί λειτουργίας των εντολών CALL και RET είναι οι εξής:

ASSEMBLY	DECIMAL	HEX	BINARY
CALL nn	205 n n	CD n n	11 001 101 n n
RET	201	C9	11 001 001

Αν κοιτάξετε το πρόγραμμα που πληκτρολογήσατε στο Κεφάλαιο 2, ή αν βρίσκεστε προστά στον υπολογιστή σας και το έχετε εισάγει, κάνετε list στη γραμμή 90. Θα δείτε ότι οι αριθμοί που βρίσκονται αμέσως μετά από εκείνους με τους οποίους πειραματιστήκατε, είναι οι 205, 90, 187.

Τώρα θα πρέπει να ξέρετε τι λένε οι αριθμοί αυτοί στη CPU. Ο 205 είναι μια εντολή CALL και η διεύθυνση που καλείται μπορεί να υπολογιστεί προσθέτοντας τον επόμενο αριθμό στον 256* τον τρίτο.

187*256=47872. 47872+90=47962 ή BB5A h.

Έτσι ξέρετε τώρα ότι η αρχή της ρουτίνας που γράψατε πρώτα φορτώνει στον καταχωρητή A τον κωδικό του χαρακτήρα που θα τυπωθεί, κατόπιν φορτώνει στον καταχωρητή B τον αριθμό που λέει πόσες φορές θα τυπωθεί ο χαρακτήρας και μετά καλεί (με την εντολή CALL) μια υπορουτίνα που αρχίζει στη διεύθυνση 47872 (BB5Ah). Αυτή η υπορουτίνα είναι τμήμα του λειτουργικού συστήματος του υπολογιστή, και είναι πιθανώς η υπορουτίνα που θα χρησιμοποιήσετε περισσότερο από κάθε άλλη. Η AMSOFT την έχει ονομάσει TXT OUTPUT, και θα τυπώσει το χαρακτήρα του οποίου ο κωδικός βρίσκεται στον καταχωρητή «A» στο τρέχον window, σ' εκείνη τη θέση του window όπου βρίσκεται εκείνη τη στιγμή ο δρομέας. Η υπορουτίνα θα υπακούσει επίσης στους κωδικούς ελέγχους, οι οποίοι εξηγούνται στο Κεφάλαιο 9 του βιβλίου Amstrad User's Instructions.

Για να έχετε ένα σύντομο παράδειγμα απόκρισης σε κωδικό ελέγχου αλλάξετε τη γραμμή 90 σε: 90 DATA 62,7,205,90,187,201, καθώς και τον αριθμό που βρίσκεται μετά το TO της γραμμής 30 σε 42585. Αν η γραμμή 40 εξακολουθεί να υπάρχει, τότε ο έλεγχος πρέπει να αλλάξει σε 752. Το πρόγραμμα σε γλώσσα assembly γίνεται τώρα:

```
LD A, 7
CALL 47962
RET
```

Όταν τρέξετε αυτό το πρόγραμμα γλώσσας μηχανής, θα ηχήσει το κουδούνι, εκπέμποντας από τον υπολογιστή ένα beep. Αν δεν ακούσετε τίποτα, δοκιμάστε πάλι δυναμώνοντας τον ήχο και πληκτρολογώντας CALL 42580 [ENTER]. Αυτό θα καλέσει απευθείας τη γλώσσα μηχανής, αντί να καλείται μέσω του προγράμματος σε BASIC.

Αντίθετα με τις εντολές LD δεν μπορείτε να καθορίσετε μια διεύθυνση χρησιμοποιώντας σαν δείκτη ένα ζεύγος καταχωρητών. Η διεύθυνση που θα κληθεί πρέπει να βρίσκεται πάντοτε στα δύο bytes που ακολουθούν τον κώδικα λειτουργίας της CALL. Το RETURN στο τέλος μιας υπορουτίνας θα βρίσκεται στη διεύθυνση που είναι αμέσως μετά τα τρία bytes της CALL (ένα byte για τον κωδικό λειτουργίας και δύο για το operand).

Για να μπορεί να εκτελέσει τη RETURN η CPU πρέπει να γνωρίζει από που καλέσατε την υπορουτίνα, και αυτό το κατορθώνει χρησιμοποιώντας τη λεγόμενη Machine Stack (Στοιβα Μηχανής), η για συντομία stack (στοίβα). Ακολουθεί μια σύντομη εξήγηση του τι είναι και πως χρησιμοποιείται όταν το πρόγραμμα συναντάει μια εντολή CALL ή ένα RETURN. Η χρήση της στοίβας θα συζητηθεί με λεπτομέρειες στο Κεφάλαιο 9.

Η στοίβα μπορεί να συγκριθεί με ένα καρφί στην οροφή, και το κάθε byte πληροφορίας που αποθηκεύεται σ' αυτήν, μπορεί να θεωρηθεί σαν ένα κομμάτι χαρτί. Όταν το χαρτί τοποθετείται στη στοίβα, η στοίβα μεγαλώνει προς τα κάτω, και η μόνη πληροφορία που μπορεί να αφαιρεθεί από τη στοίβα είναι το κομμάτι που βρίσκεται κάτω - κάτω. Για να πάρετε κάτι που δεν βρίσκεται στην κατώτερη θέ-

ση, πρέπει προηγουμένως να αφαιρέσετε αυτά που υπάρχουν κάτω.

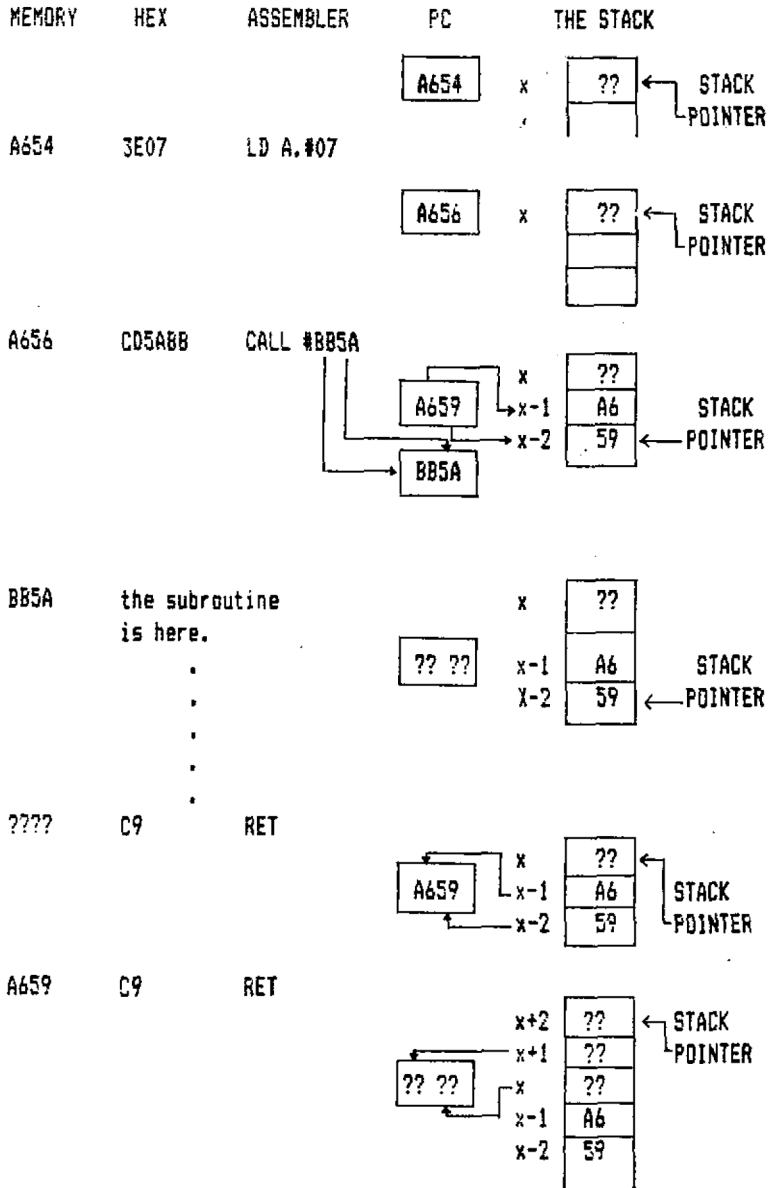
Η στοίβα καταλαμβάνει μια περιοχή της μνήμης, και η διεύθυνση της βάσης της στοίβας υπάρχει πάντοτε στον καταχωρητή των 16 bits Stack Pointer (Δείκτης Στοίβας). Η περιοχή μνήμης που προορίζεται για τη στοίβα πρέπει να προστατεύεται ώστε να μην χρησιμοποιείται από το πρόγραμμα, γιατί αν υποστεί άλλες επεμβάσεις είναι σχεδόν σίγουρο ότι το πρόγραμμα θα χαθεί. Συνήθως αυτό το κατορθώνει τοποθετώντας τη στοίβα στην κορυφή μίας μεγάλης περιοχής ελεύθερης μνήμης, γιατί έτσι η στοίβα έχει πάρα πολύ χώρο για να μεγαλώσει προς τα κάτω χωρίς να εισέρχεται σε απαγορευμένες περιοχές.

Κάθε φορά που το πρόγραμμα συναντάει μιαν εντολή CALL, η CPU, αφού βρει τον προορισμό της CALL, τοποθετεί την τρέχουσα διεύθυνση του PC (τη διεύθυνση της αμέσως επόμενης εντολής που θα εκτελεστεί) στη στοίβα, ενώ στον PC τοποθετεί τη διεύθυνση της υπορουτίνας. Έτσι η επόμενη εντολή θα έρθει από τη διεύθυνση της CALL, οπότε εκτελείται η υπορουτίνα. Στο τέλος της υπορουτίνας, όταν εκτελείται η RET, η CPU παίρνει τη διεύθυνση επιστροφής από τη στοίβα και την τοποθετεί στον PC, αναγκάζοντας κατ' αυτόν τον τρόπο να συνεχιστεί η εκτέλεση του προγράμματος στη διεύθυνση που θρυσκόταν αμέσως μετά το αρχικό CALL.

Στα διαγράμματα της άλλης σελίδας φαίνεται η σειρά των εργασιών όταν καλείται μια υπορουτίνα και όταν εκτελείται η εντολή RETurn. Το listing του assembler για το πρόγραμμα δίνεται στο Σχήμα 5.8. Η πρώτη στήλη είναι η διεύθυνση μνήμης της αρχής της εντολής, στο Hex, η δεύτερη στήλη είναι ο Hex κωδικός της εντολής και η τρίτη στήλη είναι το πραγματικό listing του assembler, όπου και σ' αυτό έχουν χρησιμοποιηθεί δεκαεξαδικοί αριθμοί. Ο λόγος για τον οποίο χρησιμοποιούνται δεκαεξαδικοί αριθμοί είναι ότι δείχνουν με μεγαλύτερη σαφήνεια αυτό που συμβαίνει, γιατί το καθένα byte μνήμης ή ο καταχωρητής του ενός byte μπορούν να περιέχουν ένα διψήφιο δεκαεξαδικό αριθμό.

Το παράδειγμα που χρησιμοποιείται είναι το προηγούμενο πρόγραμμα που κάλυπτε το κουδούνι να ηχήσει.

Κάθε φορά που τοποθετείται κάτι στη στοίβα, αυτή μεγαλώνει προς τα κάτω κατά δύο bytes, ενώ κάθε φορά που αφαιρείται κάτι από τη στοίβα, αυτή μικραίνει κατά δύο bytes. Επομένως είναι πολύ σημαντικό να βεβαιωθείτε ότι ένα πρόγραμμα δεν τοποθετεί στη στοίβα περισσότερα από όσα αφαιρεί. Το τελευταίο θα μπορούσε να κάνει τη στοίβα είτε να μεγαλώσει τόσο πολύ προς τα κάτω ώστε να αρχίσει να χρησιμοποιεί τη μνήμη που προορίζεται για το πρόγραμμα, ή να έχει στη βάση της λαθεμένες πληροφορίες όταν πρόκειται να αφαιρεθεί κάτι. Μεταξύ των πληροφοριών που τοποθετούνται στη στοίβα και εκείνων που χρειάζονται πάλι, πρέπει να υπάρχει ίσος αριθμός πραγμάτων που τοποθετούνται και αφαιρούνται από τη στοίβα. Αυτό είναι ζωτικής σημασίας και πρέπει να τονιστεί πάρα πολύ, ιδίως γιατί υπάρχουν εντολές που εξηγούνται αργότερα, διαφορετικές από τις CALL και RETurn, οι οποίες χρησιμοποιούν τη στοίβα για αποθήκευση πληροφοριών. Η πιο συνηθισμένη αιτία απώλειας ενός προγράμματος είναι τα λάθη που γίνονται κατά τη χρησιμοποίηση της στοίβας. Αντίθετα με τη BASIC, όταν το πρόγραμμα «κολλήσει», το μόνο που μπορείτε συνήθως να κάνετε είναι να κλείσετε και να ξαναοιζέτε τον υπολογιστή σας.



Σχήμα 5.8

Jumps

Η εντολή JUMP έχει δύο τύπους. Ο πρώτος που θα εξηγηθεί μιμείται με απόλυτη σχεδόν ακρίβεια την GOTO της BASIC. Σ' αυτόν τον τύπο εντολής ο απόλυτος προορισμός δίνεται μετά την εντολή. Θεωρείστε για παράδειγμα την εντολή της BASIC GOTO 100. Πρέπει να υπάρχει στο πρόγραμμα μια γραμμή με αριθμό «100» και με την εκτέλεση της GOTO 100 ο έλεγχος μεταφέρεται στη γραμμή 100. Στη γλώσσα μηχανής, όπως γνωρίζετε, δεν υπάρχουν αριθμοί γραμμής, έτσι αντί ο έλεγχος να μεταφερθεί σε αριθμό γραμμής, μεταφέρεται σε μια διεύθυνση.

Η συμβολική εντολή είναι JP, συντομογραφία του JUMP, και συνήθως ακολουθείται από δύο bytes που δίνουν τη διεύθυνση προορισμού. Το πήδημα (jump) στην πραγματικότητα γίνεται κατά τον ίδιο ακριβώς τρόπο που η εντολή CALL μεταφέρει τον έλεγχο σε μια υπορουτίνα, με τη διαφορά ότι, επειδή δεν υπάρχει εντολή RETURN, δεν χρησιμοποιείται η στοίβα. Ο τύπος αυτός έχει σαν συμβολική εντολή τη JPnn, και επιτρέπει το πήδημα σε οποιαδήποτε θέση της συγκεκριμένης σελίδας μνήμης. Η εντολή περιλαμβάνει τρία bytes, με τη σειρά που έχουν και στην εντολή CALL, αλλά το πρώτο byte, αντί να είναι 11 001 101 όπως στην CALL, γίνεται 11 000 011 για την JP. Αν η CALL του τελευταίου προγράμματος αλλάξει σε JP θα γίνει:

ASSEMBLER	DECIMAL	HEX	BINARY
JP 47962	195 90 187	C3 5A BB	11 000 011 0101 1010 1011 1011

Η εντολή JP μπορεί επίσης να χρησιμοποιηθεί σε συνδυασμό με το ζεύγος καταχωρητών HL που θα περιέχει τη διεύθυνση στην οποία θα γίνει το πήδημα. Σ' αυτή τη περίπτωση η εντολή χρειάζεται ένα μόνο byte και το πήδημα γίνεται στη διεύθυνση που περιέχεται στο ζεύγος καταχωρητών HL. Αν θυμάστε τον τρόπο με τον οποίο αντιπροσωπεύεται στη γλώσσα assembly η έκφραση «περιέχεται στο» θα πρέπει να ήδη να γνωρίζετε τη συμβολική εντολή. Ολόκληρος ο κωδικός λειτουργίας δίνεται παρακάτω:

ASSEMBLER	DECIMAL	HEX	BINARY
JP (HL)	233	E9	11 101 001

Αυτή είναι μια από τις πιο χρήσιμες εντολές της Z80 CPU και χρησιμοποιείται συχνά σε συνδυασμό με ένα CALL για να δημιουργηθεί το ανάλογο της εντολής ON GOSUB της BASIC. Το τελευταίο εξηγείται στο επόμενο κεφάλαιο.

Τα περισσότερα πήδηματά γίνονται σε μια εντολή που βρίσκεται πολύ κοντά στη διεύθυνση από την οποία γίνεται το πήδημα, και η Z80 CPU διαθέτει μια εντολή που επιτρέπει το πήδημα σε μια θέση σχετική (relative) προς τη διεύθυνση που υπάρχει στο μετρητή προγράμματος (program counter). Η εντολή αυτή καλείται Jump Relative. Στη γλώσσα assembly συντομοεύεται σε JR.

Η εντολή JR αποτελείται από δύο μόλις bytes. Το πρώτο είναι ο κωδικός λειτουργίας και το δεύτερο, η απόσταση του πηδήματος από τη διεύθυνση στην οποία ο μετρητής προγράμματος περιμένει να βρει την επόμενη εντολή, γραμμένη

σε «συμπλήρωμα του 2», όπως εξηγείται στο Κεφάλαιο 3. Αυτό επιτρέπει να γίνει ένα πήδημα από +127 έως -128 σε σχέση με τον PC. Ο κωδικός λειτουργίας της JR είναι:

ASSEMBLER	DECIMAL	HEX	BINARY
JR n	24 n	18 n	00 011 000 n

Συνήθως όταν χρησιμοποιείται ο assembler δεν χρειάζεται ο υπολογισμός του μήκους της JR. Στη θέση του χρησιμοποιείται μια LABEL, είτε δίνονται στη LABEL (με τη βοήθεια της EQU) την τιμή της διεύθυνσης στην οποία θα γίνει το πήδημα ή, συνηθέστερα, ορίζοντας τη LABEL μέσα στο πρόγραμμα (η EQU έχει εξηγηθεί στο Κεφάλαιο 3).

Για να σας βοηθήσουμε να το καταλάβετε αυτό, θεωρήστε το παρακάτω παράδειγμα:

ADDRESS	LABEL	ASSEMBLER	DECIMAL	HEX
42580 (A654)		LD A.7	62 7	3E 07
42582 (A656)	PRINT:	CALL 47962	205 90 187	CD 5A BB
42585 (A659)		LD A.65	62 65	3E 41
42587 (A65B)		JR PRINT	24 249	18 F9

Το 249 μετά τον κωδικό λειτουργίας της εντολής-JR λέει στη CPU να μεταφέρει την εκτέλεση σε μια διεύθυνση που απέχει -7 από εκείνη που βρίσκεται στον PC. Ο PC δείχνει ήδη την επόμενη εντολή γιατί το διάβασμα της UR ή τελείωσε, οπότε περιέχει τη διεύθυνση 42589. $42589 - 7 = 42582$. Το πήδημα επομένως θα γίνει στην αρχή της εντολής CALL.

ADDRESS	LABEL	ASSEMBLER	DECIMAL	HEX
42580 (A654)		JR GO	24 5	18 05
42582 (A656)		LD A.7	62 7	3E 07
42584 (A658)	PRINT:	CALL 47962	205 90 187	CD 5A BB
42587 (A65B)	GO:	LD A.65	62 65	3E 41
42589 (A65D)		JR PRINT	24 249	18 F9

Αντί να ηχήσει το κουδούνι και να ακολουθήσουν τα «A» μέχρι να κάνετε reset στον υπολογιστή, η JR GO θα υποχρεώσει τον PC να πηδήξει την LD A, 7 και την CALL 47962, και το πρώτο πράγμα που θα τυπωθεί θα είναι ένα «A». Παρατη-

ρήστε πως το πήδημα είναι μόνο 5 με το οποίο αγνοούνται τα πέντε byte μετά την JR GO.

Μπορείτε να χρησιμοποιείτε σχετικές διευθύνσεις στον assembler τις οποίες υπολογίζετε, αλλά δεν χρειάζεται εκτός κι αν έχετε μεγάλα προγράμματα οπότε είναι σημαντικό να εξοικονομείτε το χώρο που καταναλώνουν οι labels. Η Highsoft Denrac μπερδεύει λίγο τα πράγματα, με τον τρόπο που λειτουργεί ο assembler GENSA3. Οι αποστάσεις των jumps υπολογίζονται σε σχέση με τον μετρητή του assembler όχι του PC. Αυτό εξηγείται στη σελίδα 2.6 του εγχειρίδιου οδηγιών (manual) που συνοδεύει την κασέτα Denrac. Στην ουσία ο μετρητής θα δείχνει την αρχή της εντολής JR όταν γίνει ο υπολογισμός, και επομένως θα χρειαστεί να προσθέσετε 2 στην απόσταση που υπολογίζεται κατά τον κανονικό τρόπο, ώστε να γίνει το σωστό πήδημα. Στο μετρητή αυτό απευθυνόμαστε χρησιμοποιώντας το σύμβολο \$. Η εντολή JR PRINT στην παραπάνω ρουτίνα θα γραφόταν:

JRS-5

και όχι σαν JR-7 που φαίνεται λογικότερο.

Ο assembler Picturesque, που θα είναι διαθέσιμος σύντομα, θα χρησιμοποιεί τη τυπική μέθοδο υπολογισμού Z80, και η εντολή που τη χρησιμοποιεί, γίνεται επομένως η άναμενόμενη JR-7.

Αν, όπως θα συμβαίνει συχνά, εκτός αν ο χώρος έχει προτεραιότητα, χρησιμοποιείτε labels τίποτε από τα προηγούμενα δεν θα σας προβληματίσει, οπότε δεν χρειάζεται να ανησυχείτε για το ποιόν assembler χρησιμοποιείτε.

Οποιοδήποτε πήδημα (relative jump) σε κλάδο που υποδεικνύεται από label, θα γίνεται στην εντολή που βρίσκεται αμέσως μετά τη label.

Υπάρχει μια τελευταία εντολή που θα περιγραφεί σε τούτο το κεφάλαιο, και είναι μια από τις απλούστερες αλλά και χρησιμότερες εντολές που διαθέτουμε. Επιτρέπει στα περιεχόμενα του ζεύγους καταχωρητών DE να ανταλλάγουν με τα περιεχόμενα του ζεύγους καταχωρητών HL. Αυτό είναι εξαιρετικά χρήσιμο εφ' όσον το ζεύγος καταχωρητών HL χρησιμοποιείται για να δείχνει μια διεύθυνση μνήμης, αλλά κατόπιν πρέπει να χρησιμοποιηθεί για άλλο σκοπό. Στο επόμενο κεφάλαιο θα καταλάβετε γιατί είναι πιθανή η δημιουργία τέτοιων καταστάσεων. Όπως περιμένατε ίσως, η συμβολική εντολή είναι: EX DE, HL όπου το EX προέρχεται από το EXchange (ανταλλαγή). Οι πλήρεις κωδικοί λειτουργίας δίνονται παρακάτω.

ASSEMBLER	DECIMAL	HEX	BINARY
EX DE, HL	235	EB	11 101 011

Αν ο DE περιείχε 10 και ο HL 37 πριν την ανταλλαγή, κατόπιν ο DE θα περιέχει το 37 και ο HL το 10.

Ακολουθεί τώρα μια πολύ σύντομη περίληψη των εντολών που μάθατε σ' αυτό το κεφάλαιο:

r= μονός καταχωρητής (single register) των 8 bits A,B,C,D,E,H ή L
r= ζεύγος καταχωρητών που χρησιμοποιείται σαν καταχωρητής των 16 bits
n= αριθμός των 8 bits από 0 μέχρι 255
nn= αριθμός των 16 bits από το 0 μέχρι το 65535
() γύρω από αριθμό ή ζεύγος καταχωρητών = η διεύθυνση στο
PC = Program Counter (Μετρητής Προγράμματος)
SP = Stack Pointer (Δείκτης Στοίβας).

LD σημαίνει LOAD

Οποιοδήποτε r μπορεί να φορτωθεί με οποιοδήποτε n. Η εντολή είναι της μορφής LD r,n.

Οποιοδήποτε r μπορεί να φορτωθεί με οποιοδήποτε άλλο r. Η εντολή είναι της μορφής LD r,r'.

Ο καταχωρητής A μπορεί να φορτωθεί από μια διεύθυνση μνήμης. Η εντολή είναι της μορφής LD A, (nn).

Μια διεύθυνση μνήμης μπορεί να φορτωθεί από τον καταχωρητή A. Η εντολή είναι της μορφής LD (nn),A.

Στις δύο παραπάνω εντολές μπορούν να χρησιμοποιηθούν τα περιεχόμενα του ζεύγους καταχωρητών HL αντί για το nn. Η εντολή είναι της μορφής LD A,(HL) και LD (HL),A.

Οποιοδήποτε rr μπορεί να φορτωθεί με οποιοδήποτε nn. Η εντολή είναι της μορφής LD rr,nn.

Οποιοδήποτε rr μπορεί να φορτωθεί από οποιαδήποτε θέση μνήμης μαζί με την αμέσως αποπάνω. Η εντολή είναι της μορφής LD rr, (nn).

Οποιαδήποτε θέση μνήμης μαζί με την αμέσως επόμενη μπορεί να φορτωθεί από οποιοδήποτε ζεύγος καταχωρητών. Η εντολή είναι της μορφής LD (nn), rr

Αν χρησιμοποιήσετε τον HL rr, θα χρειαστείτε ένα byte λιγότερο για να κάνετε κάποιο από τα δύο προηγούμενα.

Για να καλέσουμε μια υπορουτίνα χρησιμοποιούμε την CALL nn. Η εντολή CALL μπορεί να απευθύνεται σε οποιαδήποτε θέση μνήμης. Μια υπορουτίνα τελειώνει με RET.

Και οι δύο προηγούμενες εντολές χρησιμοποιούν τη στοίβα (stack).

Ένα πήδημα (jump) μπορεί να γίνει σε οποιαδήποτε θέση μνήμης χρησιμοποιώντας την εντολή JP nn.

Το ζεύγος καταχωρητών HL μπορεί να χρησιμοποιηθεί για να περιέχει το nn, οπότε η εντολή μειώνεται από τρία bytes σε ένα byte. Σ' αυτή τη περίπτωση η εντολή παίρνει τη μορφή JP (HL).

Ένα relative jump μπορεί να γίνει σε μια θέση που απέχει κατά +127 ή -128 από το τέλος της εντολής jump, η οποία παίρνει τη μορφή JR n.

Οι αριθμοί των 16 bits περιέχονται στη μνήμη κατά την αντίστροφη σειρά. Αν ο αριθμός αποτελείται από 4 δεκαεξαδικά ψηφία, τα δύο σημαντικότερα ψηφία αποθηκεύονται στη θέση μνήμης μετά από τα δύο λιγότερο σημαντικά ψηφία. Σε ένα ζεύγος καταχωρητών αποθηκεύονται κατά την κανονική σειρά: Περισσότερο σημαντικά - Λιγότερο σημαντικά (High - Low).

Όλοι οι κωδικοί λειτουργίας (opcodes) υπάρχουν στο Παράρτημα μαζί με μια συμβολική λειτουργία (symbolic operation).

ΑΠΛΑ ΜΑΘΗΜΑΤΙΚΑ

Στο προηγούμενο κεφάλαιο μάθατε για τις εντολές που φορτώνουν ένα μονό καταχωρητή με κάποιον αριθμό των 8 bits ή με τον αριθμό που περιέχεται σε έναν άλλο μονό καταχωρητή, με κώδικες λειτουργίας LD, r, n και LD r,r. Υπάρχει επίσης μια πλήρης ομάδα εντολών για πρόσθεση και αφαίρεση, και οι κωδικοί λειτουργίας τους βρίσκονται σε πολύ στενή αντιστοιχία με τους τύπους εντολών LD r,n και LD r,r.

Θα θυμάστε ότι ο καταχωρητής «A» είναι γνωστός και σαν συσσωρευτής (accumulator), και ότι υπάρχουν ορισμένες εντολές φόρτωσης που μπορούν να γίνουν μόνο αν χρησιμοποιείται ο συσσωρευτής. Ο καταχωρητής «A» βρίσκει τον πραγματικό εαυτό του στις μαθηματικές πράξεις των 8 bits, γιατί είναι ο μόνος καταχωρητής που μπορεί να χρησιμοποιηθεί για να περιέχει το αποτέλεσμα μιας μαθηματικής πράξης της 8 bits.

Πριν προχωρήσουμε στην εξέταση των αληθινών μαθηματικών πράξεων υπάρχουν δύο εντολές οι οποίες, παρ' όλο που δεν είναι μαθηματικές με την αυστηρή έννοια, μπορούν να εκτελεστούν χρησιμοποιώντας οποιονδήποτε από τους καταχωρητές γενικής χρήσης. Αυτές είναι η αύξηση κατά 1 (increase by 1), ή η μείωση κατά 1 (decrease by 1). Φανταστείτε ένα καταχωρητή γενικής χρήσης που περιέχει για παράδειγμα την τιμή 99: Μια αύξηση κατά 1 θα τον έκανε να περιέχει το 100, και μια μείωση κατά 1 θα τον έκανε να περιέχει το 98. Οι εντολές που χρησιμοποιεί η γλώσσα assembly είναι: INC r για την αύξηση κατά 1, όπου r είναι ένας καταχωρητής γενικής χρήσης, και DEC r για τη μείωση κατά 1.

Υπάρχει μια περίπτωση κατά την οποία η INC δεν αυξάνει κατά 1 την τιμή του καταχωρητή, και μια ακόμη που η DEC δεν μειώνει κατά 1 την τιμή του καταχωρητή. Όπως ξέρετε ένας μονός καταχωρητής μπορεί να περιέχει τιμές που βρίσκονται μεταξύ 0 και 255, οπότε τι νομίζετε ότι θα συμβεί αν ο καταχωρητής περιέχει την τιμή 255 και πρέπει να εκτελεστεί η εντολή INC;

Κάθε φορά που υπερβαίνουμε την τιμή που μπορεί να γραφτεί σε 8 bits, όλα αρχίζουν πάλι από το 0. Αυτό είναι σαν να έχουμε ένα ρολόι με ένα μόνο δείκτη, αριθμημένο από το 1 μέχρι το 256 κατά τη δεξιά φορά. Αν η ώρα είναι 255 και της προσθέσουμε 1 θα έχουμε 256 αλλά, όπως συμβαίνει και με το κανονικό ρολόι των 24 ωρών σαν το ψηφιακό που υπάρχει στο video ή δίπλα στο κρεβάτι σας, δεν δείχνει 24.00 τα μεσάνυχτα αλλά 00.00. Αυτό γίνεται γιατί η ώρα δεν είναι 24.00 της προηγούμενης μέρας, αλλά 00.00 της καινούριας μέρας.

Αν εξακολουθείτε να μην γνωρίζετε την απάντηση στο τι έρχεται μετά το 255,

προσπαθήστε να σκεφθείτε ότι στο δυαδικό σύστημα ο αριθμός είναι 11111111. Αυξήστε τον κατά 00000001 και το αποτέλεσμα είναι 100000000. Ο καταχωρητής μπορεί να κρατήσει αριθμούς των 8 μόνο bits, οπότε τι υπάρχει στον καταχωρητή;

Η άλλη εξαίρεση όπως είναι πιθανό να καταλάβετε, είναι όταν μια εντολή DEC

ASSEMBLER	DECIMAL	HEX	BINARY
INC B	4	04	00 000 100
INC C	12	0C	00 001 100
INC D	20	14	00 010 100
INC E	28	1C	00 011 100
INC H	36	24	00 100, 100
INC L	44	2C	00 101 100
INC (HL)	52	34	00 110 100
INC A	60	3C	00 111 100

ASSEMBLER	DECIMAL	HEX	BINARY
DEC B	5	05	00 000 101
DEC C	13	0D	00 001 101
DEC D	21	15	00 010 101
DEC E	29	1D	00 011 101
DEC H	37	25	00 100 101
DEC L	45	2D	00 101 101
DEC (HL)	53	35	00 110 101
DEC A	61	3D	00 111 101

Σχήμα 6.1

ενεργεί σε ένα καταχωρητή που περιέχει το 0. Χρησιμοποιώντας το ίδιο με το παραπάνω σύστημα αν βρίσκεται ότι σας βοηθάει, τι θα περιέχει ο καταχωρητής μετά την εκτέλεση της DEC; Στο δυαδικό σύστημα ο καταχωρητής περιέχει το 00000000 πριν την εντολή DEC. Αν μπερδεύετε με αυτό, γυρίστε στο Κεφάλαιο 3, και ξαναδιαβάστε το τμήμα που ασχολείται με τους δυαδικούς αριθμούς «συμπληρώματα του 2».

Ο σχηματισμός των εντολών INC και DEC γίνεται πολύ απλά. Όπως μπορείτε να δείτε στο σχήμα 6.1 τα bits 5,4 και 3 καθορίζουν τον καταχωρητή που θα χρησιμοποιηθεί κατά τον ίδιο τρόπο που γίνονταν και στις εντολές LD.

Για να έχετε μια επίδειξη των εντολών INC και DEC γράψτε το παρακάτω σύντομο πρόγραμμα:

```

10 MM=HIMEM
20 MEMORY 42499
30 FOR N=42500 TO 42511:READ D:POKE N,D:A=A+D:NE
XT
40 IF A<>1353 THEN CLS:PEN 3:PRINT"DATA ERROR":P
EN 1:EDIT 90
50 INPUT"PRESS ENTER TO START":A=A-32:B=224
60 PRINT CHR$(A);:A=A+1:B=B-1:IF B<>0 THEN 60
70 PRINT:CALL 42500
90 DATA 6,224,62,32,205,90,187,60,5,32,249,201
100 END

```

Σχήμα 6.2

Οι εκδόσεις της γλώσσας μηχανής Hex και Assembly, που τοποθετούνται με POKE στη μνήμη από τη γραμμή 30 και την DATA στη γραμμή 90, φαίνονται στο Σχήμα 6.3.

Παρατηρείστε πως μια καινούρια εντολή έχει προστεθεί πριν το RET στο τέλος της ρουτίνας. Θα εξηγηθεί κανονικά στο επόμενο κεφάλαιο, αλλά θα μπορούσατε πιθανώς να διακρίνετε το νόημά της, αν κοιτάξετε τη γραμμή 60 του προγράμματος σε BASIC. Σε συντομία, το NZ σημαίνει Not Zero (μη μηδενικός), και έχει σχέση με το αποτέλεσμα της τελευταίας μαθηματικής πράξης. Η εντολή επομένως σημαίνει: αν το αποτέλεσμα της τελευταίας μαθηματικής πράξης δεν ήταν 0 τότε κάνε Jump Relative στη label «PRINT».

HEX	ASSEMBLER
06 E0	LD B,224
3E 20	LD A,32
CD 5A BB PRINT:	CALL 47962
3C	INC A
05	DEC B
20 F9	JR NZ,PRINT
C9	RET

Σχήμα 6.3

Όταν τρέξει το πρόγραμμα θα τυπώσει το σει χαρακτήρων του Amstrad στην οθόνη, αρχίζοντας με το κενό διάστημα (space) και συνεχίζοντας με ό-

λους τους χαρακτήρες που υπάρχουν στο παράρτημα III του βιβλίου Amstrad User Instructions στη σελίδα 2.

Οι εντολές για πρόσθεση στον καταχωρητή «A» ή για αφαίρεση από τον καταχωρητή «A» είναι σχεδόν το ίδιο εύκολες, στην απλούστερη μορφή τους. Η συμβολική εντολή για την απλή πρόσθεση είναι ADD (πρόσθεση), και για την απλή αφαίρεση είναι SUB (από το (SUBtract - αφαίρεση). Εφ' όσον μόνο ο καταχωρητής «A» μπορεί να χρησιμοποιηθεί για μαθηματικά στον 8 bits, θα φαινόταν περιττό να καθορίσουμε ποιός καταχωρητής χρησιμοποιείται, και στη περίπτωση της συμβολικής εντολής SUB συμβαίνει πραγματικά αυτό. Όμως η συμβολική εντολή ADD χρησιμοποιείται και στην εκτέλεση πρόσθεσης στα 16 bits χρησιμοποιώντας το ζεύγος καταχωρητών HL, όπως θα εξηγηθεί αργότερα. Με την ADD συνηθίζεται να αναφέρουμε ποιός καταχωρητής πρόκειται να χρησιμοποιηθεί, παρ' όλο που ορισμένοι assemblers δεν το χρειάζονται. Ο assembler Denpac δεν θα δεχθεί ένα ADD χωρίς να καθορίζεται ο καταχωρητής, αλλά είναι πιθανό ότι ο assembler Picturespue θα δέχεται μόνο του το ADD.

Αν χρειάζεται να προσθέσετε έναν αριθμό στον καταχωρητή A, η πλήρης εντολή είναι ADD A, n, ενώ για να αφαιρέσουμε έναν αριθμό από τον καταχωρητή A η εντολή είναι απλώς SUB n. Οι άλλες μορφές των εντολών είναι:

ASSEMBLER	DECIMAL	HEX	BINARY
ADD A,n	198 n	C6 n	11 000 110 n
SUB n	214 n	D6 n	11 010 110 n

Μπορείτε να δείτε την εντολή ADD A, ή σε χρήση αν αλλάξετε το BASIC πρόγραμμα ως εξής:

```
30 FOR N=42500 TO 42512:READ D:POKE N,D:A=A+D:NEXT
```

το 1353 στη γραμμή 40 γίνεται 1491, και η γραμμή 90 γίνεται:

```
90 DATA 6,224,62,32,205,90,187,198,1,5,32,248,201
```

Αυτό άλλαξε την INC A στο listing της assembly σε ADD A, 1 και το μήκος της JR άλλαξε επίσης, αλλιώς το παραπάνω byte θα οδηγούσε το πήδημα σε λάθος προορισμό. Όταν το τρέξετε δεν θα παρουσιαστεί διαφορά στο αποτέλεσμα, αλλά θα χρειαστεί ένα παραπάνω byte μνήμης.

Για να δείτε την εντολή SUB μπορείτε να κάνετε τις παρακάτω αλλαγές στο πρόγραμμα BASIC:

```
Αλλάξτε το 1491 στη γραμμή 40 σε 1730  
η γραμμή 55 γίνεται 55A = 255:B = 224  
στη γραμμή 60 αλλάξτε το A = A+1 σε A = A-1
```

και η γραμμή 90 να αλλάξει σε:

```
90 DATA 6,224,62,255,205,90,187,214,1,5,32,248,201
```

Η γλώσσα assembly είναι τώρα:

HEX	ASSEMBLER
06 E9	LD B,224
3E FF	LD A,255
CD 5A BB	PRINT: CALL 47962
D6 01	SUB 1
05	DEC B
20 FB	JR NZ,PRINT
C9	RET

Σχήμα 6.4

Όπως και στις εντολές INC και DEC όταν το αποτέλεσμα ενός ADD ή SUB είναι 0 το μηδενικό flag γίνεται «1» (the Zero Flag is set), αλλιώς γίνεται «0» (reset), δείχνοντας ότι το αποτέλεσμα δεν είναι μηδενικό. Αν η εκτέλεση μιας ADD χρειάζεται μεταφορά κρατούμενου σε κάποιο ένατο bit, γιατί το αποτέλεσμα είναι μεγαλύτερο από 255 ή στην περίπτωση μιας εντολής SUB, όταν χρειάζεται δανεισμός από το ένατο bit, που οφείλεται στο ότι το αποτέλεσμα είναι μικρότερο του μηδενός, υπάρχει ένα flag που θα σου πει αν η πράξη χρειάζεται αυτή τη μεταφορά, και ονομάζεται — προφανώς — το flag του κρατούμενου (the carry flag). Το carry flag είναι πάντοτε «1» αν έχει χρειαστεί κάποιο κρατούμενο ή δανεισμός μονάδας σε μαθηματική εντολή, αλλιώς είναι «0». Παρατηρήστε πως οι εντολές INC και DEC δεν έχουν καμία επίδραση στο carry flag.

Εκτός από τη δυνατότητα πρόσθεσης ή αφαίρεσης ενός αριθμού στον καταχωρητή A, είναι δυνατό να εκτελεσθούν αυτές οι πράξεις με κάποιον άλλο καταχωρητή, δίνοντας τις συμβολικές εντολές ADD A, r και SUB r. Οι άλλες μορφές είναι:

ASSEMBLER	DECIMAL	HEX	BINARY
ADD A, r	128 - 135	80 - 87	10 000 r
SUB r	144 - 151	90 - 97	10 010 r

Το r αντιπροσωπεύει κάποιον καταχωρητή γενικής χρήσης ή τον καταχωρητή A, και οι κωδικοί είναι ίδιοι με αυτούς που δόθηκαν στο κεφάλαιο 5.

B=000	H=100
C=001	L=101
D=010	
E=011	A=111

Ο κωδικός 110 χρησιμοποιείται πάλι για να σημαίνει (HL). Δηλαδή, με τα περιεχόμενα της θέσης μνήμης της οποίας η διεύθυνση βρίσκεται στο ζεύγος κατα-

χωρητών HL.

Οι SUB r και ADD A, r εργάζονται με τον ίδιο ακριβώς τρόπο που εργάζονται και οι ADD A, n και SUB n, και έχουν όμοια επίδραση στα Zero και Carry flags. Η διαφορά είναι βέβαια, ότι το r χρησιμοποιείται σαν BASIC μεταβλητή και μπορεί να χρησιμοποιηθεί στη θέση ενός προκαθορισθέντος αριθμού.

Γνωρίζοντας τα παραπάνω και έχοντας στη μνήμη σας αυτά που μάθατε στο προηγούμενο κεφάλαιο, θα πρέπει να είσαστε σε θέση να γράψετε μια σύντομη ρουτίνα σε γλώσσα assembly, που θα προσθέτει τα περιεχόμενα της μνήμης στη διεύθυνση 42594 στα περιεχόμενα της μνήμης στη διεύθυνση 42596 και θα τοποθετεί το αποτέλεσμα στη διεύθυνση 42598. Αφού το κάνετε αυτό, αν διαθέτετε assembler μπορείτε να πληκτρολογήσετε την υπορουτίνα σας. Αν όχι, θα πρέπει να την κωδικοποιήσετε με το χέρι, χρησιμοποιώντας τις πληροφορίες που σας έχουν ήδη δοθεί, ή κοιτάζοντας στο Παράρτημα που δίνει το ρεπερτόριο εντολών του Z80. Μπορείτε κατόπιν να χρησιμοποιήσετε το πρόγραμμα loader για να την εισάγετε στον υπολογιστή σας. Η αρχική διεύθυνση πρέπει να είναι 42550.

Υπάρχουν αρκετοί τρόποι με τους οποίους μπορεί να γραφτεί μια υπορουτίνα, και δύο απ' αυτούς δίνονται στο τέλος του βιβλίου για την περίπτωση που θα «κολλήσετε». Η κρίσιμη δοκιμή θα είναι εκείνη που εξετάζει αν η υπορουτίνα σας δουλεύει, και παρακάτω σας δίνουμε μια υπορουτίνα που σας επιτρέπει να κάνετε κάτι τέτοιο. Χρησιμοποιεί ορισμένες εντολές που ακόμη σας είναι άγνωστες, και οι οποίες θα σας εξηγηθούν σύντομα, αν και είναι πιθανό να καταλάβετε τη μια από αυτές αμέσως. Το τέλος του προγράμματός σας πρέπει να είναι:

ASSEMBLER	DECIMAL	HEX
CALL 42500	205 3 166	CD 03 A6
RET	201	C9

Αυτή η υπορουτίνα που τυπώνει το αποτέλεσμα, η οποία καλείται από την υπορουτίνα σας (Σχήμα 6.5).

ASSEMBLER	DECIMAL	HEX
ORG 42500		SET MEMORY TO A603
ENT 42500		START ADDRESS A604
LD A, (42598)	58 102 166	3A 66 A6 CHECK
LD L, A	111	6F
LD H, 0	38 0	26 00
LD DE, -100	17 156 255	11 9C FF
CALL REDN	205 24 166	CD 18 A6
LD E, -10	30 246	1E F6
CALL REDN	205 24 166	CD 18 A6
LD A, L	125	7D
JR PRIN	24 9	18 09
REDN: LD A, 0	62 0	3E 00

FNUM: INC	A	60	3C
ADD	HL, DE	25	19
JR	C, FNUM	56 252	38 FC
SBC	HL, DE	237 82	ED 52
DEC	A	61	3D
PRIN: ADD	A, #30	198 48	C6 30
CALL	47962	205 90 187	CD 5A BB
RET		201	C9
CHECKSUM		3966	

Σχήμα 6.5

Αυτή η υπορουτίνα μπορεί να εισαχθεί από το listing της γλώσσας assembly αν έχετε assembler, ή χρησιμοποιώντας το πρόγραμμα Hex loader που δίνεται στο τέλος του βιβλίου, το οποίο πρέπει να έχετε σε μια κασέτα έτοιμο για φόρτωμα. Αν αισθάνεσθε δυνατός, μπορείτε να αλλάξετε το BASIC πρόγραμμα που πληκτρολογήσατε ωρύτερα σ' αυτό το κεφάλαιο. Θα πρέπει να μπορείτε να το κάνετε μόνος σας αλλά, σαν μια μικρή βοήθεια, σας λέμε ότι υπάρχουν 35 bytes στο listing, οπότε η γραμμή 30 θα αρχίζει ως εξής:

```
30 FOR N=42500 TO 42534
```

Να θυμάστε ότι ο κωδικός σας θα ξεκινήσει στη 42550, γι' αυτό όταν το πρόγραμμα φορτώσει το κομμάτι σας, το N πρέπει να αρχίζει σαν 42550.

Αν αλλάξετε το πρόγραμμα BASIC, βεβαιωθείτε ότι κρατήσατε αντίγραφο της τροποποιημένης έκδοσης πριν το τρέξετε, για την περίπτωση που έχετε κάνει κάποιο λάθος που θα έκανε το πρόγραμά σας να χαθεί.

Το πρώτο πρόβλημα που θα αντιμετωπίσετε αφού φορτώσετε το πρόγραμμα γλώσσας μηχανής στη μνήμη θα είναι το πως να βάλετε αριθμούς μέσα στη ρουτίνα, για να τους προσθέσετε. Εδώ είναι που θα πρέπει να καταφύγουμε στη BASIC γι' άλλη μια φορά - μέχρι να προχωρήσετε λίγο ακόμη σ' αυτό το βιβλίο!

```
400 INPUT "FIRST NUMBER"; A: INPUT "SECOND NUMBER"; B
410 PRINT A; "+"; B; "=";
420 POKE 42594, A: POKE 42596, B
430 CALL 42550
440 GOTO 400
```

Σχήμα 6.6

Γράψτε τώρα GOTO 400 και η BASIC θα σας ζητήσει τους δύο αριθμούς, και κατόπιν θα τους τοποθετήσει με POKE στη μνήμη έτοιμους για τη γλώσσα μηχανής. Η ρουτίνα σας κατόπιν θα κληθεί από τη γραμμή 430 για να κάνει τη πρόσθεση, και το αποτέλεσμα θα τυπωθεί από την υπορουτίνα που δόθηκε προηγουμένως. Πατήστε δύο φορές το ESCAPE όταν θελήσετε να βγείτε έξω από το βρόχο που ζητάει αριθμούς και τους προσθέτει.

Αν είχατε δώσει ένα ζευγάρι αριθμών που το άθροισμα τους ξεπερνούσε το 255,

θα βλέπατε ότι παρουσιαζόταν λαθεμένο αποτέλεσμα. Θα θυμάστε το λόγο γι' αυτό από προηγούμενες περιπτώσεις, και θα θυμάστε ακόμη ότι το carry flag είναι πάντοτε «1» (set) όταν συμβαίνει αυτό. Χρειαζόμαστε λοιπόν έναν τρόπο για να χειριζόμαστε τους αριθμούς που δεν χωράνε σε ένα byte, και χρησιμοποιούμε το carry flag για να μας πληροφορεί κάθε φορά που πρέπει να ασχοληθούμε με κρατούμενο.

Αν αυτό αρχίζει να σας μπερδεύει, σκεφθείτε για ένα λεπτό τι συμβαίνει όταν προσθέτετε 9+6+8.

9+6=5 και κρατούμενο 1. Αυξάνετε επομένως τις δεκάδες κατά 1.

5 (από επάνω) + 8 = 3 και κρατούμενο 1. Αυξάνετε πάλι τις δεκάδες κατά 1. Το αποτέλεσμα είναι: δύο δεκάδες και τρεις μονάδες ή 23, που είναι σωστό.

Σκεφθείτε τώρα τι συμβαίνει στο πρόγραμμά σας όταν εκτελεί την επόμενη δυαδική πρόσθεση:

```

1010 0101 (165)
+1011 0000 (176)
-----
      1
    +0
    =1
      01
    +0
    =01

      101
    +0
    =101

      0101
    +0
    =0101

0 0101
+1
=1 0101

11 0101
+1
=101 0101

101 0101
+0
+0
=101 0101

1101 0101
+1
=0101 0101

```

Κάθε στήλη προστίθεται στην από πάνω της κατά τον ίδιο ακριβώς τρόπο που έγινε και η προηγούμενη δεκαδική πρόσθεση. Αυτή τη φορά όμως, επειδή η άθροιση γίνεται στο δυαδικό σύστημα και τραβάει πολύ σε μάκρος, το κρατούμενο προστίθεται στους δύο αριθμούς της επόμενης στήλης, όταν εμφανίζεται.

Το μόνο πραγματικό κρατούμενο, εφ' όσον οι αριθμοί μέχρι το 255 μπορούν να περιέχονται χωρίς υπερχείλιση (overflow) στον καταχωρητή, βρίσκεται στο τέλος αυτής της άθροισης. Σ' αυτό το σημείο, το κάθε bit αξίζει 256 φορές παραπάνω από το ελάχιστο σημαντικό bit (least significant bit).

Αν ένας καταχωρητής χρησιμοποιείται για να αποθηκεύει τα κρατούμενα όταν παρουσιάζονται μετράει σε μονάδες των 256.

Αυτό που χρειάζεται είναι μια σειρά από bytes, από τα οποία οποιαδήποτε υπερχείλιση προστίθεται αυτόματα στο επόμενο byte.

Γίνεται αυτό ακριβώς που θα γινόταν αν η πρόσθεση ήταν δεκαδική, μόνο που εδώ η θάση είναι 256 αντί για δέκα.

Στη πραγματικότητα αυτό είναι πολύ εύκολο να γίνει και εσείς δεν χρειάζεται να σπαταλάτε φαιά ουσία. Υπάρχουν εντολές ενσωματωμένες στη Z80 CPU που αυτόματα προσθέτουν ή αφαιρούν με κρατούμενο.

(85) με κρατούμενο 1 (στίχος 256)

Τα παραπάνω ονομάζονται πρόσθεση με κρατούμενο (add with carry) και αφαίρεση με κρατούμενο (subtract with carry). Οι συμβολικές εντολές για τον assembler, αντί να είναι ADDC και SUBC, συντομεύεται για ταχύτερη πληκτρολόγηση, σε ADC και SBC. Όταν χρησιμοποιούνται στη θέση των εντολών ADD και SUB το carry flag περιλαμβάνεται στην πράξη. Για παράδειγμα, δείτε το πρόγραμμα του σχήματος 6.7.

Φανταστείτε ότι η διεύθυνση 42594 περιέχει τον 10100101 (165), η 42596 περιέχει τον 10110000 (176) και όλες οι υπόλοιπες διευθύνσεις περιέχουν τον 00000000 πριν τρέξει το πρόγραμμα. Το πρώτο μέρος του προγράμματος θα προσθέσει τον 165 στον 176, θα υπάρχει ένα κρατούμενο και ο καταχωρητής A θα μείνει με τον 85. Αυτό θα αποθηκευτεί στη διεύθυνση 42598. Το υπόλοιπο πρόγραμμα θα προσθέσει 0 με 0 και θα αποθηκεύσει το αποτέλεσμα στη διεύθυνση 42599. Για να αποφύγουμε αυτή την απώλεια ενέργειας, ας αλλάξουμε την ADD σε ADC.

<pre>LD HL, 42596 LD A, (42594) ADD A, (HL) LD (42598), A LD A, (42595) INC HL ADD A, (HL) LD (42599), A</pre>	<p>Φορτώνεται η τιμή που υπάρχει στη διεύθυνση 42594 στον καταχωρητή A και προσθέτονται στον τελευταίο τα περιεχόμενα της διεύθυνσης που δείχνει ο HL (42596). Το αποτέλεσμα τοποθετείται στη διεύθυνση 42598.</p> <p>Η διαδικασία επαναλαμβάνεται με τον A να έχει τα περιεχόμενα της 42595 και τον HL να δείχνει την 42597. Το αποτέλεσμα πηγαίνει στη διεύθυνση 42599.</p>
--	---

Σχήμα 6.7

Το πρώτο μέρος του προγράμματος εργάζεται όπως και πριν, και όταν εκτελείται το δεύτερο τμήμα του το carry flag γίνεται «1» (set). Όταν εκτελείται το ADD with carry (ADC) ο υπολογισμός γίνεται 0+0+ κρατούμενο, αντί για 0+0. Το carry flag γίνεται «1», οπότε αυτή τη φορά το αποτέλεσμα είναι 1, και στη διεύθυνση 42599 αποθηκεύεται 1.

Προσθέτοντας την εντολή LD HL,(42598) στο τέλος του προγράμματος, ο καταχωρητής H μπορεί να υποχρεωθεί να περιέχει το σημαντικότερο byte του αποτελέσματος, και ο L το λιγότερο σημαντικό. Έτσι, μετά την εκτέλεση του προγράμματος ο HL θα περιέχει τον αριθμό 0000 0001 0101 0101b ή 0155 Hex. Σας φαίνεται γνωστός; Θα έπρεπε, γιατί θα δείτε ότι είναι το σωστό αποτέλεσμα στην προηγούμενη άθροιση.

Πριν προχωρήσουμε στην περιγραφή της χρήσης της εντολής SBC και την ADC με αριθμούς μεγαλύτερους από εκείνους που μπορούν να περιληφθούν σε δύο bytes, σας δίνουμε τους αριθμητικούς κώδικες των εντολών:

ASSEMBLER	DECIMAL	HEX	BINARY
ADC A,n	206 n	CE n	11 001 110 n
SBC A,n	222 n	DE n	11 011 110 n
ADC A,r	136 - 143	88 - 8F	10 001 r
SBC A,r	152 - 159	98 - 9F	10 011 r

Ως συνήθως, το r μπορεί να είναι οποιοσδήποτε καταχωρητής γενικής χρήσης, ο καταχωρητής A ή το (HL), των οποίων οι κώδικες εξακολουθούν να είναι οι ίδιοι. Παρατηρήστε πως η SBC πρέπει να έχει καθορισμένο τον καταχωρητή A. Αυτό γίνεται γιατί αντίθετα με την εντολή SUB, η SBC μπορεί να χρησιμοποιηθεί κατά διαφορετικό τρόπο, ο οποίος θα εξηγηθεί αργότερα.

Μπορείτε τώρα να εισάγετε το πρόγραμμα του Σχήματος 6.9 που σας επιτρέπει να πειραματιστείτε με τις εντολές ADC και SBC. Αν έχετε assembler θα μπορέσετε να τροποποιήσετε το πρόγραμμα που γράψατε για να προσθέσετε τα περιεχόμενα δύο θέσεων μνήμης,, αλλιώς θα πρέπει να χρησιμοποιήσετε το πρόγραμμα HEX LOADER. Τα δεκαδικά listings δεν είναι πια χρήσιμα γιατί τα προγράμματα τώρα αρχίζουν να έχουν πολύ μεγάλο μήκος για να εισαχθούν με τη μέθοδο των "DATA".

Αν έχετε χρησιμοποιήσει τον assembler μπορείτε να πατήσετε το R ακολουθούμενο από το [ENTER] για την εκτέλεση, αλλιώς θα πρέπει να πληκτρολογήσετε CALL 42550 και να πατήσετε το [ENTER]. Το πρόγραμμα τότε θα προσθέσει τον κωδικό ASCII του επόμενου πλήκτρου που θα πατήσετε στα περιεχόμενα της διεύθυνσης 42596 και θα αποθηκεύσει το αποτέλεσμα στη διεύθυνση 42598. Κατόπιν τα περιεχόμενα της διεύθυνσης 42595 προσθέτονται με κρατούμενο στα περιεχόμενα της διεύθυνσης 42597, και το αποτέλεσμα αποθηκεύεται στη διεύθυνση 42599.

Μια υπορουτίνα του λειτουργικού συστήματος χρησιμοποιείται για το διάβασμα του πληκτρολόγιου, η οποία καλείται με την CALL 47896. Αυτή η CALL περιμένει να πατηθεί ένα πλήκτρο, και επιστρέφει έχοντας τον κωδικό ASCII του πλήκτρου στον καταχωρητή A.

Το αποτέλεσμα θα είναι πάντοτε ο κωδικός του πλήκτρου που πατήσατε, εκτός κι αν έχετε τοποθετήσει κάτι στις διευθύνσεις που προσθέτονται από το πρόγραμμα. Μπορείτε να το ελέγξετε αυτό κοιτάζοντας τους κωδικούς στο Παράρτημα 3 του Amstrad manual.

Αν χρησιμοποιείτε τον assembler πατήστε το R ακολουθούμενο από το [ENTER]. Αυτό θα σας φέρει πίσω στη BASIC. Μπορείτε τώρα να βάλετε τιμές στις διευθύνσεις που προσθέτονται, και κατόπιν με την CALL 42550 να δείτε το αποτέλεσμα.

Θυμηθείτε ότι θα πρέπει να πατήσετε ένα πλήκτρο για να δώσετε ένα μέρος από το άθροισμα.

Για να προσθέσετε, για παράδειγμα, το 220 με το 89:

```
Γράψτε POKE 42596, 220 [ENTER] CALL 42550 [ENTER]
```

Κατόπιν πατήστε SHIFT και Y (το κεφαλαίο Y έχει κωδικό ASCII 89). Το αποτέλεσμα που θα παρουσιαστεί θα είναι 309.

Ή για να προσθέσετε το 23260 στο 345 πρέπει να γράψετε:·

```
POKE 42596, 220 [ENTER] POKE 42597, 90 [ENTER]  
POKE 42595, 1 [ENTER] CALL 42550 [ENTER]
```

Κατόπιν πατήστε SHIFT και Y. Θα πρέπει το αποτέλεσμα να είναι 23605, αλλά δεν είναι! Γιατί;

Ο 23260 είναι 5ADC στο HEX και ο 345 είναι 159 στο HEX. Έχοντας υπόψη ότι ο Z80 αποθηκεύει ανάποδα τους αριθμούς, και κοιτάζοντας το προηγούμενο πρόγραμμα, θα βρείτε ότι η διεύθυνση 42596 και το πληκτρολόγιο προσφέρουν τα λιγότερο σημαντικά bytes του αθροίσματος (το DCh και το 59h), ενώ οι διευθύν-

```

ASSEMBLER                                HEX
SET MEMORY TO
A603
START ADDRESS
A636

ORG    42550

ENT    42550
LD     HL, 42596        21 64 A6
CALL   47896           CD 18 BB
ADD    A, (HL)         86
LD     (42598), A     32 66 A6
LD     A, (42595)     3A 63 A6
INC    HL              23
ADC    A, (HL)        BE
LD     (42599), A     32 67 A6
CALL   42500          CD 04 A6
RET                                         C9

                                MORE? Y/N      Y
ORG    42500                START ADDRESS  A604
                                                CHECK

LD     HL, (42598)        2A 66 A6
NOP                                         00
NOP                                         00
NOP                                         00
NOP                                         00
NOP                                         00
NOP                                         00
LD     DE, -1000         11
                                                18 FC
CALL   REDN              CD 21 A6
LD     DE, -100         11 9C FF
CALL   REDN              CD 21 A6
LD     E, -10           1E F6
CALL   REDN              CD 21 A6
LD     A, L              7D
JR     PRIN              18 09
REDN: LD     A, 0         3E
                                                00
FNUM: INC    A           3C
      ADD    HL, DE      19
      JR     C, FNUM     3B FC

```



```

SBC      HL, DE          ED 52
DEC      A              3D
PRIN:    ADD     A, #30   C6 30      03F6
         CALL    47962    CD 5A  BB
         RET                               CS      END 02AB
MORE? Y/N      N

```

Σχήμα 6.8

σεις 42595 και 42597 προσφέρουν τα σημαντικότερα bytes (5Ah και 01h).

Ο DCh είναι το 220 και ο 59h είναι το 89, έτσι το 220 τοποθετείται με POKE στη διεύθυνση 42596 και το κεφαλαίο Y δίνει το 89 για το πρώτο μέρος του αθροίσματος. Αυτα προσθέτονται και το αποτέλεσμα τοποθετείται στη 42598. Το αποτέλεσμα θα πρέπει να ισούται με 35h γιατί $59h + DCh = 135h$. Το 1 είναι το κρατούμενο, ενώ μένει το 35h, ή 53. Αν κοιτάξετε τη διεύθυνση 42598 πληκτρολογώντας:

? PEEK (42598) [ENTER]

θα καταλάβετε ότι πραγματικά αυτή είναι η περίπτωση. Το 1h και το 5Ah (τα σημαντικότερα bytes του αθροίσματος) είναι 1 και 90 αντίστοιχα. Πρόκειται για τους αριθμούς που τοποθετήθηκαν με POKE στις διευθύνσεις 42595 και 42597 για να προστεθούν με κρατούμενο στο δεύτερο μέρος του προγράμματος. $5Ah + 01h + \text{κρατούμενο}$ (που είναι «1») και επομένως ισούται με 1) = 5Ch ή 92. Αυτό θα πρέπει να βρεθεί στη διεύθυνση 42599. Ελέγξτε το και δείτε αν είναι σωστό.

Το αποτέλεσμα της άθροισης είναι 5C35h, το 5C είναι ίσο με 92 και αποτελεί το σημαντικότερο byte με αξία 256* την ονομαστική τιμή του που είναι 23552. Το λιγότερο σημαντικό byte είναι το 35h, 53. Αυτό κατόπιν προστίθεται στην αξία του σημαντικότερου byte: $23552 + 53 = 23605$ ή 5C35h. Επομένως το άθροισμα είναι σωστό. Γιατί τότε το αποτέλεσμα παρουσιάζεται λαθεμένο;

Η απάντηση σ' αυτό θα βρεθεί με μια ανάλυση του δεύτερου τμήματος του παραπάνω προγράμματος. Πρόκειται για το κομμάτι που τυπώνει το αποτέλεσμα στην οθόνη. (Υπάρχει κάποιος σοβαρός λόγος για τη σειρά των NOPS, κανένας στα λογικά του δεν θα ξόδευε χώρο αν δεν είχε να τον γεμίσει με κάτι αργότερα!)

Υπάρχουν δύο εντολές (ADD HL, DE και SBC HL, DE) τις οποίες δεν έχετε γνωρίσει, γι' αυτό θα σας τις εξηγήσουμε σύντομα στην αρχή. Πληρέστερη εξήγηση ακολουθεί αργότερα σ' αυτό το κεφάλαιο, και οι τρόποι με τους οποίους μπορούμε να μεταχειριστούμε τις εντολές εξετάζονται με περισσότερες λεπτομέρειες σε επόμενα κεφάλαια.

Όπως αναφέρθηκε παραπάνω, η μόνη από τις εντολές πρόσθεσης ή αφαιρέσεως που χρησιμοποιεί αποκλειστικά τον καταχωρητή A είναι η εντολή SUB. Ο άλλος τρόπος με τον οποίο μπορούν να γίνουν αυτές οι μαθηματικές πράξεις, είναι με το ζεύγος καταχωρητών HL που χρησιμοποιείται σαν συσσωρευτής των 16 bits. Σε όλες τις περιπτώσεις το ζεύγος καταχωρητών HL θα περιέχει το αποτέλεσμα μετά την πράξη, κατά τον ίδιο τρόπο που το κάνει και ο καταχωρητής A μετά από μια πρόσθεση ή αφαίρεση των 8 bits.

Αντίθετα με τις εντολές των 8 bits που χρησιμοποιούν τον καταχωρητή A το ο-

operand, σ' αυτή την περίπτωση ο αριθμός που θα προστεθεί ή θα αφαιρεθεί από το ζεύγος καταχωρητών HL, μπορεί να ληφθεί μόνο από κάποιο ζεύγος καταχωρητών γενικής χρήσης, ή από τον καταχωρητή ειδικής χρήσης SP (Stack Pointer, Δείκτης Στοιβάς). Δεν είναι δυνατό να χρησιμοποιηθεί αριθμητικό operand (για παράδειγμα ADD HL, 23456), μία θέση μνήμης (για παράδειγμα, ADC HL, (23456)) ή ακόμη και μια θέση μνήμης που υποδεικνύεται από ζεύγος καταχωρητών (για παράδειγμα, SBC HL, (DE)). Με το Δείκτη Στοιβάς θα ασχοληθούμε λεπτομερώς σε επόμενο κεφάλαιο. Οι διαθέσιμες εντολές είναι:

ASSEMBLER	DECIMAL	HEX	BINARY
ADD HL, BC	9	09	00 001 001
ADD HL, DE	25	19	00 011 001
ADD HL, HL	41	29	00 101 001
ADD HL, SP	57	39	00 111 001
ADC HL, BC	237 74	ED 4A	11 101 101 01 001 010
ADC HL, DE	237 90	ED 5A	11 101 101 01 011 010
ADC HL, HL	237 106	ED 6A	11 101 101 01 101 010
ADC HL, SP	237 122	ED 7A	11 101 101 01 111 010
SBC HL, BC	237 66	ED 42	11 101 101 01 000 010
SBC HL, DE	237 82	ED 52	11 101 101 01 010 010
SBC HL, HL	237 98	ED 62	11 101 101 01 100 010
SBC HL, SP	237 114	ED 72	11 101 101 01 110 010

Σχήμα 6.9

Λειτουργούν κατά τον ίδιο τρόπο που λειτουργούν τα ανάλογά τους στα 8 bits, ADD A,B ADC A,B και SBC A,B κ.λ.π. μόνο που θρυσκόμαστε στα 16bits.

Για παράδειγμα για να προσθέσουμε τους 55536 και 2000 (δεκαδικό), το πρόγραμμα σε γλώσσα Assembly θα ήταν ως εξής:

```
LD DE, 55536
LD HL, 2000
ADD HL, DE
```

Μετά την εκτέλεση του προγράμματος το ζεύγος καταχωρητών HL θα περιέχει το αποτέλεσμα 57536 (E0C0h FOh στον H, C0h στον L) και το ζεύγος καταχωρη-

τών DE θα εξακολουθεί να περιέχει τον 55536 (D8F0h D8h στον D, FOh στον E) και το Flag του κρατουμένου (carry flag) θα γίνει «0» (reset). Αν το άθροισμα ήταν 55536+23605 τότε το αποτέλεσμα, αντί να ήταν 79141 (13525h) που δεν μπορεί να τοποθετηθεί στα 16 bits, θα ήταν 13605 (3525h) και το flag του κρατουμένου θα ήταν «1» (set). Το μεταφερόμενο από το άθροισμα έχει τιμή 65536 * την τιμή του ελάχιστου σημαντικού bit του ζεύγους καταχωρητών. Αυτό είναι $2 \uparrow 16$, ενώ όταν έχουμε μεταφορά μετά από πράξη στα 8 bits η τιμή του είναι 256^* το ελάχιστο σημαντικό bit, που είναι $2 \uparrow 8$.

Αν η ADD είχε αντικατασταθεί από την ADC, θα είχε προστεθεί και το κρατούμενο όπως ακριβώς γίνεται με τις εντολές των 8 bits.

Η εντολή SBC, όταν χρησιμοποιείται με το ζεύγος καταχωρητών HL, μπορεί επίσης να γίνει ανάλογη προς την SBC των 8 bits με τον καταχωρητή A. Εξαιτίας της απουσίας μιας εντολής SUB για τις πράξεις των 16 bits, αν το flag του κρατουμένου δεν χρειάζεται να αφαιρεθεί από το ζεύγος καταχωρητών HL, πρέπει να γίνει «0» αν είναι «1», αλλιώς το αποτέλεσμα μπορεί να είναι κατά ένα μικρότερο από το σωστό.

Υπάρχουν εντολές για να γίνει «1» (set) το flag του κρατουμένου, και για να συμπληρωθεί (complement) το flag του κρατουμένου (SCF και CCF) αλλά δεν υπάρχει εντολή που να κάνει «0» (reset) το flag του κρατουμένου. Είναι δυνατό να γίνει το flag του κρατουμένου «0» κάνοντάς το πρώτα «1» και συμπληρώνοντάς το (SCF ακολουθούμενη από την CCF) αλλά πρόκειται για μακριά διαδικασία, ενώ η εντολή AND A κάνει την ίδια δουλειά με μια εντολή λιγότερη. Αυτή είναι μια από τις λογικές πράξεις που εξηγούνται στο Κεφάλαιο 8.

Επιστρέφουμε επιτέλους στο πρόβλημα γιατί το αποτέλεσμα της άθροισης είναι λαθεμένο, και στην ανάλυση του δεύτερου μισού του προγράμματος, όπου βεβαιωθήκαμε ότι βρίσκεται το πρόβλημα.

Το πρώτο μέρος τοποθέτησε το σημαντικότερο byte του αποτελέσματος στη διεύθυνση 42599 και το λιγότερο σημαντικό byte στη διεύθυνση 42598.

Η πρώτη εντολή του δεύτερου μέρους είναι:

LD HL, (42598)

η οποία φορτώνει τα περιεχόμενα της διεύθυνσης που καθορίζει η εντολή στον καταχωρητή L, ενώ ο καταχωρητής H φορτώνεται από την επόμενη διεύθυνση.

Επομένως μετά την εκτέλεση της εντολής ο L θα περιέχει τον 35h και ο H θα περιέχει τον 5Ch, δίνοντας έτσι HL=5C35h ή 23605. Αυτό είναι σωστό, άρα το πρόβλημα δεν βρίσκεται εδώ.

Κατόπιν υπάρχουν 6 NOPs (από το No Operation, δηλαδή δεν εκτελούν καμία λειτουργία), οπότε το πρόβλημα δεν βρίσκεται εδώ.

Τώρα το ζεύγος καταχωρητών DE φορτώνεται με -1000 που είναι FC18h. Αυτό φαίνεται περίεργο αλλά μπορεί να είναι σωστό. Τι συμβαίνει μετά;

Μια υπορουτίνα που ονομάζεται REDN (συντομογραφία του REDuse Number, ελάττωση αριθμού) καλείται. Θα την εξετάσουμε σαν μια μονάδα, τεμαχισμένη σε στάδια.

- 1) LD A,0 A=0
- 2) INC A A=A+1

- 3) ADD HL,DE HL=5C35h και DE=FC18h την πρώτη φορά που καλείται η υπορουτίνα Ο FC18h είναι ο 64536 στο δεκαδικό ή ο -1000 αν είναι σε συμπλήρωμα του 2. $23605+64536=88141$. Ο μεγαλύτερος αριθμός που μπορεί να τοποθετηθεί σε 16 bits είναι ο 65535, οπότε έχουμε μια μεταφορά κρατούμενου στο 65536. $88141-65536=22605$. Ο 1000 επομένως αφαιρέθηκε από το ζεύγος καταχωρητών HL.
- 4) JR C,FNUM Το FNUM (συντομογραφία του Find NUMber, εύρεση αριθμού) είναι το δεύτερο στάδιο. Έτσι κάθε φορά που υπάρχει ένα κρατούμενο για μεταφορά σαν αποτέλεσμα της ADD HL,DE ο καταχωρητής A αυξάνεται κατά 1. Μ' άλλα λόγια ο καταχωρητής A μετράει πόσες φορές ο DE προστίθεται στον HL.
- 5) SBC HL,DE Για να φτάσουμε εδώ θα πρέπει να μην υπάρχει κρατούμενο για μεταφορά από την ADD HL,DE στο στάδιο 3. Επομένως δεν ήταν δυνατό να αφαιρεθεί ο DE από αυτό που απέμεινε στον HL, και το αποτέλεσμα είναι λαθεμένο. Χρησιμοποιώντας την εντολή SBC με αρνητική τιμή το τελικό αποτέλεσμα είναι μια πρόσθεση. (Σας έχει πει ο καθηγητής σας στο σχολείο ότι δύο πλην κάνουν ένα συν; Εδώ είναι η απόδειξη). Το Flag του κρατούμενου είναι γνωστό ότι είναι «0» οπότε δεν θα επηρεάσει τον υπολογισμό.
- 6) DEC A Εφ' όσον ο DE έχει προστεθεί με την προηγούμενη εντολή, το μέτρημα (στον A) πρέπει να ελαττωθεί. Ο A έχει αυξηθεί κατά τον αριθμό που δείχνει πόσες φορές ο DE αφαιρέθηκε από ό,τι υπήρχε στον HL στην αρχή της υπορουτίνας. Ο HL περιέχει τώρα αυτό που υπήρχε στον HL στην αρχή της υπορουτίνας $-(A*DE)$. Στην περίπτωση αυτού του παραδείγματος όπου ο DE ήταν -1000 HL=605 και A=23.
- 7) ADD A,#30 Όταν χρησιμοποιείται ο assembler Highsoft το σύμβολο # σημαίνει ότι τα ψηφία που ακολουθούν θα θεωρηθούν σαν δεκαεξαδικός αριθμός. Ο A ήταν 23 (17h) οπότε μετά την πρόσθεση του 30h(48) γίνεται 71(47h).
- 8) CALL 47962 Αυτή είναι η δοκιμασμένη ROM υπορουτίνα «τύπωσε τον χαρακτήρα του οποίου ο κωδικός βρίσκεται στον καταχωρητή A.
- 9) RET Το τέλος της υπορουτίνας.

Η αιτία για το λάθος στο αποτέλεσμα του παραδείγματος θα πρέπει να είναι τώρα προφανής. Αν το αποτέλεσμα είναι μεγαλύτερο από 9000 ο DE (που είναι -1000) θα αφαιρεθεί από τον HL περισσότερες φορές από τους αριθμούς που υπάρχουν, δηλαδή περισσότερο από 9 φορές. Ο καταχωρητής A επομένως θα περιέχει έναν αριθμό που θα ξεπερνάει τον $30h+9=39h(57)$ όταν κληθεί η υπορουτίνα εκτύπωσης. Οι αριθμοί από 30h μέχρι και 39h είναι οι κωδικοί ASCII των αριθμών 0 μέχρι και 9, με τους οποίους τυπώνεται το αποτέλεσμα, αλλά οι κωδικοί πάνω από το 39h χρησιμοποιούνται για τα σημεία στίξης και τα γράμματα. Ο χαρακτήρας με κωδικό ASCII 71 είναι ο G, επομένως ο G τυπώθηκε στη θέση των αριθμών 2 και 3.

Μπορείτε πιθανά να καταλάβετε τι πρέπει να γίνει για να κάνουμε τη ρουτίνα

να δουλεύει με οποιονδήποτε αριθμό μπορεί να παρασταθεί στα 16 bits. Οι επόμενες εντολές μπορούν να γραφτούν στη θέση των NOPs.

Αν χρησιμοποιείτε τον assembler γράψτε

```
CALL 30004 [ENTER]
```

Μετά πατήστε το L και το [ENTER] για να πάρετε λίστα του προγράμματος. Κατόπιν γράψτε τις εντολές που ακολουθούν στη θέση των πρώτων δύο NOPs και απαλείψτε (delete) τα υπόλοιπα 4 NOPs. Μετά πατήστε A [ENTER] [ENTER] [ENTER] για να ξαναεταφραστεί το πρόγραμμα.

Όσοι από σας χρησιμοποιούν το πρόγραμμα HEX LOADER, γράψτε:

```
RUN και [ENTER]  
SET MEMORY TO A603
```

και

```
START ADDRESS A607
```

ASSEMBLER	HEX
LD. DE, -10000	11 FO DB
CALL REDN	CD 21 A6 END 0386
	MORE? Y/N N

Τρέξτε τώρα το πρόγραμμα όπως και πριν, και θα πρέπει να διαπιστώσετε ότι δουλεύει με δύο οποιουδήποτε αριθμούς που μπορούν να περιέχονται σε 16 bits (<65535).

Μπορείτε τώρα να αλλάξετε το πρώτο μέρος του προγράμματος, το μέρος που προσθέτει τους αριθμούς που αρχίζουν στη διεύθυνση 42550, για να πειραματιστείτε με τις άλλες εντολές πρόσθεσης και αφαίρεσης των 8 bits. Όσο χρησιμοποιείτε πάντα το (HL) για να δείχνετε τη διεύθυνση όπου βρίσκονται οι αριθμοί, και δεν χρησιμοποιείτε της εντολές που έχουν n ή (nn) σαν μέρος της παρουσίασης του assembler, το μόνο που θα χρειαστεί να αλλάξετε είναι το byte που περιέχει την εντολή. Θυμηθείτε ότι η γλώσσα μηχανής δεν είναι σαν τη BASIC, δεν μπορείτε να παρεμβάλλετε εντολές!

Μόλις αισθανθείτε ευτυχισμένοι που καταλαβαίνετε τι συμβαίνει με κάθε μια εντολή, συνεχίστε το διάβασμα.

Θα πρέπει τώρα να έχετε εξοικειωθεί αρκετά με τα μαθηματικά των 8 bits, και αν χρειαστεί θα πρέπει να είναι μέσα στις δυνατότητές σας το γράψιμο ενός προγράμματος που θα προσθέτει δύο τυχόντες αριθμούς ή θα αφαιρεί ένα αριθμό από κάποιον άλλο, χρησιμοποιώντας αποκλειστικά πράξεις των 8 bits. Ίσως εξακολουθεί να σας είναι πρόβλημα ο τρόπος παρουσίασης του αποτελέσματος. Αν πρόκειται να χρησιμοποιηθεί η οθόνη, μια τροποποίηση του προγράμματος που χρησιμοποιείτε τώρα θα σας δώσει το επιθυμητό αποτέλεσμα.

Σε τέτοιου είδους δουλειές τα μαθηματικά των 16 bits αρχίζουν να βρίσκουν τον εαυτό τους. Για να μπορέσετε να προσθέσετε επιτυχώς δύο τυχόντες αριθμούς των

16 bits χρειάζεται να ασχοληθείτε με αποτελέσματα των 17 bits, αλλά για να μπορέσετε να πολλαπλασιάσετε δύο τυχόντες αριθμούς των 16 bits πρέπει να διαθέτετε ένα αποτέλεσμα των 32 bits. Η δυνατότητα μαθηματικών πράξεων στα 16 bits μας επιτρέπει να έχουμε αποτελέσματα των 32 bits, γιατί δεν χρειάζονται άλλες εντολές από εκείνες που χρησιμοποιούνται για τα 24 bits με τη βοήθεια των μαθηματικών των 8 και 16 bits. Αυτό δίνει τη δυνατότητα χρησιμοποίησης αριθμών μέχρι 4294967295 ($2 \uparrow 32$) με τους οποίους μπορούμε να χειριστούμε ο,τιδήποτε εκτός από το εθνικό χρέος.

Ο περιορισμός που τίθεται με το να μην μπορούμε να χρησιμοποιήσουμε μαθηματικά των 16 bits σε αριθμητικά operands εύκολα παρακάμπτεται. Το πρώτο μέρος του προγράμματος στο Σχήμα 6.8 χρησιμοποιούσε μαθηματικά των 8 bits, αλλά μπορούσε να γραφτεί όπως φαίνεται στο Σχήμα 6.10. Αυτό χρησιμοποιεί 21 bytes, που αντιστοιχεί σε οικονομία 1 byte ως προς το αρχικό. Παρ' όλο που είναι σχεδόν σίγουρο ότι το αποτέλεσμα θα χρειαστεί να αποθηκευτεί στη μνήμη για να χρησιμοποιηθεί αργότερα, είναι διαθέσιμο για άμεση χρήση στο ζεύγος καταχωρητών HL, ενώ με τη ρουτίνα των 8 bits το αποτέλεσμα δεν ήταν διαθέσιμο παρά μόνο από τη μνήμη. Αυτό έκανε αναγκαία την εντολή LD HL,(42598) στη ρουτίνα εκτύπωσης του αριθμού, που μπορεί τώρα να παραληφθεί, εξοικονομώντας έτσι τρία ακόμη bytes.

ASSEMBLER	HEX
ORG 42550	SET MEMORY TO A635
ENT 42550	START ADDRESS A636
LD HL,(42598)	21 63 A6 CHECK
LD A,(HL)	7E
INC HL	23
LD E,(HL)	5E
INC HL	23
LD D,(HL)	56
LD H,A	67
CALL 47896	CD 1B BB
LD L,A	6F
ADD HL,DE	19
LD (42598),HL	22 66 A6
CALL 42500	CD 04 A6
RET	C9

Σχήμα 6.10

Μια περαιτέρω εξοικονόμηση στη χρήση της μνήμης μπορεί να γίνει φορτώνοντας, όπου είναι δυνατό, 16 bits αντί για 8 bits. Αυτό κάνει το πρόγραμμα όπως φαίνεται στο Σχήμα 6.11

Το byte μετρητής μειώνεται τώρα σε 19 μόνο.

Αυτό το είδος της υπορουτίνας θα μπορούσε να χρησιμοποιηθεί για να κρατάει το σκορ σε ένα arcade game, ή σε ένα πλήθος άλλων χρήσεων, αλλά όπως είναι τώρα περιορίζεται από το γεγονός ότι δεν μπορεί να κληθεί σαν υπορουτίνα. Αυτό

οφείλεται στο ότι ορισμένες θέσεις χρησιμοποιούνται για τις τιμές που θα προστεθούν και για την αποθήκευση του αποτελέσματος. Αν η υπορουτίνα ήταν να χρησιμοποιηθεί για να κρατάει το σκορ στο παιχνίδι Space Invaders, τότε οι invaders διαφορετικής αξίας θα χρειάζονταν και διαφορετικές ρουτίνες. Έτσι, αν είχαμε invaders αξίας 10,20,50 και 100 και ένα διάστημόπλοιο με αξία 400, η ρουτίνα θα έπρεπε να γραφτεί 5 φορές. Ένα ακόμη πρόβλημα θα ήταν ότι το αποτέλεσμα του ενός CALL θα έπρεπε να προστίθεται στο επόμενο CALL.

ASSEMBLER		HEX	
ORG	42550	SET MEMORY TO	A635
ENT	42550	START ADDRESS	A636
LD	HL, (42596)	21 64 A6	CHECK
LD	A, (42595)	3A 63 A6	
LD	D, A	57	
CALL	47896	CD 18 BB	
LD	E, A	5F	
ADD	HL, DE	19	
LD	(42598), HL	22 66 A6	
CALL	42500	CD 04 A6	
RET		CS	END 041B

Σχήμα 6.11

Αυτό που χρειάζεται είναι μια υπορουτίνα που θα προσθέτει δύο αριθμούς που δίνονται από το πρόγραμμα που την καλεί, και κατόπιν επιστρέφει το αποτέλεσμα, έτοιμο για αποθήκευση από το κυρίως πρόγραμμα. Αυτή μπορεί να γραφτεί χρησιμοποιώντας καταχωρητές για τη μεταφορά πληροφοριών. Χρησιμοποιώντας το παραπάνω σενάριο θα μπορούσαμε να το κατορθώσουμε αναθέτοντας στον HL να κρατάει το σκορ και στον DE να περιέχει την αξία του χτυπημένου εισβολέα. Κατόπιν θα καλείται η υπορουτίνα που θα προσθέτει τους δύο αριθμούς και θα τυπώνει το σκορ στην οθόνη, και το αποτέλεσμα, στο ζεύγος καταχωρητών HL, θα αποθηκεύεται σαν νέο σκορ.

Οι υπορουτίνες στα Σχήματα 6.10 και 6.11 μπορούν ακόμη να χρησιμοποιηθούν για την αφαίρεση δύο αριθμών, αλλά το flag του κρατουμένου θα πρέπει να γίνει «0» πριν γίνει η αφαίρεση. Αν αυτό δεν γίνει τότε ίσως να παρουσιαστεί λαθεμένο αποτέλεσμα. Η εντολή AND A που αναφέρθηκε προηγουμένως χρησιμοποιείται για να κάνει «0» το flag του κρατουμένου, στο παράδειγμα που δίνεται στο Σχήμα 6.12, που απλώς ξανασυντάσσει εκείνο του Σχήματος 6.11.

ASSEMBLER		HEX	
ORG	42550	SET MEMORY TO	A635
ENT	42550	START ADDRESS	A636
LD	HL, (42596)	21 64 A6	CHECK
LD	A, (42595)	3A 63 A6	
LD	D, A	57	
CALL	47896	CD 18 BB	0496

LD	E, A	SF	
AND	A	A7	
SBC	HL, DE	ED	52
LD	(42598), HL	22	66 A6
CALL	42500	CD	04 A6
RET		CS	END 00CS

Σχήμα 6.12

Αυτό θα αφαιρέσει τον DE από τον HL και θα αφήσει το αποτέλεσμα στον HL.

Αναφέρθηκε προηγουμένως ότι, χρησιμοποιώντας εντολές των 16 bits, μπορούμε να πάρουμε αποτελέσματα των 32 bits. Μπορείτε να γράψετε ένα πρόγραμμα που να προσθέτει δύο τυχόντες αριθμούς των 16 bits και να αποθηκεύει το σωστό αποτέλεσμα στη μνήμη σαν αριθμό των 32 bits;

Το αποτέλεσμα θα πρέπει να αποθηκεύεται σε διαδοχικές θέσεις μνήμης, με το ελάχιστο σημαντικό byte αποθηκευμένο στη διεύθυνση 42596 και το σημαντικότερο στη διεύθυνση 42599.

Πριν αρχίσετε να φτιάχνετε το πρόγραμμά σας γράψτε την επόμενη ρουτίνα που θα σας επιτρέψει να παρουσιάσετε το αποτέλεσμα. Θα σας δώσει ακόμη ορισμένες ιδέες για να το γράψετε.

Ορισμένα πράγματα που πρέπει να θυμάστε, για να σας βοηθήσουν, είναι:

- 1) Το αποτέλεσμα μπορεί να είναι μεγαλύτερο από αυτό που μπορεί να συγκρατήσει ένα ζεύγος καταχωρητών. Το κάθε τμήμα του επομένως θα πρέπει να αποθηκεύεται στη μνήμη όταν δεν χρησιμοποιείται.
- 2) Στην πραγματικότητα το πρόγραμμα είναι το πρώτο μέρος του Σχήματος 6.8 με τη διαφορά ότι χρησιμοποιούνται μαθηματικά των 16 bits αντί για 8, και δεν βασίζεται σε εισαγωγή δεδομένων από πληκτρολόγιο.
- 3) Το πρόγραμμα που τυπώνει τον αριθμό κάνει μια αφαίρεση των 32 bits, και μια πρόσθεση των 32 bits θα έχει πολλές ομοιότητες.
- 4) Το πρόγραμμα που τυπώνει τον αριθμό είναι μια έκδοση των 32 bits του δεύτερου μέρους του προγράμματος στο Σχήμα 6.8

Το πρόγραμμα πρόσθεσης που θα εισάγετε στον υπολογιστή πρέπει να αρχίζει στη διεύθυνση 42540 (A62C) και να τελειώνει με την

```
CALL 42400 RET
```

Η λίστα του προγράμματος για τον assembler φαίνεται στο Σχήμα 6.13, και προέρχεται κατ' ευθείαν από τον assembler για μεγαλύτερη ακρίβεια. Αυτό που πρέπει να γράψετε είναι η στήλη στα δεξιά των αριθμών γραμμής, αλλά θυμηθείτε να βάλετε άνω και κάτω τελείες (:) μετά τις labels.

```
Hisoft GENA3 Assembler. Page 1.
Pass 1 errors 00
```

```
10 : FIG 6.14 A SUBROUTINE TO PRINT
32 BIT VALUES IN DECIMAL
```


		20		
A5A0		30	ORG	42400
A5A0		40	ENT	42400
A5A2	2A64A6	50	LD	HL, (42596)
A5A3	222AA6	60	LD	(42538), HL
A5B6	2A66A6	70	LD	HL, (42598)
A5B9	222CA6	80	LD	(42540), HL
A5CC	110036	90	LD	DE, #3600 ; THE LOW WORD AND
A5CF	0165C4	100	LD	BC, #C465 ; THE HIGH WORD OF -1,000,000,000
A5B2	CDF8A5	110	CALL	REDN
A5B5	11001F	120	LD	DE, #1F00 ; THE LOW WORD AND
A5B8	010AFA	130	LD	BC, #FA0A ; THE HIGH WORD OF -100,000,000
A5B8	CDF8A5	140	CALL	REDN
A5BE	11B069	150	LD	DE, #6980 ; THE LOW WORD AND
A5C1	0167FF	160	LD	BC, #FF67 ; THE HIGH WORD OF -10,000,000
A5C4	CDF8A5	170	CALL	REDN
A5C7	11C0BD	180	LD	DE, #BDC0 ; THE LOW WORD AND
A5CA	01F0FF	190	LD	BC, #FFF0 ; THE HIGH WORD OF -1,000,000
A5CD	CDF8A5	200	CALL	REDN
A5D0	116079	210	LD	DE, #7960 ; THE LOW WORD AND THE
A5D3	01FEFF	220	LD	BC, #FFFE ; THE HIGH WORD OF -100,000
A5E6	CDF8A5	230	CALL	REDN
A5D9	11F0DB	240	LD	DE, -10000 ; THE LOW WORD AND
A5DC	01FFFF	250	LD	BC, #FFFF ; THE HIGH WORD OF -10,000
A5DF	CDF8A5	260	CALL	REDN
A5E2	111BFC	270	LD	DE, -1000
A5E5	CDF8A5	280	CALL	REDN
A5E8	119CFF	290	LD	DE, -100
A5EA	CDF8A5	300	CALL	REDN
A5EE	1EF6	310	LD	E, -10
A5F0	CDF8A5	320	CALL	REDN
A5F3	3A2AA6	330	LD	A, (42538)
A5F6	1825	340	JR	PRIN
A5F8	3E00	350	REDN LD	A, 0
A5FA	3C	360	FNUM INC	A
A5FB	2A2AA6	370	LD	HL, (42538)
A5FE	19	380	ADD	HL, DE
A5FF	222AA6	390	LD	(42538)

A602	2A2CA6	400	LD	HL, (42540)
A605	ED4A	410	ADC	HL, BC
A607	222CA6	420	LD	(42540), HL
A60A	3BEE	430	JR	C, FNUM
A60C	2A2AA6	440	LD	HL, (42538)
A60F	ED52	450	SBC	HL, DE
A611	222AA6	460	LD	(42538), HL
A614	2A2CA6	470	LD	HL, (42540)
A617	ED42	480	SBC	HL, BC
A619	222CA6	490	LD	(42540), HL
A61C	3D	500	DEC	A
A61D	C630	510	PRIN	ADD A, #30
A61F	CD5ABB	520	CALL	47962
A622	C9	530	RET	

Σχήμα 6.13

Pass 2 errors: 00

THE CHECK-SUMS REQUIRED BY THE HEX LOADER ARE
03C9 0335 0364 03F6 0562 05D4 0575 04FB 0322 02DD 047D
04F7 0464 00C9

Αν «κολλήσετε» στην προσπάθειά σας να γράψετε το πρόγραμμα πρόσθεσης στα 32 bits, υπάρχει μια δυνατή λύση κάπου αλλού μέσα στο βιβλίο. Προσπαθήστε να παρακολουθήσετε αυτό που κάνει, και κατόπιν ξαναγράψτε το διαφορετικά. Σας προτείνουμε ακόμη να ξαναδιαβάσετε το τελευταίο μέρος αυτού του κεφαλαίου αν εξακολουθείτε να έχετε προβλήματα.

Περίληψη

Ακολουθεί τώρα μια πολύ σύντομη περίληψη των εντολών που μάθατε σ' αυτό το κεφάλαιο.

r = ένας μονός καταχωρητής των 8 bits A, B, C, D, E, H ή L

rr = ένα ζεύγος καταχωρητών που χρησιμοποιείται σαν καταχωρητής των 16 bits.

n = ένας αριθμός των 8 bits

nn = ένας αριθμός των 16 bits

() γύρω από αριθμό ή ζεύγος καταχωρητών = η διεύθυνση στο

PC = Program Counter, Μετρητής Προγράμματος

SP = Stack Pointer, Δείκτης Στοίβας

Η INC r ή DEC r προσθέτει 1 ή αφαιρεί 1 από το r. Και οι δύο επιδρούν στο μηδενικό flag (zero flag) σύμφωνα με το αποτέλεσμα. Αν το αποτέλεσμα είναι 0 τότε το flag γίνεται «1» (set), αλλιώς γίνεται «0» (reset).

Οι INCrr και DEC rr δουλεύουν όπως και οι παραπάνω, αλλά με ένα ζεύγος κα-

ταχωρητών σαν να ήταν ένας καταχωρητής των 16 bits. Οι εντολές αυτές δεν επιδρούν σε κανένα flag.

Ο καταχωρητής A ή συσσωρευτής είναι ο μόνος καταχωρητής που μπορεί να περιέχει το αποτέλεσμα μια μαθηματικής πράξης των 8 bits. Οι μαθηματικές πράξεις των 8 bits είναι:

SUBr SUBn SUB(nn) SUB (HL) που αφαιρούν από τον καταχωρητή A.

ADD A,r ADD A,n ADD A,(nn) ADD A, (HL) που προσθέτουν στον καταχωρητή A.

SBC A,r SBC A,n SBC A,(nn) SBC A,(HL) που αφαιρούν με κρατούμενο (carry) από τον καταχωρητή A.

ADC A,r ADC A,n ADC A,(nn) ADC A,(HL) που προσθέτουν με κρατούμενο (carry) στον καταχωρητή A.

Το ζεύγος καταχωρητών HL πρέπει να χρησιμοποιείται για να συγκρατεί το αποτέλεσμα μιας μαθηματικής πράξης των 16 bits.

Οι μαθηματικές πράξεις των 16 bits είναι:

ADD HL, rr που προσθέτει τα περιεχόμενα του rr στο ζεύγος καταχωρητών HL.

ADC HL, rr που προσθέτει με κρατούμενο στο ζεύγος καταχωρητών HL.

SBC HL, rr που αφαιρεί με κρατούμενο από το ζεύγος καταχωρητών HL.

Αν πρέπει να γίνει δανεισμός μονάδας ή μεταφορά κρατουμένου σε οποιαδήποτε μαθηματική πράξη το flag του κρατουμένου (carry flag) θα γίνει «1» (set), αλλιώς είναι «0» (reset). Αν το αποτέλεσμα μιας μαθηματικής πράξης εκτός από πρόσθεση των 16 bits είναι 0, το μηδενικό flag (zero flag) γίνεται «1», αλλιώς είναι «0».

Η εντολή AND A χρησιμοποιείται για να εξασφαλίσει ότι το flag του κρατουμένου είναι «0» ανεξάρτητα από την προηγούμενη κατάσταση του, όταν η εντολή SBC δεν χρειάζεται το κρατούμενο σε μια πράξη των 16 bits.

FLAGS, ΚΑΤΑΣΤΑΣΕΙΣ ΚΑΙ ΛΗΨΕΙΣ ΑΠΟΦΑΣΕΩΝ

Στο προηγούμενο κεφάλαιο εξετάσαμε με αρκετές λεπτομέρειες τη χρήση του flag του κρατουμένου (carry flag) στις μαθηματικές πράξεις, και δείξαμε πως επιδρά το αποτέλεσμα ενός μαθηματικού υπολογισμού σ' αυτό το flag. Έγινε ακόμη μια σύντομη εισαγωγή στο μηδενικό flag (zero flag).

Και τα δύο αυτά flags είναι του ενός bit και βρίσκονται σε έναν ειδικό καταχωρητή που είναι γνωστός σαν Flag Register (Καταχωρητής των Flags). Όπως όλοι γνωρίζετε σε ένα καταχωρητή υπάρχουν 8 bits, άρα τα υπόλοιπα σε τι χρησιμοποιούνται; Σωστό! Υπάρχουν και άλλα flags που δείχνουν άλλα πράγματα. Ο καταχωρητής των flags είναι φτιαγμένος ως εξής:

S SIGN M/P	Z ZERO Z/NZ	NOT USED	H HALF CARRY	NOT USED	P/V PARITY/ OVERFLOW PE/PO	N ADD/ SUBTRACT	C CARRY C/NC
------------------	-------------------	-------------	--------------------	-------------	-------------------------------------	-----------------------	--------------------

Το γράμμα στην κορυφή του καθενός flag είναι η συντομογραφία των ονομάτων που χρησιμοποιεί η Zilog για τα flags. Θα τα βρείτε στο Παράρτημα με τους κωδικούς εντολών, καθώς και στα περισσότερα βιβλία που έχουν γραφτεί για το Z80. Ακολουθεί το πλήρες όνομα του flag, και στο τέλος υπάρχουν τα μέσα που διαθέτει ο προγραμματιστής για να ελέγχει την κατάσταση (condition) του flag. Θα παρατηρήσετε, πάντως, ότι ο προγραμματιστής έχει πρόσβαση μόνο στα 4 flags. Τα υπόλοιπα χρησιμοποιούνται εσωτερικά από τη CPU.

Φανταστείτε τη CPU του υπολογιστή σας σαν τη μηχανή ενός τραίνου, και το πρόγραμμα σαν τις ράγες πάνω στις οποίες κινείται το τρένο. Το τρένο μπορεί να υποχρεωθεί να αλλάξει πορεία με την αλλαγή ορισμένων κλειδιών, και κάπου στο μήκος της γραμμής θα υπάρχει ένας άνθρωπος σε ένα σπιτάκι που θα τραβήξει ένα μοχλό για να αλλάξει τα κλειδιά. Θα γνωρίζει από πριν τον προορισμό του τραίνου και ποιά κλειδιά πρέπει να αλλάξει για να επιτύχει τη σωστή κατεύθυνση. Ενέργειες αυτού του είδους γίνονται με τις εντολές JUMP και CALL σε ένα πρόγραμμα. Αλλά τι συμβαίνει όταν το τρένο πρέπει να πάρει διαφορετικές κατευθύνσεις ανάλογα με το μέγεθος του φορτίου και τους σταθμούς που πρέπει να περάσει;

Ο μοναδικός τρόπος με τον οποίο ο άνθρωπος στο σπιτάκι μπορεί να ξέρει πως να ρυθμίζει τα κλειδιά είναι να παίρνει πληροφορίες από τον οδηγό του τραίνου. Πρόβλημα! Πως να παίρνει τις πληροφορίες αυτός ο άνθρωπος. Το τρένο δεν πρέπει να σταματήσει αλλιώς θα αργοπορήσει για να μην αναφέρουμε τα καύσιμα που θα χαθούν. Γι' αυτό επινοήθηκε το σύστημα των σημαιών (flags), αλλά ο

οδηγός μπορεί να δίνει μια μόνο πληροφορία κάθε φορά, γιατί μπορεί να ασχολείται με ένα μόνο flag, και μπορεί να πει μόνο ναι ή όχι, γιατί πηγαίνει πολύ γρήγορα και δεν μπορεί να χρησιμοποιήσει τον οπτικό τηλεγράφο.

Με τη γλώσσα μηχανής συμβαίνει το ίδιο πράγμα, και είναι τα flags που διαμορφώνουν τη βάση για τις λήψεις όλων των αποφάσεων, ενώ μπορούν να απαντήσουν μόνο με ναι ή όχι σε μια συγκεκριμένη ερώτηση. Εφ' όσον υπάρχουν 4 μόνο flags που μπορούν να χρησιμοποιηθούν, πρέπει να αφιερώσουμε λίγη σκέψη για να φτάσουμε στην απόφαση που χρειαζόμαστε. Μοιάζει λίγο με το παιχνίδι των είκοσι ερωτήσεων.

Έχουμε ήδη δείξει ότι οι αριθμητικές πράξεις επιδρούν στα flags, αλλά συμβαίνει συχνά, όταν χρειάζεται κάποιος έλεγχος, να μην επιτρέπεται να αλλάξει αυτό που ελέγχεται. Το παράδειγμα του σκορ στο προηγούμενο κεφάλαιο είναι μια τέτοια περίπτωση. Σκεφθείτε την κατάσταση όπου κρατάμε ένα πίνακα με τα υψηλότερα σκορ, και ελέγχουμε το σκορ του τελευταίου παιχνιδιού. Θα ήταν λίγο ηλίθιο αν, ο μόνος τρόπος για να βρούμε αν το νέο σκορ είναι υψηλότερο από εκείνο της κορυφής του πίνακα, ήταν να αφαιρούμε το παλιό υψηλότερο σκορ από αυτό που ελέγχουμε, και μετά να εξετάζουμε το flag του κρατουμένου. Αν δεν είναι «1» (set), δηλαδή είναι στη κατάσταση no carry (NC, όχι κρατούμενο), τότε γνωρίζουμε ότι το νέο σκορ είναι υψηλότερο από το προηγούμενο καλλίτερο.

Πριν κριτικάρουμε τα παραπάνω, σκεφθείτε τι θα συμβεί αν το νέο σκορ πρέπει να τοποθετηθεί στην κορυφή του πίνακα με τα υψηλότερα αποτελέσματα. Υποθέστε ότι το προηγούμενο καλλίτερο αποτέλεσμα ήταν 15575 και το πρόσφατο ήταν 21024, όταν πραγματοποιήθηκε ο έλεγχος. Το σκορ που θα τοποθετηθεί στην κορυφή του πίνακα είναι 5449. Ηλίθιο!

Σε μια τέτοια περίπτωση θα είναι πάντα σχεδόν δυνατό να ξαναβρούμε την τιμή που υπήρχε πριν γίνει ο έλεγχος, αλλά γιατί θα έπρεπε να κάνουμε κάτι τέτοιο; Θα ήταν πολύ καλλίτερα αν υπήρχε ένα είδος «ψεύτικης» (“dummy”) αφαίρεσης που θα έκανε τον έλεγχο. Μια εντολή που θα συγκρίνει και θα δίνει τιμές στα flags, χωρίς να αλλάζει τίποτε άλλο. Υπάρχει, και το σπουδαιότερο είναι πως ονομάζεται compare (σύγκρινε)! Διαβολικά πονηροί αυτοί της Zilog, έτσι!!

Όπως γίνεται συνήθως, η συμβολική εντολή είναι μια συντομογραφία της αντίστοιχης Αγγλικής λέξης, και το compare συντομεύεται σε CP. Ενεργεί ακριβώς όπως η εντολή SUB, αλλά δεν αλλάζει τα περιεχόμενα κανενός καταχωρητή εκτός από τον καταχωρητή των flags. Θα θυμάστε ότι η εντολή SUB λειτουργεί μόνο με τον καταχωρητή A, έτσι και η CP λειτουργεί μόνο με τον καταχωρητή A.

Όλες οι μαθηματικές πράξεις εκτός από την ADD των 16 bits επιδρούν σε όλα τα flags. Το μόνο χρησιμοποιήσιμο flag που επηρεάζεται από την ADD των 16 bits είναι το flag του κρατουμένου (carry flag). Πρόκειται επομένως για την περίπτωση όπου συνήθως δεν χρειάζεται η σύγκριση μετά από μια μαθηματική πράξη, από την οποία εξαρτάται κάποια απόφαση.

Όπως θα περιμένατε, η εντολή compare σχηματίζεται κατά τον ίδιο τρόπο με τις εντολές μαθηματικών πράξεων στα 8 bits. Η μόνη αλλαγή γίνεται στα bits 5,4 και 3. Τα bits αυτά γίνεται 111 αντικαθιστώντας ό,τιδήποτε άλλο υπήρχε προηγουμένως. Για παράδειγμα:

ASSEMBLER	BINARY				
SUB n	10 010 110 n			B	000
CP n	10 111 110 n			C	001
				D	010
SUB r	10 010 r r is, as usual.			E	011
CP r	10 111 r			H	100
				L	101
SUB (HL)	10 010 110			(HL)	110
CP (HL)	10 111 110			A	111

Στο προηγούμενο κεφάλαιο γνωρίσατε δύο flags, το carry flag (flag του κρατούμενου) και το zero flag (μηδενικό flag). Αυτά χρησιμοποιήθηκαν από το πρόγραμμα, για να δίνουν τη δυνατότητα λήψης αποφάσεων που εξαρτιόνταν από την κατάσταση τους. Ο όρος που χρησιμοποιείται για την περιγραφή εντολών που ενεργούν σύμφωνα με την κατάσταση ενός flag, είναι πάντοτε εύκολο να προβλεφτεί. Είναι γνωστές σαν εντολές υπό συνθήκη (conditional instructions). Στη BASIC το "IF αυτό κι αυτό THEN έτσι και έτσι" είναι μια εντολή υπό συνθήκη, και το THEN ακολουθείται τόσο συχνά από ένα GOTO ώστε σε πολλές διαλέκτους BASIC, συμπεριλαμβανομένης της BASIC του Amstrad, επιτρέπεται η παράλειψη του GOTO.

Και στη γλώσσα μηχανής τα πράγματα είναι περίπου ίδια. Η αναλογία μεταξύ του GOTO της BASIC και των εντολών jump της γλώσσας μηχανής έχει τονιστεί ήδη, αλλά αυτή η ομοιότητα προχωράει και πέρα από όσα έχουν ειπωθεί μέχρι τώρα. Στο πρόγραμμα του Σχήματος 6.3 έγινε ένα jump (JR) σύμφωνα με την κατάσταση του zero flag, και το carry flag χρησιμοποιήθηκε κατά τον ίδιο τρόπο, από το πρόγραμμά του Σχήματος 6.5 Είναι σχεδόν ταυτόσημα με τη δομή IF... THEN της BASIC.

Η φαινομενική έλλειψη πραγμάτων που μπορούν να ελεγχθούν δεν είναι στην πραγματικότητα περιορισμός, όπως θα φανεί γρήγορα από μια σύντομη ανάλυση των flags και των πραγμάτων που δείχνουν. Πρώτα θα εξεταστεί το carry flag, γιατί το έχετε ήδη γνωρίσει.

Με την εξαίρεση των εντολών INC και DEC, οποιαδήποτε πράξη που δημιουργεί υπερχείλιση (overflow) σε κάποιον καταχωρητή ή καταχωρητής κάνει «1» το carry flag, και αντίστροφα κάθε πράξη που θα έκανε «1» το carry flag θα το κάνει «0» αν δεν υπάρχει υπερχείλιση.

Για να γίνει αυτό σαφέστερο, δίνονται παρακάτω μερικά παραδείγματα:

```
LD A,0
```

```
DEC A Will re-set the carry flag, but LD A,0
```

```
SUB 1 will set it.
```

```

LD B,156
LD A,100
LD BC,65000
ADD A,B Will set the carry flag, as will LD HL,5536
ADC HL,BC or
LD EC,65000
LD HL,5536 LD BC,5536
SBC HL,BC but LD HL,65000 LD A,225
SBC HL,BC or ADD A,25

```

will leave the flag reset.

Με λίγα λόγια οποιαδήποτε πρόσθεση των 8 bits που δίνει αποτέλεσμα μεγαλύτερο από 255, ή οποιαδήποτε πρόσθεση των 16 bits με αποτέλεσμα μεγαλύτερο από 65535 θα κάνει «1» (set) το carry flag, και το ίδιο θα κάνει οποιαδήποτε αφαίρεση με αποτέλεσμα μικρότερο του μηδενός. Είναι όμοιο με το > (μεγαλύτερο από) ή < (μικρότερο από) της BASIC.

Οι περισσότερες από τις εντολές που επιδρούν στο carry flag κάνουν «1» και το zero flag αν το αποτέλεσμα είναι 0, ή κάνουν «0» (reset) το zero flag αν το αποτέλεσμα δεν είναι 0. Η μόνη εξαίρεση σ' αυτόν τον κανόνα, από τις εντολές που γνωρίσαμε μέχρι τώρα, είναι η ADD HL των 16 bits. Αυτός ο κώδικας λειτουργίας (opcode) αφήνει το zero flag στην ίδια κατάσταση που ήταν πριν εκτελεστεί η ADD HL. Το zero flag μπορεί να θεωρηθεί ότι αντιστοιχεί στο = (ίσον) της BASIC.

Αυτή η ομοιότητα πηγαίνει πολύ σε βάθος όταν η εντολή CP χρησιμοποιείται για να κάνει «1» τα flags. Το zero flag θα είναι πάντοτε «1» αν δεν υπάρχει διαφορά μεταξύ των περιεχομένων του καταχωρητή A και εκείνου με το οποίο συγκρίνεται ο καταχωρητής A, αλλιώς θα είναι «0».

Υπάρχουν επίσης πάρα πολλές εντολές που τροποποιούν το zero flag και όχι το carry flag, αλλά μέχρι τώρα, όπως τονίστηκε και τότε, οι δύο γνωστές περιπτώσεις είναι οι εντολές INC και DEC των 8 bits. Από τώρα και στο εξής, όσο προσθέτονται νέες εντολές, ο τρόπος που αλλάζουν τα flags στα οποία έχει προσπέλαση ο προγραμματιστής, θα εξηγείται λεπτομερώς.

Με τον απλό έλεγχο των carry και zero flags μετά από μια καλομελετημένη σύγκριση (CP) είναι δυνατό να δοθεί απάντηση σε οποιαδήποτε σχεδόν ερώτηση που μπορεί να απαντηθεί με ναι ή όχι. Στην αρχή θα δείτε ότι θα παίρνετε συχνά απρόσμενα αποτελέσματα αλλά, μόλις μάθετε να σκέφτεστε σαν μικροεπεξεργαστής, θα σας γίνει δεύτερη φύση το να κάνετε τη σωστή ερώτηση και να κοιτάτε το σωστό flag.

Προσέξτε τις βιαστικές αποφάσεις, ή τη χρήση δύο ελέγχων όπου διαφορετικά flags μπορεί να δίνουν και τις δύο δυνατές απαντήσεις, απαντώντας σε ένα προσεκτικά μελετημένο έλεγχο! Δείτε το παρακάτω πρόγραμμα που πηδάει σε διάφορες labels ανάλογα με την τιμή που υπάρχει στον καταχωρητή A. Το ζητούμενο είναι

ο κωδικός του γράμματος «Α», αλλά χρειάζονται επίσης και άλλες πληροφορίες.

- 1) Περιέχει ο καταχωρητής A έναν κωδικό ASCII;
- 2) Αν ναι, τότε (THEN) είναι ο κωδικός ενός γράμματος;
- 3) Αν ναι, τότε (THEN) είναι ο κωδικός του πρώτου γράμματος του αλφαβήτου;

CP 128	Όλοι οι ισχύοντες κωδικοί ASCII είναι μικρότεροι από 128
JR NC, NOT ASC	Αν υπάρχει κρατούμενο ο καταχωρητής A πρέπει να περιέχει μία τιμή μικρότερη από 128. Αν δεν υπάρχει, θα πρέπει να είναι 128 ή μεγαλύτερη γιατί ο A δεν περιέχει ένα κωδικό ASCII. Γι' αυτό «πήδηξε» (jump) στη label NOT ASC αν είναι NC (no carry, όχι κρατούμενο).
CP 32	Όλοι οι κωδικοί ASCII είναι πάνω από 32. Αν υπάρχει κρατούμενο ο A θα περιέχει μια τιμή κάτω από 32.
JR C, NOTLET	Αν υπάρχει κρατούμενο, «πήδηξε» στη label NOT LET (NOT LETter, όχι γράμμα).
CP 65	Αυτό θα μπορούσε να γραφτεί σαν CP «Α».
JR Z, ISA	Αν (IF) δεν υπάρχει διαφορά τότε (THEN) το zero flag θα γίνει «1» γι' αυτό, «πήδηξε» (jump) αν είναι μηδέν.

Σ' αυτό το σημείο θα μπορούσατε να σκεφθείτε ότι είναι σωστό να υποθέσουμε ότι ο καταχωρητής A περιέχει ένα κωδικό ASCII κάποιου γράμματος που δεν είναι το A, αλλά θα κάνατε λάθος. Κοιτάξτε στο Παράρτημα III του manual του Amstrad και θα διαπιστώσετε ότι ενώ όλοι οι κωδικοί είναι πάνω από το 31, τα γράμματα που τυπώνονται δεν αρχίζουν παρά από τον κωδικό 65. Γι' αυτό αν αλλάξετε τώρα την εντολή CP 32 σε CP 65, και απαλείψετε την αρχική CP 65, όχι μόνο θα έχετε εξοικονομήσει μια εντολή αλλά θα έχετε αποφύγει και ένα παραπλανητικό αποτέλεσμα. Υπάρχουν ακόμη ορισμένα λάθη. Οι κωδικοί ASCII των γραμμάτων δεν συνεχίζουν μέχρι τον κωδικό 127. Όλοι οι κωδικοί πάνω από το 122 είναι σημεία στίξης, καθώς και οι κωδικοί από το 91 μέχρι το 96. Μπορείτε να προσθέσετε τις αναγκαίες εντολές για να «πηδάτε» στη label NOTLET όταν παρουσιάζονται αυτοί οι κωδικοί; Προσπαθήστε να βρείτε μια λύση πριν συνεχίσετε το διάβασμα. Μια υπόδειξη, θα χρειαστείτε μια ακόμα label, την ISLET (IS LETter, είναι γράμμα).

Θα πρέπει να έχετε προσθέσει τις παρακάτω εντολές:

CP 123	; Ο πρώτος κωδικός που δεν είναι γράμμα
JR NC, NOTLET	; Αν δεν υπάρχει κρατούμενο (No Carry), τότε ο A περιέχει κωδικό πάνω από 123 (το τελευταίο γράμμα)
CP 91	; Ο πρώτος αριθμός που δεν αντιστοιχεί σε γράμμα μετά τα ΚΕΦΑΛΑΙΑ (CAPITALS)
JR C, ISLET	; Αν υπάρχει κρατούμενο τότε ο A πρέπει να περιέχει μια τιμή κάτω από 91. Επομένως, μιας και έχουν αποκλειστεί όλες οι τιμές κάτω από 65, ο A πρέπει να περιέχει τον κωδικό ενός

ΚΕΦΑΛΑΙΟΥ ΓΡΑΜΜΑΤΟΣ.

CR 97 ; Ο κωδικός του πρώτου «μικρού» γράμματος
 JR C, NOTLET ; Έχοντας υπόψη τις προηγούμενες εντολές, η απόφαση αυ-
 τή μπορεί να ληφθεί. Η JR NC, ISLET θα μπορούσε επίσης
 να χρησιμοποιηθεί, *αλλά!*

Αν ο έλεγχος γινόταν για κάποιον που θα πατούσε το «Α» από το πληκτρολόγιο, τι θα συνέβαινε αν το γράμμα που πιεζόταν ήταν το μικρό «α»; Τίποτα! Αυτό είναι που πρέπει να προσέχετε κάθε φορά που ψάχνετε για μια εισαγωγή από το πληκτρολόγιο ή κάποιο κείμενο. Η πρόσθεση μιας ακόμη εντολής θα διορθώσει το πρόγραμμα.

JR Z, ISA Το zero flag θα γίνει «1» αν ο A περιέχει το 97 (τον κωδικό του «α»)

Η ISLET (η label της διεύθυνσης όπου μεταφέρεται ο έλεγχος του προγράμματος αν ο καταχωρητής A περιέχει τον κωδικό ASCII κάποιου γράμματος που δεν είναι «Α» ή «α») θα πρέπει να τοποθετηθεί στο τέλος του προγράμματος. Έτσι αποφεύγεται η πρόσθεση μιας ακόμη jump, γιατί η ροή του προγράμματος θα καταλήξει φυσιολογικά σ' αυτή τη label, αν δεν έχουν γίνει καθόλου jumps, και ο καταχωρητής A θα περιέχει τον κωδικό ASCII κάποιου γράμματος που δεν είναι «Α» ή «α».

Εισάγετε αυτό το πρόγραμμα στον υπολογιστή σας και πειραματιστείτε μαζί του μέχρι να αισθανθείτε ευτυχισμένοι που καταλαβαίνετε πως δουλεύει. Κατόπιν αλλάξτε το ώστε να ψάχνει για άλλο γράμμα. Όσοι έχετε assembler θα μπορέσετε εύκολα να κάνετε τις τροποποιήσεις αφού βρείτε τη νέα κωδικοποίηση. Όσοι χρησιμοποιούν το πρόγραμμα HEX LOADER θα πρέπει να ξαναγράψουν ολόκληρο το πρόγραμμα, να βρουν όλα τα jumps, και να το ξαναεισάγουν στο σύνολό του.

Αυτό το πρώτο μέρος θα επιτρέψει στο πρόγραμμα να δεχθεί δεδομένα από το πληκτρολόγιο και τα αποτελέσματα θα εμφανιστούν στην οθόνη. Χρησιμοποιεί τις ίδιες δυο ρουτίνες του λειτουργικού συστήματος που χρησιμοποιήθηκαν και προηγουμένως. Προσέξτε τον τρόπο με τον οποίο το πρόγραμμα τυπώνει τα μηνύματα, και επιλέγει το μήνυμα που θα τυπώσει. Αυτό αναλύεται αργότερα.

```
Hisoft GENA3 Assembler. Page 1.
pass 1 errors: 00
A5A0          10          ORG      42400
A5A0          20          ENT      42400
A5B0  CD18BB  30  START  CALL    47896
A5A3  0604    40          LD      B, 4
A5A5  FEFC    50          CP      252
A5A7  CB      60          RET     Z
A5A8  FE80    70          CP      128
A5AA  3016    80          JR      NC, NOTASC
A5AC  FE41    90          CP      65
A5AE  3B11   100          JR      C, NOTLET
```

ASB0	2811	110		JR	Z,ISA
ASB2	FE7B	120		CP	123
ASB4	300B	130		JR	NC,NOTLET
ASB6	FE5B	140		CP	91
ASB8	3806	150		JR	C,ISLET
ASBA	FE61	160		CP	97
ASBC	3803	170		JR	C,NOTLET
ASBE	2803	180		JR	Z,ISA
ASCO	05	190	ISLET	DEC	B
ASC1	05	200	NOTLET	DEC	B
ASC2	05	210	NOTASC	DEC	B
ASC3	21EDAA	220	ISA	LD	HL,MESST
ASC6	3E0A	230		LD	A,#0A
ASC8	BE	240	LOOKMS	CP	(HL)
ASC9	23	250		INC	HL
ASCA	20FC	260		JR	NZ.LOOKMS
ASCC	10FA	270		DJNZ	LOOKMS
ASCE	7E	280	PRINT	LD	A,(HL)
ASCF	CDSABB	290		CALL	47962
ASD2	FE0A	300		CP	#0A
ASD4	28CA	310		JR	Z,START
ASD6	23	320		INC	HL
ASD7	18F5	330		JR	PRINT
ASD9	0A	340	MESST	DEFB	#0A
ASDA	41204C45	350		DEFM	'A LE'
ASDE	54544552	360		DEFM	'TTER'
ASE2	20425554	370		DEFM	' BUT'
ASE6	204E4F54	380		DEFM	' NOT'
ASEA	2041	390		DEFM	' A'
ASEC	0DOA	400		DEFW	#0AOD
ASEE	4E4F5420	410		DEFM	' NOT'
ASF2	41204C45	420		DEFM	'A LE'
ASF6	54544552	430		DEFM	'TTER'
ASFA	0DOA	440		DEFW	#0AOD
ASFC	4E4F5420	450		DEFM	'NOT'
A600	41534349	460		DEFM	'ASCII'
A60A	49	470		DEFM	'I'
A605	0DOA	480		DEFW	#0AOD
A607	594F5520	490		DEFM	'YOU'
A60B	50524553	500		DEFM	'PRES'
A60F	53454420	510		DEFM	'SED'
A613	4121	520		DEFM	'A!'
A615	0DOA	530		DEFW	#0AOD

Pass 2 errors: 00

Σχήμα 7.1

Θα παρατηρήσετε ότι δεν υπάρχει ξεχωριστό Hex listing στο πρόγραμμα του σχήματος 7.1 και κανένα από τα υπόλοιπα προγράμματα του βιβλίου δεν θα έχει. Σ' αυτό το στάδιο θα πρέπει να έχετε εξοικειωθεί αρκετά με το HEX LOADER ώστε να μπορείτε να χρησιμοποιείτε το Hex listing από τον assembler. Πρόκειται για την πρώτη από αριστερά στήλη, που αρχίζει — σ' αυτή τη περίπτωση — με CD18BB.

Όπως γίνεται συνήθως, αν χρησιμοποιείτε το HEX LOADER θα εισάγετε το πρόγραμμα σε ζεύγη δεκαεξαδικών (Hex) αριθμών. Οι διευθύνσεις των SET MEMORY, START ADDRESS και των ελέγχων (checksums) δίνονται παρακάτω:

```
SET MEMORY TO A59F
START ADDRESS A5A0
Checksums;
05EA,0380,036C,023A,0567,0395,02DB,0226,02A5,0248,0264,01C8
```

Αν χρησιμοποιείτε τον assembler, δεν χρειάζεται να χωρίζετε τα μηνύματα σε μικρά κομμάτια. Αυτό έχει γίνει γιατί ο assembler δίνει HEX listing των τεσσάρων πρώτων bytes κάθε γραμμής. Η γραμμή 350 θα μπορούσε επομένως να είναι:

```
DEFM "A LETTER BUT NOT A" («ΓΡΑΜΜΑ ΑΛΛΑ ΟΧΙ ΤΟ Α»)
```

και οι γραμμές από 360 μέχρι και 390 δεν θα χρειάζονταν. Το ίδιο ισχύει και για τα άλλα μηνύματα.

Υπάρχουν ορισμένα ενδιαφέροντα σημεία σ' αυτό το πρόγραμμα που αξίζει τον κόπο να εξηγηθούν.

Οποιοδήποτε σχεδόν πρόγραμμα σε γλώσσα μηχανής, αφού αρχίσει να τρέχει δεν υπάρχει τρόπος να το σταματήσετε εκτός κι αν το έχετε εφοδιάσει με ένα δρόμο διαφυγής. Το παραπάνω πρόγραμμα εκτελεί συνεχώς ένα βρόχο. Όταν καλείται από τη BASIC με την εντολή "CALL 42400", ή από τον assembler με το R, το πρόγραμμα θα συνεχίσει να δίνει πληροφορίες για τα πλήκτρα που πιέζονται στο πληκτρολόγιο μέχρι να συμβεί η Δευτέρα Παρουσία, μια διακοπή (breakdown), ένα reset ή διακοπή της λειτουργίας του υπολογιστή σας.

Επομένως έχει προσφερθεί τρόπος διαφυγής. Ο πρώτος έλεγχος που κάνει το πρόγραμμα στον κωδικό που έρχεται στον καταχωρητή A, από τη ρουτίνα WAIT KEY στη διεύθυνση 47896, είναι να δει αν είναι ο 252, ο κωδικός που αντιστοιχεί στο κόκκινο πλήκτρο ESC, κι αν συμβαίνει αυτό τότε γίνεται RETURN στο πρόγραμμα που έκανε την κλήση.

Το επόμενο μέρος του προγράμματος έχει ήδη εξηγηθεί, και οι τέσσερις labels ISLET, NOTLET, NOTASC και ISA τώρα έχουν συμπεριληφθεί. Υπάρχουν 4 μηνύματα που τυπώνονται ανάλογα με την τιμή του κωδικού στον καταχωρητή A, και στο κάθε μήνυμα έχει δοθεί ένας αριθμός από το 1 μέχρι το 4. Ο καταχωρητής B φορτώνεται με το 4 κάθε φορά που τρέχει ο βρόχος του προγράμματος, και μειώνεται ανάλογα με τη label στην οποία «πηδάει» το πρόγραμμα. Αν «πηδήξει» αμέσως στη label ISA τότε ο καταχωρητής B θα περιέχει το 4, ενώ όταν το πήδημα γίνεται στη label ISLET, ο καταχωρητής B θα μειωθεί κατά τρία, και θα περιέχει το 1 όταν το πρόγραμμα φτάσει στη label ISA. Αυτό χρησιμοποιείται για τον κα-

θορισμό του μηνύματος που θα τυπωθεί.

Κατόπιν ανιχνεύεται ο πίνακας μηνυμάτων που αρχίζει στη label MESST, και ο καταχωρητής B ελαττώνεται κάθε φορά που βρίσκεται ένα byte που περιέχει την OAh, μέχρι ο B να περιέχει το 0. Αντί να χρησιμοποιούνται δύο εντολές, χρησιμοποιείται η ημι-αυτόματη εντολή DJNZ.

Αυτή η εντολή σχηματίζεται κατά τον ίδιο ακριβώς τρόπο που σχηματίζεται και η JRNZ, αλλά είναι η πρώτη από πολλές εντολές του Z80 που δεν έχετε γνωρίσει ακόμη, η οποία συνδυάζει εργασίες περισσότερων από μιας κανονικής εντολής.

Η εντολή DJNZ εκτελεί το ανάλογο μιας DEC B ακολουθούμενης από μια εντολή JR NZ, αλλά εξοικονομώντας 1 byte, και χωρίς να επιδρά στα flags.

ASSEMBLER	DECIMAL	HEX	BINARY
-----------	---------	-----	--------

DJNZ n	16 n	10 n	00 001 010 n
--------	------	------	--------------

Όπως γίνεται συνήθως στα relative jumps το n είναι η απόσταση, σε «συμπλήρωμα του 2» (2s complement), από τη διεύθυνση της επόμενης εντολής.

Όταν ο καταχωρητής B φτάνει στο 0, το πρόγραμμα συνεχίζει και το μήνυμα τυπώνεται από το byte που ακολουθεί εκείνο που περιέχει τον OAh που υποχρέωσε τον B να φτάσει στο 0, μέχρι και το τέλος του μηνύματος, που σημαδεύεται από τον επόμενο OAh. Θα παρατηρήσετε ότι οι δείκτες του τέλους (OAh) έχουν μπροστά τους ένα byte που περιέχει τον ODh. Ο συνδυασμός τους έχει σαν αποτέλεσμα ένα carriage return (επιστροφή στην αρχή της γραμμής) και ένα line feed (νέα γραμμή), ετοιμάζοντας έτσι τον δρομέα για τον επόμενο μήνυμα. Ο έλεγχος κατόπιν επιστρέφεται στην αρχή του προγράμματος και η διαδικασία επαναλαμβάνεται για το επόμενο πλήκτρο που θα πατηθεί.

Το επόμενο flag που θα εξεταστεί είναι το sign flag (flag του προσήμου), που δείχνει αν το πρόσημο του αποτελέσματος μιας μαθηματικής πράξης είναι συν ή πλην. Το flag αυτό, καθώς και το Parity/Overflow flag (flag της Ισοτιμίας/Υπερχειλίσσης) του οποίου η λειτουργία θα εξηγηθεί αργότερα, δεν μπορεί να χρησιμοποιηθεί μαζί με την εντολή relative jump (JR). Είναι καιρός επομένως να δώσουμε λεπτομέρειες για όλες τις εντολές που μπορούν να γίνουν υπό συνθήκη (conditional) ανάλογα με την τιμή κάποιου flag.

Μέχρι τώρα, με την εξαίρεση μιας εντολής στο πρόγραμμα του Σχήματος 7.1, οι διακλαδώσεις υπό συνθήκη (conditional branches) έγιναν μόνο με relative jumps. Οι πλήρεις κώδικες λειτουργίας των relative jumps, υπό συνθήκη και μη, φαίνονται στο Σχήμα 7.2.

ASSEMBLER	DECIMAL	HEX	BINARY
DJNZ n	16 n	10 n	00 010 000 n
JR n	24 n	18 n	00 011 000 n
JR NZ, n	32 n	20 n	00 100 000 n
JR Z, n	40 n	28 n	00 101 000 n
JR NC, n	48 n	30 n	00 110 000 n
JR C, n	56 n	38 n	00 111 000 n

Σχήμα 7.2

Όπως ίσως περιμένατε, ένα απόλυτο πήδημα JP (absolute jump JP) μπορεί να γίνει υπό συνθήκη δίνοντας τιμές στα flags, όπως μπορούν και οι εντολές CALL και RET. Οι εντολές αυτές δεν περιορίζονται όπως τα relative jumps, στο να χρησιμοποιούν μόνο τα carry και zero flags. Μπορούν να χρησιμοποιούν οποιοδήποτε από τα προσπελάσιμα flags.

Θα θυμάστε ότι ο καθένας καταχωρητής γενικής χρήσης έχει ένα κωδικό των τριών bits, που χρησιμοποιείται για να τον καθορίζουμε σε όλες τις εντολές που μπορούν να χρησιμοποιήσουν ένα καταχωρητή γενικής χρήσης, αλλάζοντας τρία bits της εντολής, ανάλογα με το ποιός καταχωρητής θα χρησιμοποιηθεί. Το ίδιο σύστημα χρησιμοποιείται από τον Z80 για να καθορίζει τις συνθήκες.

NZ	(not zero, μη μηδενικό)	000
Z	(zero, μηδέν)	001
NC	(no carry, όχι κρατούμενο)	010
C	(carry, κρατούμενο)	011
PO	(parity odd, περιττή ισοτιμία)	100
PE	(parity even, άρτια ισοτιμία)	101
P	(sign positive, θετικό πρόσημο)	110
M	(sign negative, αρνητικό πρόσημο)	111

Σε καθεμία από τις επόμενες εντολές, το cc αντιπροσωπεύει τη συνθήκη, που επιλέγεται από τις παραπάνω, σύμφωνα με την οποία θα διακλαδωθεί το πρόγραμμα. Τα τρία bits που δηλώνονται σαν cc στη δυαδική εντολή, περιλαμβάνουν τρία bits από την παραπάνω, ανάλογα με τη συνθήκη που επιλέχθηκε.

ASSEMBLER	BINARY
JP cc, nn	11 cc 010 nn
CALL cc, nn	11 cc 100 nn
RET cc	11 cc 000

Η JP NC, 47962 γίνεται επομένως 11 010 010 0101, 1010 1011 1011
και η CALL Z, 47960 θα γίνει 11 001 100 0101 1000 1011 1011.

Προφανώς το sign flag (flag του προσήμου) τότε μόνο μπορεί να δείξει σωστά το πρόσημο ενός αποτελέσματος όταν χρησιμοποιείται το συμπλήρωμα του 2. Το flag δεν έχει νόημα σαν sign flag όταν οι υπολογισμοί γίνονται στο δυαδικό χωρίς πρόσημο, παρ' όλο που μπορεί ακόμη να είναι χρήσιμο σαν έλεγχος του bit 7. Για παράδειγμα: ο καταχωρητής A περιέχει τη θετική τιμή 254 μετά από μια μαθηματική πράξη. Το sign flag όμως θα γίνει «1» (set), δείχνοντας εσφαλμένα αρνητικό αποτέλεσμα. Αυτό γίνεται γιατί το sign flag απλώς αντανακλά την κατάσταση του bit 7 του αποτελέσματος. Στο μέλλον, αντί να αναφερόμαστε στο συμπλήρωμα του 2, που είναι μάλλον μακρύς όρος, θα χρησιμοποιούμε τον όρο «προσημασμένος» (“signed”). Αυτό θα σημαίνει ότι το sign flag δείχνει σωστά το πρόσημο του αποτελέσματος μια μαθηματικής πράξης.

Το sign flag επηρεάζεται και από τις 8 μαθηματικές πράξεις, συμπεριλαμβανο-

μένων των εντολών CP (compare), INC και DEC των 8 bits, και των εντολών ADC και SBC των 16 bits. Καμία άλλη από τις εντολές που μάθατε μέχρι τώρα δεν το επηρεάζει κατά κανένα τρόπο. Κατά το σχολιασμό νέων εντολών θα εξηγείται με λεπτομέρειες η επίδρασή τους στα flags όπου αυτό είναι χρήσιμο, ενώ η επίδραση όλων των εντολών στα flags φαίνεται στο Παράρτημα με τους κώδικες λειτουργίας (opcodes).

Το τελευταίο flag που μπορεί να χρησιμοποιήσει ο προγραμματιστής είναι το Parity/Overflow flag (flag ισοτιμίας/υπερχείλισης). Αυτό το flag έχει δύο ξεχωριστές χρήσεις και δεν μπορεί να χρησιμοποιηθεί συγχρόνως και για τους δύο σκοπούς. Είναι είτε overflow flag ή parity flag, ποτέ και τα δύο μαζί.

Όλες οι εντολές που επιδρούν στο zero flag επιδρούν και στο P/V flag, και όλες οι εντολές που έχουν εξηγηθεί μέχρι τώρα σ' αυτό το βιβλίο που επηρεάζουν το zero flag χρησιμοποιούν το P/V σαν overflow flag.

Το Overflow flag δείχνει ότι η εκτέλεση ενός προσημασμένου (signed) υπολογισμού υποχρέωσε το αποτέλεσμα να υπερβεί το εύρος που διατίθεται για την προσημασμένη μορφή. Μπερδευτήκατε; Δεν μας εκπλήσσει, γιατί πρόκειται πιθανώς για την πολυπλοκότερη έννοια μέχρι τώρα, αλλά αφού μπειτε στο νόημα των προσημασμένων υπερχειλίσεων (signed overflows) θα αναρωτιέστε που βρισκόταν το πρόβλημα. Δείτε το επόμενο σύντομο πρόγραμμα:

```
LD A, -80
ADD A, -80
```

Τελειώνοντας ο καταχωρητής A θα περιέχει τον δυαδικό 0110 0000, που είναι ο 96 ή ο 60h. Αυτός είναι θετικός αριθμός (το bit 7 είναι «0») και όχι η σωστή απάντηση. Σ' αυτό το παράδειγμα το carry flag θα γίνει «1», επιτρέποντάς σας να καταλάβετε ότι το αποτέλεσμα προκάλεσε μια υπερχειλίση. Αλλά τι γίνεται με το άθροισμα:

```
LD A, 80
ADD A, 80
```

Αυτή τη φορά ο καταχωρητής A θα περιέχει το αποτέλεσμα 1010 0000 στο δυαδικό, που είναι το -96. Και πάλι δεν είναι η σωστή απάντηση αλλά αυτή τη φορά το carry flag δεν θα είναι «1». Φαινομενικά, χωρίς να το ξέρουμε ενστικτωδώς, δεν υπάρχει ένδειξη ότι το άθροισμα είναι λαθεμένο. Εδώ είναι που κάνει την εμφάνισή του το Overflow flag. Οποιαδήποτε μαθηματική πράξη που υποχρεώνει ένα αποτέλεσμα να υπερβεί το αριθμητικό εύρος μιας εντολής, δηλαδή $-128 \leq n \leq 127$ για πράξη των 8 bits, ή $-32768 \leq n \leq 32767$ για πράξη των 16 bits, θα κάνει «1» το overflow flag.

Ο έλεγχος του overflow flag γίνεται με τις συμβολικές εντολές (mnemonics) PE και PO. Αν έχει συμβεί υπερχειλίση η PO θα κάνει ένα jump, και αν όχι το jump θα γίνει από την PE. Στην πραγματικότητα η PE είναι για την άρτια ισοτιμία (parity even) και η PO για την περιττή ισοτιμία (parity odd), και χρησιμοποιούνται εδώ, γιατί δεν υπάρχουν πρόσθετες συμβολικές εντολές για το flag, που να ξεχωρίζουν τη χρήση του σαν overflow flag και δείκτη ισοτιμίας (parity indicator). .I-

σως βοηθηθείτε στο να θυμάστε ποιά συμβολική εντολή θα χρησιμοποιείτε αν παρατηρήσετε ότι η συμβολική εντολή που προκαλεί διακλάδωση αν υπάρχει υπερχείλιση είναι η μόνη που περιέχει το γράμμα O.

Καμία υπερχείλιση δεν μπορεί να υπάρξει με την πρόσθεση δύο αριθμών με διαφορετικά πρόσημα, και μόνο αριθμοί με διαφορετικά πρόσημα μπορούν να προκαλέσουν υπερχείλιση κατά την αφαίρεση.

Η ισοτιμία (parity) καθορίζεται από τον αριθμό των bits που είναι 1 σε ένα byte. Αν ο αριθμός τους είναι άρτιος τότε λέμε ότι η ισοτιμία είναι άρτια (even). Το P/V flag, μετά από μια εντολή που το χρησιμοποιεί σαν parity flag, θα δείχνει την ισοτιμία του byte που ελέγχθηκε. Το flag είναι «1» (set) αν η ισοτιμία είναι περιττή. Καμία από τις εντολές που γνωρίσαμε μέχρι τώρα δεν χρησιμοποιεί το P/V flag για τον έλεγχο της parity. Θα σας επιστήσουμε την προσοχή όταν μια νέα εντολή χρησιμοποιεί το P/V flag σαν parity flag.

Υπάρχουν δύο ακόμη εντολές με τις οποίες θα ασχοληθούμε σ' αυτό το κεφάλαιο, η SCF και η CCF.

Η λειτουργία και των δύο είναι τελείως φανερή. Η SCF είναι συντομογραφία του Set Carry Flag (κάνει «1» το carry flag) και αυτό ακριβώς κάνει όταν εκτελείται. Η CCF είναι συντομογραφία του Complement Carry Flag (συμπλήρωσε το carry flag) και κάνει ακριβώς αυτό. Όχι, δεν το γεμίζει, του αλλάζει την κατάσταση. Αν το carry flag ήταν «1» (set) πριν την εκτέλεση μιας εντολής CCF θα γίνει «0» (reset) κατόπιν, και το αντίστροφο θα συμβεί αν το flag ήταν «0» πριν την εντολή CCF.

Οι κώδικες λειτουργίας είναι οι εξής:

ASSEMBLER	DECIMAL	HEX	BINARY
CCF	63	3F	00 111 111
SCF	55	37	00 110 111

Περίληψη

Ακολουθεί τώρα μια πολύ σύντομη περίληψη των εντολών που μάθατε σ' αυτό το κεφάλαιο:

r= μονός καταχωρητής των 8 bits A,B,C,D,E,H ή L

rr= ζεύγος καταχωρητών που χρησιμοποιείται σαν καταχωρητής των 16 bits

n= αριθμός των 8 bits

nn= αριθμός των 16 bits

() γύρω από αριθμό ή ζεύγος καταχωρητών = περιέχεται στο

cc= μια συνθήκη (condition)

PC= Program Counter, Μετρητής Προγράμματος

SP= Stack Pointer, Δείκτης Στοίβας.

Τα flags που χρησιμοποιούνται είναι τα CARRY, ZERO, SIGN και P/V.

Το P/V flag έχει δύο ξεχωριστές χρήσεις, την Parity και την Overflow.

Το overflow δείχνει ότι το πρόσημο έχει αλλάξει σε μια προσημασμένη αριθμητική πράξη, δίνοντας έτσι λάθος αποτέλεσμα.

Η cc είναι C,NC,Z,NZ,PE,PO,M και P.

Η CP εκτελεί μια ψεύτικη SUB στον καταχωρητή A, κάνει «1» τα flags, αλλά δεν αλλάζει τίποτε άλλο.

Η JR μπορεί να είναι υπό συνθήκη μόνο με το Carry flag ή το Zero flag.

Η DJNZ είναι το ανάλογο μιας DEC B και μιας JR NZ, αλλά δεν επιδρά σε κανένα flag.

Οι CALL JP και RET μπορούν να υποχρεωθούν να εξαρτώνται από οποιοδήποτε χρησιμοποιήσιμο flag.

Οι εντολές LD CALL JP JR ή RET δεν επιδρούν σε κανένα flag.

ΛΟΓΙΚΕΣ ΠΡΑΞΕΙΣ

Η Z80 CPU είναι προικισμένη με μια ομάδα λογικών τελεστών (logical operators) σχεδόν πανομοιότυπη με εκείνη που διαθέτει η BASIC του Amstrad CPC 464. Αυτό δεν είναι και τόσο εκπληκτικό, εφ' όσον ο Z80 είναι αυτός που κάνει όλη τη δουλειά της BASIC όταν τρέχει. Αυτό κάνει πολύ ευκολότερη την εξήγηση των εντολών γλώσσας μηχανής AND OR και XOR, γιατί χωρίς αμφιβολία ήδη γνωρίζετε, αν και χωρίς να το ξέρετε, τον τρόπο που λειτουργούν, εκτός από την επίδραση που έχουν στα flags. Αν δεν έχετε γνωριστεί ακόμη με αυτούς τους τελεστές στον Amstrad, τότε γυρίστε στο Κεφάλαιο 4 σελίδα 18 του βιβλίου Amstrad User Instructions, όπου μπορείτε να διαβάσετε γι' αυτές. Το υπόλοιπο αυτού του κεφαλαίου υποθέτει ότι έχετε γνώση των λογικών εκφράσεων της BASIC του Amstrad.

Οι λογικοί τελεστές AND, OR και XOR ταξινομούνται σαν μαθηματικοί, και μπορούν να ασχοληθούν μόνο με τιμές των 8 bits χρησιμοποιώντας τον καταχωρητή A. Αν κοιτάξετε στους κώδικες λειτουργίας που ακολουθούν, θα διαπιστώσετε ότι σχηματίζονται ακριβώς όπως και οι υπόλοιπες μαθηματικές εντολές των 8 bits, με τα bits 5,4 και 3 να αλλάζουν ανάλογα με την εντολή. Οι συμβολικές εντολές δεν χρειάζεται να αναφέρουν τον καταχωρητή A γιατί, όπως γίνεται και με τη συμβολική εντολή SUB, δεν υπάρχει άλλος καταχωρητής που μπορεί να χρησιμοποιηθεί.

Όλοι οι συμβολικοί τελεστές επιδρούν στα flags, που παίρνουν τιμές ανάλογα με τα περιεχόμενα του καταχωρητή A μετά την εκτέλεση. Προφανώς, εφ' όσον κανένα αποτέλεσμα από ένα AND OR ή XOR δεν μπορεί να δημιουργήσει αποτέλεσμα που να ξεπερνάει το εύρος των 8μπιτων αριθμών, όλες αυτές οι εντολές κάνουν «0» το carry flag. Και επειδή θα ήταν αδύνατο να δημιουργήσουν μια υπερχειλίση, το P/V flag χρησιμοποιείται σαν flag ισοτιμίας (Parity). Το flag του προσήμου (sign flag) θα δείχνει την κατάσταση του bit 7 στον καταχωρητή A μετά την πράξη, και το zero flag θα είναι «1» αν όλα τα bits του καταχωρητή A είναι «0», αλλιώς θα είναι «0».

Πριν μπορέσετε να χρησιμοποιήσετε αποτελεσματικά τους λογικούς τελεστές θα πρέπει να αρχίσετε να σκέφτεστε στο δυαδικό σύστημα. Μόνο τότε γίνονται φανερά οι πολλές και διάφορες χρήσεις τους. Αυτή τη στιγμή είναι απίθανο να σκέφτεστε στο δυαδικό, γι' αυτό είναι πιθανό να μην καταλαβαίνετε τους σκοπούς που εξυπηρετεί αυτό.

Ας θεωρήσουμε το πρόγραμμα στο Σχήμα 7.1. Εδώ πρέπει να γίνουν ξεχωριστοί έλεγχοι για τα μικρά και τα κεφαλαία γράμματα, και για τα κενά μεταξύ των δύο τύπων γραμμάτων. Στην πραγματικότητα όμως η μοναδική διαφορά (στο δυαδικό) μεταξύ των δύο τύπων γραμμάτων είναι η κατάσταση του bit 5. Όλα τα κεφαλαία γράμματα έχουν το bit 5 σαν «0» (reset), και όλα τα μικρά γράμματα έχουν το bit 5 σαν «1» (set). Χρησιμοποιώντας το λογικό τελεστή and μπορούμε να μετα-

ASSEMBLER	DECIMAL	HEX	BINARY
AND n	230	E6	11 100 110
AND r	160 - 167	A0 - A7	10 100 r
XOR n	238	EE	11 101 110
XOR r	168 - 175	AB - AF	10 101 r
OR n	246	F6	11 110 110
OR r	176 - 183	B0 - B7	10 110 r

τρέψουμε ένα μικρό γράμμα σε κεφαλαίο, ενώ με τη χρήση του λογικού τελεστή or ένα κεφαλαίο γράμμα μπορεί να μετατραπεί σε μικρό. Μπορείτε να βρείτε την πλήρη μορφή της εντολής;

Με τις αλλαγές στο πρόγραμμα του Σχ. 7.1 που δίνονται παρακάτω, δίνεται η απάντηση στο προηγούμενο ερώτημα.

Αλλάξτε τη γραμμή 220 σε:

```
ADDRESS          HEX          ASSEMBLER
A5C3             CD 17 A6      CALL EXTRA  Checksum
for this is 01A3.
```

Προσθέστε τα παρακάτω στο τέλος του προγράμματος:

```
A617             CD 5A BB      CALL 47962
A61A             00                          NOP
A61B             00                          NOP
A61C             CD 5A BB      CALL 47962
A61F             3E 20          LD  A,32 ;THE CODE
FOR SPACE
A621             CD 5A BB      CALL 47962
A624             21 ED AA      LD HL,MESST
A627             C9                          RET
```

Οι έλεγχοι (checksums) που χρειάζονται από το πρόγραμμα HEX LOADER για το δεύτερο αυτό τμήμα είναι: 0422, 0463.

Τώρα όταν εκτελείται το πρόγραμμα ο χαρακτήρας που δημιουργείται μέσω του κώδικα που καθορίζεται από το πλήκτρο που πιέζεται θα εμφανιστεί δύο φορές ακολουθούμενος από ένα κενό διάστημα, πριν το πρόγραμμα δώσει την ετυμηγορία του. Τα δύο NOPs σας αφήνουν χώρο για να τοποθετήσετε μια AND OR ή XOR εντολή, και κατόπιν να δείτε το αποτέλεσμα που έχει.

Αλλάξτε πρώτα τα δύο NOPs σε:

```
ADDRESS          HEX          ASSEMBLER
A61A             F6 20          OR #20
```

Ο ευκολότερος τρόπος για να το κάνετε αν δεν έχετε assembler είναι να γράψετε POKE & A61A, &F6:POKE & A61B, &20 σαν άμεση εντολή (direct command). Τώρα εκτελέστε πάλι το πρόγραμμα και δοκιμάστε το πιέζοντας διάφορα κλή-

κτρα, μαζί ή χωρίς το πλήκτρο SHIFT. (Θυμηθείτε να βεβαιωθείτε ότι δεν είναι ενεργό το CAPS LOCK... Γιατί αχ γιατί η Amstrad να μην έχει βάλει ένα φωτάκι στο πλήκτρο CAPS LOCK που να δείχνει πότε είναι σε χρήση):

Θα διαπιστώσετε ότι όλα τα κεφαλαία γράμματα μετατρέπονται σε μικρά, οι αριθμοί δεν αλλάζουν γιατί θεωρούνται σαν μικρά γράμματα, ενώ οι υπόλοιποι κωδικοί αλλάζουν ή δεν αλλάζουν, ανάλογα με το αν το bit 5 ήταν αρχικά «1» (set). Με τη συγχώνευση της εντολής OR#20 στο κυρίως πρόγραμμα, πολλοί από τους ελέγχους που χρησιμοποιούν την εντολή CP γίνονται περιττοί. Η αναθεωρημένη μορφή του προγράμματος στο Σχήμα 7.1 δίνεται στο Σχήμα 8.1. Θα δείτε ότι το sign flag χρησιμοποιήθηκε για να δείχνει αν ο κωδικός δεν είναι ASCII (το bit 7 γίνεται «1» (set), δηλαδή η τιμή στον A είναι 128 ή παραπάνω). Το sign flag δεν μπορούσε να χρησιμοποιηθεί προηγουμένως γιατί δεν θα μπορούσε να δείξει την κατάσταση του bit 7 του κωδικού του πλήκτρου που πατήθηκε χωρίς μίαν εντολή CP 0, πράγμα που θα πρόσθετε ένα byte στο μήκος του προγράμματος, μιας και το sign flag δεν μπορεί να ελεγχθεί από την εντολή JR. Τώρα που χρησιμοποιείται ο λογικός τελεστής, η εντολή ελέγχου που ήταν απαραίτητη στο προηγούμενο πρόγραμμα μπορεί να παραληφθεί, έχοντας έτσι καθαρή οικονομία ενός byte μετά την αντικατάσταση της JR από την JP.

Hisoft GENA3 Assembler. Page 1.

Pass 1 errors:00

```

          1 ; FIG 8,1
          2 ; AMMENDED VERSION OF PROGRAM
            IN FIG 7,1
A5A0      10 :          ORG          42400
A5A0      20          ENT          42400
A5A0  CD18BB  30  START  CALL          47896
A5A3  0604    40          LD          B,4
A5A5  FEFC    50          CP          252
A5A7  CB      60          RET          Z
A5A8  F620    90          OR          #20
A5AA  FAB9 AS 100        JR          M,NOTASC
A5AD  FE7B   120        CP          123
A5AF  3007   130        JR          NC,NOTLET
A5B1  FE61   160        CP          97
A5B3  3803   170        JR          C,NOTLET
A5B5  2803   180        JR          Z,ISA
A5B7  05     190  ISLET  DEC          B
A5B8  05     200  NOTLET DEC          B
A5B9  05     210  NOTASC DEC          B
A5BA  21E4AA 220  ISA    LD          HL,MESST
A5BD  3E0A   230          LD          A,#0A
A5BF  BE     240  LOOKMS CP          (HL)
ASC0  23     250          INC          HL
ASC1  20FC   260        JR          NZ,LOOKMS
ASC3  10FA   270          DJNZ         LOOKMS

```

A5C5	7E	280	PRINT	LD	A.(HL)
A5C6	CD5ABB	290		CALL	47962
A5C9	FEOA	300		CP	#OA
A5CB	28D3	310		JR	Z.START
A5CD	23	320		INC	HL
A5CE	1BF5	330		JR	PRINT
A5D0	0A	340	MESST	DEFB	#OA
A5D1	41204C45	350		DEFM	'A LE'
A5D5	54544552	360		DEFM	'TIER'
A5D9	20425554	370		DEFM	' BUT'
A5DD	204E4F54	380		DEFM	' NOT'
A5E1	2041	390		DEFM	' A'
A5E3	0DOA	400		DEFW	#OAOB
A5E5	4E4F5420	410		DEFM	'NOT'
A5E9	41204C45	420		DEFM	'A LE'
A5ED	54544552	430		DEFM	'TIER'
A5F1	0DOA	440		DEFW	#OAOB
A5F3	4E4F5420	450		DEFM	'NOT'
A5F7	41534349	460		DEFM	'ASCII'
A5FB	49	470		DEFM	'I'
A5FC	0DOA	480		DEFW	#OAOB
A5FE	594F5520	490		DEFM	'YOU'
A602	50524553	500		DEFM	'PRES'
A606	53454420	510		DEFM	'SED'
A60A	4121	520		DEFM	'A!'
A60C	0DOA	530		DEFW	#OAOB

Pass 2 errors:00
Table used: 110 from 184
Executes: 43700

Σχήμα 8.1

Checksums: 0582, 05B8, 0215, 04B6, 0439, 02A7, 022B, 02A2, 0251, 0268, 020D, 0608, 0278

Στη θέση της εντολής OR θα μπορούσε να έχει χρησιμοποιηθεί η AND, αλλάζοντας έτσι τα μικρά γράμματα σε κεφαλαία, μαζί με τις κατάλληλες μετατροπές που χρειάζονται. Σ' αυτή τη περίπτωση η AND#DF θα αφαιρούσε οποιοδήποτε bit 5 που θα ήταν «1» (set).

Αν η OR στο πρόγραμμα του Σχήματος 8.1 αλλαχθεί με την XOR και προστεθούν οι αλλαγές που δόθηκαν προηγουμένως για την εμφάνιση του χαρακτήρα που αντιπροσωπεύεται από τον κωδικό πριν και μετά τη λογική πράξη, θα διαπιστώσετε ότι τα κεφαλαία μετατρέπονται σε μικρά, και αντιστρόφως. Προσέξτε να μην πατήσετε κάποιο μη αλφαβητικό πλήκτρο, γιατί η XOR θα μετατρέψει ορι-

σμένα από αυτά σε κωδικούς ελέγχου (control codes).

Η εντολή AND χρησιμοποιείται συνήθως για να «καλύπτει» (“mask”) τα bits. Ο όρος mask χρησιμοποιείται όταν κάποιο bit ή bits αγνοούνται, ή γίνονται ασήμαντα. Για παράδειγμα, ένα πρόγραμμα όπου το κάθε γράμμα του αλφαβήτου να αντιπροσωπεύεται από έναν αριθμό, με το «Α» σαν 1 μέχρι το «Ζ» σαν 26, χωρίς να γίνεται διαφοροποίηση μεταξύ κεφαλαίων και μικρών. Εδώ η ευκολότερη λύση θα ήταν να καλύψουμε τα τρία bits στην κορυφή του κωδικού του γράμματος με την AND % 00011111.

Η εντολή OR έχει το αντίθετο αποτέλεσμα, και θα μπορούσε να χρησιμοποιηθεί στο «ξεσκεπάσμα» των bits που καλύπτονται από μια AND. Μια από τις πιο κοινές χρήσεις είναι να κάνουμε το γράψιμο στην οθόνη να γίνεται με τη μορφή “over”, δηλαδή, τα bits οποιουδήποτε χαρακτήρα υπήρχε προηγουμένως σε κάποιο τετραγωνάκι της οθόνης να αλλάζουν μόνον όταν καλύπτονται. Μια άλλη κοινή χρήση είναι το ξεσκεπάσμα καλυμμένων bits (masked bits) ή η μετατροπή τιμών. Για παράδειγμα, τα προγράμματα που χρησιμοποιήθηκαν για την εκτύπωση αριθμών που περιείχονταν σε καταχωρητές ή στη μνήμη, που άρχιζαν με το πρόγραμμα του Σχήματος 6.5 και έφταναν μέχρι το πρόγραμμα του Σχήματος 6.14, χρησιμοποιούσαν όλα την εντολή ADD A, # 30 για να μετασχηματίσουν έναν αριθμό στον αντίστοιχο του κωδικό ASCII. Αυτή έκανε στην πραγματικότητα την ίδια δουλειά με την εντολή OR # 30, η οποία και θα χρησιμοποιόταν αν γνωρίζατε τότε αυτήν την εντολή. Δεν θα εξοικονομούσαμε καθόλου μνήμη, αλλά θα γινόταν σαγέστερο αυτό που συνέβαινε.

Η εντολή XOR, σαν την OR, χρησιμοποιείται συχνά για λειτουργίες που έχουν σχέση με την οθόνη, και ο Amstrad σας την χρησιμοποιεί για εκτύπωση στο “Transparent” (Διαφανές) mode (Κεφάλαιο 5, σελίδα 2 του Amstrad User Instructions). Χρησιμοποιείται επίσης συχνά όταν κάποιο bit ή bits πρόκειται να μετατραπούν στα αντίθετα αυτών που ήταν.

Η λίστα των λογικών τελεστών συνεχίζεται με την εντολή συμπληρώματος (complement instruction), που αντιστοιχεί στη συμβολική εντολή CPL, και υπάρχει και πάλι η άμεση ανάλογη της στην Amstrad BASIC. Η εντολή αυτή κάνει την ίδια δουλειά με το NOT της BASIC. Όπως συμβαίνει και με όλες τις εντολές αυτού του κεφαλαίου, χρησιμοποιεί μόνο τον καταχωρητή A, και μετατρέπει απλώς το κάθε bit στην αντίθετή του κατάσταση. Στην πραγματικότητα το αποτέλεσμα είναι ίδιο ακριβώς με εκείνο της XOR#FF.

Μια γραφική επίδειξη της CPL στην πράξη μπορεί να γίνει από το πρόγραμμα του Σχήματος 8.2, το οποίο βρίσκει το συμπλήρωμα όλων των bits του screen map (χάρτη της οθόνης, περιοχής της μνήμης όπου φυλάγονται όλες οι πληροφορίες που αφορούν το τι θα παρουσιαστεί στην οθόνη). Αυτό θα αντιστρέψει τα bits των paper και pen και εφ’ όσον βρίσκεστε στο mode 2, θα υποχρεώσει την οθόνη να γίνει το αρνητικό του εαυτού της.

Στα διαφορετικά του 2 modes τα πράγματα μερδεύονται κάπως. Αυτό γίνεται γιατί υπάρχουν περισσότερα από δύο διαθέσιμα χρώματα, ενώ στο mode 2, το paper αντιστοιχεί στο 0 και το pen στο 1 και ο διαχωρισμός γίνεται από 1 bit σε ένα byte. Επομένως 1 byte ελέγχει 8 pixels στην οθόνη, όπου το κάθε bit είναι 1 όταν το pixel αντιστοιχεί στο ink 1 και 0 για το pixel που αντιστοιχεί στην τιμή 0. Αντιστρέφοντας επομένως αυτά τα bits με μια εντολή CPL θα κάνει όλα τα ink 1 ink 0

και όλα τα ink 0 ink 1.

Στο mode 1 υπάρχουν τέσσερα χρώματα που πρέπει να διαχωρίζονται μεταξύ τους, και αυτό απαιτεί δύο bits για να ορίζουν το χρώμα του καθενός pixel, 00 για ink0, 01 για ink 1, 10 για ink 2 και 11 για ink 3. Το κάθε byte μπορεί επομένως να ελέγχει μόνο 4 pixels. Αν το χρώμα PEN είναι 1, και το χρώμα PAPER 0, μετά τη συμπλήρωση τα bits PAPER θα γίνουν 3 και τα PEN 2.

Στο mode 0 τα πράγματα γίνονται ακόμη χειρότερα: έχουμε να κάνουμε με 16 χρώματα, και τώρα το κάθε pixel χρειάζεται τέσσερα bits, οπότε ένα byte μπορεί να ελέγχει δύο μόνο pixels.

Μόλις ανακαλύψατε (αν δεν το ξέρατε ήδη) γιατί η διακριτικότητα (resolution) πέφτει όσο τα χρώματα αυξάνονται. Δυστυχώς τα bits σε ένα byte έχουν τη σειρά που θα περιμένατε μόνο στο mode 0, ενώ στα άλλα modes είναι κάπως μπερδεμένα, πράγμα που σίγουρα έγινε σκόπιμα. Στο mode 1, για παράδειγμα,, τα bits 3 και 7 ελέγχουν το αριστερότερο pixel από τα τέσσερα που ελέγχονται από ένα byte, τα bits 2 και 6 το επόμενο, τα bits 1 και 5 το δεύτερο από τα δεξιά και τα bits 0 και 4 το δεξιότερο.

Για να γίνουν τα πράγματα χειρότερα, ακόμη και η σειρά των bytes που ελέγχουν την οθόνη δεν είναι αυτή που θα περιμένατε (εκτός κι αν το μυαλό σας δεν σκέφτεται σωστά!) Λεπτομέρειες και για τη σειρά των bits δίνονται στο screen map στο Παράρτημα.

Το πρόγραμμα που δίνεται στο Σχήμα 8.3 χειρίζεται την οθόνη στο mode 1, μετατρέποντάς, όταν εκτελείται, την οθόνη σε μια σειρά από στήλες πλάτους ενός pixel σε INK0, INK1, INK2 και INK3. Παρατηρήστε πως τα bits για τα χρώματα του ink είναι αποθηκευμένα με το σημαντικότερο bit στη λιγότερο σημαντική θέση. Ο άνθρωπος που επινόησε αυτόν τον screen map πρέπει να είναι σαδιστής.

Hisoft GENA3 Assembler. Page 1.

Pass 1 errors: 00

```
1 ; FIG 8.2
2 ; PROGRAM TO COMPLEMENT THE
  SCREEN MEMORY AREA
ASAO          10          ORG          42400
ASAO          20          ENT          42400
ASAO 2100C0   30          LD           HL, #C000
ASA3 7C      40          LD           A, H
ASA4 B5      50          OR           L
ASA5 CB      60          RET
ASA6 7E      70          LD           A, (HL)
ASA7 2F      80          CPL
ASA8 77      90          LD           (HL), A
ASA9 23     100          INC          HL
ASAA 18F7   110          JR           LOOP
```

Pass 2 errors: 00

THE CHECKSUMS REQUIRED BY THE HEX LOADER ARE

0421, 010F

Σχήμα 8.2

Hisoft GENA3 assembler. Page 1.
 Pass 1 errors:00

```

                                10 ; FIG 8,3
                                20 ; PROGRAM TO CREATE BANDS

DF
                                INK 0,1,2,3
A5A0                            30      ORG      42400
A5A0                            40      ENT      42400
A5A0 2100C0                      50      LD      HL,#C000
A5A3 7C                          60  LOOP  LD      A,H
A5A4 B5                          70      OR      L
A5A5 CB                          80      RET      Z
A5A6 3E5C                        90      LD      A,%01011100
A5A8 77                          100     LD      (HL),A
A5A9 23                          110     INC     HL
A5AA 1BF7                        120     JR      LOOP
Pass 2 errors: 00

```

THE CHECKSUMS REQUIRED BY THE HEX LOADER ARE
 040E, 010F

Σχήμα 8.3

Η εντολή CPL δεν επιδρά σε κανένα από τα flags που μπορούν να ελεγχθούν.

Ο τελευταίος από τους λογικούς τελεστές είναι ο αρνητικός κώδικας λειτουργίας, NEG σαν συμβολική εντολή στον assembler, καθώς και στη γλώσσα των CBers. Η εντολή αυτή είναι η απλούστερη από τους λογικούς τελεστές. Παίρνει την τιμή του καταχωρητή A και της αλλάζει το πρόσημο, βρίσκοντας το συμπληρωμά της ως προς 2. Μ' άλλα λόγια ο A γίνεται 0 πλην A.

Η εντολή NEG επιδρά στα flags ακριβώς σαν να είχε εκτελεστεί μια συνηθισμένη SUB. Τα carry, Zero, Sign και P/V flags επηρεάζονται και το P/V flag χρησιμοποιείται στο Overflow mode.

Οι κώδικες λειτουργίας είναι:

ASSEMBLER	DECIMAL	HEX	BINARY
NEG	237 68	ED 44	11 101 101 01 000 100
CPL	47	2F	00 101 111

Περίληψη

Ακολουθεί τώρα μια πολύ σύντομη περίληψη των εντολών που μάθατε σε τούτο το κεφάλαιο:

r= μονός καταχωρητής των 8 bits A,B,C,D,E,H ή L

rr= ζεύγος καταχωρητών που χρησιμοποιείται σαν καταχωρητής των 16 bits

n= αριθμός των 8 bits

nn= αριθμός των 16 bits

() γύρω από αριθμό ή ζεύγος καταχωρητών = περιέχεται στο
PC= Program Counter, Μετρητής Προγράμματος .
SP= Stack Pointer, Δείκτης Στοίβας

Όλες οι λογικές εντολές εργάζονται με την τιμή του καταχωρητή A.

Όλες οι AND ORκαι XOR μπορούν να χρησιμοποιηθούν με r ή n.

AND. Τα bits που είναι «1» (set) και στον καταχωρητή A και στο operand πριν την εκτέλεση παραμένουν «1» στον καταχωρητή A μετά την εκτέλεση.

Όλα τα άλλα bits στον καταχωρητή A γίνονται «0» (reset).

OR. Τα bits που είναι «1» είτε στον καταχωρητή A ή στο operand πριν την εκτέλεση γίνονται «1» στον καταχωρητή A μετά την εκτέλεση.

XOR. Τα bits που ήσαν «1» είτε στον καταχωρητή A ή στο operand, αλλά όχι και στα δύο, πριν την εκτέλεση γίνονται «1» μετά την εκτέλεση.

Όλες οι παραπάνω εντολές κάνουν «0» (reset) το carry flag, και επιδρούν στα υπόλοιπα flags ανάλογα με το αποτέλεσμα στον καταχωρητή A. Το P/V flag χρησιμοποιείται για τον έλεγχο της ισοτιμίας (parity).

Οι CPL και NEG δεν χρειάζονται operand.

Η CPL αλλάζει τα bits στον καταχωρητή A, ενώ δεν επιδρά σε κανένα από τα flags που μπορούν να ελεγχθούν.

Η NEG φορτώνει στον καταχωρητή A το συμπλήρωμα ως προς 2 της τιμής του. Τα flags επηρεάζονται με τον ίδιο τρόπο που επηρεάζονται και από την εντολή SUB.

ΧΡΗΣΙΜΟΠΟΙΩΝΤΑΣ ΤΗ ΣΤΟΙΒΑ ΜΗΧΑΝΗΣ

Η στοίβα μηχανής (machine stack) έχει ήδη αναφερθεί σε συντομία, στο Κεφάλαιο 5, όπου εξηγήθηκε πως μια εντολή CALL τοποθετούσε τη διεύθυνση επιστροφής στη στοίβα, για να την ξαναπάρει αργότερα με μια εντολή RETURN. Η σημασία της διατήρησης της ισορροπίας, μεταξύ του αριθμού των πραγμάτων που τοποθετούνται στη στοίβα και εκείνων που απαμακρύνονται, είχε τονιστεί, και είχε δείχτει ότι, για να πάει ένα RETURN στη σωστή διεύθυνση, τότε η επόμενη τιμή που θα απομακρυνθεί από τη στοίβα πρέπει να είναι η τιμή που τοποθετήθηκε εκεί από την CALL από την οποία επιθυμούμε να επιστρέψουμε.

Υπάρχουν επίσης εντολές που κάνουν τα πράγματα περισσότερο πολύπλοκα επιτρέποντας στον προγραμματιστή να χρησιμοποιεί τη στοίβα μηχανής για προσωρινή αποθήκευση, κατά τον ίδιο τρόπο που μια εντολή CALL αποθηκεύει προσωρινή τη διεύθυνση επιστροφής για να είναι έτοιμη να παραληφθεί από το σχετικό RETURN. Αν και μπορεί να μην είναι τελείως καταστροφική μια επιστροφή σε λαθεμένη διεύθυνση, ένα RETURN που πηγαίνει σε αυτό που ο επεξεργαστής πιστεύει ότι είναι διεύθυνση επιστροφής, ενώ στην πραγματικότητα είναι απλώς ένας αριθμός αποθηκευμένος στη στοίβα, είναι σχεδόν βέβαιο ότι θα κάνει το σύστημα να «κολλήσει».

Οι παραπάνω παράγραφοι μπορεί να φαίνονται πολύ καταστροφολογικές, αλλά η ανισορροπία της στοίβας είναι ο πιο κοινός λόγος «κολλήματος» της μηχανής, και ορισμένες φορές το παθαίνουν ακόμη και οι πιο πεπειραμένοι προγραμματιστές. Αυτός επίσης είναι ο πρώτιστος λόγος για τον οποίο πρέπει να βεβαιώνετε ότι έχετε κάνει SAVE στο πρόγραμμα γλώσσας μηχανής πριν προσπαθήσετε να το τρέξετε. Τουλάχιστον δεν θα είσαστε υποχρεωμένοι να ξαναρχίσετε από το μηδέν αν «κολλήσει».

Πιθανώς να έχετε καταλάβει ποιές είναι οι συμβολικές εντολές για να τοποθετείτε και να απομακρύνετε δεδομένα από τη στοίβα, από τα αγγλικά ρήματα που αναφέραμε παραπάνω. Οι δύο εντολές είναι:

ASSEMBLER	BINARY
PUSH rr	11 rr0 i01
POP rr	11 rr0 001

Όπως γίνεται συνήθως με τις εντολές που περιέχουν rr, μπορεί να χρησιμοποιηθεί οποιοδήποτε ζεύγος καταχωρητών γενικής χρήσης, και οι δυαδικοί κώδικες που αντικαθιστούν το rr με το καθένα ζεύγος καταχωρητών είναι όπως θα πρέπει να γνωρίζετε πια:

BC=00 DE=01 HL=10

Ακόμη, με τις εντολές PUSH και POP μπορείτε να αποθηκεύσετε στη στοίβα τον καταχωρητή A και τον καταχωρητή των flags, και να τους ξαναπάρτε. Ο μόνος κωδικός που απομένει, ο 11, χρησιμοποιείται για να δηλώνει ότι πρόκειται να χρησιμοποιηθεί ο AF.

Όταν εκτελείται μια PUSH τα περιεχόμενα του οριζόμενου ζεύγους καταχωρητών αντιγράφονται στις επόμενες θέσεις της στοίβας, και ο δείκτης στοίβας (stack pointer) μειώνεται κατά 2, ώστε να δείχνει τη νέα βάση της στοίβας. Η POP κάνει το αντίθετο, αντιγράφοντας τα περιεχόμενα της κορυφής της στοίβας στο οριζόμενο ζεύγος καταχωρητών και αυξάνοντας τον δείκτη στοίβας κατά δύο. Αυτό παρουσιάστηκε με τις εντολές CALL και RET, στο Σχήμα 5.9. Η ίδια ακριβώς διαδικασία συμβαίνει όταν η στοίβα χρησιμοποιείται από τις εντολές PUSH και POP, αλλά αντί να τοποθετείται ή να αφαιρείται από τη στοίβα ο μετρητής προγράμματος (program counter), χρησιμοποιούνται κανονικοί καταχωρητές. Το πρόγραμμα δεν «πηδάει» (jumps).

Έχει ενδιαφέρον να παρατηρήσετε ότι οι δυαδικές εντολές για τις CALL και RET είναι σχεδόν πανομοιότυπες με τις PUSH και POP, πράγμα που θα πρέπει να το περιμένατε, γνωρίζοντας πως χρησιμοποιούν την στοίβα.

```
(CALL 11 001 101  RET 11 001 101)
(PUSH 11 r0 101  POP 11 r0 001)
```

Παρακάτω σας δίνουμε ένα πρόγραμμα που δείχνει το τελευταίο πράγμα που τοποθετείται στη στοίβα και τη διεύθυνση που υποδεικνύει ο δείκτης στοίβας (stack pointer).

Οι έλεγχοι (checksums) που χρειάζονται από το πρόγραμμα HEX LOADER είναι οι εξής:

```
0581, 05B6, 0561, 0580, 04F9, 02C7, 047C, 0403, 03A9, 0300, 013D
```

Το μεγαλύτερο μέρος του προγράμματος θα σας είναι γνωστό, οπότε δεν χρειάζεται να το εξηγήσουμε, αλλά θα δείτε ότι έχουν γίνει ορισμένες αλλαγές. Ο καταχωρητής A αντί να μετράει από το μηδέν, και να πρέπει να αλλάζει για να περιέχει τον κωδικό ASCII του αριθμού που θα τυπωθεί, τώρα αρχίζει περιέχοντας τον #30. Η μοναδική περίπτωση που χρειάζεται να αλλάξει είναι όταν τυπώνεται το υπόλοιπο από τις αφαιρέσεις, και αυτό τώρα πετυχαίνεται με ένα OR.

Μια νέα εντολή έχει εισαχθεί, στη γραμμή 110, αλλά θα καταλάβετε τι κάνει από τη συμβολική εντολή. Ένα ενδιαφέρον σημείο εδώ, που ίσως έχετε προσέξει, είναι ότι η δυαδική εντολή της LD (nn), SP είναι 1110 1101 01 110 011 n n, που ταιριάζει άνετα στην ομάδα εντολών LD (nn), rr που αναφέρονται στο Σχήμα 5.8. Ο κωδικός των 2 bits για το ζεύγος καταχωρητών των 16 bits του SP είναι 11, και αυτό ταιριάζει πάλι με την εντολή LD rr, (nn). Και τότε, μπορεί να ρωτήσετε, σε τι χρησιμοποιείται ο κωδικός των δύο bits 10 που απομένει; Θα ήταν λογικό να αντιστοιχούσε στο ζεύγος καταχωρητών HL, όπως γίνεται και σε όλες τις άλλες περιπτώσεις, αλλά υπάρχει ήδη η εντολή LD HL, (nn) καθώς και η LD (nn), HL.

```

1 ;FIG 9,1
2 ;PROGRAM TO SHOW WHERE THE
  STACK POINTER
3 ;IS POINTING AND THE VALUE
  THAT WILL BE
4 ;ACCESSED BY THE NEXT
  RETRIEVAL FROM THE STACK

A410          10 :          ORG    42000
A410          20 :          ENT    42000
BB5A          30   PRIN    EQU    47962
A410 E1       60   PROG1   POP    HL
A411 E5       70          PUSH   HL
A412 2220A6   80          LD     (42528).HL
A415 CD5AA4   90          CALL   PMESS1
A418 CD22A4  100         CALL   PROG2
A418 ED7320A6 110        LD     (42528).SP
A41F CD61A4  120         CALL   PMESS2
A422 11F0D8  130   PROG2   LD     DE,-10000
A425 CD41A4  140         CALL   REDN
A428 1118FC  150         LD     DE,-1000
A42B CD41A4  160         CALL   REDN
A42E 119CFF  170         LD     DE,-100
A431 CD41A4  180         CALL   REDN
A434 1EF6    190         LD     E,-10
A436 CD41A4  200         CALL   REDN
A439 3A20A6  210         LD     A,(42528)
A43C F630    220         OR     #30
A43E C35ABB  230         JP     PRIN
A441 3E30    240   REDN    LD     A,#30
A443 3C      250   FNUM    INC    A
A444 2A20A6  260         LD     HL,(42528)
A447 19      270         ADD   HL,DE
A44B 2220A6  280         LD     (42528).HL
A44B 38F6    290         JR     C,FNUM
A44D 2A20A6  300         LD     HL,(42528)
A450 ED52    310         SBC   HL,DE
A452 2220A6  320         LD     (42528).HL
A455 3D      330         DEC   A
A456 CD5ABB  340         CALL  PRIN
A459 C9      350         RET
A45A 0607    360   PMESS1  LD     B,7
A45C 216EA4  370         LD     HL,MESS1
A45F 1805    380         JR     MLOOP

```

A461	0604	390	PMESS2	LD	B.4
A463	2175A4	400		LD	HL, MESS2
A466	7E	410	MLOOP	LD	A, (HL)
A467	CD5ABB	420		CALL	PRIN
A46A	23	430		INC	HL
A46B	10F9	440		DJNZ	MLOOP
A46D	C9	450		RET	
A46E	0A0D	460	MESS1	DEFW	#0DOA
A470	2B5350293D	470		DEFM	'(SP)='
A475	2053503D	480	MESS2	DEFM	'SP='

Pass 2 errors: 00

Table used: 132 from 196
Executes: 42000

Σχήμα 9.1

Οι υπολογιστές είναι λογικοί — πράγματι αντιπροσωπεύει το ζεύγος καταχωρητών HL — παρ' όλο που υπάρχει μια συντομότερη εντολή που κάνει ακριβώς την ίδια δουλειά! (λογικό;) Μπορείτε να το ελέγξετε πολύ εύκολα αν χρησιμοποιείτε assembler, αλλάζοντας τη γραμμή 80 σε DEFB # ED και προσθέτοντας τις παρακάτω γραμμές:

```
81 DEFB %01100011
82 DEFB # 34
83 DEFB # AB
```

Τώρα μεταφράστε πάλι το πρόγραμμα και τρέξτε το. Θα διαπιστώσετε ότι δουλεύει ακριβώς όπως και πριν.

Το πρόγραμμα δουλεύει κάνοντας POP την τιμή που βρίσκεται στην κορυφή της στοίβας (ναι, λέγεται κορυφή παρ' όλο που η κορυφή βρίσκεται σε χαμηλότερη διεύθυνση από τη βάση) στο ζεύγος καταχωρητών HL. Αυτό μετακινεί το δείκτη στοίβας ώστε να δείχνει στο προηγούμενο αντικείμενο της στοίβας και, για να αποφύγουμε αλλαγές στη στοίβα, το ζεύγος καταχωρητών HL ξανατοποθετείται με PUSH στη στοίβα. Η στοίβα βρίσκεται τώρα στην ίδια ακριβώς κατάσταση που θρυσκόταν στην αρχή του προγράμματος, αλλά το ζεύγος καταχωρητών HL περιέχει τώρα ένα αντίγραφο της τιμής του στην κορυφή της στοίβας.

Κατόπιν ο HL φορτώνεται στη διεύθυνση μνήμης 42528 και καλείται η υπορουτίνα που τυπώνει το μήνυμα 1, έπειτα καλείται η υπορουτίνα που τυπώνει έναν αριθμό (εδώ ονομάζεται PROG2), η οποία θα τυπώσει την τιμή που αντιγράφηκε από την κορυφή της στοίβας στη μνήμη. Μέχρι τώρα είχαμε δύο CALLs και δύο RETURNS, οπότε ο δείκτης στοίβας δείχνει την ίδια διεύθυνση που έδειχνε στην αρχή του προγράμματος. Η διεύθυνση που φορτώνεται τώρα στη μνήμη είναι έτοιμη να τυπωθεί με την υπορουτίνα εκτύπωσης αριθμού. Πρώτα καλείται η PMESS2 για να τυπώσει το μήνυμα 2 και κατόπιν εκτελείται πάλι η υπορουτίνα εκτύπωσης αριθμού. Αυτή τη φορά η υπορουτίνα εκτύπωσης αριθμού δεν καλείται, οπότε η

RET στο τέλος θα επιστρέψει στη BASIC ή στον assembler ανάλογα με τον τρόπο που προσπελάθηκε η PROG1.

Μια δραματική επίδειξη του πως μπορείτε να χειριστείτε τη στοίβα προς όφελός σας μπορεί να γίνει προσθέτοντας τις επόμενες γραμμές στην αρχή του προγράμματος στο Σχήμα 9.1. Αν χρησιμοποιείτε το πρόγραμμα HEX LOADER χρησιμοποιείτε τη μνήμη από την 41992 και ορίστε σαν αρχική διεύθυνση (start adress) την 41993.

A409		5	ORG	41993
A409		6	ENT	41993
A409	2110A4	7	LD	HL, PROG1
A40C	E5	8	PUSH	HL
A40D	E5	9	PUSH	HL
A40E	E5	10	PUSH	HL
A40F	E5	20	PUSH	HL

Μεταφράστε πάλι το πρόγραμμα αν χρησιμοποιείτε assembler και κατόπιν τρέξτε το. Αν χρησιμοποιήσατε το πρόγραμμα HEX LOADER παρατηρείστε πως τώρα το πρόγραμμα αρχίζει στην A409 (41993) και όχι πια στην A410 (42000).

Αυτή τη φορά το πρόγραμμα θα εκτελέσει το βρόχο πέντε φορές. Οι 4 επιπλέον φορές οφείλονται στα PUSH που γίνονται στη στοίβα, που υποχρεώνουν τα RETURNS να πηγαίνουν στο PROG1 μέχρι να εμφανιστεί η αρχική διεύθυνση επιστροφής.

Οι εντολές που περιγράφηκαν μέχρις εδώ είναι οι μόνες που ενημερώνουν αυτόματα το δείκτη στοίβας όταν χρησιμοποιείται η στοίβα. Υπάρχει όμως ένας αριθμός εντολών που επιτρέπει το χειρισμό της στοίβας, και πληροφορίες να μεταφέρονται από και προς αυτήν.

Οι πρώτοι από τους υπόλοιπους κώδικες λειτουργίας που χρησιμοποιούν τη στοίβα ή το δείκτη στοίβας με τους οποίους θα ασχοληθούμε είναι οι εντολές LD. Κι αυτό γιατί είναι οι απλούστερες, όχι στη λειτουργία τους, μιας και όλες τις εντολές του Z80 θα πρέπει να τις καταλαβαίνετε εύκολα τώρα, αλλά στις χρήσεις για τις οποίες προορίζονται.

Όταν θέτετε σε λειτουργία τον Amstrad CPC 464, ένα τμήμα της cold start (ξεκίνημα εν ψυχρώ) διαδικασίας δίνει σαν αρχική τιμή στον δείκτη στοίβας μια υψηλή διεύθυνση μνήμης, τη διεύθυνση 49144 (BFF8h), και από εδώ η στοίβα πηγαίνει προς τα κάτω. Είναι πιθανό ότι αυτή η διεύθυνση θα γίνει δεκτή σαν βάση της στοίβας, και δεν θα χρειαστεί να αλλάξει. Υπάρχουν, παρ' όλ' αυτά, περιπτώσεις όπου μπορεί να είναι ωφέλιμο, ή ακόμη και ουσιώδες, είτε το να αλλάξουμε τον δείκτη στοίβας ή να τον αποθηκεύσουμε κάπου αλλού. Υπάρχουν επομένως εντολές που το επιτρέπουν, μια από τις οποίες ήδη έχετε χρησιμοποιήσει.

Παρακαλούμε να προσέξετε πως είναι σημαντικό να εξασφαλίζεται ότι ο δείκτης στοίβας ξεκινάει πάντοτε δείχνοντας μια διεύθυνση με άρτιο αριθμό, ιδίως στον Amstrad, όπου μπορούν να αλλάζουν κομμάτια της μνήμης. Αν δεν γίνει αυτό, θα ήταν δυνατό να χαθεί το μισό κάποιου αντικειμένου στη στοίβα, ενώ το υπόλοιπο να παραμείνει. Είναι καλλίτερα να ξεκινάει δείχνοντας μια διεύθυνση που είναι πολλαπλάσιο του 256 μιας και αυτό επιτρέπει τη μέγιστη προς τα κάτω επέκταση πριν συναντηθεί με κάποιο φράγμα σελίδας μνήμης (memory page barrier).

Όλες οι εντολές LD των 16 bits μπορούν να χρησιμοποιηθούν με τον δείκτη στοίβας κάνοντας 11 τα bits 5 και 4. Η πλήρης σειρά των κανονικών εντολών LD είναι:

ASSEMBLER	HEX	BINARY
LD SP, nn	31 n n	00 110 001 n n
LD SP, (nn)	ED 76 n n	11 101 101 01 111 011 n n
LD (nn), SP	ED 73 n n	11 101 101 01 110 011 n n

Περιπτώσεις στις οποίες θα πρέπει να μετακινήσετε τον δείκτη στοίβας θα μπορούσαν να είναι, για παράδειγμα: όταν υπάρχει κάποια εντολή που έχει προτεραιότητα απέναντι σε οτιδήποτε άλλο συμβαίνει σε ένα πρόγραμμα. Το RESET που γίνεται πατώντας τα πλήκτρα [CONTROL] [SHIFT] και [ESC] στον Amstrad, είναι ένα καλό παράδειγμα γι' αυτό. Όταν εκτελείται μια εντολή προτεραιότητας δεν υπάρχει ούτε καν η δυνατότητα να βεβαιωθούμε ότι η στοίβα βρίσκεται σε ισορροπία, και ούτε λόγος για να γνωρίζουμε τι υπάρχει στην κορυφή, οπότε η στοίβα θα πρέπει να ξεκινήσει πάλι από μια γνωστή θέση, πριν χρησιμοποιηθεί σε οτιδήποτε. Εδώ θα χρησιμοποιήσουμε την LD SP, (nn).

Ένα ακόμη καλό παράδειγμα είναι το πρόγραμμα στο Σχήμα 9.2. Έχει ξαναγραφτεί από το πρόγραμμα του Σχήματος 8.3 χρησιμοποιώντας την εντολή PUSH για να γεμίσει την περιοχή μνήμης της οθόνης (screen memory area) σε κλάσμα του χρόνου που χρειάζεται το αρχικό πρόγραμμα. Εδώ ο SP αποθηκεύεται στη μνήμη για να χρησιμοποιηθεί αργότερα, και κατόπιν φορτώνεται με τη διεύθυνση 0. Η διεύθυνση που βρίσκεται από κάτω θα είναι η πρώτη που θα γεμίσει με οποιαδήποτε εντολή PUSH, και εφ' όσον από κάτω βρίσκεται το -1, και ο SP μπορεί να περιέχει μόνο αριθμούς των 16 bits, γίνεται FFFFh, η κορυφή της περιοχής μνήμης της οθόνης. Ο HL κατόπιν φορτώνεται με τον 5C5Ch (όπως και ο A στο Σχήμα 8.3 αλλά δύο φορές) που θα χρησιμοποιηθεί για το γέμισμα.

Hisoft GENA3 Assembler. Page 1.

Pass 1 errors: 00

```

1 ; FIG 9,2
2 ; SCREEN FILL MK.2
88B8      10      ORG  35000
88B8      20      ENT  35000
88B8  ED73D188  30      LD  (SPWD), SP
88BC  310000    40      LD  SP, #0
88BF  215C5C    50      LD  HL, #5C5C
88C2  0E20      60      LD  C, #20
88C4  0600      70  BLOOP LD  B, #0
88C6  E5        80  SLOOP PUSH HL
88C7  10FD      90      DJNZ SLOOP
88C9  0D        100     DEC  C
88CA  20F8      110     JR   NZ, BLOOP
88CC  ED7BD188  120     LD  SP, (SPWD)
88D0  C9        130     RET

```

88D1 0000 140 SPWD DEFW 0

Pass 2 errors: 00

Table used: 48 from 127

Executes: 35000

THE CHECK-SUMS FOR THE HEX LOADER ARE;
03C3 034B 038A

Σχήμα 9.2

Κατόπιν δημιουργείται ένας διπλός βρόχος. Πρόκειται για μια παραλλαγή της τεχνικής που χρησιμοποιείται συχνά όταν ο αριθμός των επαναλήψεων είναι μεγαλύτερος από τον μέγιστο αριθμό που μπορεί να περιέχεται για μέτρηση στους καταχωρητές. Θα συζητηθεί πλήρως στο Κεφάλαιο 16. Σε συντομία αυτό που συμβαίνει είναι με κάθε πέρασμα από το μεγάλο βρόχο BLOOP πραγματοποιείται μια πλήρης σειρά περασμάτων από το μικρό βρόχο SLOOP. (Τα BLOOP (μπλουπ) και SLOOP (σλούπ) δεν σημαίνουν ότι βουλιάζει κάποιο πλοίο!). Σ' αυτή τη περίπτωση ο μικρός βρόχος εκτελείται 256 φορές για το καθένα από τα 32 περάσματα από τον μεγάλο βρόχο. Επομένως η PUSH HL εκτελείται $32 \cdot 256$ φορές που ισούται με 8192, και εφ' όσον το κάθε PUSH γεμίζει δύο θέσεις μνήμης, συμπληρώνονται συνολικά 16384 (4000h) bytes. Τελικά, ο SP παίρνει την προηγούμενη τιμή του και εκτελείται ένα RETurn.

Ο SP μπορεί επίσης να χρησιμοποιηθεί σε όλες τις μαθηματικές εντολές των 16 bits, καθώς και στις INC και DEC των 16 bits. Και πάλι η εντολή σχηματίζεται κάνοντας 11 τα bits 5 και 4. Έτσι για παράδειγμα:

Η ADD HL, DE είναι, στο δυαδικό 00011001 και

η ADD HL, SP επομένως, είναι 00111001

Η DEC BC είναι 00001011 οπότε η DEC SP είναι 00111 011

Η επόμενη εντολή με την οποία θα ασχοληθούμε επιτρέπει στα δεδομένα της κορυφής της στοίβας να ανταλλάγουν με τα δεδομένα του ζεύγους καταχωρητών HL. Όπως πάντα η συμβολική εντολή είναι αυτή που θα περιμένατε. Είναι μια ανταλλαγή (exchange), γι' αυτό το πρώτο τμήμα της συμβολικής εντολής είναι EX, και τα πράγματα που ανταλλάσσονται είναι ο (SP) και ο HL, οπότε ο πλήρης κώδικας λειτουργίας είναι:

ASSEMBLER	HEX	BINARY
EX (SP), HL	E3	11 100 011

Αυτή είναι μια από τις πιο χρήσιμες εντολές που χρησιμοποιούνται στη στοίβα. Μπορεί να χρησιμοποιηθεί για να αλλάξει τις διευθύνσεις των RETurns όταν βρισκόμαστε σε μια υπορουτίνα ή ακόμη και για να προσθέσει νέες υπορουτίνες.

Ας θεωρήσουμε την περίπτωση όπου έχει γραφτεί ένα πρόγραμμα σε ένα τμήμα του οποίου μια υπορουτίνα εκτελεί έναν αριθμό υπολογισμών των 16 bits που το κυρίως πρόγραμμα τους χρειάζεται με συγκεκριμένη σειρά. Φυσικά όλα τα αποτε-

λέσματα υπολογίζονται στο ζεύγος καταχωρητών HL, αλλά ο HL χρειάζεται για να εκτελέσει τον επόμενο υπολογισμό. Έτσι το αποτέλεσμα πρέπει με κάποιον τρόπο να αποθηκευτεί στη μνήμη. Μια εντολή LD (nn), HL θα μας εξυπηρετούσε, αλλά χρησιμοποιεί τρία bytes για την απόθήκευση ενός αποτελέσματος και άλλα τρία κάθε φορά που το αποτέλεσμα ανακτάται. Ο ευκολότερος τρόπος για να το κάνουμε θα ήταν να αποθηκεύσουμε τα αποτελέσματα στη στοίβα, αλλά εκεί υπάρχει η διεύθυνση επιστροφής της υπορουτίνας, και έχετε προειδοποιηθεί για τις επιπτώσεις που έχει η επιστροφή σε αποτέλεσμα! Ποιά είναι λοιπόν η λύση;

Η απάντηση είναι συχνά η ανταλλαγή της διεύθυνσης επιστροφής που βρίσκεται στην κορυφή της στοίβας με το αποτέλεσμα του υπολογισμού, και κατόπιν να κάνουμε PUSH τη διεύθυνση επιστροφής πίσω στην κορυφή. Αυτό χρειάζεται μόνο δύο bytes, ενώ μόνο ένα byte θα χρησιμοποιηθεί για να φέρει το αποτέλεσμα πίσω στο κυρίως πρόγραμμα. Το RETURN θα έχει ήδη αφαιρέσει τη διεύθυνσή του. Το τμήμα του προγράμματος που τοποθετεί τα αποτελέσματα κάτω από το RETURN που βρίσκεται στην κορυφή της στοίβας θα πρέπει να είναι σαν το παρακάτω (υποθέτοντας ότι το αποτέλεσμα βρίσκεται στον HL όταν μπαίνουμε σε τούτο το τμήμα):

```
EX (SP), HL ; Ο HL ΠΕΡΙΕΧΕΙ ΤΩΡΑ ΤΗ ΔΙΕΥΘΥΝΣΗ ΕΠΙΣΤΡΟΦΗΣ (RETURN
ADDRESS) ΚΑΙ ΤΟ ΑΠΟΤΕΛΕΣΜΑ ΒΡΙΣΚΕΤΑΙ ΣΤΗΝ ΚΟΡΥΦΗ ΤΗΣ
ΣΤΟΙΒΑΣ.
PUSH HL ; ΚΑΙ ΤΩΡΑ Η ΔΙΕΥΘΥΝΣΗ ΕΠΙΣΤΡΟΦΗΣ ΒΡΙΣΚΕΤΑΙ ΠΑΛΙ ΣΤΗΝ
ΚΟΡΥΦΗ, ΜΕ ΤΟ ΑΠΟΤΕΛΕΣΜΑ ΑΠΟΚΑΤΩ.
```

Η τελευταία εντολή που περιέχει τον SP είναι περιέργη, από τη σκοπιά της CPU. Είναι η μοναδική εντολή που επιτρέπει τη φόρτωση από καταχωρητή σε καταχωρητή των 16 bits. Η εντολή είναι:

ASSEMBLER	HEX	BINARY
LD SP,HL	F9	11 111 001

Αυτή η εντολή χρησιμοποιείται συχνά όταν η διεύθυνση στην οποία πρέπει να δείχνει ο δείκτης στοίβας έχει προκύψει από υπολογισμό.

Έχετε οικονομία όταν τοποθετείτε πρώτα τα περιεχόμενα του ζεύγους καταχωρητών HL στη μνήμη και κατόπιν φορτώνετε από εκεί τον δείκτη στοίβας.

Σε τούτο το κεφάλαιο είχατε να μάθετε πάρα πολλά, γι' αυτό μην ανησυχείτε αν γυρίζει λίγο το κεφάλι σας. Δείτε τα παραδείγματα των προγραμμάτων που δίνονται, και προσπαθείστε να πειραματιστείτε μόνοι σας. Να θυμάστε να αποθηκεύετε το πρόγραμμα πριν το τρέξετε! Δεν μπορείτε να κάνετε κακό στον Amstrad, οτιδήποτε κι αν κάνει το πρόγραμμά σας. Το χειρότερο που μπορεί να συμβεί είναι να πρέπει να κλείσετε και να ανοίξετε πάλι τον υπολογιστή σας. Μετρείστε πόσες φορές χρησιμοποιείται η στοίβα πριν εκτελέσετε κάποιο πρόγραμμα. Θα πρέπει να υπάρχει ίσος αριθμός από PUSH και POP σε κάθε τμήμα η υπορουτίνα, και σε κάθε CALL πρέπει να αναλογεί ένα RETURN.

Περίληψη

Ακολουθεί τώρα μια πολύ σύντομη περίληψη των εντολών που μάθατε σ' αυτό το

κεφάλαιο:

r = μονός καταχωρητής των 8 bits, A,B,C,D,E,H ή L.

rr = ζεύγος καταχωρητών που χρησιμοποιείται σαν καταχωρητής των 16 bits.

n = αριθμός των 8 bits.

nn = αριθμός των 16 bits.

() γύρω από αριθμό ή ζεύγος καταχωρητών = η διεύθυνση στο

PC = Program Counter, Μετρητής Προγράμματος.

SP = Stack Pointer, Δείκτης Στοίβας.

Η στοίβα μηχανής κατέρχεται τη μνήμη όσο μεγαλώνει. Η κορυφή της στοίβας βρίσκεται στη χαμηλότερη διεύθυνση μνήμης, και καθορίζεται από τον SP.

Η PUSH τοποθετεί τα περιεχόμενα του συγκεκριμένου ζεύγους καταχωρητών στην κορυφή της στοίβας, και ενημερώνει τον SP ώστε να δείχνει στη νέα κορυφή.

Η POP κάνει το αντίθετο.

Οποιοσδήποτε rr γενικής χρήσης ή το ζεύγος καταχωρητών AF μπορεί να τοποθετηθεί με PUSH ή να φύγει με POP από τη στοίβα.

Όλες οι μαθηματικές πράξεις των 16 bits ή οι εντολές LD μπορούν να χρησιμοποιούν τον καταχωρητή SP, καθώς και οι INC και DEC.

Τα περιεχόμενα στην κορυφή της στοίβας μπορούν να ανταλλάγουν (EXchanged) με τα περιεχόμενα του ζεύγους καταχωρητών HL. Ο SP μπορεί να φορτωθεί κατευθείαν από τον HL.

Σε κάθε PUSH πρέπει να αντιστοιχεί ένα POP.

Κάθε CALL πρέπει να έχει ένα RETurn.

Τα περιεχόμενα της στοίβας δεν χρειάζεται να πάνε με POP στον ίδιο καταχωρητή από τον οποίο ήρθαν με PUSH.

ΧΡΗΣΙΜΟΠΟΙΩΝΤΑΣ ΕΝΤΟΛΕΣ ΠΟΥ ΑΣΧΟΛΟΥΝΤΑΙ ΜΕ ΕΝΑ BIT

Η Z80 CPU είναι κάτι το σπάνιο στο πεδίο των 8μπιτων μικροεπεξεργασιών, γιατί επιτρέπει σε ένα bits, είτε είναι στη μνήμη είτε σε καταχωρητή, να γίνει «1» (στο δυαδικό, set) ή «0» (στο δυαδικό, reset), χωρίς να επηρεάζεται καθόλου το υπόλοιπο byte που περιέχει το συγκεκριμένο bit. Υπάρχουν επίσης εντολές για τον έλεγχο της κατάστασης ενός μεμονωμένου bit.

Γιατί να παίζουμε στον κόπο να έχουμε ειδικές εντολές γι' αυτό; Μπορείτε να κάνετε «1» (set) όποιο bit θέλετε με ένα OR, ή «0» (reset) μεταμφιέζοντάς το (masking it) με ένα AND. Και πάλι μπορούν να γίνουν έλεγχοι με τις ίδιες εντολές.

Για παράδειγμα, για να κάνετε «1» το bit 5 στον καταχωρητή A χωρίς να επηρεάσετε κανένα από τα άλλα bits θα πρέπει να χρησιμοποιήσετε την:

```
OR % 00100000
```

Η εντολή που κάνει «0» το bit 5 είναι:

```
AND % 11011111
```

Αν θέλετε να ελέγξετε το bit 5 τότε η

```
AND % 00100000
```

θα έκανε «1» το zero flag αν το bit 5 ήταν μηδέν, ή θα έκανε «0» το zero flag αν το bit 5 δεν ήταν μηδέν.

Σε όλα τα επόμενα παραδείγματα το *tb* είναι η διεύθυνση του byte που ελέγχουμε ή χειριζόμαστε.

Το πρόβλημα με τις εντολές AND και OR είναι ότι πρέπει το byte με το οποίο θα ασχοληθούμε να βρίσκεται στον καταχωρητή A. Αυτό σημαίνει ότι, αν το byte δεν ήταν ήδη εκεί, χρειάζονται αρκετές εντολές για να:

- | | |
|---|---------------|
| 1) Αποθηκεύσουμε τον καταχωρητή A αν χρειάζεται | (PUSH AF) |
| 2) Τοποθετήσουμε το byte στον καταχωρητή A | ((LD A, (tb)) |
| 3) Εκτελέσουμε τη λειτουργία | (AND n) |
| 4) Επιστρέψουμε το τροποποιημένο byte | (LD (tb), A) |
| 5) Αποκαταστήσουμε τον A | (POP AF). |

Γι' αυτή την απλή δουλειά χρησιμοποιούνται συνολικά 10 bytes. Ο αριθμός θα μπορούσε να μειωθεί αν χρησιμοποιούσαμε το ζεύγος καταχωρητών HL έτσι ώστε να δείχνει στη μνήμη ως εξής:

```

PUSH    AF
LD      HL,tb
LD      A,(HL)
AND     n
LD      (HL),A
POP     AF

```

Ο αριθμός των bytes του προγράμματος έπεσε τώρα στα 9 (μεγάλη υπόθεση!). Το πιθανότερο είναι να πέσετε κι εσείς κάτω από την κούραση με τόσο πολλή πληκτρολόγηση.

Το πρόγραμμα για τον έλεγχο ενός bit θα ήταν παρόμοιο, αλλά ο καταχωρητής A δεν θα μπορούσε να αποληκευτεί με την PUSH AF γιατί, με την αποκατάσταση του καταχωρητή A, πριν γίνει έλεγχος στο zero flag για να εξακριβωθεί η κατάσταση του bit που ελέγχεται, θα αποκατασταθεί, και ο καταχωρητής των flags (flag register), καταστρέφοντας έτσι τη μορφή που πήραν τα flags από τον έλεγχο! Η τελική LD (HL),A θα πρέπει να παραληφθεί, γιατί ο έλεγχος θα έχει επιφέρει αλλαγές στο byte του καταχωρητή A, και πρέπει να παραμείνει στη μνήμη χωρίς αλλαγές.

Παρακάτω δίνεται ένα πρόγραμμα σαν παράδειγμα για τον έλεγχο του bit 5. Χρησιμοποιεί 12 bits.

```

LD      (sb),A
LD      HL,tb
LD      A,(HL)
AND     %00100000
LD      A,(sb)

```

Τα προγράμματα αυτά δόθηκαν γιατί, παρ' όλο που δεν θα τα χρησιμοποιήσετε ποτέ, αποτελούν καλά παραδείγματα των παγίδων και των λάκκων του προγραμματισμού, και προσφέρουν τεχνικές για την αντιμετώπισή τους. Χρησιμεύουν επίσης στο να δείχνουν την ανάγκη του ελέγχου των bits, με τις εντολές set και reset.

Οι συμβολικές εντολές που κάνουν «1» και «0» τα bits είναι SET και RESET αντίστοιχα. Ο δυαδικός κώδικας λειτουργίας και οι συμβολικές εντολές του assembler δίνονται παρακάτω. Σε κάθε περίπτωση το b πρέπει να αντικαθίσταται από τον αριθμό του bit με το οποίο ασχολούμαστε. 000 για τα bit 0 (το ελάχιστο σημαντικό) μέχρι 111 για το bit 7.

ASSEMBLER	BINARY
SET b,r	11 001 011 11 b r
RES b,r	11 001 011 10 b r

Το «r» είναι η συνηθισμένη ομάδα (000 για τον B, 111 για τον (HL), 111 για τον A κ.λπ). Όλες οι εργασίες σε επίπεδο bit έχουν μπροστά τους το 11001011 (CBh).

Οι πλήρεις εντολές που κάνουν «1» το bit 5 στον καταχωρητή B και κάνουν «0» το bit 3 στη θέση μνήμης που ορίζεται από τον (HL) είναι επομένως:

ASSEMBLER	HEX	BINARY
SET 5, B	CB EB	11 001 011 11 101 000
RES 3, (HL)	CB 9E	11 001 011 10 011 110

Καμία από τις εντολές που κάνουν τα bits set και reset δεν επιδρούν στα flags με οποιονδήποτε τρόπο.

Η εντολή ελέγχου ενός bit έχει την ίδια δυαδική μορφή με τις εντολές SET και RESET, έχει σαν συμβολική εντολή το BIT και στην πραγματικότητα θα έπρεπε να διαβάζεται σαν BIT? για να θγαίνει νόημα, ακόμη κι αν δεν χρησιμοποιείται το ερωτηματικό.

ASSEMBLER	BINARY
BIT b, r	11 001 011 01 b r

Για να ελέγξουμε το bit 2 στον καταχωρητή H, για παράδειγμα, η εντολή θα ήταν:

ASSEMBLER	HEX	BINARY
BIT 2, H	CB 54	11 001 011 01 010 100

Η εντολή BIT δείχνει την κατάσταση του bit που ελέγχεται με το zero flag, που θα είναι «1» αν το bit είναι 0 ή «0» αν είναι 1. Το carry flag δεν αλλάζει με τις εντολές BIT αλλά όλα τα άλλα flags, εκτός από το zero flag, παίρνουν τιμές που δεν μπορούν να προβλεφθούν.

Μια από τις χρήσεις των εντολών επιπέδου bit είναι να επιτρέπουν το «πακετάρισμα» των πληροφοριών. Πρόκειται για την τεχνική όπου ένα byte χρησιμοποιείται για να περιέχει λεπτομέρειες για περισσότερα από ένα πράγματα. Ένα παράδειγμα είναι οι εγγραφές του προσωπικού (personnel records). Φανταστείτε τη βάση δεδομένων μιας εταιρίας που περιέχουν τις παρακάτω λεπτομέρειες για το προσωπικό:

- 1) Άνδρας / Γυναίκα
- 2) Έγγαμος / Άγαμος
- 3) Παιδιά / Χωρίς παιδιά
- 4) Άδεια οδήγησης / Χωρίς άδεια οδήγησης
- 5) Με μισθό / Με την ώρα
- 6) Κρατάει κλειδιά / Δεν κρατάει κλειδιά
- 7) Ασφαλισμένος / Ανασφάλιστος

Η κάθε μία από τις παραπάνω περιπτώσεις θα μπορούσε να περιγράφεται με ένα μόνο bit, γιατί σε κάθε ερώτηση υπάρχουν δύο μόνο πιθανές απαντήσεις ναι / όχι. Το ναι θα μπορούσε να αντιπροσωπεύεται από το 1 και το όχι από το 0, ενώ 7 bits σε ένα byte αρκούν για να περιέχουν όλες τις παραπάνω πληροφορίες. Το όγδοο bit συχνά φυλάγεται για να δείχνει ότι το byte χρησιμοποιείται.

Θα μπορούσατε να δημιουργήσετε εγγραφές σαν κι αυτές χρησιμοποιώντας τις λογικές πράξεις AND και OR, αλλά θα ήταν δύσκολο να αλλάξετε ένα συγκεκριμένο bit με αυτόν τον τρόπο μετά τη συμπλήρωση της εγγραφής. Το πρόγραμμα που δίνεται στο Σχήμα 10.1 το δείχνει αυτό. Δεν σας προτείνουμε να το εισάγετε στ' αλήθεια, αλλά προσπαθήστε να παρακολουθήσετε αυτό που συμβαίνει. Το πρόγραμμα θα εργαστεί σωστά όταν οι πληροφορίες εισάγονται για πρώτη φορά αλλά για αλλαγές οι εντολές των bits είναι πολύ ευκολότερες.

Σ' αυτό το πρόγραμμα χρησιμοποιήθηκε ένας αριθμός από «βρώμικα κόλλα», καθώς και πολλές από τις τεχνικές και τις εντολές που μάθατε μέχρι τώρα. Δείτε αν μπορείτε να βρείτε πού πηγαίνει το «Y» στο τέλος του μηνύματος 8 στη γραμμή 1140.

Το bit στον καταχωρητή C, στον οποίο ενεργεί η εντολή OR ως προς τον καταχωρητή A όταν η απάντηση είναι «ναι» στην ερώτηση που έχει σχέση με το bit, μετατοπίζεται προς τα αριστερά κατά 1 bit από τις εντολές ADD A, A που βρίσκονται στη label SLA για κάθε ερώτηση, ενώ το ίδιο κόλλο χρησιμοποιείται για να γίνει «1» (set) το carry flag για τις περιπτώσεις που απαντήθηκαν με «Y», κατά την παρουσίαση των εγγραφών.

Hisoft GENA3 Assembler. Page 1.

Pass 1 errors: 00

```

10 ; FIG 10,1
20 ; PROGRAM TO DEMONSTRATE PROBLEMS
30 ; OF USING OR TO SET BITS.
88B8          40          ORG  35000
88B8          50          ENT  35000
BB5A          60 PRINT EQU  47962
BB18          70 GETKEY EQU  47896
88B8          80          LD   HL, FREE
88B8          90          NXTREC CALL CRLF
88BE          CD5689     100         CALL CRLF
88C1          0609      110         LD   B, 9
88C3          CDFB88     120         CALL PR_MSG
88C6          CD6389     130         CALL KEYIN
88C9          FE66      140         CP   "f"
88CB          284E      150         JR   Z, LSTREC
88CD          3E01      160         LD   A, #1
88CF          77        170         LD   (HL), A
88D0          0607      180         LD   B, 7
88D2          0E02      190         LD   C, 2
88D4          CD5689     200         NXTBIT CALL CRLF
88D7          CDFB88     210         CALL PR_MSG
88DA          C5        220         PUSH BC
88DB          060A      230         LD   B, 10
88DD          CDFB88     240         CALL PR_MSG
88E0          C1        250         POP  BC
88E1          CD6389     260         CALL KEYIN
88E4          FE79      270         CP   "y"
88E6          2005      280         JR   NZ, NO
88E8          7E        290         LD   A, (HL)
88E9          B1        300         OR   C

```

88EA	77	310	LD	(HL), A
88EB	1806	320	JR	SLA
88ED	FE6E	330	NO	CP "n"
88EF	2802	340	JR	Z, SLA
88F1	18E1	350	JR	NXTBIT
88F3	79	360	SLA	LD A, C
88F4	87	370	ADD	A, A
88F5	4F	380	LD	C, A
88F6	10DC	390	DJNZ	NXTBIT
88F8	23	400	INC	HL
88F9	18C0	410	JR	NXTREC
88FB	CD6C89	420	PR_MSG	CALL SAVREG
88FE	217689	430	LD	HL, MSGTBL
8901	CB7E	440	FNDMSG	BIT 7, (HL)
8903	23	450	INC	HL
8904	28FB	460	JR	Z, FNDMSG
8906	10F9	470	DJNZ	FNDMSG
8908	CD0F89	480	CALL	NXTCHR
890B	CD7189	490	CALL	RESREG
890E	C9	500	RET	
890F	7E	510	NXTCHR	LD A, (HL)
8910	E67F	520	AND	%01111111
8912	CD5ABB	530	CALL	PRINT
8915	CB7E	540	BIT	7, (HL)
8917	C0	550	RET	NZ
8918	23	560	INC	HL
8919	18F4	570	JR	NXTCHR
891B	21438A	580	LSTREC	LD HL, FREE
891E	CD5689	590	CALL	CRLF
8921	CD5689	600	CALL	CRLF
8924	0608	610	LD	B, B
8926	CDFB88	620	CALL	PR_MSG
8929	0601	630	PR_REC	LD B, 1
892B	E5	640	PUSH	HL
892C	CD5689	650	CALL	CRLF
892F	CD5689	660	CALL	CRLF
8932	CD18BB	670	CALL	GETKEY
8935	E1	680	POP	HL
8936	7E	690	LD	A, (HL)
8937	23	700	INC	HL
8938	A7	710	AND	A
8939	CB	720	RET	Z
893A	87	730	P_ITEM	ADD A, A
893B	CDFB88	740	CALL	PR_MSG
893E	F5	750	PUSH	AF
893F	3007	760	JR	NC, NOT
8941	3E59	770	LD	A, "Y"
8943	CD5ABB	780	CALL	PRINT
8946	1805	790	JR	NXTITM
8948	3E4E	800	NOT	LD A, "N"
894A	CD5ABB	810	CALL	PRINT
894D	04	820	NXTITM	INC B
894E	CD5689	830	CALL	CRLF
8951	F1	840	POP	AF
8952	28D5	850	JR	Z, PR_REC
8954	18E4	860	JR	P_ITEM

8956	F5	870	CRLF	PUSH	AF	
8957	3E0D	880		LD	A,#0D	
8959	CD5ABB	890		CALL	PRINT	
895C	3E0A	900		LD	A,#0A	
895E	CD5ABB	910		CALL	PRINT	
8961	F1	920		POP	AF	
8962	C9	930		RET		
8963	CD18BB	940	KEYIN	CALL	GETKEY	
8966	CD5ABB	950		CALL	PRINT	
8969	F620	960		OR	#20	
896B	C9	970		RET		
896C	E3	980	SAVREG	EX	(SP),HL	
896D	C5	990		PUSH	BC	
896E	F5	1000		PUSH	AF	
896F	E5	1010		PUSH	HL	
8970	C9	1020		RET		
8971	E1	1030	RESREG	POP	HL	
8972	F1	1040		POP	AF	
8973	C1	1050		POP	BC	
8974	E3	1060		EX	(SP),HL	
8975	C9	1070		RET		
8976	A0	1080	MSGTBL	DEFB	#A0	
8977	53454355	1090		DEFM	"SECURITY CLEARED?"	
8988	A0	1100		DEFB	#A0	
8989	4B455920	1110		DEFM	"KEY HOLDER ?	"
899A	A0	1120		DEFB	#A0	
899B	53414C41	1130		DEFM	"SALARIED ?	"
89AC	A0	1140		DEFB	#A0	

Hisoft GENA3 Assembler. Page 3.

89AD	44524956	1150		DEFM	"DRIVING LICENCE ?"	
89BE	A0	1160		DEFB	#A0	
89BF	4348494C	1170		DEFM	"CHILDREN ?	"
89D0	A0	1180		DEFB	#A0	
89D1	4D415252	1190		DEFM	"MARRIED ?	"
89E2	A0	1200		DEFB	#A0	
89E3	4D414C45	1210		DEFM	"MALE ?	"
89F4	A0	1220		DEFB	#A0	
89F5	0A0A	1230		DEFW	#0A0A	
89F7	464F5220	1240		DEFM	"FOR NEXT RECORD PRESS ANY	
8A14	0788	1250		DEFW	#8807	
8A16	4620544F	1260		DEFM	"F TO FINISH OR ANY OTHER K	
8A32	20544F20	1270		DEFM	" TO GO ON"	
8A3B	07A0	1280		DEFW	#A007	
8A3D	20592F4E	1290		DEFM	" Y/N"	
8A41	A0A0	1300		DEFW	#A0A0	
8A43	0000	1310	FREE	DEFW	#0000	

Pass 2 errors: 00

Table used: 257 from 326
Executes: 35000

Σχήμα 10.1

ΠΕΡΙΣΤΡΟΦΕΣ ΚΑΙ ΜΕΤΑΤΟΠΙΣΕΙΣ

Στο πρόγραμμα του Σχήματος 10.1, η εντολή OR είχε ενεργήσει στους καταχωρητές C και A για να κάνει «1» κάποιο bit, πράγμα που σημαίνει ότι η ερώτηση που είχε σχέση με το συγκεκριμένο bit είχε απαντηθεί καταφατικά. Μετά την επιστροφή στη μνήμη του byte που βρισκόταν στον καταχωρητή A, ο καταχωρητής C φορτώθηκε στον καταχωρητή A, ο καταχωρητής A τον πρόσθεσε στον εαυτό του, και το αποτέλεσμα επέστρεψε στον καταχωρητή C. Όλα αυτά έγιναν για να μετακινηθεί απλώς κατά μια θέση προς τα αριστερά το bit που έγινε «1» (set) στον καταχωρητή C, για να μπορεί να γίνει «1» και το επόμενο bit αν χρειάζεται. Η label SLA πήρε αυτό το όνομα γιατί η ενέργεια που κάνει αυτό το τμήμα του προγράμματος είναι μια αριστερή αριθμητική μετατόπιση (Shift Left Arithmetic).

Κάθε φορά που ένας δυαδικός αριθμός προστίθεται στον εαυτό του, πολλαπλασιαζόμενος στην πραγματικότητα επί 2, η σειρά των bits παραμένει η ίδια αλλά μετατοπίζεται κατά μια θέση προς τα αριστερά. Το ίδιο συμβαίνει σε οποιοδήποτε αριθμητικό σύστημα όταν ένας αριθμός πολλαπλασιάζεται με τη βάση του συστήματος.

Για παράδειγμα:

Στο Δυαδικό (βάση 2) $1110110 * 10 = 10101100$ (το 10b είναι το 2 στο δεκαδικό).

Στο Δεκαδικό (βάση 10) $1234567 * 10$ είναι 12345670.

Στο Hex (βάση 16) $789ABCD * 10 = 789ABCD0$ (το 10 h είναι το 16 στο δεκαδικό).

Είναι μάλλον ενοχλητικό αν κάθε φορά που χρειάζεται να μετατοπιστούν τα περιεχόμενα ενός byte, πρέπει να χρησιμοποιείται ο καταχωρητής A. Υπάρχει επίσης το πρόβλημα του τι θα κάνουμε όταν χρειαζόμαστε μια μετατόπιση προς τα δεξιά. Αυτό θα ήταν εξυπηρετικό στο πρόγραμμα του Σχήματος 10.1 γιατί θα επέτρεπε στις πληροφορίες να παρουσιάζονται με την ίδια σειρά που εισάγονται. Τώρα η ρουτίνα εκτύπωσης χρησιμοποιεί πάλι την εντολή ADD A,A, αυτή τη φορά στη label P-ITEM, για να μετατοπίσει τα bits στο κρατούμενο, σημαίνοντας ναι ή όχι.

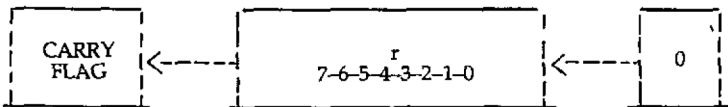
Στην πραγματικότητα υπάρχουν τρόποι με τους οποίους θα ήταν δυνατή η παρουσίαση των πληροφοριών με την ίδια σειρά που δόθηκαν. Η μετατόπιση του byte των δεδομένων στον καταχωρητή A αμέσως μετά το OR στη θέση του καταχωρητή C θα ήταν ένας τρόπος, αλλά θα δημιουργούσε άλλα προβλήματα, γιατί η μετατόπιση πρέπει να γίνεται ακόμη και όταν, μετά από αρνητική απάντηση, δεν ενεργεί κανένα OR.

Αυτό που χρειάζεται πραγματικά είναι μια ομάδα εντολών που θα επιτρέπουν τη μετατόπιση των καταχωρητών, όχι μόνο για να είναι δυνατή η αντιμετώπιση των προβλημάτων που περιγράψαμε παραπάνω, αλλά και γιατί διευκολύνεται ο υπολογισμός των διαιρέσεων.

Ανατρέξτε στην αρχή αυτού του κεφαλαίου, όπου δείξαμε ότι ο πολλαπλασιασμός ενός αριθμού επί τη βάση του αριθμητικού συστήματος τον μετατόπιζε κατά 1 ψηφίο προς τα αριστερά. Τι συμβαίνει όμως σε μια διαίρεση; Το μαντέψατε! Μετατοπίζει τον αριθμό κατά 1 ψηφίο προς τα δεξιά και το ψηφίο που βρίσκεται στη δεξιά άκρη χάνεται. Ωραία, πλησιάσατε κοντά εφόσον σκέφτεστε όπως ο υπολογιστής για τα bits ενός byte, το οποίο έχει το καθορισμένο μέγεθος των 8 bits. Και τι συμβαίνει κάθε φορά που ο αριθμός ξεπερνά το μήκος που μπορεί να συγκρατήσει ένα byte; Κάνει «1» το carry flag.

Όλα αυτά μας οδηγούν στο γεγονός ότι η Z80 CPU διαθέτει εντολές που μετατοπίζουν ένα byte προς τα αριστερά ή δεξιά. Ονομάζονται Shift Left (Αριστερή Μετατόπιση) και Shift Right (Δεξιά Μετατόπιση). Πρωτότυπο, έτσι; Η εντολή Shift Left κάνει την ίδια ακριβώς εργασία με την εντολή ADD A,A αλλά δεν περιορίζεται στο να χρησιμοποιεί μόνο τον καταχωρητή A. Η πλήρης εντολή είναι Shift Left Arithmetic, SLA για συντομία. Η εντολή σχηματίζεται από δύο bytes. Το πρώτο είναι ένα πρόθεμα, το CBh, και το δεύτερο είναι ο κωδικός λειτουργίας.

ASSEMBLER	HEX	BINARY
SLA r	CB 20-27	11 001 011 00 100 r



Σχήμα 11.1

Το r είναι, συνήθως, ένας από τους καταχωρητές γενικής χρήσης, ο A ή ο (HL). Και θα πρέπει να έχετε μάθει τώρα τους κωδικούς των 3 bits.

Πριν ασχοληθούμε στη συνέχεια με τις μετατοπίσεις προς τα δεξιά υπάρχουν ορισμένα σημεία που θέλουν προσοχή όταν χρησιμοποιείται η μετατόπιση προς τα αριστερά. Όπως ήδη γνωρίζετε, το carry flag γίνεται «1» (set) κάθε φορά που το αποτέλεσμα της μετατόπισης υποχρεώνει το σημαντικότερο bit να φύγει από τον καταχωρητή. Όλα πάνε καλά όταν το byte που μετατοπίζεται δεν είναι αριθμός, και χρησιμοποιείται απλώς για να δείχνει κάτι, όπως στο πρόγραμμα του Σχήματος 10.1. Σ' αυτή τη περίπτωση η απώλεια του MSB δεν έχει σημασία, αλλά αν γινόταν κάποιος πολλαπλασιασμός, το bit που θα περάσει στο carry flag έχει σημασία, και πρέπει να προσεχτεί.

Αυτό συνήθως αντιμετωπίζεται εύκολα, τοποθετώντας το κρατούμενο στο επόμενο σημαντικότερο byte. Το δεύτερο μέρος του προγράμματος της πρόσθεσης στο Σχήμα 6.8 δείχνει ένα τρόπο, με τη χρήση της εντολής ADC, για να τοποθετηθεί το κρατούμενο στο επόμενο byte. Αυτό είναι απλό για τους αριθμούς χωρίς πρόσημο. Μια μικρή υπορουτίνα που πολλαπλασιάζει την τιμή που υπάρχει στον καταχωρητή A επί 2 θα ήταν:

```
MULT    SLA    A
        LD    (RESULT),A
```

```

LD    A, (RESULT+1)
ADC   A,A
LD    (RESULT+1), A
RET

RESULT DEFW 0

```

Σχήμα 11.2

Το αποτέλεσμα θα τοποθετηθεί στη μνήμη στις διευθύνσεις RESULT και RESULT+1, με το σημαντικότερο byte στην RESULT+1, έτοιμο να παραληφθεί αργότερα σαν αριθμός των 16 bits. Αυτή η υπορουτίνα μπορεί να χρησιμοποιηθεί πολλές φορές ώστε να πολλαπλασιάζει με αριθμούς μεγαλύτερους του 2 αν χρειάζεται, καλώντας την ενώ ο καταχωρητής A περιέχει την (RESULT). Η κάθε διαδοχική κλήση θα υψώσει την τιμή στη δύναμη του 2. Για παράδειγμα:

```

LD    A, 1
CALL MULT ;      RESULT is now 2
LD    A, (RESULT)
CALL MULT ;      RESULT is now 4
LD    A, (RESULT)
CALL MULT ;      RESULT is now 8

```

Σχήμα 11.3

Αυτό συνεχίζεται μέχρι το αποτέλεσμα να ξεπεράσει το 65535 (που θα συμβεί μετά από 16 κλήσεις στο παραπάνω παράδειγμα). Το carry flag θα γίνει «1» (set) με την επιστροφή στο κυρίως πρόγραμμα. Αυτό το πρόγραμμα δεν είναι και πολύ καλό, αλλά παρ' όλ' αυτά μας χρησιμεύει στο να δείξουμε πώς μπορεί να χρησιμοποιηθεί η Shift Left Arithmetic στον πολλαπλασιασμό. Όταν χρησιμοποιείται αυτή η τεχνική σε αριθμό με αρνητικό πρόσημο, το σημαντικότερο byte της RESULT πρέπει να γίνει 1111111b πριν αρχίσει ο υπολογισμός, αλλιώς το τελικό αποτέλεσμα θα είναι θετικό.

Ακόμη δεν υπάρχει λόγος να προσπαθήσουμε να βελτιώσουμε το πρόγραμμα, γιατί πρόκειται να εξηγηθούν εντολές που κάνουν πολύ εύκολα τα πράγματα.

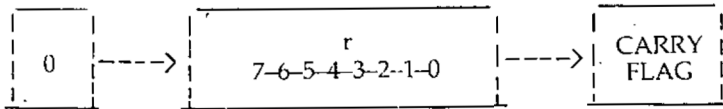
Η right shift έχει δύο μορφές, την arithmetic shift (αριθμητική μετατόπιση) και την logical shift (λογική μετατόπιση). Η logical shift είναι ακριβώς ίδια με την SLA αλλά μετατοπίζει προς τα δεξιά. Αυτό μπορεί να μην φαίνεται λογικό, αλλά όλα θα αποκαλυφθούν, γι' αυτό κάντε υπομονή!

Η δυαδική εντολή Shift Right Logical είναι ίδια με την Shift Left Arithmetic, αλλά με αλλαγμένα τα bits 5,4 και 3. Όλες οι εντολές σε τούτο το κεφάλαιο σχηματίζονται κατ' αυτόν τον τρόπο, με αυτά τα τρία bits να υπαγορεύουν τη φύση

της λειτουργίας που θα εκτελεστεί. Το πρόθεμα CBh είναι και πάλι παρόν σ' αυτήν την εντολή.

ASSEMBLER	HEX	BINARY
SRL r	CB 3B-40	11 001 011 00 111 r

Η συμβολική τους απεικόνιση φαίνεται στο Σχήμα 11.4.



Σχήμα 11.4

Με την πρώτη ματιά φαίνεται ότι προσφέρει την ευκαιρία αλλαγής της υπορουτίνας MULT με μια υπορουτίνα που θα διαιρεί με το 2. Η SLA θα χρειαστεί να αντικατασταθεί με την εντολή SRL, και θα πρέπει να αντιστραφεί η σειρά των εργασιών, ώστε να αρχίζουν με το σημαντικότερο byte. Το πρώτο πρόβλημα είναι ότι δεν υπάρχει τρόπος να παίρνουμε το κρατούμενο από τη βάση του σημαντικότερου byte, και να το χρησιμοποιούμε όταν ασχολούμαστε με το λιγότερο σημαντικό byte. Η εντολή SUB δεν μπορεί να χρησιμοποιηθεί, γιατί θα αφήνει πάντοτε τον καταχωρητή A με περιεχόμενο 0. Η διαίρεση επομένως πρέπει να περιοριστεί σε ακέραιους των 8 bits. Αυτό φαίνεται στο Σχήμα 11.5

```

DIVD   SRL   A

        LD   (RESULT+1), A

        RET

RESULT DEFW 00
    
```

Σχήμα 11.5

Υποθέτοντας ότι ο καταχωρητής A περιείχε αρχικά τον αριθμό 100 (64 h 01100100b), μετά την εκτέλεση θα περιέχει τον 50 (00110010b) που είναι σωστό. Αν διαιρεθεί ένας περιττός αριθμός, το υπόλοιπο θα υπάρχει στο carry flag. Θυμηθείτε ότι το carry flag δείχνει ότι το 1 βγήκε έξω από το byte. Έτσι, αν η παραπάνω υπορουτίνα είχε κληθεί με τον 101 (65h 01100101b) κατόπιν η RESULT+1 θα περιείχε τον 50 και το carry flag θα γινόταν «1» (set) δείχνοντας ότι υπόλοιπο = 1.

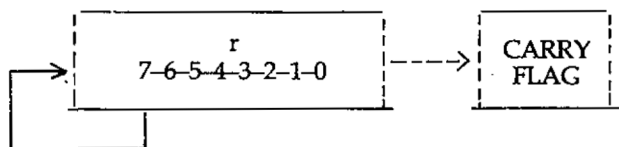
Τι συμβαίνει αν διαιρεθεί ένας αριθμός με αρνητικό πρόσημο; Ας δούμε το αποτέλεσμα αν η DIVD είχε κληθεί με τον καταχωρητή A να περιέχει το -26 (E6h 11100110b). Μετά την εκτέλεση η RESULT+1 θα περιέχει τον 01110011b ή 73h ή 115 στο δεκαδικό, που είναι τελείως λαθεμένο. Το 0 που τοποθετήθηκε για να γεμίσει το κενό μετά την μετατόπιση είναι ο αίτιος, γιατί το bit 7 είναι το bit του προσήμου (sign bit). Το γεγονός ότι αυτή η μετατόπιση προς τα δεξιά δεν μπορεί να χρησιμοποιηθεί για μια αριθμητική μετατόπιση είναι, προφανώς, είναι ο λόγος για τον οποίο η εντολή ονομάζεται Shift Right Logical (λογικό!).

Η Shift Right Arithmetic, όπως χωρίς αμφιβολία θα μαντέψατε, διατηρεί το bit

του προσήμου. Αν η παραπάνω υπορουτίνα ξαναγραφεί χρησιμοποιώντας την εντολή SRA στη θέση της SRL το αποτέλεσμα μετά την πράξη θα είναι 11110011b, -13 ή F3h, που είναι σωστό.

SRA r CB 28-30 11 001 011 00 101 r

Η συμβολική απεικόνιση φαίνεται στο Σχήμα 11.6



Σχήμα 11.6

Τώρα που γνωρίζετε να κάνετε μετατοπίσεις προς τα δεξιά και αριστερά, επόμεως να διαιρείτε ή να πολλαπλασιάζετε με το 2, το επόμενο βήμα είναι να βρείτε τρόπο να πολλαπλασιάζετε και να διαιρείτε αριθμούς με αριθμούς διάφορους του 2. Σε τούτο το στάδιο υποθέστε πως όχι μόνο τα δύο μέρη, αλλά και το αποτέλεσμα ενός υπολογισμού μπορούν να περιληφθούν σε ένα μόνο byte. Αυτό θα σας κάνει να ξεκινήσετε απλά, και θα σας δώσει την ευκαιρία να καταλάβετε τις αρχές του πολλαπλασιασμού και της διαιρέσης πριν φτάσετε στα δύσκολα. Για τους υπολογισμούς χωρίς πρόσημο αυτό σημαίνει ότι στον πολλαπλασιασμό το γινόμενο πρέπει να είναι μικρότερο του 256, και στη διαίρεση ότι και ο διαιρετέος και ο διαιρέτης πρέπει να είναι μικρότερος του 256.

Η πράξη του πολλαπλασιασμού είναι στην πραγματικότητα, απλώς μια διαδικασία πρόσθεσης του πολλαπλασιαστέου σε ένα αποτέλεσμα, που είναι 0 αρχικά, τόσες φορές όσες ορίζει ο πολλαπλασιαστής. Αυτό μπορεί ναδειχθεί αν φορτώσετε πάλι στον υπολογιστή σας το πρόγραμμα του Σχήματος 6.14 και κατόπιν προσθέσετε το σύντομο πρόγραμμα που δίνεται στο Σχήμα 11.7. Αυτό πολλαπλασιάζει μεταξύ τους τους κωδικούς δύο πλήκτρων που πιέζονται στο πληκτρολόγιο από σας.

Hisoft GENA3 Assembler. Page 1.

Pass 1 errors: 00

```

10 ; FIG 11,7
20 ; A PROGRAM TO PERFORM AN
   BX8 BIT MULTIPLICATION
30 ; WITH AN 8 BIT RESULT,
   SIMPLE ADDITION METHOD

A604          40          ORG      42500
A604          50          ENT      42500
BB1B         60  GETKEY   EQU      47896
A604  CD18BB   70          CALL    GETKEY
A607   4F      80          LD      C,A
A60B  CD18BB   90          CALL    GETKEY
A60B   47     100         LD      B,A

```

A60C	AF	110		XOR	A ;THIS
					WILL SET A TO 0
A60D	81	120	ADLOOP	ADD	A,C
A60E	10FD	130		DJNZ	ADLOOP
A610	3264A6	140		LD	(42596),A
A613	C3B4AA	150		JP	43700

Pass 2 errors: 00
Table used: 72 from 221
Executes: 43000

THE CHECK-SUMS REQUIRED BY THE HEX LOADER ARE
0506 0483

Σχήμα 11.7

Η τεράστια πλειοψηφία των πλήκτρων δεν θα μπορεί να συγκρατηθεί από ένα μόνο byte, αλλά πολλοί από τους κωδικούς ελέγχου (control codes) είναι χρήσιμοι. Η προσπέλαση σ' αυτούς γίνεται πατώντας το πράσινο πλήκτρο [CONTROL] και, ενώ εξακολουθείτε να το πατάτε, ένα ακόμη πλήκτρο.

Το [CONTROL] G θα δώσει τον κωδικό BEL (καμπανάκι) που είναι 7, και το [CONTROL] J θα σας δώσει τον κωδικό αλλαγής γραμμής 10 (OAh), οπότε όταν εκτελέσετε το πρόγραμμα με την CALL 42500 ή με την εντολή R αν χρησιμοποιείτε τον assembler, αυτό θα περιμένει να πατήσετε τα πλήκτρα, και κατόπιν θα τυπώσει το αποτέλεσμα του γινομένου τους. Για παράδειγμα: πατώντας το [CONTROL] G ακολουθούμενο από το [CONTROL] J θα έχετε σαν αποτέλεσμα 70. Το παράρτημα III του βιβλίου Amstrad User Instructions δίνει μια πλήρη λίστα των κωδικών των διαφόρων πλήκτρων.

Αν και η μέθοδος που χρησιμοποιείται στο Σχήμα 11.7 δουλεύει θαυμάσια για τους τύπους πολλαπλασιασμού που μπορεί να χειριστεί, υπάρχει και ένας άλλος τρόπος με τον οποίο μπορεί να γίνει το ίδιο πράγμα, αλλά πολύ αποτελεσματικότερα. Δεν χρειάζεται να πούμε ότι δεν χρησιμοποιεί μόνο προσθέσεις, αλλά και μετατοπίσεις.

Hisoft GENA3 Assembler. Page 1.
Pass 1 errors: 00

		10 ;	FIG 11.8		
		20 ;	A PROGRAM TO PERFORM AN		
			BXB BIT MULTIPLICATION		
		30 ;	WITH AN 8 BIT RESULT,		
			SHIFT AND ADD METHOD		
A604		40	ORG	42500	
A604		50	ENT	42500	
BB18		60	GETKEY	EQU	47896
A604	CD18BB	70	CALL	GETKEY	
A607	4F	80	LD	C,A	
A608	CD18BB	90	CALL	GETKEY	

A60B	47	100		LD	B, A
A60C	AF	110		XOR	A; THIS
				WILL	SET A TO 0
A60D	CB3B	120	ADLOOP	SRL	B
A60F	3001	130		JR	NC, NOADD
A611	B1	140		ADD	A, C
A612	CB21	150	NOADD	SLA	C
A614	20F7	160		JR	NZ, ADLOOP
A616	3264A6	170		LD	(42596), A
A619	C304A6	180		JP	42500

Pass 2 errors: 00
Table used: B4 From 230
Executes: 43000

THE CHECK-SUMS REQUIRED BY THE HEX LOADER ARE
0550 0397 02CC

Σχήμα 11.8

Το πρόγραμμα στο Σχήμα 11.7 θα έπρεπε να εκτελέσει το βρόχο της πρόσθεσης μέχρι 127 φορές πριν υπολογιστεί το αποτέλεσμα, κι αυτό για αποτελέσματα μικρότερα του 256. Σκεφτείτε το χρόνο που θα χρειαστεί αυτό το σύστημα για να φτάσει στο αποτέλεσμα ενός υπολογισμού με πολλά bytes. Ακόμη και με όριο τα 16 bits (2 bytes) στο αποτέλεσμα, έχουμε να κάνουμε με 32767 βρόχους. Οι «έξυπνοι» τώρα θα πουν: «όχι, δεν χρειάζεται! Θα το έδιναν σαν $2 \cdot 32767$ και όχι σαν $32767 \cdot 2$, και ο μέγιστος αριθμός των δυνατών βρόχων αν δίνω πάντοτε πρώτο τον μεγαλύτερο αριθμό θα είναι 256, για αποτελέσματα των 16 bits».

Εντάξει, αρκετά καλό, αλλά τι συμβαίνει όταν το πρόγραμμα το χρησιμοποιεί κάποιος άλλος; Πάντως, υπάρχει ένας πολύ καλύτερος τρόπος για να πολλαπλασιάσουμε, είναι ο τρόπος που διδαχθήκαμε και στο σχολείο. Παρακάτω φαίνεται ένας δυαδικός πολλαπλασιασμός αυτού του είδους δίπλα σε ένα δεκαδικό. Δείτε τι συμβαίνει κατά τη διάρκεια ενός πολλαπλασιασμού στο δυαδικό, καθώς και στο δεκαδικό όπου ο πολλαπλασιαστής αποτελείται από μονάδες και μηδενικά.

	BINARY	DECIMAL
To calculate	00010011	19d multiplicand
	00001011 *	11d multiplier
	10011	19
	100110	19
	0	
	10011000	
	11010001	209

Σε κάθε στάδιο ο πολλαπλασιαστέος (multiplicand) μετατοπίζεται κατά μια θέση προς τα αριστερά και αν ο πολλαπλασιαστής (multiplier) δεν είναι 0 τότε ο μετατοπισμένος πολλαπλασιαστέος προστίθεται στο αποτέλεσμα. Όταν χρησιμοποιείται το δεκαδικό σύστημα αυτό θα συμβαίνει σε λίγες μόνο περιπτώσεις, αλλά στο δυαδικό θα συμβαίνει πάντοτε, γιατί δεν υπάρχει τίποτε άλλο εκτός από 0 και 1. Χρη-

σιμοποιώντας αυτή τη μέθοδο οι προσθέσεις περιορίζονται στο ελάχιστο, γιατί δεν μπορεί να υπάρχουν περισσότερες αθροίσεις από όσες είναι οι μη μηδενικές στήλες του πολλαπλασιαστή. Για έναν οκταψήφιο αριθμό δεν μπορεί ποτέ να υπάρχουν περισσότερες από 8 προσθέσεις και για ένα δεκαεξαψήφιο αριθμό, υπάρχουν μόνο 16 προσθέσεις το πολύ.

Το πρόγραμμα στο Σχήμα 11.8 δείχνει μια μέθοδο πολλαπλασιασμού που εφαρμόζει το παραπάνω σύστημα. Μόλις έρθουν οι αριθμοί που θα πολλαπλασιαστούν και μηδενιστεί το αποτέλεσμα (ο καταχωρητής A για τους σκοπούς αυτού του προγράμματος), ελέγχεται το ελάχιστο σημαντικό bit του πολλαπλασιαστή. Το shift right (SRL) στη label ADLOOP τοποθετεί το bit στο carry flag ώστε να μπορεί να ελεγχθεί. Κατόπιν, αν το bit ήταν «1» (set), ο πολλαπλασιαστής προστίθεται στο αποτέλεσμα. Στη label NOADD, ο πολλαπλασιαστής μετατοπίζεται κατά μια θέση προς τα αριστερά, ακριβώς όπως συνέβη στον πολλαπλασιασμό που παρουσιάσαμε παραπάνω. Κατόπιν γίνεται ένας έλεγχος για να εξακριβωθεί αν υπάρχει τίποτε άλλο να γίνει, και να συμβαίνει αυτό η διαδικασία επαναλαμβάνεται με τον νέο πολλαπλασιαστή, αλλιώς ο υπολογισμός έχει τελειώσει, και το αποτέλεσμα μεταφέρεται για εκτύπωση από την υπορουτίνα.

Η διαίρεση είναι περίπου ανάλογη με τον πολλαπλασιασμό αλλά έχει την πρόσθετη περιπλοκή ότι ο υπολογισμός θα μπορούσε να συνεχίζεται χωρίς να τελειώνει ποτέ. Ένα παράδειγμα που όλοι γνωρίζουμε είναι ο υπολογισμός του π . Έχει ανατεθεί στους ισχυρότερους υπολογιστές το καθήκον του υπολογισμού του π αλλά μέχρι τώρα δεν υπήρξε ένδειξη μιας αληθινής απάντησης. Κατά πάσα πιθανότητα δεν πρόκειται να υπάρξει ποτέ, αλλά στο κάτω - κάτω ποιός χρειάζεται να γνωρίζει το π με ακρίβεια αρκετών εκατομμυρίων δεκαδικών ψηφίων;

Ακόμη και στα κανονικά μαθηματικά παρουσιάζονται συχνά, με μια απλή διαίρεση, αριθμοί με άπειρο πλήθος ψηφίων, και ο συνηθισμένος τρόπος χειρισμού τους από τους υπολογιστές είναι να δίνουν ένα ηλίκο και ένα υπόλοιπο (Το ηλίκο δείχνει πόσες φορές μπορεί να αφαιρεθεί ο διαιρέτης από τον διαιρετέο χωρίς να γίνεται αρνητικός ο διαιρετέος).

Το πρόγραμμα της διαίρεσης που είναι ανάλογο με το πρόγραμμα του πολλαπλασιασμού στο Σχήμα 11.7 σας είναι ήδη πολύ γνωστό. Τα προγράμματα που έχετε χρησιμοποιήσει για την εκτύπωση των αποτελεσμάτων σε όλα τα μαθηματικά που κάνατε μέχρι τώρα, εργάζονται με διαδοχικές διαιρέσεις, με τη μέθοδο της αφαίρεσης. Κάθε φορά που ο διαιρετέος γίνεται αρνητικός, ο διαιρέτης προστίθεται στο διαιρετέο για να γίνει ο διαιρετέος της επόμενης διαίρεσης. Ο κατάλληλος όρος για την ενέργεια που υποχρεώνει τον διαιρετέο να γίνει αρνητικός είναι «overdraw» («παρατραβάω»).

Σας έχουμε δείξει πως τα bits που «βγαίνουν» έξω από ένα καταχωρητή, ή θέση μνήμης, σαν αποτέλεσμα μιας εντολής που εκτελείται, «πέφτουν» στο carry flag, και ότι, στον πολλαπλασιασμό, μια μετατόπιση προς τα αριστερά μπορεί να επιφέρει τεράστια εξοικονόμηση χρόνου σε έναν υπολογισμό. Αυτή η μετατόπιση προς τα αριστερά μπορεί να εφαρμοσθεί σε όσα bits χρειάζεται παίρνοντας το bit από το carry flag κατά τα διαδοχικά στάδια μιας μετατόπισης.

Για να είναι δυνατή η χρησιμοποίηση στη διαίρεση αυτής της αποτελεσματικότερης μεθόδου μετατόπισης (καθώς και πολλά άλλα πράγματα) είναι αναγκαίες νέες λειτουργίες. Κι αυτό γιατί, από ολόκληρο το ρεπερτόριο εντολών που έχουν

εξηγηθεί μέχρι τώρα, δεν υπάρχει ούτε μια που να μπορεί να μεταφέρει το σημαντικότερο (most significant) bit, ουσιαστικά αν η διαίρεση πρόκειται να εκτελεστεί με μετατοπίσεις αντί για τη μέθοδο των επαναλαμβανόμενων αφαιρέσεων. Στην πραγματικότητα, ο μοναδικός τρόπος με τον οποίο το κρατούμενο μπορεί να μεταφερθεί με μια εντολή φαίνεται παρακάτω.

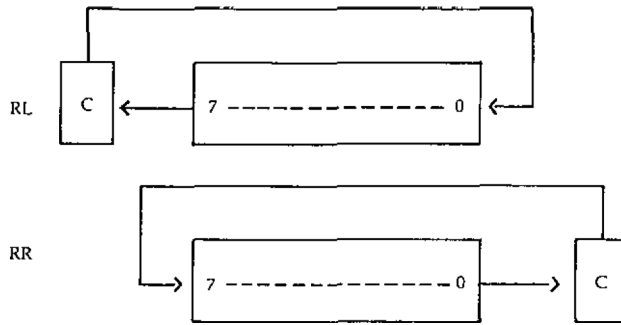
	Most Sig. Byte	CARRY	Least S. Byte
	7-6-5-4-3-2-1-0		7-6-5-4-3-2-1-0
	0 0 0 0 0 0 0 0	0	1 0 1 1 0 1 0 0
SLA LSB	0 0 0 0 0 0 0 0	1	0 1 1 0 1 0 0 0
ADC MSB, MSB	0 0 0 0 0 0 0 1	0	0 1 1 0 1 0 0 0

Σχήμα 11.9

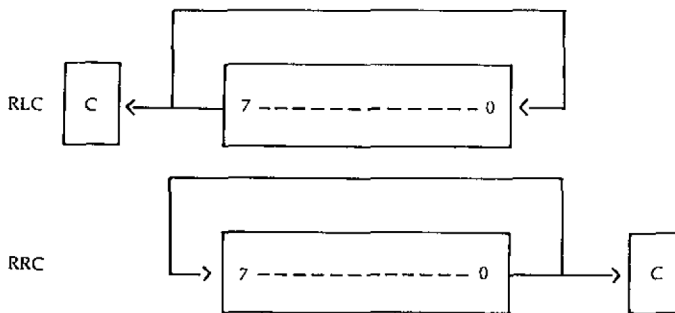
Πρόκειται πραγματικά για ένα «ζαβολιάρικο» bit, γιατί η εντολή ADC χρησιμοποιείται για την προσομοίωση μιας μετατόπισης προς τα αριστερά με κρατούμενο. Είναι ο οικονομικότερος (όσον αφορά τη μνήμη και τους χρησιμοποιούμενους καταχωρητές) τρόπος για να γίνει αυτό με έναν αριθμό των 8 bits. Η μετατόπιση προς τα αριστερά των 16 bits μπορεί να πραγματοποιηθεί με μια μόνο εντολή, χρησιμοποιώντας την ίδια με την παραπάνω τεχνική, αλλά με τις εντολές ADD HL, HL ή ADC HL, HL.

Ο Z80 προσφέρει μια σημαντική συλλογή πράξεων, ειδικά σχεδιασμένων για να είναι οι λειτουργίες αυτού του είδους ανεξάρτητες από τους συσσωρευτές (το ζεύγος καταχωρητών HL είναι στην πραγματικότητα ένας συσσωρευτής των 16 bits, όταν χρησιμοποιείται για μαθηματικές πράξεις). Όλες αυτές οι εντολές χρησιμοποιούν το carry flag, και για την παραλαβή του bit που φεύγει από το byte και για την προμήθεια του bit που θα τοποθετηθεί στη θέση που έμεινε κενή από την πράξη. Ορισμένες παίρνουν το bit από το carry flag πριν βάλουν το bit που φεύγει στο carry flag, κατορθώνοντας αυτού του είδους τη μετατόπιση με την παραπάνω εντολή ADC. Άλλες τοποθετούν το bit που φεύγει εξαιτίας της συγκεκριμένης πράξης στο carry flag πριν βάλουν το carry flag στο bit που έμεινε κενό. Και οι δύο τρόποι στην πραγματικότητα κάνουν ένα είδος Περιστροφής (Rotation), είτε μέσω του carry flag είτε περιλαμβάνοντας το carry flag.

Αυτό μπορεί να συμβολιστεί όπως φαίνεται στο Σχήμα 11.10. Πρόκειται όπως βλέπετε για πλήρεις περιστροφές, και πραγματικά οι εντολές που ενεργούν κατ' αυτόν τον τρόπο ονομάζονται «Rotate», συντομευόμενες σε RR για τη δεξιά περιστροφή (Rotate Right) και RL για την αριστερή περιστροφή (Rotate Left). Οι δύο λειτουργίες που παρουσιάζονται στο Σχήμα 11.11, καθώς τοποθετούν στο carry flag ένα αντίγραφο του bit που μεταφέρεται στην άλλη άκρη του byte, στην πραγματικότητα δεν παίρνουν καμιά καινούρια πληροφορία από το carry flag. Αυτές ονομάζονται κυκλικές περιστροφές (Rotate Circular), και όπως και οι παραπάνω συντομεύονται σε RRC και RLC, όπου το δεύτερο R είναι για δεξιά και το δεύτερο L για αριστερή κατεύθυνση.



Σχήμα 11.10



Σχήμα 11.11

Ο συσσωρευτής (ο καταχωρητής A) έχει πάλι τις δικές του ειδικές εντολές, καθώς και τους standard κωδικούς λειτουργίας που κάνουν ακριβώς την ίδια δουλειά. Οι πλήρεις εντολές για κάθε μια από τις λειτουργίες που δόθηκαν παραπάνω είναι:

ASSEMBLER	HEX	BINARY
RL r	CB 10 - 17	11 001 011 00 010 r
RLA	17	00 010 111
RR r	CB 18 - 1F	11 001 011 00 011 r
RRA	1F	00 011 111
RLC r	CB 00 - 07	11 001 011 00 000 r
RLCA	07	00 000 111
RRC r	CB 08 - 0F	11 001 011 00 001 r
RRCA	0F	00 001 111

Με τις ευκολίες που προσφέρει αυτή η νέα ομάδα εντολών ανοίγεται διάπλατα η πόρτα για τη γρήγορη διαίρεση. Η διαίρεση που εκτελείται στη ρουτίνα εκτύπωσης αριθμών δεν μπορεί ποτέ να έχει πηλίκο μεγαλύτερο του 9, οπότε δεν έχουμε μεγάλη απώλεια χρόνου χρησιμοποιώντας τη μέθοδο της αφαίρεσης, ενώ όλοι οι διαιρέτες ήσαν γνωστοί από πριν. Όταν γράφουμε ένα πρόγραμμα όπου ο διαιρέτης δεν είναι γνωστός εκ των προτέρων είναι ουσιαστικά να εξασφαλίζουμε ότι έχει αποκλειστεί οποιαδήποτε προσπάθεια διαίρεσης με το 0. Αν δεν ληφθεί αυτή η προφύλαξη η απόπειρα διαίρεσης με το 0 θα μπλοκάρει τον υπολογιστή, γιατί το αποτέλεσμα είναι άπειρο, οπότε η διαίρεση θα συνεχιστεί για πάντα.

Ο έλεγχος για το μηδέν μπορεί να γίνει με διάφορους τρόπους. Για ένα διαιρέτη των 8 bits, ο διαιρέτης μπορεί να τοποθετηθεί στον καταχωρητή A και να ελεγχθεί από ένα AND A, ενώ για ένα διαιρέτη των 16 bits, 1 byte του διαιρέτη τοποθετείται στον καταχωρητή A και πάνω του, ενεργεί η εντολή OR ως προς το άλλο byte. Και οι δύο μέθοδοι κάνουν «1» (set) το zero flag αν ο διαιρέτης είναι 0.

Hisoft GENA3 ASSEMBLER. PAGE 1.
Pass 1 errors: 00

```

10 : FIG 11.12 A SHIFT AND
      ROTATE DIVIDE BY 2
A410          20          ORG      42000
A410          30          ENT      42000
BB1B          40  GETKEY  EQU      47896
BB5A          50  PRINT   EQU      47962
A410  0604      60          LD      B,4
A412  2164A6    70          LD      HL,42596
A415  CD18BB    80  INLOOP  CALL   GETKEY
A41B  FE80      90          CP      #80
A41A  2001     100         JR      NZ,NOT_0
A41C  AF       110         XOR      A
A41D  77       120  NOT_0   LD      (HL),A
A41E  23       130         INC     HL
A41F  10F4     140         DJNZ   INLOOP
A421  CDA0A5   150         CALL   42400
A424  213CAB   160         LD      HL,D_MSG
A427  7E       170  MSG_LP  LD      A,(HL)
A428  CD5ABB   180         CALL   PRINT
A42B  23       190         INC     HL
A42C  FE00     200         CP      #00
A42E  20F7     210         JR      NZ,MSG_LP
A430  2167A6   220         LD      HL,42596
A433  AF       230         XOR      A
A434  CB3E     240         SRL     (HL)
A436  0603     250         LD      B,3
A438  2B       260  DIV_LP  DEC     HL
A439  CB1E     270         RR      (HL)

```

A43B	10FB	280		DJNZ	DIU_LP
A43D	F5	290		PUSH	AF
A43E	CDA0A5	300		CALL	42400
A441	3E20	310		LD	A, 32
A443	CD5ABB	320		CALL	PRINT
A446	3E52	330		LD	A, 'R'
A448	CD5ABB	340		CALL	PRINT
A44B	F1	350		POP	AF
A44C	CE00	360		ADC	A, 0
A44E	F630	370		OR	#30
A450	CD5ABB	380		CALL	PRINT
A453	C9	390		RET	
A454	20446976	400	D_MSG	DEFM	' Div'
A458	69646564	410		DEFM	'ided'
A45C	20627920	420		DEFM	' by'
A460	74776F3D	430		DEFM	'two-'
A464	2000	440		DEFW	#0020

Pass 2 errors: 00
Table used: 134 from 306
Executes: 43000

THE CHECK-SUMS REQUIRED BY THE HEX LOADER ARE
046C 0499 0486 041F 057D 0565 0503 0390 01E7

Σχήμα 11.12

Εφοδιασμένη μ' αυτές τις εντολές μια διαίρεση με το δύο μπορεί να εκτελεστεί σε οποιονδήποτε αριθμό από bytes χρησιμοποιώντας μια SRL ή μια SRA αν ο διαιρετέος είναι προσημασμένος, στο σημαντικότερο byte, ακολουθούμενες από μια RR για το καθένα από τα λιγότερο σημαντικά bytes στη σειρά. Αυτό φαίνεται στο Σχήμα 11.12. Για να μπορείτε να εισάγετε αριθμούς για διαίρεση σας προσφέρεται μια ρουτίνα εισαγωγής, που θα παίρνει τον κωδικό ASCII του πλήκτρου που πιέζετε και θα το χρησιμοποιεί σαν ένα byte του διαιρετέου των 32 bits. Ο κωδικός του πρώτου πλήκτρου που πιέζεται θα γίνει το ελάχιστο σημαντικό byte, και τα πλήκτρα που ακολουθούν θα είναι τα επόμενα σημαντικότερα bytes. Επειδή ο Amstrad δεν έχει τρόπο δημιουργίας του κωδικού ASCII NUL από το πληκτρολόγιο, το πλήκτρο 0 του αριθμητικού πληκτρολόγιου αποκλείεται, και χρησιμοποιείται ο κωδικός του NUL κάθε φορά που το πιέζουμε. Ναι, το Παράρτημα III του Amstrad User Instructions λέει ότι το 0 δημιουργείται από το [CONTROL] A αλλά έχουν επίσης δύο [CONTROL] C. Όταν χρησιμοποιείται με το [CONTROL] το πλήκτρο A αντιστοιχεί στον κωδικό 1, το B στον 2 και το C στον 3, και από εκεί και πέρα το βιβλίο User Instructions είναι σωστό.

Παρατηρείστε πως ο καταχωρητής των flags δεν χρειάζεται να αποθηκευτεί πριν χρησιμοποιήσει την εντολή DJNZ, γιατί αυτή η εντολή δεν επιδρά σε κανένα flag (αυτό είναι σημαντικό γιατί το carry flag περιέχει ό,τι απομένει στο τέλος του

DIV-LP), ενώ τα flags και ο καταχωρητής A, που περιέχει το 0 έτοιμο για την ADC, πρέπει να αποθηκευτούν μετά τη διαίρεση. Πειραματιστείτε με αυτό το πρόγραμμα μέχρι να βεβαιωθείτε ότι καταλαβαίνετε πως πετυχαίνεται μια διαίρεση με Μετατοπίσεις και Περιστροφές. Παρατηρήστε πως τα προγράμματα στα Σχήματα 11.12 και 11.13 χρειάζονται να έχουν την υπορουτίνα εκτύπωσης αριθμών των 32 bits που δίνεται στο Σχήμα 6.13 για να τυπώσουν τον αριθμό.

Hisoft GENA3 Assembler. Page 1.

Pass 1 errors: 00

```

10 ; FIG 11,13 A SHIFT AND
    ROTATE DIVIDE
A410          20          ORG    42000
A410          30          ENT    42000
BB1B          40    GETKEY EQU    47896
BB5A          50    PRINT  EQU    47962
A410 210000    60          LD     HL,0
A413 226AA6    70          LD     (42596),HL
A416 2266A6    80          LD     (42598),HL
A419 CD58A4    90          CALL  GETVAL
A41C 5F        100         LD     E,A
A41D 216EA4    110        LD     L,D_MSG
A420 CD64A4    120        CALL  MSG_LP
A423 CD58A4    130        CALL  GETVAL
A426 4F        140        LD     C,A
A427 217CA4    150        LD     HL,MSG2
A42A CD64A4    160        CALL  MSG_LP
A42D AF        170        XOR    A
A42E 0608      180        LD     B,0
A430 CB13      190    DIV_LP  RL     E
A432 17        200        RLA
A433 91        210        SUB    C
A434 3001      220        JR     NC,NO_ADD
A436 B1        230        ADD    A,C
A437 10F7      240    NO_ADD  DJNZ  DIV_LP
A439 47        250        LD     B,A
A43A 7B        260        LD     A,E
A43B 17        270        RLA
A43C 2F        280        CPL
A43D CD4BA4    290        CALL  P_NUMB
A440 2180A4    300        LD     HL,MSG3
A443 CD64A4    310        CALL  MSG_LP
A446 7B        320        LD     A,B
A447 CD4BA4    330        CALL  P_NUMB
A44A C9        340        RET
A44B E5        350    P_NUMB.  PUSH  HL

```

A44C	DS	360		PUSH	DE
A44D	CS	370		PUSH	BC
A43E	3264A6	380		LD	(42596),A
A451	CDAA0A5	390		CALL	42400
A454	C1	400		POP	BC
A455	D1	410		POP	DE
A456	E1	420		POP	HL
A457	C9	430		RET	
A458	CD18BB	440	GETVAL	CALL	GETKEY
A45B	F5	450		PUSH	AF
A45C	CD4BA4	460		CALL	P_NUMB
A45F	F1	470		POP	AF
A460	A7	480		AND	A
A461	C0	490		RET	NZ
A462	E1	500		POP	HL
A463	C9	510		RET	
A464	7E	520	MSG_LP	LD	A,(HL)
A465	CD5ABB	530		CALL	PRINT
A468	23	540		INC	HL
A469	FE00	550		CP	#00
A46B	20F7	560		JR	NZ,MSG_LP
A46D	C9	570		RET	
A46E	20446976	580	D_MSG	DEFM	' Div'
A472	69646564	590		DEFM	'ided'
A476	20627920	600		DEFM	' by'
A47A	2000	610		DEFW	#0020
A47C	3D0D	620	MSG2	DEFW	#0D3D
A47E	0A00	630		DEFW	#000A
A480	2052	640	MSG3	DEFW	#5220
A482	2000	650		DEFW	#0020

THE CHECK-SUMS REQUIRED BY THE HEX LOADER ARE

037A 04F4 04D4 0256 0430 0637 06AC
06D8 0692 03F0 024E 009C

Σχήμα 11.13

Στο Σχήμα 11.13 δίνεται το ανάλογο της διαίρεσης του προγράμματος στο Σχήμα 11.9. Οι ομοιότητες είναι άμεσα φανερές αλλά αυτή τη φορά ο βρόχος πρέπει να εκτελείται για το κάθε bit του υπολογισμού. Για να αρχίσουμε, ο διαιρετέος βρίσκεται στον καταχωρητή E και ο διαιρέτης είναι στον καταχωρητή C, ο B χρησιμοποιείται σαν μετρητής, με την εντολή DJNZ να μετράει τα bits της διαίρεσης. Με κάθε πέρασμα από το βρόχο το carry flag περιστρέφεται μέσα στον E (τον διαιρετέο), και το σημαντικότερο bit του E περιστρέφεται στο carry-flag. Στην αρχή

το carry flag είχε γίνει «0» (reset), από την εντολή XOR A που χρησιμοποιήθηκε για να καθαρίσει τον καταχωρητή A, οπότε το 0 περιστράφηκε στο ελάχιστο σημαντικό bit (LSB) του E. Το κρατούμενο από τον E τοποθετείται κατόπιν στο bit 0 (το ελάχιστο σημαντικό bit) του καταχωρητή A από την εντολή RLA.

Κατόπιν γίνεται μια προσπάθεια να αφαιρεθεί ο διαιρέτης από τον καταχωρητή A. Αν αυτό έχει σαν αποτέλεσμα τη μεταφορά ενός κρατούμενου τότε δεν μπορούσε να γίνει η αφαίρεση, και ο καταχωρητής A αποκαθίσταται προσθέτοντας τον διαιρέτη. Όπως και με τη ρουτίνα πολλαπλασιασμού πρόκειται για την ίδια ακριβώς διαδικασία που εκτελείται κατά τη διαίρεση, όπως φαίνεται στο Σχήμα 11.14 για τη διαίρεση του 85 με το 2. Οποιοδήποτε κρατούμενο δημιουργείται ή παραλαμβάνεται από κάποια πράξη προσδιορίζεται με ένα θέλος που δείχνει την κατεύθυνση.

			E	A
			01010101	00000000
(1)	DIV_LP	RL E	0<-10101010<-0	00000000
		RLA		0<-00000000<-0
		SUB C		1<-11111110
		JR NC,NO_ADD		
		ADD A,C		1<-00000000
	NO_ADD	DJNZ DIV_LP		
(2)	DIV_LP	RL E	1<-01010101<-1	
		RLA		0<-00000001<-1
		SUB C		1<-11111101
		JR NC,NO_ADD		
		ADD A,C		1<-00000001
	NO_ADD	DJNZ DIV_LP		
(3)	DIV_LP	RL E	0<-10101011<-1	
		RLA		0<-00000010<-0
		SUB C		0<-00000000
		JR NC,NO_ADD		
	NO_ADD	DJNZ DIV_LP		
(4)	DIV_LP	RL E	1<-01010110<-0	
		RLA		0<-00000001<-1
		SUB C		1<-11111101
		JR NC,NO_ADD		
		ADD A,C		1<-00000001
	NO_ADD	DJNZ DIV_LP		
(5)	DIV_LP	RL E	0<-10101101<-1	
		RLA		0<-00000010<-0
		SUB C		0<-00000000
		JR NC,NO_ADD		
	NO_ADD	DJNZ DIV_LP		
(6)	DIV_LP	RL E	1<-01011010<-0	
		RLA		0<-00000001<-1
		SUB C		1<-11111101
		JR NC,NO_ADD		
		ADD A,C		1<-00000001

```

NO_ADD DJNZ DIV_LP
      ( 7 )
DIV_LP RL  E          0<-10110101<-1
      RLA                      0<-00000010<-0
      SUB  C          0<-00000000
      JR   NC,NO_ADD
NO_ADD DJNZ DIV_LP
      ( 8 )
DIV_LP RL  E          1<-01101010<-0
      RLA                      0<-00000001<-1
      SUB  C          1<-11111101
      JR   NC,NO_ADD
      ADD  A,C          1<-00000001
NO_ADD DJNZ DIV_LP
      LD  B,A          B now 00000001
      LD  A,E          01101010
      RLA                      0<-11010101<-1
      CPL                      00101010

```

Which is 42d with a remainder in B of 1.

Σχήμα 11.14

Αξίζει να μελετήσετε τη διαδικασία ξανά και ξανά μέχρι να βεβαιωθείτε απόλυτα ότι καταλαβαίνετε πλήρως τα μέσα με τα οποία εκτελείται η διαίρεση. Αυτή η τεχνική διαίρεσης είναι γνωστή σαν μέθοδος της αποκατάστασης (restoring method), γιατί η αφαίρεση αποκαθίσταται σε περίπτωση που υπάρχει κρατούμενο. Υπάρχουν και άλλες μέθοδοι διαίρεσης αλλά βρίσκονται πέρα από τους στόχους αυτού του βιβλίου. Η μέθοδος της αποκατάστασης είναι αποτελεσματική και εύκολα μπορεί να εφαρμοστεί σε πολλαπλά bytes, οπότε δίνει στον προγραμματιστή τη δυνατότητα να φέρει σε πέρας οποιαδήποτε διαίρεση χρειάζεται.

Υπάρχει μια ενδιαφέρουσα διαφορετική χρήση των εντολών μετατόπισης και περιστροφής. Γράψτε το σύντομο πρόγραμμα του Σχήματος 11.15, και μετά την εισαγωγή και αποθήκευσή του, βάλτε τον Amstrad στο mode 2 και γράψτε αρκετές ανοησίες στην οθόνη. Αν είσαστε στη BASIC μερικά syntax errors κάνουν τη δουλειά μας, ή το listing του Hex loader. Κατόπιν εκτελέστε το. Αν χρησιμοποιείτε τον assembler η εντολή [W] θα αλλάξει το mode.

Θα διαπιστώσετε ότι ολόκληρη η οθόνη ρολάρει προς τα δεξιά, ένα pixel κάθε φορά, ανά 1 χαρακτήρα. Προσπαθήστε να αλλάξετε την RR (HL) με άλλους κώδικες λειτουργίας που εξηγούνται σ' αυτό το κεφάλαιο, και δείτε αν μπορείτε να προβλέψετε σωστά το αποτέλεσμα. Παρατηρήστε ότι ο καταχωρητής των flags φυλάγεται προσεκτικά από το πρόγραμμα. Τι θα συμβεί αν όλες οι εντολές PUSH και POP αφαιρεθούν; Δοκιμάστε το και θα δείτε. Δοκιμάστε επίσης και σε άλλα modes, μπορείτε να καταλάβετε γιατί παίρνете αυτά τα περίεργα αποτελέσματα στα άλλα modes; Το Σχήμα 11.16 δίνει το ίδιο πρόγραμμα για ρολάρισμα προς τα αριστερά κατά 1 pixel. Μπορείτε να καταλάβετε γιατί έπρεπε να γίνουν οι αλλαγές;

Hisoft GENA3 Assembler. Page 1.

Pass 1 errors: 00

10 : FIG 11,14 SCREEN RIGHT
SCROLL

20

A410		30	ORG	42000
A410		40	ENT	42000
A410	0608	50	LD	B, B
A412	F5	60	PUSH	AF
A413	2100C0	70	SCREEN	LD HL, #C000
A416	F1	80	PIXEL	POP AF
A417	CB1E	90	RR	(HL)
A419	F5	100	PUSH	AF
A41A	23	110	INC	HL
A41B	7D	120	LD	A, L
A41C	B4	130	OR	H
A41D	20F7	140	JR	NZ, PIXEL
A41F	10F2	150	DJNZ	SCREEN
A421	F1	160	POP	AF
A422	C9	170	RET	

Pass 2 errors: 00
Table used: 38 from 143
Executes: 43000

THE CHECK-SUMS REQUIRED BY THE HEX LOADER ARE
04B3 0527

Σχήμα 11.15

Μια ποσότητα προσεκτικών υπολογισμών με τη βοήθεια του χάρτη της οθόνης (screen map) στο παράρτημα θα σας επιτρέψει να μετακινείτε κομμάτια της οθόνης κατά βούληση. Αυτές οι ρουτίνες είναι αργές αλλά όταν σκεφθείτε ότι πρέπει να γίνονται 16384 περιστροφές για κάθε pixel προς τα αριστερά ή δεξιά, και να εκτελούνται σχεδόν 132000 εντολές για μια μετατόπιση της οθόνης κατά 1 μόνο pixel, ίσως αρχίσετε να εκτιμάτε την ταχύτητά τους.

Υπάρχουν δύο ακόμη εντολές Περιστροφής, που φαίνονται στο παράρτημα με τους κώδικες λειτουργίες. Αυτές είναι δεκαδικές περιστροφές. Βρίσκονται έξω από τους στόχους αυτού του βιβλίου, και είναι πολύ απίθανο

Hisoft GENA3 Assembler. Page 1.
Pass 1 errors: 00

		10	:	FIG 11,15 SCREEN
				LEFT SCROLL
		20		
A410		30	ORG	42000
A410		40	ENT	42000
A410	0608	50	LD	B, B
A412	F5	60	PUSH	AF
A413	21FFFF	70	SCREEN	LD HL, #FFFF
A416	F1	80	PIXEL	POP AF
A417	CB16	90	RL	(HL)

A419	F5	100	PUSH	AF
A41A	2B	110	DEC	HL
A51B	7D	120	LD	A, L
A416	A7	130	AND	A
A41D	20F7	140	JR	NZ, PIXEL
A41F	7C	150	LD	A, H
A420	FECO	160	CP	#CO
A422	20F2	170	JR	NZ, PIXEL
A424	10ED	180	DJNZ	SCREEN
A426	F1	190	POP	AF
A427	C9	200	RET	

Pass 2 errors: 00

Table used: 3B from 141

Executes: 43000

THE CHECKSUMS REQUIRED BY THE HEX LOADER ARE

05E9, 05B2, 02B7

Σχήμα 11.16

ότι θα έχετε ποτέ την ευκαιρία να τις χρησιμοποιήσετε, εκτός ίσως όταν εργάζεστε πάνω στην οθόνη. Για το σκοπό αυτό η συμβολική αναπαράσταση στο παράρτημα είναι αρκετή. Προορίζονται για χρήση σε περιπτώσεις όπου χρειάζονται αριθμοί του κώδικα B.C.D. (Binary Coded Decimal), συχνά σε βοηθητικά εξαρτήματα όπως είναι οι οθόνες των ψηφιακών ρολογιών. Ο κώδικας B.C.D. είναι ένα σύστημα όπου χρησιμοποιούνται 4 bits για να περιέχουν μια τιμή μεταξύ 0 και 9, και επομένως σε ένα byte μπορούν να περιέχονται αριθμοί μεταξύ του 0 και 99 μανάχα, και όχι μεταξύ του 0 και 255 όπως συμβαίνει με την δεκαεξαδική γραφή. Αν ενδιαφέρεστε πραγματικά να μάθετε γι' αυτό το είδος εντολής θα χρειαστεί να καταλάβετε πλήρως όλες τις έννοιες που εξηγούνται σ' αυτό το βιβλίο, και κατόπιν να συνεχίσετε διαβάζοντας ένα βιβλίο σαν το «Programming the Z80» (ISBN 0 89588 069 5) του Zak.

Περίληψη

Ακολουθεί τώρα μια πολύ σύντομη περίληψη των εντολών που μαθατε σ' αυτό το κεφάλαιο:

r= μόνος καταχωρητής των 8 bits A,B,C,D,E,H, ή L

m= οποιοσδήποτε από τους r και (HL)

rr = ζεύγος καταχωρητών που χρησιμοποιείται σαν καταχωρητής των 16 bits

n= αριθμός των 8 bits

nn= αριθμός των 16 bits

() γύρω από αριθμό ή ζεύγος καταχωρητών = που περιέχεται στην

PC= Μετρητής Προγράμματος, Program Counter

SP= Μετρητής Στοίβας, Stack Pointer

Όλες οι μετατοπίσεις και όλες οι περιστροφές μπορούν να γίνουν σε οποιοδή-

ποτε m.

Οι περιστροφές έχουν ένα ειδικό κώδικα λειτουργίας ενός byte για χρήση στον καταχωρητή A. Αυτές οι ειδικές εντολές επηρεάζουν μόνο το carry flag.

Όλες οι άλλες περιστροφές και μετατοπίσεις επηρεάζουν όλα τα flags που μπορούν να χρησιμοποιηθούν σύμφωνα με τα περιεχόμενα του m μετά την πράξη.

Το P/N flag χρησιμοποιείται για να δείχνει την ισοτιμία. Οι δεκαδικές περιστροφές δεν επηρεάζουν το carry flag.

Στις προσημασμένες διαιρέσεις το bit του προσήμου μπορεί να φυλαχτεί με τη χρήση της εντολής SRA.

Μια κυκλική περιστροφή δεν παραλαμβάνει το bit που βρίσκεται στο carry flag πριν την εκτέλεση της εντολής.

Οι δεξιές μετατοπίσεις διαιρούν με το 2.

Οι αριστερές μετατοπίσεις πολλαπλασιάζουν με το 2.

ΑΥΤΟΜΑΤΕΣ ΜΕΤΑΚΙΝΗΣΕΙΣ ΚΑΙ ΕΡΕΥΝΕΣ

Είναι πιθανό να καταλάβατε από τον τίτλο του κεφαλαίου ότι η Z80 CPU είναι καλά εφοδιασμένη με αυτόματες εντολές, και ήδη έχετε μάθει μια από αυτές, την εντολή DJNZ.

Οι υπόλοιπες αυτόματες εντολές ανήκουν σε δύο διαφορετικούς χώρους. Στην ομάδα μεταφοράς περιοχών της μνήμης και έρευνας (black transfer and search group) και στην ομάδα εντολών εισόδου/εξόδου (input and outpout group). Η πρώτη από αυτές τις ομάδες θα εξηγηθεί εδώ, και η δράση της δεύτερης θα γίνει φανερή όταν διαβάσετε το επόμενο κεφάλαιο.

Υποθέστε για μια στιγμή ότι επιθυμείτε να μετακινήσετε τα περιεχόμενα μιας περιοχής της μνήμης σε μια άλλη περιοχή μνήμης. Αυτό θα γινόταν για να δημιουργηθεί χώρος σε μια σειρά εγγραφών (records) σε ένα πρόγραμμα βάσης δεδομένων (data-base), για να αποθηκευτεί μια οθόνη ή μέρος της σε μια άλλη περιοχή ή ακόμη για να ρολάρει η οθόνη όπως έγινε στα Σχήματα 11.15 και 16. Υποθέτοντας ότι γνωρίζετε το μήκος της περιοχής μνήμης που θα μετακινηθεί, το πρόγραμμα θα πρέπει να είναι περίπου σαν κι αυτό του Σχήματος 12.1.

Παρατηρήστε πως χρησιμοποιείται η εντολή EX DE, HL για να επιτρέψει το φόρτωμα της διεύθυνσης στο DE από τον καταχωρητή A, ανταλλάσσοντας προσωρινά τα περιεχόμενά της με τον HL, και κατόπιν αλλάζοντας τα ξανά. Αυτό θα μπορούσε να γίνει το ίδιο καλά (αν όχι καλλίτερα) χρησιμοποιώντας μια εντολή LD (DE), A, αλλά έτσι φαίνεται η χρήση των ανταλλαγών (exchanges).

Αν για παράδειγμα ο καταχωρητής A δεν ήταν εύκολα διαθέσιμος, και επρόκειτο να μετακινηθούν λιγότερο από 257 bytes, το πρόγραμμα θα μπορούσε εύκολα να μετατραπεί ώστε να χρησιμοποιεί τον καταχωρητή C και την εντολή DJNZ για το βρόχο.

Σ' αυτό το πρόγραμμα ο BC χρησιμοποιείται σαν Δυαδικός Μετρητής (Binary Counter) και ο DE σαν διεύθυνση προορισμού (DEstination address) για το byte που μετακινείται. Ο HL εδώ ενεργεί με τον κανονικό του ρόλο, δηλαδή δείχνει τη διεύθυνση του byte που θα τοποθετηθεί στον καταχωρητή A. Αυτός που έγραψε την ομάδα των συμβολικών εντολών (mnemonic instruction set) για τον Z80 σίγουρα μας διευκολύνει στο να θυμόμαστε τους ρόλους που έχουν συνήθως τα ζεύγη καταχωρητών!

Hisoft GENA3 Assembler. Page 1.

Pass 1 errors: 00

1 ; FIG 12,1 UP-WARD BLOCK MOVE BY
NORMAL MEANS

```

4E20          10          ORG  20000
4E20          20          ENT  20000
0000          30  ORIGIN  EQU  #????
0000          40  DEST   EQU  #????
0000          50  COUNT  EQU  #????
4E20  210000          60          LD  HL,ORIGIN
4E23  110000          70          LD  DE,DEST
4E26  010000          80          LD  BC,COUNT
4E29  7E              90  LOOP  LD  A,(HL)
4E2A  EB             100         EX  DE,HL
4E2B  77             110         LD  (HL),A
4E2C  EB             120         EX  DE,HL
4E2D  23             130         INC  HL
4E2E  13             140         INC  DE
4E2F  0B             150         DEC  BC
4E30  78             160         LD  A,B
4E31  B1             170         OR  C
4E32  20F5           180         JR  NZ,LOOP
4E34  C9             190         RET

```

Pass 2 errors: 00

Table used: 60 from 147
Executes: 20000

Σχήμα 12.1

Το Σχήμα 12.1 δουλεύει θαυμάσια όπου η διεύθυνση προορισμού (DESTINATION address) στον DE είναι χαμηλότερη από την αρχική διεύθυνση (ORIGIN) του HL, αλλά αν ο προορισμός βρίσκεται πάνω από την αρχική διεύθυνση, κατά λιγότερο από τον αριθμό των bytes που πρόκειται να μετακινηθούν, θα πάρете περιεργα αποτελέσματα. Γράψτε το πρόγραμμα και εξισώσετε το ORIGIN EQU με το C000h, το DEST EQU με C100h και το COUNT EQU με 3FFF. Οι αριθμοί ελέγχου για το Hex loader θα είναι 036F 04CC 00C9, και κατόπιν τρέξτε το. Θα διαπιστώσετε ότι το ίδιο περιεχόμενο επαναλαμβάνεται αρκετές φορές στην οθόνη. Αυτό που έκανε το πρόγραμμα ήταν να αντιγράψει από την αρχή της μνήμης της οθόνης (screen memory), C000h, σε μια διεύθυνση 100h αργότερα, και πάλι στην οθόνη. Όλα πήγαν καλά για τις πρώτες FFh θέσεις, αλλά κατόπιν αυτό που αντιγράφηκε δεν ήταν τα αρχικά περιεχόμενα, αλλά τα περιεχόμενα της αρχής της περιοχής της οθόνης, που είχαν μετακινηθεί εκεί από τους πρώτους 100h βρόχους.

Hisoft GENA3 Assembler. Page 1.

Pass 1 errors: 00

1 ; FIG 12,2 DOWN-WARD BLOCK MOVE
BY NORMAL MEANS

```

4E20          10          ORG  20000
4E20          20          ENT  20000
FEFF          30  ORIGIN  EQU  #FEFF

```

FFFF		40	DEST	EQU	#FFFF
3EFF		50	COUNT	EQU	#3EFF
4E20	21FFFE	60		LD	HL,ORIGIN
4E23	11FFFF	70		LD	DE,DEST
4E26	01FF3E	80		LD	BC,COUNT
4E29	7E	90	LOOP	LD	A,(HL)
4E2A	EB	100		EX	DE,HL
4E2B	77	110		LD	(HL),A
4E2C	EB	120		EX	DE,HL
4E2D	2B	130		DEC	HL
4E2E	1B	140		DEC	DE
4E2F	0B	150		DEC	BC
4E30	7B	160		LD	A,B
4E31	B1	170		OR	C
4E32	20F5	180		JR	NZ,LOOP
4E34	C9	190		RET	

Pass 2 errors: 00

Table used: 60 from 147
Executes: 20000

Σχήμα 12.2

Αυτό θα ήταν ακόμη διασκεδαστικότερο αν ανοίγατε χώρο σε μια πρόταση για να προσθέσετε κάτι. Σκεφθείτε τι θα συμβεί με την παρακάτω πρόταση αν πρόκειται να διατεθεί χώρος μετά το "bro" για να τοποθετηθεί ένα "w". Η πρόταση αρχίζει στη διεύθυνση nn, οπότε ο HL θα φορτωθεί με nn+12, μιας και εκεί χρειάζεται χώρος, ενώ ο DE θα φορτωθεί με nn+13, γιατί οτιδήποτε μετά το «ο» του «bro» πρόκειται να μετακινηθεί κατά μια θέση. Εφ' όσον 5 χαρακτήρες πρόκειται να μετακινηθούν ο BC θα φορτωθεί με 5.

How now bron cow
How now bronncow
How now bronnnow
How now bronnnnnw
How now bronnnnnn
How now bronnnnnnn

Αρχίζοντας
μετά από μια μετακίνηση,
μετά από δύο μετακινήσεις,
μετά από τρεις μετακινήσεις,
μετά από τέσσερις μετακινήσεις και
μετά από πέντε μετακινήσεις.

Θα πρέπει τώρα να είναι σαφές τι δεν πάει καλά, και ο τρόπος αντιμετώπισης του προβλήματος θα πρέπει να είναι κι αυτός φανερός αν σκεφθείτε λιγάκι. Πριν προχωρήσουμε στη διόρθωση της λειτουργίας του προγράμματος, μπορούμε να γνωριστούμε με τις πρώτες από τις αυτόματες εντολές μετακίνησης περιοχών της μνήμης. Θα αντικαταστήσουν τις γραμμές 90 μέχρι και 180 του listing του assembler, που έκαναν το «φόρτωμα» (LoadIn), τις αυξήσεις (Incrementing) και τις επαναλήψεις (Repeating).

Καθώς είσαστε συνηθισμένοι στο στυλ αυτού του βιβλίου γνωρίζετε τη συμβολική εντολή και ποιών λέξεων αποτελεί συντομογραφία, καθώς κι τη λειτουργία που εκτελεί, οπότε το μόνο που απομένει να αναφέρουμε είναι η μορφή της στο Hex και στο Δυαδικό.

ASSEMBLER	HEX	BINARY
LDIR	ED B0	11 101 101 10 110 000

Υπάρχει επίσης η αντίστροφη εντολή, για τη μετακίνηση περιοχών της μνήμης κατά τον ίδιο τρόπο, αλλά αρχίζοντας από την υψηλότερη διεύθυνση και του DESTINATION και του ORIGIN, και όχι με τη χαμηλότερη. Το Σχήμα 12.2 δείχνει τον τρόπο μετακίνησης της ίδιας περιοχής μνήμης στον ίδιο προορισμό που τραβάει πολύ σε μάκρος χρησιμοποιώντας την μειούμενη μετακίνηση (Decrementing move). Η εντολή που θα μπορούσε να χρησιμοποιηθεί για να αντικαταστήσει τις γραμμές 90 μέχρι 180 σ' αυτό το πρόγραμμα είναι:

ASSEMBLER	HEX	BINARY
LDDR	ED B8	11 101 101 10 111 000

Παρ' όλο που οι εντολές LDIR και LDDR εκτελούν τις ίδιες εργασίες όπως φαίνεται στα Σχήματα 12.1 και 2 δεν εργάζονται κατά τον ίδιο τρόπο. Ο καταχωρητής A δεν χρησιμοποιείται για να κάνει τη μεταφορά και, αντί να χρησιμοποιείται το zero flag για να ελέγχει αν το ζεύγος καταχωρητών BC παίρνει την τιμή 0, χρησιμοποιείται το P/V flag. Αυτό θα γίνεται πάντα «0» (reset) μετά την εκτέλεση της εντολής. Κανένα άλλο από τα flags που μπορούν να ελεγχθούν δεν επηρεάζεται κατά κανένα τρόπο από οποιαδήποτε εντολή μετακίνησης περιοχών της μνήμης.

Όταν τα δεδομένα που μετακινούνται δεν πρέπει να υποστούν αλλαγές, η εντολή LDIR θα είναι ασφαλής για να χρησιμοποιηθεί όταν μια περιοχή της μνήμης μετακινείται προς τα κάτω, ενώ η εντολή LDDR πρέπει να χρησιμοποιείται για τις προς τα πάνω μετακινήσεις. Προφανώς η εντολή LDDR χρειάζεται το ζεύγος καταχωρητών HL για να περιέχει τη διεύθυνση της κορυφής της περιοχής που θα μετακινηθεί, και το ζεύγος καταχωρητών DE για να περιέχει την υψηλότερη διεύθυνση μνήμης και θα γεμίσει. Αντίστροφα, όταν χρησιμοποιείται η εντολή LDIR, ο HL θα περιέχει τη βάση της διεύθυνσης της προέλευσης των δεδομένων, και ο DE τη διεύθυνση της βάσης του προορισμού.

Και οι δύο εντολές μπορούν να χρησιμοποιηθούν για να γεμίσουν μια περιοχή με το ίδιο byte, και μια έκδοση του προγράμματος SCREEN FILL MK3 στο Σχήμα 9.2 χρησιμοποιώντας την εντολή LDDR δίνεται στο Σχήμα 12.3. Αυτό χρησιμοποιεί επίτηδες την τεχνική "overcopying", μετακινώντας κατά 1 byte προς τα κάτω. Το κάθε byte που γεμίζει χρησιμοποιείται με τη σειρά του σαν πρωτότυπο για το επόμενο byte που θα γεμίσει. Παρατηρήστε πως το ζεύγος καταχωρητών HL αρχίζει στη διεύθυνση FFFFh και όχι στη 0000h, γιατί ο BC μειώνεται μετά την πρώτη μεταφορά.

Hisoft GENA3 Assembler. Page 1.

Pass 1 errors: 00

	1	;	FIG 12,3	SCREEN FILL MK 3
4E20	10		ORG	20000
4E20	20		ENT	20000

FFFF		30	ORIGIN	EQU	#FFFF
FFFE		40	DEST	EQU	#FFFE
3FFF		50	COUNT	EQU	#3FFF
4E20	21FFFF	60		LD	HL,ORIGIN
4E23	11FEFF	70		LD	DE,DEST
4E26	01FF3F	80		LD	BC,COUNT
4E29	EDBB	90		LDDR	
4E2B	C9	100		RET	

Pass 2 errors: 00

Table used: 49 from 132
Executes: 20000

THE CHECK-SUMS REQUIRED BY THE HEX LOADER ARE
0659 0101

Σχήμα 12.3

Σε όλα τα παραπάνω παραδείγματα το μήκος της περιοχής της μνήμης που θα μετακινηθεί είναι γνωστό, αλλά θα ήταν μεγάλη ντροπή αν έπρεπε να ξαναγράφεται ολόκληρη η έκδοση του προγράμματος κάθε φορά που θα είναι ανάγκη να περιληφθεί κάποιος έλεγχος για το πλησίασμα του τέλους. Αυτό θα τελείωνε όπως φαίνεται στο Σχήμα 12.4, υποθέτοντας ότι ο 00 χρησιμοποιήθηκε για να δείχνει το τέλος. Ο BC εξακολουθεί να χρησιμοποιείται για να θέτει ένα όριο στη μνήμη που μπορεί να μετακινηθεί, αλλιώς σε περίπτωση που δεν υπήρχε δείκτης του τέλους για κάποιο λόγο, το πρόγραμμα δεν θα τελείωνε ποτέ. Ίσως χειρότερα ακόμη, να σβηγόταν το ίδιο το πρόγραμμα, και φυσικά θα ακολουθούσε η απώλεια του προγράμματος. Δεν υπάρχει καμία πεπονόφλουδα όταν γράφετε γλώσσα μηχανής, θα πρέπει να τις τοποθετήσετε οι ίδιοι!

Hisoft GENA3 Assembler. Page 1.

Pass 1 errors: 00

			1 ; FIG 12,4 MOVE TO END MARKED BY 0		
4E20		10	ORG	20000	
4E20		20	ENT	20000	
FFFF		30	ORIGIN	EQU	#FFFF
FFFE		40	DEST	EQU	#FFFE
3FFF		50	COUNT	EQU	#3FFF
4E20	21FFFF	60	LD	HL,ORIGIN	
4E23	11FEFF	70	LD	DE,DEST	
4E26	01FF3F	80	LD	BC,COUNT	
4E29	7E	90	LOOP	LD	A,(HL)
4E2A	12	100	LD	(DE),A	
4E2B	2B	110	DEC	HL	
4E2C	1B	120	DEC	DE	
4E2D	0B	130	DEC	BC	
4E2E	78	140	LD	A,B	
4E2F	B1	150	OR	C	
4E30	2004	160	JR	Z,LIMIT	
4E32	AF	170	XOR	A	
4E33	BE	180	CP	(HL)	
4E34	20F3	190	JR	NZ,LOOP	

4E36 C9 200 LIMIT RET

Pass 2 errors: 00

Table used: 72 from 147

Executes: 20000

Σχήμα 12.4

Ευτυχώς η Zilog έχει σκεφτεί ακόμα κι αυτό, και έχει δημιουργήσει ένα ζευγάρι εντολών που είναι ίδιο με τις LDIR και LDDR, αλλά χωρίς το Repeat, και ποτέ δεν θα μαντευάτε τη συμβολική μορφή τους, έτσι δεν είναι;

ASSEMBLER	HEX	BINARY
LDD	ED AB	11 101 101 10 101 000
LDI	ED A0	11 101 101 10 100 000

Δεν είναι και τόσο απλό να αλλάξετε το Σχήμα 12.4 για να ενσωματώσετε την LDD όσο φαίνεται με την πρώτη ματιά, γιατί το P/V flag χρησιμοποιείται για να δείχνει αν το ζεύγος καταχωρητών BC πλησιάζει το μηδέν, στη θέση του zero flag στο αρχικό μακρύ πρόγραμμα. Χρησιμοποιώντας την LDD η συνθήκη με την οποία γίνονται το «πήδημα» (jump) στο LIMIT πρέπει να αλλάξει σε PO, γιατί το flag γίνεται «0» (reset) όταν BC=0. Αυτό με τη σειρά του οδηγεί στην ανάγκη πραγματοποίησης μιας μεγαλύτερης αλλαγής. Ένα relative jump δεν έχει τη δυνατότητα να γίνει υπό συνθήκη με το P/V flag, οπότε το «πήδημα» θα πρέπει να μετατραπεί σε απόλυτο (absolute jump). Το Σχήμα 12.5 δείχνει το ξαναγραμμένο πρόγραμμα με ενσωματωμένη την εντολή LDD.

Hisoft GENA3 Assembler. Page 1.

Pass 1 errors: 00

```
1 ; FIG 12,5 MOVE TO END MARKED BY
0 MK 2
4E20 10 ORG 20000
4E20 20 ENT 20000
FFFF 30 ORIGIN EQU #FFFF
FFFE 40 DEST EQU #FFFE
3FFF 50 COUNT EQU #3FFF
4E20 21FFFF 60 LD HL,ORIGIN
4E23 11FEFF 70 LD DE,DEST
4E26 01FF3F 80 LD BC,COUNT
4E29 EDAB 90 LOOP LDD
4E2B E2324E 171 JP PO,LIMIT
4E2E AF 173 XOR A
4E2F BE 174 CP (HL)
4E30 20F7 180 JR NZ,LOOP
4E32 C9 190 LIMIT RET
```

Pass 2 errors: 00

Table used: 72 from 141

Executes: 20000

Σχήμα 12.5

Οι επόμενες αυτόματες εντολές, που είναι και οι τελευταίες με τις οποίες θα ασχοληθούμε σε τούτο το κεφάλαιο, είναι οι εντολές έρευνας μιας περιοχής της μνήμης (block search instructions). Αυτές ακολουθούν πολύ στενά τη μορφή που έχουν οι κώδικες λειτουργίας μετακίνησης περιοχών της μνήμης αλλά, όπως φαίνεται και από το όνομά τους χρησιμοποιούνται για να ψάχνουν σε περιοχές της μνήμης για κάποιο byte που περιέχει μια συγκεκριμένη τιμή. Για να δείξουμε τις ίδιες αναλογίες που παρουσιάστηκαν στις μετακινήσεις περιοχών της μνήμης το Σχήμα 12.6 δίνει τη μακριά έκδοση ενός προγράμματος που βρίσκει ένα byte στη μνήμη με περιεχόμενο 65 (τον κωδικό του «Α») αρχίζοντας την έρευνα από τη διεύθυνση FFFFh και εξετάζοντας 3FFFh bytes πριν παραιτηθεί αν δεν βρεθεί το ζητούμενο. Όταν φτάσουμε στο τέλος του βρόχου που κάνει την έρευνα (η label DONE) το zero flag θα είναι «1» (set) αν έχει βρεθεί το ζητούμενο, και «0» (reset) αν όχι.

Θα δείτε ότι υπήρξαν δυσκολίες κατά την αποθήκευση του καταχωρητή A ενώ γινόταν ο έλεγχος για την περίπτωση που ο BC γινόταν 0. Δεν μπορούσε να αποθηκευτεί στη στοίβα με PUSH γιατί θα αποθηκεύονταν και τα flags, και αυτό θα ακύρωνε τον έλεγχο όταν το POP θα γινόταν πριν το jump. Τα flags θα είχαν αποκατασταθεί με τον καταχωρητή A πριν χρησιμοποιηθεί το αποτέλεσμα του ελέγχου. Ο καταχωρητής D έχει τεθεί επομένως σε υπηρεσία για την προσωρινή αποθήκευση του A όσο γίνονται οι έλεγχοι.

Hisoft GENA3 Assembler. Page 1.

Pass 1 errors: 00

```
10 ; FIG 12,6 BLOCK SEARCH THE HARD WAY
4E20 20 ORG 20000
4E20 30 ENT 20000
FFFF 40 START EQU #FFFF
3FFF 50 COUNT EQU #3FFF
4E20 21FFFF 60 LD HL, START
4E23 01FF3F 70 LD BC, COUNT
4E26 3E41 80 LD A, 65
4E28 BE 90 LOOP CP (HL)
4E29 2B 100 DEC HL
4E2A 0B 110 DEC BC
4E2B 2807 120 JR Z, DONE
4E2D 57 130 LD D, A
4E2E 7B 140 LD A, B
4E2F B1 150 OR C
4E30 7A 160 LD A, D
4E31 20F5 170 JR NZ, LOOP
4E33 3F 180 CCF
4E34 C9 190 DONE RET
```

Pass 2 errors: 00

Table used: 59 from 143
Executes: 20000

Σχήμα 12.6

Σαν αποτέλεσμα έχουμε ένα πρόγραμμα που κάνει μια πετυχημένη σύγκριση (CompAre), μείωση (Decrement) και επανάληψη (Repeat) σε μια περιοχή της μνήμης, δείχνοντας το αποτέλεσμα με το zero flag όταν έχει τελειώσει. Η αυτόματη εντολή που θα χρησιμοποιούταν κανονικά γι' αυτή τη δουλειά, η έκδοση του ίδιου προγράμματος που κάνει έρευνα προς τα πάνω, και οι εκδόσεις χωρίς την επανάληψη είναι:

ASSEMBLER	HEX	BINARY
CPIR	ED B1	11 101 101 10 110 001
CPDR	ED B9	11 101 101 10 111 001
CPI	ED A1	11 101 101 10 100 001
CPD	ED A9	11 101 101 10 101 001

Αυτές οι εντολές χρησιμοποιούν το P/V flag για να δείχνει ότι το μέτρημα (στον BC) έφτασε στο 0, όπως ακριβώς περιγράφηκε νωρίτερα στις αυτόματες εντολές LD, αλλά επιπρόσθετα το zero flag γίνεται «1» (set) για να δείχνει ότι βρέθηκε το ζητούμενο, ή «0» (reset) στην αντίθετη περίπτωση.

Το Σχήμα 12.7 δείχνει μια ξαναγραμμένη έκδοση του Σχήματος 12.6 χρησιμοποιώντας την εντολή CPDR. Η CPIR αυξάνει (INCrements) το ζεύγος καταχωρητών HL μετά από κάθε σύγκριση αντί να μειώνεται, και οι δύο εντολές χωρίς το R για επανάληψη (Repeat), κάνουν το ίδιο αλλά χωρίς την επανάληψη.

Hisoft GENA3 Assembler. Page 1.

Pass 1 errors: 00

```
          .10 ; FIG 12,7  BLOCK SEARCH USING CPDR
4E20          20          ORG  20000
4E20          30          ENT  20000
FFFF          40 START  EQU  #FFFF
3FFF          50 COUNT  EQU  #3FFF
4E20  .21FFFF          60          LD  HL, START
4E23  01FF3F          70          LD  BC, COUNT
4E26  3E41           80          LD  A, 65
4E28  EDB9          90 LOOP  CPDR
4E2A  C9          190 DONE  RET
```

Pass 2 errors: 00

Table used: 59 from 132
Executes: 20000

Σχήμα 12.7

Είναι εύκολο να τροποποιήσουμε το πρόγραμμα στο Σχήμα 12.7 για να ψάξουμε για μια σειρά θέσεων μνήμης αντί για μια μόνο τιμή, προσθέτοντας μια μικρή ποσότητα γλώσσας μηχανής. Μια εφαρμογή θα μπορούσε να είναι το ψάξιμο για να βρεθεί συγκεκριμένη λέξη ή φράση σε αποθηκευμένα δεδομένα, ή το ψάξιμο για λέξεις-κλειδιά (keywords) που έχουν τοποθετηθεί σε κάποιο adventure game. Το Σχήμα 12.8 δείχνει ένα πρόγραμμα που κάνει ακριβώς αυτό. Επιτρέπει την εισαγωγή μιας σειράς χαρακτήρων που πρόκειται να ερευνηθούν, που τελειώνει όταν πιέζεται το πλήκτρο [ENTER], και κατόπιν επιστρέφει στην διεύθυνση όπου αρχίζει η σειρά, αν βρίσκεται στη μνήμη.

Hisoft GENA3 Assembler. Page 1.

Pass 1 errors: 00

```

                                10 ; FIG 12,8  BLOCK SEARCH FOR STRING
                                20 ; OF MATCHING LOCATIONS USING CPIR
7530                                30          ORG  30000
7530                                40          ENT  30000
BB18                                50 GETKEY EQU  47896
BB5A                                60 PRINT EQU  47962
0000                                70 START EQU  #0000
7530                                80 COUNT EQU  30000
7530 217575                          90          LD   HL, FREE
7533  E5                             100         PUSH HL
7534  D1                             110         POP  DE
7535  CD18BB                          120 INPUT  CALL GETKEY
7538  77                             130         LD   (HL), A
7539  CD5ABB                          140         CALL PRINT
753C  23                             150         INC  HL
753D  FE0D                            160         CP   #0D
753F  20F4                            170         JR   NZ, INPUT
7541  210000                          180         LD   HL, START
7544  013075                          190         LD   BC, COUNT
7547  1A                             200 LOOK  LD   A, (DE)
7548  D5                             210         PUSH DE
7549  EDB1                            220         CPIR
754B  C5                             230         PUSH BC
754C  E5                             240         PUSH HL
754D  2012                            250         JR   NZ, NOFIND
754F  13                             260 NXT_CH INC  DE
7550  1A                             270         LD   A, (DE)
7551  BE                             280         CP   (HL)
7552  23                             290         INC  HL
7553  28FA                            300         JR   Z, NXT_CH
7555  FE0D                            310 FINI? CP   #0D
7557  E1                             320         POP  HL
7558  C1                             330         POP  BC
7559  D1                             340         POP  DE

```

755A	20EB	350	JR	NZ, LOOK
755C	2B	360	FOUND	DEC HL
755D	227575	370	LD	(FREE), HL
7560	C9	380	RET	
7561	C1	390	NOFIND	POP BC
7562	E1	400	POP	HL
7563	D1	410	POP	DE
7564	23	420	INC	HL
7565	18F5	430	JR	FOUND

Pass 2 errors: 00

Table used: 145 from 184
Executes: 30000

THE CHECK-SUMS REQUIRED BY THE HEX LOADER ARE
05A5 0378 04FD 042E 055E 02E2

Σχήμα 12.8

```
10 PRINT "HELLO"
20 CALL 30000
30 N= PEEK (30069)+256*PEEK(30070) :PRINT N
40 PRINT CHR$( PEEK (N));CHR$( PEEK (N+1));CHR$( PEEK
(N+2));CHR$( PEEK (N+3))
```

Αυτό το πρόγραμμα μπορεί να αλλάξει ώστε να ταιριάζει σε οποιαδήποτε έρευνα χρειαστεί να κάνετε. Οι γραμμές 120 μέχρι 190 του listing του assembler μπορούν να ανταλλαγούν με οποιαδήποτε ρουτίνα εισαγωγής χρειάζεται, ενώ αυτό που γίνεται μετά τη συμπλήρωση της έρευνας μπορεί επίσης να αλλάξει. Όπως δίνεται, ο HL θα περιέχει είτε τη διεύθυνση όπου αρχίζει η σειρά των χαρακτήρων που ερευνάται, ή το 0 αν δεν βρέθηκε. Για τους σκοπούς αυτού του προγράμματος επίδειξης αποθηκεύεται και αυτό στη μνήμη, ώστε να μπορεί να προσπελαθεί από τη BASIC.

Η label FINI? δεν χρησιμοποιείται παρά μόνο για να δείχνει ότι εδώ είναι που γίνεται ο έλεγχος για να διαπιστωθεί αν έχει βρεθεί το ζητούμενο. Πρέπει πάντοτε να προσέχετε ώστε να αποφεύγετε ρουτίνες αυτού του τύπου που βρίσκουν το ζητούμενο από δικό τους δείγμα, αλλιώς ποτέ δεν θα πάρετε ένα "Not Found" (Δεν βρέθηκε) ακόμη κι όταν στην πραγματικότητα δεν υπάρχει αντίγραφο.

Τέλος, ένα σημαντικότερο θέμα που αφορά όλες τις αυτόματες εντολές σε τούτο το κεφάλαιο, που πρέπει να γνωρίζετε ή να χρησιμοποιείτε, είναι η σειρά των γεγονότων. Το ζεύγος καταχωρητών HL και το ζεύγος καταχωρητών DE όπου χρησιμοποιείται, αυξάνονται ή μειώνονται πριν από το ζεύγος καταχωρητών BC. Επομένως θα δείχνουν πάντοτε την επόμενη θέση σύμφωνα με τη σειρά που ακολουθούσαν, αφού εκτελέσουν την εργασία τους.

Αυτό μπορεί να διαπιστωθεί κοιτάζοντας στο Σχήμα 12.8 από τη label NXT — CH, όπου ελέγχεται ο επόμενος χαρακτήρας της σειράς. Το ζεύγος καταχωρητών DE πρέπει να αυξηθεί ώστε να δείχνει τον επόμενο χαρακτήρα του δείγματος, ενώ το ζεύγος καταχωρητών HL δείχνει ήδη τον επόμενο χαρακτήρα της σειράς χαρακτήρων που εξετάζεται, θγαίνοντας από το βρόχο CPIR. Αυτός είναι επίσης ο λόγος της μείωσης του HL πριν τελειώσει το πρόγραμμα στις label FOUND. Εφ' ό-

σον είναι γνωστό ότι το ζεύγος καταχωρητών BC θα περιέχει το 0 αν ερευνήθηκε η καθορισμένη περιοχή της μνήμης και δεν βρέθηκε το ζητούμενο, αυτό είναι που γίνεται POP στο ζεύγος καταχωρητών στη label NOFIND, εξοικονομώντας δύο bytes και δίνοντας τη δυνατότητα στο INC που το επιτρέπει αυτό να μειωθεί πάλι στο 0 μετά το «πήδημα» στο FOUND.

Πειραματιστείτε με αυτή τη ρουτίνα και κατόπιν ξαναγράψτε την για να χρησιμοποιήσετε την εντολή CPDR και δείτε αν παίρνετε τα ίδια αποτελέσματα όταν ψάχνετε για το "HELLO" στο BASIC πρόγραμμα.

Περίληψη

Ακολουθεί τώρα μια πολύ σύντομη περίληψη των εντολών που μάθατε σε τούτο το κεφάλαιο:

r= μονός καταχωρητής των 8 bits A,B,C,D,E,H ή L

r= ζεύγος καταχωρητής που χρησιμοποιείται σαν καταχωρητής των 16 bits

n= αριθμός των 8 bits

nr= αριθμός των 16 bits

() γύρω από αριθμό ή ζεύγος καταχωρητών = που περιέχουν στην

PC= Program Counter, Μετρητής Προγράμματος

SP= Stack Pointer, Δείκτης Στοίβας

Η LDIR φορτώνει τα περιεχόμενα της διεύθυνσης HL στη διεύθυνση DE, αυξάνει τα DE και HL, μειώνει τον BC και κατόπιν, αν ο BC δεν είναι 0, επαναλαμβάνει.

Η LDDR φορτώνει τα περιεχόμενα της διεύθυνσης HL στη διεύθυνση DE, μειώνει τα DE και HL, μειώνει τον BC και κατόπιν, αν ο BC δεν είναι 0, επαναλαμβάνει.

Οι LDI και LDD ενεργούν ακριβώς όπως οι εντολές LDIR και LDDR, αλλά χωρίς να επαναλαμβάνουν.

Η CPIR συγκρίνει τα περιεχόμενα του καταχωρητή A με τα περιεχόμενα της διεύθυνσης HL, αυξάνει τον HL, μειώνει τον BC και επαναλαμβάνει εκτός αν βρέθηκε το ζητούμενο ή ο BC έγινε 0. Αν έχει βρεθεί το ζητούμενο, το zero flag θα γίνει «1» (set).

Η CPDR συγκρίνει τα περιεχόμενα του καταχωρητή A με τα περιεχόμενα της διεύθυνσης HL, μειώνει τον HL, μειώνει τον BC, και επαναλαμβάνει εκτός αν βρέθηκε το ζητούμενο ή ο BC έγινε 0. Αν έχει βρεθεί το ζητούμενο, το zero flag θα γίνει «1» (set).

Οι CPI και CPD ενεργούν ακριβώς όπως οι εντολές CPIR και CPDR, αλλά χωρίς να επαναλαμβάνουν.

Σε όλες τις παραπάνω εντολές το P/V flag χρησιμοποιείται για να ελέγχει αν ο BC έχει γίνει 0, και όταν συμβαίνει από το flag γίνεται «0» (reset), επομένως η συνθήκη ελέγχου PO θα «πηδήξει» όταν είναι BC=0.

ΕΠΙΚΟΙΝΩΝΩΝΤΑΣ ΜΕ ΤΟΝ ΕΞΩΤΕΡΙΚΟ ΚΟΣΜΟ

Όλες οι εντολές μέχρι τώρα έχουν ασχοληθεί με την επεξεργασία πληροφοριών που περιέχονται στον υπολογιστή. Μπορεί να έχετε αναρωτηθεί πώς ο υπολογιστής παίρνει αυτές τις πληροφορίες για να αρχίσει, ή μπορεί να το έχετε σαν δεδομένο πως όταν πιέζετε ένα πλήκτρο, ο Amstrad το αντιλαμβάνεται. Στη πραγματικότητα, αν δεν του λέγαμε ειδικά να δέχεται πληροφορίες από έξω στον δικό του μικρό κόσμο της μνήμης, της οθόνης και του επεξεργαστή, ο υπολογιστής σας θα καθόταν εκεί ευτυχισμένος, τρέχοντας τα προγράμματα του ή ο,τιδήποτε άλλο, και αγνοώντας σας τελείως. Θα μπορούσατε να πιέζετε πλήκτρα μέχρι να μαυρίσει το μάτι σας, ή να πεθάνετε από την πείνα, και του υπολογιστή δεν θα του καιγότανε μπιτάκι (όπως λέμε καρφάκι). Το μόνο πράγμα που θα τον στεναχωρούσε, θα ήταν αν του κλείνατε το διακόπτη. Από την άλλη μεριά αν ένα πρόγραμμα χρειαζόταν να μεταφέρει πληροφορίες προς τα έξω σε ο,τιδήποτε άλλο εκτός από την οθόνη, θα έπρεπε να δοθούν στον υπολογιστή τα μέσα για να το κάνει αυτό. Το λειτουργικό σύστημα προσφέρει τα μέσα προσπέλασης στις υπάρχουσες μονάδες, όπως είναι το πληκτρολόγιο και το τσιπ του ήχου (sound chip), που βρίσκονται έξω από το άμεσο οπτικό πεδίο της CPU, οπότε είναι πολύ απίθανο να χρειαστεί να τα προσπελάσετε εσείς.

Χωρίς να εμβαθύνουμε πάρα πολύ στις τεχνικές προδιαγραφές της CPU, πράγμα που θα μας έκανε να αλλάξουμε το θέμα μας, σας παρουσιάζουμε μια πολύ στοιχειώδη εξήγηση του πώς επικοινωνεί με ό,τι βρίσκεται έξω από τα όριά της.

Θα πρέπει τώρα να είμαστε αρκετά εξοικειωμένοι με το δυαδικό σύστημα, και με το πώς μια σειρά από 1 και 0 μπορούν να σχηματίσουν έναν αριθμό. Έχετε μάθει επίσης ότι ο υπολογιστής αντιστοιχεί στο 1 έναν ανοικτό διακόπτη («on») και στο 0 ένα κλειστό («off»). Η Z80 CPU έχει σαράντα ακροδέκτες (pins), καθένας από τους οποίους χρησιμοποιείται για μια συγκεκριμένη λειτουργία. Τα δύο σύρματα που συνδέουν το πληκτρολόγιο με το monitor ενεργούν κατά παρόμοιο τρόπο. Η μια γραμμή έχει δύο σύρματα που δίνουν ισχύ (ηλεκτρισμό) στον υπολογιστή, και το άλλο σύρμα, που έχει έξι υποδοχές καθεμία από τις οποίες πηγαίνει σε έναν ακροδέκτη, στέλνει στο monitor τις πληροφορίες της εικόνας.

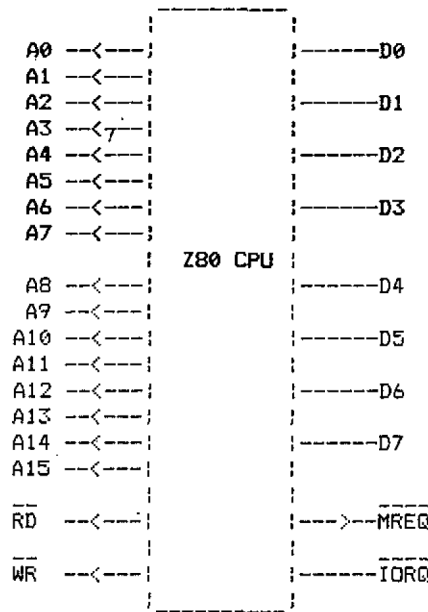
Δεκαέξι από τους ακροδέκτες της CPU χρησιμοποιούνται για να δίνουν διεύθυνση με την οποία επιθυμεί να επικοινωνήσει η CPU, ενώ οχτώ χρησιμοποιούνται για να στέλνουν ή να παίρνουν τα δεδομένα. Άλλοι ακροδέκτες χρησιμοποιούνται για να δείχνουν αν πρόκειται να χρησιμοποιηθεί η μνήμη, ή αν πρόκειται αν υπάρξει επικοινωνία με τον εξωτερικό κόσμο, και αν η CPU πρόκειται να στείλει πληροφορίες ή περιμένει να πάρει.

Οι δεκαέξι ακροδέκτες που δίνουν τη διεύθυνση είναι γνωστοί σαν Δίαυλος Διευθύνσεων (Address Bus). Συνήθως συντομεύεται σε ένα απλό A, και η αναφορά στον κάθε ακροδέκτη γίνεται σύμφωνα με τον αριθμό του bit που δίνει. Ο ακρο-

δέκτης που δίνει το bit 0 (0 ή 1 δεκαδικό) ονομάζεται A0, ο ακροδέκτης που δίνει το bit 1 (το 2 όταν είναι στην κατάσταση 1) ονομάζεται A1, κ.ο.κ. μέχρι τον A15 που δίνει το bit 15 (32768 στο δεκαδικό όταν είναι στην κατάσταση 1). Οι οχτώ ακροδέκτες μέσω των οποίων περνούν τα δεδομένα ονομάζονται Δίαυλος Δεδομένων (Data Bus) και είναι γνωστοί σαν D0 μέχρι D7. Ο Δίαυλος αυτός φαίνεται στο Σχήμα 13.1, μαζί με μερικούς από τους άλλους ακροδέκτες.

Όταν η CPU εκτελεί μια εντολή σαν την LD A, (3456) θα δώσει σήμα πως επιθυμεί να χρησιμοποιήσει τη μνήμη, και πως επιθυμεί να διαβάσει πληροφορίες από τη διεύθυνση που έχει τοποθετήσει στον Δίαυλο Διευθύνσεων. Κατόπιν θα διαβάσει τα περιεχόμενα αυτής της διεύθυνσης της μνήμης μέσω του Δίαυλου Δεδομένων.

ΜΕΡΙΚΟΙ ΑΠΟ ΤΟΥΣ ΑΚΡΟΔΕΚΤΕΣ ΤΟΥ Z80



Το RD δίνει σήμα για «διάβασμα» (Read). Το WR δίνει σήμα για «γράφισμο» (Write). Το MREQ δίνει σήμα ότι «ζητάει μνήμη» (Memory REQuest), με άλλα λόγια ότι πρόκειται να χρησιμοποιηθεί η μνήμη. Το IORQ δίνει σήμα ότι «ζητάει να πραγματοποιήσει είσοδο ή έξοδο» (Input Output REQuest). Η γραμμή πάνω από αυτούς τους ακροδέκτες σημαίνει πως είναι ενεργοί όταν είναι «κάτω» (low, το δυαδικό 0).

Σχήμα 13.1

Αν θέλετε να χρησιμοποιήσετε κάτι άλλο εκτός από τη μνήμη θα πρέπει να πείτε ειδικά στη CPU ότι θέλετε να στείλετε (send out) ή να πάρετε (receive in) από κάπου αλλού. Αυτό το κάνετε με μια εντολή OUT ή IN. Υπάρχει ένας μεγάλος αριθμός εντολών αυτού του τύπου που είναι διαθέσιμες στη CPU αλλά, εξαιτίας του τρόπου με τον οποίο σχεδιάστηκε ο Amstrad, μόνο μια IN και μια OUT εντολή έχουν ενδιαφέρον.

Η διεύθυνση για οποιαδήποτε εξωτερική επικοινωνία με τη CPU καθορίζεται

από το ζεύγος καταχωρητών BC, και οι εντολές είναι:

ASSEMBLER	BINARY
OUT (C), r	11 101 101 (EDh) 01 r 001
IN r, (C)	11 101 101 01 r 000

Ο καταχωρητής B δίνει το A8 μέχρι το A15 και ο καταχωρητής C δίνει το A0 μέχρι το A7. Αν τοποθετήσουμε τον 1234h στον BC θα κάνει επομένως τα A0 μέχρι A15:

A15) 00010010 00110100 (A0

Η διεύθυνση για κάποια εξωτερική μονάδα συνήθως δεν ονομάζεται διεύθυνση, αλλά «Θύρα» (Port). Έτσι αποφεύγεται η σύγχυση πάνω στο αν μια διεύθυνση είναι εσωτερική (δηλαδή, στη μνήμη) ή εξωτερική. Οποιαδήποτε μεταφέρεται μέσω μιας θύρας είναι γνωστό σαν I/O που είναι συντομογραφία του In/Out.

Εξαιτίας του σχεδιασμού του Amstrad πολύ λίγες τιμές μπορούν να χρησιμοποιηθούν για τον BC. Η πιθανότερη χρήση που θα έχετε γι' αυτές τις εντολές θα είναι όταν χρησιμοποιείτε κάποιο είδος περιφερειακού. Αν βρίσκεστε σε ένα επίπεδο όπου δημιουργείτε ή ελέγχετε κάτι τέτοιο, τα A8 μέχρι A15 του δίαυλου διευθύνσεων, και επομένως ο καταχωρητής B, θα πρέπει να περιέχει είτε τον F8h F9h FAh ή τον F8h. Για όλες αυτές τις γραμμές το A10 θα βρίσκεται στη λογική κατάσταση 0 (low). Όσο αυτή η γραμμή είναι «low», και τα οκτώ bits στη βάση του δίαυλου διευθύνσεων περιέχουν αριθμούς μεταξύ E0h και FEh συμπεριλαμβανομένου, δεν θα επεμβαίνετε σε οποιαδήποτε διανομή που έχει κάνει ο Amstrad για υπάρχουσες ή μελλοντικές μονάδες που θα χρησιμοποιηθούν.

Περισσότερες λεπτομέρειες για τα περιφερειακά που ήδη συνδέονται με τον Amstrad χρησιμοποιώντας τις εντολές IN και OUT μπορείτε να βρείτε στο Amstrad Programmer's Reference Manual.

Ένα γρήγορο πρόγραμμα που δεν έχει καμιά πρακτική χρησιμότητα, αλλά που δείχνει τη χρήση της εντολής OUT, δίνεται στο Σχήμα 13.2. Σας επιτρέπει να ανοίγετε και να κλείνετε τον κινητήρα του κασετόφωνου. Το κασετόφωνο βρίσκεται στην άλλη πλευρά ενός κυκλώματος interface (UPD 8255) που έχει τρία κανάλια I/O. Η πρόσβαση στο κανάλι A γίνεται με τη θύρα F4xxh, στο κανάλι B με τη θύρα F5xxh και στο κανάλι C με τη θύρα F6xxh. Ο έλεγχος γίνεται μέσω της θύρας F7xxh. Σε κάθε θύρα το xx μπορεί να πάρει οποιαδήποτε τιμή (και θα κρατηθεί στον καταχωρητή C) αλλά εκτός αν χρησιμοποιείτε κάποια από τις διευθύνσεις που δεν χρησιμοποιούνται για το A0 μέχρι A7, που αναφέρθηκαν προηγουμένως, πάτε γυρεύοντας να βρείτε μελάδες.

Hisoft GENA3 Assembler. Page 1.

Pass 1 errors: 00

	10 ; F16 13,2 PROGRAM TO TURN CASSETTE
	20 ; MOTOR ON OR OFF
BB18	30 GETKEY EQU 47896
7530	40 ORG 30000

7530		50	ENT	30000
7530	01E0F6	60	LD	BC, #F6E0
7533	3E10	70	LD	A, #10
7535	ED79	80	OUT	(C), A
7537	CD18BB	90	CALL	GETKEY
753A	AF	100	XOR	A
753B	ED79	110	OUT	(C), A
753D	C9	120	RET	

Pass 2 errors: 00

Table used: 35 from 137

Executes: 30000

THE CHECK-SUMS REQUIRED BY THE HEX LOADER ARE
052B 02DE

Σχήμα 13.2

ΑΛΛΕΣ ΕΝΤΟΛΕΣ

Η Z80 CPU διαθέτει δύο ομάδες καταχωρητών γενικής χρήσης, καθώς και ένα δεύτερο καταχωρητή A και καταχωρητή των flags. Με τους περισσότερους υπολογιστές μπορείτε να χρησιμοποιείτε τις εντολές του Z80 για να αλλάζετε μεταξύ της πρώτης και της δεύτερης ομάδας καταχωρητών γενικής χρήσης, και/ή τους δεύτερους καταχωρητές A και F, όποτε το επιθυμείτε. Αν δεν γνωρίζετε πολύ καλά το λειτουργικό σύστημα, ο Amstrad αποκλείει τη χρήση μιας ολόκληρης ομάδας αυτών των καταχωρητών.

Οι εντολές που σας επιτρέπουν να αλλάζετε την ομάδα των καταχωρητών που χρησιμοποιείτε φαίνονται στο παράρτημα, αλλά συνιστάται θερμά να ξεχάσετε την ύπαρξή τους μέχρι να κατέχετε στην εντέλεια το Programmer's Reference Manual. Αυτό δεν θα είναι εύκολη δουλειά, και κάποιος περιοδικό είπε ότι πριν κάποιος προσπαθήσει να τις χρησιμοποιήσει «θα πρέπει να πάρει αρκετά διπλώματα πάνω στα ηλεκτρονικά και στην computer science». Αυτό μάλλον μεγαλοποιούσε το βαθμό ικανότητας που απαιτείται πριν μπορέσετε να χρησιμοποιήσετε μερικές από τις πληροφορίες που δίνονται εκεί. Αν έχετε κατορθώσει να φτάσετε μέχρι εδώ, και υποθέτουμε ότι μπορείτε να καταλάβετε την ενέργεια και τη λειτουργία των εντολών που έχουν εξηγηθεί, θα βρείτε εκεί μέσα ένα τεράστιο πλούτο από δεδομένα που μπορούν να χρησιμοποιηθούν.

Interrupts (διακοπές)

Τα interrupts (διακοπές) είναι τα μέσα με τα οποία ο Amstrad μπορεί να εκτελέσει εντολές της BASIC σαν την EVERY και την AFTER. Δημιουργούνται από τον Amstrad κατά κανονικά διαστήματα. Κάθε φορά που γίνεται ένα interrupt η Z80 CPU αντιδρά με έναν από τρεις τρόπους. Οι τρόποι αυτοί ονομάζονται Interrupt Modes που συντομεύεται σε IM για να σχηματίσει τη συμβολική εντολή. Τα interrupt modes μπορούν να χρησιμοποιηθούν σε οποιοδήποτε από τα τρία modes που είναι διαθέσιμα σε ένα πρόγραμμα αλλά εξαιτίας πάλι του σχεδιασμού του Amstrad μόνο με το ένα mode αξίζει να ασχοληθούμε. Οι εντολές καθορισμού του interrupt mode δίνονται στο παράρτημα.

Η διαδικασία του «ψυχρού ξεκινήματος» (cold start) στον Amstrad, που καθαρίζει τη μνήμη και τον φέρνει στην κατάσταση του «πρωϊνού ξυπνήματος» (early-morning wake-up), δίνει επίσης αρχική τιμή 1 στο interrupt mode (IM1). Σ' αυτό το mode οποιοδήποτε interrupt προκαλεί ένα ειδικό CALL στη διεύθυνση 56(38h). Το πρόγραμμα που αρχίζει εδώ είναι γνωστό σαν Interrupt Service Routine (Υπορουτίνα Εξυπηρέτησης των Interrupts) και εφ' όσον μπορεί να κληθεί σε ενέργεια από οποιοδήποτε σημείο του κυρίως προγράμματος, μια ρουτίνα εξυπηρέτησης των interrupts πρέπει να αποθηκεύει τους καταχωρητές που πρόκειται να χρησιμοποιήσει πριν τους τροποποιήσει, και να τους αποκαθιστά πριν την επιστροφή στο κυρίως πρόγραμμα.

Το ειδικό CALL που δημιουργείται από ένα interrupt, σταματάει αυτόματα την αναγνώριση άλλων interrupts, οπότε πριν την επιστροφή από μια υπορουτίνα εξυπηρέτησης των interrupts για να αναλάβει την εκτέλεση το κυρίως πρόγραμμα πρέπει να ενεργοποιηθούν τα interrupts, ώστε να μην αγνοούνται τα μελλοντικά interrupts. Η εντολή που επιτρέπει την ενεργοποίηση των interrupts ονομάζεται Enable Interrupts, ενώ υπάρχει επίσης μια ανάλογη εντολή για την αποενεργοποίηση των Interrupts (Disable Interrupts).

ASSEMBLER	HEX	BINARY
DI	F3	11 110 011
EI	FB	11 111 011

Η Locomotive software σκέφτηκε ευτυχώς αυτούς που προγραμματίζουν σε γλώσσα μηχανής και έδωσε ένα μέσο με το οποίο μπορούν να χρησιμοποιηθούν εύκολα τα interrupts. Στις περισσότερες μηχανές που βασίζονται στον Z80 πρέπει να χρησιμοποιείτε το IM2 για να έχετε πρόσβαση στα interrupts, και όπως θα διαπιστώσετε συνεχίζοντας το διάβασμα, αυτό δεν είναι από τα ευκολότερα πράγματα που μπορείτε να κάνετε. Στον Amstrad, αφού γράψετε την υπορουτίνα εξυπηρέτησης των interrupts, το μόνο που έχετε να κάνετε είναι να προσθέσετε στο τέλος ένα JP#B939, όπου κανονικά θα έπρεπε να βάλετε την εντολή RET. Για να ενεργοποιηθεί η ρουτίνα των interrupts φορτώστε το ζεύγος καταχωρητών HL με την αρχική διεύθυνση της υπορουτίνας σας, και κατόπιν LD (#39), HL. Από τώρα και στο εξής το κάθε interrupt θα καλεί την υπορουτίνα σας. Για να αποενεργοποιηθεί ή υπορουτίνα των interrupts θα πρέπει να χρησιμοποιηθούν οι εντολές:

ASSEMBLER	HEX
LD HL, #B939	21 39 B9
LD (#39), HL	22 39 00

Μην προσπαθήσετε να αλλάξετε τη διεύθυνση στο 39h σε δύο βήματα, γιατί είναι πιθανό να συμβεί ένα interrupt στο ενδιάμεσο της αλλαγής, κάνοντας το interrupt να καλέσει τη λαθεμένη διεύθυνση.

Παρακάτω δίνεται μια σύντομη περιγραφή του IM2 που θα σας εξυπηρετήσει πολύ μελλοντικά. Μην προσπαθήσετε να χρησιμοποιήσετε αυτό το mode, ή το IMO χωρίς να δείτε και να καταλάβετε τις λεπτομέρειες που δίνονται στο Programmer's Reference Manual για τη χρήση των interrupts. Τα διπλώματα θα σας βοηθήσουν στην προσπάθειά σας να ασχοληθείτε με αυτό το τμήμα!

Με το IM2 μπορείτε να χρησιμοποιείτε τα interrupts για τους δικούς σας σκοπούς, υπό την προϋπόθεση ότι τελειώνετε την υπορουτίνα εξυπηρέτησης των interrupts ενεργοποιώντας πάλι τα interrupts πριν επιστρέψετε στο πρόγραμμα που έκανε την κλήση και τελειώσετε με μια εντολή RETI.

Να θυμάστε ότι πρέπει να επιστρέψετε στο mode IM1 και να ενεργοποιήσετε τα interrupts πριν γυρίσετε στη BASIC εκτός αν χρησιμοποιείτε μέσα στην interrupt υπορουτίνα ένα RST 56 (38H).

Το mode IM2 είναι κάπως μπερδεμένο και λειτουργεί ως εξής: Μόλις η CPU

δεχτεί ένα interrupt αποθηκεύει στη στοίβα μηχανής τη διεύθυνση της επόμενης εντολής του προγράμματος που εκτελεί, και αποενεργοποιεί τα υπόλοιπα interrupts. Κατόπιν κοιτάζει στη θέση που δείχνει ο δίαυλος δεδομένων + (256*τον καταχωρητή A) και «πηδάει» στη διεύθυνση που περιέχεται σ' αυτή τη θέση+(256*τα περιεχόμενα της επόμενης θέσης). Για παράδειγμα ο καταχωρητής I περιέχει το 10 (OAH) και ο μηχανισμός διακοπής τοποθετεί το 200 στο δίαυλο δεδομένων όταν κάνει τη διακοπή.

$$10*256=2560 \quad 2560+200=2760$$

Επομένως η διεύθυνση στην οποία θα γίνει το «πήδημα» θα ληφθεί από τα περιεχόμενα της διεύθυνσης 2760 + (256*τα περιεχόμενα της διεύθυνσης 2816). Αν η 2760 περιείχε το 90 και η 2761 περιείχε το 187 τότε η διεύθυνση όπου θα γινόταν το «πήδημα» θα ήταν 90+(256*187) που ισούται με 47962. Ή αν ο καταχωρητής I ήταν 187 και ο μηχανισμός διακοπής έδινε 90

$$187*256=47872. \quad 47872+90=47962$$

η 47962 περιέχει το 207 και η 47963 περιέχει το 0

$$0+(207*256)=52992.$$

Επομένως το «πήδημα» θα γίνει στην 52992.

Αυτό μπορεί να αναπαρασταθεί αν φανταστούμε το interrupt σαν μια αόρατη εντολή στο πρόγραμμα που «τρέχει». Τη στιγμή που γίνεται το interrupt εκτελείται η αόρατη εντολή σαν να ήταν ένα DI ακολουθούμενο από μια εντολή CALL στη διεύθυνση που βρίσκεται αμέσως πριν τη διεύθυνση που δείχνει ο καταχωρητής I και ο δίαυλος δεδομένων. Η διεύθυνση που καλείται βρίσκεται στα δύο επόμενα bytes με την τυπική σειρά του Z80, το λιγότερο σημαντικό byte πρώτο. Η εντολή, αόρατη καθώς είναι, δεν μπορεί να τοποθετήσει τη δική της διεύθυνση επιστροφής στη στοίβα μηχανής, επομένως πηγαίνει στη στοίβα η διεύθυνση που βρίσκεται μετά την τελευταία εντολή που εκτελέστηκε στο πρόγραμμά σας, και αυτή είναι η διεύθυνση στην οποία θα επιστρέψει μετά την εντολή RETI στο τέλος της υπορουτίνας εξυπηρέτησης των interrupts.

Η εντολή RETI πρέπει να ακολουθεί μια εντολή EI. Ο λόγος για τον οποίο το DI συγχωνεύεται στο CALL που ενεργεί το interrupt είναι για να εξασφαλιστεί ότι, όταν η υπορουτίνα εξυπηρέτησης των interrupts χρησιμοποιηθεί για περισσότερο χρόνο από την καθυστέρηση μεταξύ δύο interrupts, το πρόγραμμα δεν θα εγκλωβιστεί σε βρόχο.

Είναι πολύ εύκολο να γράψετε ένα πρόγραμμα που αλλάζει τη διεύθυνση στην οποία γίνεται το «πήδημα» από ένα interrupt φορτώνοντας τα κατευθυντήρια bytes (τις δύο διευθύνσεις που καθορίζουν που θα γίνει το «πήδημα») με τις επιθυμητές διευθύνσεις μέσα στο πρόγραμμα.

Κάθε φορά που χρησιμοποιούνται interrupt υπορουτίνες είναι ζωτικής σημασίας ή φύλαξη κατά την εισαγωγή των καταχωρητών που χρησιμοποιούνται, και η αποκατάστασή τους πριν την επιστροφή στο κυρίως πρόγραμμα, καθώς και το

να μη γίνει απόπειρα μεταφοράς δεδομένων στους καταχωρητές προς και από την interrupt υπορουτίνα.

Τυπικές χρήσεις των interrupt υπορουτινών είναι για τον έλεγχο των sprites καθώς και για τον συνεχή έλεγχο των πλήκτρων που πιέζονται σε ένα πρόγραμμα. Αν είναι γνωστή η συχνότητα δημιουργίας ενός interrupt είναι εύκολο να υπολογιστεί η ταχύτητα μετακίνησης ενός sprite, και εφ' όσον θα είναι ανεξάρτητο από οποιαδήποτε άλλη λειτουργία μέσα στο πρόγραμμα, η ταχύτητα συνήθως παραμένει σταθερή.

Το Σχήμα 14.1 παρουσιάζει ένα σύντομο πρόγραμμα το οποίο, όταν εκτελείται από το INIT, θα ρυθμίζει το τυπικό interrupt να δείχνει στη label START. Μετά από κάθε interrupt θα εκτελείται η υπορουτίνα μεταξύ START και FINISH. Αυτό απλώς τοποθετεί στη διεύθυνση μνήμης 31100 το 123. Για να επαναφέρετε το interrupt στην αρχική του διεύθυνση καλέστε (CALL) την υπορουτίνα με τη label DISARM.

Επιστρέψτε στη BASIC αν χρησιμοποιούσατε τον assembler, και πριν κάνετε χρήση της γλώσσας μηχανής, δοκιμάστε τις παρακάτω άμεσες εντολές (direct commands).

```
? PEEK (31100)
```

πρέπει να δώσει 0

```
POKE 31100,10:?PEEK(31100)
```

πρέπει να δώσει 10 και

```
POKE 31100,0:?PEEK (31100)
```

πρέπει να δώσει πάλι 0. Γράψτε τώρα

```
CALL 30000
```

και δοκιμάστε πάλι τα παραπάνω.

Αν η interrupt υπορουτίνα εκτελείται σωστά θα διαπιστώσετε ότι ανεξάρτητα από το τι κάνετε η ?PEEK(31100) θα σας δίνει 123 γιατί η interrupt υπορουτίνα δίνει αυτή την τιμή σε κάθε interrupt.

Τώρα γράψτε CALL 30007 για να αποενεργοποιήσετε την interrupt υπορουτίνα σας και δοκιμάστε πάλι τα παραπάνω. Όλα θα πρέπει να έχουν επιστρέψει στην κανονική τους κατάσταση.

Ένα τελικό σημείο για τα interrupts: οποιοδήποτε πρόγραμμα σε γλώσσα μηχανής που μπορεί να τρέχει χωρίς interrupts θα τρέχει γρηγορότερα. Η αποενεργ-

Hisoft GENA3 Assembler. Page 1.

Pass 1 errors: 00

1 ; FIG 14,1 DIVERTING THE INTERRUPT

```

7530          10      ORG  30000
7530          20      ENT  30000
7530 211879    30  INIT  LD   HL,31000
7533 223900    40      LD   (#39),HL
7536  C9       50      RET
7537 2139B9    60  DISARM LD   HL,#B939
753A 223900    70      LD   (#39),HL
753D  C9       80      RET
7918          90      ORG  31000
7918  F5       100     PUSH AF
7919  3E7B     110     LD   A,123
791B  327C79   120     LD   (31100),A
791E  F1       130     POP  AF
791F  C339B9   140     JP   #B939

```

Pass 2 errors: 00

Table used: 37 from 142
Executes: 30000

THE CHECK-SUMS REQUIRED BY THE HEX LOADER ARE
02E9 0124
and 057B for the second part

Σχήμα 14.1

Hisoft GENA3 Assembler. Page 1.

Pass 1 errors: 00

```

          10 ; FIG 14,2 DELAY USING HALT
7530          20      ORG  30000
7530          30      ENT  30000
7530  FB       40      EI
7531  06C8     50      LD   B,200
7533  76       60  LOOP  HALT
7534  10FD     70      DJNZ LOOP
7536  C9       80      RET

```

Pass 2 errors: 00

Table used: 24 from 136
Executes: 30000

THE CHECK-SUMS REQUIRED BY THE HEX LOADER ARE
0415

Σχήμα 14.2

ποίηση των interrupts θα αποενεργοποιήσει επίσης τις συναρτήσεις για τη μέτρη-
ση του χρόνου στη BASIC καθώς και τις εντολές AFTER και EVERY, ενώ η ε-

πίδραση σε άλλα πράγματα θα είναι ελάχιστη.

HALT:

ASSEMBLER HEX BINARY

HALT 76 01 110 110

(Παρατηρήστε πως αυτός είναι ο κωδικός που έλειπε από την ομάδα εντολών LD r,r).

Αυτή η εντολή φαίνεται να είναι η κατάλληλη για να εξηγηθεί σύμφωνα με την τελευταία παράγραφο. Η εντολή HALT σταματάει τη CPU από ο,τιδήποτε έκανε μέχρι να ληφθεί το επόμενο interrupt. Αν εκτελεστεί όταν τα interrupts είναι αποενεργοποιημένα ο υπολογιστής θα πέσει να κοιμηθεί, γι' αυτό βεβαιωθείτε ότι τα interrupts είναι *σίγουρα* ενεργοποιημένα όταν χρησιμοποιείται η εντολή HALT.

Η συνηθέστερη χρήση της HALT είναι για να επιτρέπει την πραγματοποίηση μεγάλων χρονικών καθυστερήσεων χωρίς τη χρησιμοποίηση πολλαπλών βρόχων στο πρόγραμμα της καθυστέρησης. Το Σχήμα 14.2 παρουσιάζει ένα πρόγραμμα που χρησιμοποιεί την εντολή HALT γι' αυτόν τον σκοπό.

Μπορείτε να διαπιστώσετε τη χρονική διαφορά της καθυστέρησης που πραγματοποιείται από την εντολή HALT αν γράψετε POKE & 7533,0, που αντικαθιστά την HALT με ένα NOP, και τρέξετε πάλι το πρόγραμμα.

RST

Η CPU διαθέτει οχτώ διευθύνσεις που μπορούν να κληθούν από μια ειδική εντολή του ενός byte, γνωστή σαν RST (ReStArt). Ο Amstradd χρησιμοποιεί τις περισσότερες από αυτές για τους δικούς του σκοπούς, οπότε δεν έχουν πρακτική χρησιμότητα για σας, τον προγραμματιστή. Έχει μείνει, όμως, μια RST στη διάθεσή σας. Είναι η RST 30h.

Η διεύθυνση που ακολουθεί την RST είναι η διεύθυνση που καλείται όταν εκτελείται η εντολή. Ενεργεί κατά τον ίδιο ακριβώς τρόπο που ενεργεί ένα συνηθισμένο CALL και η ρουτίνα που καλείται από το RST πρέπει να τελειώνει με ένα RET, ακριβώς όπως θα γινόταν αν είχε χρησιμοποιηθεί ένα CALL.

ASSEMBLER	BINARY	p	t	p	t
RST p	11 t 111	00h	000	20h	100
		08h	001	28h	101
RST 30h	11 110 111	10h	010	30h	110
		18h	011	38h	111

Θα έχετε προσέξει ότι το τελευταίο ReStArt είναι η διεύθυνση που χρησιμοποιήθηκε από την interrupt υπορουτίνα, ενώ η μέθοδος που χρησιμοποιήθηκε για να υπάρξει πρόσβαση στο ReStArt του χρήστη είναι σχεδόν πανομοιότυπη με εκείνη που αλλάζει την κατεύθυνση του interrupt.

Η υπορουτίνα ψυχρού ξεκινήματος (Cold Start routine) κάνει την RST 30h να

«πηδάει» πίσω στην cold start ρουτίνα όταν χρησιμοποιείται. Αν είσαστε γενναίοι μπορείτε να το ελέγξετε καλώντας την διεύθυνση 56(38h). Θα προξενήσει στο σύστημα ένα reset.

Για να το αλλάξετε ώστε να «πηδάει» στην υπορουτίνα σας πρέπει να βάλετε μια εντολή jump στη διεύθυνση 30h που θα καλεί την υπορουτίνα σας. Θα μπορούσατε, για παράδειγμα, να αποφασίσετε ότι καλείται την υπορουτίνα PRINT στη διεύθυνση 47962 (BB5A1.) τόσο συχνά, ώστε αξίζει τον κόπο να χρησιμοποιείτε την RST αντί για την CALL, εξοικονομώντας έτσι 2 bytes κάθε φορά. Για να το κάνετε αυτό θα πρέπει να τοποθετήσετε το κωδικό του JP στη διεύθυνση 30h, και τη διεύθυνση στην οποία θα γίνει το «πήδημα» στις διευθύνσεις 31h και 32h, με τον συνηθισμένο για τον Z80 τρόπο όπου το λιγότερο σημαντικό byte είναι πρώτο. Το Σχήμα 14.3 δείχνει αυτή τη διαδικασία. Προσέξτε πως το RET στη γραμμή 51 αποτελεί τμήμα ενός σχολίου και όχι μιας εντολής.

Hisoft GENA3 Assembler. Page 1.

Pass 1 errors: 00

```

10 ; FIG 14,3 DIVERTING RST 30
7530      20      ORG   30000
7530      30      ENT   30000
7530  3EC3      40      LD   A,#C3      ; the code
                                   ; for jump
7532  215ABB      50      LD   HL,#BB5A ; the address
                                   ; of the routine
                                   ; to call
7535      60      ; RET would normally come here
7535  323000      60      LD   (#30),A ; The remainder
                                   ; of this is
                                   ; just for
7538  223100      70      LD   (#31),HL ; demonstration
                                   ; purposes.
753B  3E4B      80      LD   A,72
753D  F7        90      RST   #30
753E  3E65     100      LD   A,101
7540  F7       110      RST   #30
7541  3E6C     120      LD   A,108
7543  F7       130      RST   #30
7544  3E6C     140      LD   A,108
7546  F7       150      RST   #30
7547  3E6F     160      LD   A,111
7549  F7       170      RST   #30
754A  C9       180      RET

```

Pass 2 errors: 00

Table used: 13 from 160
Executes: 30000

THE CHECK-SUMS REQUIRED BY THE HEX LOADER ARE
02EC 04BB 040E

Σχήμα 14.3

Καμιά από τις μέχρι τώρα εντολές αυτού του κεφαλαίου, δεν επηρεάζει τα flags με οποιοδήποτε τρόπο.

Indexed Addressing

Υπάρχουν δύο καταχωρητές για τους οποίους δεν έχει αναφερθεί τίποτα μέχρι τώρα. Πρόκειται για τους καταχωρητές IX και IY. Το I στο κάθε όνομα αντιπροσωπεύει τη λέξη Index (Δείκτης). Είναι επομένως ο Δείκτης X και ο Δείκτης Y και, όπως και με οποιοδήποτε συνηθισμένο δείκτη, χρησιμοποιούνται για να λένε που μπορεί να βρεθεί ένα συγκεκριμένο τμήμα πληροφορίας. Εκτός από τη χρησιμότητά τους σαν καταχωρητές - δείκτες (index registers) μπορούν επίσης να χρησιμοποιηθούν σε ο,τιδήποτε κάνει το ζεύγος καταχωρητών HL.

«Πώς», θα ρωτήσετε, «μπορεί ένας μονός καταχωρητής να χρησιμοποιηθεί στη θέση ενός ζεύγους καταχωρητών;». Η απάντηση είναι απλή, στην πραγματικότητα δεν είναι μονοί καταχωρητές αλλά δύο καταχωρητές των 8 bits που χρησιμοποιούνται μαζί. Οι κατασκευαστές της Z80 CPU δεν μπορούσαν να πάρουν αξιόπιστα αποτελέσματα από αυτά τα δύο ζεύγη καταχωρητών, όταν τα χρησιμοποιούσαν ξεχωριστά, έτσι δεν δημοσίευσαν τις εντολές για την ανεξάρτητη χρήση του καθένα καταχωρητή, γιατί ορισμένες ή όλες οι εντολές μπορεί να μην λειτουργούν.

Στον Amstrad που χρησιμοποιήθηκε για να γραφτεί αυτό το βιβλίο όλες οι εντολές μονού καταχωρητή που δοκιμάστηκαν στα ζεύγη καταχωρητών - δεικτών λειτούργησαν. Παρ' όλο που αυτές οι εντολές δεν έχουν δημοσιευτεί είναι εξαιρετικά εύκολο να ανακαλύψετε ποιές είναι, από τη στιγμή που γνωρίζετε πως σχηματίζονται όλες οι εντολές που χρησιμοποιούν αυτούς τους καταχωρητές. Όσοι από σας διαθέτετε assembler θά πέσετε στο ίδιο σχεδόν επίπεδο προγραμματισμού με αυτούς που δεν διαθέτουν, αν επιθυμείτε να χρησιμοποιήσετε ξεχωριστά τα δύο μισά των καταχωρητών-δεικτών. Εκτός από αυτό, δεν υπάρχει τρόπος που να μπορεί να εγυηθεί ότι ένα πρόγραμμα που χρησιμοποιεί αυτές τις πρόσθετες εντολές θα δουλέψει σε έναν άλλο Amstrad, οπότε πιθανώς να είναι καλλίτερα να μην τους απασχολούμε. Αφού το ξεκαθαρίσαμε αυτό μπορούμε να περιγράψουμε τη χρήση των καταχωρητών-δεικτών.

Οποιαδήποτε εντολή εκτός από τις ADC και SBC που χρησιμοποιεί το ζεύγος καταχωρητών HL μπορεί να διαμορφωθεί ώστε να ενεργεί στον καταχωρητή IX, τοποθετώντας απλώς το DDh (221) μπροστά από τον κωδικό λειτουργίας της εντολής για την HL ή, για να χρησιμοποιήσετε τον καταχωρητή IY, βάλτε το FDh(253) μπροστά από την εντολή για τον HL. Όταν η εντολή χρησιμοποιεί τον καταχωρητή - δείκτη για να δείχνει μια διεύθυνση της μνήμης χρειάζεται ένα ακόμα byte μετά το πρώτο byte της αρχικής εντολής. Αυτό το πρόσθετο byte δίνει μια προσημασμένη μετατόπιση από τη διεύθυνση που δείχνει ο καταχωρητής. Αυτό πιθανώς να φαίνεται λιγάκι μπερδεμένο. Πριν τα παρατήσετε δείτε τα παρακάτω παραδείγματα.

Η Hex εντολή για την LD HL, nn είναι 21 και ακολουθείται από δύο bytes που δίνουν τον αριθμό των 16 bits nn. Για να αλλάξετε αυτή την εντολή ώστε να την κάνετε να δουλέψει στον καταχωρητή IX θα πρέπει να βάλετε μπροστά της το DDh. Η εντολή έγινε τώρα:

ASSEMBLER HEX

LD IX, nn DD 21 n n

Για να δουλέψει στο ζεύγος καταχωρητών IY αντί για το IX, η εντολή είναι:

ASSEMBLER HEX

LD IY, nn FD 21 n n

Όλες οι εντολές που ενεργούν άμεσα στους καταχωρητές-δείκτες έχουν το ίδιο format. Παρακάτω δίνονται μερικά παραδείγματα.

ASSEMBLER	HEX	WITH IX	WITH IY
LD (nn), HL	22 n n	DD 22 n n	FD 22 n n
PUSH HL	E5	DD E5	FD E5
DEC HL	2B	DD 2B	FD 2B
JP (HL)	E9	DD E9	FD E9
ADD HL, BC	09	DD 09	FD 09

Αυτή η τελευταία εντολή τοποθετεί ένα ενδιαφέρον θέμα. Τι υποθέτετε ότι θα συμβεί όταν η εντολή: ADD HL, HL τροποποιηθεί για να ενεργήσει στον καταχωρητή IX ή IY; Αυτή ήταν η εντολή που θα μπορούσε να έχει χρησιμοποιηθεί για να εκτελέσει μια αριστερή μετατόπιση (shift left) των 16 bits στα προγράμματα πολλαπλασιασμού του Κεφαλαίου 11.

Αν η εντολή γινόταν ADD IY, HL με την τοποθέτηση του προθέματος FDh και ADD IX, HL με το DDh μπροστά δεν θα είχαμε μια άμεση αντικατάσταση (παρ' όλη τη χρησιμοποίηση ενός πρόσθετου byte) για την εντολή που χρησιμοποιεί τον HL. Στην πραγματικότητα οποιαδήποτε αναφορά στο ζεύγος καταχωρητών HL αλλάζει ώστε να αναφέρεται στον καταχωρητή - δείκτη όταν ο κωδικός λειτουργίας που χρησιμοποιεί το ζεύγος καταχωρητών HL έχει μπροστά του κάποιο από τα δύο προθέματα των καταχωρητών - δεικτών. Η ADD HL, HL γίνεται ADD IX, IX όταν έχει μπροστά της το DDh ενώ, με το πρόθεμα FDh γίνεται ADD IY, IY.

Μέχρι τώρα δεν έχει παρουσιαστεί καμία εντολή που χρησιμοποιεί τον HL για να δείχνει θέσεις μνήμης. Αυτό οφείλεται στην πρόσθετη περιπλοκή του indexed addressing (προσπέλασης με δείκτες). Όπως αναφέρθηκε προηγουμένως χρειάζεται ένα πρόσθετο byte για να δίνει την προσημασμένη μετατόπιση από την πραγματική διεύθυνση που δείχνει ο καταχωρητής.

Υποθέστε, για ένα λεπτό, ότι έχετε γράψει ένα πρόγραμμα Database, και η κάθε εγγραφή έχει ένα είδος επικεφαλίδας (header), που δίνει το μήκος του καθενός πεδίου στην εγγραφή. Ας πάρουμε ένα πολύ απλό παράδειγμα (που οι συγγραφείς βιβλίων και περιοδικών φαίνεται να πιστεύουν ότι το θέλουν όλοι οι κάτοχοι micros, αν και ένας θεός ξέρει γιατί!), το βιβλίο διευθύνσεων (Adress Book).

Δεν υπάρχει τρόπος για να πείτε ότι το μήκος του ονόματος ενός ατόμου, ο αριθ-

μός γραμμών στη διεύθυνση και το μήκος αυτών των γραμμών, ή ο αριθμός τηλεφώνου, θα είναι ίδια και για κάποιο άλλο άτομο. Αυτό μπορεί να γίνει με δύο βασικές μεθόδους.

- 1) Ορίζετε η κάθε εγγραφή να καταλαμβάνει τόσο χώρο ώστε να μπορεί να περιλαμβάνει το μακρύτερο όνομα και διεύθυνση
- 2) Καταγράφετε το μήκος κάθε γραμμής, και τον αριθμό των γραμμών, καθώς και το συνολικό μήκος της εγγραφής.

Η μέθοδος 1 είναι πολύ σπάταλη σε χώρο, η κάθε εγγραφή χρησιμοποιεί την ίδια ποσότητα μνήμης, ενώ η παρακολούθηση των στοιχείων της κάθε εγγραφής χρησιμοποιώντας τη μέθοδο 2 θα αποτελούσε πρόβλημα. Όχι όμως με τους καταχωρητές - δείκτες!

Αν αρχίσετε τις εγγραφές από μια γνωστή διεύθυνση, ας πούμε την 10000, θα μπορούσατε να έχετε ένα δείκτη στην αρχή της κάθε εγγραφής σαν τον παρακάτω.

ΔΙΕΥΘΥΝΣΗ

10000/1

10002

10003

10004

10005

10006

10007

10008

ΣΤΟΙΧΕΙΑ

μήκος αυτής της εγγραφής, σε 16 bits

μήκος του ονόματος

μήκος της διεύθυνσης στη γραμμή 1

μήκος της διεύθυνσης στη γραμμή 2

μήκος της διεύθυνσης στη γραμμή 3

μήκος της διεύθυνσης στη γραμμή 4

μήκος της διεύθυνσης στη γραμμή 5

μήκος του αριθμού τηλεφώνου

Αν η εγγραφή 1 ήταν

Martin Pudwick 14

27 New Road

Mudford

Sussex

0123 456789

10000= 49 το συνολικό μήκος της εγγραφής, στο λιγότερο σημαντικό byte του Z80

10001= 0 πρώτη μορφή

10002= 14 όνομα

10003= 11 διεύθυνση 1

10004= 7 διεύθυνση 2

10005= 6 διεύθυνση 3

10006= 0 διεύθυνση 4

10007= 0 διεύθυνση 5

10008= 11 αριθμός τηλεφώνου

Για τον δείκτη θα χρησιμοποιούνται πάντοτε εννέα bytes: έτσι $49+9=58$, η αρχή του δείκτη για την επόμενη εγγραφή θα είναι στην 10058. Τώρα μπορείτε να χρησιμοποιήσετε τους καταχωρητές δείκτες.

Αν ο IX φορτωθεί με 10000 τότε το συνολικό μήκος της εγγραφής θα είναι (IX + 0) και (IX+1), το μήκος του ονόματος θα είναι (IX+2), κ.ο.κ. Μπορείτε να γράψετε το πρόγραμμά σας που θα αυξάνει το συνολικό μήκος για κάθε χαρακτήρα που προστίθεται σε οποιαδήποτε γραμμή της εγγραφής, και θα αυξάνει επίσης το δείκτη της γραμμής στην οποία εργαζόσαστε. Κατόπιν, όταν θα θέλετε να μετακινηθείτε στην επόμενη εγγραφή, το μόνο που θα κάνετε θα είναι να προσθέσετε τα (IX+0) και (IX+1) στον καταχωρητή IX, για να έχετε στη διάθεσή σας τα στοιχεία αυτής της εγγραφής, έτοιμα να χρησιμοποιηθούν από το ίδιο πρόγραμμα, χωρίς μετατροπές.

Οι εντολές στον assembler, για τη χρησιμοποίηση των καταχωρητών-δεικτών με μετατόπιση, έχουν ακριβώς τη μορφή που θα περιμένατε. Αντί για LD A, (HL) η εντολή γίνεται LD A, (IX+d) όταν ο DDh τοποθετείται μπροστά από τον αρχικό κωδικό λειτουργίας, και LD A, (IY+d) όταν χρησιμοποιείται ο FDh. Το d είναι οποιαδήποτε τιμή από -128 μέχρι +127.

Η μετατόπιση είναι υποχρεωτική για όλες τις εντολές που χρησιμοποιούν καταχωρητές - δείκτες σε διευθύνσεις μνήμης, ακόμη και όταν υπάρχει 0 μετατόπιση, και το byte που περιέχει τη μετατόπιση βρίσκεται πάντοτε αμέσως μετά το πρώτο byte του αρχικού κωδικού λειτουργίας. Για παράδειγμα:

H LD A, (HL) είναι 7Eh. H LD A, (IX+d) είναι DDh 7Eh d
H INC (HL) είναι 34h. H INC (IY+d) είναι FDh 34h d

Το ίδιο ισχύει και όταν περιλαμβάνεται αριθμός:

H LD (HL), n είναι 36h n και η LD(IX+d),n είναι DDh36h d n

Μια ακόμη χρήση των καταχωρητών - δεικτών είναι για την αλλαγή του άξονα της οθόνης, ώστε να μπορεί να γραφτεί στον εκτυπωτή ένα αντίγραφο της οθόνης. Οι εκτυπωτές χρειάζονται ένα byte που να περιέχει τις πληροφορίες κάθετα, ενώ η οθόνη χρησιμοποιεί το byte οριζόντια. Στο mode 2 ένα byte περιέχει πληροφορίες για οχτώ pixels από την ίδια θέση του άξονα των Y αλλά από οκτώ διαδοχικές θέσεις στον άξονα των X. Η οθόνη θα πρέπει να περιστραφεί κατά 90 μοίρες πριν αντιγραφεί κατ' ευθείαν στον εκτυπωτή.

Ο οικονομικότερος τρόπος για να το κάνουμε αυτό, είναι να μην γίνει στην πραγματικότητα η περιστροφή της οθόνης, αλλά να φυλάξουμε ένα χώρο στη μνήμη και να περιστρέφουμε τα bytes με τη σειρά, μέχρι να μπορεί να μεταβιβαστεί στον εκτυπωτή μια γραμμή χαρακτήρων της οθόνης, και η διαδικασία να επαναλαμβάνεται για την επόμενη γραμμή χαρακτήρων. Το Σχήμα 14.4 δείχνει να γίνεται αυτό για τη γραμμή της οθόνης που αρχίζει στη C000h, στο mode 2. Επειδή δεν έχουν όλοι εκτυπωτή, και ακόμη κι αν είχαν, πολλοί εκτυπωτές χρειάζονται διαφορετικούς κωδικούς ελέγχου (control codes) για να βρεθούν στο graphics mode, υποθέτοντας πάντα ότι μπορούν να χρησιμοποιηθούν για την εκτύπωση γραφικών, το πρόγραμμα αυτό δουλεύει πάνω στην οθόνη.

Ο χάρτης της οθόνης (screen map) μετακινείται κάθε φορά που ρολάρει η οθόνη, οπότε για να είστε βέβαιοι ότι αρχίζει από την C000h, πατείστε το W μέχρι να βρεθείτε στο mode 2, αν χρησιμοποιείτε τον assembler, ή πηγαίνετε στο mode 2 με άμεση εντολή (direct command) αν χρησιμοποιείτε το HEX LOADER. Μην κάνετε list για να πάρετε κάτι στην οθόνη αλλά μετακινείστε τον δρομέα στην ε-

πάνω γραμμή και γράψτε εκεί μερικούς χαρακτήρες. Κατόπιν πάλι χωρίς να κάνετε την οθόνη να ρολάρει, πατείστε [ENTER]. Θα πάρετε ένα μήνυμα λάθους, αλλά μην ανησυχείτε. Τρέξτε τώρα το πρόγραμμα. Θα διαπιστώσετε ότι στη γραμμή 2 παρουσιάζεται ένα αντίγραφο της πρώτης γραμμής χαρακτήρων, αλλά ο κάθε χαρακτήρας έχει περιστραφεί κατά 90 μοίρες προς τη φορά των δεικτών του ρολογιού.

Αυτό το πρόγραμμα μεταφέρεται άνετα οπότε μπορείτε να το ενσωματώσετε σε ένα δικό σας πρόγραμμα, αν δημιουργείτε αντίγραφα οθόνης ή κάποιο παρόμοιο πρόγραμμα. Παρατηρείστε πως τα αντίγραφα της οθόνης είναι εύκολα στο mode 2 ενώ στα άλλα modes μας φέρνουν δυσκολίες, μιας και ένα bit δεν αντιπροσωπεύει ένα pixel.

Εδώ περίπου τελειώνει το βιβλίο, και το μόνο που σας μένει είναι να κάνετε πρακτική εξάσκηση πάνω σ' αυτά που μάθατε. Το επόμενο κεφάλαιο θα σας δώσει μερικές ιδέες, και τις διευθύνσεις που μπορείτε να καλέσετε (CALL) για να χρησιμοποιήσετε ορισμένες από τις υπορουτίνες του λειτουργικού συστήματος. Τα παραρτήματα υπάρχουν για να σας βοηθήσουν στις υπορουτίνες που γράφετε, γλυτώνοντας σας από το ψάξιμο για μια απλή απάντηση σε ένα ερώτημα.

Αν πρόκειται να ασχοληθείτε σοβαρά με τον προγραμματισμό θα σας βοηθήσει η αγορά ενός stencil για διαγράμματα ροής που δεν χρειάζεται να είναι από τα ακριβά. Ανεκτίμητη σχεδόν βοήθεια θα σας προσφέρει μια αριθμομηχανή που μπορεί να δουλέψει στο Hex και στο δυαδικό καθώς και στο συνηθισμένο δεκαδικό σύστημα. Η Casio fx450 είναι πολύ καλή (και η τιμή της είναι η μισή της Texas) και κοστίζει στην Μ. Βρετανία περίπου είκοσι λίρες. Αυτή η αριθμομηχανή εκτελεί επίσης τις λογικές πράξεις AND OR XOR καθώς και τις NEG και CPL (που την ονομάζουν NOT).

Hisoft GENA3 Assembler. Page 1.

Pass 1 errors: 00

```

          1 ; FIG 14,4 ROTATING THE SCREEN
          2 ; MODE 2
9C40      10      ORG  40000
9C40      20      ENT  40000
9C40      DD2100C0 30      LD   IX, #C000
9C44      2150C0   40      LD   HL, #C050
9C47      110008   50      LD   DE, #800
9C4A      0650     60      LD   B, 80
9C4C      DDE5     70 LOOP3  PUSH IX
9C4E      E5       80      PUSH HL
9C4F      C5       90      PUSH BC
9C50      0608     100     LD   B, 8
9C52      C5       110 LOOP2  PUSH BC
9C53      E5       120     PUSH HL
9C54      DDE5     130     PUSH IX
9C56      0608     140     LD   B, 8
9C58      DDCB0006 150 LOOP  RLC  (IX+00)
9C5C      CB1E     160     RR   (HL)
9C5E      19       170     ADD  HL, DE
9C5F      10F7     180     DJNZ LOOP
9C61      DDE1     190     POP  IX

```

9C63	E1	200	POP	HL
9C64	DD19	210	ADD	IX, DE
9C66	C1	220	POP	BC
9C67	10E9	230	DJNZ	LOOP2
9C69	C1	240	POP	BC
9C6A	E1	250	POP	HL
9C6B	DDE1	260	POP	IX
9C6D	DD23	270	INC	IX
9C6F	23	280	INC	HL
9C70	10DA	290	DJNZ	LOOP3
9C72	C9	300	RET	

Pass 2 errors: 00

Table used: 48 from 147

THE CHECK-SUMS REQUIRED BY THE HEX LOADER ARE

0308 057A 0467 0586 0656 00C9

Σχήμα 14.4

ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΕΣ ΙΔΕΕΣ ΚΑΙ ΧΡΗΣΙΜΟΠΟΙΗΣΗ ΤΟΥ FIRMWARE

Το λειτουργικό σύστημα του Amstrad μπορεί να χωριστεί σε εννέα βασικές περιοχές. Μερικές υπορουτίνες από κάθε μια από αυτές τις περιοχές θα σας δώσουν μια καλή αρχή για τον προγραμματισμό σας. Έχετε ήδη γνωρίσει δύο υπορουτίνες, την «text output» («εκτύπωση κειμένου») και την «wait key» («ανάμονη πλήκτρου»), που έχουν αναφερθεί σαν «GETKEY» και «PRINT» αντίστοιχα. Η «text output» είναι τμήμα του Screen Manager, και η «wait key» είναι μια από τις υπορουτίνες του Key Manager. Οι υπόλοιπες 7 περιοχές είναι:

- Graphics VDU
- Screen Pack
- Cassette Manager
- Sound Manager
- Kernel
- Machine Pack
- Jumper

Το βιβλίο αυτό δεν έχει σαν σκοπό του να μπει σε λεπτομέρειες πάνω στην κατασκευή αυτών των τμημάτων, ούτε να περιγράψει το λειτουργικό σύστημα. Το Programmer's Reference Manual τα καλύπτει αυτά λεπτομερώς αλλά σαν αρχή σας προσφέρουμε μερικά στοιχεία που μπορεί να τα βρείτε χρήσιμα.

Η προσπέλαση στο λειτουργικό σύστημα γίνεται, για το μεγαλύτερο τμήμα του, από τις διευθύνσεις που καλεί ο προγραμματιστής οι οποίες βρίσκονται στη RAM (Random Access Memory, Μνήμη Τυχαίας Προσπέλασης), γνωστές σαν «Jump-blocks». Ένα jump-block συχνά αποτελείται από μια απλή σειρά διευθύνσεων, έτοιμων να φορτωθούν στο ζεύγος καταχωρητών HL, πριν από μια εντολή JP (HL). Αυτό φαίνεται στο Σχήμα 15.1, μιας και είναι μια χρήσιμη τεχνική που μπορεί να επιθυμείτε να την χρησιμοποιήσετε στα προγράμματά σας.

.....

MAIN PROGRAM

LD A, ROUTNO ; ο αριθμός της ρουτίνας που θα κληθεί (0-255)

CALL JUMP

REST OF MAIN PROGRAM

.....

```

JUMP:  ADD A, A
         LD  D, 0
         LD  E, A
         LD  HL, JBSTRT
         ADD HL, DE ; τώρα ο HL περιέχει τη διεύθυνση μνήμης με τη διεύθυνση που
                   θα κληθεί
         LD  E, (HL)
         INC HL
         LD  D, (HL)
         EX  DE, HL
         JP  (HL) ; «πηδάει» στην υπορουτίνα, και χρησιμοποιεί το RET στο τέλος
                   της για να επιστρέψει στο κυρίως πρόγραμμα

JBSTRT: ; εδώ αρχίζουν τα ζεύγη των bytes που περιέχουν τις διευθύνσεις της υπορουτί
           νας σύμφωνα με τη μέθοδο του Z80 με το σημαντικότερο και λιγότερο ση
           μαντικό byte (Z80|high byte, low byte fashion).

```

Σχήμα 15.1

Ένας λόγος για να χρησιμοποιείτε τα jump-blocks είναι γιατί μπορείτε να αλλάξετε τον τρόπο με τον οποίο χρησιμοποιείται μια συγκεκριμένη υπορουτίνα, αλλάζοντας απλώς τη διεύθυνση στο jump-block, αντί να διατρέχετε όλο το πρόγραμμα και να αλλάζετε τις εντολές CALL προς τη σχετική υπορουτίνα. Έχουν επίσης το πλεονέκτημα ότι επιτρέπουν σε ένα πρόγραμμα να υπολογίζει τις δικές του ενέργειες.

Τυπική περίπτωση όπου το jump-block πλεονεκτεί είναι όταν ένα πρόγραμμα πρόκειται να χρησιμοποιήσει τον εκτυπωτή, ή κάποιο διαφορετικό «παράθυρο» (window), με την έξοδο των δεδομένων, από εκείνο που είχε επιλεγεί στην αρχή. Αυτό σας επιτρέπει να κάνετε εκσφαλμάτωση (debugging) και να «τρέξετε» το πρόγραμμα στην οθόνη, ενώ ο εκτυπωτής χρησιμοποιείται μόνον όταν χρειάζεται. Στη BASIC θα μπορούσατε να χρησιμοποιείτε την εντολή PRINT ακολουθούμενη από μια μεταβλητή, στην οποία θα δίνετε τον αριθμό του περιφερειακού που επιθυμείτε να χρησιμοποιήσετε εκείνη τη στιγμή. Σε ένα πρόγραμμα γλώσσας μηχανής δεν υπάρχει λειτουργικό σύστημα για την παρακολούθηση των μεταβλητών, ώστε να ανταποκρίνεται σφαιρικά όταν αλλάζει η τιμή τους, γι' αυτό υπάρχει το jump-block.

Ο Amstrad σας δεν μπορεί να χρησιμοποιήσει το παραπάνω σύστημα, γιατί έχει τη δυνατότητα να ανταλλάσει περιοχές μνήμης των ROM και RAM. Μόνον αν ο

προγραμματιστής ελέγξει πραγματικά ότι χρησιμοποιείται η ROM που περιέχει την υπορουτίνα που θα κληθεί, κι αν όχι να κάνει τις απαραίτητες ενέργειες, μπορεί να υπάρξει εγγύηση για προσπέλαση στη σωστή υπορουτίνα! Ακόμη, αν πρέπει να διαβάζεται η θόνη, πρέπει να μην χρησιμοποιείται η ανώτερη ROM (Upper ROM), που καταλαμβάνει την ίδια διεύθυνση.

Η λύση της Amstrad είναι η αφιέρωση ενός αριθμού από ReStarts στην κλήση ROM υπορουτινών. Έτσι εξασφαλίζεται η χρησιμοποίηση της σωστής ROM και ότι η θόνη είναι πρόσφορη σε ανάγνωση αν χρειαστεί. Κάνουν POP τη διεύθυνση επιστροφής από την εντολή restart έξω από τη στοίβα και κατόπιν την χρησιμοποιούν για να κοιτάξουν στο bytes που ακολουθούν το restart, τα οποία δίνουν τη διεύθυνση της υπορουτίνας που θα κληθεί και τη ROM που θα χρησιμοποιηθεί. Οι πλήρεις λεπτομέρειες του τρόπου λειτουργίας αυτών των ReStarts δίνονται στο Programmer's Reference Manual, αλλά είναι απίθανο να χρειαστεί να γνωρίζετε περισσότερα για τον τρόπο λειτουργίας τους, πριν ασχοληθείτε σε βάθος με τον προγραμματισμό. Οι ενέργειές τους είναι τελείως διαφανείς στο χρήστη, και επιστρέφοντας στο πρόγραμμα σας η ROM επαναφέρεται στην αρχική της κατάσταση.

Οποιαδήποτε από τις πολλές διαθέσιμες υπορουτίνες του λειτουργικού συστήματος μπορεί να χρησιμοποιηθεί με την άμεση εκτέλεση ενός CALL σε κάποιο jump-block. Η διεύθυνση που επιστρέφεται από το return στο τέλος της ROM ρουτίνας, είναι αυτή που τοποθετείται εκεί από την κλήση σας στο jump-block. Δεν υπάρχει επιστροφή στο ReStart του jump-block.

Υπάρχουν δύο σαφώς διαφορετικά jump-blocks, ένα που χρησιμοποιείται από τον interpreter της BASIC, και το άλλο που χρησιμοποιείται από το firmware, ή το λειτουργικό σύστημα. Μπορείτε να εναλλάσσετε αυτές τις δύο ομάδες αλλάζοντας τα τρία bytes, αρχίζοντας από τη διεύθυνση RST. Αν η νέα υπορουτίνα που θα προσπελαθεί βρίσκεται στο firmware, αντιγράψτε από τον πίνακα των jumps τα τρία bytes της νέας υπορουτίνας που θα αντικαταστήσουν την παλαιότερη εγγραφή. Αν η νέα υπορουτίνα είναι δική σας αντικαταστήστε απλώς την εγγραφή στον πίνακα των jumps με ένα JP στη διεύθυνση της υπορουτίνας που θέλετε να κληθεί για να εκτελέσει την εργασία, και όσο συνεχίζετε να τελειώνετε αυτή την υπορουτίνα με ένα RET τα πράγματα θα εξακολουθήσουν να συμβαίνουν «σωστά». Η αλλαγή του κυρίως jump-block, που βρίσκεται μεταξύ των BB00h και BDCBh, δεν θα έχει επίδραση στη λειτουργία οποιασδήποτε υπορουτίνας του λειτουργικού συστήματος. Η αλλαγή εγγραφών σε jump-blocks έξω από αυτήν την περιοχή μπορεί να έχουν απρόβλεπτες παρενέργειες. Αυτό οφείλεται στο ότι οι υπορουτίνες μπορεί να χρησιμοποιούν αυτά τα άλλα jump-blocks για να καλούν άλλες υπορουτίνες, που χρειάζονται για να εκτελεστεί μια εργασία.

Αν (πότε;) βρεθείτε σε τέτοιο μπερδεμα που να μην καταλαβαίνετε ποιός καλεί ποιόν, πού, πώς, πότε και γιατί, υπάρχει μια εγγραφή στο jump-block που είναι ζωτικής σημασίας. Αυτή επιστρέφει σε όλα τα jumps τον αρχικό τους προορισμό! CALL BD37h.

Η θόνη και ο ήχος χειρίζονται από το hardware και όχι από το software, πράγμα που δημιουργεί δυσκολίες στην άμεση προσπέλαση των λειτουργιών που επιθυμείτε να χρησιμοποιήσετε, ενώ η ανίχνευση του πληκτρολογίου (keyboard) με software γίνεται σχεδόν αδύνατη. Ευτυχώς υπάρχουν στο firmware ενσωματωμέ-

νες ευκολίες (facilities) που είναι διαθέσιμες για την εκτέλεση αυτών των ενεργειών για σας. Τα joy-sticks μπορούν να διαβαστούν για οποιονδήποτε συνδυασμό διεύθυνσης και πίεσης του fire button, επιτρέποντας έτσι ανίχνευση της κίνησης προς οχτώ κατευθύνσεις. Ορισμένες από τις βασικές διευθύνσεις των jumps του firmware δίνονται στο παράρτημα.

Προσπαθείτε πάντοτε να χρησιμοποιείτε labels στα προγράμματά σας που να έχουν σχέση με την εργασία για την οποία χρησιμοποιούνται. Παρ' όλο που η εκτεταμένη χρήση των σχολίων δεν είναι μόνο ανεπιθύμητη, αλλά κοστίζει και σε μνήμη, να θυμάστε ότι μπορεί κάποτε στο μέλλον να επιστρέψετε στον πηγαίο κώδικα, χωρίς να έχετε κάποια ένδειξη για τον τρόπο οργάνωσης του προγράμματος σας.

Παρακαλούμε να προσέξετε ότι αυτό το βιβλίο δεν προσπάθησε καθόλου να σας διδάξει ειδικές λεπτομέρειες για τον τρόπο λειτουργίας Z80, ούτε εξετάστηκαν θέματα χρόνου. Ο σκοπός ήταν να εισαχθείτε στον προγραμματισμό, να καταλάβετε την ομάδα εντολών και να μπορέσετε να χρησιμοποιήσετε το firmware. Αν επιθυμείτε να ερευνήσετε βαθύτερα στις λεπτότερες αποχρώσεις του προγραμματισμού του Z80, ή να ερευνήσετε τη λειτουργία του hardware, μπορούμε να σας προτείνουμε τα βιβλία:

- 1) Για τον προγραμματιστή το «Programming the Z80» του Rodney Zaks.
- 2) Για τον σχεδιασμό και την επέκταση του hardware το «The Z80 Technical Manual» της Zilog και το «Z80 Applications» του James Coffron.

Τα βιβλία των Zaks και Coffron είναι εκδόσεις της Sybex, και δεν είναι φτηνά (το καθένα κοστίζει στη Μ. Βρετανία πάνω από 10 λίρες) γι' αυτό δεβαιωθείτε ότι είναι αυτά που πραγματικά χρειάζεστε πριν τα αγοράσετε.

Αφού γίνετε έμπειρος στον προγραμματισμό σε γλώσσα assembly μπορεί να θελήσετε να μάθετε και άλλες γλώσσες. Η Pascal είναι πιθανώς η καλύτερη «δεύτερη γλώσσα» που μπορείτε να μάθετε για σοβαρό προγραμματισμό, η οποία είναι σχεδόν σαν την BASIC από πολλές απόψεις, αλλά προσφέρει επίσης πολλές από τις ευκολίες ενός assembler. Πρόκειται για μια μεταφραζόμενη (compiled) γλώσσα σαν την assembly, και ένα πρόγραμμα γραμμένο σε Pascal συνήθως μεταφέρεται πολύ εύκολα με τη μορφή πηγαίου κώδικα. Πολλοί υπολογιστές διαθέτουν εκδόσεις γραμμένες γι' αυτούς, και ένα πηγαίο πρόγραμμα μπορεί να μεταφραστεί για να τρέξει σε οποιονδήποτε υπολογιστή με Pascal compiler κάνοντας ελάχιστες αλλαγές. Οι βασικοί περιορισμοί της Pascal είναι η ταχύτητα, η assembly υπερτερεί, και ο χώρος, όπου η assembly είναι πολύ οικονομικότερη. Η ίδια η Pascal είναι πολύ ταχύτερη από τη Basic και ο μεταφρασμένος κώδικας καταναλώνει λιγότερο χώρο από το αντίστοιχο πρόγραμμα BASIC. Πολλοί προγραμματιστές χρησιμοποιούν ένα συνδυασμό της Pascal και αληθινής γλώσσας μηχανής, μεταφρασμένης από τη γλώσσα assembly, όπου η ταχύτητα ή η οικονομία χώρου έχουν τον πρώτο λόγο.

Και η Pascal και η γλώσσα assembly, μετά τη μετάφρασή τους, μπορούν να τρέξουν χωρίς την παρουσία του προγράμματος που χρησιμοποιήθηκε για να γραφτούν, και ο πηγαίος κώδικας μπορεί να αποθηκευτεί για περαιτέρω χρήση αν χρειαστεί, ενώ μόνον ο αντικειμενικός κώδικας χρειάζεται να αποθηκευτεί και να φορτωθεί για να χρησιμοποιηθεί το πρόγραμμα. Υπάρχει μια έκδοση της Pascal που διατίθεται για τον Amstrad, γραμμένη από την Highsoft.

Η ΟΜΑΔΑ ΕΝΤΟΛΩΝ ΤΟΥ Ζ80 ΠΡΟΣΦΟΡΑ ΤΗΣ ΖΙΛΟΓ ΙΝC

Η ομάδα εντολών

Ο μικροεπεξεργαστής Ζ80 διαθέτει μια από τις ισχυρότερες και ευστροφότερες ομάδες εντολών στο χώρο των μικροεπεξεργαστών των 8 bits. Περιέχει μοναδικές λειτουργίες όπως είναι η μετακίνηση περιοχών της μνήμης για γρήγορη, αποτελεσματική μεταφορά δεδομένων μέσα στη μνήμη ή μεταξύ μνήμης και I/O. Επιτρέπει επίσης πράξεις σε οποιοδήποτε bit οποιασδήποτε θέσης μνήμης.

Τα παρακάτω αποτελούν μια περίληψη της ομάδας εντολών του Ζ80 και παρουσιάζουν τη συμβολική εντολή της γλώσσας assembly, την λειτουργία, την κατάσταση των flags, και δίνει σχόλια για την κάθε εντολή. Τα βιβλία «Ζ80 CPU Technical Manual» (03-0029-01) και «Assembly Language Programming Manual» (03-0002-01) περιέχουν σημαντικά περισσότερες λεπτομέρειες για προγραμματιστική χρήση.

Οι εντολές χωρίζονται στις ακόλουθες κατηγορίες:

- Φορτώσεις των 8-bits (8-bit loads)
- Φορτώσεις των 16-bits (16-bit loads)
- Ανταλλαγές (exchanges), μεταφορές περιοχών μνήμης (block transfers) και έρευνες (searches)
- Αριθμητικές και λογικές πράξεις των 8 bits
- Αριθμητική γενικής χρήσης και έλεγχος της CPU
- Αριθμητικές πράξεις των 16 bits
- Περιστροφές (rotates) και μετατοπίσεις (Shifts)
- Λειτουργίες ενεργοποίησης (set), αποενεργοποίησης (reset) και ελέγχου των bits
- Πηδήματα (jumps)
- Κλησεις (calls), returns και restarts
- Λειτουργίες εισόδου και εξόδου (input and output operations).

Εφαρμόζεται μια ποικιλία μεθόδων προσπέλασης (addressing modes) για την γρήγορη και αποτελεσματική μεταφορά δεδομένων μεταξύ καταχωρητών, θέσεων μνήμης και περιφερειακών εισόδου/εξόδου. Αυτές οι μέθοδοι προσπέλασης περιλαμβάνουν τις:

- Immediate
- Immediate extended
- Modified page zero
- Relative

- Extended
- Indexed
- Register
- Register indirect
- Implied
- Bit

Ομάδα εντολών Load των 8 bits

Μνημονικό	Συμβολική παρουσίαση	Flags				Opcode			No. of Bytes	No. of M Cycles	No. of T States	Σχόλια					
		S	Z	H	P/V	N	C	76					543	210	Hex		
LD r, r'	r - r'	*	*	X	*	X	*	*	*	01	r	r'	1	1	4	r, r' Κώδικας.	
LD r, n	r - n	*	*	X	*	X	*	*	*	00	r	110	2	2	7	000 B 001 C	
LD r, (HL)	r - (HL)	*	*	X	*	X	*	*	*	01	r	110	1	2	7	010 D	
LD r, (IX+d)	r - (IX+d)	*	*	X	*	X	*	*	*	11	011	101	DD	3	5	19	011 E 100 H 101 L
LD r, (IY+d)	r - (IY+d)	*	*	X	*	X	*	*	*	11	111	101	FD	3	5	19	111 A
LD (HL), r	(HL) - r	*	*	X	*	X	*	*	*	01	110	r	1	2	7		
LD (IX+d), r	(IX+d) - r	*	*	X	*	X	*	*	*	11	011	101	DD	3	5	19	
LD (IY+d), r	(IY+d) - r	*	*	X	*	X	*	*	*	11	111	101	FD	3	5	19	
LD (HL), n	(HL) - n	*	*	X	*	X	*	*	*	00	110	110	36	2	3	10	
LD (IX+d), n	(IX+d) - n	*	*	X	*	X	*	*	*	11	011	101	DD	4	5	19	
LD (IY+d), n	(IY+d) - n	*	*	X	*	X	*	*	*	11	111	101	FD	4	5	19	
LD A, (BC)	A - (BC)	*	*	X	*	X	*	*	*	00	001	010	0A	1	2	7	
LD A, (DE)	A - (DE)	*	*	X	*	X	*	*	*	00	011	010	1A	1	2	7	
LD A, (nn)	A - (nn)	*	*	X	*	X	*	*	*	00	111	010	3A	3	4	13	
LD (BC), A	(BC) - A	*	*	X	*	X	*	*	*	00	000	010	02	1	2	7	
LD (DE), A	(DE) - A	*	*	X	*	X	*	*	*	00	010	010	12	1	2	7	
LD (nn), A	(nn) - A	*	*	X	*	X	*	*	*	00	110	010	32	3	4	13	
LD A, I	A - I	1	1	X	0	X	IFF	0	*	11	101	101	ED	2	2	9	
LD A, R	A - R	1	1	X	0	X	IFF	0	*	11	101	101	ED	2	2	9	
LD I, A	I - A	*	*	X	*	X	*	*	*	01	011	111	5F	2	2	9	
LD R, A	R - A	*	*	X	*	X	*	*	*	01	000	111	4F	2	2	9	

ΠΑΡΑΤΗΡΗΣΕΙΣ: τα r, r' αντιπροσωπεύουν έναν από τους καταχωρητές A, B, C, D, E, H, L.

IFF είναι το περιεχόμενο της flip-flop ενεργοποίησης των interrupt, το (IFF) αντιγράφεται στο P.V flag.

Για την εξήγηση της γραφής των flags και των συμβόλων των μνημονικών πινάκων, δες τους πίνακες Συμβολικής Γραφής που ακολουθούν.

Ομάδα εντολών Load των 16 bits

Μνημονικό	Συμβολική παρουσίαση	Flags				Opcode			No. of Bytes	No. of M Cycles	No. of T States	Σχόλια					
		S	Z	H	P/V	N	C	76					543	210	Hex		
LD dd, nn	dd - nn	*	*	X	*	X	*	*	*	00	d00	001	3	3	10	dd Ζεύγος 00 δε. 01 DE 10 HL 11 SP	
LD IX, nn	IX - nn	*	*	X	*	X	*	*	*	11	011	101	DD	4	4	14	
LD IY, nn	IY - nn	*	*	X	*	X	*	*	*	11	111	101	FD	4	4	14	
LD HL, (nn)	H - (nn+1) L - (nn)	*	*	X	*	X	*	*	*	00	101	010	2A	3	5	16	
LD dd, (nn)	ddH - (nn+1) ddL - (nn)	*	*	X	*	X	*	*	*	11	101	101	ED	4	6	20	

Μνημονικό	Συμβολική παρουσίαση	Flags				Opcode	No. of Bytes	No. of M Cycles	No. of T States	Σχόλια				
		S	Z	H	P/V N C									
LD IX, (nn)	IX _H - (nn+1) IX _L - (nn)	*	*	X	*	X	*	*	*	11 011 101 DD 00 101 010 2A - n - - n -	4	6	20	
LD IY, (nn)	IY _H - (nn+1) IY _L - (nn)	*	*	X	*	X	*	*	*	11 111 101 FD 00 101 010 2A - n - - n -	4	6	20	
LD (nn), HL	(nn+1) - H (nn) - L	*	*	X	*	X	*	*	*	00 100 010 22 - n - - n -	3	5	16	
LD (nn), dd	(nn+1) - dd _H (nn) - dd _L	*	*	X	*	X	*	*	*	11 101 101 ED 01 dd0 011 - n - - n -	4	6	20	
LD (nn), IX	(nn+1) - IX _H (nn) - IX _L	*	*	X	*	X	*	*	*	11 011 101 DD 00 100 010 22 - n - - n -	4	6	20	
LD (nn), IY	(nn+1) - IY _H (nn) - IY _L	*	*	X	*	X	*	*	*	11 111 101 FD 00 100 010 22 - n - - n -	4	6	20	
LD SP, HL LD SP, IX	SP - HL SP - IX	*	*	X	*	X	*	*	*	11 111 001 F9 11 011 101 DD 11 111 001 F9	1 2	1 2	6 10	
LD SP, IY	SP - IY	*	*	X	*	X	*	*	*	11 111 101 FD 11 111 001 F9	2	2	10	
PUSH qq	(SP-2) - qq _L (SP-1) - qq _H SP - SP - 2	*	*	X	*	X	*	*	*	11 qq0 101 - n - - n -	1	3	11	00 BC 01 DE 10 HL 11 AF
PUSH IX	(SP-2) - IX _L (SP-1) - IX _H SP - SP - 2	*	*	X	*	X	*	*	*	11 011 101 DD 11 100 101 E5	2	4	15	
PUSH IY	(SP-2) - IY _L (SP-1) - IY _H SP - SP - 2	*	*	X	*	X	*	*	*	11 111 101 FD 11 100 101 E5	2	4	15	
POP qq	qq _H - (SP+1) qq _L - (SP) SP - SP + 2	*	*	X	*	X	*	*	*	11 qq0 001 - n - - n -	1	3	10	
POP IX	IX _H - (SP+1) IX _L - (SP) SP - SP + 2	*	*	X	*	X	*	*	*	11 011 101 DD 11 100 001 E1	2	4	14	
POP IY	IY _H - (SP+1) IY _L - (SP) SP - SP + 2	*	*	X	*	X	*	*	*	11 111 101 FD 11 100 001 E1	2	4	14	

ΠΑΡΑΤΗΡΗΣΕΙΣ: το dd αντιπροσωπεύει ένα από τα ζεύγη καταχωρητών BC,DE,HL,SP
το qq αντιπροσωπεύει ένα από τα ζεύγη καταχωρητών AF,BC,DE,HL.
Τα (PAIR)_H,(PAIR)_L αναφέρονται στα οκτώ bits υψηλής και χαμηλής τάξης αντίστοιχα του ζεύγους καταχωρητών π.χ. BC_L=C, AF_H=A.

Ομάδες εντολών ανταλλαγής, μεταφοράς και έρευνας περιοχών μνήμης

EX DE, HL EX AF, AF EXX	DE - HL AF - AF BC - BC DE - DE	*	*	X	*	X	*	*	*	11 101 011 EB 00 301 000 08 11 011 001 D9	1 1 1	1 1 4	4 4 4	Ανταλλαγή των περιεχομένων μεταξύ των καταχωρητή και του βοηθητικού καταχωρητή.
EX (SP), HL	HL - HL H - (SP+1) L - (SP)	*	*	X	*	X	*	*	*	11 100 011 E3	1	5	19	
EX (SP), IX	IX _H - (SP+1) IX _L - (SP)	*	*	X	*	X	*	*	*	11 011 101 DD 11 100 011 E3	2	6	23	
EX (SP), IY	IY _H - (SP+1) IY _L - (SP)	*	*	X	*	X	*	*	*	11 111 101 FD 11 100 011 E3	2	6	23	
LDI	(DE) - (HL) DE - DE + 1 HL - HL + 1 BC - BC - 1	*	*	X	0	X	1	0	*	11 101 101 ED 10 100 000 A0	2	4	16	Φέρνει το (HL) στα (DE), αυξάνει τους δείκτες (pointers) και μιάσει τον μετρητή των bytes (BC).
LDI _R	(DE) - (HL) DE - DE + 1 HL - HL + 1 BC - BC - 1 επανάλαβε μέχρι BC = 0	*	*	X	0	X	0	0	*	11 101 101 ED 10 110 000 B0	2 2	5 4	21 16	Αν BC = 0 Αν BC = 0

ΠΑΡΑΤΗΡΗΣΗ: ① Το P/V flag είναι 0 αν το αποτέλεσμα του BC-1=0, αλλιώς P/V=1.

LDD	(DE) - (HL) DE - DE - 1 HL - HL - 1 BC - BC - 1	*	*	X	0	X	1	0	*	11 101 101 ED 10 101 000 A8	2	4	16	
-----	--	---	---	---	---	---	---	---	---	--------------------------------	---	---	----	--

Μνημονικό	Συμβολική περιγραφή	S	Z	Flags H	P/V	M	C	78	543	210	Hex	No. of Bytes	No. of Cycles	No. of States	Σχόλια
LDDR	(DE) ← (HL) DE ← DE-1 HL ← HL-1 BC ← BC-1 επανάλαβε μέχρι BC = 0	*	*	X 0 X 0 0 *	②			11 101 101 ED	10 111 000 B8	2	2	5	4	21 16	Αν BC ≠ 0 Αν BC = 0
CPI	A ← (HL) HL ← HL+1 BC ← BC-1	1	1	X 1 X 1 1 *	①			11 101 101 ED	10 100 001 A1	2	2	4	4	16	
CPDR	A ← (HL) HL ← HL+1 BC ← BC-1 επανάλαβε μέχρι A = (HL) BC = 0	1	1	X 1 X 1 1 *	①			11 101 101 ED	10 110 001 B1	2	2	5	4	21 16	Αν BC = 0 τότε A ← (HL) Αν BC = 0 ή A = (HL)
CPD	A ← (HL) HL ← HL-1 BC ← BC-1	1	1	X 1 X 1 1 *	①			11 101 101 ED	10 101 001 A9	2	2	4	4	16	
CPDR	A ← (HL) HL ← HL-1 BC ← BC-1 Repeat until επανάλαβε μέχρι BC = 0	1	1	X 1 X 1 1 *	①			11 101 101 ED	10 111 001 B9	2	2	5	4	21 16	Αν BC = 0 και A ← (HL) Αν BC = 0 ή A = (HL)

ΠΑΡΑΤΗΡΗΣΕΙΣ: ① Το P/V flag είναι 0 αν το αποτέλεσμα του BC-1=0, αλλιώς P/V=1

② Το P/V flag είναι 0 μόνο μετά την εκτέλεση της εντολής

③ Το Z flag είναι 1 αν A=(HL), αλλιώς Z=0

Ομάδα Λογικών και Αριθμητικών εντολών των 8 bits

ADD A, r	A ← A + r	1	1	X 1 X V 0 1	10 000 r			1	1	4	r	Κατακ.
ADD A, n	A ← A + n	1	1	X 1 X V 0 1	11 000 110			2	2	7	000 B 001 C 010 D 011 E	
ADD A, (HL)	A ← A + (HL)	1	1	X 1 X V 0 1	10 000 110			1	2	7	011 E	
ADD A, (IX+d)	A ← A + (IX+d)	1	1	X 1 X V 0 1	11 011 101	DD		3	5	19	100 H 101 L 111 A	
ADD A, (IY+d)	A ← A + (IY+d)	1	1	X 1 X V 0 1	11 111 101	FD		3	5	19		
ADC A, s	A ← A + s + CY	1	1	X 1 X V 0 1	1001							
SUB s	A ← A - s	1	1	X 1 X V 1 1	010							
SBC A, s	A ← A - s - CY	1	1	X 1 X V 1 1	011							
AND s	A ← A & s	1	1	X 1 X P 0 0	100							
OR s	A ← A s	1	1	X 0 X P 0 0	110							
XOR s	A ← A ⊕ s	1	1	X 0 X P 0 0	101							
CP s	A ← s	1	1	X 1 X V 1 1	111							
INC r	r ← r + 1	1	1	X 1 X V 0 *	00 r 100			1	1	4		
INC (HL)	(HL) ← (HL) + 1	1	1	X 1 X V 0 *	00 110 100			1	3	1		
INC (IX+d)	(IX+d) ← (IX+d) + 1	1	1	X 1 X V 0 *	11 011 101	DD		3	6	2		
INC (IY+d)	(IY+d) ← (IY+d) + 1	1	1	X 1 X V 0 *	00 110 100							
INC (IY+d)	(IY+d) ← (IY+d) + 1	1	1	X 1 X V 0 *	11 111 101	FD		3	6	23		
DEC m	m ← m - 1	1	1	X 1 X V 1 *	00 110 100							

Το s είναι κάποιο από τα r, n, (HL), (IX+d), (IY+d) όπως φαίνεται για την εντολή ADD. Τα σημειούμενα bits αντικαθιστούν τα 000 στην παραπάνω ομάδα εντολών ADD.

Το m είναι κάποιο από τα r, (HL), (IX+d), (IY+d) όπως φαίνεται για την εντολή INC. Η DEC έχει το ίδιο format και κατωτάσεις με την INC. Αντικαταστήστε στον κώδικα λειτουργίας το 100 με το 101.

Ομάδες εντολών Αριθμητικής γενικής χρήσης και ελέγχου της CPU

Μνημονικό	Συμβολική παρουσίαση	Flags								Opcode			No. of Bytes	No. of M Cycles	No. of T States	Σχόλια		
		S	Z	H	P/V	N	C	78	543	210	Hex							
DAA	Converts acc. content into packed BCD following add or subtract with packed BCD operands.	1	1	X	1	X	P	*	*	1	00	100	111	27	1	1	4	Προσρροαση εξααξιου σωωρωετης
CPL	A ← A	*	*	X	1	X	*	*	1	*	00	101	111	2F	1	1	4	συμπληρωματικοσ σωωρωετης
NEG	A ← 0 - A	1	1	X	1	X	V	1	1	1	11	101	101	ED	2	2	8	αρνητικοσ σωωρωετης
CCF	CY ← CY	*	*	X	X	X	*	0	1		01	000	100	44	1	1	4	συμπληρωματικοσ σωωρωετης
SCF	CY ← 1	*	*	X	0	X	*	0	1		00	110	111	37	1	1	4	FLAH κρατουμενου
NOP	No operation	*	*	X	*	X	*	*	*		00	000	000	00	1	1	4	καθερισε το FLAH κρατουμενου
HALT	CPU halted	*	*	X	*	X	*	*	*		01	110	110	76	1	1	4	
DI *	IFF ← 0	*	*	X	*	X	*	*	*		11	110	011	F3	1	1	4	
EI *	IFF ← 1	*	*	X	*	X	*	*	*		11	111	011	FB	1	1	4	
IM 0	Set interrupt mode 0	*	*	X	*	X	*	*	*		11	101	101	ED	2	2	8	
IM 1	Set interrupt mode 1	*	*	X	*	X	*	*	*		11	101	101	ED	2	2	8	
IM 2	Set interrupt mode 2	*	*	X	*	X	*	*	*		11	101	101	ED	2	2	8	

ΠΑΡΑΤΗΡΗΣΕΙΣ: το IFF αντιπροσωπεύει την flip-flop ενεργοποίηση των interrupts
 το CY αντιπροσωπεύει το flip-flop του κρατουμένου
 το * σημαίνει ότι τα interrupts δεν ελέγχονται στο τέλος των EI ή DI.

Ομάδα αριθμητικών εντολών των 16 bits

ADD HL, ss	HL ← HL + ss	*	*	X	X	X	*	0	1		00	ss1	001	1	3	11	κκατασ.	
ADC HL, ss	HL ← HL + ss + CY	1	1	X	X	X	V	0	1		11	101	101	ED	2	4	15	00 BC 01 DE 10 HL 11 SP
SBC HL, ss	HL ← HL - ss - CY	1	1	X	X	X	V	1	1		11	101	101	ED	2	4	15	
ADD IX, pp	IX ← IX + pp	*	*	X	X	X	*	0	1		11	011	101	DD	2	4	15	pp κκατασ. 00 BC 01 DE 10 IX 11 SP
ADD IY, rr	IY ← IY + rr	*	*	X	X	X	*	0	1		11	111	101	FD	2	4	15	rr κκατασ. 00 BC 01 DE 10 IY 11 SP
INC ss	ss ← ss + 1	*	*	X	*	X	*	*	*		00	ss0	011	1	1	6		
INC IX	IX ← IX + 1	*	*	X	*	X	*	*	*		11	011	101	DD	2	2	10	
INC IY	IY ← IY + 1	*	*	X	*	X	*	*	*		00	100	011	23	2	2	10	
DEC ss	ss ← ss - 1	*	*	X	*	X	*	*	*		00	ss1	011	1	1	6		
DEC IX	IX ← IX - 1	*	*	X	*	X	*	*	*		11	011	101	DD	2	2	10	
DEC IY	IY ← IY - 1	*	*	X	*	X	*	*	*		00	101	011	2B	2	2	10	

ΠΑΡΑΤΗΡΗΣΕΙΣ: το ss αντιστοιχεί σε ένα από τα ζεύγη καταχωρητών BC,DE,HL,SP.
 το pp αντιστοιχεί σε ένα από τα ζεύγη καταχωρητών BC,DE,IX,SP.
 το rr αντιστοιχεί σε ένα από τα ζεύγη καταχωρητών BC,DE,IY,SP.

Ομάδα περιστροφής

RLCA		*	*	X	0	X	*	0	1		00	000	111	07	1	1	4	Περιστροφη αριστερου κυκλικου σωωρωετης
RLA		*	*	X	0	X	*	0	1		00	010	111	17	1	1	4	Περιστροφη αριστερου σωωρωετης
RRCA		*	*	X	0	X	*	0	1		00	001	111	0F	1	1	4	Περιστροφη δεξιου κυκλικου σωωρωετης
RRA		*	*	X	0	X	*	0	1		00	011	111	1F	1	1	4	Περιστροφη δεξιου σωωρωετης
RLC r		1	1	X	0	X	P	0	1		11	001	011	CB	2	2	8	Περιστροφη αριστερου κυκλικου σωωρωετης
RLC (HL)		1	1	X	0	X	P	0	1		11	001	011	CB	2	4	15	r κκατασ. 000 B 001 C 010 D 011 E 100 H 101 L 111 A
RLC (IX + d)		1	1	X	0	X	P	0	1		11	011	101	DD	4	6	23	
RLC (IY + d)		1	1	X	0	X	P	0	1		11	111	101	FD	4	6	23	

Μνημονικό	Συμβολική παρουσίαση	Flags					Opcodes 78 54 210 Hex	No. of Bytes	No. of M Cycles	No. of T States	Σχόλια					
		S	Z	H	P/V	N						C				
JP Z, e	AVZ = 0 συνέχισε Z = 1. PC - PC + e	.	.	X	.	X	00 101 000 28 - e-2 -	2	2	7	Αν η συνθήκη δεν ικανοποιείται
IR NZ, e	AVZ = 1. συνέχισε AVZ = 0. PC - PC + e	.	.	X	.	X	00 100 000 20 - e-2 -	2	2	7	Αν η συνθήκη δεν ικανοποιείται
JP (HL)	PC - HL	.	.	X	.	X	11 101 001 E9	1	1	4	Αν η συνθήκη ικανοποιείται
JP (IX)	PC - IX	.	.	X	.	X	11 011 101 DD 11 101 001 E9	2	2	8	
JP (IY)	PC - IY	.	.	X	.	X	11 111 101 FD 11 101 001 E9	2	2	8	
DINZ, e	Z = B-1 AVB = 0. συνέχισε AVB = 0. PC - PC + e	.	.	X	.	X	00 010 000 10 - e-2 -	2	2	8	AVB = 0.
													2	3	13	AVB = 0.

ΠΑΡΑΤΗΡΗΣΕΙΣ: το e αντιπροσωπεύει την επέκταση (extension) στη σχετική (relative) μέθοδο προσπέλασης. το e είναι προσημασμένος αριθμός σε μορφή «συμπληρώματος του 2» εύρους <-126, 129>. το e-2 στον κωδικό λειτουργίας (opcode) δίνει την ενεργό διεύθυνση του pc+e καθώς ο PC αυξάνεται κατά 2 πριν την πρόσθεση του e.

Ομάδα εντολών Call και Return

CALL nn	(SP-1) - PC _H (SP-2) - PC _L PC - nn	.	.	X	.	X	11 001 101 CD - n - - n -	3	5	17	
CALL cc, nn	If condition cc is false continue, otherwise same as CALL nn	.	.	X	.	X	11 cc 100 - n - - n -	3	3	-10	Αν η CC είναι ψευδής
													3	5	17	Αν η CC είναι αληθής
RET	PC _L - (SP) PC _H - (SP+1)	.	.	X	.	X	11 001 001 C9	1	3	10	
RET cc	If condition cc is false continue, otherwise same as RET	.	.	X	.	X	11 cc 000	1	1	5	Αν η CC είναι ψευδής
													1	3	11	Αν η CC είναι αληθής
																Υπόθεση CC 000 NZ non-zero 001 Z zero 010 NC non-carry 011 C carry 100 PO parity odd 101 PE parity even 110 P sign positive 111 M sign negative
RETI	Return from interrupt	.	.	X	.	X	11 101 101 ED 01 001 101 4D	2	4	14	
RETI ¹	Return from non-maskable interrupt	.	.	X	.	X	11 101 101 ED 01 000 101 45	2	4	14	
RST p	(SP-1) - PC _H (SP-2) - PC _L PC _H = 0 PC _L = p	.	.	X	.	X	11 1 111	1	3	11	1 p 000 00H 001 08H 010 10H 011 18H 100 20H 101 28H 110 30H 111 38H

ΠΑΡΑΤΗΡΗΣΗ: ① η RETN φορτώνει (loads) IFF₂ → IFF₁

Ομάδα εντολών εισόδου - εξόδου

IN A, (n)	A - (n)	.	.	X	.	X	11 011 011 DB - n -	2	3	11	n to A ₀ - A ₇ Acc. to A ₈ - A ₁₅
IN r, (C)	r - (C) Εάν r=110 μόνο τότε οι flags επηρεάζονται	1	1	X	1	X	P	0	.	.	.	11 101 101 ED 01 r 000	2	3	12	C to A ₀ - A ₇ B to A ₈ - A ₁₅
INI	(HL) - (C) B - B - 1 HL - HL + 1	X	1	X	X	X	X	1	X	.	.	11 101 101 ED 10 100 010 A2	2	4	16	C to A ₀ - A ₇ B to A ₈ - A ₁₅
INR	(HL) - (C) B - B - 1 HL - HL + 1 επιπλέον μέχρι B = 0	X	1	X	X	X	X	1	X	.	.	11 101 101 ED 10 110 010 B2	2	5	21	C to A ₀ - A ₇ B to A ₈ - A ₁₅
													2	4	16	(AVB=0) (AVB=0)
IND	(HL) - (C) B - B - 1 HL - HL - 1	X	1	X	X	X	X	1	X	.	.	11 101 101 ED 10 101 010 AA	2	4	16	C to A ₀ - A ₇ B to A ₈ - A ₁₅
INDR	(HL) - (C) B - B - 1 HL - HL - 1 επιπλέον μέχρι B = 0	X	1	X	X	X	X	1	X	.	.	11 101 101 ED 10 111 010 BA	2	5	21	C to A ₀ - A ₇ B to A ₈ - A ₁₅
													2	4	16	(AVB=0) (AVB=0)

Μνημονικό	Συμβολική παρονομασία	Flags						Opcode 7B 543 210 Hex	No. of Bytes	No. of M Cycles	No. of T States	Σχόλια	
		S	Z	H	P/V	N	C						
OUT (n), A	(n) - A	*	*	X	*	X	*	*	11 010 011 D3	2	3	11	n to A ₀ - A ₇ Acc. to A ₈ - A ₁₅
OUT (C), r	(C) - r	*	*	X	*	X	*	*	11 101 101 ED 01 r 001	2	3	12	C to A ₀ - A ₇ B to A ₈ - A ₁₅
OUTI	(C) - (HL) B - B-1 HL - HL + 1	X	1	X	X	X	X	1 X	11 101 101 ED 10 100 011 A3	2	4	16	C to A ₀ - A ₇ B to A ₈ - A ₁₅
OTIR	(C) - (HL) B - B-1 HL - HL + 1 επανάλαβε μέχρι B = 0	X	1	X	X	X	X	1 X	11 101 101 ED 10 110 011 B3	2	5 4 4	21 16	C to A ₀ - A ₇ B to A ₈ - A ₁₅ (AV B=0) (AV B=0)
OUTD	(C) - (HL) B - B-1 HL - HL-1	X	1	X	X	X	X	1 X	11 101 101 ED 10 101 011 AB	2	4	16	C to A ₀ - A ₇ B to A ₈ - A ₁₅
NOTE: ① If the result of B-1 is zero the Z flag is set, otherwise it is reset. ② Z flag is set upon instruction completion only.													
OTDR	(C) - (HL) B - B-1 HL - HL-1 επανάλαβε μέχρι B = 0	X	1	X	X	X	X	1 X	11 101 101 ED 10 111 011	2	5 4 4	21 16	C to A ₀ - A ₇ B to A ₈ - A ₁₅ (AV B=0) (AV B=0)

ΠΑΡΑΤΗΡΗΣΗ: ① το Z flag γίνεται «1» (set) μόνο μετά την εκτέλεση της εντολής

Περίληψη της λειτουργίας των Flags

Instruction	D ₇	S	Z	H	P/V	N	D ₀	C	Comments
ADD A, r; ADC A, s	1	1	X	1	X	V	0	1	Πρόσθεση των 8 bits ή πρόσθεση με μεταφορά κρατούμενου (add with carry)
SUB s; SBC A, s; CP s; NEG	1	1	X	1	X	V	1	1	Αφαίρεση των 8 bits, αφαίρεση με μεταφορά κρατούμενου, σύγκριση (compare) και αρνητικός συσσωρευτής (negate accumulator)
AND s	1	1	X	1	X	P	0	0	Λογικές πράξεις.
OR s, XCR s	1	1	X	0	X	P	0	0	
INC s	1	1	X	1	X	V	0	*	
DEC s	1	1	X	1	X	V	1	*	Μείωση των 8 bits.
ADD DD, ss	*	*	X	X	X	*	0	1	Πρόσθεση των 16 bits.
ADC HL, ss	1	1	X	X	X	V	0	1	Πρόσθεση των 16 bits με μεταφορά κρατούμενου.
SBC HL, ss	1	1	X	X	X	V	1	1	Αφαίρεση των 16 bits με μεταφορά κρατούμενου.
RLA, RLCA, RRA; RRCA	*	*	X	0	X	*	0	1	Περιστροφή του συσσωρευτή.
RL m; RLC m; RR m;	1	1	X	0	X	P	0	1	Θάσεις περιστροφής και μετατόπισης.
RR m; SLA m;									Αριστερή και δεξιά περιστροφή ψηφίου.
SRA m; SRL m;	1	1	X	0	X	P	0	*	
RLD; RRD	1	1	X	1	X	P	*	1	
DAA	1	1	X	0	X	P	*	1	
CPZ	*	*	X	1	X	*	1	*	Συμπληρωματικός συσσωρευτής (Complement accumulator).
SCF	*	*	X	0	X	*	0	1	Ενεργοποίηση (set) του carry flag.
CCF	*	*	X	X	X	*	1	1	Συμπλήρωμα (complement) του carry flag
IN r (C)	1	1	X	0	X	P	0	*	Είσοδος εισαγωγή καταχωρητή (Input Register indirect).
INI, IND, OUTI, OUTD	X	1	X	X	X	X	1	*	Είσοδος και έξοδος περιοχής μνήμης. Z=0 αν B≠0, αλλιώς Z=0.
INIR; INDR; OTIR; OTDR	X	1	X	X	X	X	1	*	
LDI; LDD	X	X	X	0	X	1	0	*	
LDIR; LDDR	X	X	X	0	X	0	0	*	
CPH; CPIL; CPD; CPDR	X	1	X	X	X	1	1	1	Εντολές μεταφοράς περιοχής μνήμης Z=1 αν A=(HL), αλλιώς Z=0. P/V=1 αν BC≠0, αλλιώς P/V=0.
LD A, 1, LD A, R	1	1	X	0	X	IFF	0	*	Το περιεχόμενο του flip-flop ενεργοποίησης των interrupt (IFF) αντιγράφεται στο P/V flag.
BIT s, s	X	1	X	1	X	X	0	*	Η κατάσταση (state) του bit b της θέσης s αντιγράφεται στο Z flag.

Συμβολική γραφή

Σύμβολο

Λειτουργία

- S Το flag του προσήμου (sign flag). S=1 αν το σημαντικότερο byte (MSB) του αποτελέσματος είναι 1.
- Z Το μηδενικό flag (zero flag). Z=1 αν το αποτέλεσμα της πράξης είναι 0.
- P/V Το flag ισοτιμίας (parity) ή υπερχείλισης (overflow). Η ισοτιμία (P) και η υπερχείλιση (V) μοιράζονται το ίδιο flag. Οι λογικές πράξεις επιδρούν σ' αυτό το flag ως προς την ισοτιμία ενώ οι αριθμητικές πράξεις επιδρούν σ' αυτό το flag ως προς την υπερχείλιση. Αν το P/V περιέχει την ισοτιμία, τότε P/V=1 αν το αποτέλεσμα της πράξης είναι άρτιο, και P/V=0 αν το αποτέλεσμα είναι περιττό. Αν το P/V περιέχει την υπερχείλιση, τότε P/V=1 αν το αποτέλεσμα της πράξης προέβλεπε υπερχείλιση.
- H Το flag μισού κρατούμενου (half-carry flag). H=1 αν η πράξη της πρόσθεσης ή της αφαίρεσης παρήγαγε κρατούμενο ή δανείστηκε από το bit 4 του συσσωρευτή.
- N Το flag πρόσθεσης/αφαίρεσης (add/subtract flag). N=1 αν η προηγούμενη πράξη ήταν αφαίρεση.

H&N	Τα flag H και N χρησιμοποιούνται σε συνδυασμό με την εντολή DAA(decimal adjust instruction) για να διορθώνει κατάλληλα το αποτέλεσμα σε format packed BCD μετά από πρόσθεση ή αφαίρεση που χρησιμοποιούσε operands με format packed BCD.
C	Το flag Carry/Link. C=1 αν η πράξη παρήγαγε ένα κρατούμενο από το σημαντικότερο byte (MSB) του operand ή του αποτελέσματος.
‡	Το flag επηρεάζεται σύμφωνα με το αποτέλεσμα της πράξης.
●	Το flag δεν αλλάζει από την πράξη.
0	Το flag γίνεται «0» (reset) από την πράξη.
1	Το flag γίνεται «1» (set) από την πράξη.
x	Το flag δεν ενδιαφέρει.
V	Το P/V flag επηρεάζεται σε περίπτωση υπερχείλισης ενός αποτελέσματος.
P	Το P/V flag επηρεάζεται σύμφωνα με την ισοτιμία της πράξης.
r	Οποιοσδήποτε από τους καταχωρητές της CPU A,B,C,D,E,H,L.
s	Οποιαδήποτε θέση των 8 bits για όλες τις μεθόδους προσπέλασης (addressing modes) που επιτρέπονται για την συγκεκριμένη εντολή.
ii	Οποιοσδήποτε από τους δύο καταχωρητές - δείκτες (index registers) IX ή IY.
R	O Refresh counter.
n	τιμή των 8 bits εύρους <0,255>
nn	τιμή των 16 bits εύρους <0,65535>.

Παράρτημα Β

```
1000 REM APPENDIX B
1010 REM HEXLOADER
1020 MODE 1
1030 ER%=1:L%=-4
1040 PEN 2:PRINT"SET MEMORY TO":
1050 GOSUB 1270
1060 IF B>42600 OR B<2000 THEN ER%=-1:GOTO 1250
1070 MM=42619:MEMORY B
1080 PAPER 2:PEN 0:PRINT"MEMORY SET TO ":HEX$(HI
MEM):" HEX"
1090 L%=-4
1100 PRINT"INPUT START ADDRESS":.PAPER 3
1110 GOSUB 1270
1120 IF B<=HIMEM THEN ER%=2:GOTO 1250
1130 IF B>42619 THEN ER%=-5:GOTO 1250
1140 START=B:PEN 3:PAPER 2:PRINT"START INPUT":PA
PER 0
1150 STAD=B
1160 INAD=STAD:CHECK=0
1170 L%=-2
1180 WHILE INAD<STAD+10
1190 GOSUB 1270:POKE INAD,B:PEN 2:PRINT HEX$(INA
D,4),HEX$(B,2):PEN 1:CHECK=CHECK+B:INAD=INAD+1:I
F INAD>=MM-2 THEN INAD=STAD+20
1200 WEND:IF INAD=STAD+20 THEN ER%=4:GOTO 1250
1210 PAPER 3:PRINT"INPUT CHECK-SUM ":PAPER 0:L%-
4:GOSUB 1270
1220 IF CHECK<>B THEN ER%=3:GOTO 1250
1230 IF FIN=1 THEN PEN 2:PAPER 3:PRINT" FINISHED
":PEN 1:INPUT" MORE? Y?N":A$:PAPER 0:A$=UPPER$(A
$):IF ASC(A$)=89 THEN FIN=0:GOTO 1080 ELSE END
1240 STAD=INAD:PEN 0:PAPER 2:PRINT"CHECK-SUM ":H
EX$(B,4):" CORRECT ! CONTINUE INPUT":PEN 1:PAPER
0:GOTO 1160
1250 RESTORE 1390:PEN 3:PAPER 1:FOR N%=1 TO ER%
:READ DS:NEXT:PRINT DS:" TRY AGAIN":CHR$(7)
```

```

1260 PEN 1:PAPER 0:ON ER% GOTO 1030,1090,1160,10
30,1090
1270 A%=0:B=0
1280 PEN 1:INPUT ST$:PRINT CHR$(11)::ST$=UPPER$(
ST$):IF ST$="END" THEN 1370
1290 IF LEN(ST$)<>L% THEN 1360
1300 FOR N%=1 TO L%
1310 A$=MID$(ST$,N%,1):IF A$>"F" OR A$<"O" OR (A
$>"9" AND A$<"A") THEN 1360
1320 IF A$>"9" THEN A%=ASC(A$):A%=(A% AND &F)+9
ELSE A%=VAL(A$)
1330 IF N%<>L% THEN B=B+(A%*16^(L%-N%)) ELSE B=B
+A%
1340 NEXT
1350 RETURN
1360 PEN 3:PAPER 1:PRINT"INVALID INPUT, TRY AGAI
N":CHR$(7):PEN 1:PAPER 0:GOTO 1270
1370 REM END
1380 FIN=1:GOTO 1210
1390 DATA UNREALISTICALLY LOW OR HIGH,UNPROTECTE
D MEMORY AREA,CHECSUM DOES NOT MATCH YOU WILL H
AVE TO RE-ENTER FROM THE LAST CHECK, OUT OF MEM
ORY, TOO HIGH"

```

Παράρτημα C

ΜΕΤΑΤΡΟΠΗ HEX ΣΤΟ ΔΕΚΑΔΙΚΟ ΤΟΥ MSB (Σημαντικότερου byte)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	256	512	768	1024	1280	1536	1792	2048	2304	2560	2816	3072	3328	3584	3840
1	4096	4352	4608	4864	5120	5376	5632	5888	6144	6400	6656	6912	7168	7424	7680	7936
2	8192	8448	8704	8960	9216	9472	9728	9984	10240	10496	10752	11008	11264	11520	11776	12032
3	12288	12544	12800	13056	13312	13568	13824	14080	14336	14592	14848	15104	15360	15616	15872	16128
4	16384	16640	16896	17152	17408	17664	17920	18176	18432	18688	18944	19200	19456	19712	19968	20224
5	20480	20736	20992	21248	21504	21760	22016	22272	22528	22784	23040	23296	23552	23808	24064	24320
6	24576	24832	25088	25344	25600	25856	26112	26368	26624	26880	27136	27392	27648	27904	28160	28416
7	28672	28928	29184	29440	29696	29952	30208	30464	30720	30976	31232	31488	31744	32000	32256	32512
8	32768	33024	33280	33536	33792	34048	34304	34560	34816	35072	35328	35584	35840	36096	36352	36608
9	36864	37120	37376	37632	37888	38144	38400	38656	38912	39168	39424	39680	39936	40192	40448	40704
A	40960	41216	41472	41728	41984	42240	42496	42752	43008	43264	43520	43776	44032	44288	44544	44800
B	45056	45312	45568	45824	46080	46336	46592	46848	47104	47360	47616	47872	48128	48384	48640	48896
C	49152	49408	49664	49920	50176	50432	50688	50944	51200	51456	51712	51968	52224	52480	52736	52992
D	53248	53504	53760	54016	54272	54528	54784	55040	55296	55552	55808	56064	56320	56576	56832	57088
E	57344	57600	57856	58112	58368	58624	58880	59136	59392	59648	59904	60160	60416	60672	60928	61184
F	61440	61696	61952	62208	62464	62720	62976	63232	63488	63744	64000	64256	64512	64768	65024	65280

Παράρτημα D

ΜΕΤΑΤΡΟΠΗ HEX ΣΤΟ ΔΕΚΑΔΙΚΟ ΤΟΥ LSB (Λιγότερο σημαντικού byte)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

NIBBLES

HEX	DEC	BIN	HEX	DEC	BIN
0	0	0000	8	8	1000
1	1	0001	9	9	1001
2	2	0010	A	10	1010
3	3	0011	B	11	1011
4	4	0100	C	12	1100
5	5	0101	D	13	1101
6	6	0110	E	14	1110
7	7	0111	F	15	1111

ΜΕΤΑΤΡΟΠΗ HEX «ΣΥΜΠΛΗΡΩΜΑΤΟΣ ΤΟΥ 2» ΣΤΟ ΔΕΚΑΔΙΚΟ ΤΟΥ MSB (Σημαντικότερου byte).

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	256	512	768	1024	1280	1536	1792	2048	2304	2560	2816	3072	3328	3584	3840
1	4096	4352	4608	4864	5120	5376	5632	5888	6144	6400	6656	6912	7168	7424	7680	7936
2	8192	8448	8704	8960	9216	9472	9728	9984	10240	10496	10752	11008	11264	11520	11776	12032
3	12288	12544	12800	13056	13312	13568	13824	14080	14336	14592	14848	15104	15360	15616	15872	16128
4	16384	16640	16896	17152	17408	17664	17920	18176	18432	18688	18944	19200	19456	19712	19968	20224
5	20480	20736	20992	21248	21504	21760	22016	22272	22528	22784	23040	23296	23552	23808	24064	24320
6	24416	24672	24928	25184	25440	25696	25952	26208	26464	26720	26976	27232	27488	27744	28000	28256
7	28384	28640	28896	29152	29408	29664	29920	30176	30432	30688	30944	31200	31456	31712	31968	32224
8	-32768	-32512	-32256	-32000	-31744	-31488	-31232	-30976	-30720	-30464	-30208	-29952	-29696	-29440	-29184	-28928
9	-28672	-28416	-28160	-27904	-27648	-27392	-27136	-26880	-26624	-26368	-26112	-25856	-25600	-25344	-25088	-24832
A	-24576	-24320	-24064	-23808	-23552	-23296	-23040	-22784	-22528	-22272	-22016	-21760	-21504	-21248	-20992	-20736
B	-20480	-20224	-19968	-19712	-19456	-19200	-18944	-18688	-18432	-18176	-17920	-17664	-17408	-17152	-16896	-16640
C	-16384	-16128	-15872	-15616	-15360	-15104	-14848	-14592	-14336	-14080	-13824	-13568	-13312	-13056	-12800	-12544
D	-12288	-12032	-11776	-11520	-11264	-11008	-10752	-10496	-10240	-9984	-9728	-9472	-9216	-8960	-8704	-8448
E	-8192	-7936	-7680	-7424	-7168	-6912	-6656	-6400	-6144	-5888	-5632	-5376	-5120	-4864	-4608	-4352
F	-4096	-3840	-3584	-3328	-3072	-2816	-2560	-2304	-2048	-1792	-1536	-1280	-1024	-768	-512	-256

Όπου ένας προσημασμένος αριθμός των 16 bits είναι αρνητικός, η τιμή που φαίνεται εδώ θα πρέπει να μειωθεί κι άλλο από την χωρίς πρόσημο τιμή των 8 bits των 8 bits χαμηλής τάξης (low).

ΜΕΤΑΤΡΟΠΗ HEX «ΣΥΜΠΛΗΡΩΜΑΤΟΣ ΤΟΥ 2» ΣΤΟ ΔΕΚΑΔΙΚΟ ΤΟΥ LSB (Λιγότερο σημαντικό byte).

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	-128	-127	-126	-125	-124	-123	-122	-121	-120	-119	-118	-117	-116	-115	-114	-113
9	-112	-111	-110	-109	-108	-107	-106	-105	-104	-103	-102	-101	-100	-99	-98	-97
A	-96	-95	-94	-93	-92	-91	-90	-89	-88	-87	-86	-85	-84	-83	-82	-81
B	-80	-79	-78	-77	-76	-75	-74	-73	-72	-71	-70	-69	-68	-67	-66	-65
C	-64	-63	-62	-61	-60	-59	-58	-57	-56	-55	-54	-53	-52	-51	-50	-49
D	-48	-47	-46	-45	-44	-43	-42	-41	-40	-39	-38	-37	-36	-35	-34	-33
E	-32	-31	-30	-29	-28	-27	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17
F	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Ο ΧΑΡΤΗΣ ΟΘΟΝΗΣ ΤΟΥ AMSTRAD

Ο χάρτης οθόνης (screen map) στον Amstrad δεν βρίσκεται εύκολα, όσο κι αν χρησιμοποιήσετε την φαντασία σας. Η αρχική διεύθυνση μπορεί να αλλάξει, και ένα pixel αντιπροσωπεύεται από διαφορετικά bits ανάλογα με το τρέχον mode.

Στην οθόνη αφιερώνονται πάντοτε 16K μνήμης. Η αρχή της μνήμης της οθόνης (screen memory) θα είναι στη C000h (49152) εκτός αν κάποιο πρόγραμμα την έχει μετακινήσει. Η άλλη πιθανή αρχή είναι η 4000h (16384) αλλά αυτό πρέπει να γίνει από κάποιο πρόγραμμα. Είναι απίθανο να μετακινηθεί η περιοχή μνήμης της οθόνης έτσι όσα ακολουθούν στηρίζονται στην υπόθεση ότι η μνήμη της οθόνης αρχίζει στη C000h.

Η οθόνη αποτελείται πάντοτε από 200 γραμμές ύψους ενός pixel, και σε μια γραμμή της οθόνης αντιστοιχίζονται 80 διαδοχικά bytes από μια διεύθυνση που είναι C000h συν ένα πολλαπλάσιο του 80, από την αριστερή πλευρά προς τη δεξιά. Ένας χαρακτήρας είναι πάντοτε ένα τετράγωνο πλευράς 8 pixels, έτσι μπορεί να δείχτει ότι, στο mode 2, υπάρχει μια αναλογία ένα - προς - ένα των bits προς τα pixels. Ένα ενεργοποιημένο (set) bit δείχνει ότι το pixel είναι ink 1 και ένα αποενεργοποιημένο (reset) bit είναι ink 0.

Αρχικά (πριν το ρολάρισμα της οθόνης) το πάνω αριστερά τμήμα της οθόνης αρχίζει στη διεύθυνση C000h. Τα πρώτα 80 bytes σχηματίζουν τη γραμμή της κορυφής, ενώ τα επόμενα 80 bytes δεν σχηματίζουν τη δεύτερη γραμμή. Αποτελούν την γραμμή της κορυφής της επόμενης σειράς χαρακτήρων, δηλαδή την ένατη γραμμή των pixels, τα επόμενα 80 bytes είναι η 17η γραμμή των pixels, κ.ο.κ για το σύνολο των 25 γραμμών χαρακτήρων. Μόνον αφού τελειώσουμε με τις πρώτες γραμμές των pixels και των 25 γραμμών χαρακτήρων, καθορίζονται οι δεύτερες γραμμές των pixels.

Στην αρχή επομένως οι διευθύνσεις της οθόνης του πρώτου και του τελευταίου byte των γραμμών pixels 1 έως 24, θα είναι όπως φαίνονται στο παρακάτω Σχήμα.

Το λειτουργικό σύστημα προσφέρει υπορουτίνες που σας επιτρέπουν να υπολογίζετε τις διευθύνσεις για τη θέση ενός χαρακτήρα ή ενός pixel, και οι διευθύνσεις που καλούνται δίνονται στο Παράρτημα G.

LINE No.	ADDRESS		LINE No.	ADDRESS	
	LEFT	RIGHT		LEFT	RIGHT
1	C000	C04F	13	E050	E09F
2	CB00	CB4F	14	E850	E89F
3	D000	D04F	15	F050	F09F
4	DB00	DB4F	16	F850	F89F

5	E000	E04F	17	C0A0	C0DF
6	E800	E84F	18	CBA0	C8DF
7	F000	F04F	19	D0A0	D0DF
8	F800	F84F	20	DBA0	D8DF
9	C050	C09F	21	E0A0	E0DF
10	C850	C89F	22	E8A0	E8DF
11	D050	D09F	23	F0A0	F0DF
12	D850	D89F	24	F8A0	F8DF

Στα modes εκτός του 2 τα bits σε κάθε byte δεν έχουν μια σχέση ένα - προς - ένα με τα pixels στην οθόνη, γιατί το κάθε byte χρειάζεται να κωδικοποιεί περισσότερα από δύο χρώματα μελάνης (ink colours). Η σειρά των bytes κατά μήκος της οθόνης μένει σταθερή σε όλα τα modes αλλά στο mode 1 το κάθε byte δίνει πληροφορίες για τέσσερα pixels, και στο mode 0 το κάθε byte περιγράφει μόνο δύο pixels.

Το κάθε byte αντιπροσωπεύει pixels από αριστερά προς τα δεξιά ως εξής:

Mode 1 από αριστερά προς τα δεξιά
bits 3&72&61&50&4

Mode 0 από αριστερά προς τα δεξιά
bits 1,5,3&70,4,2&6

Τα bits δίνονται σε σειρά σημαντικότητας και κωδικοποιούν το χρώμα μελάνης στο standard δυαδικό σύστημα.

Για παράδειγμα: στο Mode 1, η διεύθυνση C000h με το byte 00110101b θα δώσει τα τέσσερα πρώτα pixels (στο πάνω αριστερό τμήμα της οθόνης) σε μελάνη (ink) 1,2,3, και 4 αντίστοιχα.

Στο mode 0, θα μας έδινε δύο pixels, το πρώτο σε ink 4 και το δεύτερο σε ink 14. Για να δοθούν 4 pixels σε ink 1,2,3, και 4 στο mode 0 θα χρειάζονταν τα εξής δύο bytes:

```
01000000 10011000
```

Το ρολάρισμα ολόκληρης της οθόνης από το hardware πετυχαίνεται μεταβάλλοντας το αρχικό pixel στην αρχή της μνήμης της οθόνης κατά 80, και επομένως η διεύθυνση του πρώτου (πάνω αριστερά) pixel θα είναι C000h + 80 έως 80*25 mod 2048. Ευτυχώς το firmware προσφέρει ρουτίνες για τον καθορισμό της αρχικής διεύθυνσης (δες Παράρτημα G).

ΧΡΗΣΙΜΕΣ ΚΛΗΣΕΙΣ ΔΙΕΥΘΥΝΣΕΩΝ

Ο Key Manager

Τα σύμβολα επέκτασης (expansion token) δεν αυξάνονται εκτός αν έτσι αναφέρεται. Το buffer του πληκτρολογίου δεν καθαρίζει εκτός αν έτσι αναφέρεται.

ΚΛΗΣΗ ΔΙΕΥΘΥΝΣΗΣ ΣΤΟ HEX	ΛΕΙΤΟΥΡΓΙΑ	ΕΠΗΡΕΑΖΟΜΕΝΟΙ ΚΑΤΑΧΩΡΗΤΕΣ
BB00	ΚΑΝΕΙ RESET ΣΕ ΟΛΟΚΛΗΡΟ ΤΟΝ KEY MANAGER ΚΑΘΑΡΙΖΕΙ ΤΟΝ BUFFER	AF BC PE HL ΕΝΕΡΓΟΠΟΙΕΙ ΤΑ INTS
BB12	ΠΑΙΡΝΕΙ ΤΟ ΑΛΦΑΡΙΘΜΗΤΙΚΟ ΕΠΕΚΤΑΣΗΣ (EXPANSION STRING). ΜΕ ΤΗΝ ΕΙΣΑΓΩΓΗ Ο Α ΘΑ ΕΙΝΑΙ ΤΟ ΣΥΜΒΟΛΟ ΕΠΕΚΤΑΣΗΣ ΚΑΙ L Ο ΑΡΙΘΜΟΣ ΤΟΥ ΧΑΡΑΚΤΗΡΑ. ΜΕ ΤΗΝ ΕΞΟΔΟ A=CHAR ΚΑΙ ΤΟ CARRY FLAG ΕΙΝΑΙ «1» (SET), ΑΛΛΟΙΩΣ ΔΕΝ ΥΠΑΡΧΕΙ ΔΙΑΘΕΣΙΜΟΣ ΧΑΡΑΚΤΗΡΑΣ	AF DE
BB18	ΠΕΡΙΜΕΝΕΙ ΤΟ ΠΑΤΗΜΑ ΕΝΟΣ ΠΛΗΚΤΡΟΥ. ΕΠΙΣΤΡΕΦΕΙ ΤΟΝ ΚΩΔΙΚΟ ΣΤΟΝ ΚΑΤΑΧΩΡΗ- ΤΗ Α	AF
BB1B	ΔΙΑΒΑΖΕΙ ΤΟ ΠΛΗΚΤΡΟΛΟΓΙΟ, ΚΑΝΕΙ «1» (SET) ΤΟ CARRY FLAG ΑΝ ΠΑΤΗΘΕΙ ΠΛΗΚΤΡΟ, ΚΑΙ ΕΠΙΣΤΡΕΦΕΙ ΤΟΝ ΚΩΔΙΚΟ ΣΤΟΝ Α. ΜΠΟΡΕΙ ΝΑ ΧΡΗΣΙΜΟΠΟΙΗΘΕΙ ΓΙΑ ΤΟ «ΚΑΘΑΡΙΣΜΑ» ΤΟΥ BUFFER	AF
BB1E	ΕΛΕΓΧΕΙ ΑΝ ΠΑΤΗΘΗΚΕ ΤΟ ΠΛΗΚΤΡΟ ΤΟΥ ΟΠΟΙΟΥ Ο ΑΡΙΘΜΟΣ ΒΡΙΣΚΕΤΑΙ ΣΤΟΝ Α. ΤΟ ZERO FLAG ΕΙΝΑΙ «1» ΑΝ ΔΕΝ ΠΑΤΗΘΗΚΕ	AF HL C (bit 7 για CTRL (bit 5 για SHIFT)
BB24	ΠΑΙΡΝΕΙ ΤΟ JOYSTICK A&H=JOY 0 L=JOY 1 ΑΝ ΓΙΝΕΙ ΕΝΕΡΓΕΙΑ ΤΟ BIT ΓΙΝΕΤΑΙ «1» (SET) 0, ΠΙΑΝΟ 1, ΚΑΤΩ 2, ΑΡΙΣΤΕΡΑ 3, ΔΕΞΙΑ 4, FIRE 2 5, FIRE 1, 6, ΧΩΡΙΣ ΑΝΑΘΕΣΗ 7, ΠΑΝΤΟΤΕ «0» (RESET)	AF HL
H TEXT VDU		
BB4E	ΑΡΧΙΚΕΣ ΤΙΜΕΣ	AFBC DE HL

BB5A	ΤΥΠΩΣΕ ΣΤΗΝ ΟΘΟΝΗ ΤΟΝ ΧΑΡΑΚΤΗΡΑ ΠΟΥ ΒΡΙΣΚΕΤΑΙ ΣΤΟΝ Α	ΚΑΝΕΙΣ
BB60	ΔΙΑΒΑΣΕ ΤΟΝ ΧΑΡΑΚΤΗΡΑ ΑΠΟ ΤΗΝ ΤΡΕΧΟΥΣΑ ΘΕΣΗ ΤΟΥ ΔΡΟΜΕΑ. Ο Α ΠΕΡΙΕΧΕΙ ΤΟΝ ΧΑΡΑΚΤΗΡΑ ΠΟΥ ΔΙΑΒΑΣΤΗΚΕ ΑΝ ΗΤΑΝ «1» (SET) ΤΟ ΣΩΣΤΟ CARRY FLAG	ΑF
BB75	ΤΟΠΟΘΕΤΗΣΕ ΤΟΝ ΔΡΟΜΕΑ ΣΤΗ ΣΤΗΛΗ ΧΑΡΑΚΤΗΡΩΝ Η ΚΑΙ ΣΤΗ ΓΡΑΜΜΗ L	ΑF HL

ΟΙ ΠΕΡΙΣΣΟΤΕΡΕΣ ΑΠΟ ΤΙΣ ΥΠΟΛΟΙΠΕΣ ΔΥΝΑΤΕΣ ΕΝΕΡΓΕΙΕΣ ΓΙΑ ΤΗΝ TEXT VDU ΜΠΟΡΟΥΝ ΝΑ ΓΙΝΟΥΝ ΜΕ ΤΟΥΣ ΚΩΔΙΚΟΥΣ ΕΛΕΓΧΟΥ ΕΚΤΥΠΩΣΗΣ (PRINTING CONTROL CODES), ΔΕΣ ΚΕΦΑΛΑΙΟ 9 ΣΕΛΙΔΑ 2 ΤΟΥ AMSTRAD USER INSTRUCTIONS.

Η GRAPHICS VDU

BB8A	ΑΡΧΙΚΕΣ ΤΙΜΕΣ	ΑF BC DE HL
BBC0	ΤΟΠΟΘΕΤΗΣΕ ΤΗΝ ΑΡΧΗ ΤΩΝ GRAPHICS ΣΤΑ DE (X) HL (Y)	ΑF BC PE HL
BBDE	ΕΝΕΡΓΟΠΟΙΗΣΕ ΤΗΝ GRAPHICS PEN. Ο Α ΠΕΡΙΕΧΕΙ ΤΟΝ ΑΡΙΘΜΟ ΜΕΛΑΝΗΣ (INK No)	ΑF
BBEA	PLOT DE (X), HL (Y)	ΑF BC DE HL
BBF6	ΤΡΑΒΗΣΕ ΓΡΑΜΜΗ ΑΠΟ ΤΗΝ ΤΩΡΙΝΗ ΑΡΧΗ ΣΤΟ DE(X), HL(Y) ΚΑΙ ΕΝΗΜΕΡΩΣΕ ΤΗΝ ΑΡΧΗ	ΑF BC DE HL
BBFC	ΓΡΑΨΕ ΕΝΑ ΧΑΡΑΚΤΗΡΑ ΣΤΗΝ ΑΡΧΗ ΤΩΝ GRAPHICS. Α=Ο ΚΩΔΙΚΟΣ ΤΟΥ ΧΑΡΑΚΤΗΡΑ, Η ΑΡΧΗ ΕΙΝΑΙ ΠΑΝΩ ΑΡΙΣΤΕΡΑ. Η ΑΡΧΗ ΜΕΤΑΚΙΝΕΙΤΑΙ ΚΑΤΑ 8 PIXELS ΠΡΟΣ ΤΑ ΔΕΞΙΑ	ΑF BC DE HL

ΤΟ SCREEN PACK

BBFF	ΑΡΧΙΚΕΣ ΤΙΜΕΣ	ΑF BC DE HL
BC05	ΤΟΠΟΘΕΤΗΣΕ ΤΗΝ ΑΡΧΗ. Ο HL ΠΕΡΙΕΧΕΙ ΤΗΝ ΑΡΧΗ ΠΟΥ ΧΡΕΙΑΖΕΤΑΙ (ΜΟΝΟ ΑΡΤΙΟΙ ΑΡΙΘΜΟΙ) Η ΑΡΧΗ MOD 80 ΘΑ ΡΟΛΑΡΕΙ ΤΗΝ ΟΘΟΝΗ	ΑF HL
BC1A	ΕΠΙΣΤΡΕΦΕΙ ΣΤΟΝ HL ΤΗΝ ΔΙΕΥΘΥΝΣΗ ΤΗΣ ΠΑΝΩ ΑΡΙΣΤΕΡΑ ΘΕΣΗΣ ΧΑΡΑΚΤΗΡΑ Η (ΣΤΗΛΗ) L (ΓΡΑΜΜΗ). Ο Β ΘΑ ΠΕΡΙΕΧΕΙ ΤΟ ΑΡΙΘΜΟ ΤΩΝ ΒΥΤΕΣ ΓΙΑ ΧΑΡΑΚΤΗΡΕΣ Α ΠΛΑΤΟΥΣ	ΑF

BC1D	ΕΠΙΣΤΡΕΦΕΙ ΣΤΟΝ ΗΛ ΤΗΝ ΔΙΕΥΘΥΝΣΗ ΤΟΥ PIXEL DE(X) HL(Y), ΜΕ ΤΗ «ΜΑΣΚΑ» ΣΤΟΝ C ΚΑΙ ΤΑ PIXELS ΑΝΑ BYTE -1 ΣΤΟΝ B.	AF DE
------	--	-------

ΟΙ ΕΠΟΜΕΝΕΣ 4 ΚΛΗΣΕΙΣ ΧΡΕΙΑΖΟΝΤΑΙ ΟΛΕΣ ΤΟ ΖΕΥΓΟΣ ΚΑΤΑΧΩΡΗΤΩΝ ΗΛ ΝΑ ΠΕΡΙΕΧΕΙ ΤΗ ΔΙΕΥΘΥΝΣΗ ΜΙΑΣ ΘΕΣΗΣ ΤΗΣ ΟΘΟΝΗΣ, ΚΑΙ ΘΑ ΕΠΙΣΤΡΕΨΕΙ ΤΟ ΑΠΟΤΕΛΕΣΜΑ ΤΟΥΣ ΣΤΟ ΖΕΥΓΟΣ ΗΛ. ΥΠΑΡΧΕΙ ΝΑ ΒΡΕΘΕΙΤΕ ΕΞΩ ΑΠΟ ΤΗΝ ΟΘΟΝΗ, ΓΕΓΟΝΟΣ ΠΟΥ ΠΡΕΠΕΙ ΝΑ ΕΛΕΓΧΕΤΑΙ ΚΑΙ ΝΑ ΑΠΟΦΕΥΓΕΤΑΙ.

BC20	ΔΕΞΙΑ 1 BYTE	AF
BC23	ΑΡΙΣΤΕΡΑ 1 BYTE	AF
BC26	ΚΑΤΩ 1 PIXEL	AF
BC29	ΠΑΝΩ 1 PIXEL	AF
BC38	ΤΟΠΟΘΕΤΗΣΕ ΤΑ ΧΡΩΜΑΤΑ ΤΟΥ BORDER ΣΤΟΥΣ B, C	AF BC DE HL
BC3E	ΤΟΠΟΘΕΤΗΣΕ ΤΑ FLASH PERIODS H, L	AF HL

Ο CASSETTE MANAGER

BC65	ΑΡΧΙΚΕΣ ΤΙΜΕΣ	AF BC DE HL
------	---------------	-------------

Ο cassette manager απαιτεί λεπτομερείς γνώσεις πριν χρησιμοποιηθεί, σαν αυτές που δίνονται στο Firmware Specification Manual, όπως συμβαίνει και με τον Sound Manager και τις εισαγωγές του Kernel. Προτείνεται οποιοσδήποτε χειρισμός καθέτως ή ήχου να γίνεται επιστρέφοντας στη BASIC, και κατόπιν να κάνετε τις απαραίτητες εργασίες χρησιμοποιώντας τη BASIC. Μετά μπορεί να γίνει ένα CALL και να ξαναπεράσουμε στο πρόγραμμα γλώσσας μηχανής. Παρατηρείστε πως αρχικά θα πρέπει να καλέσατε το πρόγραμμα γλώσσας μηχανής από τη BASIC, για να μπορείτε να επιστρέψετε στη BASIC. Δεν μπορείτε να χρησιμοποιήσετε την εντολή "RUN" για να φορτώσετε και να εκτελέσετε το πρόγραμμά σας γλώσσας μηχανής.

BD2B	ΣΤΕΛΝΕΙ ΤΟΝ ΧΑΡΑΚΤΗΡΑ ΤΟΥ ΟΠΟΙΟΥ Ο ΚΩΔΙΚΟΣ ΒΡΙΣΚΕΤΑΙ ΣΤΟΝ ΚΑΤΑΧΩΡΗΤΗ A, ΣΤΗ ΘΥΡΑ CENTRONICS (ΕΚΤΥΠΩΤΗΣ). ΑΝ Η ΑΠΟΣΤΟΛΗ ΤΟΥ ΧΑΡΑΚΤΗΡΑ ΔΕΝ ΓΙΝΕΙ ΕΠΙΤΥΧΩΣ ΜΕΤΑ ΑΠΟ 0.4 SECS ΕΚΤΕΛΕΙΤΑΙ ΕΝΑ RETURN ΜΕ ΤΟ CARRY FLAG «0» (RESET). ΑΝ ΤΟ CARRY FLAG ΕΙΝΑΙ «1» (SET) ΜΕ ΤΗΝ ΕΠΙΣΤΡΟΦΗ ΤΟΤΕ Η ΑΠΟΣΤΟΛΗ ΤΟΥ ΧΑΡΑΚΤΗΡΑ ΗΤΑΝ ΕΠΙΤΥΧΗΣ. ΤΟ BIT 7 ΑΓΝΟΕΙΤΑΙ.	AF
------	---	----

BD37	ΕΠΑΝΑΦΟΡΑ ΣΤΟ ΑΡΧΙΚΟ JUMP BLOCK	AF BC DE HL
------	---------------------------------	-------------

Οι παραπάνω ρουτίνες θα σας επιτρέψουν να έχετε πρόσβαση σε ορισμένες από τις χρησιμότερες λειτουργίες του firmware. Υπάρχουν κυριολεκτικά εκατοντάδες ακόμη διαθέσιμες ρουτίνες, μερικές από τις οποίες θα σας εξοικονομήσουν χρόνο και προσπάθεια ακόμη κι όταν χρησιμοποιείτε τις ρουτίνες που δίνονται εδώ. Το

Firmware Specification Manual δίνει πληρέστερες λεπτομέρειες για όλες τις ρουτίνες του firmware καθώς και μια γενική άποψη του hardware, και λεπτομερή εξέταση των τεχνικών που χρησιμοποιούνται από το firmware. Αν διαπιστώσετε ότι ενδιαφέρεστε σοβαρά για τον προγραμματισμό σε γλώσσα μηχανής δεν έχετε καλύτερο καταφύγιο από το να αγοράσετε το Firmware Specification Manual (SOFT 158) από την Amstrad.

ΚΥΚΛΟΦΟΡΟΥΝ ΑΠΟ ΤΟΝ ΔΕΚΕΜΒΡΙΟ 1985:

Το σύστημα δίσκων του Amstrad του Ian Sinclair

Εισχωρεί στη γνώση του συστήματος των δίσκων, περιγράφει το λειτουργικό σύστημα CP/M, τρόπους αρχειοθέτησης από την BASIC και περιέχει διάφορα UTILITIES όπως πως να ξεκλειδώσετε προστατευμένα προγράμματα κ.λπ.

Γλώσσα μηχανής για αρχάριους στον Amstrad του Steve Cramer

Απευθύνεται στον Αρχάριο που θέλει να μάθει την γλώσσα μηχανής του Amstrad. Περιγράφει τις αρχές προγραμματισμού σε γλώσσα μηχανής, τις οδηγίες του επεξεργαστού Z-80 κ.λπ.

Συστήματα αρχειοθέτησης και βάσεις δεδομένων για τον Amstrad. των A. P. Stephenson και D. J. Stephenson

Αναλύει τον τρόπο σχεδίασης των προγραμμάτων αρχειοθέτησης και βάσεων δεδομένων, με πλήρη και λεπτομερή παραδείγματα.

Όλα τα βιβλία αναφέρονται στους υπολογιστές CPC 464, CPC 664, CPC 6128

ΚΥΚΛΟΦΟΡΟΥΝ ΑΠΟ ΤΟΝ ΙΑΝΟΥΑΡΙΟ 1986:

Το Εγχειρίδιο του CP/M του Alan Miller

Το Εγχειρίδιο του MS-DOS του Allen King

Μάθετε εύκολα τον IBM-PC του Lean Scanlon

Pascal για αρχάριους του Maik James.

Αυτό το βιβλίο απευθύνεται σ' όλους τους χρήστες της Pascal, τεχνικούς, επαγγελματίες, φοιτητές αλλά και χομπίστες. Κάθε κεφάλαιο αναλύεται με πολύ σαφήνεια και απλό τρόπο. Επίσης υπάρχουν παραδείγματα αλλά και ερωτήσεις που σας βοηθούν να γράψετε πολύ σύντομα τα δικά σας προγράμματα στην Pascal.

LOGO για αρχάριους του Boris Allan.

Απευθύνεται κυρίως στους νεαρούς χρήστες που θέλουν να γνωρίσουν αυτή τη γλώσσα, όπως είναι οι κάτοχοι του Amstrad CPC 664, 6128 όπως και του Atari 520 ST.

ΚΥΚΛΟΦΟΡΟΥΝ:

Γνωρίστε το ZX-microdrive σας του Andrew Pennell

Έχει τόση σαφήνεια και τόσα πολλά παραδείγματα που το κάνουν εξίσου χρήσιμο τόσο για τον νεοφερμένο στη BASIC όσο και για τον έμπειρο προγραμματιστή

ΑΠΑΡΑΙΤΗΤΟ ΣΕ ΚΑΘΕ ΚΑΤΟΧΟ SPECTRUM

Δουλέψτε με τον Amstrad Χρήσιμα προγράμματα και υπορουτίνες του David Lawrence

Μια συλλογή από έξυπνα προγράμματα και υπορουτίνες που μπορούν να χρησιμοποιηθούν αυτούσιες στα προγράμματά σας όπως αποθήκευση δεδομένων, οικονομικά κ.λπ.

Μάθετε Basic με τον Amstrad του Wynford James

Το βιβλίο απευθύνεται στον νέο κάτοχο του Amstrad και τον εισάγει στον προγραμματισμό μεθοδικά και σταδιακά, χωρίς να έχει τις αδυναμίες του manual του υπολογιστή.

Να τι έγραψε το αγγλικό περιοδικό Personal Computer World για το αγγλικό manual: «Οι αρχάριοι μάλλον θα χρειασθούν ένα άλλο βιβλίο να τους μάθει Basic».

Basic για αρχάριους του A. P. Stephenson

Μεθοδικό και απλό απευθύνεται στον αρχάριο της Basic και τον οδηγεί στα μυστικά του προγραμματισμού.

Δουλέψτε με τον Commodore Χρήσιμα προγράμματα και υπορουτίνες του David Lawrence

Ένα βιβλίο με πολλά έτοιμα προγράμματα και υπορουτίνες που θα σας προσφέρουν μεγάλη βοήθεια στην κατασκευή των δικών σας προγραμμάτων. Περιλαμβάνει προγράμματα με αποθήκευση δεδομένων, οικονομικά, γραφήματα ακόμα και επεξεργασία κειμένου.

AMSTRAD CPC



MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA



<https://acpc.me/>