Συστήματα Μικροϋπολογιστών

Προγραμματισμός μΕ σε Συμβολική Γλώσσα

Εισαγωγή

Η ανάπτυξη ενσωματωμένου λογισμικού διαφέρει από την ανάπτυξη λογισμικού σε ένα workstation ή ένα PC γιατί πρέπει:

- Να λειτουργεί σε συγκεκριμένους ρυθμούς για να ικανοποιεί χρονικούς περιορισμούς
- Να χωράει σε συγκεκριμένο μέγεθος μνήμης
- Να ικανοποιεί συγκεκριμένες προδιαγραφές κατανάλωσης ισχύος

SNPD (Smart Negative Pressure Device)

- Software stack from drivers up to user application
- No RTOS (Real Time Operating System)
- C and assembly programming

SNPD real case: Pump control

Verification in complex lab tubing system with pressure meter and on healthy volunteer

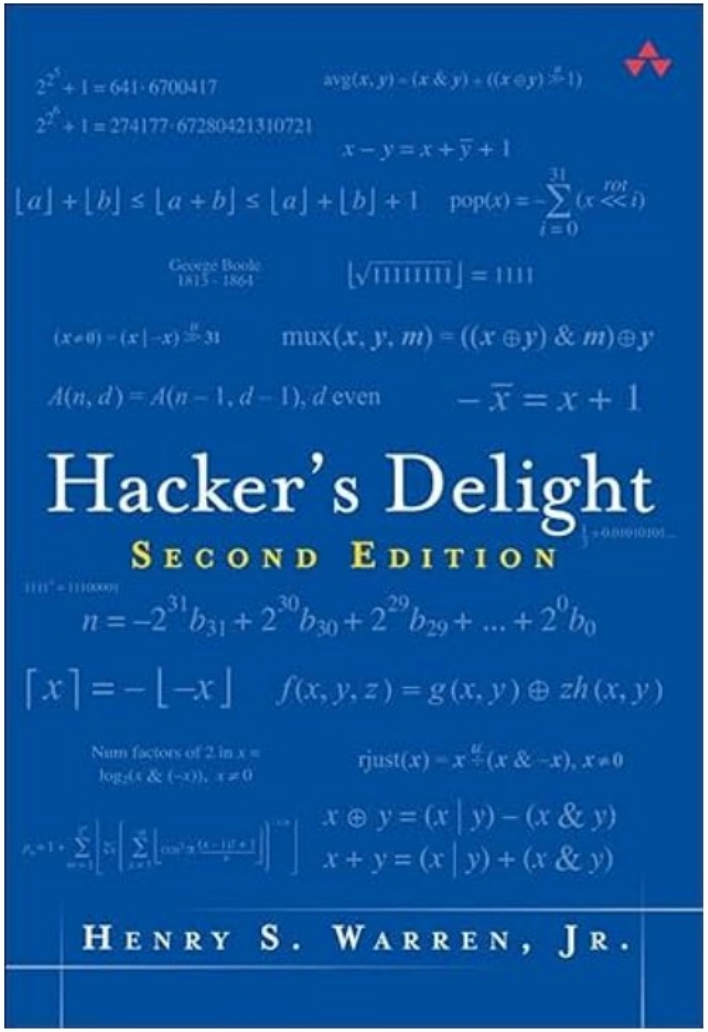
Αντιστοιχία γλώσσας υψηλού επιπέδου με συμβολική και με γλώσσα μηχανής

Γλώσσα Υψηλού Επιπέδου	Συμβολική Γλώσσα	HEX Κώδικας	Δυαδικός Κώδικας
A = A + K	LDA SUMA	3A	0011 1010
		22	0010 0010
		00	0000 0000
	MOV C,A	4F	0100 1111
	LDA SUMK	3A	0011 1010
		23	0010 0011
		00	0000 0000
	ADD C	81	1000 0001
	STA SUMA	32	0011 0010
		22	0010 0010
		00	0000 0000
	HLT	76	0111 0110

Variable instruction width
Intel CPU characteristic feature

Περιπτώσεις που απαιτείται η χρήση της Συμβολικής Γλώσσας

- Όταν ο προγραμματιστής χρειάζεται **πρόσβαση σε καταχωρητές του επεξεργαστή.**
- Στην εφαρμογή απαιτείται να πάρουμε την **μέγιστη απόδοση από τον επεξεργαστή** και το υλικό του υπολογιστή (π.χ. υψηλότερη ταχύτητα λειτουργίας, χαμηλή κατανάλωση ισχύος κλπ.).
- Χρειάζεται ο χειρισμός χρονικά κρίσιμων λειτουργιών καθώς και η επικοινωνία με τις συσκευές I/O.
- Επίσης ο χειρισμός των διακοπών απαιτεί assembly.
- Περαιτέρω, η κωδικοποίηση απευθείας στην Assembly μπορεί μερικές φορές (αλλά όχι πάντα) να οδηγήσει σε μικρότερου μεγέθους κώδικα.
- Ο μΕ ή ο μικροελεγκτής μπορεί να μη διαθέτει μεταγλωττιστή (ή και να διαθέτει δεν είναι καθόλου αποδοτικός και να μην χρησιμοποιείται).


Hacker's Delight
SECOND EDITION
HENRY S. WARREN, JR.

Modern use cases of Assembly-Level (Binary) Optimization

Meta

Research Publications Programs Datasets Careers

BOLT: A Practical Binary Optimizer for Data Centers and Beyond

International Symposium on Code Generation and Optimization (CGO)

Google Research Who we are Research areas Our work Programs & events Careers Blog

Home > Publications >

AutoFDO: Automatic Feedback-Directed Optimization for Warehouse-Scale Applications

Dehao Chen · David Xinliang Li · Tipp Moseley ·

CGO 2016 Proceedings of the 2016 International Symposium on Code Generation and Optimization, ACM, New York, NY, USA, pp. 12-23

Google Scholar Copy Bibtex

MAO

Ανάπτυξη Λογισμικού

Σχεδίαση

- Αλγόριθμος

Κώδικας

- Εντολές

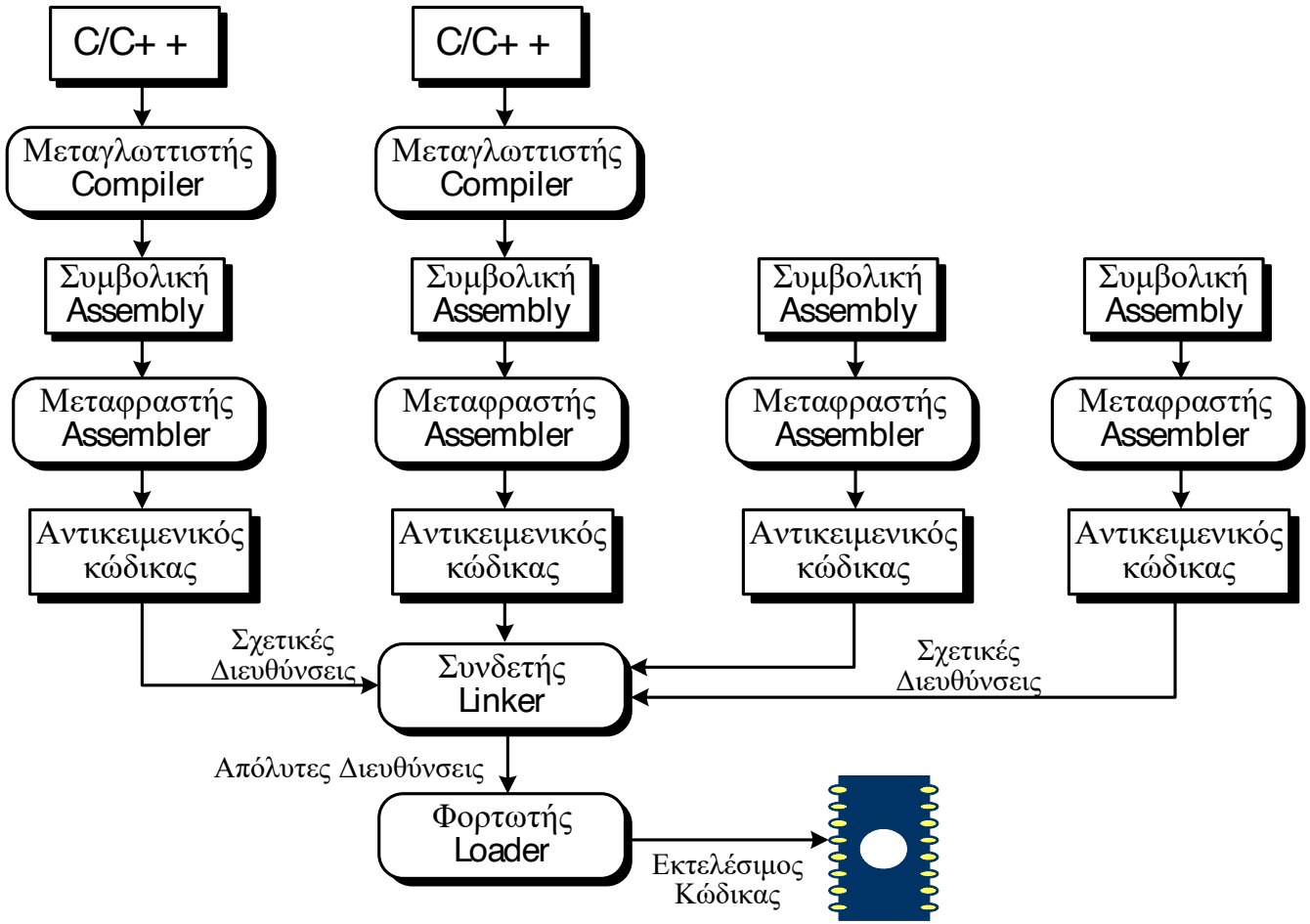
Μετάφραση

- Πηγαίος → Αντικειμενικός κώδικας

Έλεγχος – Διόρθωση

- Το τελικό αποτέλεσμα είναι σωστό;

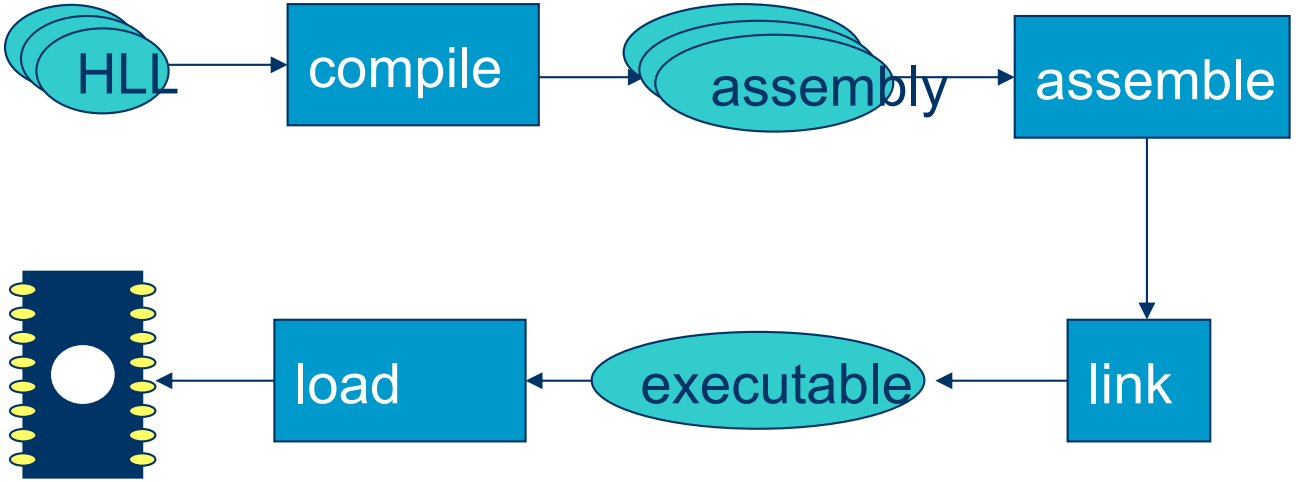
Διαδικασία Μετάφρασης



Assembly και Linking

Η μετάφραση assembly και το Linking είναι τα τελευταία βήματα στην διαδικασία μεταγλώττισης του λογισμικού.

Μετατρέπουν μια λίστα εντολών σε έναν πίνακα από bits που θα αποθηκευτούν στην μνήμη του ενσωματωμένου συστήματος.



Προγράμματα Πολλαπλών Αρχείων

Το λογισμικό μπορεί να αποτελείται από πολλά αρχεία

Οι διευθύνσεις γίνονται πιο συγκεκριμένες κατά τη διάρκεια της επεξεργασίας:

- Οι σχετικές διευθύνσεις μετρούνται σε σχέση με την αρχή (στην μνήμη) του προγράμματος που την χρησιμοποιεί
- Οι απόλυτες διευθύνσεις μετρούνται από την αρχή του χώρου μνήμης της CPU

Assemblers

Οι βασικές λειτουργίες των assemblers:

- Παραγωγή δυαδικών από τις συμβολικές εντολές
- Μετάφραση των ετικετών (labels) σε διευθύνσεις
- Επεξεργασία ψευδοεντολών

Η ύπαρξη των ετικετών επιβάλλει την επεξεργασία 2 περασμάτων

- Στο πρώτο πέρασμα καθορίζεται η διεύθυνση κάθε ετικέτας
- Στο δεύτερο πέρασμα μετατρέπονται οι συμβολικές εντολές σε δυαδικές χρησιμοποιώντας τις διευθύνσεις από το πρώτο πέρασμα

Παραγωγή Symbol Table

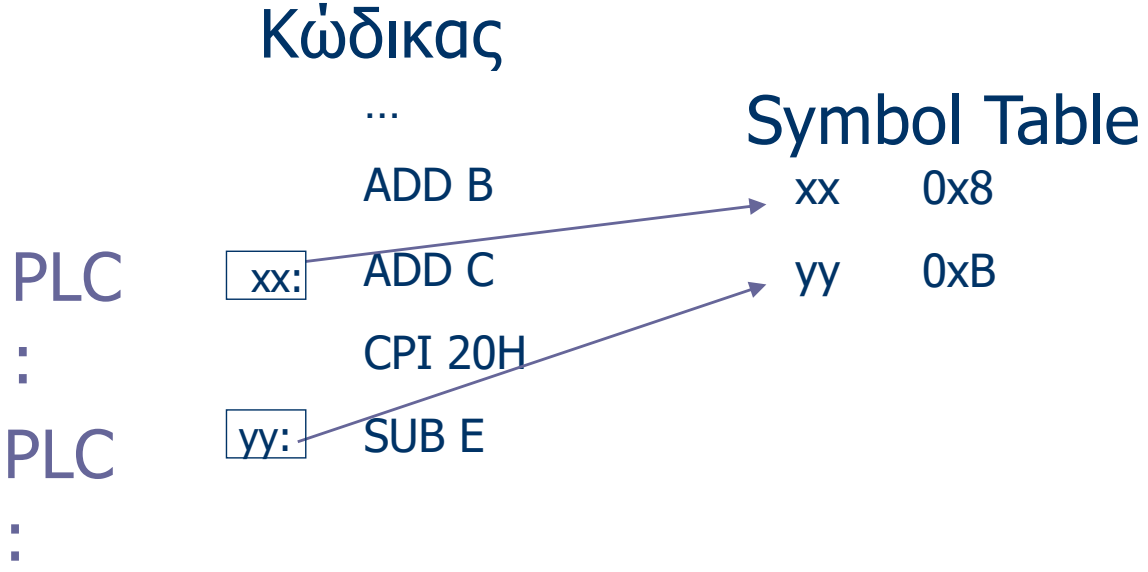
Καθώς ο assembler μεταφράζει ένα πρόγραμμα, διατηρεί έναν πίνακα συμβόλων (symbol table)

- < ετικέτα, διεύθυνση >
- Όταν ορίζεται μια ετικέτα, αυτή η ετικέτα, μαζί με την τρέχουσα τιμή του Μετρητή Θέσης (Program Location Counter, PLC), αποθηκεύονται στον πίνακα συμβόλων
- Όταν χρησιμοποιείται μια ετικέτα, ο assembler αναζητά στον πίνακα συμβόλων για να βρει την αντίστοιχη διεύθυνση Μετρητής Θέσης (PLC) — παρακολουθεί τη διεύθυνση της τρέχουσας εντολής κατά τη διαδικασία παραγωγής του κώδικα assembly
- **Ο Program Location Counter δεν είναι ο μετρητής προγράμματος (PC).** Ο μετρητής τοποθεσίας είναι μια μεταβλητή που χρησιμοποιείται κατά τη μετάφραση του προγράμματος. Ο PC είναι ένας καταχωρητής που χρησιμοποιείται κατά την εκτέλεση του προγράμματος

Παραγωγή Symbol Table (cont.)

Χρήση του Program Location Counter (PLC) για τον καθορισμό της διεύθυνσης κάθε θέσης.

Στο πρώτο πέρασμα μετράει ο PLC και αποθηκεύει τις ετικέτες στο Symbol Table.



Linking

Το ενσωματωμένο λογισμικό αναπτύσσεται σε πολλά αρχεία. Κάθε ένα από αυτά μετατρέπεται σε «γλώσσα μηχανής».

Ο Linker συνδέει όλα αυτά τα μεταφρασμένα αρχεία σε ένα μεγάλο κάνοντας τις αντίστοιχες μετατροπές στα αρχεία αυτά.

Ο Linker λειτουργεί σε 2 φάσεις:

- Αφού τοποθετήσει όλα τα αρχεία στη σειρά όπως θα μπουν στην μνήμη (user defined) καθορίζει την απόλυτη διεύθυνση που θα αρχίζει κάθε αρχείο.
- Έπειτα, συνενώνει όλα τα Symbol Tables σε ένα μεγάλο και μετατρέπει τις σχετικές διευθύνσεις κάθε αρχείου σε απόλυτες.

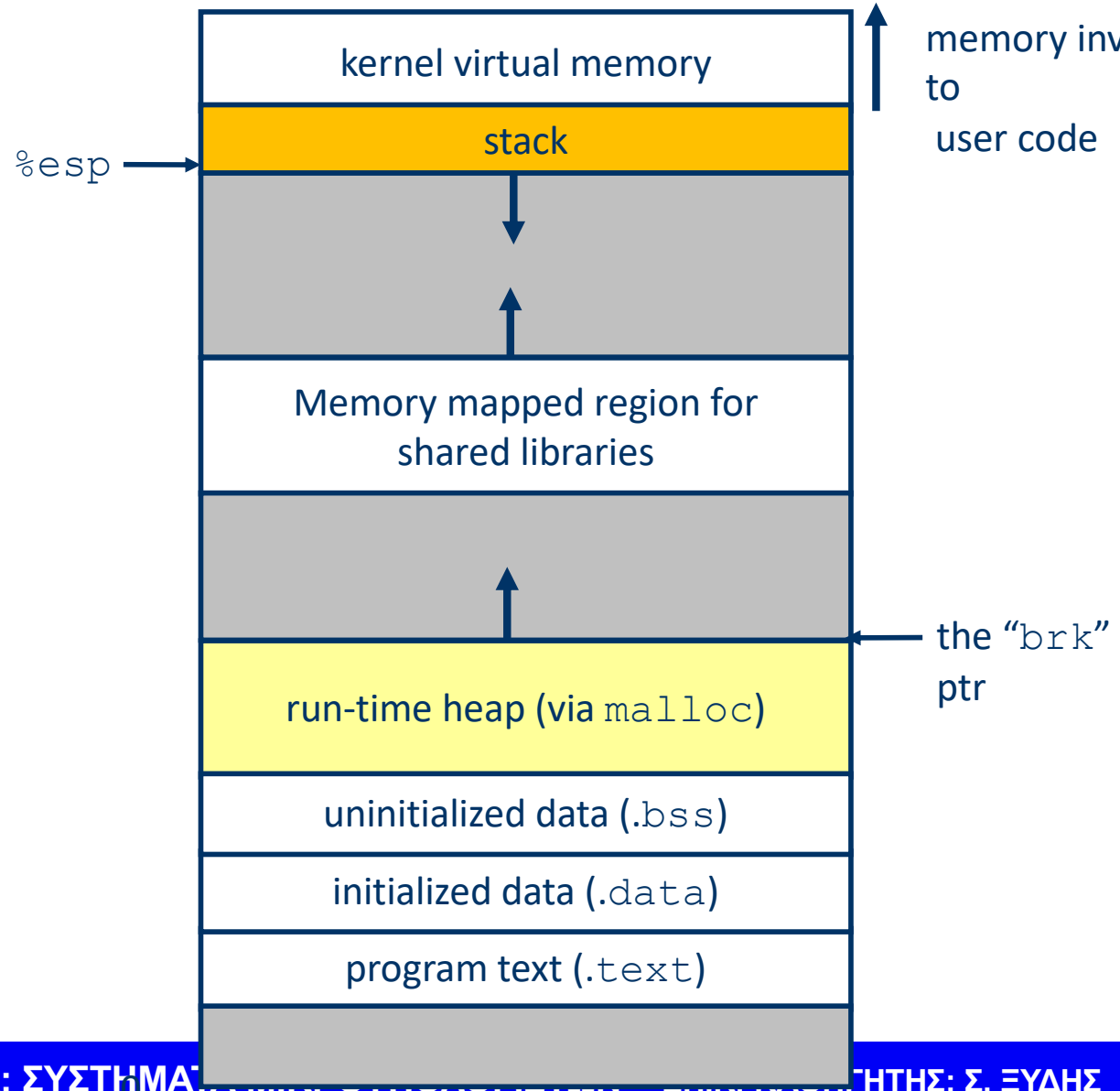
Ο έλεγχος ποια αρχεία τοποθετούνται και σε ποιο σημείο της μνήμης είναι πολύ βασικός στα ενσωματωμένα συστήματα.

- Κώδικας που επεξεργάζεται interrupts πρέπει να είναι σε συγκεκριμένη θέση μνήμης για να μπορεί να τον χρησιμοποιήσει ο μικροεπεξεργαστής.

Loader

- Ο Loader φορτώνει το ολοκληρωμένο πρόγραμμα στη μνήμη όπου μπορεί να εκτελεστεί
  - Φορτώνει τμήματα κειμένου και δεδομένων στη μνήμη σε καθορισμένη θέση
  - Επιστρέφει την τιμή της διεύθυνσης έναρξης στο λειτουργικό σύστημα (— τη διεύθυνση της πρώτης εντολής που θα εκτελεστεί)
- Εναλλακτικές λύσεις στη φόρτωση
  - Απόλυτος φορτωτής (Absolute Loader) — φορτώνει το εκτελέσιμο αρχείο σε σταθερή θέση
  - Relocatable Loader — μπορεί να φορτώσει το πρόγραμμα σε μια αυθαίρετη θέση μνήμης
    - Ο assembler και ο linker υποθέτουν ότι το πρόγραμμα θα ξεκινήσει στη θέση 0
    - Όταν το πρόγραμμα διαβάζεται από τον Loader, ο Loader τροποποιεί όλες τις διευθύνσεις προσθέτοντας την πραγματική θέση έναρξης σε αυτές τις διευθύνσεις

Address space after Loading (obj code architecture) for x86



Γλώσσα Assembly (Συμβολική)

Μνημονικές συντμήσεις των εντολών

Συμβολικά ονόματα ρουτίνων, διευθύνσεων

Γενικός τύπος Εντολής Assembly

label (:)	*Mnemonic*	*Operand(s)*	*; Comments*
	LDA	Label-1	
Label-2:	ADD	oper	
	MOV	destination, source	
	MVI	destination, data	
	ADI	data	
	JMP	Label-2	

Μακροεντολές και Υποπρογράμματα

Οδηγίες Συμβολομεταφραστή

Οι οδηγίες συντάσσονται όπως και οι εντολές, αλλά έχουν σημαντικές διαφορές από αυτές. Πρώτον, δεν ανήκουν στο σύνολο των εντολών του μΕ, (μικροεπεξεργαστή) και δεύτερον, δεν μεταφράζονται σε κώδικα. Η χρησιμότητά τους έγκειται στο να δίνουν κάποιες πληροφορίες προς τον assembler, όταν αυτός δημιουργεί τον κώδικα.

Οδηγίες του Assembler (Ψευδο-εντολές)

ORG (Origin)

Διεύθυνση επόμενης εντολής

END

Τέλος κώδικα

DB (1 byte), DW (2 byte), DD (4 byte)

Π.χ. DB 084H ➔ 84

EQU Symbol = Constant

Π.χ. LOOP EQU 0100H

SET: ίδια με την **EQU** μόνο που μπορεί να αλλάξει η τιμή του συμβολικού ονόματος

DS (define storage): Φυλάσσεται ένας αριθμός από θέσεις μνήμης για αποθήκευση δεδομένων

ORG (origin)

Σύνταξη: **ORG** έκφραση

Η ψευδοεντολή αυτή ακολουθείται από έναν αριθμό που δείχνει την απόλυτη θέση μνήμης όπου πρέπει να τοποθετηθεί η πρώτη εκτελέσιμη εντολή του προγράμματος. Αν δεν υπάρχει η ψευδοεντολή **ORG** πριν την πρώτη εντολή του προγράμματος, τότε η εντολή τοποθετείται στη θέση 0 της μνήμης. Μπορούμε να έχουμε περισσότερες από μία ψευδοεντολές **ORG,** οι οποίες θα δείχνουν τη θέση της μνήμης για την πρώτη εκτελέσιμη εντολή που τις ακολουθεί. Για παράδειγμα, στις παρακάτω εντολές ενός προγράμματος έχουμε:

```
ORG 0100H       ; Ο κώδικας των εντολών αυτών αρχίζει από τη θέση μνήμης 0100H
    ΕΝΤΟΛΕΣ ASSEMBLY
ORG 0200        ; Ο κώδικας των εντολών αυτών αρχίζει από τη θέση μνήμης 0200H
    ΕΝΤΟΛΕΣ ASSEMBLY
```

END

Σύνταξη: **END**

Η ψευδοεντολή αυτή δηλώνει το φυσικό τέλος του προγράμματος και πρέπει να είναι μοναδική για κάθε πρόγραμμα. Όταν ο assembler αναγνωρίσει το φυσικό τέλος του προγράμματος, αρχίζει τη δημιουργία του κώδικα και (πιθανώς) της λίστας του πηγαίου προγράμματος.

DB, DW, DD

DB, DW, DD

define byte, define word, define double word

Σύνταξη: <επιγραφή:> **DB** λίστα

<επιγραφή:> **DW** λίστα

<επιγραφή:> **DD** λίστα

Με τον όρο λίστα εννοούμε τιμές δεδομένων ή εκφράσεις.

Οι ψευδοεντολές αυτές έχουν σαν αποτέλεσμα να αποθηκεύει ο assembler σε μία ή περισσότερες θέσεις μνήμης τις συγκεκριμένες τιμές αρχίζοντας από τη διεύθυνση της επιγραφής. Οι θέσεις αυτές μπορεί να είναι ομάδες από bytes (8 bits), λέξεις (16 bits) ή διπλές λέξεις (32 bits). Ο όρος έκφραση εδώ αναφέρεται σε μια ή περισσότερες αριθμητικές ή λογικές εκφράσεις. Επίσης στην περίπτωση της ψευδοεντολής DB μπορούμε στη λίστα να περιλάβουμε και χαρακτήρες σε εισαγωγικά. Ο assembler αναλαμβάνει την αντικατάστασή τους σε ASCII μορφή.

Δημιουργία στατικών μεταβλητών κώδικα, π.χ. πίνακες, ακέραιοι κτλ με αρχικοποίηση

DS (define storage)

Σύνταξη: <επιγραφή:> **DS** έκφραση

Με την ψευδοεντολή αυτή φυλάσσεται ένας αριθμός από θέσεις μνήμης για αποθήκευση δεδομένων. Η πρώτη από τις θέσεις μπορεί να αναφερθεί με τη συμβολική επιγραφή. Προσοχή πρέπει να δώσουμε στις δύο τελείες μετά την επιγραφή. Ο αριθμός των bytes που δεσμεύονται είναι ίσος με την τιμή της έκφρασης.

Για παράδειγμα αν θέλουμε να κρατήσουμε δέκα bytes για αποθήκευση δεδομένων με το όνομα DEKA δηλώνουμε:

```
DEKA:    DS 10
```

Δημιουργία στατικών μεταβλητών κώδικα, π.χ. πίνακες, ακέραιοι κτλ χωρίς αρχικοποίηση

EQU (equate)

Σύνταξη: όνομα **EQU** έκφραση

Με την ψευδοεντολή αυτή ένα συμβολικό όνομα αντιστοιχείται με μια σταθερά ή διεύθυνση ή γενικότερα έκφραση. Η εντολή αυτή είναι ισοδύναμη με την εντολή const της Pascal ή #define της C και έχει σαν αποτέλεσμα όποτε συναντάται το συμβολικό όνομα στο πρόγραμμα να χρησιμοποιείται η έκφραση στην οποίαν ισοδυναμεί. Πρέπει να σημειώσουμε πως εδώ το όνομα δεν είναι επιγραφή και δεν ακολουθείται από δύο τελείες (:) αν και βρίσκεται στο πεδίο της επιγραφής. Επίσης η τιμή του ονόματος δεν μπορεί να αλλάξει με το πρόγραμμα.

```
Sunday EQU 1
Monday EQU Sunday + 1  <-> Monday EQU 2
```

SET

Σύνταξη: όνομα **SET** έκφραση

Έχει τα ίδια αποτελέσματα με την **EQU**. Η διαφορά της βρίσκεται στο ότι μπορεί η τιμή του συμβολικού ονόματος να αλλάξει μέσα στο πηγαίο πρόγραμμα. Έτσι πολλές **SET** μπορεί να αναφέρονται στο ίδιο συμβολικό όνομα μέσα στο ίδιο πρόγραμμα.

```
Sunday EQU 1
Monday SET Sunday + 1
Tuesday EQU Sunday + 2
…
Monday SET Tuesday - 1
```

IF, ENDIF

Σύνταξη: **IF** έκφραση

εντολές assembly

ENDIF

Εάν η τιμή της έκφρασης είναι 0 οι εντολές έως την **ENDIF** **αγνοούνται** ενώ εάν είναι 1 οι εντολές μεταφράζονται από τον assembler.

- Υπάρχει ακόμη ένας σημαντικός αριθμός από ψευδοεντολές/οδηγίες που όμως αλλάζουν από υλοποίηση σε υλοποίηση. Μπορούμε να αναφέρουμε ενδεικτικά γιά τον MASM 8086 τις **SEGMENT/ENDS, ASSUME, GROUP, LABEL** και **PROC/ENDP,** που θα αναφερθούν και στο κεφάλαιο 9 του παρόντος βιβλίου.

- *Τελειώνοντας αναφέρουμε πως τις ψευδοεντολές (pseudo instructions) μπορούμε να συναντήσουμε στην βιβλιογραφία και ως assembler directives, nongenerative instructions ή declaratives.*

Πολλαπλασιασμός – Διάγραμμα Ροής

Multiplicand	1011	11
Multiplier	1101	13
	1011	
	0000	Partial Products
	1011	
	1011	
Product	10001111	143

Mul N x N → Product 2N

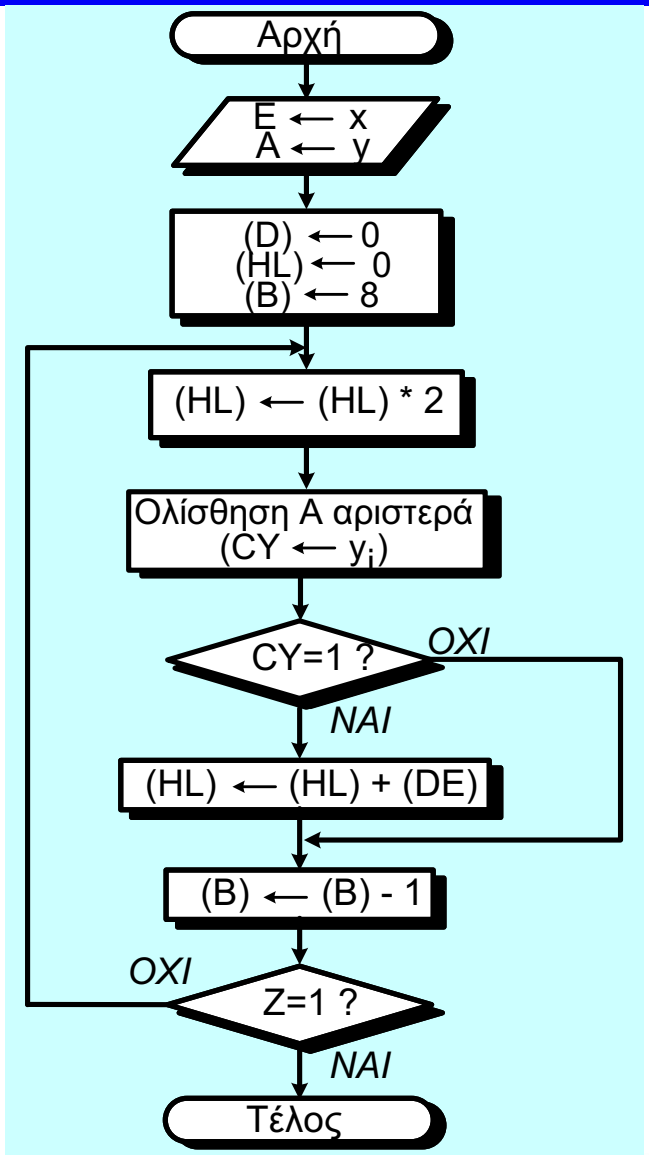


Table 6. Instruction Set Summary

Mnemonic	Instruction Code D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	Operations Description
MOVE, LOAD, AND STORE		
MOVr1 r2	0 1 D D D S S S	Move register to register
MOV M,r	0 1 1 1 0 S S S	Move register to memory
MOV r,M	0 1 D D D 1 1 0	Move memory to register
MVI r	0 0 D D D 1 1 0	Move immediate register
MVI M	0 0 1 1 0 1 1 0	Move immediate memory
LXI B	0 0 0 0 0 0 0 1	Load immediate register Pair B & C
LXI D	0 0 0 1 0 0 0 1	Load immediate register Pair D & E
LXI H	0 0 1 0 0 0 0 1	Load immediate register Pair H & L
STAX B	0 0 0 0 0 0 1 0	Store A indirect
STAX D	0 0 0 1 0 0 1 0	Store A indirect
LDAX B	0 0 0 0 1 0 1 0	Load A indirect
LDAX D	0 0 0 1 1 0 1 0	Load A indirect
STA	0 0 1 1 0 0 1 0	Store A direct
LDA	0 0 1 1 1 0 1 0	Load A direct
SHLD	0 0 1 0 0 0 1 0	Store H & L direct
LHLD	0 0 1 0 1 0 1 0	Load H & L direct
XCHG	1 1 1 0 1 0 1 1	Exchange D & E, H & L Registers
STACK OPS		
PUSH B	1 1 0 0 0 1 0 1	Push register Pair B & C on stack
PUSH D	1 1 0 1 0 1 0 1	Push register Pair D & E on stack
PUSH H	1 1 1 0 0 1 0 1	Push register Pair H & L on stack
PUSH PSW	1 1 1 1 0 1 0 1	Push A and Flags on stack
POP B	1 1 0 0 0 0 0 1	Pop register Pair B & C off stack
POP D	1 1 0 1 0 0 0 1	Pop register Pair D & E off stack
POP H	1 1 1 0 0 0 0 1	Pop register Pair H & L off stack
POP PSW	1 1 1 1 0 0 0 1	Pop A and Flags off stack
XTHL	1 1 1 0 0 0 1 1	Exchange top of stack, H & L
SPHL	1 1 1 1 1 0 0 1	H & L to stack pointer
LXI SP	0 0 1 1 0 0 0 1	Load immediate stack pointer
INX SP	0 0 1 1 0 0 1 1	Increment stack pointer
DCX SP	0 0 1 1 1 0 1 1	Decrement stack pointer
JUMP		
JMP	1 1 0 0 0 0 1 1	Jump unconditional
JC	1 1 0 1 1 0 1 0	Jump on carry
JNC	1 1 0 1 0 0 1 0	Jump on no carry
JZ	1 1 0 0 1 0 1 0	Jump on zero
JNZ	1 1 0 0 0 0 1 0	Jump on no zero
JP	1 1 1 1 0 0 1 0	Jump on positive
JM	1 1 1 1 1 0 1 0	Jump on minus
JPE	1 1 1 0 1 0 1 0	Jump on parity even
JPO	1 1 1 0 0 0 1 0	Jump on parity odd
PCHL	1 1 1 0 1 0 0 1	H & L to program counter
CALL		
CALL	1 1 0 0 1 1 0 1	Call unconditional
CC	1 1 0 1 1 1 0 0	Call on carry
CNC	1 1 0 1 0 1 0 0	Call on no carry

Mnemonic	Instruction Code D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	Operations Description
CZ	1 1 0 0 1 1 0 0	Call on zero
CNZ	1 1 0 0 0 1 0 0	Call on no zero
CP	1 1 1 1 0 1 0 0	Call on positive
CM	1 1 1 1 1 1 0 0	Call on minus
CPE	1 1 1 0 1 1 0 0	Call on parity even
CPO	1 1 1 0 0 1 0 0	Call on parity odd
RETURN		
RET	1 1 0 0 1 0 0 1	Return
RC	1 1 0 1 1 0 0 0	Return on carry
RNC	1 1 0 1 0 0 0 0	Return on no carry
RZ	1 1 0 0 1 0 0 0	Return on zero
RNZ	1 1 0 0 0 0 0 0	Return on no zero
RP	1 1 1 1 0 0 0 0	Return on positive
RM	1 1 1 1 1 0 0 0	Return on minus
RPE	1 1 1 0 1 0 0 0	Return on parity even
RPO	1 1 1 0 0 0 0 0	Return on parity odd
RESTART		
RST	1 1 A A A 1 1 1	Restart
INPUT/OUTPUT		
IN	1 1 0 1 1 0 1 1	Input
OUT	1 1 0 1 0 0 1 1	Output
INCREMENT AND DECREMENT		
INR r	0 0 D D D 1 0 0	Increment register
DCR r	0 0 D D D 1 0 1	Decrement register
INR M	0 0 1 1 0 1 0 0	Increment memory
DCR M	0 0 1 1 0 1 0 1	Decrement memory
INX B	0 0 0 0 0 0 1 1	Increment B & C registers
INX D	0 0 0 1 0 0 1 1	Increment D & E registers
INX H	0 0 1 0 0 0 1 1	Increment H & L registers
DCX B	0 0 0 0 1 0 1 1	Decrement B & C
DCX D	0 0 0 1 1 0 1 1	Decrement D & E
DCX H	0 0 1 0 1 0 1 1	Decrement H & L
ADD		
ADD r	1 0 0 0 0 S S S	Add register to A
ADC r	1 0 0 0 1 S S S	Add register to A with carry
ADD M	1 0 C 0 0 1 1 0	Add memory to A
ADC M	1 0 0 0 1 1 1 0	Add memory to A with carry
ADI	1 1 0 0 0 1 1 0	Add immediate to A
ACI	1 1 0 0 1 1 1 0	Add immediate to A with carry
DAD B	0 0 0 0 1 0 0 1	Add B & C to H & L
DAD D	0 0 0 1 1 0 0 1	Add D & E to H & L
DAD H	0 0 1 0 1 0 0 1	Add H & L to H & L
DAD SP	0 0 1 1 1 0 0 1	Add stack pointer to H & L
SUBTRACT		
SUB r	1 0 0 1 0 S S S	Subtract register from A
SBB r	1 0 0 1 1 S S S	Subtract register from A with borrow
SUB M	1 0 0 1 0 1 1 0	Subtract memory from A
SBB M	1 0 0 1 1 1 1 0	Subtract memory from A with borrow
SUI	1 1 0 1 0 1 1 0	Subtract immediate from A
SBI	1 1 0 1 1 1 1 0	Subtract immediate from A with borrow

DAD B	0	0	0	0	1	0	0	1	Add B & C to H & L
DAD D	0	0	0	1	1	0	0	1	Add D & E to H & L
DAD H	0	0	1	0	1	0	0	1	Add H & L to H & L
DAD SP	0	0	1	1	1	0	0	1	Add stack pointer to H & L

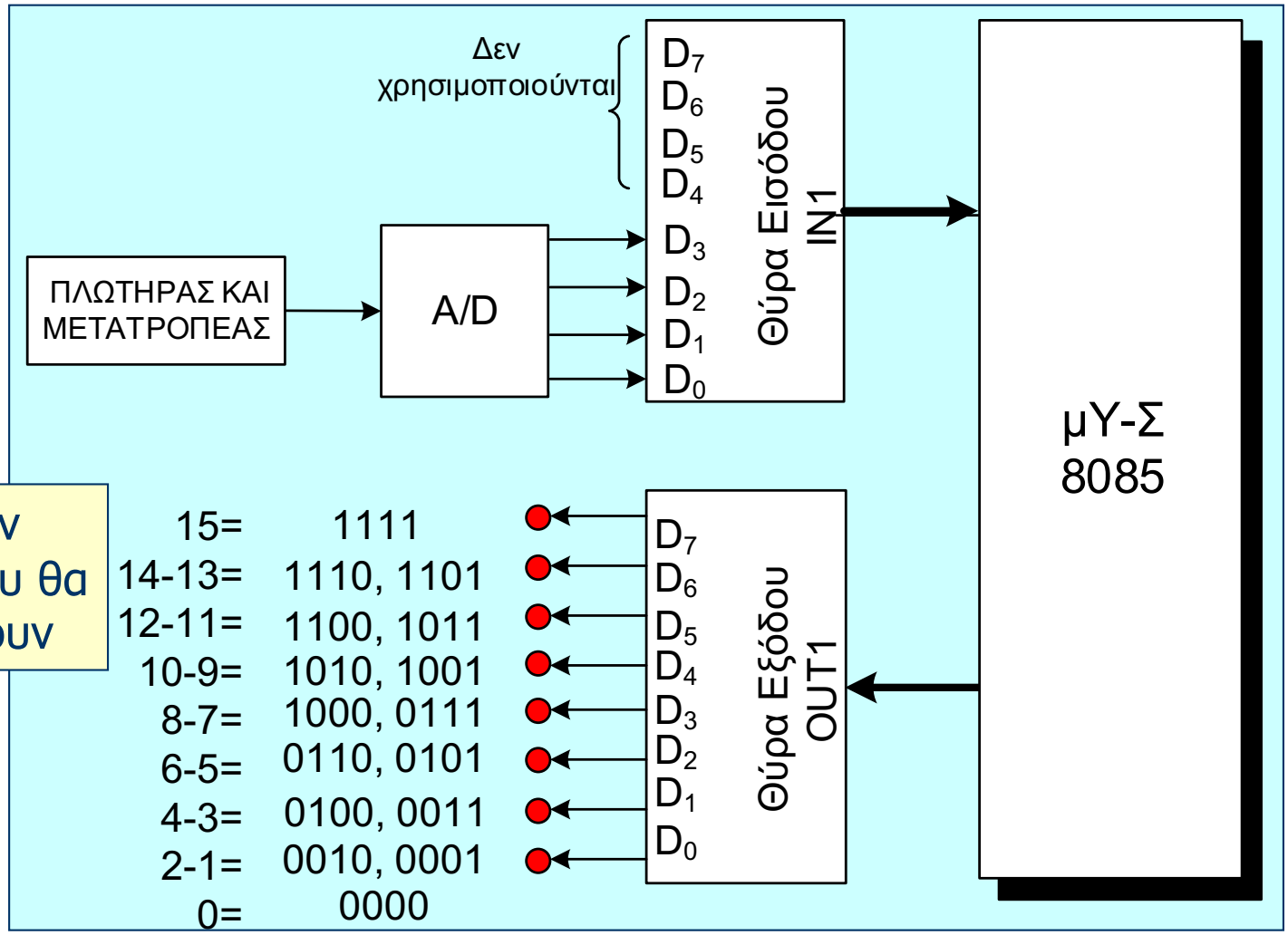
Πολλαπλασιασμός – Κώδικας

Διεύθυνση	*Obj. Code*	*Πηγαίο πρόγρ.*		*Σχόλια*
			ORG 0100H	
0100	16 00		MVI D,0	; (DE) = X
0102	21 00 00		LXI H,0	
0105	06 08		MVI B,8	; B = Μετρητής
0107	29	MULT:	DAD H	; Γινόμενο * 2
0108	17		RAL	; Ολίσθηση ←
0109	D2 **0D 01**		JNC SKIP	; CY=1?
010C	19		DAD D	; Πρόσθεσέ τον X<<Bit_{Rank}
010D	05	SKIP:	DCR B	; Αλλιώς μόνο B-1
010E	C2 **07 01**		JNZ MULT	; Τελείωσαν τα 8 bits
0111	76		HLT	; Τέλος

Παράδειγμα: Ενδείκτης Στάθμης Υγρού

Ζητείται η σχεδίαση ενός ενδείκτη στάθμης υγρού με φωτεινές στιγμές. Δίδεται ότι ο πλωτήρας με τον μετατροπέα παράγουν τάση ανάλογη της στάθμης. Επίσης δίδεται A/D των 4-bit που συνδέεται στην πόρτα εισόδου IN1. Τέλος 8 φωτεινές στιγμές από LEDs οδηγούνται από την πόρτα εξόδου OUT1.

Ενδείκτης Στάθμης Υγρού: Διάγραμμα Συστήματος



Αντιστοιχία τιμών και των LEDs που θα πρέπει να ανάψουν

15=	1111
14-13=	1110, 1101
12-11=	1100, 1011
10-9=	1010, 1001
8-7=	1000, 0111
6-5=	0110, 0101
4-3=	0100, 0011
2-1=	0010, 0001
0=	0000

Ενδείκτης Στάθμης Υγρού: Κώδικας (1)

Διεύθυνση	*Obj. Code*	*line*	*Label*	*Mnemonic*
0000		1	IN1	EQU 10H
0000		2	OUT1	EQU 20H
0000		3		ORG 0000H
0000	DB 10	4	LOOP:	IN IN1
0002	E6 0F	5		ANI 0FH
0004	5F	6		MOV E,A
0005	16 00	7		MVI D,0
0007	21 11 00	8		LXI H, TABLE
000A	19	9		DAD D
000B	7E	10		MOV A,M
000C	D3 20	11		OUT OUT1
000E	C3 00 00	12		JMP LOOP

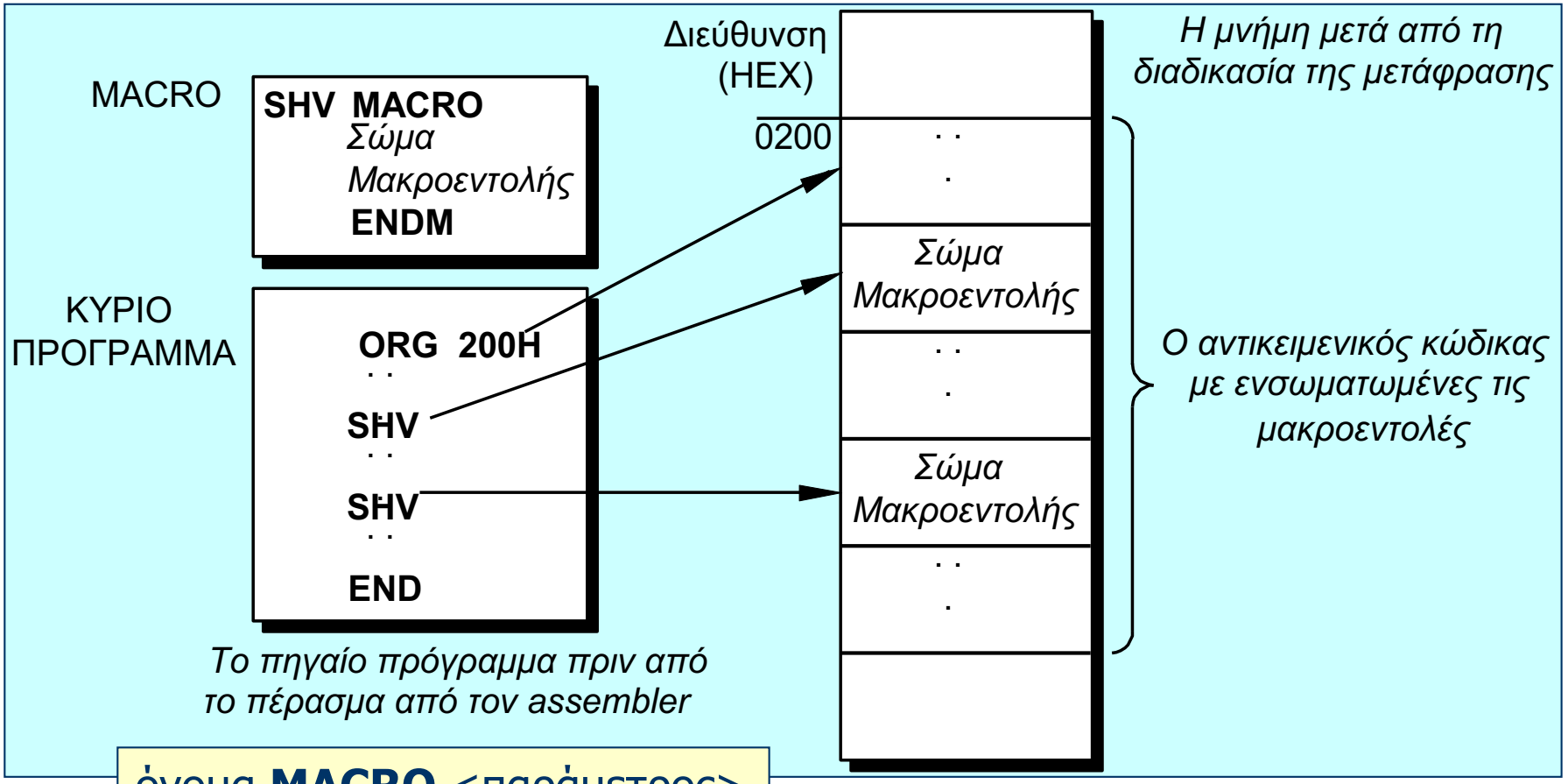
Why no obj. code here?

DE <- Data

Ενδείκτης Στάθμης Υγρού: Κώδικας (2)

Διεύθ.	*Obj. Code*	*line*	*Label*	*Mnemonic*
0011	00	15	TABLE:	DB 00H, 01H, 01H, 03H, 03H, 07H, 07H, 0FH
0012	01			
0013	01			
0014	03			
0015	03			
0016	07			
0017	07			
0018	0F			
0019	0F	16		DB 0FH, 1FH, 1FH, 3FH, 3FH, 7FH, 7FH, 0FFH
001A	1F			
001B	1F			
001C	3F			
001D	3F			
001E	7F			
001F	7F			
0020	FF			
		17		END

Μακροεντολές



όνομα **MACRO** <παράμετρος>

Σώμα της macro

ENDM

Παράδειγμα Μακροεντολής

Υλοποιήστε μια μακροεντολή, SWAP Q,R που εναλλάσσει τα περιεχόμενα οποιονδήποτε δύο καταχωρητών γενικού σκοπού B, C, D, E, H και L. Η εκτέλεση της μακροεντολής δεν πρέπει να επηρεάζει τα περιεχόμενα των υπολοίπων καταχωρητών που δεν μετέχουν στην εναλλαγή.

SWAP MACRO

```
SWAP MACRO Q,R
    PUSH PSW          ; στοίβα←A, F
    MOV A,Q
    MOV Q,R
    MOV R,A
    POP PSW           ; A, F←στοίβα
ENDM
```

Για παράδειγμα η εντολή SWAP H,B
αντικαθίσταται από το παρακάτω σύνολο εντολών:

```
    PUSH PSW
    MOV A,H
    MOV H,B
    MOV B,A
    POP PSW
```

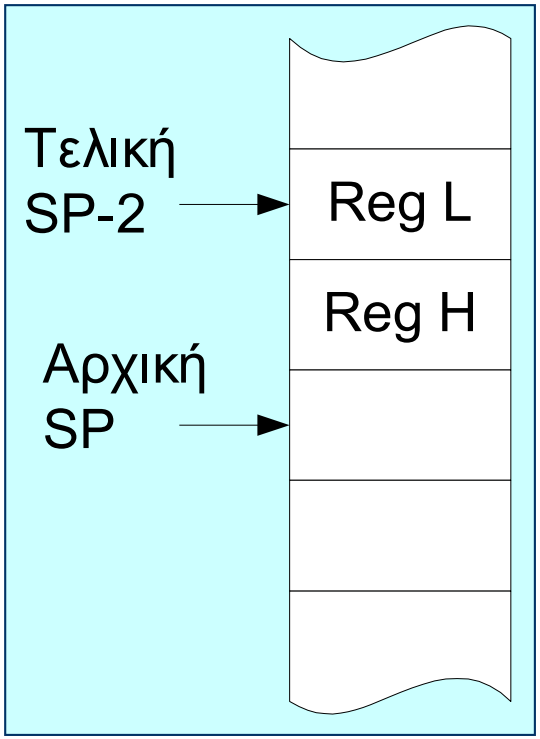
Εντολές στοίβας - PUSH

PUSH Reg Pair

((SP) - 1) ← (Reg H)

((SP) - 2) ← (Reg L)

(SP) ← (SP) - 2



	Reg H	Reg L
PUSH B	B	C
PUSH D	D	E
PUSH H	H	L
PUSH PSW	A	Flags

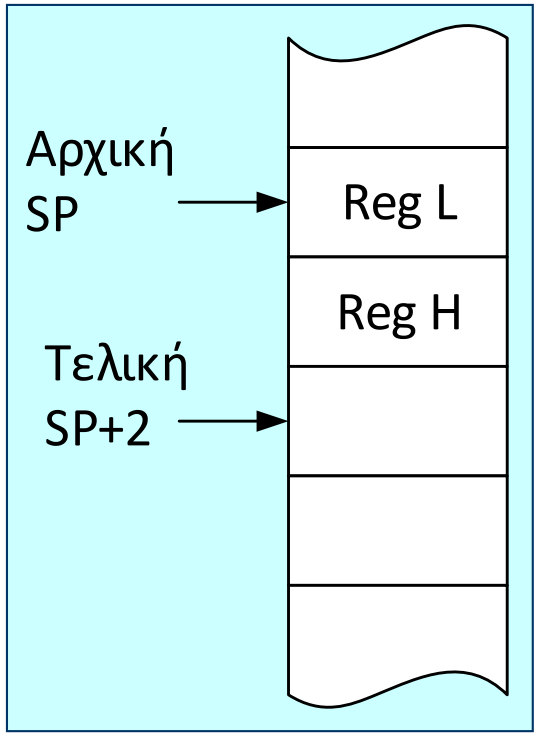
Εντολές στοίβας - POP

POP Reg Pair

(Reg L) ← ((SP))

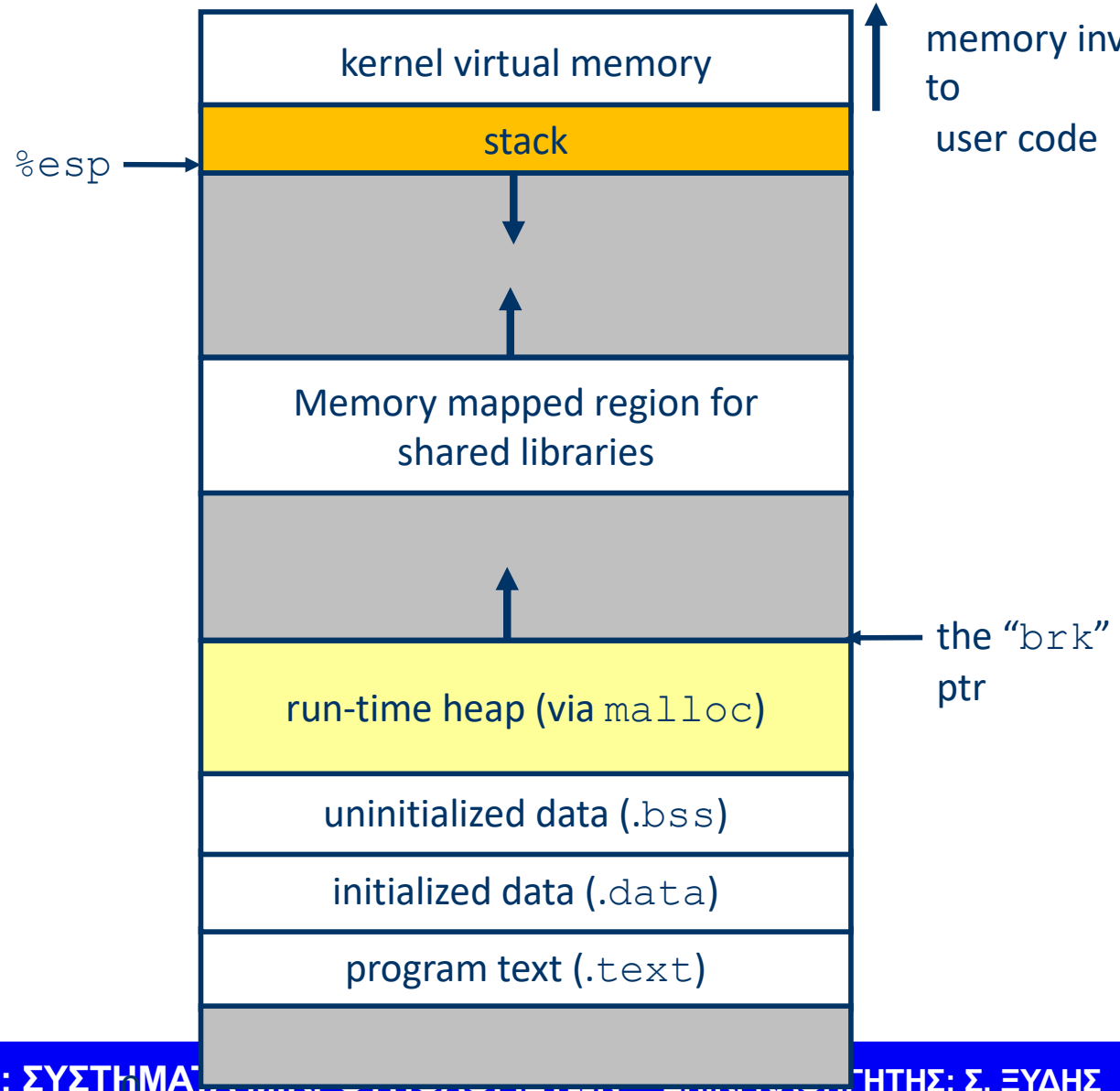
(Reg H) ← ((SP) +1)

(SP) ← (SP) + 2



	Reg H	Reg L
POP B	B	C
POP D	D	E
POP H	H	L
POP PSW	A	Flags

Address space after Loading (obj code architecture) for x86



Υπορουτίνες

CALL Address

((SP) - 1) ← (PCH)

((SP) - 2) ← (PCL)

(SP) ← (SP) - 2

(PC) ← Address

PC of next 8085 instruction

Συνθήκη Address

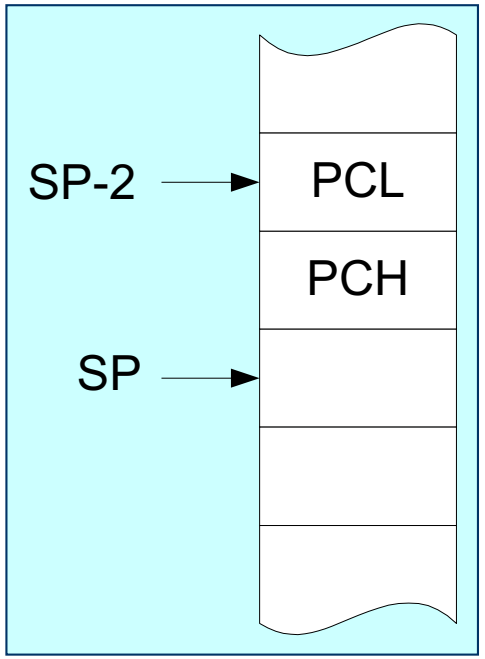
Αν ισχύει η συνθήκη (NZ, Z, NC, C, PO, PE, P, M) τότε:

((SP) - 1) ← (PCH)

((SP) - 2) ← (PCL)

(SP) ← (SP) - 2

(PC) ← Address



RET

(PCL) ← (SP)

(PCH) ← ((SP)+1)

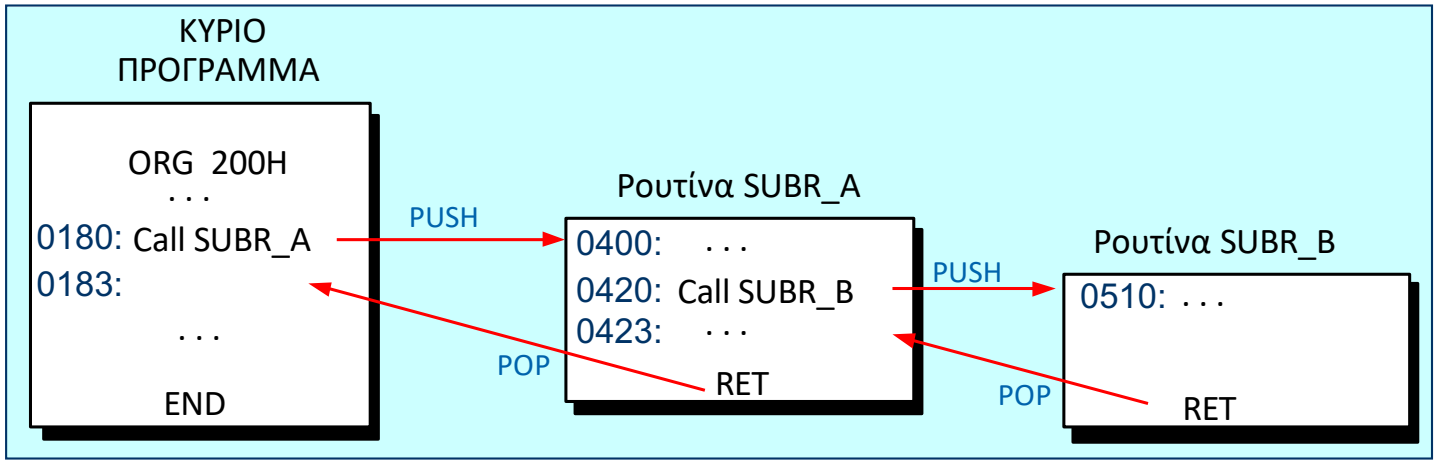
(SP) ← (SP) + 2

Φωλιασμένη Κλήση Υποροutινών

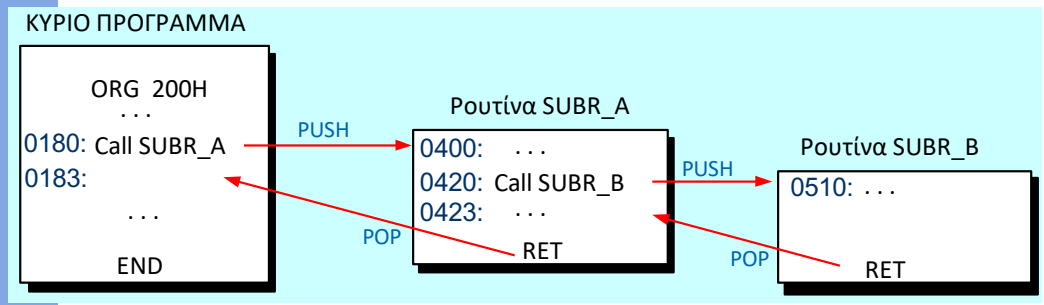
```
0000        LXI         SP,610H
      Κύριο Πρόγραμμα
                . . . . .
0180        CALL        SUBR_A
0183        (επόμενη εντολή)
      Υπόλοιπο Κυρίου Προγρ.
                . . . . .
                END

;Υποροutίνα A
0400  SUBR_A:     PUSH      H
0401              PUSH      D
0402              PUSH      B
0403              PUSH      PSW
      Κύριο Σώμα Υποp. A
                . . . . .
0420              CALL      SUBR_B
0423        Υπόλοιπες Εντολές Υποp. A
                ....
```

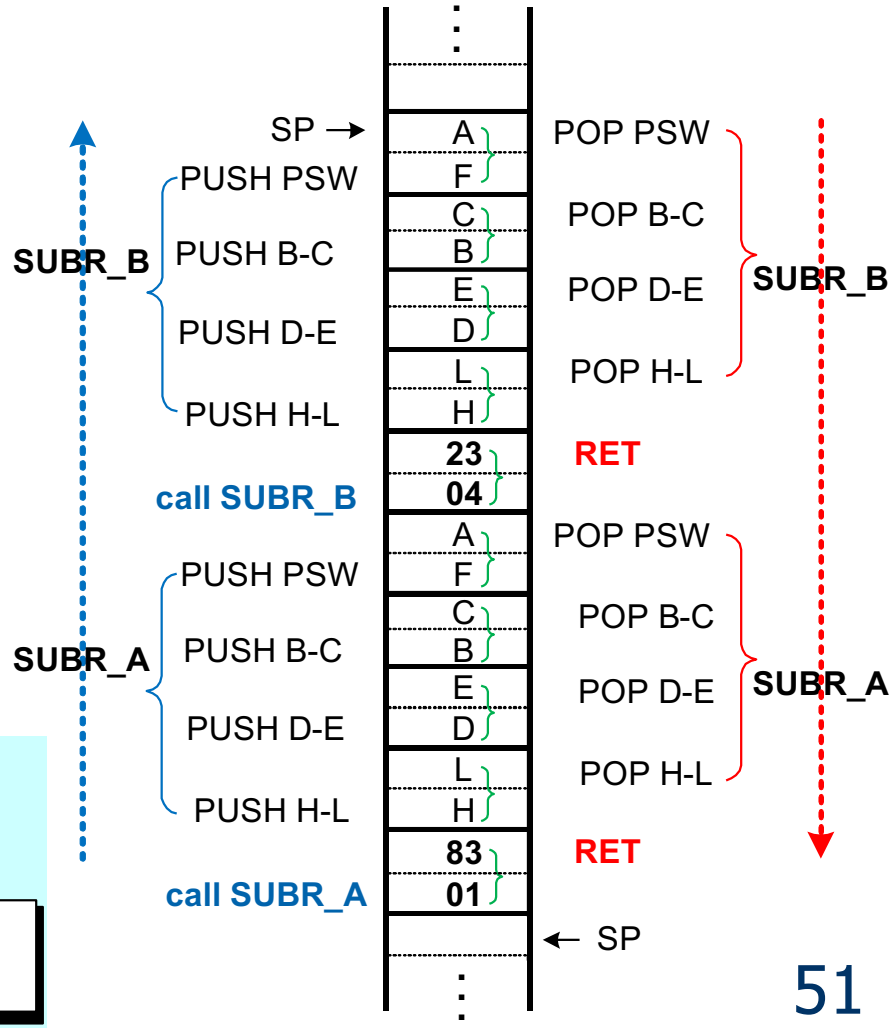
```
0436              POP       PSW
0437              POP       B
0438              POP       D
0439              POP       H
0440              RET

;Υποροutίνα B
0510  SUBR_B:     PUSH      H
0511              PUSH      D
0512              PUSH      B
0513              PUSH      PSW
            Κύριο Σώμα Υποp. B
                . . . . .
0536              POP       PSW
0537              POP       B
0538              POP       D
0539              POP       H
0540              RET
```

Φωλιασμένη Κλήση Υπορουτινών (cont.)



Στοίβα σε Φωλιασμένες Κλήσεις Υπορουτινών





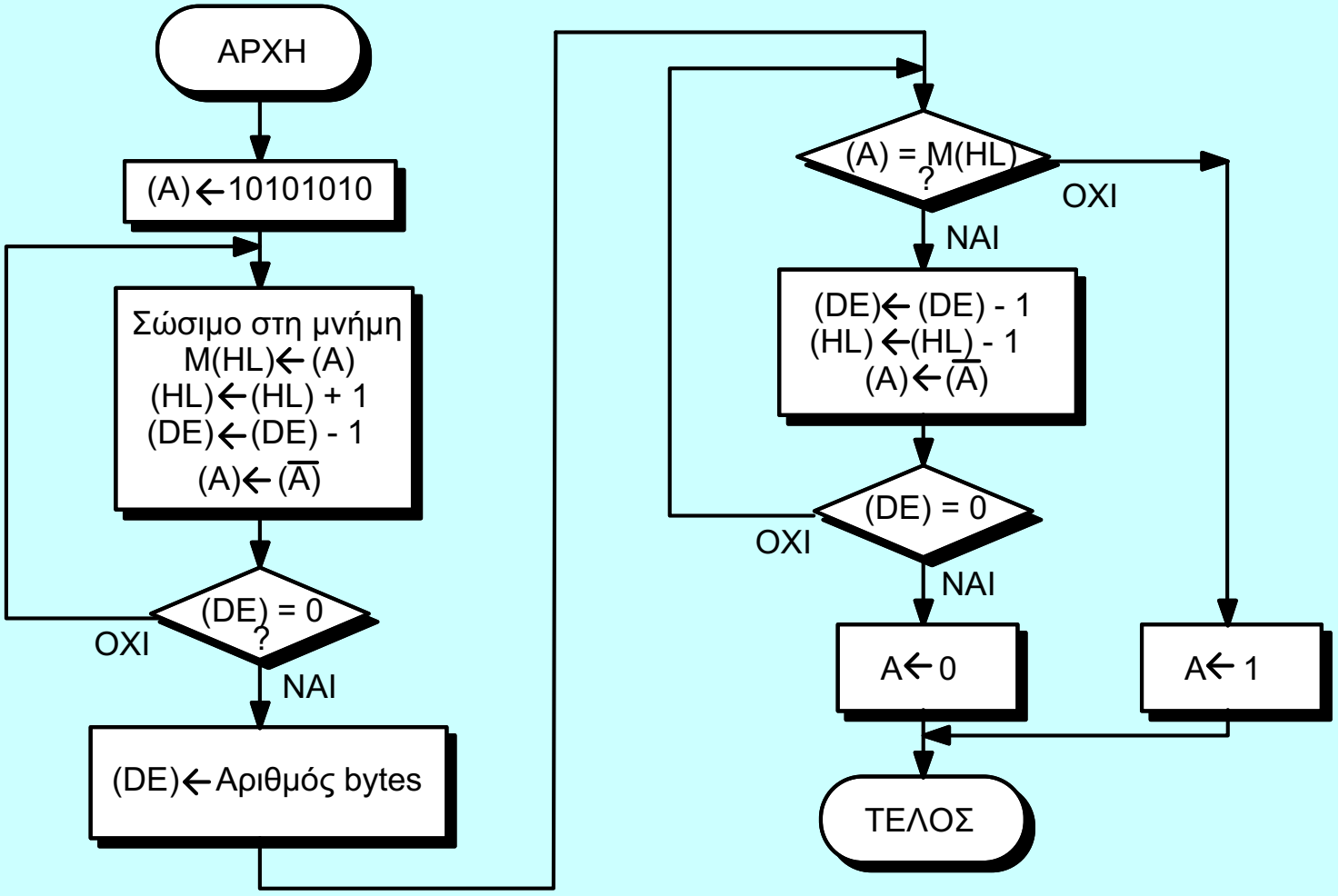
Λογισμικές Διακοπές (Software Interrupts)

RST n
((SP)-1) ← (PCH)
((SP)-2) ← (PCL)
(SP) ← (SP)-2
(PC) ← 8n

RST 0	0000
RST 1	0008
RST 2	0010
RST 3	0018
RST 4	0020
RST 5	0028
RST 6	0030
RST 7	0038

Η ροή του προγράμματος μεταπηδά σε προκαθορισμένη διεύθυνση

Παράδειγμα: Έλεγχος RAM



Έλεγχος RAM: Κώδικας Υπορουτίνας (1)

```
REG16_FLAG MACRO ; Ενεργοποίηση σημαίας Z όταν ο διπλός καταχωρητής DE=0
     MOV  B,A        ; Φυλάσσεται ο καταχωρητής A
     MOV  A,E        ; Όταν και οι δύο καταχωρητές E και D είναι 0,
     ORA  D          ; τότε η σημαία Z γίνεται 1.
     MOV  A,B
ENDM                 ; Μπορεί να γραφεί και σε παραμετρική μορφή για
                     ; το διπλό καταχωρητή X-Y (D-E, H-L)

; Υπορουτίνα ελέγχου μνήμης
     MVI   A,10101010B       ; Φόρτωση του byte ελέγχου
     PUSH D                  ; Σώσιμο του πλήθους των bytes
WRITE:
     MOV  M,A                ; Εγγραφή στη μνήμη
     CMA                     ; Δημιουργία συμπληρωματικού byte ελέγχου
     INX   H                 ; Αύξηση του δείκτη της διεύθυνσης μνήμης
     DCX   D                 ; Μείωση του μετρητή των bytes
     REG16_FLAG              ; Έλεγχος μηδενισμού καταχωρητή DE
     JNZ   WRITE             ; Αν δεν είναι 0 ξαναγράψε
     POP   D                 ; Φόρτωση του πλήθους των bytes
```

Έλεγχος RAM: Κώδικας Υπορουτίνας (2)

```
; Αντίστροφη ανάγνωση και έλεγχος της μνήμης
     DCX H                    ; Μείωση του δείκτη διεύθυνσης μνήμης
READ:                         ;  για να δείχνει το τελευταίο byte
     CMA                      ; Επαναφορά του byte ελέγχου
     CMP  M                   ; Έλεγχος ορθής εγγραφής
     JNZ  ERROR               ; Σε περίπτωση λάθους τερματισμός
     DCX  H                   ; Μείωση του δείκτη διεύθυνσης μνήμης
     DCX  D                   ; Μείωση του μετρητή των bytes
     REG16_FLAG               ; Έλεγχος μηδενισμού καταχωρητή DE
     JNZ  READ                ; Αν δεν είναι 0 ξαναδιάβασε
OK:
     MVI  A,0                 ; Δε βρέθηκε σφάλμα, επιστρέφεται το 0
     RET
ERROR:
     MVI  A,1                 ; Βρέθηκε σφάλμα, επιστρέφεται το 1
     RET
```

Σχεδιάζοντας με μικροεπεξεργαστές

Η αρχιτεκτονική ενός ενσωματωμένου συστήματος είναι συνισταμένη του:

- Hardware
- Software

Το Hardware περιλαμβάνει πολλά συστατικά:

- CPU
- Bus
- Μνήμες
- Συσκευές Εισόδου-Εξόδου (κάρτες δικτύου, αισθητήρες, κτλ)

Ποιά είναι η σωστή επιλογή για κάθε ένα από τα παραπάνω συστατικά του συστήματος;

Αναπτυξιακά Συστήματα για Κατασκευή Μικροϋπολογιστών

Hardware Εργαλεία

- Αναπτυξιακές κάρτες
- Real-time (In-Circuit) emulators
- Logic Analyzers

Software Εργαλεία

- Monitor
- Assemblers
- Cross Compilers
- Cross Debuggers

Σχεδιασμός του Hardware

Η Πολυπλοκότητα του HW σχεδιασμού μπορεί να ποικίλλει από ένα έτοιμο απλό σύστημα μέχρι ένα Full-custom σύστημα.

Αφού γίνει επιλογή του μικροεπεξεργαστή, το πρώτο βήμα είναι η χρήση **αναπτυξιακής κάρτας.** Πρόκειται για μια κάρτα με τον επεξεργαστή, μνήμη και αρκετά Interfaces ενώ συνοδεύεται από το απαραίτητο SW για τον προγραμματισμό και τον έλεγχο του μικροεπεξεργαστή. Η αναπτυξιακή κάρτα είναι διαθέσιμη από την εταιρεία κατασκευής του μικροεπεξεργαστή.

Το δεύτερο βήμα είναι η επιλογή των συσκευών εισόδου/εξόδου και των μνημών. Πρέπει να μελετηθούν τα **datasheets** των ολοκληρωμένων προσεκτικά για να γίνει κατανοητός τόσο ο τρόπος λειτουργίας τους όσο και ο τρόπος ελέγχου και επικοινωνίας.

Το τρίτο βήμα είναι να εξεταστεί η ανάγκη ύπαρξης **glue logic** (FPGAs, CPLDs) για να επιτευχθεί επικοινωνία μεταξύ κάποιων ολοκληρωμένων, ή για να υλοποιηθεί κάποια λογική που απαιτείται (π.χ. Address decoding).

Αρχιτεκτονική Software

Συνήθως οι σχεδιαστές SW Ενσωματωμένων Συστημάτων προσπαθούν να αναπτύξουν μεγάλο μέρος του SW σε μια γνωστή πλατφόρμα γιατί

- Το περιβάλλον προγραμματισμού είναι πιο φιλικό
- Είναι πιο εύκολος ο έλεγχος του SW

Πολλές versions του SW πρέπει να φτιάχνονται ενδιάμεσα για να υλοποιούν τμήματα της συνολικής λειτουργίας και να δοκιμάζονται στην πλατφόρμα του Ενσωματωμένου Συστήματος.

Εργαλεία ανάπτυξης στον Host

Cross Compiler: Πρόκειται για έναν Compiler που τρέχει σε έναν τύπο μηχανής (στο PC) αλλά παράγει κώδικα για έναν άλλο τύπο μηχανής (τον μικροεπεξεργαστή στο Ενσωματωμένο Σύστημα). Ο παραγόμενος κώδικας μπορεί να γίνει download μέσω του σειριακού link στο πραγματικό σύστημα.

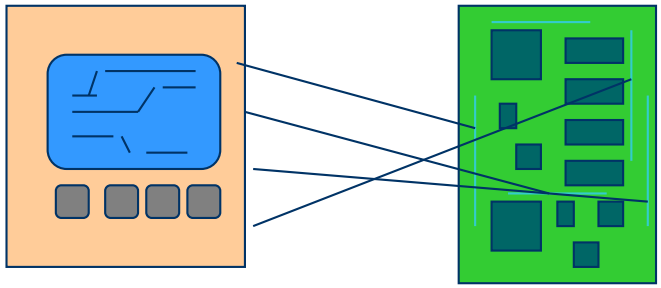
Cross Debugger: Πρόκειται για απεικόνιση σε SW στον Host, της κατάστασης του Target σε πραγματικό χρόνο. Επίσης, επιτρέπει τον απόλυτο έλεγχο του μικροεπεξεργαστή και των περιφερειακών του μέσω της σειριακής επικοινωνίας Host και Target.

Εργαλεία ελέγχου του Target

In-Circuit Emulators: Πρόκειται για έναν ειδικό μικροεπεξεργαστή που επιτρέπει την πρόσβαση στους καταχωρητές του όταν είναι σταματημένος → deprecated & replaced by JTAG tech.

Το βασικό πρόβλημα είναι το γεγονός ότι το μηχάνημα λειτουργεί μόνο με έναν συγκεκριμένο μικροεπεξεργαστή

Logic Analyzers: Πρόκειται για ένα *πλήθος χαμηλού κόστους παλμογράφων*. Χρησιμοποιείται για να απεικονίζεται η ψηφιακή στάθμη σημάτων του συστήματος



Ανάπτυξη και Έλεγχος

Για την ανάπτυξη του SW και τον έλεγχο τόσο του HW όσο και του SW χρησιμοποιείται η συνδεσμολογία **Host-Target,** όπου Host είναι ένα PC και Target το Ενσωματωμένο Σύστημα που αναπτύσσουμε. Η επικοινωνία γίνεται συνήθως μέσω ενός σειριακού Link (JTAG).