1. LD, PUSH, POP και Άλλες Εντολές στον Z80

Μέσα από τα προγράμματα αυτής της Άσκησης, ξαναβλέπουμε την **LD (Load)** και μαθαίνουμε μερικές ακόμα εντολές του **Z80,** κυρίως τις **PUSH και POP.**

Ξαναβλέπουμε πως **ο Z80 εκτελεί κάθε εντολή** σ' ένα πρόγραμμα. **Βλέπουμε πιο αναλυτικά την αρχιτεκτονική σχεδίαση του Z80:** τις βασικές μονάδες από τις οποίες αποτελείται, τη λειτουργία κάθε μονάδας.

Ξεκινάμε να γράφουμε τα πρώτα προγράμματα **στην Assembly του Z80** και μέσα από το προγραμματισμό στην Assembly, βλεπουμε **όχι μόνον** πως λειτουργεί ένας μικροπεξεργαστής, **αλλά και τις βασικές έννοιες και τεχνικές του προγραμματισμού.**

2. LD (Load): Delivery Δεδομένων στον Z80

Η πρώτη εντολή του προγράμματος,

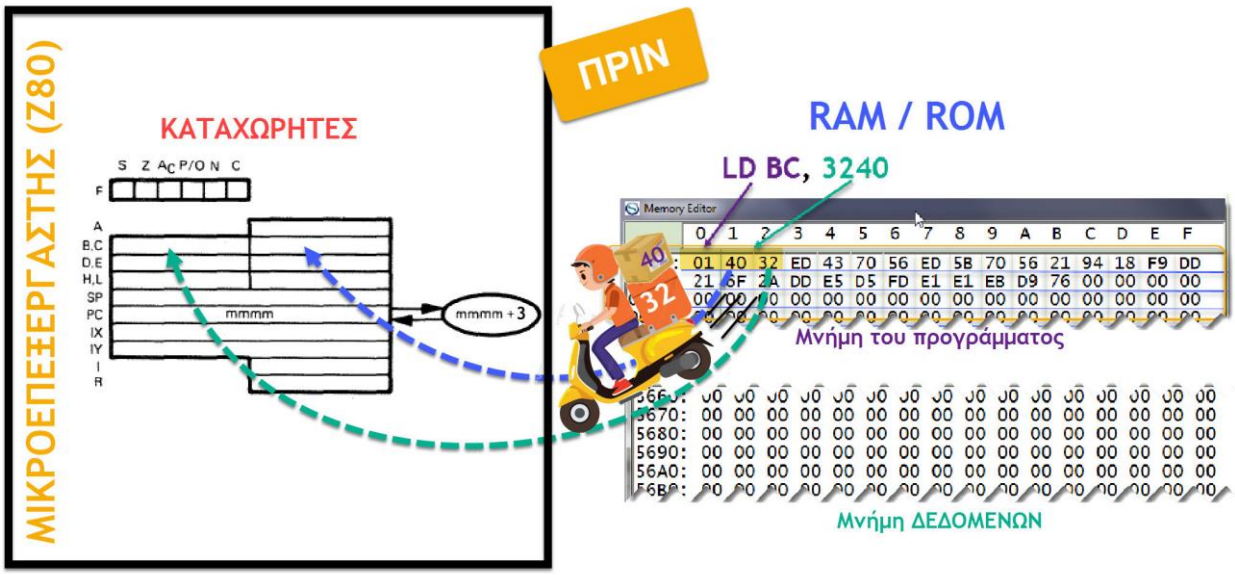
LD BC, 3240H

είναι μία εντολή **Load**. Είδαμε στο προηγούμενο μάθημα πως, γενικά, η εντολή **Load (Φόρτωσε) μεταφέρει δεδομένα** από μία τοποθεσία προέλευσης σε μία άλλη τοποθεσία που είναι η τοπθεσία προσορισμού. Έτσι, γενικά, μία εντολή LD αποτελείται από τρία μέρη:

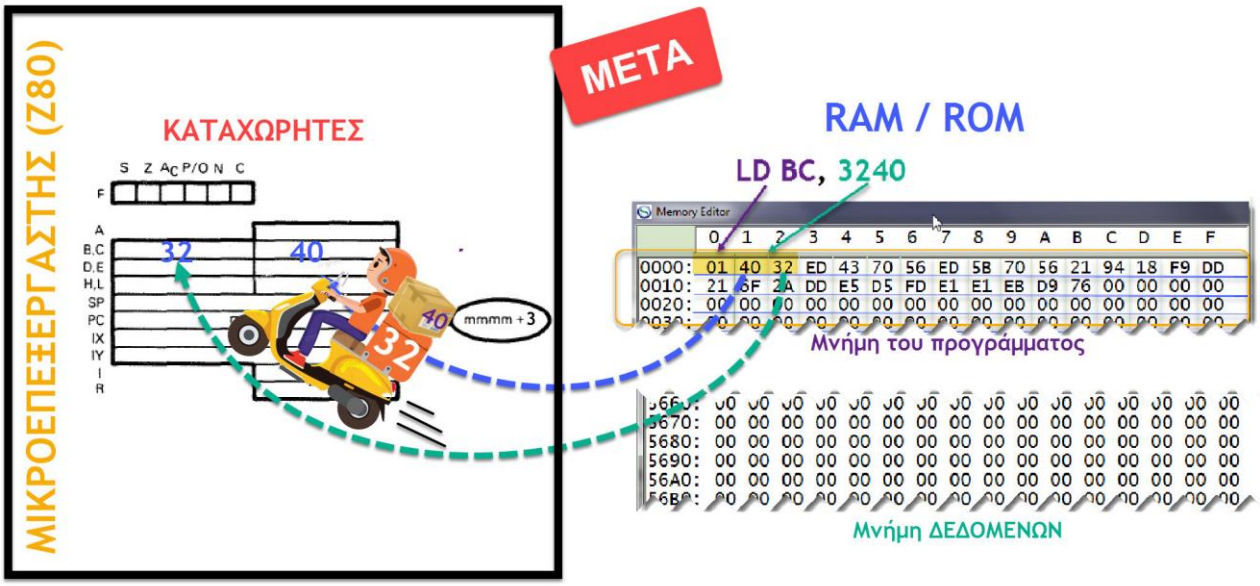
LD	Τελικός προορισμός	Προέλευση

Η τοποθεσία προορισμού, όσο και η τοποθεσία προέλευσης, μπορεί να είναι **είτε ένας καταχωρητής ή μία θέση στη μνήμη**. Έτσι, ανάλογα με το τύπο της αρχικής και της τελικής τοποθεσίας, αλλά και το τρόπο που προσδιορίζεται η διεύθυνση μίας θέσης στη μνήμη, η εντολή **Load** μπορεί να έχει διαφορετικές μορφές. Θα δούμε αυτές τις μορφές της Load, μέσα από παραδείγματα.

Εδώ, η συγκεκριμένη εντολή **LD**, παραπάνω, **είναι μία εντολή 16 Bit**. Γιατί, μεταφέρει, για την ακρίβεια, **αντιγράφει μία τιμή 16 bit**, τη τιμή 3240H, **στο ζεύγος καταχωρητών B** και **C**. Αντιγράφει δηλαδή τα χαμηλής τάξης ψηφία (low order bits), το 40H στον **C** και τα άλλα ψηφία (high order bits) αυτής της τιμής, το 32H στον καταχωρητή **B**. Σχηματικά, η λειτουργία της εντολής παριστάνεται στην Εικόνες 1 και 2.



*Εικόνα 1: Η λειτουργία της εντολής **LD** (**Load** ή **Φόρτωσε**) είναι να μεταφέρει δεδομένα μεταξύ θέσεων μνήμης, στον **Z80**. Εδώ, η **LD** μεταφέρει - αποθηκεύει την σταθερή τιμή 3240H, από την μνήμη του προγράμματος, στο ζεύγος καταχωρητών **BC.***

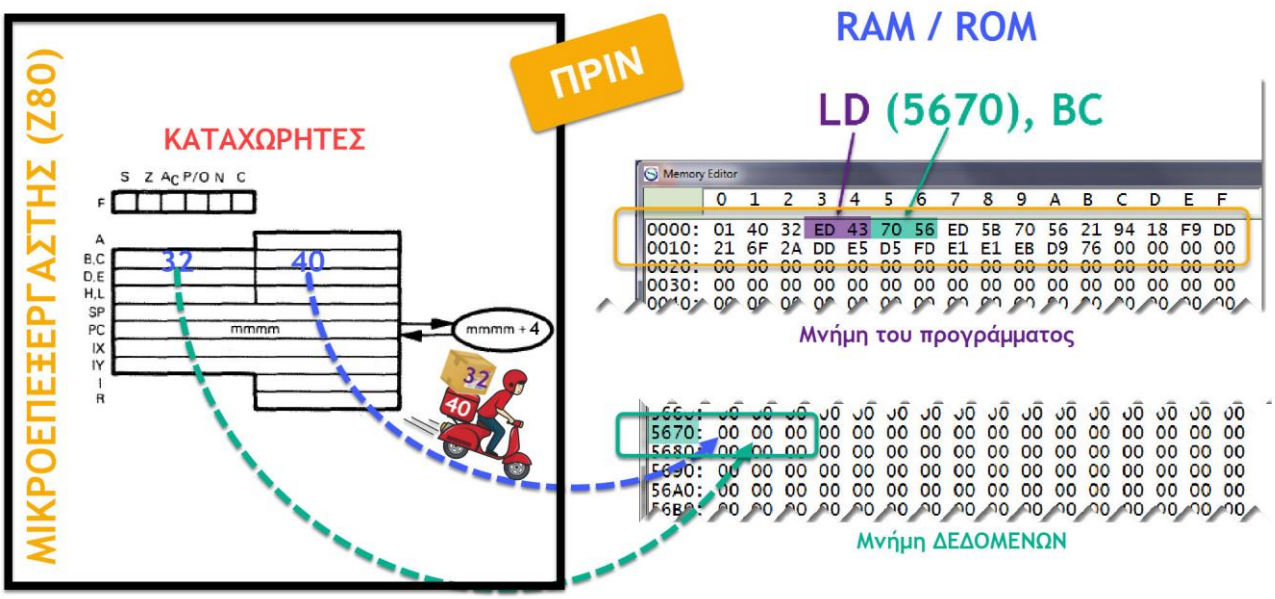


*Εικόνα 2: **Μετά την εκτέλεση** της εντολής LD BC, 3240H, το ζεύγος καταχωρητών **BC** θα πάρει τη τιμή 3240H.*

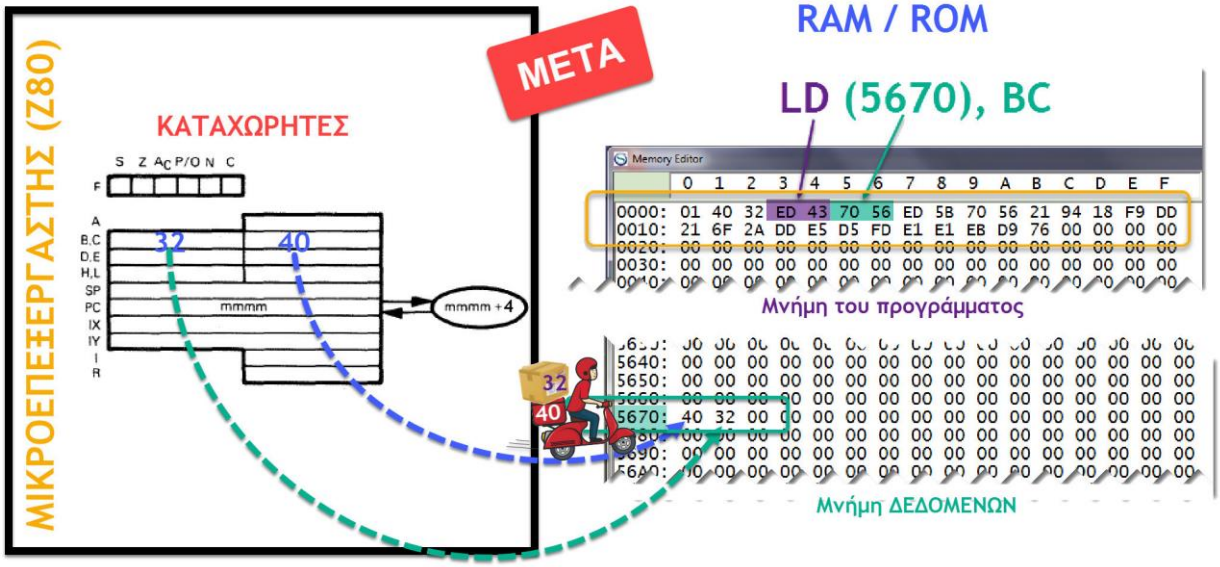
Η δεύτερη εντολή,

LD (5670H), BC

έχει μία διαφορετική μορφή. Αποθηκεύει δεδομένα **σε μία αντίθετη κατεύθυνση**, από ένα ζεύγος καταχωρητών, το **BC**, σε μία θέση στη μνήμη. Η **παρένθεση** γύρω από τη τιμή 5670H, σημαίνει τη **θέση στη μνήμη** με διεύθυνση 5670H. Έτσι, η εκτέλεση αυτής της εντολής αντιγράφει τα περιεχόμενα του ζεύγους **BC**, στη θέση μνήμης 5670H, όπως σχηματικά, παριστάνεται στις Εικόνες 3 και 4.



*Εικόνα 3: Μία **LD** (**Load** ή **Φόρτωσε**) λίγο διαφορετικής μορφής που μετακινεί - φορτώνει δεδομένα σε μία αντίθετη κατεύθυνση, από το ζεύγος καταχωρητών **BC** σε μία θέση **στη μνήμη** των Δεδομένων.*

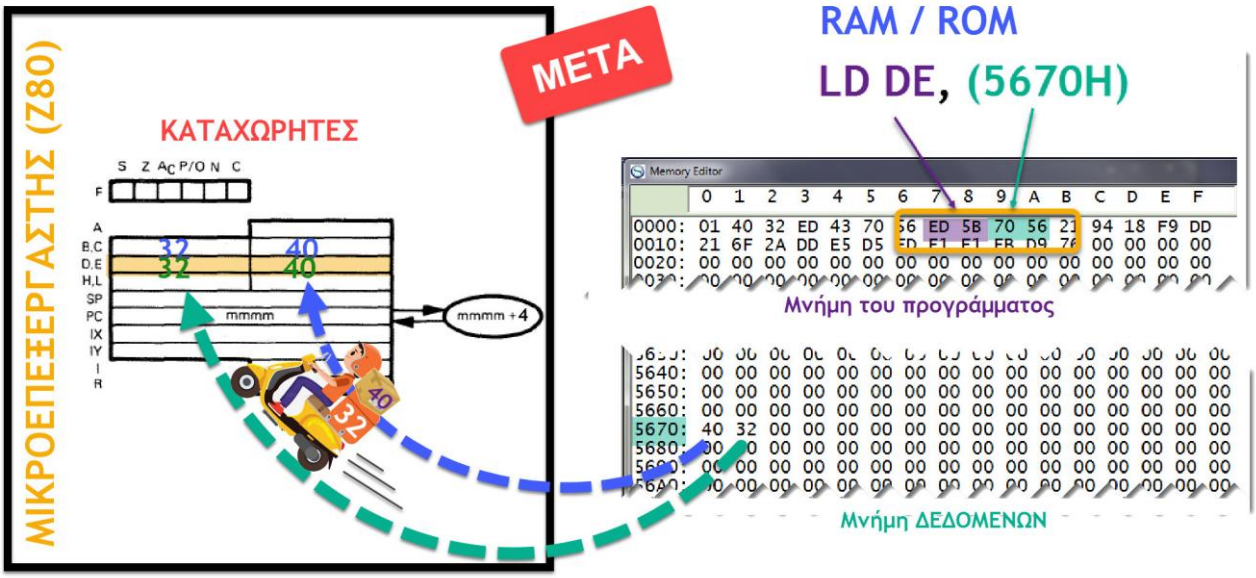


*Εικόνα 4: **Μετά την εκτέλεση της εντολής** LD (5670H), BC, η θέση μνήμης με διεύθυνση 5670H θα περιέχει τη τιμή 40 στο δεκαεξαδικό σύστημα ΚΑΙ η επόμενη θέση μνήμης 5671H, θα περιέχει τη τιμή 32 στο δεκαεξαδικό.*

Η τρίτη εντολή,

LD DE, (5670H)

αποθηκεύει τα περιεχόμενα της θέσης μνήμης με διεύθυνση 5670H στον ζεύγος καταχωρητών **DE**, αποθηκεύοντας τα χαμηλής τάξης ψηφία (low order bits) αυτής της θέσης, στον **E** και τα άλλα ψηφία (high order bits) στον **D**, όπως παριστάνεται στην Εικόνα 5.



*Εικόνα 5: Η εντολή LD DE, (5670H) φορτώνει τα περιεχόμενα της θέσης μνήμης με διεύθυνση (5670H) στο ζεύγος καταχωρητών **DE.***

Καθεμία απο τις παρακάτω εντολές,

LD HL, 1894H

LD SP, HL

LD IX, 2A6FH

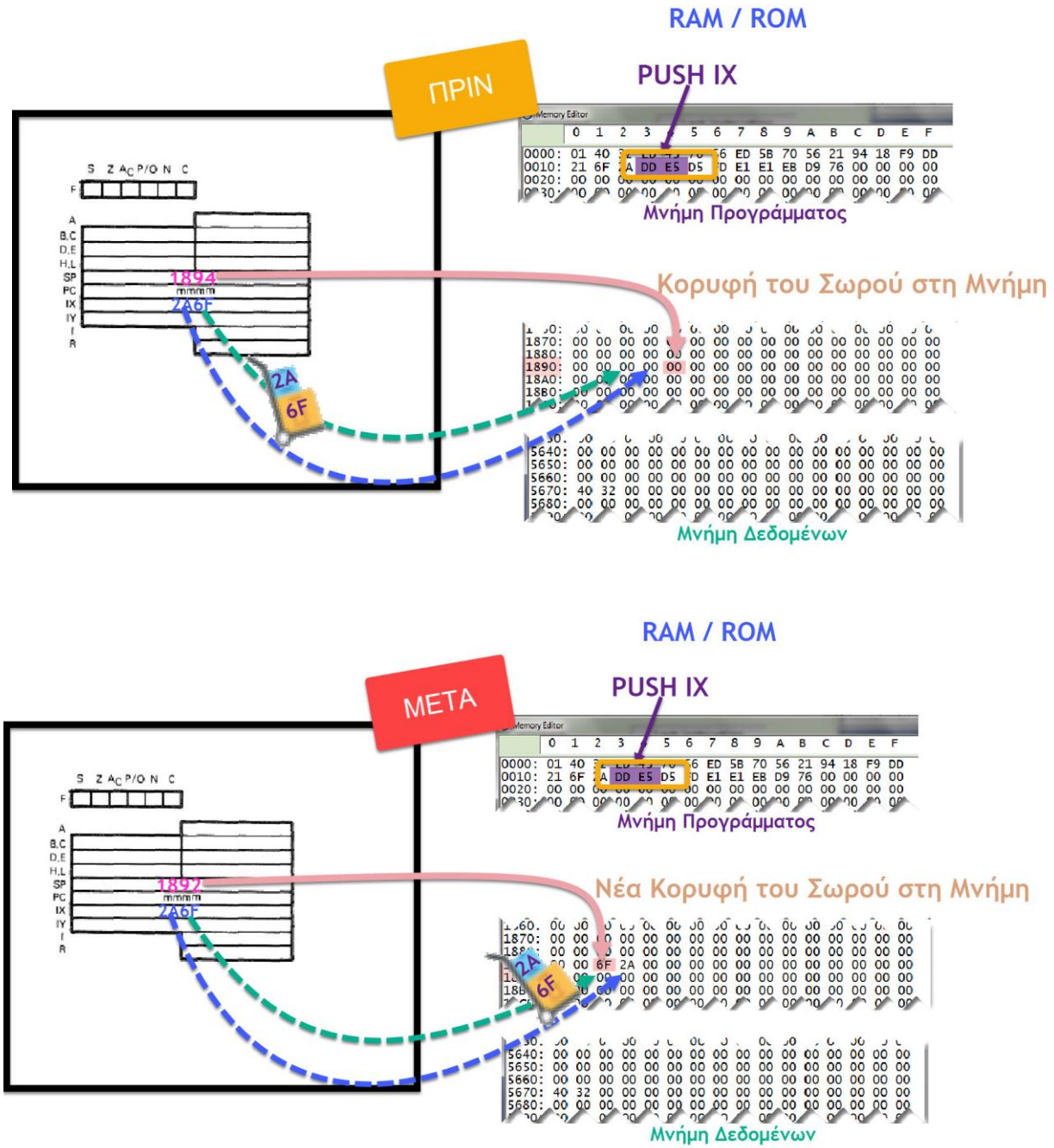
αν και λίγο διαφορετική από τις άλλες, **εκτελεί μία παρόμοια λειτουργία**, είτε αποθηκεύοντας μία σταθερή τιμή, σ' ένα καταχωρητή, τον **IX** ή ζεύγος καταχωρητών, τον **HL** ή αντιγράφοντας τη τιμή από ένα ζεύγος καταχωρητών, το **HL**, σ' έναν άλλο καταχωρητή τον **Stack Pointer** (**SP**).

3. PUSH: Στοιβάζοντας Δεδομένα στο Σωρό

Η επόμενη εντολή,

PUSH IX

αποθηκεύει τα περιεχόμενα του καταχωρητή ένδειξης **IX** στη μνήμη. Γενικά, η εντολή **PUSH** λειτουργεί ώστε να **αποθηκεύει τα περιεχόμενα καταχωρητών στη μνήμη, διατηρώντας έτσι τη κατάσταση της Μονάδας Επεξεργασίας σε κάποιο σημείο του προγράμματος.**



*Εικόνα 6: Η **PUSH** λειτουργεί ώστε να αποθηκεύει τα περιεχόμενα καταχωρητών στη μνήμη RAM, εδώ του **IX**. Έτσι, μπορούμε να διατηρούμε στη μνήμη τη κατάσταση των καταχωρητών σε κάποιο σημείο του προγραμματος.*

Ας δούμε λίγο πιο αναλυτικά τη λειτουργία της **PUSH**. Η **PUSH** αποθηκεύει τα περιεχόμενα ενός καταχωρητή, εδώ, τα περιεχόμενα του **IX** στη μνήμη, **χρησιμοποιώντας έναν άλλο καταχωρητή, τον Stack Pointer (SP)**. Η λειτουργία της **PUSH** γίνεται σε κάποια βήματα (Εικόνα 6):

Βήμα 1. Η **PUSH** μειώνει τη τιμή του **Stack Pointer** (**SP**) κατά 1.

SP <- SP –1

Βήμα 2. Η **PUSH** αποθηκεύει τα υψηλής τάξης ψηφία (high order byte) του **IX** στη θέση μνήμης που η διεύθυνσή της υποδεικνύεται από τον **SP**

(SP) <- 2A

Βήμα 3. Η **PUSH** μειώνει πάλι τη τιμή του **Stack Pointer** (**SP**) κατά 1.

SP <- SP – 1

Βήμα 4. Η **PUSH** αποθηκεύει τα χαμηλής τάξης ψηφία (low order byte) του **IX** στη θέση μνήμης που υποδεικνύεται από τον **SP**.

(SP) <- 6F

Έτσι, η **PUSH** αποθηκεύει - αντιγράφει τα περιεχόμενα του καταχωρητή ένδειξης **IX** στη θέση μνήμης που η διεύθυνση της υποδεικνύεται απο τον **Stack Pointer. Η μνήμη λειτουργεί σαν μία στοίβα που επάνω της,** η **PUSH** τοποθετεί τα περιεχόμανα ενός καταχωρητή, όπως για παράδειγμα, θα βάζαμε ένα ακόμα πιάτο σε μια στοίβα πιάτα.

Η επόμενη εντολή,

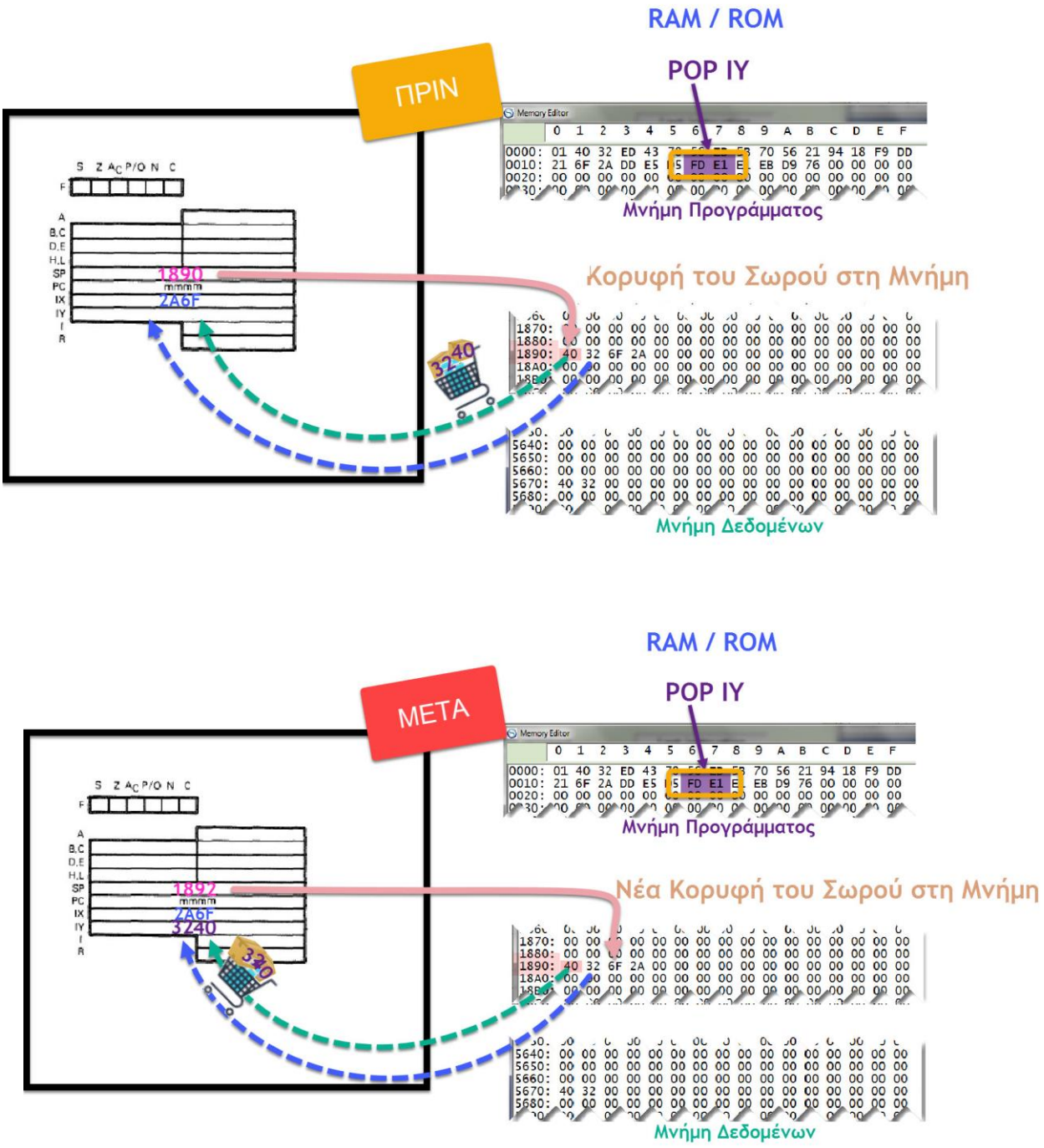
PUSH DE

Κάνει ότι και η προηγούμενη PUSH, όμως, τώρα, με το ζεύγος καταχωρητών DE, τοποθετώντας τα περιεχόμενα των D και E στη θέση μνήμης που υποδεικνύεται από τον Stack Pointer (SP) και σ’ αυτή τη διαδικασία, μειώνοντας τη τιμή του SP κατά 2, ώστε να δείχνει στην καινούργια κορυφή της στοίβας, δηλαδή, την πιο πρόσφατα κατειλημένη θέση στη μνήμη.

Η επόμενη εντολή,

POP IY

κάνει την **αντίθετη λειτουργία** της **PUSH**, μεταφέροντας δεδομένα από τη μνήμη σε καταχωρητές. Αν η **PUSH** αποθηκεύει τα περιεχόμενα των καταχωρητών στη μνήνη, **χρησιμοποιώντας τη μνήμη σαν να ήταν μια στοίβα με πιάτα** που επάνω της, σπρώχνουμε και άλλα πιάτα, η **POP** κάνει την αντίθετη λειτουργία. Μεταφέρει περιεχόμενα από τη μνήμη στους καταχωρητές, σαν να αφαιρεί πιάτα από μία στοίβα (Εικόνα 7).



*Εικόνα 7: Η **POP** έχει την **αντίθετη λειτουργία της PUSH,** μεταφέροντας περιεχόμενα από την κορυφή της στοίβας στη μνήμη, στους καταχωρητές, εδώ στον **IY.***

Ενώ η **PUSH** αποθηκεύει τα περιεχόμενα των καταχωρητών στη μνήμη, για να διατηρήσει σ' αυτήν, τη κατάσταση των καταχωρητών σε κάποιο σημείο του προγράμματος, η **POP** κάνει την αντίθετη λειτουργία. Μεταφέροντας δεδομένα από τη στοίβα της μνήμης, στους καταχωρητές, **επαναφέρει μία προηγούμενη κατάσταση του προγράμματος**.

Στην μετακίνηση δεδομένων από τη μνήμη σε καταχωρητές, η **POP**, κάθε φορά, παίρνει τα δεδομένα - τη τιμή **στην κορυφή της στοίβας της μνήμης**, δηλαδή, τη τιμή στη θέση μνήμης που

υποδεικνύεται από τον **Stack Pointer** (**SP**) και την αποθηκεύει στον καταχωρητή που ορίζεται στην εντολή, εδώ, στον **IY**. Ας δούμε, λίγο πιο αναλυτικά, τα βήματα μίας **POP** (Εικόνα 7):

Βήμα 1. Η **POP** αποθηκεύει - **αντιγράφει τη τιμή από τη θέση μνήμης** που η διεύθυνσή της υποδεικνύεται από τον **SP,** στα χαμηλής τάξης ψηφία (low order byte) του καταχωρητή που προσδιορίζεται στην εντολή, εδώ του **IY.**

$(IY)_L$ <- 32H

Βήμα 2. Η **POP** αυξάνει τη τιμή του **Stack Pointer** (**SP**) κατά 1.

SP <- SP + 1

Βήμα 3. Η **POP** αποθηκεύει - **αντιγράφει τη τιμή από τη θέση μνήμης** που, τώρα, υποδεικνύεται από τον **SP,** στα υψηλής τάξης ψηφία (high order byte) του καταχωρητή που προσδιορίζεται στην εντολή, εδώ του **IY.**

$(IY)_H$ <- 40H

Βήμα 4. Η **POP** αυξάνει πάλι τη τιμή του **Stack Pointer** (**SP**) κατά 1, ώστε να δείχνει στη καινούργια κορυφή της στοίβας της μνήμης.

SP <- SP + 1

Έτσι, οι **PUSH** και **POP** διαχειρίζονται τη μνήμη **σα να ήταν μία στοίβα**, όπου η **PUSH** σπρώχνει - τοποθετεί καινούργιες τιμές, μεταφέροντας εκεί τις τιμές κατχωρητών, ενώ η **POP** αφαιρεί τιμές απ' αυτή τη στοίβα, μεταφέροντας τες, σε καταχωρητές στη Μονάδα Επεξεργασίας. Παρακάτω, θα δούμε πως οι PUSH και POP χρησιμεύουν σ' ένα πρόγραμμα.

Η επόμενη εντολή,

POP HL

κάνει ότι και η προηγούμενη POP, όμως, τώρα, με τον **HL**, μεταφέροντας στο ζεύγος καταχωρητών **H** και **L**, τη τιμή **από τη κορυφή της στοίβας της μνήμης**, δηλαδή από τη θέση μνήμης που υποδεικνύεται από τον SP.

Η εντολή,

EX DE, HL

κάνει ότι λέει το όνομά της, **ανταλλάσσει τα περιεχόμενα των καταχωρητών DE** και **HL μεταξύ τους.**

Η εντολή,

EXX

κάνει / προσδιορίζει μια μεγαλύτερη **ανταλλαγή περιεχομένων**. Ανταλλάσσει τα περιεχόμενα αντίστοιχων ζευγών καταχωρητών, μεταξύ τους, των:

BC και BC' μεταξύ τους (BC <-> B'C')

DE και DE' μεταξύ τους (DE <-> D'E')

HL και HL' μεταξύ τους (HL <-> H'L')

Συνοπτικά, το αποτέλεσμα εκτέλεσης κάθε εντολής του πρώτου προγράμματος της Άσκησης 2, παριστάνεται στον **Πίνακα 2Α**.

Πίνακας 2Α. Το Πρόγραμμα: Με μία Ματιά

Assembly Language	Machine Language	Bytes	Αποτέλεσμα Εκτέλεσης Εντολής
START: LD BC,3240H	0000 01 40 32	3	B <— 32H C <— 40H (Εικόνα 2)
LD (5670H),BC	ED 43 70 56	4	(5670H) <— 40H (5671H) <— 32H (Εικόνα 4)
LD DE,(5670H)	ED 56 70 56	4	D <— 32H E <— 40H (Εικόνα 5)
LD HL,1894H	21 94 18	3	H <— 18H L <— 94H
LD SP, HL	F9	1	SP <— 1894H
LD IX,2A6FH	DD 21 6F 2A	4	IX <— 2A6FH
PUSH IX	DD E5	2	(1893H) <— 2AH (1892H) <— 6FH (Εικόνα 6)
PUSH DE	D5	1	(1891H) <— 32H (1890H) <— 40H
POP IY	FD E1	2	IY <— 3240H (Εικόνα 7)
POP HL	E1	1	H<— 2AH L<— 6FH
EX DE,HL	EB	1	D <— 2AH, E <— 6FH H <— 32H, L <— 40H
EXX	D9	1	B <— FFH, C <— FFH D <— FFH, E <— FFH H <— FFH, L <— FFH B' <— 32H, C' <— 40H D' <— 2AH, E' <— 6FH H' <— 32H, L' <— 40H
HALT	76	1	

4. Κύκλοι (Cycles): Τα Βήματα στην Εκτέλεση μιας Εντολής

Εκτός από λίγες εντολές, **οι περισσότερες εντολές στον Z80 εκτελούνται σε περισσότερο από ένα βήματα ή κύκλους μηχανής ή, απλά, κύκλους**. **Τα βήματα διαφέρουν από εντολή σε εντολή,** ακόμα για εντολές στην ίδια κατηγορία, όπως οι διάφορες παραλλαγές της **LD (Load).**

Όμως, **υπάρχει ένας γενικός κανόνας που με μικρές διαφοροποιήσεις ακολουθεί η εκτέλεση της κάθε εντολής.** Σύμφωνα μ' αυτό τον κανόνα, η εκτέλεση κάθε εντολής στον **Z80** γίνεται μέσα από τα παρακάτω **βήματα ή κύκλους:**

Βήμα 1: Αποκωδικοποίηση. Το πρώτο βήμα στην εκτέλεση κάθε εντολής, είναι η μεταφορά του πρώτου byte της εντολής που είναι ο κωδικός της, στη **Μονάδα Ελέγχου** του μικροεπεξεργαστή. Εκεί, **ο κωδικός της εντολής αποκωδικοποιείται**, ώστε ο μικροεπεξεργαστής να "δει" ποια εντολή πρέπει να εκτελέσει.

Σημείωση. Αν και για πολλές εντολές του Z80, ο κωδικός τους είναι ένα byte, **ορισμένες εντολές έχουν κωδικούς που είναι 2 bytes**. Έτσι, **η αποκωδικοποίηση αυτών των εντολών** δεν περιλαμβάνει ένα, αλλά **δύο βήματα**. Στο πρώτο βήμα, αποκωδικοποιείται το πρώτο byte και στο δεύτερο βήμα, μεταφέρεται και αποκωδικοποιείται στη Μονάδα Ελεγχου, το δεύτερο byte.

Βήμα 2: Μεταφορά των Δεδομένων της Εντολής σε Καταχωρητές. Αν μία εντολή δεν προσδιορίζει μία λειτουργία μεταξύ καταχωρητών, αλλά **περιλαμβάνει και δεδομένα**, τότε, **στο δεύτερο βήμα, τα δεδομένα της εντολής μεταφέρονται από τη μνήμη σε καταχωρητές, ένα byte κάθε φορά**. Αν τα δεδομένα μιας εντολής είναι 2 bytes, τότε, η μεταφορά τους από τη μνήμη σε καταχωρητές, γίνεται σε δύο βήματα.

Βήμα 3: Εκτέλεση της Εντολής. Αν μία εντολή προσδιορίζει μία αριθμητική ή λογική πράξη, τότε, σ' αυτό το βήμα, γίνεται **η εκτέλεση αυτής της αριθμητικής ή λογικής πράξης**, στη **Μονάδα Αριθμητικής & Λογικής Επεξεργασίας (ALU)**. **Αυτό είναι το μόνο παραγωγικό στάδιο, στην εκτέλεση μίας εντολής, από τη πλευρά των υπολογισμών** που κάνει ένα πρόγραμμα.

Όλα τα άλλα στάδια ή βήματα στην εκτέλεση μιας εντολής είναι **μη παραγωγικά**. Γιατί, μετακινούν δεδομένα ή κωδικούς από τη μνήμη στη Μονάδα Επεξεργασίας και αντίστροφα, **χωρίς όμως να κάνουν κάποιο υπολογισμό ή επεξεργασία**.

Αυτό, δηλαδή, ποια στάδια στην εκτέλεση μιας εντολής, είναι παραγωγικά και ποια όχι, είναι πολύ σημαντικό, για να σχεδιάσουμε καινούργιες αρχιτεκτονικές για μικροεπεξεργαστές που μειώνουν το χρόνο εκτέλεσης ενός προγράμματος, **εκτελώντας παράλληλα κάποια στάδια στην εκτέλεση μίας εντολής**.

Βήμα 4: Αποθήκευση στη Μνήμη. Αν η εντολή προσδιορίζει την αποθήκευση δεδομένων ή αποτελεσμάτων στη μνήμη, τότε, σ' αυτό το στάδιο, τα **δεδομένα ή αποτελέσματα από την εκτέλεση της εντολής, μεταφέρονται από καταχωρητές στη μνήμη, ένα byte κάθε φορά**. Έτσι, αν τα δεδομένα που πρέπει να αποθηκευτούν στη μνήμη, είναι δύο bytes, αυτό το στάδιο εκτελείται σε δύο επιμέρους βήματα. Στο πρώτο βήμα, μεταφέρεται το πρώτο byte και στο δεύτερο βήμα, το δεύτερο byte.

Ας δούμε, τώρα, πως εφαρμόζεται αυτός ο κανόνας και τις διαφοροποιήσεις του, για κάθε εντολή του προγράμματος, ξεκινώντας από την πρώτη εντολή,

Assembly	Machine Language		
LD BC, 3240H	**00000001**	**40**	**32**

Αυτή η εντολή εκτελείται σε τρία (3) βήματα ή κύκλους:

Δηλαδή, στο πρώτο βήμα, γίνεται η αποκωδικοποίηση του κωδικού της εντολής. Στο δεύτερο βήμα, μεταφέρεται το πρώτο byte από τα δεδομένα της εντολής, δηλαδή, η τιμή 32 στον **B** και στο τρίτο βήμα, μεταφέρεται το δεύτερο byte από τα δεδομένα της εντολής, δηλαδή η δεκαεξαδική τιμή 40 στον **C**.

Η εντολή,

Assembly	Machine Language			
LD (5670), BC	**ED**	**01000011**	**70**	**56**

Εκτελείται σε έξι (6) βήματα ή κύκλους:

Η αποκωδικοποίηση της εντολής **δεν γίνεται σ' ένα**, **αλλά σε δύο βήματα**, γιατί ο κωδικός της είναι δύο (2) bytes. Στο πρώτο βήμα, αποκωδικοποιείται το πρώτο byte και στο δεύτερο, το δεύτερο byte του κωδικού της. Τα **δεδομένα αυτής της εντολής** είναι η διεύθυνση μνήμης 5670. Επειδή αυτή η διεύθυνση είναι δύο bytes, **η μεταφορά της σε καταχωρητές**, στη Μονάδα Επεξεργασίας, περιλαμβάνει δύο βήματα.

Ακόμα, αυτή η εντολή περιλαμβάνει την **αποθήκευση των περιεχομένων** του ζεύγους **BC** στη μνήμη. Επειδή αυτά τα περιεχόμενα είναι δύο bytes, η **αποθήκευσή τους στη θέση μνήμης** με διεύθυνση 5670, γίνεται σε δύο βήματα.

Η τρίτη εντολή,

Assembly	Machine Language			
LD DE, (5670H)	**ED**	**01011011**	**70**	**56**

εκτελείται σε έξι (6) βήματα ή κύκλους:

Αυτό, γιατί, η **αποκωδικοποίηση του κωδικού της εντολής** που είναι δύο bytes, γίνεται σε δύο βήματα. Τα **δεδομένα της εντολής είναι η διεύθυνση** 5670. Επειδή, αυτή η διεύθυνση είναι δύο bytes, **η μεταφορά της σε καταχωρητές** γίνεται σε δύο βήματα. Η εντολή περιλαμβάνει **την μεταφορά των δεδομένων απ' αυτή τη διεύθυνση στη μνήμη, στο ζεύγος καταχωρητών DE**. Επειδή, αυτά τα περιεχόμενα είναι δύο bytes, **η μεταφορά τους από τη μνήμη γίνεται** σε δύο βήματα.

Καθεμία απο τις παρακάτω εντολές,

LD HL, 1894H

LD SP, HL

LD IX, 2A6FH

αν και λίγο διαφορετική από τις προγούμενες, **εκτελεί μία παρόμοια λειτουργία** και παρόμοια, υπολογίζουμε τα βήματα ή κύκλους στην εκτέλεση τους.

Η εντολή,

Assembly	Machine Language	
PUSH IX	**DD**	**E5**

εκτελείται σε τέσσερα (4) βήματα ή κύκλους,

.

Τα δύο πρώτα βήματα περιλαμβάνουν την **αποκωδικοποίηση του κωδικού της** που είναι δύο bytes. Τα άλλα δύο βήματα είναι η **αποθήκευση στη μνήμη των περιεχομένων** του **IX**, ένα βήμα για την αποθήκευση του χαμηλότερης τάξης byte και ένα ακόμα βήμα, για την αποθήκευση του υψηλοτερης τάξη byte του **IX.**

Αλλά, η επόμενη εντολή,

Assembly	Machine Language
PUSH DE	**11010101**

εκτελείται σε τρία (3) βήματα ή κύκλους,

Αυτό, γιατί, ο κωδικός της εντολής είναι ένα μόνον byte και επομένως, η **αποκωδικοποίηση΄του** γίνεται σ' ένα βήμα. Τα άλλα δύο βήματα, είναι η **αποθήκευση των περιεχομένων** του **DE** που είναι δύο bytes σε δύο bytes, στη μνήμη.

Η εντολή,

Assembly	Machine Language	
POP IY	**FD**	**E1**

εκτελείται σε τέσσερα (4) βήματα ή κύκλους,

Τα δύο πρώτα βήματα περιλαμβάνουν την **αποκωδικοποίηση του κωδικού της** που είναι δύο bytes. Τα άλλα δύο βήματα είναι η **μεταφορά των δεδομένων από την κορυφή της στοίβας της μνήμης, στον IY**, ένα βήμα για την μεταφορά του χαμηλότερης τάξης byte και ένα ακόμα βήμα, για την μεταφορά του υψηλοτερης τάξη byte στον **IY.**

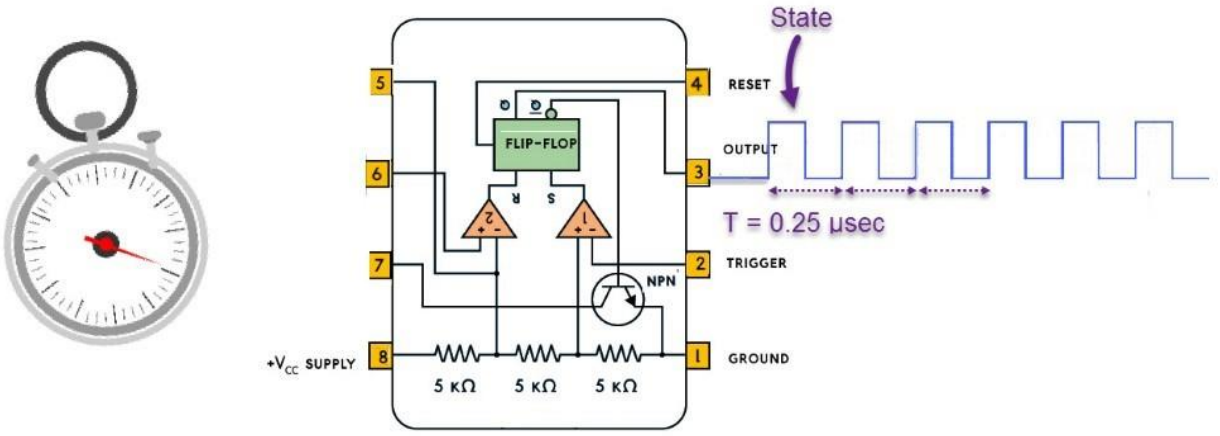
Παρόμοια υπολογίζονται τα βήματα στην εκτέλεση των υπόλοιπων εντολών του προγράμματος. Τα βήματα στην εκτέλεση κάθε εντολής σ' αυτό το πρόγραμμα, συνοψίζονται στο **Πίνακα 2Β**

5. States ή Πόσο Χρόνο Παίρνει ένα Πρόγραμμα να Εκτελεστεί?

Όταν λέμε **states** στην εκτέλεση μίας εντολής, εννοούμε το **χρόνο που απαιτείται για την εκτέλεση αυτής της εντολής**. **Όμως, χρόνο στο ρολόι** του **Z80**. Ας το δούμε, λίγο αναλυτικότερα.

Ο **Z80,** αλλά και κάθε μικροεπεξεργαστής έχει ένα **εσωτερικό ρολόι** που **μετράει το χρόνο που κάνει να εκτελέσει κάθε εντολή** ενός προγράμματος, άρα και το συνολικό χρόνο που κάνει να εκτελέσει ολόκληρο το πρόγραμμα.

Όμως, το ρολόι του **Z80,** γενικότερα, το ρολόι σ' ένα μικροεπεξεργαστή είναι **λίγο διαφορετικό από τα δικά μας ρολόγια. Τ**α δικά μας ρολόγια μετρούν το χρόνο, μέσα από την κυκλική περιστροφή δεικτών. Για παράδειγμα, μία πλήρης περιστροφή 360° του δείκτη των λεπτών, στο καντράν του ρολογιού, σημαίνει ένα λεπτό. Κάθε πλήρης περιστροφή 360° του δείκτη των δευτερολέπτων, σημαίνει ένα δευτερόλεπτο. **Το ρολόι του Z80 είναι μία γεννήτρια τετραγωνικών παλμών που μετράει το χρόνο μέσα από αυτούς τους παλμούς** (Εικόνα 8).



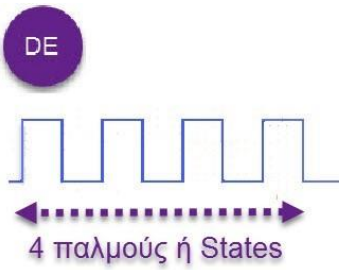
*Εικόνα 8: Το **ρολόι του Z80** που **δεν μοιάζει** μ' ένα δικό μας ρολόι (αριστερά), αλλά είναι ένα κύκλωμα (γεννήτρια) τετραγωνικών παλμών (δεξιά). Ενώ εμείς μετράμε το χρόνο μέσα από τις στροφές των δεικτών του ρολογιού, **ένας μικροεπεξεργαστής μετράει το χρόνο με παλμούς**. **Κάθε παλμός αντιστοιχεί σ' ένα state***

Η συχνότητα αυτής της γεννήτριας είναι ***f = 4 MHz***. Άρα, η **περίοδος T** του παλμού που δημιουργεί είναι:

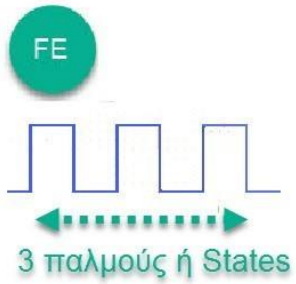
$$T = \frac{1}{4\ MHz} = \frac{1}{4 * 10^6\ Hz} = \frac{1}{4} * 10^{-6} sec = \frac{1}{4}\ \mu sec = 0.25\ \mu sec$$

Επομένως, **αντί για δείκτες, το ρολόι του Z80 μετράει το χρόνο σε παλμούς που δημιουργεί ένα κύκλωμα (γεννήτρια) παλμών, στο εσωτερικό του** (Εικόνα 8).

Έτσι, ο **Z80 μετράει το χρόνο εκτέλεσης κάθε εντολής με τους παλμούς του ρολογιού του** (Εικόνα 8). Είδαμε πως η εκτέλεση κάθε εντολής χωρίζεται σε βήματα. Κάθε βήμα στην εκτέλεση μιας εντολής παίρνει συγκεκριμμένο χρόνο. Δηλαδή:

> **Αποκωδικοποίηση (4 Παλμούς ή States)**. Η αποκωδικοποίηση του κωδικού μίας εντολής παίρνει **4 παλμούς του ρολογιού**. Η μεταφορά, δηλαδή, του κωδικού της εντολής απο την ROM στη Μονάδα Ελέγχου και η αποκωδικοποίηση του εκεί, ώστε ο μικροεπεξεργαστής να "δει" ποια εντολή πρέπει να εκτελέσει, χρειάζεται 4 παλμούς του ρολογιού.
>
> 
>
>
> **Αν ο κωδικός μίας εντολής είναι δύο bytes και έτσι, η αποκωδικοποίση του γίνεται σε δύο βήματα**, τότε και ο χρόνος για την αποκωδικοποίηση της εντολής δεν θα είναι 4 παλμοί, αλλά 2 × 4 = 8 παλμοί του ρολογιού

Μετακίνηση Δεδομένων από τη Μνήμη σε Καταχωρητές (3 παλμοί). Η **μετακίνηση των δεδομένων** μίας εντολής, από τη μνήμη σε καταχωρητές στη Μονάδα Επεξεργασίας γίνεται ένα byte κάθε φορά και η μεταφορά κάθε byte, παίρνει **3 παλμούς**.



Εκτέλεση μίας Εντολής. Αν μία εντολή προσδιορίζει μία αριθμητική ή λογική πράξη, **η εκτέλεση αυτής της πράξης** στη Μονάδα Αριθμητικής και Λογικής Επεξεργασίας **παίρνει χρόνο ανάλογα με τη συγκεκριμμένη πράξη**.

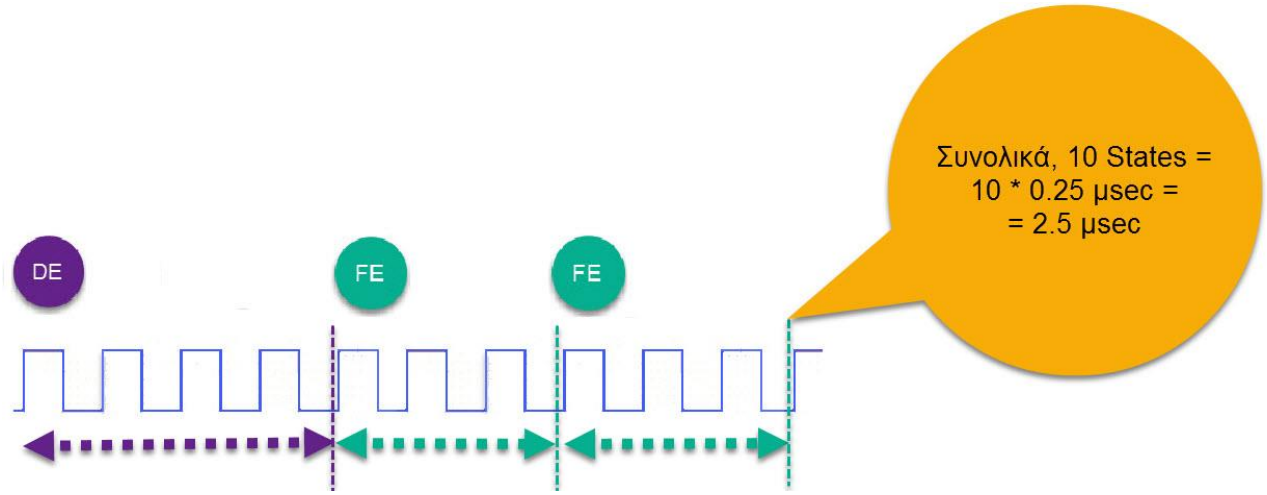
Αποθήκευση Δεδομένων ή Αποτελεσμάτων στη Μνήμη (3 παλμοί). Η **αποθήκευση περιεχομένων των κατταχωρητών στη μνήμη**, γίνεται ένα byte κάθε φορά και η ματακίνηση ενός byte από ένα καταχωρητή στη μνήμη RAM, παίρνει τρείς (3) παλμούς του ρολογιού.

Έχοντας προσδιορίσει το χρόνο κάθε βήματος στην εκτέλεση μίας εντολής, εύκολα, μπορούμε να υπολογίσουμε το χρόνο εκτέλεσης της εντολής, **απλώνοντας τα βήματα της εκτέλεσής της στο χρόνο**. Ξεκινάμε με την πρώτη εντολή, στο πρόγραμμα της Άσκησης 2:

LD BC, 3240H

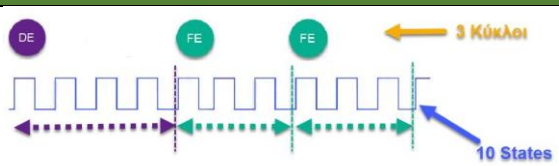
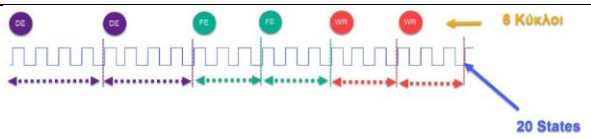
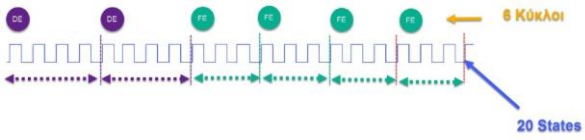
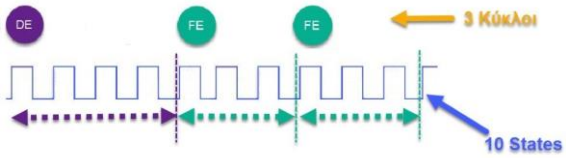
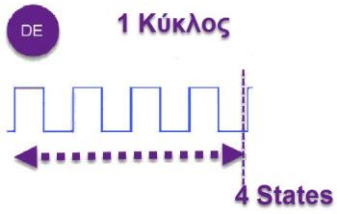
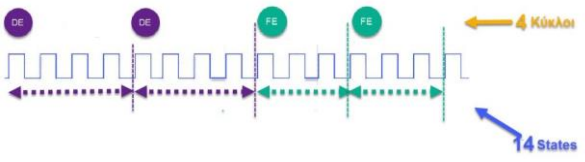
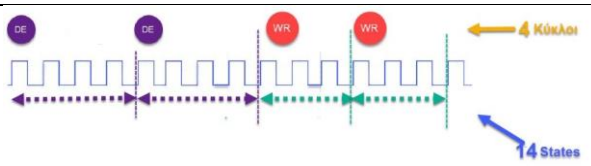
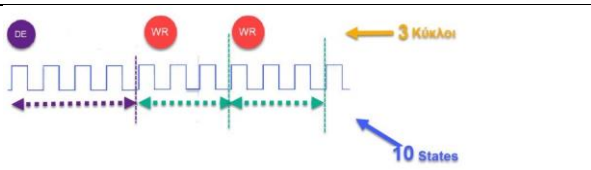
Πόσο χρόνο κάνει ο Z80 να εκτελέσει αυτή την εντολή? Είδαμε παραπάνω πως αυτή η εντολή εκτελείται σε **τρία βήματα**. Ξέρουμε το χρόνο που παίρνει κάθε ένα από αυτά τα βήματα και από το χρόνο πού παίρνει κάθε βήμα, μπορούμε να υπολογίσουμε το συνολικό χρόνο εκτέλεσης της εντολής:

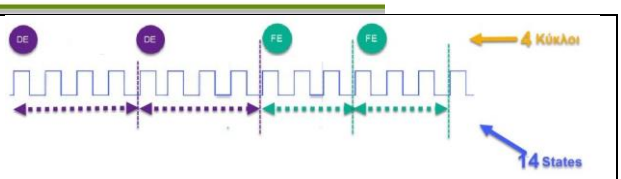
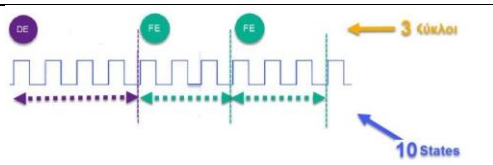
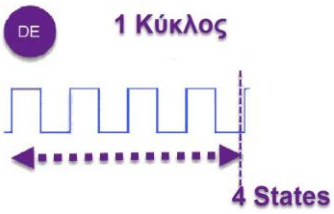
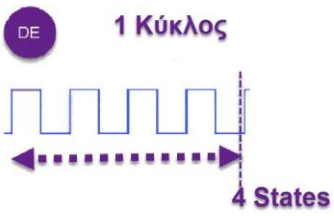
Assembly	Machine Language		
LD BC, 3240H	**00000001**	**40**	**32**



Παρόμοια, υπολογίζουμε το **χρόνο εκτέλεσης**, για κάθε άλλη εντολή στο πρόγραμμα της Άσκησης 2, στη βάση των βημάτων που περιλαμβάνει η εκτέλεσή της. Ο χρόνος για κάθε εντολή προσδιορίζεται στο **Πίνακα 2Β**.

Πίνακας 2Β: Κύκλοι & States

Assembly Language	Machine Language	Κύκλοι και States
START: LD BC,3240H	0000 01 40 32	
LD (5670H),BC	ED 43 70 56	
LD DE,(5670H)	ED 56 70 56	
LD HL,1894H	21 94 18	
LD SP, HL	F9	
LD IX,2A6FH	DD 21 6F 2A	
PUSH IX	DD E5	
PUSH DE	D5	

POP IY	**FD E1**	
POP HL	**E1**	
EX DE,HL	**EB**	
EXX	**D9**	
HALT	**76**	

NOV 11, 2020