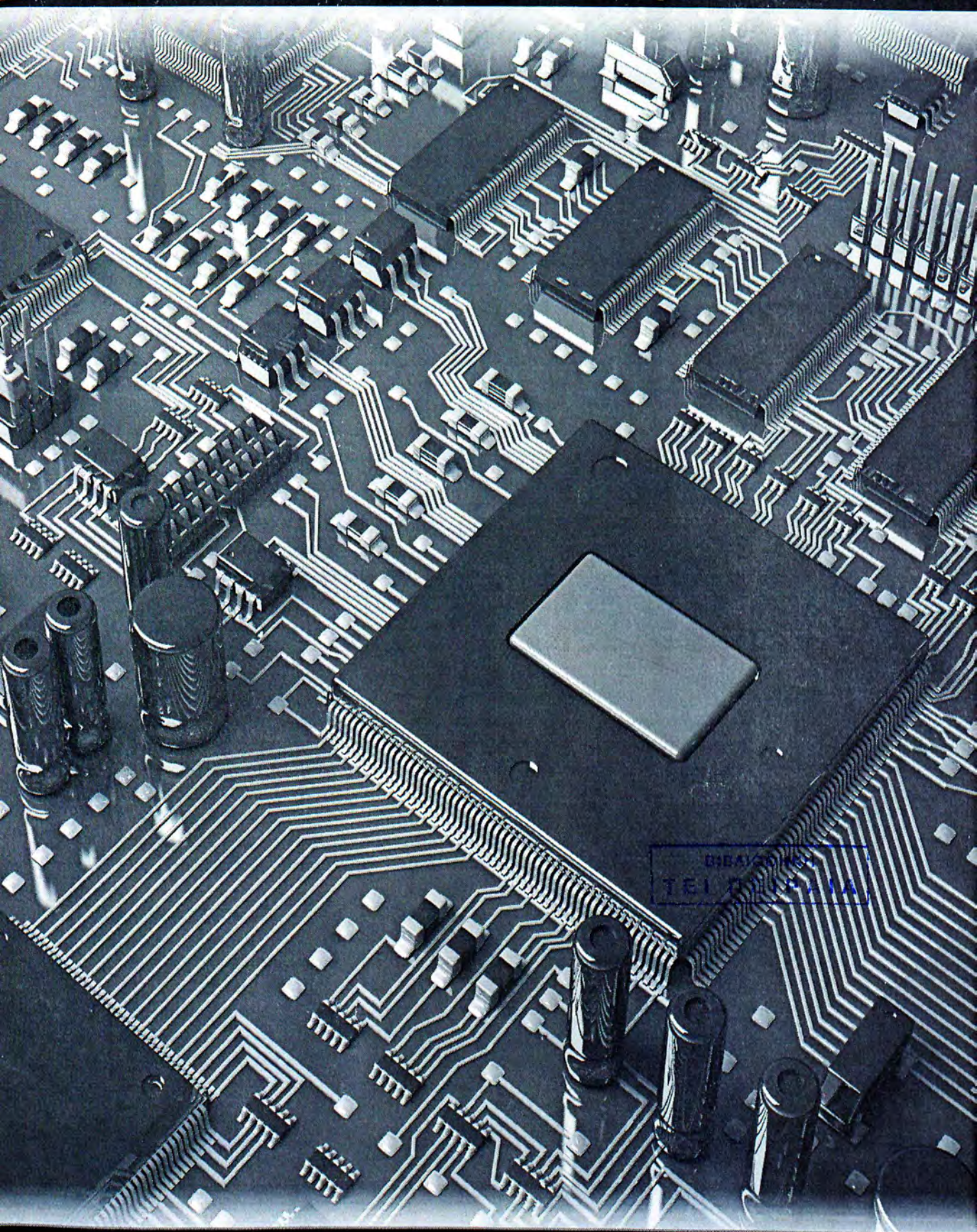
ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

χεδίαση απλού υπολογιστή με κυκλώματα SSI και MSI



Γεώργιος Χ.Ντούνας

ΠΕΡΙΕΧΟΜΕΝΑ

Πρόλογος

Σκοπός της παρούσας πτυχιακής εργασίας είναι η μελέτη και η σχεδίαση ενός απλού υπολογιστή, καθώς επίσης και η κατανόηση του τρόπου οργάνωσης και των τεχνικών σχεδίασης σύνθετων υπολογιστικών συστημάτων. Πιο συγκεκριμένα αποτελεί μία βήμα προς βήμα επεξήγηση της διαδικασίας που ακολουθήσαμε για την κατασκευή του υπολογιστή ξεκινώντας από τις λειτουργίες που επιθυμούμε να εκτελεί και καταλήγοντας στην λεπτομερή σχεδίαση και την προσομοίωση της λειτουργίας του μέσω του προγράμματος προσομοίωσης Multisim. Φυσικά προηγήθηκε η μελέτη της θεωρίας των ψηφιακών συστημάτων για αυτό και το παρόν εγχειρίδιο περιέχει βασικές γνώσεις αυτής της θεωρίας προκειμένου να γίνει περισσότερο κατανοητή η διαδικασία που ακολουθήθηκε για την σχεδίαση του συστήματος αλλά και των κυκλωμάτων που θα χρησιμοποιήσουμε για την κατασκευή του. Πρέπει να τονιστεί ότι γίνεται αναφορά κυρίως σε όσα κυκλώματα θα χρησιμοποιήσουμε στην εφαρμογή μας και όχι μια συνολική επεξήγηση όλων των βασικών ψηφιακών εξαρτημάτων. Στο cd-ROM που συνοδεύει το εγχειρίδιο μπορείτε να βρείτε το κύκλωμα του υπολογιστή έτοιμο να δεχτεί τις δικές σας εντολές και να εκτελέσει τις πράξεις που επιθυμείτε, σύμφωνα πάντα με τις δυνατότητες του. Υπάρχουν επίσης τα datasheets των ολοκληρωμένων κυκλωμάτων που χρησιμοποιήθηκαν και που παρουσιάζουν μεγάλο ενδιαφέρον. Τέλος, μπορείτε να βρείτε έναν πλήρη οδηγό του προγράμματος προσομοίωσης Multisim προκειμένου να εξοικειωθείτε με αυτό και να δημιουργήσετε τα δικά σας κυκλώματα. Για να ανοίξετε το αρχείο που περιέχει το κύκλωμα του υπολογιστή θα πρέπει να έχετε εγκατεστημένο στον υπολογιστή σας το Multisim.

Περίληψη κεφαλαίων

Στο **1° κεφάλαιο** γίνεται μία αναλυτική περιγραφή των δομικών στοιχείων τα οποία αποτελούν τους βασικούς δομικούς λίθους στη σχεδίαση ψηφιακών συστημάτων, ξεκινώντας από τις πύλες και περνώντας στη συνέχεια στα flip flop. Επίσης γίνεται μία σύντομη αναφορά στις κατηγορίες που χωρίζονται τα ψηφιακά κυκλώματα, στα χαρακτηριστικά της κάθε κατηγορίας αλλά και στη θεμελιώδη για τα ψηφιακά συστήματα έννοια του χρονισμού.

Στο **2° κεφάλαιο** με τη χρήση των στοιχείων που αναλύσαμε στο 1° κεφάλαιο κατασκευάζουμε τα εξαρτήματα τα οποία ή παραλλαγές των οποίων θα χρησιμοποιήσουμε στην κατασκευή του μικροϋπολογιστή (καταχωρητές, μετρητές κ.α.). Θα δούμε τον τρόπο λειτουργίας τους και τον τρόπο που τα συνδέουμε για να επιτύχουμε την επιθυμητή λειτουργία ανάλογα με την εφαρμογή μας.

Το **3° κεφάλαιο** αποτελεί μία πρώτη εισαγωγή στην οργάνωση του υπολογιστή που θα σχεδιάσουμε. Θα αναφερθούμε στη γλώσσα μεταφοράς καταχωρητών, μία τεχνική που θα μας βοηθήσει να επιλέξουμε τα δομικά στοιχεία που θα χρησιμοποιήσουμε, με βάση τις λειτουργίες που επιθυμούμε να εκτελεί το σύστημα και γενικά θα αποτελεί τον οδηγό μας για την υλοποίηση του συστήματος

Στο **4° κεφάλαιο** θα αναφερθούμε για πρώτη φορά στο πρόγραμμα που θα τρέχει ο υπολογιστής και στις εντολές από τις οποίες αυτό θα αποτελείται. Αποτελεί μία γενική προσέγγιση της μορφής των εντολών που μπορούν να διαχειριστούν τα κυκλώματα του υπολογιστή και του τρόπου με τον οποίο η δομή των εντολών βοηθά στην οργάνωση των κυκλωμάτων και στην διαδικασία που θα ακολουθηθεί για την εκτέλεση του προγράμματος.

Το **5ο κεφάλαιο** αποτελεί το τελευταίο βήμα πριν την ολοκλήρωση του τελικού μας σκοπού. Έχοντας αναφερθεί στα δομικά στοιχεία του υπολογιστή αλλά και στις τεχνικές για την οργάνωση του συστήματος μας θα προσπαθήσουμε να κατασκευάσουμε τις επιμέρους μονάδες από τις οποίες θα αποτελείται και θα μιλήσουμε για τις βασικές λειτουργίες και την προσφορά της κάθε μονάδας στο τελικό κύκλωμα. Επίσης θα εξηγήσουμε τη διαδικασία που εκτελεί κάθε υπολογιστής για την εκτέλεση μίας λειτουργίας και τελικά την εκτέλεση ολόκληρου του προγράμματος.

Στο **6° και τελευταίο κεφάλαιο** συνδέουμε τις μονάδες που είδαμε στα αμέσως προηγούμενα και παρουσιάζουμε το τελικό κύκλωμα του υπολογιστή. Επίσης πραγματοποιούμε τη δημιουργία ενός πρώτου προγράμματος με σκοπό να δείξουμε τον τρόπο σύνταξης των εντολών που πραγματοποιούν τις λειτουργίες που επιθυμούμε αλλά και την εισαγωγή τους στον υπολογιστή.

Κεφάλαιο 1ο
Βασικά Δομικά Στοιχεία των Ψηφιακών Συστημάτων

Όλα τα σύνθετα ψηφιακά κυκλώματα που συναντούμε στην καθημερινότητα (ηλεκτρονικοί υπολογιστές, κινητά κ.α.) κατασκευάζονται από ορισμένα βασικά δομικά στοιχεία όπως είναι οι καταχωρητές και οι πολυπλέκτες οι οποίοι με τη σειρά τους αποτελούνται από ακόμα απλούστερα δομικά στοιχεία που ονομάζονται λογικές πύλες. Στο παρόν κεφάλαιο θα παρουσιάσουμε τον τρόπο λειτουργίας των στοιχείων αυτών καθώς και τις υπηρεσίες που θα μας προσφέρουν κατά την σχεδίαση του υπολογιστή.

1.1 Λογικές πύλες

Μία λογική πύλη είναι ένα ηλεκτρονικό κύκλωμα το οποίο δέχεται στις εισόδους του δεδομένα και παράγει στην έξοδο του ένα αποτέλεσμα, δηλαδή αποτελεί την υλοποίηση σε ηλεκτρονικό κύκλωμα των διαφόρων λογικών συναρτήσεων. Οι πύλες που θα χρησιμοποιήσουμε στην εφαρμογή μας είναι οι AND, OR και Inverter.

Λογική πύλη AND

Η πύλη AND έχει δύο ή περισσότερες εισόδους και μία έξοδο και εκτελεί την λογική πράξη ΚΑΙ. Αυτό πρακτικά σημαίνει ότι για να βρεθεί σε λογικό 1 η έξοδος της θα πρέπει και οι δύο είσοδοι της να βρίσκονται σε λογικό 1. Παρακάτω φαίνεται το σύμβολο της και ο πίνακας αληθείας (Πίνακας Αληθείας: ουσιαστικά είναι ένας πίνακας ο οποίος στο ένα μέρος έχει όλους τους πιθανούς συνδυασμούς των εισόδων της πύλης και στο άλλο τις τιμές της εξόδου που αντιστοιχούν στον εκάστοτε συνδυασμό εισόδων). Το ολοκληρωμένο κύκλωμα που θα χρησιμοποιήσουμε είναι το 74LS09.

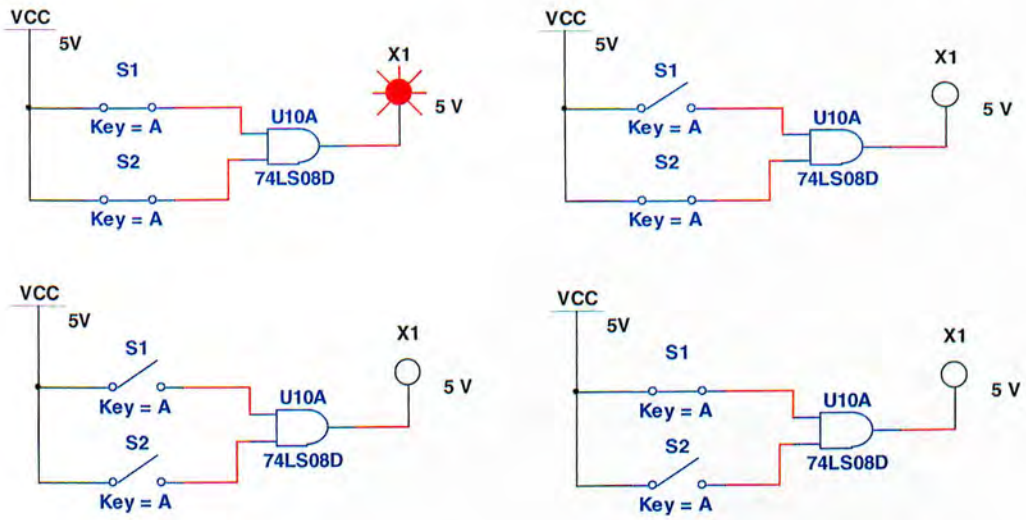
A	B	x
0	0	0
0	1	0
1	0	0
1	1	1

a) Σύμβολο b) Πίνακας Αληθείας

Σχήμα 1.1 Σύμβολο και Πίνακας Αληθείας

Για να κατανοήσουμε πλήρως τον τρόπο λειτουργίας της πύλης AND μπορούμε να σκεφτούμε ένα πολύ απλό παράδειγμα όπου στις εισόδους της πύλης συνδέουμε δύο διακόπτες και στην έξοδο της ένα λαμπάκι. Ο μόνος τρόπος για να ανάψει το λαμπάκι (να

γίνει δηλαδή η έξοδος 1) είναι να κλείσουμε και τους δύο διακόπτες (να βρεθούν δηλαδή και οι δύο σε λογικό 1). Σε κάθε άλλο συνδυασμό εισόδων η έξοδος είναι 0. Στο σχήμα 1.2 φαίνεται η κατάσταση της εξόδου για κάθε δυνατό συνδυασμό εισόδων.



Σχήμα 1.2 Προσομοίωση λειτουργίας πύλης AND

Οι πύλες AND θα μας φανούν εξαιρετικά χρήσιμες στις περιπτώσεις όπου θα θέλουμε να ισχύουν ταυτόχρονα δύο συνθήκες προκειμένου να δοθεί, μέσω της εξόδου της πύλης, το έναυσμα για την πραγματοποίηση διαφόρων λειτουργιών.

Λογική Πύλη OR

Η πύλη OR έχει, όπως φαίνεται στο σχήμα 1.3, έχει τουλάχιστον 2 εισόδους και μία έξοδο και εκτελεί τη λογική πράξη Η. Η έξοδος της δίνει λογικό 1 όταν τουλάχιστον μία από τις εισόδους της βρίσκεται σε λογική κατάσταση 1. Τις πύλες αυτές θα τις χρησιμοποιούμε όταν θα θέλουμε να πραγματοποιούνται λειτουργίες υπό την προϋπόθεση ότι τουλάχιστον μία από δύο (ή περισσότερες) συνθήκες είναι αληθής. Παρακάτω βλέπεται το σύμβολο και τον πίνακα αληθείας της πύλης OR

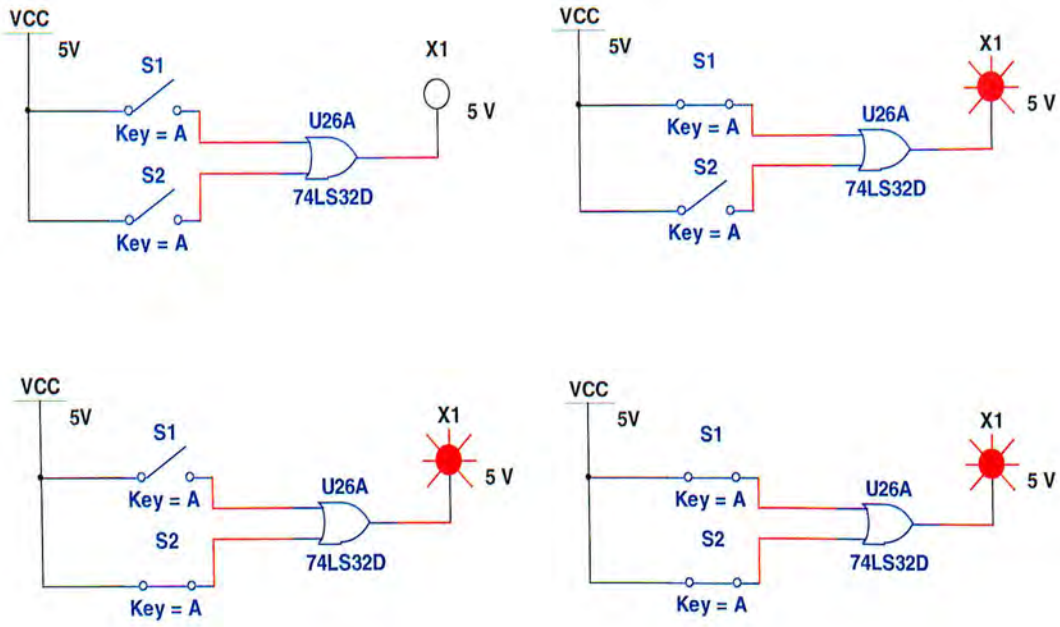
a) Σύμβολο

A	B	x
0	0	0
0	1	1
1	0	1
1	1	1

b) Πίνακας Αληθείας

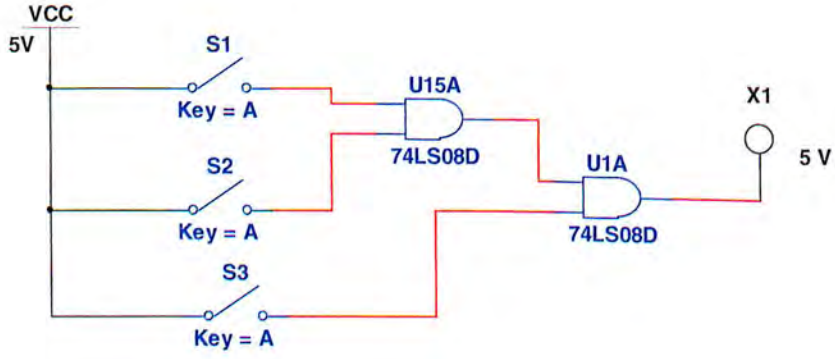
Σχήμα 1.3 Σύμβολο και Πίνακας Αληθείας πύλης OR

Ας ξαναδούμε το προηγούμενο παράδειγμα χρησιμοποιώντας πύλες OR. Η έξοδος βρίσκεται σε λογικό 1 και το λαμπάκι ανάβει όταν τουλάχιστον μία από τις εισόδους μας βρίσκεται σε λογικό 1



Σχήμα 1.4 Προσομοίωση λειτουργίας πύλης OR

Σημείωση: Στο εμπόριο υπάρχουν πύλες OR και AND με περισσότερες από δύο εισόδους. Στην εφαρμογή μας σε περίπτωση που θέλουμε να εξετάσουμε τρεις συνθήκες θα χρησιμοποιούμε δύο πύλες συνδεδεμένες σε σειρά. Για παράδειγμα στο σχήμα 1.5 εάν οι είσοδοι της 1^{ης} πύλης ΚΑΙ είναι αληθείς τότε η έξοδος της πύλης βρίσκεται σε λογικό 1, επομένως η πρώτη είσοδος της δεύτερης πύλης βρίσκεται και αυτή σε λογικό 1. Για να ανάψει το led θα πρέπει να τεθεί σε λογικό 1 και η δεύτερη είσοδος της δεύτερης πύλης ΚΑΙ. Το κύκλωμα μας, επομένως, δίνει στην έξοδο του 1 μόνο όταν και οι τρεις είσοδοι του βρίσκονται σε λογικό 1. Σε περίπτωση που θέλουμε να έχουμε στην έξοδο του κυκλώματος λογικό 1 όταν βρίσκεται σε λογικό 1 τουλάχιστον μία από τις τρεις εισόδους του, απλά αντικαθιστούμε τις πύλες ΚΑΙ με πύλης OR.



Σχήμα 1.5 Έλεγχος τριών συνθηκών

Λογική Πύλη NOT

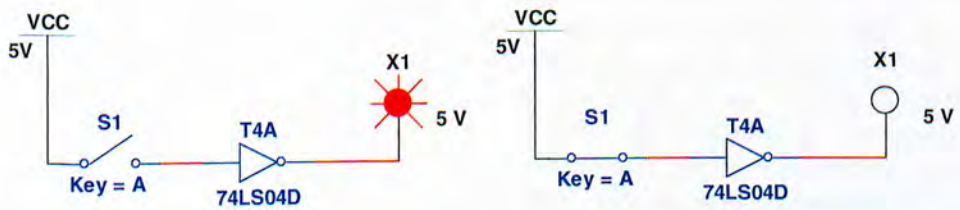
Η πύλη NOT ή αναστροφέας έχει μόνο μία είσοδο και μία έξοδο και εκτελεί τη λογική πράξη της αντιστροφής. Ουσιαστικά η πύλη αντιστρέφει την τιμή της εισόδου της (εάν η είσοδος είναι λογικό 1 η έξοδος γίνεται λογικό 0, εάν η είσοδος είναι λογικό 0 η έξοδος γίνεται λογικό 1). Στο σχήμα 1.6 βλέπεται το σύμβολο και τον πίνακα αληθείας της πύλης NOT. Ο μικρός κύκλος στην έξοδο του ολοκληρωμένου 74LS05 σηματοδοτεί ότι το σήμα στην έξοδο της πύλης είναι το αντίθετο της εισόδου του.

A	x
0	1
1	0

a) Σύμβολο b) Πίνακας Αληθείας

Σχήμα 1.6 Σύμβολο και Πίνακας Αληθείας πύλης NOT

Όπως φαίνεται και από το σχήμα 1.7 όταν ο διακόπτης είναι ανοιχτός, η είσοδος του αναστροφέα βρίσκεται σε λογικό 0, όμως η έξοδος του τίθεται σε λογικό 1, εξαιτίας της λειτουργίας της πύλης NOT. Προφανώς όταν κλείσουμε σε διακόπτη η πύλη θα αναστρέψει το σήμα που δέχεται στην είσοδο της και το led θα είναι σβηστό.



1.7 Προσομοίωση λειτουργίας πύλης NOT

1.2 Συνδυαστικά και Ακολουθιακά Κυκλώματα

Στο σημείο αυτό θεωρούμε σκόπιμο να μιλήσουμε για τις κατηγορίες στις οποίες χωρίζονται τα ψηφιακά κυκλώματα προκειμένου να εξετάσουμε τα διαφορετικά χαρακτηριστικά τους και ορισμένες θεμελιώδεις λειτουργίες τους.

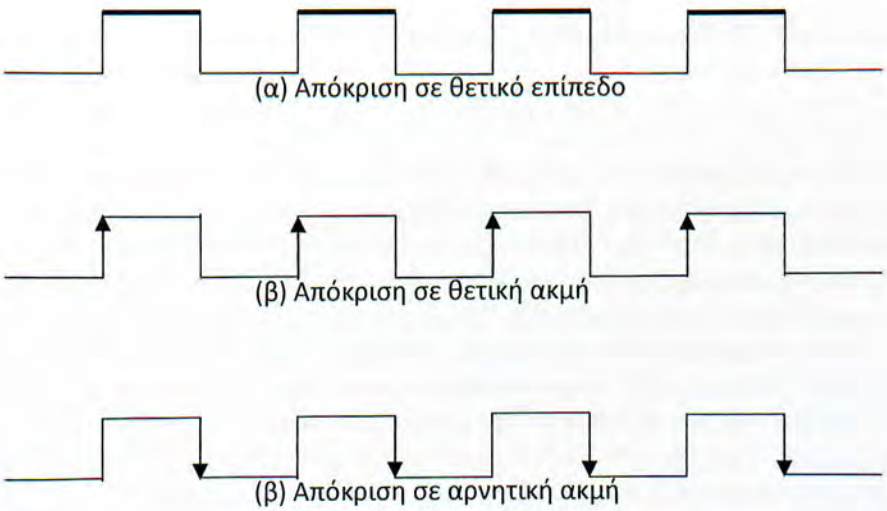
Τα ψηφιακά κυκλώματα διακρίνονται σε δύο μεγάλες κατηγορίες, στα συνδυαστικά και στα ακολουθιακά. Ένα συνδυαστικό κύκλωμα συντίθεται από λογικές πύλες, των οποίων οι έξοδοι καθορίζονται, ανά πάσα στιγμή, μόνο από τον τρέχοντα συνδυασμό των εισόδων. Τα κυκλώματα που έχουμε δει μέχρι τώρα , είτε οι πύλες είτε ο συνδυασμός τους, είναι συνδυαστικά. καθώς το αποτέλεσμα στις εξόδους τους ήταν συνάρτηση μόνο των διαφόρων τιμών που θέταμε στις εισόδους τους.

Στα ακολουθιακά κυκλώματα, σε αντίθεση με τα συνδυαστικά, χρησιμοποιούνται και στοιχεία μνήμης, δηλαδή εξαρτήματα τα οποία έχουν την ικανότητα να αποθηκεύουν δυαδικές πληροφορίες (Σε κάθε στοιχείο μνήμης αποθηκεύεται 1 bit δυαδικής πληροφορίας). Ανά πάσα στιγμή, οι λογικές καταστάσεις (0 ή 1) που είναι αποθηκευμένες στα στοιχεία μνήμης παρουσιάζονται στις εξόδους τους και χαρακτηρίζουν την κατάσταση τους. Γίνεται εύκολα κατανοητή η θεμελιώδης σημασία που έχει για τις εφαρμογές μας η δυνατότητα αυτή που μας δίνουν τα στοιχεία μνήμης, καθώς προκειμένου να επεξεργαστούμε τα διάφορα δεδομένα θα πρέπει πρώτα να τα αποθηκεύσουμε. Μία σημαντική διαφορά μεταξύ ακολουθιακών και συνδυαστικών κυκλωμάτων έγκειται στο γεγονός ότι οι έξοδοι των ακολουθιακών κυκλωμάτων εξαρτώνται όχι μόνο από τις εισόδους, αλλά και από την κατάσταση στην οποία βρίσκονται τα στοιχεία μνήμης (από τις υπάρχουσες-αποθηκευμένες σε αυτά τιμές) που διαθέτουν. Δηλαδή η επόμενη τιμή που θα αποθηκευτεί σε ένα στοιχείο μνήμης (ή με άλλα λόγια η επόμενη κατάσταση του) δεν είναι συνάρτηση μόνο των εισόδων του (όπως στα συνδυαστικά κυκλώματα), αλλά και της προηγούμενης τιμής που ήταν αποθηκευμένη σε αυτό. Αυτό επιτυγχάνεται με την ύπαρξη βρόγχου ανάδρασης από την έξοδο των στοιχείων μνήμης στην είσοδο του συνολικού κυκλώματος. Σημειώνουμε ότι ένα ακολουθιακό κύκλωμα μπορεί να έχει και συνήθως έχει και συνδυαστικό μέρος (λογικές πύλες κλπ). Αντίθετα ένα συνδυαστικό κύκλωμα δεν διαθέτει στοιχεία μνήμης.

Τα ακολουθιακά κυκλώματα με τη σειρά τους διακρίνονται σε δύο κατηγορίες, τα σύγχρονα και τα ασύγχρονα (με τα οποία δε θα ασχοληθούμε). Ο παράγοντας χρονισμός είναι μία πολύ βασική έννοια των ψηφιακών συστημάτων γι' αυτό και θα προσπαθήσουμε να εξηγήσουμε με απλό τρόπο τη λειτουργία του. Μέχρι τώρα όταν εφαρμόζαμε ένα σήμα στην είσοδο μίας πύλης βλέπαμε αυτομάτως (με κάποια μικρή χρονική καθυστέρηση εξαιτίας του χρόνου που απαιτείται για τη μετάδοση των σημάτων μέσα από τα εξαρτήματα) το αποτέλεσμα στην έξοδο του. Τα σύγχρονα ακολουθιακά κυκλώματα διαθέτουν μία επιπλέον είσοδο, την είσοδο χρονισμού. Για το συγχρονισμό των πολλών ακολουθιακών λογικών στοιχείων που υπάρχουν σε ένα ψηφιακό σύστημα χρησιμοποιείται μία κοινή παλμοσειρά παραγόμενη από ένα κύκλωμα χρονισμού (ένα ρολόι), η οποία συνδέεται με όλα τα σύγχρονα κυκλώματα της εφαρμογής μας και συγχρονίζει τις διεργασίες τους. Οι παλμοί του ρολογιού διανέμονται σε όλο το σύστημα με τέτοιο τρόπο ώστε τα στοιχεία μνήμης να επηρεάζονται μόνο κατά τη στιγμή της άφιξης κάθε νέου παλμού χρονισμού. Πρακτικά οι παλμοί του ρολογιού καθορίζουν μόνο πότε θα γίνουν αλλαγές στην κατάσταση του συστήματος και όχι ποιες ακριβώς αλλαγές θα γίνουν. Για παράδειγμα εάν θέλουμε να αποθηκεύσουμε σε ένα στοιχείο μνήμης ένα bit δυαδικής πληροφορίας πρέπει να θέσουμε στις εισόδους δεδομένων του τα κατάλληλα σήματα. Τη χρονική στιγμή που θέτουμε τα σήματα στις εισόδους δεδομένων του δε θα παρατηρήσουμε καμία αλλαγή στα περιεχόμενα του (και ως εκ τούτου στις εξόδους του). Η αποθήκευση των πληροφοριών που υπάρχουν στις εισόδους του θα γίνει μόνο όταν εμφανιστεί ο επόμενος παλμός του ρολογιού. Στα σύγχρονα ακολουθιακά κυκλώματα ο συγχρονισμός της δραστηριότητας και η τελική ενημέρωση των στοιχείων μνήμης γίνονται με το σήμα του ρολογιού.

Η λειτουργία ολόκληρου του υπολογιστικού μας συστήματος είναι άμεσα εξαρτημένη από τη γεννήτρια της παλμοσειράς (το ρολόι), η οποία είναι υπεύθυνη για την συγχρονισμένη λειτουργία όλων των στοιχείων του συστήματος μας. Αυτό που πρέπει να κατανοήσουμε είναι ότι η παλμοσειρά του ρολογιού χρησιμοποιείται ώστε να συγχρονίζεται η λειτουργία των κυκλωμάτων του συστήματος. Στο σχήμα 1.8 απεικονίζεται η παλμοσειρά του ρολογιού. Υπάρχουν ηλεκτρονικά στοιχεία τα οποία ανταποκρίνονται στο επίπεδο του παλμού του ρολογιού, δηλαδή κατά τη διάρκεια που ο παλμός βρίσκεται σε λογικό 1 (θετικό επίπεδο) ή λογικό 0 (αρνητικό επίπεδο). Στο σχήμα 1.8α βλέπεται το θετικό επίπεδο του παλμού. Ωστόσο εξαιτίας προβλημάτων που παρουσίασε η χρήση αυτών των

κυκλωμάτων σχεδιάστηκαν ηλεκτρονικά στοιχεία τα οποία έχουν την ικανότητα να αλλάζουν κατάσταση μόνο κατά τη μετάβαση του σήματος χρονισμού από 0 σε 1 (θετική ακμή) ή από 1 σε 0 (αρνητική ακμή). Το χρονικό διάστημα που διαρκεί η μετάβαση του παλμού από τη μία λογική κατάσταση στην άλλη είναι πολύ μικρότερο από τη χρονική διάρκεια του επιπέδου του ρολογιού. Αυτό σημαίνει πρακτικά ότι τα στοιχεία μνήμης μένουν για μεγάλο χρονικό διάστημα εκτεθειμένα στα σήματα που δέχονται στις εισόδους τους, με αποτέλεσμα πολλές φορές να μη μπορεί να τοποθετηθεί η επιθυμητή τιμή στο εσωτερικό τους, καθώς είναι πιθανότερο να υπάρξει κάποια μη επιθυμητή μεταβολή - αστάθεια στα σήματα εισόδου κατά το μεγαλύτερο αυτό χρονικό διάστημα. Τα στοιχεία μνήμης τα οποία επηρεάζονται από τα επίπεδα του σήματος ρολογιού ονομάζονται μανδαλωτές, ενώ αυτά που επηρεάζονται από τις ακμές ονομάζονται flip-flop. Στα 1.8β και 1.8γ βλέπεται την απόκριση στις ακμές του παλμού χρονισμού.

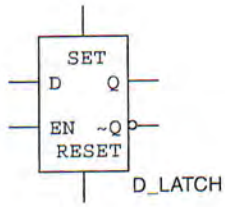


Σχήμα 1.8 Αποκρίσεις στο επίπεδο και στις ακμές του σήματος ρολογιού

Σημείωση: Ο κύρια αιτία σχεδίασης κυκλωμάτων ευαίσθητων στις ακμές του σήματος χρονισμού είναι εξαιτίας του βρόγχου ανάδρασης που διαθέτουν τα ακολουθιακά κυκλώματα, πράγμα που δε θεωρούμε σκόπιμο να αναλύσουμε στο συγκεκριμένο εγχειρίδιο.

Μανδαλωτές

Οι μανδαλωτές είναι στοιχεία μνήμης τα οποία ενεργοποιούνται από την παρουσία συγκεκριμένων επιπέδων του σήματος χρονισμού. Οι μανδαλωτές δεν είναι ιδιαίτερα χρήσιμοι στην κατασκευή σύγχρονων ακολουθιακών κυκλωμάτων εξαιτίας της ιδιότητας τους να ανταποκρίνονται στα επίπεδα του σήματος (δημιουργούνται προβλήματα εξαιτίας του βρόγχου ανάδρασης). Ωστόσο η χρήση ενός μανδαλωτή θα μας βοηθήσει στην κατασκευή της εφαρμογής μας καθώς θα χρησιμοποιηθεί για την έναρξη της λειτουργίας του υπολογιστή. Υπάρχουν διάφοροι τύποι μανδαλωτών, εμείς θα ασχοληθούμε μόνο με τον μανδαλωτή τύπου D. Στο σχήμα 1.9 φαίνεται το σύμβολο και ο πίνακας αληθείας ενός μανδαλωτή D.



a) Σύμβολο

EN	D	SET	RESET	Q
0	0	0	0	0
0	1	0	0	0
1	1	0	0	1
1	0	0	0	0
0	0	1	0	1
0	0	0	0	1
0	0	0	1	0
0	0	0	0	0

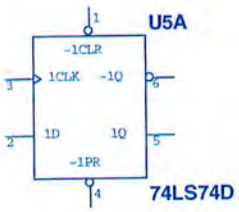
b) Πίνακας Αληθείας

Σχήμα 1.9 Σύμβολο και Πίνακας Αληθείας πύλης NOT

Ο συμπεριφορά του μανδαλωτή τύπου D είναι πολύ απλή. Όπως φαίνεται και στο σχήμα διαθέτει 6 ακροδέκτες (4 εισόδους και 2 εξόδους.). Αρχικά βλέπουμε την είσοδο D (data) στην οποία τοποθετούμε τη δυαδική πληροφορία που θέλουμε να αποθηκευτεί στον μανδαλωτή. Ωστόσο υπάρχει μία συνθήκη για να αποθηκευτούν τα δεδομένα που τοποθετούμε στην είσοδο D στο εσωτερικό του. Η συνθήκη αυτή είναι η ενεργοποίηση της εισόδου EN, η οποία ονομάζεται είσοδος επίτρεψης. Μόνο όταν η είσοδος επίτρεψης έχει τιμή 1 ο μανδαλωτής "βλέπει" τα δεδομένα που θέτουμε στην είσοδο του D. Όπως φαίνεται και στον πίνακα αληθείας όταν EN=1 η έξοδος ακολουθεί τις τιμές της εισόδου (όταν D=1 τότε Q=1, ενώ όταν D=0 τότε Q=0). Όταν EN=0 οι τιμές που θέτουμε στην είσοδο D δεν επηρεάζουν την κατάσταση του μανδαλωτή. Οι άλλες δύο είσοδοι SET και RESET είναι και αυτές που μας ενδιαφέρουν περισσότερο, καθώς θα τις αξιοποιήσουμε για την έναρξη της λειτουργίας του υπολογιστή. Όταν η είσοδος SET αποκτήσει τιμή 1 η έξοδος του μανδαλωτή γίνεται 1 και θα παραμείνει σε αυτή την κατάσταση ακόμα και αν ο είσοδος SET γίνει 0. Για να μηδενιστεί το περιεχόμενο του μανδαλωτή θα πρέπει να ενεργοποιήσουμε την είσοδο RESET. Οι είσοδοι SET και RESET είναι ανεξάρτητες από την είσοδος επίτρεψης EN. Τέλος η έξοδος Q' δεν είναι τίποτα άλλο από το αντίστροφο σήμα της εξόδου Q (Όταν η έξοδος Q βρίσκεται σε λογικό 1, τότε η έξοδος Q' βρίσκεται σε λογικό 0 και το αντίστροφο).

Flip-Flop

Τα flip-flop είναι στοιχεία μνήμης που ενεργοποιούνται από μεταβάσεις τιμών του σήματος ρολογιού. Οι τιμές που θέτουμε στις εισόδους ελέγχου των flip-flop επηρεάζουν την κατάσταση τους μόνο κατά τη μετάβαση του παλμού του ρολογιού από λογικό 0 σε λογικό 1 ή και το αντίστροφο. Ένα flip flop είναι μία διάταξη που μπορεί να αποθηκεύσει 1 bit δυαδικής πληροφορίας. Χρησιμοποιώντας, επομένως, τον αναγκαίο αριθμό flip flop μπορούμε να αποθηκεύσουμε το επιθυμητό πλήθος δυαδικών ψηφίων. Για παράδειγμα, για την αποθήκευση 1 byte, ήτοι 8 bit δυαδικής πληροφορίας, αρκεί να χρησιμοποιήσουμε 8 flip-flop. Αν αναλογιστούμε πόσο σημαντική είναι η αποθήκευση πληροφοριών σε κάθε υπολογιστικό σύστημα μπορούμε να καταλάβουμε τη σημασία που έχουν στην ψηφιακή σχεδίαση αυτά τα εξαρτήματα. Στο παρακάτω σχήμα 1.10 φαίνεται το ολοκληρωμένο και ο πίνακας αληθείας ενός D flip-flop που είναι και αυτό που κυρίως μας ενδιαφέρει.



a) Ολοκληρωμένο

D	Q(t+1)
0	0
1	1

b) Πίνακας αληθείας

Σχήμα 1.10 Ολοκληρωμένο κύκλωμα και Πίνακας Αληθείας D flip-flop

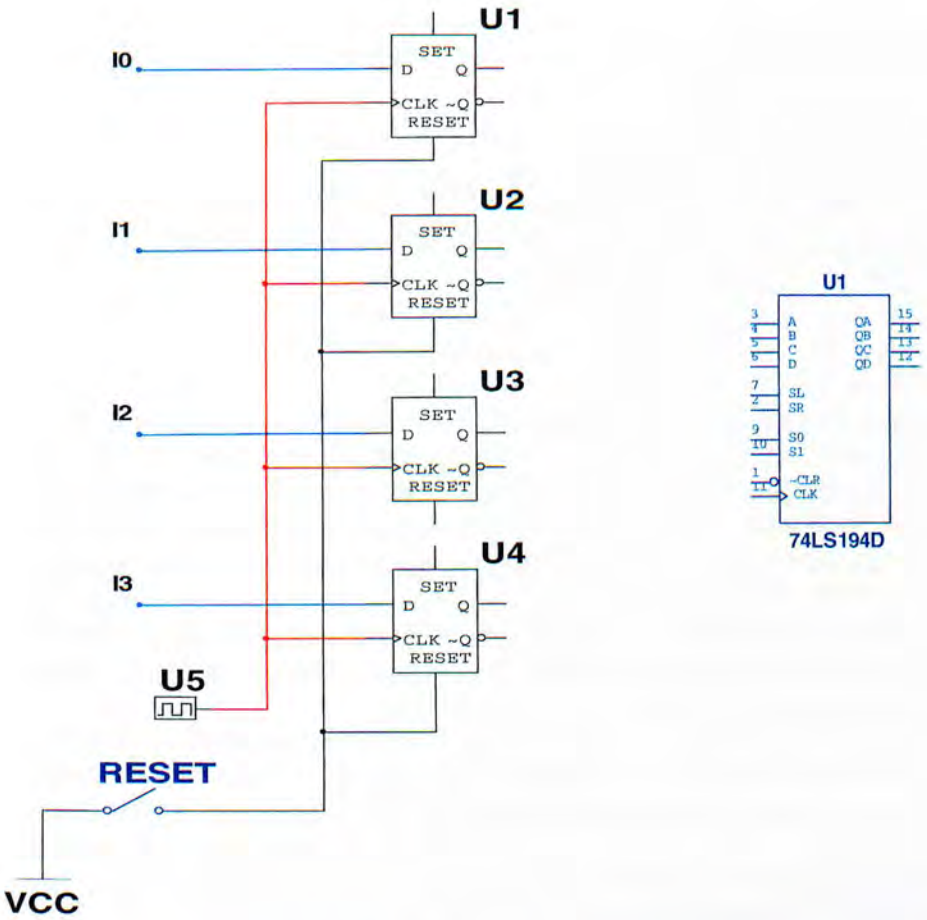
Όπως θα παρατηρείται εκτός από τους ακροδέκτες εισόδου, εξόδου και χρονισμού υπάρχουν 3 ακόμα ακροδέκτες στο flip flop. Η γνωστή μας πλέον έξοδος Q' την οποία διαθέτει και το μανδαλωτής που είδαμε προηγουμένως αλλά και οι είσοδοι CLR και PR. Η μεν είσοδος CLR κατά την ενεργοποίηση της "καθαρίζει" το περιεχόμενο του flip flop (το κάνει 0), ενώ η είσοδος PR πραγματοποιεί πρωτοποθέτηση του flip flop (κάνει το περιεχόμενο του 1). Αυτές οι δυο έχουν ένα κοινό χαρακτηριστικό που είναι ο μικρός κύκλος στην είσοδο τους. Το σύμβολο αυτό δείχνει ότι οι συγκεκριμένες είσοδοι ενεργοποιούνται από επίπεδο σήματος λογικού 0. Επομένως όταν εμείς θα θέλουμε να καθαρίσουμε το περιεχόμενο του flip flop θα πρέπει να οδηγήσουμε στην είσοδο του CLR λογικό 0. Το συγκεκριμένο χαρακτηριστικό θα το δούμε και σε άλλους ακροδέκτες διαφόρων εξαρτημάτων που θα χρησιμοποιήσουμε. Η παρούσα της κανονικής εξόδου ενός flip-flop συμβολίζεται με Q(t) ενώ η επόμενη κατάσταση σου με Q(t+1). Δηλαδή από τον πίνακα αληθείας του D flip-flop συμπεραίνουμε ότι όταν θέσουμε 1 στην είσοδο του με την έλευση του επόμενου παλμού του ρολογιού η κατάσταση της εξόδου του θα γίνει 1 ανεξάρτητα από την τιμή της εξόδου του πριν την έλευση του παλμού. Τα αντίστοιχα συμβαίνουν και όταν τεθεί 0 στην είσοδο του.

Κεφάλαιο 2ο
Σύνθετα Ψηφιακά Κυκλώματα

Ένα πολυσύνθετο σύστημα όπως οι υπολογιστές καλείται να εκτελέσει περίπλοκες λειτουργίες οι οποίες δε θα μπορούσαν να πραγματοποιηθούν μόνο με τη χρήση των απλών κυκλωμάτων που είδαμε στο 1° κεφάλαιο. Σε αυτό το 2° κεφάλαιο αξιοποιώντας τα κυκλώματα που είδαμε και αναλύσαμε θα χτίσουμε σύνθετα εξαρτήματα τα οποία εκτελούν διάφορες λειτουργίες όπως η αποθήκευση πληροφοριών, η μέτρηση προς τα άνω ή προς τα κάτω και άλλα, τα οποία θα μας φανούν εξαιρετικά χρήσιμα στη σύνθεση του υπολογιστή μας.

2.1 Καταχωρητές

Όπως αναφέραμε και στο προηγούμενο κεφάλαιο για την αποθήκευση ενός επιθυμητού πλήθους δυαδικών ψηφίων αρκεί να χρησιμοποιήσουμε τόσα flip flop όσο το μήκος της προς αποθήκευση πληροφορίας. Μία ομάδα από flip flop ονομάζεται καταχωρητής. Ένας n-bit καταχωρητής είναι μία ομάδα αποτελούμενη από n flip flop και έχει την ικανότητα να αποθηκεύει μία πληροφορία μήκους n bits. Στο παράδειγμα όπου θέλαμε να αποθηκεύσουμε 1 byte πληροφορίας χρησιμοποιήσαμε 8 flip flop. Αυτά τα 8 flip flop μπορούμε να τα δούμε σαν μία ομάδα-σύνολο το οποίο συνθέτει έναν 8-bit καταχωρητή (δηλαδή ένα κουτί αποθήκευσης 8-bit δεδομένων). Στο σχήμα 2.1.a φαίνεται η εσωτερική δομή ενός 4-bit καταχωρητή και στο 2.1.b το ολοκληρωμένο κύκλωμα που θα χρησιμοποιήσουμε.



Σχήμα 2.1 a) 4-bit Καταχωρητής **b) Ολοκληρωμένο κύκλωμα 4-bit καταχωρητή**

Όλα τα flip flop του καταχωρητή είναι συνδεδεμένα με το κοινό ρολόι του συστήματος. Όταν όλα τα flip flop του καταχωρητή φορτώνονται (αποθηκεύονται πληροφορίες στο εσωτερικό τους) ταυτόχρονα με την έλευση ενός κοινού παλμού ρολογιού, λέμε ότι η φόρτωση γίνεται παράλληλα. Στις διάφορες εφαρμογές θα υπάρχουν συχνά περιπτώσεις όπου δεν θα θέλουμε με κάθε παλμό του ρολογιού να φορτώνονται όλοι οι καταχωρητές τους συστήματος μας. Για το λόγο αυτό κάθε καταχωρητής έχει μία επιπλέον είσοδο, η οποία ονομάζεται είσοδος φόρτωσης. Η είσοδος φόρτωσης καθορίζει αν στον επόμενο παλμό ο καταχωρητής θα δεχθεί νέες πληροφορίες ή θα διατηρήσει τις πληροφορίες του αναλλοίωτες. Όπως έχουμε πει σε έναν καταχωρητή οι παλμοί του ρολογιού εφαρμόζονται συνεχώς στις εισόδους χρονισμού του. Όμως το αν τα δεδομένα που εφαρμόζονται στις εισόδους του καταχωρητή θα αποθηκευθούν στο εσωτερικό του εξαρτάται από την τιμή της εισόδου φόρτωσης. Όταν αυτή είναι 1, τα δεδομένα στις τέσσερις εισόδους εισάγονται στον καταχωρητή με την άφιξη του επόμενου παλμού του ρολογιού. Στο ολοκληρωμένο 74LS194 που θα χρησιμοποιήσουμε για να επιτευχθεί η λειτουργία της φόρτωσης πρέπει να θέσουμε τις εισόδους S0 και S1 σε λογικό 1. Οι ακροδέκτες A έως D είναι οι ακροδέκτες δεδομένων. Σε αυτούς οδηγούμε τα δεδομένα που θέλουμε να φορτώσουμε παράλληλα στον καταχωρητή. Οι ακροδέκτες SL και SR χρησιμοποιούνται για την λειτουργία της ολίσθησης η οποία δε θα μας χρειαστεί. Μέσω του CLR επιτυγχάνεται ο καθαρισμός του καταχωρητή. Παρατηρούμε και πάλι τον μικρό κύκλο στην είσοδο CLR που υποδηλώνει ότι η συγκεκριμένη είσοδος ενεργοποιείται όταν βρεθεί σε λογική κατάσταση 0. Οι ακροδέκτες

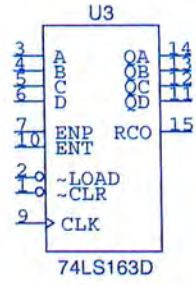
QA έως QD είναι οι ακροδέκτες εξόδου όπου εμφανίζονται τα δεδομένα που είναι αποθηκευμένα στον καταχωρητή. Τέλος, η είσοδος CLK είναι η γνωστή μας είσοδος χρονισμού.

Σημείωση: Πρέπει να κατανοήσουμε ότι η λογική κατάσταση που είναι αποθηκευμένη σε ένα flip flop (άρα και σε οποιοδήποτε εξάρτημα συντίθεται από αυτά) εμφανίζεται ανά πάσα στιγμή στην έξοδο του.

2.2 Μετρητές

Οι μετρητές είναι μία ειδική κατηγορία καταχωρητών οι οποίοι έχουν την ικανότητα να μεταβάλλουν το περιεχόμενο τους με την εφαρμογή παλμών εισόδου. Με τον όρο μεταβολή του περιεχομένου τους ουσιαστικά εννοούμε ότι με την εφαρμογή παλμών στις εισόδους χρονισμού αυξάνουν το περιεχόμενο τους κατά ένα. Για παράδειγμα όταν ένας 4-bit μετρητής έχει περιεχόμενο τον δυαδικό αριθμό 0000 με την έλευση του επόμενου παλμού του ρολογιού θα αποκτήσει περιεχόμενο 0001, στη συνέχεια 0010 κλπ μέχρις ότου φτάσει στο 1111, όπου με τον επόμενο παλμό θα επανέλθει στην αρχική του κατάσταση, δηλαδή 0000. Τους μετρητές θα τους χρησιμοποιήσουμε στην εφαρμογή μας προκειμένου να δημιουργήσουμε τους παλμούς ρολογιού του συστήματος.

Όπως αναφέραμε, οι μετρητές είναι μία ειδική κατηγορία καταχωρητών. Οι δυαδικοί μετρητές με παράλληλη φόρτωση είναι μετρητές οι οποίοι εκτός από την ικανότητα δυαδικής αρίθμησης, μπορούν να φορτώσουν παράλληλα τις πληροφορίες που υπάρχουν στις εισόδους δεδομένων τους, ακριβώς όπως και οι καταχωρητές για τους οποίους μιλήσαμε στην προηγούμενη παράγραφο. Στο σχήμα 2.2 φαίνεται το ολοκληρωμένο ενός δυαδικού μετρητή 4-bit με παράλληλη φόρτωση που θα χρησιμοποιήσουμε.

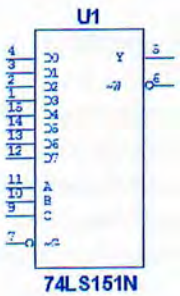


Σχήμα 2.2 Ολοκληρωμένο κύκλωμα 4-bit δυαδικού μετρητή

Ας επικεντρωθούμε στους ακροδέκτες του ολοκληρωμένου 74LS163 που χρησιμοποιούμε στο πρόγραμμα προσομοίωσης. Όπως παρατηρούμε οι ακροδέκτες εισόδου και εξόδου δεδομένων, CLK και CLR είναι ίδιοι με αυτούς του καταχωρητή που εξετάσαμε προηγουμένως. Πάνω από το CLR υπάρχει η είσοδος φόρτωσης LOAD με την οποία φορτώνεται παράλληλα ο μετρητής. Οι ακροδέκτες ENP και ENT χρησιμοποιούνται για την λειτουργίας της αρίθμησης. Για να ξεκινήσει ο μετρητής τη λειτουργία της αρίθμησης θα πρέπει και στις τέσσερις εισόδους (LOAD, CLR, ENP, ENT) να τοποθετηθεί λογικό 1.

2.3 Πολυπλέκτες

Σε ένα υπολογιστικό σύστημα κεντρικό ρόλο παίζει η επικοινωνία μεταξύ των διαφορετικών μονάδων από τις οποίες αποτελείται. Φανταστείτε έναν υπολογιστή όπου δεν υπάρχει τρόπος επικοινωνίας μεταξύ της μονάδας εισόδου (π.χ πληκτρολόγιο) και της μνήμης του. Στο εσωτερικό του συστήματος μία πολύ βασική λειτουργία είναι η μεταφορά δεδομένων μεταξύ των καταχωρητών του. Όταν θέλουμε να μεταφέρουμε τα δεδομένα ενός καταχωρητή σε έναν άλλο απλά συνδέουμε τα bit ίδιας τάξης των δύο καταχωρητών. Έτσι όταν ενεργοποιηθεί η λειτουργία φόρτωσης του καταχωρητή-προορισμού τα δεδομένα του καταχωρητή-πηγή θα μεταφερθούν σε αυτόν. Ας υποθέσουμε ότι έχουμε δύο καταχωρητές-πηγές (R1 και R2) οι οποίοι θέλουμε να μεταφέρουν τα περιεχόμενα τους σε έναν τρίτο καταχωρητή-προορισμό (R3). Πως θα γίνει η σύνδεση των αντίστοιχων bit? Ποιού καταχωρητή τα δεδομένα θα λάβει ο R3 όταν ενεργοποιηθεί η λειτουργία παράλληλης φόρτωσης? Τη λύση σε αυτό δίνουν οι πολυπλέκτες. Οι πολυπλέκτες είναι εξαρτήματα που διαθέτουν πολλές γραμμές εισόδου και μία γραμμή εξόδου. Η ιδιαίτερη λειτουργία τους, που τους καθιστά τόσο σημαντικούς, είναι ότι μας επιτρέπουν να επιλέξουμε ποιάς εισόδου τα δεδομένα θα εμφανιστούν κάθε φορά στην μοναδική του έξοδο. Στο σχήμα 2.3 φαίνεται ένας 8 σε 1 πολυπλέκτης (8 είσοδοι σε μια έξοδο)

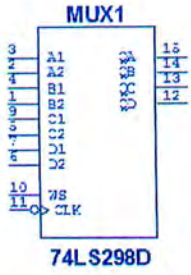


Σχήμα 2.3 Ολοκληρωμένο κύκλωμα 8-σε-1 πολυπλέκτη

Στις εισόδους D0-D7 τοποθετούμε τα δεδομένα από τα οποία θα επιλέγουμε ένα κάθε φορά για να εμφανιστούν στην έξοδο του. Ποιο δεδομένο θα περάσει στην έξοδο Y εξαρτάται από το δυαδικό αριθμό που τοποθετούμε στις εισόδους A,B,C οι οποίες για αυτό το λόγο ονομάζονται είσοδοι επιλογής. Κάθε είσοδος έχει έναν δείκτη 0,1,2...7. Για να δημιουργήσουμε το μονοπάτι μεταξύ μίας εισόδου και της εξόδου πρέπει στις εισόδους επιλογής να οδηγήσουμε το δυαδικό αριθμό που αντιστοιχεί στον δείκτη της εισόδου. Για παράδειγμα όταν θέλουμε να μεταφέρουμε στην έξοδο το περιεχόμενο της εισόδου D5 αρκεί να οδηγήσουμε στις A,B,C το δυαδικό ισοδύναμο του 5. Το A αποτελεί το λιγότερο σημαντικό ψηφίο και το C το περισσότερο σημαντικό ψηφίο. Επομένως για την D5 (5=101) οι τιμές των εισόδων επιλογής είναι C=1, B=0, A=1. Γίνεται, λοιπόν, εύκολα κατανοητό ότι ο αριθμός των εισόδων επιλογής εξαρτάται άμεσα από τον αριθμό των εισόδων δεδομένων. Εφόσον έχουμε 8 εισόδους δεδομένων και χρειαζόμαστε για κάθε μία από αυτές ένα διαφορετικό δυαδικό αριθμό, για την αναπαράσταση 8 διαφορετικών συνδυασμών bit απαιτούνται δυαδικοί αριθμοί μήκους 3 bit (Βάση του δυαδικού συστήματος είναι το 2, αριθμός εισόδων 8 άρα 8=2^3 επομένως 3 είσοδοι επιλογής)

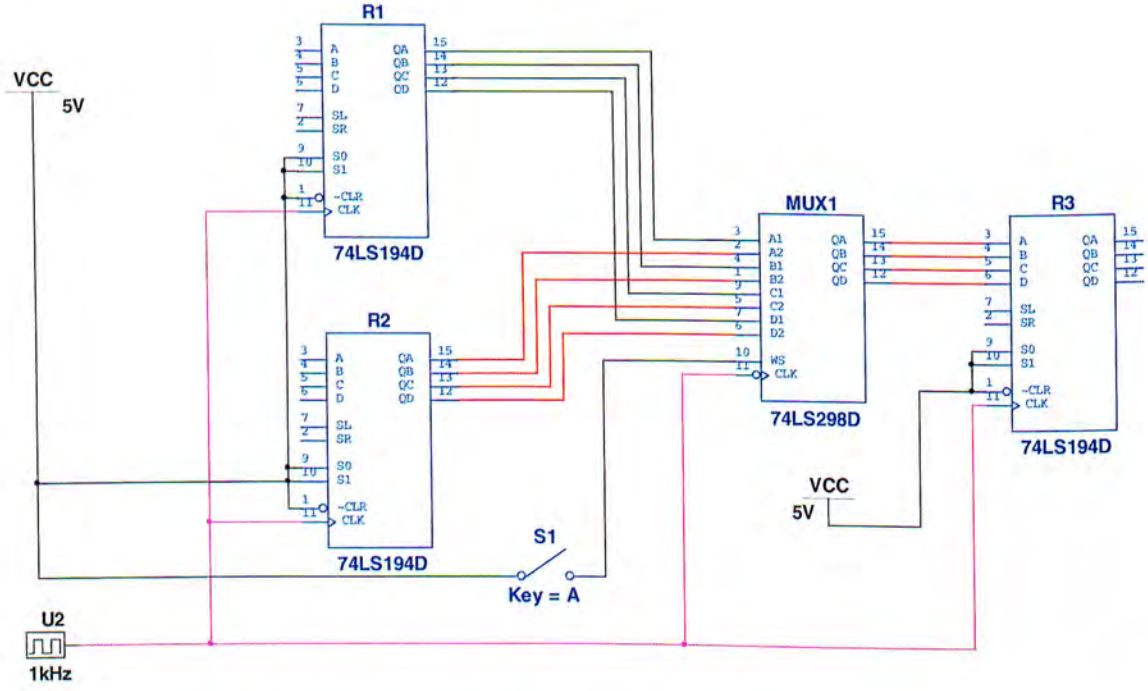
Ας επανέλθουμε στο παράδειγμα όπου επιθυμούσαμε τη μεταφορά ενός εκ των R1, R2 στον καταχωρητή R3. Πως θα το καταφέρναμε με τη χρήση πολυπλεκτών? Θα μπορούσαμε να οδηγήσουμε τα δύο λιγότερο σημαντικά bit των R1 και R2 σε έναν 2 σε 1 πολυπλέκτη (2

εισόδων και μίας εξόδου) και την έξοδο αυτού στο λιγότερο σημαντικό bit του R3. Εφόσον υπάρχουν 2 είσοδοι δεδομένων αρκεί μία είσοδος επιλογής. Όταν αυτή είναι 0 μεταφέρεται το λιγότερο σημαντικό ψηφίο του ενός (πχ του R1), ενώ όταν είναι 1 το αντίστοιχο ψηφίο του άλλου. Με την ίδια λογική μπορούμε να μεταφέρουμε το περιεχόμενο όλων των bit των R1, R2 στον R3 αρκεί να χρησιμοποιήσουμε από έναν πολυπλέκτη σε κάθε bit. Αν χρησιμοποιήσουμε για όλους τους πολυπλέκτες κοινή γραμμή για τις εισόδους επιλογής τους (άρα επιλογή των bit ίδιας θέσης κάθε φορά), τότε εύκολα διαπιστώνουμε ότι μπορούμε να μεταφέρουμε παράλληλα στον R3 όποιο περιεχόμενο των R1 ή R2 θέλουμε. Το κύκλωμα που μόλις περιγράψαμε υπάρχει έτοιμο σε ολοκληρωμένη μορφή και έχει την ονομασία Τετραπλός πολυπλέκτης 2-σε-1 και φαίνεται στο σχήμα 2.4 και είναι το 74LS298.



Σχήμα 2.4 Ολοκληρωμένο κύκλωμα Τετραπλού πολυπλέκτη 2-σε-1

Στις A1 και A2 συνδέουμε τα λιγότερο σημαντικά ψηφία των καταχωρητών-πηγής. Στις B1 και B2 τα επόμενα και ούτω καθεξής. Παρατηρούμε την είσοδο χρονισμού, που υποδηλώνει ότι πρόκειται για ένα σύγχρονο κύκλωμα, και την είσοδο επιλογής WS. Στο σχήμα 2.5 μπορείτε να δείτε πως θα υλοποιούσαμε το συγκεκριμένο παράδειγμα στο Multisim με κυκλώματα που ήδη έχουμε γνωρίσει.



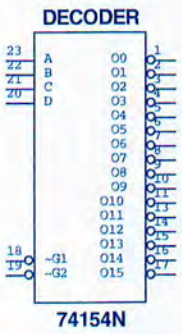
Σχήμα 2.5 Παράδειγμα χρήσης Τετραπλού πολυπλέκτη 2-σε-1

Βλέπουμε με τη ροζ γραμμή το καθολικό ρολόι του συστήματος, το οποίο εφαρμόζεται σύμφωνα με όσα έχουμε αναφέρει σε όλα τα κυκλώματα του συστήματος μας. Όταν ο διακόπτης S1 είναι ανοιχτός στις εξόδους του πολυπλέκτη θα εμφανιστούν τα δεδομένα του R1 (φυσικά με την έλευση ενός παλμού του ρολογιού). Όταν κλείσουμε τον S1 μεταφέρονται στις εξόδους του πολυπλέκτη τα δεδομένα του R2. Μπορείτε να ανατρέξετε στην αντίστοιχη παράγραφο για να δείτε τις συνδέσεις των καταχωρητών. Στις εισόδους δεδομένων των R1 και R2 μπορεί να φτάνουν δεδομένα από άλλους καταχωρητές, πολυπλέκτες, μνήμες ανάλογα με τις απαιτήσεις της εφαρμογής μας.

Σημείωση: Μέσω της εισόδου επιλογής ουσιαστικά ελέγχουμε το κύκλωμα. Η διαδικασία αυτή στα υπολογιστικά συστήματα πραγματοποιείται μέσω της μονάδας ελέγχου αλλά είναι μία καλή ευκαιρία να γνωρίσουμε την έννοια του ελέγχου που θα δούμε σε επόμενα κεφάλαια.

2.4 Αποκωδικοποιητές

Στα ψηφιακά κυκλώματα, ως γνωστών, όλα τα δεδομένα (λέξεις, αριθμοί κ.α) αναπαρίστανται στο δυαδικό σύστημα. Όταν εισάγουμε χαρακτήρες από το πληκτρολόγιο μας στον υπολογιστή, αυτοί κωδικοποιούνται σε δυαδική μορφή ώστε να μπορεί να πραγματοποιηθεί η αποθήκευση και η επεξεργασία τους. Στη συνέχεια ,πριν εμφανίσουν τα αποτελέσματα των πράξεων, αποκωδικοποιούνται καθώς στη δυαδική τους μορφή δεν έχουν κανένα απολύτως νόημα για εμάς. Στη συγκεκριμένη παράγραφο θα αναφερθούμε σε κυκλώματα που πραγματοποιούν διάφορα είδη κωδικοποιήσεων και αποκωδικοποιήσεων. Στη παράγραφο των πολυπλεκτών οδηγούσαμε στις εισόδους επιλογής έναν 3-bit δυαδικό αριθμό, το δεκαδικό ισοδύναμο του οποίου αντιστοιχούσε στον δείκτη της εισόδου της οποίας τα δεδομένα εμφανίζονταν στην μία και μοναδική έξοδο. Με τη χρήση ενός αποκωδικοποιητή 3-σε-8 (3 εισόδων και 8 εξόδων) θα μπορούμε να εισάγουμε έναν δυαδικό αριθμό στις εισόδους του και να ενεργοποιείται εκείνη η έξοδος της οποίας ο δείκτης αντιστοιχεί στο δεκαδικό ισοδύναμο του 3-bit δυαδικού αριθμού. Στο σχήμα 2.6 βλέπουμε το ολοκληρωμένο κύκλωμα ενός 4-σε-16 αποκωδικοποιητή.



Σχήμα 2.6 Ολοκληρωμένο κύκλωμα 4-σε-16 αποκωδικοποιητή

Οι τέσσερις είσοδοι δεδομένων A, B, C και D αποκωδικοποιούνται σε 16 εξόδους, με κάθε έξοδο να αντιπροσωπεύει έναν από τους 16 διαφορετικούς συνδυασμούς τιμών των εισόδων. Πολλά κυκλώματα της αγοράς όπως το 74154 που δείχνουμε παραπάνω αποκωδικοποιούν τον αριθμό δίνοντας στην αντίστοιχη έξοδο λογικό 0 ενώ κρατάνε τις άλλες εξόδους στο 1. Αυτό υποδηλώνεται από τους μικρούς κύκλους στους ακροδέκτες εξόδου του ολοκληρωμένου.

2.5 Κωδικοποιητές

Όπως είδατε περιγράψαμε τη λειτουργία της αποκωδικοποίησης. Σειρά τώρα έχει η κωδικοποίηση η οποία πραγματοποιείται με τη χρήση των κωδικοποιητών. Αυτοί έχουν 2^n εισόδους δεδομένων και n εξόδους. Με έναν 8-σε-3 κωδικοποιητή για παράδειγμα μπορούμε να δούμε ποιο είναι το δυαδικό ισοδύναμο κάθε εισόδου του κωδικοποιητή (άρα και το δυαδικό ισοδύναμο των αριθμών από το 0 έως το 7). Η σχεδίαση ενός κωδικοποιητή είναι αρκετά εύκολη και θα τη χρειαστούμε κατά την σχεδίαση του υπολογιστή. Στον πίνακα 2.1 βλέπουμε τον πίνακα αληθείας του 8-σε-3 κωδικοποιητή. Με βάση αυτόν θα κατασκευάσουμε το κύκλωμα.

Είσοδοι								Έξοδοι		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

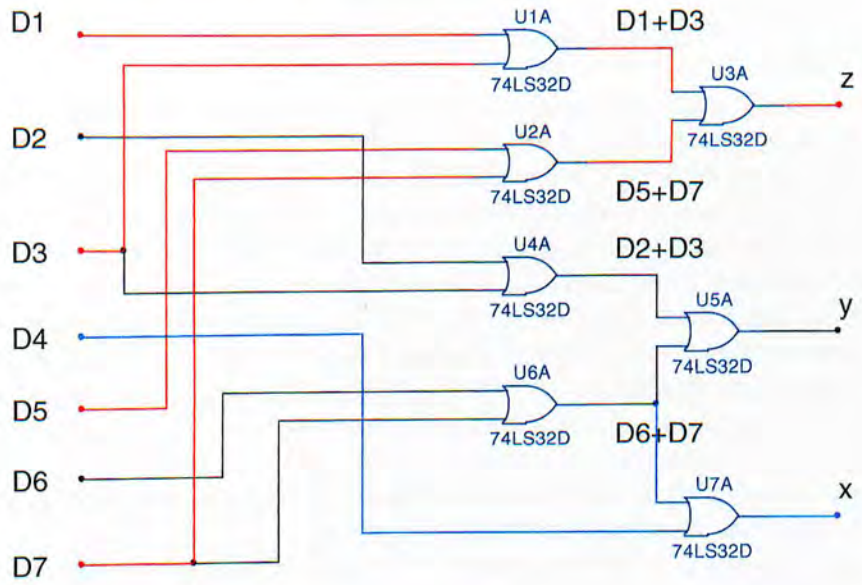
Πίνακας 2.1 Πίνακας αληθείας 8-σε-3 κωδικοποιητή

Οι έξοδοι του κωδικοποιητή μας δίνουν ένα διαφορετικό δυαδικό αριθμό για κάθε μεμονωμένη είσοδο του που είναι ενεργοποιημένη. Οι είσοδοι είναι 8, μία για κάθε αριθμό από το 0 έως το 7, και τρεις έξοδοι που παράγουν τον αντίστοιχο δυαδικό αριθμό. Φυσικά θεωρούμε ότι μόνο μία είσοδος μπορεί να είναι ενεργοποιημένη κάθε φορά, αλλιώς δεν έχει νόημα εφόσον θέλουμε να κωδικοποιούμε μόνον έναν αριθμό από το 0 ως το 7 κάθε φορά. Επομένως σε κάθε γραμμή των εισόδων μόνο ένα από τα D0-D7 είναι αληθές και τοποθετούμε τιμή 1. Στη συνέχεια στην πλευρά στήλη των εξόδων τοποθετούμε για κάθε είσοδο που έχει τιμή 1 το δυαδικό ισοδύναμο της. Έτσι αφού το δυαδικό ισοδύναμο της D0 είναι 000 στην γραμμή που αυτή είναι αληθής τοποθετούμε στις στήλες των εξόδων τρία μηδενικά. Για κάθε είσοδο ακολουθούμε ακριβώς την ίδια διαδικασία έως ότου γεμίσουμε τον πίνακα.

Σκανάρωντας τις τιμές της εξόδου z μετράμε 4 άσσους. Οι άσσοι αυτοί αντιστοιχούν στις εισόδους D1, D3, D5, D7. Επομένως η έξοδος z γίνεται 1 όταν μία εξ αυτών των εισόδων είναι 1. Αυτή η παρατήρηση μας μπορεί να γραφεί σε συνάρτηση της Άλγεβρας Boole ως εξής:

$$z = D1 + D3 + D5 + D7$$
$$y = D2 + D3 + D6 + D7$$
$$x = D4 + D5 + D6 + D7$$

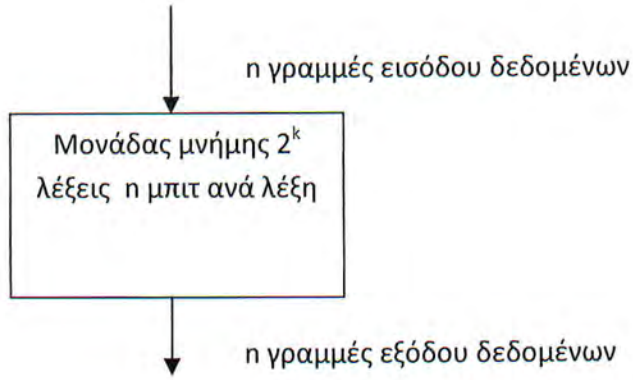
Ακολουθήσαμε την ίδια διαδικασία για όλες τις εξόδους και καταλήξαμε στις 3 συναρτήσεις Boole. Πλέον είναι πολύ εύκολο με τη χρήση πυλών OR να κατασκευάσουμε τον κωδικοποιητή. Μπορείτε να δείτε την υλοποίηση στο παρακάτω σχήμα 2.7. Η έξοδος z ενεργοποιείται όταν μία εκ των D1, D3, D5, D7. Ο συνδυασμός 3 πυλών OR μπορεί να μας δώσει την τελική συνάρτηση ελέγχου D1 + D3 + D5 + D7. Με τον ίδιο τρόπο σκέψης σχεδιάζουμε τον κωδικοποιητή που βλέπεται παρακάτω:



Σχήμα 2.7 Υλοποίηση κωδικοποιητή

2.6 Μνήμη RAM

Ως γνωστών οι μνήμες είναι το σημείο του ψηφιακού συστήματος όπου αποθηκεύουμε δεδομένα. Συνδυάζοντας flip flop καταφέραμε να κατασκευάσουμε έναν μεγαλύτερο αποθηκευτικό χώρο δεδομένων τον οποίο ονομάσαμε καταχωρητή. Θα μπορούσαμε με την ίδια λογική να χρησιμοποιήσουμε πολλούς καταχωρητές ώστε να δημιουργήσουμε έναν μεγάλο αποθηκευτικό χώρο. Η μονάδα μνήμης, λοιπόν, αποτελείται από ένα σύνολο κυττάρων αποθήκευσης και από κατάλληλα κυκλώματα, τα οποία είναι απαραίτητα για να εισάγουμε και να εξάγουμε πληροφορίες από αυτή. Μπορούμε να σκεφτούμε τη μνήμη σαν ένα σύνολο από καταχωρητές, ίδιου μεγέθους, τοποθετημένους τον έναν πάνω στον άλλο. Το περιεχόμενο του κάθε καταχωρητή αποτελεί μία ανεξάρτητη οντότητα από τα περιεχόμενα των υπολοίπων και ονομάζεται λέξη (word). Για να ξεχωρίζουμε την κάθε λέξη τις έχουμε αριθμήσει, έτσι κάθε λέξη έχει τη δική της διεύθυνση ανάλογα με τη σειρά στην οποία βρίσκεται (1η θέση, 2^n θέση...). Μέσω των διευθύνσεων επιτυγχάνουμε την αποθήκευση και την ανάγνωση δεδομένων από τη μνήμη. Στο σχήμα 2.8 φαίνεται το μπλοκ διάγραμμα μίας μνήμης RAM.



Σχήμα 2.8 Μπλοκ διάγραμμα μνήμης RAM

Όπως προείπαμε για να αποκτήσουμε το περιεχόμενο που βρίσκεται αποθηκευμένο σε κάποια θέση της μνήμης πρέπει να δώσουμε με κάποιο τρόπο στη μνήμη τη διεύθυνση στην οποία είναι αποθηκευμένο και στη συνέχεια αυτή να μας εμφανίσει στις εξόδους δεδομένων της το περιεχόμενο αυτής θέσης μνήμης . Αυτό επιτυγχάνεται μέσω γραμμών εισόδου και εξόδου δεδομένων, γραμμών επιλογής διευθύνσεων και γραμμών ελέγχου, οι οποίες προσδιορίζουν αν θα εισάγουμε ή θα εξάγουμε δεδομένα στη μνήμη. Τα παραπάνω φαίνονται στο μπλοκ διάγραμμα της μνήμης RAM στο σχήμα . Οι n γραμμές εισόδου δεδομένων (ο αριθμός n εξαρτάται από το μήκος λέξης που μπορεί να αποθηκευτεί στη μνήμη) παρέχουν τις πληροφορίες που θα αποθηκευτούν στη μνήμη και οι n γραμμές εξόδου δεδομένων παρέχουν τις πληροφορίες που διαβάζονται από τη μνήμη. Οι k γραμμές διεύθυνσης (ο αριθμός k εξαρτάται από το μέγεθος της μνήμης, δηλαδή τον αριθμό των θέσεων μνήμης που διαθέτει) προσδιορίζουν τη συγκεκριμένη λέξη, η οποία επιλέγεται ανάμεσα στις πολλές διαθέσιμες της μνήμης.

Οι δύο λειτουργίες τις οποίες μπορεί να εκτελέσει μία μνήμη RAM είναι η γραφή (εισαγωγή δεδομένων) και η ανάγνωση (εξαγωγή δεδομένων). Με την ενεργοποίηση του σήματος γραφής (ακροδέκτης Write) πραγματοποιείται η αποθήκευση του δεδομένου που βρίσκεται στις γραμμές εισόδου δεδομένων στο εσωτερικό της μνήμης. Σε ποια θέση μνήμης θα αποθηκευτεί το δεδομένο αυτό καθορίζεται από το δυαδικό αριθμό που έχουμε τοποθετήσει στις γραμμές διεύθυνσης. Με την ενεργοποίηση του σήματος ανάγνωσης επιτυγχάνεται η εμφάνιση (στις γραμμές εξόδου δεδομένων) της λέξης η οποία είναι αποθηκευμένη στη διεύθυνση της μνήμης που έχουμε θέσει στις γραμμές διευθύνσεων.

Τα βήματα που πρέπει να ακλουθούμε για την αποθήκευση ενός δεδομένου στη μνήμη είναι τα εξής:

1. Εφαρμόζουμε στις γραμμές διευθύνσεων τη διεύθυνση της μνήμης στην οποία θέλουμε να αποθηκευτούν τα δεδομένα.
2. Εφαρμόζουμε στις γραμμές εισόδου δεδομένων τα δεδομένα που θέλουμε να αποθηκευτούν στην παραπάνω διεύθυνση
3. Ενεργοποιούμε την είσοδο ελέγχου γραφής (write)

Η μονάδα μνήμης θα δεχτεί τα μπιτ από τις γραμμές εισόδου δεδομένων και θα τα αποθηκεύσει στη λέξη που ορίζεται από τις γραμμές διεύθυνσης.

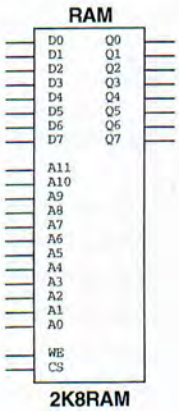
Αντίστοιχα τα βήματα για την ανάγνωση δεδομένων από τη μνήμη είναι τα εξής:

1. Εφαρμόζουμε στις γραμμές διευθύνσεων τη διεύθυνση της μνήμης τα δεδομένα της οποίας θέλουμε να διαβάσουμε.
2. Ενεργοποιούμε την είσοδο ελέγχου ανάγνωσης (read).

Η μονάδα μνήμης θα πάρει τα μπιτ της λέξης που είναι αποθηκευμένα στη συγκεκριμένη διεύθυνση (δηλαδή στη διεύθυνση που έχει εφαρμοστεί στους ακροδέκτες διευθύνσεων) και θα τα εμφανίσει στις γραμμές εξόδου δεδομένων. Τα περιεχόμενα της επιλεγμένης λέξης δεν αλλάζουν μετά την ανάγνωση.

Σημείωση: Στις γραμμές διεύθυνσης εφαρμόζουμε τη διεύθυνση της μνήμης στη δυαδική της μορφή. Στις μνήμες θα αποθηκεύουμε εκτός από αποτελέσματα πράξεων και τα προγράμματα που θα τρέχει ο υπολογιστής.

Το ολοκληρωμένο κύκλωμα που θα χρησιμοποιήσουμε για μνήμη RAM στο project μας είναι το παρακάτω:



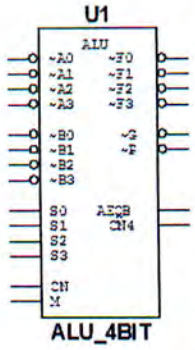
Σχήμα 2.9 Ολοκληρωμένο Κύκλωμα μνήμης RAM μεγέθους 2kByte

Οι είσοδοι D0 έως D7 αποτελούν τις εισόδους δεδομένων. Οι έξοδοι Q0 έως Q7 αποτελούν τις εξόδους δεδομένων. Οι είσοδοι A0 έως A11 είναι οι γραμμές διευθύνσεων. Τέλος υπάρχουν οι είσοδοι ελέγχου CS (chip select) και WE (write). Κάθε μνήμη διαθέτει τουλάχιστον έναν ακροδέκτη ενεργοποίησης CS που χρησιμεύει για την ενεργοποίηση του. Όταν CS=1 η μνήμη λειτουργεί κανονικά και ανάλογα με την τιμή του ακροδέκτη ελέγχου WS πραγματοποιείται γραφή ή ανάγνωση. Πιο συγκεκριμένα όταν WS=0 πραγματοποιείται ανάγνωση των δεδομένων της θέσης μνήμης που υποδεικνύεται από στις A0-A11. Όταν WS=1 πραγματοποιείται γραφή των δεδομένων που είναι τοποθετημένα στις D0-D7 στη θέση μνήμης που υποδεικνύεται στις A0-A11

Σημείωση: Παρατηρούμε ότι δεν υπάρχει είσοδος χρονισμού, επομένως η λειτουργία της μνήμης είναι ανεξάρτητη από τους παλμούς του ρολογιού.

2.7 Arithmetic Logic Unit

Οι ψηφιακοί υπολογιστές εκτελούν ποικίλες εργασίες σχετικά με την επεξεργασία πληροφοριών. Στις εργασίες αυτές ανήκει και η εκτέλεση αριθμητικών και λογικών πράξεων, οι οποίες θα είναι και η κύρια λειτουργία της δικής μας κατασκευής. Ανάλογα με την πράξη που θέλουμε να εκτελέσουμε υπάρχει και το αντίστοιχο κύκλωμα το οποίο μπορεί να την πραγματοποιήσει. Ωστόσο έχουν αναπτυχθεί κυκλώματα τα οποία μπορούν να πραγματοποιήσουν πολλές διαφορετικές πράξεις μεταξύ δύο δεδομένων. Ένα τέτοιο κύκλωμα ονομάζεται ALU (Arithmetic Logic Unit) και φαίνεται στο σχήμα 2.10.



Σχήμα 2.10 Ολοκληρωμένο κύκλωμα ALU 4-BIT

Πρόκειται για μία ALU 4-bit, πραγματοποιεί δηλαδή πράξεις μεταξύ λέξεων μήκους 4-bit. Στους ακροδέκτες A0-A3 τοποθετούνται τα bit του 1^{ου} αριθμού ενώ στους ακροδέκτες B0-B3 του δεύτερου. Η επιλογή της πράξης που θα πραγματοποιηθεί μεταξύ των δύο αριθμών γίνεται μέσω των ακροδεκτών επιλογής S0-S3. Μέσω του πίνακα λειτουργίας της ALU βρίσκουμε τις τιμές που πρέπει να τεθούν στις εισόδους επιλογής της προκειμένου να επιτευχθεί η επιθυμητή λογική ή αριθμητική πράξη.

Function Table

Mode Select Inputs				Active LOW Operands & F_n Outputs		Active HIGH Operands & F_n Outputs	
				Logic	Arithmetic (Note 2)	Logic	Arithmetic (Note 2)
S3	S2	S1	S0	(M = H)	(M = L) (C_n = L)	(M = H)	(M = L) (C_n = H)
L	L	L	L	$\overline{A}$	A minus 1	$\overline{A}$	A
L	L	L	H	$\overline{AB}$	AB minus 1	$\overline{A} + \overline{B}$	A + B
L	L	H	L	$\overline{A} + \overline{B}$	$A\overline{B}$ minus 1	$\overline{A}$ B	A + $\overline{B}$
L	L	H	H	Logic 1	minus 1	Logic 0	minus 1
L	H	L	L	$\overline{A} + \overline{B}$	A plus (A + $\overline{B}$)	$\overline{AB}$	A plus $A\overline{B}$
L	H	L	H	$\overline{B}$	AB plus (A + $\overline{B}$)	$\overline{B}$	(A + B) plus $A\overline{B}$
L	H	H	L	$\overline{A} \oplus \overline{B}$	A minus B minus 1	A ⊕ B	A minus B minus 1
L	H	H	H	A + $\overline{B}$	A + $\overline{B}$	$A\overline{B}$	AB minus 1
H	L	L	L	$\overline{A}$ B	A plus (A + B)	$\overline{A}$ + B	A plus AB
H	L	L	H	A ⊕ B	A plus B	$\overline{A} \oplus \overline{B}$	A plus B
H	L	H	L	B	$A\overline{B}$ plus (A + B)	B	(A + $\overline{B}$) plus AB
H	L	H	H	A + B	A + B	AB	AB minus 1
H	H	L	L	Logic 0	A plus A (Note 1)	Logic 1	A plus A (Note 1)
H	H	L	H	$A\overline{B}$	AB plus A	A + $\overline{B}$	(A + B) plus A
H	H	H	L	AB	$A\overline{B}$ minus A	A + B	(A + $\overline{B}$) plus A
H	H	H	H	A	A	A	A minus 1

Note 1: Each bit is shifted to the next most significant position.

Note 2: Arithmetic operations expressed in 2s complement notation.

Πίνακας 2.2 Πίνακας Λειτουργίας ALU

Για παράδειγμα αν επιθυμούμε να πραγματοποιηθεί η λογική πράξη OR μεταξύ δύο αριθμών A και B εργαζόμαστε ως εξής:

1. Επιθυμούμε οι είσοδοι και οι έξοδοι να ενεργοποιούνται με λογικό 1, δηλαδή είναι ενεργά υψηλές. Επομένως εξετάζουμε την τρίτη στήλη του πίνακα με τίτλο Active High Inputs & Outputs.
2. Επιθυμούμε να πραγματοποιηθεί λογική πράξη επομένως οδηγούμε στον ακροδέκτη M λογικό 1 (Για αριθμητικές πράξεις θέτουμε M=0).
3. Βρίσκουμε το συμβολισμό της πράξης που μας ενδιαφέρει και οδηγούμε στους ακροδέκτες S0-S3 το δυαδικό αριθμό που αντιστοιχεί σε αυτή την πράξη στη στήλη MODE SELECT INPUTS.

Ο ακροδέκτης Cn4 αποτελεί το κρατούμενο της πρόσθεσης. Σε περίπτωση που θέλουμε να πραγματοποιήσουμε πράξεις μεταξύ 8-bit αριθμών, αρκεί να συνδέσουμε το κρατούμενο που παράγεται από την πρώτη ALU (των 4-bit χαμηλής τάξης) στην είσοδο Cn που αποτελεί το κρατούμενο εισόδου.

3ο κεφάλαιο
Μεταφορά στο Επίπεδο Καταχωρητών

Το κάθε σύστημα χωρίζεται σε επιμέρους, λειτουργικά αυτόνομα υποσυστήματα, τα οποία κατασκευάζονται με ψηφιακές διατάξεις, ανάμεσα στις οποίες είναι οι καταχωρητές, οι αποκωδικοποιητές, οι πολυπλέκτες, τα υποκυκλώματα εκτέλεσης αριθμητικών πράξεων και τα λογικά κυκλώματα ελέγχου. Η διασύνδεση αυτών των υποσυστημάτων, ώστε να σχηματιστεί ένα ενιαίο σύστημα απαιτεί τη γνώση μίας σχετικής μεθοδολογίας για την περιγραφή και τη σχεδίαση μεγάλων και σύνθετων ψηφιακών συστημάτων. Στο παρόν κεφάλαιο θα μιλήσουμε για αυτή τη μεθοδολογία που ονομάζεται μεταφορά στο επίπεδο καταχωρητών.

3.1 Register Transfer Language

Ένα ψηφιακό σύστημα είναι ένα κύκλωμα διαφορετικών ψηφιακών μονάδων-υποσυστημάτων διασυνδεδεμένων μεταξύ τους, όπου η κάθε μία πραγματοποιεί μία συγκεκριμένη λειτουργία. Οι μονάδες αυτές κατασκευάζονται από κυκλώματα που γνωρίσαμε όπως οι καταχωρητές, οι μετρητές, οι αποκωδικοποιητές κ.α.

Τα υποσυστήματα αυτά προσδιορίζονται από τους καταχωρητές που διαθέτουν και τις λειτουργίες που πραγματοποιούνται στα δεδομένα αυτών των καταχωρητών. Οι λειτουργίες που εκτελούνται σε δεδομένα αποθηκευμένα σε καταχωρητές ονομάζονται μικρολειτουργίες. Παραδείγματα μικρολειτουργιών είναι η μεταφορά δεδομένων μεταξύ καταχωρητών, ο καθαρισμός, η αύξηση της τιμής των περιεχομένων κ.α.

Ο τρόπος με τον οποίο συνθέτουμε το υπολογιστικό σύστημα που επιθυμούμε είναι αρχικά να καθορίσουμε τις λειτουργίες τις οποίες θα πραγματοποιεί. Στη συνέχεια για κάθε υποσύστημα επιλέγουμε τους καταχωρητές που χρειαζόμαστε και τις μικρολειτουργίες που θα πρέπει να πραγματοποιούνται μεταξύ αυτών, προκειμένου να λειτουργεί το υποσύστημα σύμφωνα με τις απαιτήσεις μας. Τέλος συνδέουμε τα επιμέρους υποσυστήματα δίνοντας υπόσταση στο τελικό σύστημα.

Όπως ίσως θα έχετε παρατηρήσει η λειτουργία των ψηφιακών συστημάτων είναι μία πολύ λεπτομερής και δομημένη διαδικασία η οποία βασίζεται σε μία βήμα προς βήμα εκτέλεση απλών λειτουργιών (μικρολειτουργιών). Επομένως ο τρόπος με τον οποίο θα πρέπει να εργαστούμε θα πρέπει να είναι λεπτομερής, ακριβής και σαφής. Σε αυτό θα μας βοηθήσει μία τεχνική η οποία ονομάζεται γλώσσα στο επίπεδο μεταφοράς καταχωρητών. Η γλώσσα αυτή αποτελείται από απλά σύμβολα με τα οποία γράφουμε την ακολουθία των μικρολειτουργιών που πρέπει να πραγματοποιηθούν σε ένα υποσύστημα και τις συνθήκες υπό τις οποίες θα πραγματοποιηθούν. Είναι θα λέγαμε ένα συμβολικό πρόγραμμα που θα αποτελεί το σχεδιάγραμμα με βάση το οποίο θα πραγματοποιούμε τις κατάλληλες συνδέσεις μεταξύ των εξαρτημάτων του υποσυστήματος. Η μεταφορά του περιεχομένου ενός καταχωρητή σε έναν άλλο μπορεί να παρασταθεί στη γλώσσα μεταφοράς καταχωρητών με τον συμβολισμό:

$$R2 \leftarrow R1$$

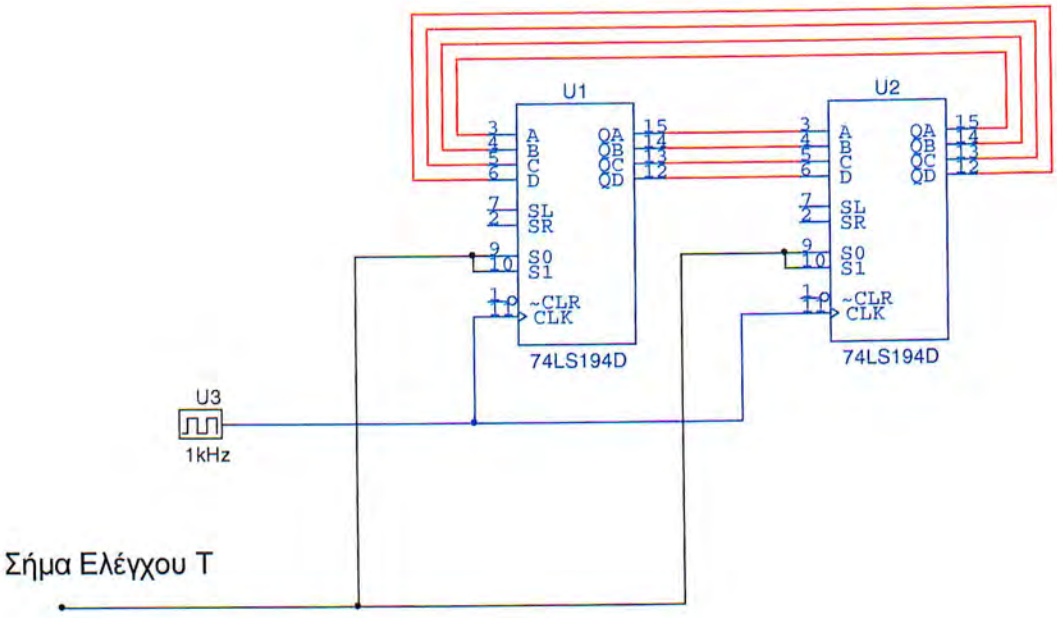
Υπενθυμίζουμε ότι οι καταχωρητές είναι σύγχρονα ακολουθιακά κυκλώματα, επομένως η λειτουργία τους εξαρτάται από την ενεργοποίηση της εισόδου χρονισμού. Σύμφωνα με την παραπάνω μικρολειτουργία το περιεχόμενο του R1 μεταφέρεται στον R2 με την έλευση του επόμενου παλμού του ρολογιού. Ωστόσο θυμηθείτε ότι στην παράγραφο των καταχωρητών είχαμε αναφερθεί στην είσοδο φόρτωσης (LOAD) του καταχωρητή. Για να πραγματοποιηθεί η μεταφορά πρέπει η είσοδος φόρτωσης του R2 να είναι ενεργοποιημένη, ώστε ο καταχωρητής να είναι σε θέση να δεχτεί τα δεδομένα που εφαρμόζονται στις εισόδους δεδομένων του. Με τον τρόπο που έχουμε γράψει την μικρολειτουργία (δεν έχουμε θέσει κάποια συνθήκη) η μεταφορά θα πραγματοποιείται σε κάθε παλμό του ρολογιού που εφαρμόζεται στον R2.
Η ιδιότητα αυτή των καταχωρητών, να φορτώνουν δεδομένα μόνο εάν είναι ενεργοποιημένη η είσοδος LOAD, μας βοηθά στο να ελέγχουμε τον καταχωρητή υπό ορισμένες συνθήκες. Συνήθως, θέλουμε η μεταφορά να πραγματοποιείται μόνο αν αληθεύει μία συνθήκη. Η συνθήκη που καθορίζει πότε θα εκτελεστεί μία μικρολειτουργία λέγεται συνάρτηση ελέγχου. Η συνάρτηση ελέγχου είναι μία λογική σχέση (μία συνάρτηση Boole) η οποία μπορεί να είναι αληθής ή ψευδείς. Ας πάρουμε για παράδειγμα την παρακάτω μικρολειτουργία:

$$P: R2 \leftarrow R1$$

Είναι η μικρολειτουργία που είδαμε πριν, με τη διαφορά ότι η μεταφορά θα συμβεί μόνο όταν η συνθήκη ελέγχου P είναι αληθής (Για να είμαστε ακριβής πρέπει να είναι αληθής η συνθήκη και στον επόμενο παλμό του ρολογιού θα γίνει η μεταφορά.)
Η συνθήκη ελέγχου P είναι μία συνάρτηση Boole η οποία "γεννιέται" σε ένα υποσύστημα του υπολογιστή που ονομάζεται κεντρική μονάδα επεξεργασίας (CPU). Σκοπός της μονάδας επεξεργασίας είναι η παραγωγή των σημάτων τα οποία θα εφαρμόζονται στα εξαρτήματα του υπολογιστή (καταχωρητές, πολυπλέκτες κλπ) και θα ελέγχουν τη λειτουργία τους. Είναι θα λέγαμε ο διευθυντής που δίνει τις οδηγίες για το **πότε** και **ποια** θα είναι η λειτουργία του κάθε εξαρτήματος.
Πολλές φορές θα χρειαστεί την ίδια χρονική στιγμή να εκτελεστούν περισσότερες από μία μικρολειτουργίες. Μπορούμε να τις γράψουμε τη μία δίπλα στην άλλη χωρίζοντας τες με κόμμα όπως φαίνεται παρακάτω:

$$T: R2 \leftarrow R1, R1 \leftarrow R2$$

Ας "διαβάσουμε" αυτή τη μικρολειτουργία για να δούμε τι πληροφορίες θα πάρουμε για να σχεδιάσουμε το κύκλωμα που την υλοποιεί. Αρχικά θα έχουμε μεταφορά δεδομένων από τον R1 στον R2. Επομένως θα πρέπει να συνδέσουμε τις εξόδους του R1 στις εισόδους δεδομένων του R2 (το λιγότερο σημαντικό bit του ενός στο αντίστοιχο bit του άλλου και ούτω καθεξής). Επίσης το περιεχόμενο του R2 μεταφέρεται στον R1, άρα συνδέουμε και τις εξόδους του R2 στον R1. Η μικρολειτουργία θα εκτελεστεί αν αληθεύει μία συνθήκη, που όπως είπαμε παράγεται στη Μονάδα Ελέγχου. Επομένως όταν T=1 θα γίνει παράλληλη φόρτωση των δεδομένων στους καταχωρητές, που σημαίνει ότι πρέπει συνδέσουμε τη συνθήκη T στους ακροδέκτες LOAD και των δύο καταχωρητών. Στο σχήμα 3.1 φαίνεται το κύκλωμα που εκτελεί τις παραπάνω μικρολειτουργίες. Οι είσοδοι CLR είναι ενεργά χαμηλές επομένως θα πρέπει να συνδεθούν με τα 5V, ενώ οι είσοδοι SL και SR είναι αδιάφορες.



Σχήμα 3.1 Ανταλλαγή δεδομένων μεταξύ δύο καταχωρητών

Σημείωση: Ο παλμός του ρολογιού είναι μία συνθήκη την οποία δεν γράφουμε στο συμβολικό πρόγραμμα καθώς οποιαδήποτε πράξη στο επίπεδο καταχωρητών εννοείται πως πραγματοποιείται μόνο με την έλευση ενός παλμού του ρολογιού.

Ένα σύγχρονο υπολογιστικό σύστημα όπως καταλαβαίνεται δε μπορεί να πραγματοποιεί μόνο απλή μεταφορά δεδομένων. Είναι απαραίτητο να μπορεί να πραγματοποιεί ορισμένες βασικές αριθμητικές και λογικές μικρολειτουργίες όπως η πρόσθεση, η αφαίρεση, το λογικό AND κλπ. Παραδείγματα μικρολειτουργιών είναι τα παρακάτω:

C ← A + B	Το άθροισμα των A και B μεταφέρεται στον C
C ← A + 1	Το περιεχόμενο το A αυξάνεται κατά 1 και μεταφέρεται στο C
M[AR] ← A	Το περιεχόμενο του A μεταφέρεται στη θέση μνήμης που υποδεικνύει ο AR
C ← M[AR]	Το περιεχόμενο της μνήμης που υποδεικνύει ο AR μεταφέρεται στον C
C ← AvB	Πραγματοποιείται η πράξη OR μεταξύ των περιεχομένων των A και B και το αποτέλεσμα τους μεταφέρεται στον C

3.2 Μεταφορά δεδομένων από και προς μνήμη

Στην παράγραφο 2.6 περιγράψαμε τον τρόπο λειτουργίας των μνημών. Η μεταφορά πληροφοριών από τη μνήμη στον "έξω κόσμο" αποτελεί τη λειτουργία της ανάγνωσης, ενώ η αντίστροφη λειτουργία ονομάζεται γραφή. Όπως αναφέραμε και στο κεφάλαιο των μνημών σε ποια θέση μνήμης θα αποθηκευτεί το προς αποθήκευση δεδομένο (λειτουργία ανάγνωσης) ή ποιας θέσης μνήμης το περιεχόμενο θα τοποθετηθεί στις εξόδους δεδομένων της, καθορίζεται από τη διεύθυνση της μνήμης που έχουμε τοποθετήσει στις

γραμμές διευθύνσεων. Τις λέξεις μνήμης θα τις συμβολίζουμε με το γράμμα M. Η ακριβής διεύθυνση μνήμης της οποίας το περιεχόμενο θα διαβαστεί/γραφεί θα τοποθετείται μέσα σε αγκύλες, δίπλα από το γράμμα M.
Ας θεωρήσουμε ότι έχουμε μία μνήμη της οποίας οι γραμμές διευθύνσεων είναι συνδεδεμένες με έναν καταχωρητή, τον οποίο ονομάζουμε AR (Address Register). Επίσης οι γραμμές δεδομένων της είναι συνδεδεμένες με έναν δεύτερο καταχωρητή DR (Data Register). Σύμφωνα με την παρακάτω μικρολειτουργία:

Read: DR← M[AR]

Εφόσον η συνθήκη Read είναι αληθής (εφόσον δηλαδή εφαρμοσθούν στη μνήμη τα κατάλληλα σήματα ώστε να πραγματοποιήσει λειτουργία ανάγνωσης), το περιεχόμενο της θέσης μνήμης που ορίζει ο AR θα εμφανισθεί στις εξόδους της μνήμης και επομένως θα αποθηκευτεί στον DR . Έτσι επιτυγχάνεται η λειτουργία της ανάγνωσης. Τα αντίστοιχα ισχύουν και για τη λειτουργία της γραφής.

Write: M[AR]←DR

Το περιεχόμενο του DR θα τοποθετηθεί στη θέση μνήμης που "δείχνει" ο AR.

4ο κεφάλαιο
Κώδικες Εντολών

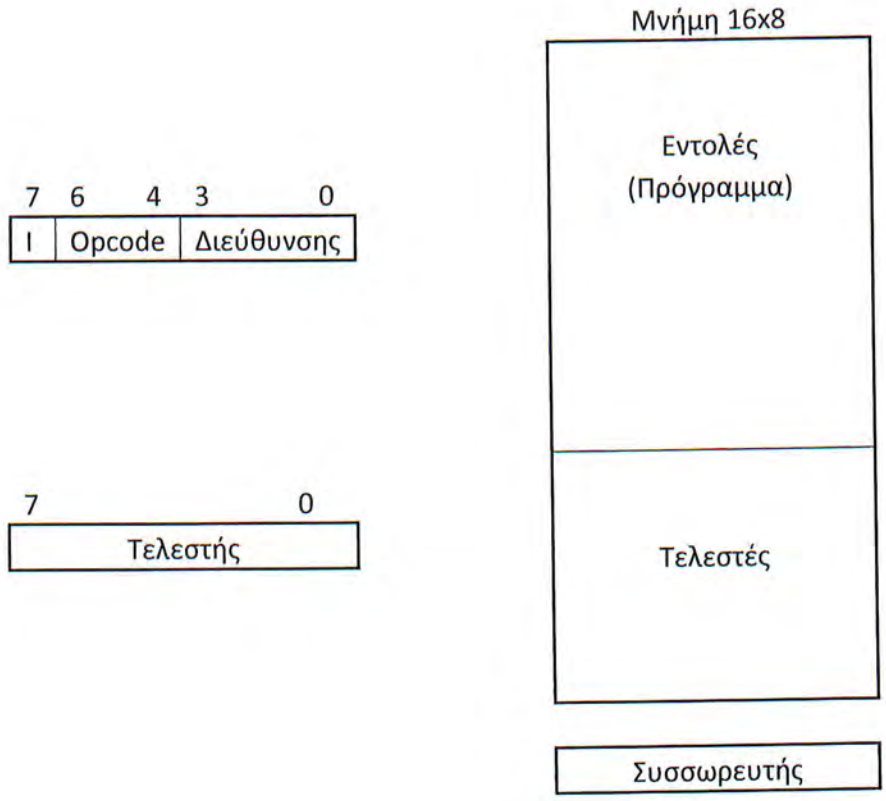
Ο τρόπος με τον οποίο οργανώνουμε και κατασκευάζουμε τα ψηφιακά συστήματα εξαρτάται από ορισμένους παράγοντες. Πρώτον τους καταχωρητές από τους οποίους αποτελούνται, δεύτερον τις εντολές τις οποίες θα εκτελούν και φυσικά τις συνθήκες ελέγχου απ τις οποίες εξαρτάται η εκτέλεση των εντολών. Έχοντας προσδιορίσει αυτά τα στοιχεία έχουμε κάνει ένα πολύ σημαντικό βήμα στη σχεδίαση κάθε συστήματος.
Ο τρόπος με τον οποίο θα είναι συνδεδεμένα τα δομικά στοιχεία κάθε ψηφιακού συστήματος (καταχωρητές, μετρητές, πολυπλέκτες κλπ.), η εσωτερική του οργάνωση με άλλα λόγια, καθορίζεται όπως αναφέραμε στο προηγούμενο κεφάλαιο από τις μικρολειτουργίες που θέλουμε να πραγματοποιούνται στα δεδομένα που είναι αποθηκευμένα στους καταχωρητές του. Ο χρήστης καθορίζει την ακολουθία των μικρολειτουγιών μέσω του προγράμματος. Πρόγραμμα είναι ένα σύνολο εντολών που καθορίζουν τη λειτουργία που θα εκτελεστεί (πρόσθεση, αφαίρεση κλπ), τους τελεστές και την ακολουθία με την οποία θα εκτελεσθεί η όλη διαδικασία. Είναι δηλαδή ένα σύνολο οδηγιών που αποθηκεύουμε στη μνήμη του συστήματος με το οποίο υπαγορεύουμε στο σύστημα μας ποια θα είναι ακριβώς η λειτουργία του.
Εντολή είναι μία ακολουθία δυαδικών ψηφίων (ένα δυαδικός αριθμός) η οποία καθορίζει μία σειρά μικρολειτουργιών. Δηλαδή, κάθε εντολή που εισάγουμε αντιστοιχεί σε ένα σύνολο μικρολειτουργιών. Κάθε εντολή του προγράμματος οδηγείται στη μονάδα ελέγχου, όπου γίνεται η "μετάφραση" ή ερμηνεία ή αποκωδικοποίηση της και με βάση αυτή την αποκωδικοποίηση γεννιούνται τα κατάλληλα σήματα ελέγχου (θυμηθείται τα σήματα ελέγχου που αναφέραμε στο υποκεφάλαιο 2.5) με τα οποία ελέγχονται οι μονάδες του συστήματος.
Πως όμως φτάνουμε από την εντολή που εισάγει ο χρήστης στις μικρολειτουργίες που τελικά θα πραγματοποιήσει ο υπολογιστής? Για να κατανοήσουμε αυτή τη διαδικασία θα πρέπει να αναλύσουμε τη δομή των εντολών αυτών ή του κώδικα εντολών (instruction code) όπως συχνά αναφέρεται. Ο κώδικας εντολής (που όπως είπαμε αποτελείται από bits) υποδιαιρείται σε τμήματα, που το καθένα έχει ξεχωριστό σκοπό. Το σημαντικότερο μέρος του κώδικα εντολής είναι ο κώδικας λειτουργίας του (συχνά θα αναφερόμαστε σε αυτόν με τον όρο opcode). Μέσω αυτού υποδυκνείουμε στο σύστημα ποια θα είναι η λειτουργία που θα πραγματοποιήσει π.χ πρόσθεση, αφαίρεση κλπ. Το μέγεθος του κώδικα λειτουργίας, δηλαδή ο αριθμός των bit που θα καταλαμβάνει μέσα στην εντολή εξαρτάται από τον συνολικό αριθμό των λειτουργιών που θα πραγματοποιεί το σύστημα, καθώς κάθε λειτουργία θα πρέπει να έχει τη δική της ξεχωριστή κωδικοποίηση (ένας συνδιασμός bit για κάθε opcode). Εάν επομένως ένας υπολογιστής μπορεί να πραγματοποιήσει 2^n διαφορετικές λειτουργίες τότε n bit απαιτούνται για την αναπαράσταση όλων των κωδικών λειτουργίας. Αντιστοιχοίζουμε, επομένως, κάθε λειτουργία σε έναν δυαδικό αριθμό. Η λειτουργία της πρόσθεσης θα μπορούσε για παράδειγμα να κωδικοποιηθεί ως 001, η αφαίρεση ως 010 κλπ. Η αποκωδικοπίηση αυτών των δυαδικών αριθμών στη μονάδα ελέγχου θα δημιουργήσει τα σήματα ελέγχου τα οποία με τη σειρά τους θα "οδηγήσουν" τα κατάλληλα κυκλώματα του συστήματος για την πραγματοποίηση της επιθυμητής λειτουργίας.
Όπως προαναφέραμε ο κώδικας εντολής προσδιορίζει επακριβώς τη λειτουργία που θέλουμε να πραγματοποιηθεί. Καταλαβαίνουμε, λοιπόν, ότι μόνο ο κώδικας λειτουργίας δεν αρκεί. Θα πρέπει να προσδιοριστούν τα δεδομένα τα οποία θα επεξεργαστεί το

σύστημα. Μέσω του κώδικα εντολής, επομένως, θα πρέπει να προσδιορίζονται τόσο η λειτουργία που θα εκτελεσθεί, όσο και οι καταχωρητές ή οι θέσεις μνήμης όπου βρίσκονται τα προς επεξεργασία δεδομένα.

Υπάρχουν πολλοί διαφορετικοί τρόποι για την οργάνωση των εντολών με βάση τις οποίες θα προγραμματίζουμε τον υπολογιστή. Ένας απλός τρόπος, τον οποίο και υιοθετήσαμε στη σχεδίαση του, είναι αρχικά ο διαχωρισμός της εντολής σε δύο τμήματα. Το πρώτο τμήμα θα αποτελεί το τμήμα του τελεστή ή operand και θα αντιστοιχεί στη διεύθυνση της μνήμης όπου είναι αποθηκευμένο το δεδομένο το οποίο θα επεξεργαστούμε. Μέσω του operand, επομένως, πληροφορούμε τη μονάδα ελέγχου για το που βρίσκεται το δεδομένο το οποίο θέλουμε να υποστεί την επεξεργασία. Τα δυαδικά ψηφία του operand καταλαμβάνουν τις θέσεις από το λιγότερο σημαντικό bit μέχρι το bit εκείνο που απαιτείται για να παρασταθούν όλες οι διευθύνσεις της μνήμης (εξαρτάται δηλαδή από το μέγεθος της χρησιμοποιούμενης μνήμης).

Ας δούμε ένα παράδειγμα για να κατανοήσουμε καλύτερα την όλη διαδικασία. Στο παρακάτω σχήμα φαίνονται στο δεξί μέρος της εικόνας η μονάδα μνήμης και ο συσσωρευτής ή processor register ενώ στο αριστερό ένας κώδικας εντολής και ένας τελεστής. Παρατηρούμε ότι πρόκειται για μία μνήμη 16 θέσεων μνήμης (ή αν προτιμάται 16Byte) στην οποία αποθηκεύονται λέξεις των 8 bit. Στο επάνω μέρος της μνήμης είναι αποθηκευμένοι οι κώδικες εντολών, ενώ στο κάτω μέρος βρίσκονται οι τελεστές (η διάταξη αυτή δεν είναι υποχρεωτική, εμείς επιλέγουμε πως θα οργανώσουμε τη μνήμη).

Ας επικεντρωθούμε στην οργάνωση των εντολών. Όπως προείπαμε η μνήμη διαθέτει 16 θέσεις μνήμης. Αυτό σημαίνει ότι απαιτούνται 4 bits για τον προσδιορισμό των διευθύνσεων της μνήμης ($16=2^4$). Επομένως προκειμένου να προσδιορίσουμε, μέσω του κώδικα εντολής, τη διεύθυνση της μνήμης όπου είναι αποθηκευμένο το προς επεξεργασία δεδομένο χρειαζόμαστε 4 bit από τα συνολικά 8 που διαθέτει ο κώδικας εντολής. Αυτά τα 4 bit αποφασίσαμε να είναι τα bit 0 έως 3. Εν συνεχεία τα υπόλοιπα 4 bit, 4 έως 7, μπορούν να χρησιμοποιηθούν για την αποθήκευση του opcode. Η χρήση 4 bit επιτρέπει στο σύστημα μας την εκτέλεση 16 διαφορετικών λειτουργιών. Ο υπολογιστής που θα κατασκευάσουμε θα πραγματοποιεί 7 διαφορετικές εντολές, άρα αρκούν 3 bit για την κωδικοποίηση του opcode. Το περισσότερο σημαντικό ψηφίο (για συντομία θα αναφέρεται ως ΠΣΨ) μπορεί να μείνει ανενεργό ή να χρησιμοποιηθεί για κάποιο άλλο σκοπό.



Σχήμα 4.1 Οργάνωση μνήμης και εντολών

Κεφάλαιο 5ο
Οι Επιμέρους Μονάδες του Υπολογιστή

Έχοντας πλέον αναφερθεί στα δομικά στοιχεία των ψηφιακών συστημάτων και έχοντας αναλύσει τη διαδικασία που θα ακολουθήσουμε για να χτίσουμε ένα υπολογιστικό κύκλωμα, δε μένει παρά να αναφερθούμε στις επιμέρους μονάδες από τις οποίες θα αποτελείται ο υπολογιστής. Κάθε μία από αυτές τις μονάδες θα επιτελεί και ένα διαφορετικό έργο και η σύνδεση και επικοινωνία μεταξύ τους συνθέτει το τελικό κύκλωμα του υπολογιστή.

5.1 Σύνολο Εντολών

Το πρώτο βήμα για την επίτευξη του σκοπού μας είναι η επιλογή των εντολών – λειτουργιών που επιθυμούμε να εκτελεί ο υπολογιστής και φυσικά η αντιστοίχιση κάθε εντολής σε έναν κώδικα λειτουργίας. Πολλές από τις εντολές αυτές τις έχουμε ήδη δει. Στον πίνακα 5.1 συνοψίζονται οι λειτουργίες και γίνεται η αντιστοίχιση των opcode.

Λειτουργία	Περιγραφή	Κώδικας Λειτουργίας
AC ← DR	Μεταφορά του DR στον AC	000
AC ← AC+DR	Μεταφορά στο AC του αθροίσματος του DR και του AC	001
AC ← AC v DR	Μεταφορά στον AC του λογικού αθροίσματος AC+DR	010
AC ← AC ^ DR	Μεταφορά στον AC του λογικού πολλαπλασιασμού AC.DR	011
AC ← AC - DR	Μεταφορά στο AC της διαφοράς του AC και του DR	100
M[AR] ← AC	Αποθήκευση στη μνήμη του περιεχομένου του AC	101
START ← CLR	Καθαρισμός του μανδαλωτή που σηματοδοτεί την έναρξη της λειτουργίας	110

Πίνακας 5.1 Λειτουργίες και κώδικες λειτουργιών

Στον πίνακα παρατηρούμε μία νέα λειτουργία την οποία δεν έχουμε αναφέρει. Πρόκειται για τον "καθαρισμό" του μανδαλωτή D, ο οποίος σηματοδοτεί την έναρξη της λειτουργίας του υπολογιστή. Με λίγα λόγια θα χρησιμοποιήσουμε ένα μανδαλωτή τον οποίο όταν θέτουμε σε 1 θα ξεκινά η μέτρηση των παλμών του ρολογιούωκαι επομένως θα ενεργοποιείται ο υπολογιστής. Είναι δηλαδή το ON της συσκευής μας. Ο καθαρισμός του, όπως καταλαβαίνεται, σταματά τη λειτουργία του συστήματος μέχρις ότου τον ξαναθέσουμε σε λογικό 1. Είναι μία εντολή σταματήματος (Halt) και θα τοποθετείται στο τέλος του προγράμματος.

5.2 Επιλογή Καταχωρητών

Το επόμενο βήμα είναι η επιλογή των καταχωρητών του υπολογιστή. Αρχικά, όπως φαίνεται και από τις λειτουργίες του, υπάρχουν οι καταχωρητές AC (συσσωρευτής) ,στον οποίο θα εμφανίζονται τα αποτελέσματα των πράξεων και DR (Data Register), όπου φορτώνονται τα περιεχόμενα της μνήμης που θα υποστούν επεξεργασία (operand). Φυσικά για να εμφανίσει η μνήμη κάποια δεδομένα στους ακροδέκτες δεδομένων της θα πρέπει να της "δείξουμε" ποιας διεύθυνσης τα δεδομένα επιθυμούμε να μας εμφανίσει. Αυτό θα γίνει μέσω του AR (Address register). Ο υπολογιστής φυσικά θα διαθέτει μνήμη στην οποία θα αποθηκεύουμε το πρόγραμμα και τα αποτελέσματα των πράξεων.
Κάθε εντολή του προγράμματος οδηγείται στη μονάδα ελέγχου, όπου γίνεται η αποκωδικοποίηση της και με βάση αυτή την αποκωδικοποίηση προσδιορίζεται ποια δεδομένα θα υποστούν επεξεργασία (operands) και τι είδους επεξεργασία θα υποστούν (opcode). Η διαδικασία αυτή της ανάκτησης μίας εντολής από την μνήμη ονομάζεται φάση ή κύκλος ανάκλησης για την οποία θα μιλήσουμε παρακάτω. Απαραίτητη, λοιπόν, κρίνεται η χρήση ενός καταχωρητή όπου θα αποθηκεύεται η εντολή προκειμένου να πραγματοποιηθεί η αποκωδικοποίηση της. Ο καταχωρητής αυτός ονομάζεται IR (instruction register).
Επόμενος πολύ σημαντικός καταχωρητής και κοινός σε όλα τα ψυφιακά κυκλώματα είναι ο μετρητής προγράμματος PC. Κάθε πρόγραμμα αποτελεί μία ακολουθία εντολών, αποθηκευμένων στην μνήμη. Οι εντολές είναι αποθηκευμένες στη μνήμη του υπολογιστή σε διαδοχικές θέσεις. Εμείς θα πρέπει να διατρέχουμε τις θέσεις μνήμης στις οποίες είναι αποθηκευμένες οι εντολές, ώστε να τις μεταφέρουμε μία μία στον IR, να αποκωδικοποιηθούν και να συνεχιστεί η διαδικασία. Ο PC είναι ένας μετρητής ο οποίος είναι υπεύθυνος για τη ροή των εντολών. Όταν ξεκινάει η εκτέλεση του προγράμματος ο PC έχει ως περιεχόμενο τη διεύθυνση της μνήμης όπου έχουμε αποθηκεύσει την πρώτη εντολή του προγράμματος μας. Ο PC αυξάνει το περιεχόμενο του κατά 1, ώστε να δείχνει την επόμενη θέση μνήμης στην οποία βρίσκεται η δεύτερη εντολή του προγράμματος και ούτω καθεξής. Έτσι επιτυγχάνεται η ανάκληση όλων των εντολών από τη μνήμη.

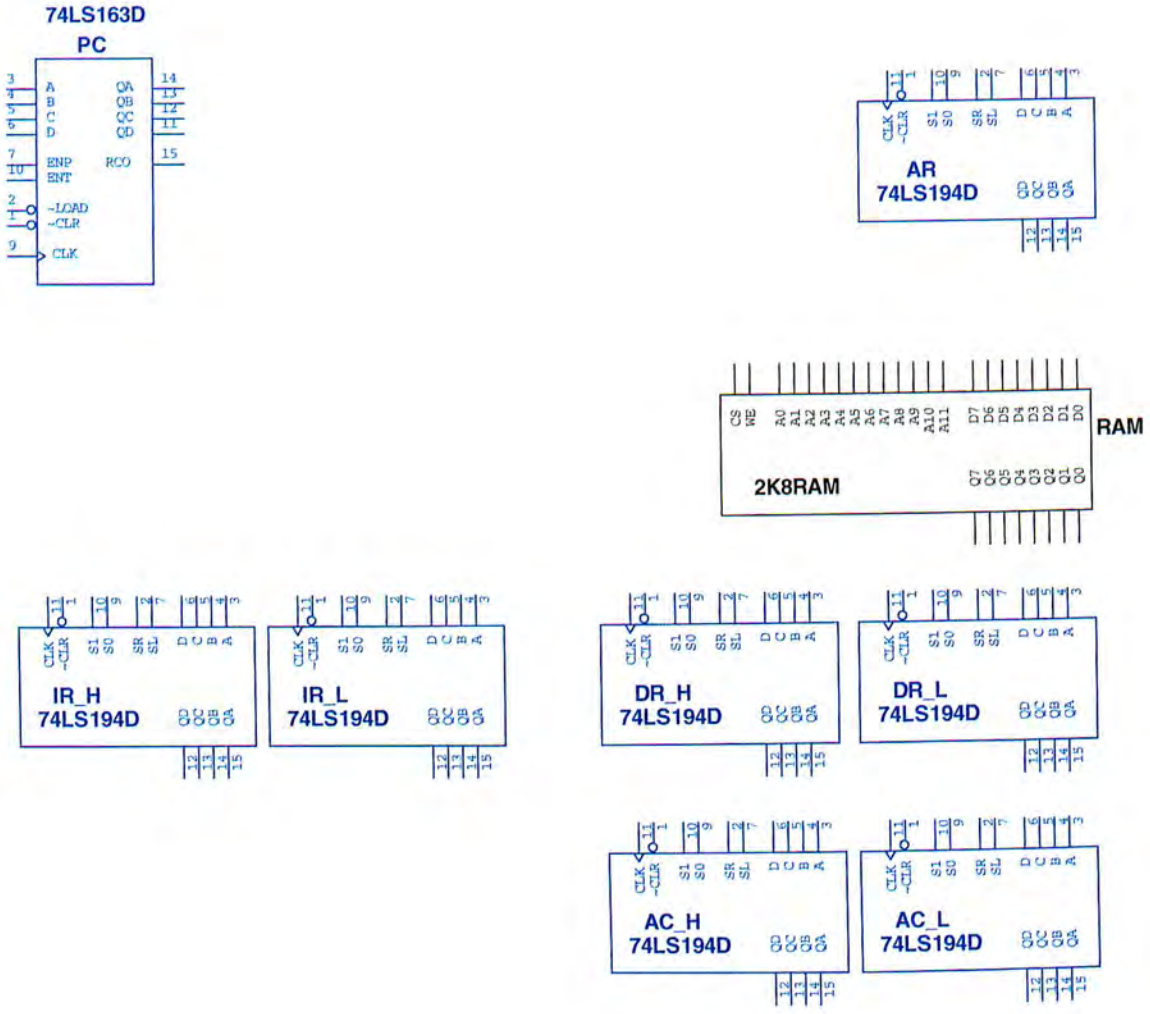
5.3 Μέγεθος καταχωρητών

Στο σημείο αυτό πρέπει να καθορίσουμε το μέγεθος των κυκλωμάτων που θα χρησιμοποιήσουμε. Για να γίνει αυτό πρέπει πρώτα να αποφασίσουμε το μήκος των δεδομένων που θέλουμε να επεξεργάζεται ο υπολογιστής. Το μήκος λέξης της εφαρμογής

μας θα είναι 8-bit. Εφόσον καταφέρουμε να σχεδιάσουμε το κύκλωμα για μήκος λέξης 8-bit στη συνέχεια η σχεδίαση κυκλωμάτων μεγαλύτερου μήκους λέξης είναι αρκετά απλή αφού αρκεί να προεκτείνουμε το μέγεθος των στοιχείων του. Επόμενο βήμα είναι η επιλογή του μεγέθους της μνήμης. Για τις λειτουργίες που έχουμε διαλέξει να εκτελούνται μία μνήμη 16 byte αρκεί. Εφόσον διαχειριζόμαστε 8-bit δεδομένα το μέγεθος των DR και AC θα είναι 8-bit (αφού σε αυτούς αποθηκεύουμε δεδομένα). Σύμφωνα με το μέγεθος μνήμης που έχουμε επιλέξει αρκούν 4-bit για να παρασταθούν όλες οι διευθύνσεις μνήμης της (2^4=16). Επομένως για τους AR και PC θα χρησιμοποιήσουμε 4-bit καταχωρητές. Όπως έχουμε αναφέρει παραπάνω ο IR αποτελείται από τον opcode (3-bit) και τη διεύθυνση της μνήμης στην οποία βρίσκεται ο operand (4-bit). Άρα ένας 8-bit καταχωρητής φτάνει και περισσεύει και για τον IR. Θα χρειαστούμε, επίσης, έναν 4-bit μετρητή για την παραγωγή των χρονικών στιγμών που θα είναι υπεύθυνες για την εκτέλεση των εντολών του προγράμματος με τη σωστή σειρά (περαιτέρω ανάλυση θα δούμε αμέσως παρακάτω). Τέλος, θα χρησιμοποιήσουμε δύο αποκωδικοποιητές έναν για την αποκωδικοποίηση του opcode της εκάστοτε εντολής (3-σε-8) και έναν για την αποκωδικοποίηση του χρονιστή (4-σε-16).
Στον πίνακα 5.2 υπάρχουν συγκεντρωμένα τα μεγέθη κάθε καταχωρητή και στο σχήμα 5.1 πιο ολοκληρωμένο θα χρησιμοποιήσουμε στο Multisim για τον καθένα.

Σύμβολο	Μέγεθος	Chip
AR	4-bit	74LS194
DR	8-bit	74LS194
IR	8-bit	74LS194
RAM	16 byte	-
PC	4-bit	74LS163
AC	8-bit	74LS194
SC	4-bit	74LS163

Πίνακας 5.2 Συνοπτικός Πίνακας Καταχωρητών



Σχήμα 5.1 Οι καταχωρητές του υπολογιστή

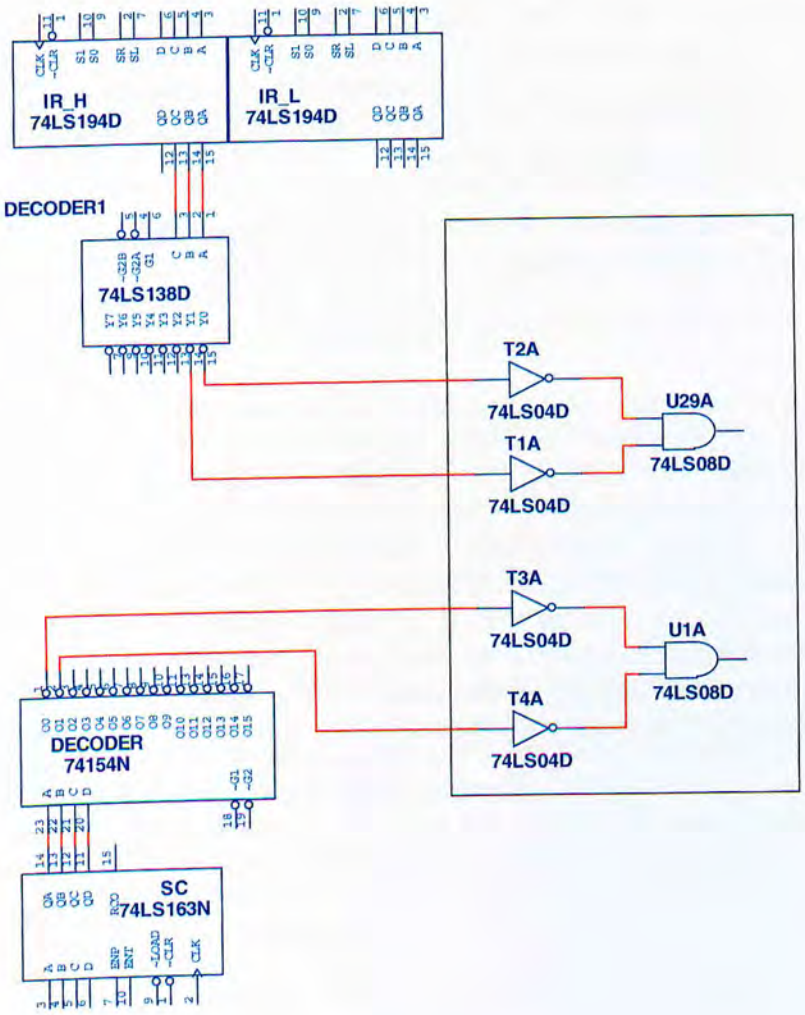
Ο τρόπος με τον οποίο θα διασυνδεθούν τα παραπάνω, καθώς και άλλα ψηφιακά στοιχεία όπως πολυπλέκτες που θα χρησιμοποιήσουμε θα παρουσιασθούν στη συνέχεια.

Σημείωση: όπως αναφέραμε και στο 1° κεφάλαιο οι καταχωρητές με μέγεθος μεγαλύτερο των 4-bit θα συνθέτονται από τον συνδυασμό 4-bitων καταχωρητών. Έτσι για τους DR, IR και AC χρησιμοποιούμε δύο ολοκληρωμένα 74LS194. Τα συγκεκριμένα ολοκληρωμένα έχουν παρουσιασθεί στο 1° κεφάλαιο.

5.4 Μονάδα ελέγχου

Το σπουδαιότερο και πιο ενδιαφέρον "κομμάτι" ενός υπολογιστή είναι η μονάδα ελέγχου. Έχουμε ήδη αναφερθεί στο καθολικό ρολόι του συστήματος, το οποίο "γεννά" τους πολύτιμους για τα υπολογιστικά συστήματα παλμούς ρολογιού. Οι παλμοί αυτοί εφαρμόζονται σε όλους τους καταχωρητές και τα flip flop και συγχρονίζουν τη λειτουργία τους ώστε να υπάρχει μία αρμονία σε όλο το σύστημα. Επίσης, έχουμε μιλήσει για την έννοια του ελέγχου και πιο συγκεκριμένα για τα σήματα με βάση τα οποία επιτυγχάνεται ο έλεγχος των κυκλωμάτων. Ως παράδειγμα θυμηθείτε την είσοδο φόρτωσης ενός καταχωρητή. Για να αποθηκεύσει τα δεδομένα που υπάρχουν στις εισόδους του, πρέπει να ενεργοποιηθεί η είσοδος φόρτωσης μέσω ενός σήματος που θα εφαρμοστεί σε αυτήν, την κατάλληλη στιγμή. Τα σήματα ελέγχου παράγονται στη μονάδα ελέγχου.

Στο σχήμα 5.2 μπορείτε να δείτε μία ενδεικτική μονάδα ελέγχου. Όπως παρατηρούμε αποτελείται από δύο αποκωδικοποιητές, ένα 4-bit μετρητή, έναν 8-bit καταχωρητή και αρκετές λογικές πύλες οι οποίες χρησιμοποιούνται για την παραγωγή των τελικών σημάτων ελέγχου που θα οδηγηθούν στις εισόδους των ολοκληρωμένων κυκλωμάτων.



Σχήμα 5.2 Μονάδα ελέγχου

Ο χρονιστής (ή **SC**: sequence counter) αυξάνει το περιεχόμενο του κατά ένα με κάθε παλμό του ρολογιού. Εφόσον πρόκειται για 4-bit μετρητή μπορεί να μετρήσει από το 0 μέχρι το 15 (0000 έως 1111). Η έξοδοι του μετρητή οδηγούνται στον 4-σε-16 αποκωδικοποιητή. Αυτός έχει την ικανότητα για κάθε δυαδικό αριθμό που δέχεται να ενεργοποιεί μόνο μία έξοδο, αυτήν που αντιστοιχεί στο δεκαδικό ισοδύναμο του δυαδικού αριθμού που εφαρμόζεται στις εισόδους του. Έτσι οι 16 διαφορετικές καταστάσεις στις οποίες μπορεί να βρεθεί ο μετρητής αποκωδικοποιούνται σε 16 ξεχωριστά σήματα τα οποία στη συνέχεια μπορούμε να τα οδηγήσουμε στις εισόδους των κυκλωμάτων ώστε να τα χρησιμοποιήσουμε σα σήματα ελέγχου. Με αυτό τον τρόπο έχουμε δημιουργήσει 16 ξεχωριστά **σήματα χρονισμού**.

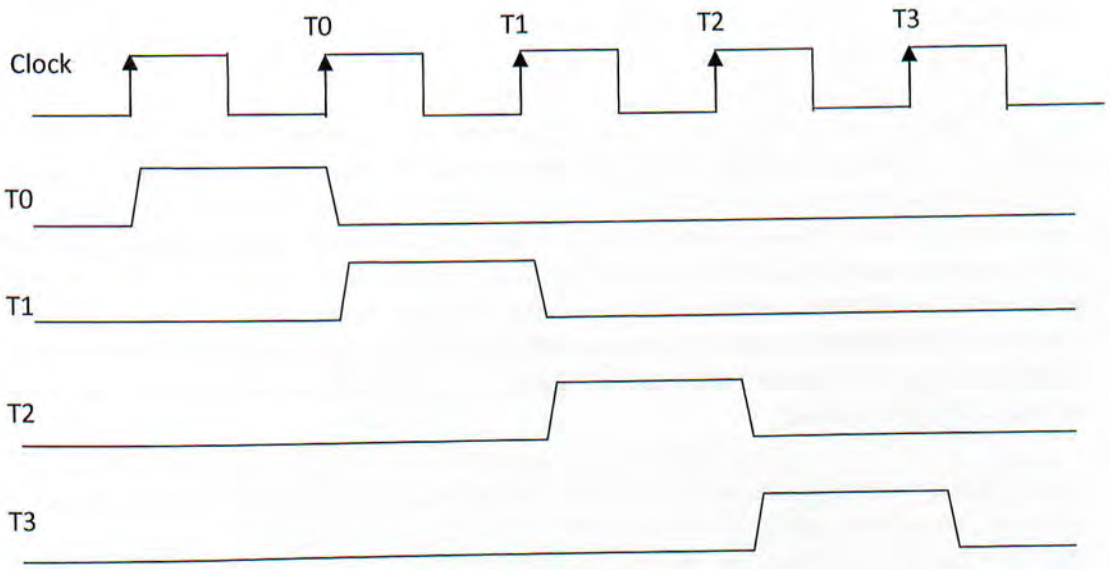
Οι εντολές του προγράμματος είναι τοποθετημένες στη μνήμη η μία μετά την άλλη. Ο χρονιστής χρησιμεύει ώστε να επιτυγχάνεται η διαδοχική εκτέλεση των μικρολειτουργιών που απαιτούνται για την πραγματοποίηση μίας εντολής με τη σωστή σειρά. Πριν εφαρμοστεί στον χρονιστή ο $1^{ος}$ παλμός του ρολογιού, είναι ενεργοποιημένη η έξοδος T_0 του αποκωδικοποιητή (θεωρούμε ότι πριν την ενεργοποίηση του υπολογιστή η τιμή του χρονιστή είναι 0). Κάθε σήμα χρονισμού είναι ενεργό για τη διάρκεια ενός κύκλου του ρολογιού. Πριν την έλευση του $1^{ου}$ παλμού του ρολογιού οδηγούνται στις εισόδους των κυκλωμάτων τα κατάλληλα σήματα σύμφωνα με τις απαιτήσεις μας. Με την έλευση του $1^{ου}$ παλμού (κατά τη θετική ακμή του) θα πραγματοποιηθούν οι μικρολειτουργίες που αντιστοιχούν στην συνθήκη T_0. Το περιεχόμενο του χρονιστή θα γίνει 0001 και θα ενεργοποιηθεί η έξοδος T_1 του αποκωδικοποιητή. Έτσι οι μικρολειτουργίες που απαιτούν να είναι αληθής η συνθήκη T_1 θα μπορούν πλέον να πραγματοποιηθούν κατά την θετική ακμή του επόμενου παλμού του ρολογιού.

Είναι πάρα πολύ σημαντικό να κατανοήσουμε τη διαφορά μεταξύ σήματος χρονισμού και παλμού του ρολογιού. Για παράδειγμα η παρακάτω εντολή:

$$T_0: AR \leftarrow PC$$

καθορίζει τη μεταφορά του περιεχομένου του PC στον AR, όταν και μόνο όταν το σήμα χρονισμού T_0 είναι ενεργό. Όπως προείπαμε το σήμα T_0 είναι ενεργό καθ' όλη τη διάρκεια μίας περιόδου του ρολογιού. Σε όλη αυτή τη διάρκεια πραγματοποιείται όλη η απαιτούμενη προετοιμασία π.χ. η ενεργοποίηση της εισόδου LD του AR, η επεξεργασία δεδομένων από τα συνδυαστικά κυκλώματα του υπολογιστή, η τοποθέτηση του περιεχομένου το PC στο δίαυλο δεδομένων (εάν είχαμε επιλέξει να διασυνδέσουμε τα κυκλώματα μας με δίαυλο) κ.α. Η φόρτωση, όμως, των δεδομένων στον AR θα πραγματοποιηθεί με την έλευση της θετικής ακμής του νέου παλμού, η οποία θα κάνει το περιεχόμενο του SC=0001 για να συνεχιστεί η ίδια διαδικασία με τις μικρολειτουργίες που αντιστοιχούν σε αυτό το σήμα χρονισμού.

Το παρακάτω διάγραμμα θα μας βοηθήσει να κατανοήσουμε καλύτερα τη διαφορά μεταξύ των δύο.



Όπως βλέπεται και στο διάγραμμα μόλις ενεργοποιήσουμε τον υπολογιστή θα είναι ενεργό το σήμα T_0, το οποίο παραμένει ενεργό μέχρι να έρθει ο πρώτος παλμός του ρολογιού. Καθ' όλη τη διάρκεια που είναι ενεργοποιημένο το T_0 γίνονται οι διάφορες προετοιμασίες όπως προείπαμε. Η στιγμή της μεταφοράς των δεδομένων, ωστόσο, είναι στην ακμή του παλμού του ρολογιού (όπου είναι σημειωμένο το T_0). Κατά την ακμή αυτή πραγματοποιούνται όλες οι μικρολειτουργίες που σχετίζονται με το σήμα χρονισμού T_0, στη συνέχεια το περιεχόμενο του SC γίνεται 0001 και ενεργοποιείται το σήμα χρονισμού T1 για να γίνουν τα ανάλογα.

Εκτός από την είσοδος Increment (αύξηση του περιεχομένου) υπάρχει και η είσοδος καθαρισμού CLR. Αυτή θα τη χρησιμοποιήσουμε όταν θα επιθυμούμε να μηδενίσουμε το περιεχόμενο του μετρητή και σαν συνέπεια να ξαναβρεθεί το σύστημα μας στην πρώτη χρονική στιγμή. Ως παράδειγμα σκεφτείτε το ενδεχόμενο να θέλουμε τη χρονική στιγμή T_4 να μηδενιστεί το περιεχόμενο του μετρητή. Αυτό μπορεί να αποτυπωθεί με την παρακάτω μικρολειτουργία:

$$T_4: SC \leftarrow 0$$

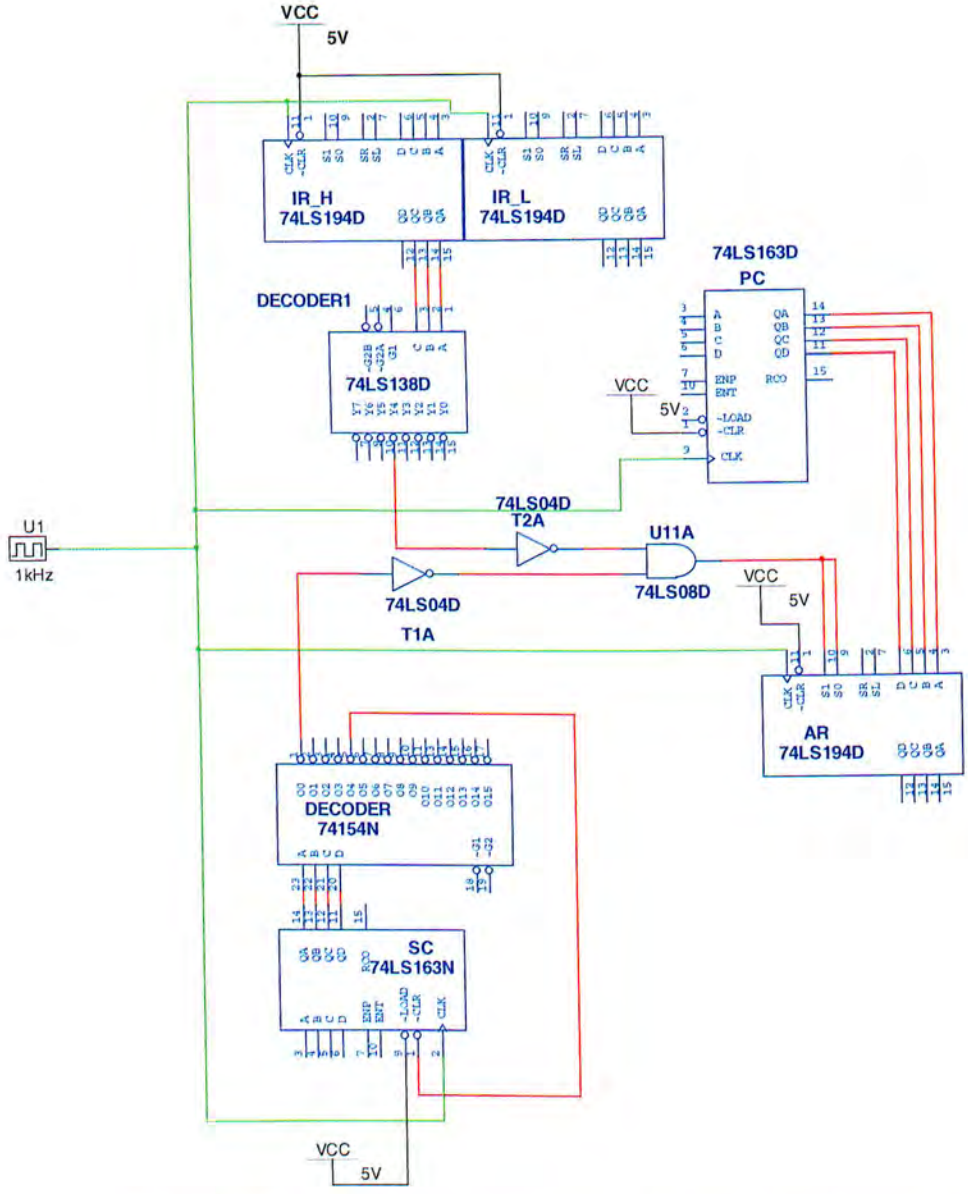
Η παραπάνω μικρολειτουργία επιτυγχάνεται αρκεί να οδηγήσουμε το σήμα χρονισμού T_4 από την έξοδο του αποκωδικοποιητή στην είσοδο καθαρισμού του μετρητή.
Όπως βλέπεται και πάλι στο διάγραμμα, αφού γίνει 1 το T_4 ενεργοποιείται η είσοδος CLR του SC. Επομένως, στον επόμενο παλμού του ρολογιού (αυτόν που σχετίζεται με το σήμα χρονισμού T4) μηδενίζεται το περιεχόμενο του SC και ενεργοποιείται το σήμα χρονισμού T_0, ξεκινώντας από την αρχή την όλη διαδικασία.

Στον IR μεταφέρονται οι εντολές του προγράμματος μας προκειμένου να αποκωδικοποιηθούν και από αυτές να προκύψουν οι τελεστές και το είδος της επεξεργασίας που αυτοί θα υποστούν (θυμηθείτε opcode και operand). Ο 3-σε-6 αποκωδικοποιητής χρησιμοποιείται και αυτός προκειμένου από τον 3-bit opcode να αποκτήσουμε ένα ξεχωριστό σήμα ελέγχου για κάθε opcode. Και πάλι τα σήματα αυτά θα

χρησιμοποιηθούν στις συναρτήσεις ελέγχου για τον έλεγχο τον κυκλωμάτων του υπολογιστή. Για τα ξεχωριστά αυτά χρησιμοποιούνται τα σύμβολα D_0, D_1, D_2... Ας δούμε τις παρακάτω μικρολειτουργίες:

$$T_0D_4: AR \leftarrow PC$$
$$T_4: SC \leftarrow 0$$

Η συνάρτηση ελέγχου είναι η T_0D_4. Όταν η τιμή αυτή της συνάρτησης είναι 1, δηλαδή την χρονική στιγμή που η έξοδος 0 του αποκωδικοποιητή χρονιστή και η έξοδος 4 του αποκωδικοποιητή λειτουργίας είναι 1 , τότε και μόνο τότε το περιεχόμενο του PC θα μεταφερθεί στον AR. Πιο συγκεκριμένα όταν $T_0D_4=1$ θα πρέπει να ενεργοποιηθεί η είσοδος φόρτωσης του AR (να γίνει και αυτή 1). Επομένως στην είσοδο φόρτωσης του AR τοποθετούμε την έξοδο μίας πύλης AND που έχει εισόδους τα T_0 και D4 αφού απαιτείται να είναι και τα δύο ενεργά για να πραγματοποιηθεί αυτή η μικρολειτουργία. Φυσικά όλα αυτά θα συμβούν με την έλευση του επόμενου παλμού του ρολογιού. Στο σχήμα 5.4 φαίνεται υλοποιημένο το κύκλωμα.



Σχήμα 5.4 Μεταφορά περιεχομένων μεταξύ καταχωρητών υπό συνθήκη

Όπως έχουμε αναφέρει στο κεφάλαιο 2 οι αποκωδικοποιητές έχουν χαμηλά ενεργές εξόδους. Επομένως απαιτούνται αναστροφείς προκειμένου να δημιουργηθούν τα επιθυμητά σήματα ελέγχου, εξού και η αναστροφείς στις εξόδους των αποκωδικοποιητών του σχήματος

Σημειώνουμε και πάλι ότι ο κύριος λόγος που χρειαζόμαστε τα σήματα χρονισμού είναι για να εκτελούνται με τη σωστή σειρά οι μικρολειτουργίες που απαιτούνται για την ολοκλήρωση μίας εντολής αλλά και για να δώσουμε στα κυκλώματα τον απαιτούμενο χρόνο για την προετοιμασία των δεδομένων πριν την μεταφορά τους. Δημιουργούμε μέσα στο σύστημα μας ξεχωριστές χρονικές στιγμές (με τη βοήθεια του ρολογιού) οι οποίες αποτελούν συνθήκες για την πραγματοποίηση των μικρολειτουργιών. Είναι θα λέγαμε μία γεννήτρια της αλληλουχίας εκτέλεσης των μικρολειτουργιών.

5.5 Κύκλος ανάκλησης και εκτέλεσης εντολών

Το πρόγραμμα αποτελείται από μία ακολουθία εντολών αποθηκευμένες η μία μετά την άλλη στη μνήμη RAM του υπολογιστή χωρίς κενά μεταξύ τους. Στόχος μας είναι η εκτέλεση των εντολών αυτών. Προκειμένου να εκτελεστεί μία εντολή θα πρέπει πρώτα να ανακληθεί – αποκτηθεί από τη μνήμη. Μετά την ανάκληση της θα βρίσκεται στον IR όπου θα αποκωδικοποιηθεί και μέσω της διεύθυνσης του operand (bit 0-3) και του opcode (bit 4-7) θα εξαχθεί από τη μνήμη ο operand (το προς επεξεργασία δεδομένο δηλαδή) και θα δημιουργηθούν τα σήματα ελέγχου με τα οποία θα πραγματοποιηθούν οι μικρολειτουργίες για την εκτέλεση της εκάστοτε εντολής.
Η διαδικασία, λοιπόν, που ακολουθείται από τον υπολογιστή για την πραγματοποίηση μίας λειτουργίας μπορεί να χωριστεί σε δύο φάσεις. Στην πρώτη ανήκουν οι μικρολειτουργίες που απαιτούνται για την ανάκληση της εντολής από τη μνήμη και στη δεύτερη όπου ανήκουν οι μικρολειτουργίες εκτέλεσης της εντολής. Οι δύο αυτές φάσεις ονομάζονται κύκλος ανάκλησης (fetch cycle) και κύκλος εκτέλεσης (execution cycle) αντίστοιχα.

Κύκλος ανάκλησης

Στόχος μας είναι να φτάσει η εντολή από τη θέση μνήμης στην οποία βρίσκεται, στον IR όπου αποκωδικοποιείται. Θα ακολουθήσουμε έναν "ανάποδο" τρόπο σκέψης. Γνωρίζοντας ποιο θέλουμε να είναι το αποτέλεσμα θα φτάσουμε μέχρι την αρχή της διαδικασίας. Επιθυμούμε στον IR να φορτωθεί η εντολή που βρίσκεται στην πρώτη θέση της μνήμης. Η μνήμη με τη σειρά της προκειμένου να εμφανίσει την εντολή αυτή στους ακροδέκτες δεδομένων εξόδου της θα πρέπει να φορτωθεί στον AR η διεύθυνση της μνήμης στην οποία η εντολή είναι αποθηκευμένη. Επομένως στον AR ΘΑ πρέπει να φορτώνουμε κάθε φορά τη διεύθυνση της μνήμης στην οποία βρίσκεται η εντολή που θέλουμε να εκτελεστεί. Εδώ πρέπει να θυμηθούμε τη λειτουργία του PC. Όπως έχουμε ξαναπεί ο PC είναι ο μετρητής προγράμματος και έχει σαν περιεχόμενο τη διεύθυνση της μνήμης στην οποία βρίσκεται η εντολή που θέλουμε να εκτελεσθεί. Επομένως η πρώτη μικρολειτουργία που θέλουμε να πραγματοποιηθεί είναι η μεταφορά του περιεχομένου του PC στον AR. Κατά την εκτέλεση της πρώτης μικρολειτουργίας το περιεχόμενο στο SC είναι 0 και επομένως το σήμα χρονισμού T_0 είναι ενεργό. Στον επόμενο παλμό του ρολογιού το περιεχόμενο του PC θα μεταφερθεί στον AR και ο SC θα αυξηθεί κατά 1 ενεργοποιώντας τη λογική μεταβλητή T_1. Η δεύτερη μικρολειτουργία του κύκλου ανάκλησης είναι η μεταφορά της εντολής από τη θέση της μνήμης που "δείχνει" ο AR, στον IR. Αφού φορτωθεί το περιεχόμενο του PC στον AR η δουλειά του PC όσον αφορά τη συγκεκριμένη εντολή έχει τελειώσει και θα πρέπει να

αυξήσουμε το περιεχόμενο του κατά 1 ώστε να δείχνει τη διεύθυνση της μνήμης όπου βρίσκεται η επόμενη εντολή. Με την έλευση του παλμού του ρολογιού όλες οι μικρολειτουργίες που αντιστοιχούν στη μεταβλητή ελέγχου T_1 εκτελούνται και το περιεχόμενο του SC αυξάνεται κατά 1 ενεργοποιώντας τη μεταβλητή ελέγχου T_2. Κατά τη διάρκεια αυτής της χρονικής περιόδου πραγματοποιείται η αποκωδικοποίηση του opcode (το κύκλωμα είναι συνδυαστικό επομένως η αποκωδικοποίηση πραγματοποιείται χωρίς την έλευση παλμού ρολογιού) και η μεταφορά των bit (0 - 3) της εντολής στον AR (θυμηθείτε ότι εκεί βρίσκεται η διεύθυνση της μνήμης στην οποία έχουμε αποθηκεύσει τον operand). Στη συνέχεια θέλουμε τη μεταφορά του operand στον DR προκειμένου να προχωρήσουμε σε περαιτέρω επεξεργασία του ανάλογα με τον opcode (κύκλος εκτέλεσης που θα δούμε παρακάτω). Επομένως το περιεχόμενο της διεύθυνσης της μνήμης που δέχεται η μνήμη στους ακροδέκτες διευθύνσεων της από τον AR θα φορτωθεί στον DR. Η μεταφορά αυτή θα συμβεί όταν είναι ενεργό το σήμα χρονισμού T_3. Όλα όσα είπαμε με λόγια μπορούν να παρασταθούν με το παρακάτω συμβολικό πρόγραμμα:

T_0: AR ← PC
T_1: IR ← M[AR], PC ← PC+1
T_2: D0, ..., D7 ← Decode IR (4-7), AR ← IR(0-3)
T_3: DR ← M[AR]

Κύκλος Εκτέλεσης

Μετά τον κύκλο ανάκλησης δε μένει παρά η εκτέλεση της εντολής. Οι μικρολειτουργίες του κύκλου ανάκλησης είναι ίδιες για όλες τις εντολές. Αντίθετα οι μικρολειτουργίες του κύκλου εκτέλεσης εξαρτώνται από τον opcode και ανάλογα με την τιμή αυτού (άρα ανάλογα με την έξοδο του αποκωδικοποιητή που είναι ενεργοποιημένη κάθε φορά) ενεργοποιούνται τα κατάλληλα σήματα τα οποία οδηγούμενα στα κατάλληλα κάθε φορά κυκλώματα πραγματοποιούν την επιθυμητή λειτουργία. Στον πίνακα 5.3 αναφέρονται όλες οι λειτουργίες που μπορεί να εκτελέσει ο υπολογιστής μας. Με βάση τον κώδικα λειτουργίας της τρίτης στήλης είναι εύκολο να συμπεράνουμε τους ακροδέκτες εξόδου του αποκωδικοποιητή που ενεργοποιούνται κάθε φορά. Για παράδειγμα εάν ο opcode της εντολής που βρίσκεται στον IR είναι ο 000 ενεργοποιείται η έξοδος D_0, για 001 ενεργοποιείται η D_1 και ούτω καθεξής. T_0 τελευταίο σήμα χρονισμού που ήταν ενεργό κατά τον κύκλο ανάκλησης ήταν το T_3 επομένως η πρώτη μικρολειτουργία του κύκλου εκτέλεσης θα πραγματοποιείται όταν είναι ενεργό το T_4. Σε περίπτωση που μία λειτουργία απαιτεί δύο ακμές του ρολογιού για να πραγματοποιηθεί τότε η αρίθμηση των σημάτων χρονισμού συνεχίζει με T_5, T_6... (Στη δική μας περίπτωση όλες οι λειτουργίες πραγματοποιούνται με τον ίδιο παλμό ρολογιού). Στον πίνακα 5.3 υπάρχουν συνοπτικά για κάθε λειτουργία του υπολογιστή το σύμβολο που θα χρησιμοποιούμε, η έξοδος που αποκωδικοποιητή που ενεργοποιείται κάθε φορά καθώς και η περιγραφή του στη γλώσσα μεταφοράς καταχωρητών.

Σύμβολο	Έξοδος Αποκωδικοποιητή	Συμβολική Περιγραφή
LDA	D_0	AC ← DR
ADD	D_1	AC ← AC+DR
OR	D_2	AC ← AC v DR
AND	D_3	AC ← AC ^ DR
SUB	D_4	AC ← AC - DR
WRITE	D_5	M[AR] ← AC
HLT	D_6	START ← CLR

Πίνακας 5.3 Οι λειτουργίες του υπολογιστή

Καθαρισμός SC

Ο SC αυξάνει το περιεχόμενο του κατά 1 με κάθε παλμό του ρολογιού. Για το λόγο αυτό δε θα γράφουμε σε κάθε γραμμή του συμβολικού μας προγράμματος τη μικρολειτουργία SC ← SC + 1. Ωστόσο αφού πραγματοποιηθεί ο κύκλος της εκτέλεσης μίας εντολής θα θέλουμε ο υπολογιστής να ξαναβρεθεί στη κύκλος ανάκλησης προκειμένου να κάνει την ίδια διαδικασία και για την επόμενη εντολή. Όπως καταλαβαίνεται αυτό μπορεί αν επιτευχθεί μηδενίζοντας το περιεχόμενο του SC, επαννεκινώντας δηλαδή τον δείκτη της σειράς των μικρολειτουργιών (T_0, T_1, T_2...). Με το τέλος της εκτέλεσης της κάθε λειτουργίας θα πραγματοποιείται και μηδενισμός του SC γι' αυτό και στην τελευταία γραμμή του συμβολικού μας προγράμματος θα συμπεριλαμβάνουμε τη μικρολειτουργία SC ← 0.

Εντολή LDA

Με τη λειτουργία αυτή θα πραγματοποιείται απλή φόρτωση του περιεχομένου του DR στον AC. Η μεταβλητή ελέγχου για αυτή την λειτουργία χρησιμοποιεί την έξοδο του αποκωδικοποιητή D_0 και το σήμα χρονισμού T_4. Τέλος, θα μηδενίζουμε το περιεχόμενο του SC για να ανακαλέσει ο υπολογιστής την επόμενη εντολή από τη μνήμη. Οι μικρολειτουργίες που πραγματοποιούν αυτή τη λειτουργία είναι:

D_0T_4: AC ← DR, SC ←0

Όσον αφορά τις συνδέσεις αρκεί να περάσουμε τα σήματα D_0 και T_4 από μία πύλη AND και το αποτέλεσμα τους να το οδηγήσουμε τόσο στην είσοδο φόρτωσης του AC όσο και στην είσοδο καθαρισμού του SC. Μην ξεχνάμε ότι δεν υπάρχει άμεση σύνδεση του DR και του AC αλλά αυτό συμβαίνει μέσω της ALU. Επομένως στις εισόδους S_0 – S_3 της ALU θα πρέπει να οδηγήσουμε τα κατάλληλα σήματα για να περάσει αναλλοίωτο το περιεχόμενο του DR. Αυτό θα συμβεί με την σχεδίαση ενός κωδικοποιητή που θα δούμε παρακάτω. Όσον αφορά τις συνδέσεις των AC και DR εργαζόμαστε ανάλογα με το σχήμα στην παράγραφο .

Εντολές ADD και SUB

Οι δύο αυτές λειτουργίες πραγματοποιούν αριθμητική πρόσθεση (ADD) και αφαίρεση (SUB) των δεδομένων του AC και του DR και το αποτέλεσμα τους αποθηκεύεται στον AC. Θα χρησιμοποιείται το ίδιο σήμα χρονισμού (T_4) με την εντολή LDA καθώς αντί για την προηγούμενη εντολή θα εκτελεσθεί αυτή, αμέσως μετά το τέλος του κύκλου ανάκλησης. Η

διαφορά έγκειται στον κώδικα λειτουργίας και άρα στην έξοδο του αποκωδικοποιητή λειτουργίας που είναι ενεργοποιημένη. Οι μικρολειτουργίες σε συμβολική γλώσσα είναι οι εξής:

Για την πράξη ADD έχουμε: D_1T_4: AC ← AC + DR, SC ← 0

Για την πράξη SUB έχουμε: D_4T_4: AC ← AC – DR, SC ← 0

Εντολές AND και OR

Με τις λειτουργίες αυτές πραγματοποιούμε λογικό πολλαπλασιασμό (AND) και λογική πρόσθεση (OR) μεταξύ των περιεχομένων του DR και του AC. . Οι μικρολειτουργίες σε συμβολική γλώσσα είναι οι εξής:

Για την πράξη AND έχουμε: D_3T_4: AC ← AC ^ DR, SC ← 0

Για την πράξη OR έχουμε: D_2T_4: AC ← AC v DR, SC ← 0

Σε ότι έχει να κάνει με τις συνδέσεις που πρέπει να γίνουν στα κυκλώματα του υπολογιστή τόσο για τις αριθμητικές όσο και για τις λογικές πράξεις είναι πολύ απλά η δημιουργία και πάλι των αντίστοιχων σημάτων ελέγχου με πύλες AND (D_1T_4, D_3T_4, ...) και η ενεργοποίηση της σωστής κάθε φορά πράξης μέσω των εισόδων της ALU.

Εντολή WRITE

Κατά τη κύκλος ανάκλησης χρησιμοποιούμε την ανάγνωσης δεδομένων από τη μνήμη. Μετά την επεξεργασία τους θα έχουμε τη δυνατότητα να αποθηκεύουμε τα αποτελέσματα των διαφόρων πράξεων από τον AC στη RAM του υπολογιστή. Αυτό επιτυγχάνεται με τη λειτουργία της εγγραφής στη μνήμη.

D_5T_4: M[AR] ← AC, SC ← 0

Για την εκτέλεση αυτής της εντολής θα πρέπει η έξοδος της πύλης AND με εισόδους τα D5 και T_4 να οδηγηθεί στις εισόδους CS και WE προκειμένου να ενεργοποιηθεί η λειτουργία γραφής της μνήμης. Η εντολή αυτή δεν απαιτεί την ανάγνωση ενός operand από τη μνήμη.. Ωστόσο, στα bit (0-3) του κώδικα εντολής τοποθετούμε τη διεύθυνση της μνήμης στην οποία θέλουμε να αποθηκευτεί το αποτέλεσμα – περιεχόμενο του AC, αντί για τη θέση της μνήμης στην οποία βρίσκεται ο operand, όπως κάνουμε στις υπόλοιπες εντολές που έχουμε δει. Αυτή η διεύθυνση θα βρίσκεται στον AR εξαιτίας της αντίστοιχης μικρολειτουργίας κατά τη χρονική στιγμή T_4 και κατά την εκτέλεση της γραφής τα δεδομένα του AC θα μεταφερθούν στη διεύθυνση της μνήμης που δείχνει ο AR.

Εντολή HLT

Έχουμε αναφέρει ότι ο υπολογιστής θα ξεκινάει τη λειτουργία του όταν θα κάνουμε ON έναν διακόπτη. Αυτός θα θέτει ένα μανδαλωτή τύπου D, η κανονική έξοδος του οποίου θα

εφαρμόζεται στις εισόδους ενεργοποίησης του SC και αυτός θα ξεκινάει την αρίθμηση του. Θεωρούμε απαραίτητο να προσθέσουμε στις λειτουργίες του υπολογιστή και το σταμάτημα της λειτουργίας του, προκειμένου να μην εκτελεί το ίδιο πρόγραμμα πολλές φορές, καθώς δε θα είχε κανένα νόημα για εμάς. Η εντολή αυτή είναι η εντολή σταματήματος HLT και η μικρολειτουργία που την πραγματοποιεί είναι η παρακάτω:

$$D_6T_4: \text{Start latch} \leftarrow \text{CLR}$$

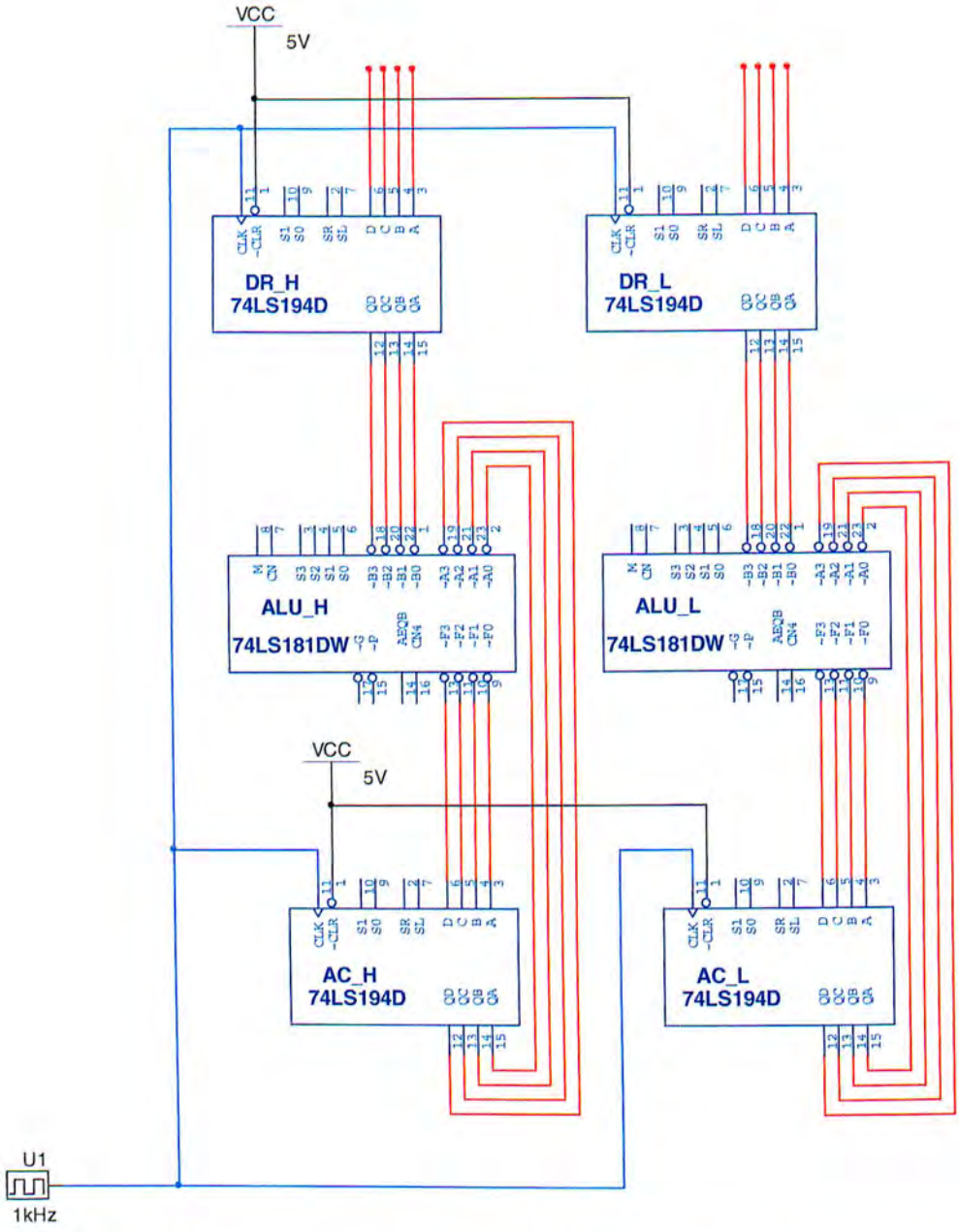
Η συνάρτηση ελέγχου D_6T_4 οδηγούμενη στην είσοδο καθαρισμού του μανδαλωτή μηδενίζει το περιεχόμενο του με αποτέλεσμα να σταματάει και η λειτουργία του SC άρα και η εκτέλεση του προγράμματος.

5.6 Μονάδα επεξεργασίας

Σε αυτή την υποενότητα θα ασχοληθούμε με τη Μονάδα Επεξεργασίας του υπολογιστή. Πρόκειται για την μονάδα εκείνη η οποία εκτελεί τις αριθμητικές και λογικές πράξεις μεταξύ των δεδομένων. Υπάρχουν αρκετοί τρόποι οργάνωσης της μονάδας επεξεργασίας και ανάλογα με τις απαιτήσεις της εφαρμογής μας επιλέγουμε τον καταλληλότερο. Ωστόσο εμείς θα αναφερθούμε μόνο σε αυτόν που θα χρησιμοποιήσουμε και δεν είναι άλλος από την Οργάνωση Συσσωρευτή, η οποία είναι και η οργάνωση που χρησιμοποιείται περισσότερο.

Έχουμε αναφερθεί ήδη στον καταχωρητή AC (accumulator ή συσσωρευτής) ως τον καταχωρητή στον οποίο εμφανίζονται τα τελικά αποτελέσματα των πράξεων που επιθυμούμε να εκτελεστούν. Ονομάζεται συσσωρευτής διότι σε αυτόν συσσωρεύονται όλα τα αποτελέσματα των διαφόρων πράξεων. Στο σχήμα 5.4 φαίνεται η οργάνωση συσσωρευτή που θα αποτελεί τη μονάδα επεξεργασίας του υπολογιστή μας. Όπως παρατηρούμε η μονάδα επεξεργασίας αποτελείται από τον DR, δηλαδή τον καταχωρητή που δέχεται τον operand από τη μνήμη, την ALU (το συνδυαστικό κύκλωμα που πραγματοποιεί τις αριθμητικές και λογικές πράξεις) και τον συσσωρευτή AC. Η επιλογή και το μέγεθος των καταχωρητών είναι ίδια με αυτή στο σχήμα 5.3. Το ολοκληρωμένο 74LS181 είναι μία 4-bit ALU, πραγματοποιεί δηλαδή πράξεις μεταξύ δύο δεδομένων μήκους 4-bit. Αντίστοιχα με τους καταχωρητές έτσι και για την ALU χρησιμοποιούμε δύο ολοκληρωμένα κυκλώματα, ένα για την επεξεργασία των τεσσάρων χαμηλής τάξης bit και ένα για την επεξεργασία των τεσσάρων υψηλής τάξης bit.

Σύμφωνα με την τελευταία μικρολειτουργία της φάσης ανάκλησης ο DR δέχεται δεδομένα από τη μνήμη. Στη συνέχεια τα αποτελέσματα των πράξεων μεταξύ των δεδομένων του AC και του DR (τα οποία υπολογίζονται στην ALU) φορτώνονται στον AC, επομένως τόσο οι έξοδοι του DR όσο και του AC οδηγούνται στις εισόδους της ALU. Τέλος, οι έξοδοι της ALU θα φορτωθούν όπως ήταν αναμενόμενο στον AC. Δε θα πρέπει όμως να ξεχνάμε ότι η ALU είναι ένα συνδυαστικό κύκλωμα, άρα εμφανίζει τα αποτελέσματα στην έξοδο της χωρίς να απαιτείται παλμός χρονισμού.



Σχήμα 5.4 Μονάδα Επεξεργασίας με Οργάνωση Συσσωρευτή

5.7 Σχεδίαση Κωδικοποιητή για τον έλεγχο της ALU

Όπως σε κάθε κύκλωμα του υπολογιστή έτσι και στην ALU ο έλεγχος της πραγματοποιείται από τις συναρτήσεις ελέγχου που παράγονται στη Μονάδα Ελέγχου. Σύμφωνα με όσα έχουμε αναφέρει, ανάλογα με τον κώδικα λειτουργίας (opcode) κάθε εντολής δημιουργείται και ένα διαφορετικό σήμα ελέγχου το οποίο ενεργοποιεί τις κατάλληλες εισόδους επιλογής της ALU. Από τον αποκωδικοποιητή λειτουργίας λαμβάνουμε 7 διαφορετικά σήματα, ένα για κάθε λειτουργία που μπορεί να εκτελέσει ο υπολογιστής. Ωστόσο η ALU διαθέτει πέντε εισόδους ελέγχου, τέσσερις οι S0 έως S3 και μία το M. Ένας διαφορετικός συνδυασμός δυαδικών αριθμών στις εισόδους της απαιτείται για την πραγματοποίηση κάθε πράξης. Για παράδειγμα για την αριθμητική πρόσθεση θα πρέπει να εφαρμόσουμε στις εισόδους επιλογής της τις παρακάτω τιμές M=0, S0=1, S1=0, S2=0, S3=1 Θα μπορούσαμε, λοιπόν, να οδηγήσουμε την έξοδο της πύλης AND που παράγει το σήμα ελέγχου D_1T_4 (είναι η συνάρτηση ελέγχου που σηματοδοτεί την αριθμητική πρόσθεση) στις

εισόδους ελέγχου S0 και S3. Ωστόσο, οι είσοδοι αυτοί ενεργοποιούνται και για την λογική πρόσθεση, μαζί όμως με τις εισόδους ελέγχου S2, S1 και M. Τη λύση θα έδινε η χρήση πυλών OR. Καταλαβαίνεται ότι για τις διάφορες πράξεις που θέλουμε να πραγματοποιεί ο υπολογιστής θα πρέπει από κάθε διακριτό σήμα ελέγχου να παράγουμε έναν διαφορετικό δυαδικό αριθμό για τις εισόδους της ALU. Η παραπάνω φράση μας οδηγεί στη σχεδίαση ενός κωδικοποιητή.

Θα δουλέψουμε ανάλογα με το υποκεφάλαιο 2.5 όπου σχεδιάσαμε τον πρώτο μας κωδικοποιητή. Αρχικά θα κάνουμε τον πίνακα αληθείας του κωδικοποιητή όπου οι διάφορες μεταβλητές ελέγχου που παράγει ο αποκωδικοποιητής λειτουργίας (D0 έως D4) θα αντιστοιχιστούν στις τιμές που απαιτεί η ALU στις εισόδους της προκειμένου να πραγματοποιήσει την εκάστοτε λειτουργία. Παραδείγματος χάριν όταν θέλουμε να πραγματοποιηθεί η φόρτωση του operand στον AC (LOAD) ο opcode έχει τιμή 000. Επομένως η έξοδος D_0 του αποκωδικοποιητή είναι ενεργοποιημένη (τοποθετούμε έναν 1 κάτω από D_0 του πινάκα αληθείας). Οι υπόλοιπες έξοδοι δεν είναι ενεργοποιημένες επομένως έχουν τιμή 0. Για τον κωδικοποιητή αυτή η τιμή θα είναι η είσοδος. Από την άλλη πλευρά για να αφήσει η ALU το δεδομένο που "βλέπει" στις εισόδους δεδομένων της να περάσει άθικτο θα πρέπει οι είσοδοι ελέγχου της να έχουν τις παρακάτω τιμές S3=1, S2=0, S1=1, S0=0, M=1. Δηλαδή στη λειτουργία φόρτωσης που έχει opcode 000 και επομένως το D0 είναι ενεργοποιημένο αντιστοιχεί ο δυαδικός αριθμός 10101 στις εισόδους επιλογής της ALU. Με αυτά τα δεδομένα μπορούμε να συμπληρώσουμε την πρώτη γραμμή του πίνακα αληθείας. Με ανάλογους συλλογισμούς μπορούμε να γεμίσουμε όλες τις γραμμές του πίνακα. Στον Πίνακα 5.4 δεν αναγράφεται στο σήμα χρονισμού καθώς είναι ίδιο για όλες μικρολειτουργίες.

	Είσοδοι					Έξοδοι				
Μικρολειτουργία	D_0	D_1	D_2	D_3	D_4	S_3	S_2	S_1	S_0	M
LOAD	1	0	0	0	0	1	0	1	0	1
ADD	0	1	0	0	0	1	0	0	1	0
OR	0	0	1	0	0	1	1	0	0	1
AND	0	0	0	1	0	1	0	1	1	1
SUB	0	0	0	0	1	0	1	1	0	0

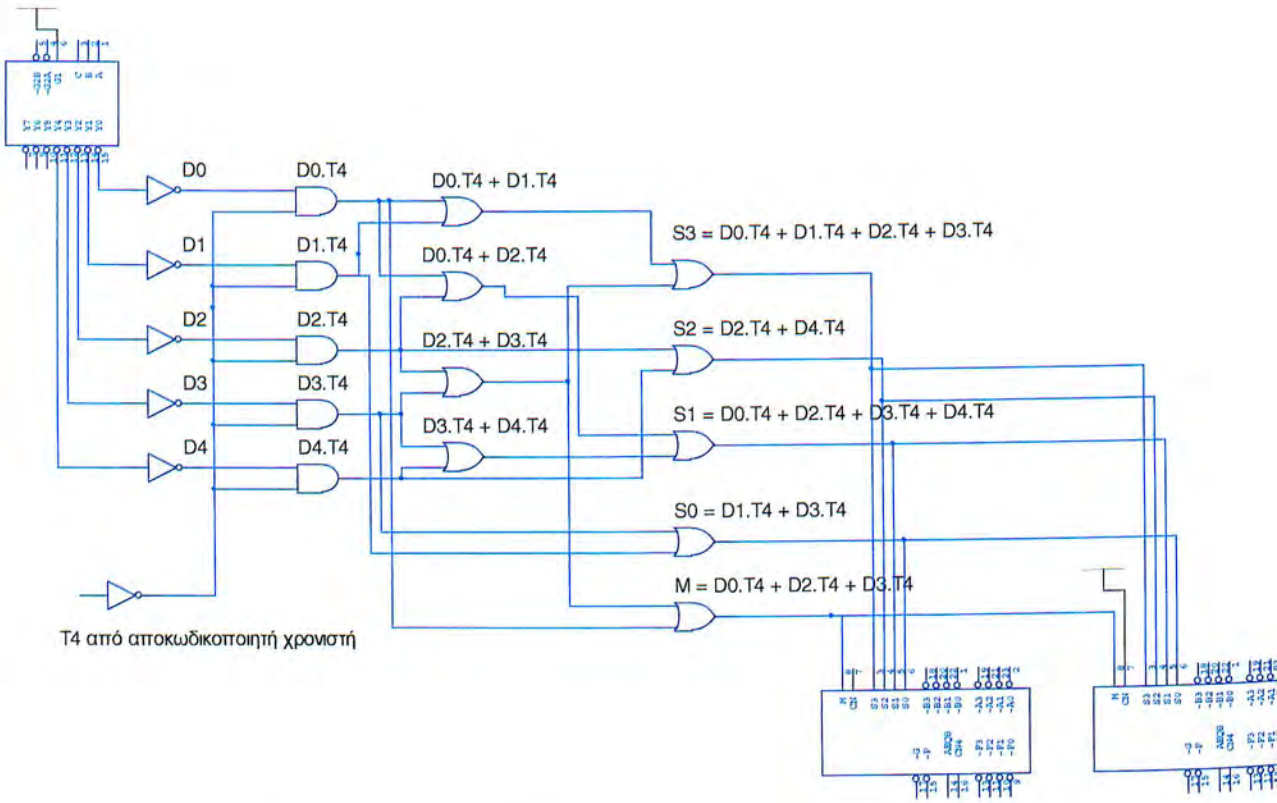
Πίνακας 5.4 Πίνακας Αληθείας Κωδικοποιητή

Από τον πίνακα αληθείας μπορούμε να εξακριβώσουμε με την ενεργοποίηση κάθε εξόδου του αποκωδικοποιητή (οι οποίες είναι και είσοδοι του κωδικοποιητή λειτουργίας), ποιες είσοδοι της ALU θα ενεργοποιηθούν (οι οποίες είναι και έξοδοι του κωδικοποιητή). Σκανάρωντας τον πίνακα μπορούμε να εξάγουμε τις συναρτήσεις Boole των εξόδων του κωδικοποιητή:

$$S3 = (D0 + D1 + D2 + D3).T4$$
$$S2 = (D2 + D4).T4$$
$$S1 = (D0 + D2 + D3 + D4).T4$$
$$S0 = (D1 + D3).T4$$
$$M = (D0 + D2 + D3).T4$$

Πλέον από τον πίνακα αληθείας είναι πολύ εύκολο να σχεδιάσουμε τον κωδικοποιητή με χρήση πυλών OR. Φυσικά δε θα πρέπει να ξεχνάμε ότι το σήμα ελέγχου δεν είναι μόνο οι

έξοδοι του αποκωδικοποιητή λειτουργίας D0...D4, καθώς οι συναρτήσεις ελέγχου που έχουμε γράψει στο συμβολικό μας πρόγραμμα εξαρτώνται από τη χρονική στιγμή T4. Στο σχήμα 5.5 μπορείτε να δείτε τον αποκωδικοποιητή λειτουργίας μαζί με τον κωδικοποιητή και την ALU



Σχήμα 5.5 Κύκλωμα Κωδικοποιητή ελέγχου ALU

Είναι προφανές ότι μόνο στις μικρολειτουργίες στις οποίες συμμετέχει η ALU ενεργοποιούνται οι είσοδοι επιλογής της. Επομένως για τις μικρολειτουργίες WRITE και HLT, στην εκτέλεση των οποίων δε συμμετέχει η ALU, δε χρειάζεται να μεριμνήσουμε για τις τιμές των εισόδων επιλογής της.

Κεφάλαιο 6ο
Σύνθεση του Τελικού Κυκλώματος

Στο 6° και τελευταίο κεφάλαιο θα παρουσιάσουμε το τελικό κύκλωμα του υπολογιστή που θα πραγματοποιεί τις αριθμητικές και λογικές πράξεις που επιθυμούμε. Όπως αναφέραμε στο κεφάλαιο 3 μέσω του συμβολικού μας προγράμματος μπορούμε να εξακριβώσουμε τη διασύνδεση των στοιχείων του υπολογιστή. Το πρόγραμμα αυτό θα είναι ο οδηγός μας για να οργανώσουμε τις εσωτερικές συνδέσεις των κυκλωμάτων (καταχωρητών, μετρητών κ.α.) που θα τον αποτελούν. Μπορούμε με λίγα λόγια να καταλαβαίνουμε, κάθε φορά, από ποιον καταχωρητή μεταφέρονται δεδομένα και σε ποιον θα φορτωθούν, ποιες μικρολειτουργίες γίνονται πάνω στα δεδομένα και φυσικά υπό ποιες συνθήκες θα γίνουν οι εκάστοτε μικρολειτουργίες.

6.1 Το Συμβολικό Πρόγραμμα του Υπολογιστή

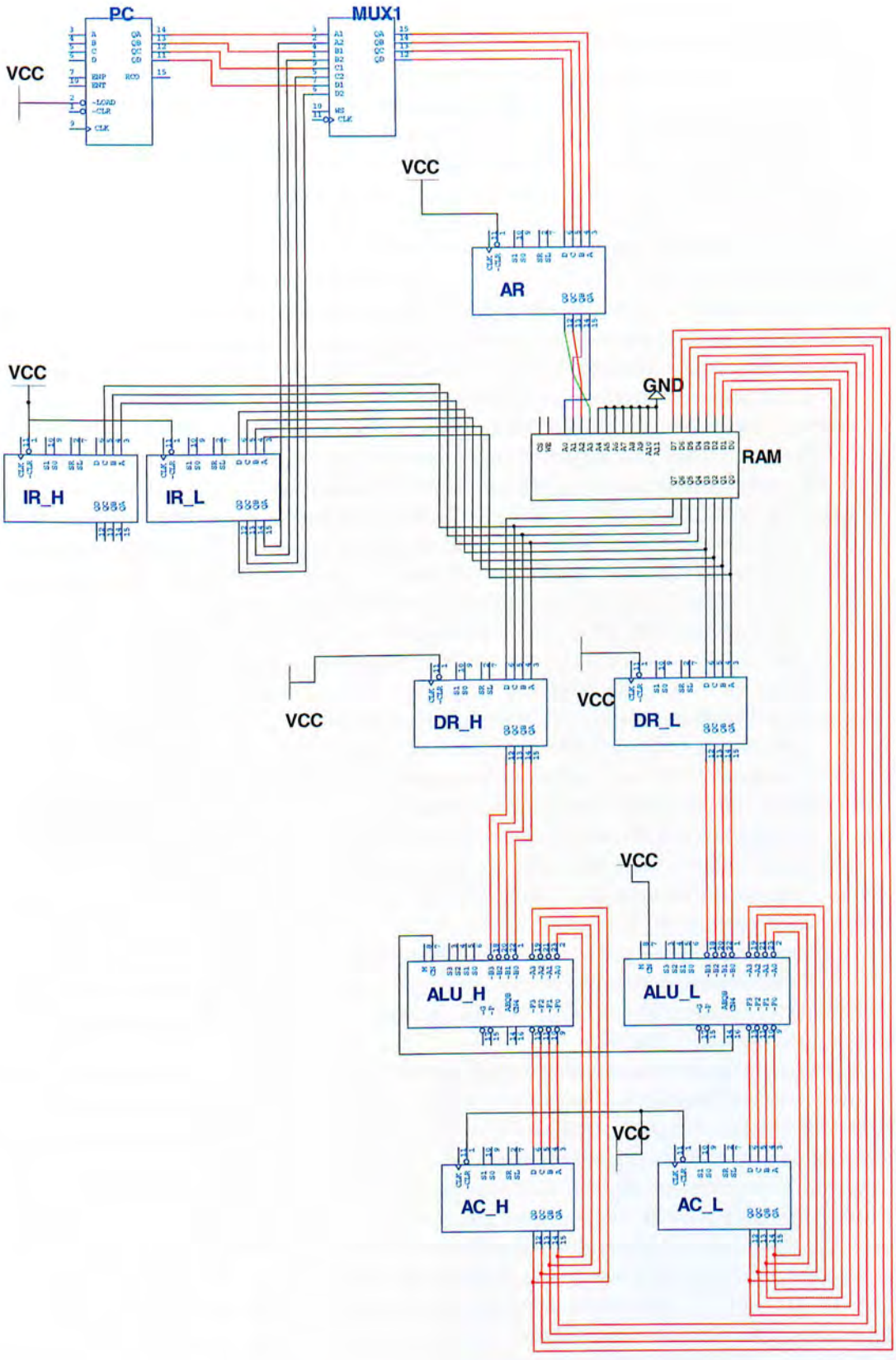
Το πρώτο βήμα για τη σχεδίαση του υπολογιστή είναι η επιλογή των λειτουργιών που θέλουμε να πραγματοποιεί, πράγμα το οποίο κάναμε στο υποκεφάλαιο 5.1. Στη συνέχεια στο υποκεφάλαιο 5.2 αποφασίσαμε με βάση τις λειτουργίες το σύνολο των καταχωρητών αλλά και το μέγεθος τους.
Επόμενο βήμα είναι η δημιουργία του συμβολικού προγράμματος, το οποίο θα περιγράφει τις μικρολειτουργίες που πρέπει να πραγματοποιηθούν προκειμένου μία εντολή να ανακληθεί από τη μνήμη (κύκλος ανάκλησης) και στη συνέχεια να εκτελεστεί (κύκλος εκτέλεσης). Η επανάληψη αυτής της διαδικασίας για κάθε εντολή θα έχει σαν αποτέλεσμα το "τρέξιμο" όλου του προγράμματος. Παρακάτω δίνουμε το συμβολικό πρόγραμμα της εφαρμογής μας, σύμφωνα με όσα έχουμε πει για τον κύκλο ανάκλησης και εκτέλεσης των εντολών:

T_0: AR ← PC
T_1: IR ← M[AR], PC ← PC+1
T_2: D0, ..., D7 ← Decode IR (4-7), AR ← IR(0-3)
T_3: DR ← M[AR]
D_0T_4: AC ← DR, SC ←0
D_1T_4: AC ← AC + DR, SC ← 0
D_2T_4: AC ← AC v DR, SC ← 0
D_3T_4: AC ← AC ^ DR, SC ← 0
D_4T_4: AC ← AC – DR, SC ← 0
D_5T_4: M[AR] ← AC, SC ← 0
D_6T_4: Start latch ← CLR, SC ← 0

6.2 Σύνδεση Δομικών Στοιχείων

Στο 3° κεφάλαιο δείξαμε ένα πρώτο παράδειγμα του τρόπου με τον οποίο από τον συμβολικό πρόγραμμα προχωρήσαμε στη σύνδεση των στοιχείων και τελικά στην σχεδίαση του κυκλώματος που μπορεί να πραγματοποιήσει τη λειτουργία που επιθυμούμε.

Ακλουθώντας τον ίδιο τρόπο σκέψης θα καταλήξουμε στη σύνδεση των δομικών στοιχείων του υπολογιστή μας με βάση το νέο συμβολικό πρόγραμμα.
Η μνήμη του υπολογιστή δέχεται διευθύνσεις μόνο από τον AR. Επομένως οι έξοδοι του AR θα συνδεθούν στις εισόδους διευθύνσεων της μνήμης. Σύμφωνα με την πρώτη γραμμή του συμβολικού προγράμματος ο PC δίνει το περιεχόμενο του στον AR (ο δείκτης προγράμματος δίνει το περιεχόμενο του στον καταχωρητή διευθύνσεων προκειμένου να αποκτηθεί η πρώτη εντολή. Πολύ λογικό!), επομένως οι έξοδοι του PC συνδέονται στις εισόδους του AR. Στη δεύτερη γραμμή του συμβολικού προγράμματος βλέπουμε ότι ο IR δέχεται δεδομένα από τη μνήμη, επομένως οι έξοδοι δεδομένων της μνήμης οδηγούνται στις εισόδους του IR. Κατά την τρίτη γραμμή γίνεται η αποκωδικοποίηση του opcode από τον αποκωδικοποιητή λειτουργίας (συνδυαστικό κύκλωμα) και η μεταφορά των τεσσάρων χαμηλής τάξης bit από τον IR στον AR. Παρατηρήστε εδώ ότι ο IR είναι ο δεύτερος καταχωρητής που δίνει τα περιεχόμενα του στον AR. Πώς όμως θα καταφέρουμε ένας καταχωρητής θα δέχεται δεδομένα από δύο καταχωρητές υπό διαφορετικές συνθήκες? Μα φυσικά με τη χρήση πολυπλέκτη όπως είχαμε δει και στο σχήμα 2.5 του 2^{ου} κεφαλαίου.
Η τέταρτη μικρολειτουργία και τελευταία του κύκλου ανάκλησης είναι η μεταφορά του operand στον DR, επομένως οι έξοδοι δεδομένων της μνήμης εκτός από τον IR θα οδηγούνται και στον DR (το σωστό κάθε φορά σήμα ελέγχου θα ενεργοποιεί την είσοδο LOAD του καταχωρητή που θέλουμε να δεχτεί τα εκάστοτε δεδομένα της μνήμης).
Οι επόμενες μικρολειτουργίες αποτελούν μέρος του κύκλους εκτέλεσης. Για τις μικρολειτουργίες με συνθήκες D_0T_4 έως D_4T_4 οι συνδέσεις δεν είναι τίποτε άλλο από το κύκλωμα που είδαμε στο υποκεφάλαιο 5.5 όπου αναλύσαμε τη μονάδα επεξεργασίας. Τα περιεχόμενα του DR (ο operand) οδηγούνται στην ALU και μαζί με τα δεδομένα του AC επεξεργάζονται από αυτήν προκειμένου να σχηματιστούν τα τελικά αποτελέσματα των πράξεων, τα οποία και οδηγούνται πίσω στον AC. Για την εντολή WRITE (D_5T_4) απαιτείται η μεταφορά των αποτελεσμάτων των πράξεων από τον AC στη μνήμη. Επομένως οι έξοδοι του AC θα οδηγηθούν στις εισόδους δεδομένων της μνήμης. Στο σχήμα 6.1 μπορείτε να δείτε τις συνδέσεις των κυκλωμάτων του υπολογιστή σύμφωνα με όσα έχουμε πει μέχρι τώρα. Από το σχήμα όπως παρατηρείται λείπουν τόσο η μονάδα ελέγχου (επομένως και τα σήματα ελέγχου στις εισόδους των κυκλωμάτων) όσο και ο κωδικοποιητής της ALU τα οποία θα προσθέσουμε στη συνέχεια.



Σχήμα 6.1 Συνδέσεις Δομικών στοιχείων

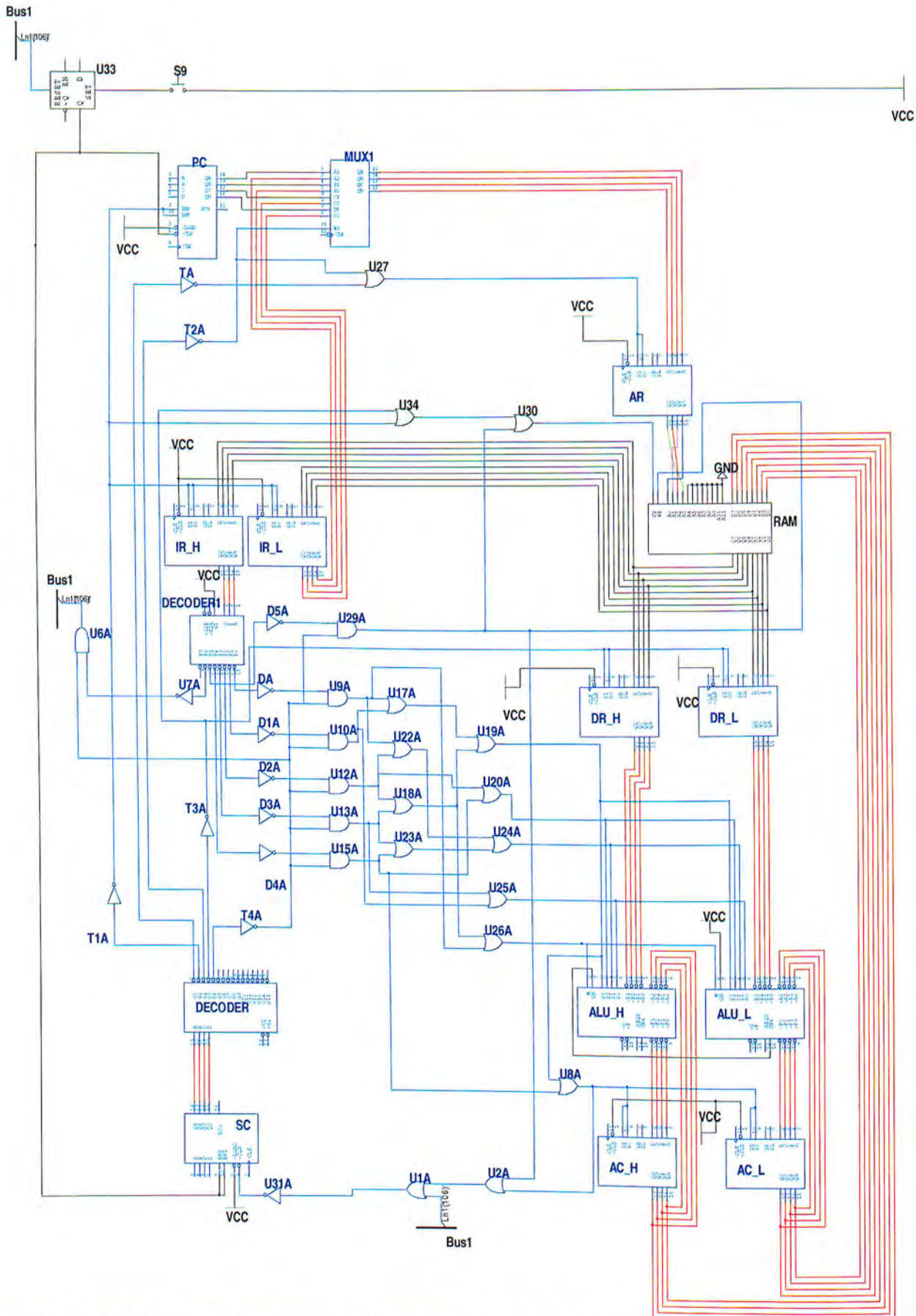
Σημείωση: Στο παραπάνω σχήμα όλους τους ακροδέκτες CLR τους τροφοδοτούμε με 5V (λογικό 1) ώστε να μη μηδενίζεται το περιεχόμενο τους. Μην ξεχνάτε ότι πρόκειται για ενεργά χαμηλές εισόδους, άρα ενεργοποιούνται με λογικό 0.

6.3 Σύνδεση Μονάδας Ελέγχου

Στα σχήματα 5.2 και 5.4 έχουμε δώσει την οργάνωση της μονάδας ελέγχου και ένα παράδειγμα μεταφοράς δεδομένων μεταξύ δύο καταχωρητών με τη χρήση ενός σήματος που δημιουργείτε από τη μονάδα ελέγχου και εφαρμόζεται στις εισόδους επιλογής του εκάστοτε καταχωρητή. Η απόφαση για το ποιο σήμα ελέγχου θα εφαρμοστεί σε ποιον καταχωρητή είναι και πάλι αποτέλεσμα των πληροφοριών που μας δίνει το συμβολικό πρόγραμμα. Ακολουθώντας τον τρόπο σκέψης που ακολουθήσαμε για τη σύνδεση των σημάτων ελέγχου στο σχήμα 5.4 μπορούμε να συμπεράνουμε τα σήματα που θα εφαρμοστούν στις εισόδους επιλογής των κυκλωμάτων του υπολογιστή μας. Διατρέχουμε και πάλι το συμβολικό πρόγραμμα και βλέπουμε ότι στην πρώτη γραμμή πραγματοποιείται η μεταφορά του περιεχομένου του PC στον AR όταν αληθεύει η συνθήκη T_0. Επομένως ο AR μόλις γίνει 1 το σήμα T_0 του αποκωδικοποιητή χρονιστή θα πρέπει να φορτώσει τα bit που "βλέπει" στις εισόδους δεδομένων του. Προκειμένου να συμβεί αυτό θα οδηγήσουμε το σήμα ελέγχου T0 στην είσοδο LOAD του AR. Στη συνέχεια έχουμε τη μεταφορά του περιεχομένου της μνήμης στο IR και την αύξηση του περιεχομένου του PC όταν ενεργοποιηθεί το σήμα ελέγχου T_1. Οδηγούμε όπως καταλαβαίνεται το T_1 στην είσοδο LOAD του IR και στις εισόδους ENP και ENT του PC σύμφωνα με το datasheet του ολοκληρωμένου 74LS163. Δεν πρέπει να ξεχνάμε ότι για να διαβάσουμε το περιεχόμενο μίας θέσης μνήμης θα πρέπει να ενεργοποιήσουμε και την είσοδο επιλογής της μνήμης CS. Άρα το ίδιο σήμα θα πρέπει να εφαρμοστεί και στην είσοδος CS της RAM. Τη χρονική στιγμή T_2 τα τέσσερα χαμηλής τάξης bit του IR φορτώνονται και πάλι στον AR (δεύτερο σήμα ελέγχου που πρέπει να εφαρμοστεί στον AR). Με ποιο τρόπο θα κάνουμε την είσοδο LOAD του AR να ενεργοποιείται όταν είναι αληθές ένα από τα σήματα ελέγχου T_0 και T_2? Μα φυσικά περνώντας τα από μία πύλη OR και συνδέοντας την έξοδο της στην είσοδο LOAD του AR. Επόμενη μικρολειτουργία είναι η μεταφορά του operand από τη μνήμη στον DR. Για να πραγματοποιηθεί αυτό οδηγούμε το σήμα χρονισμού T_3 στην είσοδο CS της RAM και στην είσοδο φόρτωσης του DR.

Συνεχίζουμε με τις μικρολειτουργίες της φάσης εκτέλεσης. Στις πέντε πρώτες συμβολικές εντολές τα αποτελέσματα των διαφόρων πράξεων φορτώνονται στον AC. Επομένως θα πρέπει τα σήματα D_0T_4, D_1T_4, D_2T_4, D_3T_4, D_4T_4 να εφαρμοστούν στην είσοδο LOAD του AC αφού φυσικά τα περάσουμε από τον κατάλληλο αριθμό πυλών OR προκειμένου να σχηματιστεί η συνάρτηση Boole $(D_0+D_1+D_2+D_3+D_4).T_4$. Προκειμένου να αποφύγουμε την επιπλέον χρήση πυλών OR μπορούμε να εκμεταλλευτούμε τις συναρτήσεις Boole που έχουμε φτιάξει στον κωδικοποιητή ελέγχου της ALU. Η συνάρτηση που οδηγούμε στην είσοδο ελέγχου S3 της ALU περιέχει τα περισσότερα σήματα απ όσα μας ενδιαφέρουν $S3 = (D_0 + D_1 + D_2 + D_3).T4$. Περνώντας το παραπάνω σήμα από μία πύλη OR μαζί με το D4T4 έχουμε την τελική συνάρτηση Boole που θα ενεργοποιεί την είσοδο φόρτωσης του AC. Εκτός από τη φόρτωση των αποτελεσμάτων στον AC υπάρχει και η μικρολειτουργία $SC \leftarrow 0$ για το μηδενισμό του περιεχομένου του μετρητή χρονιστή. Προκειμένου να συμβεί αυτό οδηγούμε το σήμα ελέγχου $(D_0+D_1+D_2+D_3+D_4).T_4$ και στην είσοδο μηδενισμού του SC, αφού το περάσουμε από έναν αναστροφέα καθώς πρόκειται για ενεργά χαμηλή είσοδο (Η

μικρολειτουργία SC ← 0 υπάρχει σε όλες τις λειτουργίες της φάσης εκτέλεσης επομένως το τελικό λογικό άθροισμα που θα εφαρμοστεί στην CLR του SC είναι ($D_0 + D_1 + D_2 + D_3 + D_4 + D_5 + D_6$).$T_4$). Για την εντολή WRITE πρέπει να ενεργοποιήσουμε και τις δύο εισόδους ελέγχου της RAM CS και WE καθώς πρόκειται για εντολή εγγραφής στη μνήμη. Καταλαβαίνεται φυσικά ότι προκειμένου να εφαρμόσουμε και αυτό το σήμα ελέγχου στην είσοδο CS θα πρέπει να χρησιμοποιήσουμε πύλες OR ώστε να σχηματιστεί το άθροισμα $T_1+T_3+T_4$. Αυτό μπορεί να γίνει με τη χρήση μία πύλης OR τριών εισόδων, ωστόσο όπως είχαμε σημειώσει και στο πρώτο κεφάλαιο για να πετύχουμε κάτι τέτοιο θα χρησιμοποιούμε δύο πύλες OR δύο εισόδων. Και πάλι με τη χρήση OR προσθέτουμε και αυτόν τον όρο στην τελική συνάρτηση που θα μηδενίζει το περιεχόμενο του SC. Τέλος με την εντολή HLT θα σταματάει η λειτουργία του υπολογιστή, η οποία σηματοδοτείται μέσω ενός latch. Όταν κάνουμε 1 την κατάσταση του μανδαλωτή θα ξεκινά η λειτουργία του υπολογιστή, ενώ όταν θα μηδενίζουμε το μανδαλωτή θα σταματάει η λειτουργία (θα αναφερθούμε περαιτέρω στα επόμενα). Άρα το σήμα ελέγχου D_6T_4 θα οδηγηθεί στην είσοδο RESET του μανδαλωτή. Παρακάτω δίνεται μία συνολική εικόνα του κυκλώματος του υπολογιστή όπου έχουμε συμπεριλάβει και τα σήματα ελέγχου αλλά και τον κωδικοποιητή της ALU που είδαμε στο προηγούμενο κεφάλαιο. Για να έχετε μία πιο αναλυτική εικόνα του κυκλώματος μπορείτε να ανατρέξετε στο cd-ROM που συνοδεύει το εγχειρίδιο και να δείτε το κύκλωμα μέσα από το πρόγραμμα Multisim.

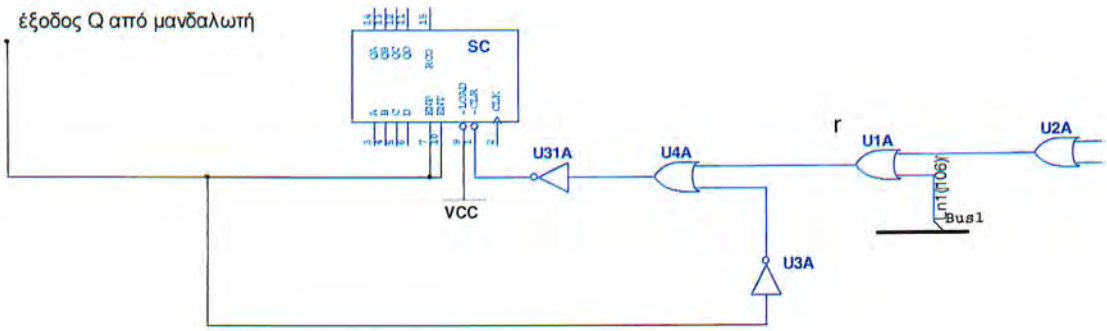


Σχήμα 6.2 Κύκλωμα υπολογιστή με μονάδα ελέγχου και κωδικοποιητή ελέγχου της ALU

Σημείωση: Προκειμένου να περιπλέξουμε όσο γίνεται λιγότερο το κύκλωμα έχουμε χρησιμοποιήσει τη δυνατότητα που μας δίνει το Multisim για την μεταφορά δεδομένων μέσω bus. Για παράδειγμα η έξοδος της πύλης AND U6A (σήμα D_6T_4, εντολή HLT) αντί να τη συνδέσουμε στην είσοδο RESET του μανδαλωτή, την οδηγούμε στο bus 1 και συνδέουμε και την είσοδο RESET σε ένα άλλο "κομμάτι" του bus 1. Για περισσότερες πληροφορίες σχετικά με το bus μπορείτε να ανατρέξετε στον Multisim guide που βρίσκεται στο cd-ROM. Επίσης δεν έχουμε συνδέσει το καθολικό ρολόι του συστήματος και πάλι για να αποφύγουμε ένα ακόμα πιο πολύπλοκο κύκλωμα, ωστόσο μας είναι πλέον γνωστό ότι σε όλες τις εισόδους χρονισμού των συνδυαστικών κυκλωμάτων συνδέουμε το σήμα ρολογιού.

6.4 Έναρξη και Παύση λειτουργίας

Η έναρξη της λειτουργίας του υπολογιστή θα γίνεται από εμάς μέσω ενός button. Το button είναι συνδεδεμένο στην είσοδο SET ενός μανδαλωτή και με το πάτημα του τοποθετείται ο μανδαλωτής σε κατάσταση θέσης (μπορείτε να το δείτε στο σχήμα 6.2). Στο σχήμα 6.2 έχουμε συνδέσει την έξοδο Q στις εισόδους ENP και ENT ώστε να μετρά προς τα πάνω. Ωστόσο όπως θα δούμε παρακάτω αυτό δεν είναι αρκετό για να έχουμε τη λειτουργία που επιθυμούμε, καθώς η λειτουργία του μετρητή εξαρτάται άμεσα από την είσοδο CLR. Εδώ θα πρέπει να θυμηθούμε ότι η λειτουργία του υπολογιστή εξαρτάται άμεσα από τους παλμούς χρονισμού. Ο μετρητής χρονιστή δέχεται τους παλμούς του ρολογιού, αυξάνει το περιεχόμενο του κατά 1 για κάθε παλμό και μέσω του αποκωδικοποιητή δημιουργεί τα σήματα ελέγχου T_0, T_1, T_2... τα οποία είναι οι συνθήκες για την ενεργοποίηση των εισόδων ελέγχου των ολοκληρωμένων κυκλωμάτων. Η ενεργοποίηση αυτή οδηγεί στην πραγματοποίηση των μικρολειτουργιών. Αν με κάποιο τρόπο δεν επιτρέπαμε στον SC να μετρά τους παλμούς που δέχεται ή για να είμαστε πιο ακριβής αν ελέγχαμε το πότε θα μετρά τους παλμούς του ρολογιού και πότε όχι, θα καταφέρναμε να ενεργοποιούμε και να απενεργοποιούμε όποτε θέλουμε τον υπολογιστή. Το ολοκληρωμένο 74LS163 (SC) είναι ένας δυαδικός μετρητής με παράλληλη φόρτωση. Οδηγώντας τις εισόδους ENP και ENT σε λογικό 1 ενεργοποιούμε τη λειτουργία μέτρησης προς τα πάνω του SC με την έλευση κάθε νέου παλμού του ρολογιού που δέχεται στην είσοδο χρονισμού του. Για να πραγματοποιείται όμως αυτή η λειτουργία θα πρέπει απαραίτητα η είσοδος καθαρισμού να βρίσκεται σε λογικό 1 (ενεργά χαμηλή), καθώς όσο η CLR είναι σε λογικό 0 ο μετρητής μηδενίζει το περιεχόμενο του σε κάθε παλμό του ρολογιού. Συνδέοντας λοιπόν την έξοδο του μανδαλωτή Q με την είσοδο CLR του SC με τέτοιο τρόπο ώστε όταν η έξοδος του πρώτου είναι 0, η είσοδος CLR του δεύτερου να είναι 0 και για Q = 1 το CLR = 1, μπορούμε να επιτύχουμε το σκοπό μας. ΠΡΟΣΟΧΗ όμως, καθώς στην είσοδο CLR έχουμε ήδη συνδέσει το σήμα ελέγχου $(D_0 + D_1 + D_2 + D_3 + D_4 + D_5 + D_6).T_4$ (ας το ονομάσουμε r, ώστε να μη το "κουβαλάμε" όποτε θέλουμε να αναφερθούμε σε αυτό), αφού το περάσαμε από τον αναστροφέα που απαιτείται. Άρα χρειαζόμαστε μία ακόμα πύλη OR. Στο σχήμα 6.3 έχουμε απομονώσει τον SC μαζί με το συνδυαστικό κύκλωμα που εφαρμόζουμε στην είσοδο CLR.



Σχήμα 6.3 Συνδυαστικό κύκλωμα Καθαρισμού του SC

Θέλουμε όταν τα r και Q είναι 0 ο υπολογιστής να είναι ανενεργός, άρα η είσοδος CLR να οδηγείται στο 0. Όταν Q = 0 η έξοδος του αναστροφέα U3A είναι 1 άρα και η έξοδος της πύλης OR U4A είναι 1 που οδηγεί την είσοδο CLR σε λογικό 0 αφού αναστρέφεται και πάλι από την πύλη U31A. Πράγματι λοιπόν ο υπολογιστής είναι ανενεργός. Όταν ενεργοποιήσουμε μέσω του button έναρξης τον υπολογιστή, έχουμε Q = 1, U3A = 0, U4A = 0 (εφόσον δεν έχει γίνει ακόμα r = 1), U31A = 1 και επομένως λειτουργία του υπολογιστή. Όταν ενεργοποιηθεί η συνάρτηση r από τη μονάδα ελέγχου τότε η εξής ακολουθία καταστάσεων θα οδηγήσει στο μηδενισμό του CLR: U1A = 1, U4A = 1, U31A = 0 και τελικά μηδενισμός του CLR. Εφόσον όμως δεν έχει μηδενιστεί και το περιεχόμενο του μανδαλωτή τότε μόλις το r επιστρέψει στο 0, καθώς δε θα ισχύει πια $T_4 = 1$, (αφού ο μηδενισμός του SC θα ενεργοποιήσει το T_0) θα συνεχίσει ο SC να μετράει παλμούς από την αρχή. Μέσω της εντολής HLT μηδενίζεται τόσο το περιεχόμενο του SC όσο και του μανδαλωτή, επομένως ο SC αφού μηδενιστεί δε του επιτρέπουμε να μετρήσει τους παλμούς του ρολογιού καθώς έχουμε θέσει την είσοδο CLR σε λογικό 0. Στον πίνακα 6.1 έχουμε συνοψίσει τις εξόδους των πυλών και τις μικρολειτουργίες που πραγματοποιούνται με βάση τις τιμές των εισόδων Q και r.

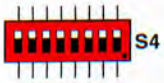
Q	r	U3A	U4A	U31A	Λειτουργία
0	0	1	1	0	SC = 0, υπολογιστής ανενεργός ή απενεργοποίηση του
0	1	1	1	0	Δε μπορεί να υπάρχει ο συγκεκριμένος συνδυασμός
1	0	0	0	1	Έναρξη μέτρησης παλμών
1	1	0	1	0	SC ← 0, υπολογιστής ανενεργός

Πίνακας 6.1 Έξοδοι πυλών για την ενεργοποίηση του CLR

Ο συνδυασμός τιμών Q = 0 και r = 1 δε μπορεί να υπάρξει καθώς εφόσον δεν έχουμε ενεργοποιημένο τον υπολογιστή δεν είναι δυνατόν να έχουμε $T_4 = 1$ που απαιτείται για να αληθεύει το r = 1. Τις εισόδους ENP και ENT θα μπορούσαμε να τις έχουμε μόνιμα στο 1 αντί να ενεργοποιούνται και αυτές μέσω της Q.

6.5 Εισαγωγή Προγράμματος

Έχουμε αναφερθεί αρκετές φορές στο πρόγραμμα που θα τρέχει ο υπολογιστής, στις εντολές από τις οποίες αυτό αποτελείται, στον τρόπο που ο υπολογιστής τις ανακαλεί από τη μνήμη του και τις εκτελεί, όμως ακόμα δεν έχουμε δει κάποιο τρόπο με τον οποίο θα το εισάγουμε στη μνήμη του υπολογιστή. Ως γνωστών το πρόγραμμα αποτελείται από εντολές και οι εντολές από τον operand και τον opcode, οι οποίοι με τη σειρά τους είναι ακολουθίες από bit. Στόχος μας λοιπόν είναι να αποθηκεύσουμε στις σωστές κάθε φορά θέσεις μνήμης τις ακολουθίες από bit που ικανοποιούν την εφαρμογή μας. Αυτό θα γίνεται με τη χρήση διακοπτών οι οποίοι στο Multisim έχουν το παρακάτω σχήμα

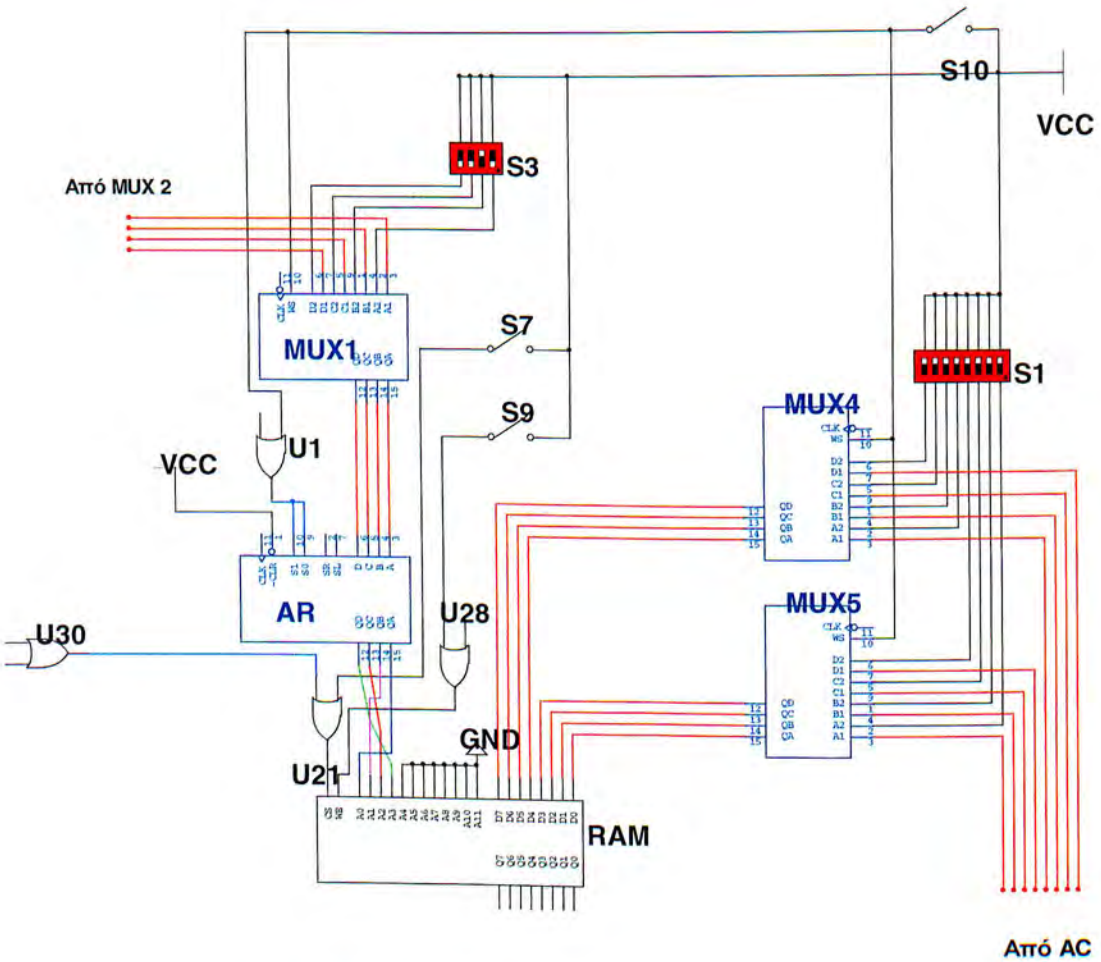


Από τη μία πλευρά εφαρμόζουμε τάση 5 Volt και από την άλλη συνδέουμε τις εισόδους του κυκλώματος στο οποίο θέλουμε να εισάγουμε δεδομένα. Η μαύρη τελεία κάτω δεξιά μας επιτρέπει να ξεχωρίζουμε το ON και OFF. Όταν το άσπρο τετράγωνο βρίσκεται στην πλευρά που βρίσκεται και η τελεία ο διακόπτης είναι κλειστός και το ρεύμα περνάει από την έξοδο του συγκεκριμένου διακόπτη. Το κύκλωμα S2 αποτελείται από τέσσερις διακόπτες ενώ το S4 από οχτώ. Δεν είναι τίποτα άλλο από τους διακόπτες που είχαμε χρησιμοποιήσει στα πρώτα κεφάλαια ως εισόδους με τη διαφορά ότι είναι περισσότεροι διακόπτες σε μικρότερα κυκλώματα.

Στο 4° κεφάλαιο έχουμε δει τη δομή των εντολών του υπολογιστή και με βάση αυτή θα δημιουργούμε τις δικές μας εντολές, ανάλογα με τις απαιτήσεις μας και φυσικά με τις λειτουργίες που μπορεί να εκτελέσει ο υπολογιστής. Οι εντολές είναι αποθηκευμένες η μία μετά την άλλη στη μνήμη RAM. Θα εισάγουμε τις εντολές πραγματοποιώντας λειτουργία γραφής στην σωστή θέση μνήμης. Επομένως θα πρέπει να έχουμε τη δυνατότητα να επιλέγουμε τα δεδομένα που θέλουμε να εισάγουμε στη μνήμη, τη διεύθυνση στην οποία θα τα αποθηκεύουμε και να ελέγχουμε τις εισόδους CS και WE της RAM. Η μνήμη επικοινωνεί με τα υπόλοιπα κυκλώματα μόνο μέσω του AR (μπορούμε να εισάγουμε τη διεύθυνση της μνήμης στην οποία θέλουμε να αποθηκεύσουμε δεδομένα και κατευθείαν στις εισόδους διευθύνσεων, ωστόσο προτιμήσαμε να το κάνουμε μέσω του AR). Προκειμένου να εισάγουμε, λοιπόν, δεδομένα στον AR θα συνδέσουμε τις εξόδους των διακοπτών στις εισόδους δεδομένων του AR. Παρατηρώντας, όμως, το υπάρχον κύκλωμα του υπολογιστή βλέπουμε ότι ο AR δέχεται δεδομένα, μέσω ενός πολυπλέκτη, από τους PC και IR. Τοποθετώντας έναν δεύτερο πολυπλέκτη MUX2 μεταξύ του MUX1 και του AR, θα μπορούμε με έναν διακόπτη να παρεμβαίνουμε χειροκίνητα στην είσοδο επιλογής του MUX2 και να εισάγουμε τη διεύθυνση που θέλουμε στον AR. Στη συνέχεια θα τοποθετούμε στις εισόδους δεδομένων της RAM την εντολή που θέλουμε και ενεργοποιώντας και πάλι χειροκίνητα, με τη χρήση διακοπτών, τις εισόδους CS και WE θα επιτυγχάνεται η αποθήκευση της εντολής σε μία θέση της μνήμης. Απαιτείται φυσικά μία πύλη OR στην είσοδο LOAD του AR, ώστε να ελέγχεται η λειτουργία του είτε από τη μονάδα ελέγχου είτε από το διακόπτη εισαγωγής προγράμματος.

Σύμφωνα με το σχήμα 6.2 η RAM δέχεται δεδομένα από τον AC προκειμένου να αποθηκεύει τα αποτελέσματα των πράξεων. Απαιτείται, λοιπόν, και πάλι χρήση πολυπλέκτη και για την ακρίβεια 2 πολυπλεκτών 8-σε-4 αφού οι εντολές είναι μήκους 8-bit (ένας για τα τέσσερα χαμηλής τάξης bit και ένας για τα τέσσερα υψηλής τάξης). Οι πολυπλέκτες δέχονται δεδομένα από τον AC και από εμάς και μέσω της εισόδου επιλογής ελέγχουμε από πού θα δέχεται δεδομένα η μνήμη. Προκειμένου να μη χρησιμοποιούμε

δεύτερο διακόπτη θα συνδέσουμε τις εισόδους επιλογής και αυτών των πολυπλεκτών στο διακόπτη που χρησιμοποιούμε για τον έλεγχο του MUX2 (ο διακόπτης αυτός θα είναι το ON της λειτουργίας εισαγωγής δεδομένων). Όταν αυτός ο διακόπτης είναι κλειστός μπορούμε να εισάγουμε δεδομένα στη μνήμη, ενώ όταν είναι ανοιχτός μπορούμε να ενεργοποιήσουμε τον υπολογιστή. Τέλος εφόσον απαιτείται χειροκίνητος έλεγχος των εισόδων επιλογής της μνήμης απαιτούνται δύο επιπλέον πύλες OR, ώστε οι είσοδοι CS και WE να ενεργοποιούνται είτε από τη μονάδα ελέγχου είτε από εμάς μέσω των διακοπτών. Στο σχήμα 6.4 μπορείτε να δείτε τα το κομμάτι του κυκλώματος που υλοποιεί την εισαγωγή προγράμματος στην μνήμη RAM.



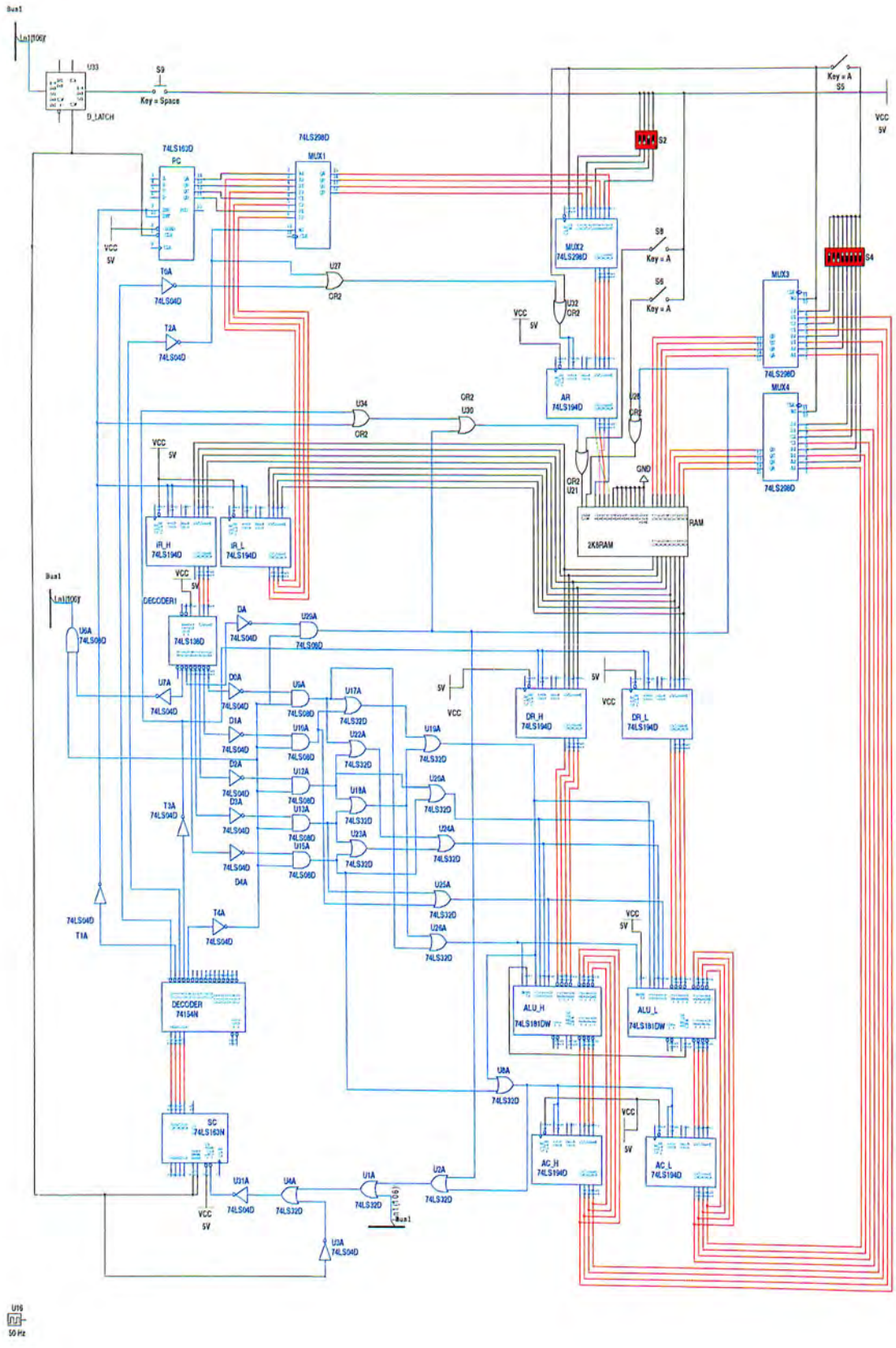
6.4 Κύκλωμα εισαγωγής προγράμματος

Και μετά από το κύκλωμα που χρησιμοποιούμε για να εισάγουμε τα προγράμματα μας στον υπολογιστή, σειρά έχει το ίδιο το πρόγραμμα. Σύμφωνα με τη δομή των εντολών που έχουμε μάθει ως τώρα ήρθε η ώρα να διαμορφώσουμε τις εντολές που ικανοποιούν τις πράξεις που επιθυμούμε. Η κάθε εντολή αποτελείται από τον opcode που αντιστοιχεί στην εκάστοτε λειτουργία και από τη διεύθυνσης μνήμης στην οποία βρίσκεται ο operand (ή τη θέση μνήμης στην οποία θέλουμε να αποθηκευτεί το αποτέλεσμα αν πρόκειται για τη

λειτουργία WRITE). Αφού εισάγουμε την εντολή θα πρέπει να τοποθετήσουμε στη σωστή θέση μνήμης τον operand. Η σύμβαση που υιοθετήσαμε είναι να αποθηκεύουμε τις εντολές στο επάνω μέρος της μνήμης και τους τελεστές στο κάτω (ανατρέξτε στο 4° κεφάλαιο). Ας εργαστούμε με ένα παράδειγμα. Υποθέτουμε ότι θέλουμε να εισάγουμε δύο αριθμούς στον υπολογιστή και αυτός να υπολογίζει και να αποθηκεύει σε συγκεκριμένες θέσεις μνήμης το αποτέλεσμα της αριθμητικής πρόσθεσης τους. Η οργάνωση συσσωρευτεί που έχουμε επιλέξει για τη μονάδα επεξεργασίας απαιτεί να φέρουμε τον πρώτο τελεστή στον AC και στη συνέχεια να προσθέσουμε σε αυτόν τον δεύτερο τελεστή. Επομένως η πρώτη εντολή θα είναι μία πράξη LOAD στον AC. Ο opcode για τη φόρτωση ενός τελεστή στον AC είναι 000. Επομένως τα bit στις θέσεις 4, 5, 6 έχουν περιεχόμενο 0. Τον τελεστή θα τον αποθηκεύσουμε στην τελευταία θέση της μνήμης η οποία έχει διεύθυνση 1111. Το ΠΣΨ δε χρησιμοποιείται επομένως έχει πάντοτε τιμή 0. Άρα η πρώτη εντολή του προγράμματος μας έχει την παρακάτω μορφή σε γλώσσα μηχανής 00001111 και θα τοποθετηθεί στην πρώτη θέση της μνήμης 0000. Στη συνέχεια θέλουμε στο νέο περιεχόμενο του AC να προσθέσουμε τον επόμενο τελεστή. Η αριθμητική πρόσθεση έχει opcode 001 και η θέση στην οποία θα αποθηκεύσουμε τον τελεστή είναι η 1110. Η δεύτερη εντολή που θα εισάγουμε στη θέση 0001 είναι η 00011110. Στη συνέχεια θα αποθηκεύσουμε το αποτέλεσμα της πράξης στη διεύθυνση μνήμης 1101 μέσω μίας εντολής WRITE. Εύκολα μπορείτε να συμπεράνετε ότι η εντολή αποτελείται από την παρακάτω ακολουθία bit 01011101 και φυσικά την τοποθετούμε στη θέση με διεύθυνση 0010. Τέλος, μία εντολή HLT στην επομένη θέση μνήμης σταματά τη λειτουργία του υπολογιστή. Τελεστής για αυτή την εντολή δε χρειάζεται επομένως τα τέσσερα λιγότερο σημαντικά ψηφία είναι αδιάφορα (θα τοποθετούμε το 0000). Στο σχήμα 6.5 φαίνεται οι μνήμη με αποθηκευμένο το παραπάνω πρόγραμμα στις αντίστοιχες διευθύνσεις. Στο σχήμα 6.6 υπάρχει το τελικό κύκλωμα του υπολογιστή χωρίς και πάλι να έχουμε συνδέσει το κοινό ρολόι στις εισόδους χρονισμού.

Πρόγραμμα	Διευθύνσεις
00001111	0000
00011110	0001
01011101	0010
01100000	0011
	0100
	0101
	0110
	0111
	1000
	1001
	1010
	1011
	1100
	1101
00001100	1110
10100011	1111

Σχήμα 6.5 Αποθηκευμένο πρόγραμμα στη μνήμη.



Σχήμα 6.6 Τελικό Κύκλωμα Υπολογιστή

1. Mano, M.M., M. D. Ciletti. 2007. Digital Design 4th ed. Pearson Education.
2. Mano, M.M. 1993. Computer System Architecture 3rd ed. Prentice Hall.
3. Καραϊσκος, Χ. 1984. Ψηφιακά Κυκλώματα ΙΙ