ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΥΠΟΛΟΓΙΣΤΩΝ

Περίγραμμα ύλης:

1. Εγκυκλοπαιδική εισαγωγή στη σύγχρονη τεχνολογία υλοποίησης των υπολογιστών.
2. Η γλώσσα μηχανής και η γλώσσα assembly, σαν το μοντέλο αφαίρεσης που το hardware παρουσιάζει προς το software.
3. Παράδειγμα της γλώσσας μηχανής μιας αρχιτεκτονικής RISC.
4. Υλοποίηση των υπολογιστών χρησιμοποιώντας καταχωρητές, πολυπλέκτες, αποκωδικοποιητές, ALU's, PLA's, RAM's, ROM's, κλπ.
5. Σχεδίαση του datapath.
6. Σχεδίαση της μονάδας ελέγχου.
7. Εικονική μνήμη.
8. Ολίγα περί της επίδοσης (ταχύτητας) των υπολογιστών.

Κύρια Βιβλιογραφία:

D. Patterson, J. Hennessy:
"Computer Organization & Design: the Hardware/Software Interface",
Morgan Kaufmann Publishers, 1994, ISBN: 1-55860-428-6

Αναφορές;

D. Patterson, J. Hennessy: "Computer Organization & Design: the Hardware/Software Interface",
Πανεπιστήμιο Κρήτης http://www.csd.uoc.gr/~hy225/
Πανεπιστήμιο Κύπρου: http://www.cs.ucy.ac.cy

Περιεχόμενα

1.Υπολογιστής : Αφαιρετικότητα και Τεχνολογία

1.1 Εισαγωγή

Ο κόσμος των υπολογιστών είναι δυναμικός, πολύ ενθουσιώδης, εξελίσσεται ραγδαία και πολλές φορές απρόβλεπτα. Η βιομηχανία των υπολογιστών καταλαμβάνει το 5-10% του ακαθάριστου εθνικού προϊόντος στις ΗΠΑ. Αυτή η ραγδαία εξέλιξη στη τεχνολογία των υπολογιστών αρχίζει από το 1940 και συνεχίζεται μέχρι και σήμερα. Αν υπήρχε παρόμοια ανάπτυξη και στη βιομηχανία των μεταφορών τότε το ταξίδι από τη μια άκρη της Αμερικής στην άλλη θα έπαιρνε 30 δευτερόλεπτα και θα στοίχιζε μόνο 50 σεντς.

Η επανάσταση στη τεχνολογία των υπολογιστών αποτελεί την τρίτη επανάσταση στο πολιτισμό μας. Προηγούνται η αγροτική και η βιομηχανική επανάσταση και ακολουθεί η πιο πρόσφατη η επανάσταση της πληροφορίας.

Η επανάσταση της πληροφορίας έχει επηρεάσει όλες τις επιστήμες. Οι επιστήμονες της πληροφορικής πέρα από τις εφαρμογές, εργάζονται στενά με θεωρητικούς και πειραματικούς επιστήμονες για την ανάπτυξη νέων κατευθύνσεων στη βιολογία, χημεία, φυσική κ.α.

Κάθε φορά που το κόστος των υπολογιστών μειώνεται κατά δέκα φορές, οι ευκαιρίες για νέες εφαρμογές πολλαπλασιάζονται. Εφαρμογές που αποτελούσαν επιστημονική φαντασία πρίν μερικές δεκαετίες, είναι σήμερα πραγματικότητα, π.χ.:

- Αυτόματες τραπεζικές μηχανές (Automatic Teller Machines - ATM)
- Υπολογιστές στα αυτοκίνητα (Computers in Automobiles)
- Φορητοί υπολογιστές (Laptop Computers)
- Χαρτογράφηση του ανθρώπινου γονότυπου (Human Genome Project - DNA)
- Ηλεκτρονικές βιβλιοθήκες (Electronic Libraries) κ.α.

Οι επιστήμονες της πληροφορικής για να είναι σε θέση να φτιάχνουν πιο αποδοτικά προγράμματα και κατά συνέπεια πετυχημένα λογισμικά πρέπει να έχουν καλή γνώση της οργάνωσης των υπολογιστών.

Στόχος αυτού του μαθήματος και του βιβλίου “Computer Organization & Design. The Hardware/Software Interface” των D.Patterson και J.Hennessy, είναι να παρουσιαστούν και να κατανοηθούν πλήρως οι έννοιες του υλικού και του λογισμικού καθώς και η εσωτερική δομή του ηλεκτρονικού υπολογιστή. Θα καλυφθούν οι έννοιες του προγραμματισμού του υπολογιστή σε επίπεδο γλώσσας μηχανής και σε συμβολική γλώσσα, η εσωτερική οργάνωση του υπολογιστή, πως επηρεάζεται η απόδοση των προγραμμάτων που γράφουμε και τέλος θα τεθούν τα βασικά στοιχεία της σχεδίασης ενός υπολογιστή.

Στη πρώτη ενότητα θα τεθούν οι αρχές για το υπόλοιπο μάθημα. Θα εισαχθούν οι βασικές ιδέες και ορισμοί, τα βασικά στοιχεία του υπολογιστή - το υλικό και το λογισμικό, τα ολοκληρωμένα κυκλώματα και η τεχνολογία.

1.2 Κάτω από το πρόγραμμα σου

Για να μπορέσει κάποιος να επικοινωνήσει απευθείας με μια ηλεκτρονική μηχανή πρέπει να στείλει ηλεκτρονικά σήματα, τα πιο απλά ηλεκτρονικά σήματα που καταλαβαίνει η μηχανή είναι το On και το Off. Άρα το αλφάβητο της μηχανής αποτελείται από δύο γράμματα που συμβολίζονται με 1 και 0 και αντιπροσωπεύουν το On και το Off, αντίστοιχα. Αυτά τα δύο σύμβολα ονομάζονται δυαδικά ψηφία.

Οι πρώτοι προγραμματιστές επικοινωνούσαν με τους υπολογιστές στο δυαδικό σύστημα. Αυτό όμως ήταν πολύ δύσκολο και χρονοβόρο.

Έτσι σε μια προσπάθεια απλοποίησης της επικοινωνίας με τον υπολογιστή, οδηγηθήκαμε στη δημιουργία της συμβολικής γλώσσας (Assembly language). Τα προγράμματα γράφονταν σε συμβολική γλώσσα και μεταφράζονταν σε γλώσσα μηχανής από το συμβολομεταφραστή (Assembler).

Για παράδειγμα η εντολή ,

ADD A , B σε συμβολική γλώσσα συμβολομεταφράζεται σε :
1000 1110 1000 σε γλώσσα μηχανής

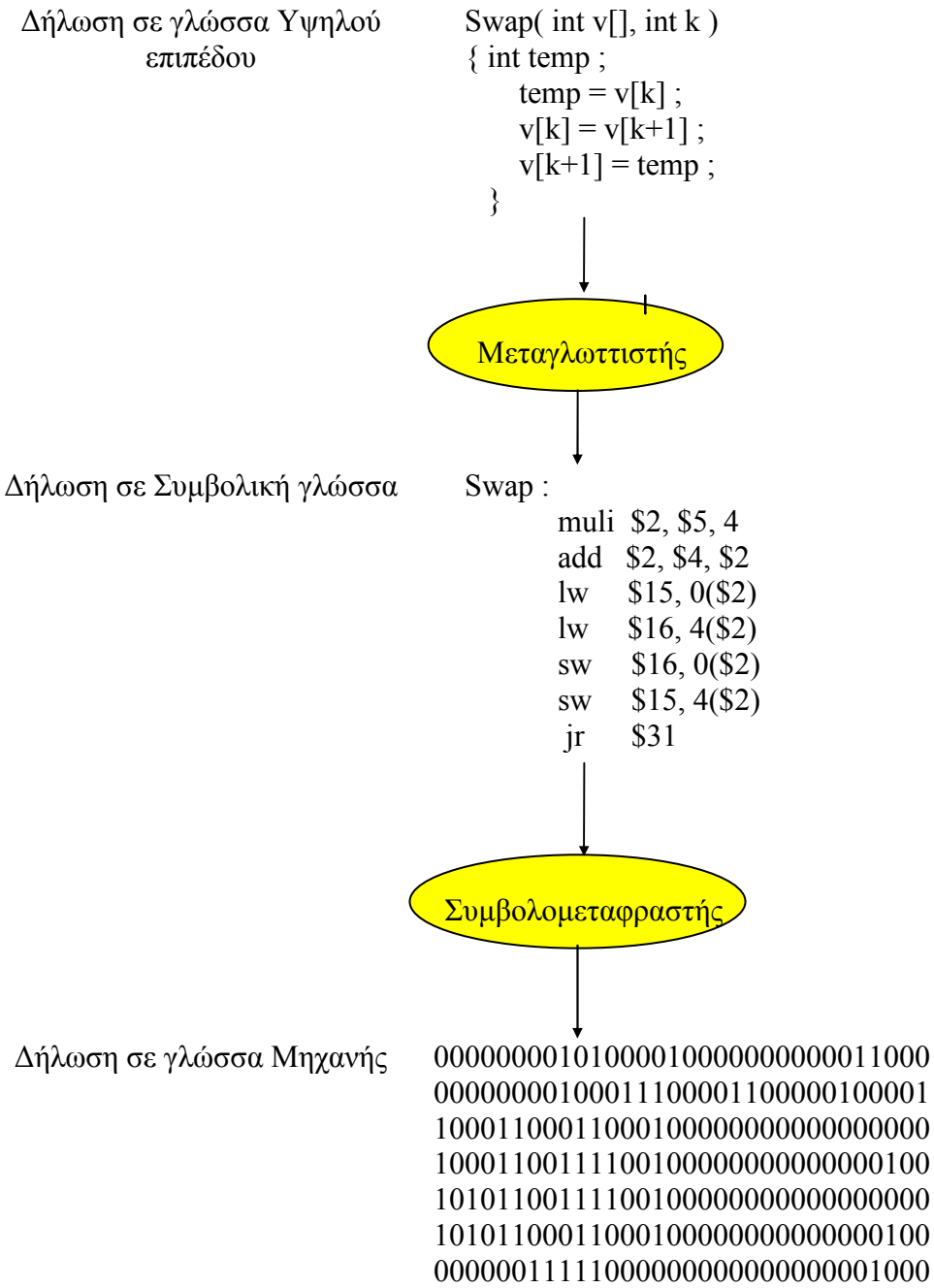
Αυτή η εντολή λεει στον υπολογιστή να προσθέσει δύο αριθμούς.

Η συμβολική γλώσσα απαιτεί από τον προγραμματιστή να γραφεί μια γραμμή για κάθε εντολή (Instruction) που πρέπει να ακολουθεί η μηχανή, δηλαδή σπρώχνει τον προγραμματιστή να σκέφτεται όπως τη μηχανή. Αυτός ο τρόπος σκέψης είναι πέραν από τον συνηθισμένο τρόπο σκέψης οποιουδήποτε επιστήμονα.

Τίθεται τότε το ακόλουθο ερώτημα : ” Μπορούμε να γράψουμε προγράμματα σε μια πιο ψηλού επιπέδου γλώσσα, η οποία θα είναι πιο κατανοητή και τα προγράμματα αυτά να μεταφράζονται σε συμβολική γλώσσα ; “

Η απάντηση είναι Ναι! Τα προγράμματα γράφονται σε μια ψηλού επιπέδου γλώσσα (high-lenel programming language) και μεταφράζονται από το μεταγλωττιστή (compiler) σε μια χαμηλού επιπέδου γλώσσα.

Το Σχήμα 1.1 δίνει τις σχέσεις μεταξύ των προγραμμάτων και των γλωσσών.



Σχήμα 1.1: Ένα πρόγραμμα στη C μεταγλωττίζεται σε χαμηλού επιπέδου γλώσσα (assembly) και ακολούθως συμβολομεταφράζεται σε δυαδική γλώσσα μηχανής. Κάθε γραμμή στη συμβολική γλώσσα αντιστοιχεί σε κάθε γραμμή στη γλώσσα μηχανής.

Τα βασικά πλεονεκτήματα των γλωσσών υψηλού επιπέδου σε σχέση με τις χαμηλού επιπέδου γλώσσες είναι :

- Επιτρέπουν στο προγραμματιστή να σκέφτεται σε μια πιο φυσική γλώσσα χρησιμοποιώντας αγγλικές λέξεις και αλγεβρικά σύμβολα.

- Επιτρέπεται ο σχεδιασμός γλωσσών ανάλογα με τη χρήση τους, π.χ. Fortran για επιστημονικά προγράμματα, Cobol για οικονομικά προγράμματα, Lisp για την επεξεργασία συμβόλων κ.α.
- Υπάρχει μεγαλύτερη αποδοτικότητα στη γραφή προγραμμάτων. Χρειάζεται λιγότερος χρόνος για την ανάπτυξη προγραμμάτων όταν γράφονται σε γλώσσα που χρειάζεται λιγότερες γραμμές για την απόδοση κάποιου αλγορίθμου.
- Τα προγράμματα μπορούν να τρέχουν ανεξάρτητα από τον υπολογιστή στον οποίο έχουν αναπτυχθεί. Αυτό είναι δυνατό δεδομένου ότι οι μεταγλωττιστές και οι συμβολομεταφραστές μεταφράζουν υψηλού επιπέδου γλώσσες σε δυαδικές εντολές οποιασδήποτε μηχανής.

Τα πιο πάνω πλεονεκτήματα είναι τόσο σημαντικά ώστε σήμερα είναι περιορισμένος ο αριθμός των προγραμμάτων που γράφονται εξ' ολοκλήρου σε συμβολική γλώσσα.

Το λογισμικό χωρίζεται σε κατηγορίες ανάλογα με τη χρήση του. Το λογισμικό το οποίο προσφέρει υπηρεσίες που χρησιμοποιούνται ευρέως ονομάζεται λογισμικό του συστήματος - Systems software, όπως π.χ. λειτουργικά συστήματα, μεταγλωττιστές, συμβολομεταφραστές κ.α. Ενώ το λογισμικό που απευθύνεται στις ανάγκες των χρηστών ονομάζεται λογισμικό εφαρμογών - applications software, όπως π.χ. λογιστικά προγράμματα, συντάκτες κειμένου κ.α.

Το Σχήμα 1.2 απεικονίζει το κλασσικό διάγραμμα στην ιεραρχία επιπέδων από το λογισμικό στο υλικό.

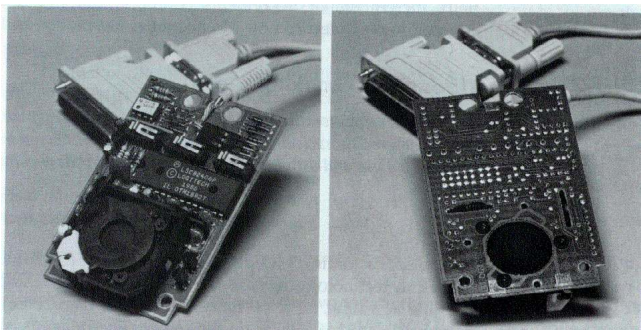
Σχήμα 1.2: Μια απλή όψη του υλικού και του λογισμικού σε ιεραρχικά επίπεδα.

Στη πραγματικότητα το λογισμικό δεν αποτελείται από μονολιθικά επίπεδα, αλλά από προγράμματα που κτίζουν το ένα πάνω στο άλλο.

1.3 Κάτω από τα καλύμματα

1.3.1 Ανατομία του ποντικιού :

Η ιδέα του ποντικιού δημιουργήθηκε πριν 30 περίπου χρόνια. Από μηχανικής άποψης, ένα ποντίκι αποτελείται από μια μεγάλη μπάλά, η οποία επικοινωνεί με ένα ζεύγος τροχών (wheels),ο ένας τροχός είναι τοποθετημένος στο X-άξονα και ο άλλος στο Y-άξονα.



Σχήμα 1.3: Το εσωτερικό του ποντικιού. (Patterson and Hennessy, 1998, σελ. 11).

1.3.2 Δια μέσου του γυαλιού που βλέπουμε - οθόνη :

Η οθόνη βασίζεται στη τεχνολογία της τηλεόρασης και αποτελεί μια από τις πιο ενδιαφέρον συσκευές εισόδου – εξόδου. Η ακτίνα ηλεκτρονίων της καθοδικής λυχνίας (raster cathode ray tube - CRT) σαρώνει την επιφάνεια της οθόνης από τα αριστερά προς τα δεξιά και από πάνω προς τα κάτω, μια γραμμή κάθε φορά, 30 ως 60 φορές κάθε δευτερόλεπτο. Η εικόνα αποτελείται από ένα πίνακα από στοιχεία της εικόνας ή pixels τα οποία μπορούν να αντιπροσωπευθούν από ένα πίνακα από bits, που ονομάζεται bit map.

Στις μαυρόασπρες οθόνες χρειάζεται 1 bit για κάθε pixel (π.χ. 1 = άσπρο, 0 = μαύρο), ενώ στις έγχρωμες χρειάζονται 8 bits για το κόκκινο, 8 bits για το πράσινο και 8 bits για το μπλε. Δηλ. Έχουμε 2^{24} διαφορετικά χρώματα.

1.3.3 Ανοίγοντας το κιβώτιο - CPU :

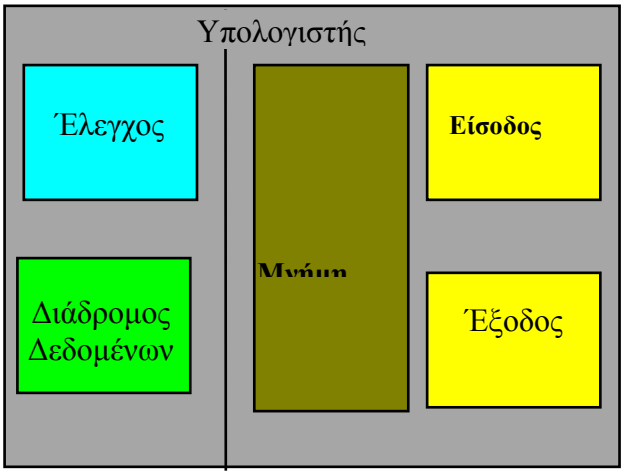
Όταν ανοίξουμε το κουτί θα αντικρίσουμε ένα ενδιαφέρον board (πλακέτα), το οποίο αποτελείται από 3 κομμάτια : το κομμάτι που ενώνεται με τις συσκευές E/E, τη μνήμη και τον επεξεργαστή. Η μνήμη είναι ο χώρος που αποθηκεύονται τα προγράμματα όταν τρέχουν καθώς και τα αντίστοιχα δεδομένα τους. Ο επεξεργαστής αποτελείται από δύο κομμάτια, τον διάδρομο δεδομένων (databath) και την μονάδα ελέγχου (control). Ο διάδρομος δεδομένων εκτελεί τις αριθμητικές λειτουργίες και ο έλεγχος καθορίζει τις λειτουργίες που πρέπει να εκτελέσουν ο διάδρομος δεδομένων, η μνήμη και οι συσκευές E/E ανάλογα με τις εντολές του προγράμματος.

Η «μεγάλη εικόνα»

Τα πέντε κλασσικά μέρη ενός υπολογιστή είναι :

- Είσοδος
- Έξοδος
- Μνήμη
- Επεξεργαστής : διάδρομος δεδομένων και έλεγχος.

Ο επεξεργαστής παίρνει εντολές και δεδομένα από τη μνήμη. Η είσοδος γράφει δεδομένα στη μνήμη και η έξοδος διαβάζει δεδομένα από τη μνήμη.



Σχήμα 1.4: Η οργάνωση ενός υπολογιστή. Φαίνονται τα πέντε μέρη του υπολογιστή

Η πιο πάνω οργάνωση είναι ανεξάρτητη από την τεχνολογία του υλικού. Μπορείς να τοποθετήσεις οποιοδήποτε κομμάτι (μέρος) του υπολογιστή, είτε του παρελθόντος είτε του παρόντος σε μια από αυτές τις πέντε κατηγορίες.

Η μεγάλη εικόνα: Και το υλικό και το λογισμικό αποτελούνται από ιεραρχικά επίπεδα, όπου το χαμηλότερο επίπεδο κρύβει λεπτομέρειες του πιο πάνω επιπέδου. Έτσι προσφέρεται ένα πιο απλό μοντέλο στα ψηλότερα επίπεδα, αυτή η τεχνική ονομάζεται αφαιρετικότητα. Η αρχή της αφαίρετικότητας (abstraction) χρησιμοποιείται από τους σχεδιαστές του υλικού και λογισμικού για να μπορέσουν να αντεπεξέλθουν στη πολυπλοκότητα των υπολογιστικών συστημάτων. Ένα κλειδί διασύνδεσης μεταξύ των επιπέδων τις αφαιρετικότητας είναι η αρχιτεκτονική του συνόλου εντολών (instruction set architecture) και αποτελεί το κλειδί διασύνδεσης μεταξύ του υλικού και του χαμηλού επιπέδου λογισμικού. Η αρχιτεκτονική της ομάδας εντολών διαφέρει από Κεντρική Μονάδα Επεξεργασίας (ΚΜΕ - CPU) σε ΚΜΕ, π.χ. οι κάτωθι επεξεργαστές έχουν διαφορετική αρχιτεκτονική ομάδας εντολών:

- Intel Pentium
- Motorola 68000
- Motorola/IBM PowerPc
- MIPS R2000
- Intel 860.

Αυτή η αφαιρετική διασύνδεση επιτρέπει ποικίλες υλοποιήσεις του υλικού με διαφορετικό κόστος και απόδοση για το ίδιο λογισμικό ή αρχιτεκτονική ομάδας εντολών, π.χ. υπολογιστικά συστήματα βασιζόμενα στους μικροεπεξεργαστές:

- Intel Pentium 200 MHz.
- Intel 80486 DX4 100 MHz.

έχουν την ίδια αρχιτεκτονική ομάδας εντολών, έχουν όμως διαφορετική οργάνωση σε επίπεδο υλικού. Ο Pentium τρέχει στα 200 MHz ενώ ο 80486 τρέχει στα 100 MHz. Επίσης ο διάδρομος διασύνδεσης ΚΜΕ και μνήμης για το σύστημα Pentium τρέχει στα 66 MHz (200/3) ενώ στο σύστημα 80486 στα 25 MHz (100/4).

1.4 Εισαγωγή στις μνήμες

Υπάρχουν δύο είδη μνήμης : η DRAM (Dynamic Random Access Memory) και η Κρυφή (Cache).

Μερικές DRAMs χρησιμοποιούνται μαζί για να φυλάγονται οι εντολές και τα δεδομένα ενός προγράμματος.

Σε αντίθεση με τις μνήμες σειριακής πρόσβασης, όπως είναι οι μαγνητικές ταινίες, στη μνήμη RAM, η πρόσβαση παίρνει τον ίδιο χρόνο ανεξάρτητα από το πιο κομμάτι της μνήμης διαβάζεται.

Η κρυφή μνήμη είναι μικρή και γρήγορη (χρόνος πρόσβασης 10-12 ns), λειτουργεί ως ενδιάμεση μνήμη (buffer) της μνήμης DRAM. Είναι ένας ασφαλής χώρος για να κρύβουμε δεδομένα και εντολές.

Χρόνος πρόσβασης (access time) είναι ο χρόνος που χρειάζεται από την στιγμή που τοποθετείται η διεύθυνση μνήμης στο διάδρομο δεδομένων, μέχρι τη στιγμή που τοποθετούνται από τη μνήμη τα δεδομένα στο διάδρομο.

Σημειώσεις :
Η ομάδα εντολών της αρχιτεκτονικής ή απλώς η αρχιτεκτονική μιας μηχανής συμπεριλαμβάνει οτιδήποτε πρέπει να ξέρει ο προγραμματιστής για να φτιάξει ένα πρόγραμμα που θα τρέχει σωστά σε γλώσσα μηχανής : ποιες εντολές υποστηρίζονται από τη μηχανή, τις συσκευές Εισόδου/Εξόδου κ.ο.κ. Για μια συγκεκριμένη αρχιτεκτονική υπάρχουν πολλές διαφορετικές οργανώσεις (υλοποιήσεις) του υπολογιστή. Οργάνωση είναι η υλοποίηση της αρχιτεκτονικής (ομάδας εντολών) και καθορίζει την σχεδίαση και υλοποίηση της μνήμης, του διαύλου και της ΚΜΕ.

Η ΚΜΕ δύναται να τρέχει σε διαφορετική ταχύτητα από την ταχύτητα του διαδρόμου (data dus) του ηλεκτρονικού υπολογιστή. Για παράδειγμα, σε ένα υπολογιστικό σύστημα που βασίζεται στην ΚΜΕ Pentium 200MHz, ο μικροεπεξεργαστής τρέχει στα 200 MHz ενώ ο διάδρομος διασύνδεσης ΚΜΕ με τη μνήμη στα 66 MHz.

Ασφαλής τοποθεσία για τα δεδομένα. Κύρια και δευτερεύουσα μνήμη.
Η κυρίως μνήμη (DRAM, RAM, κ.α.) είναι διατηρήσιμη (volatile) - τα δεδομένα, οι εντολές χάνονται όταν χαθεί η παροχή. Η δευτερεύουσα μνήμη η οποία αποτελείται από τα μαγνητικά μέσα (σκληρός δίσκος, ελαστικός δίσκος, ταινία) είναι μη διατηρήσιμη.

Οι μαγνητικοί δίσκοι αποτελούν τη κύρια πηγή δευτερεύουσας μνήμης από το 1965.

Ένας μαγνητικός δίσκος, αποτελείται από ένα αριθμό από δίσκους (disk platters) που περιστρέφονται με ταχύτητα 3600 έως 5400 στροφές το λεπτό (rpm - revolutions per minute). Οι

μεταλλικοί δίσκοι επικαλύπτονται με μαγνητικό υλικό και στις δύο πλευρές, παρόμοιο υλικό που χρησιμοποιείται στις ταινίες μαγνητοφώνου.

Η διάμετρος του δίσκου κυμαίνεται από 10.25 έως 1.3 ίντσες, με δίσκους με διάμετρο πιο μικρή από μια ίντσα να είναι τώρα διαθέσιμοι. Παραδοσιακά οι μεγάλοι δίσκοι έχουν πιο ψηλή απόδοση, ενώ οι μικρότεροι δίσκοι έχουν πιο χαμηλό κόστος.

Η ανάγνωση και η γραφή των πληροφοριών γίνεται από ένα κινούμενο άξονα (movable arm), ο οποίος περιέχει μια κεφαλή ανάγνωσης/γραφής. Ο χρόνος πρόσβασης (access time) στους μαγνητικούς δίσκους κυμαίνεται από 5 σε 20 msec, ενώ ο χρόνος πρόσβασης στη μνήμη DRAM κυμαίνεται από 50 σε 150 nsec, άρα είναι 100,000 πιο γρήγορος.

Συμπεραίνοντας, τα χαρακτηριστικά των μαγνητικών δίσκων σε σύγκριση με την κυρίως μνήμη είναι :

- Οι μαγνητικοί δίσκοι είναι μη διατηρήσιμοι, επειδή είναι μαγνητικοί, έτσι μπορούν να διατηρούν τα δεδομένα ακόμη και όταν δεν είναι υπό τάση.

- Μεγαλύτερος χρόνος πρόσβασης επειδή είναι μηχανικές συσκευές.

- Χαμηλότερο κόστος για την ίδια χωρητικότητα, λόγω του ότι το κόστος παραγωγής των σκληρών δίσκων, είναι χαμηλότερο από αυτό των ολοκληρωμένων κυκλωμάτων.

1.5 Επικοινωνία με άλλους υπολογιστές

Τα δίκτυα έχουν γίνει πολύ δημοφιλή, έτσι οι σημερινοί υπολογιστές πρέπει να παρέχουν οπωσδήποτε την επιλογή για διασύνδεση με κάποιο δίκτυο.

Τα βασικά πλεονεκτήματα των δικτυωμένων υπολογιστών είναι:

- Επικοινωνία - Ανταλλαγή πληροφοριών μεταξύ των υπολογιστών σε πολύ ψηλές ταχύτητες.

- Διαμοιρασμός των πόρων - Οι συσκευές διαμοιράζονται μεταξύ των υπολογιστών του δικτύου.

- Μη τοπική (ευρεία) πρόσβαση - ο χρήστης δεν χρειάζεται να είναι κοντά στον υπολογιστή που χρησιμοποιεί.

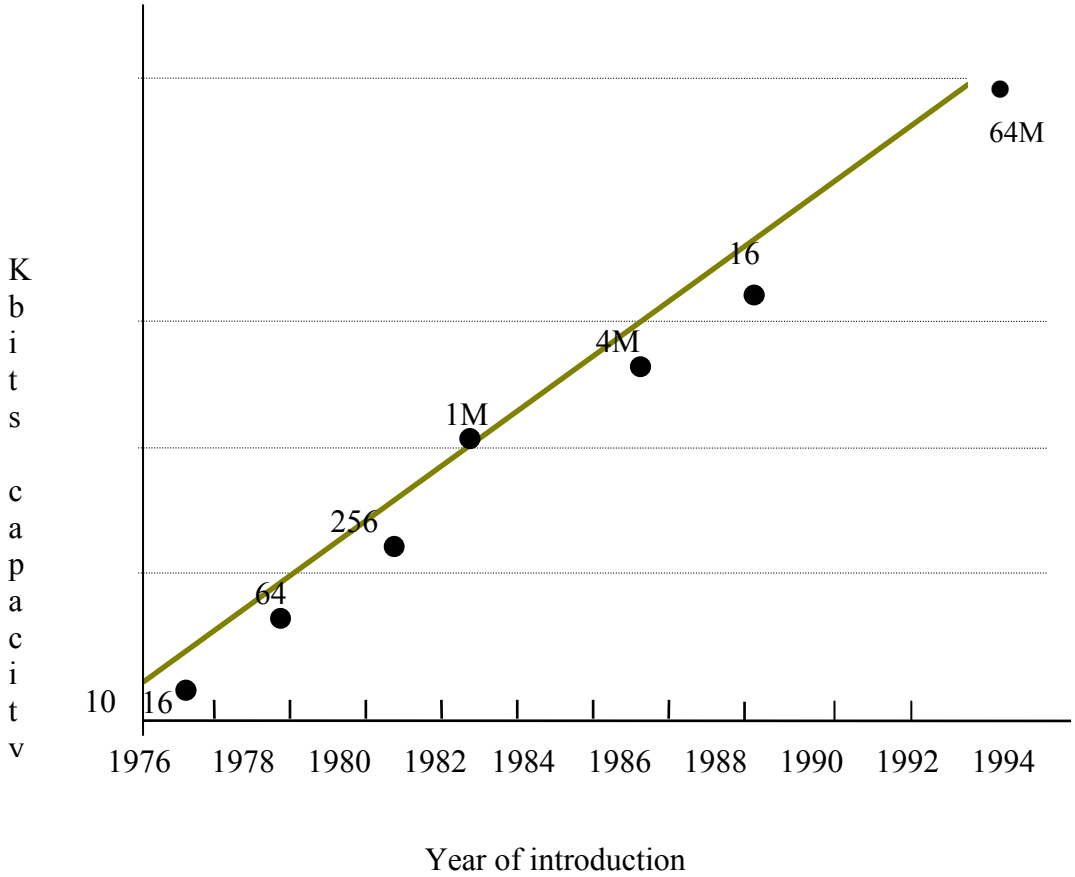
1.6 Ολοκληρωμένα Κυκλώματα

Οι επεξεργαστές και η μνήμη έχουν εξελιχθεί ραγδαία, διότι οι σχεδιαστές των υπολογιστών βασίζονται στις πιο πρόσφατες εξελίξεις της ηλεκτρονικής τεχνολογίας για το σχεδιασμό καλύτερων υπολογιστών.

Στο Σχήμα 1.6 φαίνεται η πρόοδος της τεχνολογίας και η συσχέτιση της σχετικής απόδοσης ανά μονάδα κόστους για κάθε τεχνολογία. Μια σύντομη αναδρομή στη τεχνολογία των υπολογιστών μετά το 1975, δίδεται πιο κάτω.

	Τεχνολογία που χρησιμοποιείται στους ηλεκτρονικούς υπολογιστές – Technology used in computers	Σχετική Απόδοση / Μονάδα Κόστους Relative performance / unit cost –
1951	Λυχνίες (Vacuum tube)	1
1965	Κρυσταλλοτριόδοι (Transistor)	35
1975	Ολοκληρωμένα κυκλώματα (Integrated circuit)	900
1990	Πολύ μεγάλης κλίμακας ολοκληρωμένα κυκλώματα (Very large scale integrated circuit)	400,000

Σχήμα 1.5: Σχετική απόδοση προς κάθε μονάδα κόστους των τεχνολογιών που χρησιμοποιούνται στον υπολογιστή.



Σχήμα 1.6: Η αύξηση της χωρητικότητας στο DRAM chip μέσα στο χρόνο.

Η κρυσταλλοτρίοδος (transistor) είναι απλώς ένας διακόπτης που ανοίγει και κλείνει (on/off) ανάλογα με το αν περνά ή όχι ηλεκτρισμός. Το ολοκληρωμένο κύκλωμα (intergrated circuit) αποτελείται από μερικές δεκάδες μέχρι μερικά εκατομμύρια τρανσίστορς σε ένα τσιπ (chip). Για να αποδοθεί ο μεγάλος αριθμός των κρυσταλλοτριόδων που υπάρχουν σε ένα τσιπ, χρησιμοποιείται ο όρος VLSI (Very Large Scale Integrated Circuit).

Αυτός ο ρυθμός ανάπτυξης των ολοκληρωμένων κυκλωμάτων ήταν πολύ σταθερός, όπως δίδεται και στο Σχήμα 1.7α, αναφορικά με την αύξηση της χωρητικότητας της μνήμης από το 1977.

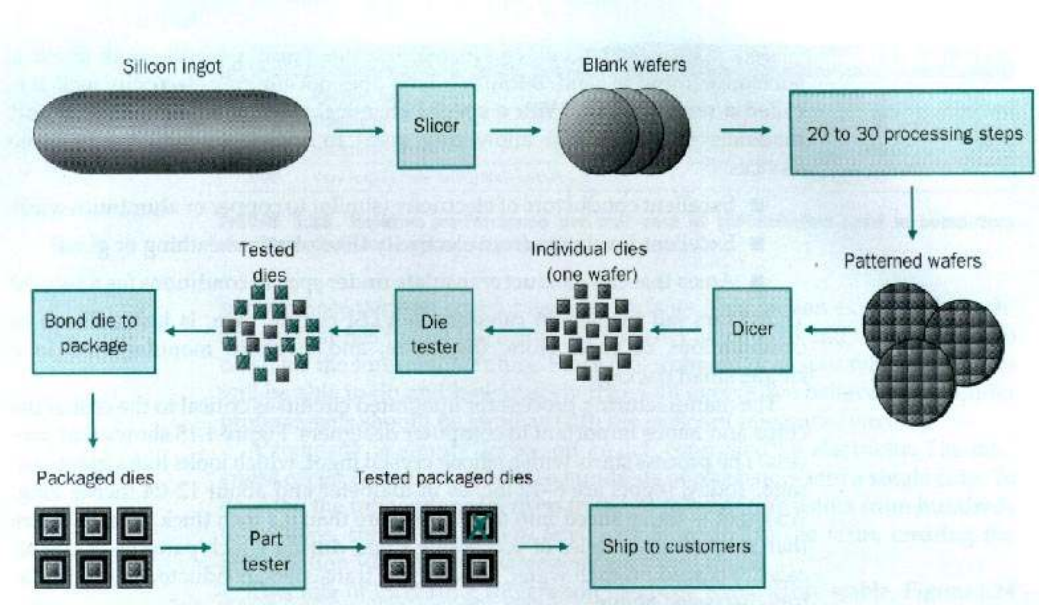
Η βιομηχανία τετραπλασίαζε τη χωρητικότητα κάθε 3 χρόνια, με αποτέλεσμα αυτή η αύξηση να ξεπερνά τις 1000 φορές, τα τελευταία χρόνια.

Αυτή η ραγδαία ανάπτυξη στη σχέση κόστους/απόδοσης και χωρητικότητας των ολοκληρωμένων κυκλωμάτων, καλύπτει τη σχεδίαση του υλικού στην ανάπτυξη της τεχνολογίας των υπολογιστών.

Η κατασκευή των τσιπς βασίζεται σε ένα υλικό που ονομάζεται σιλικόνη (silicon) το οποίο συναντάται στην άμμο. Αυτό το υλικό είναι ημιαγωγός (semiconductor), δηλαδή δεν είναι καλός αγωγός του ηλεκτρισμού, αλλά με κάποια ειδική χημική διεργασία είναι δυνατό να προσθέσουμε κάποια υλικά στη σιλικόνη, τα οποία επιτρέπουν σε μικροσκοπικές περιοχές να μετατραπούν σε μια εκ των τριών πιο κάτω συσκευών :

1. Εξαιρετικοί αγωγοί στον ηλεκτρισμό (όπως το χαλκό ή το αλουμίνιο)
2. Εξαιρετικοί απομονωτές (όπως τα πλαστικά ή το γυαλί)
3. Περιοχές που μπορούν να είναι είτε αγωγοί, είτε απομονωτές κάτω από ειδικές συνθήκες (να λειτουργεί δηλαδή σαν διακόπτης).

Τα τρανσίστορς ανήκουν στη τελευταία κατηγορία. Επομένως ένα κύκλωμα VLSI αποτελείται από εκατομμύρια συνδυασμούς από αγωγούς, μονωτές ή διακόπτες σε ένα μικρό κουτί.



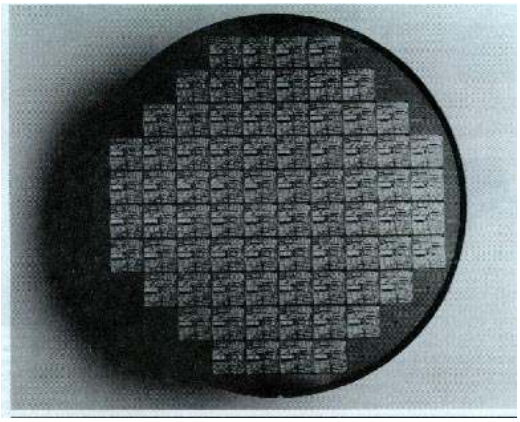
Σχήμα 1.7: Η διαδικασία κατασκευής των chip ή των dies (Patterson and Hennessy, 1998, σελ. 24).

Η ανάπτυξη ολοκληρωμένων κυκλωμάτων βασισμένων σε σιλικόνη ξεκινάει από ένα κύλινδρο σιλικόνης 5-8 ίντσες με διάμετρο x 12 ίντσες μήκος, από όπου κόβονται λεπτές πλάκες (wafers) πάχους πιο μικρού της 0.1 ίντσας. Αυτές οι πλάκες επεξεργάζονται για τη κατασκευή dies ή τσιπς. (σχήμα 1.7 β).

1.7 Κατασκευάζοντας επεξεργαστές Pentium

Σε αυτό το κεφάλαιο συνδέουμε την ιδέα των ολοκληρωμένων κυκλωμάτων με τα τσιπς που καθοδηγούν τους IBM και συμβατούς Η/Υ.

Υπάρχουν περισσότερα μικρά κομμάτια παρά μεγάλα ανά wafer: υπάρχουν 196 κομμάτια « Pentium » με διάμετρο 8 ιντσών, αλλά μόνο 78 από τα μεγάλα κομμάτια του Pentium Pro, που φαίνεται στο Σχήμα 1.8.



Σχήμα 1.8: ***Ένα 8 ιντσών wafer με επεξεργαστές Intel Pentium Pro.(Patterson and Hennessy, 1998, σελ. 26)***

Μια και το wafer στοιχίζει περίπου το ίδιο ανεξάρτητα από το τι υπάρχει σε αυτό, λιγότερα κομμάτια στοιχίζουν περισσότερα. Τα κόστος αυξάνεται περισσότερο λόγω του ότι ένα μεγαλύτερο κομμάτι είναι πιο πιθανό να έχει ελαττώματα με αποτέλεσμα να μην λειτουργήσει.

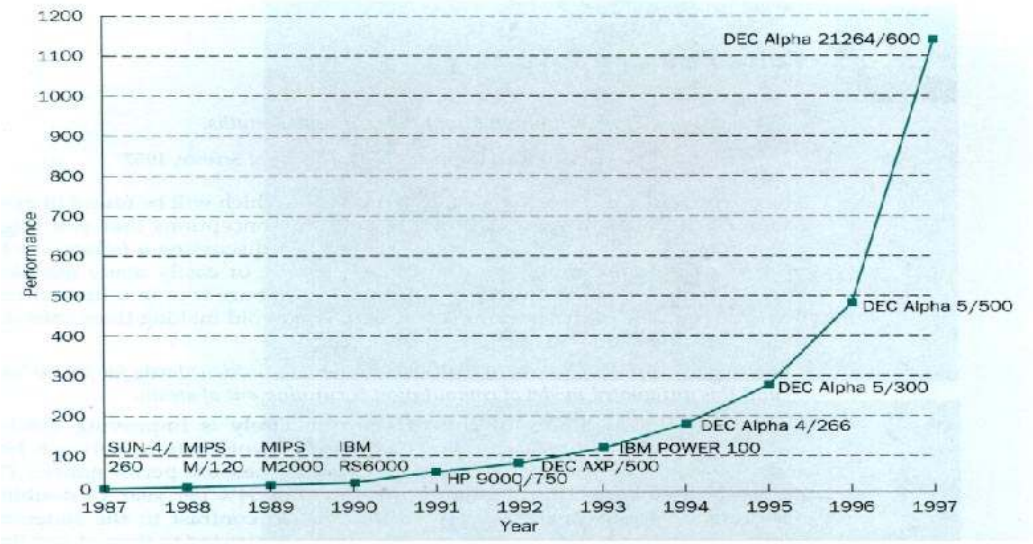
Έτσι το κόστος των τσιπς αυξάνεται γοργά καθώς το μέγεθος του τσιπ αυξάνεται. Είναι προφανές ότι οι σχεδιαστές Η/Υ πρέπει να είναι οικείοι με την τεχνολογία που χρησιμοποιούν ώστε να είναι σίγουροι ότι το προστιθέμενο κόστος των μεγαλύτερων "τσιπς" δικαιολογείται από την αυξανόμενη απόδοση.

Σημειώστε ότι το πακέτο του "Pentium Pro" στην πραγματικότητα περιέχει δύο τσιπς! Οι μηχανικοί της Intel αποφάσισαν παρά να έχουν ένα ακόμη μεγαλύτερο σε μέγεθος τσιπ για τον "Pentium Pro", να χρησιμοποιήσουν ένα δεύτερο τσιπ. Δύο τσιπς μπορούν να είναι φθηνότερα από ένα μεγαλύτερο τσιπ. Το δεύτερο κομμάτι είναι ένα εξωτερικό τσιπ κρυφής μνήμης (cache).

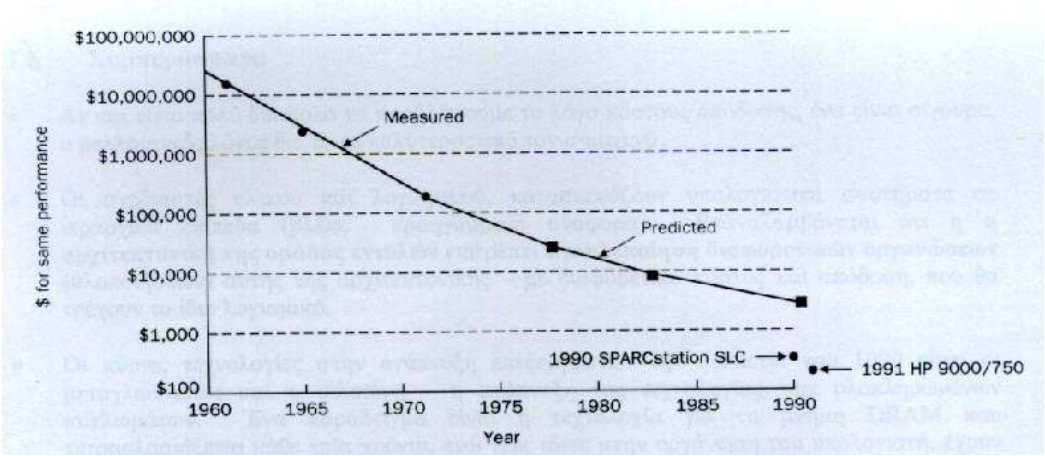
Οι σχεδιαστές των Η/Υ πρέπει να γνωρίζουν τόσο για την τεχνολογία του υλικού όσο και του λογισμικού, έτσι ώστε να κατασκευάσουν ανταγωνιστικούς Η/Υ. Οι σχεδιαστές πρέπει επίσης να γνωρίζουν τις αρχές του λογισμικού οι οποίες επηρεάζουν περισσότερο το υλικό του Η/Υ: π.χ. μεταγλωττιστές και λειτουργικά συστήματα.

1.8 Επιπλέον Έννοιες (Παρεξηγημένες έννοιες και παγίδες)

Η απόδοση των υπολογιστών αυξάνεται κατά 54% κάθε χρόνο.



Σχήμα 1.9: Παράσταση αύξησης της απόδοσης των υπολογιστών 1987– 997.(Patterson and Hennessy, 1998, σελ. 30).



Σχήμα 1.10: Πρόβλεψη του λόγου Κόστος/Απόδοσης του 1980 που έγινε το 1974

Ας υποθέσουμε ότι σχεδιάζεται μια μηχανή που θα εισαχθεί στην αγορά σε τρία χρόνια από τώρα και στόχος είναι ότι η μηχανή αυτή θα έχει τριπλάσια απόδοση από οποιαδήποτε άλλη μηχανή. Δυστυχώς η μηχανή αυτή δεν θα πωληθεί και τόσο καλά όσο αναμένεται, διότι και άλλες μηχανές θα έχουν απόδοση της ίδιας τάξης. Δεδομένου ότι ο ρυθμός ανάπτυξης της απόδοσης είναι 50% κάθε χρόνο και η απόδοση σήμερα είναι X, τότε σε τρία χρόνια η μηχανή θα έχει απόδοση $1.5^3X = 3.4X$.

Σημειώνεται ότι πολλές εταιρείες βρίσκονται ξαφνικά σε δύσκολη θέση λόγω του ότι δεν λαμβάνονται ορθά οι αποφάσεις τους.

Σε μια κυβερνητική μελέτη που έγινε στις ΗΠΑ το 1974 για την πρόβλεψη του λόγου κόστους/απόδοσης στους υπολογιστές για τη δεκαετία του 1980, ο Sun SPARC station το 1990 ήταν 10 φορές πιο γρήγορος από ότι αναμένετο και ακόμη πιο γρήγορος ήταν ο HP750. Αυτό αποδόθηκε στις απρόβλεπτες ραγδαίες εξελίξεις, με την ανάπτυξη βελτιωμένων μεταφραστών και με την εισαγωγή μηχανών RISC (Reduced Instruction Set Computer).

1.9 Συμπεράσματα

Αν και είναι πολύ δύσκολο να προβλέψουμε το λόγο κόστους/απόδοσης, ένα είναι σίγουρο, ο μελλοντικός λόγος θα είναι καλύτερος από τον σημερινό.

Οι σχεδιαστές υλικού και λογισμικού, κατασκευάζουν υπολογιστικά συστήματα σε ιεραρχικά επίπεδα (βλέπε προηγούμενη αναφορά). Επαναλαμβάνεται ότι **η αρχιτεκτονική της ομάδας εντολών** επιτρέπει την **υλοποίηση διαφορετικών οργανώσεων** (υλοποιήσεων) αυτής της αρχιτεκτονικής - με διαφορετικό κόστος και απόδοση, που θα τρέχουν το ίδιο λογισμικό.

Οι κύριες τεχνολογίες στην ανάπτυξη επεξεργαστών την δεκαετία του 1990 είναι οι μεταγλωττιστές και η σιλικόνη - η ανάπτυξη της τεχνολογίας των ολοκληρωμένων κυκλωμάτων. Ένα παράδειγμα είναι η τεχνολογία για τη μνήμη DRAM που τετραπλασιάζεται κάθε τρία χρόνια, ενώ νέες ιδέες στην οργάνωση του υπολογιστή, έχουν αυξήσει περαιτέρω το λόγο κόστους/απόδοσης.

Δύο από τις σημαντικές ιδέες που εκμεταλλεύονται τον παραλληλισμό στον επεξεργαστή είναι η διασωλήνωση και η ιεραρχία μνήμης. Αυτές οι έννοιες θα εξεταστούν με περισσότερη λεπτομέρεια αργότερα.

1.10 Γενεές Υπολογιστών

Οι ηλεκτρονικοί υπολογιστές χωρίζονται σε πέντε, μέχρι τώρα, γενεές, ανάλογα με τη τεχνολογία υλικού. Τυπικά κάθε γενιά υπολογιστών διαρκεί από οκτώ μέχρι δέκα χρόνια

Στους πιο κάτω πίνακες απεικονίζεται μια σύνοψη στις γενεές των υπολογιστών.

Γενεά	Ημερομ.	Τεχνολογία	Κύρια νέα προϊόντα
1	1950 – 1959	Λυχνίες	Εμπορικός ηλεκτρονικός υπολογιστής
2	1960 – 1968	Κρυσταλλοτριόδοι	Πιο φθηνοί ηλεκτρονικοί υπολογιστές
3	1969 – 1977	Ολοκληρωμένα κυκλώματα	Μικροϋπολογιστές
4	1978 – 199?	Μεγάλης και πολύ μεγάλης κλίμακας ολοκληρωμένα κυκλώματα	Προσωπικοί υπολογιστές και σταθμοί εργασίας
5	199? - 20??	Μικροεπεξεργαστής;	Προσωπικές, φορητές υπολογιστικές μηχανές και παράλληλοι επεξεργαστές;

Σχήμα 1.11: Οι γενεές των υπολογιστών συνήθως καθορίζονται από τις αλλαγές αλλαγές στη τεχνολογία υλικού.

Χρόνος	Όνομα	Μέγεθος (cu. ft.)	Ισχύς (watts)	Απόδοση (adds/sec)	Μνήμη (KB)	Κόστος $	Απόδοση/ Κόστος Έναντι Univac	Προσαρ-μοσμένο κόστος (1991 $)	Απόδοση/ Κόστος Έναντι Univac
1951	Univac1	1000	124, 500	1,900	48	1,000,000	1	4,533, 607	1
1964	IBM S360/ model 50	60	10, 000	500,000	64	1,000,000	263	3,756, 502	318
1965	PDP - 8	8	500	330,000	4	16,000	10,855	59,947	13,135
1976	Cray -1	58	60, 000	166,000, 000	32, 768	4,000,000	21,842	7,675, 591	51,604
1981	IBM PC	1	150	240,000	256	3,000	42,105	3,702	154,673
1991	HP9000 / model 750	2	500	50,000, 000	16, 384	7,4000	3,556, 188	7,400	16,122, 356

Σχήμα 1.12: Τα βασικά χαρακτηριστικά και η τιμή κάποιων υπολογιστών, ξεκινώντας από το 1950.

1.11 Κύρια σημεία του κεφαλαίου

- Για επιτυχή ανάπτυξη του λογισμικού χρειάζεται πολύ καλή γνώση της οργάνωσης του υπολογιστή.
- Τα δεδομένα και οι εντολές αναπαριστούνται στο δυαδικό σύστημα.
- Οι Η/Υ εκτελούν μόνο εντολές γλώσσας μηχανής.
- Τα πέντε μέρη του Η/Υ: είσοδος, έξοδος, μνήμη, διάδρομος δεδομένων και έλεγχος.
- Ιεραρχία υλικού και λογισμικού.
- Αρχιτεκτονική της ομάδας εντολών (Instruction set architecture).
- Υλοποίηση μιας αρχιτεκτονικής.
- Τα αποτελέσματα του διπλασιασμού της χωρητικότητας της τεχνολογίας κατασκευής τσιπς VLSI κάθε ενάμιση χρόνια.

2.Εντολές: Η γλώσσα μηχανής

2.1 Εισαγωγή

Οι λέξεις στη γλώσσα μηχανής ονομάζονται εντολές (instructions) και το λεξιλόγιο τους ονομάζεται ομάδα εντολών (instruction set).

Οι σχεδιαστές υπολογιστών έχουν ένα κοινό στόχο: να βρουν την γλώσσα που κάνει πιο εύκολο το κτίσιμο υλικού και μεταγλωττιστή, ενώ ταυτόχρονα μεγιστοποιεί την απόδοση και ελαχιστοποιεί το κόστος.

Η ομάδα εντολών που θα χρησιμοποιήσουμε σε αυτό το βιβλίο προέρχεται από την εταιρεία MIPS και αντιπροσωπεύει μια τυπική ομάδα εντολών που άρχισε να σχεδιάζεται από την αρχή της δεκαετίας του 1980. Στο τέλος του κεφαλαίου υπάρχουν συνοπτικοί πίνακες για την ομάδα εντολών της μηχανής MIPS.

2.2 Λειτουργίες του υλικού του υπολογιστή

Κάθε μηχανή πρέπει να είναι ικανή να κάνει αριθμητικές πράξεις. Για παράδειγμα χρησιμοποιώντας τη γλώσσα MIPS, έχουμε την εντολή :

add a, b, c

Η πιο πάνω εντολή διατάζει την μηχανή να προσθέσει δύο αριθμούς b και c και να τοποθετήσει το άθροισμα στο α.

Κάθε αριθμητική εντολή στη γλώσσα MIPS πρέπει να έχει **τρεις μεταβλητές** (τελεσταίους).

Για παράδειγμα αν θέλουμε να υπολογίσουμε την έκφραση :

$$a = b + c + d + e ,$$

στη γλώσσα MIPS θα έχουμε

```
add  a, b, c        # a= b+c
add  a, a, d        # a=a+d=(b+c)+d
add  a, a, e        #a=a+e=((b+c)+d)+e
```

Επομένως χρειάζονται τρεις εντολές για τον υπολογισμό του a.

Το σύμβολο # δηλώνει ότι ακολουθούν **σχόλια** τα οποία αγνοούνται από τον υπολογιστή. Προσέξτε ότι σε αντίθεση με άλλες προγραμματιστικές γλώσσες, κάθε γραμμή αυτής της γλώσσας (MIPS) μπορεί να περιέχει το πολύ μιαν εντολή. Επίσης, τα σχόλια τελειώνουν στο τέλος της γραμμής.

Ο φυσικός αριθμός τελεσταίων για μια λειτουργία, όπως για παράδειγμα πρόσθεση, είναι τρία: δύο τελεσταίοι που θα προστεθούν και ο τρίτος που θα κρατήσει το αποτέλεσμα. Η χρησιμοποίηση ακριβώς τριών τελεσταίων, υπακούει στη φιλοσοφία να έχουμε απλό υλικό (simple hardware). Το υλικό για ένα μεταβλητό αριθμό τελεσταίων είναι πιο πολύπλοκο από το υλικό για σταθερό αριθμό. Με βάση αυτά τα δεδομένα δίνουμε την πρώτη από τις τέσσερις αρχές, αναφορικά με την σχεδίαση υλικού:

Αρχή 1: Η ομοιότητα των λειτουργιών επιφέρει απλότητα στο hardware

Στα δύο προγράμματα που ακολουθούν δίνεται η σχέση μεταξύ των προγραμμάτων που είναι γραμμένα σε υψηλού επιπέδου γλώσσα και σε αυτά που είναι γραμμένα σε συμβολική γλώσσα. Το Σχήμα 2.1 δίνει τις εντολές MIPS που μελετήσαμε σε αυτή την ενότητα.

Category	Instructions	Example	Meaning	Comments
Arithmetic	Add	add a, b, c	a = b + c	Always 3 operators
	Subtract	sub a, b, c	a = b - c	Always 3 operators

Σχήμα 2.1: Οι εντολές της αρχιτεκτονικής MIPS που μελετήσαμε στην ενότητα 3.2

Παράδειγμα :

Δίνεται ο πιο κάτω κώδικας σε γλώσσα C (υψηλού επιπέδου) :
a = b + c
d = a - e
Τι κώδικα θα δώσει ο μεταγλωττιστής της C ;

Απάντηση*:*

Ο μεταγλωττιστής της C θα δώσει:

```
add   a, b, c
sub   d, a, e
```

Παράδειγμα:

Δίνεται η ακόλουθη εντολή σε C :
f=(g+h) - (i+j)
Τι κώδικα θα μπορούσε να παραχθεί από το μεταγλωττιστή της C ;

Απάντηση:

Ο μεταγλωττιστής της C θα μπορούσε να δώσει :

```
add t0, g, h        # προσωρινή μεταβλητή t0, περιέχει το g+h
add t1, i, j        # προσωρινή μεταβλητή t1, περιέχει το i+j
sub f, t0 , t1      # f = t0 - t1 = (g+h) - (i+j)
```

2.3 Τελεσταίοι του υλικού του υπολογιστή

Σε αντίθεση με τις γλώσσες υψηλού επιπέδου, στις συμβολικές γλώσσες οι τελεσταίοι (operators) των αριθμητικών εντολών δεν μπορούν να είναι οποιεσδήποτε μεταβλητές. Πρέπει να προέρχονται από ένα περιορισμένο αριθμό ειδικών θέσεων που ονομάζονται καταχωρητές. Οι καταχωρητές αποτελούν το κύριο στοιχείο στην κατασκευή του υπολογιστή. Το μέγεθος των καταχωρητών στην αρχιτεκτονική του MIPS είναι 32 bits. Στην αρχιτεκτονική MIPS, δίνεται συχνά η ονομασία λέξη(word) σε ομάδες των 32 bits.

Μια βασική διαφορά μεταξύ των μεταβλητών στις γλώσσες υψηλού επιπέδου και των καταχωρητών, είναι ο περιορισμένος αριθμός καταχωρητών(τυπικά, μεταξύ 16 και 32 στους σημερινούς υπολογιστές). Η αρχιτεκτονική MIPS έχει 32 καταχωρητές, οι οποίοι συμβολίζονται με $0, $1, ..., $31. Ο περιορισμένος αριθμός καταχωρητών οφείλεται στην δεύτερη αρχή σχεδίασης υπολογιστών:

Αρχή 2: Το μικρότερο είναι γρηγορότερο

Αν ο αριθμός των καταχωρητών ήταν μεγαλύτερος, τότε θα αυξανόταν η διάρκεια του κύκλου του ρολογιού, διότι τα ηλεκτρονικά σήματα χρειάζονται περισσότερο χρόνο όταν πρέπει να ταξιδέψουν πιο μακριά.

Στην αρχιτεκτονική MIPS οι εντολές γράφονται σε 4 bytes, επιπρόσθετα υπάρχει ένας άλλος καταχωρητής (PC) στον οποίο φυλάγεται η διεύθυνση της επόμενης εντολής που θα εκτελεστεί.

Πρέπει να είναι ξεκάθαρο ότι η **αποτελεσματική χρήση των καταχωρητών είναι το κλειδί στην απόδοση των προγραμμάτων.**

Παράδειγμα:

Έστω η πιο κάτω εντολή στη C :

f = (g + h) - (i + j) ;

Ο μεταγλωττιστής της C συσχετίζει τις μεταβλητές f, g, h, i, και j με τους καταχωρητές $16, $17, $18, $19, και $20 αντιστοίχως.

Ποιος θα είναι ο κώδικας σε MIPS ;

Απάντηση:

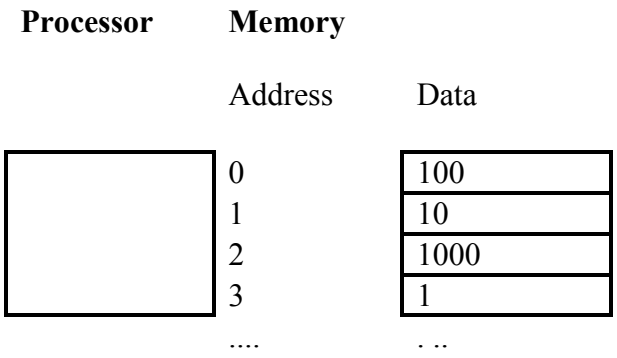
```
add $8, $17, $18        # ο καταχωρητής $8 έχει το αποτέλεσμα g+h
add $9, $19, $20        # ο καταχωρητής $9 έχει το αποτέλεσμα i+j
sub  $16, $8, $9        #  ο καταχωρητής $16 έχει το αποτέλεσμα (g+h) - (i+j),
                        #   την τιμή της μεταβλητής f
```

Οι καταχωρητές $8 και $9 αντιστοιχούν στις μεταβλητές t_0 και t_1 του προηγούμενου παραδείγματος.

Η αριθμητικές πράξεις στην αρχιτεκτονική MIPS μπορούν να γίνουν μόνο μέσω των καταχωρητών, αντίθετα με τον INTEL όπου μπορούν να γίνουν και στην μνήμη. Για αυτό το λόγο χρειαζόμαστε εντολές για την μεταφορά δεδομένων από την μνήμη στους καταχωρητές.

Ο επεξεργαστής μπορεί να κρατήσει περιορισμένο αριθμό δεδομένων. Για αυτό χρειάζεται η μνήμη για την φύλαξη μεγάλων ποσοτήτων δεδομένων (πίνακες - μεταβλητές).

Για να έχουμε πρόσβαση (access) σε μια λέξη που βρίσκεται στη μνήμη πρέπει να δώσουμε την αντίστοιχη διεύθυνση στη μνήμη. Η μνήμη στην πραγματικότητα είναι ένας μεγάλος, μονοδιάστατος πίνακας και η διεύθυνση ένας δείκτης στον πίνακα. Οι διευθύνσεις ξεκινούν από το 0. Όπως δίνετε στο Σχήμα 3.2, για να έχουμε πρόσβαση στο τρίτο στοιχείο του πίνακα θα χρησιμοποιήσουμε διεύθυνση Memory [2], τα δεδομένα του οποίου είναι 1000.



Σχήμα 2.2: ***Μερικές διευθύνσεις στη μνήμη με τα αντίστοιχα περιεχόμενα σε αυτές τις θέσεις.***

Η εντολή που μεταφέρει δεδομένα από την μνήμη σε ένα καταχωρητή, ονομάζεται **load word - lw** (εντολή φόρτωσης). Η μορφή της εντολής φόρτωσης είναι: το όνομα της λειτουργίας ακολουθούμενο πρώτα από τον καταχωρητή που θα πάρει τα δεδομένα (θα φορτωθεί), μετά από τη διεύθυνση που ξεκινά ο πίνακας στην μνήμη, και τέλος από έναν καταχωρητή που περιέχει το δείκτη (θέση) του στοιχείου του πίνακα που θα φορτωθεί. Η διεύθυνση μνήμης του στοιχείου του πίνακα καθορίζεται από το άθροισμα της βάσης (αρχικής διεύθυνσης) του πίνακα και του καταχωρητή (δείκτη στον πίνακα). Βλέπε το πιο κάτω παράδειγμα :

Παράδειγμα:

Υποθέστε ότι ο πίνακας A έχει 100 στοιχεία και ο μεταγλωττιστής έχει συσχετίσει τις μεταβλητές g, h, και i με τους καταχωρητές $17, $18, και $19. Η βάση του πίνακα(διεύθυνση του πρώτου στοιχείου στον πίνακα) δίνεται είναι η διεύθυνση Astart. Μεταφράστε την πιο κάτω εντολή σε γλώσσα MIPS :

$$g = h + A[i];$$

Απάντηση:

Σε γλώσσα MIPS έχει ως ακολούθως:

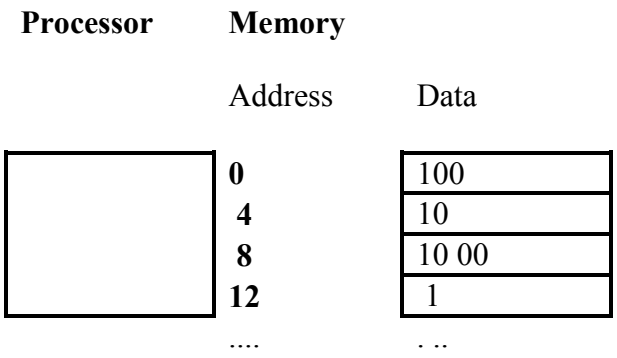
```
lw $8, Astart ($19)     # ο προσωρινός καταχωρητής $8 παίρνει το A[i]
add $17, $18, $8        # το αποτέλεσμα  h + A[i] κρατείται στον καταχωρητή $17,
                        # ο καταχωρητής $17 κρατάει την τιμή της μεταβλητής g
```

Η εντολή lw προσθέτει στην βάση της διεύθυνσης του πίνακα A (Astart) τον δείκτη i που βρίσκεται στον καταχωριτή $19, για να υπολογίσει την διεύθυνση του στοιχείου A[i]. Ο καταχωρητής που κρατεί την τιμή του δείκτη του πίνακα ονομάζεται καταχωρητής δείκτη (index register).

2.4 Διασύνδεση υλικού/λογισμικού

Ο μεταγλωττιστής είναι υπεύθυνος για την συσχέτιση μεταβλητών και καταχωρητών, και επίσης για την ανάθεση δομών δεδομένων, όπως για παράδειγμα πίνακες, στους χώρους της μνήμης. Ακολούθως ο μεταγλωττιστής μπορεί να τοποθετήσει την σωστή αρχική διεύθυνση στις εντολές μεταφοράς δεδομένων π.χ. lw. Ο επεξεργαστής MIPS μπορεί να έχει διεύθυνση για μεταφορά ενός Byte (8 bits), ή τεσσάρων Bytes (μιας λέξης). Στη δεύτερη περίπτωση, οι

διευθύνσεις διαφέρουν κατά 4 (4 bytes). Το Σχήμα 2.3 δίνει τις πραγματικές διευθύνσεις του Σχήματος 2.2.



Σχήμα 2.3: Οι πραγματικές διευθύνσεις μνήμης στον MIPS και τα αντίστοιχα περιεχόμενα τους.

Η διευθυνσιοδότηση επηρεάζει επίσης τον δείκτη i. Για τον σωστό υπολογισμό της διεύθυνσης του παραδείγματος, ο καταχωρητής $19 (στον οποίο περιέχεται η τιμή της μεταβλητής i) πρέπει να πολλαπλασιάζεται με 4 (4 x i) και να προστίθεται στο Astart (βάση του πίνακα) για τον υπολογισμό της διεύθυνσης.

Η αντίστροφη εντολή της load word (φόρτωσε), είναι η **store word - sw** (φύλαξε). Η μορφή της εντολής store είναι παρόμοια με αυτή της εντολής load, δηλαδή, αποτελείται από το όνομα της λειτουργίας, τον καταχωρητή του οποίου τα περιεχόμενα θα αποθηκευθούν(φυλαχτεί) στη μνήμη, τη βάση του πίνακα και τέλος ένας καταχωρητής στον οποίο περιέχεται ο δείκτης του στοιχείου που θα αποθηκευθεί στον πίνακα.

Παράδειγμα:

Υποθέστε ότι η μεταβλητή h συσχετίζεται με τον καταχωρητή $18. Υποθέστε επίσης ότι ο καταχωρητής $19 έχει την τιμή 4 x i. (στο επόμενο κεφάλαιο θα μελετηθεί πως γίνεται ο πολλαπλασιασμός στον MIPS). Δίνεται ο πιο κάτω κώδικας σε C :

A[i] = h + A[i];

Ποιος είναι ο αντίστοιχος κώδικας σε γλώσσα MIPS;

Απάντηση:

Ο κώδικας σε γλώσσα MIPS είναι:

```
lw $8, Astart ($19)     # ο καταχωρητής $8 παίρνει προσωρινά την τιμή του A[i]
add $8, $18 , $8        # ο καταχωρητής $8 παίρνει προσωρινά την τιμή του h+A[i]
sw $8, Astart ($19)     # Το h+A[i] φυλάγεται πίσω στη θέση A[i]
```

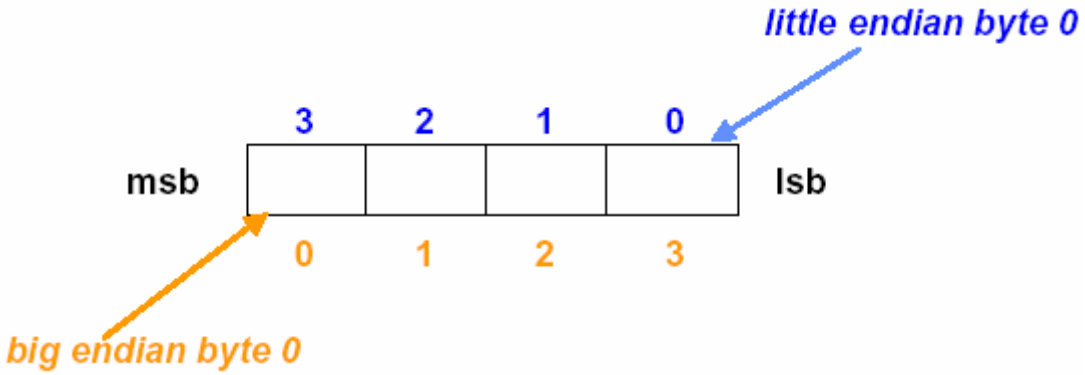
Τα περισσότερα προγράμματα χρησιμοποιούν περισσότερες μεταβλητές σε σχέση με τους καταχωρητές που διαθέτει η μηχανή. Γι' αυτό το λόγο, ο μεταγλωττιστής προσπαθεί να τοποθετήσει τις μεταβλητές που χρησιμοποιούνται συχνότερα, στους καταχωρητές, και τις υπόλοιπες (μεταβλητές) στην μνήμη. Η διαδικασία της τοποθέτησης των λιγότερα χρησιμοποιημένων μεταβλητών στην μνήμη, ονομάζεται **spilling registers**. Αν, για παράδειγμα, έχουμε περισσότερες από 32 μεταβλητές σε ένα πρόγραμμα. τότε υποχρεωτικά κάποιες θα πρέπει να βρίσκονται στην μνήμη(μιλώντας πάντα για τον MIPS).

Η αρχή του υλικού που συσχετίζει το μέγεθος με την ταχύτητα, εισηγείται ότι η μνήμη πρέπει να είναι πιο αργή από τους καταχωρητές, δεδομένου ότι ο αριθμός των καταχωρητών είναι πιο μικρός. Επίσης τα δεδομένα στους καταχωρητές πρέπει να διαχειρίζονται πολύ πιο εύκολα. Μια αριθμητική εντολή (add, sub, κ.α.) στον MIPS μπορεί να διαβάσει τα δεδομένα δύο καταχωρητών, να εκτελέσει την αντίστοιχη αριθμητική λειτουργία και να γράψει το αποτέλεσμα. Μια εντολή για μεταφορά δεδομένων (lw, sw, κ.α), διαβάζει ή γράφει χωρίς να επεξεργάζεται τα δεδομένα. Επομένως οι καταχωρητές στον MIPS είναι πιο γρήγοροι στην πρόσβαση και πιο εύκολοι στη χρήση. Συμπέρασμα: **Για επίτευξη ψηλής απόδοσης, οι μεταγλωττιστές πρέπει να χρησιμοποιούν αποδοτικά τούς καταχωρητές.**

Στον MIPS υπάρχουν εντολές για load word και store word, για 16 bit (half word) - lh/sh και 8bit - lb/sb δεδομένων, επιπρόσθετα στα 32 bit που έχουμε μελετήσει.

2.5 Διάταξη των Bytes σε μία λέξη

Μέσα σ' έναν ακέραιο αριθμό, τα bits εκείνα που πολλαπλασιάζονται επί τις μεγαλύτερες δυνάμεις του 2 για να μας δώσουν την αριθμητική τιμή του ακεραίου λέγονται "περισσότερο σημαντικά" (MS - most significant) bits, και αυτά που πολλαπλασιάζονται επί τις μικρότερες δυνάμεις του 2 λέγονται "λιγότερο σημαντικά" (LS - least significant) bits. Το byte που περιέχει τα MS bits λέγεται MS byte, και εκείνο που περιέχει τα LS bits λέγεται LS byte. Όποτε σχεδιάζουμε έναν ακέραιο στο χαρτί, οριζόντια, θα βάζουμε πάντα τα MS bits και byte αριστερά, και τα LS bits και byte δεξιά, δηλαδή όπως και στους δεκαδικούς αριθμούς (αυτό αφορά μόνο τους ανθρώπους, και όχι τους υπολογιστές...).



Μεγάλος ινδιάνος (Big endian) : Byte # 0 1 2 3
Μικρός ινδιάνος (Little endian) : Byte # 3 2 1 0

- **Big-Endian**: Σε πολλούς υπολογιστές, το MS byte του κάθε ακεραίου έχει τη μικρότερη διεύθυνση, και οι διευθύνσεις των bytes του αυξάνουν (προχωρούν) καθώς προχωράμε "δεξιά", προς το LS byte του. Αυτοί οι υπολογιστές λέγονται "big-endian" διότι η αρίθμηση των bytes ξεκινά από το "big end", δηλαδή το MS byte.
- **Little-Endian**: Σε άλλους υπολογιστές, το LS byte του κάθε ακεραίου έχει τη μικρότερη διεύθυνση, και οι διευθύνσεις των bytes του αυξάνουν (προχωρούν) καθώς προχωράμε "αριστερά", προς το MS byte του. Αυτοί οι υπολογιστές λέγονται "little-endian" διότι η αρίθμηση των bytes ξεκινά από το "little end", δηλαδή το LS byte.

Το "endian-ness" του υπολογιστή, δηλαδή το αν είναι big-endian ή little-endian, δεν μας επηρεάζει όταν πάντα γράφουμε και διαβάζουμε την κάθε ποσότητα με τον ίδιο τύπο (γράφω

string και διαβάζω string, ή γράφω integer και διαβάζω integer), ενώ μας επηρεάζει όταν αλλάζουμε τύπο μεταξύ εγγραφής και ανάγνωσης (γράφω string και διαβάζω integer, ή γράφω integer και διαβάζω string). Επίσης το endian-ness του υπολογιστή μας επηρεάζει όταν μεταφέρουμε δυαδικά δεδομένα (όχι κείμενο) μέσω δικτύου μεταξύ υπολογιστών με διαφορετικό endian-ness, και δεν πούμε στο πρόγραμμα μεταφοράς ότι αυτά που μεταφέρουμε δεν είναι κείμενο.

Ο επεξεργαστής MIPS εργάζεται και με τούς δύο τρόπους. Η οικογένεια επεξεργαστών της Intel ακολουθεί τον Μικρό ινδιάνο(Little Endian).

2.6 Αναπαράσταση εντολών στην μηχανή

Οπως είναι γνωστό, οι εντολές είναι κωδικοποιημένες στο δυαδικό σύστημα, σαν μία ακολουθία από δυαδικά ψηφία. Η κωδικοποίηση των εντολών αποφασίζεται από τον κατασκευαστή του επεξεργαστή. Η κάθε εντολή αποτελείται από ένα πεπερασμένο αριθμό ψηφίων, για τον MIPS ο αριθμός αυτός είναι 32, και χωρίζεται σε ένα αριθμό κομματιών (segments). Κάθε κομμάτι της εντολής καθορίζει μία συγκεκριμένη λειτουργία ή δήλωση. Το κάθε ένα από αυτά τα κομμάτια της εντολής ονομάζεται πεδίο (field).

Για παράδειγμα, έχουμε την πιο κάτω εντολή σε γλώσσα MIPS:

add $8, $17, $18

Η γενική μορφή της εντολής στον επεξεργαστή θα είναι :

0	17	18	8	0	32

(α) Σε δεκαδική μορφή

Το πρώτο και τελευταίο πεδίο (0 & 32) σε συνδιασμό δηλώνουν ότι η πράξη που θα εκτελεστεί είναι η πρόσθεση. Το δεύτερο πεδίο δίνει τον αριθμό του καταχωρητή ($17) που είναι η πηγή του πρώτου τελεσταίου της πρόσθεση. Το τρίτο πεδίο δίνει τον αριθμό του καταχωρητή ($18) που είναι η πηγή του δεύτερου τελεσταίου της πρόσθεσης. Το τέταρτο πεδίο δίνει τον αριθμό του καταχωρητή ($8) που θα δεκτεί το αποτέλεσμα της πρόσθεσης. Το πέμπτο πεδίο δεν χρησιμοποιείται σε αυτή την εντολή και γι'αυτό παίρνει την τιμή 0. Αυτή η εντολή προσθέτει τον καταχωρητή (τα περιεχόμενα του) $17 στον καταχωρητή (στα περιεχόμενα του) $18 και τοποθετεί το αποτέλεσμα στον καταχωρητή $8.

Η εντολή μπορεί να αναπαρασταθεί και με τα αντίστοιχα πεδία στο δυαδικό σύστημα, όπως φαίνεται στο πιο κάτω σχήμα:

000000	10001	10010	01000	00000	100000
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

(β) Σε δυαδική μορφή

Όλες οι εντολές στο MIPS έχουν μέγεθος 32 bits, άρα διατηρείται η πρώτη αρχή που θέσαμε. Τα πιο πάνω πεδία έχουν συγκεκριμένα ονόματα:

op	Rs	Rt	Rd	shamt	Funct
6bits	5 bits	5 bits	5 bits	5 bits	6 bits

(γ) Η γενική μορφή της εντολής στη μηχανή MIPS με τα αντίστοιχα ονόματα των πεδίων.

Το κάθε πεδίο στην πιο πάνω κωδικοποίηση έχει ως ακολούθως:

- **op**: η λειτουργία της εντολής (operation of the instruction)
- **rs**: ο πρώτος καταχωρητής εισόδου (the first register, source operand)
- **rt**: ο δεύτερος καταχωρητής εισόδου (the second register, source operand)
- **rd**: ο καταχωρητής που θα πάρει το αποτέλεσμα (the register destination operand)
- **shamt**: ποσότητα μετακίνησης (Shift amount – δεν θα μας απασχολήσει, το θεωρούμε 0)
- **funct**: συνάρτηση (function). Αυτό το πεδίο αποτελεί διαφοροποίηση της εντολής από το πεδίο op.

Οταν η εντολή χρειάζεται μεγαλύτερα πεδία από αυτά που δίνονται πιο πάνω παρουσιάζονται προβλήματα. Για παράδειγμα, στην εντολή lw πρέπει να προσδιορισθούν δύο καταχωρητές και μια διεύθυνση. Αν οι διευθύνσεις χρειάζονταν μόνο 5 bits ($=2^5$ ή 32 θέσεις μνήμης) τότε αυτό θα ήταν δυνατό. Όμως, αυτό το μέγεθος είναι πολύ μικρό για να μπορέσει να χρησιμοποιηθεί για διευθύνσεις δεδομένων.

Έτσι, πρέπει να υιοθετήσουμε και άλλες μορφές εντολών. Αυτό μας οδηγεί στην τρίτη αρχή της σχεδίασης υλικού:

Αρχή 3: Η καλή σχεδίαση απαιτεί συμβιβασμούς

Οι σχεδιαστές του MIPS αποφάσισαν να κρατήσουν όλες τις εντολές στο ίδιο μήκος -32 bits- επομένως, διαφορετικά είδη εντολών θα πρέπει να έχουν διαφορετική μορφή.

Για παράδειγμα, η πιο πάνω μορφή ονομάζεται **R-type (Register type)**. Ένα δεύτερο είδος εντολής ονομάζεται **I-type** και χρησιμοποιείται από τις εντολές μετακίνησης δεδομένων (π.χ. lw, sw). Τα πεδία αυτής της μορφής είναι τέσσερα:

Op	Rs	Rt	address
6 bits	5 bits	5 bits	16 bits

Για παράδειγμα, αν έχουμε την εντολή:

lw $8, Astart ($19)

τότε το 19 θα τοποθετηθεί στο πεδίο rs, το 8 θα τοποθετηθεί στο πεδίο **rt**, και η βάση της διεύθυνσης του πίνακα Astart στο **πεδίο διευθύνσεων** (address). Σημειώστε ότι το πεδίο rt στην εντολή lw δίνει το πεδίο που θα πάρει το αποτέλεσμα.

Κάθε εντολή πηγαίνει από τη μνήμη στον επεξεργαστή, αποκωδικοποιείται, δραστηριοποιεί την αριθμητική και λογική μονάδα και γίνονται οι κατάλληλες πράξεις. Στην μνήμη υπάρχει πρώτα ο κώδικας και μετά τα δεδομένα του προγράμματος.

Αν και πολλές μορφές εντολών προκαλούν πολυπλοκότητα στο υλικό μπορούμε να μειώσουμε την πολυπλοκότητα κρατώντας τις μορφές των εντολών παρόμοιες.

Για παράδειγμα, τα πρώτα τρία πεδία στις μορφές R-type και I-type έχουν τα ίδια ονόματα και το τέταρτο πεδίο στην I-type είναι ίσο με το μήκος των άλλων τριών πεδίων του R-type. Υπάρχει μια ομοιομορφία ανάμεσα στους δύο τύπους, αυτό βοηθά στην αποκωδικοποίηση του προγράμματος, χρησιμοποιώντας το ίδιο κύκλωμα.

Οι διάφορες μορφές των εντολών ξεχωρίζουν από τις τιμές που δίνονται στο πρώτο πεδίο (op). Η κάθε μορφή παίρνει συγκεκριμένες τιμές στο πεδίο op, γι'αυτό και παραδοσιακά το πεδίο αυτό ονομάζεται **opcode** -κωδικός εντολής.

Το πιο κάτω σχήμα (σχήμα 2.4) δίνει τις κωδικοποιήσεις των εντολών που μελετήσαμε μέχρι τώρα

Instruction	Format	Op	Rs	Rt	rd	shamt	funct	address
Add	R	0	Reg	Reg	reg	0	32	n.a.
Sub	R	0	Reg	Reg	reg	0	34	n.a.
Lw	I	35	Reg	Reg	n.a.	n.a.	n.a.	address
Sw	I	43	Reg	Reg	n.a.	n.a.	n.a.	address

Μεγέθη των πεδίων για τις πιο πάνω εντολές :

Field	Op	rs	rt	rd	shamt	Funct	address
Size	6 bits	6 bits	5 bits	5 bits	5 bits	6 bits	16 bits

Σχήμα 2.4:Η αποκωδικοποίηση των εντολών της μηχανής MIPS. Το reg σημαίνει ένας αριθμός καταχωρητή από το 0 μέχρι το 31, address σημαίνει μια 16-bit διεύθυνση και το n.a. σημαίνει not appear, δηλαδή το συγκεκριμένο πεδίο δεν εμφανίζεται στη μορφή της εντολής.

Στο πιο πάνω σχήμα προσέξτε ότι οι εντολές add και sub έχουν το ίδιο op πεδίο. Για το λόγο αυτό, το υλικό χρησιμοποιεί το πεδίο funct για να προσδιορίσει ποια αριθμητική πράξη (πρόσθεση ή αφαίρεση) πρέπει να εκτελέσει.

Παράδειγμα:

Υποθέστε ότι η μεταβλητή h συσχετίζεται με τον καταχωρητή $18 και επίσης ότι ο καταχωρητής $19 έχει την τιμή 4 x i (όπως στο προηγούμενο παράδειγμα). Η πιο κάτω εντολή στη C

$$A[i] = h + A[i] ;$$

μεταγλωττίζεται σε:

```
lw $8, Astart ($19)     # ο καταχωρητής $8 παίρνει προσωρινά το A[i]
add $8, $18 , $8        # ο καταχωρητής $8 παίρνει προσωρινά h+A[i]
sw $8, Astart ($19)     # Το h+A[i] φυλάγεται πίσω στη θέση A[i]
```

Ποιος είναι ο κώδικας σε γλώσσα μηχανής (MIPS) για τις πιο πάνω εντολές;

Απάντηση:

Για ευκολία θα χρησιμοποιήσουμε δεκαδικούς αριθμούς για την απεικόνιση τους σε γλώσσα μηχανής. Υποθέτουμε ότι η βάση του πίνακα (αρχική διεύθυνση) είναι το 1200 στο δεκαδικό (ή 0000 0100 1011 0000 στο δυαδικό). Η μορφή των τριών εντολών στο δεκαδικό σύστημα είναι:

op	*Rs*	*Rt*	*(rd)*	*(shamt)*	*address OR funct*
35	*19*	*8*	*1200 (Astart)*		
0	*18*	*8*	*8*	*0*	*32*
43	*19*	*8*	*1200*		

Η εντολή lw προσδιορίζεται από τον αριθμό 35 (πεδίο op). Η εντολή add προσδιορίζεται από το πρώτο και τελευταίο πεδίο (0 και 32) και η εντολή sw προσδιορίζεται από τον αριθμό 43 στο πρώτο πεδίο (βλέπε Σχήμα 3.5).

Η αντίστοιχη δυαδική μορφή των εντολών (τρόπος αναπαράστασης τους σε γλώσσα μηχανής) είναι:

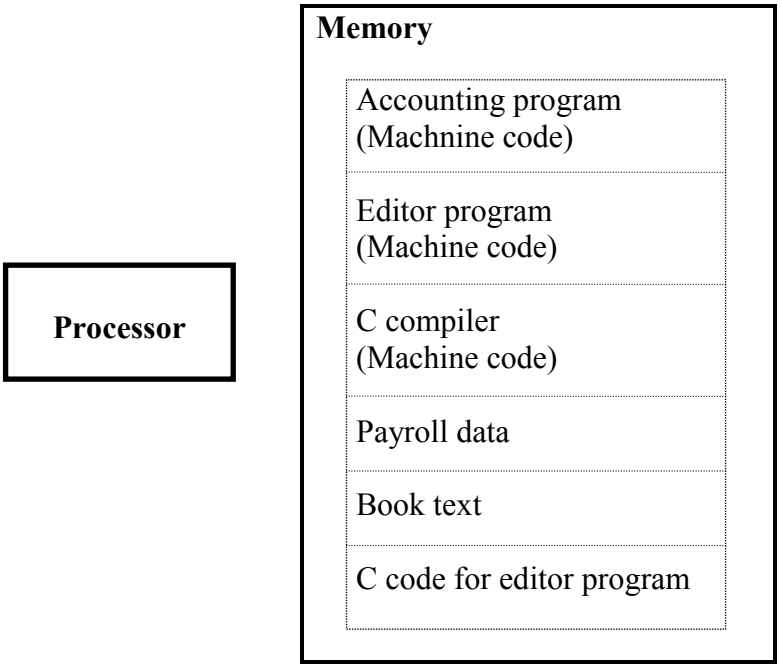
100011	*10011*	*01000*	*0000 0100 1011 0000*		
000000	*10010*	*01000*	*01000*	*00000*	*100000*
101011	*10011*	*01000*	*0000 0100 1011 0000*		

Προσέξτε την ομοιότητα της πρώτης και τρίτης εντολής στην δυαδική απεικόνιση. Η μόνη διαφορά τους βρίσκεται στο πρώτο πεδίο, στο τρίτο bit από τα αριστερά.

Σήμερα οι υπολογιστές είναι κτισμένοι πάνω σε δύο βασικές αρχές:

1. Οι εντολές απεικονίζονται ως αριθμοί.
2. Τα προγράμματα μπορούν να αποθηκευθούν στη μνήμη για να διαβαστούν, ή να γραφούν, όπως οι αριθμοί. Αυτή η αρχή ονομάζεται **"η έννοια του αποθηκευμένου προγράμματος"** (stored program concept). John Van Neuman
   Αποθηκευμένα Προγράμματα : Φυλάγονται τα προγράμματα στη μνήμη σαν κώδικας σε γλώσσα μηχανής.

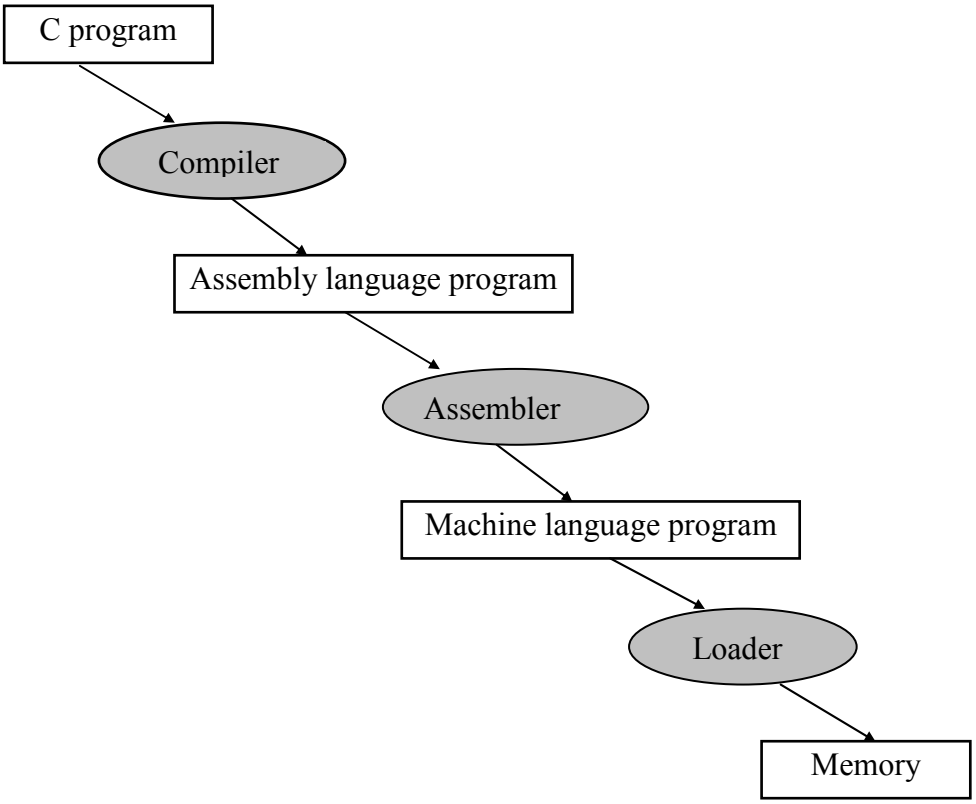
Στο Σχήμα 3.5 φαίνονται οι δυνατότητες ενός αποθηκευμένου προγράμματος. Για παράδειγμα, η μνήμη μπορεί να περιέχει, τον κώδικα σε C για ένα συντάκτη κειμένου, τον αντίστοιχο μεταγλωττισμένο κώδικα μηχανής και ακόμη τον μεταγλωττιστή που παράγει τον κώδικα μηχανής.



Σχήμα 2.5:*Η έννοια του αποθηκευμένου προγράμματος.*

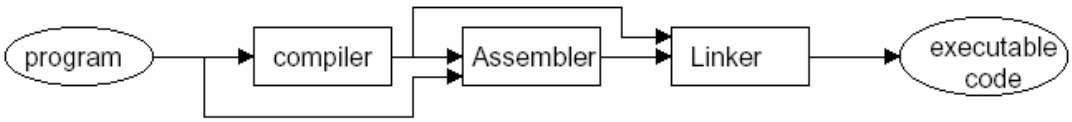
2.6.1 Διασύνδεση Υλικού/Λογισμικού

Όπως έχουμε τονίσει στο Κεφάλαιο 1, η συμβολική αναπαράσταση των εντολών ονομάζεται συμβολική γλώσσα, και η αντίστοιχη αναπαράσταση με αριθμούς ονομάζεται γλώσσα μηχανής. Στο Σχήμα 2.6 απεικονίζεται η ιεραρχία μετάφρασης για ένα πρόγραμμα από υψηλού επιπέδου γλώσσα μέχρι τη γλώσσα μηχανής. Για παράδειγμα, ένα πρόγραμμα σε C (υψηλού επιπέδου γλώσσα) μεταφράζεται σε συμβολική γλώσσα από τον μεταγλωττιστή και μετά ο συμβολομεταφραστής μεταφράζει την συμβολική γλώσσα σε γλώσσα μηχανής. Το πρόγραμμα που τοποθετεί το πρόγραμμα της γλώσσας μηχανής στη μνήμη για εκτέλεση ονομάζεται **φορτωτής (loader)**.



Σχήμα 2.6 Η ιεραρχία μετάφρασης ενός προγράμματος από ψηλού επιπέδου γλώσσα, στη γλώσσα μηχανής.

Ο Loader αναλαμβάνει να τοποθετήσει στη μνήμη τη γλώσσα μηχανής που παράγει ο assembler.



Ο Linker αναλαμβάνει να 'σπάσει' το πρόγραμμά μας σε διαδικασίες τις οποίες διαχειρίζεται χωριστά.

Ιδιομορφίες του MIPS

Ο καταχωρητής **$0** στη μηχανή MIPS έχει πάντα την **τιμή 0**. Γι' αυτό ο συμβολομεταφραστής της γλώσσας MIPS δέχεται την πιο κάτω εντολή αν και δεν ανήκει στην αρχιτεκτονική της μηχανής MIPS:

```
move $8, $18      #ο καταχωρητής 8 θα πάρει το περιεχόμενο του
                  #καταχωρητή 18.
```

Ο συμβολομεταφραστής μεταφράζει την πιο πάνω εντολή στην αντίστοιχη εντολή του MIPS που είναι:

```
add $8, $0, $18   # ο καταχωρητής 8 παίρνει το άθροισμα του
                  # καταχωρητή 0 και 18.
```

Το πιο πάνω παράδειγμα δείχνει πως μπορεί να γίνεται η μεταφορά δεδομένων από τον ένα καταχωρητή στον άλλο (move) με την χρησιμοποίηση **ψευδοεντολών** (pseudo instructions) σε συμβολική γλώσσα. Μπορούμε να χρησιμοποιήσουμε ψευδοεντολές στη θέση άλλων εντολών. Όπως και στο πιο πάνω παράδειγμα, η move δεν υποστηρίζεται από τον MIPS, είναι ψευδοεντολή και συσχετίζεται με την εντολή add.

2.7 Εντολές για ανάληψη αποφάσεων

Η μηχανή MIPS υποστηρίζει δύο εντολές για λήψη αποφάσεων, παρόμοιες με το if και το goto,

beq register1, register2, L1

Η πιο πάνω εντολή μεταφέρει την εκτέλεση του προγράμματος στην εντολή που έχει **ετικέτα (label)** L1 αν τα δεδομένα του register1 ισούνται με τα δεδομένα του register2. Η εντολή ονομάζεται **branch equal**. Η επόμενη εντολή για ανάληψη αποφάσεων είναι:

bne register1, register2, L1

Η πιο πάνω εντολή πηγαίνει στην εντολή που έχει ετικέτα L2 αν τα δεδομένα του register1 και register2 δεν ισούνται **(branch not equal)**. Αυτές οι δύο εντολές ονομάζονται **διακλαδώσεις υπό συνθήκη (conditional branches)**.

Παράδειγμα

Δίνεται ο ακόλουθος κώδικας σε γλώσσα C :

```
    if ( i = = j )  goto  L1;
    f = g + h;
L1: f = f - i;
```

Οι μεταβλητές f, g, h, i και j είναι τοποθετημένες στους καταχωρητές $16 ώες $20. Ποιος θα είναι ο κώδικας σε MIPS;

Απάντηση

```
    beq   $19, $20, L1          # go to L1  αν  i = j
    add  $16, $17, $18          # f = g + h (αγνοείται αν i = j)
L1 :  sub $16, $16, $19         # f = f - i  εκτελείται πάντοτε
```

2.7.1 Διασύνδεση υλικού / λογισμικού

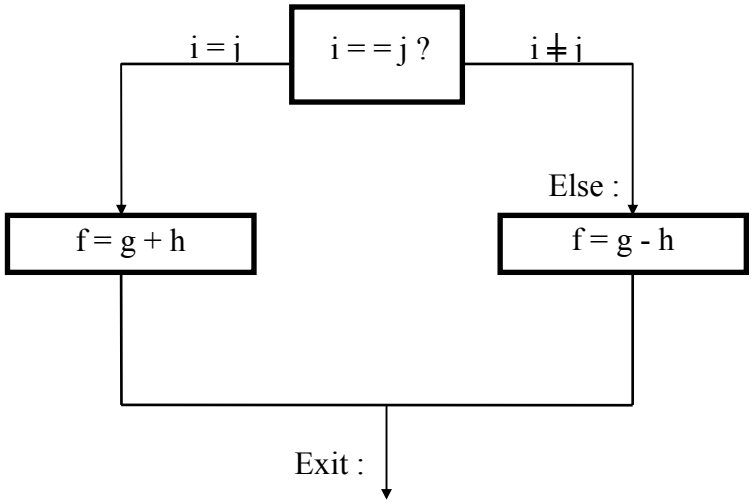
Οι μεταγλωττιστές συχνά δημιουργούν μεταπηδήσεις και ετικέτες που δεν παρουσιάζονται στις γλώσσες υψηλού επιπέδου. Για παράδειγμα, χρησιμοποιώντας τις μεταβλητές και τους καταχωρητές του πιο πάνω παραδείγματος, ο πιο κάτω κώδικας σε C:

```
if (i= =j)  f=g+h;   else  f=g-h;
```

μεταγλωττίζεται στον πιο κάτω κώδικα MIPS:

```
        bne    $19, $20, Else        # goto   Else  αν i ≠ j
        add   $16, $17, $18          #  f = g + h ( αγνοείται αν   i ≠ j )
        j  Exit                      # go to Exit
Else :  sub $16, $17, $18            #   f = g - h  ( αγνοείται αν   i = j )
Exit :
```

Το Σχήμα 2.7 δείχνει τη μορφή του κώδικα στη C, που πρέπει να μεταφραστεί στον κώδικα του MIPS. Η δεύτερη εντολή πιο πάνω, εκτελεί το "then" μέρος της εντολής "if" και η τέταρτη εντολή το "else" μέρος. Για αποφυγή εκτέλεσης του Else, όταν το i = j, χρησιμοποιούμε την εντολή **j Exit** που μεταφέρει την εκτέλεση του προγράμματος στην ετικέτα Exit, χωρίς συνθήκη (unconditional branch). Το **j** είναι το αρχικό γράμματα της λέξης **jump**.



Σχήμα 2.7: Οι επιλογές της πιο πάνω if εντολής .

Παράδειγμα:

Δίνεται ο ακόλουθος βρόγχος σε C :

```
Loop :  g = g + A[i]
            i = i + j;
if (i != h) goto Loop;
```

Υποθέστε ότι ο πίνακας Α έχει 100 στοιχεία και ότι ο μεταγλωττιστής συσχετίζει τις μεταβλητές g, h, i, και j με τους καταχωρητές $17, $18, $19 και $20. Επίσης ο πίνακας Α ξεκινά από το Astart. Ο δείκτης του πίνακα i, πρέπει να πολλαπλασιάζεται επί 4 και το 4 βρίσκεται στον καταχωρητή $10. Ποιος θα είναι ο κώδικας σε MIPS;

Απάντηση:

```
Loop :  mul  $9, $19, $10      # Προσωρινός καταχωρητής $9 = i x 4,
                               # υπολογισμός δείκτη
        lw  $8, Astart ($9)    # Προσωρινός καταχωρητής $8 = A[i]
        add  $17, $17, $8      # g=g+A[i]
        add  $19, $19, $20     # i = i+j
        bne  $19, $18, Loop    # Πήγαινε στο Loop αν  i ≠ h
```

Ο έλεγχος για ισότητα ή ανισότητα είναι ίσως ο πιο δημοφιλής έλεγχος, αλλά αρκετές φορές, θέλουμε να ελέγξουμε αν μια μεταβλητή, είναι πιο μικρή από μια άλλη μεταβλητή. Η εντολή **set on less than (slt)**, μας επιτρέπει να κάνουμε αυτήν την σύγκριση, π.χ.

slt $8, $19, $20

ο καταχωρητή $8 παίρνει την τιμή 1, αν η τιμή στον καταχωρητή $19 είναι πιο μικρή από την τιμή στον καταχωρητή $20, διαφορετικά ο καταχωρητής $8 παίρνει την τιμή 0.

Στον MIPS, οι εντολές slt, beq, bne, και η τιμή 0 στον καταχωρητή $0 χρησιμοποιούνται για τη δημιουργία όλων των σχετικών ελέγχων. Για παράδειγμα, για να ελέγξουμε αν η μεταβλητή στον καταχωρητή $16 είναι πιο μικρή από την τιμή στον καταχωρητή $17, εκτελούμε τις ακόλουθες εντολές:

```
slt    $1, $16, $17          # S1=1  if  $16 < $17
bne  $1, $0, Less            # goto   Less  if  $1 ≠ $0.
```

Το ζευγάρι των εντολών slt και bne εκτελούν την συνθήκη ελέγχου για το μικρότερο. Στην πραγματικότητα ο συμβολόμεταφραστής του MIPS, συμβολομεταφράζει την εντολή **blt (branch on less than)** στις δύο πιο πάνω εντολές. **Προσέξτε πώς δημιουργούμε εντολές στο επίπεδο του συμβολομεταφραστή που δεν υποστηρίζονται από την μηχανή.** Ο λόγος που η μηχανή MIPS δεν υποστηρίζει την εντολή blt, είναι διότι θα προκαλούσε επιπρόσθετη πολυπλοκότητα στη μηχανή, είτε θα μεγάλωνε ο κύκλος μηχανής είτε η εντολή θα χρειαζόταν επιπρόσθετους κύκλους μηχανής για κάθε εντολή. Δύο γρήγορες εντολές είναι πολύ πιο χρήσιμες. **Το μικρότερο είναι το πιο γρήγορο (ΑΡΧΗ 2).**

Σημειώστε ότι ο καταχωρητής $1 χρησιμοποιείται από τον συμβολομεταφραστή. Για αυτό το λόγο αποφεύγεται να χρησιμοποιείται από τους μεταγλωττιστές της γλώσσας MIPS.

Αρκετές γλώσσες προγραμματισμού έχουν εντολές case ή switch. Για την υποστήριξη τέτοιων εντολών, ο MIPS έχει την εντολή **jr (jump register)**, η οποία εκτελεί μεταπήδηση χωρίς συνθήκη στην διεύθυνση (αντίστοιχη ετικέτα) που δίνεται στον καταχωρητή.

Παράδειγμα:

Δίνεται ο πιο κάτω κώδικας σε C, ο οποίος χρησιμοποιεί την εντολή επιλογής switch για να επιλέξει μια συγκεκριμένη λειτουργία ανάλογα με την τιμή της μεταβλητής k. Υποθέστε ότι οι μεταβλητές f, g, h, i, j, k συσχετίζονται με τους καταχωρητές από $16 ως $21 και επίσης ο καταχωρητής $10 περιέχει τον αριθμό 4.

Ποιος θα είναι ο αντίστοιχος κώδικας σε MIPS;

```
switch (k)
    {
        case 0:   f = i + j;   break;     /*  k = 0  */
        case 1:   f = g + h; break;       /*  k =1   */
        case 2:   f = g - h;  break;      /*  k =2  */
        case 3:   f = i - j;   break;     /*  k = 3 */
    }
```

Απάντηση:

Το πρόγραμμα σε γλώσσα MIPS αρχικά θα προμηθεύσει τέσσερις λέξεις στη μνήμη, ξεκινώντας από τη θέση JumpTable και επίσης θα έχει διευθύνσεις που θα ανταποκρίνονται στις ταμπέλες L0, L1, L2, L3. Τον πίνακα τον έχει ορίσει ο μεταγλωττιστής. Κάθε εντολή είναι 4 bytes άρα ξέρουμε τις διευθύνσεις L0,L1,L2,L3.

```
Loop: mult $9, $10, $21          # Temporary register $9 = k  x 4
      lw  $8, JumpTable($9)      # Temporary register $8 = JumpTable[k]
      jr    $8                   #  Jump based on register $8
L0:   add  $16, $19, $20         #  k = 0 so f gets i + j
      j     Exit                 #  end of this case so goto Exit
L1:   add  $16, $17, $18         #  k = 1 so f gets g + h
      j     Exit               #  end of this case so goto Exit
L2:   sub $16, $17, $18          #  k = 2 so f gets g - h
      j     Exit                 #  end of this case so goto Exit
L3:   sub $16, $19, $20          #  k = 3 so f gets i - j
Exit:                            # end of switch statement
```

2.8 Υποστήριξη διαδικασιών σε επίπεδο υλικού

Μία διαδικασία ή υποπρόγραμμα (subroutine) είναι ένας τρόπος με τον οποίο οι προγραμματιστές χωρίζουν τα προγράμματα τους για να κάνουν πιο εύκολη την κατανόηση του κώδικα και για να επιτρέψουν στον κώδικα να ξαναχρησιμοποιηθεί (reusability). Μια ομάδα εντολών πρέπει να παρέχει ένα τρόπο μεταφοράς της ροής εκτέλεσης του προγράμματος σε μια διαδικασία και ακολούθως να επιτρέπει την επιστροφή στην επόμενη εντολή (αμέσως μετά από την εντολή που κάλεσε τη διαδικασία). Οι προγραμματιστές ακόμη πρέπει να έχουν τρόπο για το πέρασμα των παραμέτρων και επίσης να μπορούν να υποστηρίζουν το φώλιασμα των διαδικασιών, δηλαδή το κάλεσμα και άλλων διαδικασιών από μια διαδικασία.

Η αρχιτεκτονική MIPS υποστηρίζει μίαν εντολή που μεταπηδά σε μία διεύθυνση, και ταυτόχρονα αποθηκεύει τη διεύθυνση της εντολής που ακολουθεί στον καταχωρητή $31. Αυτή η εντολή ονομάζεται, **jump - and - link (jal)** και έχει ως ακολούθως:

jal ProcedureAddress

Το link τμήμα του ονόματος, σημαίνει ότι δημιουργείται η σύνδεση (link) με το κάλεσμα της διαδικασίας το οποίο ακολούθως θα επιτρέπει στη διαδικασία να επιστρέψει στη σωστή διεύθυνση. Αυτό το link που φυλάγεται στον καταχωρητή $31, ονομάζεται **διεύθυνση επιστροφής (return address)**. Έχουμε ήδη μιλήσει για την εντολή που μεταπηδά στην επιστροφή: **jr $31**

Υπάρχει ανάγκη για ένα καταχωρητή ο οποίος θα αποθηκεύει (κρατεί) την διεύθυνση της εντολής που εκτελείται. Η εντολή jal αυξάνει αυτό τον καταχωρητή, για να δείχνει στην επόμενη εντολή, προτού φυλαχτεί στον καταχωρητή $31. Η εντολή jr $31 απλά αντιγράφει το $31 σε αυτό τον καταχωρητή. Για ιστορικούς λόγους αυτός ο καταχωρητής ονομάζεται **program counter (PC)**, ή instruction address register.

Προσοχή: Στην περίπτωση που έχουμε κάνει jal σε εντολή την οποία ακολουθούν jal άλλων εντολών και δεν έχουμε επιστρέψει στην επόμενη εντολή που φυλάχτηκε στον καταχωρητή $ra, τότε είναι απαραίτητο να φυλαχτεί η αρχική τιμή του $ra πριν να εκτελέσουμε τα jal που ακολουθούν.

Υποθέστε ότι μια διαδικασία πρέπει να καλέσει μιαν άλλη διαδικασία. Ο ιδανικός τρόπος για επίτευξη αυτής της λειτουργίας είναι η χρησιμοποίηση της στοίβας stack. Χρειάζεται ένας δείκτης που θα δείχνει στην κορυφή της στοίβας, δηλαδή εκεί που η επόμενη εντολή πρέπει να τοποθετήσει τους καταχωρητές της. Η τοποθέτηση δεδομένων στη στοίβα ονομάζεται **push**, ενώ η μετακίνηση τους από την στοίβα **pop**.

Επιπρόσθετα στη διεύθυνση επιστροφής χρειαζόμαστε μεταφορά των δεδομένων των μεταβλητών ή των παραμέτρων σε μία διαδικασία. Στην αρχιτεκτονική MIPS η μέθοδος αυτή γίνεται με την τοποθέτηση των παραμέτρων στους καταχωρητές $4 έως $7. Αν μία διαδικασία χρειάζεται να καλέσει μιαν άλλη διαδικασία, αυτοί οι καταχωρητές μπορούν να φυλαχτούν και να καλεστούν στη συνέχεια, χρησιμοποιώντας τη στοίβα. Γενικώς μία διαδικασία, μπορεί να αλλάξει τους καταχωρητές που χρησιμοποιούνται από την τρέχουσα διαδικασία. Υπάρχουν δύο τυποποιημένες πρακτικές για αυτό:

1. **Αποθηκεύει αυτός που καλεί (caller save)**. Η διαδικασία που καλεί είναι υπεύθυνη για το φύλαγμα και την επαναφορά των καταχωρητών, οι οποίοι πρέπει να φυλαχτούν κατά την διάρκεια της εκτέλεσης της διαδικασίας. Τότε η διαδικασία που καλείται μπορεί να αλλάξει οποιοδήποτε καταχωρητή.

2. **Αποθηκεύει αυτός που καλείται (called save)**. Φυλάει και επαναφέρει αυτός που καλείται. Η διαδικασία που καλείται φυλάει τους καταχωρητές, τους χρησιμοποιεί όπως θέλει και επαναφέρει τους αρχικούς στο τέλος του καλέσματος.

2.9 Άλλα είδη διευθυνσιοδότησης στη μηχανή MIPS

Η αρχιτεκτονική MIPS υποστηρίζει δύο τρόπους για πρόσβαση στους τελεσταίους. Ο πρώτος τρόπος κάνει πιο γρήγορη την πρόσβαση σε μικρές σταθερές και ο δεύτερος τρόπος κάνει την διακλάδωση ή μεταπήδησης πιο αποδοτική.

2.9.1 Σταθεροί ή Άμεσοι Τελεσταίοι

Πολλές φορές χρησιμοποιούμε σταθερούς αριθμούς σε μια λειτουργία (μιαν εντολή). Για παράδειγμα, όταν αυξάνουμε τον δείκτη να δείχνει στο επόμενο στοιχείο σε ένα πίνακα, ή όταν μετρούμε τις επαναλήψεις σε ένα loop, ή όταν μετατρέπουμε την στοίβα σε ένα φωλιασμένο call. Στην πραγματικότητα, σε δύο προγράμματα που έχουν μελετηθεί: στο μεταγλωττιστή της C, gcc και στο πρόγραμμα για σχεδίαση ηλεκτρονικών κυκλωμάτων spice, περισσότερο από 50% των αριθμητικών λειτουργιών διεξάγεται σε σταθερούς τελεσταίους.

Με τις εντολές που έχουμε μελετήσει μέχρι τώρα, για να προσθέσουμε τον αριθμό 4 στον καταχωρητή $29 θα χρειαστούμε τις ακόλουθες δύο εντολές:

```
lw    $24, AddrConstant4 ($0)      #  $24 = 4
add  $29, $29, $24                 #  $29=$29 + $24
```

Δεδομένου ότι η διεύθυνση AddrConstant4 είναι η διεύθυνση της μνήμης που έχει τη σταθερά 4.

Μία εναλλακτική λύση είναι η δημιουργία νέων αριθμητικών εντολών, με πρόσβαση στη μνήμη όπου ένας τελεσταίος είναι μία σταθερά, η οποία φυλάγεται μέσα στην ίδια την εντολή. Ακολουθώντας την αρχή για της χρήσης της κανονικοποίησης, χρησιμοποιούμε την ίδια μορφή εντολών όπως και για την μεταφορά δεδομένων. Η μορφή εντολών **I-type** είναι για **Immediate-type** εντολές (δηλ. εντολές στις οποίες ένας από τους τελεσταίους τους είναι σταθερά). Το πεδίο του MIPS στο οποίο αποθηκεύεται η σταθερά έχει μέγεθος 16 bits.

Παράδειγμα:

Η εντολή πρόσθεσης - add η οποία έχει ένα σταθερό τελεσταίο ονομάζεται add immediate ή addi. Για να προσθέσουμε τον αριθμό 4 στον καταχωρητή 29 γράφουμε απλά:

```
addi  $29, $29, 4          # $29 = $29 + 4
```

Ποίος είναι ο αντίστοιχος κώδικας στη γλώσσα μηχανής MIPS;

Απάντηση

Ο κώδικας μηχανής για τη πιο πάνω μηχανή είναι:

op	*Rs*	*Rt*	*immediate*
8	*29*	*29*	*4*

(α) Χρησιμοποιώντας δεκαδικούς αριθμούς

001000	*11101*	*11101*	*0000 0000 0000 0100*

(β) Χρησιμοποιώντας δυαδικούς αριθμούς

Οι τελεσταίοι άμεσης πρόσβασης (immediate) είναι επίσης πολύ χρήσιμοι για τις συγκρίσεις. Όπως έχουμε τονίσει ο καταχωρητής $0 έχει πάντα την τιμή 0, έτσι μπορούμε να συγκρίνουμε με το μηδέν. Για σύγκριση με άλλες τιμές, μπορούμε να χρησιμοποιήσουμε την

εντολή **stli** (set on less than immediate), για παράδειγμα στις πιο κάτω εντολές θέλουμε να συγκρίνουμε αν το περιεχόμενο του καταχωρητή 18 είναι μικρότερο από το 10 :

```
slti   $8, $18, 10      # $8 = 1      if  $18<10
bne  $8, $0, Less       # goto  Less  if  $18<10
```

Η τελευταία αρχή στη σχεδίαση υλικού είναι:

Αρχή 4: Τις πιο συχνές περιπτώσεις τις σχεδιάζουμε έτσι ώστε να εκτελούνται γρήγορα.

Η χρήση σταθερών τελεσταιων είναι πολύ συχνή, έτσι χρησιμοποιώντας τις σταθερές ως μέρος των αριθμητικών εντολών γίνεται πολύ πιο γρήγορη η εκτέλεση τους παρά αν φορτώνονταν από την μνήμη.

Αν και οι περισσότερες σταθερές χωράνε στο πεδίο των 16bits, μερικές φορές μπορεί να έχουμε και πιο μεγάλες σταθερές. Η εντολή του MIPS **load upper immediate (lui)**, μεταφέρει τα 16 bits της σταθερά στα πιο ψηλά 16 bits του καταχωρητή, αφήνοντας τα χαμηλότερα 16 bits να προσδιορισθούν από μία άλλη εντολή, όπως φαίνεται στο Σχήμα 3.9α.

Ο κώδικας σε γλώσσα μηχανής για την εντολή lui $8, 255 :

001111	00000	01000	0000 0000 1111 1111

Τα περιεχόμενα του καταχωρητή 8 μετά από την εκτέλεση της εντολής lui:

0000 0000 1111 1111	0000 0000 0000 0000

Σχήμα 4.9α: *Το αποτέλεσμα της εντολής lui. Η εντολή lui μεταφέρει τα 16 δεξιότερα bits στα αριστερότερα 16 bits του καταχωρητή, γεμίζοντας τα χαμηλότερα 16 bits με 0.*

Παράδειγμα :

Ποίος θα είναι ο κώδικας στη συμβολική γλώσσα MIPS για να φορτώσουμε την πιο κάτω 32-bit σταθερά στο καταχωρητή $16:

0000 0000 0011 1101 0000 1001 0000 0000

Απάντηση:

Πρώτα θα φορτώσουμε τα ψηλότερα 16 bits (είναι ο αριθμός 61 στο δεκαδικό), χρησιμοποιώντας την εντολή lui:

```
lui  $16, 61  # 61 decimal =  0000  0000  0011  1101   in  binary
```

Η τιμή στο καταχωρητή 16 θα είναι τελικά:

0000 0000 0011 1101 0000 0000 0000 0000

Το επόμενο βήμα είναι να προσθέσουμε τα χαμηλότερα 16 bits, τους οποίους η δεκαδική τιμή είναι 2304:

addi $16, $16, 2304 # 2304 decimal = 0000 1001 0000 0000

Η τελική τιμή στο καταχωρητή $16 θα είναι η επιθυμητή τιμή, δηλαδή:

0000 0000 0011 1101 0000 1001 0000 0000

2.9.2 Διασύνδεση λογισμικού/υλικού

Ο μεταγλωττιστής ή ο συμβολομεταφραστής πρέπει να σπάσει τις μεγάλες σταθερές σε μικρές και μετά να τις ενώσει μαζί σε ένα καταχωρητή. Στον MIPS γι' αυτό τον σκοπό χρησιμοποιείται ο καταχωρητής **$1**.

2.10 Διευθυνσιοδότηση με διακλάδωση

Η πιο απλή διευθυνσιοδότηση στον MIPS είναι η εντολές jump. Χρησιμοποιείται η τρίτη και τελευταία μορφή στον MIPS, η **J-type**, η οποία αποτελείται από 6 bits για το πεδίο op (λειτουργίας) και τα υπόλοιπα 26 bits είναι για το πεδίο των διευθύνσεων.

Επομένως η εντολή:

j 10000 # goto location 10000

Συμβολομεταφράζεται στη πιο κάτω μορφή:

2	10000
6 bits	26 bits

Η διακλάδωση υπό συνθήκη, πρέπει να προσδιορίσει δύο επιπρόσθετους τελεσταίους στη διεύθυνση διακλάδωσης. Επομένως:

bne $8, $21, Exit # goto Exit if $8 = $21

Συμβολομεταφράζεται στη πιο κάτω μορφή, αφήνοντας μόνο 16 bits για τη διεύθυνση διακλάδωσης:

5	8	21	Exit
6 bits	5 bits	5 bits	16 bits

Αν οι διευθύνσεις ενός προγράμματος έπρεπε να χωράνε σε ένα πεδίο 16 bits, αυτό θα σήμαινε ότι κανένα πρόγραμμα δεν θα μπορούσε να είναι μεγαλύτερο από 2^{16} bits, το οποίο είναι μη ρεαλιστικό. Μία εναλλακτική λύση είναι ο προσδιορισμός ενός καταχωρητή που θα προσθέτεται πάντοτε στη διεύθυνση διακλάδωσης, ούτως ώστε η εντολή διακλάδωσης θα υπολογίζει το ακόλουθο:

PC = register + branch address

Αυτό επιτρέπει σε ένα πρόγραμμα να έχει μέγεθος μέχρι 2^{32} και να μπορεί να χρησιμοποιεί διακλάδωση υπό συνθήκη, λύνοντας έτσι το πρόβλημα του μεγέθους της διεύθυνσης. Το ερώτημα τότε είναι, ποιος καταχωρητής θα χρησιμοποιηθεί;

Η απάντηση δίνεται μελετώντας πώς λειτουργούν οι εντολές υπό συνθήκη. Οι εντολές υπό συνθήκη συναντώνται σε εντολές if και τείνουν να προκαλούν διακλάδωση σε κοντινές εντολές. Για παράδειγμα σχεδόν οι μισές εντολές διακλάδωσης υπό συνθήκη στα προγράμματα πηγαίνουν (μεταπηδούν) σε λιγότερο από 16 εντολές από το σημείο (εντολή) που ελέγχθηκε η συνθήκη. Δεδομένου ότι ο PC περιέχει την διεύθυνση της τρέχουσας εντολής, μπορούμε να έχουμε διακλάδωση μέσα σε 2^{16} από την τρέχουσα εντολή χρησιμοποιώντας τον καταχωρητή PC για να προστεθεί η διεύθυνση διακλάδωσης.

Σχεδόν όλες οι εντολές loap και if χρειάζονται πιο μικρή διεύθυνση από 2^{16} bits, επομένως η χρήση του καταχωρητή PC είναι η ιδανική λύση. Αυτό το είδος διευθυνσιοδότησης ονομάζεται **PC-relative addressing**. Όπως θα μελετήσουμε και στο Κεφάλαιο 5, το υλικό αυξάνει τον καταχωρητή PC στα αρχικά στάδια της εντολής για να δείχνει στην επόμενη εντολή. Επομένως η επόμενη διεύθυνση στη μηχανή MIPS θα είναι στη θέση PC+4 αν η προηγούμενη ήταν στη θέση που δεικνύει ο PC.

Επομένως οι μηχανές MIPS μπορούν να χρησιμοποιήσουν PC-relative addressing, καθώς επίσης και jump-and-link εντολές, για μακρινές διευθύνσεις, όπως είναι η μορφή j-type που έχουμε μελετήσει.

Παράδειγμα :
Δίνεται ο πιο κάτω κώδικας σε MIPS που αντιπροσωπεύει ένα while loop σε μια γλώσσα ψηλού επιπέδου:

```
Loop :  Mult  $9, $19, $10    # Προσωρινός καταχωρητής  $9 = i x 4
        lw    $8, Sstart ($9) # Προσωρινός καταχωρητής  $8 = save [i]
        bne   $8, $21, Exit   # go to Exit if save [i] ≠ k
        add   $19, $19, $20   # i=i+j
        j     Loop            # goto Loop
Exit:
```

Αν υποθέσουμε ότι ο πιο πάνω βρόγχος είναι αποθηκευμένος στη θέση 80000 στη μνήμη και ότι η διεύθυνση Sstart αναφέρεται στη θέση 1000, ποια θα είναι η κωδικοποίηση στη γλώσσα μηχανής για αυτόν τον βρόγχο;

Απάντηση:
Οι μεταφρασμένες εντολές και οι διευθύνσεις τους θα έχουν την μορφή:

80000	*0*	*19*	*10*	*9*	*0*	*24*
80004	*35*	*9*	*8*	*1000*		
80008	*5*	*8*	*21*	*8*		
80012	*0*	*19*	*20*	*19*	*0*	*32*
80016	*2*	*80000*				
80020	. . .					

Οι διευθύνσεις συνεχόμενων λέξεων στη γλώσσα MIPS διαφέρουν κατά 4 bytes. Η εντολή bne στην τρίτη γραμμή προσθέτει 8 bytes στη διεύθυνση της επόμενης εντολής (80012), προσδιορίζοντας τον προορισμό της διακλάδωσης σε σχέση με αυτή την εντολή αντί να χρησιμοποιήσει ολόκληρη τη διεύθυνση (80020). Η εντολή jump χρησιμοποιεί ολόκληρη την διεύθυνση (80000), που ανταποκρίνεται στην ετικέτα Loop.

2.10.1 Διασύνδεση λογισμικού / υλικού

Αν και σχεδόν όλες οι διακλαδώσεις υπό συνθήκη είναι σε κοντινές διευθύνσεις, υπάρχει ανάγκη για διακλάδωση και σε μακρινές διευθύνσεις. Για παράδειγμα η εντολή:

beq $18, $19, L1

μπορεί να αντικατασταθεί με δύο εντολές, ως ακολούθως, για να επιτευχθεί η διακλάδωση στη μακρινή διεύθυνση L1:

```
     bne $18, $19, L2
     j     L1
L2:
```

Έχουμε μελετήσει κάποιες νέες μορφές διευθυνσιοδότησης σε αυτήν την ενότητα, οι οποίες γενικώς ονομάζονται **addressing modes**. Στον MIPS αυτά τα addressing modes είναι τα ακόλουθα:

1. **Register addressing,** όπου ο τελεσταίος είναι καταχωρητής.

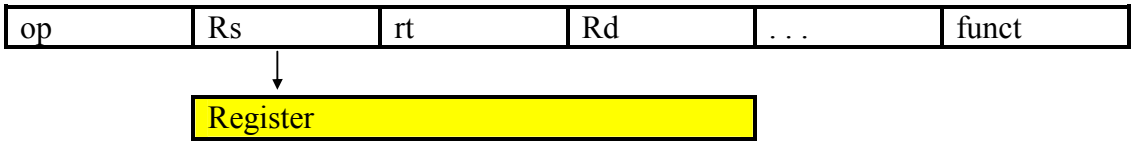
2. **Base or displacement addressing**, όπου ο τελεσταίος είναι τοποθετημένος στην μνήμη, του οποίου η διεύθυνση είναι το άθροισμα του καταχωρητή και της διεύθυνσης της εντολής.

3. **Immediate addressing**, όπου ο τελεσταίος είναι μία σταθερά μέσα στην εντολή .

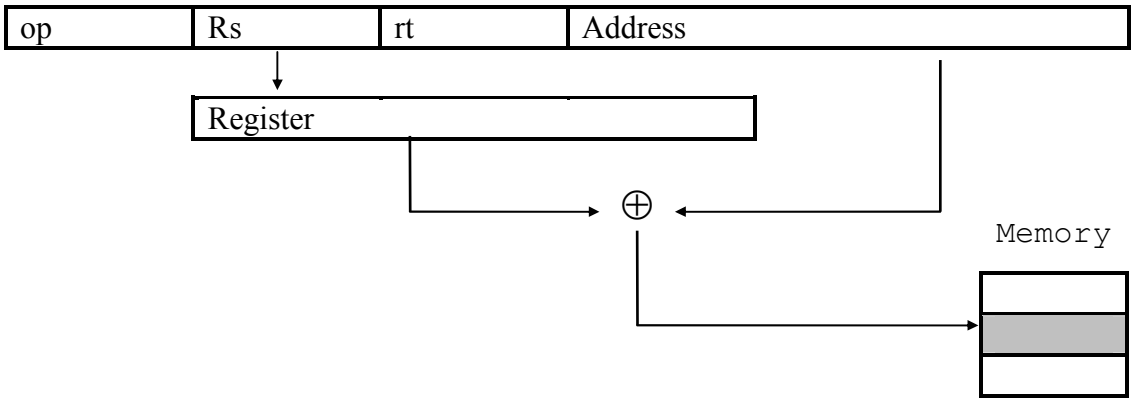
4. **PC-relative addressing**, όπου η διεύθυνση είναι το άθροισμα του καταχωρητή PC και της σταθεράς μέσα στην εντολή.

Το Σχήμα 2.8 δείχνει τον τρόπο προσδιορισμού των τελεσταίων για κάθε μορφή διευθυνσιοδότησης.

1. Register Addressing (add, sub, κ.α.)

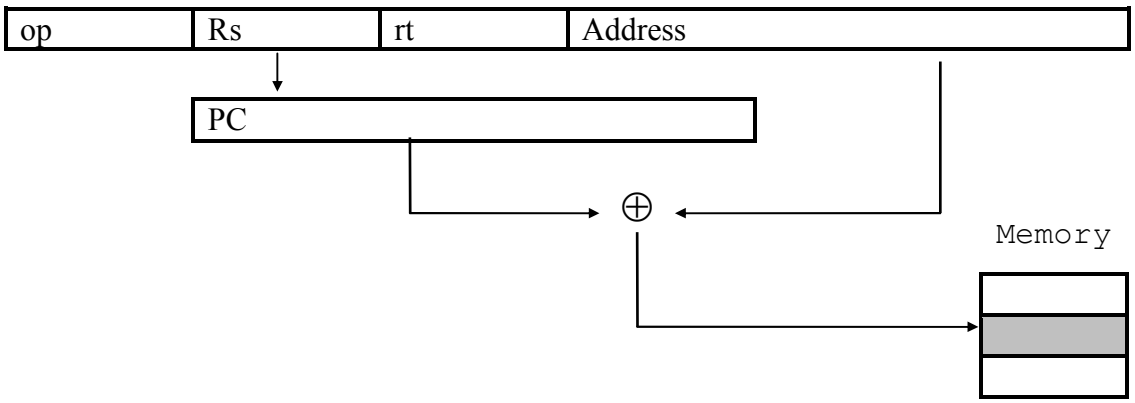


2. Base Addressing (lw,sw, κ.α.)



3. Immediate Addressing (addi, subi, κ.α.)

4. PC - Relative Addressing



Σχήμα 2.8: Οι τέσσερις μορφές διευθυνσιοδότησης στον MIPS. Οι τελεσταίοι είναι σκιασμένοι. Ο τελεσταίος στη δεύτερη και τέταρτη μορφή διευθυνσιοδότησης είναι θέση στη μνήμη, στη πρώτη μορφή είναι καταχωρητής και στη τρίτη μορφή είναι ένας 16 bits αριθμός.

- Στο Σχήμα 2.9 φαίνονται όλες οι εντολές της αρχιτεκτονικής MIPS που μελετήσαμε στο Κεφάλαιο 2.

MIPS operands

Name	Example	Comments
32 registers	$0, $1, $2, ..., $31	Fast location for data. In MIPS, data must be in registers to perform arithmetic. MIPS register $0 always equals 0. Register $1 is reserved for the assembler to handle pseudo-instructions and large constants.
2^{30} memory words	Memory[0], Memory[4], Memory[4292967292]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as arrays, and spilled registers, such as those saved on procedure calls.

MIPS assembly language

Category	Instructions	Example	Meaning	Comments
Arithmetic	Add	add $1, $2, $3	$1 = $2 + $3	3 operands; data in registers
	Subtract	sub $1, $2, $3	$1 = $2 - $3	3 operands; data in registers
	Add immediate	addi $1, $2, 100	$1 = $2 + 100	Used to add constants
Data Transfer	load word	lw $1, 100($2)	$1 = Memory [$2+100]	Data from memory to register
	store word	sw $1, 100($2)	Memory [$2+100] = $1	Data from register to memory
	load upper immediate	lui $1, 100	$1 = 100 * 2^{16}	Loads constants in upper 16 bits
Conditional Branch	branch on equal	beq $1, $2, 100	if ($1 == $2) go to PC + 4 + 100	Equal test; PC relative branch
	branch on not equal	bne $1, $2, 100	if ($1 != $2) go to PC + 4 + 100	Not equal test; PC relative
	set on less than	slt $1, $2, $3	if ($2 < $3) $1 = 1; else $1 = 0	Compare less than; for beq, bne
	set less than immediate	slti $1, $2, 100	if ($2 < 100) $1=1; else $1 = 0	Compare less than constant
Unconditional Jump	Jump	j 10000	go to 10000	Jump to target address
	Jump register	jr $31	go to $31	For switch, procedure return
	jump and link	jal 10000	$31 = PC + 4; go to 10000	For procedure call

MIPS machine language

Name	Format	Example						Comments
Add	R	0	2	3	1	0	32	add $1, $2, $3
Sub	R	0	2	3	1	0	34	sub $1, $2, $3
Addi	I	8	2	1	100			addi $1, $2, 100
Lw	I	35	2	1	100			lw $1, 100($2)
Sw	I	43	2	1	100			sw $1, 100($2)
Lui	I	15	0	1	100			lui $1, 100
Beq	I	4	1	2	100			beq $1, $2, 100
Dne	I	5	1	2	100			bne $1, $2, 100
Slt	R	0	2	3	1	0	42	slt $1, $2, $3
Slti	I	10	2	1	100			slti $1, $2, 100
J	J	2	10000					j 10000
Jr	R	0	31	0	0	0	8	jr $31
Jal	J	3	10000					jal 10000
Field size		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions 32 bits
Format R	R	op	rs	rt	rd	Shamt	funct	Arithmetic instruction format
Format I	I	op	rs	rt	address / immediate			Transfer, branch, imm. format
Format J	J	op	Target address					Jump instruction format

Σχήμα 2.9: Όλες οι εντολές της αρχιτεκτονικής MIPS που μελετήθηκαν σε αυτό το κεφάλαιο.

2.11 Άλλες λύσεις στη προσέγγιση της μηχανής MIPS

Οι σχεδιαστές του συνόλου εντολών μιας μηχανής συνήθως προσπαθούν να μειώσουν τον αριθμό των εντολών που εκτελούνται από ένα πρόγραμμα. Αυτή η προσπάθεια όμως μπορεί να επηρεάσει την απλότητα της μηχανής και να αυξήσει το χρόνο εκτέλεσης των προγραμμάτων επειδή οι εντολές θα είναι πιο αργές. Πιο κάτω θα παρουσιαστούν μερικές μέθοδοι μείωσης του αριθμού των εντολών που εκτελούνται χρησιμοποιώντας πιο δυναμικούς τρόπους πρόσβασης στους τελεστές.

2.11.1 Αυτόματη Αύξηση και Αυτόματη Μείωση

- Ένα εύχρηστο ζεύγος εντολών είναι η φόρτωση μιας λέξης και μετά η αύξηση της τιμής του καταχωρητή για να δείχνει στην επόμενη εντολή.

- Η ιδέα της αυτόματης αύξησης βασίζεται στην ύπαρξη μιας εντολής που θα αυξάνει αυτόματα τον καταχωρητή για να δείχνει στην επόμενη εντολή. Η νέα μορφή θα είναι ισοδύναμη με το ακόλουθο ζεύγος εντολών:

```
lw     $8, Sstart ($19)        # register $8 gets S[$19]
addi $19, $19, 4               #  $19 = $19 + 4
```

Οι πιο πάνω εντολές θα μπορούσαν να αντικατασταθούν με την πιο κάτω υποθετική εντολή (που δεν βρίσκεται στον MIPS):

```
lw+    $8, Sstart ($19)   # register $8 = S[$19] ;  $19 = $19 + 4
```

Το ίδιο μπορεί να γίνει και για την αυτόματη μείωση.

Τελεσταίοι εντολών βασισμένοι στη μνήμη

Μια άλλη προσπάθεια για μείωση του αριθμού των εντολών που εκτελούνται είναι η αντικατάσταση του ζεύγους εντολών lw και μιας αριθμητικής εντολής με μια αριθμητική εντολή που θα μπορούσε ο ένας της τελεσταίος να κάνει αναφορά στη μνήμη. Για παράδειγμα:

```
lw     $8, Astart ($19)   # Temporary register $8 gets A [$19]
add   $16, $17, $8        # $16 = $17 + A [I]
```

Το πιο πάνω ζεύγος εντολών θα μπορούσε να αντικατασταθεί με την πιο κάτω υποθετική εντολή που δεν βρίσκεται στην αρχιτεκτονική του MIPS:

```
addm   $16, $17, Astart ($19) #$16 = $17 + Memory [$19 + Astart]
```

2.12 Παρεξηγημένες έννοιες και παγίδες

Παγίδα: " Γράφουμε στην συμβολική γλώσσα για να επιτύχουμε ψηλότερη απόδοση. "

Υπάρχει μια διαμάχη μεταξύ των μεταγλωττιστών και των συμβολομεταφραστών για το ποιος είναι πιο αποδοτικός. Οι μεταγλωττιστές κερδίζουν συνεχώς έδαφος, αν και η μάχη δεν χάθηκε ακόμα, το σίγουρο είναι ότι προγραμματίζοντας στη συμβολική γλώσσα αντιμετωπίζεις τους πιο κάτω κινδύνους:

- Σπαταλείται περισσότερος χρόνος για την δημιουργία του κώδικα και για την εύρεση των λαθών (coding and debugging).
- Χάνει στη φορητότητα (portability).
- Υπάρχουν δυσκολίες στη συντήρηση τέτοιων προγραμμάτων.

Είναι γενικώς αποδεκτό ότι ένας κώδικας με μεγαλύτερο αριθμό γραμμών χρειάζεται περισσότερο χρόνο για να εκτελεστεί. Σίγουρα η συμβολική γλώσσα χρειάζεται πολύ περισσότερες γραμμές κώδικα από ότι η C. Επίσης η συντήρηση αλλά και η αναβάθμιση των προγραμμάτων που είναι γραμμένα σε συμβολική γλώσσα είναι χρονοβόρα και επίπονη εργασία. Όταν γράφεις σε γλώσσα ψηλού επιπέδου επιτρέπεις στους μελλοντικούς μεταγλωττιστές να προσαρμόσουν τον κώδικα για μελλοντικές μηχανές, επιπλέον η συντήρηση του λογισμικού είναι πολύ πιο εύκολη, και το πρόγραμμα μπορεί να τρέξει σε περισσότερο από ένα είδος υπολογιστή.

2.13 Συμπεράσματα

Οι τέσσερις αρχές που ακολουθούν οι σχεδιαστές του συνόλου εντολών μιας μηχανής είναι:

1. *Η ομοιομορφία επιφέρει απλότητα.*. Πολλά χαρακτηριστικά των εντολών της μηχανής MIPS οφείλονται στη ομοιομορφία, όπως: όλες οι εντολές έχουν το ίδιο μέγεθος, απαιτούνται πάντα τρεις καταχωρητές για τις αριθμητικές εντολές και τα πεδία των καταχωρητών κρατούνται στην ίδια θέση για όλα τα είδη εντολών.
2. *Το μικρότερο είναι πιο γρήγορο.* Ο λόγος που ο MIPS έχει μόνο 32 καταχωρητές για να έχει περισσότερη ταχύτητα.
3. *Η καλή σχεδίαση απαιτεί συμβιβασμούς.* Ένα παράδειγμα στον MIPS είναι ο συμβιβασμός μεταξύ της προσφοράς bytes για τις μεγαλύτερες διευθύνσεις και των σταθερών των εντολών και της απόφασης όλες οι εντολές να έχουν το ίδιο μέγεθος.
4. *Σχεδίασε τις πιο συχνές περιπτώσεις τις κάνουμε γρήγορες.* Για παράδειγμα, η χρησιμοποίηση της διευθυνσιοδότησης PC - relative addressing για τις εντολές εκλογής υπό συνθήκη (conditional branches) και οι εντολές της άμεσης διευθυνσιοδότησης (immediate addressing) για τους σταθερούς αριθμούς.

Κάθε ομάδα εντολών στον MIPS συσχετίζεται με δομές που μπορούμε να συναντήσουμε σε άλλες γλώσσες προγραμματισμού υψηλού επιπέδου:

- Οι αριθμητικές εντολές ανταποκρίνονται στις λειτουργίες που βρίσκονται στις εντολές ανάθεσης.
- Οι εντολές μεταφοράς δεδομένων είναι πιο πιθανό να χρησιμοποιηθούν όταν έχουμε δομές δεδομένων, π.χ. πίνακες.
- Οι εντολές διακλάδωσης υπό συνθήκη (conditional branches) χρησιμοποιούνται τις εντολές if και στους βρόγχους (loops).
- Οι εντολές διακλάδωσης χωρίς συνθήκη (unconditional jumps) χρησιμοποιούνται στο κάλεσμα διαδικασιών και στις επιστροφές (returns), και επίσης στις εντολές case/switch.

Μερικά σημαντικά πράγματα που πρέπει να ξέρετε για τον MIPS:

☑ Οι λέξεις (words) έχουν πλάτος 32 bits

☑ Υπάρχει ένα αρχείο καταχωρητών με 32 καταχωρητές

☑ Ο κάθε καταχωρητής έχει πλάτος 32 bits (μία λέξη)

☑ Υπάρχουν δύο ονοματολογίες για τους 32 καταχωρητές:
   - $0 - $31 (το σύμβολο «$» + το νούμερο του καταχωρητή)
   - Με συμβολικά ονόματα: $zero, $at, $t0,$t1 … $sp, $ra κλπ

   Εμείς προτιμάμε τον πρώτο τρόπο ονοματολογίας, αν και συχνά αναφερόμαστε και στο δεύτερο. Τα συμβολικά ονόματα και ο ρόλος τους αναφέρονται στο Παράρτημα Α του βιβλίου.

☑ Ο καταχωρητής $0 έχει <u>πάντα</u> τιμή 0 (μηδέν)

☑ Ο μετρητής προγράμματος (Program Counter => PC) είναι ένας ειδικός καταχωρητής ο οποίος έχει 32 bits. Ο καταχωρητής PC δείχνει στο σημείο στο οποίο βρίσκεται η εκτέλεση του προγράμματος κάθε δεδομένη χρονική στιγμή.

- ☑ Η πρόσβαση στη μνήμη γίνεται αποκλειστικά με εντολές load (φόρτωσης) και store (αποθήκευσης). Με μόνη εξαίρεση τις εντολές load και store, καμία άλλη εντολή δεν αναφέρεται στην μνήμη

- ☑ Η μνήμη είναι οργανωμένη σε λέξεις (των 32 bits).

- ☑ Οι διευθύνσεις μνήμης είναι διευθύνσεις byte (8 bits). Η μικρότερη διεύθυνση μνήμης (byte) δίνει το όνομά της σε όλη τη λέξη στην οποία περιέχεται το byte. Άρα, (εφόσον κάθε λέξη αποτελείται από 4 byte) οι διευθύνσεις λέξεων διαιρούνται ακριβώς με το 4 (4, 8, 16, 1024 κ.ο.κ.)

- ☑ Όλες οι αριθμητικές και λογικές πράξεις αποθηκεύουν το αποτέλεσμά τους σε ένα από τους 32 καταχωρητές.

- ☑ Η εκτέλεση των εντολών γίνεται στην σειρά, από μικρότερες διευθύνσεις προς μεγαλύτερες. Η σειρά αυτή 'διαταράσσεται' μόνο όταν καλέσουμε κάποια εντολή ελέγχου ροής (branch, ή jump κλπ)

- ☑ Οι εντολές έχουν πλάτος 32 bits (1 λέξη), και οι διευθύνσεις στις οποίες βρίσκονται είναι διευθύνσεις λέξεων Οι διευθύνσεις λέξεων είναι πολλαπλάσια του 4.

Βιβλιογραφία

Chapter 3 : Instructions: Language of the Machine, από το βιβλίο των David A. Patterson & John L. Hennessy, με τίτλο "Computer Organization and Design. The Hardware/Software Interface".

3. Αριθμητική για υπολογιστές

3.1 Εισαγωγή

Οι λέξεις στους ηλεκτρονικούς υπολογιστές αποτελούνται από bits. Επιπλέον οι λέξεις μπορούν να αναπαρασταθούν σαν δυαδικοί αριθμοί. Παρόλο που οι φυσικοί αριθμοί 0,1,2 μπορούν να αναπαρασταθούν είτε σε δεκαδική είτε σε δυαδική μορφή, τι γίνεται με τις περιπτώσεις των πιο κάτω αριθμών:

Πως αναπαριστώνται οι αρνητικοί αριθμοί ?

Ποιος είναι ο μεγαλύτερος αριθμός που μπορεί να αναπαρασταθεί με μια λέξη του υπολογιστή?

Τι γίνεται με τα κλάσματα και τους πραγματικούς αριθμούς ?

Οι βασικοί στόχοι αυτού του κεφαλαίου είναι να αποκαλύψει πως το υλικό πραγματικά προσθέτει, αφαιρεί, πολλαπλασιάζει ή διαιρεί αριθμούς. Παράλληλα περιλαμβάνει αναπαράσταση των αριθμών, αριθμητικούς αλγορίθμους, υλικό το οποίο υλοποιείται σύμφωνα με τους αλγορίθμους αυτούς και την υλοποίηση όλων αυτών σύνολο εντολών.

3.2 Αριθμητικά συστήματα

Υπάρχουν διάφορα συστήματα αρίθμησης που έχουν το καθένα τα πλεονεκτήματα και τα μειονοκτήματα τους. Τα μειονεκτήματα πηγάζουν κυρίως από τον τρόπο απεικόνισης μεγάλων αριθμών και τα λάθη που προκύπτουν κατά την εκτέλεση των πράξεων. Το δεκαδικό σύστημα αρίθμησης είναι το πιο "οικείο" αφού έχουμε συνηθίσει να εργαζόμαστε με αυτό. Η βάση του συστήματος είναι το 10, ενώ η πραγματική αξία ενός αριθμού που βρίσκεται στο δεκαδικό σύστημα διαμορφώνεται από τα 10 ψηφία (0,1,2,..,9) σε συνδυασμό με την θέση τους μέσα στον αριθμό : 3423(10) = 3*1000 + 4*100 + 2*10 + 3*1.

Παρόλο που στην καθημερινή μας ζωή το δεκαδικό σύστημα έχει κυριαρχήσει, δεν μπορεί να χρησιμοποιειθεί από τους Η/Υ για τον εξής απλό λόγο : Για να γίνει η αναπαράσταση των 10 διαφορετικών ψηφίων χρειαζόμαστε δέκα διαφορετικές καταστάσεις (10 διαφορετικά επίπεδα τάσης). Αυτό βέβαια είναι υλοποιήσιμο αλλά το κύκλωμα που θα δημιουργηθεί θα έχει μεγάλο βαθμό πολυπλοκότητας και αυξημένο κόστος οπότε πρέπει να χρησιμοποιηθεί ένα πιο απλό σύστημα αρίθμησης.Από τα παραπάνω πηγάζει η ανάγκη της υιοθέτησης του δυαδικού συστήματος αρίθμησης για την εσωτερική αναπαράσταση των δεδομένων στους Η/Υ.

Το δυαδικό σύστημα αρίθμησης έχει βάση το 2 και διαθέτει 2 ψηφία, το 0 και το 1. Τώρα χρειαζόμαστε μόνο 2 διαφορετικές καταστάσεις (2 επίπεδα τάσης) για να αναπαραστίσουμε τα ψηφία του δυαδικού συστήματος. Η πραγματική αξία ενός αριθμού που βρίσκεται στο δυαδικό σύστημα διαμορφώνεται από τα 2 ψηφία (0,1) σε συνδιασμό με την θέση τους μέσα στον αριθμό, ενώ κατα αντιστοιχία αντί για μονάδες, δεκάδες, εκαντοντάδες, ... που έχουμε στο δεκαδικό σύστημα αρίθμησης υπάρχουν δυνάμεις του 2 δηλαδή μονάδες, δυάδες, τετράδες, οκτάδες, ...

10110000(2) =1*27 + 0*26 + 1*25 + 1*24 + 0*23 + 0*22 + 0*21 + 0*20.

Σύστημα	Βάση	Σύμβολα
Δυαδικό	2	0,1
Τριαδικό	3	0,1,2
Οκταδικό	8	0,1,2,3,4,5,6,7
Δεκαδικό	10	0,1,2,3,4,5,6,7,8,9
Δωδεκαδικό	12	0,1,2,3,4,5,6,7,8,9,A,B
δεκαεξαδικό	16	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Σχήμα 3.1: Συστήματα αρίθμησης

3.3 Προσημασμένοι και Απρόσημοι Αριθμοί

Οι λέξεις στον MIPS είναι μεγέθους 32bits, έτσι μπορούμε να αναπαραστήσουμε 2^{32} διαφορετικούς 32bits συνδυασμούς, δηλαδή από 0 μέχρι 2^{32}.

3.3.1 Απρόσημη Αριθμητική

Αν έχουμε n bits η περιοχή των αριθμών είναι : 0 εως (2^n –1)

πχ Για n=8 0 εως 255
Για n=16 0 εως 65535
Για n=32 0 εως (2^32-1)

-2^31 εως +2^31 -1

Ετσι μπορούμε να αναπαραστήσουμε θετικούς ακέραιους από το 0 έως και το 2^{32}-1

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two}$= 0_{ten}
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two}$= 1_{ten}
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two}$= 2_{ten}

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two}$= $4,294,967,293_{ten}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two}$= $4,294,967,294_{ten}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two}$= $4,294,967,295_{ten}$

3.3.2 Προσημασμένη Αριθμητική

Για την αναπαράσταση αρνητικών αριθμών υπάρχουν διάφοροι τρόποι:

3.3.2.1 Μέτρο πρόσημο

Στην αναπαράσταση αυτή το πιο σημαντικό Bit (MSB) δεσμεύεται για την αναπαράσταση του πρόσημου ενώ τα υπόλοιπα bits αναπαριστούν το μέτρο του αριθμού.

Παράδειγμα:

+16 => 010000

-16 => 110000

Το διάστημα που μπορεί να αναπαρασταθεί από n bits με τον τρόπο αυτό είναι από -2^{n-1} έως 2^{n-1} ενώ το μηδέν μπορεί να αναπαρασταθεί με δυο τρόπους.

3.3.2.2 Συμπλήρωμα ως προς 1

Στην αναπαράσταση αυτή αν το MSB είναι Αν το MSB είναι 0 ο αριθμός είναι θετικός το μέτρο του δίνεται στα υπόλοιπα bits. Αν το MSB είναι 1 ο αριθμός είναι αρνητικός και το συμπλήρωμα ως προς 1 των υπόλοιπων n-1 bits δίνει το μέτρο του. Το συμπλήρωμα ως προς 1 ενός δυαδικού αριθμού προκύπτει αντικαθιστώντας όλα τα 0 με 1 και όλα τα 1 με 0 στον αριθμό. Το διάστημα που μπορεί να αναπαρασταθεί από n bits με τον τρόπο αυτό είναι από -2^{n-1} έως 2^{n-1} ενώ το μηδέν μπορεί να αναπαρασταθεί με δυο τρόπους.

Παραδείγματα αναπαράστασης με 8 bit

+5 = 00000101
-5 = 11111010
Το 0 αναπαρίσταται με δύο τρόπους: 00…00 και 11…11

3.3.2.3 Συμπλήρωμα ως προς 2

Χρησιμοποιείται συνήθως στις σύγχρονες υλοποιήσεις για την αναπαράσταση αρνητικών αριθμών. Αν το MSB είναι 0 ο αριθμός είναι θετικός και το μέτρο του δίνεται από τα υπόλοιπα n-1 bit. Αν το MSB είναι 1 ο αριθμός είναι αρνητικός. ια να βρούμε το μέτρο του αριθμού πρέπει να υπολογίσουμε το συμπλήρωμα ως προς 2 όλων των ψηφίων του. Για τον υπολογισμό του συμπληρώματος ως προς 2:

- Παίρνουμε το συμπλήρωμα του αριθμού ως προς 1, δηλαδή αντικαθιστούμε κάθε 1 με 0 και κάθε 0 με 1.
- Προσθέτουμε 1
- Αν προκύψει κρατούμενο στο αριστερότερο bit το αγνοούμε.

Το διάστημα που μπορεί να αναπαρασταθεί από n bits με τον τρόπο αυτό είναι από -2^{n-1} έως $2^{n-1}-1$.

Παράδειγμα αναπαραστάσεων 8 bit
21 = 00010101
-21 = 11101010 +1 = 11101011

Αν έχουμε n bits η περιοχή των αριθμών είναι : -2^{n-1} έως $2^{n-1}-1$]

πχ.	Για	n=8	-128	έως	+129
	Για	n=16	-32768	έως	+32767
	Για	n=32	-2147483648	έως	+ 2147483647

Η πρόσθεση δυο αριθμών στην παράσταση συμπληρώματος ως προς 2 γίνεται απευθείας, ανεξάρτητα από το πρόσημό τους και χωρίς καμία μετατροπή. Η διαδικασία της πρόσθεσης είναι

η ίδια με αυτή στην παράσταση θετικών αριθμών. Αν υπάρξει κρατούμενο από την πρόσθεση των ποιο σημαντικών ψηφίων αυτό αγνοείται. Για την αφαίρεση ενός αριθμού Β από έναν Α, προσθέτουμε στον Α το συμπλήρωμα του Β ως προς 2.

• Παράδειγμα: Να εκτελεστεί η αφαίρεση 7-5 (=7+ (-5)) χρησιμοποιώντας

την αριθμητική του συμπληρώματος ως προς 2 με παραστάσεις αριθμών

8 bit.

7= 00000111

Αναπαράσταση του -5:

+5 = 00000101

-5 = 11111010+1 = 11111010

00000111 + 11111011 = 0010 = 2 (το κρατούμενο αγνοείται).

Επειδή οι αριθμητικές πράξεις δεν αλλάζουν όταν οι αρνιτικοί εκφράζονται με συμπλήρωμα ως προς δυο, η αναπαράσταση αυτή είναι η συνηθέστερη αναπαράσταση προσημασμένων αριθμών. Ο MIPS χρησιμοποιεί το συμπλήρωμα ως προς δυο.

3.3.3 Διασύνδεση Υλικού /Λογισμικού

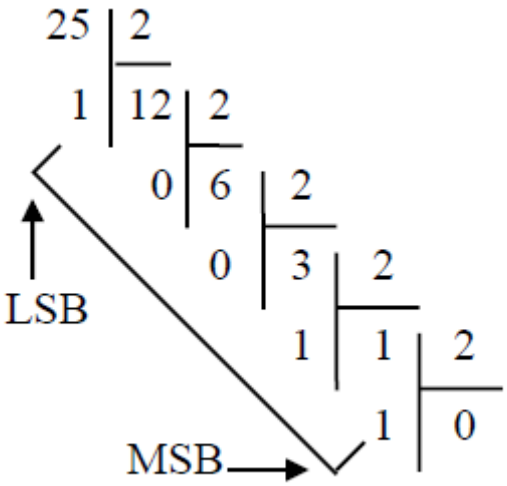
Η βάση 2 δεν είναι φυσική για τους ανθρώπους. Έχουμε 10 δάχτυλα και έτσι βρίσκουμε φυσικό τη βάση 10. Γιατί όμως οι υπολογιστές δεν μπορούν να χρησιμοποιούν δεκαδικούς αριθμούς; Στην πραγματικότητα ο πρώτος εμπορικός υπολογιστής πράγματι προσέφερε δεκαδική αριθμητική. Το πρόβλημα ήταν ότι ο υπολογιστής χρησιμοποιούσε ακόμη το σήμα του ανοικτού και κλειστού διακόπτη. Έτσι τα δεκαδικά ψηφία αναπαριστούνταν από πολλά δυαδικά ψηφία. Το δεκαδικό σύστημα αποδείχτηκε μη αποδοτικό και έτσι οι μηχανές μετατράπηκαν ώστε να χρησιμοποιούν το δυαδικό σύστημα, μετατρέποντας από δυαδικό σε δεκαδικό τις εξόδους και αντίστροφα τις εισόδους.

3.4 Μετατροπή από δεκαδικό σε δυαδικό .

Για την μετατροπή ενός θετικού ακεραίου αριθμού του δεκαδικού συστήματος στο δυαδικό, διαιρούμε διαδοχικά τον αριθμό καθώς και όλα τα πηλίκα που προκύπτουν δια 2, μέχρις ότου το πηλίκο να μηδενισθεί. Ο αντίστοιχος δυαδικός είναι εκείνος που προκύπτει αν γράψουμε τα υπόλοιπα που προέκυψαν σε αντίστροφη σειρά

Παράδειγμα

25_{ten}



Σχήμα 3.2: Μετατροπή από δεκαδικό σε δυαδικό

$25_{ten}=11001_{two}$

3.5 Μετατροπή από δυαδικό σε δεκαδικό.

Για τη μετατροπή από δυαδικό σε δεκαδικό προσθέτουμε τα γινόμενα που προκύπτουν από τα ψηφία του προς μετατροπή αριθμού επί τη δύναμη του 2 που κάθε ψηφίο αντιπροσωπεύει.
Παράδειγμα (μη Προσημασμένη αναπαράσταση)

Ποια η δεκαδική τιμή του πιο κάτω δυαδικού αριθμού:

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{two}$

Απάντηση

$(1x\ 2^{31})+ (1x2^{30})+ (1x2^{29})+\ldots+ (1x2^{2})+ (0x2^{1})+ (0x2^{0})$

$= -2^{31} + 2^{30} + 2^{29} +\ldots+ 2^{2} + 0 + 0$

$= 429497292_{ten}$

Στην προσημασμένη αναπαράσταση η δύναμη του ποιο συμαντικού Bit αφαιρείται.

Παράδειγμα (Προσημασμένη αναπαράσταση)

Ποια η δεκαδική τιμή του πιο κάτω δυαδικού αριθμού:

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{two}$

Απάντηση

$(1x -2^{31})+ (1x2^{30})+ (1x2^{29})+\ldots+ (1x2^{2})+ (0x2^{1})+ (0x2^{0})$

$= -2^{31} + 2^{30} + 2^{29} +\ldots+ 2^{2} + 0 + 0$

$= -2{,}147{,}483{,}648_{ten} + 2{,}147{,}483{,}644_{ten}$

$= -4_{ten}$

3.6 Προσημασμένη εναντίον απρόσημων συγκρίσεων

Υποθέτουμε ότι ο καταχωρητής $s0 περιέχει τον δυαδικό αριθμό

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two}$

και ο καταχωρητής $s1 περιέχει τον αριθμό

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two}$

ποιες είναι οι τιμές των καταχωρητών $t0, $t1 μετά από αυτές τις εντολές

```
slt     $t0, $s0, $s1 #singed comparison
sltu    $t1, $s0, $s1 #unsinged comparison
```

Απάντηση

Η τιμή του καταχωρητή $s0 αναπαριστά το –1 εάν είναι ακέραιος και το $4,294,967,295_{ten}$ εάν είναι απρόσημος ακέραιος. Η τιμή του καταχωρητή $s1 αναπαριστά το 1 σε οποιαδήποτε περίπτωση. Ο καταχωρητής $t0 έχει τη τιμή 1, αφου $-1_{ten} < 1_{ten,}$ και ο καταχωρητής $t1 έχει τη τιμή 0, αφού $4,294,967,295_{ten} > 1_{ten.}$

3.7 Μετατροπή προσήμου

$2_{ten} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two}$

βρίσκουμε το συμπλήρωμα του 2 και προσθέτουμε 1

$$\begin{array}{rl} & 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} \\ + & \underline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad 1_{two}} \\ = & 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} \\ = & -2_{ten} \end{array}$$

επαληθεύοντας,

$$\begin{array}{rl} & 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} \\ & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} \\ + & \underline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad 1_{two}} \\ = & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} \\ = & 2_{ten} \end{array}$$

Hexadecimal	Binary	Hexadecimal	Binary	Hexadecimal	Binary	Hexadecimal	Binary
0_{hex}	0000_{two}	4_{hex}	0100_{two}	8_{hex}	1000_{two}	C_{hex}	1100_{two}
1_{hex}	0001_{two}	5_{hex}	0101_{two}	9_{hex}	1001_{two}	D_{hex}	1101_{two}
2_{hex}	0010_{two}	6_{hex}	0110_{two}	A_{hex}	1010_{two}	E_{hex}	1110_{two}
3_{hex}	0011_{two}	7_{hex}	0111_{two}	B_{hex}	1011_{two}	F_{hex}	1111_{two}

Σχήμα 3.3: Πίνακας μετατροπής δεκαεξαδικού αριθμού σε δυαδικό

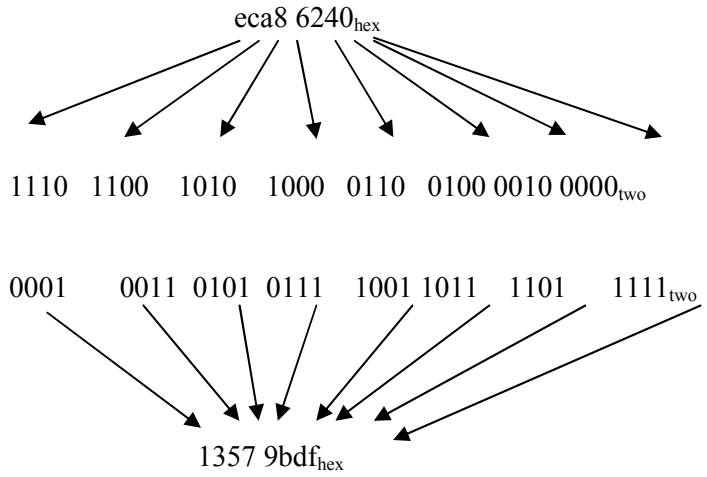
Παράδειγμα

Μετάτρεψε τον ακόλουθο δεκαεξαδικό και δυαδικό αριθμό στην αντίστροφη βάση :

$eca8\ 6240_{hex}$

$0001\ 0011\ 0101\ 0111\ 1001\ 1011\ 1101\ 1111_{two}$

Απάντηση



3.8 Πρόσθεση και Αφαίρεση

Τα ψηφία προσθέτονται ανά bit από τα δεξιά προς τα αριστερά. Τα κρατούμενα (carries) μεταφέρονται στο επόμενο ψηφίο στα αριστερά.

Παράδειγμα

Προσθέστε το 6_{ten} με το 7_{ten} και μετά αφαιρέστε το 6_{ten} απο το 7_{ten}.

Απάντηση

$$\begin{array}{rl} & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\ + & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110_{two} = 6_{ten} \\ \hline & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1101_{two} = 13_{ten} \end{array}$$

Η αφαίρεση του 6_{ten} απο το 7_{ten} μπορεί να γίνει απευθείας:

$$\begin{array}{rl} & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\ - & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110_{two} = 6_{ten} \\ \hline & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten} \end{array}$$

ή μπορεί να γίνει μέσω της πρόσθεσης χρησιμοποιώντας το συμπλήρωμα ως προς 2 (two's complement) του -6,

$$\begin{array}{rl} & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\ + & 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1010_{two} = -6_{ten} \end{array}$$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}$

Υπάρχει ο κίνδυνος το άθροισμα των δύο 32 - bit αριθμών να είναι πολύ μεγάλο και να μην μπορεί να αναπαρασταθεί κανονικά απο τα 32 - bit. Το φαινόμενο αυτό ονομάζεται **υπερχείλιση (overflow)**.

Η υπερχείλιση μπορεί επίσης να συμβεί και στην αφαίρεση. Για παράδειγμα, για να αφαιρέσουμε 2 απο το $-2,147,483,647_{ten}$ μετατρέπουμε το 2 σε -2 και το προσθέτουμε στο $-2,147,483,647_{ten}$. Κανονικά το αποτέλεσμα θα έπρεπε να ήταν $-2,147,483,649_{ten}$ αλλά δεν μπορούμε να αναπαραστήσουμε αυτό το αποτέλεσμα σε 32 bits, έτσι παίρνουμε την λανθασμένη θετική τιμή του $2,147,483,647_{ten}$.

Η υπερχείλιση δεν μπορεί να συμβεί στη πρόσθεση δύο αριθμών με διαφορετικό πρόσημο και κατά συνέπεια δεν μπορεί να συμβεί ούτε στην αφαίρεση δυο αριθμών με το ίδιο πρόσημο.

Παράδειγμα

Έστω ότι έχουμε μια μηχανή των 4 bits

5 + 5 = ?

Προσημασμένη αριθμητική

5 -> 0101

5 -> 0101 +

1010

overflow flag = $1 \oplus 0 = 1$ Αποτέλεσμα λάθος

carry out flag = 0

sing flag = 1

zero flag = 0 (υποδηλώνει κατα πόσο το αποτέλεσμα δεν είναι μηδέν)

Operation	Operand A	Operand B	Result
A + B	>= 0	>= 0	< 0
A + B	< 0	< 0	>= 0
A – B	>=0	< 0	< 0
A – B	< 0	>= 0	>= 0

Σχήμα 3.4: Οι συνδυασμοί των λειτουργιών και των τελεστών που οδηγούν σε υπερχείλιση.

Οι σχεδιαστές μηχανών πρέπει να βρουν ένα τρόπο που θα αγνοεί την υπερχείλιση σε κάποιες περιπτώσεις (π.χ. απρόσημους αριθμούς) και σε κάποιες άλλες να την αναγνωρίζει (π.χ. προσημασμένους αριθμούς). Η μηχανή MIPS έχει δύο είδη αριθμητικών εντολών για να αναγνωρίζει τις δύο επιλογές:

- Η πρόσθεση (add), η add immediate (addi) και η αφαίρεση (sub) προκαλούν exceptions στην υπερχείλιση, δηλαδή αναγνωρίζεται.
- Η απρόσημη πρόσθεση (addu), η άμεση απρόσημη πρόσθεση (addiu) και η απρόσημη αφαίρεση (subu) δεν προκαλούν exceptions στην υπερχείλιση, δηλαδή δεν αναγνωρίζεται.

3.9 Λογικές Λειτουργίες

- Η μηχανή MIPS παρέχει εντολές για επεξεργασία χαρακτήρων μέσα σε μια λέξη (32 bits).

- Μια ομάδα από τέτοιου είδους εντολές ονομάζονται μετακινήσεις (shifts). Αυτές οι εντολές μεταφέρουν όλα τα bits μιας λέξης στα αριστερά ή στα δεξιά, γεμίζοντας τα κενά με μηδενικά. Για παράδειγμα, αν ο καταχωρητής $16 περιέχει:

  0000 0000 0000 0000 0000 0000 0000 1101

  και εκτελέσουμε την εντολή που μεταφέρει 8 bits αριστερά θα πάρουμε:

  0000 0000 0000 0000 0000 1101 0000 0000

- Οι δύο εντολές μετακίνησης στον MIPS ονομάζονται *λογική μετακίνηση προς τα αριστερά* (**shift left logical - sll**) και *λογική μετακίνηση προς τα δεξιά* (**shift right logical - srl**). Αν θέλαμε να εκτελέσουμε την πιο πάνω λειτουργία, υποθέτοντας ότι το αποτέλεσμα θα αποθηκευτεί στο καταχωρητή $10, θα έχουμε:

  sll $10, $16, 8 # reg $10 + reg $16 << 8 bits

- Η αναπαράσταση της πιο πάνω εντολής στη γλώσσα μηχανής θα είναι:

Op	rs	Rt	Rd	Shamt	funct
0	0	16	10	8	0

- Μια άλλη εντολή που είναι χρήσιμη για απομόνωση πεδίων είναι η εντολή **AND**. Η εντολή AND είναι μια bit προς bit λειτουργία που βάζει 1 στο αποτέλεσμα μόνο αν και τα δύο bits των τελεστών είναι 1. Αν για παράδειγμα ο καταχωρητής $10 περιέχει:

  0000 0000 0000 0000 0000 1101 0000 0000

  και ο καταχωρητής $9 περιέχει:

  0000 0000 0000 0000 0011 1100 0000 0000

  Μετά την εκτέλεση της εντολής,

  and $8, $9, $10 # reg $8 = reg $9 & reg $10

  Η τιμή στο καταχωρητή $8 θα είναι:

  0000 0000 0000 0000 0000 1100 0000 0000

- Η εντολή **OR** είναι μια bit προς bit λειτουργία η οποία βάζει 1 στο αποτέλεσμα αν το bit οποιουδήποτε από τους τελεσταίους είναι 1. Αν για παράδειγμα έχουμε τους καταχωρητές $9 και $10 με τα ίδια περιεχόμενα όπως στο πιο πάνω παράδειγμα, το αποτέλεσμα της εντολής MIPS:

  or $8, $9, $10 # reg $8 = reg $9 | reg $10

  είναι το αποτέλεσμα στο καταχωρητή $8:

  0000 0000 0000 0000 0011 1101 0000 0000

- Το Σχήμα 3.5 απεικονίζει τις λογικές εντολές στη C και τις αντίστοιχες εντολές στη μηχανή MIPS.
- Στο σχήμα 3.6 βλέπουμε τις Logical εντολές καθώς επίσης και όλες τις εντολές του MIPS που έχουμε μέχρι τώρα δει.

Logical Operations	C Operators	MIPS Instructions
Shift Left	<<	sll
Shift Right	>>	srl
AND	&	and, andi
OR	|	or, ori

Σχήμα 3.5: Οι λογικές εντολές στη γλώσσα C και MIPS.

MIPS operands

Name	Example	Comments
32 registers	$s0–$s7, $t0–$t9, $gp, $fp, $zero, $sp, $ra, $at	Fast locations for data. In MIPS, data must be in registers to perform arithmetic. MIPS register $zero always equals 0. Register $at is reserved for the assembler to handle large constants.
2^{30} memory words	Memory[0], Memory[4], ..., Memory[4294967292]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as arrays, and spilled registers, such as those saved on procedure calls.

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three operands; overflow detected
	subtract	sub $s1,$s2,$s3	$s1 = $s2 - $s3	Three operands; overflow detected
	add immediate	addi $s1,$s2,100	$s1 = $s2 + 100	+ constant; overflow detected
	add unsigned	addu $s1,$s2,$s3	$s1 = $s2 + $s3	Three operands; overflow undetected
	subtract unsigned	subu $s1,$s2,$s3	$s1 = $s2 - $s3	Three operands; overflow undetected
	add immediate unsigned	addiu $s1,$s2,100	$s1 = $s2 + 100	+ constant; overflow undetected
	move from coprocessor register	mfc0 $s1,$epc	$s1 = $epc	Used to copy Exception PC plus other special registers
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	and immediate	andi $s1,$s2,100	$s1 = $s2 & 100	Bit-by-bit AND reg with constant
	or immediate	ori $s1,$s2,100	$s1 = $s2 | 100	Bit-by-bit OR reg with constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $$s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Data transfer	load word	lw $s1,100($s2)	$s1 = Memory[$s2 + 100]	Word from memory to register
	store word	sw $s1,100($s2)	Memory[$s2 + 100] = $s1	Word from register to memory
	load byte unsigned	lbu $s1,100($s2)	$s1 = Memory[$s2 +100]	Byte from memory to register
	store byte	sb $s1,100($s2)	Memory[$s2 + 100] = $s1	Byte from register to memory
	load upper immediate	lui $s1,100	$s1 = 100 * 2^{16}	Loads constant in upper 16 bits
Conditional branch	branch on equal	beq $s1,$s2,25	if ($s1 != $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	bne $s1,$s2,25	if ($s1 == $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; two's complement
	set less than immediate	slti $s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare < constant; two's complement
	set less than unsigned	sltu $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; natural numbers
	set less than immediate unsigned	sltiu $s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare < constant; natural numbers
Unconditional jump	jump	j 2500	go to p10000	Jump to target address
	jump register	jr $ra	go to $ra	For switch, procedure return
	jump and link	jal 2500	$ra = PC + 4; go to 10000	For procedure call

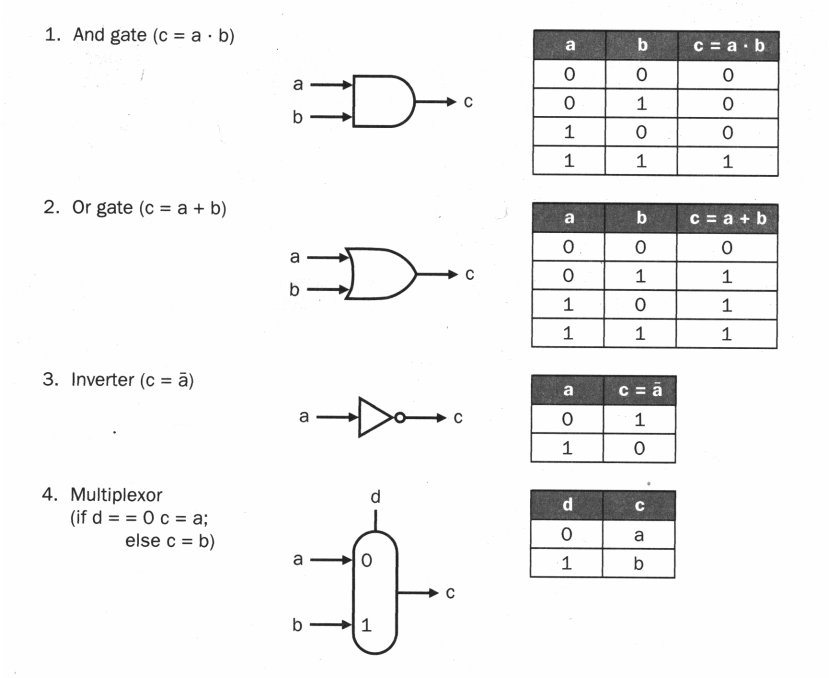
FIGURE 4.7 MIPS architecture revealed thus far. Color indicates the portions since Figure 4.5 on page 224. MIPS machine language is also listed on the back endpapers of this book.

Σχήμα 3.6: εντολές του MIPS που έχουμε μέχρι τώρα δει.

Κατασκευή της αριθμητικής και λογικής μονάδα

- Η αριθμητική και λογική μονάδα είναι μια συσκευή που εκτελεί τις αριθμητικές λειτουργίες όπως πρόσθεση και αφαίρεση και τις λογικές εντολές όπως and και or στον υπολογιστή.

- Θα κατασκευάσουμε την ALU από τα τέσσερα κομμάτια υλικού που φαίνονται στο Σχήμα 3.7

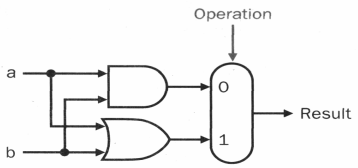


Σχήμα 3.7:Τα τέσσερα κομμάτια υλικού που χρησιμοποιούνται για την κατασκευή μιας αριθμητικής λογικής μονάδας.

- Επειδή οι λέξεις στον MIPS έχουν μέγεθος 32 bit, πρέπει και η ALU να έχει μέγεθος 32 bit. Ας υποθέσουμε ότι θα συνδέσουμε 32 ALU μεγέθους 1 bit η κάθε μια.

Μια ALU του 1 bit

Η ενός bit λογική μονάδα για τις λειτουργίες AND και OR, απεικονίζεται πιο κάτω:



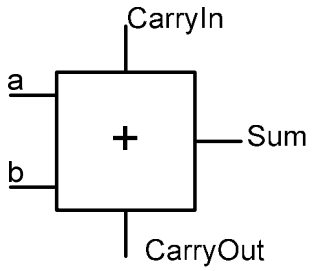
Σχήμα 3.8:Τα Η μεγέθους 1 bit λογική μονάδα για τις λειτουργίες AND και OR

Ο πολυπλέκτης στα δεξιά διαλέγει την λειτουργία a AND b ή a OR b, ανάλογα με την τιμή της Operation, αν είναι 1 ή 0.

Το επόμενο βήμα είναι να συμπεριλάβουμε και την λειτουργία της πρόσθεσης στο υλικό που κατασκευάζουμε. Ένας αθροιστής (adder) πρέπει να έχει δύο εισόδους για τους τελεσταίους και μια έξοδο 1 bit για το αποτέλεσμα (Sum). Άρα πρέπει να υπάρχει και μια δεύτερη έξοδος που θα μεταφέρει το κρατούμενο της πρόσθεσης αν υπάρχει. Αυτή η έξοδος θα ονομάζεται CarryOut. Αφού το CarryOut του γειτονικού αθροιστή πρέπει να συμπεριληφθεί ως είσοδος θα χρειαστούμε μια τρίτη είσοδο που θα ονομάσουμε CarryIn.

- Μπορούμε να εκφράσουμε τις συναρτήσεις εξόδου CarryOut και Sum ως λογικές εξισώσεις, και αυτές οι εξισώσεις μπορούν να υλοποιηθούν με τα building blocks του σχήματος 4.8.

Στο Σχήμα 9 φαίνονται οι εισόδοι και οι εξόδοι του ενός bit αθροιστή.



Σχήμα 3.9: Ένας αθροιστής του ενός bit.

Ας κάνουμε το CarryOut. Ο πιο κάτω πίνακας δείχνει τις τιμές των εισόδων όταν το CarryOut είναι 1:

Inputs		
A	b	CarryIn
0	1	1
1	0	1
1	1	0
1	1	1

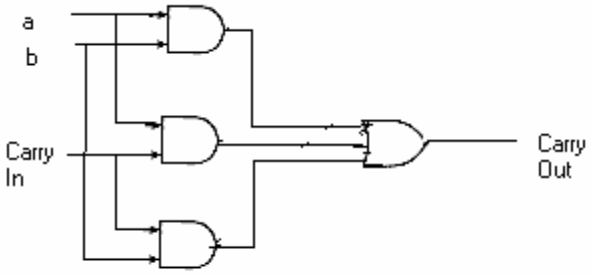
Μπορούμε να μετατρέψουμε αυτόν τον αληθοπίνακα στην εξίσωση:

CarryOut = (b . CarryIn) + (a . CarryIn) + (a . b) + (a . b . CarryIn)

Η οποία μετατρέπεται στην εξίσωση:

CarryOut = (b . CarryIn) + (a . CarryIn) + (a . b)

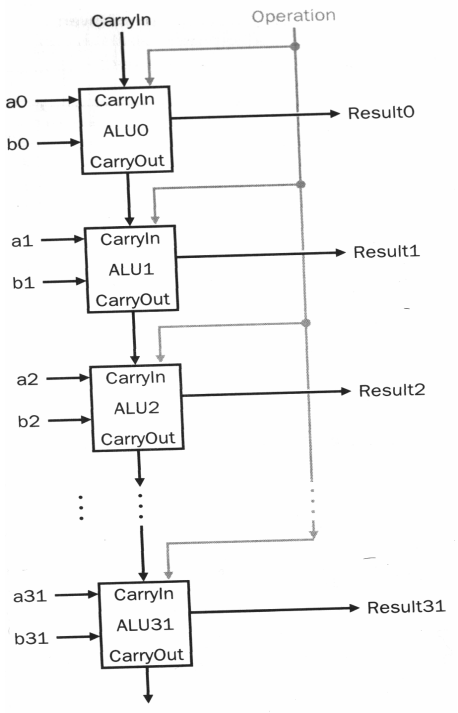
- Το υλικό (hardware) του αθροιστή στο Σχήμα 3.10 αποτελείται από τρεις πύλες AND και μια πύλη OR. Οι τρείς AND gates ανταποκρίνονται στις τρεις παρενθέσεις της πιο πάνω εξίσωσης για τον υπολογισμό του CarryOut και η πύλη OR αθροίζει τους τρεις όρους.



Σχήμα 3.10: Υλοποίηση του CarryOut με πύλες

32 - bit ALU

- Η ολοκληρωμένη 32 - bit ALU δημιουργείται ενώνοντας γειτονικά 1-bit ALU. Το Σχήμα 3.11 απεικονίζει την 32-bit ALU.



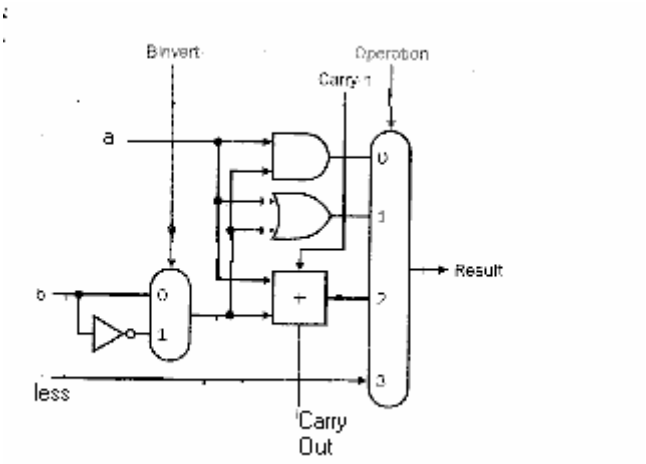
Σχήμα 3.11: Η 32 - bit ALU δημιουργείται απο 32 1 - bit ALUs .

Μετατροπή της 32-bit ALU για τον MIPS

- Το σύνολο εντολών add, subtrack, AND, OR βρίσκεται στην ALU σε όλους σχεδόν τους υπολογιστές.

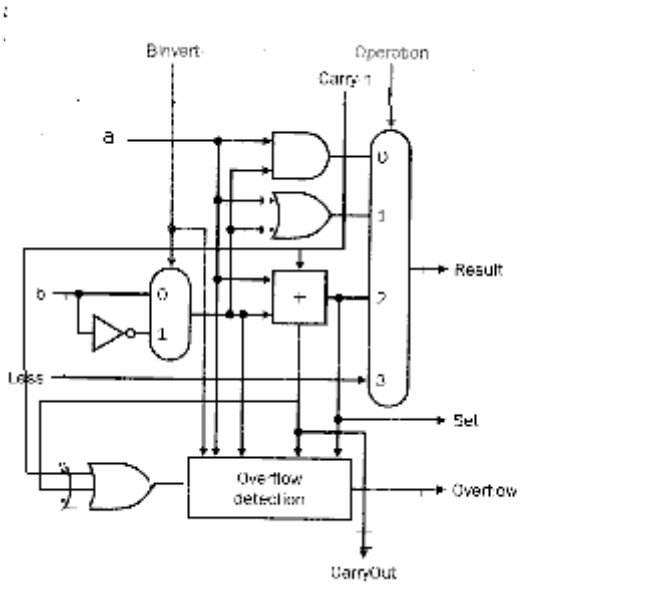
- Όλες σχεδόν οι εντολές του MIPS μπορούν να εκτελεστούν από την πιο πάνω ALU. Μια εντολή που χρειάζεται ενίσχυση είναι η set-on-less-than εντολή του MIPS. Η πιο πάνω εντολή επιστρέφει 1 αν Rs < Rt, διαφορετικά επιστρέφει 0. Άρα η εντολή set on less than θα δώσει τιμή 0 σε όλα τα bits, εκτός από το least significant bit το οποίο θα πάρει τιμή ανάλογα με το αποτέλεσμα της σύγκρισης. Πρέπει να επεκτείνουμε τον πολυπλέκτη, για να δίνει μια τιμή για την σύγκριση less than, για κάθε bit στην ALU.

- Στο Σχήμα 3.12 φαίνεται η καινούρια 1-bit ALU με τον επεκταμένο πολυπλέκτη.



Σχήμα 3.12: Η 1-bit ALU η οποία εκτελεί τις λέιτουργίες AND, OR, και πρόσθεσης.

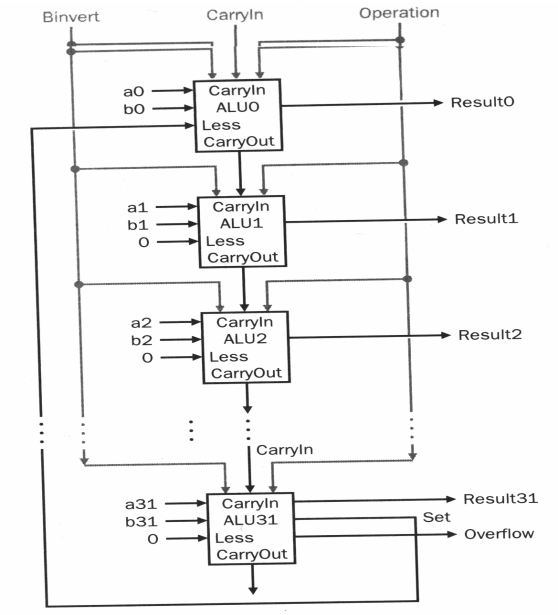
- Χρειαζόμαστε μια νέα ALU για το most significant bit που κάνει την έξοδο του αθροιστή διαθέσιμη για την πρόσθεση με το καθορισμένο αποτέλεσμα εξόδου (standard result output). Στο Σχήμα 4.17 φαίνεται ο σχεδιασμός με αυτή την καινούρια έξοδο στον αθροιστή. Αφού χρειαζόμαστε μια καινούρια ALU για το most significant bit πρέπει να προσθέσουμε την τεχνική για ανίχνευση της υπερχείλισης (overflow), αφού σχετίζεται με αυτό το bit.



Σχήμα 3.13: Μια 1-bit ALU για το most significant bit.

- Το Σχήμα 4.18 δείχνει την τελική μορφή της 32 bit ALU. Προσέξτε ότι κάθε φορά που θέλουμε η ALU να εκτελέσει την λειτουργία της αφαίρεσης, το CarryIn και το Binvert παίρνουν τιμή 1. Για την πρόσθεση ή τις λογικές λειτουργίες θέλουμε και οι δύο γραμμές ελέγχου να έχουν τιμή 0. Άρα μπορούμε να απλοποιήσουμε την σχεδίαση της ALU, ενώνοντας την CarryIn και Binvert σε μια γραμμή που θα ονομάζεται Bnegate.

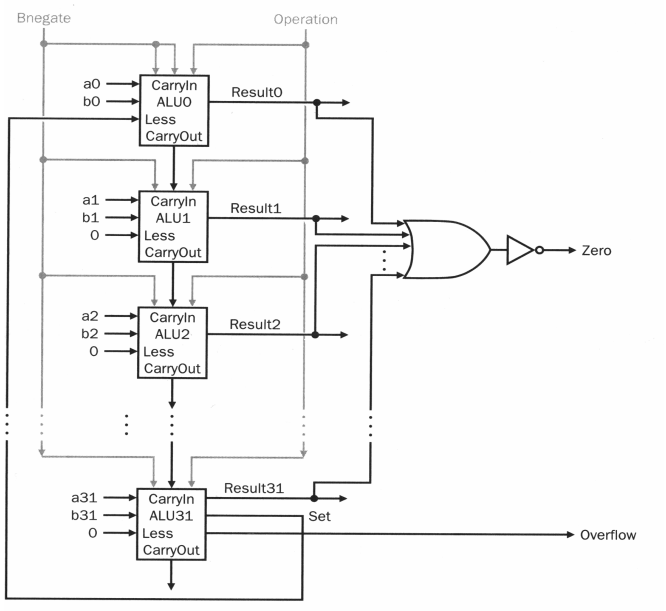
Υλοποίηση της εντολής set on less than (slt)



Σχήμα 3.14: Μια 32-bit ALU η οποία δημιουργείται απο 31 αντίγραφα της 1-bit ALU του σχήματος 3.12 και μια 1-bit ALU του σχήματος 3.13

- Η ALU για την μηχανή MIPS πρέπει επίσης να υποστηρίζει τις εντολές διακλάδωσης υπό συνθήκη (conditional branch). Αυτές οι εντολές διακλαδώνονται αν τα περιεχόμενα των δύο καταχωρητών είναι ίσα ή αν δεν είναι ίσα. Ο πιο εύκολος τρόπος για έλεγχο της ισότητας με την ALU είναι να αφαιρέσεις τα περιεχόμενα του ενός καταχωρητή από τον άλλο και μετά να ελέγξεις αν το αποτέλεσμα είναι ίσο με μηδέν. Πρέπει να προσθέσουμε υλικό που θα ελέγχει αν η έξοδος είναι ίση με μηδέν. Αυτό μπορεί να γίνει με μια OR πύλη η οποία θα μαζεύει όλες τις εξόδους. Στο Σχήμα 4.19 φαίνεται η νέα υλοποίηση της ALU.

Υλοποίηση των conditional branches (εντολών υπο συνθήκη)



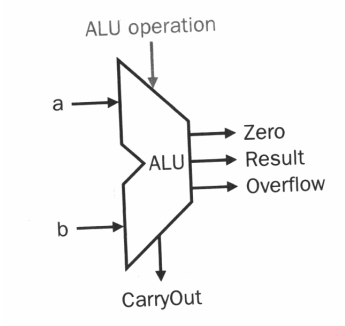
Σχήμα 3.15: Η τελική 32 – bit ALU

- Στο Σχήμα 3.16 φαίνονται οι γραμμές ελέγχου της ALU και τις αντίστοιχες ALU λειτουργίες.

ALU Control lines	Function
000	And
001	Or
010	Add
110	Subtract
111	Set-on-less-than

Σχήμα 3.16: Οι τιμές των τριών γραμμών ελέγχου της ALU, Bnegate και Operation και η αντίστοιχη ALU λειτουργία.

- Στο Σχήμα 3.17 φαίνεται ο παγκόσμιος συμβολισμός για την ολοκληρωμένη ALU. Οι τρεις γραμμές λειτουργία της ALU που αποτελούνται από τους συνδυασμούς της 1-bit Bnegate γραμμής και της 2-bit γραμμή λειτουργίας (operation line), κάνουν την ALU να παράγει της επιθυμητές πράξεις: πρόσθεση, αφαίρεση, AND, OR, set on less than.



Σχήμα 3.17: Το σύμβολο που αντιπροσωπεύει μια ALU του σχήματος 4.19

4.Ο επεξεργαστής : Διάδρομος Δεδομένων και Έλεγχος

4.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα μελετήσουμε την υλοποίηση του διαδρόμου δεδομένων και της μονάδας ελέγχου για το σύνολο εντολών της μηχανής MIPS.

Πιο συγκεκριμένα, θα σχεδιάσουμε μια υλοποίηση για τις βασικές εντολές του MIPS, συμπεριλαμβανομένων :

- Τις εντολές που κάνουν αναφορά στη μνήμη (memory reference), δηλ. την load word(lw) και store word(sw).
- Τις αριθμητικές και λογικές εντολές, δηλ. add, sub, and, or και slt.
- Την εντολή σύγκρισης branch equal(beq) και τέλος την εντολή jump(j).

Θα έχουμε την ευκαιρία να δούμε πώς η αρχιτεκτονική του συνόλου εντολών επηρεάζει μια υλοποίηση και πώς η επιλογή των διαφόρων στρατηγικών υλοποίησης επηρεάζει το χρόνο ρολογιού και το CPI μιας μηχανής.

Βλέποντας την υλοποίηση που θα σχεδιάσουμε, θα μπορέσουμε να επισημάνουμε τις βασικές αρχές που είχαμε δει προηγούμενα (“Make the common case fast” και “Simplicity favors regularity”).

4.2 Μια περίληψη της υλοποίησης

Ο μετρητής προγράμματος (PC-Program Counter) είναι ένας 32-bit καταχωρητής ο οποίος έχει αποθηκευμένη τη διεύθυνση της επόμενης εντολής που θα εκτελεστεί.

Για κάθε εντολή τα πρώτα δύο βήματα είναι τα ίδια :

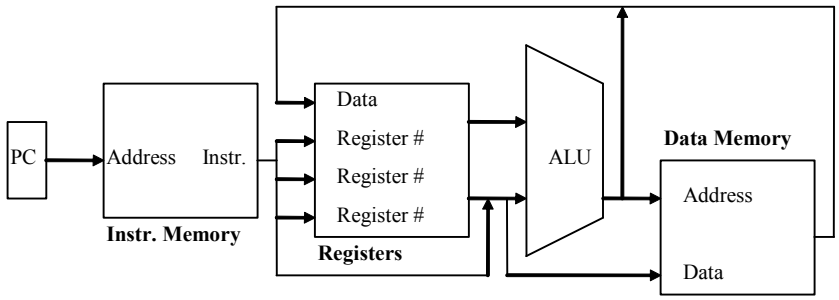
1. Στέλλεται ο PC στη μνήμη που περιέχει τον κώδικα και προσκομίζεται (fetch) η εντολή από τη μνήμη.

2. Διαβάζεται ένας ή δύο καταχωρητές χρησιμοποιώντας τα πεδία της εντολής, για να εντοπιστούν οι καταχωρητές που θα διαβαστούν τα δεδομένα τους, π.χ. για την εντολή load χρειάζεται να διαβαστεί μόνο ένας καταχωρητής αλλά για τις υπόλοιπες εντολές πρέπει να διαβαστούν δύο καταχωρητές.

Μετά από αυτά τα βήματα, οι ενέργειες που θα ακολουθήσουν για την ολοκλήρωση της εκτέλεσης της κάθε εντολής εξαρτώνται από τον τύπο της.

Το Σχήμα 5.1 σκιαγραφεί μια αφηρημένη όψη της υλοποίησης του MIPS. Διαφαίνονται οι κυριότερες λειτουργικές μονάδες και οι συνδέσεις μεταξύ τους. Όλες οι εντολές αρχίζουν χρησιμοποιώντας το μετρητή προγράμματος (PC). Ο PC παρέχει τη διεύθυνση της εντολής που θα εκτελεστεί, στη μνήμη εντολών (instruction memory). Προσδιορίζονται οι καταχωρητές που θα χρησιμοποιηθούν, από τα πεδία της εντολής και εκτελούνται λειτουργίες ανάλογα με την εντολή (υπολογισμός μιας διεύθυνσης μνήμης ή ενός αριθμητικού αποτελέσματος ή μιας

σύγκρισης). Το αποτέλεσμα από την ALU (Αριθμητική και Λογική μονάδα) ή τη μνήμη αποθηκεύεται πίσω στο αρχείο των καταχωρητών (register file).

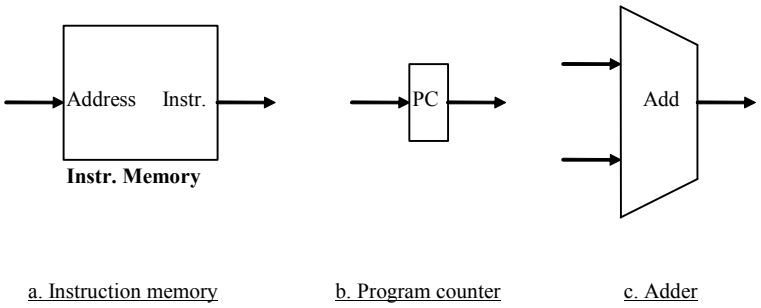


Σχήμα 4.1: ***Αφηρημένη όψη της υλοποίησης του MIPS***

Το αρχείο καταχωρητών είναι μια δομή που περιέχει τη κατάσταση των 32 καταχωρητών του επεξεργαστή σε μια μηχανή. Παρέχει δύο εισόδους για διάβασμα και μια είσοδο για εγγραφή.

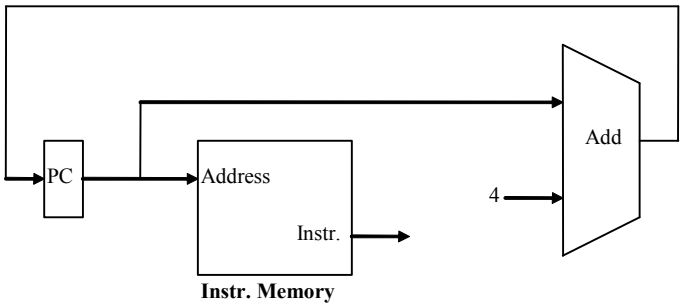
5.2 Κτίζόντας το διάδρομο δεδομένων (Datapath)

Στο Σχήμα 4.2 φαίνονται τα βασικά στοιχεία που θα χρειαστούμε για την υλοποίηση του διαδρόμου δεδομένων. Θα χρειαστούμε (a) μια μνήμη εντολών για να αποθηκεύσουμε τις εντολές ενός προγράμματος, η διεύθυνση της εντολής πρέπει επίσης να φυλάγεται στο (b) μετρητή του προγράμματος(PC), τέλος θα χρειαστούμε (c) ένα αθροιστή (adder) για να αυξάνει το PC στη διεύθυνση της επόμενης εντολής. Ο αθροιστής μπορεί να δημιουργηθεί από την ALU και θα ονομάσουμε αυτή τη λειτουργία της ALU ADD (Πρόσθεση).



Σχήμα 4.2: Βασικά στοιχεία για την υλοποίηση του διαδρόμου δεδομένων

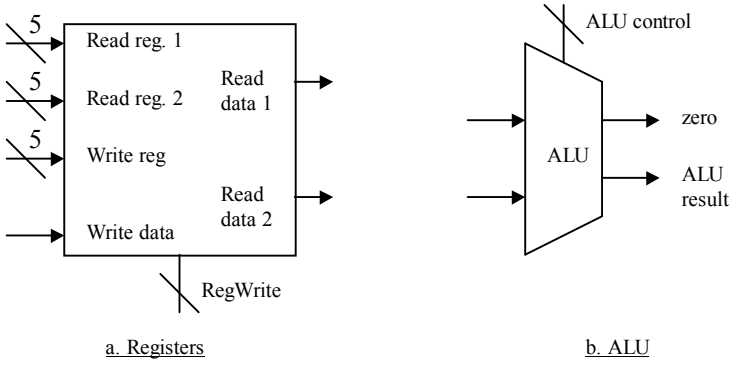
Στο Σχήμα 4.3 φαίνεται ένα μέρος του διαδρόμου δεδομένων που χρησιμοποιείται για την προσκόμιση των εντολών από τη μνήμη και την αύξηση του μετρητή προγράμματος (PC) κατά 4 bytes.



Σχήμα 4.3: Προσκόμιση εντολών από τη μνήμη και αύξηση του PC

4.3 Εντολές τύπου R - Αριθμητικές και Λογικές Εντολές

Οι αριθμητικές και λογικές εντολές διαβάζουν δύο καταχωρητές, εκτελούν μια λειτουργία ALU και γράφουν το αποτέλεσμα. Τέτοιες εντολές είναι οι add, sub και slt. Στο Σχήμα 4.4 φαίνονται τα δύο στοιχεία που χρειάζονται για την υλοποίηση της λειτουργίας των εντολών τύπου R που είναι το αρχείο των καταχωρητών και η ALU.



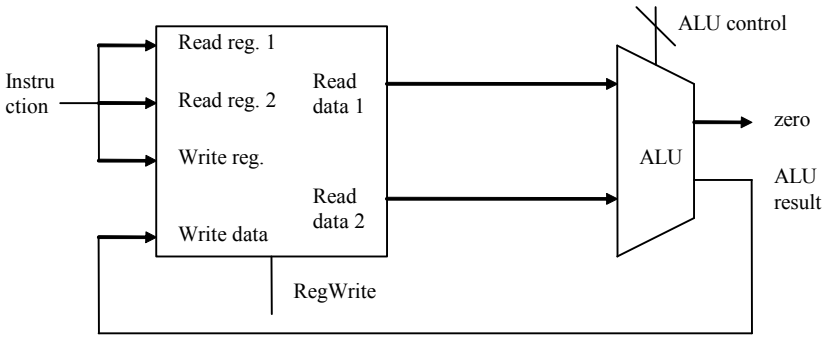
Σχήμα 4.4: Στοιχεία που χρειάζονται για την υλοποίηση εντολών τύπου R

Το μέγεθος του αριθμού των καταχωρητών εισόδου στο αρχείο των καταχωρητών είναι 5bits και προσδιορίζει έναν από τους 32 καταχωρητές, ενώ τα δεδομένα εισόδου και τα δύο δεδομένα εξόδου είναι μεγέθους 32 bits.

Η ALU παίρνει δύο εισόδους μεγέθους 32 bits και παράγει ένα αποτέλεσμα 32-bit. Η ALU ελέγχεται από ένα 3-bit σήμα το οποίο καθορίζεται από τη μονάδα ελέγχου.

Στο Σχήμα 4.5φαίνεται ο διάδρομος δεδομένων για τις εντολές τύπου R. Η ALU μπορεί να εκλεχθεί για να παρέχει όλες τις βασικές διαδικασίες που απαιτούνται από τις εντολές τύπου R.

Επίσης χρειαζόμαστε και το μηδέν (zero flag) ως έξοδος στο ALU για υλοποίηση των branches.



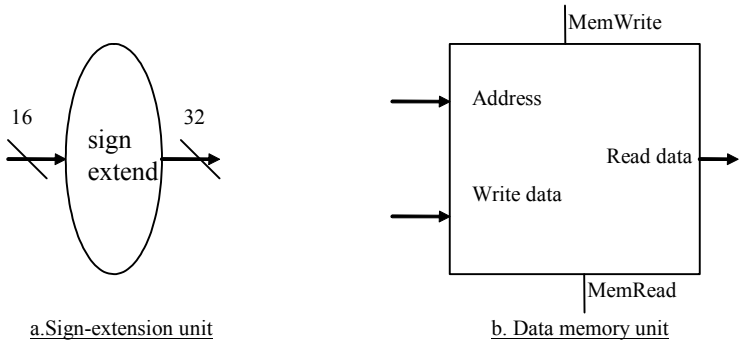
Σχήμα 4.5:: Διάδρομος δεδομένων για τις εντολές τύπου R

4.4 Εντολές με αναφορά στη μνήμη - load και store Instructions

Παράδειγμα οι εντολές: lw $8,Astart($2) ή sw $8,Astart ($2).

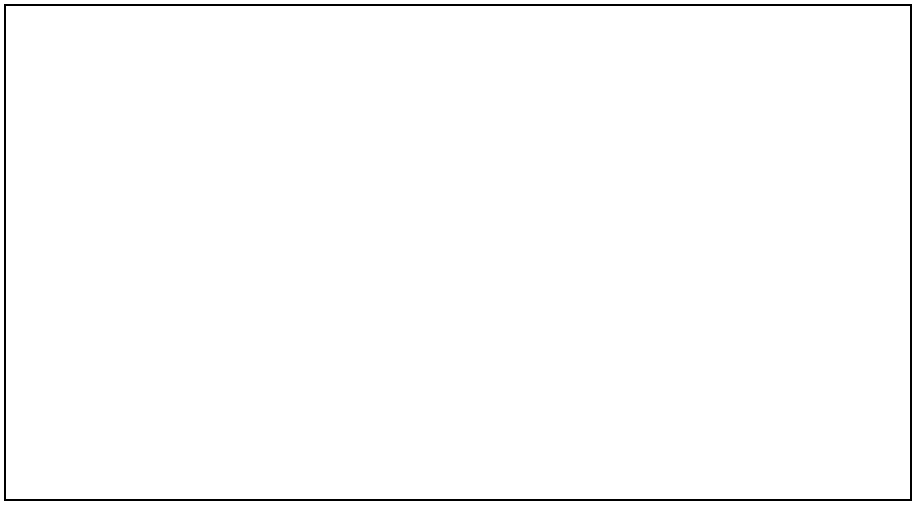
Αυτές οι εντολές υπολογίζουν τη διεύθυνση της μνήμης προσθέτοντας την τιμή του καταχωρητή $2 στο 16-bit πεδίο που δίνει τη διεύθυνση της βάσης του πίνακα(Astart).

Σύμφωνα με το Σχήμα 4.6 για την υλοποίηση των εντολών αυτού του τύπου θα χρειαστούμε το αρχείο των καταχωρητών, την ALU και επιπλέον θα χρειαστούμε (a) μια λειτουργική μονάδα για προέκταση σήματος(sign-extend) για το 16-bit πεδίο που πρέπει να μετατραπεί σε 32-bit για να μπορέσει να είναι είσοδος στην ALU και (b) μια μνήμη δεδομένων για γραφή ή ανάγνωση διευθύνσεων και γραφή δεδομένων.



Σχήμα 4.6: Στοιχεία που χρειάζονται για τις εντολές load - store

Το Σχήμα 4.7δείχνει το διάδρομο δεδομένων για τις εντολές load και store. Υπολογίζεται η διεύθυνση μνήμης στην οποία βρίσκονται τα δεδομένα, μετά διαβάζεται ή γράφεται από τη μνήμη δεδομένων και τέλος αν η εντολή είναι load γράφεται στο αρχείο των καταχωρητών.



Σχήμα 4.7: ***Διάδρομος δεδομένων για τις εντολές load - store***

4.5 Εντολή Σύγκρισης - Branch Equal Instruction

Για παράδειγμα η εντολή: beq $1, $2, offset.

Έχει τρεις τελεσταίους: δύο καταχωρητές που συγκρίνονται για ισότητα και ένα 16 bit offset για υπολογισμό της διεύθυνσης-στόχου (target address) που θα κατευθυνθεί η εκτέλεση του προγράμματος.

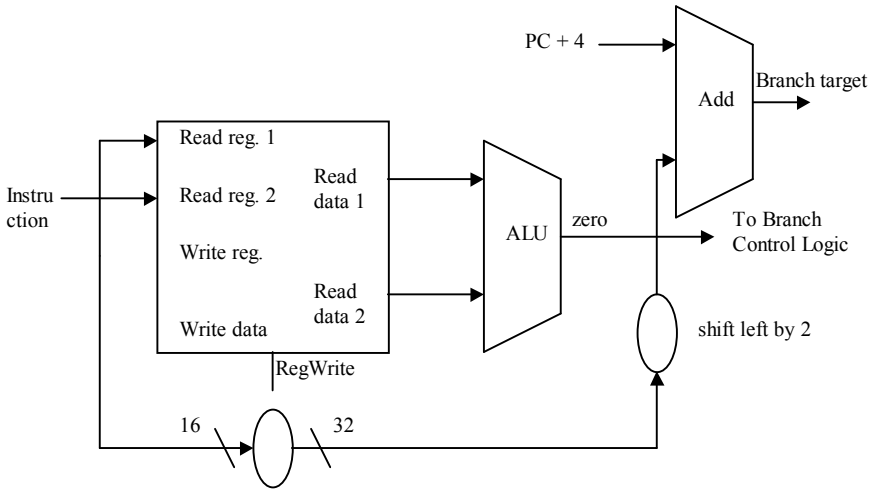
Υπάρχουν δύο σημεία που πρέπει να προσέξουμε στην αρχιτεκτονική αυτού του συνόλου εντολών :

- Η αρχιτεκτονική καθορίζει τη βάση για υπολογισμό της διεύθυνσης της συνθήκης (branch address) ως τη διεύθυνση που ακολουθεί τη συνθήκη (target address). Αφού υπολογίζουμε το PC+4 (η διεύθυνση της επόμενης εντολής) για την προσκόμιση της εντολής στο διάδρομο δεδομένων είναι εύκολο να χρησιμοποιήσουμε αυτή τη τιμή ως τη βάση για υπολογισμό της διεύθυνσης-στόχος της συνθήκης (branch target address).

- Επίσης το πεδίο offset μετακινείται 2 bits αριστερά για να είναι word offset. Αυτή η μετακίνηση είναι βοηθητική επειδή αυξάνεται η αποτελεσματικότητα του πεδίου offset κατά ένα παράγοντα του 4.

Η εντολή υπό συνθήκη πρέπει να κάνει δύο πράγματα : να υπολογίζει τη διεύθυνση-στόχος (όπως στο Σχήμα 5.8 + τον αθροιστή) και να συγκρίνει τα περιεχόμενα των καταχωρητών. Η σύγκριση μπορεί να γίνει χρησιμοποιώντας το αρχείο καταχωρητών του Σχήματος 5.7, χρησιμοποιώντας το zero flag της ALU. Η ALU επιστρέφει σήμα όταν το αποτέλεσμα από την αφαίρεση των τιμών των δύο καταχωρητών είναι μηδέν, αν είναι μηδέν τότε ξέρουμε ότι οι δύο τιμές είναι οι ίσες.

Στο Σχήμα 4.8 φαίνεται η υλοποίηση του διαδρόμου δεδομένων για την εντολή υπό συνθήκη.



Σχήμα 4.8: Διάδρομος δεδομένων για τις εντολές υπό συνθήκη

4.6 Ένα Απλό Σχήμα Υλοποίησης

4.6.1 Δημιουργία ενός μονού διαδρόμου δεδομένων

Θα ορίσουμε μια καινούργια μέχρι τώρα έννοια, την έννοια του πολυπλέκτη (multiplexor). Ο πολυπλέκτης διαλέγει από μια συλλογή δεδομένων τα δεδομένα που ικανοποιούν τις συνθήκες των γραμμών ελέγχου του.

Παράδειγμα 1

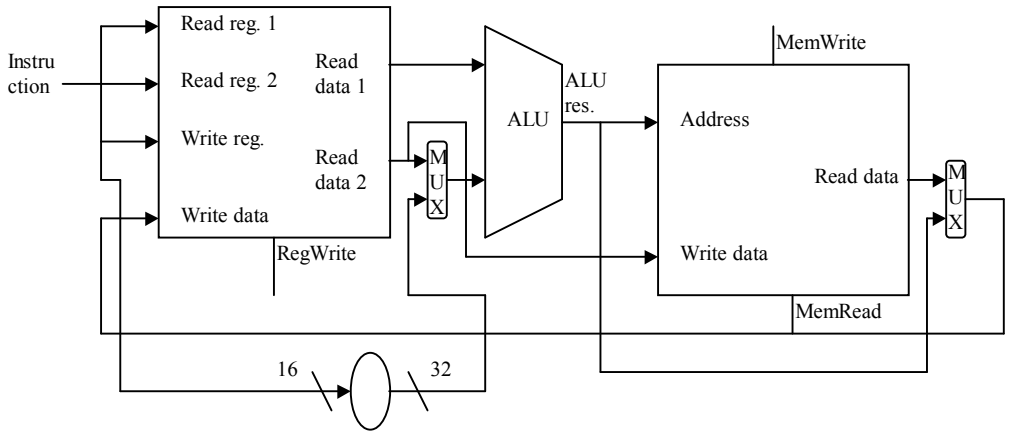
Ο διάδρομος δεδομένων των εντολών τύπου R στο Σχήμα 4.5 και ο διάδρομος δεδομένων των εντολών μνήμης στο Σχήμα 4.7 είναι σχεδόν ο ίδιος. Οι μόνες διαφορές είναι :

1. Η δεύτερη είσοδος για την ALU είναι είτε καταχωρητής (R type instructions) είτε sign-extended (memory instructions).
2. Η τιμή που γράφεται στο καταχωρητή αποτελέσματος προέρχεται από την ALU (R type) ή από τη μνήμη (load).

Δείξε πως μπορείς να συνδυάσεις τους δύο διαδρόμους δεδομένων χρησιμοποιώντας πολυπλέκτες χωρίς να επαναλάβεις τις λειτουργικές μονάδες που είναι ίδιες στα δύο σχήματα. Μπορείς να αγνοήσεις τον έλεγχο των πολυπλεκτών.

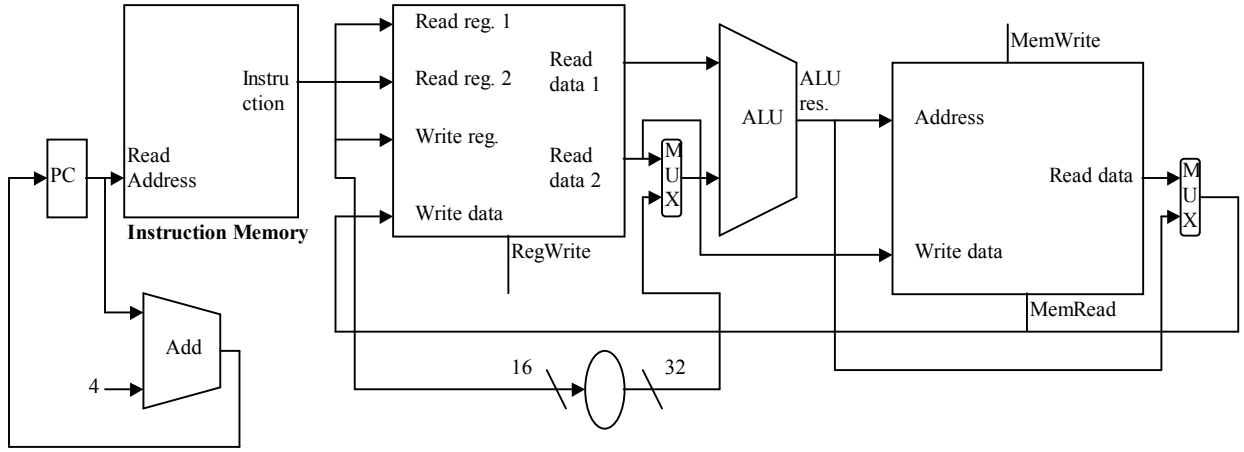
Απάντηση

Για να συνδυάσουμε τους δύο διαδρόμους δεδομένων και να χρησιμοποιήσουμε μόνο ένα αρχείο καταχωρητών και μια ALU πρέπει να υποστηρίξουμε δύο διαφορετικές πηγές για τη δεύτερη είσοδο της ALU και δύο διαφορετικές πηγές για τα δεδομένα που αποθηκεύονται στο αρχείο των καταχωρητών. Έτσι ένας πολυπλέκτης τοποθετείται στην είσοδο της ALU και ένας άλλος για τα δεδομένα εισόδου του αρχείου των καταχωρητών. Στο Σχήμα 5.11 φαίνεται ο συνδιασμένος διάδρομος δεδομένων.



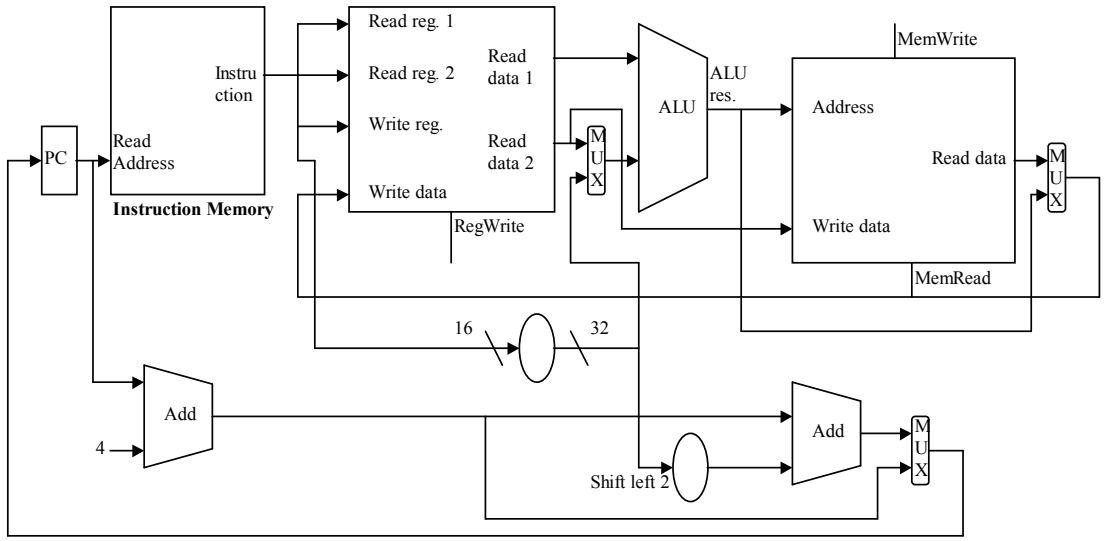
Σχήμα 4.9: Συνδυασμένος διάδρομος δεδομένων

Στο Σχήμα 4.10 φαίνεται η πρόσθεση του Σχήματος 4.3 στο Σχήμα 4.9, δηλ. ο διάδρομος δεδομένων για τη προσκόμιση της εντολής από τη μνήμη προστίθεται στο διάδρομο δεδομένων που χειρίζεται τις εντολές μνήμης και τις R-type εντολές.



Σχήμα 4.10: Συνδυασμένος διάδρομος δεδομένων

Στο Σχήμα 4.11 (αποτελεί τη πρόσθεση του διαδρόμου δεδομένων για διακλάδωση στο Σχήμα 4.10) φαίνεται ο απλός διάδρομος δεδομένων για την αρχιτεκτονική MIPS. Αυτός ο διάδρομος δεδομένων εκτελεί τις βασικές εντολές του MIPS σε ένα κύκλο ρολογιού.



Σχήμα 4.11: Συνδυασμένος διάδρομος δεδομένων

4.7 Έλεγχος της Αριθμητικής και Λογικής Μονάδας (ALU control)

Η ALU έχει τρεις εισόδους ελέγχου, αλλά μόνο πέντε από τις οκτώ (2^3) συνδυασμένες εισόδους στο δυαδικό σύστημα χρησιμοποιούνται, όπως φαίνεται στο πιο κάτω πίνακα. Ανάλογα με το τύπο της εντολής η ALU πρέπει να εκτελέσει μια από αυτές τις πέντε λειτουργίες (πράξεις).

ALU control input	Function
000	And
001	Or
010	Add
110	Subtract
111	Set-on-less-than

Για τις εντολές μνήμης χρησιμοποιούμε την ALU για υπολογισμό της διεύθυνσης μνήμης εκτελώντας τη πράξη της πρόσθεσης.

Για τις εντολές τύπου R η ALU χρειάζεται να εκτελέσει μια από τις πέντε πράξεις ανάλογα με την τιμή του 6-bit πεδίου της εντολή (function field).

Για την branch equal εντολή η ALU πρέπει να εκτελέσει την πράξη της αφαίρεσης.

Τα 3-bit ελέγχου της ALU (ALU control inputs) μπορούν να παραχθούν από μια μικρή μονάδα ελέγχου που έχει ως εισόδους το πεδίο function της εντολής και ένα πεδίο ελέγχου 2-bit που θα ονομάζουμε ALUOp.

Το ALUOp καθορίζεται από το opcode πεδίο της εντολής και προσδιορίζει τη πράξη που πρέπει να εκτελεστεί, π.χ 00 - πρόσθεση για την εντολή load και store, 01 - αφαίρεση για την εντολή beq, 10 - λειτουργία που προσδιορίζεται από το πεδίο function της εντολής.

Στον πιο κάτω πίνακα φαίνεται πως θέτονται τα 3-bit ελέγχου της ALU βασιζόμενοι στο 2-bit πεδίο ALUOp και στο 6-bit πεδίο function.

Instruction opcode	ALUOp	Instruction operation	Function code	Desired ALU action	ALU control Input
LW	00	load word	xxxxxx	add	010
SW	00	store word	xxxxxx	add	010
Branch equal	01	branch equal	xxxxxx	subtract	110
R-type	10	add	100000	add	010
R-type	10	subtract	100010	subtract	110
R-type	10	AND	100100	and	000
R-type	10	OR	100101	or	001
R-type	10	set-on-less-than	101010	set-on-less-than	111

Σχήμα 4.12: Οι τιμές των 3 - bit ελέγχου της ALU.

Στο Σχήμα 4.13 φαίνεται το μπλοκ ελέγχου της ALU (ALU control block) το οποίο παράγει τα 3 bits βασιζόμενο πάνω στο function code και στο ALUOp πεδίο.

ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	Operation
0	0	x	x	x	x	x	x	010
x	1	x	x	x	x	x	x	110
1	x	x	x	0	0	0	0	010
1	x	x	x	0	0	1	0	110
1	x	x	x	0	1	0	0	000
1	x	x	x	0	1	0	1	001
1	x	x	x	1	0	1	0	111

Σχήμα 4.13: Ο πίνακας αληθείας για τα 3 bits ελέγχου του ALU

4.8 Σχεδιασμός του Κύριου Μέρους της Μονάδας Ελέγχου

Στο Σχήμα 4.14 φαίνεται η δομή του κάθε τύπου εντολής, δηλαδή, των εντολών R-type, των εντολών διακλάδωσης (Branch) και των εντολών μνήμης (Load & Store).

0	rs	rt	rd	shamt	funct
31 - 26	25 - 21	20 - 16	15 - 11	10 - 6	5 - 0

35 or 43	rs	rt	address
31 - 26	25 - 21	20 - 16	15 - 0

4	rs	rt	address
31 - 26	25 - 21	20 - 16	15 - 0

Σχήμα 4.14: ***Δομή εντολών***

Σημαντικές παρατηρήσεις από το πιο πάνω σχήμα :

1. Το opcode πεδίο περιέχεται πάντοτε στα bits 31-26. Θα αναφερόμαστε σε αυτό το πεδίο με τον όρο Op[5-0].

2. Οι δύο καταχωρητές που πρέπει να διαβαστούν καθορίζονται πάντα από τα πεδία rs και rt στις θέσεις 25-21 και 20-16, αντίστοιχα.

3. Ο βασικός καταχωρητής (base register) για τις εντολές load και store είναι πάντοτε στη θέση 25-21 (rs).

4. Το 16-bit πεδίο offset για τις εντολές branch equal, load και store βρίσκεται στη θέση 15-0.

5. Ο καταχωρητής προορισμού (destination) βρίσκεται στη θέση 20-16 (rt) για την εντολή load ενώ για τις εντολές R-type βρίσκεται στη θέση 15-11 bits. Έτσι χρειάζεται να προσθέσουμε ένα πολυπλέκτη για να διαλέγει πιο πεδίο της εντολής χρησιμοποιείται για προσδιορισμό του αριθμού του καταχωρητή που θα διαβάσει.

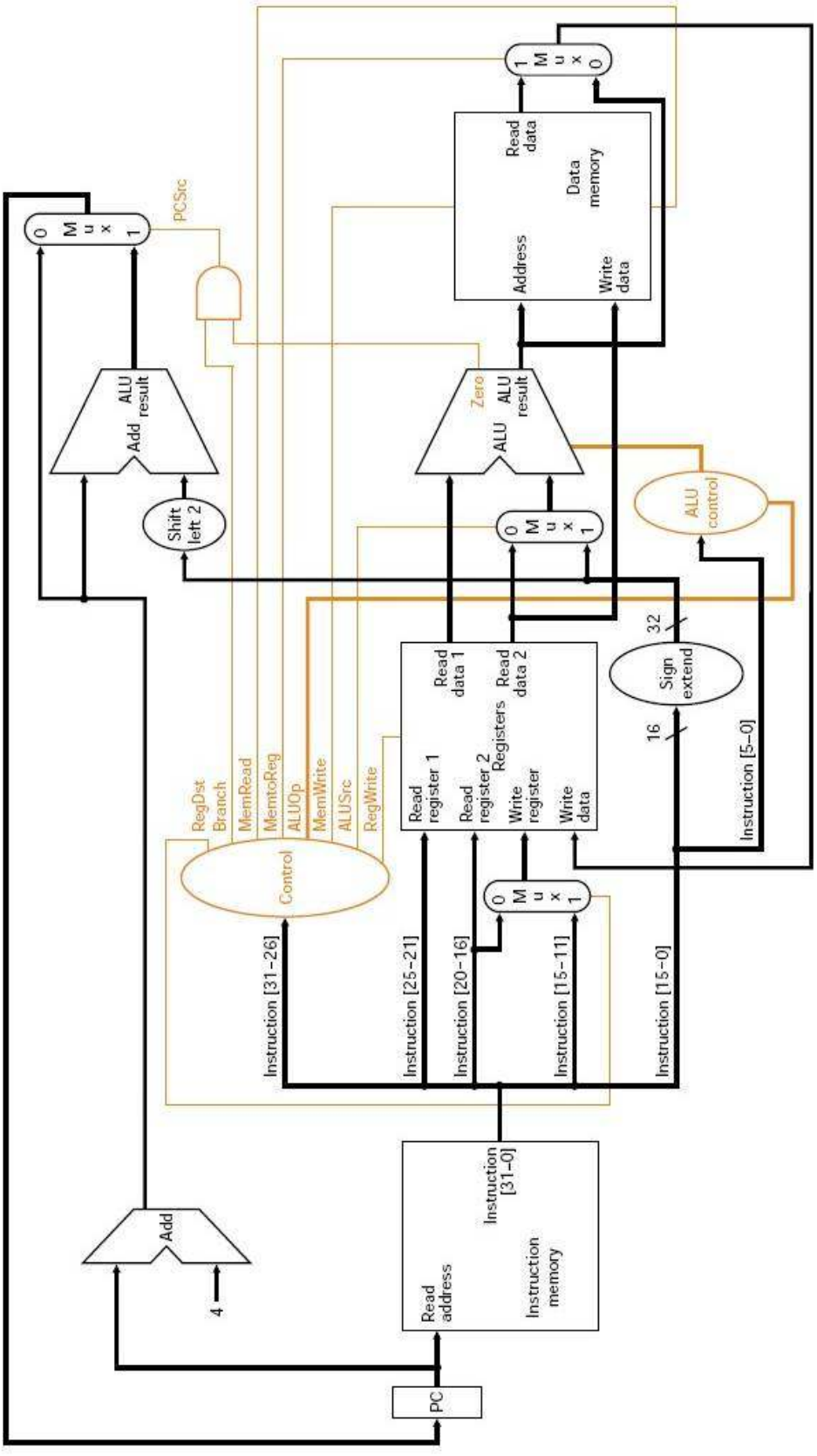
Το επόμενο σχήμα δείχνει τον μονό (απλό) διάδρομο δεδομένων για την αρχιτεκτονική MIPS, συνδυάζοντας όλα τα στοιχεία που χρειάζονται για κάθε διαφορετικό τύπο εντολής (Σχήμα 5.13), συμπεριλαμβανομένων όλων των απαραίτητων πολυπλεκτών, όλες τις γραμμές ελέγχου και το μπλοκ ελέγχου της ALU. Όλοι οι πολυπλέκτες έχουν δύο εισόδους, ο καθένας χρειάζεται μια γραμμή ελέγχου. Στο σχήμα φαίνονται επίσης οι 7 single-bit γραμμές ελέγχου και το 2-bit σήμα ελέγχου ALUOp.

Ο παρακάτω πίνακας περιγράφει τη λειτουργία των πιο πάνω επτά γραμμών ελέγχου:

Signal name	Effect when deasserted	Effect when asserted
MemRead	None	Data memory contents at the read address are put on read data output.
MemWrite	None	Data memory contents at address given by write address is replaced by value on write data input.
ALUSrc	The second ALU operand comes from the second register file output.	The second ALU operand is the sign-extended lower 16-bits of the instruction.
RegDst	The register destination number for the Write register comes from the rt field.	The register destination number for the Write register comew from the rd field.
RegWrite	None	The register on the Write register input is written into with the value on the write data input.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC+4.	The PC is replaced by the output of the adder that computes the branch target.
MemtoReg	The value fed to the register write data input comes from the ALU.	The value fed to the register write data input comes from the data memory.

Σχήμα 4.15: ***Λειτουργία των επτά γραμμών ελέγχου.***

Στο επόμενο σχήμα (3.16) φαίνεται ο μονός (απλός) διάδρομος δεδομένων μαζί με την μονάδα ελέγχου. Η είσοδος στη μονάδα ελέγχου είναι το 6-bit πεδίο opcode της εντολής. Η έξοδος της μονάδας ελέγχου αποτελείται από 3 σήματα (RegDst,AluScr,MemtoReg) του 1 bit που χρησιμοποιούνται για έλεγχο των πολυπλεκτών, 3 σήματα (RegWrite,MemRead,MemWrite) για έλεγχο γραφής και ανάγνωσης στο αρχείο των καταχωρητών και στη μνήμη δεδομένων., ένα σήμα του 1-bit που χρησιμοποιείται για καθορισμό του branch (Branch) και ένα σήμα των 2-bits για την ALU (ALUOp). Μια πύλη AND χρησιμοποιείται για συνδυασμό του σήματος ελέγχου branch με την έξοδο μηδέν από την ALU. Η έξοδος της πύλης AND ελέγχει την εκλογή του επόμενου PC.



Σχήμα 4.16: ***Δομή εντολών*****Διάδρομος δεδομένων (υλοποίηση ενός κύκλου) + μονάδα ελέγχου**

Το Σχήμα 5.20 είναι πολύ σημαντικό. Σ'αυτό το πίνακα ορίζονται οι τιμές (0,1,X) που πρέπει να πάρουν τα σήματα ελέγχου για κάθε πεδίο opcode της εντολής (αυτή η πληροφορία παράγεται κατευθείαν από τα Σχήματα 5.14, 5.18, 5.19).

Instruction	Reg Dst	ALU Src	Mem to Reg	Reg Write	Mem Read	Mem Write	Branch	ALU Op1	ALU Op0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	x	1	x	0	0	1	0	0	0
beq	x	0	x	0	0	0	1	0	1

Σχήμα 4.17: Τιμές των σημάτων ελέγχου για το πεδίο opcode.

Θα παρατηρήσουμε τώρα προσεκτικά τη ροή των τριών διαφορετικών τύπων εντολών διαμέσου του διαδρόμου δεδομένων.

Εκτέλεση της εντολής R-type

Δίνεται η εντολή add $x,$y,$z.
Τα τέσσερα βήματα της εντολή τύπου R :

1. Μια εντολή προσκομίζεται από τη μνήμη εντολών και ο PC αυξάνεται
2. Οι καταχωρητές $y και $z διαβάζονται από το αρχείο των καταχωρητών. Επίσης η κύρια μονάδα ελέγχου υπολογίζει τις τιμές για τις γραμμές ελέγχου.
3. Το ALU λειτουργεί πάνω στα δεδομένα που διαβάστηκαν από το αρχείο των καταχωρητών χρησιμοποιώντας το function code της εντολής, για να δημιουργήσει την λειτουργία της ALU.
4. Το αποτέλεσμα από την ALU γράφεται στο αρχείο των καταχωρητών χρησιμοποιώντας bits 15-11 από την εντολή για να επιλέξει τον καταχωρητή προορισμού (εξόδου) $x .

Εκτέλεση της εντολής load

Δίνεται η εντολή lw $x,offset($y). Η εκτέλεση της εντολής load αποτελείται από πέντε βήματα :

1. Η εντολή προσκομίζεται από τη μνήμη εντολών και αυξάνεται η τιμή του PC.
2. Η τιμή κάποιου καταχωρητή ($y) διαβάζεται από το αρχείο των καταχωρητών.
3. Η ALU υπολογίζει το άθροισμα της τιμής που διαβάστηκε από το αρχείο των καταχωρητών και την προέκταση σήματος 16bits της εντολής.
4. Το άθροισμα από το ALU χρησιμοποιείται ως η διεύθυνση για τη μνήμη των δεδομένων.
5. Τα δεδομένα από τη μονάδα μνήμης γράφονται στο αρχείο του καταχωρητή. Ο καταχωρητής προορισμού (εξόδου) δίνεται από τα 20-16 bits της εντολής ($x).

Η εντολή store λειτουργεί περίπου το ίδιο. Η κυριότερη διαφορά είναι στο έλεγχο μνήμης όπου πρέπει να προσδιοριστεί έλεγχος γραφής και όχι διαβάσματος. Η δεύτερη τιμή που

διαβάζεται από το καταχωρητή (offset) θα χρησιμοποιείται για αποθήκευση δεδομένων και η λειτουργία γραφής των δεδομένων της μνήμης στο αρχείο των καταχωρητών δεν θα υπάρχει.

Εκτέλεση της εντολής branch-on-equal (υπό συνθήκη):
Δίνεται η εντολή beq $x,$y,offset. Τα τέσσερα βήματα της εκτέλεσης της εντολής είναι:

1. Η εντολή προσκομίζεται από τη μνήμη εντολών και αυξάνεται η τιμή του PC.
2. Δύο καταχωρητές ($x,$y) διαβάζονται από το αρχείο των καταχωρητών.
3. Η ALU εκτελεί μια αφαίρεση ανάμεσα στις τιμές των δεδομένων που διάβασε από το αρχείο των καταχωρητών. Η τιμή του PC+4 προστίθεται στο sign-extended lower 16 bits της εντολής (offset), το αποτέλεσμα είναι η διεύθυνση - στόχος του branch.
4. Το αποτέλεσμα μηδέν από την ALU χρησιμοποιείται για να αποφασίσουμε το αποτέλεσμα του αθροιστή που θα αποθηκεύσουμε στο PC.

Τώρα που είδαμε τα βήματα λειτουργίας των εντολών, μπορούμε να συνεχίσουμε με την υλοποίηση της μονάδας ελέγχου. Η λειτουργία ελέγχου καθορίζεται πλήρως από το Σχήμα 5.20. Οι έξοδοι είναι οι γραμμές ελέγχου και η είσοδος είναι το 6-bits πεδίο opcode. Έτσι μπορούμε να δημιουργήσουμε ένα πίνακα αληθείας για κάθε έξοδο.

Στον πιο κάτω πίνακα φαίνεται η κωδικοποίηση της κάθε εντολής (opcode field) σε δυαδική και σε δεκαδική μορφή.

Opcode in binary

Name	Opcode in decimal	Op5	Op4	Op3	Op2	Op1	Op0
R-format	0	0	0	0	0	0	0
lw	35	1	0	0	0	1	1
sw	43	1	0	1	0	1	1
beq	4	0	0	0	1	0	0

Χρησιμοποιώντας αυτό το πίνακα μπορούμε να περιγράψουμε τη λογική της υλοποίησης της μονάδας ελέγχου σε ένα μεγάλο πίνακα αληθείας που συνδυάζει όλες τις εξόδους, όπως φαίνεται στο Σχήμα 4.18. Ο πίνακας καθορίζει πλήρως τη λειτουργία ελέγχου και μπορούμε να τον υλοποιήσουμε κατευθείαν με πύλες.

		R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	x	x
	ALUSrc	0	1	1	0
	MemtoReg	0	1	x	x
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

Σχήμα 4.18: Πίνακας αληθείας της λειτουργίας της μονάδας ελέγχου.

Παράδειγμα 2

Μέχρι τώρα είδαμε την υλοποίηση πολλών από τις εντολές που εξετάσαμε στο κεφάλαιο 3. Μια τάξη των εντολών που απουσιάζει είναι η εντολή jump. Περιγράψτε πως μπορεί να επεκταθεί η υλοποίηση του datapath ώστε να περιέχει και την εντολή jump.

Απάντηση

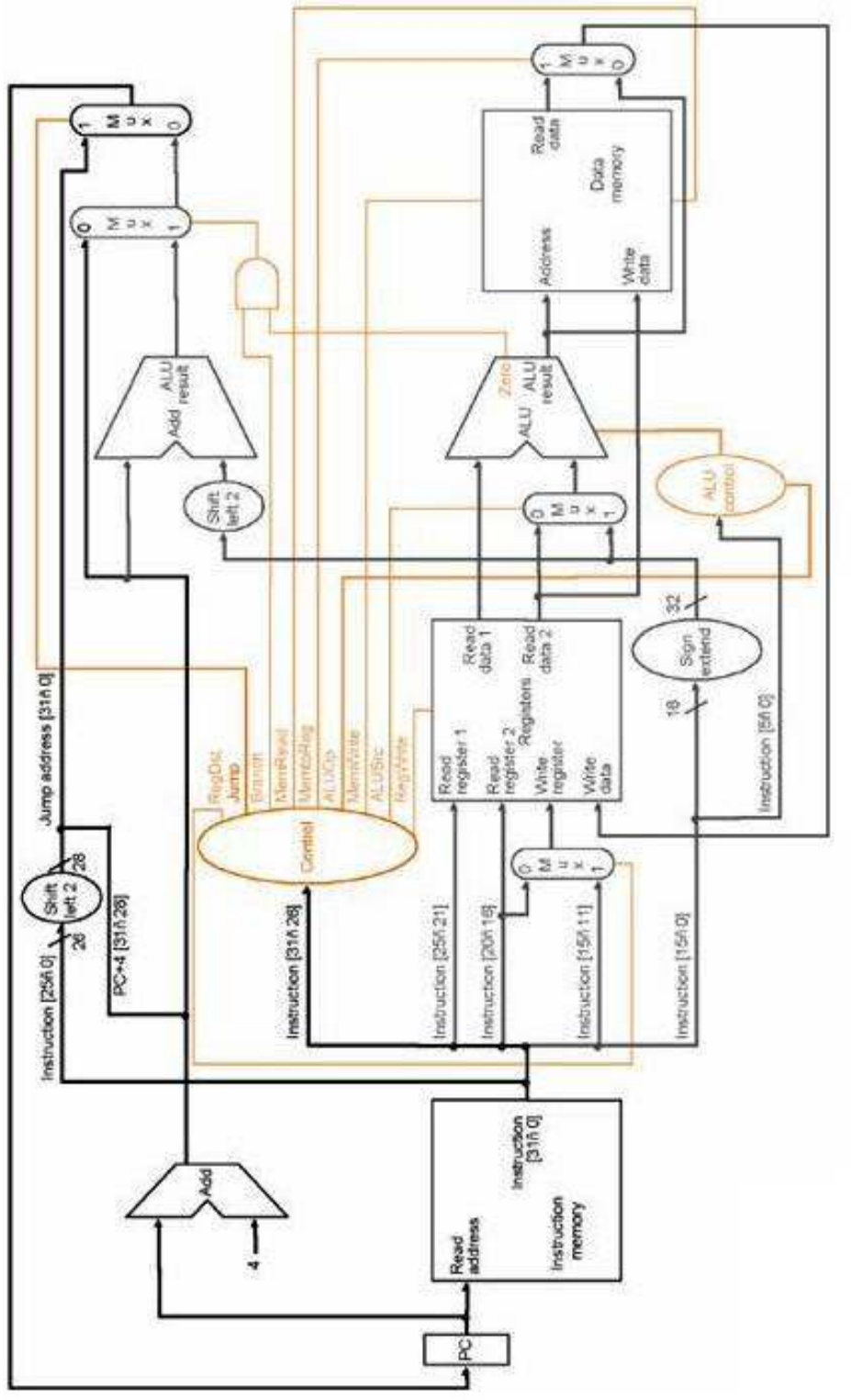
Η εντολή jump μοιάζει με την εντολή branch, αλλά υπολογίζει διαφορετικά τη διεύθυνση-στόχος PC και δεν είναι υποθετική. Όπως στη branch, τα low order 2 bits της διεύθυνσης (της jump) είναι πάντοτε 00. Τα υπόλοιπα lower 26 bits από αυτά της 32 bits διεύθυνσης προέρχονται από το 26 bit immediate πεδίο της εντολής. Τα upper 4 bits της διεύθυνσης που θα πρέπει να αντικαταστήσουν το PC προέρχονται από το τρέχον PC. Έτσι μπορούμε να υλοποιήσουμε την jump αποθηκεύοντας στο PC την ένωση των :

- upper four bits του τρέχον PC (bits 31-28)
- τα 26 bits του πεδίου immediate της εντολής jump.
- τα bits 00

Χρειαζόμαστε έναν επιπλέον πολυπλέκτη για επιλογή της πηγής για τη καινούργια τιμή του PC που θα είναι είτε PC+4 , είτε branch target PC, είτε jump target PC. Ένα επιπλέον σήμα ελέγχου χρειάζεται για τον επιπρόσθετο πολυπλέκτη. Αυτό το σήμα ελέγχου ονομάζεται Jump και είναι asserted μόνο όταν η εντολή είναι jump, δηλ. όταν το opcode είναι 2.

2	address
31 - 26	25 - 0

Σχήμα 4.19: Η μορφή της εντολής jump (opcode = 2)



Σχήμα 4.20: Datapath το οποίο υποστηρίζει και την jump

4.9 Τι είναι λάθος με την υλοποίηση ενός κύκλου

Εξ΄ορισμού ο κύκλος ρολογιού πρέπει να έχει το ίδιο μήκος (μέγεθος) για κάθε εντολή, άρα το CPI θα είναι ίσο με 1. Φυσικά ο κύκλος ρολογιού καθορίζεται από το μεγαλύτερο πιθανό

μονοπάτι στη μηχανή. Αυτό το μονοπάτι είναι σίγουρα η εντολή load που χρησιμοποιεί πέντε λειτουργικές μονάδες στη σειρά : μνήμη εντολών, αρχείο των καταχωρητών, ALU, μνήμη δεδομένων και ξανά το αρχείο καταχωρητών. Αν και το CPI είναι 1, η ολική αποδοτικότητα της υλοποίησης ενός κύκλου δεν είναι πολύ καλή αφού μερικοί τύποι εντολών μπορούσαν να εκτελεστούν σε μικρότερο κύκλου ρολογιού.

Παράδειγμα 3

Υπέθεσε ότι ο χρόνος λειτουργίας των κύριων λειτουργικών μονάδων για αυτή την υλοποίηση είναι :

- Μονάδα μνήμης: 10ns
- Αριθμητική και λογική μονάδα (ALU) και αθροιστές (Adders) : 10ns
- Αρχείο των καταχωρητών (διάβασμα ή γραφή) : 5ns

Υποθέτοντας ότι οι πολυπλέκτες, η μονάδα ελέγχου, οι προσβάσεις του PC, η μονάδα προέκτασης του πρόσημου και τα καλώδια δεν έχουν καθυστέρηση, ποία από τις ακόλουθες υλοποιήσεις θα είναι πιο γρήγορη και κατά πόσο;

1. Μια υλοποίηση στην οποία κάθε εντολή εκτελείται σε ένα κύκλο ρολογιού καθορισμένου μήκους.
2. Μια υλοποίηση στην οποία κάθε εντολή εκτελείται σε ένα κύκλο ρολογιού χρησιμοποιώντας μεταβλητό μήκος ρολογιού, το οποίο για κάθε εντολή είναι το μικρότερο δυνατό.

Απάντηση

Συγκρίνουμε αρχικά τους χρόνους εκτέλεσης του CPU.

CPU execution time = Instruction count x CPI x Clock cycle time

Αφού το CPI πρέπει να είναι 1, απλοποιούμε τον τύπο σε:

CPU execution time = Instruction count x Clock cycle time

Χρειάζεται να βρούμε το Clock cycle time για τις δύο υλοποιήσεις. Το critical path για τις διαφορετικές εντολές φαίνεται στο πιο κάτω πίνακα:

Instruction Type	Functional units used by the Instruction type				
R-format	Instruction fetch	Register Access	ALU	Register access	
Load word	Instruction fetch	Register Access	ALU	Memory access	Register access
Store word	Instruction fetch	Register Access	ALU	Memory access	
Branch	Instruction fetch	Register Access	ALU		
Jump	Instruction fetch				

Χρησιμοποιώντας αυτούς τους διαδρόμους μπορούμε να υπολογίσουμε το απαιτούμενο μέγεθος για κάθε τύπο εντολής :

Instruction type	Instruction memory	Register read	ALU operation	Data memory	Register write	Total
R-format	10	5	10	0	5	30ns
Load word	10	5	10	10	5	40ns
Store word	10	5	10	10		35ns
Branch	10	5	10	0		25ns
Jump	10					10ns

Έτσι ο κύκλος ρολογιού για τη μηχανή με ένα κύκλο για όλες τις εντολές είναι 40ns, ενώ για τη μηχανή με μεταβλητό κύκλο θα έχουμε ένα κύκλο μεταξύ των 10ns και 40ns.

Μπορούμε να βρούμε το μέσο μέγεθος του κύκλου ρολογιού για τη δεύτερη μηχανή χρησιμοποιώντας τις πιο πάνω πληροφορίες και ένα καταμερισμό της συχνότητας εντολών. Αυτός ο καταμερισμός μπορεί να υπολογιστεί από το Σχήμα 4.46 του Κεφαλαίου 4, προσθέτοντας τις ξεχωριστές συχνότητες σε κατηγορίες :

- 22% loads,
- 11% stores,
- 49% R-format λειτουργίες,
- 16% branches
- 2% jumps.

Έτσι ο μέσος χρόνος κάθε εντολής με μεταβλητό κύκλο είναι :

$$\text{CPU clock cycle} = 40\text{x}22\% + 35\text{x}11\% + 30\text{x}49\% + 25\text{x}16\% + 10\text{x}2\% = 31.6 \text{ ns.}$$

Αφού η υλοποίηση του μεταβλητού κύκλου έχει μικρότερο μέσο κύκλο ρολογιού σημαίνει ότι είναι και γρηγορότερη. Ας βρούμε τώρα την αναλογιά της απόδοσης :

$$\frac{\text{CPU performance variable clock}}{\text{CPU performance single clock}} = \frac{\text{CPU execution time single clock}}{\text{CPU execution time variable clock}}$$

$$= \frac{\text{IC x CPU clock cycle single clock}}{\text{IC x CPU clock cycle variable clock}}$$

$$\frac{\text{CPU performance variable clock}}{\text{CPU performance single clock}} = \frac{\text{CPU clock cycle single clock}}{\text{CPU clock cycle variable clock}}$$

$$= \frac{40}{31.6} = 1.27$$

Η υλοποίηση με μεταβλητό ρολόι θα ήταν 1.27 φορές γρηγορότερη, δυστυχώς όμως η υλοποίηση του μεταβλητού κύκλου ρολογιού για κάθε εντολή είναι πολύ δύσκολη και το κόστος πολύ μεγάλο.

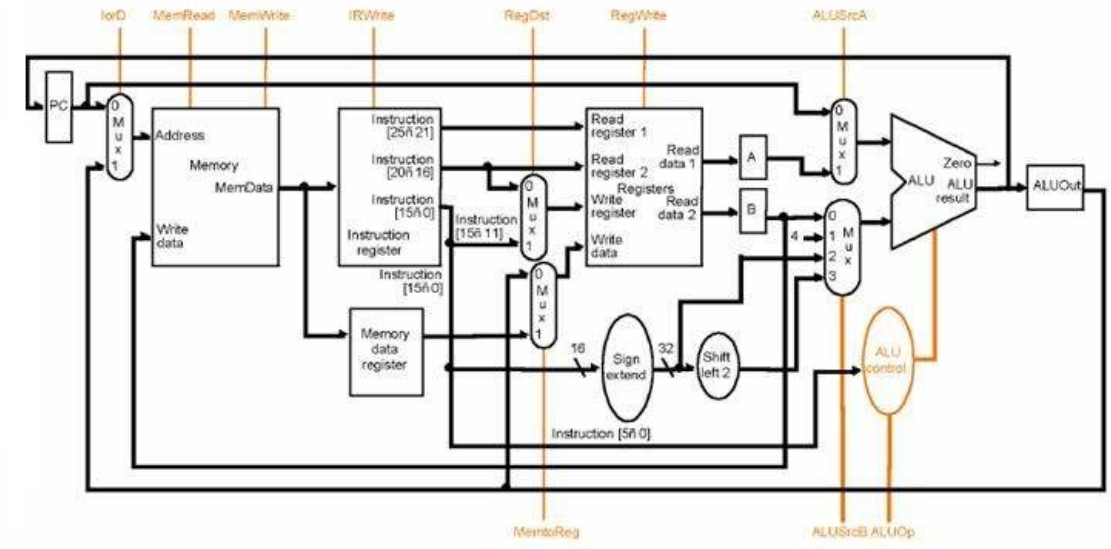
4.10 Υλοποίηση πολλαπλών κύκλων

Στην πραγματικότητα, ο έλεγχος δεν μπορεί να είναι καθαρά συνδυαστικός, και δεν μας αρκεί μία μόνο ακμή ρολογιού ανά εντολή, καθώς για παράδειγμα το σήμα WE της μνήμης δεν επιτρέπεται να είναι ασταθές κατά το πρώτο μέρος του κύκλου. Η επίλυση αυτού του προβλήματος βρίσκεται στην χρησιμοποίηση πρόσθετων ακμών ρολογιού μέσα στον κύκλο, δηλαδή ουσιαστικά σε υλοποίηση με περισσότερους από ένα κύκλο ρολογιού ανά εντολή!

4.10.1 Αναχρησιμοποίηση Μονάδων

Με 1 κύκλο/ εντολή (υλοποίηση ενός κύκλου ρολογιού), κάθε μονάδα hardware μπορεί να χρησιμοποιηθεί μόνο μια φορά ανά κύκλο. Έτσι είμαστε υποχρεωμένοι να χρησιμοποιούμε ίδιου τύπου hardware περισσότερες από μία φορές (πχ 2 ALU,), πράγμα που δημιουργεί σπατάλη και αυξάνει το κόστος κατασκευής. Αντίθετα, στην υλοποίηση πολλαπλών κύκλων ρολογιού χρησιμοποιείται το ίδιο harware, αλλά με διαφορετικό τρόπο σε κάθε κύκλο καθώς μπορεί να ανατροφοδοτηθεί περισσότερες από μία φορές σε κάθε εντολή. Για παράδειγμα, μία ALU μπορεί να χρησιμοποιηθεί και για αύξηση PC, υπολογισμό διακλαδώσεων, αριθμητικές πράξεις κλπ.

Λόγω της δυνατότητας επαναχρησιμοποίησης στοιχείων hardware σε διαφορετικούς κύκλους, στην υλοποίηση πολλαπλών κύκλων χρειαζόμαστε ένα στοιχείο μνήμης, μία ALU και αρκετούς πολυπλέκτες! Επίσης, πρέπει να χρησιμοποιούμε επιπλέον καταχωρητές ώστε να αποθηκεύουμε τις τιμές μονάδων όπως η μνήμη οι οποίες ενδέχεται να αλλάξουν κατάσταση κατά την εναλλαγή των κύκλων εκτέλεσης μίας εντολής και έτσι να χαθούν πολύτιμα δεδομένα. Ένας τέτοιος καταχωρητής είναι ο 'Instruction Register' ο οποίος κρατάει σταθερή την εντολή που εκτελείται κάθε στιγμή



Σχήμα 4.21: ***Γενικό Datapath υλοποίησης πολλαπλών***

Ο κύκλος θα έχει τόση διάρκεια όσο η μακρύτερη εργασία, αλλά πρέπει να προσέξουμε να κατανέμουμε τις εργασίες με τέτοιο τρόπο ώστε να εκμεταλλευτούμε όλη τη διάρκεια του κύκλου διασφαλίζοντας όσο το δυνατόν καλύτερη εξισορρόπηση διάρκειας εργασιών ανά κύκλο. Για παράδειγμα, σε κάποιες περιπτώσεις θα χρησιμοποιήσουμε hardware για προετοιμασία αποτελεσμάτων που είναι πιθανό (αλλά όχι σίγουρο) να χρειαστούν. Σε κάθε περίπτωση εμείς σε

κάθε κύκλο θα κάνουμε το πολύ μία προσπέλαση στη μονάδα μνήμης, μία προσπέλαση στο αρχείο καταχωτητών (RF) και μία χρήση της αριθμητικής/λογικής μονάδας (ALU).

ΚΥΚΛΟΣ 1 *(ίδιος για όλες τις εντολές)*

Ανάγνωση της εντολής από τη μνήμη Instruction Fetch: IR←M[PC]
Αύξηση του PC ώστε να 'δείξει' την επόμενη εντολή: PC←PC+4

ΚΥΚΛΟΣ 2 *(ίδιος για όλες τις εντολές)*

Ανάγνωση καταχωρητών: read RF [rs], RF[rt]
Υπολογισμός της διεύθυνσης Branch: Target←PC+4*Imm16
Αποκωδικοποίηση του Opcode, '*αναγνώριση της εντολής*'

ΚΥΚΛΟΣ 3 *(διαφέρει ανάλογα με την εντολή)*

i. Αν η εντολή είναι ALU/LW/SW: πράξη ALU

ii. Αν η εντολή είναι jump: PC←Imm26

iii. Αν η εντολή είναι Branch:
ALU: συγκρίνει τα περιεχόμενα των δύο καταχωρητών αφαιρώντας τα. Αν το αποτέλεσμα είναι 0 (zero flag) τότε: PC←Target

Σημείωση: Στον τρίτο κύκλο των branch, η ALU είναι απασχολημένη. Χρειαζόμαστε το PC+4*Imm16. Αυτό το κάνουμε νωρίτερα- είτε χρειάζεται είτε όχι (πριν μάθουμε αν η εντολή ήταν branch!)- στον 2ο κύκλο που η ALU ¨κάθονταν¨ ο προσωρινός καταχωρητής ¨Target¨ χρειάζεται για να κρατήσει αυτό το προσωρινό αποτέλεσμα, μέχρι να γίνει γνωστό αν αυτό πρέπει να μπει στο PC ή όχι.

ΚΥΚΛΟΣ 4 *(διαφέρει ανάλογα με την εντολή)*

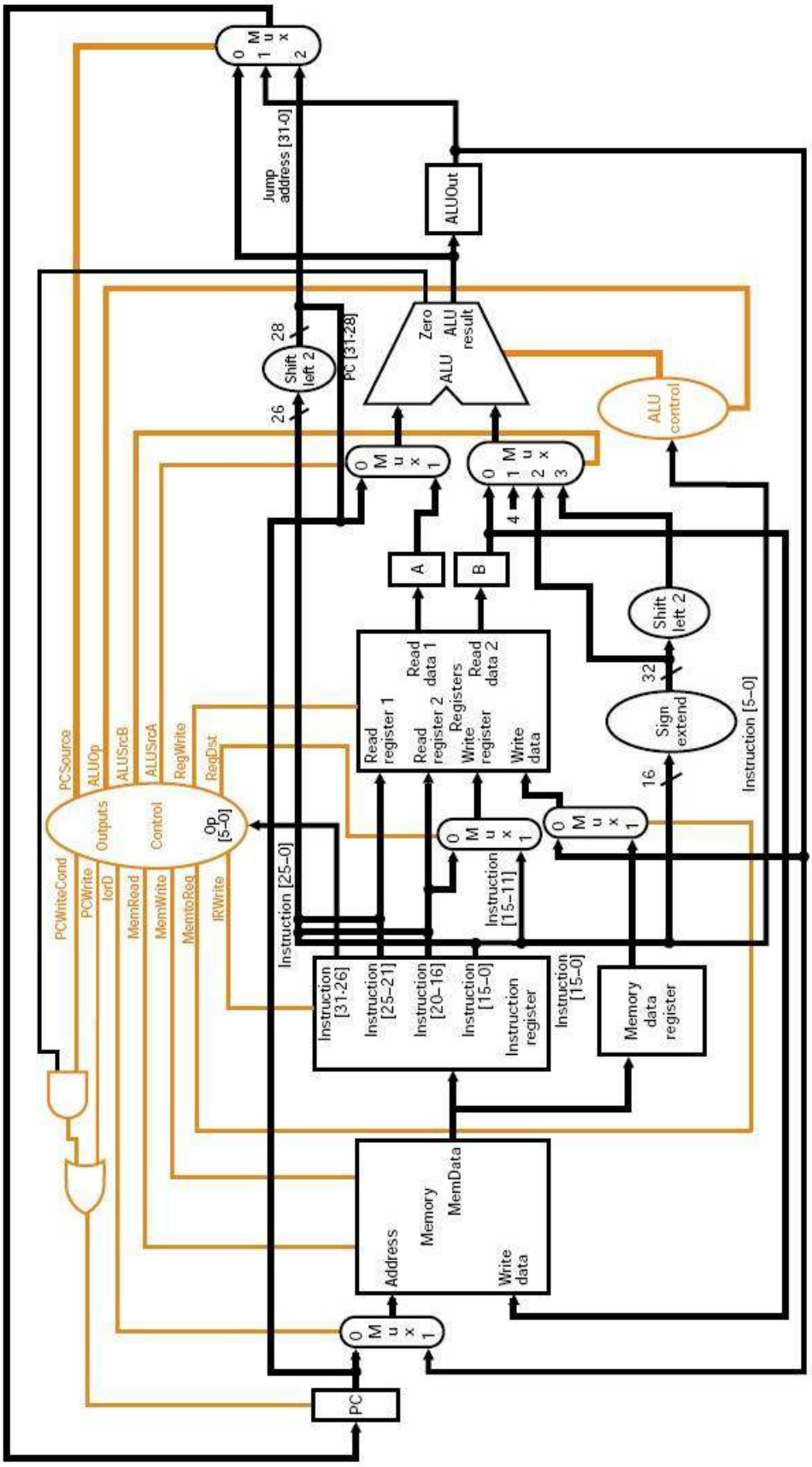
i. Αν η εντολή είναι ALU: Το αποτέλεσμα της ALU πηγαίνει σε κάποιο καταχωρητή: RF[rd/rt]←ALU

ii. Αν η εντολή είναι η lw: Το αποτέλεσμα της ALU είναι μία θέση μνήμης, από την οποία πρέπει να διαβάσουμε: Memory Read: read M [ALU]

iii. Αν η εντολή είναι η sw: Το αποτέλεσμα της ALU είναι μία θέση μνήμης, στην οποία πρέπει να γράψουμε το περιεχόμενο κάποιου καταχωρητή: Memory Write: M[ALU]←RF[rt]

ΚΥΚΛΟΣ 5 (μόνο για την lw)

i. Το περιεχόμενο της μνήμης που διαβάσαμε στον 4ο κύκλο οδηγείται σε κάποιο καταχωρητή: Memory Read: RF[rt]← M[ALU]



Σχήμα 4.22: Datapath υλοποίησης πολλαπλών κύκλων με τη μονάδα ελέγχου

5.Ιεραρχίες μνήμης, και η εκμετάλλευση τους

5.1 Ιεραρχίες Μνημών Και η Ιδιότητα Της Τοπικότητας

Ας δούμε ένα παράδειγμα: ένας μαθητής κάθεται σένα τραπέζι μίας βιβλιοθήκης και γράφει μία εργασία. Χρησιμοποιεί μερικά βιβλία από την βιβλιοθήκη, που τα έχει φέρει μπροστά του στο τραπέζι προκειμένου να έχει άμεση και γρήγορη πρόσβαση στις πληροφορίες που περιέχουν. Αν χρειαστεί κάποιο άλλο βιβλίο, μπορεί να πάει και το φέρνει από τα ράφια, αλλά τότε χρειάζεται περισσότερο χρόνο πρόσβασης στις πληροφορίες. Όμως το τραπέζι έχει πεπερασμένη χωρητικότητα και άρα ο μαθητής δεν μπορεί να κρατήσει το νέο βιβλίο που έφερε στο τραπέζι μαζί με τα προηγούμενα, οπότε παίρνοντας το νέο βιβλίο από το ράφι επιστρέφει ταυτόχρονα πίσω στα ράφια ένα από τα βιβλία που είχε πριν στο τραπέζι. Το βιβλίο που φέρνει τελευταίο στο τραπέζι, έχει μεγάλη πιθανότητα να το ξανάχρειαστεί στο άμεσο μέλλον, ενώ είναι πολύ πιθανό να τον ενδιαφέρουν και τα θέματα που αναλύονται στα υπόλοιπα κεφάλαια του βιβλίου.

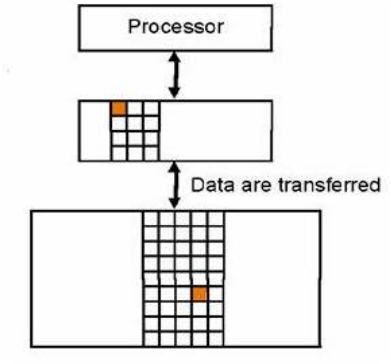
Με τον ίδιο ακριβώς τρόπο που ενεργεί ο μαθητής, λειτουργεί και ένας επεξεργαστής, προσπαθώντας να κρατάει κοντά του τις πληροφορίες που θεωρεί ότι πρόκειται να χρησιμοποιήσει σύντομα. Για το λόγω αυτό ο επεξεργαστής διαρκώς φέρνει κοντά του αντίτυπα των εντολών (λέξεων) από την κύρια μνήμη (ράφια βιβλιοθήκης) στην **κρυφή μνήμη** (cache) η οποία παίζει το ρόλο του τραπεζιού.

Η λειτουργία της κρυφής μνήμης στηρίζεται στην αρχή της τοπικότητας στο χρόνο και στο χώρο:

¨Χρονική Τοπικότητα¨(Temporal Locality):
μια λέξη μνήμης που χρησιμοποιήθηκε πρόσφατα στο παρελθόν, έχει μεγάλη πιθανότητα να ξανά χρησιμοποιηθεί σύντομα στο μέλλον.

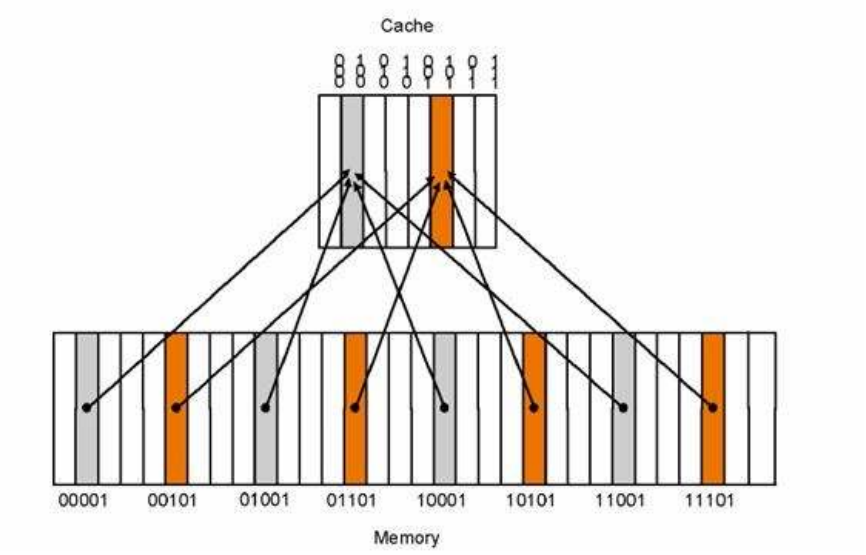
Τοπική Τοπικότητα: (Spatial Locality):
οι λέξεις κοντά σε μία λέξη που χρησιμοποιήθηκε πρόσφατα στο παρελθόν έχουν μεγάλη πιθανότητα να χρησιμοποιηθούν σύντομα στο μέλλον. (π.χ. διπλανές ή επόμενες εντολές κώδικα). Για το λόγο αυτό μεταφέρουμε ανάμεσα στην κύρια και στη κρυφή μνήμη ομάδες γειτονικών λέξεων που τις ορίζουμε ‘block’.

Όταν τοποθετούμε ένα αντίτυπο μιας λέξης από την κύρια μνήμη στην κρυφή μνήμη, αυτό μπορεί να τοποθετηθεί σε ορισμένες μόνο "επιτρεπτές" θέσεις της κρυφής μνήμη. Εάν όλες οι θέσεις της κρυφής μνήμης είναι επιτρεπτές, τότε η κρυφή μνήμη ονομάζεται ‘πλήρως προσεταιριστικές (fully associative cache). Το μειονέκτημα όμως της πλήρως προσεταιριστικής κρυφής μνήμης είναι το υψηλό της κόστος, καθώς κατά την αναζήτηση κάποιας λέξης πρέπει να ψάξουμε σε όλες τις θέσεις της κρυφής μνήμης.



Σχήμα 5.1: Σύνολο γειτονικών λέξεων που είναι παρούσες είτε όλες είτε καμία, σε κάποιο επίπεδο.

Για να επιτύχουμε μικρότερο κόστος υλοποίησης οργανώνουμε την κρυφή μνήμη ως ‘μονοσήμαντης απεικόνισης’ (direct mapped cache). Σε αυτή την οργάνωση, υπάρχει μία μόνο επιτρεπτή θέση στην κρυφή μνήμη για την κάθε λέξη της κύριας μνήμης. Έτσι, το ψάξιμο για μια λέξη είναι απλό: πηγαίνουμε στην μοναδική επιτρεπτή θέση για αυτή τη λέξη, και ελέγχουμε αν η λέξη που αναζητούμε βρίσκεται εκεί εκείνη τη στιγμή. Η επιτρεπτή αυτή θέση υποδηλώνεται από τα LS bit της διεύθυνσης όπως εξηγήσαμε στο μάθημα. Δεν πρέπει να ξεχνάτε ότι το πλήθος των LS bit που χρησιμοποιούμε καθορίζεται από το μέγεθος της κρυφής μνήμης, ενώ τα δύο τελευταία LS bit της διεύθυνσης είναι πάντα ‘00’ και άρα δεν τα λαμβάνουμε υπόψη κατά την μεταφορά λέξεων...



Σχήμα 5.2: Οργάνωση της μνήμης μονοσήμαντης απεικόνισης

Κάθε φορά ξέρουμε ποια λέξη έχει μεταφερθεί στην κρυφή μνήμη ελέγχοντας το Valid bit (το ‘bit εγκυρότητας’ valid υποδεικνύει αν αυτή η θέση της cache περιέχει κάτι ή είναι άδεια) και συγκρίνοντας το Address Tag με τα MS bit της διεύθυνσης που δεν χρησιμοποιήσαμε για να εντοπίσουμε το δείκτη (index) του πίνακα.

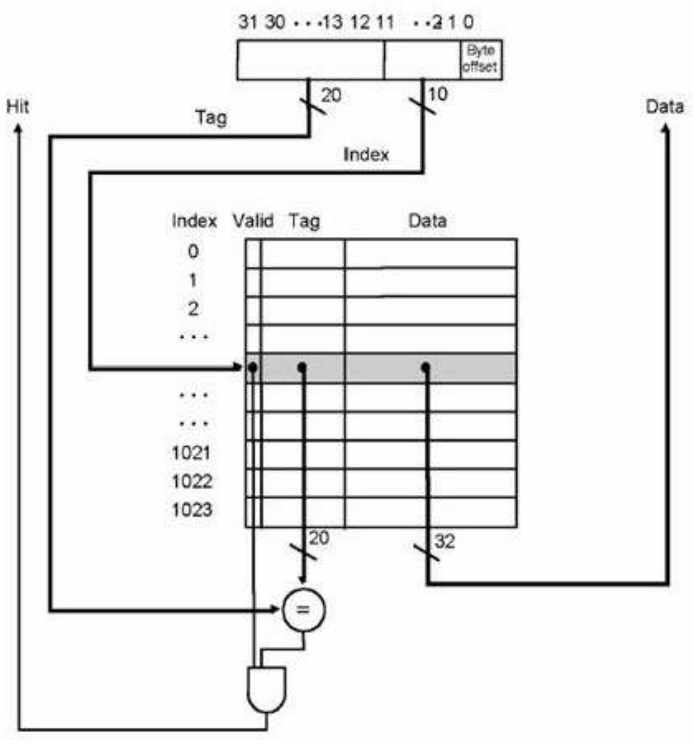
Φυσικά, το πρόβλημα με την μονοσήμαντη απεικόνιση είναι ότι μόνο μία από τις πολλές λέξεις της κύριας μνήμης που έχουν όλες την ίδια επιτρεπτή θέση στην κρυφή μνήμη μπορεί να βρίσκεται στην κρυφή μνήμη σε κάποια δεδομένη χρονική στιγμή.

Αν η αναζήτηση κάποιας λέξεις στην κρυφή μνήμη είναι επιτυχής, τότε έχουμε HIT (ευστοχία) και άρα βρίσκω αυτό που αναζητώ στην μικρή και γρήγορη κρυφή μνήμη. Σε αντίθετη περίπτωση έχω MISS (αστοχία), άρα το δεδομένο που ζητώ δεν το βρίσκω στην κρυφή μνήμη, και αναγκάζομαι να πάω στη μεγαλύτερη και αργότερη μνήμη να το αναζητήσω και να το μεταφέρω (μαζί με τα άλλα μέλη του block) στην κρυφή μνήμη.

5.2 ΑΝΑΓΝΩΣΗ κρυφής μνήμης:

Διαβάζουμε την θέση της κρυφής μνήμης που υποδεικνύουν τα LS bits της διεύθυνσης: διαβάζουμε τα: data, tag, valid bit και κάνουμε τον εξής έλεγχο:

Αν valid=1 ΚΑΙ tag= = MS bits διεύθυνσης Τότε έχουμε hit, άρα τα δεδομένα που αναζητώ τα διαβάζω από την cashe
Διαφορετικά: διαβάζουμε από κεντρική μνήμη και γράφουμε σε κρυφή μνήμη τα δεδομένα (data), και παράλληλα ορίζουμε νέες τιμές για το Address Tag, και valid (το κάνουμε 1 αμέσως ή αφού πρώτα αποθηκεύσουμε τα παλιά δεδομένα της cashe αν χρειάζεται...).



Σχήμα 5.3: ***Ανάγνωση από την κρυφή μνήμη***