ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 6

ΛΟΓΙΚΕΣ ΛΕΙΤΟΥΡΓΙΕΣ

ΜΕΡΟΣ Α

Οι περισσότερες λογικές λειτουργίες τις οποίες μπορεί να εκτελέσει ο MIPS, είναι γνωστές από την γλώσσα προγραμματισμού C:

Λογικές λειτουργίες	Τελεστές C	Εντολές MIPS
Shift left (ολίσθηση αριστερά)	<<	sll (shift left logical)
Shift right (ολίσθηση δεξιά)	>>	srl (shift right logical)
AND	&	and
OR	|	or
NOT	~	nor

Εκτός των ανωτέρω, στις εντολές MIPS περιλαμβάνονται και άλλες λογικές λειτουργίες, όπως η sra (shift right arithmetic) και οι ψευδοεντολές περιστροφής ror (rotate) right και rol (rotate left).

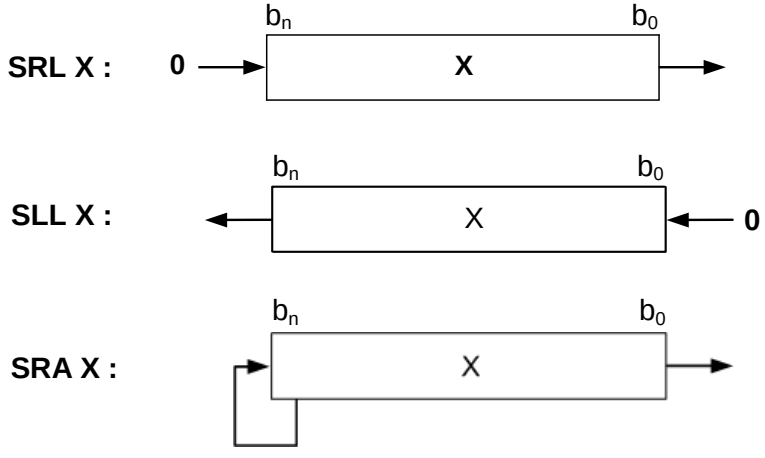
ΛΕΙΤΟΥΡΓΙΕΣ ΟΛΙΣΘΗΣΗΣ

Ονομασία των εντολών Ολίσθησης

Το πρώτο γράμμα δηλώνει την Ολίσθηση (**S**-Shift), το δεύτερο την κατεύθυνση (**L**-Left ή **R**-Right) και το τρίτο τον τρόπο που παίρνει τιμή το bit από το οποίο αρχίζει η ολίσθηση (**L**-Logical ή **A**-Arithmetic). Π.χ. SRA – Shift Right Arithmetic, SLL – Shift Left Logical. Για μεταβλητό αριθμό ολισθήσεων (τιμή εντός καταχωρητή), προστίθεται ως 4ο γράμμα το V (Value).

Λειτουργία της Ολίσθησης

Στην λειτουργία ολίσθησης οι τιμές των bit της θέσης αποθήκευσης μετατοπίζονται κατά την φορά της ολίσθησης. Οι τιμές των τελευταίων στην σειρά ολίσθησης bit χάνονται, ενώ τα πρώτα παίρνουν τιμή οριζόμενη από την συγκεκριμένη εντολή: 0 για τις εντολές τύπου logic ή την ίδια τιμή που είχαν πριν την ολίσθηση για τις εντολές τύπου arithmetic. Σχηματικά:



Αριθμητική σημασία της Ολίσθησης

Εκτός από την λογική λειτουργία της, η Ολίσθηση έχει και μία σημαντική αριθμητική σημασία:

- **Αριστερή Λογική Ολίσθηση = Πολλαπλασιασμός με την Βάση του ΑΣ**

  Π.χ (για αποθηκευτικό χώρο των 5 bit) :

  Αρχικά: 00111_2 (= 7_{10})

  Μετά το 1^o SLL: 01110_2 (= 14_{10} = 7 **x** 2)

  Μετά το 2^o SLL: 11100_2 (= 28_{10} = 14 **x** 2)

- **Δεξιά Λογική Ολίσθηση = Διαίρεση με την Βάση του ΑΣ**

  Π.χ (για αποθηκευτικό χώρο των 5 bit) :

  Αρχικά: 01110_2 (= 14_{10})

  Μετά το 1^o SRL: 00111_2 (= 7_{10} = 14 ÷ 2)

  Μετά το 2^o SRL: 00011_2 (= 3_{10} = 7 ÷ 2) (θεωρούμε το αποτέλεσμα της διαίρεσης μόνο ως προς το ακέραιο μέρος του)

  Παρατήρηση 1: για την τελευταία SRL, αν το bit που «πετιέται» το λάβουμε υπόψη ως ψηφίο μετά την υποδιαστολή, θα έχουμε $00011,1_2$, δηλαδή $3,5_{10} = 7 ÷ 2$ ακριβώς.

  *Παρατήρηση 2: Η συγκεκριμένη λειτουργία βρίσκει εφαρμογή μόνο για **μη** προσημασμένους αριθμούς*

- **Δεξιά Αριθμητική Ολίσθηση = Διαίρεση με την Βάση του ΑΣ (για προσημασμένους)**

  Έστω ότι έχουμε φορτώσει τον αριθμό -4_{10} σε έναν καταχωρητή των 8 bit. Τότε η τιμή του καταχωρητή θα είναι 1111 1100_2 (διότι αυτό δίνει το -4_{10} στο ΠΣ2 και στα 8 bit).

  Εκτέλεση της SRL θα δώσει 0111 1110_2 , δηλαδή $+126_{10}$, που είναι λάθος, αν το θεωρήσουμε ως αποτέλεσμα της διαίρεσης -4 ÷ 2.

  Εκτέλεση της SRA θα δώσει 1111 1110_2 , δηλαδή -2_{10}, που είναι σωστό ως αποτέλεσμα της διαίρεσης -4 ÷ 2.

  Δηλαδή, εάν χειριζόμαστε προσημασμένους αριθμούς και επιθυμούμε να υποδιπλασιάσουμε έναν αριθμό, πρέπει να χρησιμοποιήσουμε την SRA και όχι την SRL.

Κώδικας Μηχανής για τις εντολές Ολίσθησης

Ο κώδικας μηχανής (object code ή machine code) των εντολών ολίσθησης του MIPS, ακολουθεί τη γενική μορφή R:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
opcode						**rs**					**rt**					**rd**					**shamt**					**funct**					
6 bits						5 bits					5 bits					5 bits					5 bits					6 bits					

Όπου στο πεδίο shamt ορίζεται η ποσότητα (θέσεις) ολίσθησης.

Παράδειγμα Εντολής Ολίσθησης:

Έστω η εντολή: sll $t2, $s0, 4

Ολισθαίνουμε αριστερά το περιεχόμενο του $s0 κατά 4 θέσεις και το αποτέλεσμα το τοποθετούμε στον $t2.

Έστω ο $s0 πριν: $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1001_2 \equiv 9_{10}$

Ο $t2 μετά: $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1001\ 0000_2 \equiv 144_{10}$

Να υπενθυμίσουμε ότι το περιεχόμενο του $s0 μένει αμετάβλητο μετά την εκτέλεση της εντολής.

ΛΕΙΤΟΥΡΓΙΕΣ AND, OR, NOT

- Η εντολή AND είναι χρήσιμη όταν θέλουμε να μηδενίσουμε κάποια bits σε μια λέξη και τα υπόλοιπα να μείνουν αμετάβλητα.
- Η εντολή OR είναι χρήσιμη όταν θέλουμε να θέσουμε την τιμή 1 σε κάποια bits μιας λέξης και τα υπόλοιπα να μείνουν αμετάβλητα.
- Η εντολή NOR συνήθως χρησιμοποιείται όταν θέλουμε να αντιστρέψουμε κάποια bits σε μια λέξη.

Για κάθε μία από τις προαναφερθείσες πράξεις, προϋπόθεση βέβαια είναι η επιλογή του κατάλληλου «φίλτρου», δηλ. της κατάλληλης τιμής του $2^{ου}$ τελεστέου.

Παραδείγματα

Έστω: $t2 = $0000\ 0000\ 0000\ 0000\ 0000\ 1101\ 0000\ 0000_2$

$t1 = $0000\ 0000\ 0000\ 0000\ 0011\ 1100\ 0000\ 0000_2$ (φίλτρο)

τότε μετά την εκτέλεση της εντολής:

- and $t0,$t1,$t2 # $t0 = $t1 & $t2
  ο $t0 θα περιέχει: $0000\ 0000\ 0000\ 0000\ 0000\ 1100\ 0000\ 0000_2$
- or $t0,$t1,$t2 # $t0 = $t1 | $t2
  ο $t0 θα περιέχει: $0000\ 0000\ 0000\ 0000\ 0011\ 1101\ 0000\ 0000_2$
- nor $t0,$t1,$t2 # $t0 = ~ ($t1 | $t2)

Αν ο ένας καταχωρητής είναι ίσος με 0 τότε η εντολή είναι ισοδύναμη με NOT. Γι' αυτό, στις εντολές του MIPS δεν υπάρχει ξεχωριστή εντολή NOT. Π.χ.

Αν $t1=0 και $t2 όπως πριν,

ο $t0 θα περιέχει: $1111\ 1111\ 1111\ 1111\ 1111\ 0010\ 1111\ 1111_2$

- andi $t0,$t1,100 # $t0=$t1&100
- ori $t0,$t1,7 # $t0=$t1|7

ΜΕΡΟΣ Β

Ασκήσεις

Γράψτε πρόγραμμα σε συμβολική γλώσσα MIPS το οποίο θα μπορεί:

1. Να διαβάζει ένα ακέραιο και να εμφανίζει αν είναι άρτιος ή περιττός (υπόδειξη: ελέγξτε το χαμηλής τάξης ψηφίο).

2. Να διαβάζει ένα ακέραιο και να εμφανίζει αν είναι πολλαπλάσιο του 4 (υπόδειξη: τα δύο χαμηλής τάξης ψηφία να είναι 00).

3. Να διαβάζει ένα ακέραιο και να εμφανίζει αν είναι θετικός ή αρνητικός ελέγχοντας το υψηλής τάξης ψηφίο (υπόδειξη: μπορείτε να το ολισθήσετε προς τα δεξιά).

4. Να διαβάζει ένα ακέραιο μεταξύ 0 και 15 και να εμφανίζει τη δυαδική του τιμή (έλεγχος στα 4 χαμηλής τάξης ψηφία. Χρησιμοποιήστε επανάληψη).

5. Γράψτε ένα πρόγραμμα σε συμβολική γλώσσα MIPS το οποίο θα διαβάζει έναν ακέραιο αριθμό από 1 έως 255 μετά την εμφάνιση σχετικού μηνύματος. Με βάση αυτόν τον αριθμό να αναζητάει στο table και να εμφανίζει τον αντίστοιχο διψήφιο δεκαεξαδικό του.
Υπόδειξη: Να ορίσετε αρχικά στο πεδίο των δεδομένων μία σειρά χαρακτήρων η οποία θα περιλαμβάνει όλα τα 16δικά ψηφία, π.χ. table: . ascii "0123456789abcdef ".
Στη συνέχεια να διαμορφώσετε το πρόγραμμά σας έτσι ώστε:

- να απομονώνει τα 4 περισσότερο σημαντικά ψηφία του αριθμού που εισάγεται. Τα ψηφία αυτά θα αποτελούν έναν δείκτη στο table ώστε να εντοπιστεί το πρώτο από τα δύο δεκαεξαδικά ψηφία. Το ψηφίο αυτό θα διαβάζεται από τη μνήμη και θα εμφανίζεται στο παράθυρο Run I/O.
- στη συνέχεια να απομονώνει τα 4 λιγότερο σημαντικά ψηφία του αριθμού που εισάγεται. Τα ψηφία αυτά θα αποτελούν ένα διαφορετικό δείκτη στο table ώστε να εντοπιστεί το δεύτερο από τα δύο δεκαεξαδικά ψηφία. Το ψηφίο αυτό θα διαβάζεται από τη μνήμη και θα εμφανίζεται στο παράθυρο Run I/O, δίπλα από το πρώτο.

Παράδειγμα 1
Αν δοθεί ο αριθμός 3 (00000011_2), η απομόνωση των 4 περισσότερο σημαντικών ψηφίων του, δίνει ως δείκτη το 0. Ο δείκτης αυτός δείχνει το στοιχείο 0 του table, το οποίο θα προσπελάσει στη μνήμη και θα εμφανίσει. Στη συνέχεια η απομόνωση των 4 λιγότερο σημαντικών ψηφίων του 3 δίνει ως δείκτη το 3. Ο δείκτης αυτός δείχνει το στοιχείο 3 του table, το οποίο θα προσπελάσει στη μνήμη και θα εμφανίσει. Έτσι δίνοντας τον ακέραιο αριθμό 3 θα μας εμφανίσει το 03.

Παράδειγμα 2
Αν δοθεί ο αριθμός 254 (11111110_2), η απομόνωση των 4 περισσότερο σημαντικών ψηφίων του, δίνει ως δείκτη το 15. Ο δείκτης αυτός δείχνει το στοιχείο f του table, το οποίο θα προσπελάσει στη μνήμη και θα εμφανίσει. Στη συνέχεια η απομόνωση των 4 λιγότερο σημαντικών ψηφίων του 254 δίνει ως δείκτη το 14. Ο δείκτης αυτός δείχνει το στοιχείο e του table, το οποίο θα προσπελάσει στη μνήμη και θα εμφανίσει. Έτσι δίνοντας τον ακέραιο αριθμό 254 θα μας εμφανίσει το fe.