Φύλλο οδηγιών MIPS

Αριθμητικές και Bitwise Εντολές

Όλες οι αριθμητικές και bitwise εντολές μπορούν να γραφτούν με δύο τρόπους:

1. `add t0, t1, t2`
    - προσθέτει δύο καταχωρητές και τοποθετεί το αποτέλεσμα σε έναν τρίτο καταχωρητή.
    - αυτό κάνει `t0 = t1 + t2`
2. `add t0, t1, 4`
    - Προσθέτει έναν καταχωρητή και μια σταθερά και τοποθετεί το αποτέλεσμα σε έναν δεύτερο καταχωρητή.
    - αυτό κάνει `t0 = t1 + 4`

Το `i` σημαίνει «άμεσος», αφού οι αριθμοί μέσα σε εντολές ονομάζονται άμεσοι.

Μερικές φορές για αυτήν τη δεύτερη μορφή, θα τη δείτε γραμμένη ως **`addi`** ή **`subi`**. Αυτά είναι εντελώς ισοδύναμα, απλώς ένα διαφορετικό όνομα για την ίδια εντολή.

Εντολή	**Λειτουργία**	**Περιγραφή**
`neg a, b`	`a = -b`	δίνει το αρνητικό του b.
`add a, b, c`	`a = b + c`	προσθέτει προσημασμένους αριθμούς.
`sub a, b, c`	`a = b - c`	αφαιρεί τους προσημασμένους αριθμούς.
`mul a, b, c`	`a = b * c`	δίνει μόλις 32 bit προσημένου πολλαπλασιασμού.
`div a, b, c`	`a = b / c`	δίνει το πηλίκο της προσημασμένης διαίρεσης.
`rem a, b, c`	`a = b % c`	δίνει το υπόλοιπο της προσημασμένης διαίρεσης.
`addu a, b, c`	`a = b + c`	προσθέτει αριθμούς χωρίς πρόσημο.
`subu a, b, c`	`a = b - c`	αφαιρεί αριθμούς χωρίς πρόσημο.
`mulu a, b, c`	`a = b * c`	δίνει μόλις 32 bit πολλαπλασιασμού χωρίς πρόσημο.
`divu a, b, c`	`a = b / c`	δίνει το πηλίκο της μη πρόσημης διαίρεσης.
`remu a, b, c`	`a = b % c`	δίνει το υπόλοιπο της μη πρόσημης διαίρεσης.

Εντολή	Λειτουργία	Περιγραφή
`mfhi a`	`a = HI`	μετά `mul`, δίνει υψηλά 32 bit. μετά `div`, δίνει υπόλοιπο.
`mflo a`	`a = LO`	μετά `mul`, δίνει χαμηλά 32 bit. μετά `div`, δίνει πηλίκο.
`not a, b`	`a = ~b`	δίνει το bit-συμπλήρωμα του b (όλα τα bit έχουν αντιστραφεί).
`and a, b, c`	`a = b & c`	αριθμοί AND ανά bit.
`or a, b, c`	`a = b | c`	αριθμοί OR ανά bit.
`xor a, b, c`	`a = b ^ c`	αριθμοί XOR ανά bit.

Εντολές κύλησης

Το MIPS αποφάσισε να υλοποιήσει τις μετατοπίσεις λίγο διαφορετικά από τις υπόλοιπες αριθμητικές και bitwise εντολές.

Εντολή	Λειτουργία	Περιγραφή
`sll a, b, imm`	`a = b << imm`	μετατόπιση προς τα αριστερά κατά ένα σταθερό ποσό.
`srl a, b, imm`	`a = b >>> imm`	μετατόπιση προς τα δεξιά χωρίς πρόσημο (λογική) κατά μια σταθερή ποσότητα.
`sra a, b, imm`	`a = b >> imm`	μετατόπιση της αριθμητικής προς τα δεξιά κατά μια σταθερή ποσότητα.
`sllv a, b, reg`	`a = b << reg`	μετατόπιση προς τα αριστερά κατά το ποσό σε ένα μητρώο.
`srlv a, b, reg`	`a = b >>> reg`	μετατόπιση προς τα δεξιά χωρίς πρόσημο (λογική) κατά το ποσό σε ένα καταχωρητή.
`srav a, b, reg`	`a = b >> reg`	Μετατόπιση προς τα δεξιά στην αριθμητική κατά το ποσό σε έναν καταχωρητή.

Εντολές Μεταφοράς Δεδομένων

Υπάρχουν δύο εντολές «φόρτωσης» **που δεν έχουν πρόσβαση στη μνήμη.** Επίσης η **`move`** δεν μετακινεί, αντιγράφει

Εντολή	Λειτουργία	Περιγραφή
`li a, imm`	`a = imm`	τοποθετήστε μια σταθερή τιμή σε ένα μητρώο.
`la a, label`	`a = &label`	εισάγετε τη διεύθυνση στην οποία δείχνει μια ετικέτα σε ένα μητρώο.
`move a, b`	`a = b`	αντιγραφή τιμής από ένα μητρώο σε ένα άλλο.

Οι υπόλοιπες εντολές φόρτωσης/αποθήκευσης **έχουν πάντα πρόσβαση στη μνήμη.** Όλες αυτές οι εντολές μπορούν να γραφτούν με τέσσερις διαφορετικούς τρόπους:

1. `lw t0, var`
    - αντιγράφει μια λέξη (τιμή 32-bit) από τη μεταβλητή μνήμης **`var`** στον καταχωρητή **`t0`**
    - **`var`** πρέπει να έχει δηλωθεί ως κάτι σαν:

```
.data
var: .word 0
```

2. `lw t0, (t1)`
    - αντιγράφει μια λέξη από τη διεύθυνση μνήμης που δίνεται από **`t1`** στον καταχωρητή **`t0`**
3. `lw t0, 4(t1)`
    - αντιγράφει μια λέξη από τη διεύθυνση μνήμης που δίνεται από **`t1`** + 4στον καταχωρητή **`t0`**
4. `lw t0, arrayname(t1)`
    - αντιγράφει μια λέξη από τη διεύθυνση μνήμης που δίνεται από **`arrayname`** + **`t1`** στον καταχωρητή **`t0`**

ΥΠΕΝΘΥΜΙΣΗ: αποθηκεύει τιμές αντιγραφής ΑΠΟ τους καταχωρητές ΣΤΗ μνήμη. Άρα ΑΠΟ την αριστερή πλευρά ΣΤΗ διεύθυνση στη δεξιά πλευρά.

Εντολή	**Λειτουργία**	**Περιγραφή**
`lw reg, addr`	`reg = MEM[addr]`	φορτώνει τα 4 byte ως `addr`τιμή 32-bit σε `reg`.
`lh reg, addr`	`reg = sxt(MEM[addr])`	φορτώνει τα 2 byte ως `addr`υπογεγραμμένη τιμή 16-bit στο `reg`.
`lb reg, addr`	`reg = sxt(MEM[addr])`	φορτώνει το 1 byte στο `addr`ως υπογεγραμμένη τιμή 8-bit στο `reg`.
`lhu reg, addr`	`reg = zxt(MEM[addr])`	φορτώνει τα 2 byte ως `addr`μη υπογεγραμμένη τιμή 16-bit στο `reg`.
`lbu reg, addr`	`reg = zxt(MEM[addr])`	φορτώνει το 1 byte at `addr`ως μη υπογεγραμμένη τιμή 8-bit στο `reg`.
`sw reg, addr`	`MEM[addr] = reg`	αποθηκεύει την τιμή του `reg`στη μνήμη ως 4 byte ξεκινώντας από `addr`.
`sh reg, addr`	`MEM[addr] = lo16(reg)`	αποθηκεύει τα χαμηλά 16 bit του `reg`στη μνήμη ως 2 byte ξεκινώντας από `addr`.
`sb reg, addr`	`MEM[addr] = lo8(reg)`	Αποθηκεύει τα 8 χαμηλότερα bit `reg`στη μνήμη ως 1 byte στο `addr`.

Τέλος, υπάρχουν δύο ψευδο-εντολές στοίβας που χρησιμοποιούνται για την αποθήκευση και την επαναφορά τιμών σε συναρτήσεις:

Εντολή	**Λειτουργία**	**Περιγραφή**
`push reg`	`sp -= 4; MEM[sp] = reg`	ωθεί την τιμή του `reg`στη στοίβα κλήσεων

Εντολή	Λειτουργία	Περιγραφή
`pop reg`	`reg = MEM[sp]; sp += 4`	εμφανίζει την κορυφαία τιμή στοίβας κλήσεων και την τοποθετεί σε`reg`

Εντολές Ροής Ελέγχου Χωρίς Όρους

Αυτά αλλάζουν πάντα τον υπολογιστή σε μια νέα τοποθεσία.

Μνημονικός	Λειτουργία	Περιγραφή
`j label`	`PC = label`	πηγαίνει στην οδηγία στο `label`.
`jal label`	`ra = PC + 4; PC = label`	Η κλήση της συνάρτησης στο `label`. αποθηκεύει τη διεύθυνση επιστροφής στο `ra`.
`jr reg`	`PC = reg`	πηγαίνει στην εντολή της οποίας η διεύθυνση είναι στο `reg`, συχνά `ra`.
`syscall`	(δείτε την περιγραφή)	εκτελεί τη συνάρτηση κλήσης συστήματος της οποίας ο αριθμός είναι σε `v0`.

Εντολές Ροής Ελέγχου Υπό Όρους

Όλες αυτές οι οδηγίες ελέγχουν τη δεδομένη συνθήκη και, αν είναι:

- **true,** πηγαίνει στην δεδομένη ετικέτα
- **false,** πηγαίνει στην **επόμενη εντολή** (δηλαδή δεν κάνει τίποτα)

Επίσης, όλες αυτές οι οδηγίες μπορούν να γραφτούν με δύο τρόπους:

1. `blt t0, t1, label`
    - συγκρίνει δύο καταχωρητές (βλέπει αν `t0 < t1`)
2. `blt t0, 10, label`
    - συγκρίνει έναν καταχωρητή με μια σταθερά (ελέγχει αν `t0 < 10`)

Εντολή	Λειτουργία	Περιγραφή
`beq a, b, label`	`if(a == b) { PC = label }`	αν `a`είναι ίσο με `b`, πηγαίνει στο `label`.
`bne a, b, label`	`if(a != b) { PC = label }`	αν `a`ΔΕΝ είναι ίσο με `b`, πηγαίνει στο `label`.
`blt a, b, label`	`if(a < b) { PC = label }`	αν `a`είναι μικρότερο από `b`, πηγαίνει στο `label`.
`ble a, b, label`	`if(a <= b) { PC = label }`	αν `a`είναι μικρότερο ή ίσο με `b`, πηγαίνει στο `label`.

Εντολή	Λειτουργία	Περιγραφή
`bgt a, b, label`	`if(a > b) { PC = label }`	αν `a`είναι μεγαλύτερο από `b`, πηγαίνει στο `label`.
`bge a, b, label`	`if(a >= b) { PC = label }`	αν `a`είναι μεγαλύτερο ή ίσο με `b`, πηγαίνει στο `label`.
`bltu a, b, label`	`if(a < b) { PC = label }`	το ίδιο με `blt`αλλά κάνει μια ανυπόγραφη σύγκριση.
`bleu a, b, label`	`if(a <= b) { PC = label }`	το ίδιο με `ble`αλλά κάνει μια ανυπόγραφη σύγκριση.
`bgtu a, b, label`	`if(a > b) { PC = label }`	το ίδιο με `bgt`αλλά κάνει μια ανυπόγραφη σύγκριση.
`bgeu a, b, label`	`if(a >= b) { PC = label }`	το ίδιο με `bge`αλλά κάνει μια ανυπόγραφη σύγκριση

[Πηγή]