Πανεπιστήμιο Θεσσαλίας
Τμήμα Πληροφορικής

Εισαγωγή στους Η/Υ

Πρώτη Σειρά Ασκήσεων

10 Απριλίου 2019
παράδοση: *6μμ 8 Μαΐου 2019*

Για τις παρακάτω ασκήσεις θεωρήστε καταχωρητές και λειτουργίες ακεραίων εύρους 32 δυαδικών ψηφίων, εκτός εάν αναφέρεται διαφορετικά.

Άσκηση 1:

Θεωρήστε το παρακάτω πρόγραμμα σε συμβολική γλώσσα MIPS:

```
        addi    $t0, $zero, 0
        addi    $v0, $zero, 0
for1:   slt     $t1, $t0, $a0
        beq     $t1, $zero, finish
        lw      $v1, 0($a1)
        bne     $v1, $zero, cont
        addi    $v0, $v0, 1
cont:   addi    $t1, $t0, 1
        or      $t2, $a1, $zero
for2:   slt     $t3, $t1, $a0
        beq     $t3, $zero, next
        lw      $t3, 4($t2)
        add     $v1, $v1, $t3
        addi    $t1, $t1, 1
        addi    $t2, $t2, 4
        j       for2
next:   sw      $v1, 0($a1)
        addi    $t0, $t0, 1
        addi    $a1, $a1, 4
        j       for1
finish: jr      $ra
```

Α. Εξηγήστε τι κάνει το πρόγραμμα αυτό, (i) σε επίπεδο εντολής – ξεχωριστά για κάθε εντολή, και (ii) συνολικά. Υποθέστε ότι ο καταχωρητής $a0 περιέχει ένα μη προσημασμένο ακέραιο n και ο καταχωρητής $a1 περιέχει την αρχική διεύθυνση ενός διανύσματος ακεραίων V που περιέχει n στοιχεία.

Β. Δώστε έναν όσο το δυνατό πιο σύντομο κώδικα γλώσσας C που να αντιστοιχεί στον πιο πάνω κώδικα MIPS.

Άσκηση 2:

Χρησιμοποιώντας *μόνο το σύνολο εντολών MIPS του Πίνακα 1*, γράψτε *το συντομότερο* κώδικα συμβολικής γλώσσας MIPS ο οποίος να υλοποιεί τις ακόλουθες λειτουργίες:

Α. `orn $z, $x, $y` ; $z := $x OR NOT $y

Β. `rpt $x, loop` ; IF $x > 0 THEN $x := $x – 1, GOTO loop ENDIF

Γ. `trpl $z, $x, $y` ; IF (($z := MEM[$x]) == 0) THEN MEM[$x] := $y ENDIF

όπου $x, $y, $z οποιοιδήποτε καταχωρητές γενικού σκοπού. Να εξηγήσετε επαρκώς τους κώδικές σας.

Άσκηση 3:

Χρησιμοποιώντας *μόνο το σύνολο εντολών MIPS του Πίνακα 1*, γράψτε *το συντομότερο* κώδικα συμβολικής γλώσσας MIPS ο οποίος να απομονώνει ένα τμήμα λέξης 32-bit μεταφέροντάς το στα λιγότερο σημαντικά ψηφία ενός καταχωρητή γενικού σκοπού, συμπληρώνοντας με 0 τα υπόλοιπα ψηφία, στις ακόλουθες δύο περιπτώσεις:

Α. Το τμήμα είναι γνωστό και καλύπτει τα ψηφία σημαντικότητας από 12 έως και 21.

Β. Το τμήμα δεν είναι γνωστό, ξεκινάει από το ψηφίο σημαντικότητας που δίνει ο καταχωρητής $a0, και τελειώνει στο ψηφίο σημαντικότητας που δίνει ο καταχωρητής $a1.

Και στις δύο περιπτώσεις θέλουμε το αποτέλεσμα να τοποθετείται στον καταχωρητή $v0. Σε κάθε λέξη μετράμε τα ψηφία από τα δεξιά προς τα αριστερά, με αντίστοιχη σημαντικότητα από 0 έως 31. Να εξηγήσετε επαρκώς τους κώδικές σας.

Άσκηση 4:

Θεωρήστε ένα μονοδιάστατο πίνακα 128 bytes, του οποίου τα διαδοχικά αποθηκευμένα 1024 bits απεικονίζουν τα διαδοχικά 1024 pixels μιας γραμμής μιας παλιάς μονόχρωμης οθόνης. Χρησιμοποιώντας *μόνο το σύνολο εντολών MIPS του Πίνακα 1*, γράψτε έναν κώδικα MIPS για ένα παιχνίδι που γράφεται για τέτοια οθόνη, ο οποίος να εξομοιώνει την κίνηση δεξιά-αριστερά ως προς ένα σταθερό σημείο με αντίστροφη μετατόπιση των αποθηκευμένων bit, τόσο προς τα δεξιά όσο και προς τα αριστερά, (α) κατά 8 και (β) κατά 3 bits.

Ο ζητούμενος κώδικας για κάθε μία από τις δύο περιπτώσεις θα πρέπει να έχει τη μορφή συνάρτησης, στην οποία η πρώτη παράμετρος θα περιέχει τη διεύθυνση μνήμης όπου βρίσκεται ο πίνακας και η δεύτερη παράμετρος θα περιέχει τη φορά της κίνησης, έστω 0 για κίνηση προς τα αριστερά και 1 για κίνηση προς τα δεξιά. Κάθε συνάρτηση θα πρέπει να τηρεί τις συμβάσεις κλήσης συναρτήσεων MIPS.

Προσπαθήστε να γράψετε γρήγορο κώδικα. Στο τέλος υπολογίστε το πλήθος των εντολών που εκτελεί ο κώδικάς σας σε κάθε περίπτωση.

Άσκηση 5:

Έστω η παρακάτω εντολή μιας ψευδογλώσσας υψηλού επιπέδου:

```
for i from 1 to 999 do {
   if a[i] < 0 then a[i] := -a[i]+c*a[i-1]
   if a[i] = 0 then exit loop
   a[i] := (b[i]-c*a[i-1])/a[i]
   c := c+i
}
```

όπου a και b είναι διανύσματα ακεραίων, τοποθετημένα στη μνήμη.

Α. Γράψτε τον αντίστοιχο κώδικα σε συμβολική γλώσσα MIPS, χρησιμοποιώντας *μόνο εντολές από τον Πίνακα 1*. Υποθέστε ότι η αρχική διεύθυνση του a βρίσκεται στον καταχωρητή $a0, και του b στον καταχωρητή $a1. Ακόμα υποθέστε ότι οι μεταβλητές i και c δεν τοποθετούνται στη μνήμη, αλλά αποθηκεύονται στους καταχωρητές $t0 kai $s0 αντίστοιχα.

Β. Πόσες εντολές MIPS εκτελούνται από αυτόν τον κώδικα κατ' ελάχιστο και κατά μέγιστο; Πόσες προσπελάσεις στη μνήμη γίνονται από αυτόν τον κώδικα κατ' ελάχιστο και κατά μέγιστο; Αν ο πρώτος έλεγχος δίνει αληθές αποτέλεσμα σε ποσοστό x% και ο δεύτερος έλεγχος δίνει αληθές αποτέλεσμα σε ποσοστό y%, πόσες εντολές εκτελούνται και πόσες προσπελάσεις μνήμης γίνονται από τον κώδικα;

Άσκηση 6:

Έστω ότι σας δίνεται μια συνάρτηση, δηλωμένη σε C ως πρωτότυπο:

```
int func(int, int);
```

και μεταφρασμένη σε MIPS τηρώντας τις γνωστές συμβάσεις κλήσης συναρτήσεων. Γράψτε τον κώδικα MIPS που μεταφράζει τις ακόλουθες συναρτήσεις γλώσσας C, ο οποίος θα πρέπει επίσης να τηρεί τις ίδιες συμβάσεις κλήσης συναρτήσεων, ώστε να υπάρχει συμβατότητα στη χρήση των συναρτήσεων:

Α.
```
int f(int a, int b, int c, int d) {
   return func(func(a,b),c+d);
}
```

Β.
```
int f(int a, int b, int c, int d) {
   if (a+b>c+d) return func(1+func(a+b,c+d),a+b-c-d);
   return func(a+c,b+d);
}
```

Γ.
```
int f(int a, int b, int c, int d) {
   if (a>b) return f(func(a+b,a-b),0,func(c,d),0);
   if (c>d) return f(func(a,b),func(c,d),c+d,c-d);
   return f(func(func(a,b),a+b),c,func(c,d),d);
}
```

Για κάθε περίπτωση να αναφέρετε επακριβώς τις συμβάσεις κλήσης συναρτήσεων που τηρείτε.

Άσκηση 7:

Θέλουμε να γράψουμε έναν κώδικα σε συμβολική γλώσσα MIPS, ο οποίος να υπολογίζει την παράσταση:

$$\sum_{i=0}^{n-1} a_i$$

για κάποια ακολουθία ακεραίων a_i. Εξηγήστε εάν και πώς ο παρακάτω κώδικας υλοποιεί το ζητούμενο υπολογισμό:

```
      ori   $t0, $zero, 0
      ori   $v0, $zero, 0
      la    $t2, X
      lw    $t3, 0($t2)
X:    lw    $t1, 0($a1)
      add   $v0, $v0, $t1
      addi  $t3, $t3, 4
      sw    $t3, 0($t2)
      addi  $t0, $t0, 1
      bne   $t0, $a0, X
```

όπου οι καταχωρητές $a0 και $a1 περιέχουν αρχικά την τιμή του n και τη διεύθυνση στη μνήμη όπου ξεκινάει η ακολουθία a_i, αντίστοιχα. Με την ολοκλήρωση του κώδικα, το αποτέλεσμα πρέπει να βρίσκεται στον καταχωρητή $v0.
Για να απαντήσετε, γράψτε πρώτα το συμβολικό κώδικα MIPS σε *γλώσσα μηχανής δυαδικού συστήματος* και περιγράψτε πώς εκτελείται κάθε εντολή στον επεξεργαστή. Βρείτε έτσι τι περιέχει ο καταχωρητής $t2, τι φορτώνει στον καταχωρητή $t3 η πρώτη από τις δύο εντολές φόρτωσης από τη μνήμη, και τι αποθηκεύει η εντολή αποθήκευσης στη μνήμη. Στη συνέχεια εξηγήστε εάν ο κώδικας όντως υπολογίζει την παραπάνω έκφραση.
Ποια είναι η μέγιστη τιμή του n για την οποία ο κώδικας MIPS μπορεί να λειτουργήσει σωστά για πρώτη φορά;
Μπορεί ο κώδικας να ξαναεκτελεστεί μέσα στα πλαίσια ενός εξωτερικού βρόχου; Τι πρόβλημα υπάρχει, και πώς θα το αντιμετωπίσετε;

Άσκηση 8:

Γράψτε ένα υποπρόγραμμα σε συμβολική γλώσσα MIPS το οποίο να υπολογίζει το νιοστό όρο της παρακάτω ακολουθίας ακεραίων:

$$F_\nu = \begin{cases} 0, & \text{εάν } \nu = 0, \\ 1, & \text{εάν } \nu = 1, \\ \mathrm{F}_{\nu-1} - \mathrm{F}_{\nu-2}, & \text{εάν } \nu \text{ άρτιος και } \nu > 0, \\ \mathrm{F}_{\nu-1} + \mathrm{F}_{\nu-2} + \mathrm{F}_{\nu-3}, & \text{διαφορετικά.} \end{cases}$$

χρησιμοποιώντας μόνο εντολές από τον Πίνακα 1, με τρεις διαφορετικούς τρόπους:

Α. Το υποπρόγραμμα να υλοποιεί την παρακάτω αναδρομική συνάρτηση της γλώσσας C:

```
int F(int n) {
      if (n == 0) return 0;
      else if (n == 1) return 1;
      else if (n&1 == 0) return F(n-1) - F(n-2);
      return F(n-1) + F(n-2) + F(n-3);
}
```

η οποία εκφράζει απ' ευθείας τον πιο πάνω ορισμό της ακολουθίας F_ν.

Β. Το ζητούμενο υποπρόγραμμα να υλοποιεί τώρα την πιο κάτω συνάρτηση της γλώσσας C:

```
int F(int a, int b, int c, int n, int i) {
      if (i == n) return c;
      return F(i&1?a-b:a+b+c, a, b, n, i+1);
}
```

η οποία μετατρέπει την πολλαπλή αναδρομική κλήση σε απλή, με αρχική κλήση την:

```
F(1,1,0,n,0);
```

Γ. Επειδή στον κώδικα C του ερωτήματος Β η αναδρομική κλήση είναι η τελευταία εντολή της συνάρτησης, η μετάφραση μπορεί να γίνει χωρίς αναδρομική κλήση, με βρόχο αντί αναδρομής, ως εξής:

```
int F(int n) {
      int a=1, b=1, c=0, i=0;
      while (i < n) {
            int tmp = i&1 ?  a-b : a+b+c;
            c = b;
            b = a;
            a = tmp;
            i++;
      }
      return c;
```

```
}
```

Δ. Ο παραπάνω τρόπος μετάφρασης αναδρομικών συναρτήσεων χωρίς αναδρομική κλήση – όταν βέβαια αυτό είναι εφικτό – είναι πολύ προτιμότερος από τους προηγούμενους, και εφαρμόζεται σε όλους τους καλούς μεταγλωττιστές. Υπολογίστε λοιπόν τη διαφορά επίδοσης των τριών παραπάνω λύσεων με βάση τον αριθμό των εντολών MIPS που εκτελούνται για το νιοστό όρο της ακολουθίας F_ν σε κάθε περίπτωση.

Προσοχή: Όλες οι ασκήσεις θα πρέπει να παραδοθούν χειρόγραφα για να γίνουν δεκτές.

Πίνακας 1: Συμβολικές Εντολές Αρχιτεκτονικής MIPS R2000

Εντολή	Αντίστοιχος κώδικας μηχανής						Περιγραφή
Α. Αριθμητικές και λογικές εντολές:							
add rd,rs,rt	0x0	rs	rt	rd	0	0x20	rd = rs+rt, με υπερχείλιση
addu rd,rs,rt	0x0	rs	rt	rd	0	0x21	rd = rs+rt, χωρίς υπερχείλιση
addi rt,rs,imm	0x8	rs	rt	imm			rt = rs+imm, με υπερχείλιση
addiu rt,rs,imm	0x9	rs	rt	imm			rt = rs+imm, χωρίς υπερχείλιση
and rd,rs,rt	0x0	rs	rt	rd	0	0x24	rd = rs AND rt
andi rt,rs,imm	0xc	rs	rt	imm			rt = rs AND imm
div rs,rt	0x0	rs	rt	0	0	0x1a	rs = lo*rt+hi, με υπερχείλιση
divu rs,rt	0x0	rs	rt	0	0	0x1b	rs = lo*rt+hi, χωρίς υπερχείλιση
lui rt,imm	0xf	0	rt	imm			rt = imm << 16
mult rs,rt	0x0	rs	rt	0	0	0x18	hi,lo = rs*rt, με υπερχείλιση
multu rs,rt	0x0	rs	rt	0	0	0x19	hi,lo = rs*rt, χωρίς υπερχείλιση
nor rd,rs,rt	0x0	rs	rt	rd	0	0x27	rd = rs NOR rt
or rd,rs,rt	0x0	rs	rt	rd	0	0x25	rd = rs OR rt
ori rt,rs,imm	0xd	rs	rt	imm			rt = rs OR imm
sll rd,rt,sh	0x0	0	rt	rd	sh	0x0	rd = rt << sh
sllv rd,rt,rs	0x0	rs	rt	rd	0	0x4	rd = rt << rs
slt rd,rs,rt	0x0	rs	rt	rd	0	0x2a	rd = rs<rt, με πρόσημο
sltu rd,rs,rt	0x0	rs	rt	rd	0	0x2b	rd = rs<rt, χωρίς πρόσημο
slti rt,rs,imm	0xa	rs	rt	imm			rt = rs<imm, με πρόσημο
sltiu rt,rs,imm	0xb	rs	rt	imm			rt = rs<imm, χωρίς πρόσημο
sra rd,rt,sh	0x0	0	rt	rd	sh	0x3	rd = rt >> sh, με πρόσημο
srav rd,rt,rs	0x0	rs	rt	rd	0	0x7	rd = rt >> rs, με πρόσημο
srl rd,rt,sh	0x0	0	rt	rd	sh	0x2	rd = rt >> sh, χωρίς πρόσημο
srlv rd,rt,rs	0x0	rs	rt	rd	0	0x6	rd = rt >> rs, χωρίς πρόσημο
sub rd,rs,rt	0x0	rs	rt	rd	0	0x22	rd = rs-rt, με υπερχείλιση
subu rd,rs,rt	0x0	rs	rt	rd	0	0x23	rd = rs-rt, χωρίς υπερχείλιση
xor rd,rs,rt	0x0	rs	rt	rd	0	0x26	rd = rs XOR rt
xori rt,rs,imm	0xe	rs	rt	imm			rt = rs XOR imm
Β. Εντολές άλματος:							
bczf label	0x10|z	8	0	off			if (!cz) goto label
bczt label	0x10|z	8	1	off			if (cz) goto label
beq rs,rt,label	0x4	rs	rt	off			if (rs==rt) goto label
bgez rs,label	0x1	rs	1	off			if (rs>=0) goto label
bgtz rs,label	0x7	rs	0	off			if (rs>0) goto label
blez rs,label	0x6	rs	0	off			if (rs<=0) goto label
bltz rs,label	0x1	rs	0	off			if (rs<0) goto label
bne rs,rt,label	0x5	rs	rt	off			if (rs!=rt) goto label
j target	0x2	target					goto target
jal target	0x3	target					goto target, $ra = PC+1
jalr rs,rd	0x0	rs	0	rd	0	0x9	goto (rs), rd = PC+1
jr rs	0x0	rs	0	0	0	0x8	goto (rs)
Γ. Εντολές μεταφοράς δεδομένων:							
lb rt,off(rs)	0x20	rs	rt	off			rt = M[rs+off]^{8},με προέκταση
lbu rt,off(rs)	0x24	rs	rt	off			rt = M[rs+off]^{8},χωρίς προέκταση
lh rt,off(rs)	0x21	rs	rt	off			rt = M[rs+off]^{16},με προέκταση
lhu rt,off(rs)	0x25	rs	rt	off			rt = M[rs+off]^{16},χωρίς προέκταση
lw rt,off(rs)	0x23	rs	rt	off			rt = M[rs+off]^{32}
lwcz rt,off(rs)	0x30|z	rs	rt	off			rtz = M[rs+off]^{32}
mfhi rd	0x0	0	0	rd	0	0x10	rd = hi

Πίνακας 1: Συμβολικές Εντολές Αρχιτεκτονικής MIPS R2000 (συνέχεια)

mflo rd	0x0	0	0	rd	0	0x12	rd = lo	
mthi rs	0x0	rs	0	0	0	0x11	hi = rs	
mtlo rs	0x0	rs	0	0	0	0x13	lo = rs	
mfcz rt,rd	0x10|z	0	rt	rd	0	0x0	rt = rdz	
mtcz rd,rt	0x10|z	4	rt	rd	0	0x0	rdz = rt	
sb rt,off(rs)	0x28	rs	rt	off			$M[rs+off]^8$ = rt	
sh rt,off(rs)	0x29	rs	rt	off			$M[rs+off]^{16}$ = rt	
sw rt,off(rs)	0x2b	rs	rt	off			$M[rs+off]^{32}$ = rt	
swcz rt,off(rs)	0x38|z	rs	rt	off			$M[rs+off]^{32}$ = rtz	
Δ. Εντολές μονάδας κινητής υποδιαστολής:								
abs.d fd,fs	0x11	1	0	fs	fd	0x5	fd = |fs|	64 bits
abs.s fd,fs	0x11	0	0	fs	fd	0x5	fd = |fs|	32 bits
add.d fd,fs,ft	0x11	1	ft	fs	fd	0x0	fd = fs+ft	64 bits
add.s fd,fs,ft	0x11	0	ft	fs	fd	0x0	fd = fs+ft	32 bits
c.eq.d fs,ft	0x11	1	ft	fs	0	0x12	c1 = fs==ft	64 bits
c.eq.s fs,ft	0x11	0	ft	fs	0	0x12	c1 = fs==ft	32 bits
c.le.d fs,ft	0x11	1	ft	fs	0	0x22	c1 = fs<=ft	64 bits
c.le.s fs,ft	0x11	0	ft	fs	0	0x22	c1 = fs<=ft	32 bits
c.lt.d fs,ft	0x11	1	ft	fs	0	0x32	c1 = fs<ft	64 bits
c.lt.s fs,ft	0x11	0	ft	fs	0	0x32	c1 = fs<ft	32 bits
cvt.d.s fd,fs	0x11	1	0	fs	fd	0x21	fd = fp64($fs^{(fp32)}$)	μετατροπή
cvt.d.w fd,fs	0x11	0	0	fs	fd	0x21	fd = fp64($fs^{(int)}$)	μετατροπή
cvt.s.d fd,fs	0x11	1	0	fs	fd	0x20	fd = fp32($fs^{(fp64)}$)	μετατροπή
cvt.s.w fd,fs	0x11	0	0	fs	fd	0x20	fd = fp32($fs^{(int)}$)	μετατροπή
cvt.w.d fd,fs	0x11	1	0	fs	fd	0x24	fd = int($fs^{(fp64)}$)	μετατροπή
cvt.w.s fd,fs	0x11	0	0	fs	fd	0x24	fd = int($fs^{(fp32)}$)	μετατροπή
div.d fd,fs,ft	0x11	1	ft	fs	fd	0x3	fd = fs/ft	64 bits
div.s fd,fs,ft	0x11	0	ft	fs	fd	0x3	fd = fs/ft	32 bits
mov.d fd,fs	0x11	1	0	fs	fd	0x6	fd = fs	64 bits
mov.s fd,fs	0x11	0	0	fs	fd	0x6	fd = fs	32 bits
mul.d fd,fs,ft	0x11	1	ft	fs	fd	0x2	fd = fs*ft	64 bits
mul.s fd,fs,ft	0x11	0	ft	fs	fd	0x2	fd = fs*ft	32 bits
neg.d fd,fs	0x11	1	0	fs	fd	0x7	fd = -fs	64 bits
neg.s fd,fs	0x11	0	0	fs	fd	0x7	fd = -fs	32 bits
sub.d fd,fs,ft	0x11	1	ft	fs	fd	0x1	fd = fs-ft	64 bits
sub.s fd,fs,ft	0x11	0	ft	fs	fd	0x1	fd = fs-ft	32 bits
Ε. Άλλες εντολές:								
break n	0x0	n				0xd	ειδική περίπτωση με κώδικα n	
nop	0x0	0				0x0	μηδενική εντολή	
rfe	0x10	1	0			0x20	επιστροφή από ειδική περίπτωση	
syscall	0x0	0				0xc	κλήση συστήματος $v0	

Παρατηρήσεις:

1. Η αρχιτεκτονική MIPS R2000 είναι αρχιτεκτονική 32-bit, κάτι που σημαίνει ότι όλοι οι καταχωρητές της είναι μεγέθους 32 bits.
2. Οι rd, rs και rt είναι καταχωρητές γενικού σκοπού.
3. Ο κώδικας κάθε εντολής σε γλώσσα μηχανής περιλαμβάνει 6 bits που είναι ο κωδικός λειτουργίας, ομάδες των 5 bits που είναι συνήθως διευθύνσεις καταχωρητών όπως οι παραπάνω, και είτε έναν κωδικό τελεστή των 6 bits είτε μια σταθερά των 16 bits. Μερικές εντολές περιέχουν διαφορετικά από τα πιο πάνω πεδία.

4. Οι εντολές αριθμητικών πράξεων στις οποίες υπάρχει η ένδειξη «με υπερχείλιση» οδηγούν σε ειδική περίπτωση (exception) όταν εμφανιστεί υπερχείλιση. Σε όσες υπάρχει η ένδειξη «χωρίς υπερχείλιση», δε γίνεται έλεγχος υπερχείλισης.
5. Οι εντολές αριθμητικών ή λογικών πράξεων στις οποίες υπάρχει η ένδειξη «με πρόσημο» λαμβάνουν τα τελούμενά τους σαν προσημασμένους αριθμούς. Ειδικότερα, ολίσθηση προς τα δεξιά στην περίπτωση αυτή εισάγει από τα αριστερά το πρόσημο του αριθμού. Η ένδειξη «χωρίς πρόσημο» λαμβάνει τα τελούμενα των πράξεων σα μη προσημασμένους αριθμούς.
6. Οι εντολές ολίσθησης με σταθερό αριθμό ψηφίων ολίσθησης sh κωδικοποιούν τον αριθμό αυτό στη λέξη εντολής στην τελευταία ομάδα των 5 bits πριν τον κωδικό τελεστή.
7. Οι εντολές πολλαπλασιασμού και διαίρεσης ακεραίων αποθηκεύουν το αποτέλεσμά τους σε δύο καταχωρητές ειδικού σκοπού, τους lo και hi. Στο μεν πολλαπλασιασμό, ο πρώτος περιέχει τα λιγότερο σημαντικά 32 bits του αποτελέσματος, ενώ ο δεύτερος τα υπόλοιπα. Στη διαίρεση, ο πρώτος περιέχει το πηλίκο και ο δεύτερος το υπόλοιπο. Προσπέλαση στους καταχωρητές αυτούς γίνεται με ειδικές εντολές μεταφοράς δεδομένων.
8. Ορισμένες εντολές αναφέρονται σε συνεπεξεργαστή. Η αρχιτεκτονική MIPS υποστηρίζει μέχρι 4 συνεπεξεργαστές και η προσπέλασή τους γίνεται με ειδικές εντολές που είτε αφορούν λειτουργίες του συνεπεξεργαστή, είτε αφορούν την αλληλεπίδραση με τον κυρίως επεξεργαστή. Οι τελευταίες από αυτές τις εντολές περιέχουν στην αναπαράστασή τους τον δείκτη z που παίρνει τιμές μεταξύ 0 και 3. Η υπομονάδα κινητής υποδιαστολής της αρχιτεκτονικής MIPS θεωρείται ότι είναι ο συνεπεξεργαστής 1.
9. Κάθε συνεπεξεργαστής προσφέρει στον κυρίως επεξεργαστή τη δυνατότητα ελέγχου κάποιας συνθήκης cz για εκτέλεση εντολής άλματος με συνθήκη. Ο έλεγχος γίνεται για τιμή της συνθήκης «αληθής» ή «ψευδής».
10. Στις εντολές άλματος με συνθήκη, η κωδικοποίηση γίνεται έτσι ώστε: off = label – (pc + 1), σε επίπεδο διεύθυνσης λέξης.
11. Στις εντολές άμεσου άλματος η διεύθυνση λέξης του στόχου άλματος (target) κωδικοποιείται στα 26 λιγότερο σημαντικά bits της λέξης εντολής. Τα πλέον σημαντικά bits της διεύθυνσης target θεωρείται ότι είναι ταυτόσημα με τα αντίστοιχα της διεύθυνσης λέξης pc + 1.
12. Οι εντολές ανάγνωσης από τη μνήμη που έχουν ένδειξη «με προέκταση» εφαρμόζουν προέκταση προσήμου στην τιμή που φέρνουν από τη μνήμη.
13. Η αριθμητική ένδειξη στην περιγραφή των εντολών προσπέλασης της μνήμης δηλώνει το μέγεθος του δεδομένου που μεταφέρεται. Για μεταφορά δεδομένου μεγαλύτερου του 1 byte θεωρούμε ότι η τοποθέτησή του στη μνήμη γίνεται σε διαδοχικά bytes με τα πιο σημαντικά τμήματα αυτού να βρίσκονται στις μικρότερες διευθύνσεις.
14. Η προσπέλαση στη μνήμη θεωρείται ότι είναι ευθυγραμμισμένη. Αυτό σημαίνει ότι για προσπέλαση δεδομένου των 16 bits (2 bytes) η τελική διεύθυνση προσπέλασης είναι πολλαπλάσιο του 2, ενώ για προσπέλαση δεδομένου των 32 bits (4 bytes) η τελική διεύθυνση προσπέλασης είναι πολλαπλάσιο του 4.
15. Οι rtz και rdz είναι καταχωρητές γενικού σκοπού του συνεπεξεργαστή z.
16. Οι fd, fs και ft είναι καταχωρητές γενικού σκοπού της υπομονάδας κινητής υποδιαστολής.
17. Οι εντολές κινητής υποδιαστολής που χειρίζονται 64 bits θεωρούν ότι η λέξη των 64 bits είναι τοποθετημένη σε δύο διαδοχικούς καταχωρητές, από τους οποίους ο πρώτος είναι άρτιος και περιέχει το πλέον σημαντικό τμήμα του αριθμού.
18. Οι εντολές σύγκρισης κινητής υποδιαστολής τοποθετούν το αποτέλεσμα στη συνθήκη c1.
19. Οι μετατροπή στην υπομονάδα κινητής υποδιαστολής περιλαμβάνει μετατροπή μεταξύ πραγματικών διαφορετικού μεγέθους και μεταξύ πραγματικών και ακεραίων.