Branch and Jump Instructions part B

Στη MIPS, οι jump instructions (οδηγίες μετάβασης) χρησιμοποιούνται για να αλλάξουν τη ροή της εκτέλεσης του προγράμματος, αλλάζοντας όμως και τον μετρητή του προγράμματος (PC).

Επιτρέπουν στο πρόγραμμα να μεταβεί σε μια συγκεκριμένη διεύθυνση χωρίς να αξιολογηθούν οποιεσδήποτε συνθήκες. Αυτές οι οδηγίες χρησιμοποιούνται συνήθως για την υλοποίηση κλήσεων συναρτήσεων, βρόγχων και αλλαγών ροής ελέγχου.

1. `j target` (Jump)

Η εντολή j μεταβαίνει στη διεύθυνση που καθορίζεται από το target, η οποία είναι μια ετικέτα στο πρόγραμμα.

Τροποποιεί τον μετρητή προγράμματος (PC) στη νέα διεύθυνση. Αυτό σημαίνει ότι η εκτέλεση του προγράμματος συνεχίζεται από τη νέα διεύθυνση.

ΠΑΡΑΔΕΙΓΜΑ Jumping Between Labels

```
.data
msg1: .asciiz "Jumping to Label 1!\n"
msg2: .asciiz "Jumping to Label 2!\n"
.text
.globl main
main:
        j label1        # Jump to 'label1'
label1:
        li $v0, 4       # Load system call for printing a string
        la $a0, msg1    # Load address of msg1
        syscall         # Print message
j label2        # Jump to 'label2'
label2:
        li $v0, 4       # Load system call for printing a string
        la $a0, msg2    # Load address of msg2
        syscall         # Print message
        j exit          # Jump to exit
exit:
        li $v0, 10      # Load system call for exit
        syscall         # Terminate the program
```

∑το προηγούμενο παράδειγμα, το πρόγραμμα ξεκινά από τη main.

Μεταβαίνει “άνευ όρων” στο label1 χρησιμοποιώντας το j label1. ∑το label1, εκτυπώνει "Jumping to Label 1!" και μετά μεταβαίνει στο label2.

∑το label2, εκτυπώνει "Jumping to Label 2!". Τέλος, μεταβαίνει για έξοδο για τερματισμό του προγράμματος.

Output:

Jumping to Label 1!

Jumping to Label 2!

2. `jr $rs` (Jump Register)

Η εντολή jr μεταβαίνει στη διεύθυνση που είναι αποθηκευμένη σε έναν καταχωρητή (π.χ. $ra για επιστροφή από συναρτήσεις). Χρησιμοποιείται συνήθως σε ακολουθίες επιστροφής συναρτήσεων.

Ο μετρητής προγράμματος (PC) ορίζεται στην τιμή σε $rs. Η εκτέλεση συνεχίζεται από τη νέα διεύθυνση

ΠΑΡΑΔΕΙΓΜΑ Function Return Using `jr $ra`

```
.text
.globl main

main:
        li $a0, 5       # Load 5 into argument register $a0
        jal square      # Jump to function 'square' (stores return address in $ra)
        move $t0, $v0       # Store return value in $t0

        li $v0, 1       # Print result
        move $a0, $t0
        syscall

        li $v0, 10      # Exit
        syscall

# Function: square(x) = x * x
square:
        mul $v0, $a0, $a0   # Multiply x * x and store in $v0
        jr $ra              # Return to caller (main)
```

∑το προηγούμενο παράδειγμα

Το jal Square αποθηκεύει τη διεύθυνση επιστροφής σε $ra και μεταβαίνει στο τετράγωνο.

Το τετράγωνο υπολογίζει το x * x και το αποθηκεύει σε $v0.

jr $ra μεταπηδά πίσω στην εντολή μετά το jal στο main.

Το αποτέλεσμα εκτυπώνεται.

3. `jal target` (Jump and Link)

Το jal (Jump and Link) χρησιμοποιείται για κλήσεις συναρτήσεων.

α) αποθηκεύει τη διεύθυνση επιστροφής (διεύθυνση επόμενης εντολής) στον καταχωρητή $ra (Return Address Register).

β) μεταβαίνει στην ετικέτα στόχο (σημείο εισόδου συνάρτησης).

jal function_label
Αποθηκεύει τη διεύθυνση επιστροφής σε $ra.
Μεταφέρει τον έλεγχο στο function_label.

Η διεύθυνση επιστροφής έχει οριστεί σε PC + 4 (η οδηγία μετά το jal)

Παράδειγμα Function Call Using `jal` and `jr`

```
.text
.globl main

main:
      li $a0, 10        # Load 10 into $a0 (first argument)
      li $a1, 20        # Load 20 into $a1 (second argument)

      jal sum           # Call function 'sum'

      move $t0, $v0     # Store return value from $v0 to $t0

      li $v0, 1         # Print integer syscall
      move $a0, $t0     # Move sum result to $a0
      syscall           # Print result

      li $v0, 10        # Exit syscall
      syscall

# Function: sum(a, b) = a + b
sum:
      add $v0, $a0, $a1  # $v0 = $a0 + $a1 (return value)
      jr $ra            # Return to caller (main)
```

∑το προηγούμενο παράδειγμα

jal sum: Αποθηκεύει τη διεύθυνση επιστροφής σε $ra

Μετάβαση στη συνάρτηση sum

sum: Υπολογίζει $a0 + $a1.

Το αποτέλεσμα αποθηκεύεται στο $v0.

Επιστρέφει χρησιμοποιώντας jr $ra, το οποίο επιστρέφει στη main.

Το αποτέλεσμα 30 εκτυπώνεται.

Βιβλιογραφία: MIPS Assembly Language using QtSpim Ed Jorgensen Version 1.0 January 2013