

# ΗΡΥ 201– Ψηφιακοί Υπολογιστές

## Αρχιτεκτονική Συνόλου Εντολών MIPS

Γιάννης Παπαευσταθίου  
2016

# Μοντέλο Προγραμματισμού 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, ... \$sp, \$ra
  - Προς το παρόν χρησιμοποιούμε την πρώτη ονοματολογία (\$0 - \$31). Αργότερα θα δούμε την λογική και την χρήση της δεύτερης ονοματολογίας
- Ο καταχωρητής \$0 έχει *πάντα* την τιμή 0 (μηδέν)
- Ο μετρητής προγράμματος (Program Counter => PC) είναι ένας ειδικός καταχωρητής ο οποίος έχει 32 bits

# Μοντέλο Προγραμματισμού MIPS #2

- Η πρόσβαση στη μνήμη γίνεται αποκλειστικά με εντολές load (φόρτωσης) και store (αποθήκευσης), από και προς την μνήμη και καταχωρητές
- Η μνήμη είναι οργανωμένη σε λέξεις (των 32 bits).
- Οι διευθύνσεις μνήμης είναι διευθύνσεις byte (8 bits). Η πρώτη λέξη έχει διεύθυνση 0, η δεύτερη έχει διεύθυνση 4, κ.λ.π.
- Όλες οι αριθμητικές και λογικές πράξεις αποθηκεύουν το αποτέλεσμα τους σε καταχωρητή
- Με μόνη εξαίρεση την ανάγνωση και την εγγραφή στην μνήμη, καμμία εντολή δεν αναφέρεται στην μνήμη

# Μοντέλο Προγραμματισμού MIPS #3

- Η εκτέλεση των εντολών γίνεται στην σειρά, από μικρότερες διευθύνσεις προς μεγαλύτερες, εκτός εάν η ροή του προγράμματος αλλάξει λόγω εντολής ελέγχου ροής (branch, ή jump)
- Ο καταχωρητής PC δείχνει στο σημείο στο οποίο βρίσκεται η εκτέλεση του προγράμματος.
- Οι εντολές έχουν πλάτος 32 bits (1 λέξη), και οι διευθύνσεις στις οποίες βρίσκονται είναι διευθύνσεις λέξεων (δηλαδή είναι πολλαπλάσια του 4)
- Στον ορισμό των όλων των εντολών εκτός των εντολών ελέγχου ροής, υπονοείται και η αύξηση του PC κατά 4 ώστε η εκτέλεση να προχωρήσει στην επόμενη εντολή

# MIPS Instruction Formats

Υπάρχουν 3 format εντολών στην αρχιτεκτονική συνόλου εντολών του MIPS:

- R-type (Register)

6	5	5	5	5	6
op	rs	rt	rd	shmt	func

  - Χρησιμοποιείται από εντολές που χρησιμοποιούν μόνο καταχωρητές (1 προορισμού και 2 πηγής).
- I-type (Immediate)

6	5	5	16
op	rs	rt	immed

  - Χρησιμοποιείται από εντολές που χρησιμοποιούν καταχωρητές και σταθερή (16 bit).
- J-type (Jump)

6	26
op	disp

  - Χρησιμοποιείται από εντολές ελέγχου ροής jump.

# Πεδία των εντολών

- op                    6 bits    κύριος κωδικός εντολής
- rt                    5 bits    N° καταχωρητή πηγής 2/προορισμού
- rs                    5 bits    N° καταχωρητή πηγής 1
- rd                    5 bits    N° καταχωρητή προορισμού
- shmt                5 bits    N° bits για ολίσθηση
- func                6 bits    πράξη για εντολές R-type
- immed              16 bits   σταθερές για εντολές I-type
- disp                26 bits   σταθερές για εντολές J-type

# R-type: add, sub

- add rd, rs, rt                      Πρόσθεση καταχωρητών
  - Σύνταξη: add \$1, \$2, \$3
  - Functionality
    - $RF[rd] = RF[rs] + RF[rt]$
    - Εάν υπάρχει υπερχείλιση, δημιουργία διακοπής/εξαίρεσης
- sub rd, rs, rt                      Αφαίρεση καταχωρητών
  - Σύνταξη: sub \$1, \$2, \$3
  - Functionality
    - $RF[rd] = RF[rs] - RF[rt]$
    - Εάν υπάρχει υπερχείλιση, δημιουργία διακοπής/εξαίρεσης

# I-type: addi, subi

- `addi rt, rs, Immed16` Πρόσθεση καταχωρητή-σταθερής
  - Σύνταξη: `addi $1, $2, 100`
  - Functionality
    - $RF[rt] = RF[rs] + SignExtend_{32}(Immed_{16})$
    - Εάν υπάρχει υπερχείληση, δημιουργία διακοπής/εξαίρεσης
- Παρατήρηση: δεν υπάρχει εντολή `subi`. Η αφαίρεση σταθερής γίνεται με την πρόσθεση του (-Σταθερή) χρησιμοποιώντας την εντολή `addi`



# R-type: or, and

- Bitwise λογικές πράξεις: η πράξεις (and, or, xor, κ.λ.π.) γίνονται στα bits των καταχωρητών της ίδιας θέσης, ανεξάρτητα από τα υπόλοιπα bits.
- or rd, rs, rt                      Λογική διάζευξη καταχωρητών
  - Σύνταξη: or \$1, \$2, \$3
  - Functionality
    - $RF[rd] = RF[rs] \mid RF[rt]$
- and rd, rs, rt                      Λογική σύζευξη καταχωρητών
  - Σύνταξη: and \$1, \$2, \$3
  - Functionality
    - $RF[rd] = RF[rs] \& RF[rt]$

# I-type: ori, andi

- ori rt, rs, Immed<sub>16</sub>    Λογική διάζευξη καταχωρητή-σταθερής
  - Σύνταξη: ori \$1, \$2, 100
  - Functionality
    - $RF[rt] = RF[rs] \mid ZeroExtend_{32}(Immed_{16})$
- andi rt, rs, Immed<sub>16</sub>    Λογική σύζευξη καταχωρητή-σταθερής
  - Σύνταξη: andi \$1, \$2, 100
  - Functionality
    - $RF[rt] = RF[rs] \& ZeroExtend_{32}(Immed_{16})$

## R-type: xor, nor

- xor rd, rs, rt                      Λογική πράξη XOR καταχωρητών
  - Σύνταξη: xor \$1, \$2, \$3
  - Functionality
    - $RF[rd] = RF[rs] \wedge RF[rt]$
- nor rd, rs, rt                      Λογική πράξη NOR καταχωρητών
  - Σύνταξη: nor \$1, \$2, \$3
  - Functionality
    - $RF[rd] = \neg (RF[rs] \vee RF[rt])$

# I-type: xori

- xori rt, rs, Immed<sub>16</sub> Λογική πράξη XOR καταχωρητή-σταθεράς
  - Σύνταξη: xori \$1, \$2, 100
  - Functionality
    - $RF[rt] = RF[rs] \wedge ZeroExtend_{32}(Immed_{16})$

## R-type: addu, subu (unsigned)

- addu rd , rs, rt      Πρόσθεση μη-προσημασμένων καταχωρητών
  - Σύνταξη: addu \$1, \$2, \$3
  - Functionality
    - $RF[rd] = RF[rs] + RF[rt]$
    - Η addu **δεν** ελέγχει για υπερχείληση και **δεν** δημιουργεί διακοπές/εξαιρέσεις
- subu rd, rs, rt      Αφαίρεση μη-προσημασμένων καταχωρητών
  - Σύνταξη: subu \$1, \$2, \$3
  - Functionality
    - $RF[rd] = RF[rs] - RF[rt]$
    - Η subu **δεν** ελέγχει για υπερχείληση και **δεν** δημιουργεί διακοπές/εξαιρέσεις

# I-type: addiu, subiu

- $\text{addiu } rt, rs, \text{Immed}_{16}$  Πρόσθεση καταχωρητή-σταθερής
  - Σύνταξη:  $\text{addiu } \$1, \$2, 100$
  - Functionality
    - $\text{RF}[rt] = \text{RF}[rs] + \text{SignExtend}_{32}(\text{Immed}_{16})$  **!! Ναι, SignExtend**
    - Η διαφορά από την  $\text{addi}$  είναι ότι η  $\text{addiu}$  δεν προκαλεί υπερχείληση, και δεν δημιουργία ποτέ διακοπή/εξαίρεση
- $\text{subiu}$  Δεν υπάρχει!

# R-type: sll (shift left logical)

- sll rd, rt, shamt      Ολίσθηση καταχωρητή προς αριστερά

- Σύνταξη : sll \$1, \$2, 5

- Functionality

$RF[rd] = LowOrderZeroFill_{32}(RF[rt] \ll shamt)$

- Παράδειγμα:

001100011101010101010101010101 << 5 =>

00110 00111010101010101010101010000

Τα 5 περισσότερα σημαντικά bits χάνονται, και τα 5 λιγότερο σημαντικά bits γεμίζουν με μηδενικά.

Παρατήρηση: η ολίσθηση προς τα αριστερά κατά N θέσεις είναι ισοδύναμη με πολλαπλασιασμό επί  $2^N$  χωρίς έλεγχο υπερχείλησης

# R-type:sllv (shift left logical variable)

- sllv rd, rt, rs                      Ολίσθηση καταχωρητή προς αριστερά μεταβλητών θέσεων (shift left logical variable)
  - Σύνταξη : sllv \$1, \$2, \$3
  - Functionality:  
(Ιδιο με την sll αλλά ο αριθμός των θέσεων της ολίσθησης δίνεται σε καταχωρητή)  
$$RF[rd] = LowOrderZeroFill_{32}(RF[rt] \ll RF[rs])$$



# R-type: srl (shift right logical)

- srl rd, rt, shamt      Ολίσθηση καταχωρητή προς δεξιά
  - Σύνταξη : srl \$1, \$2, 5
  - Functionality
    - $RF[rd] = ZeroFill_{32}(RF[rt] \gg shamt)$
  - Παράδειγμα:

00110001110101010101010100010101 >> 5

00000001100011101010101010101000 10101

Τα 5 λιγότερο σημαντικά bits χάνονται, και τα 5 περισσότερα σημαντικά bits γεμίζουν με αντίγραφο του περισσότερου σημαντικού bit.

10110001110101010101010100010101 >> 5

00000101100011101010101010101000 10101

Τα 5 λιγότερο σημαντικά bits χάνονται, και τα 5 περισσότερα σημαντικά bits γεμίζουν με αντίγραφο του περισσότερου σημαντικού bit.

# R-type: sra (shift right arithmetic)

- sra rd, rt, shamt      Δεξιά ολίσθηση καταχωρητή με πρόσημο
  - Σύνταξη : sra \$1, \$2, 5
  - Functionality
    - $RF[rd] = SignExtend_{32}(RF[rt] \gg shamt)$
  - Παρατήρηση: η ολίσθηση προς τα δεξιά κατά N θέσεις είναι **σχεδόν** ισοδύναμη με διαίρεση δια του  $2^N$   
( $1 \gg 5 = 0$ ,  $-1 \gg 5 = -1$ )
  - Παράδειγμα:

00110001110101010101010100010101 >> 5 =>

00000001100011101010101010101000 10101

Τα 5 λιγότερο σημαντικά bits χάνονται, και τα 5 περισσότερα σημαντικά bits γεμίζουν με αντίγραφο του περισσότερου σημαντικού bit.

10110001110101010101010100010101 >> 5 =>

11111101100011101010101010101000 10101

Τα 5 λιγότερο σημαντικά bits χάνονται, και τα 5 περισσότερα σημαντικά bits γεμίζουν με αντίγραφο του περισσότερου σημαντικού bit

## R-type: srlv, srav (shift right log/arithmetic variable)

- srlv rd, rt, rs      Δεξιά ολίσθηση καταχωρητή μεταβλητών μεταβλητών θέσεων (shift right logical variable)
  - Σύνταξη : srlv \$1, \$2, \$3
  - Functionality: (ίδιο με την srl αλλά ο αριθμός των θέσεων της ολίσθησης δίνεται σε καταχωρητή)  
$$RF[rd] = ZeroFill_{32}(RF[rt] \gg RF[rs])$$
- srav rd, rt, rs      Δεξιά ολίσθηση καταχωρητή μεταβλητών θέσεων με πρόσημο (shift right arithmetic variable)
  - Σύνταξη : srav \$1, \$2, \$3
  - Functionality: (ίδιο με την sra αλλά ο αριθμός των θέσεων της ολίσθησης δίνεται σε καταχωρητή)  
$$RF[rd] = SignExtend_{32}(RF[rt] \gg RF[rs])$$

# I-type: lui (load upper immediate)

- lui rt, Immed<sub>16</sub> Φόρτωση σταθερής σε άνω 16 bits καταχωρητή

- Σύνταξη : lui \$1, 0x100

- Functionality

- $RF[rt] = LowOrderZeroFill_{32}(Immed_{16} \ll 16)$

- Παράδειγμα:

**lui \$1, 0x100**

**=> \$1 = 0x01000000** ή ισοδύναμα σε δυαδικό :

**0000 0001 0000 0000 0000 0000 0000 0000**

# R-type: slt, sltu (set less-than)

- `slt rd, rt, rs`      σύγκριση ανισότητας καταχωρητών με πρόσημο
  - Σύνταξη : `slt $1, $2, $3`
  - Functionality
    - `/* σύγκριση ΜΕ πρόσημο */`
    - `if (RF[rt] < RF[rs]) RF[rd] = 1`
    - `else RF[rd] = 0`
- `sltu rd, rt, rs`      σύγκριση ανισότητας καταχωρητών χωρίς πρόσημο
  - Σύνταξη: `sltu $1, $2, $3`
  - Functionality
    - `/* σύγκριση ΧΩΡΙΣ πρόσημο */`
    - `if (RF[rt] < RF[rs]) RF[rd] = 1`
    - `else RF[rd] = 0`

# I-type: slti, sltiu (set less-than immediate)

- `slti rt, rs, Immed16` σύγκριση ανισότητας καταχωρητή-σταθερής με πρόσημο
  - Σύνταξη : `slti $1, $2, 100`
  - Functionality
    - /\* σύγκριση προσημασμένων αριθμών \*/*
    - if ( $RF[rs] < SignExtend_{32}(Immed_{16})$ )      $RF[rt] = 1$
    - Else      $RF[rt] = 0$
- `sltiu rt, rs, Immed16` σύγκριση ανισότητας καταχωρητή-σταθερής χωρίς πρόσημο
  - Σύνταξη: `sltiu $1, $2, $3`
  - Functionality
    - /\* σύγκριση χωρίς πρόσημο \*/*
    - if ( $RF[rs] < SignExtend_{32}(Immed_{16})$ )      $RF[rt] = 1$
    - else      $RF[rt] = 0$

# I-type: lw (load word)

- lw rt, Immed<sub>16</sub>(rs) ανάγνωση λέξης από την μνήμη
  - Σύνταξη : lw \$1, 100(\$2)
  - Functionality
    - MemAddr = RF[rs] + *SignExtend*<sub>32</sub>(Immed<sub>16</sub>)
    - if (*IsValidWordAddress*(MemAddr))
      - RF[rt] = *ReadMemoryWord*(MemAddr)
    - else
      - SignalIllegalAddressException*()

# I-type: sw (store word)

- $\text{sw rt, Immed}_{16}(\text{rs})$  εγγραφή λέξης από την μνήμη
  - Σύνταξη :  $\text{sw } \$1, 100(\$2)$
  - Functionality
    - $\text{MemAddr} = \text{RF}[\text{rs}] + \text{SignExtend}_{32}(\text{Immed}_{16})$
    - if ( $\text{IsValidWordAddress}(\text{MemAddr})$ )
      - $\text{WriteMemoryWord}(\text{MemAddr}, \text{RF}[\text{rt}])$
    - else
      - $\text{SignalIllegalAddressException}()$



# I-type: lb, lbu, sb (load/store byte)

- lb rt, Immed<sub>16</sub>(rs)                      Ανάγνωση byte από την μνήμη
  - Σύνταξη : lb \$1, 100(\$2)
  - Functionality
    - MemAddr = RF[rs] + *SignExtend*<sub>32</sub>(Immed<sub>16</sub>)
    - if (*IsValidByteAddress*(MemAddr))
    - RF[rt] = *SignExtend*<sub>32</sub>(*ReadMemoryByte*(MemAddr))
    - else
    - SignalIllegalAddressException*()
- lbu (load byte unsigned) όπως η lb, **χωρίς** επέκταση προσήμου
- sb rd, Immed<sub>16</sub>(rt)                      Εγγραφή λέξης από την μνήμη
  - Σύνταξη : sw \$1, 100(\$2)
  - Functionality
    - MemAddr = RF[rs] + *SignExtend*<sub>32</sub>(Immed<sub>16</sub>)
    - if (*IsValidByteAddress*(MemAddr))
    - WriteMemoryByte*(MemAddr, *LeastSignificant8Bits*(RF[rt]))
    - else
    - SignalIllegalAddressException*()

# I-type: beq (branch equal)

- `beq rs, rt, Offset16` Διακλάδωση εάν οι καταχωρητές είναι ίσοι
  - Σύνταξη : `beq $1, $2, 100`
  - Παρατήρηση: η 16-bit σταθερή που εμφανίζεται ως «offset» αναφέρεται στον αριθμό των εντολών που θα υπερπηδηθούν εάν η διακλάδωση είναι επιτυχής. Το offset μπορεί να είναι αρνητικό, οπότε έχουμε διακλάδωση προς τα πίσω.
  - Functionality

```
nextPC = PC + SignExtend32(Offset16 << 2)
if ((RF[rt] == RF[rs])) {
    PC = nextPC
} else /* condition is false, just execute next instruction: */
    PC = PC + 4
```

# I-type:bne (branch not equal)

- `bne rs, rt, Offset16` Διακλάδωση εάν οι καταχωρητές *δεν* είναι ίσοι
  - Σύνταξη : `bne $1, $2, 100`
  - Παρατήρηση: η συμπεριφορά της `bne` είναι ίδια με αυτή της `beq`, με μόνη διαφορά την συνθήκη που ελέγχεται (ανισότητα αντί για ισότητα)
  - Το «offset» είναι ο αριθμός (+/-) των εντολών που θα υπερπηδηθούν εάν η διακλάδωση είναι επιτυχής.
  - Functionality

```
nextPC = PC + SignExtend32(Offset16 << 2)
if ((RF[rt] != RF[rs])) {
    PC = nextPC
} else /* condition is false, just execute next instruction: */
    PC = PC + 4
```

# I-type: bgez (branch greater than or equal to zero)

- bgez rs, Offset<sub>16</sub> Διακλάδωση εάν ο καταχωρητής είναι μεγαλύτερος ή ίσος με το μηδέν
  - Σύνταξη : bgez \$1, 100
  - Παρατήρηση: η συμπεριφορά της bne είναι ίδια με αυτή της beq, με μόνη διαφορά την συνθήκη που ελέγχεται
  - Το «offset» είναι ο αριθμός (+/-) των εντολών που θα υπερπηδηθούν εάν η διακλάδωση είναι επιτυχής.
  - Functionality

```
nextPC = PC + SignExtend32(Offset16 << 2)
if ((RF[rs] >= 0) {
    PC = nextPC
} else /* condition is false, just execute next instruction: */
    PC = PC + 4
```

# I-type:bgtz (branch greater than zero)

- bgtz rs, Offset<sub>16</sub> Διακλάδωση εάν ο καταχωρητής είναι μεγαλύτερος από το μηδέν
  - Σύνταξη : bgtz \$1, 100
  - Παρατήρηση: η συμπεριφορά της bne είναι ίδια με αυτή της beq, με μόνη διαφορά την συνθήκη που ελέγχεται
  - Το «offset» είναι ο αριθμός (+/-) των εντολών που θα υπερπηδηθούν εάν η διακλάδωση είναι επιτυχής.
  - Functionality

```
nextPC = PC + SignExtend32(Offset16 << 2)
if ((RF[rs] > 0) {
    PC = nextPC
} else /* condition is false, just execute next instruction: */
    PC = PC + 4
```

# I-type: blez (branch less than or equal to zero)

- `blez rs, Offset16` Διακλάδωση εάν ο καταχωρητής είναι μικρότερος ή ίσος με το μηδέν
  - Σύνταξη : `blez $1, 100`
  - Παρατήρηση: η συμπεριφορά της `blez` είναι ίδια με αυτή της `bgez` , με μόνη διαφορά την συνθήκη που ελέγχεται
  - Το «offset» είναι ο αριθμός (+/-) των εντολών που θα υπερπηδηθούν εάν η διακλάδωση είναι επιτυχής.
  - Functionality

```
nextPC = PC + SignExtend32(Offset16 << 2)
if ((RF[rs] <= 0) {
    PC = nextPC
} else /* condition is false, just execute next instruction: */
    PC = PC + 4
```

# I-type: bltz (branch less than zero)

- $\text{bltz rs, Offset}_{16}$  Διακλάδωση εάν ο καταχωρητής είναι μικρότερος ή ίσος με το μηδέν
  - Σύνταξη :  $\text{bltz } \$1, 100$
  - Παρατήρηση: η συμπεριφορά της  $\text{bltz}$  είναι ίδια με αυτή της  $\text{bgtz}$  , με μόνη διαφορά την συνθήκη που ελέγχεται
  - Το «offset» είναι ο αριθμός (+/-) των εντολών που θα υπερπηδηθούν εάν η διακλάδωση είναι επιτυχής.
  - Functionality

```
nextPC = PC + SignExtend32(Offset16 << 2)
if ((RF[rs] < 0) {
    PC = nextPC
} else /* condition is false, just execute next instruction: */
    PC = PC + 4
```

# J-type: j (jump)

- $j \text{ disp}_{26}$  Διακλάδωση (χωρίς συνθήκη)
  - Σύνταξη :  $j \ 100$
  - Το «disp» είναι η διεύθυνση της εντολής προορισμού.
  - Functionality
$$PC = PC_{31-28} \parallel (\text{disp}_{26} \ll 2)$$
  - Παρατήρηση: ο τελεστής  $\parallel$  στην παραπάνω γραμμή είναι το «bit concatenation». Η εντολή J συνθέτει την διεύθυνση προορισμού ολισθαίνοντας κατά δύο θέσεις (δηλ. πολλαπλασιάζοντας επί 4) το displacement, και γεμίζει τα 4 περισσότερο σημαντικά bits της διεύθυνσης με τα αντίστοιχα bits του PC.



J-type: jr (jump register)

- jr rs                                  Διακλάδωση (χωρίς συνθήκη)
  - Σύνταξη : jr \$12
  - Η διεύθυνση προορισμού δίνεται από τα περιεχόμενα του καταχωρητή rs.
  - Functionality

```
if (IsValidWordAddress(RF[rs]))  
    PC = RF[rs]  
else  
    SignalIllegalAddressException()
```

# J-type: jal (jump and link)

- $\text{jal disp}_{26}$  Διακλάδωση με αποθήκευση διεύθυνσης επιστροφής (κλήση υπορουτίνας)
  - Σύνταξη :  $\text{jal } 10000$
  - Το « $\text{disp}$ » είναι η διεύθυνση της εντολής προορισμού (πρώτη εντολή υπορουτίνας).
  - Functionality
    - $\text{RF}[31] = \text{Διεύθυνση Επόμενης Εντολής (μετά το } \text{jal}, \text{ δηλ. } \text{PC} + 4)$
    - $\text{PC} = \text{PC}_{31-28} \parallel (\text{disp}_{26} \ll 2)$
  - Παρατήρηση: ο τελεστής  $\parallel$  στην παραπάνω γραμμή είναι το «bit concatenation». Η εντολή  $\text{jal}$  πρώτα αποθηκεύει την διεύθυνση της επόμενης εντολής στον καταχωρητή 31 (return address,  $\$ra$ ), και μετά διακλαδίζεται στην διεύθυνση που ορίζει το displacement, με τον ίδιο τρόπο όπως και η  $\text{j}$

# J-type: jalr (jump and link register)

- jalr rs      Διακλάδωση μέσω καταχωρητή με αποθήκευση διεύθυνσης επιστροφής (κλήση υπορουτίνας)
  - Σύνταξη : jalr \$12
  - Η διεύθυνση της υπορουτίνας δίνεται από τα περιεχόμενα του καταχωρητή rs.
  - Functionality
    - RF[31] = ΔιεύθυνσηΕπόμενηςΕντολής (μετά το jalr, δηλ.  $PC + 4$ )
    - if (*IsValidWordAddress*(RF[rs]))
      - PC = RF[rs]
    - else
      - SignalIllegalAddressException()*