
# Εργαστήριο 8.1

**Χειμερινό Εξάμηνο 2015-2016**

## 1. Επέκταση της μικρο-αρχιτεκτονικής διοχέτευσης του MIPS (7 μονάδες)

Στο εργαστήριο αυτό θα πρέπει να κάνετε επεκτάσεις στην αρχιτεκτονική του MIPS που ολοκληρώσατε στο Εργαστήριο 6.

### Υλοποίηση εντολών αλλαγής ροής προγράμματος (7 μονάδες)

Να επεκταθεί η προηγούμενη μικρο-αρχιτεκτονική όπως αναπαρίσταται και στην εικόνα 1) ώστε να μπορεί να εκτελέσει και τις παρακάτω εντολές διακλάδωσης MIPS:

- **`j Label`**
- **`beq rs, rt, Label`**
- **`bne rs, rt, Label`**

Εκτός από την σωστή εκτέλεση των εντολών αυτών, θα πρέπει να υλοποιήσετε και οποιαδήποτε επέκταση μηχανισμών προώθησης (forwarding), καθυστέρησης (stalling) και εκκένωσης (flushing) που είναι δυνατόν να προκαλέσουν οι επιπλέον αυτές εντολές. Για παράδειγμα, σκεφτείτε την επίδραση που έχουν οι εντολές διακλάδωσης στις εντολές που είναι ήδη μέσα στο pipeline. Είστε ελεύθεροι να υλοποιήσετε την εκτέλεση των εντολών διακλάδωσης σε όποιο στάδιο της μικρο-αρχιτεκτονικής επιθυμείτε, με την προϋπόθεση να πραγματοποιείται σωστά η προώθηση, καθυστέρηση και εκκένωση σε όλες τις περιπτώσεις.

Θα πρέπει να χρησιμοποιήσετε και να επεκτείνετε το αρχείο *control.v* και το αρχείο *cpu.v*. Ο κώδικας σας θα πρέπει να τρέχει σωστά για κάθε περίπτωση όπως για παράδειγμα για τον κώδικα που φαίνεται παρακάτω. Στα σχόλια μπορείτε να βρείτε και τις αναμενόμενες τιμές των καταχωρητών και της μνήμης μετά από κάθε εκτελούμενη εντολή και στις δύο επαναλήψεις του loop. Θεωρούμε ότι κάθε καταχωρητής αρχικοποιείται με την τιμή `reg[i] = i`.

```
    add $t0, $t0, $s0    # $t0 = $8 = 24 (Decimal)
    sw $ra, 8($t2)       # Mem[$t2+8] = 31
    lw $t7, 8($t2)       # $t7 = $15 = Mem[$t2+8] = 31
    sub $t1, $t1, $a0    # $t1 = $9 = 5
    or $t6, $t7, $t5     # $t6 = $14 = 31
    and $s3, $s0, $s2    # $s3 = $19 = 16
L1:
    lw $t6, 8($t2)      # $t6 = Mem[$t2+8] = 31,  $t6 = $14 = Mem[$t2+8] = 28
    sw $gp, 8($t2)       # Mem[$t2+8] = 28, Mem[$t2+8] = 28
    sll $s0, $t5, 1      # $s0 = $16 = 26,  $s0 = $16 = 28
    lw $v0, 8($t2)       # $v0 = $2 = 28, $v0 = $2 = 28
    beq $v0, $s0, L2    # $2, $16. RAW stall, First pass NOT TAKEN, SECOND
PASS TAKEN
    addi $t5, $t5, 1     # $t5 = $13 = 14
    and $a0, $v0, $t5    # $a0 = $4 = 12
    or $a0, $a0, $t3     # bypass from ALU. $a0 = $4 = 15
    add $t1, $a0, $v0    # bypass from ALU. $t1 = $9 = 43
    slt $sp, $a0, $t1    # $sp = $29 = 1
    lw $v1, 8($t2)       # $v1 = $3 = Mem[$t2+8] = 28
    addi $t4, $v1, -1020    # $t4 = $12 = -992
    add $t4, $t4, $t4       # $t4 = $t4 = -1984
    sll $s4, $v0, 12        # $s4 = $20 = 114688
    sllv $s6, $s4, $sp      # $s6 = $22 = 229376
    j L1                    # jump once
L2:
    add $t5, $t5, $t5       # $t5 = $13 = 28
    xor $t0, $t0, $t1       # $t0 = $8 = 51
    addi $t4, $t3, 2        # $t4 = $12 = 13
    or  $t6, $t5, $t4       # $t6 = $14 = 29
```

## Ερώτηση Bonus

### Αρχείο Καταχωρητών (Register File) (4 μονάδες)

Μέχρι τώρα, έχουμε υποθέσει ότι οι καταχωρητές του MIPS γράφονται στην αρνητική ακμή του ρολογιού. Αυτή η υπόθεση μας βοηθάει στο να μειώσουμε τον αριθμό των εξαρτημένων εντολών που διαβάζουν έναν καταχωρητή που γράφεται από μία προηγούμενη εντολή.

Σε πραγματικές όμως μικρο-αρχιτεκτονικές, οι καταχωρητές διαβάζονται και γράφονται στην θετική ακμή του ρολογιού όπως όλοι οι υπόλοιποι καταχωρητές. Η επέκταση της μικρο-αρχιτεκτονικής είναι να πραγματοποιήσετε ότι αλλαγές απαιτούνται στο αρχείο καταχωρητών ώστε αυτοί και να διαβάζονται και να γράφονται στην θετική ακμή του ρολογιού.

Να προσέξετε ιδιαίτερα την περίπτωση που στον ίδιο κύκλο μηχανής μία εντολή γράφει σε έναν καταχωρητή από τον οποίο διαβάζει μία άλλη εντολή. Ενώ η εγγραφή στην αρνητική ακμή του ρολογιού αντιμετώπιζε αυτό το πρόβλημα στις προηγούμενες υλοποιήσεις, σε αυτήν την υλοποίηση θα πρέπει εσείς να επιλύσετε αυτό το πρόβλημα και να εξασφαλίσετε ότι τα δεδομένα που διαβάζονται είναι τα σωστά. Επίσης η ανάγνωση των καταχωρητών θα πρέπει να είναι σύγχρονη με την θετική ακμή του ρολογιού.

Η νέα μικρο-αρχιτεκτονική θα πρέπει να εκτελεί σωστά τον κώδικα της προηγούμενης ερώτησης.

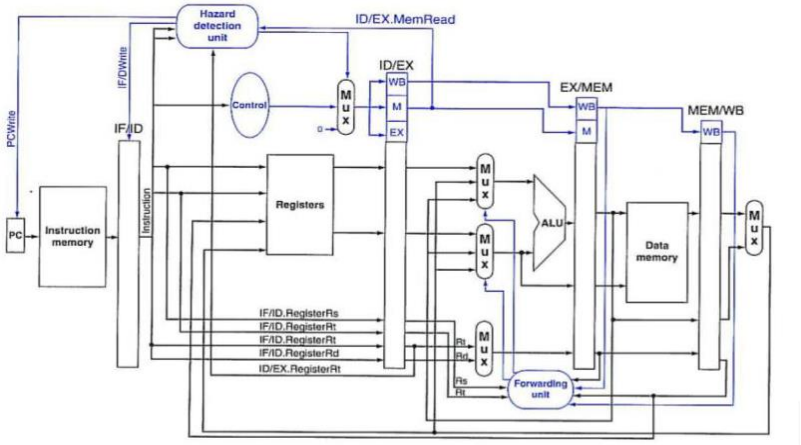


Figure 1. Αρχιτεκτονική MIPS με μηχανισμό διοχέτευσης (pipeline), μονάδα ανίχνευσης κινδύνου και μονάδα προώθησης (bypass). Η αρχιτεκτονική αυτή μπορεί να εκτελέσει εντολές format-R, καθώς και εντολές φόρτωσης, αποθήκευσης (load/store). Η εικόνα είναι όμοια με την εικόνα 4.60 του βιβλίου.

## 2. Θεωρητική άσκηση: Αρχιτεκτονική Διοχέτευσης και Καθυστερήσεις (3 μονάδες)

Έστω το παρακάτω loop το οποίο χρησιμοποιείται για να διατρέξουμε ένα δυαδικό δέντρο αναζήτησης ύψους *H*. Υπενθυμίζουμε ότι σε ένα τέτοιο δέντρο με ρίζα R, όλοι οι κόμβοι του αριστερού υποδέντρου αποθηκεύουν τιμές που είναι μικρότερες ή ίσες της τιμής του R, και όλοι οι κόμβοι του δεξιού υποδέντρου αποθηκεύουν τιμές που είναι μεγαλύτερες ή ίσες της τιμής του R:

```
loop: lw  $t0, 0($a0)
      beq $a1, $t0, exit
      bge $a1, $t0, gt
      lw  $a0, 4($a0)
      j   end
gt:   lw  $a0, 8($a0)
end:  bne $a0, $0, loop
exit:
```

a) Κυκλώστε <u>όλες</u> τις εξαρτήσεις δεδομένων (data dependencies) μέσα σε μία επανάληψη του loop. Ενώστε τους κύκλους με ένα βέλος που να δείχνει την πηγή και τον προορισμό της εξάρτησης.

b) Θεωρείστε ότι ο κώδικας εκτελείται στην γνωστή μας μικρο-αρχιτεκτονική διοχέτευσης των 5 σταδίων με την χρήση των τεχνικών προώθησης (forwarding) και καθυστέρησης (stalling), καθώς και την χρήση δυναμικής πρόβλεψης διακλάδωσης (branch prediction) των 2-bits. Θεωρείστε ότι η πραγματική κατεύθυνση των εντολών διακλάδωσης αποφασίζεται στο στάδιο EX και ότι η αρχική κατάσταση της FSM δυναμικής πρόβλεψης για την εντολή bge είναι "Predict Strongly Not Taken". Για τις υπόλοιπες εντολές διακλάδωσης θεωρούμε ότι η πρόβλεψη είναι πάντα επιτυχής από τις αντίστοιχες FSMs δυναμικής πρόβλεψης.

Συμπληρώστε έναν πίνακα σαν τον παρακάτω για 2 επαναλήψεις του loop θεωρώντας ότι η εντολή bge είναι Taken (T) την πρώτη φορά και Not Taken (NT) την δεύτερη φορά που εκτελείται το loop. Η εντολή beq είναι Not Taken και η εντολή bne είναι Taken και τις δύο φορές. Σε ποιόν κύκλο τερματίζεται η δεύτερη επανάληψη του loop?

| **Εντολή** | **Επανάληψη** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **0** | **1** | **2** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| lw | 1 | F | D | E | M | W | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | |

c) Έστω η εκτέλεση του παραπάνω κώδικα σε ένα δυαδικό δέντρο έτσι ώστε η εντολή bge να εκτελείται σύμφωνα με το pattern: 0000111100001111…, όπου το 0 αντιστοιχεί σε εκτέλεση NT και το 1 σε εκτέλεση T. Η συγκεκριμένη εκτέλεση φτάνει μέχρι το φύλλο ενός δέντρου και θεωρούμε ότι το *H* είναι ένας πολύ μεγάλος αριθμός. Συνεπώς, δεν μας ενδιαφέρει η αρχική κατάσταση της FSM δυναμικής πρόβλεψης, αλλά μόνο η σταθερή της κατάσταση (steady state).

Υπολογίστε το μέσο χρόνο εκτέλεσης κάθε επανάληψης του loop.

**Σημείωση**: θεωρείστε ότι η εντολή bge είναι κανονική εντολή και όχι ψευδοεντολή. Επίσης, θεωρείστε ότι οι εντολές beq και bne είναι πάντα Not Taken και Taken, αντίστοιχα και ότι προβλέπονται πάντα σωστά και χωρίς ποινή.

[4]