Εθνικο & Καποδιστριακο Πανεπιστημιο Αθηνων
Τμημα Πληροφορικης & Τηλεπικοινωνιων
Μαθημα: **Αρχιτεκτονικη Υπολογιστων ΙΙ**

# Εργασία 1 (υποχρεωτική) – Διοχέτευση

**Ακαδημαϊκο Ετος 2023 – 2024**
(εκφωνηση) Παρασκευη 10 Νοεμβριου 2023
(παραδοση στο eclass μεχρι) **Δευτερα 11 Δεκεμβριου 2023**

| Επώνυμο | Όνομα | Αριθμός Μητρώου | Email |
|---|---|---|---|
| Νταής | Παύλος | 1115202100122 | pavlosdais@gmail.com |
| Αρκουλής | Κωνσταντίνος | 1115202100008 | arkoulis.kostas02@gmail.com |
| | | | |

**Πληροφορίες για τις Υποχρεωτικές Εργασίες του μαθήματος**

- Οι υποχρεωτικές εργασίες του μαθήματος είναι **δύο**. Σκοπός τους είναι η κατανόηση των εννοιών του μαθήματος με χρήση αρχιτεκτονικών προσομοιωτών. Η πρώτη υποχρεωτική εργασία (αυτή) αφορά τη διοχέτευση (pipelining) και η δεύτερη θα αφορά τις κρυφές μνήμες (cache memories).
- Οι δύο εργασίες είναι υποχρεωτικές και η βαθμολογία του μαθήματος θα προκύπτει από το γραπτό (60%), την εργασία της διοχέτευσης (20%), και την εργασία των κρυφών μνημών (20%).
- Κάθε ομάδα μπορεί να αποτελείται **από 1 έως και 3 φοιτητές**. Συμπληρώστε τα στοιχεία όλων των μελών της ομάδας στον παραπάνω πίνακα. Όλα τα μέλη της ομάδας πρέπει να έχουν ισότιμη συμμετοχή και να γνωρίζουν τις λεπτομέρειες της υλοποίησης της ομάδας.
- Για την εξεταστική Σεπτεμβρίου δε θα δοθούν άλλες εργασίες. Το Σεπτέμβριο εξετάζεται μόνο το γραπτό.
- Σε περίπτωση αντιγραφής θα μηδενίζονται όλες οι ομάδες που μετέχουν σε αυτή.
- Η παράδοση της **Εργασίας Διοχέτευσης** πρέπει να γίνει μέχρι τα **μεσάνυχτα της προθεσμίας ηλεκτρονικά** και μόνο στο eclass (να ανεβάσετε ένα μόνο αρχείο zip ή rar με την τεκμηρίωσή σας σε PDF και τον κώδικά σας). **Μην περιμένετε μέχρι την τελευταία στιγμή. Η εκφώνηση της εργασίας 2 των κρυφών μνημών θα ανατεθεί αμέσως μετά**.

**Ζητούμενο**

Το ζητούμενο της εργασίας είναι η συγγραφή **ενός μόνο προγράμματος συμβολικής γλώσσας MIPS** και η εκτέλεσή του στον προσομοιωτή QtMips με σκοπό την εξαγωγή πληροφορίας για τη διαμόρφωση του μικροεπεξεργαστή μόνο με χρήση των εξής πληροφοριών: (α) **πλήθος κύκλων** που διαρκεί η εκτέλεση του προγράμματος και (β) **αποτέλεσμα του προγράμματος** (που θα είναι ονοματισμένες μεταβλητές σας στο data segment). Το πρόγραμμα δεν θα πρέπει να εκτελεί είσοδο ή έξοδο και οι υπολογισμοί του πρέπει να γίνονται υποχρεωτικά επί μεταβλητών εισόδου στο data segment και τα αποτελέσματά του να αποθηκεύονται σε μεταβλητές εξόδου και πάλι στο data segment.

Φανταστείτε (όπως αναφέρθηκε στο μάθημα) ότι δεν γνωρίζετε ποια διαμόρφωση έχει ο επεξεργαστής MIPS (μεταξύ αυτών που αναφέρονται παρακάτω) και πρέπει να κατασκευάσετε ένα πρόγραμμα που θα εκτελεστεί σε αυτόν. Μετά την εκτέλεση θα έχετε στη διάθεσή σας τον αριθμό κύκλων ρολογιού της εκτέλεσης και το αποτέλεσμα που υπολογίστηκε. Με βάση μόνο αυτά πρέπει να μπορείτε να «μαντέψετε» ποια διαμόρφωση έχει ο επεξεργαστής.

Οι πιθανές διαμορφώσεις του μικροεπεξεργαστή είναι οι εξής: **(Α) για τους κινδύνους δεδομένων**: (α1) καμία ενέργεια, (α2) ανίχνευση και καθυστέρηση σε κίνδυνο δεδομένων (stall on data hazard), (α3) ανίχνευση και πλήρης προώθηση (forwarding), **(Β) για τους κινδύνους ελέγχου**: (β1) καθυστέρηση σε διακλάδωση (stall on branch), (β2) υποδοχή καθυστέρησης (delay slot), (β3, β4) πρόβλεψη διακλάδωσης (branch predictor) με 1 ή με 2 bit. Όταν υπάρχει πρόβλεψη διακλάδωσης τα bit του BTB είναι υποχρεωτικά 5 και η επίλυση της διακλάδωσης (branch resolution) γίνεται υποχρεωτικά στο στάδιο EX. Σε αυτή την εργασία θεωρούμε ότι η μνήμη είναι ιδανική και συνεπώς σε όλους τους επεξεργαστές οι κρυφές μνήμες πρέπει να είναι απενεργοποιημένες και η κύρια μνήμη να έχει χρόνο προσπέλασης 1 κύκλο ρολογιού.

**Τεκμηρίωση**

[ Σύντομη τεκμηρίωση της λύσης σας μέχρι **10 σελίδες ξεκινώντας από την επόμενη σελίδα** – μην αλλάζετε τη μορφοποίηση του κειμένου (**και παραδώστε την τεκμηρίωση σε αρχείο PDF**). Η τεκμηρίωσή σας πρέπει να περιλαμβάνει παραδείγματα ορθής εκτέλεσης του προγράμματος και σχολιασμό για την επίλυση του προβλήματος και την επίτευξη του ζητούμενου. Μπορείτε να χρησιμοποιήσετε εικόνες, διαγράμματα και ό,τι άλλο μπορεί να βοηθήσει στην εξήγηση της δουλειάς σας. ]

**Παραδοτέο**

Ο φάκελος της εργασίας περιλαμβάνει τα εξής αρχεία:

- Prj1_Report.pdf: η εκφώνηση της εργασίας μαζί με την αναφορά σχετικά με την υλοποίηση μας.
- Results.xlsx: ο πίνακας των τελικών αποτελεσμάτων του προγράμματος για το mapping του configuration (περισσότερα παρακάτω).
- config_finder.S : το πρόγραμμα της εργασίας.

**Ζητούμενο**

Σκοπός της εργασίας αυτής, είναι η εύρεση των ρυθμίσεων (configuration) που έχει ο χρήστης στον προσομοιωτή Qtmips, μέσω της εκτέλεσης ενός μόνο προγράμματος.

Τα δυνατά configurations που μπορεί να έχει ο χρήστης, στα πλαίσια αυτής της εργασίας, είναι τα εξής δώδεκα :

| | **Data Hazards** | **Control Hazards** |
|---|---|---|
| 1. | Nothing | Stall on Branch |
| 2. | Nothing | Delay Slot |
| 3. | Nothing | Branch Predictor 1 BIT |
| 4. | Nothing | Branch Predictor 2 BIT |
| 5. | Stall | Stall on Branch |
| 6. | Stall | Delay Slot |
| 7. | Stall | Branch Predictor 1 BIT |
| 8. | Stall | Branch Predictor 2 BIT |
| 9. | Forwarding | Stall on Branch |
| 10. | Forwarding | Delay Slot |
| 11. | Forwarding | Branch Predictor 1 BIT |
| 12. | Forwarding | Branch Predictor 2 BIT |

Σε αυτή την υλοποίηση, ακολουθώντας τις οδηγίες που δίνονται, φέρνουμε εις πέρας τον παραπάνω στόχο εκμεταλλευόμενοι τρία πράγματα:

1. Total CPU Cycles (CC)
2. Την τιμή του πρώτου παραγώμενου αποτελέσματος που αποθηκεύεται στην ονοματισμένη μεταβλητή *results* του data segment, στην θέση μνήμης *0x2000*. Στην περίπτωση που η τιμή αυτή είναι ίση με 0, αυτό σημαίνει ότι θα έχουμε Nothing για τα Data Hazards, αντίθετα, αν η τιμή αυτή είναι 1, μπορεί να έχουμε Stall ή Full Forwarding για την αντιμετώπιση των προαναφερόμενων κινδύνων. Ο λόγος για αυτή την διαφοροποίηση θα δοθεί λεπτομερώς παρακάτω.
3. Την τιμή του δεύτερου παραγώμενου αποτελέσματος που αποθηκεύεται στην ονοματισμένη μεταβλητή *results+4* του data segment. Στην περίπτωση που η τιμή αυτή είναι ίση με 1, αυτό σημαίνει ότι θα έχουμε Delay Slot για τα Control Hazards, αντίθετα, αν η τιμή αυτή είναι 0, μπορεί να έχουμε Stall on

Branch ή Branch Prediction 1 bit ή Branch Prediction 2 bit, για την αντιμετώπιση των προαναφερόμενων κινδύνων. Και σε αυτή την περίπτωση, ο λόγος για αυτή την διαφοροποίηση θα δοθεί λεπτομερώς παρακάτω.

Οι τιμές στο 2. και 3. που μόλις αναφέραμε, λειτουργούν ως "tie breaks" για τα αποτελέσματα μας που έχουν ίδιο αριθμό από συνολικούς κύκλους επεξεργαστή.

**Μετάφραση των αποτελεσμάτων σε configuration**

Μετά την εκτέλεση του προγράμματος, αρκεί ο χρήστης να ψάξει στο Results.xlsx για το συγκεκριμένο tuple που του δίνει το πρόγραμμα και το εργαλείο Qtmips, κοιτώντας τα Total Cycles (**CC**) στο παράθυρο Cycle Statistics, την τιμή στη διεύθυνση μνήμης 0x2000 (ο λόγος για την επιλογή αυτής της διεύθυνσης δίνεται παρακάτω), έστω **R1** και την τιμή στη διεύθυνση μνήμης 0x2004 έστω **R2.** Έτσι λοιπόν, έχουμε φτιάξει τον αντίστοιχο πίνακα από tuples της μορφής (**CC**, **R1**, **R2**) για να γίνει το mapping: ***tuple -> configuration***.

| | A | B | C | D |
|---|---|---|---|---|
| 1 | (CC, R1, R2) | Nothing | Stall | Full Forwarding |
| 2 | Stall on branch | 33/0/0 | 39/1/0 | 33/1/0 |
| 3 | Delay slot | 18/0/1 | 22/1/1 | 18/1/1 |
| 4 | Branch prediction 1 | 31/0/0 | 37/1/0 | 31/1/0 |
| 5 | Branch prediction 2 | 29/0/0 | 35/1/0 | 29/1/0 |

*Figure 1: Πίνακας αποτελεσμάτων για mapping configuration*

Έτσι για παράδειγμα, αν το παραγώμενο tuple είναι το ***(31/1/0)***, τότε οι ρυθμίσεις του χρήστη είναι **Full Forwarding** για Data Hazards και **Branch Prediction με 1 bit** στο BHT για Control Hazards.

**Επεξήγηση λειτουργίας προγράμματος και λεπτομέρειες υλοποίησης**

Η ονοματισμένη μεταβλητή results του data segment, ξεκινά από την διεύθυνση 0x2000, επιλεγμένη συγκεκριμένα ώστε να μπορεί να φορτωθεί σε καταχωρητή μόνο με μια εντολή addi.

```
.data                                                   # Data segment, we store a tuple (x/y)
.org 0x2000                                             # x = 0 if we have Nothing for Data Hazards/ 1 if not
          results:                                      # y = 1 if we have Delay Slot for Control Hazards/ 1 if not
                    .word -1, -1
```

Εδώ πέρα, οι τιμές -1, -1 υποδηλώνουν uninitialized τιμές, τις οποίες θα αντικαταστήσουν τα πεδία **R1** και **R2** του παραγώμενου tuple, κατά την εκτέλεση του προγράμματος.

Παρακάτω θα γίνει ακριβής εξήγηση των παραγόμενων αποτελεσμάτων, χωρίζοντας το πρόγραμμα σε τρία κομμάτια (sections).

- Section 1

```
# Entry point of the program
__start:
          addi $s0, $zero, results              # Load the results' address into s0

DATA_HZRDS:
          addi $t3, $t3, 1                      # t3 = 1
          addi $t4, $t4, 2                      # t4 = 2

          addi $t0, $t0, 1                      # t0 = 1
          addi $t1, $t0, 0                      # t1 should be 1 here, if not we have Nothing for Data Hazards
          sw $t1, 0($s0)                        # Store t1's value
```

*Figure 2: Section 1*

Σε αυτό το κομμάτι κάνουμε trigger δύο data hazards:

1. Στο πρώτο, χρησιμοποιείται ο καταχωρητής *$to* ως καταχωρητής προορισμού στην εντολή *addi $to, $to, 1* και αμέσως μετά ως operand καταχωρητής σε μια άλλη πράξη στην εντολή ακριβώς από κάτω: *addi $t1, $to, 0*. Αν έχουμε Nothing για τα Data Hazards, τότε, αυτομάτως ο *$t1* θα έχει λάθος τιμή και ίση με 0 αντί για 1.

2. Στη δεύτερη πυροδότηση κινδύνου δεδομένων -αμέσως μετά τις παραπάνω δύο εντολές- εκτελείται η εντολή *sw $t1, 0($s0)*, όπου πάλι στην περίπτωση που έχουμε Nothing στο configuration για τα data hazards, θα αποθηκευτεί η τιμή *$to* = 0 (λάθος τιμή) αντί της τιμής *$to* = 1 (σωστή τιμή).

Τον ρόλο του *$to*, στο tuple των αποτελεσμάτων του Results.xlsx, διαδραματίζει η μεταβλητή **R1**.

Επομένως, λαμβάνοντας υπόψη τις διάφορες λεπτομέρειες υλοποίησης των 12 δυνατών ρυθμίσεων, παράγουμε τον παρακάτω πίνακα από συνολικούς κύκλους ρολογιού. Προσοχή, όμως, γιατί δεν προσμετράμε τους 4 τελευταίους κύκλους για την πλήρη εκτέλεση των τελευταίων 4 εντολών του section. Αυτοί οι κύκλοι θα προσμετρηθούν κατά την λήξη του προγράμματος στο εκάστοτε configuration. Θα πρέπει, στο τέλος, η άθροιση των κύκλων και από τα τρία αυτά sections σε αυτή την μελέτη, να δίνουν τους συνολικούς κύκλους στο tuple των αποτελεσμάτων του Results.xlsx.

| **CC of Section 1** | Nothing | Stall | Forwarding |
|---|---|---|---|
| Stall on Branch | 6 | 3+1+**2**+1+**2**+1 = 10 | 6 |
| Delay Slot | 6 | 3+1+**2**+1+**2**+1 = 10 | 6 |
| Branch Predictor 1 BIT | 6 | 3+1+**2**+1+**2**+1 = 10 | 6 |
| Branch Predictor 2 BIT | 6 | 3+1+**2**+1+**2**+1 = 10 | 6 |

Όπου με **κόκκινο** χρώμα σημειώνουμε τους κύκλους λόγω stalling από τα data hazards.

- Section 2

```
CONTROL_HZRDS:
          beq $zero, $zero, SKIP
          addi $t2, $t2, 1                    # If we have Delay Slot, then t2 will be 1
          nop
SKIP:     nop

          sw $t2, 4($s0)                      # Store t2's value

          beq $t2, $t3, EXIT                  # Configuration found with Delay Slot for Control Hazards
          nop
          nop
```

*Figure 3: Section 2*

Σε αυτό το κομμάτι δεν γίνεται πυροδότηση κανενός κινδύνου δεδομένων, φροντίζοντας να εισάγουμε εντολές nop στα σημεία που θα είχαμε τέτοιο κίνδυνο κανονικά. Αυτό που κάνουμε είναι φτιάξουμε την δεύτερη τιμή του παραγώμενου αποτελέσματος (**R2**) που θα είναι η τιμή του καταχωρητή *$t2* μετά την εκτέλεση του συγκεκριμένου section, διερευνώντας τη ρύθμιση την οποία έχει ο επεξεργαστής για τα Control Hazards. Συγκεκριμένα:

1. Μετά την εντολή *beq $zero, $zero, SKIP* στην περίπτωση που έχουμε Delay Slot, θα ληφθούν και οι επόμενες δύο εντολές μετά την παραπάνω εντολή διακλάδωσης, καθώς το Branch Resolution, σύμφωνα με την εκφώνηση, γίνεται (κλειδωμένα για όλα τα configurations) στο στάδιο EX.

2. Προφανώς η σωστή επίλυση του συγκεκριμένου branch είναι *TAKEN* για όλες τις εκτελέσεις του προγράμματος. Επομένως, στην περίπτωση του Delay Slot πάλι, μετά τις εντολές *addi $t2, $t2, 1* και ενός *nop*, θα αντιληφθεί το λάθος το control unit και θα πάει στο label SKIP που πάλι έχουμε άλλο ένα *nop*, για να καλύψουμε την περίπτωση που έχουμε Nothing για Data Hazards και να διασφαλίσουμε ότι ο καταχωρητής *$t2* έχει την σωστή του τιμή, που ως σωστή εδώ πέρα ορίζονται οι εξής δύο πιθανές τιμές :

    a. *$t2* = 1, αν έχουμε Delay Slot για τα Control Hazards.

    b. *$t2* = 0, αν έχουμε οποιαδήποτε άλλη ρύθμιση για την αντιμετώπιση των κινδύνων ελέγχου.

3. Έπειτα, αποθηκεύουμε (με ασφάλεια) την τιμή του *$t2* για την δημιουργία της τιμής **R2**.

4. Τέλος, ελέγχουμε αν η τιμή του καταχωρητή *$t2* είναι ίση με την *$t3* = 1. Δηλαδή, σε αυτή την εντολή διακλάδωσης, *TAKEN* σημαίνει ότι έχουμε Delay Slot. Για την σωστή εκτέλεση εδώ αυτού του branch -ανεξαρτήτως configuration- έχουν προστεθεί και δύο *nop* μετά την διακλάδωση. Επειδή έχουμε ήδη βρει πιο πάνω από την τιμή **R1** και την διαφορά κύκλων μεταξύ Stall και Forwarding, δηλαδή την ρύθμιση του επεξεργαστή για τα Data Hazards, στην περίπτωση που βρεθεί ότι έχουμε Delay Slot (TAKEN), μπορούμε με ασφάλεια να κάνουμε έξοδο από το πρόγραμμα μιας και έχει ήδη βρεθεί το configuration (με Delay Slot για τους κινδύνους ελέγχου και ένα από τα 3 δυνατά configurations για τους κινδύνους δεδομένων).

Σχόλιο: Η διαφορά εδώ πέρα ως προς την τιμή του *$t2* μεταξύ των ρυθμίσεων: Delay Slot - Branch Prediction 1 bit - Branch Prediction 2 bit, είναι ότι μετά το στάδιο EX της διακλάδωσης *beq $zero, $zero, SKIP*, οι τελευταίες δύο ρυθμίσεις, ακυρώνουν τις δύο εντολές που έχουν μπει στην επεξεργαστή (λόγω του αρχικού NOT TAKEN), αποτρέποντας την διαξεγωγή του σταδίου WB της εντολής *addi $t2, $t2, 1*. Αντίθετα η πρώτη ρύθμιση (Delay Slot) δεν τις ακυρώνει, οπότε και η τιμή **R2** του tuple εδώ θα είναι ίση με 1 ενώ, στις άλλες περιπτώσεις, ίση με την αρχική τιμή του *$t2* δηλαδή ίση με 0.

Επομένως, είμαστε σε θέση τώρα να παράγουμε τον παρακάτω πίνακα από συνολικούς κύκλους ρολογιού του συγκεκριμένου section προσέχοντας όμως, όπως και πριν, να μην προσμετρήσουμε τους 4 τελευταίους κύκλους για την πλήρη εκτέλεση των τελευταίων 4 εντολών του section αυτού, καθώς οι κύκλοι αυτοί θα προσμετρηθούν κατά την λήξη του προγράμματος στο εκάστοτε configuration.

| **CC of Section 2** | Nothing | Stall | Forwarding |
|---|---|---|---|
| Stall on Branch | 1+**2**+2+1+**2**+2 = 10 | 1+**2**+2+1+**2**+2 = 10 | 1+**2**+2+1+**2**+2 = 10 |
| Delay Slot | 3+2+3 = 8 | 3+2+3 = 8 | 3+2+3 = 8 |
| Branch Predictor 1 BIT | 3+2+3 = 8 | 3+2+3 = 8 | 3+2+3 = 8 |
| Branch Predictor 2 BITS | 3+2+3 = 8 | 3+2+3 = 8 | 3+2+3 = 8 |

Όπου, με **πράσινο** χρώμα σημειώνουμε τους κύκλους λόγω stalling από τα control hazards.

- Section 3

```
SOB_OR_BP1_OR_BP2:
        addi $t4, $t4, 0                    # Distinguish Stall or Forwarding for Data Hazards
        LOOP:                               # Iterations = 2
                addi $t4, $t4, -1
                nop
                nop
                bne $t4, $zero, LOOP

EXIT:
        break                               # Exit program
```

*Figure 4: Section 3*

Σε αυτό το τελευταίο κομμάτι κάνουμε τα εξής:

1. Πυροδοτούμε ένα *"artificial"* data hazard μέσω των διαδοχικών εντολών *addi $t4, $t4, 0* και *addi $t4, $t4, -1* για να προστεθούν δύο επιπλέον κύκλοι (στην περίπτωση που υπάρχει Stall για Data Hazards). Με τον όρο *"artificial"* εννοούμε ότι στην πραγματικότητα αφού προσθέτουμε 0 στο 0 (αρχική τιμή του uninitialized καταχωρητή *$t4*), για οποιαδήποτε ρύθμιση του χρήστη στα Data Hazards (Nothing, Stall, Forwarding) δεν θα επηρρεαστεί η διεξαγωγή του βρόγχου και η ολοκλήρωση του προγράμματος, παρά μόνο, θα επηρρεαστούν οι συνολικοί κύκλοι του (+2 για Stall στα Data Hazards).

   Ο λόγος που το κάνουμε αυτό είναι επειδή, χωρίς αυτήν την ενέργεια, παρατηρήθηκε ότι μερικά tuples της 2ης και 3ης στήλης που έχουν κοινές τιμές στα πεδία **R1**, **R2**, προέκυπταν επίσης με κοινές τιμές στο πεδίο **CC** με αποτέλεσμα να χάναν την μοναδικότητα τους και άρα και την ιδιότητα κλειδιού στο mapping μας. Η εξήγηση είναι απλή: το άθροισμα των stalls που πρόσθετε το πρόγραμμα μας λόγω data, control hazards πριν το LOOP και μετα το LOOP, αλληλοσυμπληρωνόντουσαν και το άθροισμα γινόταν ίσο για τους συνδυασμούς: Stall – Branch Prediction 1 bit και Forwarding - Branch Prediction 2 bit. Επομένως, επιλέξαμε να διαφοροποίησουμε αυτά τα tuples ως προς το **CC** πεδίο, δημιουργώντας την διαφορά αυτή μεταξύ των ρυθμίσεων Stall, Forwarding.

2. Εκτελούμε ένα βρόγχο δύο επαναλήψεων (LOOP) για να διαφοροποιήσουμε τους κύκλους που θα κάνει ο επεξεργαστής αναλογά με το αν έχει Stall on Branch, Branch Prediction 1 bit, Branch Prediction 2 bit για control hazards. Λαμβάνοντας υπόψη ότι έχουμε δύο iterations και λόγω της διακλάδωσης *bne $t4, $zero, LOOP*, προστίθονται τα εξής Control Hazard Stalls:

   a. Με Stall on Branch, 2 + 2 = 4 stalls. Αφού για κάθε iteration εκ των δύο, ο επεξεργαστής «σταλλάρει» μέχρι η εντολή διακλάδωσης να φτάσει στο branch resolution stage της δηλαδή το EX και άρα κάνει 2 stalls x 2 iterations.

   b. Με Branch Prediction 1 bit, 2 + 2 = 4 stalls. Διότι, στη πρώτη επανάληψη ο Branch History Table (BHT) έχει NOT TAKEN με αποτέλεσμα να κάνουμε mispredict και να μπούν 2 stalls. Έπειτα, στην δεύτερη επανάληψη, που κανονικά πρέπει μετά να κάνουμε exit από το loop, ο BHT έχει την τιμή TAKEN λόγω του προηγούμενου mispredict. Αποτέλεσμα εδώ είναι να γίνει και δεύτερο mispredict οπότε και να προστεθούν άλλα 2 stalls.

   c. Με Branch Prediction 2 bit, 2 stalls. Γιατί, στη πρώτη επανάληψη ο BHT έχει NOT TAKEN με αποτέλεσμα να κάνουμε mispredict και να μπούν 2 stalls. Έπειτα, στην δεύτερη επανάληψη ο BHT έχει την τιμή WEAK NOT TAKEN λόγω του προηγούμενου mispredict. Αποτέλεσμα

εδώ είναι η διακλάδωση να γίνει και πάλι NOT TAKEN και άρα να κάνουμε exit του LOOP όπως και πρέπει άλλωστε.

| **CC of Section 3** | Nothing | Stall | Forwarding |
|---|---|---|---|
| Stall on Branch | 1+2(4+2)+4 = 17 | 1+2+2(4+2)+4 = 19 | 1+2(4+2)+4 = 17 |
| Delay Slot | 4 | 4 | 4 |
| Branch Predictor 1 BIT | 1+4+2+4+2+4 = 17 | 1+2+4+2+4+2+4 = 19 | 1+4+2+4+2+4 = 17 |
| Branch Predictor 2 BIT | 1+4+2+4+4 = 15 | 1+2+4+2+4+4 = 17 | 1+4+2+4+4 = 15 |

Όπου, όπως προηγουμένως, με **κόκκινο** χρώμα σημειώνουμε τους κύκλους λόγω stalling από τα data hazards και με **πράσινο** χρώμα τους κύκλους λόγω stalling από τα control hazards. Τώρα, αφού έχουμε και την λήξη του προγράμματος με την εντολή *break*, που κάνει raise hardware exception στο 4ο στάδιο της (MEM) για να ολοκληρωθούν οι 3 εντολές πριν από αυτήν, θα προσθέσουμε 4 επιπλέον κύκλους (1 για το IF της break και άλλοι 3 μέχρι και το MEM).

Τελικά, αθροίζοντας τους παραπάνω πίνακες, επιβεβαιώνονται και οι συνολικοί κύκλοι καθενός configuration που βρίσκονται στο Results.xlsx:

| Total number of cycles | Nothing | Stall | Forwarding |
|---|---|---|---|
| Stall on Branch | 6+10+17 = **33** | 10+10+19 = **39** | 6+10+17 = **33** |
| Delay Slot | 6+8+4 = **18** | 10+8+4 = **22** | 6+8+4 = **18** |
| Branch predictor 1 BIT | 6+8+17 = **31** | 10+8+19 = **37** | 6+8+17 = **31** |
| Branch predictor 2 BITS | 6+8+15 = **29** | 10+8+17 = **35** | 6+8+15 = **29** |

Στην επόμενη σελίδα δίνουμε μια εικόνα ολόκληρου του προγράμματος με τα sections που έχουμε χωρίσει

Section 1

Section 2

Section 3

```
# Entry point of the program
__start:
        addi $s0, $zero, results        # Load the results' address into s0

DATA_HZRDS:
        addi $t3, $t3, 1                # t3 = 1
        addi $t4, $t4, 2                # t4 = 2

        addi $t0, $t0, 1                # t0 = 1
        addi $t1, $t0, 0                # t1 should be 1 here, if not we have Nothing for Data Hazards
        sw $t1, 0($s0)                  # Store t1's value

CONTROL_HZRDS:
        beq $zero, $zero, SKIP
        addi $t2, $t2, 1                # If we have Delay Slot, then t2 will be 1
        nop
SKIP:   nop

        sw $t2, 4($s0)                  # Store t2's value

        beq $t2, $t3, EXIT              # Configuration found with Delay Slot for Control Hazards
        nop
        nop

SOB_OR_BP1_OR_BP2:
        addi $t4, $t4, 0                # Distinguish Stall or Forwarding for Data Hazards
        LOOP:                           # Iterations = 2
                addi $t4, $t4, -1
                nop
                nop
                bne $t4, $zero, LOOP

EXIT:
        break                           # Exit program
```

*Figure 5: Program and its sections*

**Συμπεράσματα/Σχολιασμός**

Λόγω της φύσης της εργασίας αυτής, δεν υπάρχει κάτι που θα μπορούσαμε να σχολιάσουμε ως προς τα τελικά αποτέλεσματα. Αυτό, όμως, που μπορούμε να πούμε, είναι ότι αξιοποιώντας τον «όμορφο» προσομοιωτή Qtmips και έχοντας μια καλή διαίσθηση του pipeline 5 σταδίων (και των κινδύνων του) όπως αυτό έχει περιγραφεί στο μάθημα και στο εργαστήριο, μπορούμε να είμαστε σε θέση να διακρίνουμε τις ρυθμίσεις του επεξεργαστή του χρήστη και να κάνουμε ένα πολύ απλό “reverse engineering”. Η εργασία αυτή φάνηκε πολύ ενδιαφέρουσα και αν και απλή ως προς το ζητούμενο, εύκολα μπορούσε κανείς (όπως και εμείς κάναμε αρχικά εξάλλου) να φτιάξει ένα πρόγραμμα που κάνει διάφορα loops και εν τέλει να βγαίνουν διαφορετικοί κύκλοι για κάθε configurations. Τέτοια όμως προσέγγιση θεωρήσαμε ότι θα έδινε μεγάλη δυσκολία μετά στην εξήγηση των αποτελεσμάτων στο Report και παρόλο που είναι απλή στην υλοποίηση, θεωρήσαμε ότι είναι προτιμότερο να κάνουμε ένα πιο μικρό (σε εντολές και κύκλους) αλλά και πιο έξυπνο πρόγραμμα, για να φέρουμε εις πέρας το ζητούμενο. Έτσι λοιπόν, μπορέσαμε στο συγκεκριμένο Report να είμαστε σχολαστικοί και να εξηγήσουμε επακριβώς την συμπεριφορά του επεξεργαστή μέσα από το πρόγραμμα μας, εντολή προς εντολή.

**Καταμερισμός φόρτου**

Γενικότερα, τόσο η δημιουργία του προγράμματος όσο και η συγγραφή του συγκεκριμένου Report έγινε από κοινού και από τα δύο μέλη της ομάδας, βρισκόμενοι σε διαρκή συννενόηση.