A black background with a black square

Description automatically generated with medium confidenceΕθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Εαρινό Εξάμηνο 2023-2024

Προηγμενα ΘΕΜΑΤΑ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Λύσεις Θεμάτων

Ιωάννης (Χουάν) Τσαντήλας

03120883

Contents

[Κανονική 24 4](#_Toc179129383)

[Θέμα 1 4](#_Toc179129384)

[Ερώτημα Α 4](#_Toc179129385)

[Ερώτημα Β 7](#_Toc179129386)

[Σχόλιο Θεωρίας για το Ερώτημα 1.Α.1 9](#_Toc179129387)

[Θέμα 2 11](#_Toc179129388)

[Ερώτημα Α 11](#_Toc179129389)

[Ερώτημα Β 14](#_Toc179129390)

[Θέμα 3 2](#_Toc179129391)

[Ερώτημα Α 2](#_Toc179129392)

[Θέμα 4 3](#_Toc179129393)

[Ερώτημα Α 3](#_Toc179129394)

[Ερώτημα Β 4](#_Toc179129395)

[Κανονική 16 6](#_Toc179129396)

[Δική μου Λύση 6](#_Toc179129397)

[Κανονική 23 10](#_Toc179129398)

[Θέμα 1 10](#_Toc179129399)

[Ερώτημα Α 10](#_Toc179129400)

[Ερώτημα Β 11](#_Toc179129401)

[Ερώτημα Γ 13](#_Toc179129402)

[Θέμα 2 16](#_Toc179129403)

[Κανονική 19 19](#_Toc179129404)

[Θέμα 2 19](#_Toc179129405)

[Θέμα 4 22](#_Toc179129406)

[Κανονική 21 24](#_Toc179129407)

[Θέμα 3 24](#_Toc179129408)

A black background with a black square

Description automatically generated with medium confidenceΕθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Εαρινό Εξάμηνο 2023-2024

Προηγμενα ΘΕΜΑΤΑ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Λύσεις Θεμάτων

Ιωάννης (Χουάν) Τσαντήλας

03120883

[Github-Repo](https://github.com/ntua-el20883/ece-ntua2020)

Contents

[Κανονική 24 2](#_Toc177330570)

[Θέμα 1 2](#_Toc177330571)

[Ερώτημα Α 2](#_Toc177330572)

[Ερώτημα Β 5](#_Toc177330573)

[Σχόλιο Θεωρίας για το Ερώτημα 1.Α.1 7](#_Toc177330574)

[Θέμα 2 9](#_Toc177330575)

[Ερώτημα Α 9](#_Toc177330576)

[Ερώτημα Β 12](#_Toc177330577)

[Θέμα 3 2](#_Toc177330578)

[Κανονική 16 3](#_Toc177330579)

[Δική μου Λύση 3](#_Toc177330580)

# Κανονική 24

## Θέμα 1

### Ερώτημα Α

Δίνεται επεξεργαστής που για πρόβλεψη αλμάτων χρησιμοποιεί ένα (2,2) global history predictor με συνολικά 4 entries, κάθε ένα από τα οποία είναι ένας 2-bit saturating counter. Δίνεται επίσης ο παρακάτω κώδικας:

for (i = 0; i < 100000; i++) { //Branch B1

//B1 Taken path

if (i % 3 == 0) //Branch B2

//B2 Taken path

x[i] = y[i] + 2\*x[i]

}

1. Δώστε τις αρχικές τιμές των 2-bit saturating counters, ώστε ο επεξεργαστής να αποτυγχάνει 100% για τις 5 πρώτες επαναλήψεις του βρόχου. Υποθέστε πως αρχικά ο BHR είναι ίσος με 0.
2. Όταν προχωρήσει η εκτέλεση του κώδικα, είναι δυνατό στη μόνιμη κατάσταση (steady state) να εμφανίσει ο predictor 100% επιτυχία; Αν ναι δώστε τις αρχικές τιμές που οδηγούν σε αυτή, διαφορετικά υπολογίστε το μέγιστο ποσοστό επιτυχίας που μπορεί να επιτευχθεί.

#### Ερώτημα Α.1

Πρέπει να ορίσουμε τις αρχικές τιμές του counter έτσι ώστε να προβλέπουμε πάντα το αντίθετο από τα πραγματικά αποτελέσματα των κλάδων:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BHR | 00 | 01 | 10 | 11 |
| Counter |  |  |  |  |

Παρατηρούμε πως το Β1 Branch είναι πάντα Taken για όλες τις 100.000 επαναλήψεις (και άρα για τις 5 πρώτες). Το Β2 Branch είναι Taken μόνο στις επαναλήψεις 0 και 3.

Αφού είναι 2-bit counter, ο μετρητής παίρνει τις εξής τιμές:

* 00: Strongly NT
* 01: Weakly NT
* 10: Weakly T
* 11: Strongly T

Ξεκινούμε με BHR=00.

* **Επανάληψη 0 (Β1, Β2: Τ)**

Θέλουμε Β1: Τ, άρα counter=00 (ΝΤ):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BHR | 00 | 01 | 10 | 11 |
| Counter | 00 |  |  |  |

Ο counter θα πρέπει να γίνει 01 και το BHR θα γίνει 01.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BHR | 00 | 01 | 10 | 11 |
| Counter | **01** |  |  |  |

Θέλουμε Β2: Τ, άρα counter=01 (ΝΤ):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BHR | 00 | 01 | 10 | 11 |
| Counter | 01 | 01 |  |  |

Ο counter θα πρέπει να γίνει 10 και το BHR θα γίνει 11.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BHR | 00 | 01 | 10 | 11 |
| Counter | 01 | **10** |  |  |

* **Επανάληψη 1 (Β1: Τ, Β2: ΝΤ)**

Θέλουμε Β1: Τ, άρα counter=01 (ΝΤ):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BHR | 00 | 01 | 10 | 11 |
| Counter | 01 | 10 |  | 01 |

Ο counter θα πρέπει να γίνει 10 και το BHR θα παραμείνει 11.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BHR | 00 | 01 | 10 | 11 |
| Counter | 01 | 10 |  | **10** |

Θέλουμε Β2: ΝΤ. Ο counter είναι 10 (Τ). Ο counter θα πρέπει να γίνει 01 και το BHR θα γίνει 10.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BHR | 00 | 01 | 10 | 11 |
| Counter | 01 | 10 |  | **01** |

* **Επανάληψη 2 (Β1: Τ, Β2: ΝΤ)**

Θέλουμε Β1: Τ, άρα counter=00 (ΝΤ):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BHR | 00 | 01 | 10 | 11 |
| Counter | 01 | 10 | **00** | 01 |

Ο counter θα πρέπει να γίνει 01 και το BHR θα γίνει 01.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BHR | 00 | 01 | 10 | 11 |
| Counter | 01 | 10 | **01** | 01 |

Θέλουμε Β2: ΝΤ. Ο counter είναι 10 (Τ). Ο counter θα πρέπει να γίνει 01 και το BHR θα γίνει 10.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BHR | 00 | 01 | 10 | 11 |
| Counter | 01 | **01** | 01 | 01 |

* **Επανάληψη 3 (Β1, Β2: Τ)**

Θέλουμε Β1: Τ. Ο counter είναι 01 (ΝΤ). Ο counter θα πρέπει να γίνει 10 και το BHR θα γίνει 01.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BHR | 00 | 01 | 10 | 11 |
| Counter | 01 | 01 | **10** | 01 |

Θέλουμε Β2: Τ. Ο counter είναι 01 (ΝΤ). Ο counter θα πρέπει να γίνει 10 και το BHR θα γίνει 11.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BHR | 00 | 01 | 10 | 11 |
| Counter | 01 | **10** | 10 | 01 |

* **Επανάληψη 4 (Β1: Τ, Β2: ΝΤ)**

Θέλουμε Β1: Τ. Ο counter είναι 01 (ΝΤ). Ο counter θα πρέπει να γίνει 10 και το BHR θα παραμείνει 11.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BHR | 00 | 01 | 10 | 11 |
| Counter | 01 | 10 | 10 | **10** |

Θέλουμε Β2: ΝΤ. Ο counter είναι 10 (Τ). Ο counter θα πρέπει να γίνει 01 και το BHR θα γίνει 10.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BHR | 00 | 01 | 10 | 11 |
| Counter | 01 | 10 | 10 | **01** |

Επομένως, οι αρχικές τιμές του μετρητή είναι:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BHR | 00 | 01 | 10 | 11 |
| Counter | 00 | 01 | 00 | 01 |

#### Ερώτημα Α.2

***Consulted GPT-4o September 2024***

Η πρόβλεψη βασίζεται σε έναν 2-bit Branch History Register (BHR), ο οποίος παρακολουθεί μόνο τις δύο τελευταίες διακλαδώσεις (B1 και B2). Αυτό σημαίνει ότι ο predictor δεν μπορεί να διακρίνει ένα πλήρες μοτίβο 3 επαναλήψεων που απαιτείται για την ακριβή πρόβλεψη του B2 κάθε φορά. Ο BHR θα έχει πληροφορίες μόνο για τους δύο τελευταίους κλάδους, οπότε δεν θα γνωρίζει πάντα αν ο επόμενος κλάδος αντιστοιχεί στο (όπου το B2 είναι Taken) ή όχι.

Στην πράξη, για τα 2/3 των επαναλήψεων, το B2 είναι NT, γεγονός που θα κάνει τον predictor να συγκλίνει προς την πρόβλεψη NT για το B2 τις περισσότερες φορές. Ωστόσο, για το 1/3 των επαναλήψεων, το B2 είναι T, οδηγώντας σε λανθασμένες προβλέψεις κατά τη διάρκεια αυτών των περιπτώσεων.

Έτσι, ο προγνώστης μπορεί στην καλύτερη περίπτωση να προβλέψει σωστά το B2 περίπου 2/3 του χρόνου, το οποίο αντιστοιχεί σε ακρίβεια 66,67% για το B2.

Το συνολικό ποσοστό επιτυχίας είναι ο μέσος όρος των ποσοστών επιτυχίας για τα B1 και B2:

### Ερώτημα Β

Δίνεται GPU με 32 SIMD lanes και warp (δηλ. το σύνολο των παράλληλων νημάτων που εκτελούν την ίδια εντολή στον multithreaded SIMD επεξεργαστή) μεγέθους 32, όπου εκτελείται ο παρακάτω κώδικας:

for (i = 0; i < 1024; i++) {

A[i] = B[i] \* C[i]; //εντολή 1

if (/\* συνθήκη ελέγχου \*/){ //εντολή 2

//εντολή 3

...

//εντολή k + 2

}

C[i] = C[i] - 1; //εντολή k + 3

}

Κατά την εκτέλεση του κώδικα, κάθε νήμα αναλαμβάνει μια μοναδική επανάληψη του παραπάνω βρόχου, ενώ οι τιμές των πινάκων A, B και C έχουν φορτωθεί από την αρχή σε vector καταχωρητές και άρα δεν απαιτείται η εκτέλεση loads και stores. Υποθέστε επίσης πως οι B και C είναι πίνακες ακεραίων και όλες οι τιμές τους έχουν απόλυτη τιμή μικρότερη του 10.

1. Πόσα warps απαιτούνται για την εκτέλεση του προγράμματος;
2. Υποθέστε πως η συνθήκη ελέγχου είναι (). Ποια η τιμή του k (δηλ. το πλήθος των εντολών μέσα στο if block) για την οποία το SIMD utilization είναι ίσο με 11/32; (Ορισμός: SIMD utilization ονομάζουμε το ποσοστό των SIMD lanes που χρησιμοποιούνται από τα νήματα κατά μέσο όρο κατά τη διάρκεια της εκτέλεσης του προγράμματος). Υποθέστε πως το if block δεν περιέχει καμία εντολή ελέγχου ροής.
3. Υποθέστε τώρα πως η συνθήκη ελέγχου της εντολής if είναι (i % 8 == 0 && i < 256). Ποια τώρα η τιμή του k για την οποία το SIMD utilization είναι ίσο με 25/32;

***Consulted GPT-4o (09/2024)***

#### Ερώτημα Β.1

Ένα **warp** είναι ένα σύνολο νημάτων που εκτελούνται με βήμα κλειδώματος στον επεξεργαστή SIMD. Η GPU σε αυτή την περίπτωση διαθέτει:

* **32 SIMD lanes**, που σημαίνει ότι μπορεί να εκτελέσει 32 νήματα παράλληλα.
* Μέγεθος **warp** 32, που σημαίνει ότι κάθε warp αποτελείται από 32 νήματα.

Δεδομένου ότι ο βρόχος έχει 1024 επαναλήψεις και κάθε νήμα εκτελεί μία επανάληψη του βρόχου, ο συνολικός αριθμός των απαιτούμενων νημάτων είναι 1024. Εφόσον κάθε warp περιέχει 32 νήματα, ο αριθμός των απαιτούμενων warps είναι .

#### Ερώτημα Β.2

Επομένως, συνολικά, για την εκτέλεση «εκτός» και «εντός» του if έχουμε:

* Average Utilization:
* Outside Utilization:
  + **Εκτός** του if-block χρησιμοποιούνται 32 από τα 32 νήματα, άρα 1.
* Inside Utilization:
  + **Εντός** του if-block χρησιμοποιούνται 4 από τα 32 νήματα, άρα .
* Outside Instructions:
  + **Εκτός** του if-block, έχουμε μόνο 3 εντολές (εντολές 1, 2, k+3).
* Inside Instructions:
  + **Εντός** του if-block, έχουμε εντολές.
* Total Instructions:
  + Συνολικά έχουμε k+3 εντολές.

Επομένως:

#### Ερώτημα Β.3

## Σχόλιο Θεωρίας για το Ερώτημα 1.Α.1

* **Branch History Register (BHR)**: αποθηκεύει τα αποτελέσματα των x-bit πιο πρόσφατων διακλαδώσεων (Taken: 1, Not Taken: 0). Αφού είναι 2-bit, ο BHR αποθηκεύει τις δύο τελευταίες. Για παράδειγμα, εάν η τελευταία ήταν Taken και η προτελευταία ήταν Not Taken, ο BHR θα ήταν 01.
* **Saturating counter**: αυτός που χρησιμοποιεί στην πραγματικότητα ο επεξεργαστής για να προβλέψει το αποτέλεσμα της επόμενης διακλάδωσης. Κάθε πιθανή τιμή BHR αντιστοιχεί σε μία τιμή του counter. Αφού είναι 2-bit counter, παίρνει τις εξής τιμές:
  + 00: Strongly NT
  + 01: Weakly NT
  + 10: Weakly T
  + 11: Strongly T

Ο μετρητής ενημερώνεται ανάλογα με το αν η πρόβλεψη της διακλάδωσης ήταν σωστή. Μετά από κάθε εκτέλεση διακλάδωσης, είτε αυξάνεται είτε μειώνεται για να κινηθεί προς μια πιο σίγουρη πρόβλεψη. Μπορούμε να τους απεικονίσουμε σε έναν πίνακα όπως ο παρακάτω (όπου η γραμμή counter έχει διάφορες τιμές από 00 έως 11):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BHR | 00 | 01 | 10 | 11 |
| Counter | 01 | 11 | 01 | 00 |

Με κάθε διακλάδωση, το BHR και ο μετρητής ίσως πρέπει να αλλάξουν:

* Για το BHR, το παλιό LSB του γίνεται το νέο MSB του. Το νέο LSB του είναι 1 εάν το branch είναι πράγματι Τ ή 0 εάν το branch είναι πράγματι ΝΤ. Π.χ. εάν είχαμε BHR=10 και το branch έγινε ΝΤ, τότε το νέο BHR=00.
* Για το counter, ο επεξεργαστής κοιτάζει τον BHR για να βρει την αντίστοιχη τιμή του μετρητή. Π.χ., εάν το BHR είναι 00, ο επεξεργαστής ελέγχει την αντίστοιχη τιμή του μετρητή (εδώ είναι 01, που σημαίνει Weakly ΝΤ, άρα θα προβλέψουμε NT).

Αναλόγως του πραγματικού αποτελέσματος, ο μετρητής μετακινείται προς το άκρο του αποτελέσματος:

|  |  |  |  |
| --- | --- | --- | --- |
| NT | Weakly NT | Weakly T | T |
| 00 | 01 | 10 | 11 |

Εάν το αποτέλεσμα είναι Τ, ο μετρητής «κινείται» δεξιά στον παραπάνω πίνακα (εάν έχει φτάσει στο 11, μένει εκεί, δεν πάει στο 00 κυκλικά). Αντίστοιχα, εάν το αποτέλεσμα είναι ΝΤ, ο μετρητής «κινείται» αριστερά στον παραπάνω πίνακα.

Άρα, όλες οι πιθανές τιμές του μετρητή αναλόγως το αποτέλεσμα είναι:

|  |  |  |
| --- | --- | --- |
| Counter | Result | New Counter |
| 00 | T | 01 |
| 00 | NT | 00 |
| 01 | T | 10 |
| 01 | NT | 00 |
| 10 | T | 11 |
| 10 | NT | 01 |
| 11 | T | 11 |
| 11 | NT | 10 |

Για παράδειγμα, έστω πως βρισκόμαστε σε αυτή την κατάσταση (με κίτρινο σημειώνω σε ποιο BHR είμαστε ενώ με κόκκινο σημειώνω τις αλλαγές του counter):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BHR | 00 | 01 | 10 | 11 |
| Counter | 01 | 11 | 01 | 00 |

Φτάνουμε σε μία διακλάδωση, με BHR=01. Ο μετρητής μας λέει 11, δηλαδή Τ. Εάν το αποτέλεσμα είναι ΝΤ, τότε το BHR θα γίνει 10, ενώ ο counter θα γίνει 10:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BHR | 00 | 01 | 10 | 11 |
| Counter | 01 | **10** | 01 | 00 |

Φτάνουμε σε μία διακλάδωση. Λόγω του προηγούμενου, το BHR=10. Ο μετρητής μας λέει 01, δηλαδή ΝΤ. Εάν το αποτέλεσμα είναι ΝΤ, τότε το BHR θα γίνει 00, ενώ ο counter θα γίνει 00:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BHR | 00 | 01 | 10 | 11 |
| Counter | 01 | 10 | **00** | 00 |

## Θέμα 2

### Ερώτημα Α

Έστω μια out-of-order αρχιτεκτονική η οποία υλοποιεί τον αλγόριθμο Tomasulo και περιλαμβάνει ένα functional unit για πρόσθεση και ένα για πολλαπλασιασμό. Μετά τη δρομολόγηση ***πέντε*** εντολών, η εικόνα των δομών του επεξεργαστή είναι η παρακάτω (οι παύλες στα RS δείχνουν ότι το πεδίο έχει καθαριστεί από προηγούμενες τιμές):

**Reservation Stations**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Op | Busy | Qj | Qk | Vj | Vk |
| ADD1 | - | 0 | - | - | - | - |
| ADD2 | add | 1 |  |  | 28 | 1 |
| MUL1 | mul | 1 |  |  | 40 | 1 |
| MUL2 | mul | 1 | MUL1 |  |  | 12 |
| MUL3 | mul | 1 |  |  | 23 | 40 |

**Register Result Status**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 | R8 | R9 |
| - | - | - | - | - | MUL1 | - | MUL2 | ADD2 | MUL3 |

**Register File** (μετά τις 5 εντολές)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 | R8 | R9 |
| 127 | 28 | 1 | 40 | 30 | 25 | 23 | 90 | 12 | 17 |

**Register File** (αρχική κατάσταση)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 | R8 | R9 |
| 127 | 28 | 1 | 3 | 30 | 25 | 23 | 90 | 12 | 17 |

Υποθέτοντας τις αρχικές τιμές των καταχωρητών που φαίνονται στο πιο πάνω πίνακα, δώστε ένα πρόγραμμα, η εκτέλεση του οποίου έχει οδηγήσει τον επεξεργαστή σε αυτήν την κατάσταση.

#### Λύση

Από τον πίνακα Reservation Stations, οι εντολές έχουν την εξής μορφή:

|  |  |  |  |
| --- | --- | --- | --- |
| Εντολή | 1ο Όρισμα | 2ο Όρισμα | 3ο Όρισμα |
| ADD1 |  |  |  |
| ADD2 |  | 28 | 1 |
| MUL1 |  | 40 | 1 |
| MUL2 |  | MUL1 | 12 |
| MUL3 |  | 23 | 40 |

Από τον 1ο πίνακα Register File μπορούμε να αντικαταστήσουμε τις τιμές με τους registers:

|  |  |  |  |
| --- | --- | --- | --- |
| Εντολή | 1ο Όρισμα | 2ο Όρισμα | 3ο Όρισμα |
| ADD1 |  |  |  |
| ADD2 |  | R1 | R2 |
| MUL1 |  | R3 | R2 |
| MUL2 |  | MUL1 | R8 |
| MUL3 |  | R6 | R3 |

Από τον πίνακα Register Result Status μπορούμε να συμπληρώσουμε το 1ο όρισμα κάποιων εντολών:

|  |  |  |  |
| --- | --- | --- | --- |
| Εντολή | 1ο Όρισμα | 2ο Όρισμα | 3ο Όρισμα |
| ADD1 |  |  |  |
| ADD2 | R8 | R1 | R2 |
| MUL1 | R5 | R3 | R2 |
| MUL2 | R7 | MUL1 | R8 |
| MUL3 | R9 | R6 | R3 |

Έτσι, μπορούμε να αντικαταστήσουμε τα ονόματα των εντολών στην στήλη 2ο όρισμα:

|  |  |  |  |
| --- | --- | --- | --- |
| Εντολή | 1ο Όρισμα | 2ο Όρισμα | 3ο Όρισμα |
| ADD1 |  |  |  |
| ADD2 | R8 | R1 | R2 |
| MUL1 | R5 | R3 | R2 |
| MUL2 | R7 | R5 (MUL1) | R8 |
| MUL3 | R9 | R6 | R3 |

Όσον αφορά την εντολή ADD1 και τον R3, που είναι ο μόνος που αλλάζει τιμή μετά τις 5 εντολές. Παρατηρούμε από τον Reservation Station πίνακα πως όλες οι εντολές είναι busy εκτός της ADD1. Αυτό σημαίνει πως μόνο το 1ο όρισμα της εντολής ADD1 είναι πιθανό να αλλάξει τιμή. Επομένως, το 1ο όρισμα της ADD1 είναι ο R3.

Για να έχουμε άθροισμα 40 στον R3, μπορούμε να προσθέσουμε τους R6,R9 ή τους R1,R8. Επιλέγουμε τους R6,R9. Άρα:

|  |  |  |  |
| --- | --- | --- | --- |
| Εντολή | 1ο Όρισμα | 2ο Όρισμα | 3ο Όρισμα |
| ADD1 | R3 | R6 | R9 |
| ADD2 | R8 | R1 | R2 |
| MUL1 | R5 | R3 | R2 |
| MUL2 | R7 | R5 (MUL1) | R8 |
| MUL3 | R9 | R6 | R3 |

Μένει να καθορίσουμε την σειρά στην οποία πρέπει να μπουν οι εντολές.

* Η μόνη εξάρτηση που έχουμε μεταξύ των εντολών είναι η MUL2 να είναι μετά της MUL1.

Μία σειρά που ικανοποιεί αυτές τις προϋποθέσεις είναι:

ADD1 R3,R1,R8

ADD2 R8,R1,R2

MUL1 R5,R3,R2

MUL2 R7,R5,R8

MUL3 R9,R6,R3

***Άλλη σειρά***:

ADD1 R3,R1,R8

MUL1 R5,R3,R2

MUL3 R9,R6,R3

MUL2 R7,R5,R8

ADD2 R8,R1,R2

### Ερώτημα Β

Δίνεται αρχιτεκτονική η οποία υλοποιεί τον αλγόριθμο Tomasulo χρησιμοποιώντας ROB για in-order commit εντολών. Το pipeline του επεξεργαστή περιέχει τα στάδια Issue (IS), Execute (EX), Write Result (WR) και Commit (CMT), αγνοούμε δηλαδή τα IF και ID. Ισχύουν επίσης τα ακόλουθα:

* Τα IS, WR, CMT απαιτούν 1 κύκλο.
* Το σύστημα περιέχει περιορισμένο αριθμό από reservation stations (RS). Συγκεκριμένα, περιέχει 1 RS για προσθέσεις/αφαιρέσεις και 1 RS για πολλαπλασιασμούς/διαιρέσεις floating point αριθμών. Αντίστοιχα, για integer αριθμούς περιλαμβάνονται 3 RS για εντολές διακλάδωσης, αριθμητικές και λογικές εντολές καθώς και 1 RS για πολλαπλασιασμούς/διαιρέσεις.
* Το σύστημα περιλαμβάνει 1 functional unit για πράξεις integer αριθμών. Όλες οι εντολές μεταξύ integer αριθμών διαρκούν 1 κύκλο.
* Το σύστημα περιλαμβάνει 2 non-pipelined floating point functional units, 1 για ADDD/SUBD και 1 για MULD/DIVD. Οι εντολές πρόσθεσης/αφαίρεσης διαρκούν 3 κύκλους, ενώ οι εντολές πολλαπλασιασμού/διαίρεσης 7 κύκλους.
* Για τις εντολές αναφοράς στη μνήμη, στο στάδιο EX γίνεται τόσο ο υπολογισμός της διεύθυνσης αναφοράς όσο και η προσπέλαση στη μνήμη. Το σύστημα περιλαμβάνει ένα Load και ένα Store Queue, καθένα από τα οποία διαθέτει 2 θέσεις. Οι εντολές χρησιμοποιούν ένα ξεχωριστό pipelined functional unit για τον υπολογισμό της διεύθυνσης και διαρκούν 2 κύκλους στην περίπτωση Hit στην cache και 4 κύκλους σε περίπτωση Miss.
* Οι εντολές διακλάδωσης υπό συνθήκη χρησιμοποιούν τα κατάλληλα RS και FU για αφαίρεση, προκειμένου να υπολογίσουν αν ισχύει η συνθήκη. Η πρόβλεψη για μια εντολή διακλάδωσης υπό συνθήκη γίνεται ταυτόχρονα με τη δρομολόγηση της εντολής. Ο έλεγχος της πρόβλεψης γίνεται αμέσως μόλις γίνει γνωστό το αποτέλεσμα της εντολής, δηλαδή στο στάδιο WR (κύκλος k). Σε περίπτωση σφάλματος, σταματά η εκτέλεση των εντολών του miss-predicted execution path και στον επόμενο κύκλο (κύκλος k+1) δρομολογείται η σωστή εντολή.
* Ο ROB έχει 7 θέσεις.
* Το σύστημα περιλαμβάνει 1 CDB. Σε περίπτωση που παραπάνω από μια εντολές θέλουν να το χρησιμοποιήσουν, τότε προτεραιότητα αποκτά η "παλαιότερη" εντολή (αυτή που έγινε issued πρώτη). Θεωρείστε ότι τα branches δεν χρησιμοποιούν το CDB κατά τη διάρκεια του WR σταδίου.
* Για τις εντολές διακλάδωσης υπό συνθήκη, το σύστημα χρησιμοποιεί έναν (2, 2) global history predictor με συνολικά 8 entries, τα οποία φαίνονται στον παρακάτω πίνακα. Ο BHR έχει τιμή ίση με 0. Η δεικτοδότηση του πίνακα γίνεται χρησιμοποιώντας τον BHR καθώς και τον κατάλληλο αριθμό low order bits από το PC της εντολής. Θεωρείστε πως το αποτέλεσμα του πιο πρόσφατου άλματος βρίσκεται στο LSB του BHR.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 |
| 0 | 11 | 10 | 01 | 00 |
| 1 | 01 | 11 | 00 | 10 |

* Το σύστημα περιλαμβάνει μια fully associative cache μεγέθους 512B με 16 cache blocks και πολιτική αντικατάστασης LRU. Αρχικά η cache είναι άδεια.
* Ο καταχωρητής R1 περιέχει τη διεύθυνση του πρώτου στοιχείου του πίνακα Α, στον οποίο έχουν αποθηκευτεί αριθμοί διπλής ακρίβειας (μήκους 8 bytes ο καθένας). Ο πίνακας είναι ευθυγραμμισμένος.

Δίνεται ο παρακάτω κώδικας:

0x00448404 LOOP: LD F1, 0(R1)

0x00448408 SUBD F2, F2, F1

0x0044840C DIVD F4, F2, F1

0x00448410 SUBI R2, R2, 0x1

0x00448414 XORI R4, R2, 0x1

0x00448418 ADDI R1, R1, 0x10

0x0044841C BNEZ R4, L1

0x00448420 SUBI R6, R6, 0x1

0x00448424 BNEZ R6, LOOP

0x00448428 L1: LD F1, 0(R1)

0x0044842C DIVD F4, F4, F1

0x00448430 ADDI R1, R1, 0x10

0x00448434 SUBI R6, R6, 0x1

0x00448438 BNEZ R6, LOOP

0x0044843C ADDI R1, R1, 0x8

0x00448440 SD F4, 0(R1)

Δίνονται οι αρχικές τιμές . Εκτελέστε τον παραπάνω κώδικα και δώστε τους χρόνους δρομολόγησης, εκτέλεσης και ολοκλήρωσης των εντολών σε έναν πίνακα όπως το παρακάτω. Ποια τα τελικά περιεχόμενα της cache;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| OP | IS | EX | WR | CMT | Comments |
| LD F1, 0(R1) | 1 | 2-?? | ?? | ?? |  |

Στο πεδίο "Σχόλιο" δικαιολογήστε τυχόν καθυστερήσεις μεταξύ IS-EX, EX-WR και WR-CMT καθώς και ακυρώσεις εντολών.

#### Λύση

|  |  |  |  |
| --- | --- | --- | --- |
|  | RS | FU | Cycles |
| ADDD/SUBD | 1 | 1 | 3 |
| MULD/DIVD | 1 | 1 | 7 |
| BR/ADD/SUB/AND | 3 | 1 | 1 |
| MUL/DIV | 1 | 1 |
| LOAD | 2 |  | H2/M4 |
| STORE | 2 |  | H2/M4 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 | 01 | 10 | 11 |
| 0 | 11 | ~~10~~ 01 | 01 | 00 |
| 1 | ~~01~~ 10 | 11 | 00 | 10 |

Μνήμη μεγέθους 512Β, 16 blocks 32B block size. Κάθε μπλοκ «χωράει» 32/8=4 στοιχεία του Α.

|  |  |
| --- | --- |
| 0 | A[0-3] |
| 1 | A[4-7] |
| … |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| OP | IS | EX | WR | CMT | Comments |
|  |  |  |  |  | ROB=7, BHR=00, LRU=0  R1=A[0], R2=1, R6=1 |
| LD F1,0(R1) | 1 | 2-5 | 6 | 7 | Miss, LRU=1 |
| SUBD F2,F2,F1 | 2 | 7-9 | 10 | 11 | RAW F1 |
| DIVD F4,F2,F1 | 3 | 11-17 | 18 | 19 | RAW F1,F2 |
| SUBI R2,R2,0x1 | 4 | 5 | 7 | 20 | R2=0, CDB Conflict |
| XORI R4,R2,0x1 | 5 | 8 | 9 | 21 | RAW R2, R4=1 |
| ADDI R1,R1,0x10 | 6 | 7 | 8 | 22 | R1=A[2] |
| BNEZ R4,L1 | 7 | 10 | 11 | 23 | RAW R4  Pred=NT, Res=T, BHR=01 |
|  |  |  |  |  |  |
| **X** SUBI R6,R6,0x1 | 10 | X | X | X | FLUSH\*\* |
| **X** BNEZ R6,LOOP | X | X | X | X |  |
|  |  |  |  |  |  |
| LD F1,0(R1) | 12 | 13-14 | 15 | 24 | RAW R1, Hit |
| DIVD F4,F4,F1 | 19 | 20-26 | 27 | 28 | RAW F1,F4, RS FULL |
| ADDI R1,R1,0x10 | 20 | 21 | 22 | 29 | R1=A[4] |
| SUBI R6,R6,0x1 | 21 | 22 | 23 | 30 | R6=0 |
| BNEZ R6,LOOP | 22 | 23 | 24 | 31 | RAW R6  Pred=T, Res=ΝΤ, BHR=10 |
|  |  |  |  |  |  |
| **X** LD F1,0(R1) | 23 | 24-X | X | X | Miss, LRU=2 |
| **X** SUBD F2,F2,F1 | X | X | X | X | FLUSH |
|  |  |  |  |  |  |
| ADDI R1,R1,0x8 | 25 | 26 | 27 | 32 | R1=A[5] |
| SD F4,0(R1) | 26 | 27-28 | 29 | 33 | RAW R1, Hit |

\*\*Θεωρώ πως το R6 δεν πρόλαβε να μειωθεί

## Θέμα 3

Δίνεται σύστημα παράλληλης επεξεργασίας 2 επεξεργαστών κοινής μνήμης, το οποίο υλοποιεί ακολουθιακή συνέπεια (Sequential Consistency). Οι επεξεργαστές εκτελούν τις παρακάτω εντολές:

|  |  |
| --- | --- |
| P0 | P1 |
|  |  |
|  |  |
|  |  |
|  |  |

Όλες οι μεταβλητές βρίσκονται στην κύρια μνήμη και έχουν αρχική τιμή 0 και κάθε ανάγνωση ή εγγραφή οποιασδήποτε μεταβλητής υλοποιείται μέσω μιας αντίστοιχης πρόσβασης στη μνήμη.

1. Εξηγήστε ποιες οι πιθανές τιμές της μεταβλητής y μετά το τέλος της εκτέλεσης του κώδικα και των 2 επεξεργαστών.
2. Εξηγήστε ποιες οι τελικές τιμές της μεταβλητής b μετά το τέλος της εκτέλεσης του κώδικα και των 2 επεξεργαστών.
3. Αν το σύστημα υλοποιούσε το πλέον relaxed memory model (π.χ. RMO) θα άλλαζε η απάντησή σας στο ερώτημα i); Αν ναι, εξηγήστε ποιες διαφορετικές τιμές θα μπορούσε να έχει η μεταβλητή y στο τέλος της εκτέλεσης.

### Ερώτημα Α

## Θέμα 4

Θεωρήστε ένα σύστημα παράλληλης επεξεργασίας 2 επεξεργαστών κοινής μνήμης, το οποίο χρησιμοποιεί το πρωτόκολλο συνοχής MESI επιτρέποντας cache-to-cache transfers και υλοποιεί το μοντέλο της ακολουθιακής συνέπειας. Κάθε επεξεργαστής διαθέτει μία μόνο L1 data cache μεγέθους 128B, direct-mapped, write-back αποτελούμενη από 4 blocks. Αρχικά όλα τα blocks σε όλες τις caches είναι σε κατάσταση Ι. Στους επεξεργαστές εκτελείται παράλληλα το εξής C πρόγραμμα:

A math equations on a white background

Description automatically generated

Ο πίνακας x είναι ευθυγραμμισμένος και το μέγεθος ενός ακεραίου είναι 4 bytes. Το πρόγραμμα εκτελείται με τέτοιο τρόπο ώστε κάθε λειτουργία μνήμης (read ή write) να εναλλάσσεται ανάμεσα στους επεξεργαστές με round-robin τρόπο (κυκλικά). π.χ. P1:read x[0].a, P2:write x[0].c, P1:write x[0].a, P2:write x[0].a, κ.ο.κ. Η εκτέλεση του προγράμματος είναι τέτοια ώστε να μην υπάρχουν επικαλύψεις (overlaps) μεταξύ των λειτουργιών, δηλ. πρώτα ολοκληρώνεται η εκτέλεση μιας λειτουργίας του ενός επεξεργαστή (καθώς και ό,τι αυτή απαιτεί) και μετά ακολουθεί η λειτουργία μνήμης του άλλου επεξεργαστή. Τέλος, ισχύουν οι ακόλουθοι χρόνοι για τις διάφορες λειτουργίες της μνήμης και του πρωτοκόλλου:

|  |  |
| --- | --- |
| Λειτουργία | Κύκλοι |
| Cache hit | 1 |
| BusRdX/BusRd (περιλαμβάνει το cache access) | 2 |
| Μεταφορά 1 cache line (προς cache ή προς κύρια μνήμη) | 8 |

1. Ποιος είναι ο συνολικός χρόνος εκτέλεσης του προγράμματος; Εξηγήστε αναλυτικά την απάντησή σας παρουσιάζοντας την κατάσταση των caches μετά από την εκτέλεση κάθε αναφοράς. Συγκεκριμένα, για όλες τις caches, δείξτε την κατάσταση MESI της cache line που εμπλέκεται σε κάθε αναφορά καθώς και το χρόνο που χρειάζεται για την εκτέλεση της αναφοράς, συμπληρώνοντας έναν πίνακα όπως ο παρακάτω:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Iteration | Operation | Cache States | | Op. Latency | Iteration Cycles |
| P1 | P2 |
| i=0 | P1: read x0.c |  |  |  |  |
| P2: read x0.a |  |  |  |
| P1: write x0.c |  |  |  |
| … |  |  |  |

1. Τι αλλαγές θα κάνατε στον κώδικα ή στη δομή προκειμένου να βελτιστοποιήσετε τον χρόνο εκτέλεσης; Δικαιολογήστε την απάντησή σας και υπολογίστε και πάλι τον νέο αριθμό των κύκλων.

### Ερώτημα Α

Μνήμη 128Β με 4 blocksblock size 32B. Ο int έχει μέγεθος 4Β8 ints/block

|  |  |  |  |
| --- | --- | --- | --- |
| x[0,1] | x[2,3] | x[4,5] | X[6,7] |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Iteration | Operation | Cache States | | Op. Latency | Iteration Cycles |
| P1 | P2 |
|  |  | IIII | IIII |  |  |
| I=0 | read x0.c | EIII | IIII | 8+2 |  |
|  | read x0.a | SIII | SIII | 8+2 |  |
|  | write x0.c | MIII | IIII | 2 |  |
|  | write x0.a | IIII | MIII | 8+2 |  |
|  | read x0.d | SIII | SIII | 8+2 |  |
|  | read x0.b | SIII | SIII | 1 |  |
|  | write x0.d | MIII | IIII | 2 |  |
|  | write x0.b | IIII | MIII | 8+2 | 55 |

Για i=1, είναι η ίδια διαδικασία.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Iteration | Operation | Cache States | | Op. Latency | Iteration Cycles |
| P1 | P2 |
|  |  | IIII | MIII |  |  |
| I=2 | read x0.c | SIII | SIII | 8+2 |  |
|  | read x2.a | SIII | SEII | 8+2 |  |
|  | write x0.c | MIII | IEII | 2 |  |
|  | write x2.a | MIII | IMII | 1 |  |
|  | read x0.d | MIII | IMII | 1 |  |
|  | read x2.b | MIII | IMII | 1 |  |
|  | write x0.d | MIII | IMII | 1 |  |
|  | write x2.b | MIII | IMII | 1 | 27 |
|  |  |  |  |  |  |
| I=3 | read x1.c | MIII | IMII | 1 |  |
|  | read x3.a | MIII | IMII | 8+2 |  |
|  | write x1.c | ΜΙΙΙ | ΙΜΙΙ | 1 |  |
|  | write x3.a | ΜΙΙΙ | ΙΜΙΙ | 1 |  |
|  | read x1.d | ΜΙΙΙ | ΙΜΙΙ | 1 |  |
|  | read x3.b | ΜΙΙΙ | ΙΜΙΙ | 1 |  |
|  | write x1.d | ΜΙΙΙ | ΙΜΙΙ | 1 |  |
|  | write x3.b | ΜΙΙΙ | ΙΜΙΙ | 1 | 17 |

Για i=4,5 έχουμε την ίδια διαδικασία με τις i=0,1.

Για i=6,7 έχουμε την ίδια διαδικασία με τις i=2,3.

Για i=8-15 έχουμ την ίδια διαδικασία με τις i=0-7.

Άρα συνολικοί κύκλοι είναι:

### Ερώτημα Β

Θα βάζουμε στην cache τα εξής:

|  |  |  |  |
| --- | --- | --- | --- |
| x[0-7].c | x[0-7].a | x[0-3].d | x[0-3].b |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Iteration | Operation | Cache States | | Op. Latency | Iteration Cycles |
| P1 | P2 |
|  |  | IIII | IIII |  |  |
| I=0 | read x0.c | EIII | IIII | 8+2 |  |
|  | read x0.a | EIII | IEII | 8+2 |  |
|  | write x0.c | MIII | IEII | 1 |  |
|  | write x0.a | MIII | IMII | 1 |  |
|  | read x0.d | MIEI | IMII | 8+2 |  |
|  | read x0.b | MIEI | IMIE | 8+2 |  |
|  | write x0.d | MIMI | IMIE | 1 |  |
|  | write x0.b | MIMI | IMIM | 1 | 44 |

Για i=1-3 έχουμε την ίδια διαδικασία.

# Κανονική 16

Έστω μια out-of-order αρχιτεκτονική η οποία υλοποιεί τον αλγόριθμο Tomasulo και περιλαμβάνει ένα functional unit για πρόσθεση και ένα για πολλαπλασιασμό. Μία συγκεκριμένη στιγμή της εκτέλεσης ενός προγράμματος, η εικόνα των δομών του επεξεργαστή είναι η παρακάτω:

**Reservation Stations**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Op | Busy | Qj | Qk | Vj | Vk |
| ADD1 | add | 0 |  |  | **15** | **5** |
| ADD2 | add | 1 | MUL1 |  |  | 3 |
| ADD3 | add | 1 | MUL2 | ADD2 |  |  |
| MUL1 | mul | 1 |  |  | 20 | 20 |
| MUL2 | mul | 1 | ADD2 | ADD2 |  |  |
| MUL3 | mul | 1 | MUL2 |  |  | 5 |

**Register Result Status**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 |
| 0 | ADD2 | 0 | 0 | MUL2 | ADD3 | 0 | 0 |

**Register File**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 |
| 18 | 15 | 5 | 20 | 10 | 25 | 3 | 9 |

Δώστε ένα πρόγραμμα, η εκτέλεση του οποίου έχει οδηγήσει τον επεξεργαστή σε αυτήν την κατάσταση. Δίνεται ότι η πρώτη εντολή του προγράμματος έχει ολοκληρωθεί και είναι ADD με destination register τον R3.

# Δική μου Λύση

Από τον πίνακα Reservation Stations, οι εντολές έχουν την εξής μορφή:

|  |  |  |  |
| --- | --- | --- | --- |
| Εντολή | 1ο Όρισμα | 2ο Όρισμα | 3ο Όρισμα |
| ADD1 |  | 15 | 5 |
| ADD2 |  | MUL1 | 3 |
| ADD3 |  | MUL2 | ADD2 |
| MUL1 |  | 20 | 20 |
| MUL2 |  | ADD2 | ADD2 |
| MUL3 |  | MUL2 | 5 |

Από τον πίνακα Register File μπορούμε να αντικαταστήσουμε τις τιμές με τους registers:

|  |  |  |  |
| --- | --- | --- | --- |
| Εντολή | 1ο Όρισμα | 2ο Όρισμα | 3ο Όρισμα |
| ADD1 |  | R1 | R2 |
| ADD2 |  | MUL1 | R6 |
| ADD3 |  | MUL2 | ADD2 |
| MUL1 |  | R3 | R3 |
| MUL2 |  | ADD2 | ADD2 |
| MUL3 |  | MUL2 | R2 |

Από τον πίνακα Register Result Status μπορούμε να συμπληρώσουμε το 1ο όρισμα κάποιων εντολών:

|  |  |  |  |
| --- | --- | --- | --- |
| Εντολή | 1ο Όρισμα | 2ο Όρισμα | 3ο Όρισμα |
| ADD1 |  | R1 | R2 |
| ADD2 | R1 | MUL1 | R6 |
| ADD3 | R5 | MUL2 | ADD2 |
| MUL1 |  | R3 | R3 |
| MUL2 | R4 | ADD2 | ADD2 |
| MUL3 |  | MUL2 | R2 |

Έτσι, μπορούμε να αντικαταστήσουμε τα ονόματα των εντολών στις στήλες 2ο και 3ο όρισμα:

|  |  |  |  |
| --- | --- | --- | --- |
| Εντολή | 1ο Όρισμα | 2ο Όρισμα | 3ο Όρισμα |
| ADD1 |  | R1 | R2 |
| ADD2 | R1 | MUL1 | R6 |
| ADD3 | R5 | R4 | R1 |
| MUL1 |  | R3 | R3 |
| MUL2 | R4 | R1 | R1 |
| MUL3 |  | R4 | R2 |

**Σημείωση**: ακόμα και αν το αποτέλεσμα μιας πράξης αποθηκεύεται π.χ. στον R1, το περιεχόμενο αυτού στον RF δεν αλλάζει, παραμένει 15 και με βάση αυτό γίνονται οι πράξεις.

**1η εντολή**

Μας δίνεται από την εκφώνηση «ότι η πρώτη εντολή του προγράμματος έχει ολοκληρωθεί και είναι ADD με destination register τον R3». Από αυτό, μπορούμε να συμπεράνουμε πως η ADD1 είναι η 1η εντολή και πως το 1ο όρισμα της είναι ο R3. Μάλιστα, εάν δούμε τις τιμές των R1, R2, R3 στο Register File θα δούμε πως . Άρα:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Εντολή | 1ο Όρισμα | 2ο Όρισμα | 3ο Όρισμα | Σειρά |
| ADD1 | R3 | R1 | R2 | 1η |
| ADD2 | R1 | MUL1 | R6 |  |
| ADD3 | R5 | R4 | R1 |  |
| MUL1 |  | R3 | R3 |  |
| MUL2 | R4 | R1 | R1 |  |
| MUL3 |  | R4 | R2 |  |

**2η εντολή**

Παρατηρούμε πως η επόμενη εντολή που δεν χρειάζεται να περιμένει κάποια άλλη είναι η MUL1. Άρα:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Εντολή | 1ο Όρισμα | 2ο Όρισμα | 3ο Όρισμα | Σειρά |
| ADD1 | R3 | R1 | R2 | 1η |
| ADD2 | R1 | MUL1 | R6 |  |
| ADD3 | R5 | R4 | R1 |  |
| MUL1 |  | R3 | R3 | 2η |
| MUL2 | R4 | R1 | R1 |  |
| MUL3 |  | R4 | R2 |  |

Η MUL1 δεν έχει κάποιο αυστηρό 1ο όρισμα. Μπορούμε λοιπόν να επιλέξουμε από τους Registers του RRS που **δεν** έχουν τιμή 0 (δηλαδή R1, R4, R5). Έστω πως επιλέγουμε τον R1.

**3η εντολή**

Παρατηρούμε πως με την MUL1 υπολογισμένη, η επόμενη εντολή που μπορεί να εκτελεστεί είναι η ADD2:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Εντολή | 1ο Όρισμα | 2ο Όρισμα | 3ο Όρισμα | Σειρά |
| ADD1 | R3 | R1 | R2 | 1η |
| ADD2 | R1 | R1 | R6 | 3η |
| ADD3 | R5 | R4 | R1 |  |
| MUL1 | R1 | R3 | R3 | 2η |
| MUL2 | R4 | R1 | R1 |  |
| MUL3 |  | R4 | R2 |  |

**4η εντολή**

Παρατηρούμε πως με την ADD2 υπολογισμένη, η επόμενη εντολή που μπορεί να εκτελεστεί είναι η MUL2:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Εντολή | 1ο Όρισμα | 2ο Όρισμα | 3ο Όρισμα | Σειρά |
| ADD1 | R3 | R1 | R2 | 1η |
| ADD2 | R1 | R1 | R6 | 3η |
| ADD3 | R5 | R4 | R1 |  |
| MUL1 | R1 | R3 | R3 | 2η |
| MUL2 | R4 | R1 | R1 | 4η |
| MUL3 |  | R4 | R2 |  |

**5η εντολή**

Παρατηρούμε πως με την MUL2 υπολογισμένη, η επόμενη εντολή που μπορεί να εκτελεστεί είναι είτε η ADD3 είτε η MUL3. Παρατηρούμε πως , επομένως η τελευταία θα είναι η ADD3 και πρώτα θα είναι η MUL3:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Εντολή | 1ο Όρισμα | 2ο Όρισμα | 3ο Όρισμα | Σειρά |
| ADD1 | R3 | R1 | R2 | 1η |
| ADD2 | R1 | R1 | R6 | 3η |
| ADD3 | R5 | R4 | R1 | 6η |
| MUL1 | R1 | R3 | R3 | 2η |
| MUL2 | R4 | R1 | R1 | 4η |
| MUL3 | R5 | R4 | R2 | 5η |

Το πρόγραμμα λοιπόν έχει ως:

ADD1 R3,R1,R2

MUL1 R1, R3, R3

ADD2 R1, R1, R6

MUL2 R4, R1, R1

MUL3 R5, R4, R2

ADD3 R5, R4, R1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Εντολή | 1ο Όρισμα | 2ο Όρισμα | 3ο Όρισμα | Σειρά |
| ADD1 | **R3** | **R1** | R2 | 5 |
| ADD2 | **R1** | **R1** | R6 | 6 |
| ADD3 | R5 | **R4** | **R1** | 1 |
| MUL1 | **R1** | **R3** | **R3** | 4 |
| MUL2 | **R4** | **R1** | **R1** | 3 |
| MUL3 | R5 | **R4** | R2 | 2 |

# Κανονική 23

Θέμα 1

Ερώτημα Α

Απαντήστε συνοπτικά στις παρακάτω ερωτήσεις:

1. Τι ονομάζουμε register renaming και με ποιο τρόπο είναι βοηθητικό;
2. Πως αντιμετωπίζει ο αλγόριθμος Tomasulo τους RAW hazards;
3. Για ποιο λόγο χρειάζονται μηχανισμούς πρόβλεψης εντολών άλματος οι επεξεργαστές που υλοποιούν Tomasulo;
4. Όταν μια εντολή ολοκληρώνει την εκτέλεση της, το αποτέλεσμα της αποθηκεύεται στο ROB αλλά και στα Value πεδία όσων Reservation Stations χρειάζεται. Για ποιο λόγο την αποθηκεύουμε στο R5 από τη στιγμή που η τιμή είναι διαθέσιμη στο ROB;

**Λύση**

*Υποερώτημα (i)*

**Register Renaming**: τεχνική που χρησιμοποιείται στη σχεδίαση CPU για την εξάλειψη των ψευδών εξαρτήσεων δεδομένων μεταξύ εντολών, συγκεκριμένα αυτών που προκαλούνται από την επαναχρησιμοποίηση των ίδιων καταχωρητών για διαφορετικούς σκοπούς από διαφορετικές εντολές. Αυτές οι ψευδείς εξαρτήσεις είναι γνωστές ως εξαρτήσεις ονόματος και περιλαμβάνουν τύπους όπως οι κίνδυνοι Write-After-Write (WAW) και Write-After-Read (WAR). Χρήση:

* **Εξαλείφει τις ψευδείς εξαρτήσεις**: Με τη δυναμική μετονομασία καταχωρητών, η CPU μπορεί να διακρίνει μεταξύ διαφορετικών περιπτώσεων του ίδιου καταχωρητή, επιτρέποντας στις εντολές να εκτελούνται εκτός σειράς χωρίς περιττές καθυστερήσεις.
* **Αυξάνει τον παραλληλισμό**: Η μετονομασία καταχωρητών επιτρέπει σε περισσότερες εντολές να εκτελούνται ταυτόχρονα, βελτιώνοντας τον παραλληλισμό σε επίπεδο εντολών (ILP).
* **Βελτιώνει την αποδοτικότητα του αγωγού**: Αφαιρώντας τις ψευδείς εξαρτήσεις, η CPU μπορεί να αξιοποιήσει καλύτερα τα στάδια του αγωγού της, μειώνοντας τους χρόνους αδράνειας και αυξάνοντας τη συνολική απόδοση.

*Υποερώτημα (ii)*

Ο **Tomasulo** αντιμετωπίζει τους κινδύνους Read-After-Write (RAW) χρησιμοποιώντας έναν συνδυασμό δυναμικού χρονοπρογραμματισμού και χρήσης σταθμών κράτησης και κοινού διαύλου δεδομένων (CDB):

* **Reservation Stations**: Οι εντολές τοποθετούνται σε σταθμούς κράτησης όπου περιμένουν να γίνουν διαθέσιμοι οι τελεστές τους. Αυτό επιτρέπει στην ΚΜΕ να εκδίδει εντολές εκτός σειράς.
* **Tagging Mechanism**: Κάθε τελεστής συνδέεται με μια ετικέτα που υποδεικνύει την πηγή που θα παράγει την τιμή του. Εάν ο τελεστής δεν είναι ακόμη διαθέσιμος, η εντολή περιμένει στο σταθμό κράτησης.
* **Common Data Bus**: Όταν μια εντολή ολοκληρώνει την εκτέλεσή της, μεταδίδει το αποτέλεσμά της στον CDB μαζί με την αντίστοιχη ετικέτα. Οι σταθμοί κράτησης που αφουγκράζονται αυτή την ετικέτα μπορούν στη συνέχεια να συλλάβουν το αποτέλεσμα.
* **Forwarding**: Με τη χρήση του CDB για τη μετάδοση των αποτελεσμάτων αμέσως μόλις είναι έτοιμα, οι επόμενες εντολές που περιμένουν αυτές τις τιμές μπορούν να προχωρήσουν χωρίς να περιμένουν την εγγραφή του αποτελέσματος πίσω στο αρχείο καταχωρητών, επιλύοντας έτσι τους κινδύνους RAW.

*Υποερώτημα (iii)*

Οι επεξεργαστές που εφαρμόζουν τον αλγόριθμο του Tomasulo χρειάζονται **μηχανισμούς πρόβλεψης εντολών άλματος** επειδή:

* **Διατήρηση της αποδοτικότητας του pipeline**: Οι εντολές διακλάδωσης (άλματος) μπορούν να διαταράξουν σημαντικά τη ροή εκτέλεσης εντολών προκαλώντας την καθυστέρηση του pipeline αν η CPU πρέπει να περιμένει μέχρι να επιλυθεί η συνθήκη διακλάδωσης.
* **Speculative εκτέλεση**: Για να μετριάσει τον αντίκτυπο των διακλαδώσεων, ο επεξεργαστής χρησιμοποιεί την πρόβλεψη διακλαδώσεων για να μαντέψει την πιθανή έκβαση της διακλάδωσης (αν θα πραγματοποιηθεί ή όχι). Αυτό επιτρέπει στην CPU να συνεχίσει να φέρνει και να εκτελεί εντολές κατά μήκος της προβλεπόμενης διαδρομής χωρίς αναμονή.
* **Ελαχιστοποίηση των καθυστερήσεων και των διακλαδώσεων**: Η ακριβής πρόβλεψη διακλάδωσης μειώνει τον αριθμό των διακλαδώσεων και των καθυστερήσεων του αγωγού που συμβαίνουν όταν μια διακλάδωση προβλέπεται εσφαλμένα. Αυτό βελτιώνει τη συνολική απόδοση και την απόδοση των εντολών.

*Υποερώτημα (iv)*

Η αποθήκευση του αποτελέσματος στον **καταχωρητή R5** (ή σε οποιονδήποτε άλλο καταχωρητή καθορίζεται από την εντολή) μόλις η τιμή είναι διαθέσιμη στον Reorder Buffer εξυπηρετεί πολλαπλούς σκοπούς:

* **Update της αρχιτεκτονικής κατάστασης**: Το αποτέλεσμα πρέπει να εγγραφεί στο αρχείο καταχωρητών για να ενημερωθεί η αρχιτεκτονική κατάσταση του επεξεργαστή, διασφαλίζοντας ότι οι επόμενες εντολές που προσπελαύνουν αυτόν τον καταχωρητή λαμβάνουν τη σωστή τιμή.
* **Διατήρηση της σειράς του προγράμματος**: Η εγγραφή του αποτελέσματος στο αρχείο καταχωρητών (R5) με τη σειρά του προγράμματος, όπως τη διαχειρίζεται η ROB, διασφαλίζει τη σωστή εκτέλεση του προγράμματος σύμφωνα με την αρχική ακολουθία των εντολών του.
* **Συνέπεια**: Ο ROB παρέχει έναν μηχανισμό που εξασφαλίζει ότι όλες οι εντολές ολοκληρώνονται με τη σωστή σειρά. Μόλις το αποτέλεσμα δεσμευτεί στο αρχείο καταχωρητών, θεωρείται μέρος της μόνιμης αρχιτεκτονικής κατάστασης της μηχανής, καθιστώντας το διαθέσιμο για όλες τις μελλοντικές εντολές.
* **Ορθότητα**: Με την ενημέρωση του αρχείου καταχωρητών (R5) μόνο αφού η εντολή έχει φτάσει στην κεφαλή του ROB και έχει δεσμευτεί, ο επεξεργαστής διατηρεί ορθή συμπεριφορά ακόμη και με την παρουσία εξαιρέσεων ή διακοπών.

Ερώτημα Β

Αναλαμβάνετε επικεφαλής σχεδίασης ενός καινούργιου out-of-order επεξεργαστή που υλοποιεί Tomasulo χρησιμοποιώντας ROB για in-order commit εντολών. Το προηγούμενο μοντέλο του επεξεργαστή έχει τα εξής χαρακτηριστικά:

* Βαθύ pipeline 25 σταδίων.
* 8 RS και 12 FUs για integer αριθμούς.
* 4 RS kai 6 FUs για floating point αριθμούς.
* 8 θέσεις στον ROB.
* 256KB on chip caches.
* 2-bit saturating counter branch predictor με σχετικά καλή ακρίβεια.

Διαπιστώνετε πως οι εφαρμογές που θα εκτελούνται στον καινούργιο επεξεργαστή έχουν μικρό κώδικα, επεξεργάζονται μικρά datasets της τάξης των 64ΚΒ και περνούν σχεδόν την περισσότερη ώρα μέσα σε loops, οι επαναλήψεις των οποίων είναι ανεξάρτητες μεταξύ τους, ενώ η κάθε επανάληψη εμφανίζει περιορισμένο ILP. Για κάθε μία από τις παρακάτω αλλαγές, εξηγήστε συνοπτικά αν θα την επιλέγατε για να βελτιώσετε την επίδοση του επεξεργαστή σε σχέση με το προηγούμενο μοντέλο.

1. Αντικατάσταση του branch predictor με μηχανισμό υψηλότερης ακρίβειας.
2. Αύξηση του μεγέθους της cache.
3. Προσθήκη περισσότερων FUs.
4. Αύξηση των θέσεων του ROB.

**Λύση**

*Υποερώτημα (i)*

Η εξέταση της αντικατάστασης εξαρτάται από την υπάρχουσα απόδοση. Αν ο τρέχων προβλεπτικός μηχανισμός διακλάδωσης είναι ήδη υψηλής ακρίβειας (όπως προτείνεται), το όφελος από την αντικατάστασή του με έναν μηχανισμό υψηλότερης ακρίβειας μπορεί να είναι οριακό. Ωστόσο, αν υπάρχει σημαντική ευκαιρία για βελτίωση, η αύξηση της ακρίβειας πρόβλεψης διακλαδώσεων μπορεί να μειώσει τις λανθασμένες προβλέψεις, τα ξεπλύματα αγωγών και τα αδιέξοδα, οδηγώντας σε καλύτερη απόδοση, ιδίως σε βρόχους.

Οι εφαρμογές περνούν το μεγαλύτερο μέρος του χρόνου τους σε βρόχους. Η πρόβλεψη διακλαδώσεων υψηλής ακρίβειας είναι ζωτικής σημασίας για την ελαχιστοποίηση των αδιεξόδων αγωγού λόγω λανθασμένων προβλέψεων διακλαδώσεων. Ακόμη και μια μικρή βελτίωση στην ακρίβεια πρόβλεψης μπορεί να οδηγήσει σε σημαντικά κέρδη απόδοσης σε αυτό το πλαίσιο. Παρόλα αυτά, εάν η τρέχουσα πρόβλεψη κορεσμένου μετρητή 2-bit αποδίδει καλά, η επένδυση πόρων σε άλλους τομείς μπορεί να αποφέρει καλύτερες αποδόσεις.

*Υποερώτημα (ii)*

Όχι, δεν είναι απαραίτητο να αυξηθεί το μέγεθος της κρυφής μνήμης.

Οι εφαρμογές επεξεργάζονται μικρά σύνολα δεδομένων (γύρω στα 64KB), τα οποία βρίσκονται ήδη εντός της κρυφής μνήμης 256KB. Επιπλέον, το τρέχον μέγεθος της κρυφής μνήμης θα πρέπει να επαρκεί για την αποτελεσματική επεξεργασία αυτών των μικρών συνόλων δεδομένων, ελαχιστοποιώντας τις αστοχίες της κρυφής μνήμης. Η αύξηση του μεγέθους της κρυφής μνήμης ενδέχεται να μην παρέχει σημαντική βελτίωση των επιδόσεων και θα μπορούσε να οδηγήσει σε περιττή χρήση της περιοχής πυριτίου και κατανάλωση ενέργειας.

*Υποερώτημα (iii)*

Όχι, η προσθήκη περισσότερων FU μπορεί να μην είναι επωφελής.

Οι εφαρμογές παρουσιάζουν περιορισμένο παραλληλισμό σε επίπεδο εντολών (ILP), πράγμα που σημαίνει ότι η προσθήκη περισσότερων λειτουργικών μονάδων (FUs) ενδέχεται να μην αξιοποιηθεί πλήρως. Επιπλέον. η τρέχουσα ρύθμιση με 12 FU ακέραιων αριθμών και 6 FU κινητής υποδιαστολής είναι πιθανώς επαρκής για να χειριστεί το επίπεδο παραλληλισμού που υπάρχει στα συγκεκριμένα φορτία εργασίας. Αντ' αυτού, η βελτιστοποίηση της χρήσης των υφιστάμενων FUs μέσω άλλων μέσων (όπως η βελτίωση του χρονοπρογραμματισμού ή η μείωση της καθυστέρησης) μπορεί να είναι πιο αποτελεσματική.

*Υποερώτημα (iv)*

Ναι, η αύξηση του αριθμού των θέσεων ROB μπορεί να είναι επωφελής.

Ένα μεγαλύτερο ROB επιτρέπει την εκτέλεση περισσότερων εντολών, γεγονός που μπορεί να βοηθήσει στην εκμετάλλευση κάθε διαθέσιμου ILP και στη βελτίωση της συνολικής απόδοσης. Αυτό είναι ιδιαίτερα χρήσιμο για την εκτέλεση εκτός σειράς, καθώς επιτρέπει στον επεξεργαστή να διατηρήσει ένα μεγαλύτερο παράθυρο εκτέλεσης και να αναδιατάξει περισσότερες εντολές, οδηγώντας σε καλύτερη αξιοποίηση του αγωγού και των λειτουργικών μονάδων. Επιπλέον, δεδομένου του βαθύ αγωγού 25 σταδίων, ένα μεγαλύτερο ROB μπορεί επίσης να βοηθήσει στον μετριασμό των επιπτώσεων των αδιεξόδων του αγωγού και να διασφαλίσει ότι περισσότερες εντολές είναι έτοιμες για εκτέλεση.

Ερώτημα Γ

Εργάζεστε στη σχεδίαση του AVP (ACME Vector Processor). Η εφαρμογή ενδιαφέροντος είναι το SAXPY (a\*x[i]+y[i]) για διανύσματα διπλής ακρίβειας (double) μεγάλου μεγέθους. Η καρδιά του διανυσματικού κώδικα είναι η παρακάτω, όπου για λόγους απλότητας αγνοούμε το κόστος περιφερειακού κώδικα (ενημέρωση δεικτών και διακλαδώσεις):

|  |  |
| --- | --- |
| LV V1, Rx | ; load vector X |
| MULVS.D V2, V1 | ; vector-scalar multiply |
| LV V3, Ry | ; load vector Y |
| ADDVV.D V4, V2, V3 | ; add two vectors |
| SV Ry, V4 | ; store the sum |

Η αρχική σχεδίαση έχει τα εξής χαρακτηριστικά:

* Vector length: 128 στοιχεία.
* Μνήμη: 8 banks, καθυστέρηση πρόσβασης 8 κύκλοι, πλάτος 64 bit, με ξεχωριστές πόρτες ανάγνωσης/εγγραφής.
* Αθροιστής: Πλήρως pipelined, καθυστέρηση 2 κύκλοι.
* Πολλαπλασιαστής: Πλήρως pipelined, καθυστέρηση 4 κύκλοι.
* Πλάτος έκδοσης: 1 εντολή ανά κύκλο, υποστηρίζει chaining, 1 lane.

1. Υπολογίστε αναλυτικά τον χρόνο εκτέλεσης για αυτή τη βασική σχεδίαση και για μήκος διανύσματος 128 στοιχεία (ένα iteration του loop). Υπόδειξη: ένα διάγραμμα όπως αυτό του χρονισμού pipeline μπορεί να είναι χρήσιμο – αν δεν είναι απόλυτα απαραίτητο.
2. Προτείνεται η βελτίωση του συστήματος με την προσθήκη ενός ακόμα lane (σύνολο 2 lanes) για υπολογισμούς αλλά και στο σύστημα μνήμης. Υπολογίστε τον νέο χρόνο εκτέλεσης και εκτιμήστε το κόστος και την πολυπλοκότητα υλοποίησης της νέας σχεδίασης. Ποιο κομμάτι της σχεδίασης θα γίνει πιο πολύπλοκο/ακριβό/κλπ. και με ποιο τρόπο;

**Λύση**

*Υποερώτημα (i)*

Ανάλυση λειτουργίας SAXPY:

1. **Load Vector X** (LV V1, Rx):
   * Φορτώνει 128 στοιχεία στο V1.
   * Καθυστέρηση μνήμης: 8 κύκλοι.
   * Μήκος vector: 128 στοιχεία, πλάτος μνήμης: 64 bits.
   * Κάθε φόρτωση φέρνει 64 bits = 8 bytes (αφού διπλή ακρίβεια σημαίνει ότι κάθε στοιχείο είναι 8 bytes).
   * Συνεπώς, 128 στοιχεία \* 8 bytes/στοιχείο / 64 bits/πλάτος τράπεζας = 16 πράξεις μνήμης.
   * Με 8 memory banks, κάθε λειτουργία παραλληλίζεται πλήρως στις τράπεζες.
2. **Vector-scalar multiply** (MULVS.D V2, V1, F0):
   * Πολλαπλασιάζει κάθε στοιχείο του V1 με το κλιμάκιο F0.
   * Καθυστέρηση του πολλαπλασιαστή: 4 κύκλοι.
   * Fully pipelined, οπότε η απόδοση είναι 1 στοιχείο ανά κύκλο μετά την αρχική καθυστέρηση.
3. **Load Vector Y** (LV V3, Ry):
   * Παρόμοια με τo load vector X, απαιτεί 16 πράξεις μνήμης.
   * Κάθε φόρτωση απαιτεί επίσης 8 κύκλους καθυστέρησης λόγω της διαμόρφωσης της μνήμης.
4. **Vector Sum** (ADDVV.D V4, V2, V3):
   * Προσθέτει τα στοιχεία των V2 και V3 για να προκύψει το V4.
   * Καθυστέρηση του αθροιστή: 2 κύκλοι.
   * Fully pipelined, η απόδοση είναι 1 στοιχείο ανά κύκλο μετά την αρχική καθυστέρηση.
5. **Store Sum** (SV Ry, V4):
   * Αποθηκεύει 128 στοιχεία από το V4 πίσω στη μνήμη.
   * Παρόμοιος χρόνος πρόσβασης στη μνήμη: 16 πράξεις, καθυστέρηση 8 κύκλων.

Διάγραμμα χρονισμού αγωγού:

|  |  |  |
| --- | --- | --- |
| Operation | Cycle | Comment |
| LV V1, Rx | 1-16 | 16 operations over 8 banks, 8 cycles latency |
| MULVS.D V2, V1 | 17-20 | Initial 4 cycles latency, then 1 element per cycle |
| LV V3, Ry | 21-36 | 16 operations over 8 banks, 8 cycles latency |
| ADDVV.D V4, V2, V3 | 37-38 | Initial 2 cycles latency, then 1 element per cycle |
| SV Ry, V4 | 29-54 | 16 operations over 8 banks, 8 cycles latency |

Κάθε λειτουργία ολοκληρώνεται διαδοχικά λαμβάνοντας υπόψη τις εξαρτήσεις και το chaining.

* Η Load V1 ολοκληρώνεται στον κύκλο 16.
* Η MULVS.D ξεκινά στον κύκλο 17 και είναι fully pipelined, οπότε ολοκληρώνει το πρώτο στοιχείο στον κύκλο 20 και το τελευταίο στοιχείο στον κύκλο 147 (130 κύκλοι για όλα τα στοιχεία).
* Η Load V3 ξεκινά στον κύκλο 21 (λόγω chaining με το MULVS.D), ολοκληρώνεται στον κύκλο 36.
* H ADDVV.D ξεκινά στον κύκλο 37, ολοκληρώνει το πρώτο στοιχείο στον κύκλο 38 και το τελευταίο στοιχείο στον κύκλο 166 (130 κύκλοι για όλα τα στοιχεία).
* Η Load V4 ξεκινά στον κύκλο 39 (λόγω chaining με την ADDVV.D), ολοκληρώνεται στον κύκλο 54 και ολοκληρώνει το τελευταίο στοιχείο στον κύκλο 184 (130 κύκλοι για όλα τα στοιχεία).

Συνολικός χρόνος εκτέλεσης: 184 κύκλοι για μία επανάληψη του βρόχου με μήκος διανύσματος 128 στοιχείων.

*Υποερώτημα (ii)*

Νέα χαρακτηριστικά: Δύο lanes για υπολογισμό και μνήμη. Αυτό σημαίνει ότι:

* Οι λειτουργίες Load και Store μπορούν να μειωθούν κατά το ήμισυ σε χρόνο, επειδή τώρα μπορούν να επεξεργαστούν ταυτόχρονα δύο στοιχεία.
* Αριθμητικές πράξεις (MULVS.D και ADDVV.D) μπορούν επίσης να μειωθούν στο μισό λόγω των δύο lanes.

|  |  |  |
| --- | --- | --- |
| Operation | Cycle | Comment |
| LV V1, Rx | 1-8 | 16 operations halved, each cycle does 2 operations |
| MULVS.D V2, V1 | 9-12 | Initial 4 cycles latency, then 2 elements per cycle |
| LV V3, Ry | 13-20 | 16 operations halved, each cycle does 2 operations |
| ADDVV.D V4, V2, V3 | 21-22 | Initial 2 cycles latency, then 2 elements per cycle |
| SV Ry, V4 | 23-30 | 16 operations halved, each cycle does 2 operations |

Συνολικός χρόνος εκτέλεσης με 2 lanes: 30 κύκλοι. Το νέο κόστος και πολυπλοκότητα υλοποίησης της νέας σχεδίασης:

* *Memory System*:
  + **Κόστος**: Ο διπλασιασμός των lanes θα απαιτούσε διπλασιασμό του εύρους ζώνης της μνήμης, πράγμα που σημαίνει προσθήκη πρόσθετων θυρών ή τραπεζών μνήμης για την υποστήριξη ταυτόχρονων αναγνώσεων και εγγραφών. Αυτό αυξάνει την πολυπλοκότητα του ελεγκτή μνήμης.
  + **Πολυπλοκότητα**: Η διαχείριση της ταυτόχρονης πρόσβασης στη μνήμη και η εξασφάλιση της συνοχής γίνεται πιο πολύπλοκη.
* *Functional Units*:
  + **Κόστος**: Ο διπλασιασμός του αριθμού των FUs αυξάνει την επιφάνεια του πυριτίου και την κατανάλωση ισχύος.
  + **Πολυπλοκότητα**: Η λογική ελέγχου για την αποστολή, τον συγχρονισμό και την αλυσιδωτή σύνδεση εντολών σε πολλαπλές λωρίδες γίνεται πιο πολύπλοκη. Η εξασφάλιση της αποτελεσματικής χρήσης και των δύο lanes χωρίς bottlenecks προσθέτει επίσης πολυπλοκότητα.
* *Pipeline*:
  + **Κόστος**: Πρόσθετα pipelines και μηχανισμοί ελέγχου για τη διαχείριση δύο lanes.
  + **Πολυπλοκότητα**: Αυξημένη πολυπλοκότητα στην ανίχνευση κινδύνων, την προώθηση δεδομένων και τον συνολικό έλεγχο του pipeline για την εξασφάλιση ομαλής εκτέλεσης σε όλα τα lanes.

Θέμα 2

Δίνεται αρχιτεκτονική η οποία υλοποιεί τον αλγόριθμο Tomasulo χρησιμοποιώντας ROB για in-order commit εντολών. Το pipeline του επεξεργαστή περιέχει τα στάδια Issue (IS), Execute (EX), Write Result (WR) Commit και (CMT), αγνοούμε δηλαδή τα IF και ID. Ισχύουν επίσης τα ακόλουθα:

* Τα IS, WR, CMT απαιτούν 1 κύκλο.
* Το σύστημα διαθέτει περιορισμένο αριθμό από reservation stations (RS). Συγκεκριμένα, περιέχει 2 RS για προσθέσεις/αφαιρέσεις και 2 RS για πολλαπλασιασμούς/διαιρέσεις floating point αριθμών. Αντίστοιχα, για integer αριθμούς περιλαμβάνονται 4 RS για εντολές διακλάδωσης, αριθμητικές και λογικές εντολές καθώς και 1 RS για πολλαπλασιασμούς/διαιρέσεις.
* Το σύστημα περιλαμβάνει 4 non-pipelined functional unit για πράξεις integer αριθμών. Όλες οι εντολές μεταξύ integer αριθμών διαρκούν 2 κύκλους.
* Το σύστημα περιλαμβάνει 2 non-pipelined floating point functional units, 1 για ADD/SUBD και 1 για MULD/DIVD. Οι εντολές πρόσθεσης/αφαίρεσης διαρκούν 3 κύκλους, ενώ οι εντολές πολλαπλασιασμού/διαίρεσης 5 κύκλους.
* Για τις εντολές αναφοράς στη μνήμη, στο στάδιο EX γίνεται τόσο ο υπολογισμός της διεύθυνσης αναφοράς όσο και η προσπέλαση στη μνήμη. Το σύστημα περιλαμβάνει ένα Load και ένα Store Queue καθένα από τα οποία διαθέτει 2 θέσεις. Οι εντολές χρησιμοποιούν ένα ξεχωριστό pipelined functional unit για τον υπολογισμό της διεύθυνσης και διαρκούν 1 κύκλο στην περίπτωση Hit στην cache και 5 κύκλους σε περίπτωση Miss.
* Οι εντολές διακλάδωσης υπό συνθήκη χρησιμοποιούν τα κατάλληλα RS και FU για αφαίρεση, προκειμένου να υπολογίσουν αν ισχύει η συνθήκη. Η πρόβλεψη για μια εντολή διακλάδωσης υπό συνθήκη γίνεται ταυτόχρονα με τη δρομολόγηση της εντολής. Ο έλεγχος της πρόβλεψης γίνεται αμέσως μόλις γίνει γνωστό το αποτέλεσμα της εντολής, δηλαδή στο στάδιο WR (κύκλος k). Σε περίπτωση σφάλματος, σταματά η εκτέλεση των εντολών του miss-predicted execution path και στον επόμενο κύκλο (κύκλος k+1) δρομολογείται η σωστή εντολή.
* Ο ROB έχει 9 θέσεις.
* Το σύστημα περιλαμβάνει 1 CBD. Σε περίπτωση που παραπάνω από 1 εντολές θέλουν να το χρησιμοποιήσουν, τότε προτεραιότητα αποκτά η παλαιότερη εντολή (αυτή που έγινε issued πρώτη). Θεωρήστε ότι τα branches **δεν** χρησιμοποιούν το CDB κατά τη διάρκεια του WR σταδίου.
* Για τις εντολές διακλάδωσης υπό συνθήκη, το σύστημα χρησιμοποιεί έναν (2,2) global history predictor με συνολικά 8 entries, τα οποία φαίνονται στον παρακάτω πίνακα. Ο BHR έχει τιμή ίση με 0. Δίνεται επίσης το FSM διάγραμμα του 2-bit predictor, ο οποίος προβλέπει Τ για τιμές και NT για τις υπόλοιπες.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 = 00 | 1 = 01 | 2 = 10 | 3 = 11 |
| 0 | 00 | 00 | 01 | 01 |
| 1 | ~~01~~ 00 | 11 | 10 | 00 |

* Η δεικτοδότηση του πίνακα γίνεται χρησιμοποιώντας τον BHR καθώς και τον κατάλληλο αριθμό low order bits από το PC της εντολής. Ο επεξεργαστής υλοποιεί το ISA του MIPS.
* Το σύστημα περιλαμβάνει μια fully associative cache μεγέθους 32Β με block size 16 bytes και πολιτική αντικατάστασης LRU. Αρχικά, η cache είναι άδεια.
* Οι καταχωρητές R1, R2 περιέχουν τις διευθύνσεις των 1ων στοιχείων των πινάκων Α και Β αντίστοιχα, στους οποίους έχουν αποθηκευτεί αριθμοί διπλής ακρίβειας (μήκους 8Β ο καθένας).
* Οι πίνακες είναι ευθυγραμμισμένοι.
* Δίνεται R8=1. Εκτελέστε τον κώδικα και δώστε τους χρόνους δρομολόγησης, εκτέλεσης και ολοκλήρωσης των εντολών σε έναν πίνακα όπως ο παρακάτω. ***Ποια τα τελικά περιεχόμενα της cache;***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| OP | IS | EX | WR | CMT | Comment |
| LD F0,0(R1) |  |  |  |  |  |

Στο πεδίο «Σχόλιο» δικαιολογήστε τυχόν καθυστερήσεις μεταξύ IS-EX, EX-WR, WR-SMT καθώς και ακυρώσεις και ακυρώσεις εντολών.

**Λύση**

* IS, WR, CMT: 1 cycle
* ROB: 9 θέσεις
* 1 CBD:
  + Προτεραιότητα η παλαιότερη εντολή.
  + Το branches δεν χρησιμοποιούν το CDB στο WR.
* Πρόβλεψη διακλάδωσης ταυτόχρονη με δρομολόγηση εντολής.
* Έλεγχος πρόβλεψης: WR
* Δρομολόγηση σωστής εντολής: WR+1
* BHR = 00
* R8 = 1
* Cache: Fully Associative
  + Μέγεθος: 32 Β
  + Block Size: 16 B
* Πολιτική Αντικατάστασης:
  + R1 → A[0] → 8 bytes
  + R2 → B[0] → 8 bytes

|  |  |  |  |
| --- | --- | --- | --- |
|  | RS | FU | Cycles |
| ADDD/SUBD | 2 | 1 | 3 |
| MULD/DIVD | 2 | 1 | 5 |
| BR/ADD/SUB/AND | 4 | 4 | 2 |
| MUL/DIV | 1 | 2 |
| LOAD | 2 |  | H:1, M: 5 |
| STORE | 2 |  | H:1, M: 5 |

* IS: Χώρος στο FU + Χώρος στο RS
* EX: Χώρος στο FU + Εξαρτήσεις
* WR: Χώρος στο CBD
* CMT: Μετά από το προηγούμενο

|  |  |  |
| --- | --- | --- |
|  | 8 bytes | 8 bytes |
| 0 | ~~A[0]~~ B[4] | ~~A[1]~~ B[5] |
| 1 | ~~B[0]~~ A[2] | ~~B[1]~~ A[3] |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Εντολή | IS | EX | WR | CMT | Σχόλια |
| LD F0, 0(R1) | 1 | 2-6 | 7 | 8 | Miss A[0], LRU=1 |
| ADDD F4, F4, F0 | 2 | 8-10 | 11 | 12 | RAW F0 |
| LD F1, 8(R2) | 3 | 4-8 | 9 | 13 | Miss B[1], LRU=0 |
| ADDI R1, R1, 0x10 | 4 | 5-6 | 8 | 14 | R1=A[2], CDB Conflict |
| MULD F4, F4, F1 | 5 | 12-16 | 17 | 18 | RAW F1, F4 |
| ADDI R2, R2, 0x10 | 6 | 7-8 | 10 | 19 | R2=B[2], CDB Conflict |
| SUBI R8, R8, 0x1 | 7 | 8-9 | 12 | 20 | R8=0, CDB Conflict |
| ANDI R9, R8, 0x1 | 8 | 13-14 | 15 | 21 | RAW R8, R9=0 |
| BNEZ R9, REP **(Dir: 4)** | 9 | 16-17 | 18 | 22 | RAW R9, Pred=NT, Res=NT, (4=0**1**00, BHR=00), BHR’=00 |
| ADDI R2, R2, 0x10 | 11 | 12-13 | 14 | 23 | R2=B[4] |
| LD F1, 8(R2) | 13 | 15-19 | 20 | 24 | RAW R2, Miss B[5], LRU=1 |
| MULD F1, F1, F1 | 14 | 17-21 | 22 | 25 | RAW F1 |
| MULD F4, F4, F1 | 18 | 23-27 | 28 | 29 | RAW F1, F4 |
| ADDD F4, F4, F4 | 19 | 29-31 | 32 | 33 | RAW F4 |
| BEZ R8, END **(Dir: C)** | 20 | 21-22 | 23 | 34 | Pred=NT, Res=T (C=1**1**00, BHR=00) |
| **X** LD F0, 8(R1) | 21 | 22-X | X | X | Miss A[3], LRU=0 |
| **X** MULD F4, F4, F0 | 22 | X | X | X | RAW F0, F4 |
| **X** SUBI R8, R8, 0x1 | 23 | X | X | X | Flush |
| SD F4, 0(R1) | 24 | 25 | 26 | 35 | Hit, LRU=0 |

# Κανονική 19

Θέμα 2

Δίνεται αρχιτεκτονική η οποία υλοποιεί τον αλγόριθμο Tomasulo χρησιμοποιώντας ROB για in-order commit εντολών. Το pipeline του επεξεργαστή περιέχει τα στάδια Issue (IS), Execute (EX), Write Result (WR) Commit και (CMT), αγνοούμε δηλαδή τα IF και ID. Ισχύουν επίσης τα ακόλουθα:

* Τα IS, WR, CMT απαιτούν 1 κύκλο.
* Το σύστημα διαθέτει περιορισμένο αριθμό από reservation stations (RS). Συγκεκριμένα, περιέχει 3 RS για προσθέσεις/αφαιρέσεις και 2 RS για πολλαπλασιασμούς/διαιρέσεις floating point αριθμών. Αντίστοιχα, για integer αριθμούς περιλαμβάνονται 3 RS για εντολές διακλάδωσης, αριθμητικές και λογικές εντολές καθώς και 1 RS για πολλαπλασιασμούς/διαιρέσεις.
* Το σύστημα περιλαμβάνει 3 pipelined functional unit για πράξεις integer αριθμών. Όλες οι εντολές μεταξύ integer αριθμών διαρκούν 2 κύκλους.
* Το σύστημα περιλαμβάνει 2 non-pipelined floating point functional units, 1 για ADD/SUBD και 1 για MULD/DIVD. Οι εντολές πρόσθεσης/αφαίρεσης διαρκούν 3 κύκλους, ενώ οι εντολές πολλαπλασιασμού/διαίρεσης 5 κύκλους.
* Για τις εντολές αναφοράς στη μνήμη, στο στάδιο EX γίνεται τόσο ο υπολογισμός της διεύθυνσης αναφοράς όσο και η προσπέλαση στη μνήμη. Το σύστημα περιλαμβάνει ένα Load και ένα Store Queue καθένα από τα οποία διαθέτει 2 θέσεις. Οι εντολές χρησιμοποιούν ένα ξεχωριστό pipelined functional unit για τον υπολογισμό της διεύθυνσης και διαρκούν 1 κύκλο στην περίπτωση Hit στην cache και 4 κύκλους σε περίπτωση Miss.
* Οι εντολές διακλάδωσης υπό συνθήκη χρησιμοποιούν τα κατάλληλα RS και FU για αφαίρεση, προκειμένου να υπολογίσουν αν ισχύει η συνθήκη. Η πρόβλεψη για μια εντολή διακλάδωσης υπό συνθήκη γίνεται ταυτόχρονα με τη δρομολόγηση της εντολής. Ο έλεγχος της πρόβλεψης γίνεται αμέσως μόλις γίνει γνωστό το αποτέλεσμα της εντολής, δηλαδή στο στάδιο WR (κύκλος k). Σε περίπτωση σφάλματος, σταματά η εκτέλεση των εντολών του miss-predicted execution path και στον επόμενο κύκλο (κύκλος k+1) δρομολογείται η σωστή εντολή.
* Ο ROB έχει 10 θέσεις.
* Το σύστημα περιλαμβάνει 1 CBD. Σε περίπτωση που παραπάνω από 1 εντολές θέλουν να το χρησιμοποιήσουν, τότε προτεραιότητα αποκτά η παλαιότερη εντολή (αυτή που έγινε issued πρώτη). Θεωρήστε ότι τα branches δεν χρησιμοποιούν το CDB κατά τη διάρκεια του WR σταδίου.
* Για τις εντολές διακλάδωσης υπό συνθήκη, το σύστημα χρησιμοποιεί έναν (2,2) global history predictor με συνολικά 8 entries, τα οποία φαίνονται στον παρακάτω πίνακα. Ο BHR έχει τιμή ίση με 1. Δίνεται επίσης το FSM διάγραμμα του 2-bit predictor, ο οποίος προβλέπει Τ για τιμές και NT για τις υπόλοιπες.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 = 00 | 1 = 01 | 2 = 10 | 3 = 11 |
| 0 | 00 | 00 | 01 | 01 |
| 1 | 01 | ~~11~~ 10 | ~~10~~, 01 | 00 |

* Η δεικτοδότηση του πίνακα γίνεται χρησιμοποιώντας τον BHR καθώς και τον κατάλληλο αριθμό low order bits από το PC της εντολής. Ο επεξεργαστής υλοποιεί το ISA του MIPS.
* Το σύστημα περιλαμβάνει μια fully associative cache μεγέθους 32Β με block size 16 bytes και πολιτική αντικατάστασης LRU. Αρχικά, η cache είναι άδεια.
* Οι καταχωρητές R1, R2 περιέχουν τις διευθύνσεις των 1ων στοιχείων των πινάκων Α και Β αντίστοιχα, στους οποίους έχουν αποθηκευτεί αριθμοί διπλής ακρίβειας (μήκους 8Β ο καθένας).
* Οι πίνακες είναι ευθυγραμμισμένοι.
* (Έβαλα τον κώδικα παρακάτω)

**Λύση**

* IS, WR, CMT: 1 cycle
* ROB: 10 θέσεις
* 1 CBD:
  + Προτεραιότητα η παλαιότερη εντολή.
  + Το branches δεν χρησιμοποιούν το CDB στο WR.
* Πρόβλεψη διακλάδωσης ταυτόχρονη με δρομολόγηση εντολής.
* Έλεγχος πρόβλεψης: WR
* Δρομολόγηση σωστής εντολής: WR+1
* BHR = ~~01~~, ~~10~~, 01
* R8 = ~~1~~, ~~0~~, ~~1~~, 0
* Cache: Fully Associative
  + Μέγεθος: 32 Β
  + Block Size: 16 B
* Πολιτική Αντικατάστασης:
  + R1 → A[0] → 8 bytes
  + R2 → B[0] → 8 bytes

|  |  |  |  |
| --- | --- | --- | --- |
|  | RS | FU | Cycles |
| ADDD/SUBD | 3 | 1 | 3 |
| MULD/DIVD | 2 | 1 | 5 |
| BR/ADD/SUB/AND | 3 | 3 | 2 |
| MUL/DIV | 1 | 3 | 2 |
| LOAD | 2 |  | H:1, M: 4 |
| STORE | 2 |  | H:1, M: 4 |

* IS: Χώρος στο FU + Χώρος στο RS
* EX: Χώρος στο FU + Εξαρτήσεις
* WR: Χώρος στο CBD
* CMT: Μετά από το προηγούμενο

|  |  |  |
| --- | --- | --- |
|  | 8 bytes | 8 bytes |
| 0 | ~~Α[0]~~ Β[2] | ~~Α[1]~~ Β[3] |
| 1 | ~~B[0]~~ ~~Β[2]~~ Β[0] | ~~B[1]~~ ~~Β[3]~~ Β[1] |

C = 1**(1)**00

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Εντολή | IS | EX | WR | CMT | Σχόλια |
| LD F0, 0(R1) | 1 | 2-5 | 6 | 7 | Cache Miss, LRU = 1 |
| ADDD F4, F4, F0 | 2 | 7-9 | 10 | 11 | RAW F0, (WAR F4) |
| LD F1, 0(R2) | 3 | 4-7 | 8 | 12 | Cache Miss, LRU = 0 |
| MULD F4, F4, F1 | 4 | 11-15 | 16 | 17 | RAW F1, F4, (WAR F4) |
| ANDI R9, R8, 0x2 | 5 | 6-7 | 9 | 18 | CDB Conflict |
| BNEZ R9, NEXT | 6 | 10-11 | 12 | 19 | RAW R9, Pred = 1, Res = NT |
| **X** LD F5, 8(R1) | 7 | 8 | 11 | X | Cache Hit, CDB Conflict, LRU = 1 |
| **X** ADDD F4, F4, F5 | 8 | 12-Χ | X | X | RAW F5, F4, (WAR F4) |
| **X** ADDI R1, R1, 0x8 | 9 | 10-11 | 12 | X | (WAR R1) |
| **X** SUBI R8, R8, 0x1 | 10 | 11-12 | X | X | (WAR R8) |
| **X** BNEZ R8, LOOP | 11 | 12-Χ | X | X | RAW R8 |
| LD F2, 16(R2) | 13 | 14-17 | 18 | 20 | Cache Miss, LRU = 0, |
| MULD F2, F2, F5 | 14 | 19-23 | 24 | 25 | RAW F2, (WAR F2) |
| ADDD F4, F4, F2 | 15 | 25-27 | 28 | 29 | RAW F2, (WAR F4) |
| LD F5, 8(R1) | 16 | 17 | 19 | 30 | Cache Hit, LRU = 1, (WAR F5) , CDB Conflict |
| ADDD F4, F4, F5 | 17 | 29-31 | 32 | 33 | RAW F4, F5, (WAR F4) |
| ADDI R1, R1, 0x8 | 18 | 19-20 | 21 | 34 | Cache Hit, LRU = 0 |
| SUBI R8, R8, 0x1 | 19 | 20-21 | 22 | 35 |  |
| BNEZ R8, LOOP | 20 | 23-24 | 25 | 36 | RAW R8, Pred T, Res NT |
| **X** LD F0, 0(R1) | 21 | 22 | 23 | 37 | Cache Hit, LRU = 1 |
| **X** ADDD F4, F4, F0 | 22 | X | X | X | RAW F0, (WAR F4) |
| **X** LD F1, 0 (R2) | 23 | 24-X | X | X | Cache Miss, LRU = 0 |
| **X** MULD F4, F4, F1 | 24 | X | X | X | ROB Full |
| SD F4, 16(R2) | 26 | 33-36 | 37 | 38 | Cache Miss, LRU = 1, RAW F4 |

Θέμα 4

**Λύση**

|  |  |  |  |
| --- | --- | --- | --- |
|  | P1 | P2 | Cycles |
|  | IIII | IIII |  |
| RD x[0].a | EIII | IIII | 2+8 |
| RD x[0].c | EIII | IEII | 2+8 |
| RD x[0].b | EIII | IEII | 1 |
| RD x[0].d | EIII | IEII | 1 |
| WR x[0].a | MIII | IEII | 1 |
| WR x[0].c | MIII | IMII | 1 |
| RD x[1].a | MSII | ISII | 2+8 |
| RD x[1].c | MSII | ISEI | 2+8 |
| RD x[1].b | MSSI | ISSI | 2+8 |
| RD x[1].d | MSSI | ISSI | 1 |
| WR x[1].a | MMSI | IISI | 1+1 |
| WR x[1].c | MMII | IIMI | 1+1 |
| RD x[2].a | MMIE | IIMI | 2+8 |
| RD x[2].c | MMIE | EIMI | 2+8 |
| RD x[2].b | MMIE | EIMI | 1 |
| RD x[2].d | MMIE | EIMI | 1 |
| WR x[2].a | MMIM | EIMI | 1 |
| WR x[2].c | MMIM | MIMI | 1 |
| RD x[3].a | SMIM | SIMI | 2+8+8 |
| RD x[3].c | SMIM | SEMI | 2+8 |
| RD x[3].b | SSIM | SSMI | 2+8+8 |
| RD x[3].d | SSIM | SSMI | 1 |
| WR x[3].a | MSIM | ISMI | 1+1 |
| WR x[3].c | MIIM | IMMI | 1+1 |
| RD x[4].a | MIEM | IMMI | 2+8 |
| RD x[4].c | MIEM | IMME | 2+8 |
| RD x[4].b | MIEM | IMME | 1 |
| RD x[4].d | MIEM | IMME | 1 |
| WR x[4].a | MIMM | IMME | 1 |
| WR x[4].c | MIMM | IMMM | 1 |
| RD x[5].a | MIMS | IMMS | 2+8+8 |
| RD x[5].c | MIMS | EMMS | 2+8 |
| RD x[5].b | SIMS | SMMS | 2+8+8 |
| RD x[5].d | SIMS | SMMS | 1 |
| WR x[5].a | SIMM | SMMI | 1+1 |
| WR x[5].c | IIMM | MMMI | 1+1 |
| **Total** |  |  | **209** |

|  |  |
| --- | --- |
| B0 | x[0].a x[0].b |
| B1 | x[0].c x[0].d x[1].a |
| B2 | x[1].b x[1].c x[1].d |
| B3 | x[2].a x[2].b |
| B0 | x[2].c x[2].d x[3].a |
| B1 | x[3].b x[3].c x[3].d |
| B2 | x[4].a x[4].b |
| B3 | x[4].c x[4].d x[5].a |
| B0 | x[5].b x[5].c x[5].d |

|  |  |  |
| --- | --- | --- |
| Block | L1 | L2 |
| 0 | ~~x[0].a x[0].b~~  ~~x[2].c x[2].d x[3].a~~  x[5].b x[5].c x[5].d | ~~x[2].c x[2].d x[3].a~~  x[5].b x[5].c x[5].d |
| 1 | ~~x[0].c x[0].d x[1].a~~  x[3].b x[3].c x[3].d | ~~x[0].c x[0].d x[1].a~~  x[3].b x[3].c x[3].d |
| 2 | ~~x[1].b x[1].c x[1].d~~  x[4].a x[4].b | x[1].b x[1].c x[1].d |
| 3 | ~~x[2].a x[2].b~~  x[4].c x[4].d x[5].a | x[4].c x[4].d x[5].a |

# Κανονική 21

Θέμα 3

* IS, WR, CMT: 1 cycle
* ROB: 8 θέσεις
* 1 CBD:
  + Προτεραιότητα η παλαιότερη εντολή.
  + Το branches δεν χρησιμοποιούν το CDB στο WR.
* Πρόβλεψη διακλάδωσης ταυτόχρονη με δρομολόγηση εντολής.
* Έλεγχος πρόβλεψης: WR
* Δρομολόγηση σωστής εντολής: WR+1
* BHR (Τ, >=2) = 1,
* R8 = 1
* Cache: Fully Associative
  + Μέγεθος: 32 Β
  + Block Size: 16 B
* Πολιτική Αντικατάστασης:
  + R1 → A[0] → 8 bytes
  + R2 → B[0] → 8 bytes

|  |  |  |  |
| --- | --- | --- | --- |
|  | RS | FU | Cycles |
| ADDD/SUBD | 3 | 1 | 3 |
| MULD/DIVD | 2 | 2 | 7 |
| BR/ADD/SUB/AND | 3 | 3 | 1 |
| MUL/DIV | 1 | 3 | 1 |
| LOAD | 2 |  | H:1, M: 3 |
| STORE | 2 |  | H:1, M: 3 |

* IS: Χώρος στη ROB + Χώρος στο RS
* EX: Χώρος στο FU + Εξαρτήσεις
* WR: Χώρος στο CBD
* CMT: Μετά από το προηγούμενο

|  |  |  |
| --- | --- | --- |
|  | 00 | 01 |
| 0 | ~~10~~ 01 | 01 |
| 1 | 00 | ~~11~~ 10 |

|  |  |  |
| --- | --- | --- |
|  | 8 bytes | 8 bytes |
| 0 | ~~A[0]~~ A[2] | ~~A[1]~~ A[3] |
| 1 | ~~B[2]~~ B[0] | ~~B[3]~~ B[1] |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Εντολή | IS | EX | WR | CMT | Σχόλια |
| LD F0, 0(R1) | 1 | 2-4 | 5 | 6 | Miss, LRU=1 |
| ADDD F4, F4, F0 | 2 | 6-8 | 9 | 10 | RAW F0, (WAR F4) |
| LD F1, 16(R2) | 3 | 4-6 | 7 | 11 | Miss, LRU=0 |
| MULD F4, F4, F1 | 4 | 10-16 | 17 | 18 | RAW F1, F4, (WAR F4) |
| ANDI R9, R8, 0x2 | 5 | 6 | 8 | 19 | R8=1, R9=0, CDB Conflict |
| BNEZ R9, NEXT | 6 | 9 | 10 | 20 | RAW R9, Pred T, Res NT,  (416 = 0**1**00, BHR = 1 => 11), BHR=0 |
| **X** LD F5, 16(R1) | 7 | 8-10 | X | X | Miss A[2], LRU=1, CDB Conflict |
| **X** ADDD F4, F4, F5 | 8 | X | X | X | RAW F4, F5, (WAR F4) |
| **X** ADDI R1, R1, 0x8 | 9 | 10 | X | X | R1=A[1] |
|  |  |  |  |  | ROB Full, Flush |
| **X** SUBI R8, R8, 0x1 | X | X | X | X |  |
| LD F2, 16(R2) | 11 | 12 | 13 | 21 | Hit B[3], LRU=0 |
| MULD F4, F2, F5 | 12 | 13-19 | 20 | 22 | RAW F2 |
| LD F5, 16(R1) | 13 | 14 | 15 | 23 | Hit A[3], LRU=1, (WAR F5) |
| ADDD F4, F4, F5 | 14 | 21-23 | 24 | 25 | RAW F4, F5, (WAR F4) |
| ADDI R1, R1, 0x8 | 15 | 16 | 18 | 26 | R1=A[2] |
| SUBI R8, R8, 0x1 | 19 | 20 | 21 | 27 | R8=0, ROB Full |
| BNEZ R8, LOOP | 20 | 22 | 23 | 28 | RAW R8, Pred T, Res NT  (016=0, BHR=0 => 10), BHR=1 |
| **X** LD F0, 0(R1) | 21 | 22 | X | X | Hit A[2], LRU=1 |
| **X** ADDD F4, F4, F0 | 22 | X | X | X | RAW F0, (WAR F4) |
| **X** LD F1, 0 (R2) | 23 | X | X | X | Flush |
| SD F4, 8(R2) | 24 | 25-27 | 28 | 29 | Miss B[1], LRU=0 |