Προηγμένα Θέματα Αρχιτεκτονικής Υπολογιστών

Άσκηση Παράλληλης Επεξεργασίας

Ομαδική εργασία - Ομάδα 1η

Ντούλας Δημήτριος (ΑΜ: 271)

Αναστασίου Χρήστος (ΑΜ: 265)

Καπετανάκης Κωνσταντίνος - Ραφαήλ (ΑΜ: 269)

Επιφάνιος Παπαδόπουλος (ΑΜ: 272)

Ρούσης Παναγιώτης (ΑΜ: 274)

1. Ένας επεξεργαστής με δυναμικό χρονοπρογραμματισμό και πολιτική ανάκτησης τελεστών δεσμευμένη (issue-bound fetch policy) έχει 3 μονάδες εκτέλεσης – μία μονάδα LOAD/STORE, μία μονάδα ADD/SUB και μία μονάδα MUL/DIV. Διαθέτει σταθμό κρατήσεων (reservation station) με 1 θέση (slot) ανά μονάδα εκτέλεσης και ένα ενιαίο αρχείο καταχωρητών. Οι κύκλοι εκτέλεσης που απαιτούνται για κάθε τύπο εντολής είναι:

LOAD/STORE : 2
ADD/SUB : 1
MUL : 2
DIV : 4

α. Ξεκινώντας με την παρακάτω ακολουθία εντολών στην προσωρινή μνήμη ανάκτησης εντολών και άδειους σταθμούς κρατήσεων, για κάθε εντολή βρείτε τον κύκλο στον οποίο θα εκκινήσει (issue cycle) και τον κύκλο στον οποίο θα "γράψει" το αποτέλεσμα (result write cycle).

load R6, 34(R12)
load R2, 45(R13)
mul R0, R2, R4
sub R8, R2, R6
div R10, R0, R6
add R6, R8, R2

Απάντηση (α)

Στα παρακάτω σχεδιαγράμματα ο οριζόντιος άξονας είναι τα **machine cycles** ενώ ο κάθετος οι **εντολές**. Θα υποθέσουμε ότι όλα τα instructions έχουν γίνει fetched για βελτιωμένη απόδοση.

	Instruction					1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	LOAD	R6,	34(R12)		F	D	EX1	EX1	WB											
2	LOAD	R2,	45(R13)		F	STALL	STALL	STALL	D	EX1	EX1	WB								
3	MUL	R0,	R2,	R4	F	D	RAW	RAW	RAW	RAW	RAW	RAW	EX3	EX3	WB					
4	SUB	R8,	R2,	R6	F	D	RAW	RAW	RAW	RAW	RAW	RAW	EX2	WB						
5	DIV	R10,	R0,	R6	F	STALL	STALL	STALL	STALL	STALL	STALL	STALL	STALL	STALL	D	EX3	EX3	EX3	EX3	WB
6	ADD	R6,	R8,	R2	F	STALL	STALL	STALL	STALL	STALL	STALL	STALL	STALL	D	EX2	WB				
	Dependencies RAW																			

- **Εντολή Πρώτη:**
  - Όσον αφορά αυτήν την πρώτη εντολή και την εκτέλεση της, δεν υπάρχει κάτι άξιο σχολιασμού.
  - **Issue cycle = 1 και write-back cycle = 4**

- **Εντολή Δεύτερη**:
  - Επειδή το LOAD/STORE έχει μόνο μία μονάδα εκτέλεσης, η δεύτερη load εντολή πρέπει να περιμένει μέχρι να ολοκληρωθεί η εκτέλεση της πρώτης.
  - **Issue cycle = 4 και write-back cycle = 7**

- **Εντολή Τρίτη:**
  - Η εντολή τώρα multiply, επειδή εκτελείται σε διαφορετική execution unit μπορεί να γίνει άμεσα decode χωρίς να χρειάζεται να περιμένει την ολοκλήρωση της εντολής LOAD όπως έγινε στην προηγουμένως, αλλά υπάρχει μία RAW (true) dependency. Ειδικότερα, η εντολή mul χρησιμοποιεί ως operand την τιμή του καταχωρητή R2, πρέπει λοιπόν να περιμένει (stall) μέχρι το τέλος της εκτέλεσης της εντολής load R2, 45(R13), να αποθηκεύσει αυτήν την τιμή στον καταχωρητη, την οποία έπειτα θα χρησιμοποιήσει η mul R0, R2, R4. Διαφορετικά το αποτέλεσμα της θα ήταν λανθασμένο.
  - **Issue cycle = 1 και write-back cycle = 10**

- **Εντολή Τέταρτη:**

  - Η εντολή sub, επειδή όπως και στην περίπτωση της mul, εκτελείται σε διαφορετική ALU μπορεί να γίνει άμεσα decode χωρίς να χρειάζεται να περιμένει την ολοκλήρωση των προηγουμένων εντολών

- Παρομοίως με την προηγούμενη εντολή, υπάρχει μια RAW true dependency, συνεπώς πρέπει να περιμένει.
- **Issue cycle = 1 και write-back cycle = 9**

- **Εντολή Πέμπτη:**

  - Η εντολή DIV μοιράζεται την μία και μοναδική ALU με την MUL, συνεπώς πριν γίνει decode πρέπει να περιμένει την ολοκλήρωση της προηγούμενης
  - Επίσης υπάρχει και εδώ μια RAW true dependency, συνεπώς η εντολή να περιμένει την ολοκλήρωση της mul R0, R2, R4, να γράψει το αποτέλεσμα στον καταχωρητή (πράγμα που γίνεται στον κύκλο 10 - πράσινο) έτσι κ' αλλιώς
  - **Issue cycle = 10 και write-back cycle = 15**

- **Εντολή Έκτη:**

  - Η εντολή ADD μοιράζεται την μία και μοναδική ALU με την SUB, συνεπώς πριν γίνει decode πρέπει να περιμένει την ολοκλήρωση της.
  - Υπάρχουν 2 true dependencies, συνεπώς πρέπει έτσι κ' αλλιώς να περιμένει την ολοκλήρωση των εντολών load R2, 45(R13) και sub R8, R2, R6
  - **Issue cycle = 9 και write-back cycle = 1**

Instruction	issue cycle	result write cycle
LOAD	1	4
LOAD	4	7
MUL	1	10
SUB	1	9
DIV	10	15
ADD	9	11

β. Αν υπήρχαν 2 θέσεις (slot) ανά μονάδα εκτέλεσης πως θα άλλαζε το παραπάνω;

Απάντηση (β)

	Instruction					1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	LOAD	R6,	34(R12)		F	D	EX1	EX1	W											
2	LOAD	R2,	45(R13)		F	STALL	D	STALL	STALL	EX1	EX1	W								
3	MUL	R0,	R2,	R4	F	D	RAW	RAW	RAW	RAW	RAW	RAW	EX3	EX3	W					
4	SUB	R8,	R2,	R6	F	D	RAW	RAW	RAW	RAW	RAW	RAW	EX2	W						
5	DIV	R10,	R0,	R6	F	RAW	RAW	RAW	RAW	RAW	RAW	RAW	RAW	RAW	D	EX3	EX3	EX3	EX3	W
6	ADD	R6,	R8,	R2	F	RAW	RAW	RAW	RAW	RAW	RAW	RAW	RAW	D	EX2	W				

- Επειδή υπάρχει και δεύτερο slot διαθέσιμο η load R2, 45(R13) γίνεται decode αμέσως μόλις η προηγούμενη ξεκινήσει το execution.
- Αργότερα, αμέσως μόλις η προηγούμενη ολοκληρώσει το execution και περάσει στο write-back, η ίδια ξεκινά να εκτελείται με αποτέλεσμα να κάνει load το αποτέλεσμα της στον καταχωρητή 1 κύκλο νωρίτερα συγκριτικά με πριν.
- Παρά την δυνατότητα των δύο slot, δεν έχουμε βελτίωση του χρόνου εκτέλεσης των παραπάνω εντολών. Υποθετικά μιλώντας, εάν υπήρχαν περισσότερες εντολές, τα περισσότερα slot θα βελτίωναν τον χρόνο εκτέλεσης.

Instruction	issue cycle	result write cycle
LOAD	1	4
LOAD	2	7
MUL	1	10
SUB	1	9
DIV	10	15
ADD	9	11

γ. Με ποιο τρόπο μπορεί να μειωθεί ο χρόνος εκτέλεσης της ακολουθίας εντολών;

Απάντηση (γ)

Instruction					1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LOAD	R2,	45(R13)		F	D	EX1	EX1	W											
LOAD	R6,	34(R12)		F	STALL	STALL	STALL	D	EX1	EX1	W								
MUL	R0,	R2,	R4	F	D	RAW	RAW	RAW	EX3	EX3	W								
SUB	R8,	R2,	R6	F	D	RAW	RAW	RAW	RAW	RAW	RAW	EX2	WB						
DIV	R10,	R0,	R6	F	RAW	RAW	RAW	RAW	RAW	RAW	D	EX3	EX3	EX3	EX3	W			
ADD	R6,	R8,	R2	F	RAW	RAW	RAW	RAW	RAW	RAW	RAW	RAW	D	EX2	W				

Αλλάξανε θέσεις μεταξύ τους οι LOAD οπότε το depentacy ολοκληρώνεται σε λιγοτερους κύκλους

Είναι φανερό από τα παραπάνω πως τα dependencies (ή η έλλειψη αυτών) παίζει καθοριστικό ρόλο στην απόδοση ενός επεξεργαστή. Για αυτό και όλοι οι σύγχρονοι επεξεργαστές χρησιμοποιούν την τεχνική του δυναμικού προγραμματισμού (dynamic processing) και Out-of-Order execution ωστέ να την βελτιώσουν. Πιο συγκεκριμένα θα διαβαστούν εκ των προτέρων οι εντολές, θα βρεθούν τυχόν dependencies και θα αλλάξει η σειρά που οι εντολές εκτελούνται εάν αυτά το επιτρέπουν και μόνο εφόσον το τελικό αποτέλεσμα του προγράμματος παραμένει αναλλοίωτο. Ας δούμε πως μπορούμε να εκμεταλλευτούμε τις δυνατότητες αυτές στο πρόγραμμα μας

- Αλλάζοντας την σειρά που οι δύο πρώτες εντολές γίνονται load, χωρίς αυτό να αλλάζει τα τελικά αποτελέσματα των εντολών, καταφέραμε αρχικά να επισπεύσουμε την εκτέλεση της mul R0, R2, R4 η οποία περιμένει την τιμή του καταχωρητή R2.
- Επειδή η mul R0, R2, R4 έγραψε την τιμή της στον καταχωρητή νωρίτερα, μια τιμή που τώρα η div R10, R0, R6 χρειάζεται, μπορεί η div να ξεκινήσει την εκτέλεση της νωρίτερα (προσοχή και στην R6 RAW dependency, δεν μας επηρεάζει τώρα αλλά θα μπορούσε υπό διαφορετικές συνθήκες)

Αλλάζοντας λοιπόν την σειρά που οι δύο (LOAD) πρώτες εντολές εκτελούνται, και μέσω των αλυσιδωτών αντιδράσεων που αυτή η αλλαγή επέφερε, καταφέραμε στο τέλος να μειώσουμε τον χρόνο εκτέλεσης του προγράμματος απο 15 σε 12 κύκλους. Υποθετικά, εάν ακολουθούσαν και άλλες εντολές που δεν θα είχαν dependencies, θα μπορούσαμε ίσως να τις εκτελέσουμε

out-of-order κάπου ανάμεσα στις ήδη υπάρχουσες αυξάνοντας έτσι το throughput το επεξεργαστή μας.

Instruction	issue cycle	result write cycle
LOAD	1	4
LOAD	4	7
MUL	1	7
SUB	1	9
DIV	7	12
ADD	9	11