-9/96 - 1/96 - 9/94 - 9/95 - 2/95

Πώς ταξινομούνται οι παράλληλες αρχιτεκτονικές και τί σημαίνει speedup σε παράλληλες αρχιτεκτονικές (ποιές είναι οι 2 βασικές κατηγορίες παράλληλων αρχιτεκτονικών και ποιά τα χαρακτηριστικά τους) : οι παράλληλες αρχιτεκτονικές ταξινομούνται με βάση την οργάνωση της μνήμης τους. Έτσι υπάρχουν 2 κύριες κατηγορίες. Στην πρώτη που ονομάζεται centralized shared memory uniform οι επεξεργαστές μοιράζονται μια κεντρική μνήμη. Οι επεξεργαστές είναι συνδεδεμένοι σε ένα bus και έχουν ομογενή προσπέλαση στη μνήμη. Στη δεύτερη κατηγορία ανήκουν τα συστήματα που η μνήμη είναι διαμοιρασμένη στους επεξεργαστές, οι οποίοι τώρα είναι περισσότεροι. Χαρακτηριστικό είναι ότι χρειάζεται ένας πιο πολύπλοκος τρόπος διασύνδεσης των επεξεργαστών. Οι παράλληλες αρχιτεκτονικές χωρίζονται σε ειδικού και γενικού σκοπού. Οι ειδικού σκοπού χωρίζονται σε systolic και FFT και οι γενικού σε shared memory και σε distributed (message passing). Οι shared χωρίζονται σε multi's και dance hall. Για το speedup σε παράλληλες αρχιτεκτονικές έχουμε : speedup = (χρόνος εκτέλεσης σε μία CPU) / (χρόνος εκτέλεσης σε n CPU).

2/95

Γιατί όταν αυξηθεί ο αριθμός των επεξεργαστών σε ένα παράλληλο υπολογιστή δεν αυξάνεται ανάλογα και η απόδοση του συστήματος. Ποιό είναι το ανώτατο όριο αύξησης της απόδοσης : δύο κύριοι λόγοι δεν προκαλούν ανάλογη αύξηση της απόδοσης ενός υπολογιστικού συστήματος με την αύξηση των επεξεργαστών. Η αύξηση του αριθμού των επεξεργαστών οδηγεί σε αυξημένες απαιτήσεις επικοινωνίας. Όταν το προσφερόμενο bandwidth αδυνατεί να καλύψει τις ανάγκες των επεξεργαστών τότε παρουσιάζονται καθυστερήσεις και οι επεξεργαστές είναι σε κατάσταση αναμονής των δεδομένων. Ο δεύτερος είναι το ποσοστό παραλληλοποίησης του εκάστοτε προγράμματος. Όσο πιο μικρό είναι τόσο μικρότερη η συμβολή των περισσότερων επεξεργαστών στην απόδοση. Το ανώτατο όριο απόδοσης δίνεται από το νόμο του Amdahl. Άν το ρ% του προγράμματος είναι παραλληλοποιήσιμο η μέγιστη απόδοση είναι : speedup = 1 / (ρ/κ + (1-ρ)), η παραπάνω σχέση δίνει το θεωρητικό μέγιστο και δεν περιλαμβάνει το κόστος επικοινωνίας. Λαμβάνοντας υπόψη το κόστος επικοινωνίας από ένα σημείο και μετά δεν υπάρχει αύξηση της απόδοσης με αύξηση του κ (αριθμοί επεξεργαστών). Δηλαδή υπάρχει κάποιο μέγιστο της συνάρτησης speedup (κ).

Ποιοί οι λόγοι που δεν έχουν διαδοθεί ευρέως οι παράλληλοι υπολογιστές. Ποιές εξελίξεις θα οδηγήσουν στη διάδοση των παράλληλων υπολογιστών : 2/95

- Το μεγάλο κόστος ανάπτυξης τέτοιων συστημάτων που τα περιόρισε σε ειδικές εφαρμογές. Επίσης τα αποτελέσματα χρήσης τους δύσκολα δικαιολογούν το κόστος απόκτησής τους.
- Η έλλειψη προγραμμάτων που εκμεταλλεύονται την παραλληλία και το γεγονός ότι όλα τα προβλήματα δεν μπορούν να παραλληλοποιηθούν ή δεν είναι κατάλληλα για συγκεκριμένα συστήματα
- Όλη η τεχνολογία βρίσκεται ακόμα στα σπάργανα με αποτέλεσμα τα πράγματα να είναι πολύ ρευστά και πολλές προσεγγίσεις να αποδεικνύονται ασύμφορες.

Η εξάπλωση της χρήσης τους θα επωφεληθεί από τα εξής :

- Καθώς οι μικροεπεξεργαστές θα παραμείνουν ο κύριος τρόπος υλοποίησης επεξεργαστών, ο πιο λογικός τρόπος αύξησης της απόδοσης είναι η διασύνδεση πολλών επεξεργαστών. Αυτό θα είναι και πιο συμφέρον από πλευράς κόστους
- Δεν γνωρίζουμε για πόσο καιρό ακόμη τα επιτεύγματα της αρχιτεκτονικής θα συντηρήσουν την αλματώδη πρόοδο της απόδοσης των μικροεπεξεργαστών. Ήδη μοντέρνοι επεξεργαστές με έκδοση πολλών εντολών έχουν γίνει τόσο πολύπλοκοι που η αύξηση στην απόδοση φαίνεται να έχει ισοσκελιστεί από την πολυπλοκότητά τους και το κόστος.
- Αρχίζει να διαφαίνεται μια κινητικότητα και πρόοδος στο λογισμικό το οποίο αποτελεί την τροχοπέδη της εξάπλωσης των παράλληλων συστημάτων.
- Οι εξελίξεις ωθούν την τεχνολογία στα όριά της και οι μικροεπεξεργαστές πλησιάζουν όλο και περισσότερο στο όριο της ταχύτητας του φωτός.

Κατηγοριοποίηση παράλληλων πολυεπεξεργαστών :

1. SISD : αυτή η κατηγορία περιλαμβάνει τους μονούς επεξεργαστές.
2. SIMD : η ίδια εντολή εκτελείται από πολλαπλούς επεξεργαστές χρησιμοποιώντας διαφορετικές ροές δεδομένων. Κάθε επεξεργαστής έχει τη δική του μνήμη δεδομένων (οπότε και πολλαπλά δεδομένα), αλλά υπάρχει μία απλή εντολή μνήμης και ελέγχου επεξεργαστή η οποία φορτώνει και αποστέλλει εντολές. Αρχιτεκτονικές που στηρίζονται στη χρήση vectors αποτελούν τη μεγαλύτερη ομάδα επεξεργαστών αυτού του τύπου.
3. MISD : δεν έχουν φτιαχτεί μέχρι σήμερα πολυεπεξεργαστές αυτού του τύπου αλλά αυτό μπορεί να γίνει στο μέλλον. Μερικοί ειδικού σκοπού επεξεργαστές προσεγγίζουν μια περιορισμένη μορφή αυτού του τύπου (υπάρχει μόνο μια μονή ροή δεδομένων η οποία εκτελείται από επιτυχείς μονάδες συναρτήσεων).
4. MIMD : κάθε επεξεργαστής φορτώνει τις δικές του εντολές και εκτελεί τα δικά του δεδομένα. Οι μικροεπεξεργαστές αυτοί είναι συχνά μη συνήθεις στο εμπόριο.

Ποιοί λόγοι είναι υπεύθυνοι για την άνοδο των MIMD πολυεπεξεργαστών :

- Οι MIMD προσφέρουν ευελιξία με τη σωστή υποστήριξη λογισμικού και υλικού. Μπορούν να λειτουργήσουν ως πολυεπεξεργαστές μονού χρήστη και να παρέχουν υψηλή απόδοση για μια εφαρμογή, αφού οι πολυπρογραμματιζόμενοι πολυεπεξεργαστές τρέχουν πολλες διεργασίες ταυτόχρονα, ή να αποτελούν συνδυασμό τέτοιων λειτουργιών.
- Οι MIMD μπορούν να βασιστούν στα πλεονεκτήματα του κόστους / απόδοσης των μικροεπεξεργαστών που δεν είναι διαθέσιμοι στο εμπόριο. Στην πραγματικότητα όλοι οι πολυεπεξεργαστές που φτιάχτηκαν μέχρι σήμερα, χρησιμοποιούν τους ίδιους μικροεπεξεργαστές που βρίσκονται σε σταθμούς εργασίας και εξυπηρετητές μονού επεξεργαστή.

Δώστε τον ορισμό του speedup. Διατυπώστε και εξηγήστε μαθηματικά το νόμο του Amdahl : Υποθέτουμε ότι κάνουμε μια βελτίωση σε μια μηχανή που αυξάνει την απόδοσή της όταν αυτή χρησιμοποιείται. Τότε speedup είναι ο λόγος :

Speedup = (απόδοση για το συνολικό έργο χρησιμοποιώντας τη βελτίωση όπου είναι δυνατό) / (απόδοση για το συνολικό έργο χωρίς τη χρήση της βελτίωσης),

ή εναλλακτικά :

Speedup = (χρόνος εκτέλεσης για το συνολικό έργο χωρίς τη χρήση της βελτίωσης) / (χρόνος εκτέλεσης για το συνολικό έργο με τη χρήση της βελτίωσης όπου είναι δυνατό).

Το speedup μας λέει πόσο πιο γρήγορα μπορεί να τρέξει ένα έργο χρησιμοποιώντας τη μηχανή με τη βελτίωση από την αρχική μηχανή.

Ο νόμος του Amdahl λέει ότι η βελτίωση της απόδοσης που κερδίζεται από τη χρήση μιας τεχνικής που τρέχει πιο γρήγορα περιορίζεται από το ποσοστό χρήσης της τεχνικής αυτής. Αυτό σημαίνει ότι η χρήση μιας τεχνικής επιτάχυνσης δεν επιφέρει την αναμενόμενη βελτίωση αν η χρήση της δεν είναι εκτεταμένη.

Παράδειγμα : έστω ότι μπορούμε να βελτιώσουμε την ταχύτητα της CPU στη μηχανή μας κατά ένα κλάσμα 5 (χωρίς να επηρεάζεται το I/O). Θεωρούμε ότι η CPU χρησιμοποιείται κατά το 40% του χρόνου και το υπόλοιπο παραμένει για I/O. Έτσι αν time_old είναι ο προηγούμενος χρόνος εκτέλεσης και time_new ο νέος, τότε θα έχουμε :

$Speedup_{overall} = Exec\_time_{old} / Exec\_time_{new} = 1 / [(1-fraction_{enhanced}) + fraction_{enhanced} / speedup_{enhanced}]$ και

$Exec\_time_{new} = Exec\_time_{old} * [(1-fraction_{en}) + fraction_{en} / speedup_{en}] = (1-0.4)*Exec\_time_{old} + 0.4/5 * Exec\_time_{old}$.

Άρα : speedup = 1 / (0.6 + 0.4/5) = 1.47 < 5

Μάθημα 4.

Περιγράψτε την τεχνική Very Long Instruction Word (VLIW). Ποιά τα πλεονεκτήματα και ποιά τα μειονεκτήματα της προσέγγισης αυτής έναντι της superscalar :

Στην τεχνική VLIW ο επεξεργαστής πολλαπλής έκδοσης (multiple-issue) παρέχει τη ροή εντολών σαφώς οργανωμένη για την αποφυγή των εξαρτήσεων (dependencies) και χρησιμοποιεί ευρείες εντολές (64 ή 128 bits ή περισσότερο) με πολλαπλές λειτουργίες για κάθε εντολή. Έτσι, ο VLIW επεξεργαστής εκδίδει ένα καθορισμένο αριθμό από εντολές οι οποίες φορμάρονται είτε σαν μια μεγάλη εντολή είτε σαν ένα σταθερό πακέτο εντολών, με τον παραλληλισμό ανάμεσα στις εντολές να υποδεικνύεται σαφώς από την εντολή. Από την άλλη, οι superscalar επεξεργαστές εκδίδουν μεταβλητούς αριθμούς εντολών σε κάθε κύκλο και είναι είτε στατικά είτε δυναμικά χρονοπρογραμματισμένοι.

Πλεονεκτήματα : ένας VLIW επεξεργαστής χρησιμοποιεί πολλαπλές και ανεξάρτητες συναρτησιακές μονάδες, βασίζεται στην τεχνολογία του μεταγλωττιστή (compiler) μέσω του οποίου μπορεί να ελαχιστοποιούνται τα ενδεχόμενα data hazard stalls. Επίσης, λόγω της οργάνωσης των εντολών που επιτυγχάνει, το hardware δε χρειάζεται να ελέγχει για τυχόν εξαρτήσεις γι' αυτό και είναι απλούστερο στην υλοποίηση ενόσω ο VLIW μπορεί να προσφέρει καλή απόδοση.

Μειονεκτήματα : απαιτείται επαναμεταγλώττιση των προγραμμάτων από διαφορετικές εκδόσεις από το hardware, χρησιμοποιούνται πιο πολύπλοκοι γενικοί αλγόριθμοι χρονοπρογραμματισμού, υπάρχουν τεχνικά προβλήματα όπως αύξηση στο μέγεθος του κώδικα και περιορισμοί της λειτουργίας lockstep. Η σχεδίαση καθίσταται πιο δύσκολη λόγω του προβλήματος της συμβατότητας με δυαδικό κώδικα (binary code compatibility).

Ποιά είναι τα επίπεδα ιεραρχίας μνήμης, ποιοί οι λόγοι που επιβάλλουν τη χρήση ιεραρχικής μνήμης σε ένα υπολογιστικό σύστημα και πώς εξηγείται η σωστή λειτουργία της. Δώστε τη μαθηματική έκφραση που περιγράφει το μέσο χρόνο προσπέλασης σε ιεραρχία μνήμης n επιπέδων ;

Τα επίπεδα ιεραρχίας της μνήμης είναι 4 : οι καταχωρητές που βρίσκονται μέσα στη CPU, η κρυφή μνήμη που βρίσκεται κοντά στη CPU και μέσω αυτής ο επεξεργαστής επικοινωνεί με τον έξω κόσμο, η κύρια μνήμη του συστήματος όπου είναι αποθηκευμένο το πρόγραμμα και τα δεδομένα, η δευτερεύουσα μνήμη του συστήματος που μπορεί να είναι δίσκος ή άλλα μέσα. Στη δευτερεύουσα μνήμη τα δεδομένα αποθηκεύονται με σκοπό τη διατήρησή τους και μετά την παύση της λειτουργίας του συστήματος.

Οι 2 κύριοι λόγοι είναι ότι : (1) η μνήμη πρέπει να είναι τόσο γρήγορη όσο και η CPU ώστε η CPU να μην καθυστερεί κατά την επικοινωνία τους. Μνήμες που λειτουργούν σε ταχύτητες συγκρίσιμες της CPU είναι πολύ ακριβές. (2) η αρχή της τοπικότητας των αναφορών. Η τοπικότητα στο χρόνο (αν σε ένα δεδομένο γίνει αναφορά τείνει να του ξαναγίνει σύντομα) και η τοπικότητα στο χώρο (αν σε ένα δεδομένο γίνει αναφορά τείνει να γίνει αναφορά σε κοντινά του δεδομένα).

Η τεχνολογία και η τοπικότητα είναι οι λόγοι που επιβάλλουν τη χρήση ιεραρχικής μνήμης, που αποτελείται από διαφορετικές διατάξεις με διαφορετικούς λόγους κόστους / απόδοσης. Στόχος είναι η βελτίωση της απόδοσης και η μείωση του κόστους υλοποίησης. Οι σημερινές απαιτήσεις μας οδηγούν στη χρήση όλο και μεγαλύτερων ποσοτήτων μνήμης. Αυτό σημαίνει χρήση μεγάλων και άρα αργών μνημών. Η χρήση ιεραρχικής μνήμης και η κατανομή των δεδομένων ανάλογα με τη χρησιμότητά τους στα κατάλληλα επίπεδα της ιεραρχίας, οδηγεί σε βελτίωση της απόδοσης. Καθώς οι επιδόσεις των CPU αυξάνουν, απαιτείται γρήγορη επικοινωνία με τα δεδομένα, που συνεπάγεται χρήση γρήγορης μνήμης. Όμως χρήση τέτοιας μνήμης οδηγεί σε διόγκωση του κόστους λόγω της χρήσης μεγάλων ποσοτήτων. Με την ιεραρχική μνήμη χρησιμοποιείται η κατάλληλη ποσότητα γρήγορης μνήμης που οδηγεί σε αποδεκτό κόστος.

Η ιεραρχική μνήμη εκμεταλλεύεται την τοπικότητα των δεδομένων είτε αυτή είναι χρονική είτε χωρική. Συγκεκριμένα, αν ένα δεδομένο χρησιμοποιηθεί, τότε μπορεί να χρησιμοποιηθεί σύντομα στο μέλλον και άρα διατηρείται κοντά στον τόπο χρήσης του. Επίσης, η μεταφορά ενός δεδομένου για χρήση συνεπάγεται και ταυτόχρονη μεταφορά άλλων, καθώς η μεταφορά γίνεται κατά blocks. Έτσι μέσα στο block μπορεί να υπάρχουν

δεδομένα που πρόκειται να χρησιμοποιηθούν σύντομα. Η ιεραρχία έχει υλοποιηθεί με τέτοιο τρόπο που συχνά χρησιμοποιούμενα δεδομένα βρίσκονται πιο κοντά στον τόπο χρήσης τους, μειώνοντας έτσι τις καθυστερήσεις που συνεπάγονται από τη μεταφορά. Βασικά, η ιεραρχία μνήμης αποτελεί εφαρμογή του κανόνα : κάνε την πιο συχνά εμφανιζόμενη περίπτωση γρήγορη.

Ονομάστε τα 4 επίπεδα στην ιεραρχία της μνήμης και δώστε το τυπικό μέγεθος, χρόνο προσπέλασης και bandwidth; 1196-9196

Τύπος	**Καταχωρητές**	**Κρυφή μνήμη**	**Κύρια μνήμη**	**Δίσκοι**
Μέγεθος	**< 1KB**	**< 4MB**	**< 4GB**	**> 1GB**
Χρόνος προσπέλασης	**2-5ns**	**3-10ns**	**80-400**	**5ms**
Bandwidth MB/sec	**4000-32000**	**800-5000**	**400-2000**	**4-32**
Managed by	**Compiler**	**H/W**	**Op sys**	**Op sys / user**
Backed by	**Cache**	**Main memory**	**Disk**	**Tape**

Μπορούν ποτέ να καταργηθούν τα ιεραρχικά επίπεδα και να υπάρχει μόνο ένα;
Τα ιεραρχικά επίπεδα μνήμης δεν μπορούν να καταργηθούν όσο υπάρχει μεγάλη διαφορά μεταξύ των ταχνολογιών μνήμης και μικροεπεξεργαστών. Πρός το παρόν η τεχνολογία που αφορά μνήμες με επιδόσεις συγκρίσιμες με τον επεξεργαστή, παραμένει πολύ ακριβή για να χρησιμοποιείται σε μεγάλες ποσότητες. Επίσης η επιθυμία μόνιμης αποθήκευσης δεδομένων αποτελεί εμπόδιο για την κατάργηση της ιεραρχίας, αν και έχουν γίνει βήματα πρός τη μεριά της επίτευξης non_volatility σε αυτές τις μνήμες. Όμως παραμένει το πρόβλημα του μεγέθους που πρέπει να έχει η μνήμη αυτή. Έτσι μάλλον είναι αδύνατο να καταργηθούν όλα τα ιεραρχικά επίπεδα και να μείνει μόνο ένα.

Ποιός είναι ο καλύτερος τρόπος μέτρησης της απόδοσης ενός υπολογιστικού συστήματος ; 2|95
Ο καλύτερος τρόπος μέτρησης της απόδοσης ενός υπολογιστικού συστήματος είναι η μέτρηση του χρόνου εκτέλεσης αληθινών προγραμμάτων. Ο χρόνος εκτέλεσης αποτελεί το χρονικό διάστημα που μεσολαβεί από την αρχή μέχρι το τέλος της εκτέλεσης και μπορεί να οριστεί με διάφορους τρόπους ανάλογα με το τί μετράμε. Θεωρείται ως το καλύτερο μέτρο απόδοσης καθώς λαμβάνει όλα τα επιμέρους στοιχεία του συστήματος όπως CPU, ιεραρχία μνήμης, είσοδος / έξοδος, λειτουργικό σύστημα. Ένα εναλλακτικό του χρόνου σαν μετρική είναι και το MIPS. Επειδή το MIPS είναι ο ρυθμός των λειτουργιών ανα μονάδα χρόνου, η απόδοση μπορεί να καθοριστεί σαν το αντίστροφο του execution time, με γρηγορότερες μηχανές να έχουν υψηλότερο MIPS. Το MIPS είναι κατανοητό, ειδικά για τον καταναλωτή, αφού γρηγορότερες μηχανές σημαίνουν μεγαλύτερο MIPS. Βέβαια το MIPS έχει πρόβλημα όταν χρησιμοποιείται στη σύγκριση προγραμμάτων. Είναι δύσκολο να συγκριθούν μηχανές με διαφορετικά Instruction Sets αφού το MIPS εξαρτάται άμεσα από το χρησιμοποιούμενο IS. Τα MIPS είναι δυνατόν να διαφέρουν από πρόγραμμα σε πρόγραμμα στον ίδιο H/Y. Επιπλέον, τα MIPS μπορεί να είναι αντίθετα με την απόδοση, π.χ. αν μια μηχανή χρησιμοποιεί H/W για FP λειτουργίες και μια άλλη S/W τότε η 1η θα έχει λιγότερα MIPS αλλά προφανώς πολύ καλύτερη απόδοση.

Ποιές είναι οι διαφορετικές κατηγορίες benchmark προγραμμάτων και πώς επιλέγουμε προγράμματα για τη μέτρηση της απόδοσης Η/Υ; 1996 - 9796

Επιλέγουμε προγράμματα ανάλογα με το τμήμα του Η/Υ που θέλουμε να μετρήσουμε την απόδοση (π.χ μεμονωμένα χαρακτηριστικά Η/Υ), το επίπεδο σχεδιασμού που βρισκόμαστε και ανάλογα με το computer που έχουμε. Βασικές κατηγορίες :

- Real programs : είναι τα προγράμματα που χρησιμοποιούν οι χρήστες για να λύνουν τα προβλήματα τους. Π.χ. Compilers, Text processing, CAD. Είναι προγράμματα με Input / Output και λειτουργίες που ο χρήστης μπορεί να εκτελέσει.
- Modified applications : τροποποίηση με σκοπό να αυξήσουν την portability και να εστιάσουν σε μια όψη της απόδοσης.
- Kernels : μικρά κομμάτια από real programs. Είναι προγράμματα μέτρησης της απόδοσης ατομικών χαρακτηριστικών των μηχανών. Μετρώντας αυτά, προσπαθούν να εξηγήσουν τις διαφορές στην απόδοση των real programs.
- Toy Benchmarks : είναι 10-100 γραμμές κώδικα και παράγουν αποτέλεσμα που ο χρήστης ήδη το γνωρίζει. Τρέχουν σχεδόν σε όλους τους υπολογιστές. Η καλύτερη χρήση των προγραμμάτων αυτών είναι ξεκινώντας programming assignments. (puzzle, quicksort)
- Synthetic Benchmarks : η φιλοσοφία τους είναι παρόμοια με αυτή των Kernels και προσπαθούν να ταιριάξουν τη μέση συχνότητα λειτουργιών και τελεστών σε ένα μεγάλο σύνολο προγραμμάτων. Δεν υπολογίζουν κάτι που μπορεί να δεί ο user. Δεν είναι κομμάτια από real programs αλλά δημιουργούνται ξεχωριστά. Εξομοιώνουν γενική συμπεριφορά προγραμμάτων.

Πώς μετριέται η απόδοση και το κόστος ; 9195

Μέτρηση απόδοσης : ο χρόνος εκτέλεσης είναι το μέτρο απόδοσης ενός Η/Υ. Η απόδοση μετράται σαν το πηλίκο κάποιου αριθμού από γεγονότα ανα sec (MIPS). Άρα μικρότερος χρόνος σημαίνει μεγαλύτερη απόδοση. Ο Η/Υ που εκτελεί το ίδιο ποσό δουλειάς στο λιγότερο χρόνο είναι πιο αποδοτικός.

Μέτρηση κόστους : λόγω της ταχείας αλλαγής στο κόστος των Η/Υ, οι σχεδιαστές πρέπει να εκτιμούν το κόστος στις τιμές που θα ισχύουν όταν ολοκληρώνεται το προϊόν. Γι' αυτό χρησιμοποιούνται learning curves που βοηθούν στη μείωση του κόστους. Επίσης, λαμβάνεται υπόψη και η επίδραση της συσκευασίας στο συνολικό κόστος (μικρότερες κάρτες, μικρότερο κουτί, κλπ.). Η συγκέντρωση όλο και περισσότερων λειτουργιών σε ένα chip μειώνει το κόστος.

Τί σημαίνει ο όρος cost / performance στο σχεδιασμό ενός Η/Υ ; 9/94 - 9/95 - 1/96 -

Το κόστος και η απόδοση αποτελούν τη βάση για την απόφαση αγοράς των Η/Υ. Έτσι οι σχεδιαστές Η/Υ πρέπει να καταλάβουν καλά τόσο το κόστος όσο και την απόδοση, για να μπορέσουν να σχεδιάσουν ένα ανταγωνιστικό προϊόν. Στην cost / performance σχεδίαση, ο σχεδιαστής προσπαθεί πάντα να βρεί την καλύτερη ισορροπία μεταξύ του κόστους και της απόδοσης, πράγμα που είναι και η τέχνη του computer design. Στο high-performance design το κόστος δεν λαμβάνεται υπόψη στην επίτευξη του στόχου. Στη low cost design η απόδοση θυσιάζεται για να επιτευχθεί χαμηλό κόστος.

Πώς κατανέμεται το κόστος ενός σταθμού εργασίας σε ποσοστά ανάλογα με τα εξαρτήματα που περιέχει : 9/95

system	**subsystem**	**Fraction of total (%)**
Cabinet	Sheet metal, plastic	1
	Power supply, fans	2
	Cables, nuts, bolts	1
	Shipping box, manuals	0
	subtotal	4
Processor board	Processor	6
	DRAM 64M	36
	Video system	14
	I/O system	3
	Printed circuit Board	1
	subtotal	60
I/O devices	Keyboard, mouse	1
	Monitor	22
	Hard disk 1M	7
	DAT drive	6
	subtotal	36

Πώς μετριέται η απόδοση του I/O σε ένα σύστημα Η/Υ; 9/96-

Η μέτρηση της απόδοσης του I/O εκτός από τα response time και throughput χρησιμοποιεί και πιο εξειδικευμένους όρους. Η ποικιλία (diversity) είναι ένας από αυτούς και αφορά το ποιές συσκευές μπορούν να συνδεθούν στο σύστημα και η χωρητικότητα (capacity) που αφορά το πόσες συσκευές μπορούν να συνδεθούν. Ο χρόνος απόκρισης (response time) ορίζεται ως ο χρόνος από τη στιγμή που μια διεργασία είναι έτοιμη για επεξεργασία μέχρι όταν ολοκληρωθεί η εργασία της. Το throughput είναι ο μέσος ρυθμός ολοκλήρωσης εργασιών σε ένα χρονικό διάστημα. Τέλος ένας ακόμη τρόπος μέτρησης είναι η παρεμβολή του I/O με την εκτέλεση στη CPU. Η μεταφορά δεδομένων μπορεί να παρεμβαίνει στην εκτέλεση μιας άλλης διεργασίας. Εδώ θέλουμε να ξέρουμε πόσα clock cycle παραπάνω θα χρειαστούν, εξαιτίας κάποιας άλλης διεργασίας.

Ποιός είναι ο ορισμός του MIPS και πώς υπολογίζουμε το χρόνο εκτέλεσης ενός προγράμματος σε σχέση με τα MIPS. Τί πρόβλημα έχει η χρήση των MIPS για σύγκριση Η/Υ συστημάτων; 9/96- 9/94 - 9/95 - 1/96

Το MIPS σημαίνει εκατομμύρια εντολές το δευτερόλεπτο και αφορά το ρυθμό εκτέλεσης των εντολών σε ένα υπολογιστικό σύστημα. Για δεδομένο πρόγραμμα, το MIPS δίνεται από τη σχέση :

MIPS = Instruction_count / Exec_time*10^6 .

Από αυτή τη σχέση ο χρόνος εκτέλεσης ενός προγράμματος είναι το πηλίκο του αριθμού των εντολών με τα MIPS, δηλαδή : Exec_time = Instuction_count / MIPS*10^6.

Τα MIPS εξαρτώνται από το σύνολο εντολών μιας συγκεκριμένης μηχανής, κάνοντάς τα δύσκολο να συγκρίνουν MIPS από υπολογιστές με διαφορετικά σύνολα εντολών. Τα MIPS επίσης διαφέρουν μεταξύ προγραμμάτων του ίδιου υπολογιστή. Τέλος, τα MIPS μπορεί να διαφέρουν αντιστρόφως με την απόδοση (πιο σημαντικό).

Πώς ορίζεται ο όρος CPI και τί σημαίνει ;
Το CPI είναι μέσος αριθμός των κύκλων ρολογιού που απαιτούνται για την ολοκλήρωση μιας εντολής.
Δηλαδή : **CPI = CPU clock cycles for a program / Instruction_count**, όπου IC είναι ο αριθμός των instructions που εκτελούνται. Πολλές φορές είναι χρήσιμο να υπολογίσουμε τον αριθμό των ολικών CPU clock cycles ως εξής : CPU clock cycles = $\sum_{1}^{n} CPI * ICi$. Το IC_i αναπαριστά τον αριθμό των φορών που η εντολή i εκτελείται στο πρόγραμμα και το CPI_i αναπαριστά το μέσο αριθμό των clock cycles για την εντολή i. Οπότε :
CPU time = $(\sum_{1}^{n} CPI * ICi)$ * clock_cycle_time και το ολικό CPI είναι : **CPI = $(\sum_{1}^{n} CPI * ICi)$ / IC**

Γιατί όταν αυξηθεί ο αριθμός των σταδίων pipeline, αυξάνεται η απόδοση του επεξεργαστή ;
Όταν αυξηθεί ο αριθμός των σταδίων pipeline, αυξάνεται η απόδοση του επεξεργαστή καθώς μικραίνει το clock cycle, με την προϋπόθεση ότι κάθε στάδιο έχει όμοιο φόρτο εργασίας, έτσι ώστε το ρολόϊ του συστήματος να είναι το μικρότερο δυνατό. Με τον τρόπο αυτό πετυχαίνουμε, στην ιδανική περίπτωση, εκτέλεση μιας εντολής ανα κύκλο ρολογιού. Στην πραγματικότητα η αύξηση της απόδοσης επιβαρύνεται από stall που μπορεί να προκύψει στο pipeline.

Τί σημαίνει pipeline, superpipeline & superscalar; Δώστε σχηματικά τα διαγράμματα σε κάθε περίπτωση. 9|94 -1|96 - 9|96
Η pipeline τεχνική εκμεταλλεύεται το γεγονός ότι κάθε εντολή αποτελείται από διάφορα κομμάτια τα οποία είναι ανεξάρτητα μεταξύ τους και άρα μπορεί η εκτέλεσή τους να αλληλοκαλύπτεται. Έτσι η εκτέλεση της εντολής χωρίζεται σε αυτά τα κομμάτια και η οργάνωση της CPU χωρίζεται σε στάδια τα οποία αναλαμβάνουν την εκτέλεση αυτών των κομματιών. Η αλληλοκάλυψη δεν αφορά την ίδια την εντολή αλλά διαφορετικές. Μόλις η εκτέλεση ενός κομματιού τελειώσει, τα αποτελέσματα προωθούνται στο επόμενο στάδιο και το τρέχον περιλαμβάνει του προηγούμενου. Π.χ., στην DLX έχουμε :

I1	IM	ID	EX	MEM	WB		
I2		IM	ID	EX	MEM	WB	
I3			IM	ID	EX	MEM	WB

Σε κάθε κύκλο ρολογιού μια άλλη εντολή προσκομίζεται και ξεκινά την 5-σταδίων εκτέλεσή της. Άν μια εντολή ξεκινά σε κάθε κύκλο ρολογιού, τότε η απόδοση είναι 5 φορές μεγαλύτερη από μια μηχανή μη pipeline.
Η superpipeline είναι περαιτέρω αποσύνθεση διαφόρων σταδίων του pipeline σε μικρότερα. Ένα παράδειγμα είναι το pipeline του R4000 όπου τα IM και MEM έχουν χωριστεί σε 2 και 3 στάδια αντίστοιχα.
Τέλος, το superscalar είναι η τεχνική στην οποία ο επεξεργαστής εκδίδει περισσότερες της μιας εντολές ανά κύκλο ρολογιού. Ο αριθμός των εντολών δεν είναι συγκεκριμένος αλλά ποικίλλει.

Γιατί χρειάζονται περισσότερα από ένα επίπεδα cache σε ένα σύστημα ; 9|95
Σε ένα σύστημα χρειάζονται περισσότερα από ένα επίπεδα cache για να μειωθεί το miss penalty. Επειδή οι CPUs γίνονται γρηγορότερες και η κύρια μνήμη μεγαλύτερη αλλά αργότερη από τη CPU, θα πρέπει είτε να γίνει η cache γρηγορότερη για να συμβαδίζει με τη CPU, είτε μεγαλύτερη για να ξεπεράσει το χάσμα εύρους ανάμεσα στη CPU και την κύρια μνήμη. Επειδή πρέπει να γίνουν και τα 2, χρειαζόμαστε να προσθέσουμε άλλο ένα επίπεδο cache ανάμεσα στο αρχικό και στην κύρια μνήμη. Η 1ου επιπέδου cache θα είναι αρκετά μικρή για να ταιριάζει με τον κύκλο ρολογιού της CPU ενώ η 2ου επιπέδου θα είναι αρκετά μεγάλη ώστε να «πιάνει» πολλές προσπελάσεις που θα πάνε στην κύρια μνήμη.

Τί είναι block size και τί associativity σε μια cache ; 9|95-

Το block size είναι το μέγεθος της μικρότερης μονάδας πληροφορίας που μπορεί να είναι παρούσα στην cache (υποπολλαπλάσιο του μεγέθους της cache). Τυπική τιμή είναι 16-128 bytes. Associativity είναι ο τρόπος τοποθέτησης του block στην cache. Άν αυτό μπορεί να τοποθετηθεί οπουδήποτε στην cache τότε λέμε ότι η cache αυτή είναι fully associative. Άν το block μπορεί να τοποθετηθεί μόνο σε ένα συγκεκριμένο block της cache τότε αυτή είναι direct mapped. Άν το block μπορεί να τοποθετηθεί οπουδήποτε σε ένα συγκεκριμένο set από blocks της cache, τότε λέμε ότι αυτή είναι set associative.

1|96 - 9|96 -

9|97 **Ποιές είναι οι βασικές παράμετροι που καθορίζουν μια cache; Δώστε τυπικές τιμές για κάθε παράμετρο** ✓

Οι βασικές παράμετροι που χαρακτηρίζουν μια cache είναι το μέγεθος του block (16-128 bytes), το hit time (1-2 cycles), το miss penalty (8-100 cycles), το miss rate (0,5-10%) και τέλος το μέγεθος της ίδιας της cache (0,016-1MB)

Ποιοί είναι οι διαφορετικοί τρόποι μέτρησης της απόδοσης της cache; 2|98

Ο CPU χρόνος χωρίζεται στους κύκλους του ρολογιού που ξοδεύει η CPU εκτελώντας το πρόγραμμα και σ' αυτούς που η CPU περιμένει το σύστημα μνήμης. Οι διαφορετικοί τρόποι μέτρησης της cache είναι :

- Ο νόμος του Amdahl (απλή εφαρμογή του)
  Speedup = Performance for entire task using enhancements / Performance for entire task without using enhancements
- Το miss rate που μας δίνει μια ιδέα για το πόσο καλά συμπεριφέρεται η cache στις προσπελάσεις.
- Ο μέσος χρόνος προσπέλασης μνήμης. Συνδέεται με το miss rate και μας δείχνει τί χάνουμε όταν εμφανίζεται ένα miss. Ισχύει η σχέση :
  Average memory access time = HitTime + MissRate * MissPenalty
  Αυτή η σχέση συνδέει τα διάφορα στοιχεία της ιεραρχίας μνήμης και μας δίνει μια σφαιρικότερη άποψη της απόδοσης της cache.
- CPU time = (CPU exec_clock cycles + Mem_stall_clock cycles) * Clock_cycle time
  Για τη μέτρηση της απόδοσης της cache οι σχεδιαστές υποθέτουν ότι όλες οι memory stalls οφείλονται στην cache. Αυτό μερικές φορές ισχύει και μερικές όχι, γι' αυτό για τη μέτρηση της τελικής απόδοσης πρέπει να μετράμε όλα τα memory stalls. Οι hit clock cycles συμπεριλαμβάνονται στο CPU exec_clock cycles.
  Mem_stall_clock cycles = Mem_accesses * MissRate * MissPenalty. Άρα :
  CPU time = IC * [CPI_{exec} + (Mem_accesses / Instuction)* MissRate * MissPenalty] * Clock_cycle time
  Αυτή η σχέση δείχνει την επίδραση της cache με βάση τη χρονική εκτέλεση ενός προγράμματος. Έτσι μπορεί να αφορά διαφορετικά προγράμματα με διαφορετικές παραμέτρους.
- Μερικοί προτιμούν να μετρούν το misses per instruction αντί των misses σε αναφορά μνήμης :
  Misses / Instruction = (Mem_accesses / Instruction) * MissRate
  Το πλεονέκτημα της μεθόδου αυτής είναι ότι είναι ανεξάρτητη της H/W υλοποίησης. Το μειονέκτημα είναι ότι εξαρτάται από την αρχιτεκτονική (είναι διαδεδομένη στους σχεδιαστές που δουλεύουν με μια μόνο οικογένεια υπολογιστών).

Σε ποιές περιπτώσεις η cache δεν βοηθάει την απόδοση ενός H/Y;

Όταν η κρυφή μνήμη είναι πολύ μικρή, οπότε χρειάζονται πολλές μεταφορές σελίδων. Έτσι, η απόδοση του υπολογιστή είναι μικρότερη από έναν ίδιο H/Y χωρίς cache.

Τί σημαίνει normalized time και πώς ορίζεται; 9/94 -9/95-11/96-9/96

Τρόπος μέτρησης της απόδοσης για χρήση άνισων ομάδων προγραμμάτων. Γίνεται κανονικοποίηση του execution time για κάθε πρόγραμμα ως πρός την αναφερόμενη μηχανή και μετά υπολογίζεται η γεωμετρική μέση τιμή αυτών των τιμών σύμφωνα με τον τύπο :

Normalized time = ($\Pi_{i = 1,\ldots,n}$ Exec_time_ratio$_i$)$^{1/n}$, όπου Exec_time_ratio$_i$ είναι ο χρόνος εκτέλεσης του i προγράμματος (από n συνολικά) κανονικοποιημένος σε σχέση με τη μηχανή αναφοράς. Γι' αυτό το λόγο δεν μπορούμε να χρησιμοποιήσουμε αριθμητικό μέσο για τον υπολογισμό του normalized time.

Ποιές είναι οι διαφορετικές κατηγορίες αρχιτεκτονικής Instruction set και τα χαρακτηριστικά τους ; 1/96

	Πλεονεκτήματα	Μειονεκτήματα
Stack (push and pop onto or from stack)	Είναι απλό μοντέλο εκτίμησης έκφρασης (reverse polish). Μικρές εντολές μπορούν να παράγουν πολύ καλό κώδικα (πυκνότητα)	Δεν υπάρχει random access. Δυσκολία στην παραγωγή αποδοτικού κώδικα. Υπάρχει δυσκολία στην αποδοτική λειτουργία λόγω συνωστισμού
Accumulator (L/S accumulator)	Ελαχιστοποιεί την εσωτερική κατάσταση της μηχανής. Μικρές εντολές.	Αφού ο acc είναι μόνο προσωρινή αποθήκευση έχουμε μεγαλύτερο memory traffic
Register (L/S of registers or memory)	Τα περισσότερα γενικά μοντέλα για παραγωγή κώδικα	Όλα τα operands πρέπει να έχουν όνομα => μεγαλύτερες εντολές.

Η τελευταία περίπτωση (general purpose register architecture), η οποία τείνει να επικρατήσει γιατί οι καταχωρητές είναι γρηγορότεροι και πιο εύκολοι στη χρήση από τους compilers, χωρίζεται σε 3 κατηγορίες : register – memory, register – register, memory – memory (κρατά όλα τα operands στην κύρια μνήμη – έχει καταργηθεί).

Ποιά τα πλεονεκτήματα και μειονεκτήματα των I-set που βασίζονται σε Mem-Mem λειτουργία ; 1/96

Ένα από τα πλεονεκτήματα των I-set είναι ότι παράγουν συμπιεσμένο κώδικα. Επίσης δε σπαταλούν τους καταχωρητές χρησιμοποιώντας τους για αποθήκευση προσωρινών τιμών. Έτσι βοηθούν το έργο των μεταφραστών. Τέλος το instruction set έχει καλή πυκνότητα.

Τα μειονεκτήματα είναι οι μεγάλες διακυμάνσεις στο μέγεθος των εντολών και ιδιαίτερα για εντολές με τρία έντελα, οι μεγάλες διακυμάνσεις στην εργασία που απαιτείται για κάθε εντολή και οι πολλές αναφορές στη μνήμη που προκαλούν συμφόρηση, με αποτέλεσμα φτωχή απόδοση του προγράμματος.

Ποιός είναι ο τρόπος αξιολόγησης μιας συγκεκριμένης αρχιτεκτονικής; Ποιά είναι τα μετρήσιμα μεγέθη και σε τί μονάδες μετριούνται ; 9/94

Τρείς ποσότητες έχουν επινοηθεί για την εξέταση των αρχιτεκτονικών :

- S – ο αριθμός των bytes για τον κώδικα του προγράμματος
- M – ο αριθμός μεταφερόμενων bytes μεταξύ της μνήμης και της CPU, κατά την εκτέλεση του προγράμματος, για κώδικα και δεδομένα (το S μετράει το μέγεθος του κώδικα κατά τον compile time, ενώ το M είναι memory traffic κατά τη διάρκεια της εκτέλεσης του προγράμματος).
- R – ο αριθμός των μεταφερόμενων bytes μεταξύ των registers σε ένα απλουστευμένο (canonical) μοντέλο για μια CPU.

Μόλις μετρηθούν οι παραπάνω ποσότητες, ένας αυξητικός παράγοντας εφαρμόζεται σ' αυτές για να καθορίσει ποιά αρχιτεκτονική ήταν καλύτερη.

CACHE MISSES

Κατηγορίες των cache misses :

- compulsory : στην 1η προσπέλαση ένα μπλόκ δεν είναι ποτέ στην cache, επομένως το μπλόκ θα πρέπει να το φέρουμε στην cache (cold start misses ή first reference misses)
- capacity : αν η cache δεν περιέχει όλα τα μπλόκ που χρειάζονται κατά τη διάρκεια της εκτέλεσης ενός προγράμματος, θα συμβούν capacity misses επειδή ένα μπλόκ μπορεί απορριφθεί και να επανέλθει αργότερα.
- conflict : σε set associative ή direct mapped caches, επειδή αντικαθίστανται μπλόκ όταν άλλα διαφορετικά μεταφέρονται στον ίδιο χώρο.

Τεχνικές μείωσης των cache misses : επιτυγχάνουμε μείωση των cache misses μέσω μείωσης του miss rate, μείωσης του miss penalty, μείωσης και των 2 μέσω παραλληλισμού και μέσω μείωσης του χρόνου για την επίτευξη hit time στην cache. Πιο συγκεκριμένα έχουμε :

1. **Τεχνικές μείωσης miss rate**
   - Larger block size : μειώνει τα υποχρεωτικά (compulsory) misses λόγω spatial τοπικότητας. Επίσης, για μικρό cache size, αυξάνονται τα conflict και capacity misses, γιατί μειώνεται ο αριθμός των blocks στην cache. Αυξάνεται το miss penalty, αφού η μεταφορά μεγαλύτερων blocks απαιτεί πολλαπλάσιο χρόνο.
   - Larger caches : για να μειώσουμε τα capacity misses μπορούμε να αυξήσουμε τη χωρητικότητα της cache (αυξηση hit time, περισσότερο κόστος, χρησιμοποιείται σε off-chip caches)
   - Higher associativity : η eight way associative cache είναι εξίσου αποτελεσματική με τη fully associative. Κανόνας 2:1 για caches : η direct mapped cache μεγέθους N έχει ίδιο miss rate με μια two way set associative μεγέθους N/2.
   - Way prediction and Pseudoassociative caches : extra bits φυλάγονται στην cache για να προβλέπουν το set (way) ή το μπλόκ του επόμενου cache access. Επίσης, με διαίρεση cache σε κάθε miss, γίνεται έλεγχος του άλλου μισού της cache αν είναι εκεί. Στην περίπτωση που είναι έχουμε pseudo-hit (slow hit) – (καλύτερο για L2 caches)
   - Compiler optimizations : για εντολές, έχουμε αναδιάταξη procedures στη μνήμη για μείωση των conflict misses και profiling έλεγχο για conflicts. Για δεδομένα έχουμε merging arrays, loop interchange, loop fusion και blocking.
2. **Τεχνικές μείωσης miss penalty**
   - Multilevel caches
   - Critical word first and early restart : ζητείται από τη μνήμη πρώτα το word που απαιτείται, και μόλις φτάσει στέλνεται στη CPU – ενώ συνεχίζεται η φόρτωση των υπόλοιπων words του μπλόκ. Για το early restart, μόλις φτάσει το word που ζητήθηκε από το μπλόκ, στέλνεται στη CPU.
   - Priority to Read Misses over Writes
   - Merging Write Buffer : αν ο buffer περιέχει κάποια τροποποιημένα μπλόκς, ελέγχεται η διεύθυνση των νέων δεδομένων αν συμπίπτει με αυτή των έγκυρων περιεχομένων του buffer. Αν συμπίπτουν, τα νέα δεδομένα συγχωνεύονται με τα υπάρχοντα.
   - Victim Caches : προσθήκη buffer όπου τοποθετούνται data που αντικαθίστανται από την cache.
3. **Τεχνικές μείωσης μέσω παραλληλισμού :** Non-blocking cache ή lockup-free cache για μείωση stalls σε misses, HW prefetching για instructions and data, Compiler – controlled prefetching.
4. **Τεχνικές μείωσης hit time :** Small and simple Caches, Avoiding Address Translation during Indexing of the Cache, Pipelined Cache Access, Trace Caches.

Τρόποι βελτίωσης της κύριας μνήμης : Wider main memory, Simple Interleaved Memory, Independent Memory Banks.

Γιατί χρησιμοποιούνται ιεραρχίες μνήμης με πολλά επίπεδα : τα προγράμματα απαιτούν όλο και περισσότερη και πιο γρήγορη μνήμη. Όμως η γρήγορη και μεγάλη μνήμη είναι πιο ακριβή. Αυτό οδηγεί σε ιεραρχία μνήμης. Η ιεραρχία αυτή βασίζεται σε temporal και spatial locality.

Data & instruction caches έναντι unified cache : αν σπάσουμε μια unified cache σε δύο κομμάτια (data και instruction) τότε η σπασμένη cache προσφέρει δύο ports μνήμης σε κάθε κύκλο ρολογιού, οπότε αποφεύγει τα structural hazards. Επίσης, έχει καλύτερο μέσο χρόνο προσπέλασης στην κύρια μνήμη από ότι έχει η μονού port unified cache. Παρόλα αυτά, έχει χειρότερο miss rate.
Συνήθως, μια unified cache έχει μικρότερο miss rate από μια data – instruction cache. Ξέρουμε ότι miss rate = [(misses/1000 instructions) / 1000] / (mkemory accesses / instruction). Από τον τύπο και από το γεγονός ότι οι περισσότερες προσπελάσεις μνήμης αφορούν αναφορές σε εντολές, μια unified cache θα έχει μικρότερο miss rate.

Τί σημαίνει για τη λειτουργία ενός pipeline η εμφάνιση hazards ; Πόσα είδη hazards εμφανίζονται και πώς αντιμετωπίζεται το καθένα από αυτά ;
Υπάρχουν καταστάσεις, που ονομάζονται hazards (εμπόδια), οι οποίες εμποδίζουν την εκτέλεση της επόμενης σε σειρά εντολής, στην ακολουθία εντολών ενός προγράμματος, κατά τη διάρκεια του προσχεδιασμένου κύκλου ρολογιού της. Τα hazards μειώνουν την απόδοση του ιδανικού speedup που θα κερδίζαμε μέσω του pipeline. Υπάρχουν 3 κατηγορίες hazards :

- **δομικά (structural) hazards :** εμφανίζονται λόγω resource conflicts (σύγκρουση στα δεδομένα εισόδου) όταν το hardware δεν μπορεί να υποστηρίξει όλους τους πιθανούς συνδυασμούς εντολών ταυτόχρονα σε επικαλυπτόμενη εκτέλεση.
- **hazards δεδομένων :** εμφανίζονται όταν μια εντολή εξαρτάται από το αποτέλεσμα μιας προηγούμενης, με έναν τρόπο που καθορίζεται από την επικάλυψη των εντολών στο pipeline. Τρόποι μείωσης : forwarding (1. ειδικό hardware περνάει τα αποτελέσματα της ALU πίσω στην είσοδό της, 2. δεν επιτρέπει τη χρήση παλιών τιμών). Ταξινόμηση data hazards : i,j εντολές όπου η i προηγείται της j
  1. RAW(read after write) : η j διαβάζει μεταβλητή πρίν τη γράψει η i
  2. WAW(write after write) : η j γράφει μεταβλητή πρίν τη γράψει η i
  3. WAR(write after read) : η j γράφει μεταβλητή πρίν τη διαβάσει η I
- **hazards ελέγχου :** εμφανίζονται στις διακλαδώσεις και σε άλλες εντολές που αλλάζουν τον PC (branch, jump). Τρόποι μείωσης των χαμένων κύκλων από branch : flush, predict-not-taken, predict-taken, delayed branch.

Τα hazards στα pipelines μπορούν να προκαλέσουν stall στο pipeline. Για την αποφυγή ενός hazard συχνά απαιτείται μερικές εντολές στο pipeline να τους επιτραπεί να συνεχίσουν ενόσω άλλες καθυστερούνται.

Ποιά είναι η διαφορά της write-back από την write-trough πολιτική εγγραφής στην cache; Ποιό πρόβλημα λύνει στη σχεδίαση του συστήματος μνήμης η προσθήκη write buffer :

Οι πολιτικές εγγραφής στην cache διαχωρίζουν τον εκάστοτε σχεδιασμό της cache. Υπάρχουν 2 βασικές επιλογές όταν γράφουμε στην cache :

Write-through : η πληροφορία γράφεται και στο block της cache και στο block της μνήμης χαμηλότερου επιπέδου

Write-back : η πληροφορία γράφεται μόνο στο block της cache. Το τροποποιημένο block της cache γράφεται στην κύρια μνήμη μόνο όταν αυτό αντικατασταθεί.

Όταν η CPU πρέπει να περιμένει για την ολοκλήρωση των εγγραφών κατά τη διάρκεια του write-trough, αυτή βρίσκεται σε κατάσταση write stall. Μια κοινή βελτιστοποίηση για τη μείωση των write stalls είναι ένας write buffer, ο οποίος επιτρέπει στη CPU να συνεχίσει αφού τα δεδομένα γράφονται στο buffer, και ως εκ τούτου να συμπίπτει η εκτέλεση του επεξεργαστή με την ενημέρωση της μνήμης.

Πλεονεκτήματα της χρήσης των πολιτικών εγγραφής write back και write through :

	WT	WB
Πολλαπλές εγγραφές μέσα σε ένα block απαιτούν μόνο μια εγγραφή στη lower level μνήμη		X
Μικρότερο εύρος μνήμης (bandwidth) αφού μερικές εγγραφές δεν πάνε στη μνήμη		X
Γλυτώνει ενέργεια, αφού χρησιμοποιεί το υπόλοιπο της ιεραρχίας μνήμης		X
Ευκολότερη υλοποίηση	X	
Απλοποιεί τη συνοχή των δεδομένων με το γεγονός ότι το lower level έχει το μεγαλύτερο αντίγραφο των δεδομένων	X	

Ποιά η σχέση των dependencies με τα hazards ; Ονοματίστε και συσχετίστε τα είδη των data hazards με τα true, anti & output dependencies αντίστοιχα :

Υπάρχουν 3 τύποι εξαρτήσεων (dependencies) : data dependencies (true), name και control dependencies. Μια εντολή j είναι data dependent από μια άλλη i αν η i παράγει ένα αποτέλεσμα που μπορεί να χρησιμοποιηθεί από την j ή αν η j είναι data dependent από την k, και η k είναι data dependent από την i. Αν 2 εντολές είναι data dependent τότε δεν μπορούν να εκτελεστούν ταυτόχρονα ή να επικαλυφθούν πλήρως. Η εξάρτηση δείχνει ότι υπάρχει μια αλυσίδα από 1 ή περισσότερα data hazards μεταξύ των 2 εντολών. Η σημασία του data dependence είναι ότι μια εξάρτηση 1) δείχνει τη δυνατότητα να συμβεί ένα hazard, 2) καθορίζει τη σειρά με την οποία τα αποτελέσματα πρέπει να υπολογιστούν, 3) θέτει ένα άνω όριο στον παραλληλισμό που μπορεί να επιτευχθεί. Ένα name dependence συμβαίνει όταν 2 εντολές χρησιμοποιούν τον ίδιο register ή την ίδια θέση μνήμης (name), αλλά δεν υπάρχει ροή δεδομένων μεταξύ των εντολών που να συνδέεται με αυτό το name. Υπάρχουν 2 τύποι : anti και output dependencies. Ένα control dependence καθορίζει τη σειρά μιας εντολής i σε συνάρτηση με μια εντολή branch, έτσι ώστε η εντολή i να εκτελεστεί στη σωστή σειρά και μόνο όταν χρειάζεται.

Ένα hazard δημιουργείται κάθε φορά που υπάρχει μια εξάρτηση μεταξύ εντολών, και αυτές είναι αρκετά κοντά έτσι ώστε η πιθανή επικάλυψη που προκαλείται από το pipeline, ή από κάποια άλλη επαναδιάταξη των εντολών, να αλλάζει τη σειρά της προσπέλασης στην τιμή που εμπλέκεται με την εξάρτηση.

Τα data hazards μπορούν σε ταξινομηθούν σε 3 κατηγορίες, ανάλογα με τη σειρά των προσπελάσεων για διάβασμα ή εγγραφή στις εντολές. Θεωρούμε 2 εντολές i και j όπου η i προηγείται στη σειρά του προγράμματος. Τα πιθανά data hazards είναι :

RAW (read after write) : η j προσπαθεί να διαβάσει μια τιμή πρίν η i τη γράψει. Δηλαδή η j διαβάζει λάθος τιμή. Αυτός ο τύπος hazard είναι ανάλογος με μια true dependence.

WAW (write after write) : η j προσπαθεί να γράψει μια τιμή πρίν η i γράψει στην ίδια θέση. Δηλαδή η τελική τιμή είναι λανθασμένα αυτή που γράφει η i. Αυτός ο τύπος hazard είναι ανάλογος με μια output dependence.

WAR (write after read) : η j προσπαθεί να γράψει μια τιμή πρίν η i διαβάσει από την ίδια θέση. Δηλαδή η i λαμβάνει λάθος τιμή. Αυτός ο τύπος hazard είναι ανάλογος με μια anti dependence.

Δυσκολίες υλοποίησης μιας pipeline :

- dealing with exceptions (arithmetic over / underflow)
- stopping and restarting execution (virtual memory page fault)
- instruction set complications (multicycle instructions)

Να αποφεύγονται : variable instruction length, πολύπλοκα addressing modes, self-modifying code, condition codes.

Dynamic scheduling : υπάρχει H/W που αναδιοργανώνει την εκτέλεση εντολών αφού έχει γίνει το compilation. Στόχος του είναι η μείωση των stalls. Τα πλεονεκτήματά του είναι η απλούστευση του compiler, το γεγονός ότι κώδικας που έχει μεταγλωττιστεί σε συγκεκριμένο pipeline μπορεί να τρέξει σε διαφορετικό pipeline και ότι αντιμετωπίζει τα προβλήματα κατά την εκτέλεση του προγράμματος με αποτέλεσμα να μην επιβαρύνει τον compiler.

Περιγράψτε την τεχνική RAID και δώστε τους λόγους που οδήγησαν στη δημιουργία της; Προβλέπετε ότι θα επικρατήσει στα μοντέρνα υπολογιστικά συστήματα και γιατί; 2195

Η τεχνική RAID χρησιμοποιείται για να αυξήσει την αξιοπιστία και την διαθεσιμότητα των δεδομένων σε ένα υπολογιστικό σύστημα. Αφορά τη φύλαξη δεδομένων σε συστήματα δίσκων. Χρησιμοποιούμε disk array από πολλούς δίσκους, πράγμα που αυξάνει τη διαθεσιμότητα των δεδομένων αφού μπορούν πιο πολλοί χρήστες να προσπελαύνουν τα δεδομένα ταυτόχρονα. Επίσης χρησιμοποιούμε επιπρόσθετους δίσκους οι οποίοι φυλάγουν πλεονάζουσες πληροφορίες (redundant information) για να αντικαταστήσουμε τα δεδομένα μας σε περίπτωση που υπάρξει disk fail. Το disk array N από μόνο του χωρίς redundancy είναι λιγότερο αξιόπιστο από ένα μόνο δίσκο ίσης χωρητικότητας γιατί η αξιοπιστία του disk array είναι το 1/N της αξιοπιστίας του ενός δίσκου ίσης χωρητικότητας. Για να αποκτήσει το σύστημα αξιοπιστία σπάμε το disk array σε αξιοπιστία groups και κάθε group έχει επιπλέον δίσκους που περιέχουν πληροφορίες για να αντικαταστήσουμε τα δεδομένα που θα χάσουμε σε περίπτωση disk fail.

Οι λόγοι που οδήγησαν στην εμφάνισή της είναι η ανάπτυξη των συστημάτων αποθήκευσης καθώς τα δεδομένα αποθηκεύονται σε διαφορετικούς δίσκους. Επίσης υπήρχε η ανάγκη για μεγαλύτερο throughput. Η τεχνική αυτή θα επικρατήσει στα μοντέρνα υπολογιστικά συστήματα και θα προσφέρει υψηλότερο throughput σε συνδυασμό με τη δυνατότητα αντιμετώπισης λαθών. Επίσης ο συνδυασμός της τεχνικής με συσκευές μικρού μεγέθους και μικρότερης κατανάλωσης προσφέρει μια ελκυστική λύση για τα συστήματα αποθήκευσης.

Disk Arrays - Μεγάλοι αριθμοί data & I/O

Μείωση Αξιοπιστίας - Αξιοπιστία N δίσκων = Αξιοπιστία ενός δίσκου /N

Η ιδέα αποθήκευση πλεονάζουσων πληροφοριών

- Πρόσθεση χρησιμοποιούμενων χωρητικοτήτων
- Πρόσθετο bandwidth ανανέωσης πληροφορίας
- Πολύ υψηλή αξιοπιστία

Υψηλό transfer rate

Μείωση overhead

Επιδεικνύονται εφαρμογές

Ποιά είναι τα επίπεδα RAID και ποιά τα πλεονεκτήματα / μειονεκτήματά τους 2|9

1) **Mirrored :** Είναι η πιο ακριβή αφού κάθε data disk έχει και 'ένα check disk άρα κάθε εγγραφή στο data είναι επίσης και εγγραφή στο check. Μεγάλη αξιοπιστία όμως έχω overhead κόστος 100% και μόνο 50% usable storage capacity.
2) **memory style ECC :** Προσφέρει ρυθμους διαμεταγωγής δεδομένων και αυτόματη διαδικασία διόρθωσης λαθών.Τα αρχεία χωρίζονται σε λέξεις για κάθε μια απο τις οποίες παράγεται κώδικας διόρθωσης (Hamming) ο οποίος καταγράφεται στις αντίστοιχες μονάδες(ξεχωριστές απο τις μονάδες αποθήκευσης)
3) **bit interleaved parity :** Data disks + 1 check disk. Στον πλεονάζον δίσκο εγγράφεται το sum mod 2 όλων των data disks. Δίσκοι δεδομένων (a,b,c,d) ,check disk (k). Ισχύει a+b+c+d=k. Όταν καταστραφεί ο ένας απο τους δίσκους (π.χ. d) τότε τα δεδομένα του βρίσκονται με την αφαίρεση d=k-(a+b+c). Μειονέκτημα : έχει πολύπλοκο ελεγκτή.
4) **block interleaved parity**
5) **block interleaved distributed parity :** Όπως και το RAID 4 αλλά η ισοτιμία μοιράζεται στους δίσκους δεδομένων. Είναι πιο οικονομικό αλλά έχει πιο πολύπλοκο ελεγκτή.