$$CPU\ TIME = \frac{SECONDS}{PROGRAM} = \frac{INSTRUCTIONS}{PROGRAM} X \frac{CYCLES}{INSTRUCTION} X \frac{SECONDS}{CYCLE}$$

	Instr. Count	CPI	Clock Rate
Program	X		
Compiler	X	X	
ISA	X	X	X
Οργάνωση		X	X
Τεχνολογία			X

Simulation

•Functional VS Performance/Timing
F: Visible Arch. State, Προσομοίωση λειτουργικότητας εντολών, Μεταβολή κατάστασης (reg, mem), Σωστό output →Software dev &/| Emulation
P/T: Microarch details, Λεπτομερής υλοποίηση δομών, Χρονισμός
•Trace VS Execution-driven
T: Εκτέλεση σε πραγματική πλατφόρμα, traces as inputs, προσομοίωση proprietary apps & input sets.
E-D: O simulator εκτελεί την εφαρμογή, διατήρηση app&arch state
•User-Code VS Full System

ΠΕΡΙΟΡΙΣΜΟΙ ΒΑΘΜΩΤΩΝ ΑΡΧΙΤΕΚΤΟΝΙΚΩΝ

•Μέγιστο throughput 1 - $IPC \leq 1$ → Superscalar
•Υποχρεωτική ροή όλων των τύπων εντολών μέσα από κοινή σωλήνωση → Multicycle operations
•Εισαγωγή STALLS σε in-order-execution→δυναμικές αρχιτεκτονικής

Multi-cycle Operation & Hazards

•Non-pipelined units→Structural Hazards
•Different Latency→Structural Hazards
•Wrong Order in WB→WAW Hazards
•Out-of-order completion
•Too much latency→Stalls→RAW

Instruction Level Parallelism

$$Aver.ILP = \frac{\#instructions}{\#cycles\ required}$$

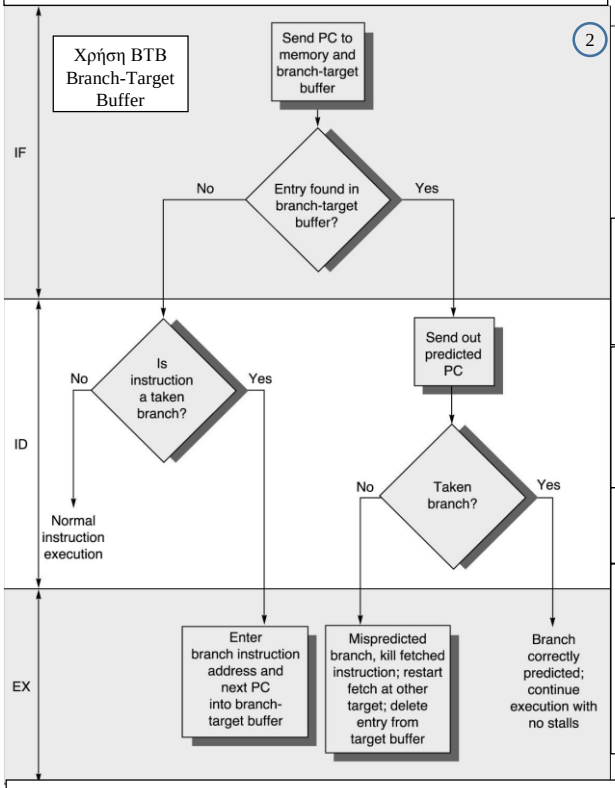
Oper. Latency (OL), Machine Parallelism (ML), Issue Latency (IL), Issue Parallelism (IP)

Αντιμετώπιση Control Depedencies (2)

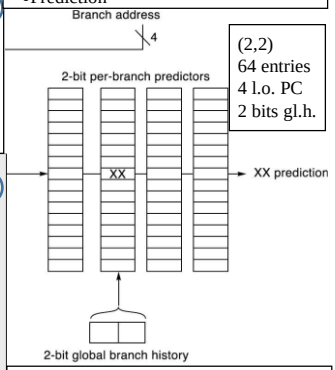
•Stalls
•Branch Delay Slots
•Predicated execution
•Fine-grained Multithreading
•Multipath execution
•Prediction

Κατηγοριοποίηση με βάση ILP (1)

•Superpipelined: cycle = 1/m of baseline, IP = 1 instr./minor cycle, OL=1 Maj.c. = m minor c, IL=1mc, MP=m x k, IPC(max)= m instr./ M.c.
•Superscalar: IP=n instr./cycle, OP=1cycle,IPC(max)= n instr./cycle
•VLIW: IP=n instr./cycle, OP=1cycle,IPC(max)= n VLIW/cycle
•Superpipelined-Superscalar: IP = n inst../m.c., OP= m m.c., IPC(max)= n x m



(2)



(2,2)
64 entries
4 l.o. PC
2 bits gl.h.

Αντιμετώπιση OOO Exec Problems (3)

•**Scoreboarding** (1963 CDC6600)
•**Tomasulo's** Algor. (renaming + RS)
ISSUE→EXECUTE→WRITE RESULT
VS **Explicit Renaming**: Συνδυασμός

Precise Interupts/Exceptions (4)

Precise: Έχουν εκτελεστεί ΟΛΕΣ οι προηγούμενες εντολές, &, Καμία εντολή δεν έχει αλλάξει την κατάσταση της μηχ.
Γιατί; -Διότι: TLB faults, Underflow, Εύκολη επανεκκίνηση, απλοποιεί το ΛΣ

ReOrderBuffer (ROB)

FIFO και Commit InOrder !
Σε περίπτωση br. mispred.κάνω flush

Πολυνηματισμός

+: Δε χρειάζεται dependency checking, branch prediction, αποφυγή bubbles, καλύτερο throughput, utilization, latency tolerance
-: Πολύπλοκο Hardware, Χειρότερο single-thread-performance, Υψηλός ανταγωνισμός για πόρους

Ταξινόμηση Παράλληλων Αρχιτεκτονικών

SISD SIMD MISD **MIMD**

MIMD: **Process** → (κοινός κώδικας και χώρος διευθύνσεων) **Thread**

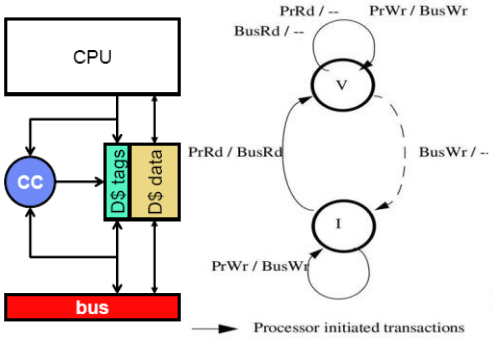
Cache Coherence

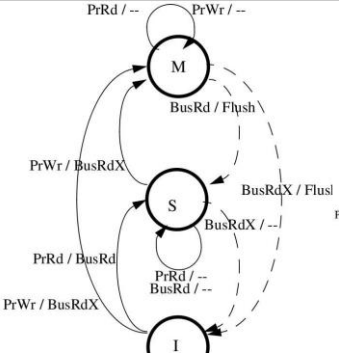
- HW: Cache invalidation
- SW: Flushing Cache Lines
- Non-Cacheable DMA

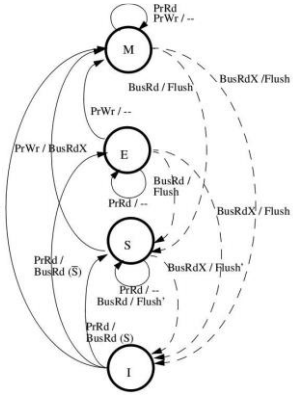
Ορισμός: Coherent αν για κάθε εκτέλεση τα αποτελέσματα είναι τέτοια, ώστε σε κάθε θέση να μπορούμε να κατασκευάσουμε μια υποθετική ακολουθιακή σειρά όλων των λειτουργιών στη θέση αυτή, που να είναι συνεπής, ώστε:

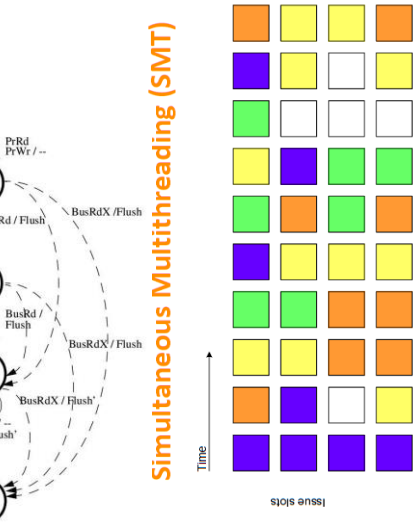
•Οι λειτουργίες κάθε thread έχουν γίνει με τη σειρά αυτή
•Η τιμή είναι η τιμή της τελευταίας εγγραφής στη συγκεκριμένη θέση.

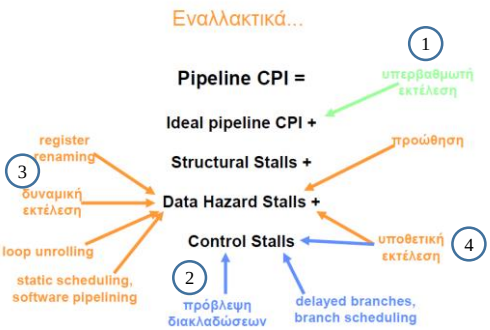
ΣΥΝΘΗΚΕΣ: 1. Διατήρηση σειράς προγράμματος 2. Write propagation 3.Write serialization

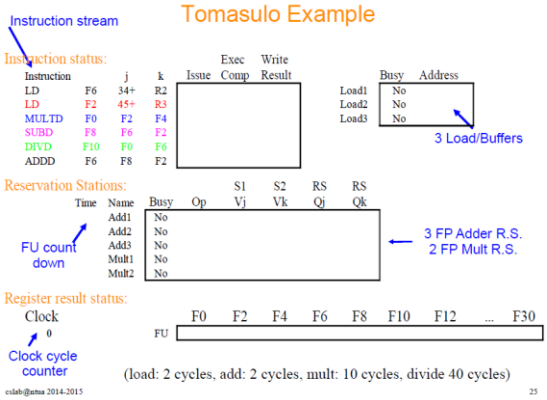


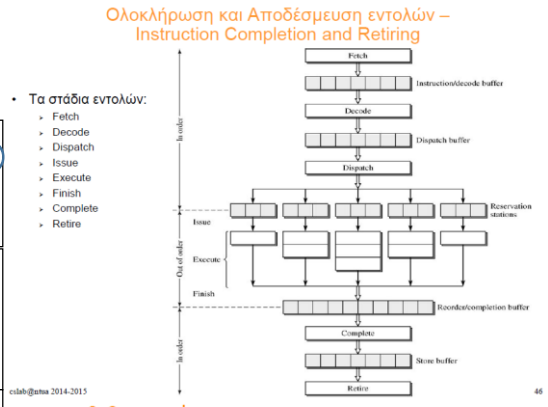


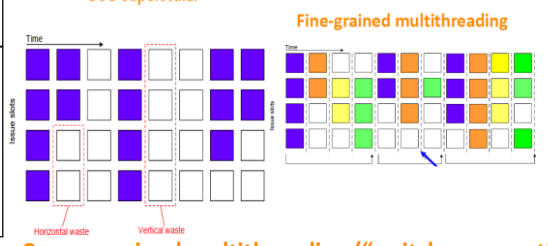


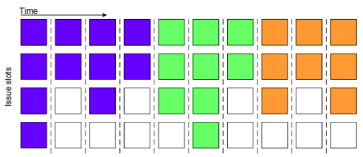


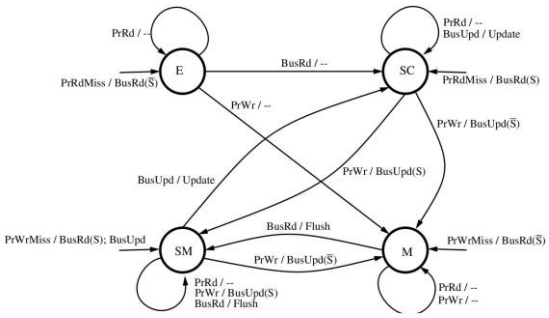












Invalidation VS Update Protocol

Σε κάποια cache γίνεται εγγραφή σε ένα block. Πριν την επόμενη εγγραφή στο ίδιο block, θέλει κάποιος άλλος να το διαβάσει;.

NAI: - Invalidation: Read miss → Πιθανώς πολλαπλά transactions
- Update: Read hit →Ενημέρωση μόνο με ένα transaction

OXI: - Invalidation: Πολλαπλές εγγραφές χωρίς επιπλέον κίνηση στο bus / Εκκαθάριση αντιγράφων που δε χρησιμοπ.
- Update: Πολλαπλές αχρείαστες ενημερώσεις

4C Cache Misses Model

Compulsory: Πρώτη πρόσβαση → Αύξηση block size **Capacity:** To block δε χωράει → Αύξηση cache size
Conflict: To block δε χωρά στο "mapped" set → Αύξηση associativity
Coherence: Communiction, True Sharing: 1 Data word σε 2+ επεξεργαστές / False Sharing: Ανεξάρτητα DW από ίδιο c.b

Cache Coherence VS Memory Consistency

Coherence:
- Διασφαλίζει ότι η τιμή της τελευταίας εγγραφής σε μια θέση μνήμης θα γνωστοποιηθεί σε όλους τους τυχόν αναγνώστες
- Δημιουργεί μια καθολικά ενιαία εικόνα για ΜΙΑ συγκεκριμένη θέση μνήμης (ή cache line)

Consistency:
- Καθορίζει ΠΟΤΕ θα γίνεται ορατή μια εγγραφή
- Δημιουργεί μια καθολικά ενιαία εικόνα για ΟΛΕΣ τις θέσεις μνήμης, όσον αφορά τις μεταξύ τους τροποποιήσεις

Memory Consistency Models

Sequential Consistency:
- Ορίζουμε PROGRAM ORDER
- Ελέγχουμε τα πιθανά αποτελέσματα με ATOMICITY

Relaxed Memory Models:
- Total Store Order (TSO), Processor Consistency (PC)
- Partial Store Order (PSO)
- Weak Ordering (WO), Release Consistency (RC), Relaxed Memory Order (RMO)

Use of: $FENCES_{SS}$ ή $FENCES_{LL}$ ή SYNCH

Αμοιβαίος Αποκλεισμός (Acquiring Locks) σε επεκταμένη ISA

Spin-lock (Test-and-Set):
- Επηρεάζει αρνητικά την απόδοση και αυξάνει την άχρηστη κίνηση στο δίαυλο

Test-and-Test-and-Set:
- Παρακολουθεί την τιμή του lock και μόνο όταν «φαίνεται» να είναι ελεύθερο επιχειρεί να το δεσμεύσει.
- Λιγότερη άχρηστη κίνηση στο δίαυλο

Συστήματα με πολλούς επεξεργαστές:

Βασικές αρχιτεκτονικές

Υπενθύμιση: