Requirements for and Design of a Processor with Predictable Timing

Christoph Berg

Reinhard Wilhelm

Universität des Saarlandes

Jakob Engblom

Uppsala University, Sweden

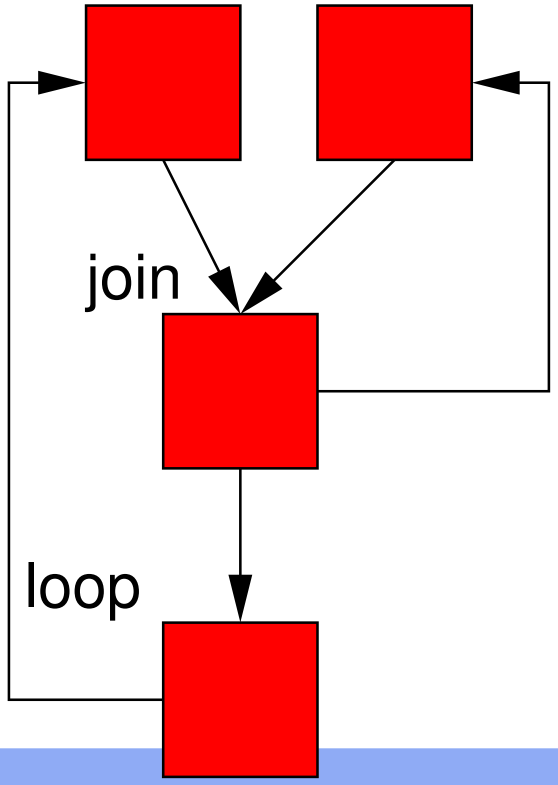
June 2, 2003

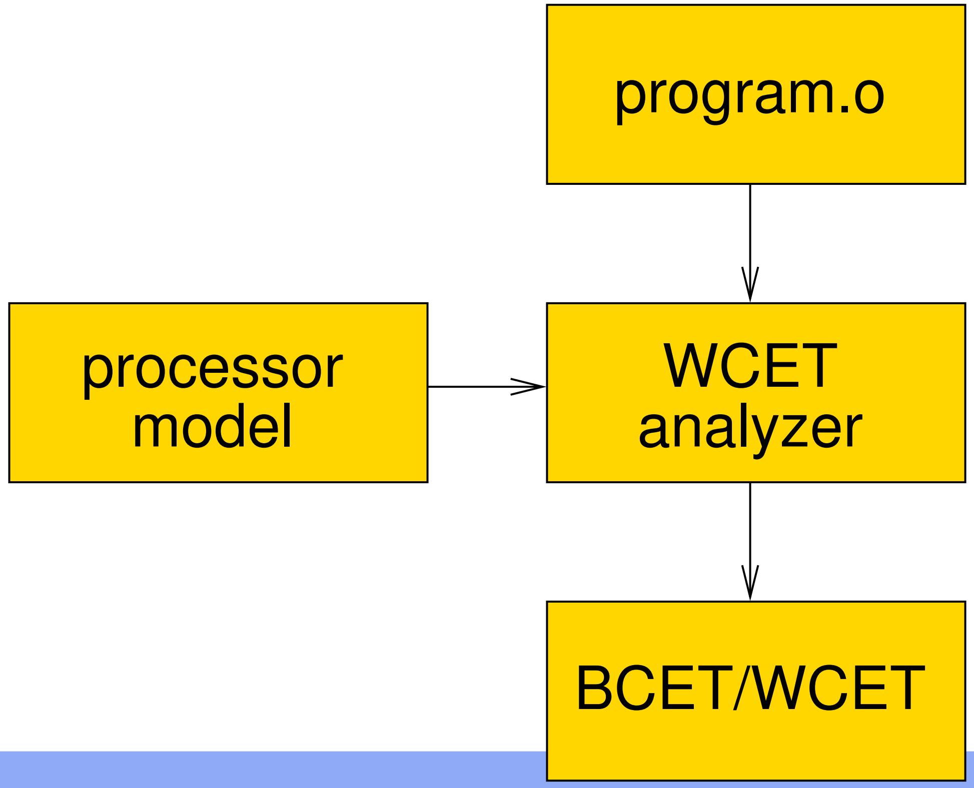
- hard real-time systems require bounds on program execution time
  - airbag controller, drive-by-wire, . . .
- lower/upper bounds on execution time
  - BCET/WCET = “best/worst case execution time”
- tight bounds, average performance not interesting
- many processor features hinder precise prediction
  - caches, OOO pipeline, . . .

- identify problems
- formulate principles for predictable processors
- give processor design that is predictable

Static analysis

- analyze executables by static analysis
- extract information from control flow graph
- merge information at control flow joins
- analysis precision $\leftrightarrow$ processor model (main problem)

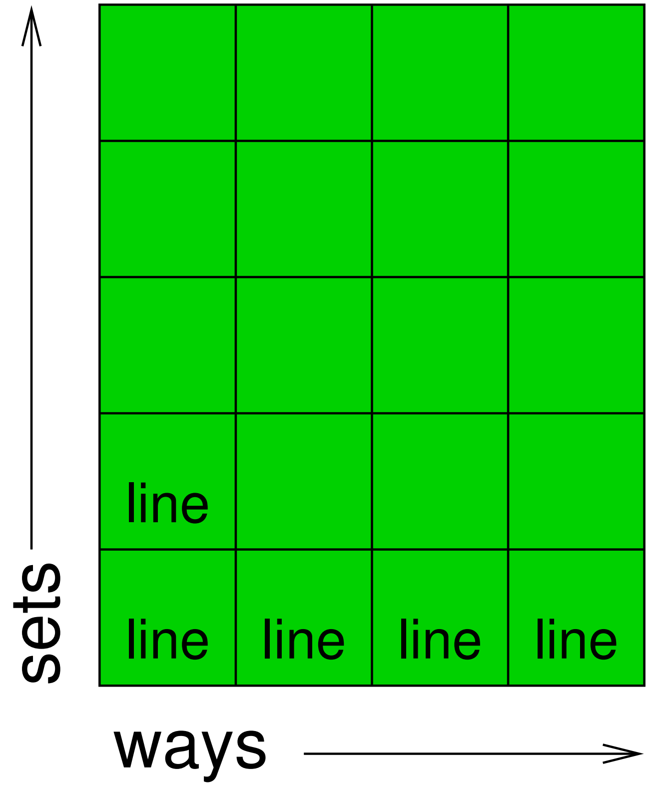




Contents

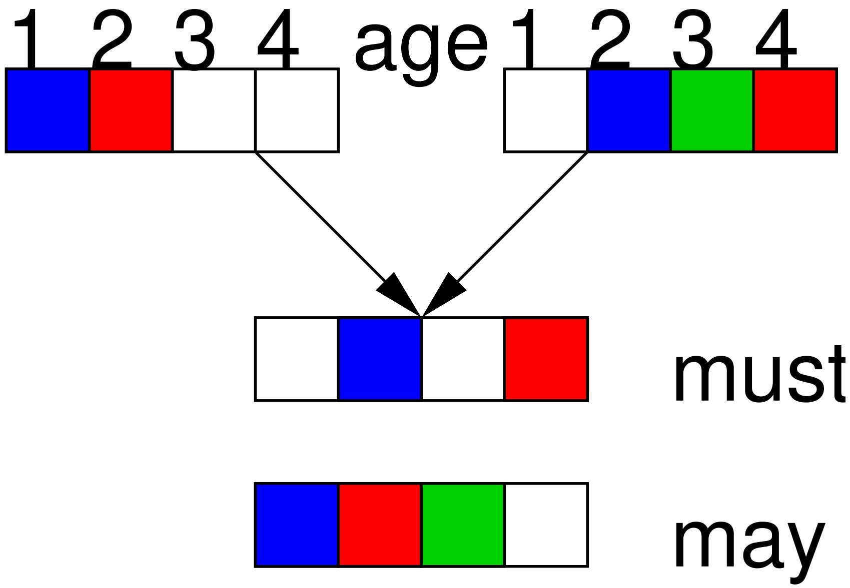
Caches

- caches store instructions and data
- caches consist of *sets* and *ways* containing *lines*
- each memory address is mapped to one set
- replacement strategy determines way to put new line in set
  - least recently used (LRU)
  - round robin
  - random . . .



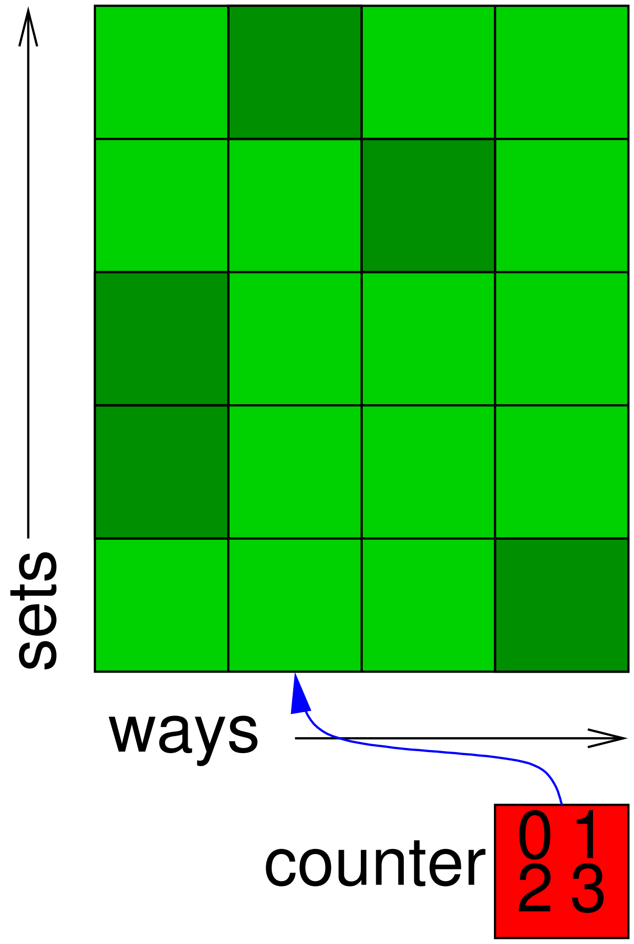
Cache analysis

- cache model: keep track of cache contents in analysis
- determine memory access addresses
- must analysis
  - lines that must be in set
  - cache hits
- may analysis
  - lines that may be in set
  - otherwise: cache misses
- LRU example: control flow join



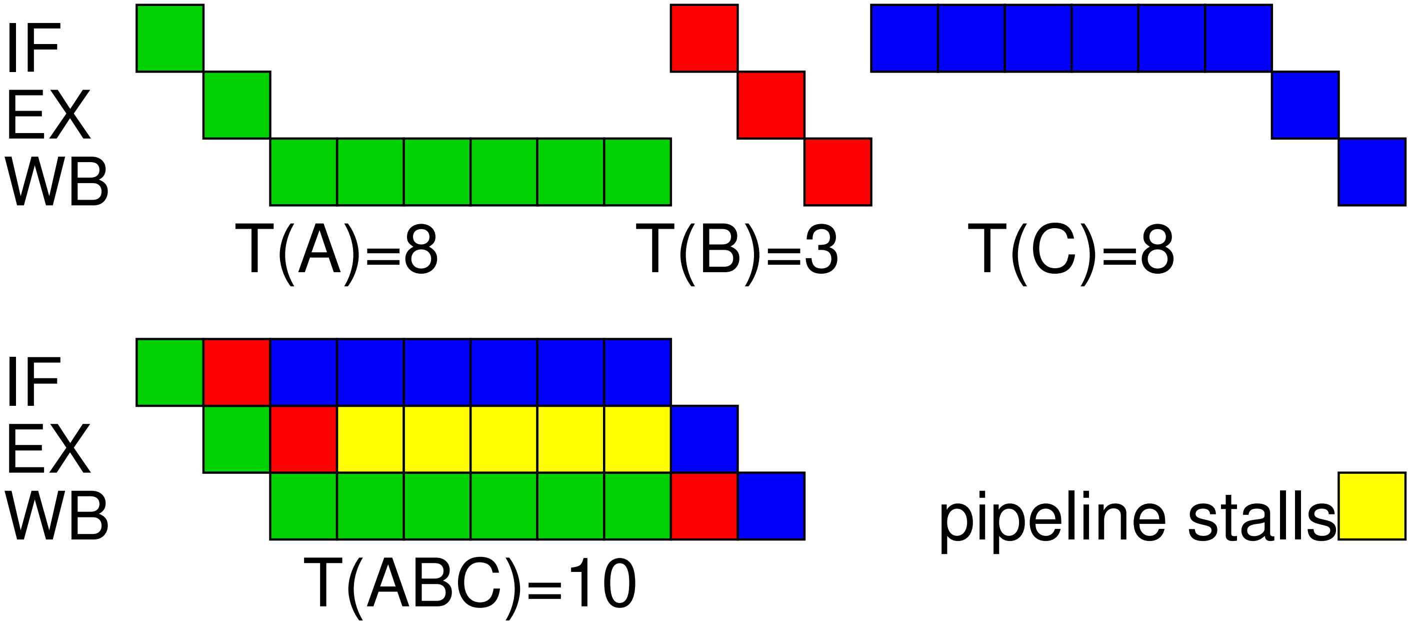
The Motorola ColdFire 5307 cache [FHL+01]

- "global" or "pseudo" round robin replacement, 4-way
- global counter selects way to replace
- counter is incremented on misses
- counter value gets lost if we don't know whether it's a miss
  - *any* way could be replaced
- only one way definitely in cache
  - 1-way cache model in must analysis
- nothing known in may analysis!
- cache is *unified*: instructions/data evict each other



Pipeline Analysis

- estimating execution time of instruction sequence
  - $T(A_1 \ldots A_m)$
- fill pipeline model with instructions
- sum up execution times for pairs of instructions
- problems: long timing effects, timing anomalies

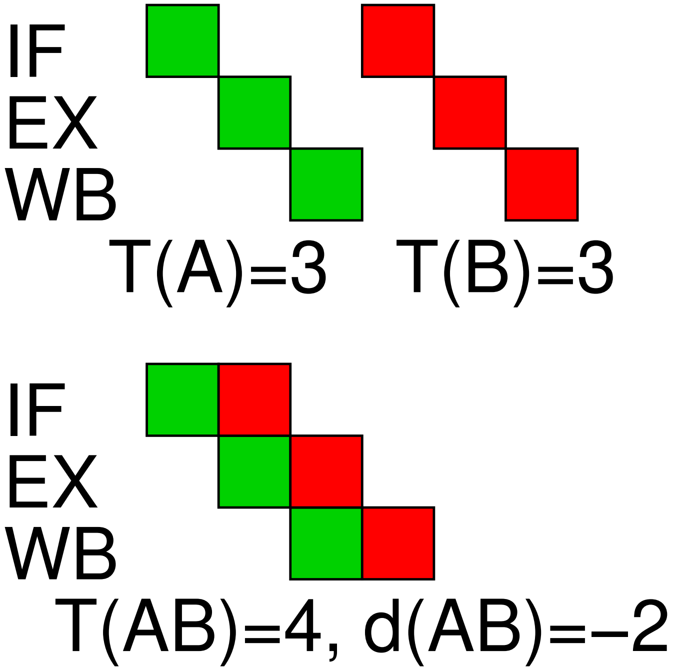


Long timing effects (LTEs) [EJ02]

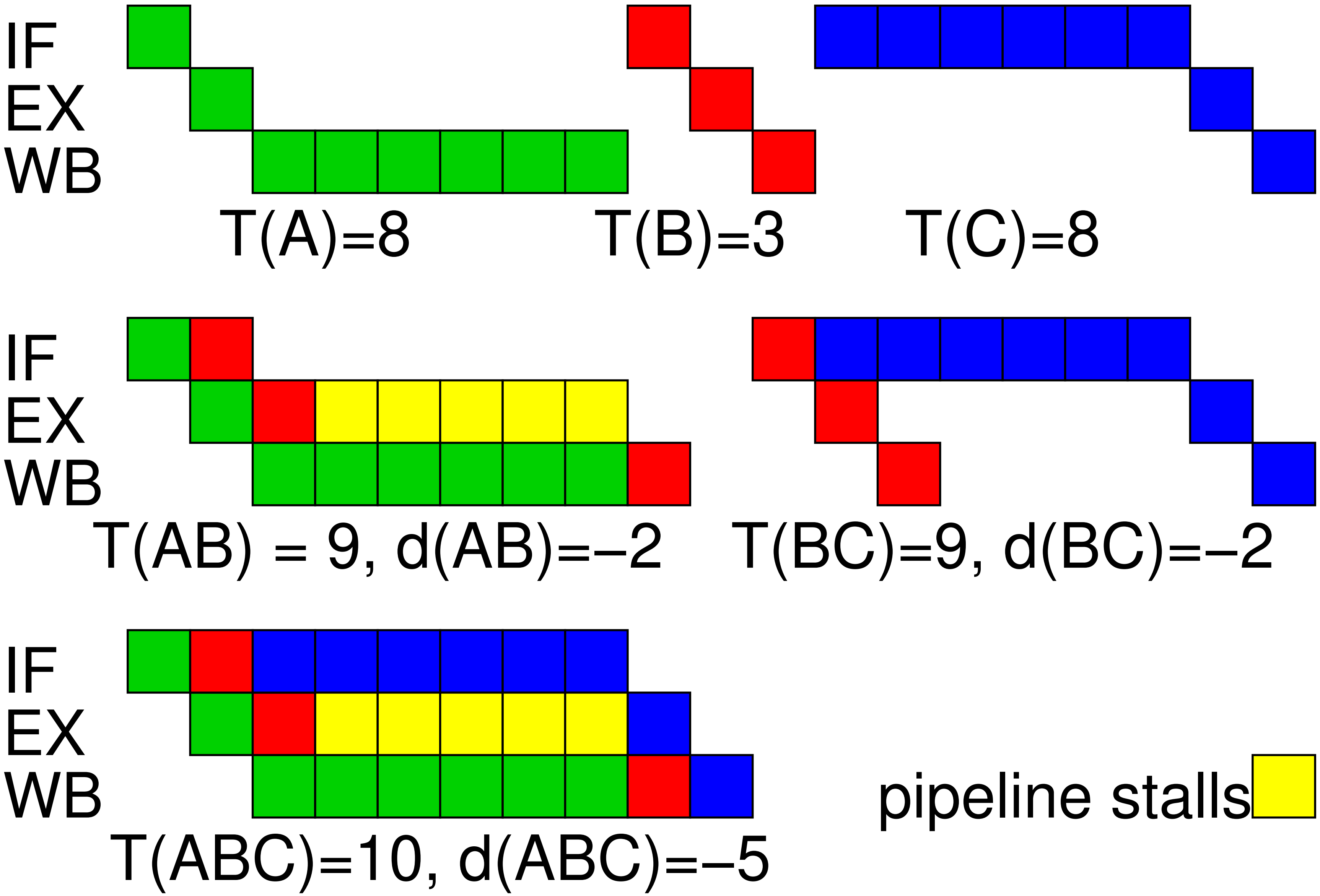
- timing effect of $A_1 \ldots A_m$

$$\delta(A_1 \ldots A_m) := T(A_1 \ldots A_m) + T(A_2 \ldots A_{m-1}) - T(A_1 \ldots A_{m-1}) - T(A_2 \ldots A_m)$$

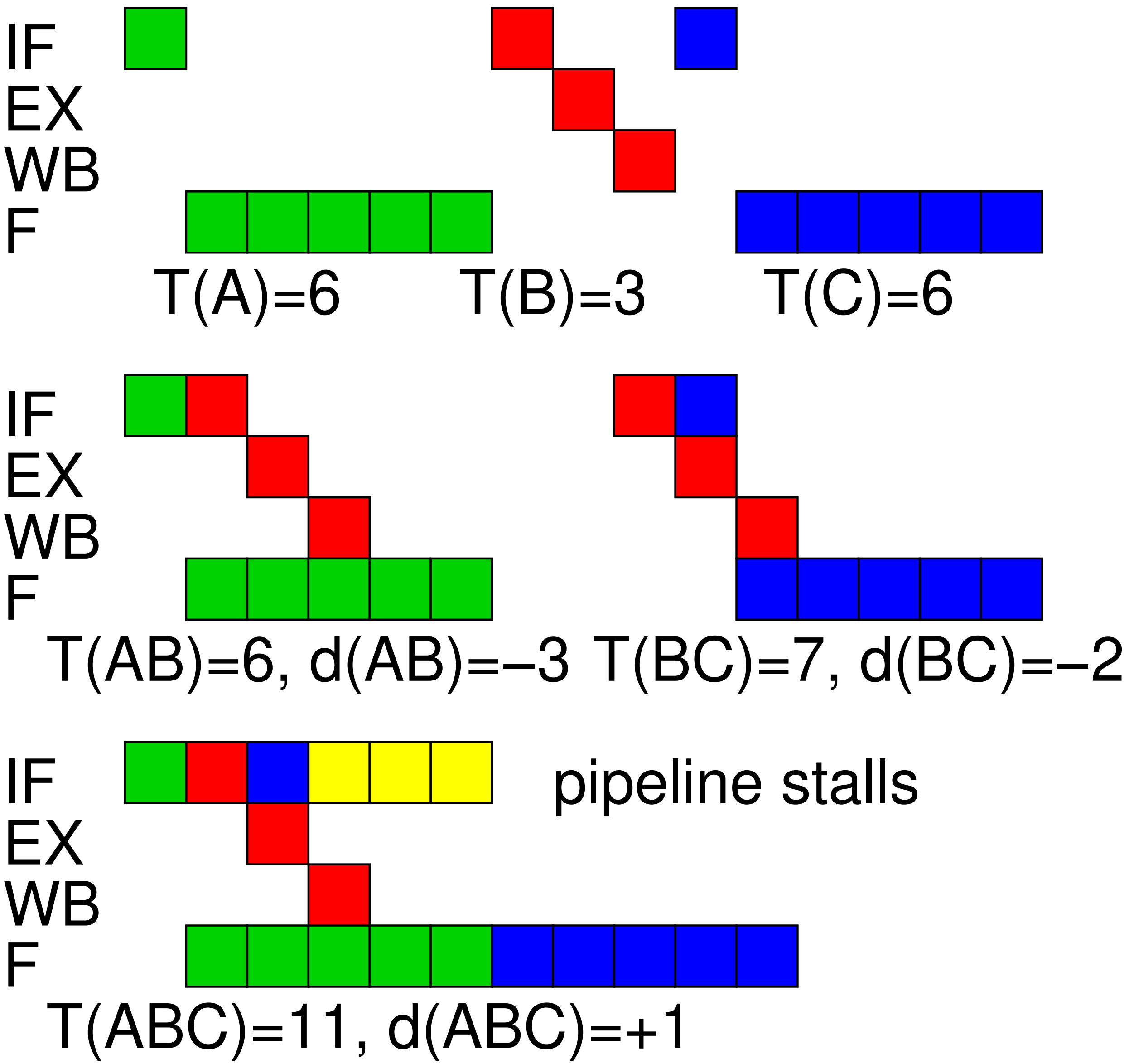
- $\delta(AB)$ = pipelining speedup for AB
- LTE := $\delta(A_1 \ldots A_m) \neq 0$ for $m > 2$
  - timing influenced by non-adjacent instructions
  - can be negative and positive
  - can be unbounded
- LTEs make pipeline analysis difficult
- examples



Long timing effects: negative example



Long timing effects: positive example



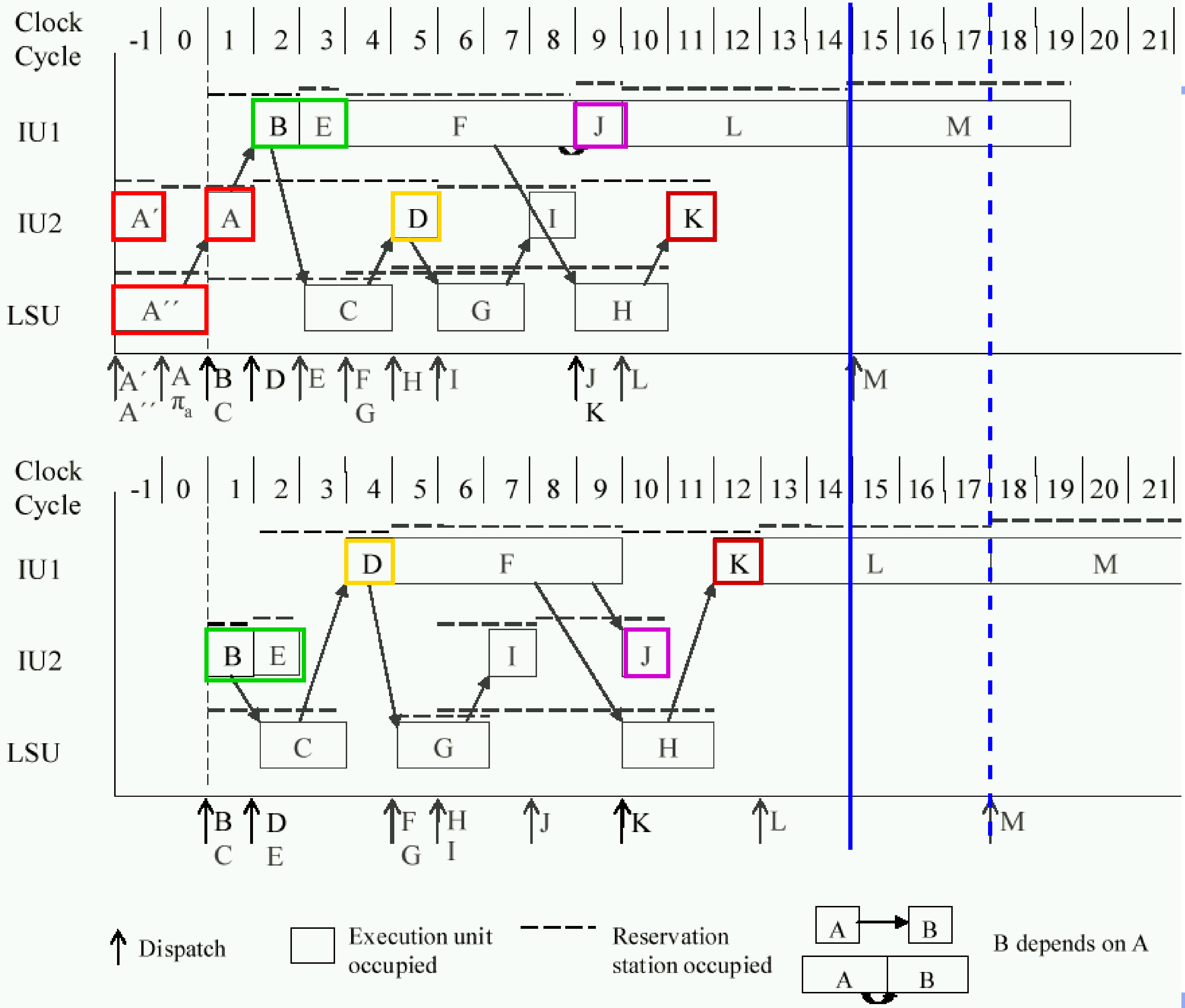
Avoiding long timing effects

- LTEs make it necessary to look at arbitrarily long sequences
- few LTEs: [EJ02]
  - no hardware interlocks (insert nops)
  - no parallel pipelines $\rightsquigarrow$ no positive LTEs
  - few stages with variable latency, pack them close

Timing anomalies [LS99]

- timing anomaly := for a sequence of instructions
  - *faster* execution by *increasing* latency or
  - *slower* execution by *decreasing* latency

  of the first instruction or other resources
- make pipeline behavior prediction even more difficult
  - more instructions might take *less* time
- example: PowerPC 755
  - two integer units
  - long-running instructions in IU1 only



Clock Cycle
-1 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
IU1
IU2
LSU
B E F J L M
A′ A D I K
A′′ C G H
A′ A′′ A π_a B C D E F G H I J K L M
Clock Cycle
-1 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
IU1
IU2
LSU
D F K L M
B E I J
C G H
B C D E F G H I J K L M
Dispatch
Execution unit occupied
Reservation station occupied
A B
A B
B depends on A

Contents

1. Recoverability

- parts of state are/become unknown in analysis
  - unknown state at beginning (e.g. unknown cache contents)
  - unknown transitions (e.g. cache hit/miss)
  - control flow joins: merging of states
- recoverability := knowledge can be recovered by continuing analysis
- example: cache fills with each memory access
- counter-examples
  - ColdFire cache (bad replacement strategy)
  - branch prediction with hash table

2. Minimal variation

- we are analyzing best/worst case, not average case
- assume worst case if timing is unknown
- minimal variation := eliminate variation in instruction timing
  - more analysis precision
- bad idea: “shortcut multiplications” for small operands

3. Non-interference

- interference between processor components makes analysis harder
- abstracting away parts of state harder
  - separate analyses not possible
- non-interference := decoupling of behavior
- examples
  - branch prediction $\leftrightarrow$ cache contents
  - OOO scheduling $\rightarrow$ order of memory accesses $\rightarrow$ cache analysis
  - unified caches (self-interference)

Tradeoffs

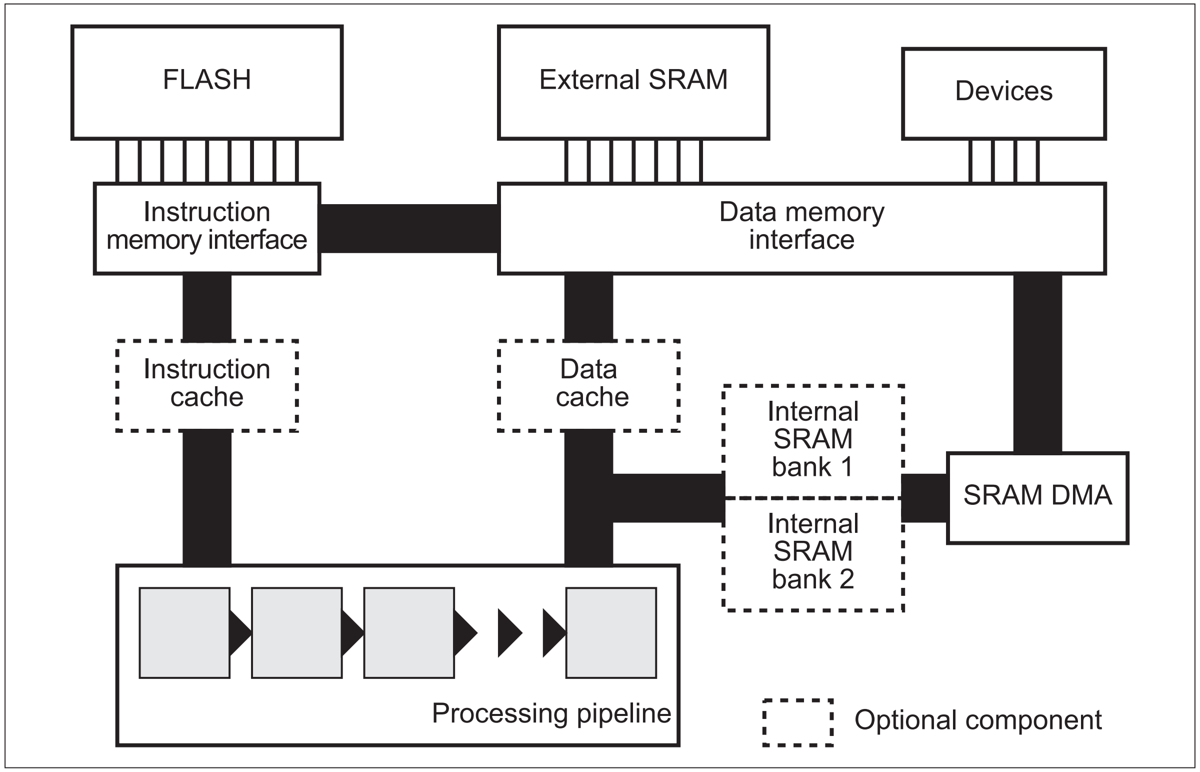
- processor strictly adhering would have no caches, pipeline, . . .
- prinicples $\leftrightarrow$ performance tradeoff
  - ▷ caches violate minimal variation, but give performance
- processor complexity $\leftrightarrow$ analysis effort tradeoff
  - ▷ time needed to design and execute analysis
- average case speedup must translate in a lower WCET

Contents

Instruction set

- fixed/variable length: predictability vs. code density
- variable length:
  - more compact code
  - harder instruction cache analysis
  - misalignments
- choose 32-bit RISC-style instruction set

Memory System



- separate instruction/data interfaces to eliminate interference (constant memory timing)
- no data accesses to instruction space
- FLASH for instruction memory, SRAM for data (no DRAM)

Caches

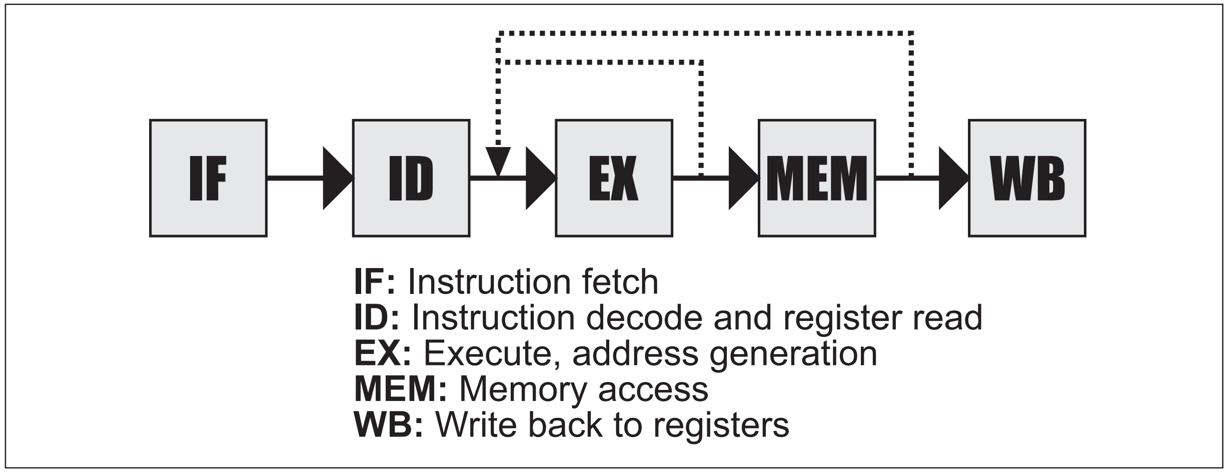
- best we can do: (analysis/performance tradeoff)
  - replacement strategy: least-recently-used (LRU)
  - associativity: 4
  - separate caches
- instruction cache: unrolling loops in analysis yields good precision [FHL+01]
- data cache: >90% of data addresses statically predictable (Airbus benchmarks)
- use caches if FLASH/SRAM does not fit on-chip/has large latency (no data cache for streaming applications)

Branch prediction

- interacts with instruction cache
- dynamic branch prediction is hard to predict
- BTFN (backward branches taken, forward not taken)
  - 60-70% accurate for embedded applications
  - can be modeled tightly, good precision [FHL+01]

Pipeline

- standard 5-state pipeline with forwarding
  - ▷ IF/fetch: variable latency (I cache)
    - ▷ ID/decode: 1 cycle
    - ▷ EX/execute: 1 cycle (more for divisions)
    - ▷ MEM/memory: 1 cycle/more (SRAM/D cache)
    - ▷ WB/write back: 1 cycle
  - 500 MHz in current processes
  - LTEs limited [EJ02]



Contents

Last slide

- contributions
  - principles are design goals for real-time processors
  - design is proposed, guideline for computer architects
- future work
  - more practical work on predictability of branch prediction, pipelines, . . .
  - extract theoretical background of principles

References

[EJ02] Jakob Engblom and Bengt Jonsson. Processor Pipelines and Their Properties for Static WCET Analysis. In *Proc. Second International Workshop on Embedded Software (EMSOFT 2002), LNCS 2491*, October 2002.

[FHL+01] Christian Ferdinand, Reinhold Heckmann, Marc Langenbach, Florian Martin, Michael Schmidt, Henrik Theiling, Stephan Thesing, and Reinhard Wilhelm. Reliable and Precise WCET Determination for a Real-Life Processor. In *Proc. First International Workshop on Embedded Software (EMSOFT 2001), LNCS 2211*. Springer-Verlag, October 2001.

[LS99] Thomas Lundqvist and Per Stenström. Timing Anomalies in Dynamically Scheduled Microprocessors. In *Proc. 20th IEEE Real-Time Systems Symposium (RTSS'99)*, December 1999.

Documentation of timing semantics

- documentation describes functional semantics of instruction set
  - ISA semantics, ISA state
  - visible: memory, registers
- documentation covers timing behavior only vaguely
  - timing semantics, timing state
  - visible: cache/pipeline contents, interrupts, branch prediction, data prefetch, . . .
- examples
  - DRAM timing wrong in study of influence of DRAM refreshes
  - Pentium 4: branch prediction not really documented

4. Determinism

- timing behavior must only depend on information that is observable in analysis
  - recoverability
- analysis efficiency $\rightsquigarrow$ timing state must have a compact approximate representation (abstraction)
  - still good analysis precision
  - unknown parts must have bounded influence
  - prerequisites: minimal variation/non-interference
  - in particular: few LTEs

5. Comprehensive documentation

- we need precise and complete documentation
- prefered: cycle-accurate reference simulator (validated against RTL code)
- printed documentation: all pipeline profiles, comprehensive description of all functional units and interdependencies

Other memory issues

- MMUs with page tables have great latencies, TLB analysis needed
  - address physical memory directly
  - use protection registers
- use memory-mapped I/O for devices
  - must complete in a single cycle (buffer registers)
- DMA: divide SRAM into two blocks, fill/read one block with DMA while computing on the other (WCET estimate for DMA transfers needed!)

System level

- short interrupt latency important in RT systems
- “random” cache pollution, pipeline flushes
- use either
  - static non-preemptive scheduling, or
  - preemption points (with preemption check instruction)