Pipeline Modeling for Timing Analysis

Markus Klein, Johannes Kasberger

Johannes

Caches & Pipelines

- Überbrücken Geschwindigkeitslücke zwischen CPU und RAM
- Erhöhen Geschwindigkeit
- Verschlechtern Vorhersagbarkeit

Ohne Caches?

- Ohne Caches hat eine 300MHz CPU teilweise Performance einer 20MHz CPU mit Caches
- z.B. fly-by-wire benötigt viel Rechenleistung
- → Caches sind notwendig

Motorola ColdFire 5307

- non-trivial real-life processor
  - instruction prefetching
  - branch prediction
  - caches
  - pipeline
  - memory bus

Motorola ColdFire 5307

- very popular microcontroller
- Verwendung z.B. bei Airbus Flugzeugen

Probleme bei WCET

- Vereinfachende Annahmen verhindern präzise Abschätzung
- Mit Pipeline&Caches Model bekommt man bessere Ergebnisse

Bisherige Ansätze

- ILP wurde überall verwendet
- Zu einfache Betrachtung von Pipelines
- Fehlende Betrachtung von emergenten Effekten von Cache-Pipeline-Kombination
- Branch Prediction wurde ignoriert

Überabschätzung

- Nur ein Model das branch prediction, instruction prefetching, speculative execution und data access zusammen betrachtet, kann gute Ergebnisse liefern

Ansatz

- Zwei Phasen bei Analyse
  - execution modeling
  - program path analysis

Markus

Ansatz

- Fertig gelinktes Executeable als Ausgangspunkt
- Vorteil: Toolcain für Luftfahrtindustrie muss nicht neu zertifiziert werden
- Control Flow Graph (CFG) aus Executeable extrahiert
- Leicht portierbar

Analyse

- Wertanalyse über Registerinhalte/ Speicherzugriffe
- Kombinierte Caches&Pipeline Zustandsanalyse (mögliche Kombinationen)
- Mit Loop Bounds → ILP Lösung für WCET

Wenn Caches & Pipeline separat betrachtet wird durch instruction prefetching und branch prediction Ergebnis schlechter

Analyse

- CFG in basic blocks teilen
- Semantic der Instruction wird in abstrakte Darstellung umgewandelt
- Knoten im CFG sind Instructions
  Kanten sind transfer functions

z.b. nicht Ergebnis von Multiplikation sondern nur Dauer interessant

Transfer Function

- Simuliert Effekt der Instruction auf Pipeline
- State von Pipeline&Caches wird berücksichtigt

Effekt von v auf Pipeline so lange simuliert bis v abgeschlossen ist, weiter zu v‘ sobald v fertig ist

Pipeline Model

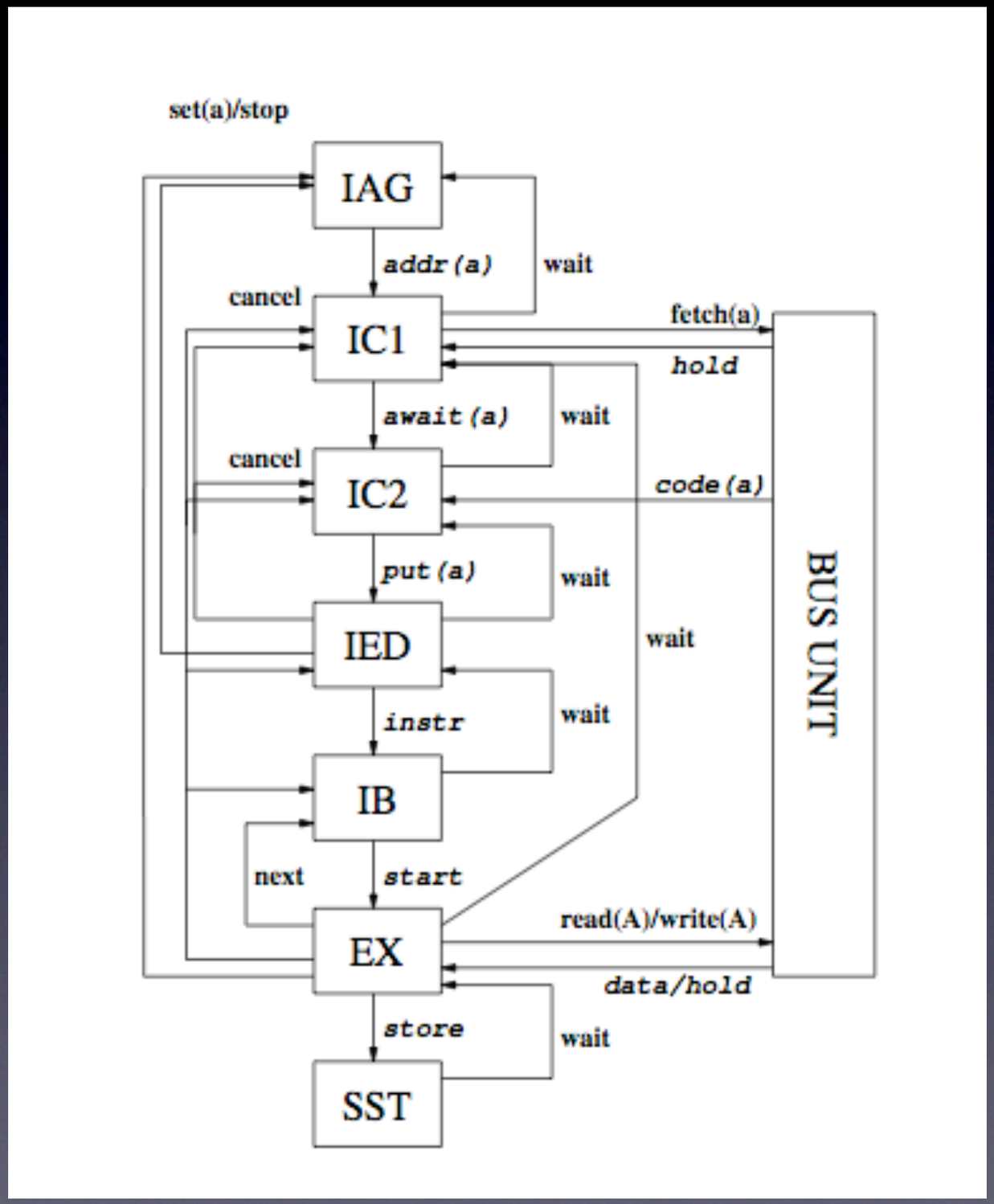
- Wird als State Machine modelliert
- State enthält Informationen über das Timing
- Jeder State wird aufgeteilt in Units (mit eigenen State)
- Units kommunizieren über Signale

Für jeden Clock Cycle ein State weiter

Transitions sind kompliziert

Komplexität wird einfacher mit Units

Formales Model ColdFire



IAG= instruction address generation, IC1/2 instruction fetch 1/2, IED=instruction early decode, IB=instruction buffer, EX=execution unit, SST=store stall timer (virtuell)
BUS Unit = Verbindung CPU mit static RAM, cache und main memory

Manche States 1:1 in Hardware vorhanden, manche Virtuell

Benötigte Zeit für Instruction Cycles zählen die benötigt werden (v=>v‘)

Abstract Pipeline State

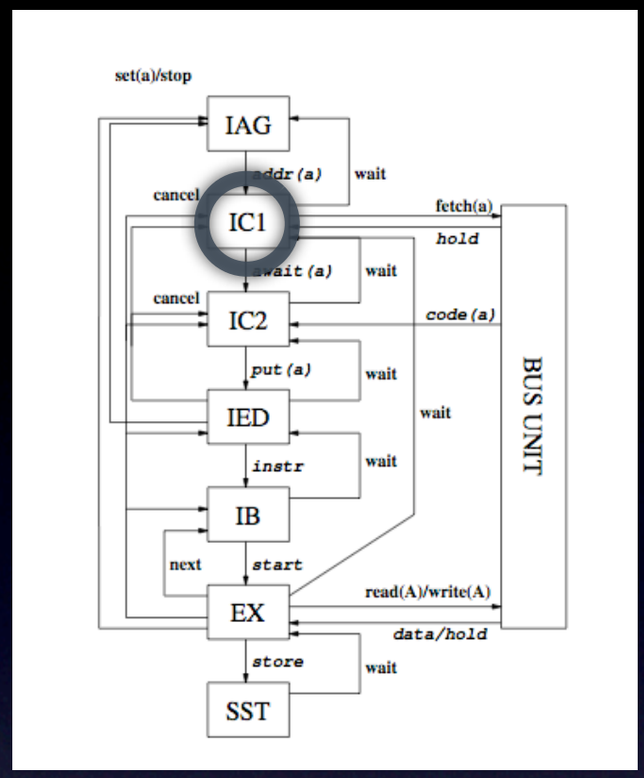
- alle möglichen konkreten Pipeline States für eine Instruction
- z.B. Abhängig von unbekannten Adressen
- Pro konkreter Pipeline State auch ein eigener Cache State

während Analyse Adresse nicht bekannt => Cache Hit/Miss offen

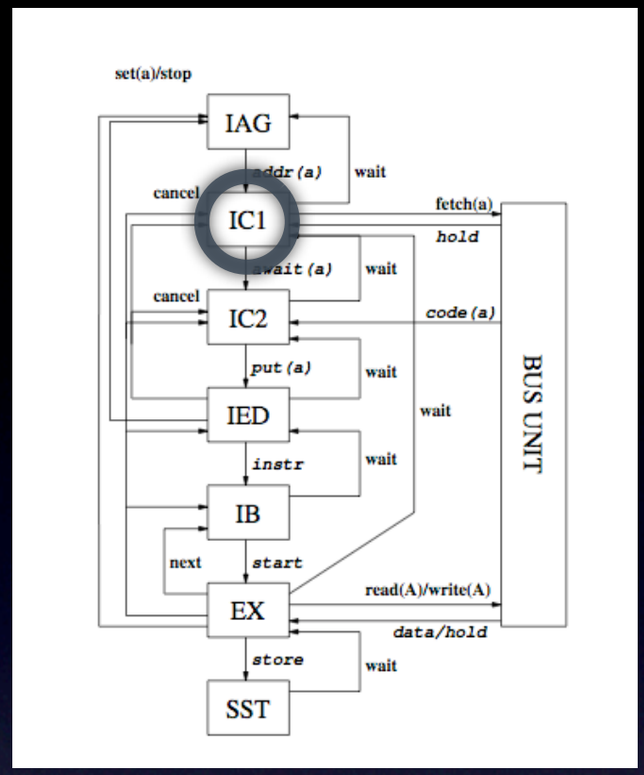
Interessant: Cache Miss nicht immer WorstCase Fall

Abstract State

t



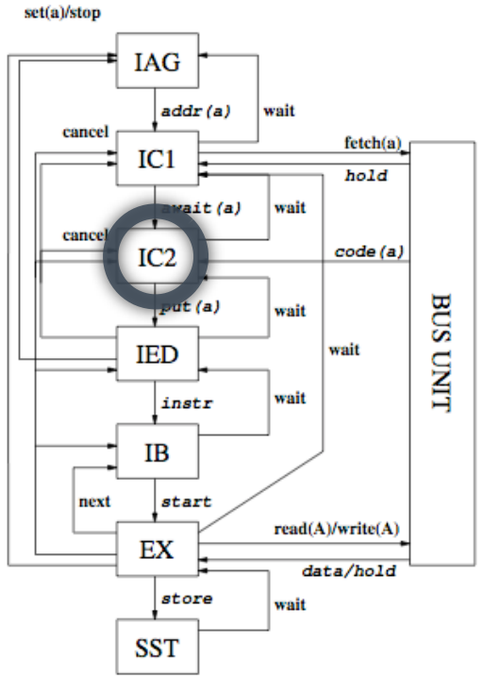
Abstract State

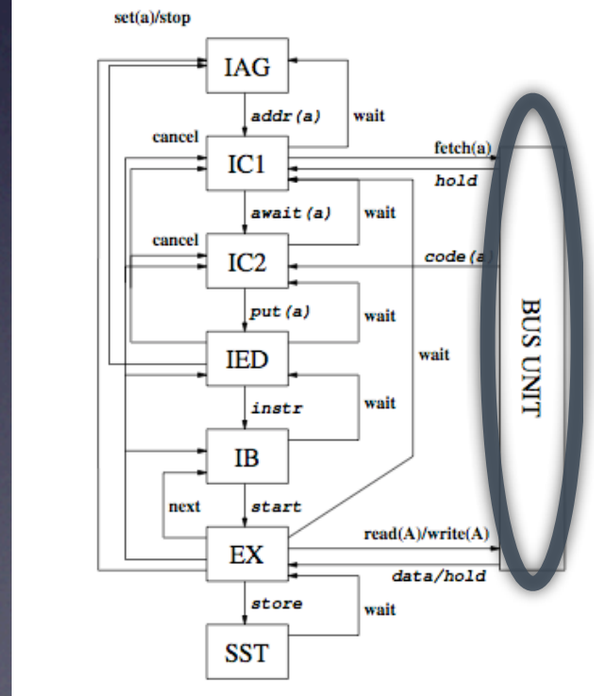


t

t‘

Abstract State





Zusammenfassung

- Gute Ergebnisse bei Airbus Benchmark
- Bei Toy Programmen größere Unterschiede zu gemessener Ausführungszeit

Grund: analysefreundliche Hardware für Toy Probleme und keine realitätsnahen Probleme

Zusammenfassung

- Gute Ergebnisse bei Airbus Benchmark
- Bei Toy Programmen größere Unterschiede zu gemessener Ausführungszeit

Vermutung: Resultiert in aiT

Grund: analysefreundliche Hardware für Toy Probleme und keine realitätsnahen Probleme