# JOP: a Java Processor for Embedded Real-Time Systems

Martin Schoeberl

ESE Seminar

# Overview

- Motivation
- JOP architecture
- WCET analysis
- Conclusion
- Current and future work

# RT System Properties

- Often safety critical
- Execution time has to be known
  - Analyzable system
    - Application software
    - Scheduling
    - Hardware properties
  - Worst case execution time (WCET)

# Issues with COTS

- COTS are for average case performance
  - *Make the common case fast*
  - Very complex to analyze WCET
    - Pipeline (out-of-order)
    - Cache
    - Multiple execution units

# The Idea

- Build a processor for RT System
  - *Optimize for the worst case*
- Design philosophy
  - Only WCET analyzable features
    - No unbound pipeline effects
    - New cache structure
  - Shall not be *slow*

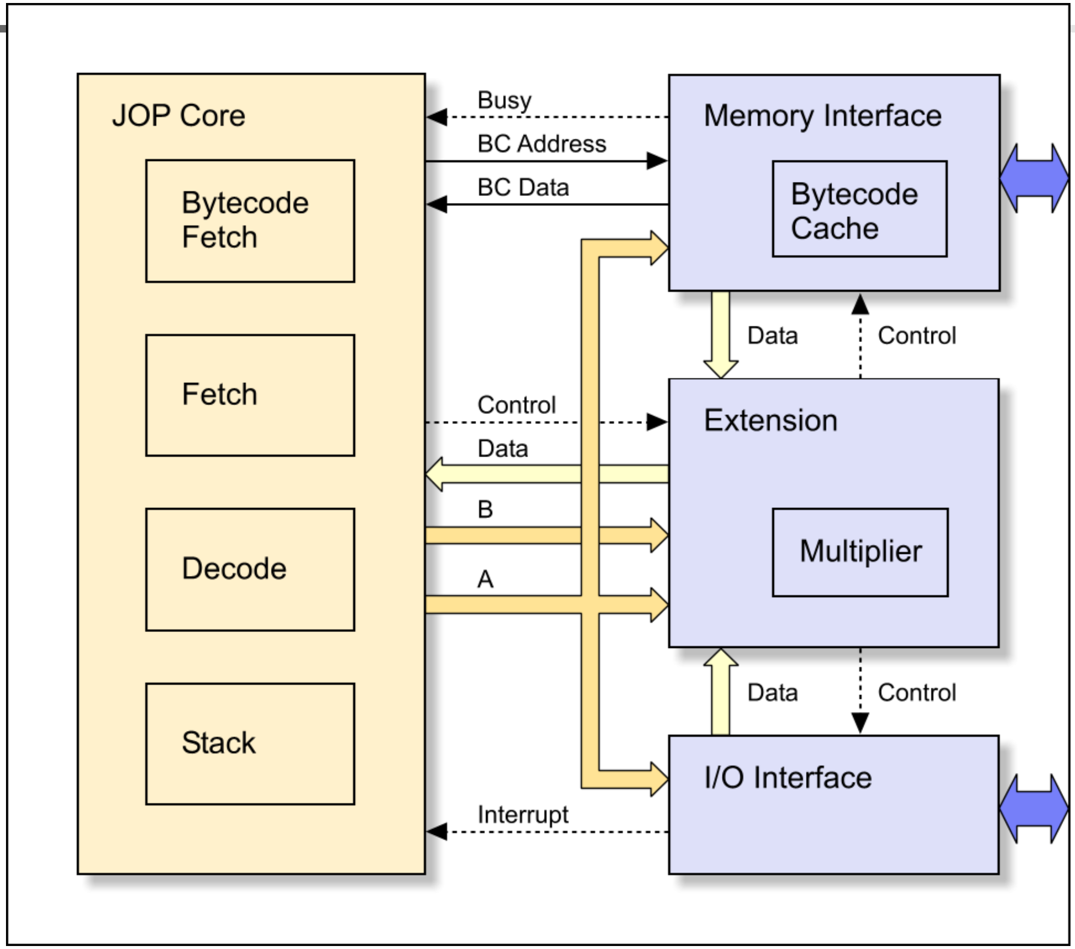
# Related Work

- picoJava
  - SUN, never released
- aJile JEMCore
  - Available, two versions
- Komodo
  - Multithreaded Java processor
- FemtoJava
  - Application specific processor

# JOP Architecture

- Overview
- Microcode
- Processor pipeline
- An efficient stack machine
- Instruction cache

# JOP Block Diagram



# JVM Bytecode Issue

- Simple and complex instruction mix
- No bytecodes for *native* functions
- Common solution (e.g. in picoJava):
  - Implement a subset of the bytecodes
  - SW trap on complex instructions
  - Overhead for the trap – 16 to 926 cycles
  - Additional instructions (115!)

# JOP Solution

- Translation to microcode in hardware
- Additional pipeline stage
- No overhead for complex bytecodes
  - 1 to 1 mapping results in single cycle execution
  - Microcode sequence for more complex bytecodes
- Bytecodes can be implemented in Java

# Microcode

- Stack-oriented
- Compact
- Constant length
- Single cycle
- Low-level HW access

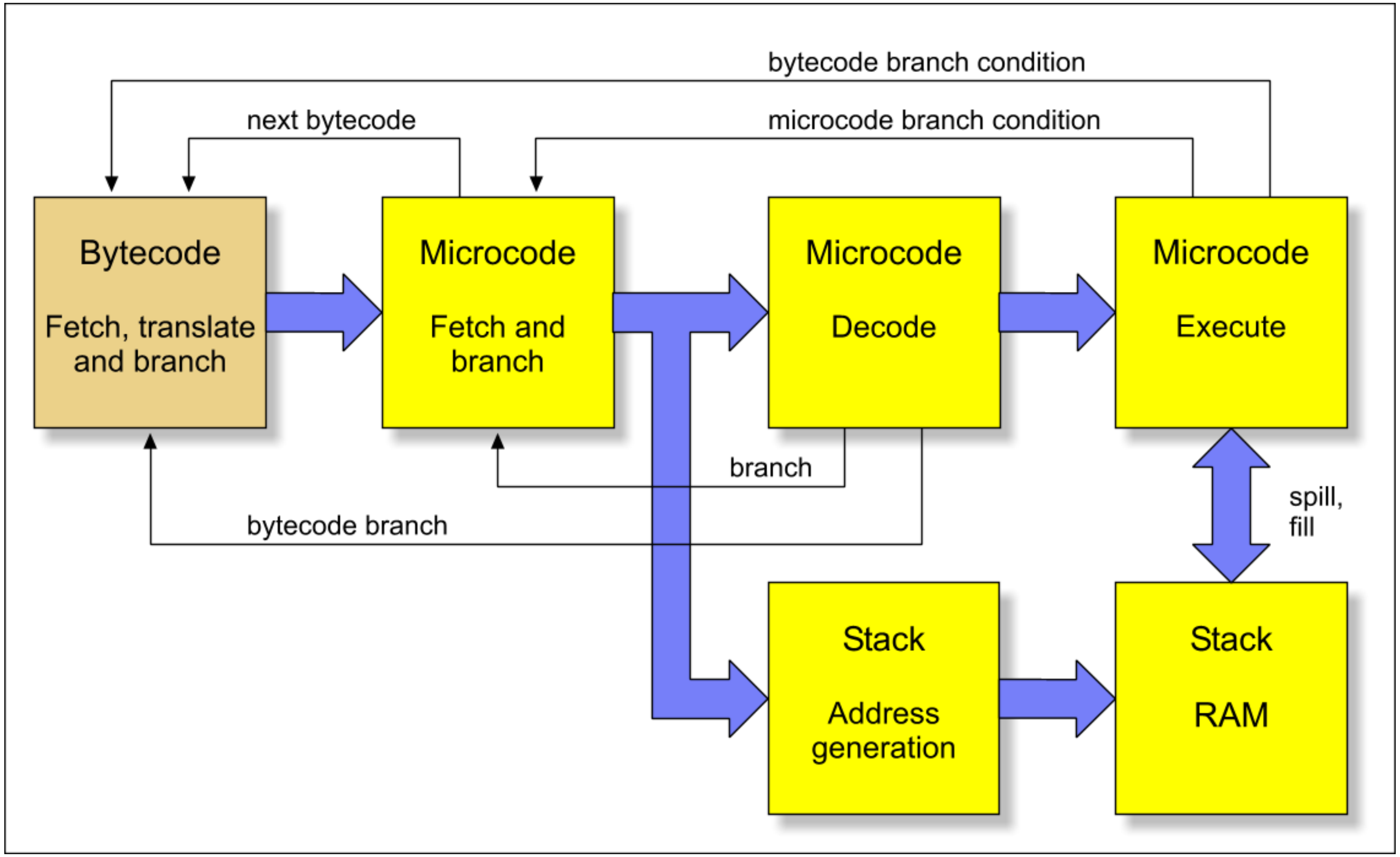
- Two examples

```
dup: dup nxt // 1 to 1 mapping

// a and b are scratch variables
// for the JVM microcode.

dup_x1: stm a      // save TOS
        stm b      // and TOS-1
        ldm a      // duplicate TOS
        ldm b      // restore TOS-1
        ldm a nxt  // restore TOS
        // and fetch next bytecode
```

# Processor Pipeline



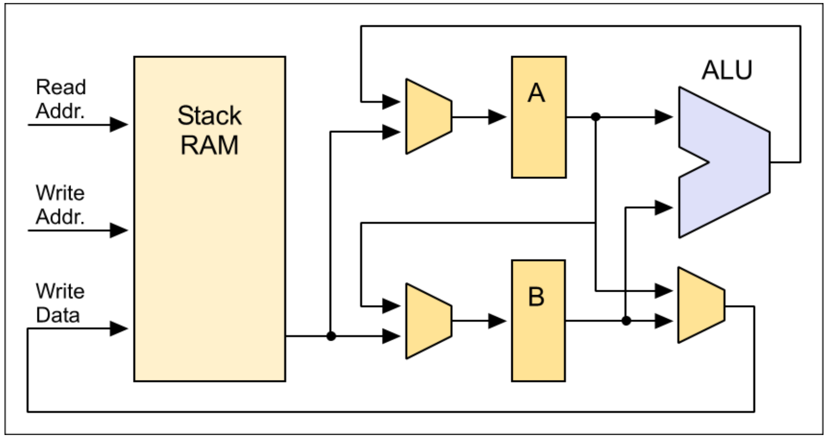
# An Efficient Stack Machine

- JVM stack is a logical stack
  - Frame for return information
  - Local variable area
  - Operand stack
- Argument-passing regulates the layout
- Operand stack and local variables need caching

# Stack Access

- Stack operation
  - Read TOS and TOS-1
  - Execute
  - Write back TOS
- Variable load
  - Read from deeper stack location
  - Write into TOS
- Variable store
  - Read TOS
  - Write into deeper stack location

# Two-Level Stack Cache




- Dual read only from TOS and TOS-1
- Two register (A/B)
- Dual-port memory
- Simple Pipeline
- No forwarding logic

- Instruction fetch
- Instruction decode
- Execute, load or store

# JVM Properties

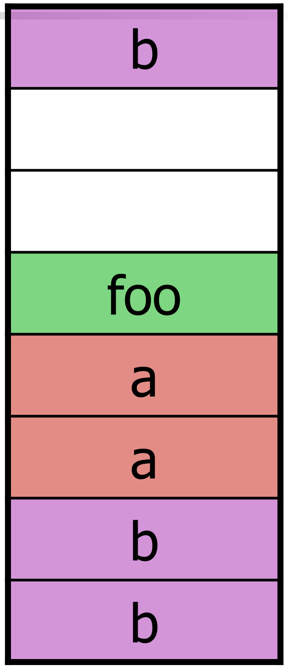
- Short methods
- Maximum method size is restricted
- No branches out of or into a method
- Only relative branches

# Proposed Cache Solution

- Full method cached
- Cache fill on call and return
  - Cache misses only at these bytecodes
- Relative addressing
  - Any position in the cache
- No fast tag memory
- Simpler WCET analysis

# Method Cache

- Whole method loaded
- Cache is divided in blocks
- Method can span several blocks
- Continuous blocks for a method
- Replacement
  - LRU not useful
  - *Free* running next block counter
  - Stack oriented next block
- Tag memory: One entry per block



# Size of Java Processors

| Processor | Resources (LC) | Memory (KB) | $f_{max}$ (MHz) |
|---|---|---|---|
| JOP | 2-3000 | 3-6 | 100 |
| Lightfoot | 3400 | 1 | 40 |
| Komodo | 2600 | ? | 33/4 (?) |
| FemtoJava | 2000 | ? | 4 (?) |
| picoJava-II | 27500 | ~45 | 40 |
| NIOS/MB | 2-3000 | ~5 | 100+ |

# Architecture Summary

- Microcode
- 1+3 stage pipeline
- Two-level stack cache
- Method cache

*The JVM is a CISC stack architecture, whereas JOP is a RISC stack architecture.*

# WCET Analysis

- WCET has to be known
  - Needed for schedulability analysis
  - Measurement usually not possible
    - Would require test of all possible cases
- Static analysis
  - Theory is mature
  - Low-level analysis is the issue

# WCET Analysis

- Path analysis
- Low-level analysis (bytecodes)
- Global low-level analysis
- WCET Calculation

# WCET Analysis for JOP

- Simple low-level analysis
- Bytecodes are independent
  - No shared state
  - No timing anomalies
- Bytecode timing is known and documented
- Simpler caches

# WCET Tool

- Execution time of basic blocks
- Annotated loop bounds (or use DFA)
- ILP problem solved
- Simple method cache analysis included
  - All methods fit in local scope
    - Single miss
  - Expand local scope

# Applications

- Kippfahrleitung
  - Distributed motor control


- ÖBB
  - Vereinfachtes Zugleitsystem
  - GPS, GPRS, supervision
- TeleAlarm
  - Remote tele-control
  - Data logging
  - Automation

# JOP in Research

- University of Lund, SE
  - Application specific hardware (Java->VHDL), HW GC
- Technical University Graz, AT
  - HW accelerator for encryption
- University of York, GB
  - Javamen – HW for real-time systems, hardware methods
- Institute of Informatics at CBS, DK
  - WCET Analyzer, embedded RT Machine Learning
- Aalborg University, DK
  - SC Java, Java HAL, Scheduling/WCET analysis with UppAal
- University of California Irvine, USA
  - WCET tool Volta
- Università della Svizzera Italiana, CH
  - Cross-profiling for embedded systems
- EU Project JEOPARD

# JOP for Teaching

- Easy access – open-source
  - Computer architecture
  - Embedded systems
- DTU: JVM in hardware
- UT Vienna
  - JVM in hardware course
  - Digital signal processing lab
- CBS, Copenhagen
  - Distributed data mining (WS 2005)
  - Very small information systems (SS 2006)

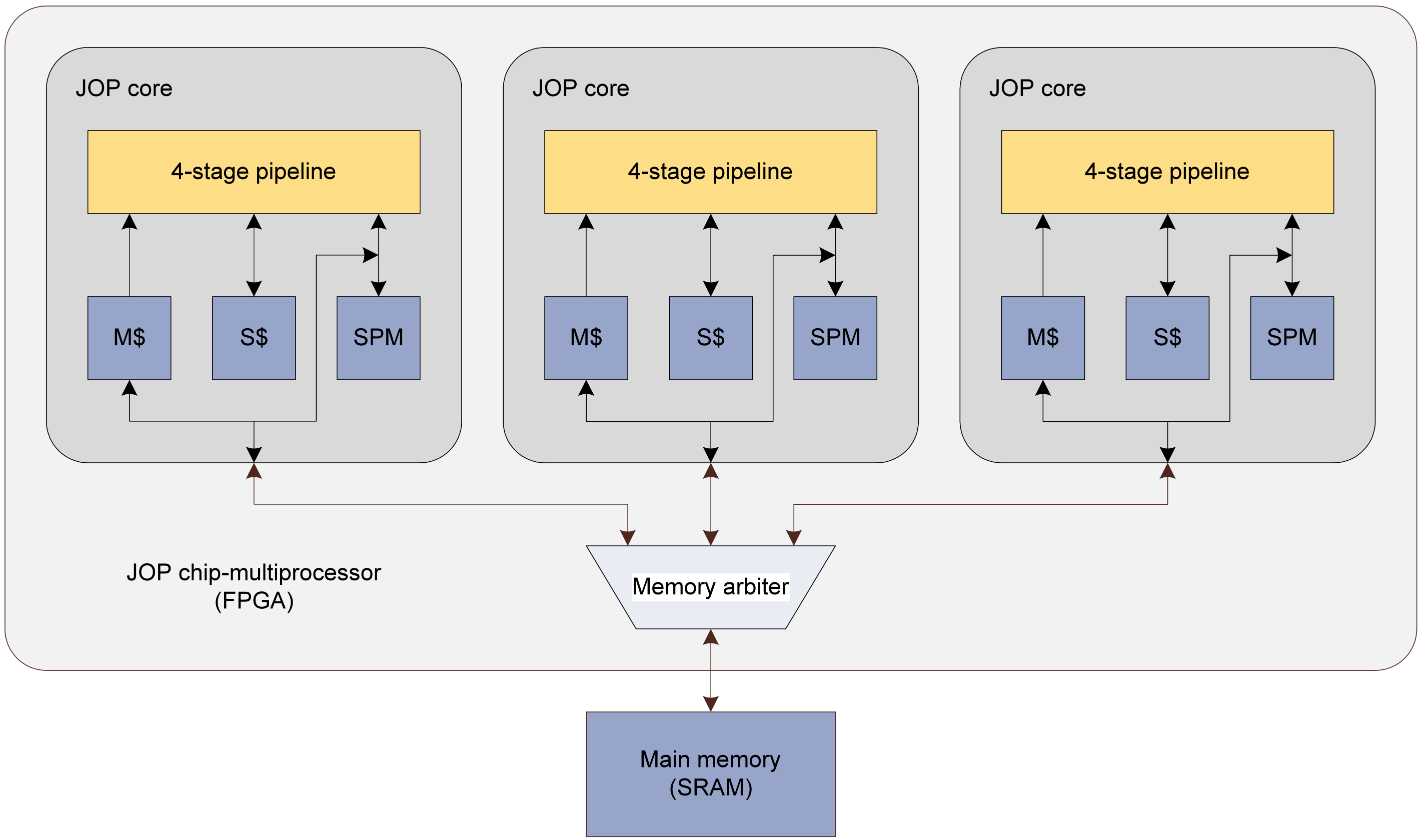
# Current/Future Work

- JOP CMP
- Analyzable D$
- Transactional memory

# Chip-Multiprocessor

- Hot topic on PC and server
- Two Flavors
  - Intel/AMD 2/4 OOO, super-scalar cores
  - 8 simple cores
    - Sun Niagara: simple 6-stage RISC
    - IBM CELL: synergistic processors
- We go the simple core approach

# JOP CMP System



# CMP Prototype

- Up to 8 cores in Cyclone-II (EP2C35)
- In Altera DE2 board
  - 90-110 MHz
- Simple synchronization
  - Global HW lock
- Pressure on memory bandwidth
  - Now we need better caching

# Caching

- Classic feature for average case throughput
- Instruction and data cache split
  - Avoid structural hazard between
    - Instruction fetch from I$
    - Load/store on D$
- Now up to three levels
  - 1st level shared in chip multi-threading
  - Next levels shard in (chip) multi-processing
  - Analysis nightmare

# Cache WCET Analysis

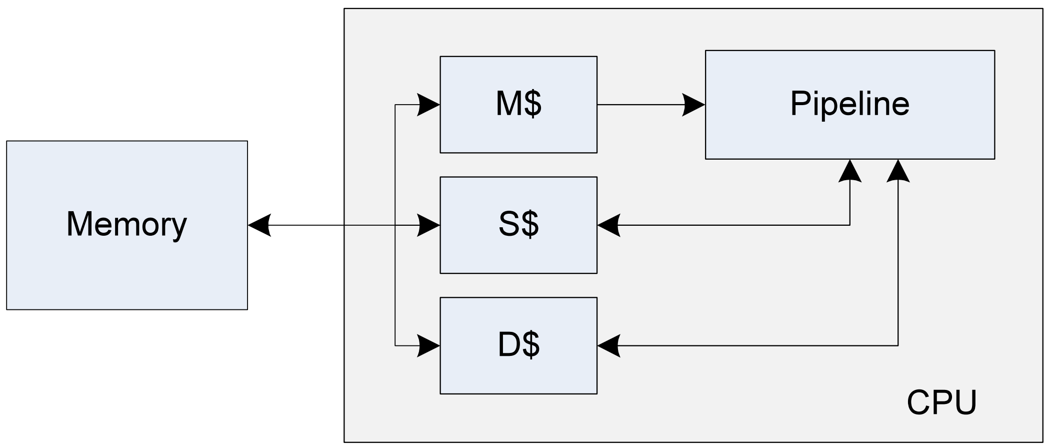
- Depends on replacement strategy
  - Direct mapped is fine, LRU is ok
  - Random, PLRU is useless
- Depends on static address estimation
  - I$ is analyzable
  - D$ is hard to analyze
- We need a new organization for D$

# D$ Issues

- Data areas
  - Static data  ok
  - Constants  ok
  - Stack data  not so hard
  - Heap data  addresses not known statically
  - A single unknown heap access destroys all known, abstract cache states from one cache way!
- Let’s split the D$!

# Cache Split

- Different caches for different areas
  - Avoid analysis influences
  - Different characteristics (size vs. associativity)
  - Independent, composable analysis



# Transactional Memory

- Automatic fine grain concurrency control
  - Simpler than locks
- Analysis of max. # retries (RTS bounds)
- Local transaction buffer (= cache)
  - Global lock on overflow
- Burst write on commit
- Status
  - Analysis published
  - Prototype implementation with JOP (Paper at FPL)

# Conclusions

- Real-time Java processor
  - Exactly known execution time of the BCs
  - Time-predictable method cache
  - WCET analysis possible
- Resource-constrained processor
  - RISC stack architecture
  - Efficient stack cache
- Platform for RT architecture research

# More Information

- JOP Thesis and source
  - http://www.jopdesign.com/thesis/index.jsp
  - http://www.jopdesign.com/download.jsp
- Various papers
  - http://www.jopdesign.com/docu.jsp
- Web sites
  - http://www.jopdesign.com/
  - http://www.jopwiki.com/

# Thank You!

Questions

and

Suggestions