JOP: A Java Optimized Processor for Embedded Real-Time Systems

Martin Schöberl

JOP Research Targets

- Java processor
- Time-predictable architecture
- Small design
- Working solution (FPGA)

Overview

- Motivation
- Research objectives
- Java and the JVM
- Related work
- JOP architecture
- Results
- Conclusions, future work

Current Praxis

- C and assembler
- Embedded systems are RT systems
- Different RTOS
- JIT is not possible
- JVM interpreter are slow
- => Java processor

Why Java?

- Safe OO language
  - No pointers
  - Type-safety
  - Garbage collection
- Built in model for concurrency
- Platform *independent*
- Very rich *standard* library

Research Objectives

- Primary objectives:
  - Time-predictable Java platform
  - Small design
  - A working processor
- Secondary objectives:
  - Acceptable performance
  - A flexible architecture
  - Real-time profile for Java

Java and the JVM

- Java language definition
- Class library
- The Java virtual machine (JVM)
  - An instruction set – the *bytecodes*
  - A binary format – the *class file*
  - An algorithm to *verify* the class file

The JVM instruction set

- 32 (64) bit stack machine
- Variable length instruction set
- Simple to very complex instructions
- Symbolic references
- Only relative branches

Memory Areas for the JVM

- Stack
  - Most often accessed
  - On-chip memory as cache
- Code
  - Novel instruction cache
- Class description and constant pool
- Heap

Implementations of the JVM

- Interpreter
- Just-in-time compilation
- Batch compilation
- Hardware implementation

Related Work

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

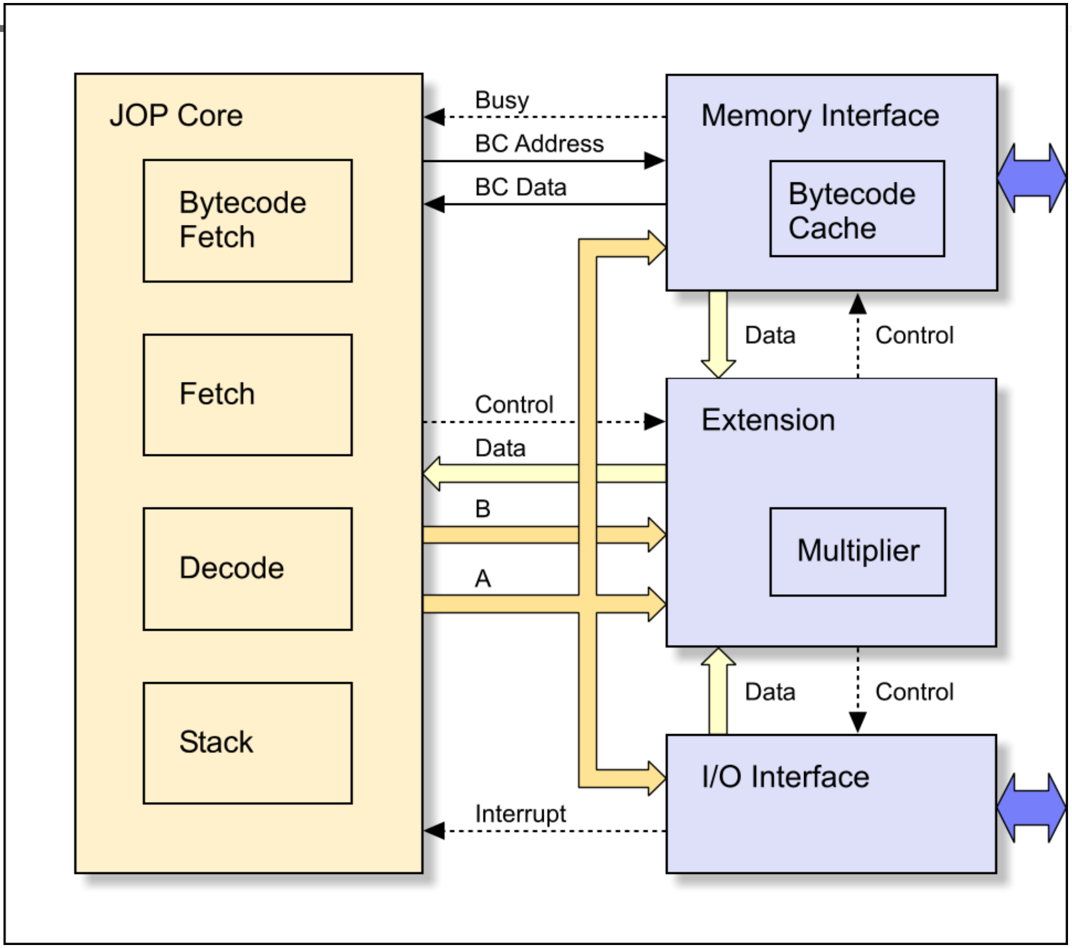
Research Objectives

	picoJava	aJile	Komodo	FemtoJava	JOP
Predictability	- -	.	-	.	+ +
Size	- -	-	+	-	+ +
Performance	+ +	+	-	- -	+
JVM conf.	+ +	+	-	- -	.
Flexibility	- -	- -	+	+ +	+ +

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

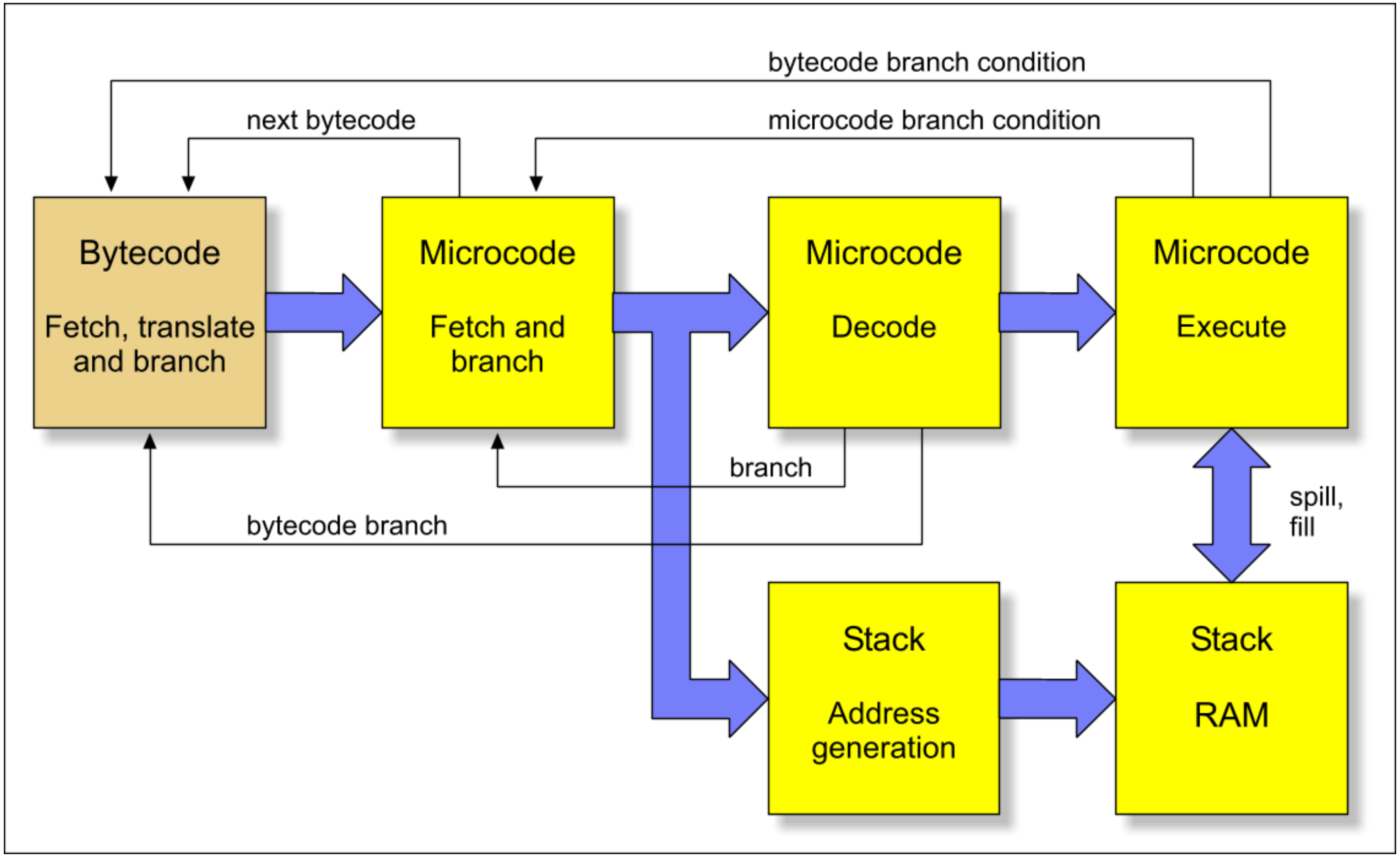
- An example

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

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

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



Interrupts

- Interrupt logic at bytecode translation
  - Special bytecode
  - Transparent to the core pipeline
- Interrupts under scheduler control
  - Priority for device drivers
  - No additional blocking time
  - Integration in schedulability analysis
  - Jitter free timer events
  - Bound to a thread

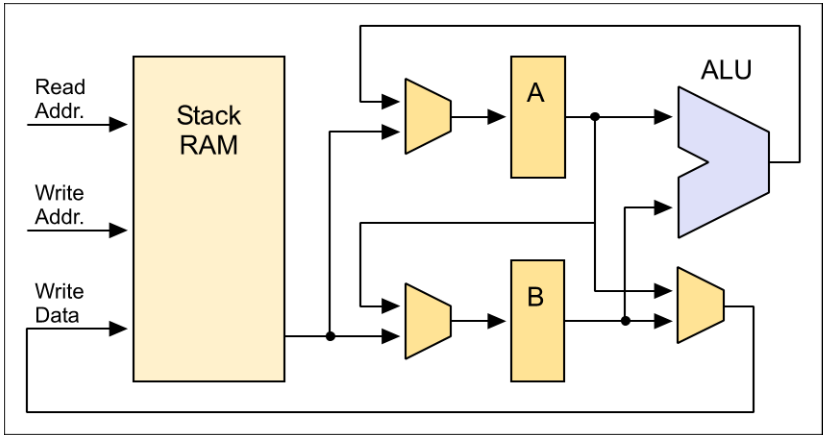
An Efficient Stack Machine

- 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
- Simpler 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
  - No address translation necessary
- No fast tag memory

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.

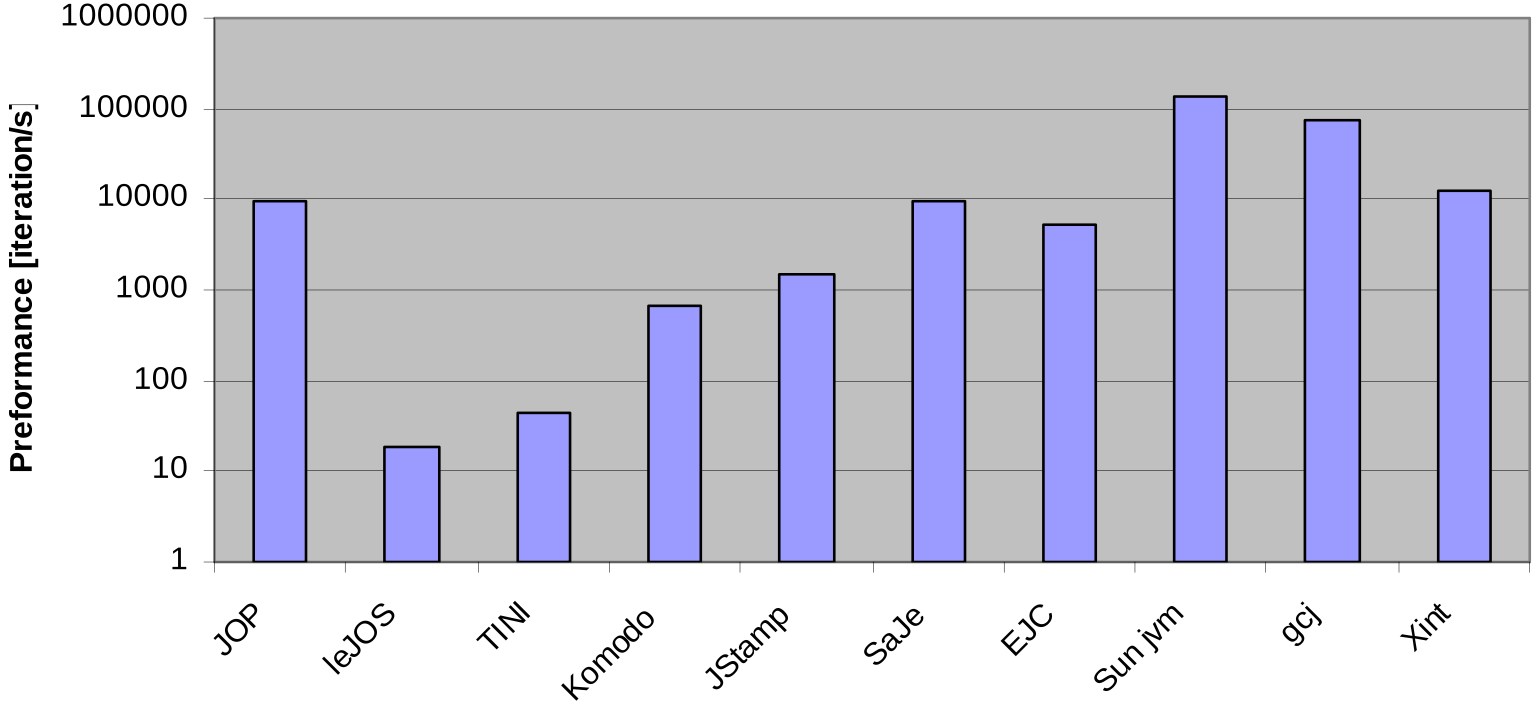
Results

- Size
  - Compared to soft-core processors
- General performance
  - Application benchmark (KFL & UDP/IP)
  - Various Java systems
- Real-time performance
  - 100MHz JOP – 266MHz Pentium MMX
  - Simple RT profile – RTSJ/RT-Linux

Size of FPGA processors

Processor	Resources [LC]	Memory [KB]	f_{max} [MHz]
JOP min.	1077	3.25	98
JOP typ.	1831	3.25	101
Lightfoot	3400	1	40
Komodo	2600	?	33/4
FemtoJava	2000	?	4
NIOS	2923	5.5	119
SPEAR	1700	8	80

Application Benchmark



Periodic Thread Jitter

Period	JOP		RTSJ/Linux	
	Min.	Max.	Min.	Max.
50 us	35 us	63 us	-	-
70 us	70 us	70 us	-	-
100 us	100 us	100 us	-	-
5 ms	5 ms	5 ms	0.017 ms	19.9 ms
10 ms	10 ms	10 ms	0.019 ms	19.9 ms
30 ms	30 ms	30 ms	29.7 ms	30.3 ms
35 ms	35 ms	35 ms	29.8 ms	40.3 ms

Context Switch

- Low priority thread records current time
- High priority periodic/event thread measures elapsed time after unblocking
- Time in cycles

	JOP		RTSJ/Linux	
	Min.	Max.	Min.	Max.
Thread	2676	2709	11529	21090
SW Event	2773	2935	63060	101292

Applications

- Kippfahrleitung
  - Distributed motor control

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

Contributions

- Real-time Java processor
  - Exactly known execution time of the BCs
  - No mutual dependency between BCs
  - Time-predictable method cache
- Resource-constrained processor
  - RISC stack architecture
  - Efficient stack cache
  - Flexible architecture

Future Work

- Real-time garbage collector
- Instruction cache WC analysis
- Hardware accelerator
- Multiprocessor JVM
- Java computer

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