Assembling for the JVM

Type safe, efficient, and low level programming for the Java Virtual Machine

Michael Wiedeking

michael.wiedeking@mathema.de

Independence

- Machine independence
- Problem independence
- Human independence
- Time independence

Daniel J. Salomon

Four Dimensions of Programming-Language Independence

ACM SIGPLAN Notices, Volume 27, No. 3, March 1992

Definition of Independence

A programming language can be said to be independent of a classification of the elements of a domain if it:

(1) supplies the same level of computational power to all groups in the classification, and

(2) meets the computational needs of each of the groups in the classification.

Architecture Independence

Applying the given definition of independence, an architecture-independent language would allow one to:

(1) Run any program written for one architecture on any other architecture.

(2) Take advantage of all the special features of any particular architecture.

A short introduction to the

AL_X Language Family

Unified Syntax Elements

- Command Statement

  keyword *token-list*

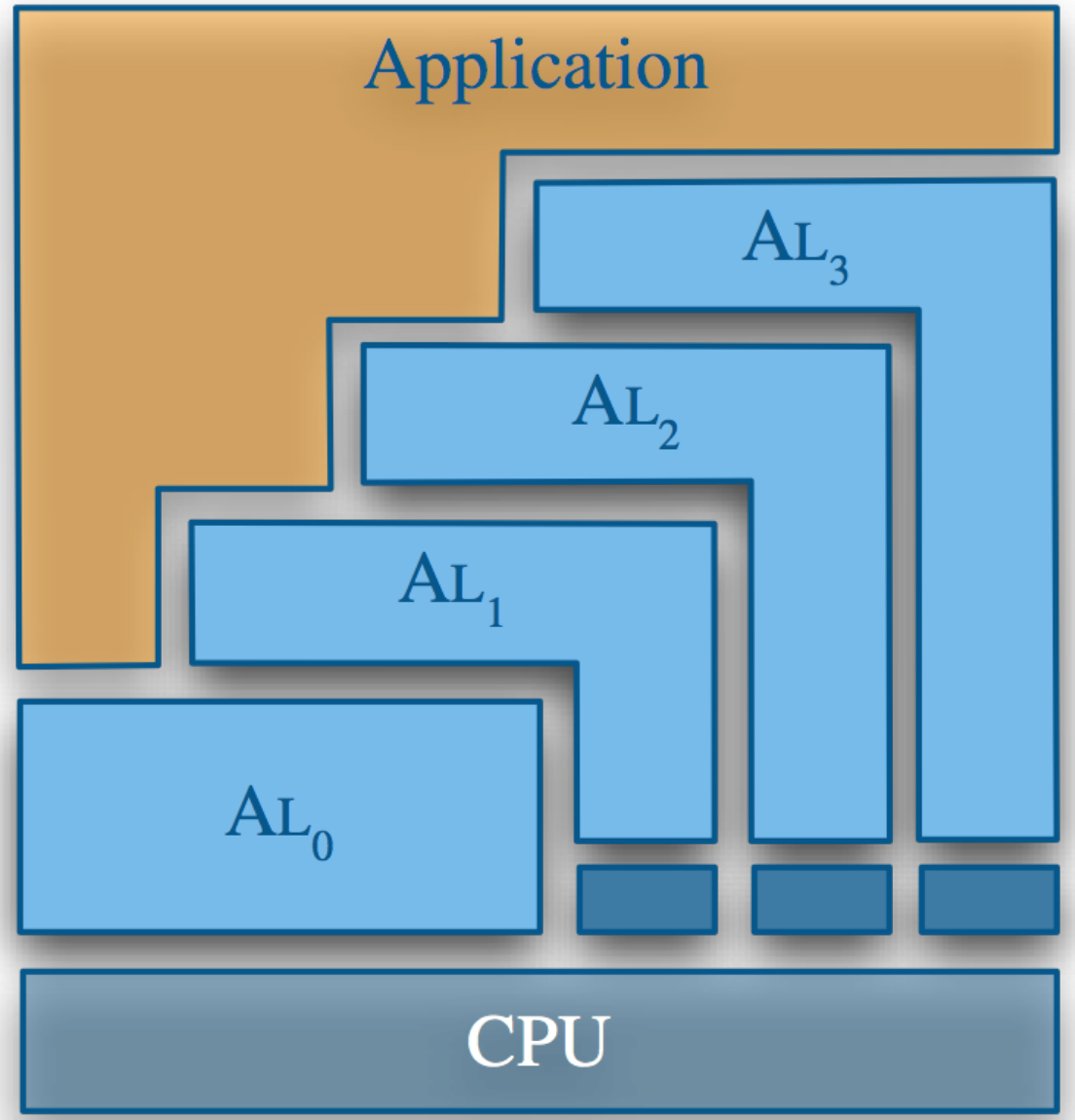
- Block Statement

  keyword [*token-list* **begin-keyword**]$^?$

  …

  end [**keyword** *optional-tokens*]$^?$

- Expression (Statement)
  - Assignment
  - Procedure call
  - Everything else



AL_x-Levels

- AL_0

  random memory access (LLVM, C, …)

- AL_1

  managed memory (JVM, CLR, …)

- AL_2

  control structures, records (Pascal, C, …)

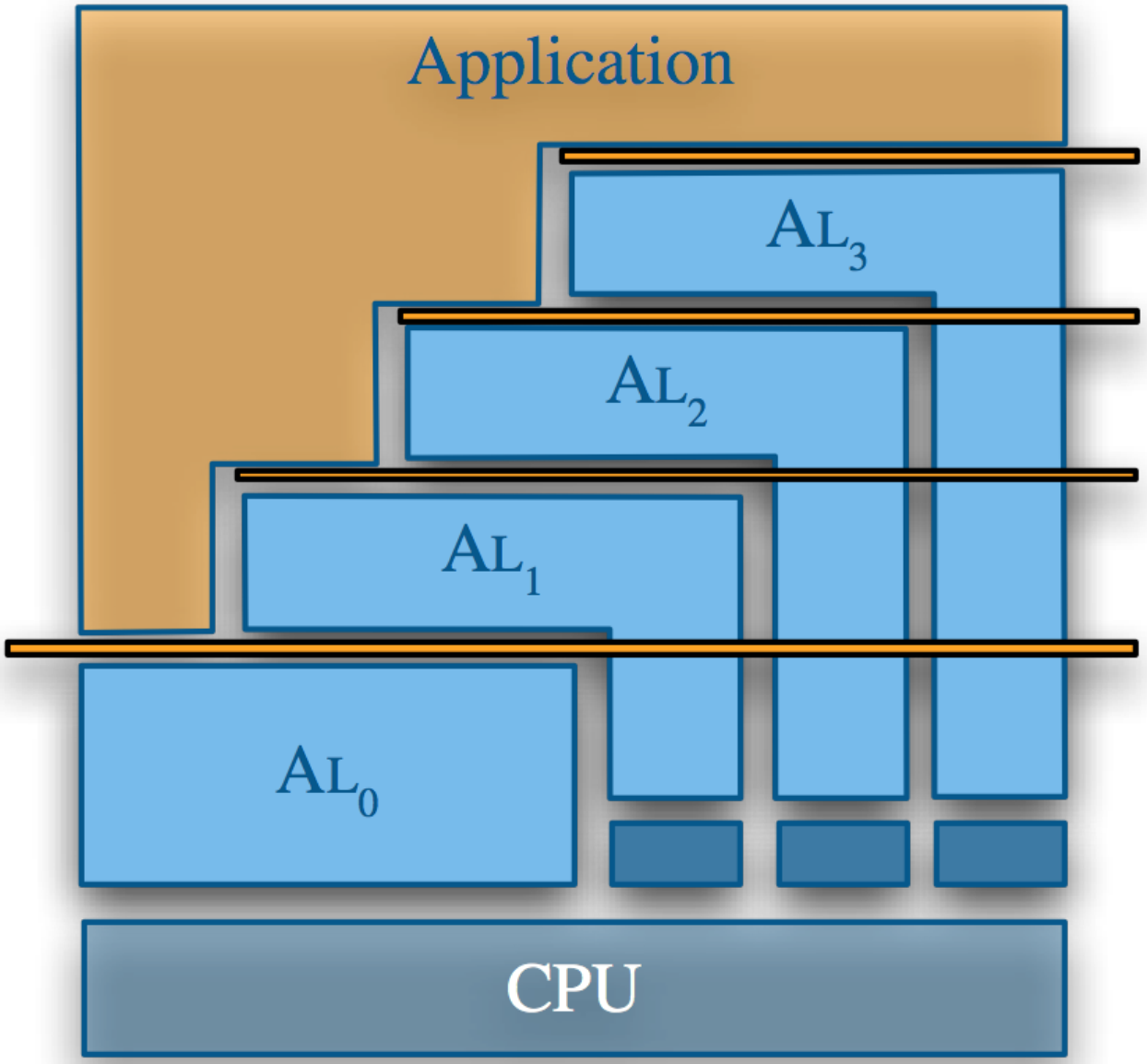
- AL_3

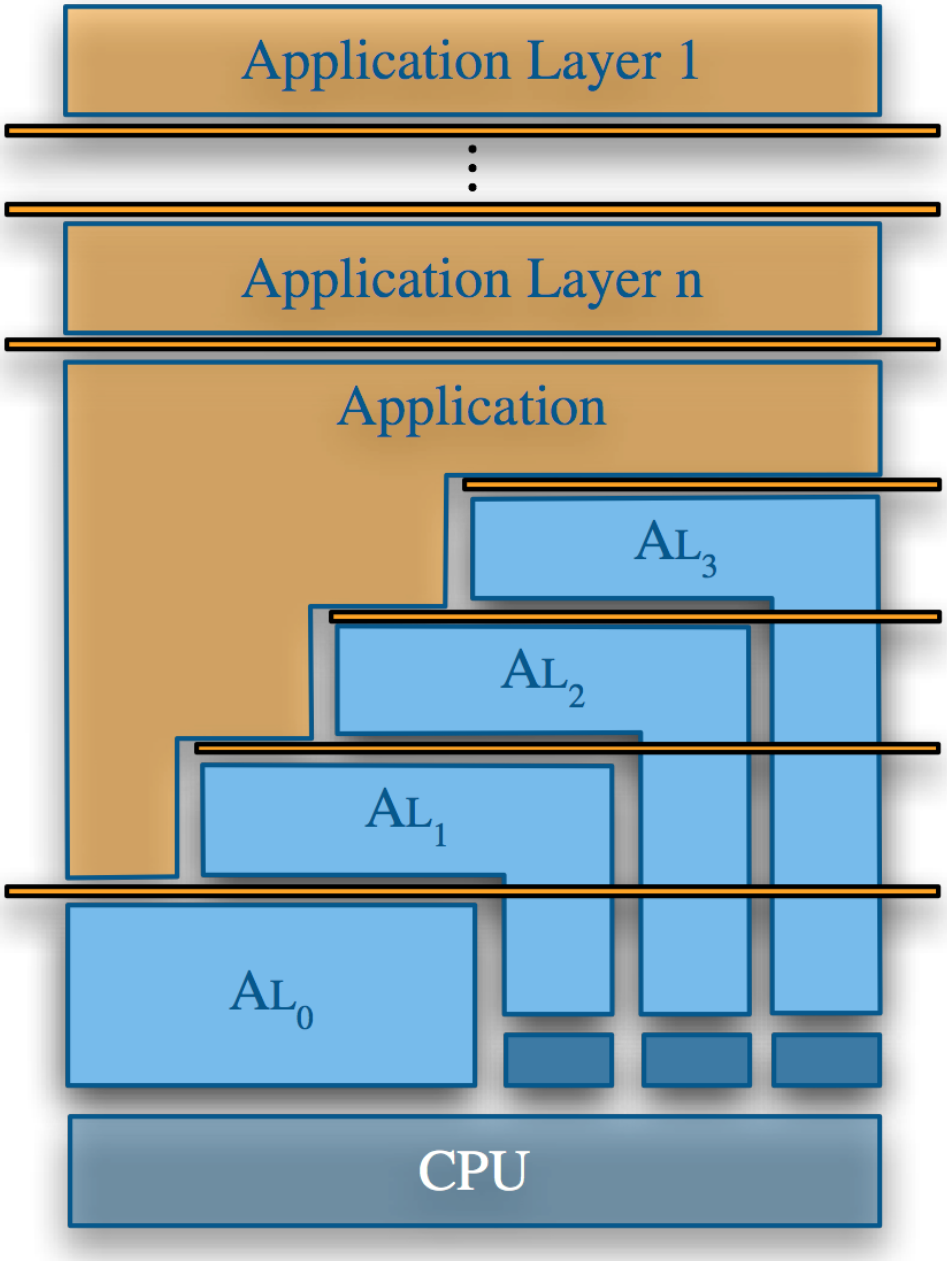
  objects (Java, C#, C++, …)

- $AL_?$

  everything else (Fortress, X10, …)

Protection Domains





Source File Format

- supported AL_x version
- XML like format
- UTF-8 encoding
- supported Unicode version and reference to Private Use Area
- owner, copyright, license, …
- signature, …

function f($x : \mathbb{Z}, y : \mathbb{N}$) $\rightarrow \mathbb{N}$

requires $\{x \leq y, x < 0 \Rightarrow y \neq 0\}$

ensures $\{\textit{result} > 0\}$

is

return $s \;/\; \sqrt{s}$ **where** $s = x^2 + y^2$

end function

$$(a, b, x_0) \mapsto x_0 \cdot \sum[1 \leq i \leq n] \sqrt{(a[i] - \mathrm{b}[i])}$$

function $\dot{:}\quad :x(: \mathbb{Z}, n : \mathbb{Z}) \to \mathbb{Z}$ **is**

- (0, 0):

  throw ArithmeticException

- (0, ?):

  return 0

- (?, 0):

  return 1

- (x, 1):

  return x

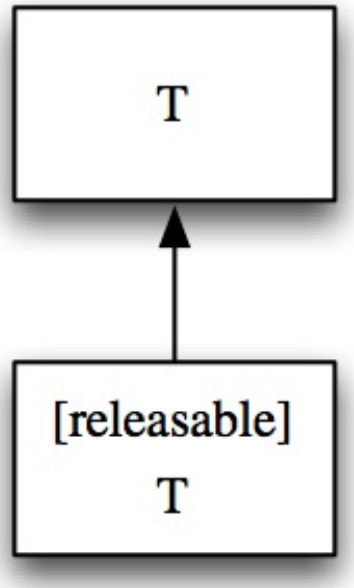
- (x, n):

  return npow(x, n)

end function

Annotated Types

- “covariant”



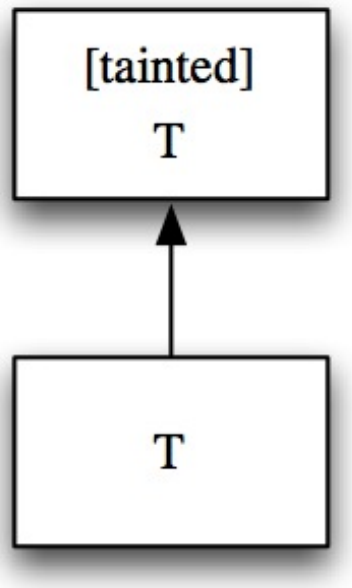
t : [releasable] T

u : T

t ← *u* // ✗

u ← *t* // ✓

- “contravariant”



t : [tainted] T

u : T

t ← *u* // ✓

u ← *t* // ✗

“Batch”

```
int limit = 500;

Remote<Mailer> connection = …;
batch (Mailer mailer : connection) {
    for (Message m : mailer.Messages) {
        if (m.Size > limit) {
            print(m.Subject + ": " + m.Sender.Name);
            m.delete();
        }
    }
}
```

"Batch"

```
limit := 500

connection : Remote[[Mailer]] ← …
batch mailer :↞ connection do
    for m :↞ mailer.messages do
        if m.size > limit then
            print(m.subject, ": ", m.sender.name);
            m.delete();
        end if
    end for
end batch
```

syntax Batch **extends** Block **is**

‘**batch**’ *e* : Expr [[Remote [[?]]]] ‘**do**’

b : Block

end(‘batch’)

semantics

end syntax

syntax Unless **extends** Statement **is**

’**unless**‘ *condition* : $\mathscr{E}[\![\mathbb{B}]\!]$ ’**then**‘

block : $\mathscr{B}$

⌈’**else**‘ ⌉?

⌊ *else_block* : $\mathscr{B}$ ⌋

end(’**unless**‘)

semantic …

…

end syntax

semantic in terms of AL_0

if *else_block*.isDefined() then

($t :\leftarrow$ **(**|*condition*.eval()|**)**

(if notFalse(t) **then goto** ①↓ **end)**

((|*block*.eval()|**)**

(goto ⓪↓**)**

(①:**)**

((|*else_block*.eval()|**)**

(⓪:**)**

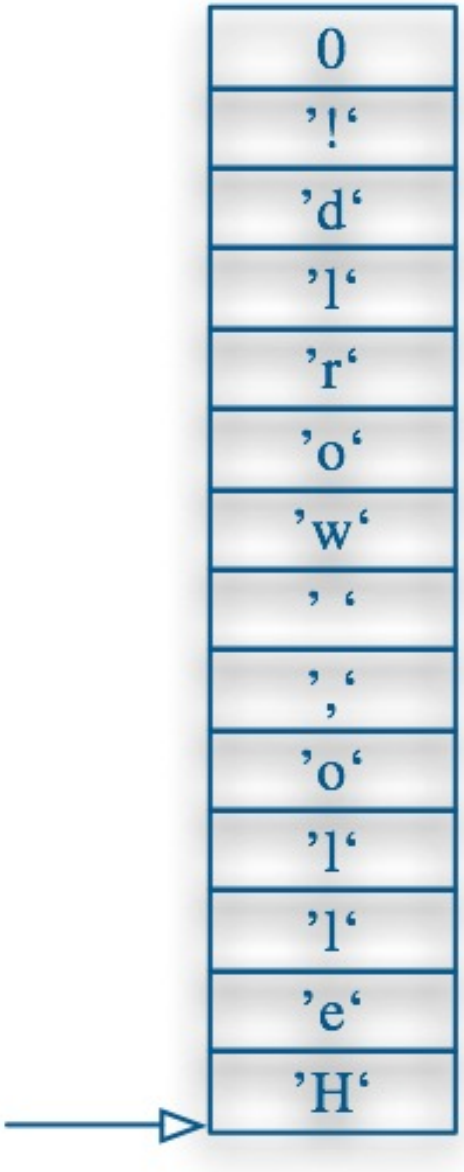
else

```
else
        ⦇t :← ⦇condition.eval()⦈⦈
        ⦇if notFalse(t) then goto ⓪↓ end⦈
            ⦇⦇block.eval()⦈⦈
        ⦇⓪:⦈
    end if
end syntax
```

A short Introduction to AL_1

Hello World

```
namespace //mathema.de/hello/Hello

import //alx/compatibility/c/Stdio/print

[[read_only]]
data DataSegment is
    greetings : Sq[[Byte]] ← "Hello, world!" ∘ 0
end data

callable main() is

    s⃗ :← ⊕greetings

    print(s⃗)

end callable
```



- $\mathbb{M}$ MachineWord
- $\mathbb{M}_1$ Byte
- $\mathbb{M}_2$ Wyde
- $\mathbb{M}_4$ Tetra
- $\mathbb{M}_8$ Octa
- $\mathbb{M}_{16}$ Hexadec
- $\mathbb{M}_{32}$ Doxadex
- $\mathbb{M}_{64}$ Texadex
- $\mathbb{M}_{128}$ Oxadex
- $\mathbb{M}_{10}$ Deca
- $\mathbb{M}_{12}$ Dodeca
- $\mathbb{A}$ Address
- $\mathbb{F}_2$ HalfFloat
- $\mathbb{F}_4$ SingleFloat
- $\mathbb{F}_8$ (Double)Float
- $\mathbb{F}_{10}$ ExtendedFloat
- $\mathbb{F}_{16}$ QuadrupleFloat

data DataSegment **is**

$a_0 : \text{Byte} \leftarrow (01)_{16}$

$a_1 : \text{Byte} \leftarrow (23)_{16}$

$a_2 : \text{Byte} \leftarrow (45)_{16}$

$a_3 : \text{Byte} \leftarrow (67)_{16}$

end data

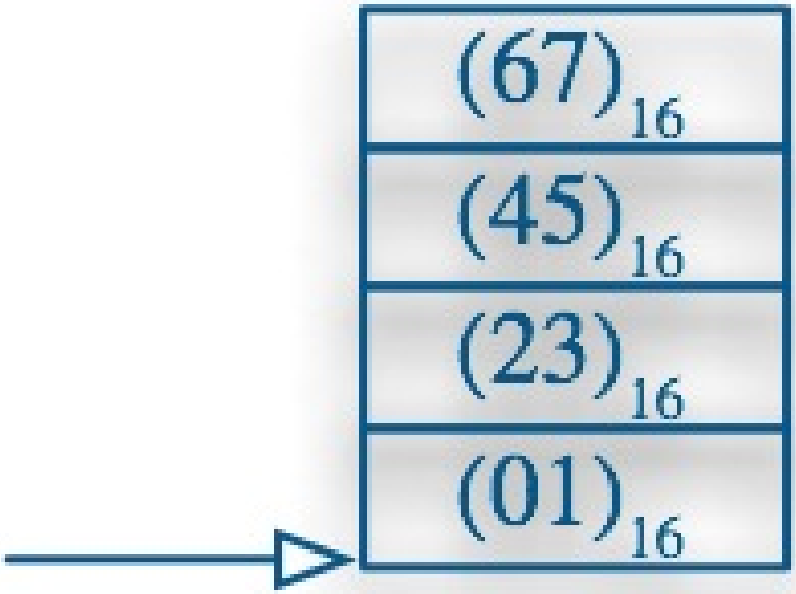
callable main() **is**

$t :\leftarrow \text{load}[\![\text{Tetra}]\!](\oplus a_0)$

// $t = (67452301)_{16}$ (*little-endian*)

// $t = (01234567)_{16}$ (*big-endian*)

end callable



assume *config*.*byte_order* = BIG_ENDIAN

data DataSegment **is**

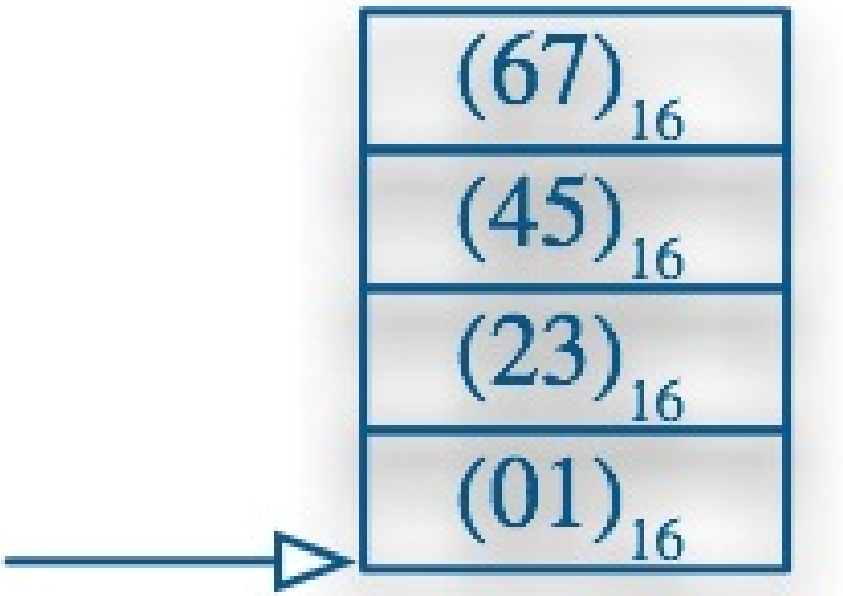
$a_0 : \text{Byte} \leftarrow (01)_{16}$

$a_1 : \text{Byte} \leftarrow (23)_{16}$

$a_2 : \text{Byte} \leftarrow (45)_{16}$

$a_3 : \text{Byte} \leftarrow (67)_{16}$

end data



callable main() **is**

$t :\leftarrow \text{load}[\![\text{Tetra}]\!](\oplus a_0)$

// $t = (01234567)_{16}$

end callable

- Tetra
  - ⟦const⟧ *width* : ℕ[Bit] = 32
  - ⟦const⟧ *size* : ℕ[Byte] = 4
  - ⟦const⟧ *element_size* : ℕ[Byte] = 4
  - ⟦const⟧ *alignment* : ℕ = 4
  - *signed* : $\mathbb{I}_{32}$
  - *unsigned* : $\mathbb{U}_{32}$
  - *high* : Wyde
  - *low* : Wyde
  - *wydes* : Sq⟦Wyde, 2⟧
  - *big_endian.wydes* : Sq⟦Wyde, 2⟧
  - *little_endian.wydes* : Sq⟦Wyde, 2⟧
  - *bytes* : Sq⟦Byte, 4⟧
  - *big_endian.bytes* : Sq⟦Byte, 4⟧
  - *big_endian.bytes* : Sq⟦Byte, 4⟧
  - *bits* : BitSq⟦32⟧
  - *big_endian.bits* : BitSq⟦32⟧
  - *little_endian.bits* : BitSq⟦32⟧

[[inline]]

callable markZeroBytes($x : \mathbb{M}$) $\rightarrow \mathbb{M}$ **is**

$M := (7\text{F}\ldots7\text{F})_{16}$

$r :\leftarrow \text{and}(x, M)$

$r \leftarrow \text{addUnsigned}(r, M)$

$r \leftarrow \text{or}(r, x)$

$r \leftarrow \text{or}(r, M)$

$r \leftarrow \text{invert}(r)$

return r

end callable

```
[[inline]]
callable findZeroByte(x : ℕ) → ℕ is
    r :← markZeroBytes(x)
    variant
    ■ “big endian” …
        …
    ■ “little endian” …
        …
    end variant
    return r
end callable
```

variant

- "big endian"

  requires {*config*.*byte_order* = BIG_ENDIAN}:

  nlz :← numberOfLeadingZeroBits(*r*)

  r ← shiftRightUnsigned(*nlz*, 3)

- "little endian"

  requires {*config*.*byte_order* = LITTLE_ENDIAN}:

  ntz :← numberOfTrailingZeroBits(*r*)

  r ← [[unchecked]] shiftRight(*ntz*, 3)

end variant

Predicates

- zero
- nonZero
- positive
- nonPositive
- negative
- nonNegative
- even
- odd

Relations

- equal
- nonEqual
- less
- lessOrEqual
- greater
- greaterOrEqual
- lessUnsigned
- lessOrEqualUnsigned
- greaterUnsigned
- greaterOrEqualUnsigned

(Conditional) Jumps and Assignment

- unconditional jump

  goto L

- conditional jump

  if condition **then**
    goto L
  end if

- conditional assignment

  if condition **then**
    $v \leftarrow r$
  end if

A Note on Labels

goto ①↓

①:

goto ①↑

①:

goto ①↑

Basic Functions

- abs
- nabs
- sign
- min
- max
- pow
- sqrt
- log
- log

- signUnsigned
- minUnsigned
- maxUnsigned
- powUnsigned
- sqrtUnsigned

Bit Manipulation

- invert(*x*)
- and(*x*, *y*)
- or(*x*, *y*)
- xor(*x*, *y*)
- andNot(*x*, *y*)
- orNot(*x*, *y*)
- xorNot(*x*, *y*)
- nand(*x*, *y*)
- nor(*x*, *y*)
- nxor(*x*, *y*)

Bit Counts

- numberOfLeadingZeroBits(x)
- numberOfTrailingZeroBits(x)
- numberOfOneBits(x)

- numberOfOneBits.*is_native*

- add($x : \mathbb{I}, y : \mathbb{I}$) $\rightarrow \mathbb{I}$ **throws** OverflowException
- addUnsigned($x : \mathbb{U}, y : \mathbb{U}$) $\rightarrow \mathbb{U}$
- addWithCarry($x : \mathbb{I}, y : \mathbb{I}, c : \mathbb{J}\{0, 1\}$) $\rightarrow (\mathbb{I}, \mathbb{J}\{0, 1\})$
- addWithCarryUnsigned($x : \mathbb{U}, y : \mathbb{U}, c : \mathbb{J}\{0, 1\}$) $\rightarrow (\mathbb{U}, \mathbb{J}\{0, 1\})$

- [[unchecked]] add($x : \mathbb{I}, y : \mathbb{I}$) $\rightarrow \mathbb{I}$
- [[checked]] addUnsigned($x : \mathbb{U}, y : \mathbb{U}$) $\rightarrow \mathbb{U}$
  throws OverflowException

Shift

- $\text{shift}_2(x, i)$ **throws** OverflowException
- $\text{shift}_{10}(x, i)$ **throws** OverflowException
- shiftLeft(x, n) **throws** OverflowException
- shiftLeftUnsigned(x, n)
- shiftRight(x, n)
- shiftRightUnsigned(x, n)

Division and Shift

- $x / 2^n$ should be identical to $x \gg n$
  (is not true for Java et al. if $x < 0$)
- different division and modulo functions
  - quot(x, y), mod(x, y) // $\lfloor x / y \rfloor$ $\longrightarrow -\infty$
  - ratio(x, y), residue(x, y) // round(x / y) $\longrightarrow$ nearest
  - group(x, y), pad(x, y) // $\lceil x / y \rceil$ $\longrightarrow +\infty$
  - div(x, y) // sign($x \cdot y$) $\cdot$ quot($|x|$, $|y|$) $\longrightarrow 0$

Optimized Offsets

- times2plus(*x*, *y*)
- times4plus(*x*, *y*)
- times8plus(*x*, *y*)
- times16plus(*x*, *y*)
- times32plus(*x*, *y*)
- times64plus(*x*, *y*)
- …

The Syntactical Exceptions

- Iverson operator

$$[\,.\,] : (\mathbb{B}) \to \mathbb{J}$$

$$\text{sign}(x, y) \mapsto [x > y] - [x < y]$$

- Address-of operator

$$\oplus. : ([\![\text{var}]\!]\ \text{T}) \to \text{Pt}\,[\![\text{T}]\!]$$

```
data X is
    x : Tetra
end data
v ← load [[Tetra]] (⊕x) … ¶ v ← X:x
```

Standard Annotations

- uncached
- unchecked
- checked
- tail_call
- unaligned
- volatile
- …

AL_1 and the JVM

Conditional branches

if $(x = a) \vee (x = b)$ **then**

$\cdots$

end if

```
if ((a == x) | (x == b)) {
    ...
}
```

if $([x = a] \otimes [x = b]) \neq 0$ **then**

$\cdots$

end if

Implementing [$x < y$]

```
0:    iload x
1:    iload y
2:    if_icmpge  +7          (9)
5:    iconst_1
6:    goto   +4              (10)
9:    iconst_0
10:   ireturn
```

Implementing [$x < y$]

```
0:    iload x
1:    iload y
2:    if_icmpge  +5          (7)
5:    iconst_1
6:    ireturn
7:    iconst_0
8:    ireturn
```

Implementing [$x < y$]

```
0:    iload x
1:    iload y
2:    iconst_1
3:    istore z
4:    if_icmplt  +5          (9)
7:    iconst_0
8:    istore z
9:    iload z
10:   ireturn
```

Implementing [$x < y$]

```
0:    iconst_0
1:    iload x
2:    iload y
3:    if_icmpge  +5          (8)
6:    iconst_1
7:    iadd
8:    ireturn
```

Implementing the *long* Version of [$x < y$]

```
0:    lload x
1:    lload y
2:    lcmp
3:    ldc2_w 63
6:    lushr
7:    lreturn
```

compareAndSet(int a, int b, int c_1, int c_2)

```
0:  iload_0
1:  iload_1
2:  invokestatic compare(II)I
5:  bipush 31
7:  iushr
8:  istore  5
10: iinc    5, -1
13: iload   5
15: iload_2
16: iload_3
17: isub
18: iand
19: istore  5
21: iload   5
23: iload_3
24: iadd
25: istore  5
27: iload   5
29: ireturn
```

int x = Integer.compare(a, b) >>> 31;

$x = x - 1$;

$x = x \,\&\, (c_1 - c_2)$;

$x = x + c_2$;

return x;

compareAndSet(int a, int b, int c_1, int c_2)

```
int x;
if (a < b) {
    x = c1;
} else {
    x = c2;
}
return x;
```

```
 0:  iload_0
 1:  iload_1
 2:  if_icmpge  11
 5:  iload_2
 6:  istore   5
 8:  goto     14
11:  iload_3
12:  istore   5
14:  iload    5
16:  ireturn
```

return $(a < b)\ ?\ c_1 : c_2$

```
 0:  iload_0
 1:  iload_1
 2:  if_icmpge  9
 5:  iload_2
 6:  goto     10
 9:  iload_3
10:  ireturn
```

Chapter 3 – Compiling for the Java Virtual Machine

3.3 Arithmetic

… (Recall that ~x == -1^x.) …

…
iconst_m1
ixor
…

Tim Lindholm, Frank Yellin, Gilad Bracha, Alex Buckley

The JavaTM Virtual Machine Specification – Java SE 7 Edition

Oracle, 2012-02-06

Questions?

Contact

Michael Wiedeking
michael.wiedeking@mathema.de

MATHEMA Software GmbH
Henkestraße 91
91052 Erlangen
Germany