

Beispiel – Java Virtual Machine

Dr.-Ing. Volkmar Sieh

Department Informatik 4
Verteilte Systeme und Betriebssysteme
Friedrich-Alexander-Universität Erlangen-Nürnberg

WS 2016/2017



„Java Virtual Machine“ (JVM): konsequentes Design für hohe JIT-Performance:

- gut spezifizierte, einfache Instruktionen, eine Adressierungsart
- leicht emulierbare Instruktionen
- kaum Exceptions (nur Div-by-Zero, NULL-Pointer)
- keine Interrupts statt dessen Multi-Threading
- keine Condition-Code-Flags
- keine MMU, keine Segmentierung, sichere Pointer
- keine berechneten Sprünge
- kein selbst-modifizierender Code



| byte | short | int | long | float | double | char | reference |
|---------|---------|---------|---------|---------|---------|---------|-----------|
| bipush | sipush | | | | | | |
| | | iconst | lconst | fconst | dconst | | aconst |
| | | iload | lload | fload | dload | | aload |
| | | istore | lstore | fstore | dstore | | astore |
| baload | saload | iaload | laload | faload | daload | caload | aaload |
| bastore | sastore | iastore | lastore | fastore | dastore | castore | aastore |



| byte | short | int | long | float | double | char | reference |
|------|-------|------|-------|-------|--------|------|-----------|
| | | iinc | | | | | |
| | | iadd | ladd | fadd | dadd | | |
| | | isub | lsub | fsub | dsub | | |
| | | imul | lmul | fmul | dmul | | |
| | | idiv | ldiv | fdiv | ddiv | | |
| | | irem | lrem | frem | drem | | |
| | | ineg | lneg | fneg | dneg | | |
| | | ishl | lshl | | | | |
| | | ishr | lshr | | | | |
| | | ushr | lushr | | | | |
| | | iand | land | | | | |
| | | ior | lor | | | | |
| | | ixor | lxor | | | | |



Java Virtual Machine – Instruktionen

| byte | short | int | long | float | double | char | reference |
|------|-------|-----------|---------|---------|---------|------|-----------|
| i2b | i2s | | i2l | i2f | i2d | | |
| | | l2i | | l2f | l2d | | |
| | | f2i | f2l | | f2d | | |
| | | d2i | d2l | d2f | | | |
| | | | lcmp | | | | |
| | | | | fcmpl | dcmpl | | |
| | | | | fcmpg | dcmpg | | |
| | | ifOP | | | | | |
| | | if_icmpOP | | | | | if_acmpOP |
| | | ireturn | lreturn | freturn | dreturn | | areturn |



Java Virtual Machine – Instruktionen

| | | | | |
|---------------|-----------------|---------------|----------------|------|
| new | newarray | anewarray | multianewarray | |
| getfield | putfield | getstatic | putstatic | |
| arraylength | instanceof | checkcast | | |
| pop | pop2 | dup | dup2 | |
| dup_x1 | dup2_x1 | dup_x2 | dup2_x2 | swap |
| ifeq | iflt | ifle | ifne | |
| ifgt | ifge | ifnull | ifnonnull | |
| if_icmpeq | if_icmpne | if_icmplt | if_icmpgt | |
| if_icmple | if_icmpge | if_acmpeq | if_acmpne | |
| tableswitch | lookupswitch | goto | goto_w | |
| jsr | jsr_w | ret | | |
| invokevirtual | invokeinterface | invokespecial | invokestatic | |
| athrow | | | | |
| monitorenter | monitorexit | | | |



```
// a = -(b + c);  
...  
iload_1    // push variable 1  
iload 20   // push variable 20  
iadd       // add variables  
ineg       // negate result  
istore_2   // pop as variable 2  
...
```



```
// for (i = n; i >= 0; i--) { XXX }  
iload 4  
istore 3  
goto L15
```

L14:

```
XXX  
iinc 3 by -1
```

L15:

```
iload 3  
ifge L14
```



Code-Prüfung: Idee:

- Wenn eine bestimmte Code-Zeile ausgeführt wird,
 - liegen immer die gleichen Typen von Argumente auf dem Stack,
 - liegen immer die gleichen Typen in den lokalen Variablen.egal auf welchem Weg die Code-Zeile erreicht wurde.
- Es liegen die richtigen Typen von Argumenten auf dem Stack,
 - wenn eine Rechenoperation ausgeführt wird,
 - wenn eine Unterprogramm aufgerufen wird.

=> sichere Pointer, kein selbst-modifizierender Code



JVM Byte-Code wurde als

- leicht emulierbarer,
- leicht zu compilierender

Code entwickelt.

Er ist ungeeignet für Hardware! Z.B. sind

- die Garbage-Collection oder
- die Code-Prüfung

praktisch unmöglich in Hardware zu bauen.



Lizenz: „kostenlos“

Mehr Infos unter

Allgemeines: <http://java.sun.com>

JVM Interna: <http://java.sun.com/docs/books/jvms/index.html>

