Chapter 1: Why Build Another Programming Language?
Chapter 2: Programming Language Design

- assignment
  - += -= *= /= %=

- conditional
  - ? :

- logical
  - ||
  - &

- bitwise_logical
  - /
  - &

- relational
  - == !=
  - <= >=
  - instanceof

- bitwise_shifts
  - << >> >>>

- arithmetic
  - + -
  - * / %

- unary_prefix
  - ++ -- + - ! ~ (type)

- unary_suffix
  - ++ --

- ultimate
  - ( ) [ ] .
Chapter 3: Scanning Source Code
Chapter 4: Parsing

- Parse stack: $S_0$, $S_i$, $S_j$, $S_k$
  - what parser has seen (shifted or reduced)

- Input:
  - public
  - void
  - main
  - input
  - remaining/waiting to be read

- Symbols: $\$
Chapter 5: Syntax Trees
Chapter 6: Symbol Tables

```java
public class xy5 {
    static int y = 5;
    public static void main(String argv[]) {
        int x;
        x = y + 5;
        System.out.println("y + 5 = " + x);
    }
}
```
Chapter 7: Checking Base Types

```java
dpublic class hello {
    public static void main(String argv[]) {
        int x;
        x = 0;
        x = x + "hello";
        System.out.println("hello, jzero!");
    }
}
```

Line 4: typecheck = on a int and a int -> OK
Line 5: typecheck + on a String and a int -> FAIL
Chapter 8: Checking Types on Arrays, Method Calls, and Structure Accesses

```java
public class funtest {
    public static int foo(int x, int y, String z) {
        return 0;
    }
    public static void main(String argv[]) {
        int x;
        x = foo(0, 1, "howdy");
        x = x + 1;
        System.out.println("hello, jzero!");
    }
}
```

> java ch8.j0 funtest.java
line 3: typecheck return on a int and a int -> OK
checking the type of a call to foo
line 7: typecheck param on a String and a String -> OK
line 7: typecheck param on a int and a int -> OK
line 7: typecheck param on a int and a int -> OK
line 7: typecheck = on a int and a int -> OK
line 8: typecheck + on a int and a int -> OK
line 8: typecheck = on a int and a int -> OK
line 9: typecheck param on a String and a String -> OK
no errors
Chapter 9: Intermediate Code Generation

<table>
<thead>
<tr>
<th>Opcode</th>
<th>C equivalent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD, SUB, MUL, DIV</td>
<td>x = y op z</td>
<td>Store result of binary operation on y and z to x</td>
</tr>
<tr>
<td>NEG</td>
<td>x = -y</td>
<td>Store result of unary operation on y to x</td>
</tr>
<tr>
<td>ASN</td>
<td>x = y</td>
<td>Store y to x</td>
</tr>
<tr>
<td>ADDR</td>
<td>x = &amp;y</td>
<td>Store address of y to x</td>
</tr>
<tr>
<td>LCON</td>
<td>x = *y</td>
<td>Store contents pointed to by y to x</td>
</tr>
<tr>
<td>SCON</td>
<td>*x = y</td>
<td>Store y to location pointed to by x</td>
</tr>
<tr>
<td>GOTO</td>
<td>goto L</td>
<td>Unconditional jump to L</td>
</tr>
<tr>
<td>BLT, BLE, BGT, BGE</td>
<td>if(x rop y)goto L</td>
<td>Test relation and conditionally jump to L</td>
</tr>
<tr>
<td>BIF</td>
<td>if (x) goto L</td>
<td>Conditionally jump to L if x != 0</td>
</tr>
<tr>
<td>BNIF</td>
<td>if (!x) goto L</td>
<td>Conditionally jump to L if x == 0</td>
</tr>
<tr>
<td>PARM</td>
<td></td>
<td>Store x as a parameter (push onto call stack)</td>
</tr>
<tr>
<td>CALL</td>
<td>x = p(...)</td>
<td>Call procedure p with n words of parameters</td>
</tr>
<tr>
<td>RET</td>
<td>return x</td>
<td>Return from function with result x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Declaration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>glob x, n</td>
<td>Declare a global variable named x that refers to offset n in the global region</td>
</tr>
<tr>
<td>proc x, n1, n2</td>
<td>Declare a procedure x with n1 words of parameters and n2 words of locals</td>
</tr>
<tr>
<td>loc x, n</td>
<td>Declare a local variable named x that refers to offset n in the local region</td>
</tr>
<tr>
<td>lab Ln</td>
<td>Declare a label Ln that will be a name for an instruction in the code region</td>
</tr>
<tr>
<td>end</td>
<td>Declare the end of the current procedure</td>
</tr>
<tr>
<td>Production</td>
<td>Semantic Rules</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Assignment : IDENT &quot;=&quot; AddExpr</td>
<td></td>
</tr>
</tbody>
</table>
  Assignment.addr = IDENT.addr  
  Assignment.icode = AddExpr.icode  
  gen(ASN, IDENT.addr, AddExpr.addr) |
| AddExpr : AddExpr1 '+' MulExpr | 
  AddExpr.addr = newtemp()  
  AddExpr.icode = AddExpr1.icode  
  MulExpr.icode  
  gen(ADD, AddExpr.addr, AddExpr1.addr, MulExpr.addr) |
| AddExpr : AddExpr1 '-' MulExpr | 
  AddExpr.addr = newtemp()  
  AddExpr.icode = AddExpr1.icode  
  MulExpr.icode  
  gen(SUB, AddExpr.addr, AddExpr1.addr, MulExpr.addr) |
| MulExpr : MulExpr1 '*' UnaryExpr | 
  MulExpr.addr = newtemp()  
  MulExpr.icode = MulExpr1.icode  
  UnaryExpr.icode  
  gen(MUL, MulExpr.addr, MulExpr1.addr, UnaryExpr.addr) |
| MulExpr : MulExpr1 '/' UnaryExpr | 
  MulExpr.addr = newtemp()  
  MulExpr.icode = MulExpr1.icode  
  UnaryExpr.icode  
  gen(DIV, MulExpr.addr, MulExpr1.addr, UnaryExpr.addr) |
| UnaryExpr : '-' UnaryExpr1 | 
  UnaryExpr.addr = newtemp()  
  UnaryExpr.icode = UnaryExpr1.icode  
  gen(NEG, UnaryExpr.addr, UnaryExpr1.addr) |
| UnaryExpr : '(' AddExpr ')' | 
  UnaryExpr.addr = AddExpr.addr  
  UnaryExpr.icode = AddExpr.icode |
| UnaryExpr : IDENT | 
  UnaryExpr.addr = IDENT.addr  
  UnaryExpr.icode = emptylist() |

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```
<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>AndExpr : AndExpr1 &amp;&amp; EqExpr</td>
<td>EqExpr.first = newlabel(); AndExpr1.onTrue = EqExpr.first; AndExpr1.onFalse = AndExpr1.onFalse; EqExpr.onTrue = AndExpr1.onTrue; EqExpr.onFalse = AndExpr1.onFalse; AndExpr1.icode = AndExpr1.icode</td>
</tr>
<tr>
<td>OrExpr : OrExpr1</td>
<td></td>
</tr>
<tr>
<td>UnaryExpr : ! UnaryExpr1</td>
<td>UnaryExpr1.onTrue = UnaryExpr1.onFalse UnaryExpr1.onFalse = UnaryExpr1.onTrue UnaryExpr1.icode = UnaryExpr1.icode</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>WhileStmt : while '(' Expr ')' Stmt</td>
<td>Expr.onTrue = newlabel(); Expr.first = newlabel(); Expr.false = WhileStmt.follow; Stmt.follow = Expr.first; WhileStmt.icode = gen(LABEL, Expr.first)</td>
</tr>
<tr>
<td>ForStmt : for( ForInit; Expr; ForUpdate ) Stmt a.k.a. ForInit; while (Expr) { Stmt ForUpdate }</td>
<td>Expr.true = newlabel(); Expr.first = newlabel(); Expr.false = S.follow; Stmt.follow = ForUpdate.first; S.icode = ForInit.icode</td>
</tr>
</tbody>
</table>
Chapter 10: Syntax Coloring in an IDE
## Chapter 11: Bytecode Interpreters

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HALT</td>
<td>Halt</td>
</tr>
<tr>
<td>2</td>
<td>NOOP</td>
<td>Do nothing</td>
</tr>
<tr>
<td>3</td>
<td>ADD</td>
<td>Add the top two integers on the stack, push the sum</td>
</tr>
<tr>
<td>4</td>
<td>SUB</td>
<td>Subtract the top two integers on the stack, push the difference</td>
</tr>
<tr>
<td>5</td>
<td>MUL</td>
<td>Multiply the top two integers on the stack, push the product</td>
</tr>
<tr>
<td>6</td>
<td>DIV</td>
<td>Divide the top two integers on the stack, push the quotient</td>
</tr>
<tr>
<td>7</td>
<td>MOD</td>
<td>Divide the top two integers on the stack, push the remainder</td>
</tr>
<tr>
<td>8</td>
<td>NEG</td>
<td>Negate the integer at the top of the stack</td>
</tr>
<tr>
<td>9</td>
<td>PUSH</td>
<td>Push a value from memory to the top of the stack</td>
</tr>
<tr>
<td>10</td>
<td>POP</td>
<td>Pop a value from the top of the stack and place it in memory</td>
</tr>
<tr>
<td>11</td>
<td>CALL</td>
<td>Call a function with n parameters on the stack</td>
</tr>
<tr>
<td>12</td>
<td>RETURN</td>
<td>Return to the caller with a return value of x</td>
</tr>
<tr>
<td>13</td>
<td>GOTO</td>
<td>Set the instruction pointer to location L</td>
</tr>
<tr>
<td>14</td>
<td>BIF</td>
<td>Pop the stack; if it is non-zero, set the instruction pointer to L</td>
</tr>
<tr>
<td>15</td>
<td>LT</td>
<td>Pop two values, compare, push 1 if less than, else 0</td>
</tr>
<tr>
<td>16</td>
<td>LE</td>
<td>Pop two values, compare, push 1 if less or equal, else 0</td>
</tr>
<tr>
<td>17</td>
<td>GT</td>
<td>Pop two values, compare, push 1 if greater than, else 0</td>
</tr>
<tr>
<td>18</td>
<td>GE</td>
<td>Pop two values, compare, push 1 if greater or equal, else 0</td>
</tr>
<tr>
<td>19</td>
<td>EQ</td>
<td>Pop two values, compare, push 1 if equal, else 0</td>
</tr>
<tr>
<td>20</td>
<td>NEQ</td>
<td>Pop two values, compare, push 1 if not equal, else 0</td>
</tr>
<tr>
<td>21</td>
<td>LOCAL</td>
<td>Allocate n words on the stack</td>
</tr>
<tr>
<td>22</td>
<td>LOAD</td>
<td>Indirect push; reads through a pointer</td>
</tr>
<tr>
<td>23</td>
<td>STORE</td>
<td>Indirect pop; writes through a pointer</td>
</tr>
</tbody>
</table>
Chapter 12: Generating Bytecode

No images...
# Chapter 13: Native Code Generation

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addq</td>
<td>Add a 64-bit into another 64-bit value</td>
</tr>
<tr>
<td>call</td>
<td>Store a return address to (%rsp), decrement %rsp, goto function</td>
</tr>
<tr>
<td>cmpq</td>
<td>Compare two values and set condition code bits</td>
</tr>
<tr>
<td>goto</td>
<td>Jump to a new location in the code</td>
</tr>
<tr>
<td>jle</td>
<td>Jump if less than or equal</td>
</tr>
<tr>
<td>leaq</td>
<td>Compute an address</td>
</tr>
<tr>
<td>movq</td>
<td>Move a 64-bit value from source to destination</td>
</tr>
<tr>
<td>negq</td>
<td>Negate a 64-bit value</td>
</tr>
<tr>
<td>popq</td>
<td>Fetch a value from (%rsp) and increment %rsp</td>
</tr>
<tr>
<td>pushq</td>
<td>Store a value to (%rsp) and decrement %rsp</td>
</tr>
<tr>
<td>ret</td>
<td>Fetch a value from (%rsp), increment %rsp and goto the address</td>
</tr>
<tr>
<td>.global</td>
<td>This symbol should be visible from other modules</td>
</tr>
<tr>
<td>.text</td>
<td>Place the bytes to follow in the code region</td>
</tr>
<tr>
<td>.type</td>
<td>This symbol is the following type</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Access Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k</td>
<td>Immediate mode, value given in the instruction</td>
</tr>
<tr>
<td>k(r)</td>
<td>Indirect mode, fetch memory k bytes relative to register r</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Register</th>
<th>Description/Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>rip</td>
<td>Instruction pointer.</td>
</tr>
<tr>
<td>rax</td>
<td>Accumulator. Also: function return value.</td>
</tr>
<tr>
<td>rbx</td>
<td>A secondary accumulator.</td>
</tr>
<tr>
<td>rbp</td>
<td>Frame pointer. Local variables are relative to this pointer.</td>
</tr>
<tr>
<td>rsp</td>
<td>Stack pointer. Memory between rbp and rsp is the local region.</td>
</tr>
<tr>
<td>rdi</td>
<td>Destination index. Holds parameter #1.</td>
</tr>
<tr>
<td>rsi</td>
<td>Source index. Holds parameter #2.</td>
</tr>
<tr>
<td>rdx</td>
<td>A secondary accumulator. Holds parameter #3.</td>
</tr>
<tr>
<td>rcx</td>
<td>Holds parameter #4.</td>
</tr>
<tr>
<td>r8</td>
<td>Holds parameter #5.</td>
</tr>
<tr>
<td>r9</td>
<td>Holds parameter #6.</td>
</tr>
<tr>
<td>r10-r15</td>
<td>Open registers usable for any purpose.</td>
</tr>
<tr>
<td>return value parameter</td>
<td>earlier activation record</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>parameter previous frame pointer (FP) saved registers</td>
<td></td>
</tr>
<tr>
<td>%rbp → saved PC local</td>
<td></td>
</tr>
<tr>
<td>local</td>
<td></td>
</tr>
<tr>
<td>temporaries</td>
<td></td>
</tr>
<tr>
<td>%rsp →</td>
<td></td>
</tr>
</tbody>
</table>

"top" of stack grows down by subtracting from %rsp calls create new activation records here

01111111 01000101 01001100 01000110 00000010 00000001 .ELF. ...
00000001 00000000 00000000 00000000 00000000 00000000 .......
00000000 00000000 00000001 00000000 00000000 00000000 >...
00000000 00000000 00000000 00000000 00000000 00000000 .......
00000000 00000000 00000000 00000000 00000000 00000000 .......
00000000 00000000 00000000 00000000 00000000 0010000 00000000 .@
00000000 00000000 00000000 00000000 00000000 0100000 00000000 .@
00010111 00000000 00010100 00000000 00101011 01010000 01010000 UH
10001001 11100101 11000111 01000101 11111100 00001000 ...E...
00000000 00000000 00000000 10001011 01000101 11111100 ...E
01011101 11000111 00000000 01000111 01000011 01000111 ...
00111010 01010000 01010000 01010101 01101011 00101000 ...
01110110 01110100 01101011 0100000 01101111 01011110 ntu 7.
00110101 00101110 00110000 00101101 00110011 01110101 5.0-3u
Chapter 14: Implementing Operators and Built-In Functions

No images...
Chapter 15: Domain Control Structures

For example, suppose string s contains &pos=1

For example, suppose string s contains &pos=14

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>any(C)</td>
<td>is the character at the position a member of a character set</td>
</tr>
<tr>
<td>many(C)</td>
<td>are 1+ characters at the position members of a character set</td>
</tr>
<tr>
<td>match(s)</td>
<td>do the characters at the position match a search string</td>
</tr>
<tr>
<td>find(s)</td>
<td>produce position(s) at which characters match a search string</td>
</tr>
<tr>
<td>upto(C)</td>
<td>produce position(s) at which the character is a member of a character set</td>
</tr>
<tr>
<td>bal()</td>
<td>produce position(s) where characters are balanced with respect to delimiters</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Production</th>
<th>Semantic Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsection : WSECTION expr_1 DO expr_2</td>
<td>wsection.code = &quot;1(WSection(&quot;</td>
</tr>
</tbody>
</table>


Chapter 16: Garbage Collection

String
len : int = 39
refcount : int = 1
contents : char[] = Omnia Gallia in tres partes divida est

MethodCall≠0

QualifiedName≠0

STRINGLIT
text = "hello"
lineno = 4

IDENTIFIER
text = String
lineno = 4

IDENTIFIER
text = pool
lineno = 4

refcount=1

word size
pointer base
pointer end
pointer free
region *next
region *free

free space

...
Chapter 17: Final Thoughts

No images...
### Appendix: Unicon Essentials

**Code** | **Character** | **Code** | **Character** | **Code** | **Character** | **Code** | **Character**
--- | --- | --- | --- | --- | --- | --- | ---
`\b` | backspace | `\d` | delete | `\e` | escape | `\f` | form feed
`\l` | line feed | `\n` | newline | `\r` | carriage return | `\t` | tab
`\w` | vertical tab | `\'` | quote | `\"` | double quote | `\` | backslash
`\ooo` | octal | `\xhh` | hexadecimal | `\x` | Control-x

### Environment variable
<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLKSIZE</td>
<td>Bytes in the block heap</td>
</tr>
<tr>
<td>IPATH</td>
<td>List of directories to search for linking</td>
</tr>
<tr>
<td>LPATH</td>
<td>List of directories to search for includes</td>
</tr>
<tr>
<td>MSTKSIZE</td>
<td>Bytes on the main stack</td>
</tr>
<tr>
<td>STKSIZE</td>
<td>Bytes on co-expression stacks</td>
</tr>
<tr>
<td>STRSIZE</td>
<td>Bytes in the string heap</td>
</tr>
<tr>
<td>TRACE</td>
<td>Initial value of &amp;trace</td>
</tr>
<tr>
<td>Defined macro</td>
<td>Meaning</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>_CO_EXPRESSIONS</td>
<td>synchronous threads</td>
</tr>
<tr>
<td>CONSOLE_WINDOW</td>
<td>emulated terminal</td>
</tr>
<tr>
<td>DBM</td>
<td>DBM</td>
</tr>
<tr>
<td>DYNAMIC_LOADING</td>
<td>code can be loaded</td>
</tr>
<tr>
<td>EVENT_MONITOR</td>
<td>code is instrumented</td>
</tr>
<tr>
<td>_GRAPHICS</td>
<td>Graphics</td>
</tr>
<tr>
<td>_KEYBOARD_FUNCTIONS</td>
<td>kbhit(), etc.</td>
</tr>
<tr>
<td>LARGE_INTEGER</td>
<td>arbitrary precision</td>
</tr>
<tr>
<td>MACINTOSH</td>
<td>Macintosh</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mode letter(s)</th>
<th>Description</th>
<th>Mode letter(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>add/append</td>
<td>n1</td>
<td>listen on a TCP port</td>
</tr>
<tr>
<td>b</td>
<td>open for both reading and writing</td>
<td>n2</td>
<td>connect to a UDP port</td>
</tr>
<tr>
<td>c</td>
<td>make a new file</td>
<td>n3</td>
<td>connect to messaging server</td>
</tr>
<tr>
<td>d</td>
<td>GDBM database</td>
<td>o</td>
<td>ODBC (SQL) connection</td>
</tr>
<tr>
<td>g</td>
<td>2D graphics window</td>
<td>p</td>
<td>execute a command line and pipe it</td>
</tr>
<tr>
<td>gl</td>
<td>3D graphics window</td>
<td>r</td>
<td>read</td>
</tr>
<tr>
<td>n</td>
<td>TCP client</td>
<td>t</td>
<td>translate newlines</td>
</tr>
<tr>
<td>na</td>
<td>accept TCP connection</td>
<td>u</td>
<td>use a binary untranslated mode</td>
</tr>
<tr>
<td>nau</td>
<td>accept UDP datagrams</td>
<td>w</td>
<td>write</td>
</tr>
</tbody>
</table>