Code Generation

Intermediate Code Generation

Intermediate Code

- An **abstract machine language** that can express the target-machine operations without committing to too much machine-specific details.

Why Intermediate Code?

- **Retargeting**
- Machine independent **code improvement** (optimization)

Intermediate Code

- Intermediate codes can take many forms.
  - Graphical: AST can be a form of IR!
  - Linear
- **Linear Intermediate Representation (IR)**
  - Linearization of a syntax tree
  - A representation of the syntax tree in a linear form

Linear Intermediate Codes

- Three Address Code
- P-code (Stack Machine code)

Three Address Code

- **General form:** **x = y op z**
  - **Many other forms may be needed**
- Source:
  2*a+(b-3)
- Syntax tree:

- Three Address Code:
  t1, t2 & t3 are temporaries

```
t1 = 2 * a
t2 = b – 3
t3 = t1 + t2
```


Example: Three Address Code

- Source: Figure 8.1 (p. 401)
- Syntax tree: Figure 3.9 (p. 138)
- Three Address Code: Figure 8.1 (p. 401)

Representation of Three Address Code

- **Quadruple**
  - Figure 8.3 (p. 403)
  - Temporary variables
- **Triple**
  - Figure 8.5 (p. 404)
  - Less space
  - No temporary variables (using index)

Quadruples

```
(rd,x,_,_)
(gt,x,0,t1)
(if_f,t1,L1,_)
(assn,1,fact,_)
(lab,L2,_,_)
(mul,fact,x,t2)
(asn,t2,fact,_)
(sub,x,1,t3)
(asn,t3,x,_)
(eq,x,0,t4)
(if_f,t4,L2,_)
(wri,fact,_,_)
(lab,L1,_,_)
(halt,_,_,_)
```


Triples

```
(0)  (rd,x,_)
(1)   (gt,x,0)
(2)  (if_f,(1),(11))
(3)  (assn,1,fact)
(4)  (mul,fact,x)
(5)  (asn,(4),fact)
(6)  (sub,x,1)
(7)  (asn,(6),x)
(8)  (eq,x,0)
(9 ) (if_f,(8),(4))
(10) (wri,fact,_)
(11) (lab,L1,_)
```


P-Code (Stack Machine Code)

- **A stack machine code**
  - Most instructions take their operands from the stack, and place results back on the stack.
  - No variables or registers
  - JVM
- Source:
  2*a+(b-3)
- Syntax tree:

- Three Address Code:

```
ldc 2
lod a
mpi
lod b
ldc 3
sbi
adi
```

P-Code (Stack Machine Code)

- Source:
  x = y + 1
- Syntax tree:

- Three Address Code:

```
lda x
lod y
ldc 1
adi
sto
```


P-Code (Stack Machine Code)

ldc x - load constant x
lda x - load address x
lod x - load variable x
sto - store value in address
stn - store & push
mpi - multiply integers
sbi - subtract integers
adi - add integers

P-Code (Stack Machine Code)

rdi - read int
wri - write int
lab - label
fjp - jump if false
ujp - unconditional jump
grt - >
equ - =
stp - stop

P-Code (Stack Machine Code)

ind x – (Indirect load)
add x to top of stack, use result as an address of item to push onto the stack
ixa x – (Indexed address)
calculate address as (top)*x+(top-1), push address onto stack

See P. 417!

Example: P-Code (Stack Machine Code)

- Source: Figure 8.1 (p. 401)
- Syntax tree: Figure 3.9 (p. 138)
- P-Code: Figure 8.6 (p. 406)

Intermediate Code Generation

- Intermediate code generation can be described as an attribute grammar!
  - Synthesized Attribute: A string
- Intermediate code generation is the attribute computation.
- **Syntax-directed translation!**

Intermediate Code Generation as Attribute Grammar

- The attribute grammar -Intermediate code generation - is SAG.

Example: The Language

- A language for expressions
  - Syntax: CFG (p. 407)

Example: AG for P-Code Generation

- **Table #1 (p. 408)**

Example: AG for Three Address Code Generation

- **Table # 2 (p. 409)**

Intermediate Code Generation as Attribute Computation

- The attribute grammar -Intermediate code generation - is SAG.
- The attribute computation – Code - can be generated in the bottom-up fashion.

Intermediate Code Generation: During a Bottom-Up Parsing

- Directly during a bottom-up parsing (without generating a syntax tree)
- Example:
  - yacc
  - **See Figure 8.8 (p. 412)**

Intermediate Code Generation: By A Postorder Syntax Tree Traversal

- By a postorder traversal of the syntax tree
- Example:
  - **The syntax tree - p. 410**
  - **CodeGen - See Figure 8.7 (p. 411)**

A Postorder Syntax Tree Traversal Algorithm

```
procedure GenCode (T: ASTreeNode);
begin
if T is not nil then
Generate code to prepare for code of left child of T;
GenCode (LeftChild of T);
Generate code to prepare for code of right child of T;
GenCode (RightChild of T);
   ...
   ...
   ...
Generate code to implement the action of T;
end;
```


Intermediate Code Generation for Arithmetic Expressions & Assignment

```
(x=x+3)+4
```

```
t1=x+3
x=t1
t2=t1+4
```


Intermediate Code Generation for Arithmetic Expressions & Assignment

```
(x=x+3)+4
```

```
lda x
lod x
ldc 3
adi
stn
ldc 4
adi
```


Example: IC Gen for Arithmetic Expressions & Assignments

- A language for expressions & assignment
  - Syntax: CFG (p. 407)
  - The syntax tree - p. 410
  - CodeGen - See Figure 8.7 (p. 411)

Intermediate Code Generation for Data Structure References

- Array
- Record
- Pointers

Intermediate Code Generation for Array References

```
t2 = a[t1]
```

a + t1*elem_size(a)

```
lda t2
lda a
lod t1
ixa elem_size(a)
ind 0
sto
```


Intermediate Code Generation for Array References

```
a[t2] = t1
```

a + t2*elem_size(a)

```
lda a
lod t2
ixa elem_size(a)
lod t1
sto
```


Intermediate Code Generation for Array References

```
a[i+1]= a[j*2] + 3
```

```
lda a
lod i
ldc 1
adi
ixa elem_size(a)
lda a
lod j
mpi
ixa elem_size(a)
ind 0
ldc 3
adi
sto
```


Intermediate Code Generation for Array References

```
(a[i+1]= 2) + a[j]
```

```
lda a
lod i
ldc 1
adi
ixa elem_size(a)
ldc 2
stn
lda a
lod j
ixa elem_size(a)
ind 0
adi
```


Example: IC Gen for Array References

- Syntax
- Syntax Tree
- IC – P-Code
- genICode
- Read **8.4.4 (p. 422)!**

Intermediate Code Generation for Control Statements

- Selection
- Iteration

Intermediate Code Generation for Selection If-Statements

- If (Exp) Statement1 else Statement2

Intermediate Code Generation for Selection If-Statements

```
if ( E ) S1 else S2
```

```
<code to evaluate E to t1>
if_false t1 goto L1
<code for S1>
goto L2
label L1
<code for S2>
label L2
```


Intermediate Code Generation for Selection If-Statements

```
if ( E ) S1 else S2
```

```
<code to evaluate E to t1>
fjp L1
<code for S1>
ujp L2
label L1
<code for S2>
label L2
```


Intermediate Code Generation for Iteration While-Statements

- While (Exp) Statement

Intermediate Code Generation for Selection While-Statements

```
while ( E ) S
```

```
label L1
<code to evaluate E to t1>
if_false t1 goto L2
<code for S>
goto L1
label L2
```


Intermediate Code Generation for Selection While-Statements

```
while ( E ) S
```

```
label L1
<code to evaluate E>
fjp L2
<code for S>
ujp L1
label L2
```

Example: IC Gen for Control Statements

- Syntax
- Syntax Tree
- IC – P-Code
- genICode
- Read **8.4.4 (p. 433)!**

Intermediate Code Generation for Logical Expressions

- **a and b =**
  - **if (a) b else false**
- **a or b =**
  - **if (a) true else b**

Intermediate Code Generation for Logical Expressions

```
(x!=0)&& (y==x)
```

```
lod x
ldc 0
neq
fjp L1
lod y
lod x
equ
Ujp L2
lab L1
lod FALSE
lab L2
```


Intermediate Code Generation for Procedure/Function Definition

- Entry instruction
- [Code for the procedure/function body]
- Return instruction

Intermediate Code Generation for Procedure/Function Definition

```
int f(int x, int y)
{ return x+y+1;}
```

```
entry f
t1 = x + y
t2 = t1 + 1
return t2
```


Intermediate Code Generation for Procedure/Function Definition

```
int f(int x, int y)
{ return x+y+1;}
```

```
ent f
lod x
lod y
adi
ldc 1
adi
ret
```

Intermediate Code Generation for Procedure/Function Calls

- Argument computation instruction
- Code to compute the argument
- Call instruction

Intermediate Code Generation for Procedure/Function Calls

```
f(2+3,4)
```

```
begin args
t1 = 2 + 3
arg t1
arg 4
call f
```


Intermediate Code Generation for Procedure/Function Calls

```
f(2+3,4)
```

```
mst
ldc 2
ldc 3
adi
dc 4
cup f
```


Example: IC Gen for Procedure/Function Definition & Calls

- Syntax
- Syntax Tree
- IC – P-Code
- genICode
- Read **8.5.2 (p. 439)!**

Machine Independent Code Optimization (Improvement)

- On the intermediate codes

Target Code Generation

Target Code Generation from Intermediate Code

- Instruction (code) selection
- Instruction (code) scheduling
- Register allocation

Machine Dependent Code Optimization (Improvement)

- On the target codes

Stack Machine Code to MIPS Code

- Stack machine codes
  - P-code
  - JVM code
- MIPS architecture
  - RISC (Reduced Instruction Set Computer) architecture
  - SPIM (A MPS simulator)