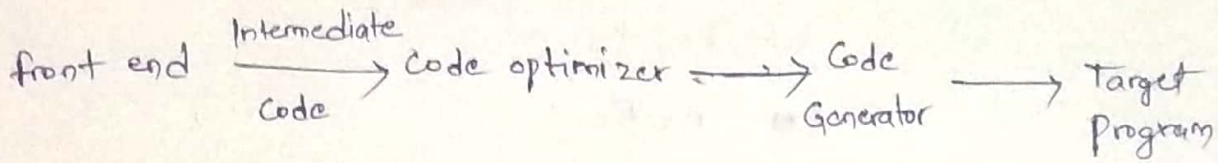


Code Generation

Requirements :-

preserve semantics

Effectively use available resources

itself must be efficient.

primary tasks :-

1. Instruction selection

I/p to code generator

2. Register Allocation & assign

3 address code: quadrapile, triple

virtual machine: bytecodes

3. Evaluation order:

⇒ Target program:- RISC, asc, stackbased Machine

stackbased machine [only ^{push} ~~push~~ & Pop]1. Instruction selection:-

Given a 3 address code, we should map this statements to a sequence of assembly language machine

$$x = y + z$$

$$\rightarrow$$

LD	R0, y
ADD	R0, R0, Z
ST	x, R0

T. Yashwanth

$a = b + c; \quad d = a + e;$

```
LD R0, b
ADD R0, R0, C
ST a, R0
LD R0, a      } same
ADD R0, R0, C
ST d, R0
```

2. Register Allocation

- allocation
- registers assignments

3. Evaluation Order

- fewer registers
- Best NP

Queue Target lang:

```
LD dst, addr (LD, r, x)
ST x, r
OP dst, src1, src2 (operations)
BR L (unconditional jump)
Bcond r, L (conditional)
(L is Label)
```

Addressing mode: $LD R_1, a(R_2) \quad R_1 = \text{content}(\text{content}(R_2) + a)$

$LD R_1, 100(R_2) \quad R_1 = \text{content} + (100 + \text{content}(R_2))$

Array → $LD R_1, *100(R_2) \quad R_1 = \text{cont}(\text{cont}(100 + \text{cont}(R_2)))$

$LD R_1, \#100 \quad (\text{immediate})$

T. Yashwanth

$z = y = z$
 LD R1, y
 LD R2, z
 SUB R1, R1, R2
 ST x, R1

b: a[i] LD R1, i
 MUL R1, R1, 8
 LD R2, a(R1)
 ST b, R2

$a[j] = c$
 LD R1, j
 MUL R1, R1, 8
 LD R8, c
 ST a(R1), R2

$x = *p$
 LD R1, p
 LD R2, 0(R1)
 ST x, R2

$*p = y$
 LD R1, p
 LD R2, y
 ST 0(R1), R2

If $x < y$ goto L

LD R1, x
 LD R2, y
 SUB R1, R2, R2
 BLTZ R1, L

[calculate the cost
 of inst.]

1. $x = a[i]$
 $y = b[i]$
 $z = x + y$

2. $y = *q$
 $q = q + 4$
 $*p = y$
 $p = p + 4$

T. Yoshida

4 byte

```
A1) LD R1, i
     MUL R1, R1, 4
     MUL R2, A(R1), b(R1)
     ST Z, R2
```

A simple code Generator

- generate code for single basic block

How to use register:-

- Either one of op should be in Register
or both in Register

- Register \rightarrow good temp

- Register \rightarrow global values, stored in only
as well

- run time management \leftarrow Reg

uses: Register Descriptor

keeps track of vars whose current value is that
reg

Address Descriptor

location (current value of variable)

Code gen Algo

ex:- $x = y + z$

step 1: get Reg ($x = y + z$)

\rightarrow gives the reg. used for holding the value
for x, y, z

T. Yashwanth

- if y is not in R_y , issue an inst. $LD\ R_y, y$

- issue $ADD\ R_x, R_y, R_z$

copy stmt

$x = y$

if y is not already in reg, $LD\ R_y, y$

Adjust RD for R_y ; so it include r

3. Ending the loopback

Ref: Managing Registers & Address Description

for $LD\ R, x$

- change RD for R so it holds only x

- change AD for x by adding R as add

[follow these steps]

get Reg: $x = y + 2$

- if y is in a reg: do nothing

- if y not in a reg, there is an empty one, choose R_y .

- let v be one of vars in R

→ we're OK if v is somewhere beside R

→ we're OK if v is x

→ we're OK if v is not used later

~~→ we're OK if v is~~

→ spill: $ST\ v, R$

T. Yashwanth

⇒ Peephole Optimization :-

Replaces instr. with shorter/faster sequence

Steps:

1. Eliminating Redundant Load & store

LD a, R0

ST R0, a

2. Eliminating unreachable code

3. Flow of control ops

4. Optimal code Gen for expression.

T. Yashwanth