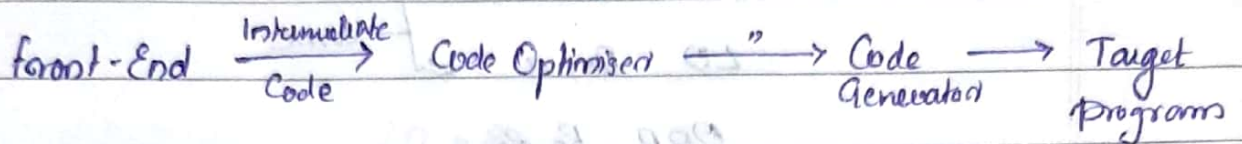


Online Class Lectures⇒ Code Generation:

- Requirements:
- # preserve semantics
 - # Effectively use available resources.
 - # Itself must be efficient

⇒ Primary tasks:1. Instruction Selection

J/P to the Code Generator

2. Register Allocation & assign.
- # 3 address code: quadruple, triple
 - # Virtual machine: bytecodes

3. Evaluation Order⇒ Target Program: RISC, CISC, Stack-based Machine

Stack-based machine [Only push & pop]

1. Instruction Selection:

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

$$x = y + z$$

$$\hookrightarrow$$

[LD R0, y
	ADD R0, R0, z
	ST R0, R0

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

LD R₀, b

ADD R₀, R₀, c

ST a, R₀

LD R₀, a

ADD R₀, R₀, e

ST d, R₀

2) ⇒ Register Allocation

→ allocation
→ register assignments

3) Evaluation Order

- fewer register
- Best NP

Ques: Target lang. : LD dst, addr (LD r, r)

ST r, r

→ should be register

OP dst, src1, src2 (Operations)

BR ←

(Unconditional Jump)

Bcond r, L

(Conditional)

(L is the Label)

Addressing Mode:

LD R₁, a(R₂) $R_1 = \text{Content}(\text{Content}(R_2) + a)$

LD R₁, 100(R₂) $R_1 = \text{Content}(100 + \text{Content}(R_2))$

Array → LD R₁, *100(R₂) $R_1 = \text{Cont}(\text{Cont}(100 + \text{Cont}(R_2)))$

LD R₁, #100 [immediate]

Date ____ / ____ / ____

Eg.

$$x = y - z$$

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 R₂, 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

Calculate the Cost
of Instruction

LD R₁, x

LD R₂, y

SUB R₁, R₂, R₂

BLTZ R₁, L

1)

$$x = a[i]$$

$$y = b[i]$$

$$z = x * y$$

$$2) y = *q$$

$$q = q + 4$$

$$*p = y$$

$$p = p + 4$$

4 byte

Date: / /

Saathi

A1) LD R₁, i
MUL R₁, R₁, 4
ST x, a(R₁)
ST y, b(R₁)
MUL R₂, a(R₁), b(R₁)
ST Z, R₂

A Simple Code Generator

- Generate code for single basic block.

How to use registers:

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

- Register \rightarrow good temp.

- Registers \rightarrow global values, stored in memory as well.

- run-time management \leftarrow Register.

What it Uses: Register descriptor:

keeps track of vars whose current value in that reg.

Address descriptor:

location (Current value of the variable)

Code gen. Algo

Eg: $x = y + z$

step 1: getReg(x = y + z)

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

3-addr

Date ____ / ____ / ____

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

- Issue ADD R_x, R_y, R_z

2. ~~start~~
Copy

$x = y$

if y is not already in reg; LD R_y, y'

Adjust RD for R_y , so it includes y .

3. Ending the loopback.

Ref: Managing Registers & Address Descriptors

for LD R, x

Example: - change RD for R so it holds only x

- change AD for x by adding R as add. option

[follow these steps]

get Reg: $x = y + z$

- 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 the var in R

→ we're OK if v is somewhere besides R

→ We're OK if v is x

→ We are OK if v is not used later

→ Spill: ST v, R

Date ___ / ___ / ___

Saathi

⇒ Peephole Optimization:
replaces inst. 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.