T. YASHWANTH
AM.EN.U4CSE17340
CSE-D

Code Generation

$$\begin{tikzcd}
\text{front end} \arrow[r, "\text{Intermediate}", "\text{Code}"'] & \text{Code optimizer} \arrow[r] & \text{Code Generator} \arrow[r] & \text{Target Program}
\end{tikzcd}$$

Requirements :- # preserve semantics
Effectively use available resources
# itself most be effecient.

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 & 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   x, R0
```

[illegible]

a=b+c ; d=a+e ;

```
LD   R0, b
ADD  R0, R0, c
ST   a, R0   }
LD   R0, a   } same
ADD  R0, R0, e
ST   d, R0
```

2. Register Allocation
   - $\rightarrow$ allocation
   - $\rightarrow$ register assignments

3. Evaluation Order
   - fewer register
   - Best NP

Ques: Target lang :
```
LD dst, addr (LD, r, x)
ST x, r         -> should be register
OP dst, src1, src2   (operations)
BR L   (unconditional jump)
Bcond r, L   (conditional)
             (L is Label)
```

Addressing mods:
```
LD R1 a(R2)   R1 = content(content(R2)+a)
LD R1,100(R2)   R1 = content (100 + content(R2))
Array ----> LD R1, *100 (R2) . R1 = cont (cont (100 + cont (R2)))
LD R1, #100 (Immediate)
```

T. Yashwanth

Scanned with CamScanner

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 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 R, 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. Yashwanth

4 byte

A 1)
```
    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 → good temp
- Register → global values, stored in only
  as well
- run time management ← Reg

uses: Register descriptor

keeps track of vars whose current value in that reg

Address descriptor

location (current value of variable)

Code gen Algo

eg:- $x = y + z$

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

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

T. Yashwanth

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

- Issue ADD Rx, Ry, Rz

copy stmt

x = y

if y is not already in reg, LD Ry, y'

Adjust RD for Ry; so it include x

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 + z

- if y is in a reg: do nothing
- if y not in a reg, three is an empty one, choose Ry.
- Let v be one of var 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~~ [illegible]
  - → spill : ST v, R

[illegible]

⇒ Peephole Optimization :-

Replaces instr. with shorter/faster sequence

<u>steps</u>: 1. Eliminating Redundant Load & store

LD a, R_0
ST R_0, a

2. Eliminating unreachable code
3. Flow of control ops
4 optimal code Gen for expression.

T. [illegible]