Notes

A. Sri deepthi
17205

→ Code Generation

front End —Intermediate code→ Code Optimizer —"→ Code Generator → Target Program

Requirements :- # preserve semantics
Effectively use available resources
itself must be effective

→ Primary tasks:

1. Instruction selection

I/P to the code generator

2. Register Allocation & assign # 3 address code: quadruples, triples
Virtual machine bytecode.

3. Evaluation order:

⇒ Target Program : RISC, CISC, stack based
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

```
→ [ LD  R0, y
    ADD R0, R0, z
    ST  x, R0
```

a=b+c; d=a+c

```
LD  R0, b
ADD R0, R0, c
```

ST a, RO]
LD RO, a] — Same

ADD RO, RO, c
ST d, RO

2) ⇒ Register Allocation

→ allocation
→ register assignment

3) Evaluation Order
- fewer register
- Best NP

Que : Target lang : LD dist, addr (LD, r, x)

ST x, r

OP dst, src1, src2 (operations) — should be register

BR L (unconditional Jump)

Bcond r, L (conditional)
(L is the label)

Addressing Mode : 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]

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

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

1) $x = a[i][j]$
   $y = b[i][j]$
   $z = x * y$

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

A:) LD R1, i

MUL R1, R1, 4

MUL R2, a(R1), b(R1)

ST Z, R2

A Simple Code Generator

- Generator code for single basic block

How to use registers:

- Either one of the op should be in Register or both in Register
- Register → good temp
- Registers → global values, stored in m/y loc. as well
- run-time manage Register.

What it uses:- Register descriptor:

Keeps track of vars whole current value in that reg.

Address descriptor:

Location (current value of the variable)

Code gen. Algo

Eg : $x = y + z$ Step 1: getReg($x = y + z$)

is gives the register used for holding the value for x, y, z.

- if y is not in Ry, issue an inst

2. x = y

Optimal

if y is not already in reg: LD Ry, y'
Adjust RD for Ry, so, 7 indicate

3. Ending the loopback

Ref Managing Register & Address Descriptors
for LD R, x

- change RD for R so it holds only x.
- change AD for x by adding R as add option

Get Reg. x = y + z

- if y is in a reg, do nothing
- if y not is a reg, there is an empty one, choose reg
- Let v be one of the var in R
  - → we're ok if v is somewhere besides R
  - → we've ok if v is x
  - → we are ok if v is not used later
  - → spill: ST V, R

→ peephole optimization

replaces inst, with shorter/ fastes sequence

Steps: 1. Eliminating Redundent load. & Store

```
LD a, R0
ST R0, a
```

2. Eliminating Unreachable code
3. flow-of-control opts
4. optimal code Gen. for Expression.