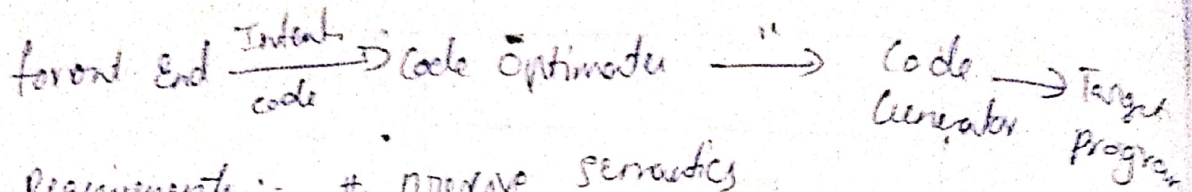


## Notes

'A.Sri' chapter

17205

→ Code Generation



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: quadreg,  
# virtual machine <sup>3ops</sup> hybrid.

3. Evaluation order:

→ Target Program: RISC, CISC, stack based  
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

$x = y + z$

→  $\left[ \begin{array}{l} \text{LD } R_0, y. \\ \text{ADD } R_0, R_0, z \\ \text{ST } R_0, x \end{array} \right.$

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

LD  $R_0, b$

ADD  $R_0, R_0, c$

ST a, R0  
 LD R0, a } — same  
 ADD R0, R0, c  
 ST d, R0

2)  $\Rightarrow$  Register Allocation

$\rightarrow$  allocation  
 $\rightarrow$  register assignments

3) Evaluation Order

- fewer register
- Best NP

Ques: Target lang: LD dst, addr (LD<sub>0</sub> r, n)

ST n, r

OP dst, src1, src2 (operations)

BR k (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  $\rightarrow$  LD R1, 100(R2)  $R_1 = \text{cont}(\text{cont}(100 + \text{cont}(R_2)))$
- LD R1, +100 [immediate]



Eg:

$z = y - z$

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 R2, c

ST a(R1), R2

$x = *p$

LD R1, p

LD R2, 0(R1)

ST x, R2

$*p = y$ : LD R1, y

LD R2, 0

ST 0(R1), R2

if  $x < y$  goto L

LD R1, x

LD R2, y

SUB R1, R2, R2

BLTZ R1, L

1)  $x = a[i]$

$y = b[i]$

$z = x * y$

2)  $y = *x$

$z = z + 7$

$*p = y$

$p = p + 4$

```

A1) LD R1, i
    MUL R1, R1, 4
    MUL R2, a(R1), b(R1)
    ST 2, 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  $\rightarrow$  good temp

— Register  $\rightarrow$  global values, stored in memory as well

— non-time magnetic Register

What it uses: — Register descriptor:

Keeps track of vars whose current value is in that reg.

Address descriptor:

Location (current value of the variable)

## Code gen. Algo

Eg:  $x = y + z$

Step 1: get reg( $x = y + z$ )

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



- if  $y$  is not in  $Ry$ , issue an inst

2. assign  $x = y$

if  $y$  is not already in reg: LD  $Ry, y$   
adjust RD for  $Ry$ , so, find  $ld$

3. Ending the loopback

ref Managing Register & Address Descriptions  
for LD  $R, x$

- change RD for  $R$  so it holds only  $x$
- change AD for  $x$  by adding  $R$  a  
add option

get Reg.  $n - y + 2$

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

→ 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 ,