Date 27 / 04 / 2020

Online Class Lectures

⇒ Code Generation :

$$\begin{tikzcd} \text{Front-End} \arrow[r, "\text{Intermediate}", "\text{Code}"'] & \text{Code Optimiser} \arrow[r, "\text{''}"] & \text{Code Generator} \arrow[r] & \text{Target program} \end{tikzcd}$$

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

→ Primary tasks :

1. Instruction Selection

I/P to the Code Generator

2. Register Allocation & asgn.

3 address code : quadraple, 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.

$x = y + z$
↳
```
LD  R0, y
ADD R0, R0, z
ST  x, R0
```

Page No.

Date ___ /___ /______

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

```
LD  R0, b
ADD R0, R0, c
ST  a, R0     <-|
                |- same
LD  R0, a     <-|
ADD R0, R0, c
ST  d, R0
```

2) ⇒ Register Allocation
- → allocation
- → register assignments

3) Evaluation Order
- fewer register
- Best NP

Que: 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 the label)

Addressing Mode: LD R_1, a(R_2) R_1 = Content(Content(R_2)+a)

LD R_1, 100(R_2) R_1 = Content(100+ Content(R_2))

Array → LD R_1, *100(R_2) R_1 = Cont(Cont (100+ Cont(R_2)))

LD R_1, #100 [immediate]

Page No.

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 R, 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 ___ / ___ / ___

A.) LD R_1, i
MUL R_1, R_1, 4
ST x, a(R_1)
ST y, b(R_1)
~~MUL~~

MUL R_2, a(R_1), b(R_1)
ST z, R_2

A. 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 → good temps
- Registers → global values, stored in mly loc as well.
- run-time mangmt ← Registers.

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

3-addr

Eg: $x = y + z$ step 1: getReg($x = y + z$)
↳ gives the register used for holding the value for x, y, z

Page No.

- if y is not in Ry , issue an inst. LD Ry, y'
- Issue ADD Rx, Ry, Rz

2. 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 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 Ry.
- 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 ___ / ___ / ______

=) Peephole Optimization:

replaces inst. with shorter/faster sequence.

Steps: 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.