27/04/2020 M.Jishnu

Online class lecture:

→ code generation:

$$\begin{tikzcd} \text{front-end} \arrow[r, "\text{Intermidiate}", "\text{code}"'] & \text{code optimiser} \arrow[r, "\text{"}"] & \text{code generator} \end{tikzcd}$$

→ Target Program

Requirements: # Preserve semantic
# effictively use available resources
# itself must be effective

→ Primary tasks:
1. Instruction selection
I/p to the code generator
2. Register Allocation & asign
3 address code: Quadraple, triple
# virtual machine: byte code
3. Evaluation order
⇒ parges Program: RISC, CISC, Stack based Machine
Stack base Machine [only push & pop]

1. Instruction selection:
Given a 3 address code, we should map this statements to a sequence of assemby language machine

$x = y + z$

↳ LD R0, y
ADD R0, R0, z
ST x, R0

M. Jishnu

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

```
LD   R0, b
ADD  R0, R0, c
ST   a, R0  ⌉
LD   R0, a  ⌋ — Jone
ADD  R0, R0, c
ST   d, R0
```

2) Register Allocation

$$\begin{tikzcd} \text{} \arrow[r] \arrow[dr] & \text{allocation} \\ & \text{register assignments} \end{tikzcd}$$

3) Evaluation order
→ fewer register
→ Best NP

puy. Target long: LD dst addr (LD r, x)
ST, x, r ⟶ should be register
Op dst, src1, src2 (operations)
BR (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]

M. Jishnu

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) $y = *a$
$a = a + 4$
$*p = y$
$p = p + 4$
4 byte

1)
```
LD R1, i
MUL R1, R1, 4
MUL R2, a(R1), b(R1)
ST z, R2
```

M. Jithnu

A Simple code generator

- Generate code for single basic block

How to use register :-

- Either one of the op. should be in register or both in register
- Register → good temp
- Register → global value, stored in my locatn as well
- run-time managment ← Register

What it uses: Register descriptor:
Keeps track of vars whose current value in the reg

Address descripter:
location (current value of the variable)

Code generation Algorithm

eg: $x = y+z$ Step1: get Reg ($x = y+z$)
↳ gives the register used for holding the value for x,y,z

- If y is not in Ry, issue an inst LD Ry, y
- issue . ADD Rx, Ry, Rz

2)
Copy statement
$x = y$
if y is not already in reg: LD Ry - y'
Adjust RD for Ry, so it includes x.

M.Jishnu

3. ending the loop back
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. 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

⇒ Peephole optimization:
replace 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