Code Generation

Menghui Wang

April 11, 2012

Code Generation

Convert intermediate codes into assembly language.

First, translate intermediate codes, assuming there are infinitely many registers (temps).

Next, perform register allocation.

Binary Op

Intermediate representation

`t1` $\leftarrow$ `t2 + t3`

Mips assembly language

```
add t1, t2, t3
```

Binary Op With Constant Oprand

Intermediate representation

```
t1 ← t2 + 1
```

Mips assembly language

```
li t3, 1
add t1, t2, t3
```

An Alternate Translation

Intermediate representation

```
t1 ← t2 + 1
```

Mips assembly language

```
addi t1, t2, 1
```

Binary Op With Memory Access

Intermediate representation

```
t1 ← t2 + Mem[t3, 4]
```

Mips assembly language

```
lw t4, 4(t3)
add t1, t2, t4
```

Binary Op With Memory Access

Intermediate representation

```
Mem[t1, 0] ← t2 + t3
```

Mips assembly language

```
add t4, t2, t3
sw t4, 0(t1)
```

Jump

Intermediate representation

```
goto L1
```

Mips assembly language

```
j L1
```

Branch

Intermediate representation

```
if t1 = t2 then goto L1
```

Mips assembly language

```
beq t1, t2, L1
```

Branch

Intermediate representation

```
if Mem[t1,0] < 1 then goto
L1
```

Mips assembly language

```
lw t2, 0(t1)
li t3, 1
blt t2, t3, L1
```

Label Address

Intermediate representation

```
t1 ← LabelAddress(L0)
```

Mips assembly language

```
la t1, L0
```

Mips Call Convention

http://www.cs.washington.edu/education/courses/cse410/09sp/examples/MIPSCallingConventionsSummary.pdf

On function entry

foo(t1, t2, t3)

```
foo:

lw t1, 0($sp) #load parameters
lw t2, 4($sp) #load parameters
lw t3, 8($sp) #load parameters

addiu $sp, $sp, -160 #push stack frame

sw $ra, 28($sp) #save the return address

sw $s0, 16($sp) #save preserved registers
sw $s1, 20($sp)
```

⋮

On function exit

```
_foo_exit:

lw $s1, 20($sp), $s1 #restore saved registers
lw $s0, 16($sp), $s0

lw $ra, 28($sp) #restore the return address

addiu $sp, $sp, 160 #pop stack frame

jr $ra #return
```

Call

Intermediate representation

```
t1 = foo(t2, t3, t4)
```

Mips assembly language

```
sw t2, 0($sp)
sw t3, 4($sp)
sw t4, 8($sp)
jal foo
move t1, $v0
```

Return

Intermediate representation

```
return t1
```

Mips assembly language

```
move $v0, t1
j _foo_exit
```

Register Allocation

We may put all variables (temps) in memory for this phase.
Each temp should be assigned a place in its stack frame, relative to the stack pointer.
Perform load operations before each original instruction, and store after the original instruction.

Pseudo Register Allocation

Assume that `t1`, `t2`, `t3` are stored in `8($sp)`, `12($sp)`, `16($sp)` respectively.

Before PRA

```
add t1, t2, t3
```

After PRA

```
lw $t1, 12($sp)
lw $t2, 16($sp)
add $t0, $t1, $t2
sw $t0, 8($sp)
```