Lecture #28

Code Generation

Code Generation

- In production compilers:
  - Emphasis is on keeping values in registers
    - Especially the current stack frame
  - Intermediate results are laid out in the AR, not pushed and popped from the stack
    - The code generator must assign a location in the AR for each temporary

Code Generation – Handling Temporaries

- Let NT(e) = Number of temporaries needed to evaluate e
- NT($e_1 + e_2$)
  - Needs at least as many temporaries as NT(e_1)
  - Needs at least as many temporaries as NT(e_2) + 1
- Space used for temporaries in e_1 can be reused for temporaries in e_2

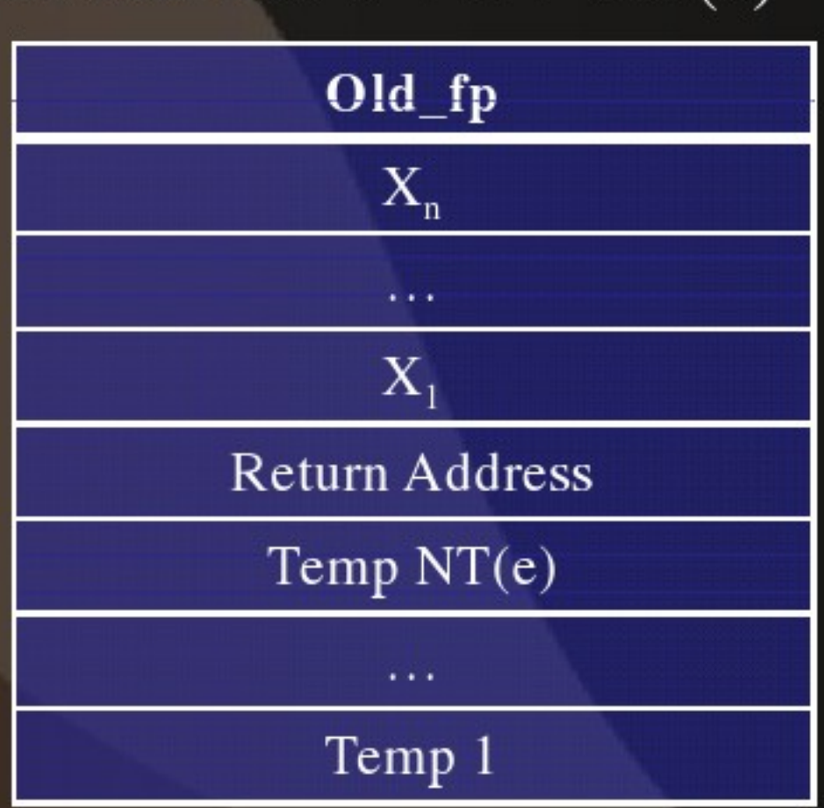
- NT($e_1 + e_2$) = max(NT(e_1), 1 + NT(e_2))
- NT(if $e_1 = e_2$ then e_3 else e_4) = max(NT(e_1),1 + NT(e_2), NT(e_3), NT(e_4))
- NT(id(e_1,…,e_n) = max(NT(e_1),…,NT(e_n))
- NT(int / id) = 0

Code Generation

- def fib(x) = if x = 1 then 0 else
  if x = 2 then 1 else
  fib(x - 1) + fib(x – 2)

2 Temporary variables required

- For a function definition $f(x_1, \ldots, x_n) = e$ the AR has 2 + n + NT(e) elements
- Return address
- Frame pointer
- n arguments
- NT(e) locations for intermediate results



Code Generation

- Code generation must know how many temporaries are in use at each point
- Add a new argument to code generation
  - The position of the next available temporary
- The temporary area is used like a small, fixed-size stack

```
cgen(e1 + e2) =
        cgen(e1)
        sw $a0 0($sp)
        addiu $sp $sp - 4
        cgen(e2)
        lw $t1 4($sp)
        add $a0 $t1 $a0
        addiu $sp $sp 4
```

```
cgen(e1 + e2, nt) =
        cgen(e1, nt)
        sw $a0 nt($fp)
        cgen(e2, nt + 4)
        lw $t1 nt($fp)
        add $a0 $t1 $a0
```

Code Generation Example

```
def sumto(x) = if x = 0 then 0
                        else x + sumto(x - 1)
```