

Compiladores

CS3025

Julio Eduardo Yarasca Moscol

- Unidad 5: Generación de código
 - 5.1. Llamadas a procedimientos y métodos en envío.
 - 5.2. Compilación separada; vinculación. Selección de instrucciones.
 - 5.3. Asignación de registros. Optimización por rendija (peephole)

Logro de la sesión:

Identificar todos los pasos esenciales para convertir automáticamente código fuente en código ensamblador o otros lenguajes de bajo nivel. Generar código de bajo nivel para llamadas a funciones en lenguajes modernos.



Offset

- ▶ Un offset es un desplazamiento en bytes desde una posición base en memoria.
- ▶ Se usa para ubicar datos relativos a un punto fijo, como un registro o una dirección base.
- ▶ Ejemplo

```
void f() { int x = 10; int y = 20 }
```

- ▶ dirección: %rbp+0 <- base del marco
- ▶ dirección: %rbp-8 <- variable x
- ▶ dirección: %rbp-16 <- variable y



Ejemplo

```
fun void main()
var int a, b, c;
a = 1;
b = 2;
c = a + b;
print(c);
return(0)
endfun
```



Ejemplo

```
.data
print_fmt: .string "%ld \n"
.text
.globl main
main:
pushq %rbp
movq %rsp, %rbp
subq $32, %rsp
movq $1, %rax
movq %rax, -8(%rbp)
```

```
movq $2, %rax
movq %rax, -16(%rbp)
movq -8(%rbp), %rax
pushq %rax
movq -16(%rbp), %rax
movq %rax, %rcx
popq %rax
addq %rcx, %rax
movq %rax, -24(%rbp)
movq -24(%rbp), %rax
```

```
movq %rax, %rsi
leaq print_fmt(%rip), %rdi
movl $0, %eax
call printf@PLT
movq $0, %rax
jmp .end_main
.end_main:
leave
ret
.section .note.GNU-stack,"",@progbits
```



Programa

Instrucción	Descripción
.data	Inicia la sección de datos, donde se declaran datos estáticos como strings y variables globales.
print_fmt: .string "%ld \n"	Define una cadena de formato para imprimir enteros
program->vardecs->accept(this);	Genera el código para las declaraciones de variables globales
.text	Inicia la sección de código .
program->fundecs->accept(this);	Genera el código para las funciones definidas en el programa.
.section .note.GNU-stack,,@progbits	Marca que la pila no es ejecutable. Requisito moderno para seguridad en Linux.



Declaración de función

Instrucción	Descripción
.globl f->nombre y f->nombre:	Declara la función como global y define su etiqueta de entrada.
pushq %rbp	Guarda el valor anterior de %rbp.
movq %rsp, %rbp	Establece el nuevo marco de pila: %rbp apunta a la cima del stack actual.
movq argReg[i], offset(%rbp)	Copia el argumento desde el registro a la ubicación correspondiente en la pila.
f->cuerpo->vardecs->accept(this);	Visita las declaraciones de variables locales .
subq \$stackSize, %rsp	Reserva espacio en la pila para las variables locales.
f->cuerpo->slist->accept(this);	Genera el código para el cuerpo de la función.
.end_f->nombre:	Etiqueta final de la función .
leave	Limpia el stack frame
ret	Devuelve el control al llamador.



Llamada y retorno de una función

Instrucción / Línea	Descripción
argRegs = { ... }	Define la lista de registros usados para pasar argumentos según la convención de llamadas
exp->argumentos[i]->accept(this);	Genera el código para evaluar el argumento i.
movq %rax, argRegs[i]	Mueve el resultado del argumento evaluado a su respectivo registro de argumentos.
call exp->nombre	Llama a la función. El control salta a la dirección de la función, retornando a la siguiente instrucción.

Instrucción / Línea	Descripción
stm->e->accept(this);	Si hay una expresión, la evalúa. El valor queda en %rax, que es el registro de retorno por convención.
jmp .end_<nombre>	Salta directamente al final de la función, evitando ejecutar el resto del cuerpo.



Ejemplo

```
fun int suma(int a, int b, int c)
return(a + b + c)
endfun
```

```
fun void main()
print(suma(1,2,3));
return(0)
endfun
```



Ejemplo

```

.data
print_fmt: .string "%ld \n"
.text
.globl suma
summa:
pushq %rbp
movq %rsp, %rbp
movq %rdi, -8(%rbp)
movq %rsi, -16(%rbp)
movq %rdx, -24(%rbp)
subq $32, %rsp
movq -8(%rbp), %rax
pushq %rax
movq -16(%rbp), %rax
movq %rax, %rcx
popq %rax

addq %rcx, %rax
pushq %rax
movq -24(%rbp), %rax
movq %rax, %rcx
popq %rax
addq %rcx, %rax
jmp .end_summa
.end_summa:
leave
ret
.globl main
main:
pushq %rbp
movq %rsp, %rbp
movq $3, %rax
movq %rax, %rdx

movq $2, %rax
movq %rax, %rsi
movq $1, %rax
movq %rax, %rdi
call suma
movq %rax, %rsi
leaq print_fmt(%rip), %rdi
movl $0, %eax
call printf@PLT
movq $0, %rax
jmp .end_main
.end_main:
leave
ret
.section .note.GNU-stack,"",@progbits

```



Ejemplo

```
fun int suma(int a)
var int b;
b = 10;
return(a + b)
endfun
```

```
fun void main()
print(suma(5));
return(0)
endfun
```



Ejemplo

```

.data
print_fmt: .string "%ld \n"
.text
.globl suma
suma:
pushq %rbp
movq %rsp, %rbp
movq %rdi, -8(%rbp)
subq $16, %rsp
movq $10, %rax
movq %rax, -16(%rbp)
movq -8(%rbp), %rax
pushq %rax

        movq -16(%rbp), %rax
        movq %rax, %rcx
        popq %rax
        addq %rcx, %rax
        jmp .end_suma
.end_suma:
        leave
        ret
.globl main
main:
pushq %rbp
movq %rsp, %rbp
movq $5, %rax

        movq %rax, %rdi
        call suma
        movq %rax, %rsi
        leaq print_fmt(%rip), %rdi
        movl $0, %eax
        call printf@PLT
        movq $0, %rax
        jmp .end_main
.end_main:
        leave
        ret
.section .note.GNU-stack,"",@progbits

```



Ejemplo

```
fun int fib(int n)
if (n < 2) then
return(n)
else
return(fib(n - 1) + fib(n - 2))
endif
endfun

fun void main()
var int x;
x=6;
print(fib(x));
return(0)
```



```

.data
print_fmt: .string "%ld \n"
.text
.globl fib
fib:
pushq %rbp
movq %rsp, %rbp
movq %rdi, -8(%rbp)
subq $16, %rsp
movq -8(%rbp), %rax
pushq %rax
movq $2, %rax
movq %rax, %rcx
popq %rax
cmpq %rcx, %rax
movl $0, %eax
setl %al
movzbq %al, %rax
cmpq $0, %rax
je else_0
movq -8(%rbp), %rax
jmp .end_fib
jmp endif_0

else_0:
movq -8(%rbp), %rax
pushq %rax
movq $1, %rax
movq %rax, %rcx
popq %rax
subq %rcx, %rax
movq %rax, %rdi
call fib
pushq %rax
movq -8(%rbp), %rax
pushq %rax
movq $2, %rax
movq %rax, %rcx
popq %rax
subq %rcx, %rax
movq %rax, %rdi
call fib
movq %rax, %rcx
popq %rax
addq %rcx, %rax
jmp .end_fib
endif_0:

.end_fib:
leave
ret
.globl main
main:
pushq %rbp
movq %rsp, %rbp
subq $16, %rsp
movq $6, %rax
movq %rax, -8(%rbp)
movq -8(%rbp), %rax
movq %rax, %rdi
call fib
movq %rax, %rsi
leaq print_fmt(%rip), %rdi
movl $0, %eax
call printf@PLT
movq $0, %rax
jmp .end_main
.end_main:
leave
ret
.section .note.GNU-stack,"",@progbits

```



Ejemplo

```
fun int fac(int n)
if n < 2 then
return(n)
else
return(fac(n - 1) * n)
endif
endfun
```

```
fun void main()
var int x;
x=1;
while x<10 do
print(fac(x));
x = x+1
endwhile;
return(0)
```

►Reinventa el mundo◀



```

print_fmt: .string "%ld \n"
.text
.globl fac
fac:
pushq %rbp
movq %rsp, %rbp
movq %rdi, -8(%rbp)
subq $16, %rsp
movq -8(%rbp), %rax
pushq %rax
movq $2, %rax
movq %rax, %rcx
popq %rax
cmpq %rcx, %rax
movl $0, %eax
setl %al
movzbq %al, %rax
cmpq $0, %rax
je else_0
movq -8(%rbp), %rax
jmp .end_fac
jmp endif_0
else_0:
movq -8(%rbp), %rax
pushq %rax
movq $1, %rax

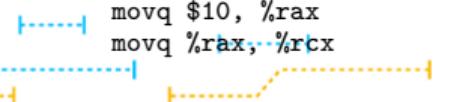
```



```

.popq %rax, %rcx
.popq %rax
.subq %rcx, %rax
.movq %rax, %rdi
.call fac
.pushq %rax
.movq -8(%rbp), %rax
.movq %rax, %rcx
.popq %rax
.imulq %rcx, %rax
.jmp .end_fac
.endif_0:
.end_fac:
.leave
.ret
.globl main
main:
.pushq %rbp
.movq %rsp, %rbp
.subq $16, %rsp
.movq $1, %rax
.movq %rax, -8(%rbp)
while_1:
.movq -8(%rbp), %rax
.pushq %rax
.movq $10, %rax
.movq %rax, %rcx

```



> Reinventa el mundo <

```

.popq %rax
.cmpq %rcx, %rax
.movl $0, %eax
.setl %al
.movzbq %al, %rax
.cmpq $0, %rax
.je endwhile_1
.movq -8(%rbp), %rax
.movq %rax, %rdi
.call fac
.movq %rax, %rsi
.leaq print_fmt(%rip), %rdi
.movl $0, %eax
.call printf@PLT
.movq -8(%rbp), %rax
.pushq %rax
.movq $1, %rax
.movq %rax, %rcx
.popq %rax
.addq %rcx, %rax
.movq %rax, -8(%rbp)
jmp while_1
endwhile_1:
.movq $0, %rax
.jmp .end_main
.end_main:
.leave
.ret
.section .note.GNU-stack,"",@progbits

```

Preguntas y cierre de la sesión

- ▶ ¿En qué consiste el llamado de una función ?
- ▶ ¿En qué consiste el retorno de una función ?

GRACIAS

JULIO EDUARDO YARASCA MOSCOL