Compiladores
Laboratorio 16

Objetivo

Trabajar con herramientas de lujo de control en el lenguaje Assembly para la arquitectura x86-64.

Operadores de Relación

cmpq %rcx, %rax

Instrucción	Operación lógica	Evaluación	Flags evaluados
sete %al	rax == rcx	Igualdad	ZF == 1
setne %al	rax != rcx	Desigualdad	ZF == 0
setl %al	rax < rcx (con signo)	Menor	SF ≠ OF
setle %al	rax <= rcx (con signo)	Menor o igual	ZF == 1 o SF ≠ OF
setg %al	rax > rcx (con signo)	Mayor	ZF == 0 y SF == OF
setge %al	rax >= rcx (con signo)	Mayor o igual	SF == OF
setb %al	rax < rcx (sin signo)	Menor (unsigned)	CF == 1
setbe %al	rax <= rcx (sin signo)	Menor o igual (unsigned)	CF == 1 o ZF == 1
seta %al	rax > rcx (sin signo)	Mayor (unsigned)	CF == 0 y ZF == 0
setae %al	rax >= rcx (sin signo)	Mayor o igual (unsigned)	CF == 0

Flag	Significado	Activación
ZF	Zero Flag	Si rax == rcx
SF	Sign Flag	Si el resultado es negativo
OF	Overflow Flag	Si hay overflow aritmético con signo
CF	Carry Flag	Si hay préstamo (sin signo)

Implementación a<b

Instrucción	Descripción
Genera código para el operando izquierdo **a**, resultado en %rax	
pushq %rax	Guarda temporalmente a en la pila
Genera código para el operando derecho **b**, resultado en %rax	
movq %rax, %rcx	Mueve **b** a %rcx para preparar comparación
popq %rax	Restaura **a** desde la pila a %rax
cmpq %rcx, %rax	Compara **a - b**: actualiza los flags según el resultado
movl $0, %eax	Limpia %eax
setl %al	Si **a < b**, coloca 1 en %al; si no, deja 0
movzbq %al, %rax	Extiende el byte %al a 64 bits en **%rax**

Implementación a/b

Genera código para el dividendo **a**, deja el valor en %rax	
pushq %rax	Guarda el dividendo temporalmente en la pila
Genera código para el divisor **b**, deja el valor en %rax	
movq %rax, %rcx	Mueve el divisor a %rcx
popq %rax	Restaura el dividendo desde la pila a %rax
cqto	Extiende el signo de %rax a %rdx (prepara %rdx:%rax para división)
idivq %rcx	Divide %rdx:%rax entre %rcx Resultado: cociente en %rax, resto en %rdx

Implementación and

Instrucción	Descripción
Genera código para **a**, deja el resultado booleano (0 o 1) en %rax	
pushq %rax	Guarda temporalmente **a** en la pila
Genera código para **b**, deja el resultado booleano (0 o 1) en %rax	
movq %rax, %rcx	Mueve **b** a %rcx para evitar sobrescribirlo
popq %rax	Restaura **a** desde la pila a %rax
andq %rcx, %rax	Realiza **a & b**, resultado en %rax (1 si ambos son 1, si no 0)

Implementación IF:

Instrucción	Descripción
Genera código para la condición	
cmpq $0, %rax	Compara el resultado con 0
je else	Salta al else si la condición es falsa
Genera código para el bloque then	
jmp endif_	Salta al final
else	Etiqueta para el bloque else
Genera código para el bloque else	
endif	Etiqueta de fin de la estructura if

OBS:

Los registros no se pueden borrar, solo se pueden sobrescribir con un nuevo valor o simplemente ignorar su contenido anterior.

Instrucción	Efecto
xor %rax, %rax	Asigna 0 a %rax. Muy eficiente, sin uso de memoria inmediata.
movq $0, %rax	Asigna 0 a %rax usando una constante inmediata.
sub %rax, %rax	Resta %rax consigo mismo, resultado 0.

Laboratorio

Implementar

- Program ::= Body
- Body ::= VarDecList StmtList
- VarDecList ::= (VarDec)*
- VarDec ::= “var” Type VarList **;**
- Type ::= id
- VarList :: id (“,” id)*
- StmtList ::= Stmt **(;** Stmt **)***
- Stmt ::= **id =** CExp |
  print (CExp **)**
  if CExp **then** Body [**else** Body]
- CExp ::= Exp [(**<**|**<=**|**==**) Exp]
- Exp ::= Term ((**+** | **-**) Term)*
- Term ::= Factor ((*****|**/**) Factor)*
- Factor ::= **id** | **Num | (** Exp **)**

Ejemplo

```
var bool x;
x = 4<5;
print(x)
```

```
.data
print_fmt: .string "%ld\n"
.text
.globl main
main:
pushq %rbp
movq %rsp, %rbp
subq $8, %rsp
movq $4, %rax
pushq %rax
movq $5, %rax
movq %rax, %rcx
popq %rax
cmpq %rcx, %rax
movl $0, %eax
setl %al
movzbq %al, %rax
movq %rax, -8(%rbp)
movq -8(%rbp), %rax
movq %rax, %rsi
leaq print_fmt(%rip), %rdi
movl $0, %eax
call printf@PLT
movl $0, %eax
leave
ret
.section .note.GNU-stack,"",@progbits
```

```
var int x,y;
x = 10;
y = 20;
if x<y then
print(100)
else
print(200)
endif
```

```
.data
print_fmt: .string "%ld\n"
.text
.globl main
main:
pushq %rbp
movq %rsp, %rbp
subq $16, %rsp
movq $10, %rax
movq %rax, -8(%rbp)
movq $20, %rax
movq %rax, -16(%rbp)
movq -8(%rbp), %rax
pushq %rax
movq -16(%rbp), %rax
movq %rax, %rcx
popq %rax
cmpq %rcx, %rax
movl $0, %eax
setl %al
cmpq $0, %rax
je else_0
movq $100, %rax
movq %rax, %rsi
leaq print_fmt(%rip), %rdi
movl $0, %eax
call printf@PLT
jmp endif_0
else_0:
movq $200, %rax
movq %rax, %rsi
leaq print_fmt(%rip), %rdi
movl $0, %eax
call printf@PLT
endif_0:
movl $0, %eax
leave
ret
.section .note.GNU-stack,"",@progbits
```