Compiladores
Laboratorio 15

Objetivo

Familiarizarse con las herramientas fundamentales del lenguaje Assembly para la arquitectura x86-64.

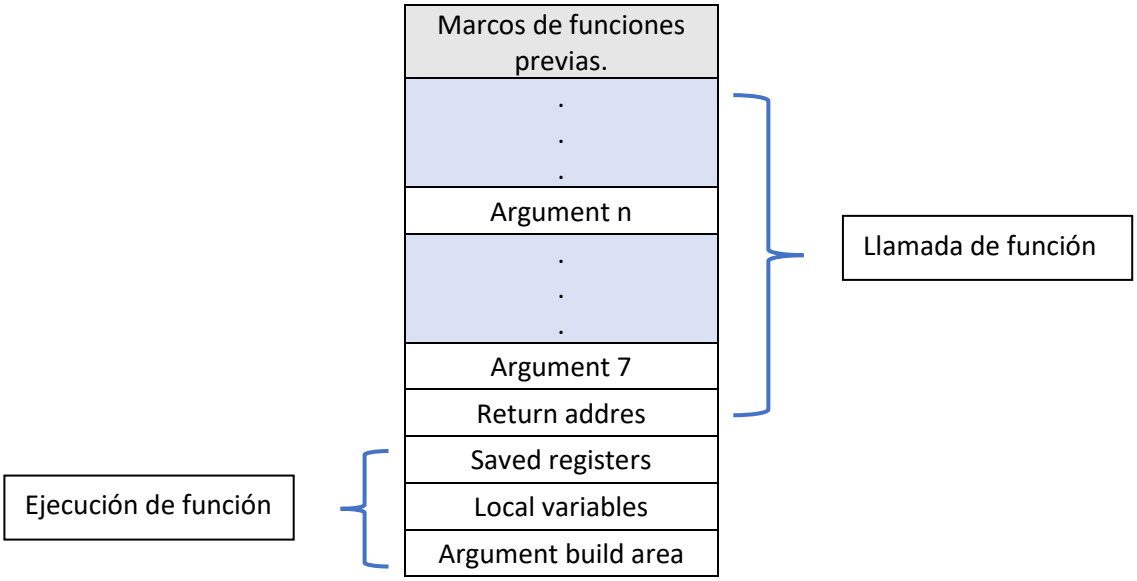
Registros

La arquitectura x86-64 cuenta con 16 registros de propósito general, cada uno de 64 bits de tamaño, que se utilizan para almacenar datos, direcciones y controlar el flujo de ejecución.

Registro (64-bit)	32-bit	16-bit	8-bit	Nombre	Propósito Principal
%rax	%eax	%ax	%al	Acumulador	Valor de retorno. Operaciones aritméticas.
%rbx	%ebx	%bx	%bl	Base	Preservado entre llamadas; se usa para datos o punteros. Callee-saved
%rcx	%ecx	%cx	%cl	Contador	4° argumento de función
%rdx	%edx	%dx	%dl	Datos	3° argumento de función
%rsi	%esi	%si	%sil	Source	2° argumento de función
%rdi	%edi	%di	%dil	Destination	1° argumento de función
%rbp	%ebp	%bp	%bpl	Base Pointer	Apunta al inicio del marco de pila
%rsp	%esp	%sp	%spl	Stack Pointer	Apunta al tope de la pila
%r8	%r8d	%r8w	%r8b	Registros extendidos	5° argumento de función Uso general
%r9	%r9d	%r9w	%r9b		6° argumento de función Uso general
%r10	%r10d	%r10w	%r10b		Registro temporal Caller-saved
%r11	%r11d	%r11w	%r11b		Registro temporal Caller-saved
%r12	%r12d	%r12w	%r12b		General persistente Callee-saved
%r13	%r13d	%r13w	%r13b		General persistente Callee-saved
%r14	%r14d	%r14w	%r14b		General persistente Callee-saved
%r15	%r15d	%r15w	%r15b		General persistente Callee-saved

Pila

La pila en x86-64 organiza la información necesaria para la ejecución de funciones, guardando la dirección de retorno, registros, variables locales y argumentos adicionales. Crece hacia direcciones bajas y reserva espacio para mantener la alineación y facilitar el paso de parámetros según la convención de llamadas.



Prologue y epilogue

El prologue es el conjunto de instrucciones al inicio de una función que prepara el marco de pila, guardando el puntero base anterior y reservando espacio para variables locales; mientras que el epilogue es el bloque al final que restaura el puntero base y limpia la pila para devolver el control a la función llamante, asegurando así un manejo ordenado y seguro del contexto de ejecución.

```
my_function:
    push rbp          Guarda el antiguo puntero de marco en la pila
    mov rsp, rbp      Establecer el nuevo puntero de marco
    sub rsp, 16       Reservar para variables locales

    Cuerpo de la función aquí

    leave             Equivale a: mov rsp, rbp seguido de pop rbp
    ret               Regresar al llamador
```

Ejemplo

```
.globl  suma
.type   suma, @function
suma:
push    %rbp
mov     %rsp, %rbp
mov     %edi, %eax
add     %esi, %eax
leave
ret

.globl  main
.type   main, @function
main:
push    %rbp
mov     %rsp, %rbp
sub     $16, %rsp

movl    $5, -4(%rbp)
movl    $10, -8(%rbp)

movl    -4(%rbp), %edi
movl    -8(%rbp), %esi
call    suma

movl    %eax, -12(%rbp)

movl    -12(%rbp), %eax
mov     %eax, %esi
lea     .LC0(%rip), %rdi
mov     $0, %eax
call    printf@PLT

mov     $0, %eax
leave
ret
```

Métodos

Categoría	Instrucción	Descripción / Comentario
Movimiento de Datos	mov src, dst	Copia el valor de src a dst
	movsbl src, dst	Mueve byte a entero con extensión de signo
	movzbl src, dst	Mueve byte a entero con extensión cero
	cmovcc src, reg	Mueve condicional: reg = src si se cumple condición
	lea addr, dst	Carga la dirección efectiva calculada en dst
Aritmética	add src, dst	dst += src
	sub src, dst	dst -= src
	imul src, dst	dst *= src
	neg dst	dst = -dst (inverso aritmético)
	imulq S	Multiplicación con signo completa
	mulq S	Multiplicación sin signo completa
	idivq S	División con signo
	divq S	División sin signo
Desplazamientos	sal count, dst	Desplazamiento lógico a la izquierda
	sar count, dst	Desplazamiento aritmético a la derecha
	shr count, dst	Desplazamiento lógico a la derecha
Lógica	and src, dst	dst &= src
	or src, dst	`dst
	xor src, dst	dst ^= src
	not dst	dst = ~dst (inverso bit a bit)
Comparación	cmp a, b	Compara b - a y actualiza flags
	test a, b	a & b y actualiza flags
	setcc dst	Establece byte en dst a 1 si condición se cumple, 0 si no
Saltos	jmp label	Salto incondicional
	je label	Salto si igual
	jne label	Salto si no igual
	js label	Salto si negativo
	jns label	Salto si no negativo
	jg label	Salto mayor (signe)
	jge label	Salto mayor o igual (signed)
	jl label	Salto menor (signed
	jle label	Salto menor o igual (signed)
	ja label	Salto mayor (unsigned)
	jae label	Salto mayor o igual (unsigned)
	jb label	Salto menor (unsigned)
	jbe label	Salto menor o igual (unsigned)
Pila	push src	Decrementa %rsp y almacena src en la cima de la pila
	pop dst	Lee valor de la cima de la pila en dst y luego incrementa %rsp
	call fn	Llama función: almacena %rip y salta a fn
	ret	Retorna de función: recupera %rip de la pila
	Leave	Limpia el marco de pila de la función: copia %rbp a %rsp (libera variables locales) y luego hace pop %rbp (restaura el valor anterior de rbp)

Laboratorio

Implementar un visitor Gencode para la gramática:

- Program ::= StmtList
- StmtList := Stmt ('**;**' Stmt)*
- Stmt ::= **id** '**=**' Exp | '**print**' '**(**' Exp '**)**'
- Exp ::= Term ((´**+**´ | ´**-**´) Term)*
- Term ::= Factor ((´*****´|´**/**´) Factor)*
- Factor ::= **id** | **Num** | ´**(**´ Exp ´**)**´

Ejemplos

```
print(4);
print(5)
```

```
.data
print_fmt: .string "%ld\n"
.text
.globl main
main:
  pushq %rbp
  movq %rsp, %rbp
  movq $4, %rax
  movq %rax, %rsi
  leaq print_fmt(%rip), %rdi
  movl $0, %eax
  call printf@PLT
  movq $5, %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
```

print(4+5)

```
.data
print_fmt: .string "%ld\n"
.text
.globl main
main:
  pushq %rbp
  movq %rsp, %rbp
  movq $4, %rax
  pushq %rax
  movq $5, %rax
  movq %rax, %rcx
  popq %rax
  addq %rcx, %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
```

```
x = 5;
y = 10;
z = 20;
print(x+y);
print(x-z)
```

```
.data
print_fmt: .string "%ld\n"
.text
.globl main
main:
  pushq %rbp
  movq %rsp, %rbp
  movq $5, %rax
  movq %rax, -8(%rbp)
  movq $10, %rax
  movq %rax, -16(%rbp)
  movq $20, %rax
  movq %rax, -24(%rbp)
  movq -8(%rbp), %rax
  pushq %rax
  movq -16(%rbp), %rax
  movq %rax, %rcx
  popq %rax
  addq %rcx, %rax
  movq %rax, %rsi
  leaq print_fmt(%rip), %rdi
  movl $0, %eax
  call printf@PLT
  movq -8(%rbp), %rax
  pushq %rax
  movq -24(%rbp), %rax
  movq %rax, %rcx
  popq %rax
  subq %rcx, %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
```

```
x = 120;
y = 50;
z = x/y;
w = x-y+6;
print(z);
print(w)
```

```
.data
print_fmt: .string "%ld\n"
.text
.globl main
main:
  pushq %rbp
  movq %rsp, %rbp
  movq $120, %rax
  movq %rax, -8(%rbp)
  movq $50, %rax
  movq %rax, -16(%rbp)
  movq -8(%rbp), %rax
  pushq %rax
  movq -16(%rbp), %rax
  movq %rax, %rcx
  popq %rax
  cqto
  idivq %rcx
  movq %rax, -24(%rbp)
  movq -8(%rbp), %rax
  pushq %rax
  movq -16(%rbp), %rax
  movq %rax, %rcx
  popq %rax
  subq %rcx, %rax
  pushq %rax
  movq $6, %rax
  movq %rax, %rcx
  popq %rax
  addq %rcx, %rax
  movq %rax, -32(%rbp)
  movq -24(%rbp), %rax
  movq %rax, %rsi
  leaq print_fmt(%rip), %rdi
  movl $0, %eax
  call printf@PLT
  movq -32(%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
```