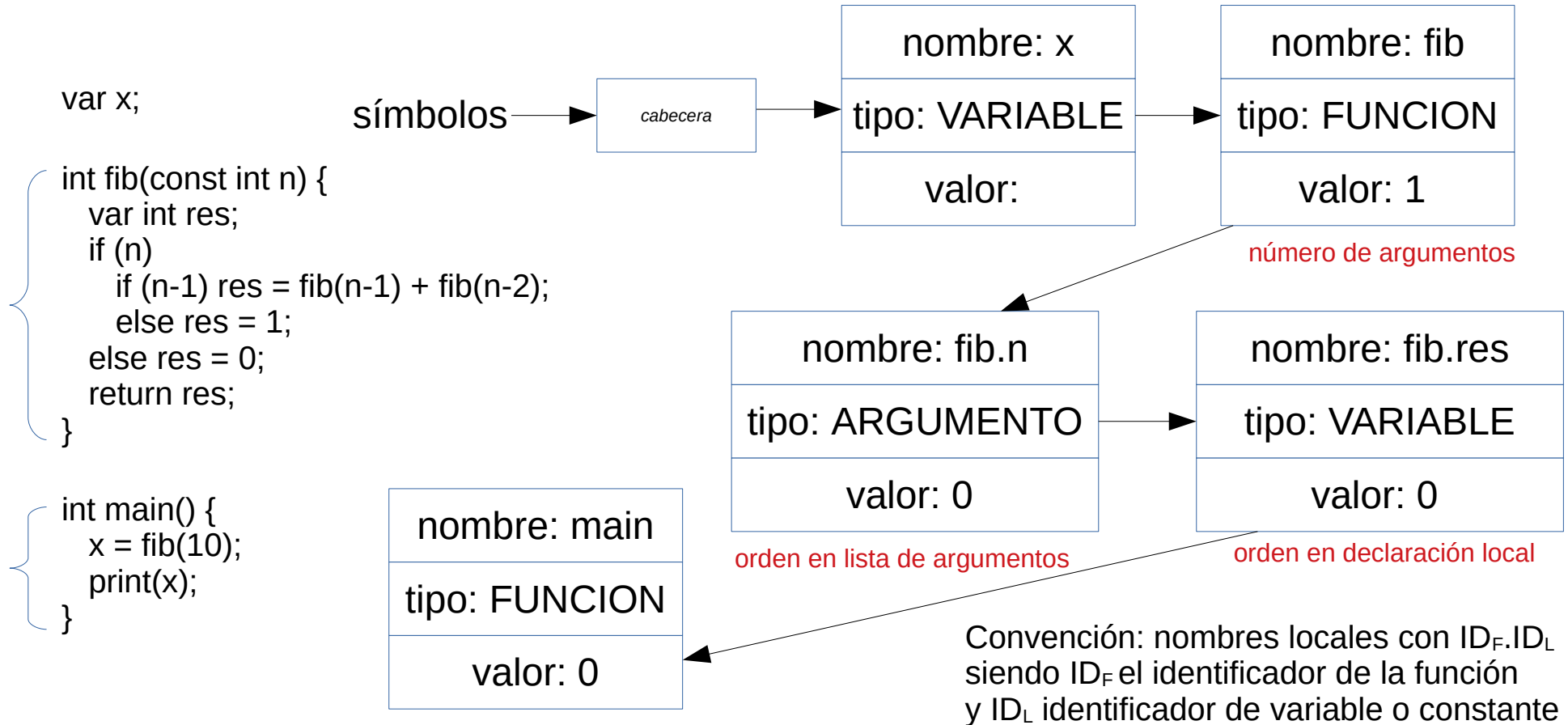


Generación de código

Opción: declaración de funciones
y llamadas a funciones

Ejemplo y tabla de símbolos



Cambios en la gramática

program -> declarations functions

functions -> functions function
| λ

function -> tipo ID "(" arguments ")" "{" declarations statement_list "}"

arguments -> CONS tipo arg_list
| %empty

arg_list -> ID
| arg_list "," ID

statement -> ID "(" call_arguments ")"

expression -> ID "(" call_arguments ")"

call_arguments -> expressions
| %empty

expressions -> expression
| expressions "," expression

Una variable global puede guardar el ID_F de la función que se está traduciendo.

En el análisis semántico se puede buscar primero:

$ID_F.ID_L$

Si no se encuentra, se busca ID_L

Estructura de la traducción

```
var x;

int fib(const int n) {
    var int res;
    if (n)
        if (n-1) res = fib(n-1) + fib(n-2);
        else res = 1;
    else res = 0;
    return res;
}

int main() {
    x = fib(10);
    print(x);
}
```

```
#####
.data

$str0: .asciiz "\n"
_x: .word 0

#####
.text
.globl main

_fib:
    # Salvar registros en pila
    ...
    # Recuperar registros de pila
    jr $ra

main:
    ...
```

Generación de código funciones (1)

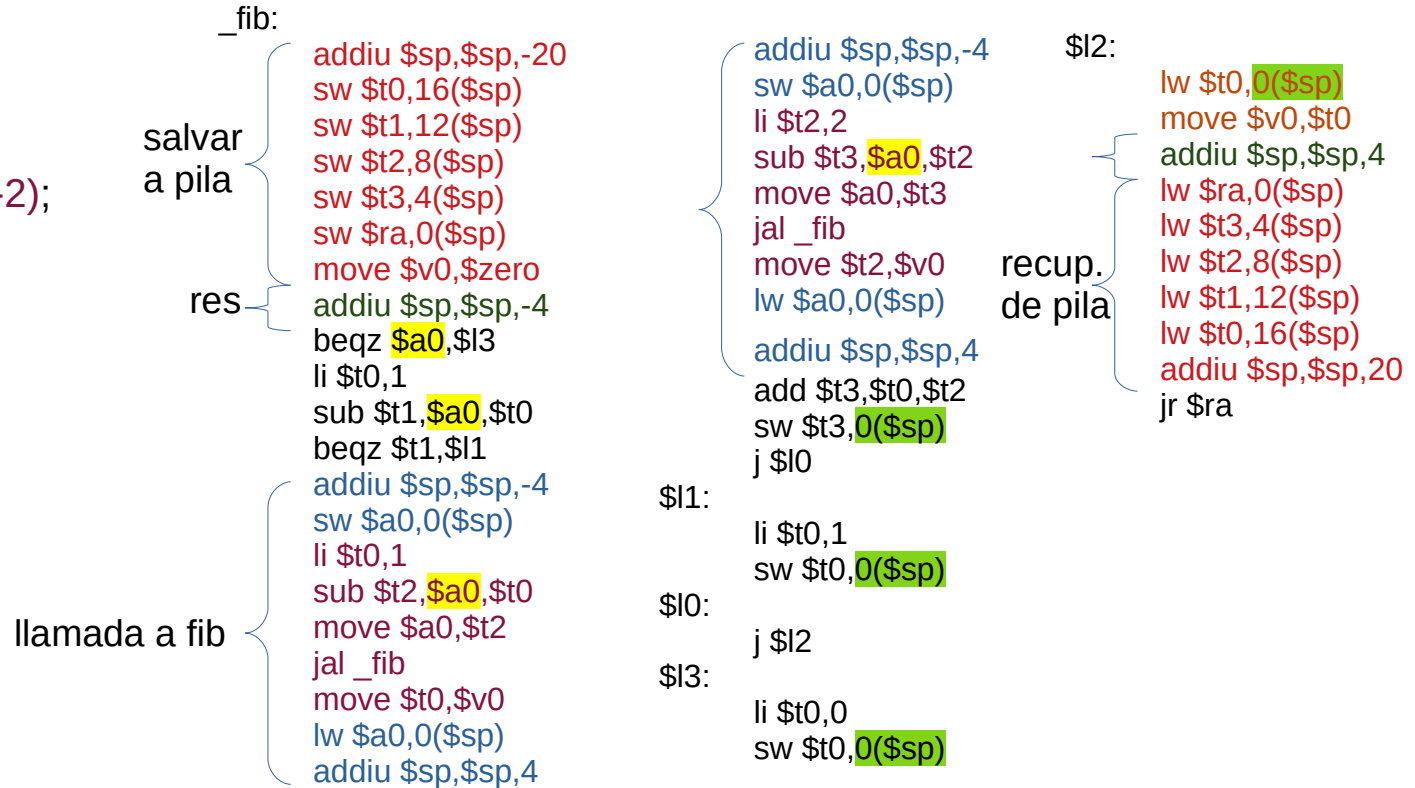
- ¿Cómo se pueden referenciar los argumentos?
 - En registros **\$aX**, $0 \leq X \leq 3$, siendo X el valor del argumento.
- Implicación importante:
 - El uso de identificadores que son argumentos se convierte en **\$aX** en lugar de la etiqueta del identificador.
- ¿Cómo se devuelve un valor desde la función?
 - En el registro **\$v0**

Generación de código funciones (2)

- ¿Cómo se referencian las variables y constantes locales?
 - Hay que usar la pila para permitir llamadas recursivas.
- Implicación importante:
 - El uso de identificadores que son variables y constantes **locales** se convierte en $X(\$sp)$ donde:
$$X = 4 * \text{simbolo.valor}$$
- Conviene usar una función `char * get_ref_id(char *id)` que devuelva la cadena para hacer referencia a un identificador.

Traducción de una función

```
int fib(const int n) {
    var int res;
    if (n)
        if (n-1) res = fib(n-1) + fib(n-2);
        else res = 1;
    else res = 0;
    return res;
}
```



print y read (1)

Peligro con print y read

```
int y(const int a) {  
    print(2);  
    ...  
}  
  
int main() {  
    print(y(1));  
}
```


```
_y:  
    addiu $sp,$sp,-8  
    sw $t0,4($sp)  
    sw $ra,0($sp)  
    move $v0,$zero  
    li $t0,2  
    li $v0,1  
    move $a0,$t0  
    syscall  
    ...
```

Antes del syscall
es necesario salvar
el contenido de los
registros \$a0 y \$v0
si estamos en una función

print y read (2)

```
int y(const int a) {  
    print(2);  
}
```

```
int main() {  
    print(y(1));  
}
```



```
_y:  
    addiu $sp,$sp,-8  
    sw $t0,4($sp)  
    sw $ra,0($sp)  
    move $v0,$zero  
    li $t0,2  
    move $s0,$a0  
    move $s1,$v0  
    li $v0,1  
    move $a0,$t0  
    syscall  
    move $a0,$s0  
    move $v0,$s1  
    lw $ra,0($sp)  
    lw $t0,4($sp)  
    addiu $sp,$sp,8  
    jr $ra
```

Por simplificar, se
pueden salvar en
los registros
\$s0 y \$s1