Lista de código

- Disponible en el Aula Virtual → Prácticas → Lista para código
- Lista simplemente enlazada con un nodo por cada operación MIPS:

```
typedef struct {
        char *op;
        char *res;
        char *arg1;
        char *arg2;
} Operacion;
```

```
syscall

b $label

li $t0, 1

add $t0, $t1, $t2
```

Integración en miniC.y

1. Añadir el fichero de cabecera

```
%{ #include “listaCodigo.h” %}

%code requires {
      #include “listaCodigo.h”
}
```

2. Añadir el tipo ListaC a la unión de tipos

```
%union {
      char *str;
      ListaC codigo;
}
```

3. Asociar el tipo ListaC a expression

```
%type <codigo> expression
```

Modificar makefile

- Añadir listaCodigo.c a la instrucción de compilación:

```
miniC : lex.yy.c miniC.tab.c main.c listaSimbolos.c listaCodigo.c
	gcc -g lex.yy.c miniC.tab.c main.c listaSimbolos.c listaCodigo.c -lfl -o miniC
```

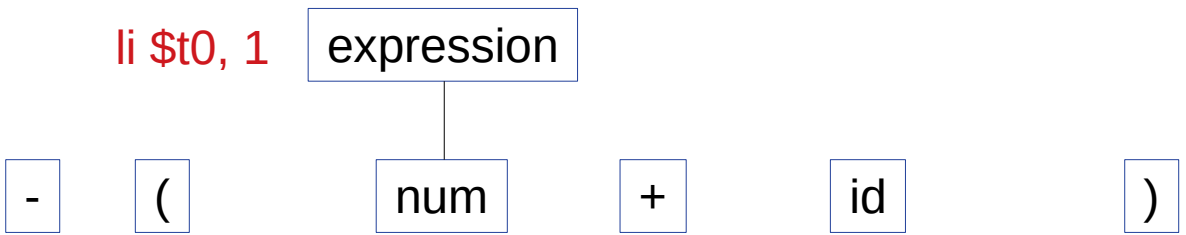
Objetivo: generar código de expresiones

```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```

-	(	num	+	id	)

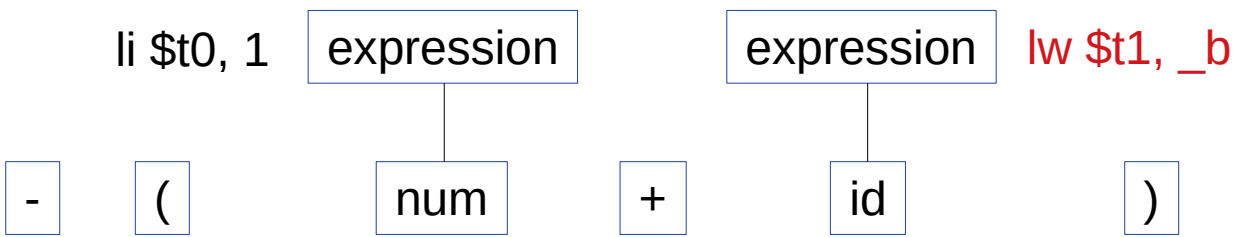
Objetivo: generar código de expresiones

```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```



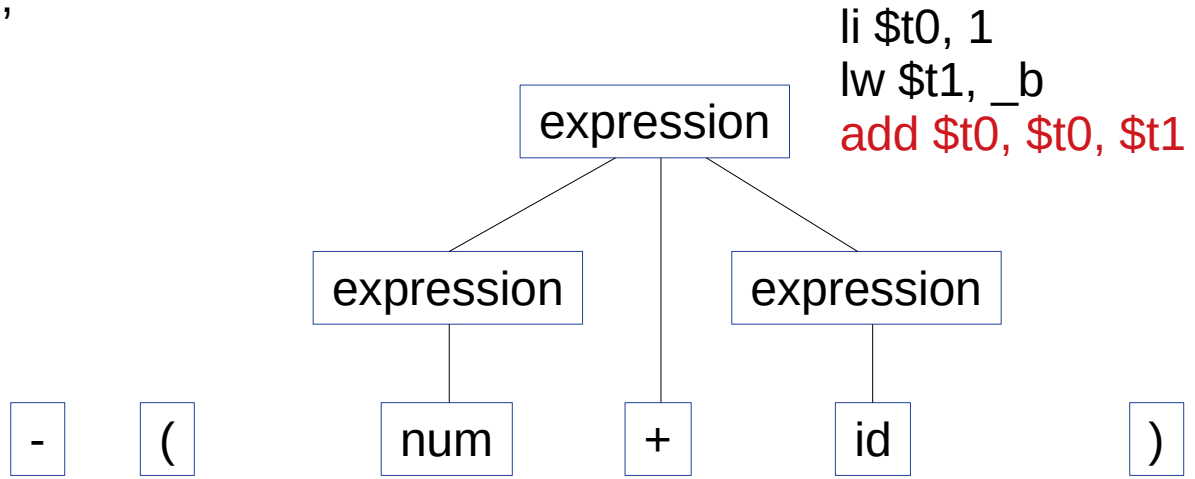
Objetivo: generar código de expresiones

```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```



Objetivo: generar código de expresiones

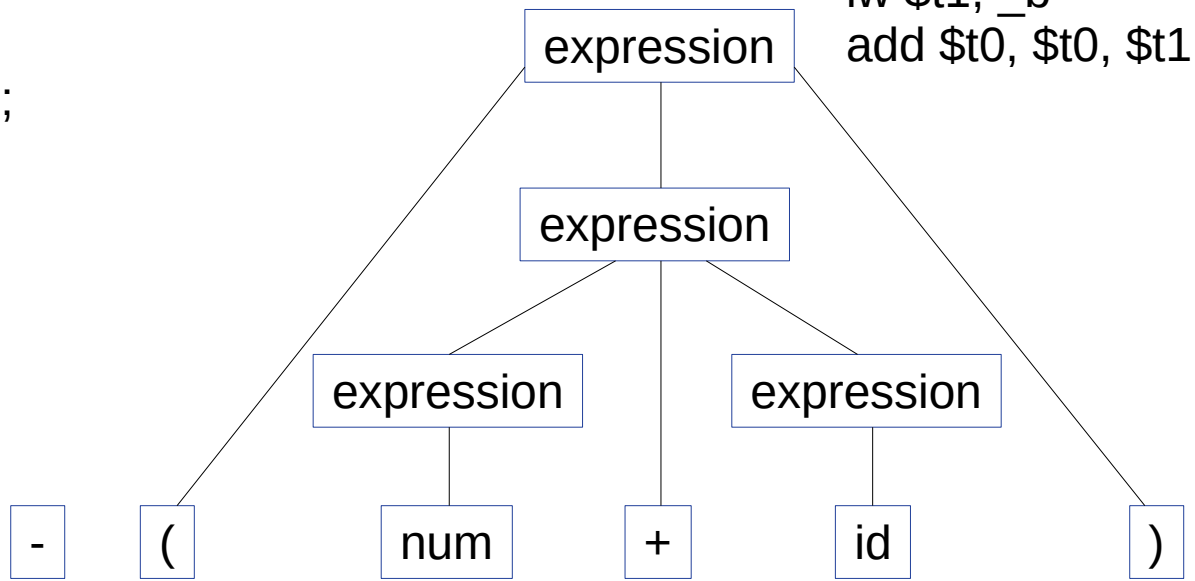
```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```



Objetivo: generar código de expresiones

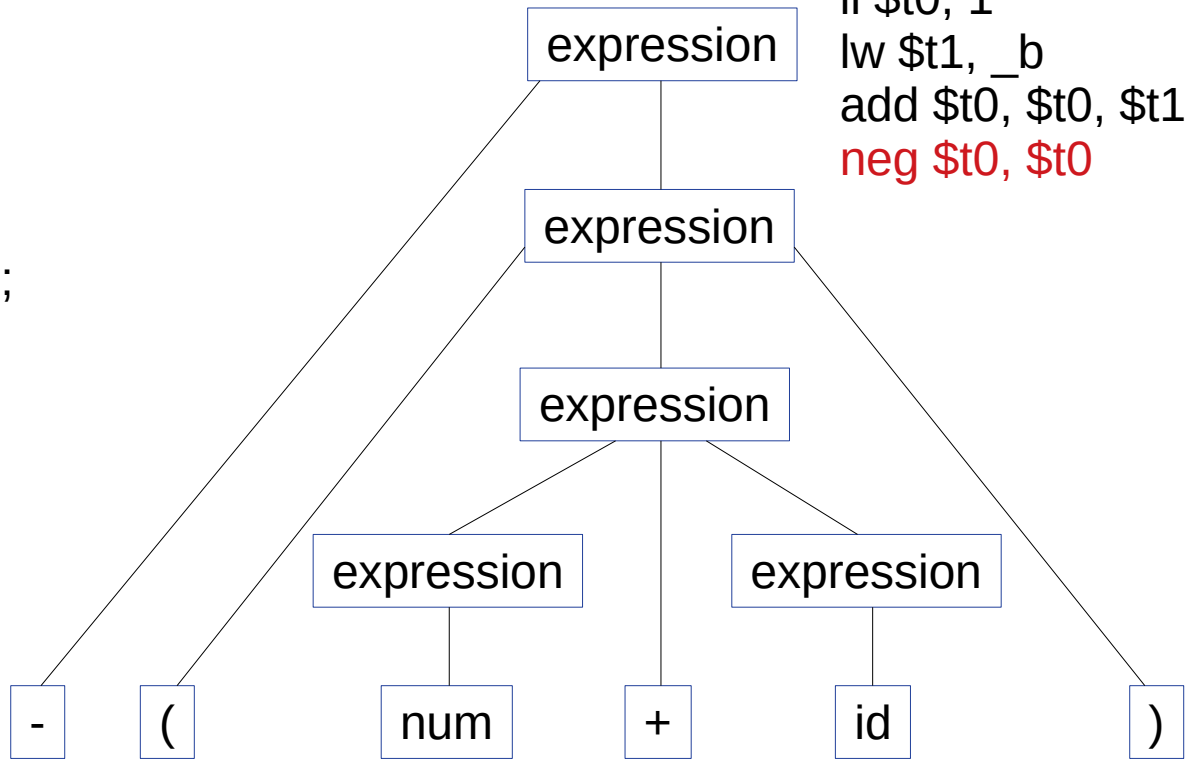
```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```

```
li $t0, 1
lw $t1, _b
add $t0, $t0, $t1
```



Objetivo: generar código de expresiones

```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```




```
li $t0, 1
lw $t1, _b
add $t0, $t0, $t1
neg $t0, $t0
```

Generación de código de expresiones (1)

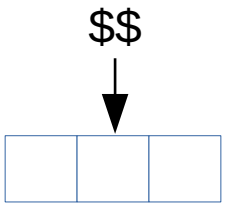
```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```

expression → **num** { }
$$

Generación de código de expresiones (2)

```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```

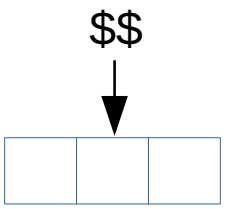
expression → **num** { $$ = creaLC(); }



Generación de código de expresiones (3)

```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```

expression → **num** { $$ = creaLC();
Operacion oper;
oper.op = “li”
oper.res = obtenerReg();
oper.arg1 = $1;
oper.arg2 = NULL; }



li	$t0	1	

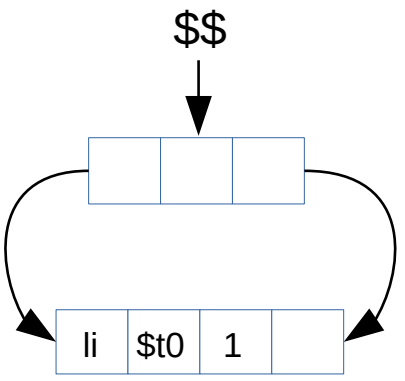
registros

1	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9

Generación de código de expresiones (4)

```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```

expression → **num** { $$ = creaLC();
Operacion oper;
oper.op = “li”
oper.res = obtenerReg();
oper.arg1 = $1;
oper.arg2 = NULL;
insertaLC($$,finalLC($$),oper); }



registros

1	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9

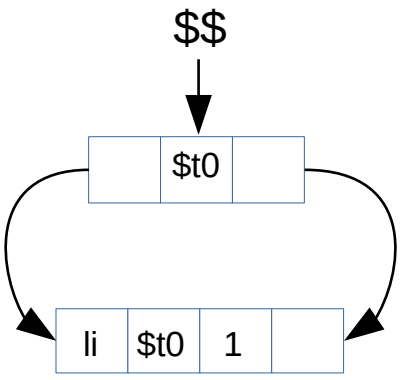
Generación de código de expresiones (5)

```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```

registros

1	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9

expression → **num** { $$ = creaLC();

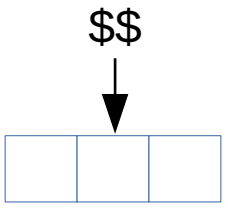



```
Operacion oper;
oper.op = “li”
oper.res = obtenerReg();
oper.arg1 = $1;
oper.arg2 = NULL;
insertaLC($$,finalLC($$),oper);
guardaResLC($$,oper.res);   }
```

Generación de código de expresiones (6)

```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```

expression → **id** { $$ = creaLC(); }



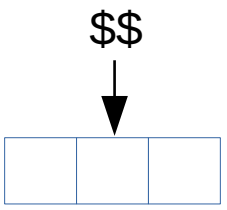
registros

1	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9

Generación de código de expresiones (7)

```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```

expression → **id** { $$ = creaLC();
Operacion oper;
oper.op = “lw”
oper.res = obtenerReg();
oper.arg1 = concatena(“_”,$1);
oper.arg2 = NULL; }



lw	$t1	_b	

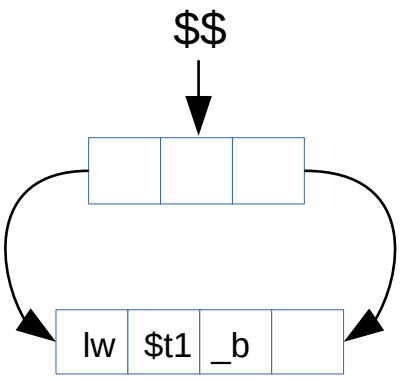
registros

1	1	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9

Generación de código de expresiones (8)

```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```

expression → **id** { $$ = creaLC();
Operacion oper;
oper.op = “lw”
oper.res = obtenerReg();
oper.arg1 = concatena(“_”,$1);
oper.arg2 = NULL;
insertaLC($$,finalLC($$),oper); }



registros

1	1	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9

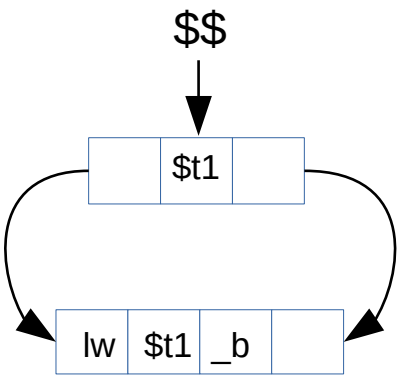
Generación de código de expresiones (9)

```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```

registros

1	1	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9

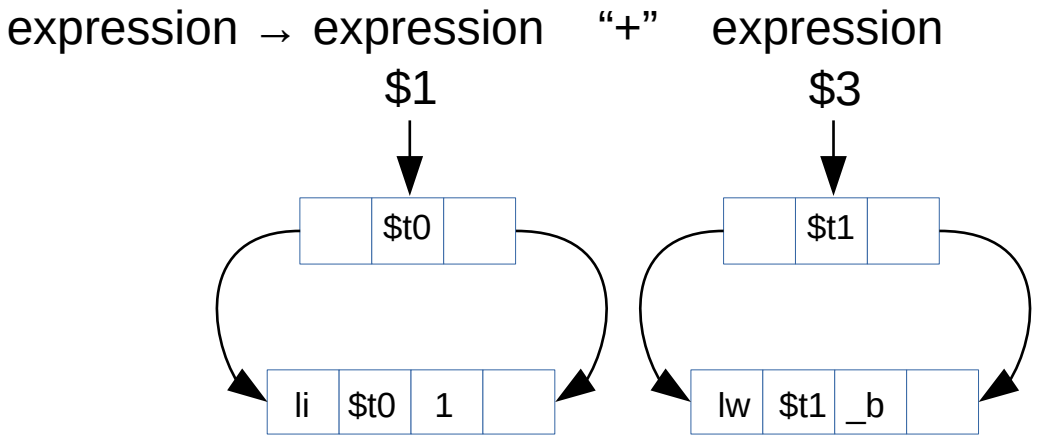
expression → **id** { $$ = creaLC();




```
Operacion oper;
oper.op = “lw”
oper.res = obtenerReg();
oper.arg1 = concatena(“_”,$1);
oper.arg2 = NULL;
insertaLC($$,finalLC($$),oper);
guardaResLC($$,oper.res);       }
```

Generación de código de expresiones (10)

```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```

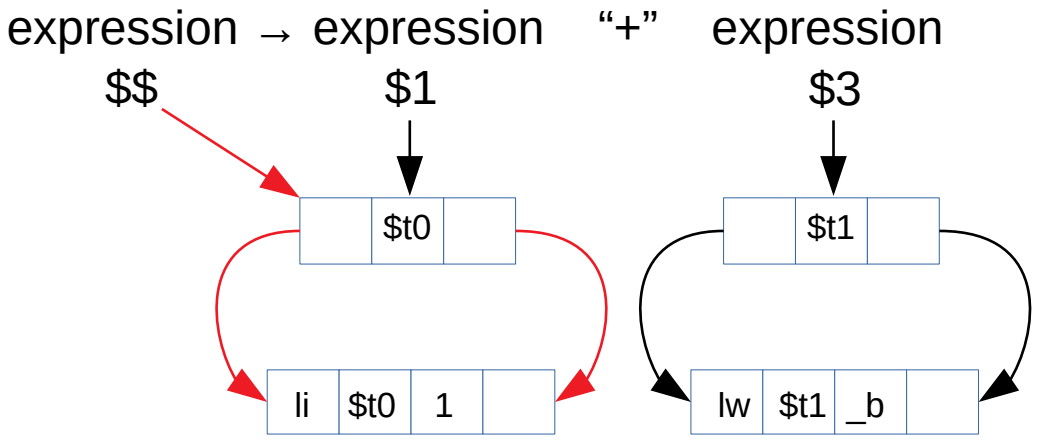


registros

1	1	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9

Generación de código de expresiones (11)

```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```



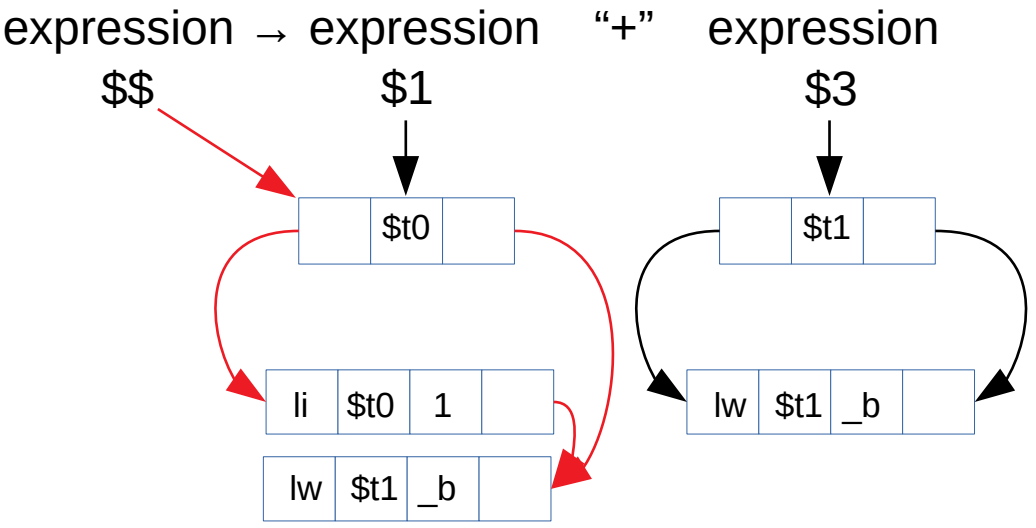
registros

1	1	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9

{ $$ = $1; }

Generación de código de expresiones (12)

```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```



registros

1	1	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9

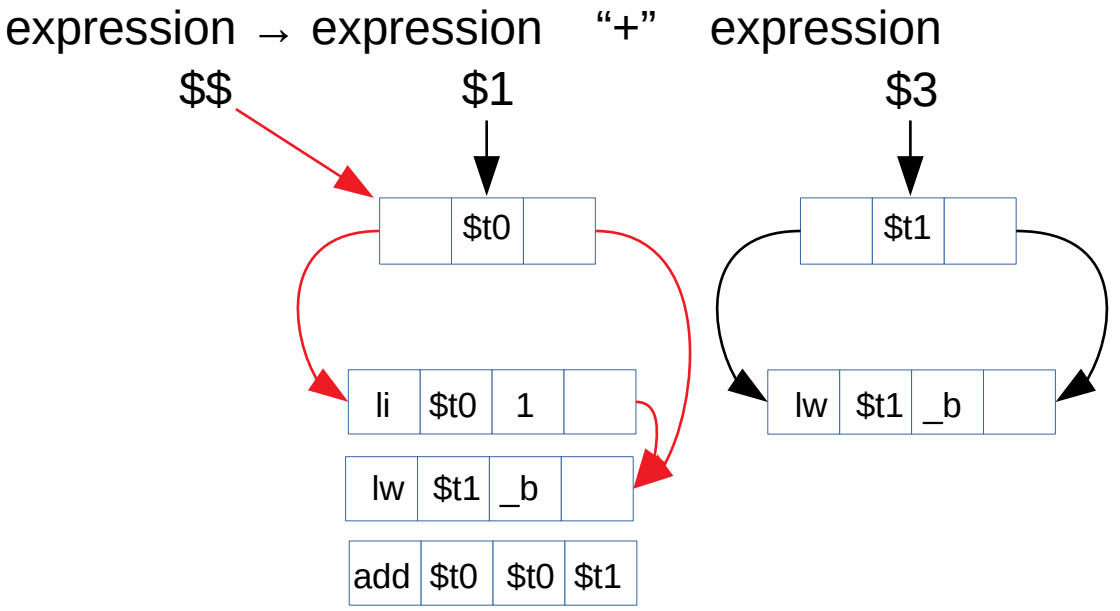
{ $$ = $1; concatenaLC($$,$3); }

Generación de código de expresiones (13)

```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```

registros

1	1	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9



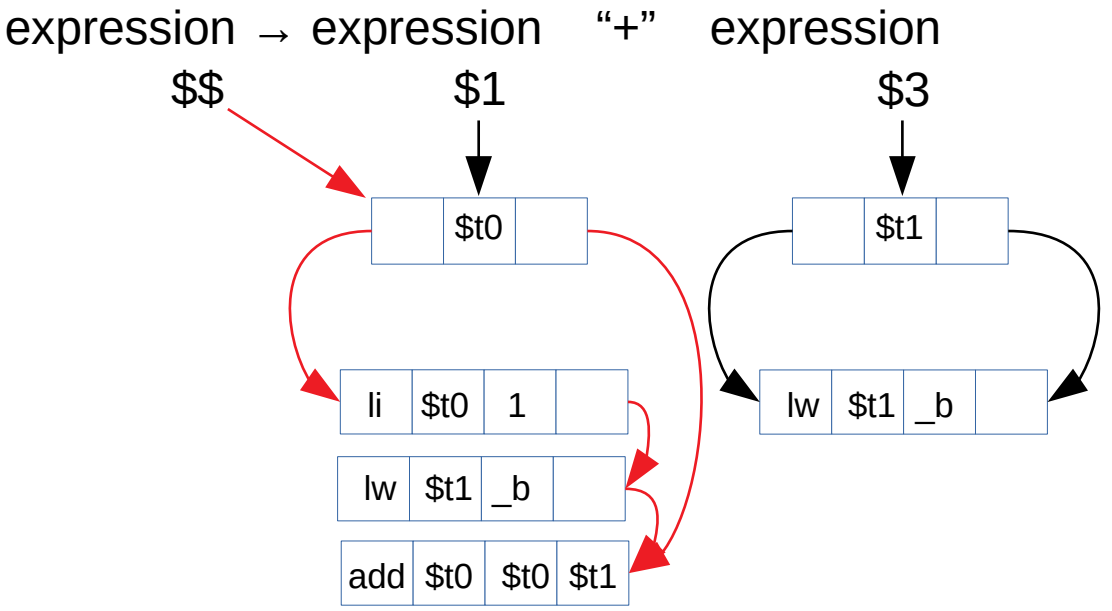
{ $$ = $1; concatenaLC($$,$3);
Operacion oper; oper.op = "add"; oper.res = recuperaResLC($1);
oper.arg1 = recuperaResLC($1); oper.arg2 = recuperaResLC($3); }

Generación de código de expresiones (14)

```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```

registros

1	1	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9



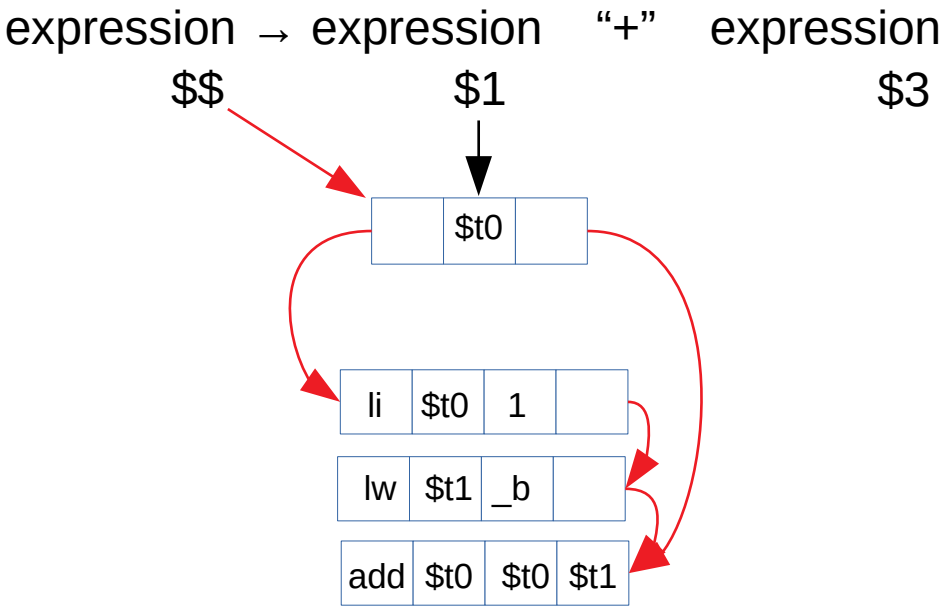
{ $$ = $1; concatenaLC($$,$3);
Operacion oper; oper.op = "add"; oper.res = recuperaResLC($1);
oper.arg1 = recuperaResLC($1); oper.arg2 = recuperaResLC($3);
insertaLC($$,finalLC($$),oper); }

Generación de código de expresiones (15)

```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```

registros

1	1	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9



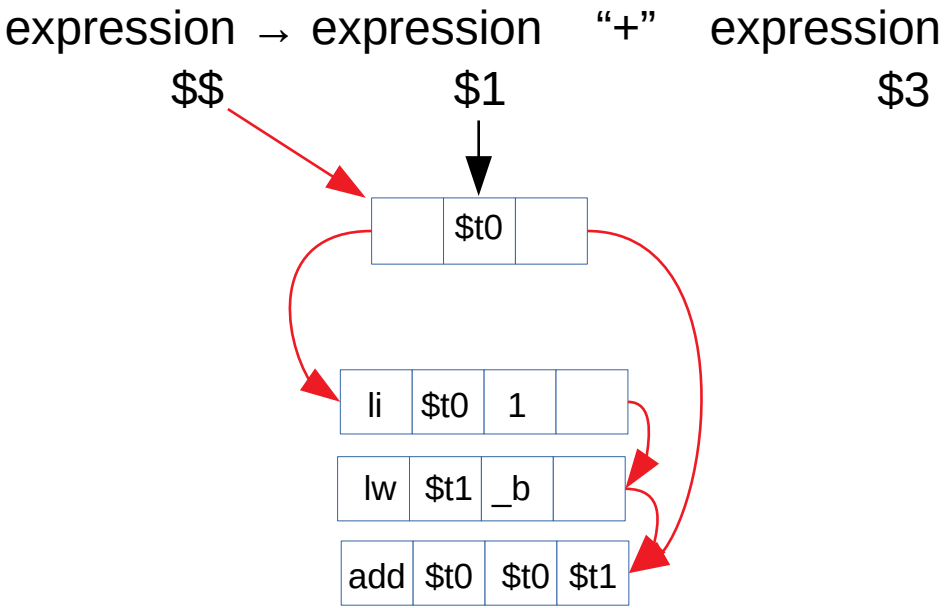
{ $$ = $1; concatenaLC($$,$3);
Operacion oper; oper.op = "add"; oper.res = recuperaResLC($1);
oper.arg1 = recuperaResLC($1); oper.arg2 = recuperaResLC($3);
insertaLC($$,finalLC($$),oper); liberaLC($3); }

Generación de código de expresiones (16)

```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```

registros

1	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9



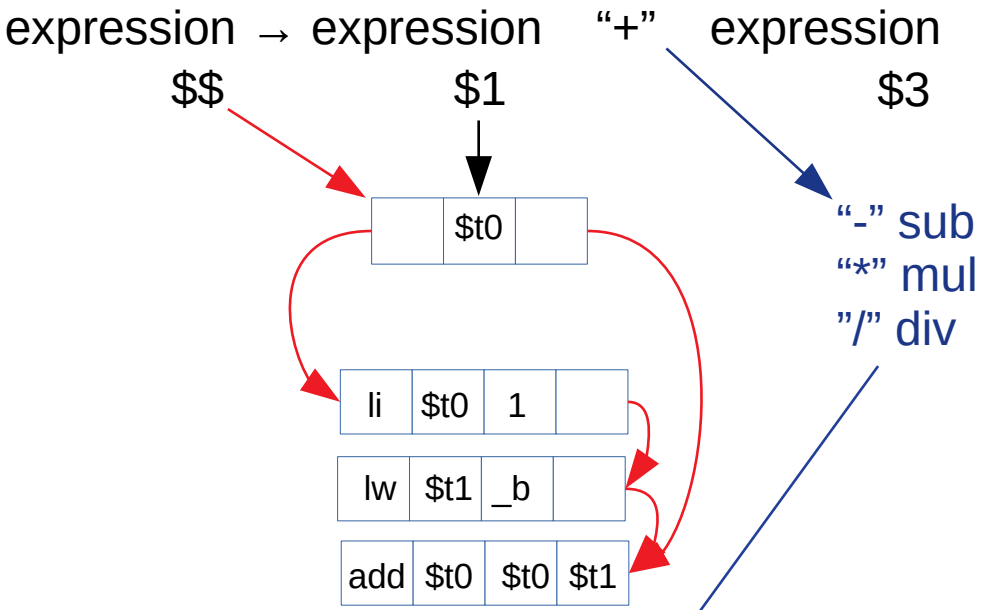

```
{ $$ = $1; concatenaLC($$,$3);
  Operacion oper; oper.op = "add"; oper.res = recuperaResLC($1);
  oper.arg1 = recuperaResLC($1); oper.arg2 = recuperaResLC($3);
  insertaLC($$,finalLC($$),oper); liberaLC($3);
  liberarReg(oper.arg2); }
```

Generación de código de expresiones (17)

```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```

registros

1	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9



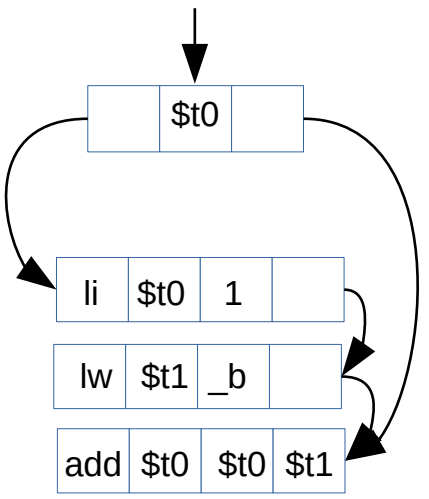

```
{ $$ = $1; concatenaLC($$,$3);
  Operacion oper; oper.op = “add”; oper.res = recuperaResLC($1);
  oper.arg1 = recuperaResLC($1); oper.arg2 = recuperaResLC($3);
  insertaLC($$,finalLC($$),oper); liberaLC($3);
  liberarReg(oper.arg2); }
```

Generación de código de expresiones (18)

```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```

expression → (expression) { }

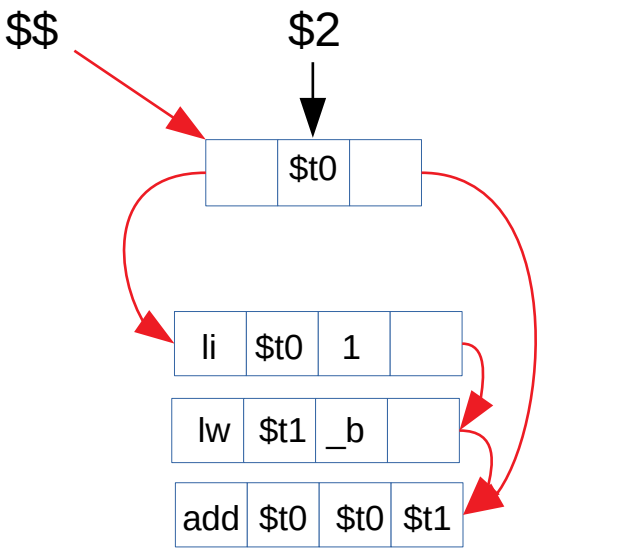
$$ $2



Generación de código de expresiones (19)

```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```

expression → (expression) { ?? }

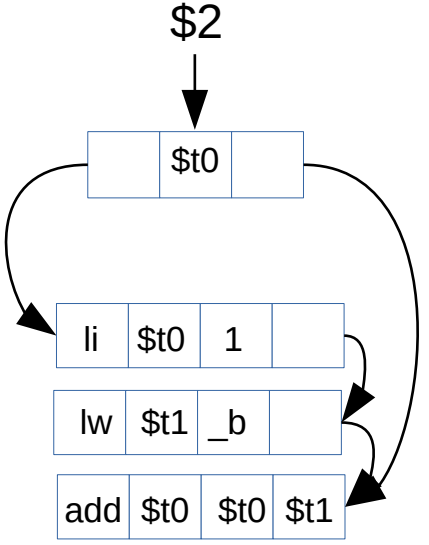


Generación de código de expresiones (20)

```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```

expression → - expression { }

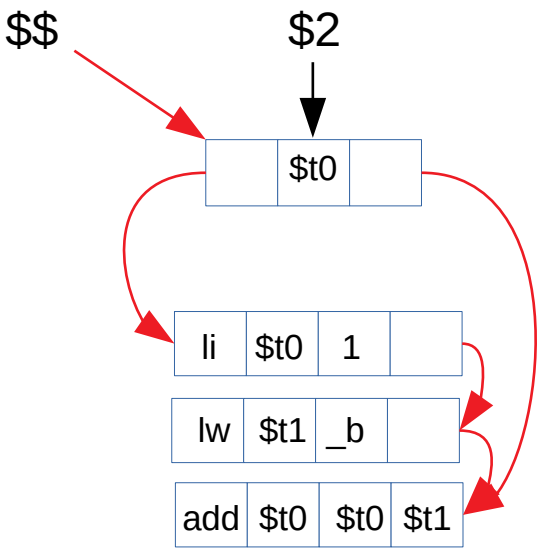
$$ $2



Generación de código de expresiones (21)

```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```

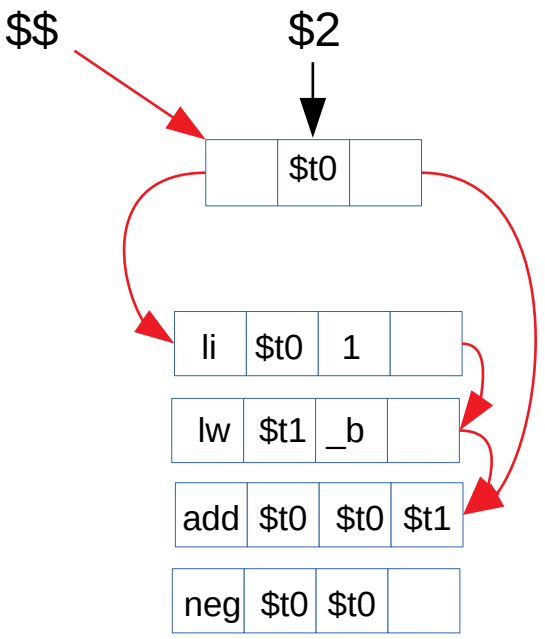
expression → - expression { ?? }



Generación de código de expresiones (22)

```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}
```

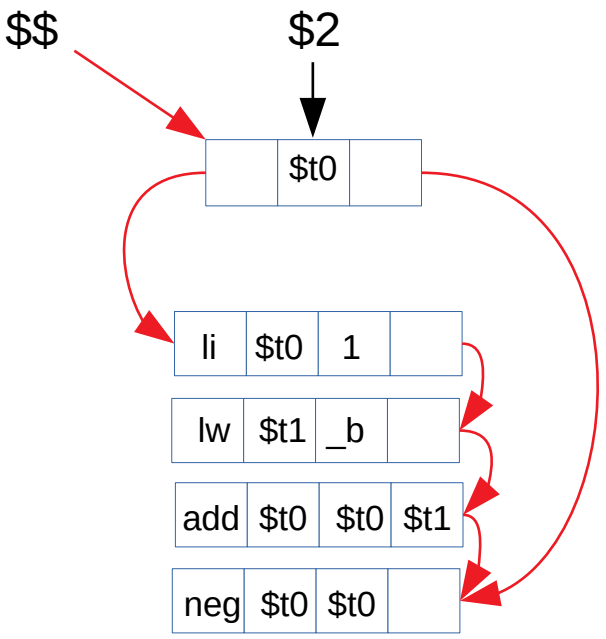
expression → - expression { ?? }



Generación de código de expresiones (23)

```
test1() {

        var int a;
        const int b = 3;

        a = -(1 + b);

}.
```

expression → - expression { ?? }



Generación de código de expresiones (24)

expression → expression ? expression : expression

```
# Traducción de 1 ? 2 : 3

        li $t0, 1
        beqz $l1
        li $t1, 2
        move $t0, $t1
        b $l2
$l1:
        li $t2, 3
        move $t0, $t2
$l2:
```

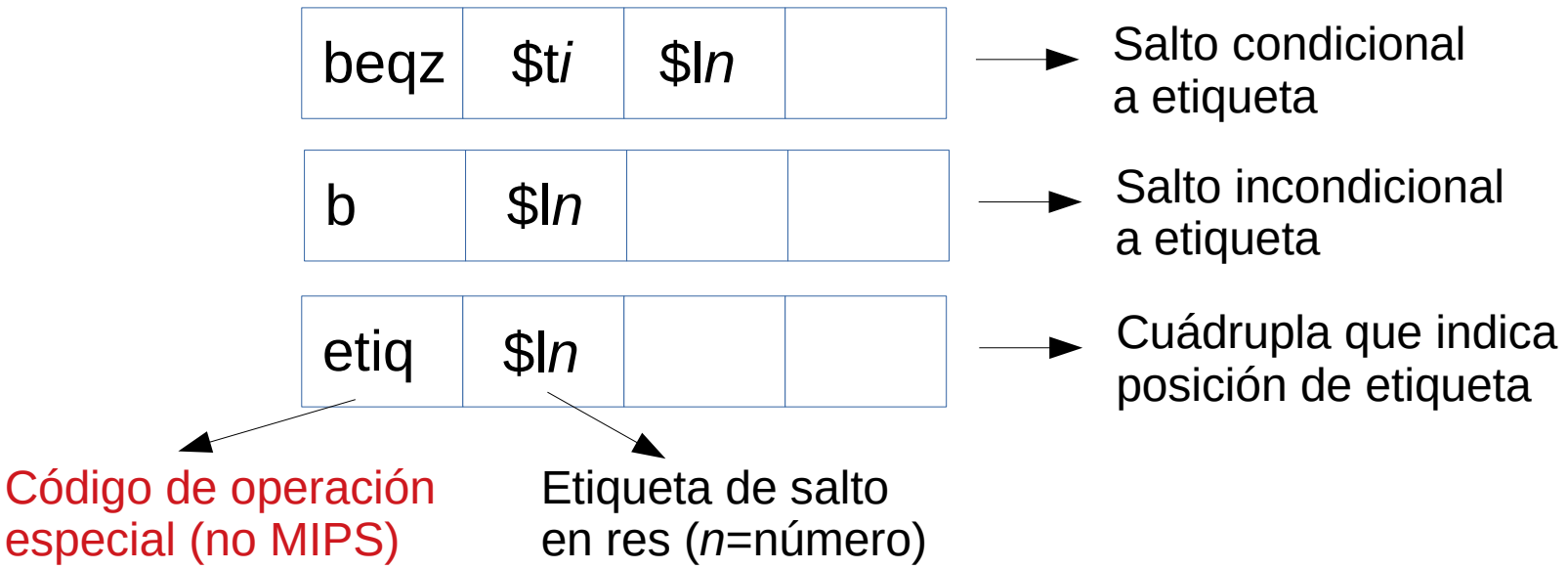
Etiquetas de salto (1)

- La generación de código con control de flujo necesita producir etiquetas de salto. Nuevo método en miniC.y:

```
%{ … int contador_etiq = 1; … %}

%%
…
%%

char *nuevaEtiqueta() {
        char *aux;
        asprintf(&aux,”$l%d”,contador_etiq++);
        return aux;
}
```

Etiquetas de salto (2)

- ¿Cómo representamos las etiquetas en la lista de operaciones?



¿Cómo puedo verificar el código?

- En fase de desarrollo, se puede hacer en:

  statement → ID “=” expression “;” { imprimirLC($3); }

Añadir a miniC.y el método imprimirLC(ListaC codigo)

El código del método se puede encontrar en testLC.c

```
void imprimirLC(ListaC codigo) {
    PosicionListaC p = inicioLC(codigo);
    while (p != finalLC(codigo)) {
      Operacion oper = recuperaLC(codigo,p);
      printf("%s",oper.op);
      if (oper.res) printf(" %s",oper.res);
      if (oper.arg1) printf(",%s",oper.arg1);
      if (oper.arg2) printf(",%s",oper.arg2);
      printf("\n");
      p = siguienteLC(codigos,p);
    }
}
```