

# Práctica 1 - Compiladores

Hecho por: *Elvis André Cruces Gómez y Yoshiro Milton Miranda Valdivia*

## Ejercicio 1

1. Redacta el siguiente código, genera el código ensamblador y explica en qué parte (del código ensamblador) se definen las variables *c* y *m*. (2 puntos).

---

```
int main(){
    char* c = "abcdef";
    int m = 11148;

    return 0;
}
```

---

## Código Ensamblador

```
1      .file      "Ejercicio_1.cpp"
2      .def       __main;      .scl   2;  .type   32; .endef
3      .section   .rdata,"dr"
4      LC0:
5          .ascii  "abcdef\0"
6          .text
7          .globl  __main
8          .def     __main;      .scl   2;  .type   32; .endef
9      __main:
10     LFB0:
11         .cfi_startproc
12         pushl   %ebp
13         .cfi_def_cfa_offset 8
14         .cfi_offset 5, -8
15         movl    %esp, %ebp
16         .cfi_def_cfa_register 5
17         andl    $-16, %esp
18         subl    $16, %esp
19         call    __main
20         movl    $LC0, 12(%esp)
21         movl    $11148, 8(%esp)
22         movl    $0, %eax
23         leave
24         .cfi_restore 5
25         .cfi_def_cfa 4, 4
26         ret
27         .cfi_endproc
28     LFE0:
29         .ident   "GCC: (MinGW.org GCC-6.3.0-1) 6.3.0"
```

La definición de la variable *c* se da en la línea de código 5 y la definición de la variable *m* se da en la línea de código 21.

## Ejercicio 2

2. Redacta el siguiente código, genera el código ensamblador y explica en qué parte (del código ensamblador) se define la división entre 8. (2 puntos).

---

```
int main(){
char* c = "abcdef";
int m = 11148;
int x = m/8;

return 0;
}
```

---

## Código Ensamblador

```
1      .file "Ejercicio_2.cpp"
2      .def __main; .scl 2; .type 32; .endef
3      .section .rdata,"dr"
4      LC0:
5          .ascii "abcdef\0"
6          .text
7          .globl _main
8          .def _main; .scl 2; .type 32; .endef
9      _main:
10     LFB0:
11         .cfi_startproc
12         pushl %ebp
13         .cfi_def_cfa_offset 8
14         .cfi_offset 5, -8
15         movl %esp, %ebp
16         .cfi_def_cfa_register 5
17         andl $-16, %esp
18         subl $16, %esp
19         call _main
20         movl $LC0, 12(%esp)
21         movl $11148, 8(%esp)
22         movl 8(%esp), %eax
23         cltd
24         andl $7, %edx
25         addl %edx, %eax
26         sarl $3, %eax
27         movl %eax, 4(%esp)
28         movl $0, %eax
29         leave
30         .cfi_restore 5
31         .cfi_def_cfa 4, 4
32         ret
33     .cfi_endproc
34     LFE0:
35     .ident "GCC: (MinGW.org GCC-6.3.0-1) 6.3.0"
```

La definición de la división entre 8 se da en las líneas de código desde la 22 a la 27.

## Ejercicio 3

3. Redacta el siguiente código, genera el código ensamblador y explica en qué parte (del código ensamblador) se define la división entre 4. (2 puntos).

```
int main(){
char* c = "abcdef";
int m = 11148;
int x = m/8;
int y = m/4;
int z = m/2;
return 0;
}
```

## Código Ensamblador

```
1      .file      "Ejercicio_3.cpp"
2      .def      __main;      .scl      2;      .type      32; .endef
3      .section  .rdata,"dr"
4      LC0:
5          .ascii  "abcdef\0"
6      .text
7      .globl    __main
8      .def      __main;      .scl      2;      .type      32; .endef
9      __main:
10     LFB0:
11         .cfi_startproc
12         pushl   %ebp
13         .cfi_def_cfa_offset 8
14         .cfi_offset 5, -8
15         movl    %esp, %ebp
16         .cfi_def_cfa_register 5
17         andl    $-16, %esp
18         subl    $32, %esp
19         call    __main
20         movl    $LC0, 28(%esp)
21         movl    $11148, 24(%esp)
22         movl    24(%esp), %eax
23         cltd
24         andl    $7, %edx
25         addl    %edx, %eax
26         sarl    $3, %eax
27         movl    %eax, 20(%esp)
28         movl    24(%esp), %eax
29         cltd
30         andl    $3, %edx
31         addl    %edx, %eax
32         sarl    $2, %eax
33         movl    %eax, 16(%esp)
34         movl    24(%esp), %eax
35         movl    %eax, %edx
36         shr    $31, %edx
37         addl    %edx, %eax
38         sarl    %eax
39         movl    %eax, 12(%esp)
40         movl    $0, %eax
41         leave
42         .cfi_restore 5
43         .cfi_def_cfa 4, 4
44         ret
45     .cfi_endproc
46     LFE0:
47     .ident      "GCC: (MinGW.org GCC-6.3.0-1) 6.3.0"
```

La definición de la división entre 4 se da en las líneas de código desde la 28 a la 33.

## Ejercicio 4

4. Redacta el siguiente código, genera el código ensamblador y explica en qué parte (del código ensamblador) se define la división entre 2. (2 puntos).

```
int main(){
    char* c = "abcdef";
    int m = 11148;
    int x = m/8;
    int y = m/4;
    int z = m/2;
    return 0;
}
```

## Código Ensamblador

```
1      .file      "Ejercicio_4.cpp"
2      .def       __main;      .scl      2;      .type      32; .endif
3      .section   .rdata,"dr"
4      LC0:
5          .ascii  "abcdef\0"
6          .text
7          .globl  __main
8          .def     __main;      .scl      2;      .type      32; .endif
9      __main:
10     LFB0:
11         .cfi_startproc
12         pushl   %ebp
13         .cfi_def_cfa_offset 8
14         .cfi_offset 5, -8
15         movl    %esp, %ebp
16         .cfi_def_cfa_register 5
17         andl    $-16, %esp
18         subl    $32, %esp
19         call    __main
20         movl    $LC0, 28(%esp)
21         movl    $11148, 24(%esp)
22         movl    24(%esp), %eax
23         cltd
24         andl    $7, %edx
25         addl    %edx, %eax
26         sarl    $3, %eax
27         movl    %eax, 20(%esp)
28         movl    24(%esp), %eax
29         cltd
30         andl    $3, %edx
31         addl    %edx, %eax
32         sarl    $2, %eax
33         movl    %eax, 16(%esp)
34         movl    24(%esp), %eax
35         movl    %eax, %edx
36         shrl    $31, %edx
37         addl    %edx, %eax
38         sarl    %eax
39         movl    %eax, 12(%esp)
40         movl    $0, %eax
41         leave
42         .cfi_restore 5
43         .cfi_def_cfa 4, 4
44         ret
45     .cfi_endproc
46     LFE0:
47     .ident      "GCC: (MinGW.org GCC-6.3.0-1) 6.3.0"
```

La definición de la división entre 2 se da en las líneas de código desde la 34 a la 39.

## Ejercicio 5

5. Redacta el siguiente código, genera el código ensamblador y explica: (4 puntos):

- En qué parte del código ensamblador se define la función *div4*.
- En qué parte del código ensamblador se invoca a la función *div4*.
- En qué parte del código ensamblador dentro de la función *div4* se procesa la división.

```
int div4(int x){
return x/4;
}

int main(){
char* c = "abcdef";
int m = 11148;
int x = m/8;
int y = m/4;
int z = m/2;

int rpt = div4(5);

return 0;
}
```

## Código Ensamblador

```
1  .file "Ejercicio_5.cpp"
2  .text
3  .globl _Z4div4i
4  .def _Z4div4i; .scl 2; .type 32; .endif
5  _Z4div4i:
6  LFB0:
7  .cfi_startproc
8  pushl %ebp
9  .cfi_def_cfa_offset 8
10 .cfi_offset 5, -8
11 movl %esp, %ebp
12 .cfi_def_cfa_register 5
13 movl 8(%ebp), %eax
14 cld
15 andl $3, %edx
16 addl %edx, %eax
17 sarl $2, %eax
18 popl %ebp
19 .cfi_restore 5
20 .cfi_def_cfa 4, 4
21 ret
22 .cfi_endproc
23 LFE0:
24 .def __main; .scl 2; .type 32; .endif
25 .section .rdata,"dr"
26 LC0:
27 .ascii "abcdef\0"
28 .text
29 .globl _main
30 .def _main; .scl 2; .type 32; .endif
31 _main:
32 LFB1:
33 .cfi_startproc
34 pushl %ebp
35 .cfi_def_cfa_offset 8
36 .cfi_offset 5, -8
37 movl %esp, %ebp
38 .cfi_def_cfa_register 5
39 andl $-16, %esp
40 subl $48, %esp
41 call __main
42 movl $LC0, 44(%esp)
43 movl $11148, 40(%esp)
44 movl 40(%esp), %eax
45 cld
46 andl $1, %edx
47 addl %edx, %eax
48 sarl $1, %eax
49 movl %eax, 36(%esp)
50 movl 40(%esp), %eax
51 cld
52 andl $1, %edx
53 addl %edx, %eax
54 sarl $2, %eax
55 movl %eax, 32(%esp)
56 movl 40(%esp), %eax
57 movl %eax, %edx
58 shrll $31, %edx
59 addl %edx, %eax
60 sarl %eax
61 movl %eax, 28(%esp)
62 movl $5, (%esp)
63 call _Z4div4i
64 movl %eax, 24(%esp)
65 movl $0, %eax
66 leave
67 .cfi_restore 5
68 .cfi_def_cfa 4, 4
69 ret
70 .cfi_endproc
71 LFE1:
72 .ident "GCC: (MinGW.org GCC-6.3.0-1) 6.3.0"
```

La definición de la función *div4* se da en las líneas de código desde la 5 hasta la 22.

La invocación de la función *div4* se da en la línea de código 63.

El proceso de división de la función *div4* se da en las líneas de código desde la 13 hasta la 17.

## Ejercicio 6

6. Redacta el siguiente código, genera el código ensamblador y explica: (4 puntos):

- En qué parte del código ensamblador se define la función *div*.
- En qué parte del código ensamblador se invoca a la función *div*.
- En qué parte del código ensamblador dentro de la función *div* se procesa la división.

```
int div(int x, int y){
    return x/y;
}

int div4(int x){
    return x/4;
}

int main(){
    char* c = "abcdef";
    int m = 11148;
    int x = m/8;
    int y = m/4;
    int z = m/2;

    int rpt = div(5,4);
    int rpt2 = div4(5);

    return 0;
}
```

## Código Ensamblador

```
1  .file "Ejercicio_6.cpp"
2  .text
3  .globl __Zdivii
4  .def __Zdivii; .sol 2; .type 32; .endef
5  __Zdivii:
6  LFB0:
7
8  .cfi_startproc
9  pushl %ebp
10 .cfi_def_cfa_offset 8
11 .cfi_offset 7, -8
12 movl %esp, %ebp
13 .cfi_def_cfa_register 5
14 movl 8(%ebp), %eax
15 cltd
16 idivl 4(%ebp)
17 popl %ebp
18 .cfi_restore 5
19 .cfi_def_cfa 4, 4
20 ret
21 .cfi_endproc
22 LFE0:
23 .globl __Zdivii
24 .def __Zdivii; .sol 2; .type 32; .endef
25 __Zdivii:
26 LFB1:
27 .cfi_startproc
28 pushl %ebp
29 .cfi_def_cfa_offset 8
30 .cfi_offset 7, -8
31 movl %esp, %ebp
32 .cfi_def_cfa_register 5
33 movl 8(%ebp), %eax
34 cltd
35 andl $0, %edx
36 addl %edx, %eax
37 sarl $0, %eax
38 popl %ebp
39 .cfi_restore 5
40 .cfi_def_cfa 4, 4
41 ret
42 .cfi_endproc
43 LFE1:
44 .def __main; .sol 2; .type 32; .endef
45 .section ".data","dr"
46 LC0:
47 .ascii "abcdef\0"
48 .text
49 .globl __main
50 .def __main; .sol 2; .type 32; .endef
51 __main:
52 LFB2:
53 .cfi_startproc
54 pushl %ebp
55 .cfi_def_cfa_offset 8
56 .cfi_offset 7, -8
57 movl %esp, %ebp
58 .cfi_def_cfa_register 5
59 andl $-16, %esp
60 subl $40, %esp
61 call __main
62 movl $LC0, 40(%esp)
63 movl $11148, 40(%esp)
64 movl 40(%esp), %eax
65 cltd
66 andl $0, %edx
67 addl %edx, %eax
68 sarl $0, %eax
69 movl %eax, 36(%esp)
70 movl 40(%esp), %eax
71 andl $0, %edx
72 addl %edx, %eax
73 sarl $0, %eax
74 movl %eax, 32(%esp)
75 movl 40(%esp), %eax
76 movl %eax, %edx
77 shril $31, %edx
78 addl %edx, %eax
79 sarl %eax
80 movl %eax, 28(%esp)
81 movl $0, 4(%esp)
82 movl $1, (%esp)
83 call __Zdivii
84 movl %eax, 24(%esp)
85 movl $0, (%esp)
86 call __Zdivii
87 movl %eax, 20(%esp)
88 movl $0, %eax
89 leave
90 .cfi_restore 5
91 .cfi_def_cfa 4, 4
92 ret
93 .cfi_endproc
94 LFE2:
95 .ident "GCC: (MinGW.org GCC-6.3.0-1) 6.3.0"
```

La definición de la función *div* se da en las líneas de código desde la 5 hasta la 20.

La invocación de la función *div* se da en la línea de código 83.

El proceso de división de la función *div* se da en las líneas de código desde la 13 hasta la 15.

## Ejercicio 7

7. De las preguntas anteriores, se ha generado código por cada función, ambas dividen entre 4, pero difieren un poco en su implementación. Investigue a qué se debe dicha diferencia y comente cuáles podrían ser las consecuencias. **(4 puntos)**

### Explicación

La primera diferencia que encontramos entre las funciones `div` y `div4` es que: La función `div` cuenta con 2 parámetros (`x`, `y`) de tipo entero (`int`), que tiene como finalidad dividir el primer parámetro sobre el segundo (`x / y`); mientras que la función `div4` solo cuenta con un parámetro (`x`) de tipo entero (`int`), que tiene la finalidad de dividir el parámetro sobre 4. Y en ambas funciones se retornará un entero, que es el tipo de dato que se declararon en estas.

Entre las consecuencias destacamos que, al usar 2 parámetros para realizar alguna operación, se pueden definir los dos lados de la operación, ya sea en una suma, resta, multiplicación, división, potencia, raíz, etc., como en el ejercicio 6, donde se definen el numerador y el denominador, los cuales cambian dependiendo el valor que se le asigne. Por otro lado, al usar un solo parámetro en la función, se tendría un valor estático en la operación, como en el ejercicio 6, donde el único valor que cambia en la función `div4`, es el de `x`, que siempre se dividirá entre 4.

En resumen, el uso de 1 operador suele determinar un curso para la solución del problema, pero al usar 2 o más parámetros (como en este caso, enteros) la solución puede ser de diferentes formas. Basándonos en el ejercicio 6, con la función `div4`, siempre tendremos una división sobre 4; por otro lado, con la función `div`, se puede obtener una división sobre 2, 3, 4, 5, etc., obteniendo así números enteros o decimales.