

Práctica 01

DOCENTE	CARRERA	CURSO
MSc. Vicente Enrique Machaca Arceda	Escuela Profesional de Ingeniería de Software	Compiladores

PRÁCTICA	TEMA	DURACIÓN
01	Introducción	3 horas

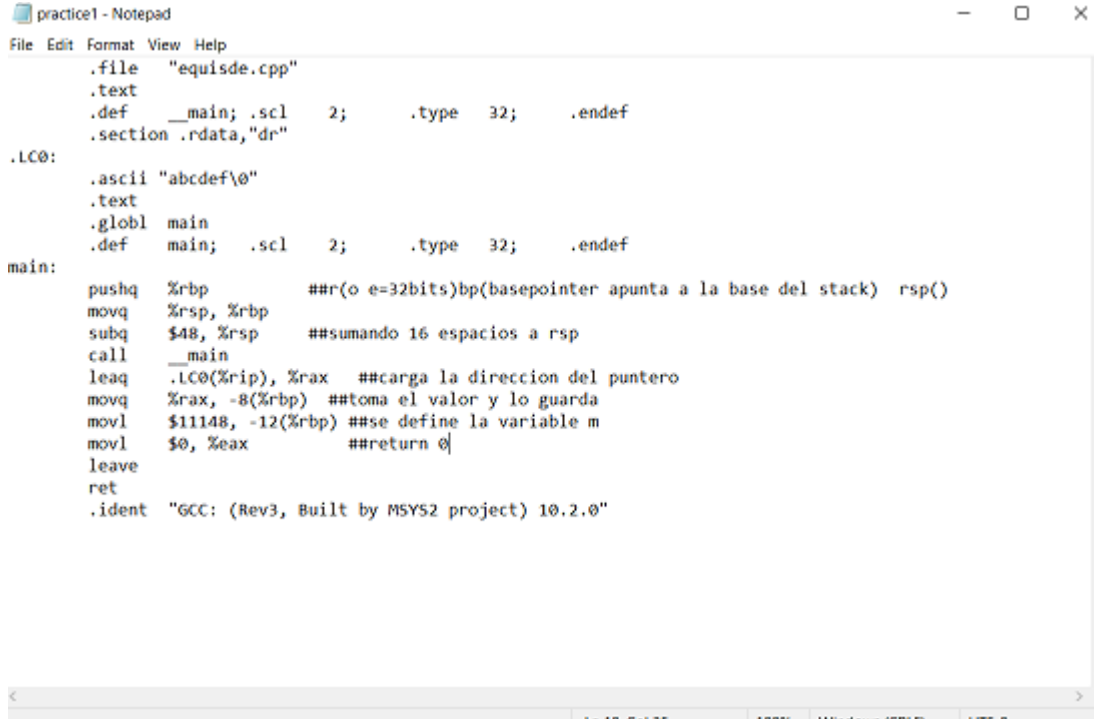
1. Datos de los estudiantes

- Grupo: x
- Integrantes:
 - Medina Pauca, Walther Mauricio

2. Ejercicios

1. Redacta el siguiente código, genera el código ensamblador y explica que parte (del código ensamblador) se definen las variables c y m.

Solución



```
.file "equisde.cpp"
.text
.def __main; .scl 2; .type 32; .endef
.section .rdata,"dr"
.LC0:
.ascii "abcdef\0"
.text
.globl main
.def main; .scl 2; .type 32; .endef
main:
    pushq %rbp                ##(o e=32bits)bp(basepointer apunta a la base del stack) rsp()
    movq %rsp, %rbp
    subq $48, %rsp            ##sumando 16 espacios a rsp
    call __main
    leaq .LC0(%rip), %rax      ##carga la direccion del puntero
    movq %rax, -8(%rbp)        ##toma el valor y lo guarda
    movl $11148, -12(%rbp)    ##se define la variable m
    movl $0, %eax              ##return 0
    leave
    ret
.ident "GCC: (Rev3, Built by MSYS2 project) 10.2.0"
```

- Redacta el siguiente código, genera el código ensamblador y explica que parte (del código ensamblador) se definen la división entre 8.

Solución

```
practice2-2 - Notepad
File Edit Format View Help
.file "equisde.cpp"
.text
.def __main; .scl 2; .type 32; .endef
.globl main
.def main; .scl 2; .type 32; .endef

main:
    pushq %rbp    rbp##apunta a la base del stackframe actual
    movq %rsp, %rbp    rsp##apunta al top, parte superior del stackframe actual
    subq $48, %rsp    ##apunta a una direccion y luego suma 48 espacios a rsp para poder almacenar
    call __main
    movl $11148, -4(%rbp)
    movl -4(%rbp), %eax
    leal 7(%rax), %edx    ##carga de direccion
    testl %eax, %eax    ##comparacion
    cmovs %edx, %eax    ##condicional para que continue
    sarl $3, %eax    ##desplazamiento aritmetico a la derecha
    movl %eax, -8(%rbp)    ##mueve el valor del 2do al 1er opeprando
    movl $0, %eax    ##return
    leave
    ret
.ident "GCC: (Rev3, Built by MSYS2 project) 10.2.0"
```

- Redacta el siguiente código, genera el código ensamblador y explica que parte (del código ensamblador) se definen la división entre 4.

Solución

```
practice3 - Notepad
File Edit Format View Help
main:
    pushq %rbp
    movq %rsp, %rbp
    subq $64, %rsp
    call __main

    leaq .LC0(%rip), %rax
    movq %rax, -8(%rbp)
    movl $11148, -12(%rbp)
    movl -12(%rbp), %eax
    leal 7(%rax), %edx
    testl %eax, %eax
    cmovs %edx, %eax
    sarl $3, %eax
    movl %eax, -16(%rbp)
    movl -12(%rbp), %eax    ##genera espacio para la operacion (empieza div4)
    leal 3(%rax), %edx    ##carga la direccion del valor
    testl %eax, %eax    ##realiza una comparacion entre los valores
    cmovs %edx, %eax    ##hace una copia pero con condicion, si esta no se cumple se pasa al siguiente
    sarl $2, %eax    ##desplazamiento aritmetico a la derecha
    movl %eax, -20(%rbp)    ##toma el valor generado (termina div4)
    movl -12(%rbp), %eax
    movl %eax, %edx
    shrl $31, %edx
    addl %edx, %eax
    sarl %eax
    movl %eax, -24(%rbp)
    movl $0, %eax
    leave

<
Ln 37, Col 15 100% Windows (CRLF) UTF
```

4. Redacta el siguiente código, genera el código ensamblador y explica que parte (del código ensamblador) se definen la división entre 2.

Solución

```
practice4 - Notepad
File Edit Format View Help
subq $64, %rsp
call __main

leaq .LC0(%rip), %rax ##carga la direccion
movq %rax, -8(%rbp)
movl $11148, -12(%rbp)
movl -12(%rbp), %eax
leal 7(%rax), %edx
testl %eax, %eax
cmovs %edx, %eax
sarl $3, %eax
movl %eax, -16(%rbp)
movl -12(%rbp), %eax
leal 3(%rax), %edx
testl %eax, %eax
cmovs %edx, %eax
sarl $2, %eax
movl %eax, -20(%rbp)
movl -12(%rbp), %eax ##genera espacio para la operacion (empieza div2)
movl %eax, %edx ##toma el valor para la operacion
shrl $31, %edx ##desplazamiento a la derecha
addl %edx, %eax ##suma los valores
sarl %eax ##desplazamiento aritmetico a la derecha
movl %eax, -24(%rbp) ##toma el valor generado (termina div2)
movl $0, %eax
leave
ret
.ident "GCC: (Rev3, Built by MSYS2 project) 10.2.0"
```

5. Redacta el siguiente código, genera el código ensamblador y explica que parte (del código ensamblador) se definen la función *div4*, se invoca la función *div4* y en que parte de la función *div4* se procesa la división.

Solución

```
practice5 - Notepad
File Edit Format View Help
.def __Z4div4i; .scl 2; .type 32; .endef

__Z4div4i: ##define la funcion div4 con un parametro(i)
pushq %rbp
movq %rsp, %rbp
movl %ecx, 16(%rbp)
movl 16(%rbp), %eax ##crea espacio para la operacion (empieza div)
leal 3(%rax), %edx ##lee la direccion del valor
testl %eax, %eax ##compara los valores
cmovs %edx, %eax ##toma el valor condicionando
sarl $2, %eax ##desplazamiento aritmetico a la derecha
popq %rbp ##recuperar valor de la pila
ret
.def __main; .scl 2; .type 32; .endef
.section .rdata,"dr"
```

6. Redacta el siguiente código, genera el código ensamblador y explica que parte (del código ensamblador) se definen la función *div*, se invoca la función *div* y en que parte de la función *div* se procesa la división.

Solución

```
practice6 - Notepad
File Edit Format View Help
.file "equisde.cpp"
.text
.globl __Z3divii
.def __Z3divii; .scl 2; .type 32; .endef

__Z3divii: ##se define la funcion div con 2 parametros(ii)
pushq %rbp ##guarda el valor
movq %rsp, %rbp ##toma el valor guardado
movl %ecx, 16(%rbp)
movl %edx, 24(%rbp)
movl 16(%rbp), %eax ##genera espacio para la operacion
cld
idivl 24(%rbp) ##division entera
popq %rbp ##recupera el valor de la pila
ret
.globl __Z4div4i
.def __Z4div4i; .scl 2; .type 32; .endef

sarl $2, %eax
movl %eax, -20(%rbp)
movl -12(%rbp), %eax
movl %eax, %edx
shrl $31, %edx
addl %edx, %eax
sarl %eax
movl %eax, -24(%rbp)
movl $4, %edx
movl $5, %ecx
call __Z3divii ##se llama la funcion div
movl %eax, -28(%rbp)
movl $5, %ecx
call __Z4div4i
movl %eax, -32(%rbp)
movl $0, %eax
```

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.

Solución

```
pushq   %rbp
movq    %rsp, %rbp
subq    $64, %rsp
call    __main
leaq    .LC0(%rip), %rax
movq    %rax, -8(%rbp)
movl    $111148, -12(%rbp)
movl    -12(%rbp), %eax
leal    7(%rax), %edx
testl   %eax, %eax
cmovs   %edx, %eax
sarl    $3, %eax
movl    %eax, -16(%rbp)
movl    -12(%rbp), %eax
leal    3(%rax), %edx
testl   %eax, %eax
cmovs   %edx, %eax
sarl    $2, %eax
movl    %eax, -20(%rbp)
movl    -12(%rbp), %eax
movl    %eax, %edx
```

##La diferencia entre dichas implementaciones se encuentra en la cantidad de parametros que recibe, mientras que una recibe 1 la otra 2, esto puede implicar mucho al tiempo de ejecutar ya que al esperar solo 1 esta proxivamente a ser realizada, mientras que la otra al tener que esperar dos, esta mas condicionado a que los valores terminen de ser generados ya que dicho proceso podria alterar el resultado esperado. Por consiguiente vemos que una consecuencia del uso es el tiempo que tarda o podria tardar el proceso.