|  |  |
| --- | --- |
| **Apellidos, Nombre:** | RUIZ FERNANDEZ, ANGEL |
| **DNI:** | 23836363Z |
| **Puntuación (rellenar sólo en autoevaluación):** | **TEST:: P1:: P2:: TOTAL:** |

**Tarea 5 de Fundamentos de Computadores**1° curso de Grado en Ingeniería Informática  
Fecha de entrega: *4 de diciembre de 2023*

**Test (3.0 puntos; 0.3 por apartado)** Rellene la siguiente tabla con la respuesta correcta a las preguntas de test que siguen. Escriba para ello **una X** en la celda correspondiente a cada respuesta correcta. Cada pregunta tiene una y sólo una respuesta correcta (una X para cada una de las columnas T1-T10). Cada 3 respuestas incorrectas anularán 1 correcta.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **T1** | **T2** | **T3** | **T4** | **T5** | **T6** | **T7** | **T8** | **T9** | **T10** |
| **a** |  |  |  |  |  |  |  | X | X |  |
| **b** | X |  |  |  |  | X |  |  |  |  |
| **c** |  |  | X |  |  |  |  |  |  | X |
| **d** |  | X |  | X | X |  | X |  |  |  |

**T1.** Con respecto al código objeto, señala la afirmación VERDADERA:  
 a) Contiene código máquina que puede ser directamente ejecutado por la CPU.  
 b) Es generado por el ensamblador.  
 c) No contiene referencias a funciones o variables no definidas  
 d) Es generado por el enlazador.

**T2.** Con respecto a los operandos de las instrucciones en x86-64, señala la afirmación VERDADERA:  
 a) Ninguna de las otras tres respuestas es correcta.  
 b) Hay instrucciones que emplean únicamente operandos constantes  
 c) Una instrucción de escritura en memoria siempre tiene al menos un operando de tipo registro  
 d) Los operandos constantes se codifican en la propia instrucción

**T3.** En la jerarquía de traducción, el compilador se ejecuta inmediatamente antes del:  
 a) planificador  
 b) cargador  
 c) ensamblador  
 d) enlazador

**T4.** Con respecto a las instrucciones de movimiento de datos en el ISA x86-64, señala la afirmación VERDADERA:  
 a) Ninguna de las otras tres respuestas es correcta.  
 b) La instrucción movl 0x123456,%eax pone el valor 0x123456 en el registro EAX.  
 c) La instrucción movl %eax,0x123456, pone el valor 0x123456 en el registro EAX.  
 d) La instrucción movl $0x123456,%eax carga el valor constante 0x123456 en los 32 bits de menos peso del registro RAX.

**T5.** Con respecto a las instrucciones de salto en el ISA x86-64, señala la afirmación VERDADERA:  
 a) Los saltos condicionales comprueban la condición en la propia instrucción de salto.   
 b) Ninguna de las otras tres respuestas es correcta.  
 c) Las únicas instrucciones que rompen el flujo de ejecución secuencial son los saltos condicionales y incondicionales  
 d) Las instrucciones de salto condicional leen el registro RFLAGS.

**T6.** En lo referido el código ejecutable, señala la afirmación VERDADERA:  
 a) Ninguna de las otras tres respuestas es correcta.  
 b) Es generado por el ensamblador.  
 c) Es independiente de la arquitectura de la máquina sobre la que se programa.  
 d) No contiene referencias a funciones o variables no definidas

**T7.** Señala la respuesta FALSA sobre el enlazado con librerías estáticas:  
 a) Da lugar a ficheros ejecutables de mayor tamaño que el enlazado dinámico  
 b) Da lugar a ficheros ejecutables que no tienen dependencias de ninguna librería externa  
 c) Conduce a un peor aprovechamiento de la memoria.  
 d) El enlazador busca las librerías estáticas en ficheros llamados lib\*.so.

**T8.** En relación al código ensamblador, señala la afirmación VERDADERA:  
 a) Cada instrucción tiene una correspondencia directa con una instrucción en código máquina.  
 b) Es independiente de la arquitectura de la máquina sobre la que se programa.  
 c) Ninguna de las otras tres respuestas es correcta.  
 d) Las instrucciones están codificadas en lenguaje máquina.

**T9.** En relación al lenguaje ensamblador, señala la afirmación VERDADERA:  
 a) El comienzo del segmento de datos viene marcado por la directiva .data.  
 b) Ninguna de las otras tres respuestas es correcta.  
 c) Las variables locales de un procedimiento se almacenan en el segmento de datos.  
 d) Las etiquetas generadas automáticamente por el compilador aparecen en el segmento de código pero no en el de datos.

**T10.** En relación a la reubicación del código, señala la afirmación VERDADERA:  
 a) Sólo las instrucciones de movimiento de datos necesitan ser reubicadas.  
 b) La reubicación de direcciones es llevada a cabo por el ensamblador  
 c) La información de reubicación es generada por el programa ensamblador.  
 d) Ninguna de las otras tres respuestas es correcta.

**P1. (2.0 puntos; 0.25 por apartado)** Indicar el valor correcto con el que rellenar cada hueco en la siguiente tabla. Ten en cuenta que las comillas sólo se usan como delimitadores de la orden, nombre de fichero, o similar:

|  |  |
| --- | --- |
| **Pregunta** | **Respuesta** |
| **En x86-64, las instrucciones que operan con los 8 bits de menos peso del registro RAX usan como nombre de registro \_\_\_\_\_.** | al |
| **En x86-64, la instrucción que lee el contenido de la dirección de memoria apuntada por el registro RBP y lo guarda en el registro RAX es \_\_\_\_\_\_.** | No pregunta que dialecto de ensamblador usar, pero ya que en las diapositivas se suele ver gas o AT&T usaré ese (pero estoy mas acostumbrado a notación Intel)  movl (%rbp), %rax |
| **En x86-64, el registro que guarda el resultado de una instrucción de comparación es \_\_\_\_\_.** | rflags |
| **Al registro que indica en qué instrucción del programa se encuentra actualmente la ejecución se le llama de manera genérica \_\_\_\_\_ (aunque en el ISA de Intel se llama forma distinta).** | Contador de programa (PC)  En x86\_64 rip, instruction pointer |
| **En x86-64, la instrucción que se utiliza para terminar un procedimiento es \_\_\_\_\_\_.** | ret |
| **En x86-64, el registro que permite acceder a las variables locales en la pila durante la ejecución de una subrutina es \_\_\_\_\_.** | pop  o mov del stack relativo a rbp |
| **En x86-64, la instrucción que se utiliza para saltar a la etiqueta fin\_for es \_\_\_\_\_\_.** | jmp fin\_for |
| **En x86-64, la instrucción que se utiliza para guardar el valor del registro RCX en la pila es \_\_\_\_\_\_.** | pushl %rcx |

**P2. (5.0 puntos; 0.25 por apartado)** Considérese el siguiente programa fuente escrito en C (en el que se ha ocultado con XX el valor de una determinada constante del array):

#include<stdio.h>  
  
// El valor XX del siguiente array es en realidad un número entero entre   
// -100 y 100 (que está oculto porque deberá deducirse posteriormente):  
long array[6] = {-8,4,3,XX,-8,2};   
  
int funcion(long varA, int varB) {  
 return (((varB + 6) >> 2) \* (-(17 \* varA)));  
}  
  
int main() {  
 int i, value;  
 for(i=5;i>=0;i--) {  
 array[i] = funcion(array[i], -18);  
 }  
}

El anterior programa ha sido compilado y enlazado, para generar un ejecutable. A continuación se muestra el desensamblado del código de la función principal del mismo, main. Obsérvese que el código máquina de la instrucción callq ha sido también ocultado (indicado con puntos suspensivos ...):

00000000046346b5 <main>:  
 46346b5: 55 push %rbp  
 46346b6: 48 89 e5 mov %rsp,%rbp  
 46346b9: 48 83 ec 10 sub $0x10,%rsp  
 46346bd: c7 45 fc 05 00 00 00 movl $0x5,-0x4(%rbp)  
 46346c4: eb 2e jmp 46346f4 <main+0x3f>  
 46346c6: 8b 45 fc mov -0x4(%rbp),%eax  
 46346c9: 48 98 cltq   
 46346cb: 48 8b 04 c5 a0 18 99 06 mov 0x69918a0(,%rax,8),%rax  
 46346d3: be ee ff ff ff mov $0xffffffee,%esi  
 46346d8: 48 89 c7 mov %rax,%rdi  
 46346db: ... callq 463468d <funcion>  
 46346e0: 48 63 d0 movslq %eax,%rdx  
 46346e3: 8b 45 fc mov -0x4(%rbp),%eax  
 46346e6: 48 98 cltq   
 46346e8: 48 89 14 c5 a0 18 99 06 mov %rdx,0x69918a0(,%rax,8)  
 46346f0: 83 6d fc 01 subl $0x1,-0x4(%rbp)  
 46346f4: 83 7d fc 00 cmpl $0x0,-0x4(%rbp)  
 46346f8: 79 cc jns 46346c6 <main+0x11>  
 46346fa: c9 leaveq   
 46346fb: c3 retq

Una vez cargado el programa en memoria, utilizando gdb se ha inspeccionado el contenido de la misma. En particular, se muestran a continuacion la salida de los comandos x/64bx main y x/48bx array respectivamente, que han sido introducidos justo después de cargar el ejecutable en memoria, pero antes de comenzar la ejecución (y donde, de nuevo, se han ocultado unos cuantos bytes, sustituyéndolos por las cadenas 0xYY y 0xZZ):

0x46346b5 <main>: 0x55 0x48 0x89 0xe5 0x48 0x83 0xec 0x10  
0x46346bd <main+8>: 0xYY 0xYY 0xYY 0xYY 0xYY 0xYY 0xYY 0xeb  
0x46346c5 <main+16>: 0x2e 0x8b 0x45 0xfc 0x48 0x98 0x48 0x8b  
0x46346cd <main+24>: 0x04 0xc5 0xa0 0x18 0x99 0x06 0xbe 0xee  
0x46346d5 <main+32>: 0xff 0xff 0xff 0x48 0x89 0xc7 0xe8 0xad  
0x46346dd <main+40>: 0xff 0xff 0xff 0x48 0x63 0xd0 0x8b 0x45  
0x46346e5 <main+48>: 0xfc 0x48 0x98 0x48 0x89 0x14 0xc5 0xa0  
0x46346ed <main+56>: 0x18 0x99 0x06 0x83 0x6d 0xfc 0x01 0x83

0x69918a0 <array>: 0xZZ 0xZZ 0xZZ 0xZZ 0xZZ 0xZZ 0xZZ 0xZZ  
0x69918a8 <array+8>: 0x04 0x00 0x00 0x00 0x00 0x00 0x00 0x00  
0x69918b0 <array+16>: 0x03 0x00 0x00 0x00 0x00 0x00 0x00 0x00  
0x69918b8 <array+24>: 0xf7 0xff 0xff 0xff 0xff 0xff 0xff 0xff  
0x69918c0 <array+32>: 0xf8 0xff 0xff 0xff 0xff 0xff 0xff 0xff  
0x69918c8 <array+40>: 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00

Finalmente, se comienza a ejecutar el programa durante un número indeterminado de instrucciones. En un momento determinado, en mitad de dicha ejecución, se ejecutan los dos siguientes comandos de gdb con el fin de inspeccionar tanto los contenidos del array como los contenidos de los registros %rax, %edx y %rip en ese preciso momento:

... El programa está detenido en cierto momento en mitad de su ejecución...  
(gdb) p array  
$1 = {-8, 4, 3, -9, -408, 102}  
(gdb) info registers rax edx rip  
rax 0x3 3  
edx 0xfffffe35 -459  
rip 0x46346e8 0x46346e8 <main+51>

A la vista de toda la información anterior, e inspeccionándola con atención, se deben deducir las contestaciones todas las preguntas siguientes:

|  |  |
| --- | --- |
| **Pregunta** | **Respuesta** |
| **¿En qué dirección comienza el código de 'main'? (Expresa dicha dirección en hexadecimal, por ejemplo 0x1234AB)** | 0x00000000046346b5 |
| **¿Crees que, simplemente a la vista del código de 'main', se puede deducir la dirección de comienzo del array? (SI/NO, y en caso afirmativo, expresa dicha dirección en hexadecimal, por ejemplo 0x1234AB)** | Si, 0x69918a0 |
| **¿En qué dirección del código se encuentra la instrucción que realiza el primer acceso en escritura a dicho array? (exprésala en hexadecimal, por ejemplo 0x1234AB)** | 0x46346e8 |
| **Indica el código máquina de la instrucción de la subrutina 'main' que realiza la inicialización del registro adecuado para pasar una determinada constante entera como segundo parámetro a la subrutina 'funcion' (expresa dicho código máquina como lista de bytes en hexadecimal, ejemplo 0x1a 0x55 0x2b)** | 0xbe 0xee 0xff 0xff 0xff |
| **¿Qué crees que están codificando exactamente los 4 últimos bytes de dicho código máquina?** | La constante inmediata (en little endian) |
| **¿Cuál es la dirección de la instrucción de salto CONDICIONAL que cierra el código ensamblador generado por el bucle for? (exprésala en hexadecimal, por ejemplo 0x1234AB)** | 0x46346f8 |
| **¿Cuál es la dirección de DESTINO del salto al que se refiere la pregunta anterior? (exprésala en hexadecimal, por ejemplo 0x1234AB)** | 0x46346C6 |
| **Indica el valor que se almacenará en la pila en el momento en que se ejecute la instrucción callq de llamada a la subrutina 'funcion' (expresa dicho valor en hexadecimal, por ejemplo 0x1234AB)** | 0x46346e0 (próxima instrucción, rip + tamaño de la instrucción call) |
| **El código máquina de la anterior instrucción callq ha sido deliberadamente omitido, pero su TAMAÑO en número de bytes se puede deducir perfectamente con la información mostrada. ¿Cuál es dicho tamaño en bytes? (Expresarlo como entero en decimal)** | 5 bytes (opcode + pointer near de 32 bits) |
| **¿Cuántos bytes está ocupando exactamente TODO el código máquina completo de la subrutina 'main'? (expresa el número de bytes en decimal, por ejemplo, 99)** | 71 bytes |

|  |  |
| --- | --- |
| **Pregunta** | **Respuesta** |
| **¿Cuál era el valor de la constante entera XX contenida en el array en el programa en C original? (expresar la respuesta como un entero en decimal)** |  |
| **¿Entre qué dos direcciones exactamente (ambas inclusive), se encuentra almacenado en memoria dicho valor? (ejemplo, direcciones 0x123456 a 0x123459)** |  |
| **¿Cuál es el código máquina de la instrucción call, que aparecía oculto con puntos suspensivos (...) en el listado desensamblado de la subrutina 'main'?** |  |
| **¿Qué valores tienen los siete bytes ocultos con 0xYY en el volcado de bytes a partir de la etiqueta 'main'? (expresar la lista de bytes en hexadecimal, ejemplo 0x12 0x34 0x56 0x78 0x9a 0xbc 0xde 0xf0 )** |  |
| **¿Qué valores tienen los ocho bytes ocultos con 0xZZ en el volcado de bytes a partir de la etiqueta 'array'? (expresar la lista de bytes en hexadecimal, ejemplo 0x12 0x34 0x56 0x78 0x9a 0xbc 0xde 0xf0)** |  |
| **¿En qué instrucción exactamente está detenido el programa? (indicar su código ensamblador completo, ejemplo push %rbp)** |  |
| **Indica la dirección de memoria exacta donde comienzan los 8 bytes a los que accede dicha instrucción (expresa dicho valor en hexadecimal, por ejemplo 0x1234AB)** |  |
| **Expresa en castellano lo que hace exactamente esa instrucción (su funcionalidad al ejecutarse con respecto a las variables del programa)** |  |
| **¿Cuál crees que es el valor de la variable i del programa en C en el momento en que está parado el programa?** |  |
| **Indica la salida que proporcionará el gdb si, tras ejecutar la instrucción en la que el programa está parado, volviésemos a imprimir el array con 'p array'.** |  |