Nombre: José Francisco Altamirano Zevallos Clave única: 000163314 Calif:\_\_\_\_\_\_\_\_

Cada una de las preguntas: La 1 vale 3.0 pts., las 2 y 3 valen 2.0 pts cada una y la 4 vale 3 pts. Duración 1 hs 55 min.

**Deberá subir a Canvas, en el Módulo OPCExámenesAD20, en Entrega2doExmAD20, el archivo de nombre genérico “SgEx967456ARBI.zip”, conteniendo el archivo del programa ensamblador y este archivo con las respuestas de las tres preguntas restantes. 967456 representa su clave única y ARBI son las inciales de su nombre.**

1. Elaboración de programa en ensamblador

En el archivo “*SgEx.asm*” se encuentran el siguiente código con un arreglo de cadenas de caracteres; la siguiente muestra es sólo un parcial del programa en el archivo.

|  |
| --- |
| .DATA  AryStr BYTE "on1on", ?  BYTE "tw2tw", ?  BYTE "th3tw", ?  BYTE "fo4fo", ?  BYTE "fi5fi", ?  BYTE "si6si", ?  BYTE "se7se", ?  BYTE "ei8ei", ?  BYTE "ni9ni", ?  N SDWORD 9 ; Total de strings  T SDWORD ? ; Tamanho de cada string |

Se le pide imprimir las N cadenas de caracteres (strings) del arreglo ArgStr, junto con otra información que se despliega más abajo. El algoritmo que usted implemente debe servir genéricamente para cualquier cantidad de cadenas. Las N cadenas antes definidas son todas del mismo tamaño T, 5 caracteres; pero usted deberá calcular de manera automática, este valor, a partir de .DATA. De igual manera, cualquier modificación a los nueve elementos del arreglo (AryStr) deberá hacerla mediante programa ensamblador.

El programa deberá imprimir, al principio de la corrida, su clave única seguido de sus iniciales, por ejemplo, supongamos “967456ARBI”.

El archivo “.asm” deberá renombrarlo con sus iniciales y clave única, “SgEx967456ARBI.asm”.

A continuación, un ejemplo de salida:

|  |
| --- |
| Soy: 967456ARBI  Longitud de cada string: 5  Lista de strings:========  String 1: on1on  String 2: tw2tw  . . . . .  String 9: ni9ni  ADIOS |

Nombre: José Francisco Altamirano Zevallos Clave única: 000163314 Calif:\_\_\_\_\_\_\_\_

Cada una de las preguntas: La 1 vale 3.0 pts., las 2 y 3 valen 2.0 pts cada una y la 4 vale 3 pts.

**CONTESTAR CON TINTA**. Duración 1 hs 55 min o de 7-8:55 hs.

1. Instrucciones y pedazos de programa.

Conteste el siguiente inciso basándose en un ejemplo de uso. Indique la base en que está expresando los contenidos de los registros.

1. Suponga que no cuenta con la instrucción NEG, ¿Con cuál o cuáles instrucciones puede substituir la operación de la instrucción NEG?

Debido a que lo que hace la instrucción NEG es realizar un complemento a 2, esto puede ser substituido al juntar la instrucción NOT con la instrucción INC; esto se debe a que la instrucción NOT realiza un complemento a 1, y al sumarle un uno con la instrucción INC, se consigue un complemento a 2.

**B)** ¿Cuáles son los valores finales de las banderas ZF y CF así como de AL? ¿Por qué?

MOV AL, 4

CMP AL, 5

Debido a que CMP funciona realizando una resta entre el primer operando y el segundo, sabemos entonces que realizaría la operación: 4 – 5. Por ello, la bandera ZF = 0, y la bandera CF = 1.

**C)** ¿Cuál o cuáles son los valores de EAX y ECX, así como de algún otro registro involucrado?

MOV EAX,-48

CDQ

Sabemos que EAX contiene el valor FFFFFFD0h (-48), y como CDQ extiende el bit de signo del EAX al EDX, sabemos entonces que EDX contiene el valor FFFFFFFFh

MOV ECX, 5

IDIV ECX

Sabemos que ECX contiene el valor 00000005h, y como IDIV lo que hace es que realiza una división en la que EAX contiene lo que se va a dividir, en EDX se pone el sobrante y en ECX se pone el divisor y finalmente en EAX se guarda el resultado de la división. Por ello, los valores finales son: EAX contiene FFFFFFF7h (-9), ECX contiene 00000005h y EDX contiene FFFFFFFCh (-4).

1. Directivas y operadores. Dado el siguiente pedazo de código

.DATA

byte1 BYTE 10, 20, 30 ;suponga un offset 0 para la etiqueta "byte1"

cade3 BYTE "GFEDCBA", 0

array2 WORD 1, 2, 3, 4, 5

array1 DWORD 20 DUP(?), 0, 0

**A)** Explique el valor contenido en el registro EAX después de ejecutar cada instrucción. En caso de error descríbalo.

MOV EAX, SIZEOF array1

Debido a que SIZEOF me regresa TYPE\*LENGTHOF, entonces sabemos que SIZEOF de array1 regresa el valor 88 (4\*22); este valor se puede guardar en EAX, por lo que EAX contiene el valor 00000058h (88).

**B)** Explique el valor contenido en el registro ECX después de ejecutar cada instrucción. En caso de error, descríbalo.

MOV ECX, DWORD PTR array2+2

Debido a que array2+2 me coloca en el valor 02h, al pasarlo a DWORD mediante el PTR, ECX contendrá el valor 00030002h.

**C)** Interprete el siguiente código, indicando los valores finales de ESI y EDX así como su respectivo significado, y lo impreso en la consola. En caso de error, descríbalo.

MOV ESI, 3

SUB BYTE PTR [cade3+ESI], 2

MOV EDX, OFFSET cade3

CALL WriteString

Al final ESI contendrá el valor 00000003h; EDX contendrá el valor 00000003h, ya que ese es el offset de cade3; como sabemos que en cade3+ESI se encuentra la letra “D” cuyo valor en hexadecimal es 44h, entonces al restarle 2 se convertirá en el valor 42h, el cual representa a la letra “B”; por lo tanto, la impresión en consola dirá: “GFEBCBA”

1. Traduzca el siguiente programa de algún lenguaje de alto nivel a lenguaje ensamblador.

main(...) {

---------------------------------

int uno, dos;

int tres [] = {4, 5, 6, 7, 8, 9};

int cuatro = 3;

int cinco = 2;

uno = cuatro + 3 \* cinco;

if( uno <= 12 ) {

dos = -34;

} else {

cinco = tres[1];

}

while( cuatro == 3 ) {

dos = tres[2];

}

---------------------------------

}

La implementación de las estructuras algorítmicas debe quedar muy claras, usando saltos condicionales; no puede utilizar macro directivas de control de flujo (.IF, :WHILE, …).

Los int considérelos como SDWORD.

En ningún caso use LOOP.

En la siguiente página viene la información sobre los saltos condicionales.

uno SDWORD ?

dos SDWORD ?

tres SDWORD 4, 5, 6, 7, 8, 9

cuatro SDWORD 3

cinco SDWORD 2

MOV EAX, cinco

MOV EBX, 3

IMUL EBX

ADD EAX, cuatro

MOV uno, EAX

CMP uno, 12

JG else

MOV dos, -34

JMP finif

else:

MOV cinco, tres + 4

finif:

while:

CMP cuatro, 3

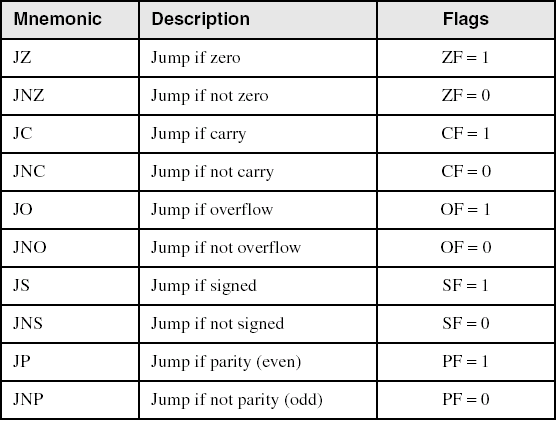
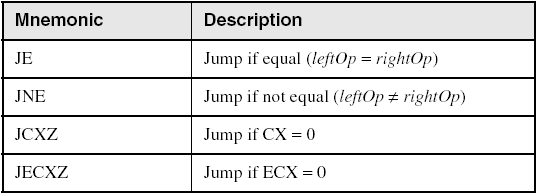
JNE finwhile

MOV dos, tres + 8

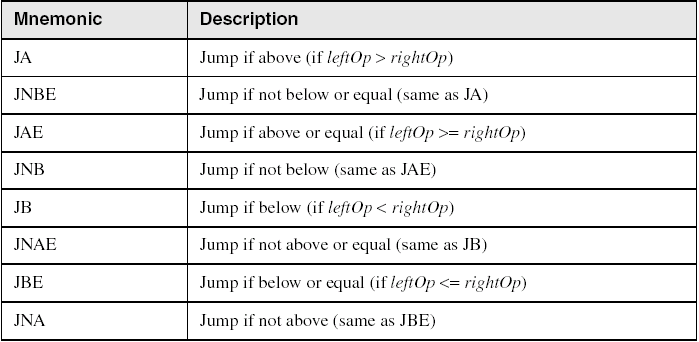
JMP while

finwhile:

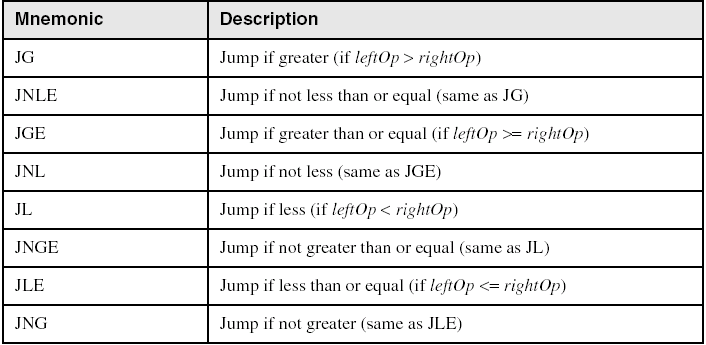
exit

Jconds based on Specific Flags. Jconds based on Equality



Jconds based on Unsigned operands.



Jconds based on Signed operands.