Universidad Autónoma de Baja California Facultad de Ciencias Químicas e Ingeniería



ORGANIZACIÓN DE LAS COMPUTADORAS Y LENGUAJE ENSAMBLADOR

Practica 11

Interfaz entre el lenguaje C89 y el lenguaje ensamblador procesador 8086

Docente: Sanchez Herrera Mauricio Alonso

Alumno: Gómez Cárdenas Emmanuel Alberto

Matricula: 1261509

Contenido

TEORIA	2
DESARROLLO	3
Parte 1	3
Parte 2	3
CONCLUSIONES	3
REFERENCIA	3
ANEXO	4
Parte 1 myputchar	4
Parte 1 (pract11.c)	4
Parte 2 (mulMat3x3)	5
Parte 2 (Matrices.c)	

TEORIA

Funciones del lenguaje de ensamblador pueden ser enlazadas con un programa C. Aunque se tienen que tomar muchas precauciones al usar registros, obtener parámetros, llamar a otras funciones y a la hora de entrar o salir de funciones.

Cuando los registros son utilizados en lenguaje ensamblador, deben seguirse las reglas al obtener o modificarlos. Especialmente cuando se utilizan registros de propósitos especiales.

El compilador hace asunciones sobre como las funciones tratan los registros.

Unas de las tantas reglas que existen al momento de guardar o recuperar valores en los registros son:

Los registros deben ser salvados haciéndoles push al stack

Al inicio de una función de lenguaje ensamblador, todos los registros que han de ser usados en la función deben ser guardados. Al final de la función, estos han de ser recuperados.

Antes de una llamada a función, todos los registros que contienen datos importantes deben de ser guardados y recuperados después de la función.

En cuanto a C, los argumentos son pasados a las funciones poniéndolos en registros o el stack, siguiendo las reglas:

Hasta 3 argumentos pueden ser pasados en registros

Una vez que un argumento sea pasado al stack, todos los demás (a la derecha) están en el stack.

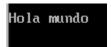
Todos los valores mayores a 32 bits son pasados por el stack.

Cuando argumentos son pasados usando el stack, se hace push de derecha a izquierda, por ej. test(int a, char b, float c);

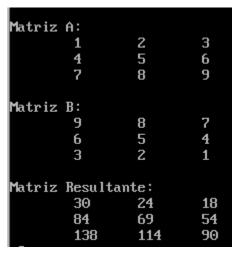
Primero se mandan el parámetro c, luego el b y por último el a.

DESARROLLO

Parte 1



Parte 2



CONCLUSIONES

Esta práctica nos abrió la posibilidad de poder implementar un código en ensamblador para posteriormente ser ejecutado en código C, lo cual es muy útil si se quiere intentar implementar un procedimiento o función lo más eficiente posible, gracias a que ensamblador nos ofrece un control total (o casi total) de los procedimientos que ejecuta el procesador. Tambien es útil para aquellos momentos en los que no existe una funcion especifica en lenguaje C de algo que se quiere realizar (desplazamiento aritmético a la derecha).

REFERENCIA

Signal.uu.se. (2020). from http://www.signal.uu.se/Staff/pd/DSP/Doc/ctools/chap4.pdf.

ANEXO

Parte 1 (myputchar.asm)

```
.model small
.code
public _myputchar
_myputchar PROC
    push bp
    mov bp,sp
    mov dl,[bp+4]
    mov ah,2
    int 21h
    pop bp
    ret
_myputchar ENDP
END
```

Parte 1 (pract11.c)

```
#include <stdio.h>
extern void myputchar(char x);
char *str = {"Hola mundo\n"};
void main(void){
    while(*str) myputchar(*str++);
    getchar();
}
```

Parte 2 (mulMat3x3.asm)

```
MODEL small
LOCALS
.DATA
    aux dw 0
.CODE
public _mulMat3x3
mulMat3x3 PROC
           push bp
            mov bp,sp
            push ax
            push bx
            push cx
            push dx
            push di
@loop:
           push cx
            mov ch,0
            mov bl,3
            sub bl,cl
            mov bh,0
            mov aux,0
           mov di,[bp+4]
            mov ah,0
            mov al,3
            mov dl,bl
            mov dh, bh
            call getnumMat
            push ax
            mov di,[bp+6]
            mov ah,0
            mov al,3
            mov dl,bh
            mov dh, ch
            call getnumMat
            pop dx
            mul dx
```

```
add aux, ax
            inc bh
            cmp bh,3
            jne @@nxtnum
            push cx
            mov di, [bp+8]
            mov al,3
            mov dl,bl
            mov dh,ch
            mov cx, aux
            call setnumMat
            pop cx
            inc ch
            cmp ch,3
            jne @@nxt3
            pop cx
            loop @@loop
            pop di
            pop dx
            pop cx
            pop bx
            pop ax
            pop bp
            ret
 mulMat3x3 ENDP
getnumMat PROC
            push bx
            push di
            mul dl
            add al,dh
            mov bx,ax
            mov al,di[bx]
            mov ah,0
            pop di
            pop bx
            ret
            ENDP
getnumMat
```

```
setnumMat
            PROC
            push ax
            push bx
            push cx
            push di
            mul dl
            add al,dh
            mov bx,ax
            mov di[bx],cl
            pop di
            pop cx
            pop bx
            pop ax
            ret
            ENDP
setnumMat
END
```

Parte 2 (Matrices.c)

```
#include<stdio.h>
extern void mulMat3x3(char[], char[]);
void printMat(unsigned char[], int);
void main(){
    //Uso de chars, ya que el codigo en asm esta hecho para 8 bits
    char matA[9] = \{1,2,3,4,5,6,7,8,9\};
    char matB[9] = \{9,8,7,6,5,4,3,2,1\};
    char matR[9] = \{0\};
    printf("\nMatriz A:\n");
    printMat(matA,9);
    printf("\nMatriz B:\n");
    printMat(matB,9);
    //Matrices enviadas a ensamblador a multiplicarse
    mulMat3x3(matA, matB, matR);
    printf("\nMatriz Resultante:\n");
    printMat(matR,9);
    getchar();
void printMat(unsigned char mat[9], int size){
   int i,j;
    for (i = 0; i < size/3; i++){}
    for (j = 0; j < size/3; j++) printf("\t%d", mat[i*3+j]);</pre>
    printf("\n");
```