

TP Organizacion del computador II

Integrantes

- Evelin Aragon - eve_aragon@hotmail.com
- Franco Disabato - francodisabatobook@gmail.com

Profesores

- Carloz Jimenez
- Alessia Katerina Lescano Horbik

Codigo

El codigo se divide en 2 partes.

- Codigo en main.c --> interaccion con el usuario.
- Codigo en operaciones.asm --> calculo de operaciones matematicas.

Durante la compilacion se linkea el codigo de c con el de assembler.

main.c

Declaramos la funcion de asm que llamaremos desde c. La misma recibe 2 numeros y 1 operador. Los operadores pueden ser + - * /

```
extern int recibir_Operacion(int Operando1, char Operador, int Operando2);
```

Luego declaramos 2 funciones que usaremos en el codigo.

LeerPregunta() se le pasan las variables que contendran los input del usuario y devolvera un resultado.

CalcularOperacion() se declara para poder encapsular a la funcion de assembler y llamarla localmente.

```
void LeerPregunta(int *a, char *op, int *b);

int CalcularOperacion(int Operando1, char Operador, int Operando2) {
    return recibir_Operacion(Operando1, Operador, Operando2);
}
```

LeerPregunta() es utilizado para pedir al usuario que ingrese una operacion matematica siguiendo el formato '10 + 10' por ejemplo. Se valida que lo ingresado tenga el formato de %d %c %d , el cual significa digito caracter digito . Si no cumple este formato, se le informa al usaurio con un mensaje de error y se vuelve a pedir que ingrese el input con formato correcto.

Tambien da la opcion de cerrar el programa ingresando la letra q

```
void LeerPregunta(int *a, char *op, int *b) {
    char buffer [50];
    printf("Ingrese una operación aritmética (ej. 10 + 10) o 'q' para salir: ");
    fgets(buffer, sizeof(buffer), stdin);

    if (strcmp(buffer, "q\n") == 0){
        *op = 'q';
        return;
    }
    //parse the input
    if(sscanf(buffer, "%d %c %d", a, op, b) != 3){
        printf("Formato invalido. Por favor intente nuevamente. \n");
        LeerPregunta(a, op, b);
    }
}
```

La funcion principal main() se encarga de declarar los int y char que se usaran para capturar el input del usuario. Luego entra en un loop en el cual le pide al usaurio ingresar datos. Con estos datos se realizan validaciones de casos borde.

- Si el usuario quiere salir del programa se espera la letra q .
- Si el usaurio intenta dividir por cero se devuelve un mensaje de error.
- Si pasa todas las validaciones se calcula la operacion en assembler y se devuelve el resultado por pantalla.

```
int main() {
    int a, b;
    char op;

    while (1) {
        LeerPregunta(&a, &op, &b);

        //chequear si el usuario quiere salir
        if(op == 'q'){
            break;
        }
        if(op == '/' && b == 0){
            printf("Error: no se permite dividir por 0. \n");
            continue;
        }
        int result = CalcularOperacion(a, op, b);
        printf("%d %c %d = %d\n", a, op, b, result);

    }

    return 0;
}
```

operaciones.asm

recibir_Operacion espera recibir 3 parametros. Operando1, operador y operando2. Disponemos de 4 registros de 32-bit para realizar todas las operaciones aritmeticas, que pueden usarse como 4 registros de 32-bit EAX, EBX, ECX, EDX o 4 registros 16-bit AX, BX, CX, DX o 8 registros de 8-bit AH, AL, BH, BL, CH, CL, DH, DL .

Utilizamos los registro EAX y EBX de 32-bit para almacenar los numeros. Y el registro AL de 8-bit para el simbolo del operando.

Se compara el ascii del operando en AL con el ascii de + - * / y si son iguales se salta a la etiqueta correspondiente. En caso de no coincidir con ninguno, se devuelve 0 como resultado

```
recibir_Operacion:
    ; Guardar registros
    push rbp
    mov rbp, rsp
    push rbx

    ; Recibir parámetros
    mov ebx, edi        ; Operando1 en ebx
    mov al, sil         ; Operador en al (sil es el byte menos significativo de esi)
    mov ecx, edx        ; Operando2 en ecx

    ; Comparar el operador y saltar a la operación correspondiente
    ; Se puede usar los numeros de la tabla ascii o el caracter
    cmp al, 43          ; Comparar con el valor ASCII de '+'
    je sum
    cmp al, 45          ; Comparar con el valor ASCII de '-'
    je res
    cmp al, 42          ; Comparar con el valor ASCII de '*'
    je mul
    cmp al, 47          ; Comparar con el valor ASCII de '/'
    je div

    ; Si no es un operador válido, devolver 0 y salir
    mov eax, 0
    jmp fin
```

Se utiliza la instruccion add para sumar los registros ebx y ecx que contienen los valores de operando1 y operando2. El resultado de la suma queda en ebx, luego se pasa el mismo a eax para devolverlo como resultado de la funcion.

```
sum:
    add ebx, ecx        ; Sumar Operando1 y Operando2
    mov eax, ebx        ; Guardar resultado en eax
    jmp fin
```

Se utiliza la instruccion sub para restar al registro ebx con el registro ecx. El resultado de la resta queda en ebx, luego se pasa el mismo a eax para devolverlo como resultado de la funcion.

```
res:
    sub ebx, ecx        ; Restar Operando2 de Operando1
    mov eax, ebx        ; Guardar resultado en eax
```

```
jmp fin
```

Se utiliza la instrucción `imul` para multiplicar al registro `ebx` con el registro `ecx`. El resultado de la multiplicación queda en `ebx`, luego se pasa el mismo a `eax` para devolverlo como resultado de la función.

```
mul:
    imul ebx, ecx      ; Multiplicar Operando1 por Operando2
    mov eax, ebx       ; Guardar resultado en eax
    jmp fin
```

Primero nos aseguramos que el divisor no sea 0 para que no rompa la división. En caso de que sea cero saltamos a la etiqueta `error`.

Para la división se coloca al dividendo en `eax` y se limpia `edx` para el resto. Luego se utiliza la instrucción `div` y se le pasa el divisor. Se divide al registro `ebx` con el registro `ecx`. El resultado de la división queda en `eax`, y se lo devuelve como resultado de la función.

```
div:
    cmp ecx, 0
    je error
    mov eax, ebx        ; Mover Operando1 a eax (dividendo)
    xor edx, edx        ; Limpiar edx para la división
    div ecx             ; Dividir Operando1 entre Operando2
    ; El cociente está ahora en eax, el residuo en edx
    jmp fin
```

Se llama a `error` en caso de que el divisor sea 0 y se devuelve cero por defecto.

```
error:
    mov eax, 0          ; En caso de error, devolver 0
    jmp fin
```

Al final de la operación se llama a `fin` para restaurar los registros y devolver el resultado de la función.

```
fin:
    ; Restaurar registros y retornar
    pop rbx
    mov rsp, rbp
    pop rbp
    ret
```