Arquitectura de computadores

UnalMed 2012-03

Carlos Daniel Sánchez Ramírez
Santiago Pinzón Correa

Explicación del algoritmo usado para la programación de la práctica 1 en ensamblador

Pseudo código

Los requisitos de la práctica eran varios, por lo que hicimos primero, antes de empezar a escribir el programa en ensamblador, hacer el programa en un pseudocódigo ejecutable en Python.

Aquí el código:

```
#Flag para el ciclo infinito del programa.
ciclo = True

#Mensaje de bienvenida.
print "Hola, este es el programa para ensamblasdor, pero escrito en python."
print "Bienvenido/a :D"
print "\n\n"


#inicio del ciclo.
while(ciclo):
    print "Introduzca los valores de las siguientes variables: "

    #Captura de variables desde el teclado.
    A=int(raw_input(" >Ingrese variable A: "))
    B=int(raw_input(" >Ingrese variable B: "))
    C=int(raw_input(" >Ingrese variable C: "))
    D=int(raw_input(" >Ingrese variable D: "))
    E=int(raw_input(" >Ingrese variable E: "))
    F=int(raw_input(" >Ingrese variable F: "))
    G=int(raw_input(" >Ingrese variable G: "))

    #comprovación para evitar divisiones por cero
    while 5*C-2*D == 0 :
        C=int(raw_input("** La variable C genera divisiones por cero.\
        Ingresela nuevamente: "))

    #Realmente, eso jamás será cero con aritmetica entera.
    #while 4*G-3 == 0 :
    #    G=int(raw_input("** La variable G genera divisiones por cero.\
    #    Ingresela nuevamente: "))

    print "\n"

    #Empieza la calculadera ~_~
    print "\t3*A=",3*A
    print "\t3*A+2*B=",3*A+2*B
    print "\t5*C=",5*C
    print "\t5*C-2*D=", 5*C-2*D
    print "\t(3*A+2*B)/(5*C-2*D)=",(3*A+2*B)/(5*C-2*D)

    print "\t2*E*F=",2*E*F
    print "\t4*G=",4*G
    print "\t4*G-3=",4*G-3
    print "\t(2*E*F)/(4*G-3)=",(2*E*F)/(4*G-3)

    print "\t(3*A+2*B)/(5*C-2*D) + (2*E*F)/(4*G-3)",\
            (3*A+2*B)/(5*C-2*D) + (2*E*F)/(4*G-3)
    #Fin de la calculadera *^_^*
```

```
print "\n"
aux= raw_input("desea continuar ? (y/n): ")
if aux == "n":
    print "Gracias por usar el programa ^_^"
    ciclo = False
print "\n"
```

Explicación del algoritmo en Ensamblador MASM

1. Lo primero que hacemos es incluir la biblioteca Irvine32.
2. Después iniciamos la sección `.data`, en donde definimos primero las variables comenzando por los Strings de los mensajes, luego los Strings para mostrar cómo se irá solucionando el problema, y terminando en las variables que guardarán los valores de la función que se solucionará.
3. Empieza la sección `.code`, donde se desarrolla el programa.
4. Borramos la pantalla.
5. Mostramos el mensaje de bienvenida, donde están los nombres de los integrantes, y el semestre actual.
6. Se hace una etiqueta, la cual hará las veces de "`while`", ya que al final, será la etiqueta desde la cual se volverá a ejecutar el programa.
7. Se muestra el mensaje para pedir los datos que se guardarán en las variables, y se empieza con cada una en orden (`A`, luego `B`, hasta `G`).
8. Una vez obtenidos los valores, se procede a revisarlos en busca de divisiones por cero.
   NOTA: Aquí solo se revisan las variables `C` y `D` (de la operación *`5C-2D`*), y no la variable `G` (de la operación `4G-3`), dado que ésta, en la aritmética entera, jamás puede ser cero.
9. En caso de hallarse una división por cero, se pedirá el ingreso de la variable `C`, con el fin de cambiar su valor. Luego se procede a volver a revisar la operación por si de pronto aun persiste el error.
10. Cuando se halla que no hay división por cero, se continúa con el cálculo de la función.
11. Al calcular la función, la haremos paso a paso, comenzando con el lado izquierdo, y siguiendo con el derecho, y mostrando a cada pazo, el resultado intermedio.
12. En el lado izquierdo, comenzamos con la parte de arriba, guardando el valor de la variable `A`, en el registro `EAX`, y luego multiplicándolo por `3` con la instrucción `imul` (multiplicación con signo).
13. Después se guarda la variable `B` en el registro `EBX`, para luego multiplicarlo con sigo con `imul`. Seguidamente se suman con la instrucción `add`, los registros `EAX` y `EBX`.
14. Guardamos en `ECX` el resultado anterior para poder hacer la división más adelante.
15. Guardamos en el registro `EAX` la variable `C` y la multiplicamos por 5 con `imul`.
16. Luego guardamos en `EBX` la variable `D` y la multiplicamos por 2 con `imul`, para después restar son `sub`, el registro `EAX` con `EBX`.
17. Para poder dividir, se necesita tener el divisor en el registro `EAX`, y se le pasa el dividendo a la instrucción `idiv` (división con números con signo).
    Se debe extender el registro `EAX` antes de dividir, por lo que se usa la instrucción `cdq`.
18. Luego salvamos el resultado del lado izquierdo en una variable auxiliar, para empezar a operar el lado derecho.
19. El lado derecho es similar al izquierdo. Guardamos la variable `E` en `EAX`, y luego multiplicamos

por 2 con imul, después guardamos la variable F en EBX, y multiplicamos ambos registros.
20. Salvamos en ECX el resultado, que es la parte de arriba.
21. Guardamos en el Registro EAX la variable G, y la operamos multiplicándola por 4 con imul, para luego restar EAX con 3 usando sub.
22. Finalmente vamos a dividir, de forma igual como se hizo con el lado izquierdo, organizando los registros y extendiéndolos con cdq.
23. Para acabar y dar el resultado final, guardamos en EBX la variable auxiliar (con el resultado del lado izquierdo) para sumarlo con el lado derecho.
24. En cada paso anterior se imprimía en la pantalla la operación y el resultado correspondientes.
25. Luego se inicia el bucle para preguntar si se desea evaluar de nuevo la función, o si se desea terminar con el programa.
26. Se lee el teclado con la función de *Irvine32* ReadChar, y luego se compara con las posibles entradas validas (y ó Y, n ó N). Si no es alguna de las entradas validas, se pregunta de nuevo.
27. Cuando se dice que sí (y) se salta a la etiqueta al inicio del programa para volverlo a ejecutar.
28. Si se dice que no (n) se salta al final del programa donde hay un mensaje de despedida y donde se finaliza el programa.