Instituto Tecnológico de Costa Rica

Sede Central de Cartago

Escuela de Ingeniería en Computación

Proyecto – Simulador CPU HAL 9000

IC-3101 Arquitectura de Computadoras

Estudiante: Deyan Sanabria Fallas #2021046131

Profesor: Esteban Arias Méndez

Fecha de entrega: 1 - 11 - 2021

Segundo Semestre

Abstract: The following document contains information about a CPU simulator for people who doesn't necessarily know about computer, perfectly for those wanting to learn how a CPU works and to program with an "assembly like language" created for the purpose of this little project. Everything about how to execute it and how it works is within this document

Índice

1. Introducción

Como Proyecto del curso de Arquitectura de Computadoras (IC-3101) se plantea la creación de un simulador de CPU, dicho CPU debe tener como mínimo:

- Un Registro Program Counter
- Un Registro Instruction Register
- Dos o más registros de uso general (En este caso se usaron 4. Son: R1, R2, R3 y R4)
- Flags: Overflow, Carry, Sign y Zero
- Mínimo cada registro debe ser de 4Bits (En este caso son de 5)
- Un lenguaje propio con las instrucciones básicas como los siguientes equivalentes en NASM: mov, loop, pop, push, cmp, rotaciones, corrimientos, test, and y saltos (condicionales)

El objetivo de este CPU es enseñar a las personas que quieren saber el funcionamiento interno de un CPU y como este interactúa con diferentes instrucciones, por ello, en este documento se hablara sobre el proyecto desarrollado, como fue su desarrollo y como cualquier persona puede ejecutarlo sin problemas.

Para este proyecto se usó y se necesita para su uso las siguientes herramientas:

- Un sistema operativo Linux, ya sea montado en el hardware como tal o en una máquina virtual (Necesario para su ejecución y desarrollo)
- El programa Visual Studio Code para realizar la codificación del programa en C con ligas de NASM (Necesario para su desarrollo, opcional para su ejecución)
- Instalar el ensamblador de NASM en el sistema operativo Linux (Necesario para su ejecución y desarrollo)
- El compilador de C GCC en el sistema operativo de Linux (Necesario para su ejecución y desarrollo)
- La librería de NCURSES (Necesario para su ejecución y desarrollo).

2. Desarrollo

2.1. Explicación del código

El desarrollo del proyecto se dividió en tres partes principales:

- **NCurses:** Como requisito de la elaboración de este proyecto, se solicitó utilizar la librería NCurses para la interfaz en consola del proyecto, dicha librería tenía una curva de aprendizaje al ser nueva para el autor, por ende, se recurrió a investigar en internet como utilizar NCurses. Se llego a la siguiente página [1], la cual fue la única consultada por la falta de otros sitios donde fuese más claro la explicación
- **Lectura de Archivo:** Se solicita que el CPU simulado pueda ejecutar archivos de código en un lenguaje tipo ensamblador, pero en español, por ende, una gran parte del trabajo iba ser en la lectura de estos archivos y su ejecución, por ello, se optó por conseguir leer y ejecutar un archivo antes de proceder a imprimir el CPU en la interfaz. La lectura de dichos archivos se aprendió de [2]
- **NASM y la Ejecución de los archivos:** Para ejecutar los archivos, C se encarga de identificar la instrucción a ejecutar y obtener sus operadores, posteriormente estos datos los recibe funciones hechas en NASM, que se encargar de simular estas operaciones de la mejor forma posible.

Durante la elaboración del proyecto, se utilizó un ejemplo de código del libro [3], el cual se encuentra en la página 429-430, el programa en cuestión es el “hll_minmax” que se separa en “hll_minmaxc.c” y “hll_minmaxa.asm”. dicho programa explica cómo se pueden hacer funciones externas de NASM para ejecutarlas en C y como a su vez se pueden pasar direcciones de memoria para poder manipular las variables dentro de NASM por si se necesita retornar más de un elemento.

El código del programa está separado en una cantidad inmensa de funciones, hay 17 funciones solo en NASM, así que para empezar se hablara de dichas funciones de NASM. Dentro del archivo “operaciones.asm” se pueden encontrar 15 operaciones diferentes destinadas a simular el funcionamiento del CPU creado, estas funciones empiezan con la palabra “nasm” seguidas de la instrucción que simular, por ejemplo “nasmSumar” o “nasmComparar”, cada función hace lo que su equivalente de esa instrucción de NASM debería de hacer.

Como ejemplo la función “nasmSumar”(referirse a la seccion 6.1 línea 29), dicha función, recibe dos operadores que son puntero tipo int, y 4 flags, las cuales son overflow, carry, sign y zero, cada una siendo un char debido a que solo almacenaran valores booleanos.

Dentro de esta función, se hace una suma sobre ambos operadores, siendo el primero donde se guardará la suma y se actualizan las flags dependiendo del resultado de dicha suma. A su vez se simula lo que pasaría si ocurre overflow, que la cantidad obtenida probablemente sea el opuesto negativo o positivo más cercano.

El resto de las funciones tiene un comportamiento similar, siempre intentando esa simulación de que pasaría si se pasa de la cantidad de bits y actualizando flags si estas lo hacen en su equivalente NASM.

Las otras últimas dos funciones de las 17 se llaman hex5Bits y decToBin. hex5bits se encarga de cortar los bits después del 5to del valor binario de una variable y lo almacena en otra variable, esto para usar la funcionalidad de la función “printw()” de ncurses que transforma lo que contiene una variable a una representación hexadecimal, pero si la variable tipo int se deja como esta, esta puede tener “1s” extra porque guarda 4 bytes (32bits) y un numero negativo cubre todos los bits después del quinto con “0s”, por ende, para no imprimir en pantalla algo como “0xffffff12”, se cortan los bits hasta el 5to y queda como “0x12”

La función decToBin recibe un puntero a int y un arreglo de char (la función asume que el char tiene como mínimo 6 de espacio), donde con corrimientos y test, se guarda en el arreglo de char, el valor binario de la variable introducida para poder imprimirla en pantalla.

Funciones importantes en C se encuentra la función “guardarEnMemoria()” (referirse a la sección 6.2 línea 314) que lee el archivo 1 veces completo para almacenar variables y las instrucciones en memoria. Esta está compuesta por múltiples funciones, como una llamada “comparar()” que sirve para comparar dos arreglos de char.

Por último, la función “ejecucion()” (sección 6.3 línea 193), que se encarga de volver a leer el archivo e interpretar cada instrucción y ejecutarla, dentro de esta función se usa otra muy importante llamada “correrInstruccion()” (sección 6.2 línea 507) la cual retorna un char (1 o 0 para simular un booleano) que indica si encontró un error al intentar ejecutar una instrucción, y esta se

encarga de toda la ejecución de instrucciones, de forma que identifica la instrucción, sus operadores y luego las pasa a NASM para procesarlas.

2.2. Especificaciones y limitaciones del CPU

Este CPU es muy limitado, cuenta con registros de 5bits y por ende todo se maneja con 5bits, su memoria principal se compone por 16 celdas de memoria, cada una almacena 5bits también, aunque en la realidad, los procesadores almacenan en la memoria principal, un byte el cual equivale a 8 bits, por ello, la unidad de memoria mínima a la que se puede acceder como programador es un byte, no más, no menos. Esto es fácilmente deducible con la explicación del libro [3], donde se habla como los registros que son las unidades de almacenamiento más flexibles a las que se puede acceder desde ensamblador como programador, y el registro más pequeño accesible es de 8 bits.

Una vez explicado eso, la memoria al almacenar muy pocos datos solo se pueden hacer programas relativamente pequeños, donde no excedan 16 instrucciones y variables. También, solo se pueden almacenar 8 variables y etiquetas (para saltos), máximo y de 10 caracteres cada una máximo.

Esta limitante además de ser un impedimento artificial simula como las computadoras tienen recursos limitados, pero aun con esos recursos limitados, se pueden hacer cosas increíbles y también puede servir como forma de promover código más eficiente, porque, aunque tengamos maquinas cada vez más rápidas, no significa que hay que dejar la eficiencia de lado, además que un programador se diferencia por como crea su código y que tan bueno es.

Por último, los strings para comunicación con el usuario máximo de 49 caracteres. También el CPU solo puede acceder a 256 caracteres por línea en el archivo del programa.

Los registros PC e IR no muestran necesariamente la dirección de memoria donde se encuentra la siguiente instrucción o la actual, si no la línea del archivo que se está leyendo, esto incluye cuando hace saltos.

2.3. Instrucciones del lenguaje propio del CPU

El CPU cuenta con las siguientes 18 instrucciones, con su valor binario y hexadecimal:

Instrucción	Código Binario	Código Hexadecimal
sumar *operador,operador*	00000	0x00
mover *operador,operador*	00001	0x01
ciclar *etiqueta:*	00010	0x02
meter *operador*	00011	0x03
sacar *operador*	00100	0x04
comparar *operador,operador*	00101	0x05
rotarI *operador*	00110	0x06
rotarD *operador*	00111	0x07
correrI *operador*	01000	0x08
correrD *operador*	01001	0x09
test *operador,operador*	01010	0x0A
and *operador,operador*	01011	0x0B
saltar *etiqueta:*	01100	0x0C
saltarI *etiqueta:*	01101	0x0D
saltarNI *etiqueta:*	01110	0x0E
DSF *operador*	01111	0x0F
entrada *string*	10000	0x10
salida *operador*	10001	0x11

Para empezar a programar en este procesador, por favor ingresar al archivo llamado “progcustom.asf”, que es el archivo el cual el programa usa para programas creados por el usuario. Una vez en ese archivo, para almacenar datos en memoria se empieza colocando “::Datos”, para indicar al procesador que lo que sigue es una sección de apartado de datos. Este procesador solo guarda datos numéricos y variables individuales, por ende, después de la palabra clave “::Datos”, debajo de ella podrá colocar el nombre de una variable junto al valor que se le quiera asignar

Ejemplo:

```
::Datos
X 10
Y 20
```

De esta forma se crearon dos variables, una llamada “X” con valor 10 y otra llamada “Y” con valor 20. Estas pueden ser accedidas con dichos nombres en todo el programa.

Para empezar la sección de código, esta debe estar en medio de dos palabras clave, las cuales son “-INICIO” y “-FIN”, ejemplo:

```
-INICIO
     *Sección de código*
-FIN
```

Las instrucciones que van en medio de esas dos palabras clave son las que serán ejecutadas por el programa. Por ultimo las etiquetas, cualquier palabra de máximo 10 caracteres con un “:” al final (los “:” cuentan como carácter dentro de los 10) será considerada una etiqueta, y para ser referenciada en una instrucción de salto debe ser puesta junto a los “:” ejemplo:

```
-INICIO
     sumar R3,R2
etiqueta:
     mover R4,R3
     saltar etiqueta:
-FIN
```

El ejemplo anterior crea un ciclo infinito, pero es solo para ejemplo, de esa forma se usan las etiquetas. Cabe destacar la funcionalidad de la instrucción DSF, esta instrucción especial fue solicitada como requisito para el proyecto. Su función es recibir un número y elevarlo al cuadrado con sumas consecutivas.

Por último, las instrucciones como CorrerD y CorrerI, la D significa Derecha y la I significa Izquierda, para realizar los corrimientos de los bits, lo mismo para con las rotaciones. En los Saltos, saltarI significa “saltar si es igual” y saltarNI significa “saltar si NO es igual” y estas revisan el estado de la ZeroFlag para realizar los saltos.

2.4. Lógica del programa

La lógica detrás del programa es un concepto muy simple, la idea principal siempre fue primero lograr leer el archivo y ejecutar las instrucciones, después acomodar el código para poder pausarlo cuando se necesite con una instrucción como el “usleep()” ubicada dentro de las librerías de C, “namps()” ubicada dentro de la librería de NCurses o getch() también ubicada en la librería de NCurses.

Después crear con NCurses una sección donde se vieran todos los datos solicitados, los cuales son registros, flas, memoria principal, instrucción a ejecutar, el ciclo de fetch y un lugar para la interacción del programa ejecutado por el CPU al usuario.

La sección donde se viera el CPU fuera acompañada con un menú, donde se seleccione 3 de los programas base de ejemplo dentro del paquete del simulador, junto a una opción extra para un programa creado por el usuario que quiera verlo, y por último una forma de seleccionar si se quiere que se ejecute de forma automática o con pausas donde puede avisar al programa que pase a la siguiente instrucción presionando cualquier tecla.

Esa sería la forma abstracta y general del pensamiento detrás del programa, si se adentra un poco más se puede hablar sobre la lógica detrás de lectura y ejecución del archivo. En este se planteó dos fases, una donde se “cargara el programa en memoria” y otra donde se ejecutarán las instrucciones como tal.

La parte de cargar el programa en memoria no es más que relacionar una variable con el espacio en memoria al que corresponde y después leer todas las instrucciones y colocarlas en memoria de forma que se vieran después como un código hexadecimal en la simulación.

La parte de ejecutar el programa, una vez los datos necesarios se cargan en memoria, se pueden ejecutar las instrucciones, esto se hace devolviendo el archivo que se leyó al inicio con la función “rewind()” [4], y posteriormente ir instrucción por instrucción interpretándola y obteniendo sus operadores para después pasarlo a funciones de NASM que se encarguen de procesarlas.

Eso fue básicamente la lógica de cómo funciona el programa y como se fue desarrollando.

2.5. ¿Cómo ejecutar el programa?

Abrir el programa se puede hacer siguiendo los siguientes pasos:

1. Abrir la terminal de Linux
2. Con el comando “cd” dirigirse a la ubicación/directorio del programa
3. Una vez la terminal se ubique en el directorio del programa, ejecutar el siguiente comando: “nasm -felf operaciones.asm”
4. Posterior al paso anterior ejecutar el siguiente comando: “gcc -m32 -fno-stack-protector main.c -lncurses -o main operaciones.o”
5. Como paso final, puede ejecutar el programa colocando “./main” en la terminal estando en el directorio del programa, una vez se hacen los 4 pasos anteriores, no hace falta repetirlos para próximas ejecuciones, solo hacer el paso 5.

3. Análisis de Resultados

3.1. Problemas encontrados

- Entre los problemas encontrados se puede destacar la no utilización de algún tipo de dato string, esto complico el proceso de lectura del archivo, teniendo que recurrir a funciones que identificaran partes de un string de la forma que se buscaba, que hubiera sido más sencillo con una estructura de datos tipo String.
- Un problema bastante crucial es que no hay demasiada información muy bien documentada sobre la librería NCURSES a excepción de la página encontrada, por ende, se improvisó un poco a la hora de hacer la interfaz, tomando como base el cómo se hacen las interfaces con en consola con prints y scroll de por medio, usando la función "clear()" se puede limpiar la pantalla anterior y volver a imprimir todo, de esa forma se realizó la interfaz.
- El uso de múltiples archivos es un problema, debido a que hay que tener cuidado donde se hacen los "#include" y que no se repitan haciendo múltiples, sin lugar a duda, algo que se puede mejorar en el futuro es el manejo de dichos archivos.

3.2. Algoritmos usados

Dentro de los algoritmos usados, no hay nada demasiado interesante, a excepción de partes de algunas funciones de NASM, por ejemplo la función "decToBin()" (referirse a la seccion 6.1 línea 671) aplicar una máscara con la instrucción TEST de forma que solo el 5to bit de un valor determinado, quede en su valor original, en caso de que este sea 1, se guarda en un arreglo, el carácter '1' y se incrementa la posición del arreglo, si es 0 se repite lo mismo guardando el carácter '0', posteriormente se hace un corrimiento a la izquierda y se repite 5 veces, de esa forma se obtiene el valor binario contenido dentro de una variable.

Otra parte de algoritmo interesante seria que en varias funciones se utilizó la misma instrucción de test para verificar cual es el valor del 5to bit (referirse a la seccion 6.1 línea 399), si este es 1, se fuerzan los bits después del 5to bit a 1 con la instrucción OR, si el 5to bit es 0, se fuerzan los valores a 0 con la instrucción AND. De esta forma, el simulador usa 5 bits, pero el CPU real mantiene constancia con los valores que hay, por ejemplo, no es lo mismo 0000 0000 0000 0000 0000 0000 0001 1011 que 11011 debido al complemento de 2, por ende, cuando se manipula los registros del simulador, hay que también hacer que el resto de los bits concuerden para que el CPU real los identifique como realmente se quiere.

3.3. Muestras de ejecución

A la hora de ejecutar el programa se puede visualizar un menú principal donde se pueden ejecutar 4 programas, tres de ellos son de ejemplo solicitados para la realización del proyecto y las dos siguientes para cambiar el modo en el que pasan las instrucciones.

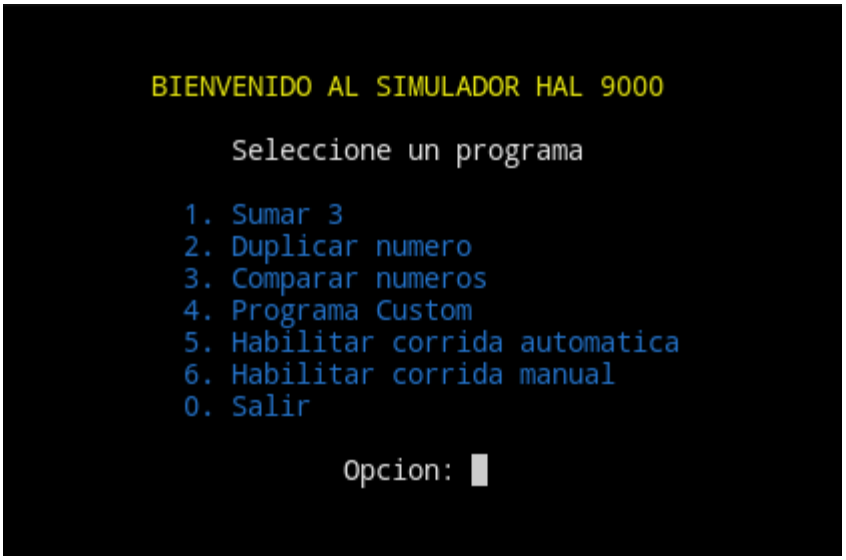


Fig. 1. Menu principal del programa creado

Cuando se ejecuta una de las 4 primeras opciones se muestra lo siguiente:

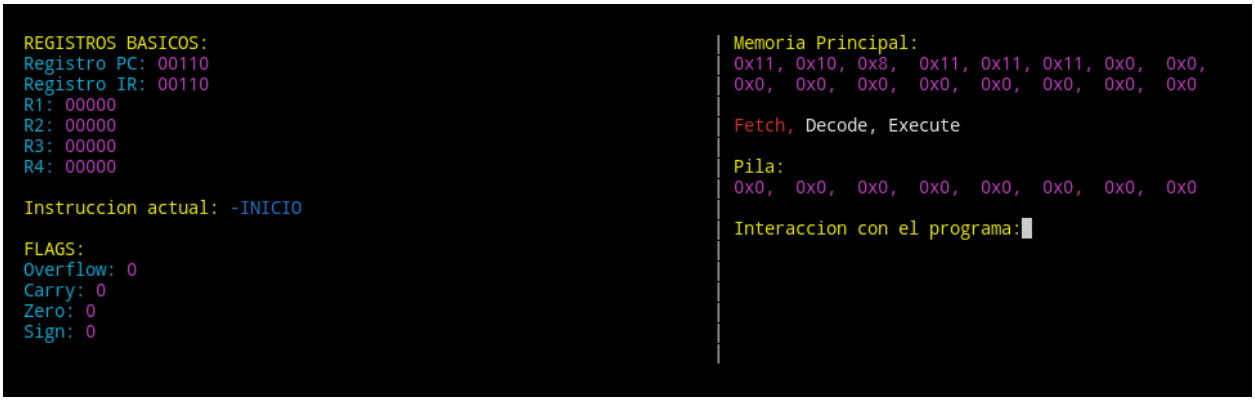


Fig. 2. Programa en ejecución

En este caso se muestras todo lo del CPU y se puede ver la instrucción actual, registros, flags, memoria principal, la pila y la interacción con el usuario. Cuando se interactúa con el usuario se puede visualizar los siguiente:

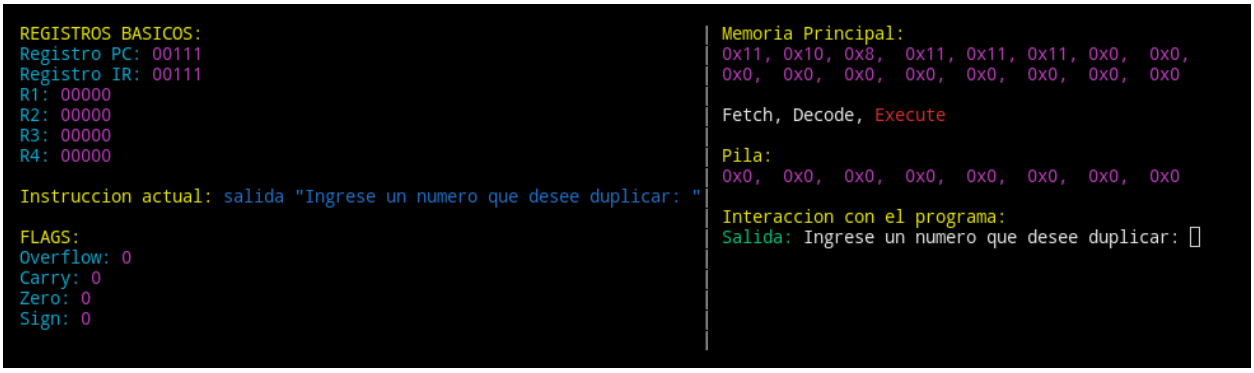


Fig. 3. Interacción con el usuario

En el apartado de interacción se indica al usuario que lo que está viendo es una salida, en este caso no se recolectan datos, solo se visualizan, independiente de si está en modo manual (presionar una tecla para ir a la siguiente instrucción) o modo automático (hace pausas de unos segundos entre instrucciones), cuando se presenta una Salida, hay que presionar alguna tecla para avisar al simulador que se pudo leer correctamente lo que hay en pantalla y en caso del modo manual, seria volver a presionar una tecla para avisar que pase a la siguiente instrucción.

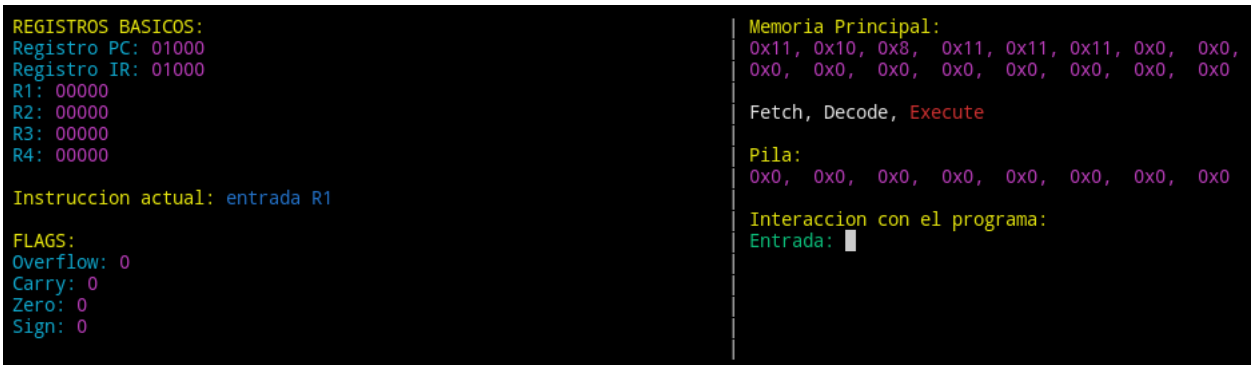


Fig. 4. Lo que sucede cuando se piden datos

En la imagen anterior se puede ver que la instrucción actual está pidiendo datos, en la sección de interacción con el usuario, se muestra una leyenda que dice “Entrada: “, eso indica al usuario que el programa está preparado para recibir una entrada, cuando el usuario quiera, puede introducir una entrada y posteriormente presionar una tecla para avisar al simulador que pase a la siguiente instrucción. En todo momento el usuario puede visualizar cual es el estado de las flags, registros, memoria y demás.

4. Conclusiones y observaciones

En la elaboración del proyecto presentado, se aprendieron muchas cosas a la hora de realizar código, así como el ver lo increíble que es el funcionamiento interno de un procesador, realmente es impresionante lo que se puede hacer con cosas tan simples y un poco de matemáticas.

El programa resultante cumple con las expectativas de ayudar a las personas a comprender un poco mas el mundo de la computación, siendo o no conocedor de esta área.

La librería de NCURSES parece ser una muy buena opción para interfaces por consola, aunque sea algo vieja, cumple realmente con todo lo necesario para hacer una interfaz bonita y con funcionamiento correcto, sin embargo, debido a su vejez, no existe muchos lugares para aprender dicha librería para gente novata que le gustaría aprender algo nuevo.

Sin mucho más por agregar, he de destacar que el lenguaje C es un lenguaje increíblemente flexible y altamente capaz, no por nada es un lenguaje del cual se basan muchos otros, tanto que se puede juntar con lenguaje ensamblador y hacer programas muy eficientes, sin lugar un lenguaje que debe ser necesario aprender para ser profesionales en el desarrollo de software.

5. Bibliografía

[1] P. Padala, «NCURSES Programming HOWTO,» 20 Junio 2005. [En línea]. Available: https://tldp.org/HOWTO/NCURSES-Programming-HOWTO/.

[2] «C Read Text File,» [En línea]. Available: https://www.learnc.net/c-tutorial/c-read-text-file/.

[3] S. P. Dandamudi, Guide to Assembly Language Programming in Linux, New York: Springer, 1989.

[4] «C library function - rewind(),» [En línea]. Available: https://www.tutorialspoint.com/c_standard_library/c_function_rewind.htm.

6. Apéndices

6.1. operaciones.asm

```
1   ; ------------------------------------------------------------------
2   ; Programa creado por Deyan Sanabaria Fallas con el objetivo de
3   ; usar funciones creadas en NASM dentro de C, habran funciones para
4   ; procesar instrucciones del lenguaje de programacion hecho
5   ; para la simulacion de un procesador
6   ; ------------------------------------------------------------------
7   segment .text
8
9   global nasmSumar
10  global nasmMover
11  global nasmCiclar
12  global nasmSacar
13  global nasmMeter
14  global nasmComparar
15  global nasmSaltar
16  global nasmSaltarI
17  global nasmSaltarNI
18  global nasmRotarI
19  global nasmRotarD
20  global nasmCorrerI
21  global nasmCorrerD
22  global nasmAND
23  global nasmDSF
24  global hex5Bits
25  global decToBin
26
27  ; Funcion para hacer el procedimiento de suma del
28  ; lenguaje del simulador de procesador
29  nasmSumar:
30      enter 0,0
31      push EBX ; Guardar registros importantes
32      push EAX
33      push EDX
34
35      ; Procedimiento de suma
36      mov EAX,[EBP+8]     ; EAX = &Operador1
37      mov EBX,[EBP+12]    ; EBX = &Operador2
38      mov EDX,[EBX]
39      add [EAX],EDX
40
41      ; Se mueve a EBX el valor de lo que hay en EAX
42      ; para su uso eficiente
43      mov EBX,[EAX]
44
45      ; Comparaciones, detectan si el numero esta entre
46      ; -16 y 15, esto para el overflow y las flags
47  sumComparacion1:
48      cmp EBX,-16
49      jge sumComparacion2     ; Primero se verifica que sea
50      jmp SHORT sumNoEnRango  ; mayor o igual a -16
51
52  sumComparacion2:
53      cmp EBX,15              ; Luego menor o igual a 15
54      jle sumEstaEnRango      ; en ambos si no se cumple salta
55      jmp SHORT sumNoEnRango  ; a "NoEnRango" pero si cumple
56                              ; a "EstaEnRango"
57
58      ; Ingreso de valores de flags:
59      ; Overflow = 0, Carry = 0
60  sumEstaEnRango:
61      mov EBX,[EBP+16]    ; EBX = &Overflow
62      mov byte [EBX],0    ; Overflow = 0
63      mov EBX,[EBP+20]    ; EBX = &Carry
64      mov byte [EBX],0    ; Carry = 0
65      jmp SHORT sumZeroSign
```

```
66
67          ; Overflow = 1, Carry = 1
68      sumNoEnRango:
69          mov EBX,[EBP+16]    ; EBX = &Overflow
70          mov byte [EBX],1    ; Overflow = 0
71          mov EBX,[EBP+20]    ; EBX = &Carry
72          mov byte [EBX],1    ; Carry = 0
73
74          ; Se le aplica una mascara para simular lo que
75          ; pasaria si se llega al limite de bits
76          mov EBX,[EAX]
77          xor EBX,0xFFFFFFE0
78          mov [EAX],EBX
79
80          ; ZeroFlag y SignFlag
81      sumZeroSign:
82          mov EBX,[EAX]   ; EBX = Operador1
83          cmp EBX,0       ; Unica comparacion para saber si es
84          je sumEsZero    ; 0, si es mayor y si es menor
85          jmp SHORT sumNoZero
86
87          ; ZeroFlag = 1
88      sumEsZero:
89          mov EBX,[EBP+28]
90          mov byte [EBX],1
91          jmp SHORT sumEsPositivo
92
93          ; ZeroFlag = 0
94      sumNoZero:
95          mov EBX,[EBP+28]
96          mov byte [EBX],0
97          jg sumEsPositivo
98          jl sumEsNegativo
99
100         ; SignFlag = 0
101     sumEsPositivo:
102         mov EBX,[EBP+24]
103         mov byte [EBX],0
104         jmp SHORT sumTerminar
105
106         ; SignFlag = 1
107     sumEsNegativo:
108         mov EBX,[EBP+24]
109         mov byte [EBX],1
110
111     sumTerminar:
112         pop EDX ; Restauracion de registros
113         pop EAX
114         pop EBX
115         leave
116         ret
117
118     ; Funcion para hacer la operacion de mover valores de un lugar del CPU
119     ; al otro del lenguaje creado para la simulacion de CPU
120     nasmMover:
121         enter 0,0
122         push EAX    ; Guardar valores
123         push EBX
124         push EDX
125
126         mov EAX,[EBP+8]     ; EAX = &Operador1
127         mov EBX,[EBP+12]    ; EBX = &Operador2
128         mov EDX,[EBX]
129
130         ; Verificacion de que el dato a mover este en el rango deseado
131     moverComparacion1:
132         cmp EDX,-16
133         jge moverComparacion2
134         jmp SHORT moverFueraRango
135
136     moverComparacion2:
```

```
137         cmp EDX,15
138         jle moverMovimiento
139         jmp SHORT moverFueraRango
140
141     moverFueraRango:
142         ; Se le aplica una mascara para simular lo que
143         ; pasaria si se llega al limite de bits
144         xor EDX,0xFFFFFFE0
145
146     moverMovimiento:
147         mov [EAX],EDX       ; Operador1 = Operador2
148
149     moverFin:
150         pop EDX     ; Restaurar valores
151         pop EBX
152         pop EAX
153         leave
154         ret
155
156
157     ; Funcion que opera los datos del simulador de CPU
158     ; para lograr Hacer ciclos
159     nasmCiclar:
160         enter 0,0
161         push EAX    ; Guardar valores
162         push EBX
163         push EDX
164
165         mov EAX,[EBP+8]     ; EAX = &operador1
166         mov EBX,[EBP+12]    ; EBX = &Registro3
167         mov EDX,[EBX]       ; EDX = Registro3
168
169         cmp EDX,1
170         jle cicloFin
171
172         dec dword [EBX]
173         mov EBX,[EBP+16]    ; EBX = &pCounter
174         mov EDX,[EAX]       ; EDX = operador1
175         dec EDX
176         mov [EBX],EDX       ; pCounter = operador1
177
178         mov EBX,[EBP+20]
179         mov byte [EBX],1
180
181     cicloFin:
182         pop EDX     ; Restaurar valores
183         pop EBX
184         pop EAX
185         leave
186         ret
187
188     ; Funcion destinada a simular una pila para el CPU simulado
189     ; su funcion es hacer push dentro de la pila
190     nasmMeter:
191         enter 0,0
192         push EAX    ; Guardar valores
193         push EBX
194         push EDX
195         push ESI
196
197         mov EAX,[EBP+8]     ; EAX = &pila
198         mov EBX,[EBP+16]    ; EBX = &posPila
199         mov ESI,[EBX]       ; ESI = posPila
200         inc ESI
201         cmp ESI,8
202         jge meterFin
203
204         mov EBX,[EBP+12]    ; EBX = &operador1
205         mov EDX,[EBX]       ; EDX = operador1
206         mov [EAX+ESI*4],EDX ; pila[ESI] = operador1
207
```

```
208         mov EBX,[EBP+16]    ; EBX = &posPila
209         mov [EBX],ESI       ; posPila++;
210
211     meterFin:
212         pop ESI
213         pop EDX     ; Restaurar valores
214         pop EBX
215         pop EAX
216         leave
217         ret
218
219     ; Funcion para simular el funcionamiento de una pila
220     ; saca un elemento de la pila
221     nasmSacar:
222         enter 0,0
223         push EAX    ; Guardar valores
224         push EBX
225         push EDX
226         push ESI
227
228         mov EAX,[EBP+8]     ; EAX = &pila
229         mov EBX,[EBP+16]    ; EBX = &posPila
230         mov ESI,[EBX]       ; ESI = posPila
231         cmp ESI,0
232         jl sacarFin
233         mov EDX,[EAX+ESI*4]     ; EDX = pila[posPila]
234         mov [EAX+ESI*4],dword 0 ; pila[posPila] = 0
235         mov EAX,[EBP+12]        ; EAX = &operador1
236         mov [EAX],EDX           ; operador1 = pila[posPila]
237         dec dword [EBX]
238
239     sacarFin:
240         pop ESI
241         pop EDX     ; Restaurar valores
242         pop EBX
243         pop EAX
244         leave
245         ret
246
247     ; Funcion para comparar dos elementos en el lenguaje del
248     ; simulador de CPU
249     nasmComparar:
250         enter 0,0
251         push EAX    ; Guardar valores
252         push EBX
253         push EDX
254
255         mov EAX,[EBP+8]     ; EAX = &operador1
256         mov EBX,[EBP+12]    ; EBX = &operador2
257         mov EDX,[EBX]       ; EDX = operador2
258
259         cmp [EAX],EDX
260         jne comNoZero
261
262     comEsZero:
263         mov EBX,[EBP+16]
264         mov byte [EBX],1
265         jmp SHORT comPositivo
266
267     comNoZero:
268         mov EBX,[EBP+16]
269         mov byte [EBX],0
270
271     comSignFlag:
272         jl comNegativo
273
274     comPositivo:
275         mov EBX,[EBP+20]
276         mov byte [EBX],0
277         jmp SHORT comFin
278
```

```
279   comNegativo:
280       mov EBX,[EBP+20]
281       mov byte [EBX],1
282
283   comFin:
284       pop EDX     ; Restaurar valores
285       pop EBX
286       pop EAX
287       leave
288       ret
289
290   ; Funcion para simular un salto en el simulador de CPU
291   nasmSaltar:
292       enter 0,0
293       push EAX    ; Guardar valores
294       push EBX
295       push EDX
296
297       mov EAX,[EBP+8]     ; EAX = &operador1
298       mov EBX,[EBP+12]    ; EBX = &pCounter
299       mov EDX,[EAX]
300       mov [EBX],EDX       ; pCounter = operador1
301
302       pop EDX     ; Restaurar valores
303       pop EBX
304       pop EAX
305       leave
306       ret
307
308   ; Funcion para simular un salto condicional
309   ; en el simulador de CPU, en este caso, salta
310   ; si la comparacion anterior es igual
311   ; (saltar si es igual)
312   nasmSaltarI:
313       enter 0,0
314       push EAX    ; Guardar valores
315       push EBX
316       push EDX
317
318       mov EAX,[EBP+16]    ; EAX = &zeroFlag
319       mov BL,[EAX]       ; EBX = zeroFlag
320       cmp BL,0
321       je sIFin
322
323       mov EAX,[EBP+8]     ; EAX = &operador1
324       mov EBX,[EBP+12]    ; EBX = &pCounter
325       mov EDX,[EAX]
326       mov [EBX],EDX       ; pCounter = operador1
327
328       mov EAX,[EBP+20]    ; EAX = &reiniciar
329       mov byte [EAX],1         ; reiniciar = 1
330
331   sIFin:
332       pop EDX     ; Restaurar valores
333       pop EBX
334       pop EAX
335       leave
336       ret
337
338   ; Funcion para simular un salto condicional
339   ; en el simulador de CPU, en este caso, salta
340   ; si la comparacion anterior NO es igual
341   ; (saltar si NO es igual)
342   nasmSaltarNI:
343       enter 0,0
344       push EAX    ; Guardar valores
345       push EBX
346       push EDX
347
348       mov EAX,[EBP+16]    ; EAX = &zeroFlag
349       mov BL,[EAX]        ; EBX = zeroFlag
```

```
350         cmp BL,1
351         je sNIFin
352
353         mov EAX,[EBP+8]     ; EAX = &operador1
354         mov EBX,[EBP+12]    ; EBX = &pCounter
355         mov EDX,[EAX]
356         mov [EBX],EDX       ; pCounter = operador1
357
358         mov EAX,[EBP+20]    ; EAX = &reiniciar
359         mov byte [EAX],1         ; reiniciar = 1
360
361     sNIFin:
362         pop EDX     ; Restaurar valores
363         pop EBX
364         pop EAX
365         leave
366         ret
367
368
369     ; Funcion para simular una rotacion izquierda
370     ; para la simulacion del procesador de 5 bits
371     nasmRotarI:
372         enter 0,0
373         push EAX    ; Guardar valores
374         push EBX
375         push EDX
376
377         mov EAX,[EBP+8]     ; EAX = &operador1
378         mov EBX,[EBP+12]    ; EBX = &carryFlag
379         mov EDX,[EAX]       ; EDX = operador1
380
381         shl EDX,1
382
383         test EDX,0x20       ; se revisa el 6to bit si es 1 o 0
384         jz rIColocar0
385
386     rIColocar1:             ; si es 1 el 6to bit se cambia el primer bit
387         xor EDX,0x1         ; que quedo como 0 a 1 (simular rotacion)
388         mov byte [EBX],1    ; y en la carry flag se pone 1
389         jmp SHORT rIRestoBits
390
391     rIColocar0:
392         mov byte [EBX],0    ; si es 0 el 6to bit, como es un shift izquierdo
393                             ; el primer bit ya es 0
394
395        ; En esta seccion se compensa porque el CPU real maneja
396        ; 32bits por ende, si el 5to bit es 1, el resto de bits
397        ; del procesador real tienen que ser 1 para que de el
398        ; resultado correcto a la hora de usar el numero
399     rIRestoBits:
400         test EDX,0x10   ; Se revisa el 5to bit para hacer que el
401         jz rIResto0     ; resto de bits despues de el sean
402                         ; 0 o 1 con una mascara
403
404     rIResto1:
405         or EDX,0xFFFFFFE0   ; Fuerza 1 en los bits despues del 5to
406         jmp SHORT rIFin
407
408     rIResto0:
409        and EDX,0x1F         ; Fuerza 0 en los bits despues del 5to
410
411     rIFin:
412         mov [EAX],EDX
413         pop EDX     ; Restaurar valores
414         pop EBX
415         pop EAX
416         leave
417         ret
418
419
420     ; Funcion para simular una rotacion derecha
```

```
421   ; para la simulacion del procesador de 5 bits
422   nasmRotarD:
423       enter 0,0
424       push EAX    ; Guardar valores
425       push EBX
426       push EDX
427
428       mov EAX,[EBP+8]     ; EAX = &operador1
429       mov EDX,[EAX]       ; EDX = operador1
430       mov EBX,EDX         ; EBX = operador1
431
432       shr EDX,1
433       test EBX,0x1
434       jz rDColocar0
435
436   rDColocar1:
437       or EDX,0xFFFFFFF0   ; Fuerza 1 en los bits despues del 4to
438       mov EBX,[EBP+12]    ; EBX = &carryFlag
439       mov byte [EBX],1
440       jmp rDFin
441
442   rDColocar0:
443       and EDX,0xF         ; Fuerza 0 en los bits despues del 4to
444       mov EBX,[EBP+12]    ; EBX = &carryFlag
445       mov byte [EBX],0
446
447   rDFin:
448       mov [EAX],EDX
449       pop EDX     ; Restaurar valores
450       pop EBX
451       pop EAX
452       leave
453       ret
454
455   ; Funcion para simular un corrimiento a la
456   ; izquierda para un procesador de 5bits
457   nasmCorrerI:
458       enter 0,0
459       push EAX    ; Guardar valores
460       push EBX
461       push EDX
462
463       mov EAX,[EBP+8]     ; EAX = &operador1
464       mov EDX,[EAX]       ; EDX = operador1
465       mov EBX,[EBP+12]    ; EBX = &carryFlag
466
467       test EDX,0x10       ; Revisa el valor para colocar
468       jz cICarry0         ; en carryFlag
469
470   cICarry1:
471       mov byte [EBX],1        ; carryFlag = 1
472       jmp SHORT cICorrimiento
473
474   cICarry0:
475       mov byte [EBX],0        ; carryFlag = 0
476
477   cICorrimiento:
478       shl EDX,1
479       test EDX,0x10           ; ocurre el corrimiento y para
480       jz cIColocar0           ; mantener constancia se fuerza el
481                               ; resto de bits despues del 5to
482                               ; a tener el mismo valor que el 5to
483
484   cIColocar1:
485       or EDX,0xFFFFFFE0   ; Fuerza 1 en los bits despues del 5to
486       jmp SHORT cIFin
487
488   cIColocar0:
489       and EDX,0x1F         ; Fuerza 0 en los bits despues del 4to
490
491   cIFin:
```

```
492         mov [EAX],EDX
493         pop EDX     ; Restaurar valores
494         pop EBX
495         pop EAX
496         leave
497         ret
498
499     ; Funcion para simular un corrimiento a la
500     ; derecha para un procesador de 5bits
501     nasmCorrerD:
502         enter 0,0
503         push EAX    ; Guardar valores
504         push EBX
505         push EDX
506
507         mov EAX,[EBP+8]     ; EAX = &operador1
508         mov EDX,[EAX]       ; EDX = operador1
509         mov EBX,[EBP+12]    ; EBX = &carryFlag
510
511         test EDX,0x1
512         jz cRCarry0
513
514     cRCarry1:
515         mov byte [EBX],1    ; carryFlag = 1
516         jmp SHORT cRCorrimiento
517
518     cRCarry0:
519         mov byte [EBX],0    ; carryFlag = 0
520
521     cRCorrimiento:
522         shr EDX,1
523         and EDX,0xF ; Forzar a 0 despues del 4to bit
524
525     cRFin:
526         mov [EAX],EDX
527         pop EDX     ; Restaurar valores
528         pop EBX
529         pop EAX
530         leave
531         ret
532
533     ; Funcion para simular la instruccion
534     ; AND y TEST para procesador de 5 bits
535     nasmAND:
536         enter 0,0
537         push EAX    ; Guardar valores
538         push EBX
539         push EDX
540
541         mov EAX,[EBP+8]     ; EAX = &operador1
542         mov EBX,[EBP+12]    ; EBX = &operador2
543         mov EDX,[EBX]       ; EDX = operador2 (mask)
544         mov EBX,EDX         ; EBX = operador2 (mask)
545         mov EDX,[EAX]       ; EDX = operador1 (aplicar la mask)
546         and EDX,EBX         ; aplicacion de mask
547
548         ; Set de flag:
549         cmp EDX,0
550         jne andFlag0
551
552     andFlag1:   ; Si es 0 despues de la mask la zeroFlag sera 1
553         mov EBX,[EBP+16]    ; EBX = &zeroFlag
554         mov byte [EBX],1
555         jmp SHORT andOrTest
556
557     andFlag0:   ; Si no es 0 despues de la mask la zeroFlag sera 0
558         mov EBX,[EBP+16]    ; EBX = &zeroFlag
559         mov byte [EBX],0
560
561     andOrTest:              ; Comprueba si el ultimo parametro es
562         mov EBX,[EBP+20]    ; 0 para hacer la operacion de AND
```

```
563         cmp EBX,1           ; o 1 para la operacion de TEST
564         je andFin
565
566     and:
567         mov [EAX],EDX
568
569     andFin:
570         pop EDX     ; Restaurar valores
571         pop EBX
572         pop EAX
573         leave
574         ret
575
576
577     ; Funcion especial para elevar al
578     ; cuadrado un numero para la simulacion
579     ; de un procesador de 5 bits, con sus
580     ; limitantes
581     nasmDSF:
582         enter 0,0
583         push EAX    ; Guardar valores
584         push EBX
585         push EDX
586         push ECX
587
588         mov EAX,[EBP+8] ; EAX = &operador1
589         mov EBX,[EAX]   ; EBX = operador1
590
591         cmp EBX,0
592         jge dsfNoNegar
593
594     dsfNegar:
595         neg EBX
596
597     dsfNoNegar:
598         mov EDX,EBX     ; EDX = operador1
599         mov ECX,EBX     ; ECX = operador1
600
601     dsfCasosBase:   ; si es 1 o 0 queda igual el numero
602         je dsfFin
603         cmp EBX,1
604         je dsfFin
605
606         dec ECX     ; se le decrementa 1 para que haga la cantidad correcta
607                     ; de ciclos
608     dsfCiclo:
609         add EBX,EDX     ; Elevacion al cuadrado apartir
610         loop dsfCiclo   ; de sumas consecutivas
611
612     dsfComparacion1:    ; Revision de si esta fuera del rango de
613         cmp EBX,-16     ; 5 bits
614         jl dsfNoRango
615
616     dsfComparacion2:
617         cmp EBX,15
618         jg dsfNoRango
619
620     dsfEnRango:
621         mov EDX,[EBP+12]    ; Si esta en rango no hay overflow y
622         mov byte [EDX],0        ; no hay problemas
623         jmp dsfFin
624
625     dsfNoRango:
626         mov EDX,[EBP+12]    ; Si esta fuera de rango
627         mov byte [EDX],1        ; se coloca el overflow en la flag
628         test EBX,0x10       ; si se simula un fuera de rango
629         jz dsfColocar0
630
631     dsfColocar1:
632         or EBX,0xFFFFFFE0
633         jmp dsfFin
```

```
634
635    dsfColocar0:
636        and EBX,0x1F
637
638    dsfFin:
639        mov [EAX],EBX
640        pop ECX     ; Restaurar valores
641        pop EDX
642        pop EBX
643        pop EAX
644        leave
645        ret
646
647
648    ; Funcion que aplica una mascara de 0x1F
649    ; en el primer parametro y lo retorna
650    ; en el segundo parametro
651    hex5Bits:
652        enter 0,0
653        push EAX    ; Guardar valores
654        push EBX
655        push EDX
656
657        mov EAX,[EBP+8] ; EAX = &numero
658        mov EBX,[EAX]   ; EAX = numero
659        and EBX,0x1F    ; Mascara
660        mov EAX,[EBP+12] ; EAX = &retorno
661        mov [EAX],EBX
662
663        pop EDX     ; Restaurar valores
664        pop EBX
665        pop EAX
666        leave
667        ret
668
669    ; Funcion que retorna el valor binario guardado
670    ; en una variable, a un string
671    decToBin:
672        enter 0,0
673        push EAX    ; Guardar valores
674        push EBX
675        push EDX
676
677        mov EAX,[EBP+8]     ; EAX = &variable
678        mov EBX,[EAX]       ; EBX = variable
679        mov EAX,EBX         ; EAX = variable
680        mov EBX,[EBP+12]    ; EBX = char[]
681
682        mov ECX,5
683    binCiclo:
684        test EAX,0x10
685        jz binColocar0
686
687    binColocar1:
688        mov byte [EBX],49
689        jmp SHORT binCiclar
690
691    binColocar0:
692        mov byte [EBX],48
693
694    binCiclar:
695        inc EBX
696        shl EAX,1
697        loop binCiclo
698        mov byte [EBX],0
699
700        pop EDX     ; Restaurar valores
701        pop EBX
702        pop EAX
703        leave
704        ret
```

6.2. cpu.c

```
1   /*
2   Creado por Deyan Sanabria Fallas
3   Este archivo contiene las funciones y variables necesarias para
4   administrar, leer y ejecutar un archivo con instrucciones de un
5   lenguaje artifical creado por el autor
6   */
7
8   #include <stdio.h>
9   #include <unistd.h>
10  #include <ncurses.h>
11  #define MAX 256
12  #define AMARILLO 1
13  #define ROJO 2
14  #define AZUL 3
15  #define VERDE 4
16  #define MORADO 5
17  #define CYAN 6
18
19  // Funciones ubicadas en NASM
20  extern void nasmSumar(int*,int*,char*,char*,char*,char*);
21  extern void nasmMover(int*,int*);
22  extern void nasmCiclar(int*,int*,int*,char*);
23  extern void nasmMeter(int[],int*,int*);
24  extern void nasmSacar(int[],int*,int*);
25  extern void nasmComparar(int*,int*,char*,char*);
26  extern void nasmSaltar(int*,int*);
27  extern void nasmSaltarI(int*,int*,char*,char*);
28  extern void nasmSaltarNI(int*,int*,char*,char*);
29  extern void nasmRotarI(int*,char*);
30  extern void nasmRotarD(int*,char*);
31  extern void nasmCorrerI(int*,char*);
32  extern void nasmCorrerD(int*,char*);
33  extern void nasmAND(int*,int*,char*,int);
34  extern void nasmDSF(int*,char*);
35  extern void hex5Bits(int*,int*);
36  extern void decToBin(int*,char[]);
37
38  // Variables locales encontradas
39  FILE *archivo;
40  int max_x,max_y;
41  int memoria[16];
42  int pila[8];
43  int posPila = -1;
44  char variables[8][11];
45  char etiquetas[8][11];
46  int etiquetasPos[8];
47  int pCounter = 0;
48  int iRegister = 0;
49  char overflow = 0;
50  char zeroFlag = 0;
51  char signFlag = 0;
52  char carryFlag = 0;
53  int registro1 = 0;
54  int registro2 = 0;
55  int registro3 = 0;
56  int registro4 = 0;
57
58  /*
59  funcion para inicializar el CPU
60  */
61  void initCPU() {
62      posPila = -1;
63      pCounter = 0;
64      iRegister = 0;
65      overflow = 0;
66      zeroFlag = 0;
67      signFlag = 0;
68      carryFlag = 0;
```

```
69          registro1 = 0;
70          registro2 = 0;
71          registro3 = 0;
72          registro4 = 0;
73      }
74
75      /*
76      funcion para señalizar el estado del ciclo de fetch
77      */
78      void fetch(int ciclo) {
79          usleep(500000);
80          char binNumber[6];
81          if(ciclo == 1) {
82              attron(COLOR_PAIR(CYAN));
83              mvprintw(3,max_x/2 - 74,"Registro PC: ");
84              attroff(COLOR_PAIR(CYAN));
85              attron(COLOR_PAIR(MORADO));
86              decToBin(&pCounter,binNumber);
87              printw("%s",binNumber);
88              attroff(COLOR_PAIR(MORADO));
89              move(6,max_x/2 - 5);
90              attron(COLOR_PAIR(ROJO));
91              printw("Fetch, ");
92              attroff(COLOR_PAIR(ROJO));
93              printw("Decode, ");
94              printw("Execute");
95          } else if(ciclo == 2) {
96              attron(COLOR_PAIR(CYAN));
97              mvprintw(4,max_x/2 - 74,"Registro IR: ");
98              attroff(COLOR_PAIR(CYAN));
99              attron(COLOR_PAIR(MORADO));
100             decToBin(&iRegister,binNumber);
101             printw("%s",binNumber);
102             attroff(COLOR_PAIR(MORADO));
103             move(6,max_x/2 - 5);
104             printw("Fetch, ");
105             attron(COLOR_PAIR(ROJO));
106             printw("Decode, ");
107             attroff(COLOR_PAIR(ROJO));
108             printw("Execute");
109         } else if(ciclo == 3) {
110             move(6,max_x/2 - 5);
111             printw("Fetch, ");
112             printw("Decode, ");
113             attron(COLOR_PAIR(ROJO));
114             printw("Execute");
115             attroff(COLOR_PAIR(ROJO));
116         }
117         refresh();
118     }
119
120     /*
121     Funcion para abrir uno de los tres archivos conteniendo el programa
122     a ejecutars
123     */
124     char abrirPrograma(int num) {
125         char *direccion;
126         switch(num) {
127             case 1:
128                 direccion = "prog1.asf";
129                 break;
130             case 2:
131                 direccion = "prog2.asf";
132                 break;
133             case 3:
134                 direccion = "prog3.asf";
135                 break;
136             case 4:
137                 direccion = "progcustom.asf";
138                 break;
139             default:
```

```
140             return 0;
141         }
142         archivo = fopen(direccion, "r");
143         if(archivo == NULL)
144             return 0;
145         return 1;
146     }
147 
148     void cerrarArchivo() {
149         fclose(archivo);
150     }
151 
152     /*
153     Funcion de comparacion de strings, recibe dos arrays de char y sus
154     tamaños. Esto debido a que al pasar un puntero a una funcion, se pierde
155     los datos del tamaño y no puede ser recuperado con "sizeof()"
156     */
157     char comparar(char str1[],char str2[],int size1,int size2) {
158         if(size1 != size2)
159             return 0;
160         for(int i = 0; i < size1; i++) { // compara cada caracter
161             if(str1[i] != str2[i])
162                 return 0;
163             if(str1[i] == 0 && str2[i] == 0)
164                 return 1;
165         }
166         return 0;
167     }
168 
169     /*
170     Funcion auxiliar que interpreta un numero en un string
171     a un numero normal
172     */
173     int stringNumero(char string[], int size) {
174         int numero = 0;
175         char negativo = 0;
176         for(int i = 0; i < size; i++) {
177             if(string[i] == 0) {
178                 break;
179             }
180             if(string[i] == '-') {
181                 negativo = 1;
182                 continue;
183             }
184             numero *= 10;
185             numero += (string[i] - 48);
186         }
187         if(negativo) {
188             numero *= -1;
189         }
190         return numero;
191     }
192 
193     /*
194     funcion auxiliar para guardar en memoria, contiene todas las verificaciones
195     para asociar un "neumonico" con su "opcode"
196     */
197     char instrucciones(int memoria[], char operacion[], int celda) {
198         if(comparar(operacion,"sumar",6,6)) {
199             memoria[celda] = 0;
200             return 1;
201         } if(comparar(operacion,"mover",6,6)) {
202             memoria[celda] = 1;
203             return 1;
204         } if(comparar(operacion,"ciclar",7,7)) {
205             memoria[celda] = 2;
206             return 1;
207         } if(comparar(operacion,"meter",6,6)) {
208             memoria[celda] = 3;
209             return 1;
210         } if(comparar(operacion,"sacar",6,6)) {
```

```
211             memoria[celda] = 4;
212             return 1;
213         } if(comparar(operacion,"comparar",9,9)) {
214             memoria[celda] = 5;
215             return 1;
216         } if(comparar(operacion,"rotarI",7,7)) {
217             memoria[celda] = 6;
218             return 1;
219         } if(comparar(operacion,"rotarD",7,7)) {
220             memoria[celda] = 7;
221             return 1;
222         } if(comparar(operacion,"correrI",8,8)) {
223             memoria[celda] = 8;
224             return 1;
225         } if(comparar(operacion,"correrD",8,8)) {
226             memoria[celda] = 9;
227             return 1;
228         } if(comparar(operacion,"test",5,5)) {
229             memoria[celda] = 10;
230             return 1;
231         } if(comparar(operacion,"and",4,4)) {
232             memoria[celda] = 11;
233             return 1;
234         } if(comparar(operacion,"saltar",7,7)) {
235             memoria[celda] = 12;
236             return 1;
237         } if(comparar(operacion,"saltarI",8,8)) {
238             memoria[celda] = 13;
239             return 1;
240         } if(comparar(operacion,"saltarNI",9,9)) {
241             memoria[celda] = 14;
242             return 1;
243         } if(comparar(operacion,"DSF",4,4)) {
244             memoria[celda] = 15;
245             return 1;
246         } if(comparar(operacion,"entrada",8,8)) {
247             memoria[celda] = 16;
248             return 1;
249         } if(comparar(operacion,"salida",7,7)) {
250             memoria[celda] = 17;
251             return 1;
252         }
253         return 0;
254     }
255 
256     /*
257     Funcion auxiliar que ayuda a obtener la siguiente palabra despues
258     de un espacio
259     */
260     int siguientePalabra(char linea[], char almacen[], int inicio) {
261         int espacio = 0;
262         int retorno;
263         for(int i = inicio; i < MAX; i++) {
264             retorno = i;
265             if(linea[i] != ' ' && linea[i] != '\n') {
266                 almacen[espacio] = linea[i];
267                 espacio++;
268             } else if (espacio > 0)
269                 break;
270         }
271         almacen[espacio] = 0;
272         return retorno;
273     }
274 
275     /*
276     Funcion auxiliar para guardar en un arreglo bidimensional strings
277     */
278     void guardarString(char almacen[][11], char string[], int tamaño,int pos) {
279         for(int i = 0; i < tamaño; i++) {
280             almacen[pos][i] = string[i];
281         }
```

```
282    }
283
284    /*
285    Funcion auxiliar para verificar si algo es una etiqueta
286    */
287    char esEtiqueta(char string[], int tamaño){
288        for(int i = 0; i < tamaño; i++) {
289            if(string[i] == 0) {
290                if(string[i-1] == ':') {
291                    return 1;
292                }
293            }
294        }
295        return 0;
296    }
297
298    // Funcion que recibe una linea leida del archivo
299    // y revisa si es una linea vacia
300    char esLineaVacia(char linea[]) {
301        for(int i = 0; i < MAX; i++) {
302            if(linea[i] == 0)
303                return 1;
304            if(linea[i] != ' ' && linea[i] != '\n') {
305                return 0;
306            }
307        }
308        return 0;
309    }
310
311    /*
312    Funcion para "guardar en memoria" datos del programa
313    */
314    char guardarEnMemoria() {
315        // variables necesarias para el funcionamiento del programa
316        char linea[MAX];
317        char palabra[11];
318        char inicio = 0;
319        char datos = 0;
320        int variable = 0;
321        int celda = 0;
322        int etiqueta = 0;
323        int ultimaPos;
324        int ultimaLinea = -1;
325
326        // El ciclo se mantiene hasta que se llega al EOF
327        while (fgets(linea,MAX,archivo)) {
328            ultimaLinea++;
329            ultimaPos = siguientePalabra(linea,palabra,0); // se obtiene una palabra de la linea leida
330
331            // Par de ifs para verificar si se esta en el segmento de datos
332            // o en la parte del codigo
333            if(comparar(palabra,"::Datos",8,8)) {
334                datos = 1;
335                continue;
336            } else if(comparar(palabra,"-INICIO",8,8)) {
337                datos = 0;
338                inicio = 1;
339                continue;
340            }
341
342            // Si se llena la memoria retorna la funcion
343            if(celda == 16 || variable == 8 || etiqueta == 8)
344                return 0;
345
346            // Si se esta en el segmento de datos
347            if(datos) {
348                if(esLineaVacia(linea)) // se revisa por lineas vacias para saltarlas
349                    continue;
350                guardarString(variables,palabra,11,variable); // se guarda nombre de variables
351                variable++;
352                siguientePalabra(linea,palabra,ultimaPos);  // y se busca por el valor a guardar
```

```
353             memoria[celda] = stringNumero(palabra,11);
354             celda++;
355         }
356
357         // Si se esta en el segmento de codigo
358         if(inicio) {
359             if(instrucciones(memoria,palabra,celda)) // Comprueba que lo leido es una instruccion
360                 celda++;
361             else if (esEtiqueta(palabra,11)) {      // Comprueba si es una etiqueta
362                 guardarString(etiquetas,palabra,11,etiqueta); // Si lo es la guarda junto
363                 etiquetasPos[etiqueta] = ultimaLinea;              // a su posicion en el codio
364                 etiqueta++;
365             } else if(comparar(palabra,"-FIN",5,5))
366                 return 1;                           // La lectura del programa solo es exitosa si
367         }                                           // encuentra el final del codio
368     }
369     return 0;
370 }
371
372 // Funcion que separa dos operadores separados por comas, tambien
373 // sirve para encontrar solo el primer operador
374 char obtenerOperadores(char op1[],char op2[],char operadores[]) {
375     // variables locales
376     char *temp = op1;
377     int charPos = 0;
378     char retorno = 0;
379
380     // busca por el string donde se encuentran los operadores
381     for(int i = 0; i < 22; i++) {
382         if(operadores[i] == ',') { // si encuentra una coma, cambia al
383             temp[charPos] = 0;     // segundo operador y reinicia
384             temp = op2;            // la posicion del string del operador
385             charPos = 0;
386             retorno = 1;
387             continue;
388         }
389         if(operadores[i] == 0 || charPos == ' ') {
390             temp[charPos] = 0;   // este if busca por el final
391             charPos++;           // del string del operador
392             break;
393         }
394         temp[charPos] = operadores[i];
395         charPos++;
396
397     }
398     return retorno;
399 }
400
401 // Funcion que retorna un registro si el parametro se
402 // refiere a uno o NULL si este no es un registro
403 int *esRegistro(char operador[]) {
404     int *temp = NULL;
405
406     // Compara los strings con el parametro
407     // si coincide con alguno asigna la direccion
408     // de memoria del registro, de lo contrario
409     // retorna NULL
410     if(comparar(operador,"R1",3,3)) {
411         temp = &registro1;
412         return temp;
413     } if(comparar(operador,"R2",3,3)) {
414         temp = &registro2;
415         return temp;
416     } if(comparar(operador,"R3",3,3)) {
417         temp = &registro3;
418         return temp;
419     } if(comparar(operador,"R4",3,3)) {
420         temp = &registro4;
421         return temp;
422     }
423     return temp;
```

```
424    }
425
426    // Funcion para obtener la direccion de memoria
427    // de una variable en la "memoria principal"
428    int *obtenerDirMem(char operador[]) {
429        int *temp = NULL; // Se almacena la direccion de memoria si se encuentra
430        for(int i = 0; i < 8; i++) {
431            if(comparar(variables[i],operador,11,11)) {
432                temp = &(memoria[i]);  // se busca por todas las variables si coincide
433            }                          // con el parametro, se asigna a temp la direccion
434        }                              // de memoria y se retorna
435        return temp;
436    }
437
438    // Funcion para obtener la linea a la que pertenece
439    // una etiqueta
440    int *obtenerLineaEjecucion(char operador[]) {
441
442        // Variable donde se guarda la linea de la etiqueta
443        int *temp = NULL;
444
445        // Ciclo for para buscar en el arreglo de etiquetas si estas
446        // si lo que se recibe de parametro es una etiqueta
447        for(int i = 0; i < 8; i++) {
448            // se compara la entrada con las etiquetas del arreglo
449            if(comparar(etiquetas[i],operador,11,11)) {
450                temp = &etiquetasPos[i];    // Si la encuentra asigna a temp
451                break;                      // la direccion de memoria donde
452            }                               // esta el numero de linea a saltar
453        }
454        return temp;
455    }
456
457    // Funcion para obtener un string dentro del lenguaje creado
458    char obtenerString(char almacen[],char linea[]) {
459
460        // Variables
461        char comilla1 = 0;
462        char retorno = 0;
463        int posAlmacen = 0;
464
465        // Ciclo for para recorrer toda la linea leida
466        for(int i = 0; i < MAX; i++) {
467            // Si se encuentra un null, no hay mas string
468            // por lo que se rompe el ciclo y retorna
469            if(linea[i] == 0)
470                break;
471
472            // Si se encuentra una comilla y no se habia encontrado
473            // previamente, se altera una variable para avisar que
474            // empezo el string
475            if(linea[i] == '"' && !(comilla1)) {
476                comilla1 = 1;
477                continue;
478            }
479
480            // Si la primer comilla se encuentra
481            if(comilla1) {
482
483                // Revisa si el caracter que se esta leyendo se es un
484                // backslash, y no se ha llegado al final de la linea
485                // para comprobar si el caracter que sigue despues del
486                // backslash es una 'n' indicando que se quiere imprimir un
487                // newline
488                if(linea[i] == '\\' && i != MAX - 1) {
489                    if(linea[i+1] == 'n') {
490                        almacen[posAlmacen] = '\n';
491                        i++;
492                        posAlmacen++;
493                        continue;
494                    }
```

```
            } else if(linea[i] == '"') { // Si no revisa si se encontro el final
                retorno = 1;             // del string, si es asi rompe el ciclo
                break;                   // y retorna con exito
            }
            almacen[posAlmacen] = linea[i]; // En esta seccion copia cada caracter del string
            posAlmacen++;                   // en otro lugar
        }
    }
    almacen[posAlmacen] = 0; // se coloca un NULL al final del string para saber su final
    return retorno;
}
// Funcion que lee la instruccion y si esta la sintaxis correcta, se ejecuta
char correrInstruccion(char operacion[],char linea[],int pos, int *lineaPos) {

    // Variables indispensables
    char operadores[22];
    char opStr1[11];
    char opStr2[11];
    int *operador1 = NULL;
    int *operador2 = NULL;
    char string[50];
    char esString = 1;

    // Se intenta obtener un string, si esto resulta no se uno
    // se ejecuta la obtencion de operadores
    fetch(2);
    if(!(obtenerString(string,linea))) {
        esString = 0;
        siguientePalabra(linea,operadores,pos);
        obtenerOperadores(opStr1,opStr2,operadores); // se separan los operadores
        operador1 = esRegistro(opStr1);     // Se revisa si son registros algun operador
        operador2 = esRegistro(opStr2);

        // si estos no son registros se intenta leer como una variable
        if(operador1 == NULL) {
            operador1 = obtenerDirMem(opStr1);
        }
        if(operador2 == NULL) {
            operador2 = obtenerDirMem(opStr2);
        }

        // Si no es una variable se intenta como si fuera una etiqueta
        if(operador1 == NULL) {
            operador1 = obtenerLineaEjecucion(opStr1);
        }

        // Si no se interpreta como un numero
        if(operador1 == NULL) {
            int temp = stringNumero(opStr1,11);
            operador1 = &temp;
        }
        if(operador2 == NULL) {
            int temp = stringNumero(opStr2,11);
            operador2 = &temp;
        }
    }

    // Esta seccion se usa para ejecutar la operacion que se encontro
    // pasando los parametros, registros y flags que se necesiten alterar
    fetch(3);
    if(comparar(operacion,"sumar",6,6)) {
        if(operador1 == NULL && operador1 == NULL)
            return 0;
        nasmSumar(operador1,operador2,&overflow,&carryFlag,&signFlag,&zeroFlag);
        return 1;
    } if(comparar(operacion,"mover",6,6)) {
        if(operador1 == NULL && operador1 == NULL)
            return 0;
        nasmMover(operador1,operador2);
        return 1;
    } if(comparar(operacion,"ciclar",7,7)) {
```

```
566             if(operador1 == NULL)
567                 return 0;
568             char reiniciar = 0;
569             nasmCiclar(operador1,&registro3,&pCounter,&reiniciar);
570             if(reiniciar) {
571                 *lineaPos = -1;
572                 rewind(archivo);
573             }
574             return 1;
575         } if(comparar(operacion,"meter",6,6)) {
576             if(operador1 == NULL)
577                 return 0;
578             nasmMeter(pila,operador1,&posPila);
579             return 1;
580         } if(comparar(operacion,"sacar",6,6)) {
581             if(operador1 == NULL)
582                 return 0;
583             nasmSacar(pila,operador1,&posPila);
584             return 1;
585         } if(comparar(operacion,"comparar",9,9)) {
586             if(operador1 == NULL && operador1 == NULL)
587                 return 0;
588             nasmComparar(operador1,operador2,&zeroFlag,&signFlag);
589             return 1;
590         } if(comparar(operacion,"rotarI",7,7)) {
591             if(operador1 == NULL)
592                 return 0;
593             nasmRotarI(operador1,&carryFlag);
594             return 1;
595         } if(comparar(operacion,"rotarD",7,7)) {
596             if(operador1 == NULL)
597                 return 0;
598             nasmRotarD(operador1,&carryFlag);
599             return 1;
600         } if(comparar(operacion,"correrI",8,8)) {
601             if(operador1 == NULL)
602                 return 0;
603             nasmCorrerI(operador1,&carryFlag);
604             return 1;
605         } if(comparar(operacion,"correrD",8,8)) {
606             if(operador1 == NULL)
607                 return 0;
608             nasmCorrerD(operador1,&carryFlag);
609             return 1;
610         } if(comparar(operacion,"test",5,5)) {
611             if(operador1 == NULL && operador2 == NULL)
612                 return 0;
613             nasmAND(operador1,operador2,&zeroFlag,1);
614             return 1;
615         } if(comparar(operacion,"and",4,4)) {
616             if(operador1 == NULL && operador2 == NULL)
617                 return 0;
618             nasmAND(operador1,operador2,&zeroFlag,0);
619             return 1;
620         } if(comparar(operacion,"saltar",7,7)) {
621             if(operador1 == NULL)
622                 return 0;
623             nasmSaltar(operador1,&pCounter);
624             *lineaPos = -1;
625             rewind(archivo);
626             return 1;
627         } if(comparar(operacion,"saltarI",8,8)) {
628             if(operador1 == NULL)
629                 return 0;
630             char reiniciar = 0;
631             nasmSaltarI(operador1,&pCounter,&zeroFlag,&reiniciar);
632             if(reiniciar) {
633                 *lineaPos = -1;
634                 rewind(archivo);
635             }
636             return 1;
```

```
637         } if(comparar(operacion,"saltarNI",9,9)) {
638             if(operador1 == NULL)
639                 return 0;
640             char reiniciar = 0;
641             nasmSaltarNI(operador1,&pCounter,&zeroFlag,&reiniciar);
642             if(reiniciar) {
643                 *lineaPos = -1;
644                 rewind(archivo);
645             }
646             return 1;
647         } if(comparar(operacion,"DSF",4,4)) {
648             if(operador1 == NULL)
649                 return 0;
650             nasmDSF(operador1,&overflow);
651             return 1;
652         } if(comparar(operacion,"entrada",8,8)) {
653             if(operador1 == NULL)
654                 return 0;
655             move(12,max_x/2 - 5);
656             attron(COLOR_PAIR(VERDE));
657             printw("Entrada: ");
658             attroff(COLOR_PAIR(VERDE));
659             scanw("%d",operador1);
660             while (*operador1 < -16 || *operador1 > 15) {
661                 attron(COLOR_PAIR(ROJO));
662                 mvprintw(13,max_x/2 - 5,"Numeros entre -16 y 15, intente de nuevo: ");
663                 attroff(COLOR_PAIR(ROJO));
664                 move(12,max_x/2 - 5);
665                 attron(COLOR_PAIR(VERDE));
666                 printw("Entrada:                                                  ");
667                 attroff(COLOR_PAIR(VERDE));
668                 move(12,max_x/2 + 4);
669                 scanw("%d",operador1);
670             }
671             return 1;
672         } if(comparar(operacion,"salida",7,7)) {
673             if(esString) {
674                 move(12,max_x/2 - 5);
675                 attron(COLOR_PAIR(VERDE));
676                 printw("Salida: ");
677                 attroff(COLOR_PAIR(VERDE));
678                 printw("%s",string);
679             } else {
680                 if(operador1 == NULL)
681                     return 0;
682                 move(12,max_x/2 - 5);
683                 attron(COLOR_PAIR(VERDE));
684                 printw("Salida: ");
685                 attroff(COLOR_PAIR(VERDE));
686                 printw("%d",*operador1);
687             }
688             refresh();
689             noecho();
690             getch();
691             echo();
692             return 1;
693         }
694         return 1;
695     }
```

6.3. main.c

```
1   /*
2   Creado por Deyan Sanabria Fallas
3   Este archivo contiene la parte principal del menu
4   para la ejecucion del simulador de CPU, para su uso
5   se necesita el archivo cpu.c y operaciones.o (compilado
6   por NASM y enlazado por GCC a la hora de la compilacion
7   de main.c) por favor compilar todo en 32bits.
8   */
9
10  #include <stdio.h>
11  #include <stdlib.h>
12  #include "cpu.c"
13
14  char automatico = 1;
15
16  /*
17  Funcion para inicializar la memoria y la pila
18  en 0, pasa por todos sus espacios y asigna 0
19  */
20  void inicializarMemoria() {
21      for(int i = 0; i < 16; i++) {
22          memoria[i] = 0;
23          if(i < 8)
24              pila[i] = 0;
25      }
26  }
27
28  /*
29  Funcion que retorna el indice donde empieza
30  un string, ignorando los espacios iniciales
31  */
32  int primerCaracterLinea(char linea[]) {
33      int temp = -1;
34      for(int i = 0; i < MAX; i++) {
35          if(linea[i] == 0) {         // si encuentra null significa que
36              break;                  // no hay mas string
37          }
38          if(linea[i] != ' ') {       // cuando encuentre algo diferente
39              temp = i;               // a espacio, rompe el ciclo y retorna
40              break;                  // el indice
41          }
42      }
43      return temp;
44  }
45
46  /*
47  Funcion para imprimir el CPU en pantalla
48  */
49  void printCPU(char strArch[]) {
50      clear(); // Limpia la pantalla anterior
51      char binNumber[6];
52
53      // Imprimir registros
54      attron(COLOR_PAIR(AMARILLO));
55      mvprintw(2,max_x/2 - 74,"REGISTROS BASICOS:");
56      attroff(COLOR_PAIR(AMARILLO));
57      attron(COLOR_PAIR(CYAN));
58      mvprintw(3,max_x/2 - 74,"Registro PC: ");
59      attroff(COLOR_PAIR(CYAN));
60      attron(COLOR_PAIR(MORADO));
61      decToBin(&pCounter,binNumber);
62      printw("%05s",binNumber);
63      attroff(COLOR_PAIR(MORADO));
64      attron(COLOR_PAIR(CYAN));
65      mvprintw(4,max_x/2 - 74,"Registro IR: ");
66      attroff(COLOR_PAIR(CYAN));
67      attron(COLOR_PAIR(MORADO));
68      decToBin(&iRegister,binNumber);
```

```
69        printw("%05s",binNumber);
70        attroff(COLOR_PAIR(MORADO));
71        attron(COLOR_PAIR(CYAN));
72        mvprintw(5,max_x/2 - 74,"R1: ");
73        attroff(COLOR_PAIR(CYAN));
74        attron(COLOR_PAIR(MORADO));
75        decToBin(&registro1,binNumber);
76        printw("%05s",binNumber);
77        attroff(COLOR_PAIR(MORADO));
78        attron(COLOR_PAIR(CYAN));
79        mvprintw(6,max_x/2 - 74,"R2: ");
80        attroff(COLOR_PAIR(CYAN));
81        attron(COLOR_PAIR(MORADO));
82        decToBin(&registro2,binNumber);
83        printw("%05s",binNumber);
84        attroff(COLOR_PAIR(MORADO));
85        attron(COLOR_PAIR(CYAN));
86        mvprintw(7,max_x/2 - 74,"R3: ");
87        attroff(COLOR_PAIR(CYAN));
88        attron(COLOR_PAIR(MORADO));
89        decToBin(&registro3,binNumber);
90        printw("%05s",binNumber);
91        attroff(COLOR_PAIR(MORADO));
92        attron(COLOR_PAIR(CYAN));
93        mvprintw(8,max_x/2 - 74,"R4: ");
94        attroff(COLOR_PAIR(CYAN));
95        attron(COLOR_PAIR(MORADO));
96        decToBin(&registro4,binNumber);
97        printw("%05s",binNumber);
98        attroff(COLOR_PAIR(MORADO));
99
100       // Imprimir la instruccion actual
101       attron(COLOR_PAIR(AMARILLO));
102       mvprintw(10,max_x/2 - 74,"Instruccion actual: ");
103       attroff(COLOR_PAIR(AMARILLO));
104       int indice = primerCaracterLinea(strArch);  // busca que no hayan espacios sobrantes
105       if(indice != -1) {                          // antes, debido a que se toma del archivo
106           attron(COLOR_PAIR(AZUL));               // la linea para imprimirla
107           printw("%s",&strArch[indice]);
108           attroff(COLOR_PAIR(AZUL));
109       }
110
111       // Imprimir flags
112       attron(COLOR_PAIR(AMARILLO));
113       mvprintw(12,max_x/2 - 74,"FLAGS:");
114       attroff(COLOR_PAIR(AMARILLO));
115       attron(COLOR_PAIR(CYAN));
116       mvprintw(13,max_x/2 - 74,"Overflow: ");
117       attroff(COLOR_PAIR(CYAN));
118       attron(COLOR_PAIR(MORADO));
119       printw("%c",overflow?'1':'0');
120       attroff(COLOR_PAIR(MORADO));
121       attron(COLOR_PAIR(CYAN));
122       mvprintw(14,max_x/2 - 74,"Carry: ");
123       attroff(COLOR_PAIR(CYAN));
124       attron(COLOR_PAIR(MORADO));
125       printw("%c",carryFlag?'1':'0');
126       attroff(COLOR_PAIR(MORADO));
127       attron(COLOR_PAIR(CYAN));
128       mvprintw(15,max_x/2 - 74,"Zero: ");
129       attroff(COLOR_PAIR(CYAN));
130       attron(COLOR_PAIR(MORADO));
131       printw("%c",zeroFlag?'1':'0');
132       attroff(COLOR_PAIR(MORADO));
133       attron(COLOR_PAIR(CYAN));
134       mvprintw(16,max_x/2 - 74,"Sign: ");
135       attroff(COLOR_PAIR(CYAN));
136       attron(COLOR_PAIR(MORADO));
137       printw("%c",signFlag?'1':'0');
138       attroff(COLOR_PAIR(MORADO));
139       for(int i = 2; i < 18; i++) {
```

```
140             mvprintw(i,max_x/2 - 7,"|");
141         }
142 
143         // Imprimir memoria principal
144         attron(COLOR_PAIR(AMARILLO));
145         mvprintw(2,max_x/2 - 5,"Memoria Principal:");
146         attroff(COLOR_PAIR(AMARILLO));
147         int j = 5;
148         int linea = 3;
149         int hex;
150 
151         attron(COLOR_PAIR(MORADO));
152         for(int i = 0; i < 16; i++) {
153             hex5Bits(&memoria[i],&hex);
154             mvprintw(linea,max_x/2 - j,"0x%x",hex);
155             if(i != 15) {
156                 printw(", ");
157             }
158             j -= 6;
159             if(j < -37) {
160                 j = 5;
161                 linea++;
162             }
163         }
164         attroff(COLOR_PAIR(MORADO));
165 
166         // Imprimir la pila
167         j = 5;
168         attron(COLOR_PAIR(AMARILLO));
169         mvprintw(8,max_x/2 - 5,"Pila:");
170         attroff(COLOR_PAIR(AMARILLO));
171         attron(COLOR_PAIR(MORADO));
172         for(int i = 0; i < 8; i++) {
173             hex5Bits(&pila[i],&hex);
174             mvprintw(9,max_x/2 - j,"0x%x",hex);
175             if(i != 7) {
176                 printw(", ");
177             }
178             j -= 6;
179         }
180         attroff(COLOR_PAIR(MORADO));
181 
182         // Imprimir ciclo fetch
183         fetch(1);
184         attron(COLOR_PAIR(AMARILLO));
185         mvprintw(11,max_x/2 - 5,"Interaccion con el programa:");
186         attroff(COLOR_PAIR(AMARILLO));
187         refresh();
188     }
189 
190     /*
191     Funcion para ejecutar el programa seleccionado
192     */
193     void ejecucion() {
194         // Variables necesarias
195         char linea[MAX];
196         char palabra[22];
197         int ultimaPos;
198         int lineaPos = -1;
199         char seccionCodigo = 0;
200         linea[0] = 0;
201         printCPU(linea);
202         // Se inicia el archivo desde el incio
203         rewind(archivo);
204 
205         // En cada ciclo se obtiene una linea del archivo con
206         // la funcion "fgets()", dicha funcion retorna 0 cuando
207         // llega a EOF, de lo contrario retorna 1
208         while (fgets(linea,MAX,archivo)) {
209             lineaPos++;             // se cuenta por cual linea del archivo se esta
210 
```

```
211             ultimaPos = siguientePalabra(linea,palabra,0); // se obtiene la primera palabra de la linea
212                                                            // y retorna la ultima posicion leida
213
214             if(comparar(palabra,"-INICIO",8,8))  // revisa si llego al inicio del codigo
215                 seccionCodigo = 1;
216             if(seccionCodigo) {     // una vez en el inicio del codigo ejecuta la instruccion leida
217                 iRegister = pCounter;
218                 if(iRegister != lineaPos) { // se usa para poder saltar a otra linea que
219                     continue;               // no sea la siguiente
220                 }
221                 fetch(1);
222                 printCPU(linea);
223                 if(!(correrInstruccion(palabra,linea,ultimaPos,&lineaPos)))// Si la instruccion no se
224                     return;                                     // pudo ejecutar, se retorna la funcion
225                 printCPU(linea);
226                 // Parar el programa
227                 if(!automatico) {
228                     noecho();
229                     getch();
230                     echo();
231                 } else {
232                     usleep(1000000);
233                 }
234
235             }
236             pCounter++;
237         }
238     }
239
240     void correrPrograma() {
241
242     }
243
244     /*
245     Imprime en pantalla el menu cada vez que se invoca.
246     Antes de imprimir, se limpia la pantalla.
247     */
248     void printMenu() {
249         clear(); // Limpia la pantalla anterior
250         attron(COLOR_PAIR(AMARILLO));
251         mvprintw(2,max_x/2 - 16,"BIENVENIDO AL SIMULADOR HAL 9000");
252         attroff(COLOR_PAIR(AMARILLO));
253
254         mvprintw(4,max_x/2 - 11,"Seleccione un programa");
255
256         attron(COLOR_PAIR(AZUL));
257         mvprintw(6,max_x/2 - 14,"1. Sumar 3");
258         mvprintw(7,max_x/2 - 14,"2. Duplicar numero");
259         mvprintw(8,max_x/2 - 14,"3. Comparar numeros");
260         mvprintw(9,max_x/2 - 14,"4. Programa Custom");
261         mvprintw(10,max_x/2 - 14,"5. Habilitar corrida automatica");
262         mvprintw(11,max_x/2 - 14,"6. Habilitar corrida manual");
263         mvprintw(12,max_x/2 - 14,"0. Salir");
264         attroff(COLOR_PAIR(AZUL));
265
266         mvprintw(14,max_x/2 - 4,"Opcion: ");
267         refresh(); // Muestra en pantalla
268     }
269
270     /*
271     muestra el menu principal y recibe las opciones presentados
272     */
273     char menu_principal() {
274         printMenu();
275         int opcion = -1;
276         do {
277             opcion = -1;
278             scanw("%d", &opcion); // Obtiene la opcion que quiere el usuario
279             switch (opcion) { // se elige en este lugar
280             case 1:
281                 initCPU();
```

```
282                 abrirPrograma(1);
283                 guardarEnMemoria();
284                 ejecucion();
285                 printMenu();
286                 return 1;
287             case 2:
288                 initCPU();
289                 abrirPrograma(2);
290                 guardarEnMemoria();
291                 ejecucion();
292                 printMenu();
293                 return 1;
294             case 3:
295                 initCPU();
296                 abrirPrograma(3);
297                 guardarEnMemoria();
298                 ejecucion();
299                 printMenu();
300                 return 1;
301             case 4:
302                 initCPU();
303                 abrirPrograma(4);
304                 guardarEnMemoria();
305                 ejecucion();
306                 printMenu();
307                 return 1;
308             case 5:
309                 automatico = 1;
310                 printMenu();
311                 attron(COLOR_PAIR(VERDE));
312                 mvprintw(16,max_x/2 - 12,"Corrida automatica habilitada");
313                 attroff(COLOR_PAIR(VERDE));
314                 move(14,max_x/2 + 4);
315                 refresh();
316                 break;
317             case 6:
318                 automatico = 0;
319                 printMenu();
320                 attron(COLOR_PAIR(VERDE));
321                 mvprintw(16,max_x/2 - 12,"Corrida manual habilitada");
322                 attroff(COLOR_PAIR(VERDE));
323                 move(14,max_x/2 + 4);
324                 refresh();
325                 break;
326             default:
327                 // En caso de alguna opcion que no este dentro de las solicitadas
328                 printMenu(); // Se imprime el menu haciendo que se borre lo que haya escrito el usuario
329                 attron(COLOR_PAIR(ROJO));
330                 mvprintw(16,max_x/2 - 12,"Error vuelva a intentarlo");// mensaje de error
331                 attroff(COLOR_PAIR(ROJO));
332                 move(14,max_x/2 + 4);
333                 refresh();
334                 break;
335             }
336         } while (opcion != 0);
337         return 0;
338     }
339
340     int main() {
341         char salir;
342         inicializarMemoria();
343         while (salir) { // Se mete el reinicio de ventana en ciclo por errores
344             // Inicio de ventana
345             initscr();
346
347             if (has_colors() == false) {// Mensaje la ventana no soporta colores si fuese el caso
348                 endwin();
349                 printf("La terminal no soporta colores\n");
350                 return 0;
351             }
352
```

```
353             // Se obtienen las dimensiones de la consola
354             getmaxyx(stdscr,max_y,max_x);
355 
356             // Se crean los colores que se usaran enla ventana
357             start_color();
358             init_pair(AMARILLO,COLOR_YELLOW,COLOR_BLACK);
359             init_pair(ROJO,COLOR_RED,COLOR_BLACK);
360             init_pair(AZUL,COLOR_BLUE,COLOR_BLACK);
361             init_pair(VERDE,COLOR_GREEN,COLOR_BLACK);
362             init_pair(MORADO,COLOR_MAGENTA,COLOR_BLACK);
363             init_pair(CYAN,COLOR_CYAN,COLOR_BLACK);
364 
365             // Inicio de menu principal
366             salir = menu_principal();
367 
368             // Fin de la ventana
369             endwin();
370         }
371 
372         return 0;
373     }
```

6.4. prog1.asf

```
1    PROGRAMA CREADO POR DEYAN SANABRIA FALLAS, ESTE PROGRAMA
2    PIDE 3 NUMEROS AL USUARIO Y LOS SUMA, LUEGO DA EL RESULTADO
3    DE DICHA SUMA
4
5    ::Datos
6    numero 0
7
8    -INICIO
9        mover R3,3
10   ciclo:
11       salida "Ingrese un numero: "
12       entrada R1
13       sumar numero,R1
14       ciclar ciclo:
15       salida "El resultado de la suma es: "
16       salida numero
17       salida "\n"
18
19   -FIN
```

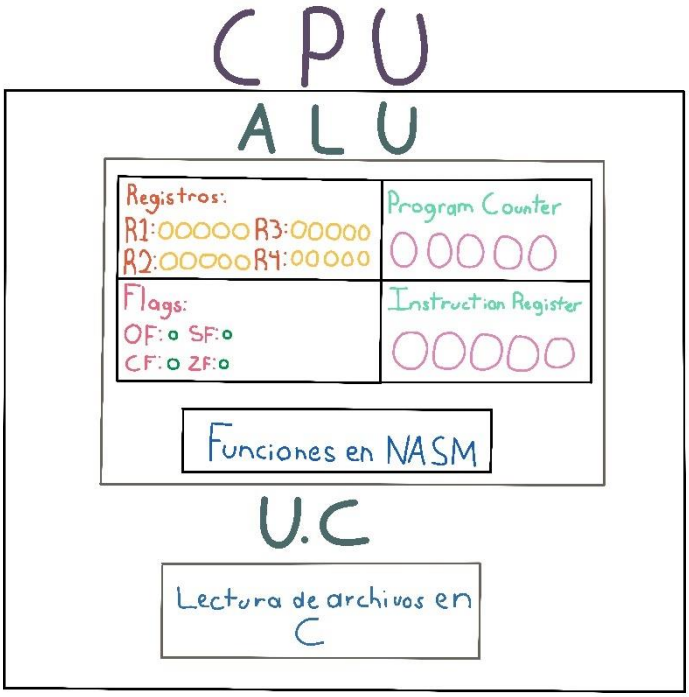
6.5. prog2.asf

```
1    PROGRAMA CREADO POR DEYAN SANABRIA FALLAS, ESTE PROGRAMA SOLICITA UN
2    NUMERO AL USUARIO Y LO DUPLICA, POSTERIORMENTE LO IMPRIME EN PANTALLA
3    PARA QUE EL USUARIO LO RECIBA
4
5    ::Datos
6
7    -INICIO
8        salida "Ingrese un numero que desee duplicar: "
9        entrada R1
10       correrI R1
11       salida "El numero duplicado es: "
12       salida R1
13       salida "\n"
14   -FIN
```

6.6. prog3.asf

```
PROGRAMA CREADO POR DEYAN SANABRIA FALLAS, DICHO PROGRAMA SE ENCARGA DE
COMPARAR DOS NUMEROS INGRESADOS POR EL USUARIO, EL PROGRAMA LE DICE
AL USUARIO SI DICHOS NUMEROS SON IGUALES O SON DIFERENTES

::Datos

-INICIO
    salida "Ingrese un numero a comparar: "
    entrada R1
    salida "Ingrese otro numero para comparar: "
    entrada R2

    comparar R1,R2
    saltarNI no:

Igual:
    salida "Los numeros son iguales\n"
    saltar final:

no:
    salida "Los numeros no son iguales\n"

final:
-FIN
```

6.7. Diagrama del Procesador



Este diagrama explica como esta compuesto el procesador, empezando por los datos de los registros, flags, PC e IR que componen una parte de la ALU, la otra parte la controla las funciones de NASM, las cuales realizan las instrucciones que salen en el código, en conjunto crean la ALU.

Al final, la Unidad de Control, que esta establecida por la lectura de archivos escrita en C y en conjunto con la ALU, crean el CPU