UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Informe Tarea
ARM Assembly

Grupo 22
Martín Arancibia 201973517-9
Miguel Soto 201973623-K

6 de diciembre de 2022

Índice

1. Resumen

En esta tarea se nos pedía el aprendizaje y la utilización de lenguajes de descripción de hardware, en este caso Assembly. Debido a que el lenguaje puede ser un poco engorroso de aplicar a una problemática específica, la manera en la que se reforzarán las habilidades necesarias para poder demostrar aprendizaje del lenguaje será por medio de la creación de 3 funciones distintas, las cuales cada una nos permitirá demostrar de maneras complementarias que se cumplieron los objetivos deseados sobre esta parte de la asignatura.

Es importante recalcar que esta tarea es la culminación de todo lo aprendido en el curso, haciendo uso de todos los conocimientos previamente adquiridos con el objetivo de desarrollar esta misma.

2. Desarrollo

Lo primero que se hizo fue instalar la herramienta de `https://pypi.org/project/qtarmsim/`, QtARMSim. La herramienta contó con una serie de dificultades, ya que pedía una versión anterior de Python para poder funcionar, sin embargo, una vez que tuvimos el programa instalado este funcionó sin mayores inconvenientes.

Una vez que tenemos todo preparado para poder empezar, empezamos a desarrollar los ejercicios que nos son planteados en la tarea. Para cada uno de estos, a la hora de inicializar los datos con los casos de prueba se debe indicar que función se desea ocupar y entregar todos los valores solicitados de manera correcta para el funcionamiento apropiado de la tarea.

2.1. Primera función

Lo primero que se nos pide es trabajar con el uso de la memoria RAM, para ello, debemos extraer todos los datos de una matriz de 3x3 con el objetivo de sacar el determinante de la misma. Recordemos que para obtener el determinante de una matriz se puede obtener multiplicando las diagonales de la matriz hacia la derecha y restándole la multiplicación de las diagonales de la izquierda de la matriz.

Sabiendo esto, se parte cargando en la memoria toda la matriz como un arreglo, de manera que podamos trabajar con los valores específicos en cada una de las posiciones. Junto a esto, inicializamos en 0 el valor del determinante de la matriz, para poder luego guardar el resultado en un registro.

Llamaremos "pendiente positivas.ª las pendientes que van de derecha a izquierda, siendo lo primero que calcularemos. Para ello, vamos cargando en memoria los distintos valores de la matriz, sabiendo que esta es una matriz de 3x3 y que necesitamos posiciones específicas para cada número (Por lo que tomamos sus posiciones a la hora de usar el LDR con sus respectivos registros).

Una vez multiplicamos los valores específicos de cada diagonal, como estamos en la parte de la pendiente positiva sumamos el multiplo al registro R5 que habíamos inicializado antes para poder guardar este resultado, y lo repetimos para las 3 diagonales.

Ahora continuamos con las "diagonales negativas", las diagonales contrarias para la resta. Repetimos el proceso, hacemos un LDR a las posiciones de la matriz especificas, se multiplican entre sí y empezamos a restarle los múltiplos a R5 que es el registro donde estamos almacenando el valor. Con todo esto, luego de que se realizaron las últimas restas queda guardado en R5, dándonos así el resultado del determinante.

```
56 @ Primera función (memoria) : Determinante de una matriz
57 funcionuno:
58         @ Carga la matriz en memoria como un arreglo
59         LDR r1, =nums
60
61         @ Definimos la determinante de la siguiente forma
62         @ DET = A - B
63         @ Donde
64         @       A = Suma de las diagonales negativas
65         @       B = Suma de las diagonales positivas
66
67         @ (Determinante por metodo de Sarruss)
68
69         @ Valor final del Determinante
70         MOV r5, #0
71
72         @ Suma de las diagonales negativas
73         LDR r2, [r1, #0]
74         LDR r3, [r1, #16]
75         LDR r4, [r1, #32]
76         MUL r2, r2, r3
77         MUL r2, r2, r4
78         ADD r5, r5, r2
79
```

- r_1 : Guarda el arreglo de numeros
- r_5 : Guarda el valor final de la determinante
- r_2 : Guarda el valor más 'arriba' de el vector

- r_3 : Guarda el valor del 'centro' de el vector
- r_4 : Guarda el valor más 'abajo' de el vector

La función luego toma dichos registros con el mísmo patrón para las diagonales

2.2. Segunda función

Se asume que la palabra más grande que se entregará a la hora de revisar si es un anagrama es de 8 letras diferentes, en cualquier otro caso tirará error.

Para poder hacer está función se parten creando tres subrutinas diferentes; .^nag", "Vecz Çont", cada una con una tarea diferente a la hora de realizar lo pedido. En el registro R1 se carga el largo 1 que correponde a la primera palabra, y luego se inicializa el largo de la palabra 2 en R2.

Luego de esto, saltamos a la función .ªnag", en la cual se inicializa un vector vacío en el registro R0. En este vector, cada posición de nuestro hexadecimal representa cuantas veces se repite el caracter en la palabra.

Una vez generado este registro hacemos una comparación los dos largos de las palabras entre R1 y R2. En el caso de que ambos sean iguales se hace un branch a la función .ªnagTrue", en cualquier otro caso se hace un branch a la función .ªnagfalse". En .ªnagfalse"simplemente terminamos nuestra función, pero en caso de que nos vayamos a .ªnagTrue"se inicia un ciclo FOR.

```
117
118 @ Segunda función (subrutinas) : Ver si dos Strings son anagramas
119 funciondos:
120         @ Carga el primer largo en memoria
121         LDR r1, =len1
122         LDR r1, [r1, #0]
123
124         @ Carga el segundo largo en memoria
125         LDR r2, =len2
126         LDR r2, [r2, #0]
127
128         @ Salta a la funcion anagrama
129         B anag
130
131         B CONTINUE
```

- r_1 : Guarda la longitud del primer string
- r_2 : Guarda la longitud del segundo string

La función luego da un salto a 'anag'

El registro R1 ahora se convierte en el índice para poder leer el primer string, y el R2 lo mantenemos ya que sabemos que los largos son iguales. Dentro del mismo ciclo usamos R3 para cargar el string 1 completo, y comparamos si es que el índice es igual al largo de la palabra. En caso de no serlo, cargamos en el registro R4 el n-ésimo caracter de la palabra. Avanzamos en uno el índice y desplazamos el vector a la izquierda (Esto se hace ya que, recordando, cada posición indica la repetición del caracter que estamos leyendo).

```
@ La funcion retorna true y sigue
anagtrue:
        @ Nuestro índice para leer el str1
        MOV r1, #0

        anagloop:
                @ Carga el primer string
                MOV r3, #0
                LDR r3, =str1

                @ Revisa si el indice es menor al largo del string
                CMP r1, r2
                BEQ anagdone

                @ Carga el siguiente caracter
                LDRB r4, [r3, r1]

                @ Avanza el índice
                ADD r1, r1, #1

                @ Desplaza el vector a la izquierda
                LSL r0, r0, #4

                @ Salta a la funcion vect
                B vect

                B anagloop
        anagdone:

        B CONTINUE
```

Aquí la función hace un ciclo que va seleccionando cada caracter de la primera palabra

Luego de esto hacemos un salto a la función "vect", en la cual el registro 3 se encarga de guardar el segundo string, y después de esto usamos el registro R6 para leer el segundo string.

Nuevamente entramos a otro ciclo, en el cual se va a comparar cada letra del primer string con cada letra del segundo, de igual manera que se hizo anteriormente en la función .ªnag". Cada vez que comparamos un caracter de cada una de las palabras saltamos a la función çont", la cual está encargada de verificar si ambos caracteres son iguales. En caso de ser iguales le sumamos 1 al vector y repetimos todo el ciclo.

```

@ Retorna 2 si los dos vectores son iguales
vect:
        @ Carga el segundo string
        MOV r3, #0
        LDR r3, =str2

        @ Nuestro índice para leer el str2
        MOV r6, #0

        vectloop:
                @ Carga el segundo string
                MOV r3, #0
                LDR r3, =str2

                @ Revisa si el indice es menor al largo del string
                CMP r6, r2
                BEQ vectdone

                @ Carga el siguiente caracter
                LDRB r5, [r3, r6]

                @ Avanza el índice
                ADD r6, r6, #1

                @ Salta a cont
                B cont

                B vectloop
        vectdone:

        B anagloop

```

Luego, se compara cada palabra de la primera con un ciclo anidado, así compara la cantidad de repeticiones por letra de los 'strings'

```
214 @ Cuenta la cantidad de veces que se repite un caracter
215 @ retorna un vector con las cantidades
216 cont:
217         @ Usamos el registro r0 como vector
218         @ cada posición de r0 indica la
219         @ repeticion de ese caracter
220         CMP r4, r5
221
222         @ Si los registros r4 y r5 son iguales
223         @ entonces va a la branch contadd
224         BEQ contadd
225
226         @ Se devuelve a la branch original
227         @ independiente del resultado
228         B vectloop
229
230 @ Suma al vector si es que los caracteres son iguales
231 contadd:
232         ADD r0, r0, #1
233         B vectloop
```

Finalmente, en 'cont' se verifica que los caracteres apuntados dentro del ciclo anidado sean iguales, si lo son, sumará al registro 'r_0', y luego se shifteará la posición de dicho registro cada vez que cambie la palabra que selecciona el primer ciclo, así formaremos el vector que nos indica la repetición de cada letra

De esta manera, al terminarse la función .ªnag"tendremos un vector con cada repetición de cara caracter de ambos strings, en caso de que la suma de cada una de las posiciones de ambos vectores sea igual, para ambos strings, significa que es un anagrama.

2.3. Tercera función

La función número 3 dice que se nos entregarán una serie de números que representan movimientos dentro de un plano cartesiano, donde los índices pares representan movimientos en la horizontal y los índices impares representan los movimientos en la vertical. Sabiendo eso, debemos interpretar las distintas instrucciones y calcular la distancia euclidiana del punto final al origen, pero truncada a la unidad.

Comenzamos cargando todo el arreglo de instrucciones en la memoria, junto con cargar número de movimientos que hay que realizar (Ambos datos nos son entregados como parte de los datos de entrada a la función)

```
235 @ Tercera (vectores) : Suma de vectores y càlculo de magnitud
236 funciontres:
237         @ Carga el arreglo en memoria
238         LDR r1, =nums
239
240         @ Carga el largo en memoria
241         LDR r2, =len1
242         LDR r2, [r2, #0]
243
244         @ Carga el primer elemento del arreglo
245         LDR r3, [r1, #0]
246
247         @ Indice del arreglo
248         MOV r4, #0
249
250         @ Coordenada eje X
251         MOV r5, #0
252         @ Coordenada eje Y
253         MOV r6, #0
254
255         @ Multiplica el largo por 4
256         LSL r2, r2, #2
```

- r_1 : Guarda el arreglo de numeros
- r_2 : Guarda el largo del arreglo
- r_3 : Carga el i-ésimo elemento del arreglo
- r_4 : El índice del arreglo

- r_5 : Coordenada del eje X (se asume posición inicial como 0)
- r_5 : Coordenada del eje Y (se asume posición inicial como 0)

El largo se multiplica por 4 ya que la matríz de memoria para numeros avanza de 4 en 4, decidimos que para ahorrarnos un registro donde haya que multiplicar para obtener la dirección, era mejor multiplicar el largo por 4 para avanzar el índice y que éste funcionase también como nuestra dirección de memoria

Luego de esto, usamos la implementación de FOR dado como materia de los ppts de la asignatura, dentro de la cual se le va sumando a la posición de X y a la posición de Y intercaladamente los valores que nos van pasando. Luego de esto, saltamos a una función que eleva cada función al cuadrado, y con esto podemos guardar en un registro diferente la raíz de la suma, obteniendo así la distancia entre ambos puntos.

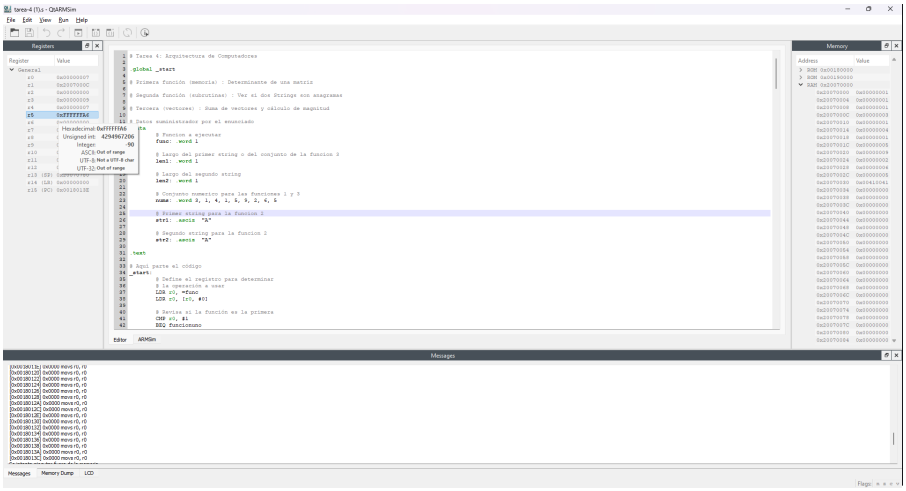
```
258         FOR:
259                 @ Primero la coordenada X
260
261                 @ Verifica que el indice
262                 @ sea menor al largo
263                 CMP r4, r2
264                 BEQ DONE
265
266                 @ Carga el i-esimo
267                 @ valor en memoria
268                 LDR r3, [r1, r4]
269
270                 @ Suma el movimiento a X
271                 ADD r5, r5, r3
272
273                 @ Avanza el indice
274                 ADD r4, r4, #4
275
276
277                 @ Segundo la coordenada Y
278
279                 @ Verifica que el indice
280                 @ sea menor al largo
281                 CMP r4, r2
282                 BEQ DONE
283
284                 @ Carga el i-esimo
285                 @ valor en memoria
286                 LDR r3, [r1, r4]
287
288                 @ Suma el movimiento a Y
289                 ADD r6, r6, r3
290
291                 @ Avanza el indice
292                 ADD r4, r4, #4
293
294                 B FOR
295
296         DONE:
```

Este primer ciclo recorre el arreglo y toma cada iteración par como una suma a la coordenada X. A su vez, cada iteración del ciclo impar lo toma como una suma a la coordenada Y. Esto nos permite acceder a las direcciones correctas sin tener que verificar que el índice sea un valor par o impar.

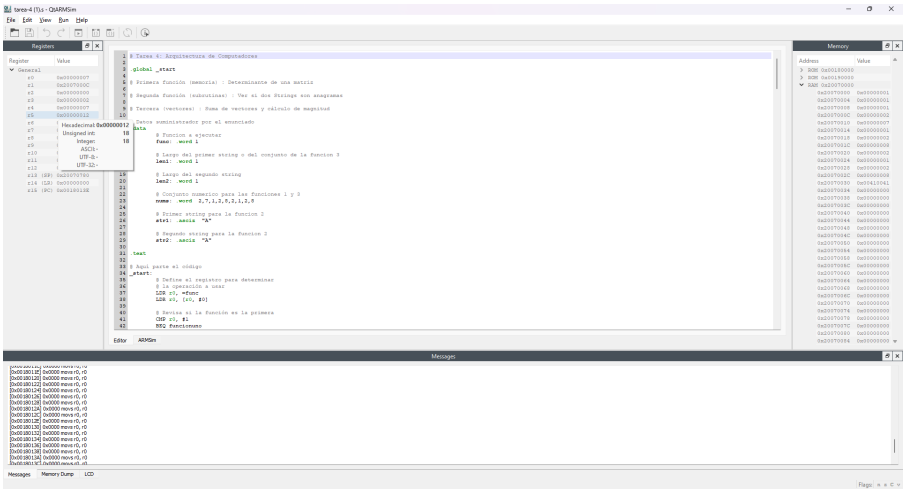
```
298         @ Eleva X al cuadrado
299         MUL r5, r5, r5
300
301         @ Eleva Y al cuadrado
302         MUL r6, r6, r6
303
304         @ Guarda la suma de X e Y
305         @ al cuadrado
306         MOV r2, #0
307         ADD r2, r5, r6
308
309         @ Inicia la raiz
310         MOV r3, #1
311
312         WHILE:
313                 @ Almacena la potencia de
314                 @ la raiz
315                 MOV r4, #0
316                 ADD r4, r4, r3
317                 MUL r4, r4, r4
318
319                 CMP r4, r2
320
321                 @ Si la raiz al cuadrado
322                 @ es mayor a la potencia
323                 @ entonces encontramos la
324                 @ raiz
325                 BGT EXIT
326
327                 @ Agrega 1 a la raiz y
328                 @ repite el loop
329                 ADD r3, #1
330
331                 B WHILE
332         EXIT:
333
334         @ Le resta 1 a la raiz ya que
335         @ queremos que trunque el valor
336         @ al entero mas cercano
337
338         @ Finalmente el valor de la magnitud
339         @ del vector queda en r3
340         SUB r3, r3, #1
```

Esta segunda parte nos permite saber la magnitud. El algoritmo basicamente toma el cuadrado de cada coordenada, los suma y luego entra a un ciclo que busca que número al cuadrado es mayor a nuestra magnitud al cuadrado, una vez se pase, se le resta 1 a dicha raíz, dejando en el registro 'r_3' el valor de nuestra magnitud

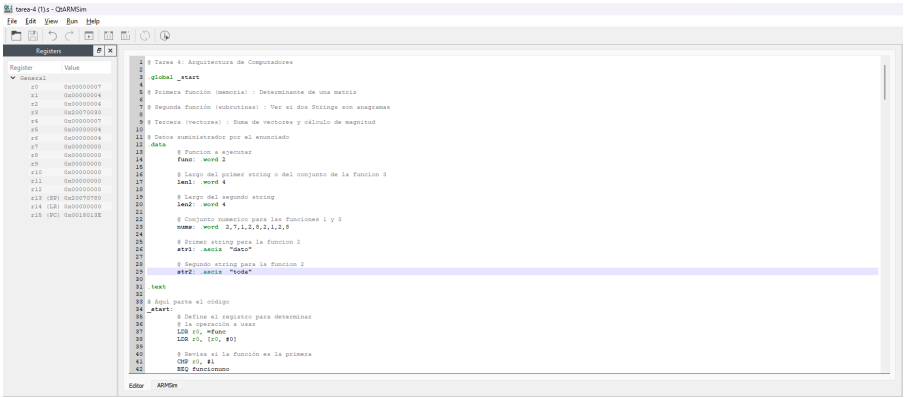
3. Resultados y análisis



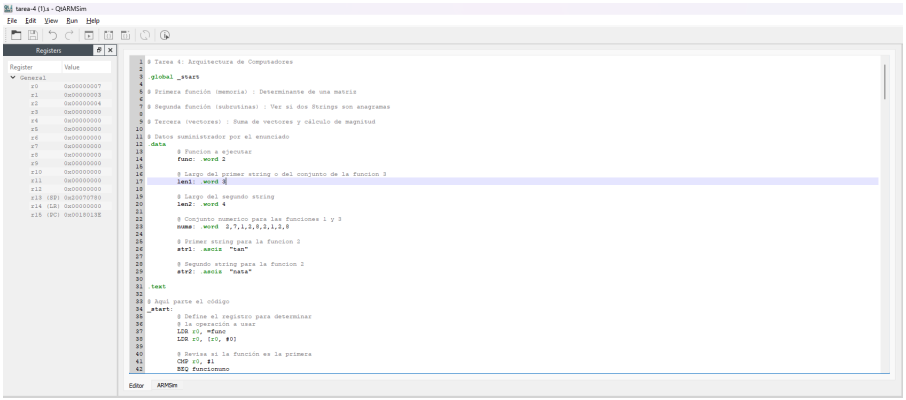
Caso de prueba 1



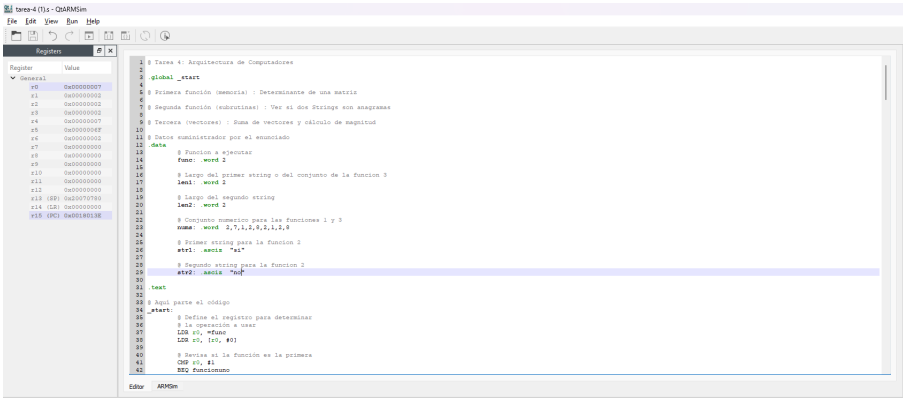
Caso de prueba 2



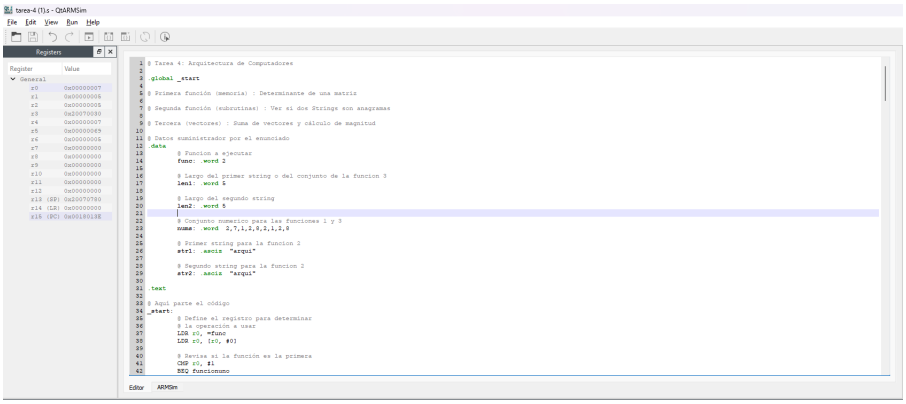
Caso de prueba 3



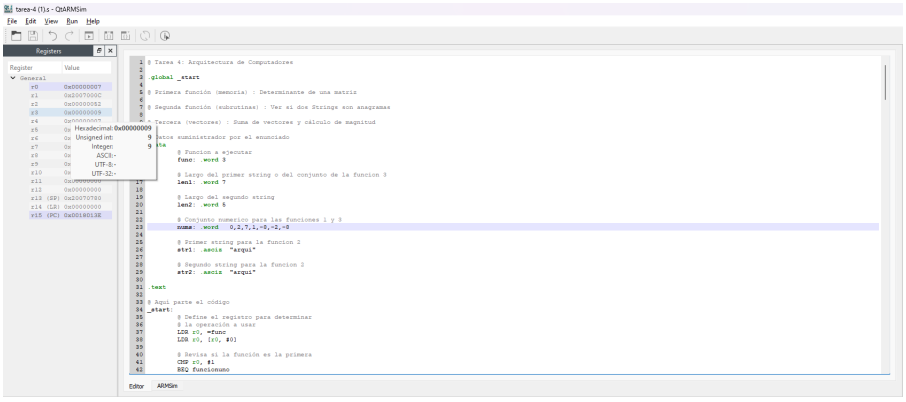
Caso de prueba 4



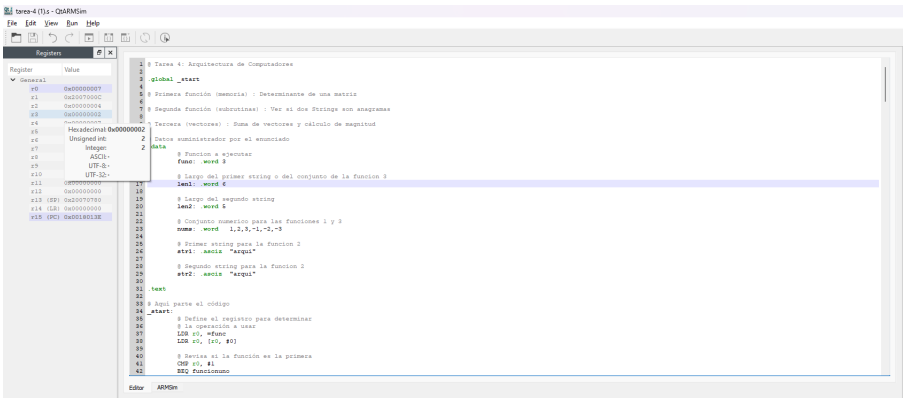
Caso de prueba 5



Caso de prueba 6



Caso de prueba 7



Caso de prueba 8

Prueba de la primera función mostrando por pantalla el resultado

4. Conclusiones

Pudimos apreciar que el lenguaje de descripción de hardware es una manera mucho más especifíca de las ramas de la programación que se habían visto previamente. Si bien esta puede resultar algo anticuada en el gran esquema de lenguajes de alto nivel como C o Java, creemos como grupo que es importante el conocer todo desde la base para poder tener un mayor entendimiento de nuestro objetivo a futuro con la programación.

No fue una tarea sencilla, la dificultad de aprender algo tan específico y la necesidad de aprender a utilizar instrucciones, memoria y otros desde la parte más básica de nuestro sistema realmente desafiaron nuestras habilidades, pero dentro de lo que se cumplió nos parece que se notan los frutos de un trabajo continuo y metódico a lo largo de la tarea y del curso en general, tomando esta culminación como un punto bastante fuerte para el ramo.

Si bien no pudimos terminar de manera completa todo lo pedido en la tarea, sentimos que el aprendizaje existió, y que tal vez con un poco más de tiempo podríamos haber solucionado esos pequeños errores que tuvimos en la tarea, sobretodo en la función 2.