ESTRUCTURA DE COMPUTADORES

Memoria Proyecto Ensamblador 88110

Pablo Agustín Myrick Ruiz
220247

I. Resumen diario

- 07/10/2024
  Antes de nada, leo el enunciado del proyecto y trato de comprender de manera general el propósito del conjunto de subrutinas que voy a implementar.
  Empiezo a implementar la subrutina *LongCad*, la cual termino ya que su complejidad no requiere mucho más tiempo. En total hoy habré invertido unas 3 horas.

- 10/10/2024
  Hoy comienzo a implementar la subrutina *BuscaCar*. Me ha costado un poco el tema de los parámetros 'from' y 'to' ya que no entendía muy bien en un principio su funcionamiento en la subrutina, pero tras leerlo detenidamente he conseguido comprenderlo. He estado unas 2 horas.

- 11/10/2024
  Continuo con la implementación de *BuscaCar*. Tras ayer entender bien dicha subrutina consigo acabarla sin mayor problema, realizando también varias pruebas para comprobar que funcione correctamente. He estado 3 horas más.

- 17/10/2024
  Comienzo con la lectura e implementación de *CoincidenCad*. No me ha costado mucho y creo que ha sido una subrutina bastante sencilla ya que solo he tardado 2 horas en total. La más sencilla hasta ahora junto con *LongCad*, a mi parecer.

- 21/10/2024
  Hoy empiezo con la implementación de *PoneBitA1*, tras la lectura de dicha subrutina en el enunciado comienzo a implementarla. Después de escribir varias líneas de código, no entiendo como puedo hacer que se ponga a 1 el bit que yo quiero, ya que en el campo de bits estos van de forma creciente, mientras que los bits de un byte van del 7 al 0. El tiempo empleado han sido 2 horas.

- 22/10/2024
  Continuo con *PoneBitA1*. He descubierto como puedo hacer que el bit que yo quiera se ponga a 1, básicamente la operación para saber el bit que tenemos que cambiar es [7 – (bit a cambiar – (bytes avanzar * 8))]. Esto lo podemos sumar en un registro junto con el número 32 (100000 en binario) y usarlo en una instrucción set.
  He conseguido implementar la subrutina y realizado un par de pruebas para comprobar su funcionamiento. He tardado 3 horas.

- 27/10/2024
  Empiezo a implementar la subrutina *LeeBit*, y en un principio no sabía cómo hacer para leer los bits. Tras pensar un rato y usando la misma operación que en *PoneBitA1*, utilizo la instrucción 'rot', para poner el bit que se quiere leer en la posición 0 del byte. Luego basta con hacer un 'and' entre dicho byte rotado y el número 1. He tardado unas 3 horas.

- 9/11/2024
  Empiezo con la lectura e implementación de *Comprime*. Hago los cuatro primeros puntos del enunciado de la subrutina, sin mayor complicación. 3 horas.

- 13/11/2024
  Comienzo con el primer bucle en el que lo más me cuesta es comprender *BuscaMax*, ya que no me había parado a mirar dicha subrutina y por lo demás no he tenido mayor complicación. Termino haciendo el sexto punto que es rellenar la cabecera del texto comprimido. 3 horas.

- 14/11/2024
  Termino la subrutina *Comprime* copiando los bytes de pilaZona3 al texto comprimido. Hago varias pruebas y compruebo que la subrutina funciona. He tardado otras 3 horas.

- 17/11/2024
  Leo y comprendo *Descomprime* y empiezo a implementarla. He seguido los pasos del enunciado y he tenido cierto problemas a la hora de avanzar los punteros que recorren 'com' y 'desc', los cuales no he conseguido corregir. 4 horas.

- 21/11/2024
  Termino de implementar *Descomprime*, sin mucha mayor dificultad. 3 horas.

- 22/11/2024
  He implementado *Verifica*, no ha sido demasiado complicado, pero algo estoy haciendo mal ya que tras pasar el corrector no me ha pasado las pruebas. 2 horas.

- 28/11/2024
  He conseguido solucionar un problema con *Comprime* ya que al ejecutar me salían errores de instrucción incorrecta y alineamiento a palabra. 2 horas.

- 9/12/2024
  En el taller de hoy con Manuel Nieto me ha advertido que en la subrutina *Comprime*, no estaba devolviendo en r29, el valor esperado. 2 horas.

- 16/12/2024
  Termino la implementación de Verifica, tenía algunos errores de alineamiento a palabra e instrucción incorrecta. 2 horas.

  Consultas:
  Hice una consulta por correo electrónico (pr_ensamblador@datsi.fi.upm.es) el día 28 de noviembre sobre un problema que tenía al ejecutar que se resolvió eficazmente tras intercambias un par de correos más al día siguiente.
  También en el taller que tuvo lugar el día 9 de diciembre, Manuel Nieto me resolvió la duda de por qué *Comprime* no pasaba ninguno de las pruebas, y la razón era que no dejaba la longitud del texto comprimido en r29.

II. Resumen de pruebas realizadas.

a) LongCad:

He probado esta subrutina con cadenas de una palabra ("Hola\0"), la cadena vacía ("\0") y el inicio de El Quijote ("En un lugar de la Mancha, de cuyo nombre...\0").

b) BuscaCar:

He realizado las pruebas:

- Buscar un carácter que no está dentro de los parámetros *from* y *to*.
- Buscar un carácter que sí está dentro de los parámetros *from* y *to*.
- Buscar un carácter que está en el límite (dir(ref[from) o dir(ref[to])) de los parámetros.

c) CoincidenCad:

He realizado las siguiente pruebas:

- Cadenas iguales.
- Una cadena cualquiera ("Hola\0") y la cadena vacía ("\0").
- Cadenas que comienzan igual pero luego difieren ("Buenas noches\0") y ("Buenos dias\0").

d) PoneBitA1:

He realizado las pruebas:

- Poner a 1 el bit 0 de un byte.
- Poner a 1 el bit 7 de un byte.
- Poner a 1 el bit 32 de un conjunto de 5 bytes.
- Poner a 1 el bit 63 de un conjunto de 8 bytes.

e) LeeBit:

He realizado las pruebas:

- Leer un bit intermedio de un byte.
- Leer el último bit de un byte.
- Leer un bit intermedio de un conjunto de 16 bytes.

f) Comprime:

He realizado las siguientes pruebas:

- Comprime una cadena vacía (“\0”)
- Comprime una cadena que no admite compresión (“123456789\0”) con N=4.
- Comprime la cadena (“tres tristes tigres comen trigo en un trigal, el primer tigre que...\0”) con N=4.
- Comprime el inicio de El Quijote (“en un lugar de la Mancha, de cuyo nombre no quiero acordarme, … y amigo de la caza.\0”) cadena de 885 caracteres más el terminador nulo, con N=4.

g) Descomprime:

He realizado las siguientes pruebas:

- Descomprime una cadena que no admite descompresión (data 0x07010014, 0x30000000, 0x34333231, 0x38373635 data 0x37383939, 0x33343536, 0x00303132) cadena resultante “01234567899876543210\0”.
- Descomprime (data 0x0b010044, 0x10102400, 0x74004000, 0x20736572 data 0x73697274, 0x04000274, 0x00016769, 0x6d6f6304 data 0x00046e65, 0x206f6704, 0x75206e65, 0x61060018 data 0x65202c6c, 0x7270206c, 0x72656d69, 0x2006000c data 0x2e657571, 0x00002e2e) cadena resultante “tres tristes tigres comen trigo en un trigal, el primer tigre que...\0”.

h) Verifica:

He realizado pruebas:

- Verifica una cadena que no admite compresión (“0123456789\0”).
- Verifica la cadena ("tres tristes tigres comen trigo en un trigal, el primer tigre que...\0”).

III. Observaciones finales y comentarios personales.

En este proyecto el mayor reto es entender el lenguaje ensamblador. Es algo fundamental ya que sin ello no vas a poder hacer nada, y parece algo obvio, pero creo que es tan importante porque las distintas instrucciones de manera individual parece que no pueden llevar a ninguna parte, o al menos eso me pasaba a mí al principio, (sumar, restar, or, and, etc.) pero si las entiendes en conjunto ahí es cuando empiezas a entender todo. La mayor dificultad para mi ha sido esa, entender ensamblador y a partir de ahí implementar las subrutinas te puede llevar más tiempo o menos, pero si no lo entiendes no vas a ninguna parte.

En total me ha supuesto un trabajo de 42 horas para implementar todas las subrutinas, más probablemente un par de horas extra para la memoria. Son bastantes horas y nunca había trabajado tantas horas en un mismo proyecto, pero aun así no se me ha hecho pesado. Aunque haya habido momentos de frustración por no entender algo, en general ha sido un proyecto que me ha gustado realizar.

Yo, Pablo Agustín Myrick Ruiz como miembro del grupo 220247 conozco y estoy de acuerdo con las siguientes normas sobre copia y fraude académico:

- el uso de cualquier material de otros grupos/alumnos, por cualquier motivo, para el desarrollo del proyecto está explícitamente prohibido.
- el departamento comprueba la copia entre todas las entregas de todos los grupos que intervienen en la convocatoria y se considera copia cuando aparezca esa condición en cualquiera de las entregas.
- el departamento realizará finalmente un análisis global de posibles casos de copia, incluyendo la inspección visual de implementaciones de los grupos cuyo código presenta un mayor índice de coincidencias. Igualmente, se comprobarán algunas implementaciones elegidas al azar.