Memoria

Proyecto de Ensamblador 2017-18

Estructura de Computadores

Autores:

X

Y

Usuario en el Sistema de Entrega:

x

Histórico

Comienzo:

Empezamos a trabajar en este proyecto el 30 de Octubre de 2018. Esto coincide con una de las correcciones adicionales. En este punto todavia no disponiamos de pruebas propias, y al entregarlo, no obtuvimos los resultados esperados.

Duración: Trabajamos juntos durante aproximadamente 2 horas.

Primer Hito (31 de Octubre 2018):

Llegado este hito, ya teniamos alguna prueba. Sin embargo, debido a nuestra falta de experiencia, tuvimos muchas dificultades a la hora de usar el emulador para comprobar que los resultados eran los correctos. En este hito solo tuvimos correcto la subrutina *ActualizaFiltro*.

Duración: Y trabajó durante 1 hora por su cuenta. Luego, intentamos arreglar el código juntos durante aproximadamente 2 horas.

Corrección Adicional:

Tras una pausa para mejorar nuestros conocimientos acerca del Emulador y el Ensamblador, retomamos el desarrollo. No tenemos constancia de cuantas horas cada uno de nosotros le dedicó, pero fue un proceso principalmente individual (cada uno aprendiendo por su cuenta). Para mejorar nuestro sistema de pruebas, decidí usar Macros para facilitar su creación. X se encargó de diseñar estas Macros y de actualizar el Código, y le llevó cerca de 1 hora.

A partir de este momento, empezamos a usar software externo para tener colaboración en tiempo real y llamada de voz. Esto nos ayudó a ser mucho mas eficientes y, en general, perder menos tiempo compartiendo nuestro código.

Arreglamos las subrutinas del Primer Hito, lo que nos llevó 2 horas a cada uno. X hizo la Subrutina de *SubMatriz*, y le llevó 2 horas. Y hizo *ValorPixel* y le llevó 1 hora.

Desafortunadamente, SubMatriz tenía muchos fallos y decidimos reescribirla. ValorPixel fallaba parcialmente.

Duración: 3 horas Y, 5 horas X.

Segundo Hito (16 de Noviembre de 2018) :

Le dedicamos más de 6 horas en conjunto a arreglar todos los errores que pudimos, y a reescribir SubMatriz. Durante el proceso, nos dimos cuenta de que en el futuro podríamos tener problemas si los registros se llegan a solapar. Además, nuestro código empezaba a ser muy denso. Decidimos diseñar nuestro propio "estándar" para este ensamblador. Este proceso nos llevó 1 hora en conjunto. Despues tuvimos que adaptar la mayor parte del código para que funcionara correctamente siguiendo estas directrices. Y se encargó personalmente de esto mientras que X terminaba de arreglar *SubMatriz*. Le llevó 1 hora, y X tardó 2.

Estándar "M88K-HK":

Registros

- r1 : Reservada para almacenar la direccion de retorno.
- r2 y r3 los tomamos como reservados, para cosas que sean absolutamente necesarias y que no se puedan modificar mas tarde. Me parece adecuado tener un par de registros exclusivamente dedicados a esto. Suelen ser usadas en MACROs.
- r4 a r10 : Uso general para variables
- r11 a r16 : Aritmetica
- r17, r18, r19 : Logica (Para comparaciones)
- r20 a r24 : Registros de los argumentos y parametros de las funciones.
- r25 a r28 : Auxiliares, se pueden usar en caso de que se necesiten registros adicionales.
- r29 : Para almacenar el valor de retorno.
- r30, r31 : Reservadas para Pila.

MACROS

- **cPILA()** crea el marco de pila.
- **dPILA()** deshace el marco de pila.
- **IF(ra, condicion, rb, eti)** actua como un if.
- **IFNOT(ra, condicion, rb, eti)** actua como un if negado.

Resultados: Completamos ValorPixel. Sin embargo, SubMatriz falló.

Duración: Y 8 horas, X 9 horas.

Correcciones Adicionales:

Durante estas correcciones intentamos llevar el proyecto lo más al día posible. Trabajamos juntos en esto mediante colaboración en tiempo real. Lo primero que hicimos fue arreglar errores. Debido a la velocidad o quizá los nervios, X cometió un error al escribir un registro en *SubMatriz*, y puso r20 en lugar de r25, lo que hacia que la subrutina fallase. Tardamos 1 hora en encontrar este error tan tonto. En nuestra experiencia, los errores más absurdos y pequeños son los más dificiles de identificar. Durante estas correcciones, le dedicamos aproximadamente 15 horas cada uno. Además, Y le dedicó 3 horas exclusivamente a encontrar el error que teníamos en FiltRec. Sin embargo, no consiguió identificar el error y decidimos reescribir la subrutina. X se encargó de esto mientras Y atendía otros compromisos. Esto le llevó a X 2 horas.

Perdímos una hora intentando encontrar un error con *FiltRec*. Al final resultó que nuestra prueba estaba mal, y no nuestra subrutina. Pasamos todas las pruebas.

Duración: Y 18 horas, X 17 horas.

Último Hito (Viernes 20 de Diciembre):

Ya que todas nuestras pruebas pasaban correctamente, dedicamos 2 horas de nuestro tiempo a darle un repaso a cada parte del proyecto para asegurarnos de que estaban hechos adecuadamente.

Duración: 2 horas.

Duración Total

X: 37 horas aproximadamente.

Y: 36 horas aproximadamente.

Se puede observar que la cantidad de horas dedicadas por cada uno de los integrantes del equipo es similar. Esto se debe a que la mayoría del tiempo hemos desarrollado el proyecto simultaneamente mediante colaboracion en tiempo real.

Visitas a Profesores

Solicitamos una tutoría un día, pero resultó imposible. Al final conseguimos encontrar nuestros errores, asi que no hubo ninguna visita a ningún profesor.

RESUMEN DEL JUEGO DE ENSAYO

En esta parte, se expondrán las pruebas usadas para comprobar cada subrutina. Están basadas en los diferentes casos expuestas en el documento en formato .pdf proporcionado en la página web de la asignatura, aunque hemos añadido más pruebas en la mayoría de las subrutinas para asegurarnos de un correcto funcionamiento antes de realizar las entregas.

Debido a ello, decidimos crear macros para realizar las llamadas a las subrutinas y la creación de las pruebas de una manera más rápida y sencilla, además de ordenar los distintos elementos de las pruebas (es decir, un bloque de imágenes, otro de filtros, etc). Por ello expondremos una prueba (que corresponderá con el primer caso de cada subrutina

proporcionado en la web con los mismos elementos) y las macros por cada respectiva subrutina, tal y como se pedía en el enunciado.

NFiltrados

```
_func1: MACRO(subr, arg1)
    PUSH(arg1)
    bsr subr
    POP(arg1)
ENDMACRO
PRUEBANF:   MACRO(dato, subr)
            LEA(r30, 65000)
            LOAD(r20, dato)
            _func1(nFiltrados, r20)
ENDMACRO
PPAL15: PRUEBANF(oper1, nFiltrados)
        stop
oper1:      data    4
```

Llama a *nFiltrados* pasándole un parámetro no nulo para iniciar la variable nF, que tiene un valor nulo.

Comp

```
_func2:     MACRO(subr, arg1, arg2)
            PUSH(arg2)
            PUSH(arg1)
            bsr subr
            POP(arg1)
            POP(arg2)
ENDMACRO
PRUEBA:     MACRO(dato1, dato2, subr)
            LEA(r30, 65000)
            LEA(r20, dato2)
            LEA(r21, dato1)
```

```
            _func2(subr, r21, r20)
ENDMACRO
PPAL8:  PRUEBA(IMAGEN1, IMAGEN2, Comp)
        stop
```

Llama a *Comp*, pasándole dos imágenes de 4x8 elementos que difieren en uno solo de ellos.

ActualizaFiltro

```
_func2:     MACRO(subr, arg1, arg2)
            PUSH(arg2)
            PUSH(arg1)
            bsr subr
            POP(arg1)
            POP(arg2)
ENDMACRO
PRUEBA:     MACRO(dato1, dato2, subr)
            LEA(r30, 65000)
            LEA(r20, dato2)
            LEA(r21, dato1)
            _func2(subr, r21, r20)
ENDMACRO
PPAL1:  PRUEBA(Dupla, FILTRO1, ActualizaFiltro)
        stop
```

Llama a *ActualizaFiltro* pasándole una matriz de filtro formada por nueve elementos distintos y una dupla que modifica el filtro dividiendo a la mitad cada uno de sus elementos.

ValorPixel

```
_func2:     MACRO(subr, arg1, arg2)
            PUSH(arg2)
            PUSH(arg1)
            bsr subr
            POP(arg1)
            POP(arg2)
ENDMACRO
PRUEBA:     MACRO(dato1, dato2, subr)
            LEA(r30, 65000)
            LEA(r20, dato2)
            LEA(r21, dato1)
```

```
            _func2(subr, r21, r20)
ENDMACRO
PPAL28: PRUEBA(SUBIMAGEN1, FILTRO8, ValorPixel)
        stop
```

Llama a *ValorPixel*, pasándole una subimagen nula excepto en su elemento central y un filtro identidad.

SubMatriz

```
_func4:     MACRO(subr, arg1, arg2, arg3, arg4)
            PUSH(arg4)
            PUSH(arg3)
            PUSH(arg2)
            PUSH(arg1)
            bsr subr
            POP(arg1)
            POP(arg2)
            POP(arg3)
            POP(arg4)
ENDMACRO
PRUEBASUBM: MACRO(img, subm, i, j)
            LEA(r30, 65000)
            LOAD(r20, j)    ; Se pasa por valor => LOAD
            LOAD(r21, i)    ; Se pasa por valor => LOAD
            LEA(r22, subm)  ; Se pasa por & => LEA
            LEA(r23, img)   ; Se pasa por & => LEA
            _func4(SubMatriz, r23, r22, r21, r20)
ENDMACRO
PPAL20: PRUEBASUBM(IMAGEN14, SUBIMAGEN, __1, __1)
        stop
```

Llama a *SubMatriz*, pasándole una imagen de 3x3 elementos de la que se ha de extraer la subimagen correspondiente al elemento central.

FilPixel

```
_func4:     MACRO(subr, arg1, arg2, arg3, arg4)
            PUSH(arg4)
            PUSH(arg3)
            PUSH(arg2)
            PUSH(arg1)
            bsr subr
            POP(arg1)
            POP(arg2)
            POP(arg3)
            POP(arg4)
```

```
ENDMACRO
PRUEBAFILPXL:   MACRO(img, i, j, MFiltro)
                LEA(r30, 65000)
                LEA(r20, MFiltro)   ; r20 <- &MFiltro
                LOAD(r21, j)        ; r21 <- j
                LOAD(r22, i)        ; r22 <- i
                LEA(r23, img)       ; r23 <- &Imagen
                _func4(FilPixel, r23, r22, r21, r20)
ENDMACRO
PPAL35: PRUEBAFILPXL(IMAGEN18, __2, __3, FILTRO8)
        stop
```

Llama a *FilPixel*, pasándole una imagen de 5x5 elementos y un filtro identidad que se aplica a un elemento del interior de la imagen.

Filtro

```
_func3: MACRO(subr, arg1, arg2, arg3)
    PUSH(arg3)
    PUSH(arg2)
    PUSH(arg1)
    bsr subr
    POP(arg1)
    POP(arg2)
    POP(arg3)
ENDMACRO
PRUEBAFILTRO:   MACRO(img, imFiltrada, MFiltro)
    LEA(r30, 65000)
```

```
    LEA(r20, MFiltro)    ; r20 <- &MFiltro
    LEA(r21, imFiltrada) ; r21 <- &imFiltrada
    LEA(r22, img)        ; r22 <- &img
    _func3(Filtro, r22, r21, r20)   ; /Filtro
ENDMACRO
PPAL40: PRUEBAFILTRO(IMAGEN21, FILTRADA1, FILTRO14)
        stop
```

Llama a *Filtro* pasándole una imagen no nula de 4x8 elementos y un filtro que multiplica por 4 cada elemento y le resta tres veces el valor del situado en la misma columna de la fila anterior. Algunos elementos alcanzan el valor máximo y otros el mínimo.

FiltRec

```
_func5:     MACRO(subr, arg1, arg2, arg3, arg4, arg5)
            PUSH(arg5)
            PUSH(arg4)
            PUSH(arg3)
            PUSH(arg2)
            PUSH(arg1)
            bsr subr
            POP(arg1)
            POP(arg2)
            POP(arg3)
            POP(arg4)
```

```
            POP(arg5)
ENDMACRO
PRUEBAFILTREC:  MACRO(ImagenIn, ImagenOut, MFiltro, ModMFiltro, NCambios)
    LEA(r30, 65000)
    LOAD(r20, NCambios) ; r20 <- NCambios
    LEA(r21, ModMFiltro); r21 <- &ModMFiltro
    LEA(r22, MFiltro)   ; r22 <- &MFiltro
    LEA(r23, ImagenOut) ; r23 <- &ImagenOut
    LEA(r24, ImagenIn)  ; r24 <- &ImagenIn
    _func5(FiltRec, r24, r23, r22, r21, r20)
ENDMACRO
PPAL43: PRUEBAFILTREC(IMAGEN24, FILTRADA4, FILTRO10, Dupla8, __40)
        Stop
```

Llama a *Filtro* pasándole una imagen no nula de 4x6 elementos y un filtro identidad, que devuelve la misma imagen recibida.

Observaciones Finales

Como práctica ha estado muy bien. Tuvimos varios problemas, y nos costó mucho empezar; pero una vez ya nos habiamos familiarizado con el entorn todo se hizo mucho mas ameno. Como comentario personal, hemos tenido muchos problemas con el emulador en Ubuntu 18.04, particularmente al intentar borrar comandos como “t 5000”. Si bien borraba el comando a efectos prácticos, visualmente nos añadía “ “ (espacios). Además, hemos tenido muchos errores tontos que nos han hecho perder muchísimo más tiempo del necesario. Finalmente, estamos muy contentos con nuestro código, y sobretodo con el estándar y las MACROs que hemos diseñado, ya que hacen el código más legible, y permiten un desarrollo más ágil y eficiente del proyecto.