MEMORIA DEL PROYECTO ES_INT.s DE ARQUITECTURA DE COMPUTADORES:

Alejandro Ayuso Expósito 190238
Hernán Calvo Aguiar 190090

El objetivo del proyecto es que nos familiaricemos con la realización de Entrada/Salida en un periférico mediante interrupciones.
Cabe mencionar que esta es la segunda vez que realizamos esta práctica, por lo que la gran parte del tiempo lo hemos invertido en corregir y realizar trazas del código ya escrito por nosotros el año pasado.
En cuanto al salvado de todos los registros en pila al entrar en una subrutina, sabemos que es subóptimo ya que hay registros que no necesariamente se tocan, lo hicimos en testeo y desarrollo se nos coló en la entrega final.

El proyecto se divide en la elaboración de las siguientes subrutinas:

- **INIT:** Subrutina que inicializa los dispositivos. Prepara las dos líneas por las que se van a recibir y transmitir caracteres y notificar las interrupciones.

  Al principio parecía que esta subrutina era simple y la tuvimos escrita bastante rápido, de lo que no nos dimos cuenta y fue un error bastante grande fue de añadir una entrada, la de la RTI, esto causo varios problemas para testear el código ya que siempre nos fallaba en lo mismo. Después de escribir un correo y que nos aclararan las dudas, logramos corregirlo.
  En escribir y corregir esta subrutina estuvimos alrededor de media hora ya que la teníamos escrita del proyecto del año pasado.
  La teníamos planteada de la siguiente manera:

```
Inicilizar lineas A y B con parámetros
  Init(){
   {
    Dar acceso a reg. modo 1
    8 bits por caracter
    Desactivar el eco
    Modo full duplex
    Velocidad = 38400 bps
    Buffer 2000 B
  }
  Inicializaciones globales{
    Conjunto de veloc. 1
    0x40 -> registro de vectores de interrupcion
    Activar bits 1 y 5 para interr. RX
    Actualiza copia del IMR
    RTI -> primer vector de interrupcion ($100)
  }
   //Inicializacion de buferes transmision y recepcion
   ini_bufs(); // viene en bib_aux.s iniciañiza punteros de
   inserción y extraccION
    Inicializacion de los contadores a cero y reseteo de A0
 RTS
```

- **SCAN:** Subrutina en la que se lee de un dispositivo. Devuelve el bloque de caracteres previamente escrito por una de las dos líneas.

  Al igual que con INIT, la teníamos escrita del año pasado, el problema era que el año pasado no nos paso muchas pruebas por lo que nos pusimos a arreglarla, el principal problema que tuvimos con esta subrutina y en general con todas las demás era como se trataba el marco de pila, so comprendíamos muy bien la instrucción MOVEM.L y esto nos generaba varios problemas al extraer los datos, para arreglar este problema sustituimos el MOVEM.L por varios MOVE.L de los registros que íbamos a utilizar en la subrutina.

```
*MOVEM.L        A0-A5/D1-D5,-(A6)
MOVE.L          D1,-4(A6)
MOVE.L          D2,-8(A6)
MOVE.L          D3,-12(A6)
MOVE.L          D4,-16(A6)
MOVE.L          D5,-20(A6)
MOVE.L          A0,-24(A6)
MOVE.L          A1,-28(A6)
MOVE.L          A2,-32(A6)
MOVE.L          A3,-36(A6)
MOVE.L          A4,-40(A6)
MOVE.L          A5,-44(A6)
```

Ilustración 1: solución al problema del marco de pila

  Tras arreglar este problema, procedimos a probar la subrutina, y vimos que fallaba cuando se le pasaba un descriptor invalido, para simular esto, escribíamos directamente en la dirección de memoria correspondiente al descriptor un numero distinto de 0 o 1, después de trazar la subrutina, adelantamos un MOVE donde poníamos #FFFFFFFF en un registro que cuyo contenido posteriormente lo pasamos a D0 y añadimos un par de etiquetas para que D0 no se sobrescribiera.

```
**ERROR EN CARACTER**
MOVE.L          D4,D0
BRA             FN_SCER
```

Ilustración 2: Etiqueta nueva que salva D0

  Con estos mismos cambios también arreglamos el resto de fallos que tenia la subrutina. El tiempo invertido en trazar la subrutina varias veces y corregir los errores que tenía no fue bastante, tardamos unas cinco horas.

La teníamos planteada de la siguiente manera:

```
int SCAN (Buffer b,int descriptor ,int tamanyo){
    int num_char_leido = -1; // caso descriptor incorrecto
    int c = 0;
    int car = 0;
    boolean vacio = false;
        switch(descriptor){
        case 0{
            while(tamanyo!=c && !vacio){
                car = LEECAR(descriptor);   // en este caso
descriptor == num para seleccionar descriptor de leecar
                if(car!=-1){
                    b.add(char);
                    c++;
                    num_char_leido = c;
                } else {
                    vacio = true;
                }
            }
        }
        case 1{
            while(tamanyo!=c && !vacio){
                car = LEECAR(descriptor);   // en este caso
descriptor == num para seleccionar descriptor de leecar
                if(car!=-1){
                    b.add(char);
                    c++;
                    num_char_leido = c;
                } else {
                    vacio = true;
```

```
                }
            }
        }
    }
    return num_char_leido;
}
```

- **PRINT:** Subrutina en la que se escribe en un dispositivo. Manda la escritura de el bloque de caracteres previamente escrito en una de las dos líneas.
  Esta subrutina, también la teníamos del año pasado, aunque también nos tocó trazarla y corregirla. Hicimos el mismo cambio del marco de pila que en SCAN por el mismo motivo.

```
LINK A6,#-48
*MOVEM.L        A0-A5/D1-D5,-(A6)
MOVE.L          D1,-4(A6)
MOVE.L          D2,-8(A6)
MOVE.L          D3,-12(A6)
MOVE.L          D4,-16(A6)
MOVE.L          D5,-20(A6)
MOVE.L          D6,-24(A6)
MOVE.L          A0,-28(A6)
MOVE.L          A1,-32(A6)
MOVE.L          A2,-36(A6)
MOVE.L          A3,-40(A6)
MOVE.L          A4,-44(A6)
MOVE.L          A5,-48(A6)
```

Ilustración 3: Solucion marco de pila

Después de comprobar que este cambio funcionaba y que ahora sí que se accedía bien a la pila procedimos a seguir trazando y corregir el primer error que aparecía en los resultados de las correcciones, el cual era el mismo que en SCAN, que no se devolvía #FFFFFFFF en D0 cuando de proporcionaba un descriptor erróneo, al tener el mismo problema, hicimos lo mismo para corregirlo, añadir una etiqueta que salvaba a D0 de sobrescribirse.

```
ERROR_PR:
                MOVE.L          #$ffffffff,D0
                JMP             P_FER
```

Ilustración 4: Solucion a etiqueta errónea

Luego corregimos el siguiente error que tenia que ver con el tamaño y las comprobaciones, si el tamaño del bloque que se enviaba a escribir era 0, no obteníamos el resultado esperado, esto lo arreglamos cambiando el orden de las comprobaciones y dejando solo las necesarias para saber en qué línea se va a escribir y dejando la comprobación del tamaño en el comienzo del bucle

```
CMP.W     #0,D1
BEQ       BUCLE_PA
CMP.W     #1,D1
BEQ       BUCLE_PB
```

Ilustración 5:comparaciones

```
BUCLE_PA:
          CMP.L     D3,D2
          BEQ       A_SET
```

Ilustración 6: comprobación tamaño (línea A)

Y a continuación procedimos a solucionar los problemas relacionados con la escritura y las interrupciones, esto fue lo mas costoso de corregir ya que tuvimos que reescribir el bucle y separarlo en dos uno para la línea A y otro para la B, con esto se nos arreglo el problema de la escritura en memoria pero no llegaba a escribir en ninguna de las dos líneas por como tratábamos el IMR, después de añadir un registro en el cual se guardaba el bit que íbamos a activar e introducirlo en IMR e IMRDUP conseguimos solucionar el problema de la escritura y de las interrupciones. Para poder pasar estas pruebas introducíamos manualmente en memoria algunos valores a demás de usar unas pruebas a parte

```
OR.B      #%00000001,D4
MOVE.B    D4,IMRDUP
MOVE.B    D4,IMR
BRA       P_FIN
```

Ilustración 7: Correcion IMR

La corrección y la traza de esta subrutina fue lamas larga y tediosa, en ella ocupamos la mayor parte del tiempo dedicado a este proyecto tardamos unas 9 horas (varios días trabajando por la tarde).

La subrutina la planteamos de la siguiente manera:

```
      Print (buffer, descriptor, tamaño){
  guardarPila();
  int contador;
//descriptor ilegal
  if (descriptor != 0 || descriptor != 1){
    D0 = FFFFFFFF;

  } else {
// Print por linea B
    if (descriptor == 0) {
      D0 = 2;
      while (contador != tamaño) {
        buffer ++;
        ESCCAR(D0,buffer)
        if(D0 == FFFFFFFF) {
          break;
        }
        contador ++;
      }
      if (!buffer vacio){
        bit0 IMRDUP -> 1
        actualizar IMR
      }
    }
// Print por linea B
      else {
      D0 = 3;
      while (contador != tamaño) {
        buffer ++;
        ESCCAR(D0,buffer)
        if(D0 == FFFFFFFF) {
          break;
        }
        contador ++;
      }
```

```
      if (!buffer vacio){
        bit4 IMRDUP -> 1
        actualizar IMR
      }
    }
  }
  contador -> D0;
  return D0;
  restablecerPila();
  RTS
}
```

- **RTI:** Subrutina que se ocupa del tratamiento de las interrupciones.

  RTI trata los casos de interrupción en ambas líneas para la recepción y transferencia de caracteres. Nuestros principales errores fueron dos y tardamos poco en resolverlos gracias a los profesores que atienden el correo de dudas. El primero era que nuestra subrutina no era concurrente, lo que significa que en vez de comprobar si había más interrupciones antes de salir de RTI, salía y volvía a entrar en el antes del fetch de la siguiente instrucción. Algo que no afecta al funcionamiento pero si lo reduce considerablemente. El segundo error fue realizar un AND entre dos registros y comprobar en el que no guardaba el resultado de la operación. Ambos fallos muy triviales.

Pseudocódigo:

```
RTI(){
    guardarPila();
    int = IMRDUP && ISR;
    int resultado;
    // %variable% simboliza que se trata el dato como una tira
de bits
    // cada caso del switch comprueba que el bit asociado al
número es 1
    switch(%int%){
        // caso de tranferencia en A
        case 0 {
            //ejecuramos la lectura
            resultado = LEECAR(2);
            if(resultado == -1){
                IMRDUP = %11101111 && 0; // % simboliza el
comienzo de un byte en binario
            } else {
            TBA.addFirst(resultado);
            }
            RTI();
        }
        // caso de recepción en A
        case 1{
            resultado = RBA.getChar();
            ESCCAR(resultado,0);    //no se pasan por pila
pero es pseudo, imaginemos que se entiende.
            RTI();
        }
        // caso de tranferencia en B
        case 4{
            int resultado;
            //ejecuramos la lectura
```

```
            resultado = LEECAR(3);
            if(resultado == -1){
                IMRDUP = %11101111 && 0; // % simboliza el
comienzo de un byte en binario
            } else {
            TBB.addFirst(resultado);
            }
            RTI();
        }
        // caso de recepción en B
        case 5{
            resultado = RBB.getChar();
            ESCCAR(resultado,3);    //no se pasan por pila
pero es pseudo, imaginemos que se entiende.
            RTI();
        }
    }
    restablecerPila();
    RTE
}
```

En resolver y detectar estos dos errores nos llevó unas 3-4h de tiempo de trabajo (gracias otra vez al correo de dudas).

- **Tests:**

Realizamos pruebas manuales para comprobar los casos en los que debería de fallar.

El resto de casos los hicimos trazando manualmente e introduciendo a mano caracteres en las líneas con la prueba dada en el manual.

*** Manejadores de excepciones**

```
INICIO: MOVE.L #BUS_ERROR,8 * Bus error handler
        MOVE.L #ADDRESS_ER,12 * Address error handler
        MOVE.L #ILLEGAL_IN,16 * Illegal instruction handler
        MOVE.L #PRIV_VIOLT,32 * Privilege violation handler
        MOVE.L #ILLEGAL_IN,40 * Illegal instruction handler
        MOVE.L #ILLEGAL_IN,44 * Illegal instruction handler
        BSR INIT
        MOVE.W #$2000,SR * Permite interrupciones


BUCPR:  MOVE.W #TAMBS,PARTAM * Inicializa par´ametro de tama~no
        MOVE.L #BUFFER,PARDIR * Par´ametro BUFFER = comienzo del buffer
* Prueba 1:
        *Introduce en scan un parámetro incorrecto en el descriptor
*D0 = ffffffff
PRUEBA1:MOVE.W PARTAM,-(A7)     * Tama~no de bloque
        MOVE.W #3,-(A7)         * Descriptor no valido
        MOVE.L PARDIR,-(A7)     * Direcci´on de lectura
        BSR SCAN                * Llamamos a scan
* Prueba 1:
        * Tamaño = 0 con descriptor correcto
        * D0= 00000000
PRUEBA2:MOVE.W #0,-(A7)     * Tama~no de bloque
        MOVE.W #1,-(A7)         * Linea B
```

```
    MOVE.L PARDIR,-(A7)     * Direcci´on de lectura
    BSR SCAN                * Llamamos a scan
SALIR: BRA BUCPR
BUS_ERROR: BREAK * Bus error handler
    NOP
ADDRESS_ER: BREAK * Address error handler
    NOP
ILLEGAL_IN: BREAK * Illegal instruction handler
    NOP
PRIV_VIOLT: BREAK * Privilege violation handler
    NOP
```