SEDG

Carpeta Montero

Apuntes y exámenes ETSIT UPM

Si alguna vez estos apuntes te sirvieron de ayuda, piensa que tus apuntes pueden ayudar a muchas otras personas.

Comparte tus apuntes en simplyjarod.com

SEDG
Teoría

TEMA 1: EL SISTEMA MICROPROCESADOR

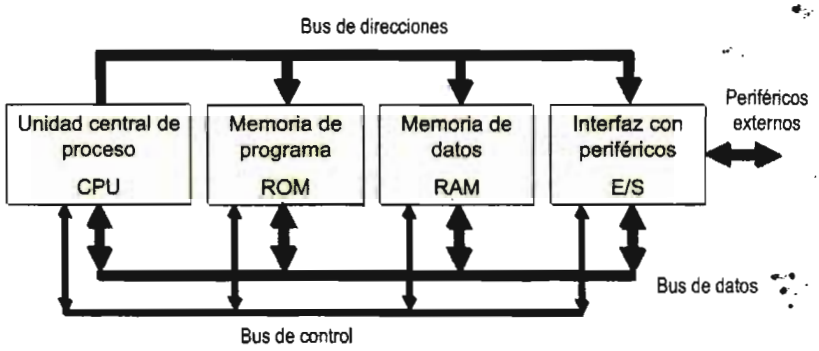
1.1 Elementos de un sistema microprocesador
1.2 El mapa de memoria
1.3 Introducción al microcontrolador ColdFire MCF5272

1.1 Elementos de un sistema microprocesador

De manera muy básica son:

- Microprocesador o unidad central de proceso (CPU). (En FDOR: UCP) Controla el sistema y ejecuta las instrucciones del código.
- Líneas de conexión: bus de datos, bus de direcciones y bus de control.
- Memoria central. Puede estar formada por dos tipos:
  - Memoria de programa (sólo lectura) que almacena instrucciones y datos invariables.
  - Memoria de datos (lectura y escritura). Almacena variables.
- Interfaces de Entrada/Salida (E/S). Comunican la CPU con los periféricos (impresora, disco, teclado...)

La estructura funcional de un sistema microprocesador es:



Notas:

- El valor de los bits del bus de direcciones indican la posición dentro de la memoria o interfaz de E/S en la que escribir o desde la que leer.

- Los datos que se van a leer o escribir en esta posición se ponen en el bus de datos.

- Mediante el bus de control, la CPU da órdenes a las distintas partes del sistema.
Por ejemplo: cuando la CPU quiere escribir en la memoria de datos, una vez ha puesto la dirección de memoria adecuada en el bus de direcciones y el dato que se desea escribir en el bus de datos, activa la señal de escritura y la hace llegar hasta el chip de memoria correspondiente. Una de las líneas del bus de control está reservada para esa señal.
Otro ejemplo: si la controladora de una impresora (que es un interfaz de E/S) genera una interrupción porque se ha quedado sin papel, se lo comunica a la CPU a través de una línea del bus de control destinada a la petición de interrupción.

Vemos ahora las características más importantes de cada elemento del sistema:

1.1.1 Microprocesador o CPU

Controla el sistema y ejecuta las instrucciones. Se caracteriza por:

la longitud del mayor dato que puede procesar coincide con el tamaño del bus de datos

- Longitud del dato procesado. Por ejemplo: 32 bits.

coincide con el tamaño del bus de direcciones

- Longitud de dirección y capacidad de direccionamiento. Por ejemplo: con una longitud de dirección de 16 bits se consigue una capacidad de direccionamiento de $2^{16} = 2^6 x 2^{10} = 64x1K$ = 64 Kbytes (porque las memorias se direccionan por bytes)

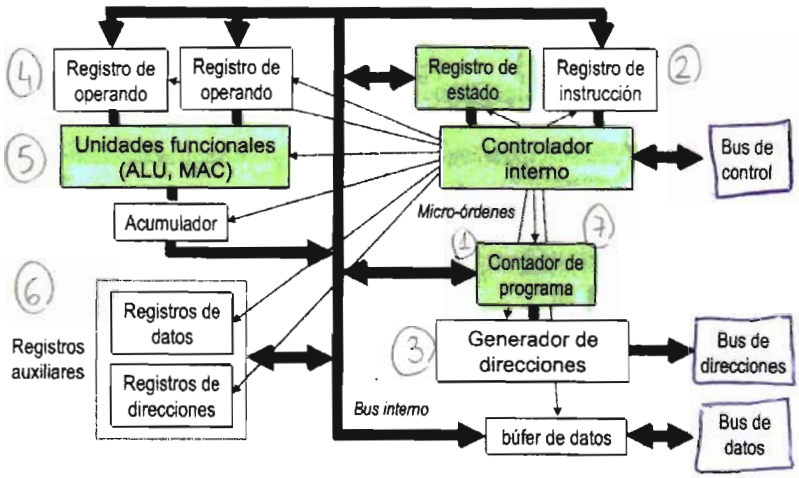
- Juego de instrucciones

-Registros internos: registros de datos, de direcciones, acumulador...

cada instrucción requiere un cierto número de ciclos de reloj para ejecutarse

- Velocidad de ejecución. Se puede medir de varias formas. Por ejemplo: Frecuencia de reloj (MHz), Millones de Instrucciones Por Segundo (MIPS).

El diagrama de bloques interno de la CPU es:



Los elementos más importantes son:

CP en FDOR
PC en SEDG

- Contador de programa: es un registro que contiene la dirección de la instrucción que se va a ejecutar. Cada vez que se termina de ejecutar una instrucción pasa a contener la dirección de la siguiente.

RE en FDOR
SR en SEDG

- Registro de estado: contiene información del estado del sistema. Por ejemplo: si las interrupciones están permitidas o no, si el resultado de la última operación ha sido cero...

Secuenciador en el tema de Micro-Símplez de FDOR

- Controlador interno: interpreta la instrucción que se va a ejecutar teniendo en cuenta el valor del registro de estado (por ejemplo: una instrucción de salto condicionada a que el resultado de la anterior instrucción haya sido cero, como BZ en FDOR, se ejecutara diferente dependiendo del valor del bit Z dentro del registro de estado, que indica si el último resultado ha sido cero). Conoce las micro-órdenes que debe activar y en qué momento de la ejecución de la instrucción debe activarlas.

UAL en FDOR
ALU en SEDG

- Unidad funcional: realiza operaciones. Por ejemplo: la Unidad Aritmético Lógica.

1.1.2. Líneas de conexión (buses)

Permiten la conexión de los distintos componentes del sistema microprocesador. Cada bus tiene un cierto número de bits.

° Conversión entre sistemas

Decimal → Hexadecimal

Dividir por 16.

Hexadecimal → Decimal

Numeración en base 16.

Hexadecimal → Binario

Grupos de 4 bits

★ **Buses:**

Bus = conjunto de líneas al que acceden dos o más dispositivos

A_0 / D_0 → bit menor peso

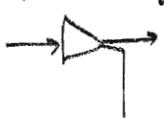
$A_7 \dots A_0$
(MSB) (LSB)

Unidireccional (direcciones)
Bidireccional (datos y control)

★ **Búfers:** (BUS DRIVERS)

- Aportan corriente al bus para incrementar su FANOUT
- Permiten añadir dispositivos sin cargar el bus

★ **Búfers tri-estado:** 0, 1, Z (alta impedancia) → evitan conflictos de acceso



- Unidireccional
- Bidireccional

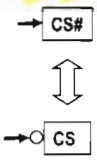
1.1.3 Memoria central

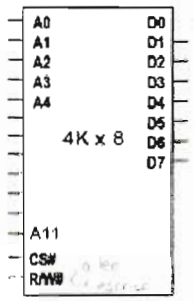
Está compuesta normalmente de dos tipos de memorias:

- Memoria de datos
  - Lectura/Escritua (RAM: Random Access Memory)
  - Contenido volátil (se pierde al desconectar la alimentación)
  - Terminales: bits del bus de direcciones, bits del bus de datos (bidireccional), habilitación o "Chip Select" (CS), y Read/Write (R/W)

  Por ejemplo: Memoria de 4K x 8.
  12 bits de direcciones $\Rightarrow 2^{12}= 2^2 \times 2^{10}= 4 \times 1K= 4$ K
  8 bits de datos
  4 Kbytes de memoria

En esta asignatura son equivalentes (indica señal activa a nivel bajo)





- Memoria de programa:
  - Sólo lectura (ROM: Read Only Memory)
  - Contenido permanente (permanece al desconectar la alimentación)
  - Terminales: bits del bus de direcciones, bits del bus de datos (sólo salida porque no se puede escribir en ella), habilitación o "Chip Select" (CS)

  El esquema es igual sólo que sin terminal R/W#

1.1.4 Interfaces de E/S

- Conexión con los periféricos

¡OJO! Es distinto a como se hace en FDOR

- A los registros de estos interfaces se les asignan posiciones del espacio de direccionamiento ⇒ ¡¡Se accede a ellos como si fuesen posiciones de memoria!!

- Ejemplo: un controlador de disquetes

1.2 El mapa de memoria

¡OJO! Esas direcciones no son sólo para memoria, también para interfaces de E/S

- Es la distribución del margen de direccionamiento (N bits del bus de direcciones⇒2^N direcciones posibles del microprocesador)

- El margen de direccionamiento es normalmente más grande que el número de direcciones que se necesitan para la memoria y los interfaces de E/S ⇒ Hay rangos de direcciones sin utilizar.

- Se puede hacer:
  - ♦ Mapa funcional o software (programa, datos, tablas, E/S ...)
  - ♦ Mapa físico o hardware (pastilla RAM, pastilla ROM, dispositivo de E/S ...)

Por ejemplo:

MAPA FUNCIONAL (SW)	MAPA FISICO (HW)	DIRECCIONES Decimal	Hex	Binario
Tablas y datos fijos	Pastilla 1: ROM 1Kx8	00000 01023	0000 03FF	0000 0000 0000 0000 0000 0011 1111 1111
Programa	Pastilla 2: ROM 1Kx8	01024 02047	0400 07FF	0000 0100 0000 0000 0000 0111 1111 1111
(Zona no usada)	----	02048 40959	0800 9FFF	0000 1000 0000 0000 1001 1111 1111 1111
Entrada/salida	Pastillas 5 a 8 (E/S)	40960 45055	A000 AFFF	1010 0000 0000 0000 1010 1111 1111 1111
(Zona no usada)	----	45056 57343	B000 DFFF	1011 0000 0000 0000 1101 1111 1111 1111
Variables y pila	Pastilla 3: RAM 4Kx8	57344 61439	E000 EFFF	1110 0000 0000 0000 1110 1111 1111 1111
Datos temporales	Pastilla 4: RAM 4Kx8	61440 65535	F000 FFFF	1111 0000 0000 0000 1111 1111 1111 1111

- En cada rango de direcciones del mapa:
  - ♦ Los bits superiores del bus de direcciones habilitan el chip
  - ♦ Los bits inferiores del bus de direcciones acceden al chip

- Se necesita que las pastillas de memoria o los interfaces de E/S sólo se activen cuando se quiera acceder a ellos. Hay que generar las señales de CS (Chip-Select) a partir de los bits del bus de direcciones. Hay dos posibilidades:

  - ♦ Por software se configuran los registros de un dispositivo que genera las señales de Chip-Select.
  - ♦ Con circuitos combinacionales: multiplexores, puertas...

1.3 Introducción al microcontrolador ColdFire MCF5272

- Microcontrolador: microprocesador (CPU) + periféricos + memoria, integrado en un solo chip

- Características de la CPU del MCF5272
  - Frecuencia de reloj: 66 MHz (en procesador y bus)
  - 63 MIPS
  - Datos de 8, 16 y 32 bits ⇒ bus de datos de 32 bits
  - Direcciones de 16 y 32 bits ⇒ bus de direcciones de 32 bits
  - Unidades funcionales: ALU y MAC

A parte de estas se podrán añadir otras externamente

- Memorias internas
  - 4 KB de RAM
  - 16 KB de ROM
  - Memoria caché (memoria oculta en FDOR) de 1 KB

- Incluye interfaces de E/S y otros módulos

TEMA 2:PROGRAMACIÓN DE LA FAMILIA COLDFIRE

2.1 Programación en ensamblador

Componentes de un programa

- Instrucciones: en el ensamblado se transforman en código máquina. Son de la forma:
  OPCODE OP1, OP2

  - OPCODE: código de operación de la instrucción. Puede tener un sufijo que represente el tamaño de los operandos
    - .B = 1 Byte = 8 bits
    - .W = 2 Bytes = 16 bits
    - .L = 4 Bytes = 32 bits
  - OP1 y OP2: operandos fuente y destino

Ejemplos:

```
MOVE.W 9300,D3
MOVE.B ETIQUETA,D4
```

- Etiquetas: representan una dirección de memoria o una constante y el proceso de ensamblado las convierte en ello.
- Comentarios: el proceso de ensamblado los ignora
- Directivas: son órdenes para el programa ensamblador.

Directivas del ensamblador

- `ORG expresión`: describe la situación de memoria para el código que sigue, para situar las instrucciones o variables en la posición que se desee dentro del margen de direccionamiento.
- `EXTERNAL lista`: las etiquetas de la lista (que se usan en el fichero) en el que esté esta directiva, están definidas en otros ficheros.
- `GLOBAL lista`: las etiquetas de la lista (que se definen en este fichero) se pueden usar en otros ficheros.

Es igual en FDOR

- `Etiqueta EQU expresión`: para definir constantes. Asigna a la etiqueta el valor de expresión y ¡¡*no ocupa espacio en memoria!!*

Es RES en FDOR

- `Etiqueta DS.X expresión`: Reserva un número de posiciones consecutivas en memoria = expresión. El tamaño de cada posición reservada es .X (.B, .W, .L)

Es DATA en FDOR

- `Etiqueta DC.X expresión`: Reserva una posición de memoria del tamaño q indica .X y le asigna el valor inicial de expresión.

Ejemplos:

```
VARIABLE        DC.L    $33
ABCD            DC.B    'A','B','C','D'
CDEFGH          DC.B    "CDEF","GH"
```

- END: indica el final del programa y es obligatoria.

Nota: es muy importante distinguir entre EQU, DC, y DS.

- EQU no ocupa memoria mientras que DC y DS sí.
- DS sólo reserva memoria, no asigna valores. DC asigna valores iniciales.
- Las etiquetas que acompañan a instrucciones representan siempre direcciones de memoria
- Resumiendo: todas las etiquetas representan direcciones de memoria excepto las que van al lado de un EQU

Si tenemos:

```
CIEN            EQU     100
DATO1           DC.B    $65
VARIABLE1       DS.W    2
...
INSTRUC1        MOVE.B  D1,D3
```

Los valores de las etiquetas serán:

CIEN= 100 (no es una dirección de memoria)

DATO1= valor de la dirección de memoria donde se ha almacenado el $65

VARIABLE1= valor de la dirección de memoria a partir de la que se han reservado dos palabras de 16 bits (.W)

INSTRUC1= valor de la dirección de memoria a partir de la que está almacenada la instrucción MOVE.B D1,D3

2.2 El modelo de programación del ColdFire

2.2.1 Registros de la CPU accesibles desde programa

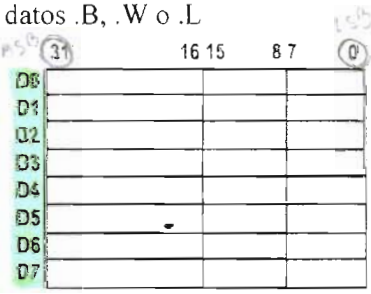
Esto quiere decir que se empiezan llenando desde el bit 0.

Los datos se alinean con el registro con el bit menos significativo (bit 0). Si se lleva al registro un dato .B ocupa en el mismo del bit 0 al bit 7. Si es .W del bit 0 al bit 15, y si es .L del bit 0 al bit 31.

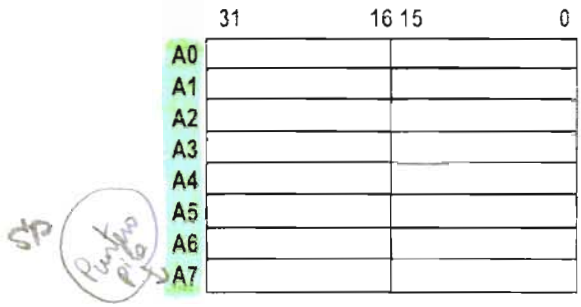
8 registros de datos (*D0-D7*)

¡OJO! Con .B o W los bits más significativos no cambian

- 32 bits
- Se pueden usar para guardar datos .B, .W o .L



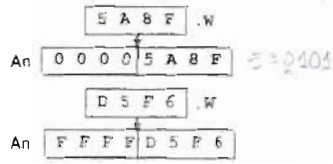
8 registros de direcciones (A0-A7)



No se usan direcciones .B

- 32 bits
- Sirven para guardar *direcciones* .W o .L
- Para cargar una dirección en un registro de direcciones se usa la instrucción MOVEA (no se usa MOVE).
- Cuando se manejan direcciones .W hay que tener en cuenta la extensión de signo. La realiza automáticamente la instrucción MOVEA

A diferencia de los registros de datos aquí al meter una dirección .W los bits más significativos **sí** que cambian

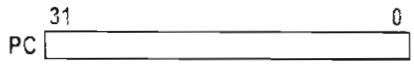


SP stack pointer

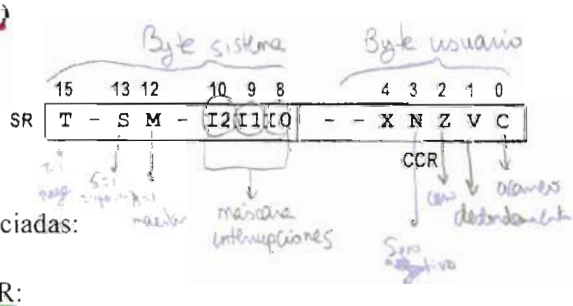
- Importante: ¡A7 es el puntero de pila!

Contador de programa (PC)

- Dirección de la instrucción en ejecución o de la siguiente
- Tiene 32 bits



Registro de estado (SR)



Tiene dos partes diferenciadas:

CCR: Condition Code Register

- Byte de usuario CCR:
  - C: indica acarreo (C=1) en aritmética sin signo
  - V: indica desbordamiento (V=1) en aritmética con signo
  - Z: indica un valor cero (Z=1)
  - N: expresa el signo o bit de mayor peso (si el valor es negativo, N=1)

Los valores negativos se expresan en *complemento a 2*

El bit X *suele* coincidir con el bit C

  - X: bit de extensión en aritmética y desplazamientos

- Byte del sistema (sólo accesible en modo supervisor)
  - I2-I0: máscara de interrupción (nivel de prioridad de la interrupción que se está atendiendo ⇒ sirve para anidar interrupciones)
  - M: modo maestro (M=1) ó interrupción (M=0)
  - S: modo usuario (S=0) ó supervisor (S=1)

Sirve para depurar el código

- T: modo traza (T=1) que provoca una excepción tras cada instrucción.

2.2.2 Modos de ejecución

No se puede pasar de modo usuario a modo supervisor poniendo S=1 en el registro de estado porque eso se hace con *instrucciones privilegiadas*

- Modo usuario (S=0)
  - Cuando se produce una excepción se pasa a modo supervisor automáticamente
  - Se regresa a modo usuario al finalizar la excepción

Sí se puede pasar de modo supervisor a usuario poniendo S=0

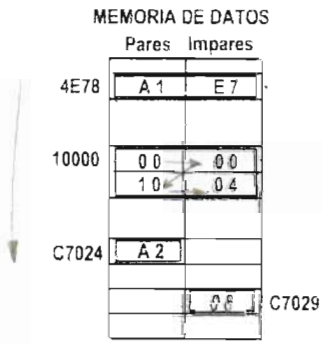
- Modo supervisor (S=1)
  - En modo supervisor se pueden ejecutar instrucciones privilegiadas (acceso total al registro de estado, e instrucciones del sistema como parada, reset...)
  - Se vuelve a modo usuario si finaliza la excepción originada en modo usuario o si se modifica el bit S del registro de estado

2.2.3 Organización en memoria

Una dirección cercana a la de la instrucción en curso (PC), es muy probable que sea otra instrucción

- Memoria de programa (sólo lectura) ROM
  - Contiene las instrucciones que se tienen que ejecutar
  - Las instrucciones tienen longitud variable
  - Protección para prevenir intentos de escritura en la memoria de programa: ¡¡no se pueden escribir posiciones relativas al contador de programa!!

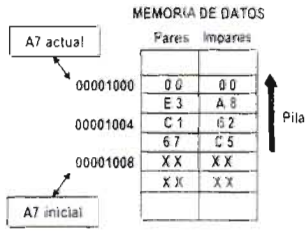
- Memoria de datos (lectura y escritura) RAM
  - Acceso .B, .W ó .L
  - Extremista mayor (big-endian)
    - Byte más significativo en la dirección más baja
    - La dirección del dato será la del byte más significativo
  - Datos alineados (deseable para accesos más rápidos)
    - Byte: en cualquier posición
    - Word: en direcciones pares
    - Long: en direcciones múltiplo de 4



2.2.4 La pila y el puntero de pila

- La pila es una zona de memoria RAM de tamaño variable para el almacenamiento de datos temporales.
- ¡¡Se llena hacia direcciones más bajas.!!
- Hay una única pila para modo supervisor y modo usuario.
- Para acceder a la pila se usa el puntero de pila: A7 (SP) que se inicializa al arrancar.
- ¡¡A7 apunta al último dato introducido!!

¡OJO! El puntero de pila **NO** apunta a la primera posición libre de la pila



2.2.5 Modos de direccionamiento

- Notación que usaremos:
  - f, d: fuente, destino
  - Dn: registro de datos n
  - An: registro de direcciones n
  - Rn: registro genérico n (por ejemplo: los registros de configuración del módulo de temporización).
  - EA: dirección efectiva
  - dspB: desplazamiento (offset) con respecto al contador de programa de B bits ⇒ Rango: $[-2^{B-1},2^{B-1}-1]$

Modos de direccionamiento simples

- **Modo directo a registro: Dn, An**
  - La dirección es un registro de datos o de dirección
  - Ejemplo:

```
MOVE.W    D2,D4
MOVEA.L   D5,A1
```

- **Modo absoluto o directo:**
  - La instrucción incluye la dirección absoluta (que puede ser .W o .L)
  - Sólo f o d puede usar este modo de direccionamiento. Nunca los dos a la vez.
  - Ejemplos:

```
MOVE.W    800,D4          * EA=800
MOVE.L    D5,$2AB         * EA=$2AB
```

No sería correcto por ejemplo:
MOVE.W $10,$80

- **Modo implícito o inherente: SR, CCR**
  - La dirección es un registro de control (CCR, SR) o de configuración de un módulo.
  - Ejemplos:

```
MOVE   D0,CCR
MOVE   SR,D3
```

- **Modo inmediato o literal: #n**
  - La instrucción incluye el operando y no su dirección.
  - Ejemplos:

```
MOVE.B    #$16,D0
MOVEQ.L   #-16,D5
MOVE.L    #%100,D4
```

Modos de direccionamiento indirectos

- **Modo indirecto: (An)**
  - EA=(An)
  - La instrucción ocupa menos bytes porque en su codificación no hay que incluir la dirección absoluta del operando.
  - Ejemplos:

```
MOVE.B    (A1),$2CE
MOVE.L    8000,(A1)
```

- **Modo indirecto con postincremento: (An)+**

Muy cómodo para recorrer tablas o búfers en memoria y para mover datos desde la pila

  - EA=(An)
  - El contenido de An se incrementa tras leerlo: .B(1 byte), .W (2 bytes), .L (4 bytes)
  - Ejemplos:

```
MOVE.W    (A3)+,D4
MOVE.L    #72,(A5)+
```

- **Modo indirecto con predecremento: -(An)**

Se usa sobre todo para mover datos a la pila

  - EA=(An)
  - El contenido de An se decrementa ~~tras~~ antes de leerlo: .B(1 byte), .W (2 bytes), .L (4 bytes)
  - Ejemplos:

```
MOVE.B    -(A3),1600
MOVE.L    -(A5),-(A6)
```

Modos de direccionamiento indirectos con desplazamiento

- **Modo indirecto con desplazamiento: N(An)**

Se usa mucho para acceder a datos que están en la pila si no se quiere modificar el valor del puntero de pila

  - EA= (An) + N
  - N= dsp16 con rango entre [-32768, 32767]
  - Ejemplos:

```
MOVE      -$B(A1),D1
MOVE.W    D0,9(A2)
```

- **Modo indirecto con índice, escala y desplazamiento: N(An, Rm*s)**
  - EA= (An) + N +(Rm)*s
  - Rm es un registo .L que actúa de índice, s es una escala (1, 2 ó 4) y N=dsp8 con rango entre [-128,127]
  - Ejemplo:

```
MOVE.W    8(A4,D6*2),D1
```

- **Modo relativo al PC con desplazamiento: N(PC)**

No se usa mucho

No es aplicable al operando destino porque no se debe escribir en direcciones "cercanas" a la de la instrucción en curso

  - EA= (PC) + N
  - N=dsp16
  - Sólo aplicable al operando fuente
  - Ejemplo:

```
MOVE      -$2(PC), D4
```

- **Modo relativo al PC con índice, escala y desplazamiento: N(PC,Rm*s)**

Mismos comentarios que con el anterior

  - EA= (PC) + N + (Rm)*s
  - N=dsp8
  - Sólo aplicable al operando fuente
  - Ejemplo:

```
MOVE.W 6(PC, D3*2), D2
```

2.3 El juego de instrucciones del ColdFire: datos

2.3.1 Instrucciones de movimiento de datos

Instrucción	Tamaños	Modos	XNZVC	Función
MOVE f,d	B,W,L	*,*	-**00	Mover datos
MOVEA f,d	W,L	*,An	-----	Mover dirección
MOVEQ f,d	L=B,d=L	#n,Dn	-**00	Mover rápido
MOVEM f,d	L	*,R's|R's,*	-----	Mover múltiple
LEA f,d	L	*,An	-----	Cargar EA en An
SWAP d	W	Dn	-**00	Intercambiar palabras

- **MOVE**: mover datos
  - El modo destino no puede ser relativo al contador de programa
  - f y d no pueden ser a la vez direcciones de memoria, ni valor inmediato y dirección de memoria. Por ejemplo, serían incorrectas:

```
MOVE.W    #3,$800
MOVE.L    $100,$200
```

Recuerda que al cargar un An se produce extensión de signo

- **MOVEA**: mover dirección (cargar An)

¡OJO! Con MOVEQ aunque el destino sea un Dn, se produce extensión de signo

- **MOVEQ**: cargar datos rápido (a Dn)
  - Si es un número negativo se codifica en Ca2 y con extensión de signo

Muy útil para guardar en la pila los registros que vayas a utilizar en una subrutina o en una rutina de servicio para luego poder restaurarlos a sus valores originales

- **MOVEM**: movimiento múltiple
  - N(An) indirecto a registro con desplazamiento opcional
  - Rs: conjunto de registros (de datos y/o direcciones)
  - ¡¡Orden fijo independientemente del orden en que se escriban en la instrucción!!: D0, D1, ... , D7, A0, A1, ... , A7
  - Mucho más rápido que varios MOVE consecutivos

- **LEA**: cargar dirección efectiva. Lleva la dirección efectiva de f a An
  - Si f es una etiqueta, equivale a `MOVEA.L #f, An`

- **SWAP**: intercambiar palabras alta y baja de un registro de datos

2.3.2 Transferencia de datos con la pila

Aquí no existen PUSH y POP como en FDOR

- No existen las instrucciones PUSH (llevar dato a la pila) y POP (sacar dato de la pila)
  - ¡¡ **PUSH=** `MOVE.X f,-(A7)` !!
  - ¡¡ **POP=** `MOVE.X (A7)+,d` !!

- Si se quieren guardar/sacar varios registros de golpe en/desde la pila usar MOVEM
  - Para guardarlos: decrementar A7 antes del `MOVEM` con `ADDA #neg,A7` (neg<0)
  - Para sacarlos: incrementar A7 después de l `MOVEM` con `ADDA #pos,A7` (pos>0)

Instrucción		Tamaños	Modos	XNZVC	Función
PEA	f	L	*	-----	Cargar EA en pila
LINK	d,#m	L,W	An	-----	Reservar zona de pila
UNLK	d	L	An	-----	Liberar zona de pila

- **PEA**: lleva la dirección efectiva de f a la pila y decrementa A7 en 4 bytes
  - si f es una etiqueta equivale a MOVE.L #f, -(A7)
- **LINK**: reservar zona de pila (m bytes). Pasos que realiza:
  - Lleva An a la pila y decrementa A7 en 4 bytes
  - Guarda A7 en An (An ahora apunta al inicio de la zona reservada)
  - Decrementa A7 en m posiciones
- **UNLK**: liberar zona de pila
  - Lleva An a A7
  - Recupera An de la pila e incrementa A7 en 4 bytes

Con esta instrucción puedes crear una "nueva pila" de tamaño m bytes cuyo "puntero de pila" es An

Instrucción complementaria de la anterior. Si ejecutas LINK An, #-m y después UNLK An te quedas como estabas

2.3.3 Instrucciones de manipulación de bits

Instrucción		Tamaños	Modos	XNZVC	Función
BTST	f,d	f=B,d=B|L(d=Dn)	#n,*|Dn,*	--*--	Test bit
BCLR	f,d	f=B,d=B|L(d=Dn)	#n,*|Dn,*	--*--	Poner bit a 0
BSET	f,d	f=B,d=B|L(d=Dn)	#n,*|Dn,*	--*--	Poner bit a 1
BCHG	f,d	f=B,d=B|L(d=Dn)	#n,*|Dn,*	--*--	Complementar bit

- **BTST**: pone Z=1 si el bit número f de d es 0 y Z=0 si es 1
- **BCLR**: pone Z=1 si el bit número f de d es 0 y Z=0 si es 1 y después lo pone a 0
- **BSET**: pone Z=1 si el bit número f de d es 0 y Z=0 si es 1 y después lo pone a 1
- **BCHG**: pone Z=1 si el bit número f de d es 0 y Z=0 si es 1 y después lo cambia al valor complementario

¡OJO! A diferencia del resto de casos, con estas instrucciones el valor del bit Z depende del valor inicial de f y no del que tiene tras la instrucción

2.3.4 Instrucciones lógicas

Instrucción		Tamaños	Modos	XNZVC	Función
NOT	d	L	*	-**00	Complemento lógico
AND	f,d	L	Dn,*|*,Dn	-**00	AND lógico
ANDI	f,d	L	#n,Dn	-**00	AND lógico inmediato
OR	f,d	L	Dn,*|*,Dn	-**00	OR lógico
ORI	f,d	L	#n,Dn	-**00	OR lógico inmediato
EOR	f,d	L	Dn,*	-**00	OR exclusivo lógico
EORI	f,d	L	#n,Dn	-**00	OR excl. Inmediato

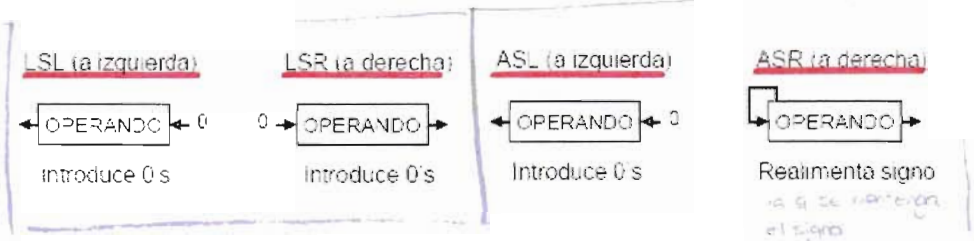
Se usan mucho para implementar máscaras

2.3.5 Instrucciones de desplazamiento

Instrucción		Tamaños	Modos	XNZVC	Función
LSL	f,d	f=B|W,d=L	#n,Dn|Dn,Dn	***0*	Despl. lógico a izq.
LSR	f,d	f=B|W,d=L	#n,Dn|Dn,Dn	***0*	Despl. lógico a der.
ASL	f,d	f=B|W,d=L	#n,Dn|Dn,Dn	***0*	Despl. aritmético izq.
ASR	f,d	f=B|W,d=L	#n,Dn|Dn,Dn	***0*	Despl. aritmético der.

Se usan mucho para multiplicar (desplazamiento de n bits a la izquierda) o dividir (n bits a la derecha) por 2^n

- El último bit que sale siempre va a X y C del CCR



2.3.6 Instrucciones aritméticas

- Los números con signo se expresan en Ca2. El bit más significativo representa el signo (si es 1 se puede considerar como número negativo).
- Las restas en realidad son sumas que usan el Ca2 del sustraendo
- Sumas: (f) + (d) → (d)
- Restas: (d) – (f) → (d)

Instrucción	Tamaños	Modos	XNZVC	Función
ADD f,d	L	Dn,*|*,Dn	*****	Sumar datos
ADDA f,d	L	*,An	-----	Sumar direcciones
ADDI f,d	L	#n,Dn	*****	Sumar inmediato
ADDQ f,d	f=3,d=L	#n,*	*****	Sumar rápido
SUB f,d	L	Dn,*|*,Dn	*****	Restar datos
SUBA f,d	L	*,An	-----	Restar direcciones
SUBI f,d	L	#n,Dn	*****	Restar inmediato
SUBQ f,d	f=3,d=L	#n,*	*****	Restar rápido

2.3.7 Instrucciones de signo y borrado

Instrucción	Tamaños	Modos	XNZVC	Función
NEG d	L	*	*****	Negación o Ca2
EXT d	B?W,W?	Dn	-**00	Extensión de signo
EXTB d	B?L	Dn	-**00	Extensión de signo 2
CLR d	B,W	*	-0100	Borrado

No confundir con NOT que hace el complemento lógico, no el complemento a 2

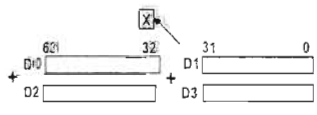
- **NEG**: 0 – (d) → (d) (obtiene el Ca2)
- **EXT**: extensión de signo duplicando tamaño (EXT.X donde X es el tamaño final de d. .X=.W o .X=.L)
- **EXTB**: extensión de signo de byte a long (EXT.L siempre)
- **CLR**: 0 → (d)

2.3.8 Instrucciones de aritmética extendida

Instrucción	Tamaños	Modos	XNZVC	Función
ADDX f,d	L	Dn,Dn	*****	Suma extendida
SUBX f,d	L	Dn,Dn	*****	Resta extendida
NEGX d	L	*	*****	Negación extendida

- Permiten operar con datos > 32 bits. El acarreo de la suma/resta de los 32 bits menos significativos de los operandos (que se hacen con las instrucciones del apartado 2.3.6) se lleva al bit X (que es copia del C) y después se usan las instrucciones de este apartado para los bits más significativos de los operandos
- **ADDX**: (f) +(d) + (X) → (d)
- **SUBX**: (d) - (f) – (X) → (d)
- **NEGX**: 0 – (d) – (X) → (d)
- Ejemplo de ADDX:

```
* Suma de 64 bits sin signo:
*   D0/D1 + D2/D3 - D2/D3

    ADD.L   D1,D3   * D1+D3=D3
    ADDX.L  D0,D2   * D0+D2+X=D2
```



2.3.9 Instrucciones de comparación y test

Instrucción	Tamaños	Modos	XNZVC	Función
CMP f,d	L	*,Dn	-****	Comparación de datos
CMPA f,d	L	*,An	-----	Comp. de direcciones
CMPI f,d	L	#n,Dn	-****	Comparac. Inmediata
TST d	B,W,L	*	-**00	Test y actualizar CCR

Antes de instrucciones de salto condicional, se usan mucho

- El resultado sirve para actualizar los bits de CCR, pero ¡¡no se guarda en d!!
- Instrucciones de comparación (d) - (f) → (bits del CCR)
- **TST**: (d) – 0 → (bits del CCR)

2.3.10 Instrucciones de multiplicación y división

Instrucción		Tamaños	Modos	XNZVC	Función
MULS	f,d	W,L	*,Dn	-**0C	Multiplicación con signo
MULU	f,d	W,L	*,Dn	-**0C	Multiplicación sin signo
DIVS	f,d	W,L	*,Dn	-***C	División con signo
DIVU	f,d	W,L	*,Dn	-***C	División sin signo
REMS	f,d1:d2	L	*,Dn,Dm	-***C	Resto con signo
REMU	f,d1:d2	L	*,Dn,Dm	-***C	Resto sin signo

¡OJO! Para multiplicar y dividir por potencias de 2 es mucho más rápido usar las instrucciones de desplazamiento que éstas

- Multiplicación: (f)*(d) → (d)
  - El resultado siempre es de 32 bits (se queda con los menos significativos en caso de que no quepa)

- División: (d)/(f) → (d)
  - Si f es de tamaño .W: el cociente se almacena en los bits 0-15 y el resto en los bits 16-31 de d
  - Si f es de tamaño .L: el cociente en los 32 bits de d. Para el cálculo del resto se deben usar las instrucciones REMS y REMU

- Resto: resto de (d2)/(f) → (d1)

2.4 El juego de instrucciones de ColdFire: control

2.4.1 Instrucciones de salto incondicional

Instrucción		Tamaños	Modos	XNZVC	Función
JMP	d	L	*	-----	Salto incondicional
BRA	d	B,W	N(PC)	-----	Salto incondicional relativo al PC
JSR	d	L	*	-----	Salto a subrutina
BSR	d	B,W	N(PC)	-----	Salto a subrutina relativo al PC
RTS		-	-	-----	Retorno de subrutina

- **JMP**: salto simple
  - La dirección de destino se codifica en código máquina como una dirección de memoria absoluta (32 bits).

La extensión .S en el código de instrucción indica un desplazamiento pequeño respecto al PC (8 bits)

- **BRA**: salto simple relativo al PC
  - La dirección de destino se codifica en código máquina como un desplazamiento (8 ó 16 bits) respecto a la posición actual del PC.
  - Utiliza internamente un modo de direccionamiento relativo al PC con desplazamiento N(PC), pero es transparente si se usan etiquetas.

Igual que CALL en FDOR

- **JSR**: salto a subrutina
  - La dirección de inicio de la subrutina se codifica en código máquina como una dirección de memoria absoluta (32 bits).
  - Guarda el valor del PC en la pila para poder retornar al programa principal tras el RTS

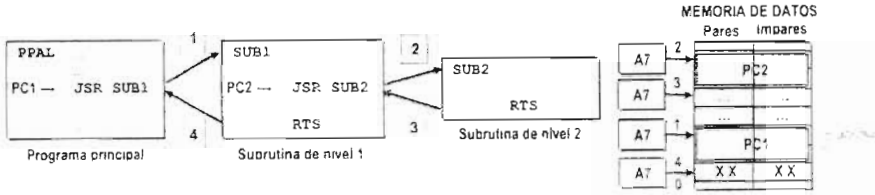
La extensión .S en el código de instrucción indica un desplazamiento pequeño respecto al PC (8 bits)

- **BSR**: salto a subrutina relativo al PC
  - La dirección de inicio de la subrutina se codifica en código máquina como un desplazamiento (8 ó 16 bits) respecto a la posición actual del PC.
  - Utiliza internamente un modo de direccionamiento relativo al PC con desplazamiento N(PC), pero es transparente si se usan etiquetas.

- Guarda el valor del PC en la pila para poder retornar al programa principal tras el RTS.

- **RTS**: retorno de subrutina
  - Es necesaria cuando previamente se usa JSR o BSR
  - Recupera el PC de la pila.

- Ejemplo de saltos a subrutinas anidados



2.4.2 Instrucciones de salto condicional

Códigos de condición

- Caracterizan resultados previos a partir de los bits de CCR

CC	CONDICION	FORMULA
HI	Superior	C#&Z#
LS	Inferior o igual	C | Z
CC	Acarreo = 0	C#
CS	Acarreo = 1	C
NE	Distintos (de 0)	Z#
EQ	Iguales (a 0)	Z
VC	Desbord. = 0	V#
VS	Desbord. = 1	V
PL	Positivo	N#
MI	Negativo	N
GE	Mayor o igual	(N&V) | (N#&V#)
LT	Menor	(N&V#) | (N#&V)
GT	Mayor	(N&V&Z#) | (N#&V#&Z)
LE	Menor o igual	Z | (N&V#) | (N#&V)

Instrucciones de salto condicional

¡Muy importante! Se suelen usar después de las instrucciones CMP y TST

Instrucción	Tamaños	Modos	XNZVC	Función
Bcc d	B,W	N(PC)	-----	Salto si se cumple cc
Scc d	B	*	-----	Pone Fs ó 0s según cc

- **BCC**: salto si se cumple la condición cc

CMP f,d

Condición de salto	CMP sin signo	CMP con signo
d = f	BEQ	BEQ
d ≠ f	BNE	BNE
d > f	BHI	BGT
d ≥ f	BCC	BGE
d < f	BCS	BLT
d ≤ f	BLS	BLE

TST d

Condición de salto	TST sin signo	TST con signo
d = 0	BEQ	BEQ
d ≠ 0	BNE	BNE
d > 0	BNE	BGT
d ≥ 0	-	BGE, BPL
d < 0	-	BLT, BMI
d ≤ 0	-	BLE

- **SCC**:
  - Si se cumple cc: pone el byte destino a $FF
  - Si no se cumple cc: pone el byte destino a $00

2.4.3 Instrucciones de control del sistema

En modo usuario

Instrucción	Tamaños	Modos	XNZVC	Funcion
MOVE CCR,d	W	Dn	-----	Mover de CCR a registro
MOVE f,CCR	W	#n|Dn	*****	Mover a CCR
NOP	-	-	-----	Sin operacion

También lo son las instrucciones generadoras de excepciones (tema 4)

- **NOP**: sin operación
  - Ocupa memoria e incrementa el PC

En modo supervisor (instrucciones privilegiadas)

Instrucción	Tamaños	Modos	XNZVC	Funcion
MOVE SR,d	W	Dn	-----	Mover SR a registro
MOVE f,SR	W	*	*****	Mover a SR
MOVEC f,Cntrl	L	Rn	-----	Mover a registro de control

- **MOVEC**: mover a registro de control
  - Cntrl: registro de control de módulo, cada uno tiene su propio nombre.

También lo son las instrucciones de depuración (Tema 3), algunas relacionadas con excepciones (Tema 4) y las de la memoria caché (Tema 7)

Ejemplo 1.1

Describir cualitativamente la secuencia de pasos que se produce en un sistema microprocesador genérico al ejecutar la instrucción ADD 2000, D0. En este sistema ADD es una instrucción que suma los dos operandos indicados. 2000 es la dirección del primer operando en memoria, y D0 es un registro de datos. El resultado se almacena en D0

Sugerencia: fíjate en el diagrama de bloques del sistema microprocesador completo y de la CPU.

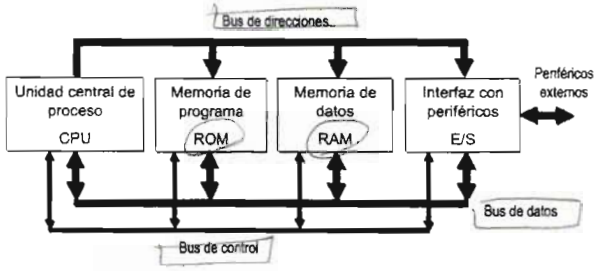


Diagrama de bloques del sistema microprocesador

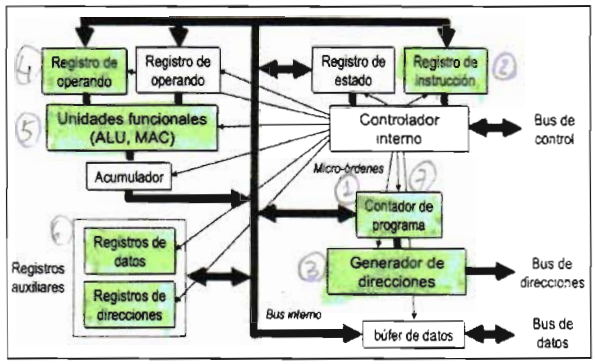


Diagrama de bloques de la CPU

ADD 2000, D0

1) Leer instrucción q está en la dirección = [CP] y llevarla al reg de instruc. (RI)

2) El cont interno interpreta la instrucción ES una suma.

3) Se calculan las direcciones de los operandos
   - El primero está en la dirección de memoria 2000
   - El segundo está en el registro interno D0

4) El generador de direcciones pone 2000 en binario en el bus de direcciones y el controlador interno genera la señal de activación de la memoria. La memoria pone [M(2000)] en el bus de datos y de ahí pasa a uno de los registros operando (RO)

5) El controlador da la orden de cargar [D0] en el otro reg de operando

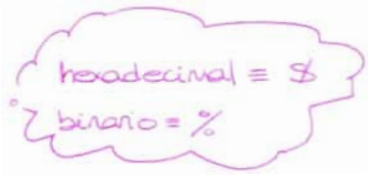
6) El controlador da la orden de sumar a la ALU

7) El controlador da la orden de cargar el resultado en D0

8) Se incrementa el valor de CP

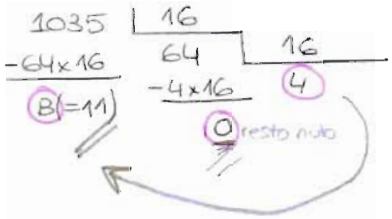
Ejemplo 1.2

a) Convertir el número decimal 1035 a hexadecimal.
b) Convertir el número hexadecimal C3A a decimal.
c) Convertir el número hexadecimal C3A a binario.



(a) hexadecimal ≡ $

$A = 10
$B = 11
$C = 12
⋮
$F = 15



1035 = $40B

(b) $C3A = A×16⁰ + 3×16¹ + C×16² = 10 + 48 + 12×256 = 3130

(c) $C3A = %1100 0011 1010
C 3 A

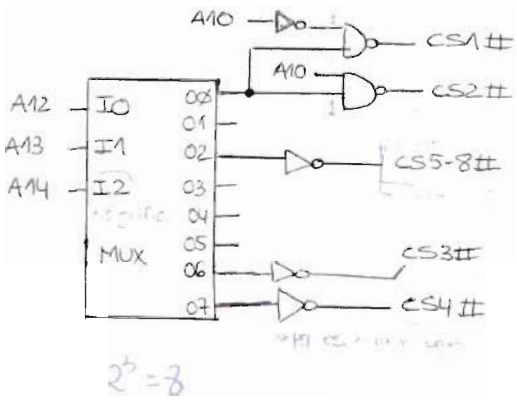
Ejemplo 1.3

Diseñar un circuito combinacional que genere las señales de CS# necesarias para el mapa de memoria del apartado 1.2 utilizando un multiplexor de 3 entradas y las puertas NOT y NAND que sean necesarias

Nota: Las salidas del multiplexor son activas a nivel alto. Las señales de Chip-Select son activas a nivel bajo.

MAPA FUNCIONAL	MAPA FISICO	DIRECCIONES Decimal	Hex	Binario
Tablas y datos fijos	Pastilla 1: ROM 1Kx8	00000 01023	0000 03FF	0000 0000 0000 0000 0000 0011 1111 1111
Programa	Pastilla 2: ROM 1Kx8	01024 02047	0400 07FF	0000 0100 0000 0000 0000 0111 1111 1111
(Zona no usada)	----	02048 40959	0800 9FFF	0000 1000 0000 0000 1001 1111 1111 1111
Entrada/salida	Pastillas 5 a 8 (E/S)	40960 45055	A000 AFFF	1010 0000 0000 0000 1010 1111 1111 1111
(Zona no usada)	----	45056 57343	B000 DFFF	1011 0000 0000 0000 1101 1111 1111 1111
Variables y pila	Pastilla 3: RAM 4Kx8	57344 61439	E000 EFFF	1110 0000 0000 0000 1110 1111 1111 1111
Datos temporales	Pastilla 4: RAM 4Kx8	61440 65535	F000 FFFF	1111 0000 0000 0000 1111 1111 1111 1111

	A15 A14 A13 A12	A11 A10 A9 A8	A7 A6 A5 A4	A3 A2 A1 A0
CS1#	0 0 0 0	0 0	X X	X X X X	X X X X
CS2#	0 0 0 0	0 1	X X	X X X X	X X X X
CS5-8#	1 0 1 0	X X	X X	X X X X	X X X X
CS3#	1 1 1 0	X X	X X	X X X X	X X X X
CS4#	1 1 1 1	X X	X X	X X X X	X X X X



Ejemplo 2.1

Observe el siguiente listado en el que además del código se indican las posiciones de memoria y el contenido de las mismas en hexadecimal, y conteste a las preguntas

Posición de memoria	Contenido de memoria (o bytes reservados)	Nº de línea	Programa		
00000000		1	* Suma de los 25 primeros enteros		
00000000		2			
00000000		3	TAMPILA	EQU	256
00000000		4			
00000000		5		ORG	$0
00000000	00004108	6		DC.L	FINPILA
00000004	00002000	7		DC.L	INICIO
00000008		8			
00002000		9		ORG	$2000
00002000	42B9 00004000	10	INICIO	CLR.L	SUMA *Pone a 0
00002006	2039 00004004	11		MOVE.L	CUENTA,D0
0000200C	D1B9 00004000	12	BUCLE	ADD.L	D0,SUMA
00002012	5380	13		SUB.L	#1,D0 *Decrementa
00002014	66F6	14		BNE	BUCLE *Salta si no 0
00002016		15			
00004000		16		ORG	$4000
00004000	00000004	17	SUMA	DS.L	1
00004004	00000019	18	CUENTA	DC.L	25
00004008	00000100	19	PILA	DS.B	TAMPILA
00004108		20	FINPILA		
00004108		21		END	

a) Valor de todas las etiquetas que aparecen en el listado.
b) Explicar el contenido de la memoria en las posiciones $0, $4, $4000, $4004, $4008 y $2014.

a) TAMPILA = 256
INICIO = $2000
BUCLE = $200C
SUMA = $4000
CUENTA = $4004
PILA = $4008
FINPILA = $4108

b) [M($0)] = $4108 = FINPILA
[M($4)] = $2000
[M($4000)] = no lo sabemos ($4 es el nº de bytes reservados)
[M($4004)] = $19
[M($4008)] = no lo sabemos ($100 es el nº de bytes reservados)
[M($2014)] = $66F6

! las etiquetas son siempre direc de MEM excepto cuando van seguidas de EQU
- tienen el valor de la pos de MEM en la q están, menos en el caso de EQU
q es una cte

Ejemplo 2.2

Con este ejemplo se pretende que se entiendan los modos de direccionamiento simples.

Diga que modo de direccionamiento se usa en cada instrucción para cada operando. Dibuje el contenido de los registros implicados y las posiciones de memoria tanto de datos como de programa, implicadas en las siguientes instrucciones, que poseen modos de direccionamiento simples, y describa lo que ocurre con las mismas al ejecutarse la instrucción en cuestión.

A la derecha de las instrucciones aparecen los valores iniciales necesarios

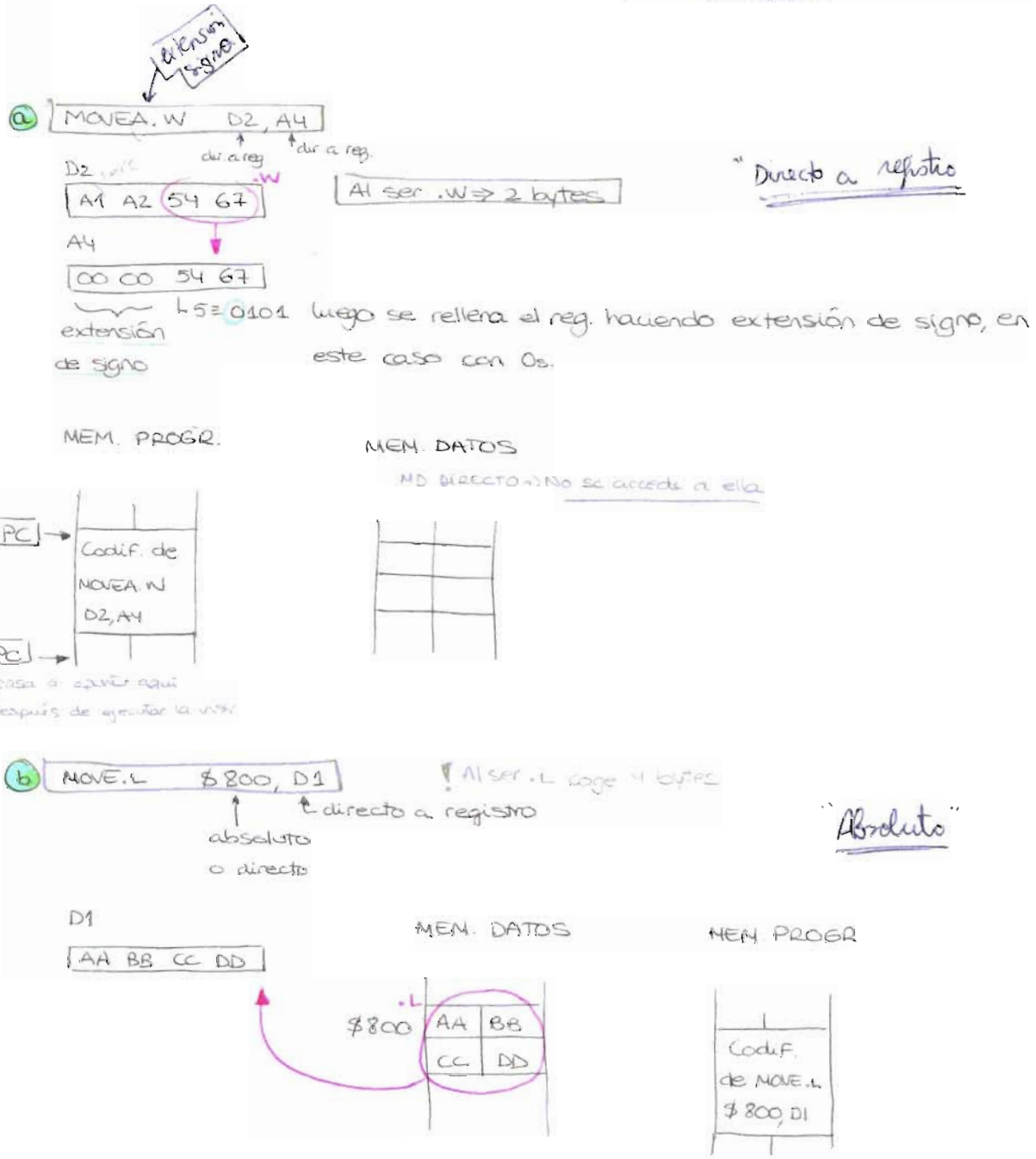
```
a) MOVEA.W  D2,A4    * Valor inicial de D2: $A1 A2 54 67
b) MOVE.L   $800,D1  * Valor inicial de la dirección $800: $AA BB CC DD
c) MOVE     D1,CCR   * Valor inicial de D1: %10101
d) MOVE.B   #$3B,D0  * Valor inicial de D0: $AA BB CC DD
```

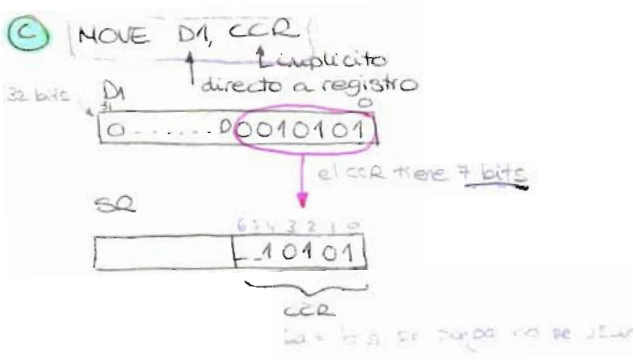
→ Directo a ~~directo~~ registro

→ ~~Inmediato~~ Absoluto

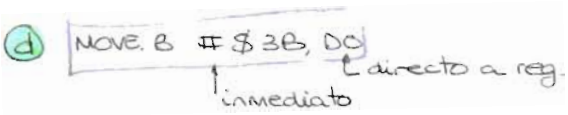
→ Implícito.

→ Inmediato.

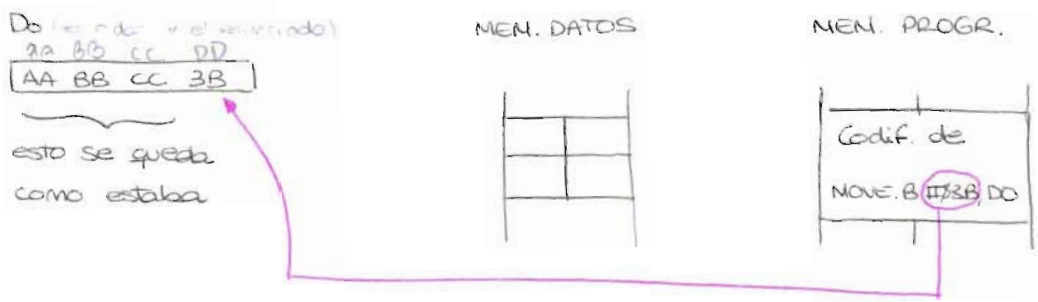




"Implícito"



"Inmediato"



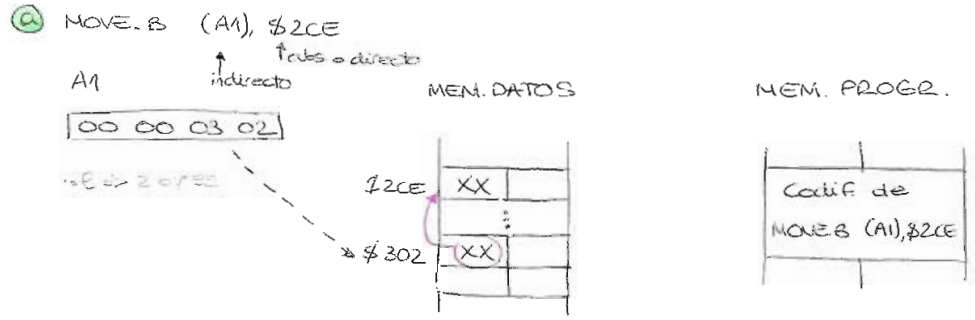
Ejemplo 2.3

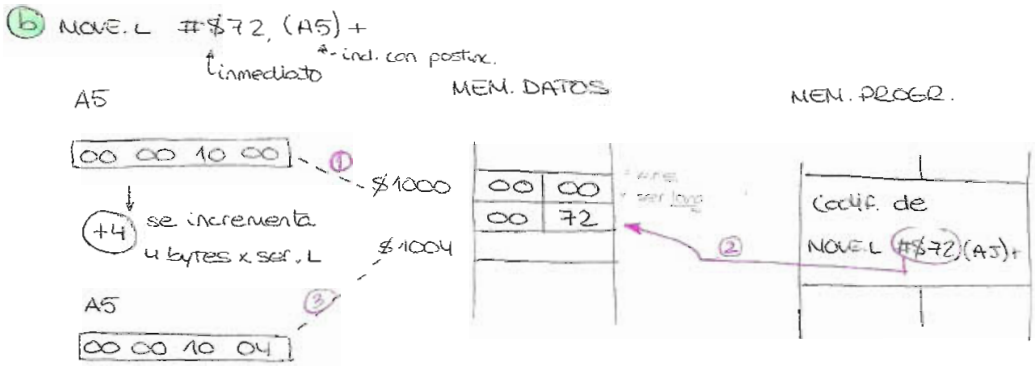
Con este ejemplo se pretende que se entiendan los modos de direccionamiento indirectos e indirectos con desplazamiento.

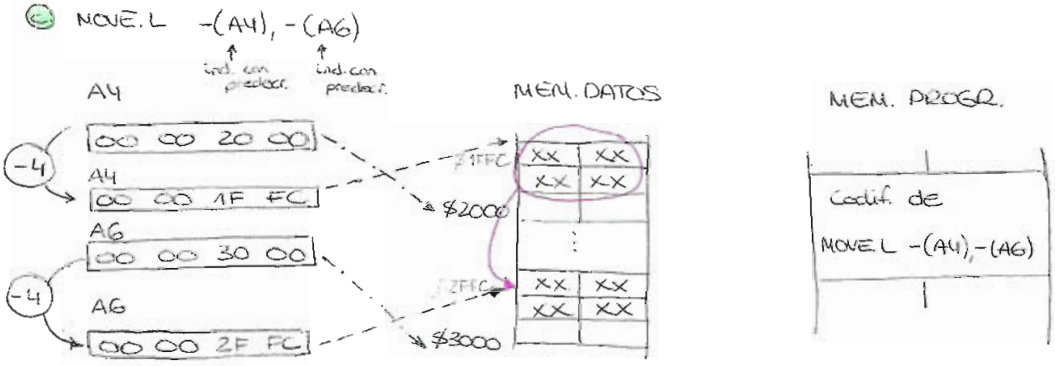
Diga que modo de direccionamiento se usa en cada instrucción para cada operando. Dibuje el contenido de los registros implicados y las posiciones de memoria tanto de datos como de programa implicadas en las siguientes instrucciones, y describa lo que ocurre con las mismas al ejecutarse la instrucción en cuestión.

A la derecha de las instrucciones aparecen los valores iniciales necesarios

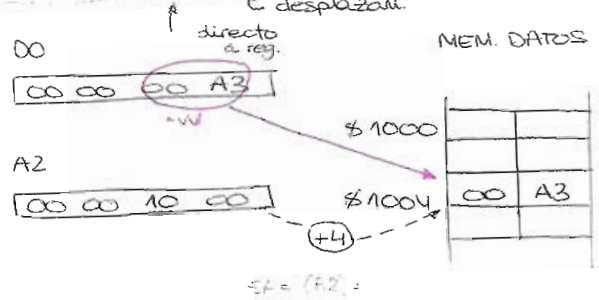
```
a) MOVE.B   (A1),$2CE       * Valor inicial de A1: $302
b) MOVE.L   #$72,(A5)+      * Valor inicial de A5: $1000
c) MOVE.L   -(A4),-(A6)     * Valor inicial de A4: $2000; valor inicial de A6: $3000
d) MOVE.W   D0, 4(A2)       * Valor inicial de D0: $A3; valor inicial de A2: $1000
e) MOVE.L   6(A4,D6*2),D1   * Valor inicial de D6: $2; valor inicial de A4: $2000
```



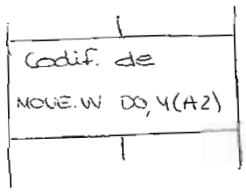




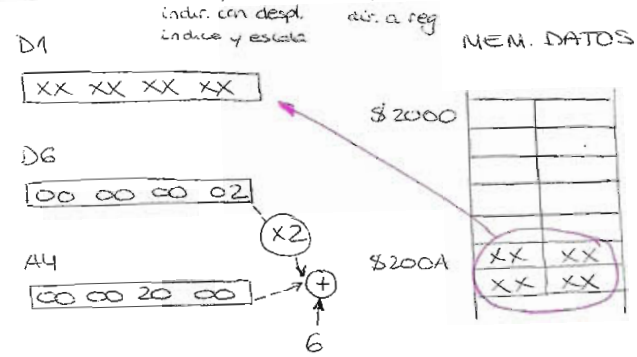
d) MOVE.W D0, 4(A2) ind. con desplazam.



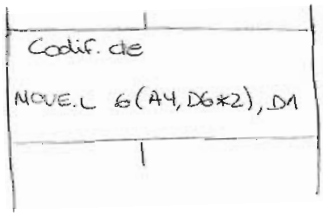
MEM. PROGR



e) MOVE.L 6(A4, D6*2), D1



MEM. PROGR



EA = (A4) + (D6)X2 + 6

Ejemplo 2.4

El objetivo de este ejemplo es adquirir soltura con las instrucciones de movimiento de datos.

Se pide dibujar los registros y posiciones de memoria, tanto de datos como de programa implicados en las siguientes secuencias de instrucciones, y describir lo que ocurre con los mismos al ejecutar dichas instrucciones. Se proporcionan los valores iniciales de registros y posiciones de memoria en caso de ser necesarios.

a)

```
MOVEQ    #-5,D1
MOVE.B   D1,$3001
MOVEA.W #$9008,A5
```

b)

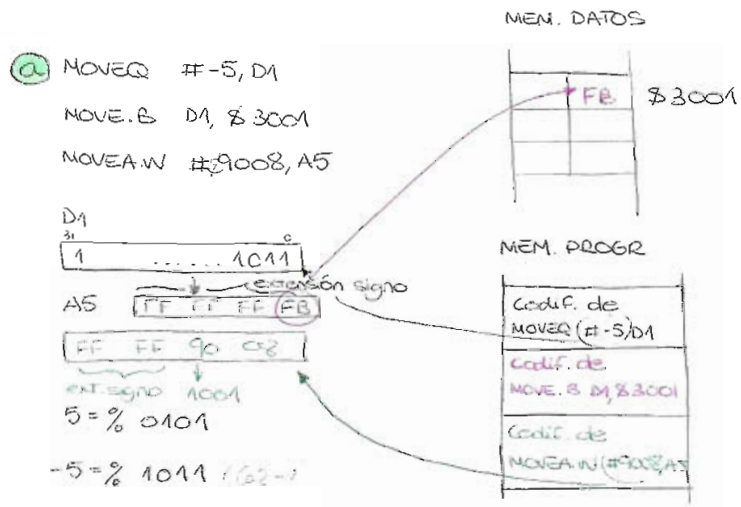
```
* (A1)=$ 00 00 10 00
* Contenido de la dirección $1000 y sucesivas: 5A 13 23 08 A1 E7 F7 25  00 00 10 04
* 0E 66 A2 0E

MOVEM (A1),A0/D0-D2
MOVEM D2/D1,2(A0)
```

c)

```
VEINTE EQU $14
       ...
       LEA VEINTE,A0
```

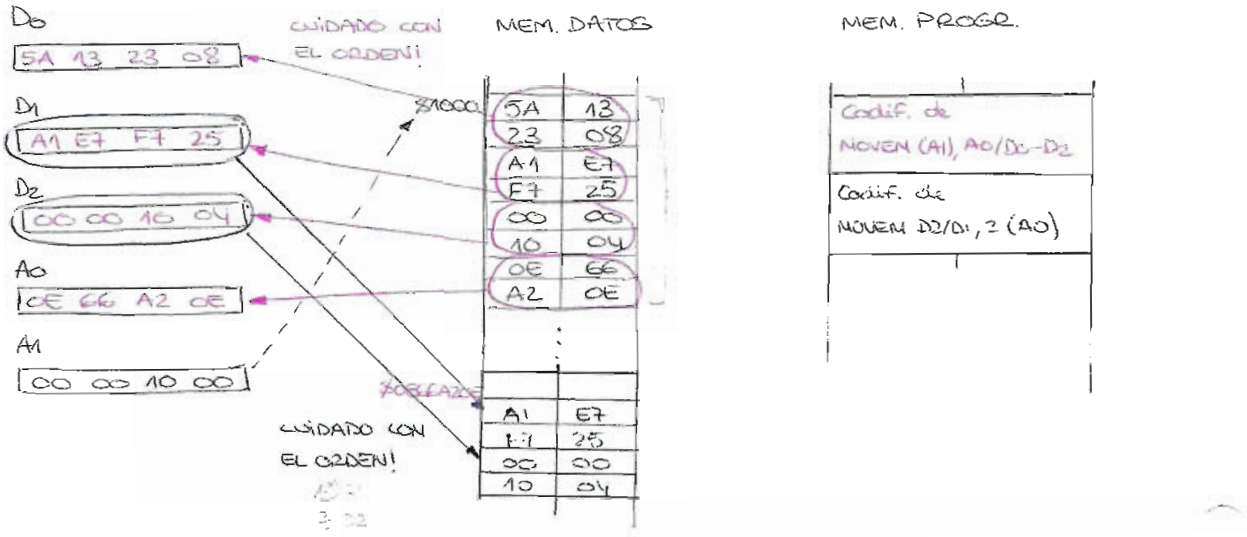
d)

```
* VIA=$00A6400
* Contenido de la dirección $000A6400 y sucesivas: FF FB 37 55

VIA    DS.W 1
       ...
       LEA        VIA,A5
       MOVE.W    (A5)+,D0
       MOVE.W    (A5),D1
       SWAP       D1
```



b) MOVEM (A1), A0/D0-D2 & MOVEM (A1), D0-D2/A0

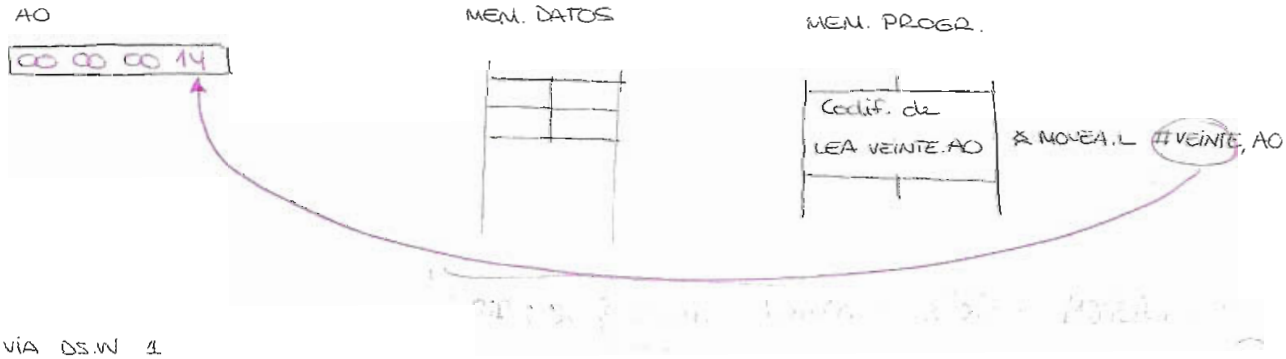
MOVEM D2/D1, 2(A0) & MOVEM D1-D2, 2(A0)



c) VEINTE EQU $14

...

LEA VEINTE, A0



d) VIA DS.W 1

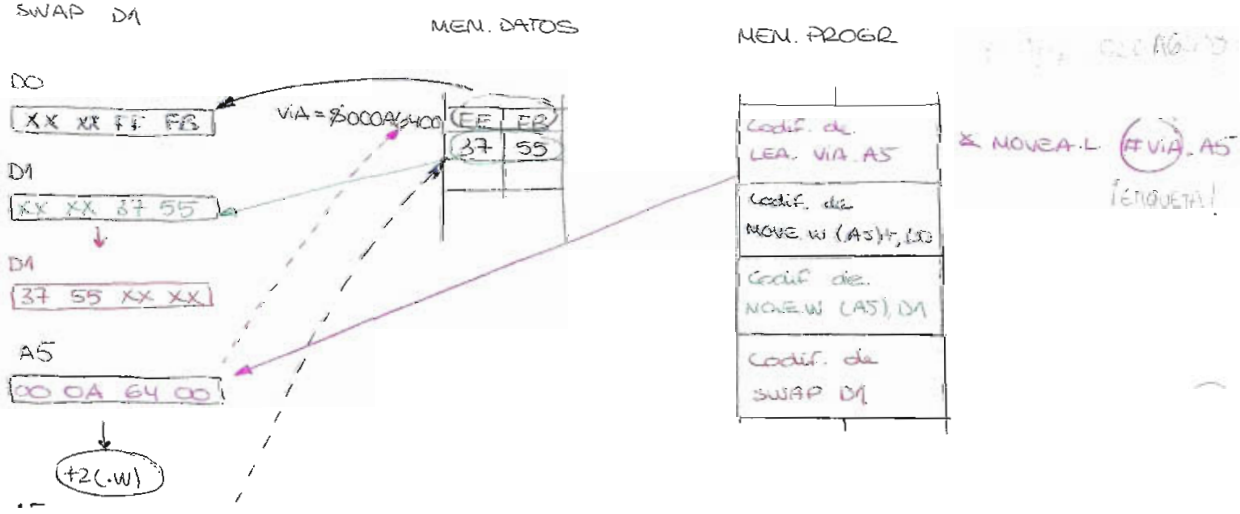
...

LEA VIA, A5

MOVEM (A5)+, D0

MOVE.W (A5), D1

SWAP D1



Ejemplo 2.5

El objetivo de este ejemplo es adquirir soltura con las instrucciones que se usan para transferir datos a la pila.

Se pide dibujar los registros y posiciones de memoria, tanto de datos como de programa implicados en las siguientes secuencias de instrucciones, y describir lo que ocurre con los mismos al ejecutar dichas instrucciones. Se proporcionan los valores iniciales de registros y posiciones de memoria en caso de ser necesarios.

a)

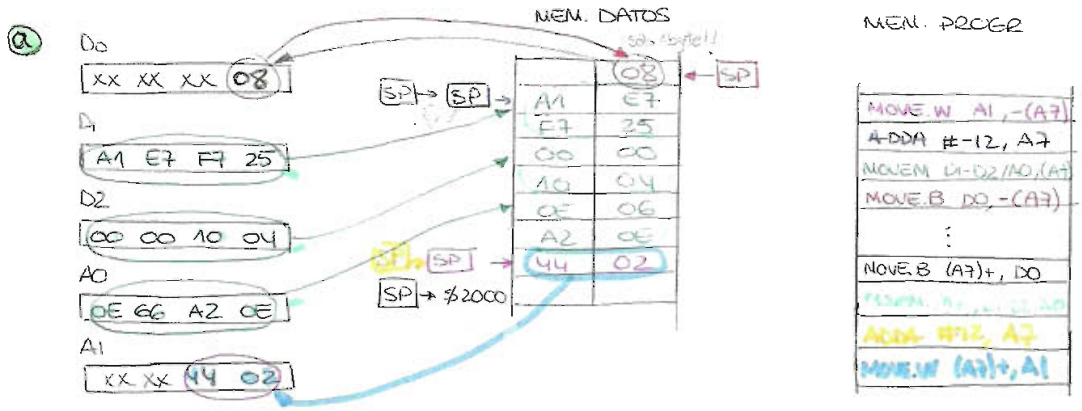
```
* (A1)=$XX XX 44 02   (A7)=$2000          (D1)=$A1 E7 F7 25
* (D2)=$00 00 10 04   (A0)=$0E 66 A2 0E   (D0)=$XX XX XX 08

MOVE.W A1,-(A7)
ADDA   #-12,A7
MOVEM  D1-D2/A0,(A7)
MOVE.B D0,-(A7)
...
MOVE.B (A7)+,D0
MOVEM  (A7),D1-D2/A0
ADDA   #12,A7
MOVE.W (A7)+,A1
```

b)

```
* (A7)=$2004

VEINTE EQU $14
       PEA VEINTE
```

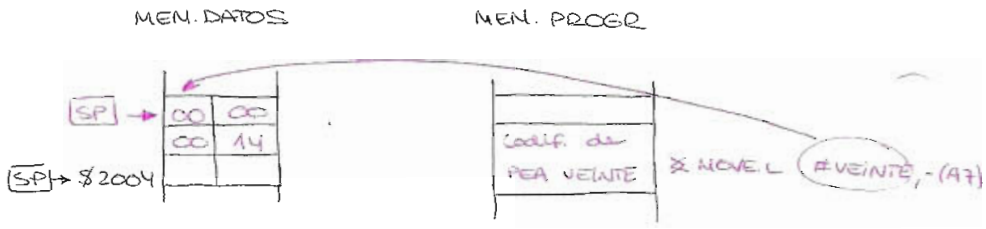
c)

```
* REF=$3000
* (A7)=$00FC900C

REF    DC.B 8
       ...
       PEA      REF
       MOVEA.L #$770E,A1
       LINK     A1,#-4
```



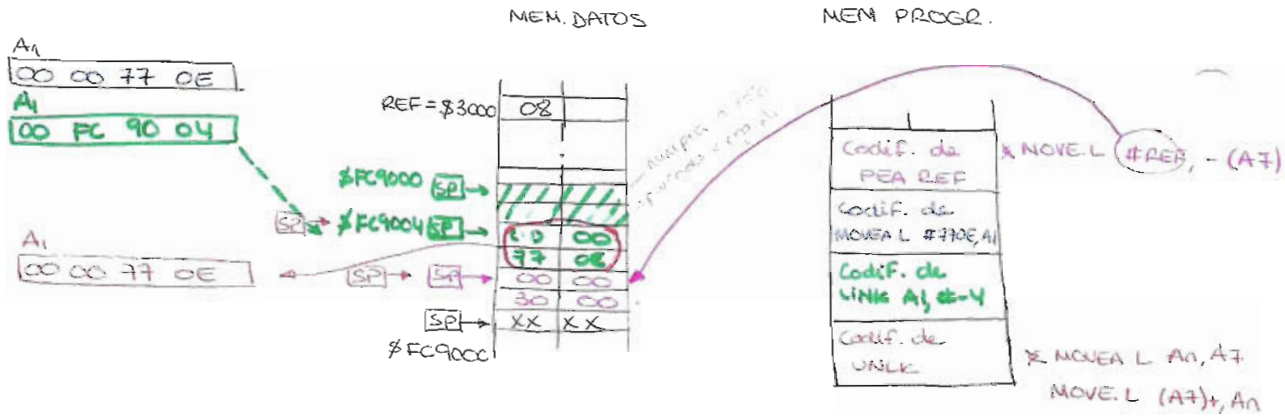
(b)

```
VEINTE EQU $14
       PEA VEINTE
```



(c)

```
REF  DC B 8
...
       PEA REF
       MOVEA.L #$770E, A1
       LINK A1, #-4
       ...
       UNLK A1
```



Ejemplo 2.6

El objetivo de este ejemplo es adquirir soltura con las instrucciones de manipulación de bits, lógicas, de desplazamiento, y de signo y borrado.

Se pide dibujar los registros implicados en las siguientes secuencias de instrucciones, y describir lo que ocurre con los mismos tras la ejecución de cada instrucción.

a) En este caso además se pide el valor del bit Z del CCR tras la ejecución de cada instrucción

```
MOVE.B #%10100110,D0
BTST   #3,D0
BCLR   #2,D0
BSET   #3,D0
BCHG   #4,D0
```

b)

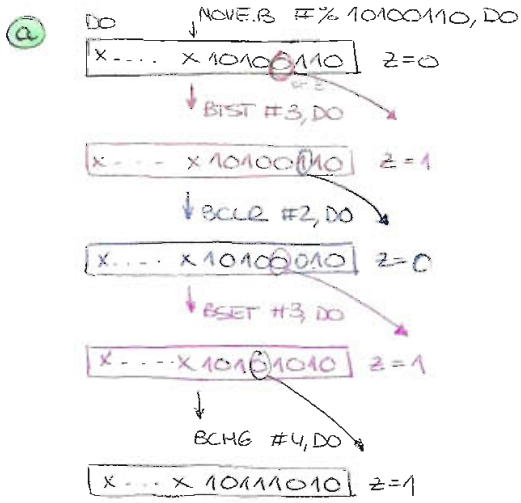
```
MOVE.L #$F0,D3
ORI    #%10010110,D3
EORI   #$F0,D3
NOT    D3
ANDI   #%11010011,D3
```

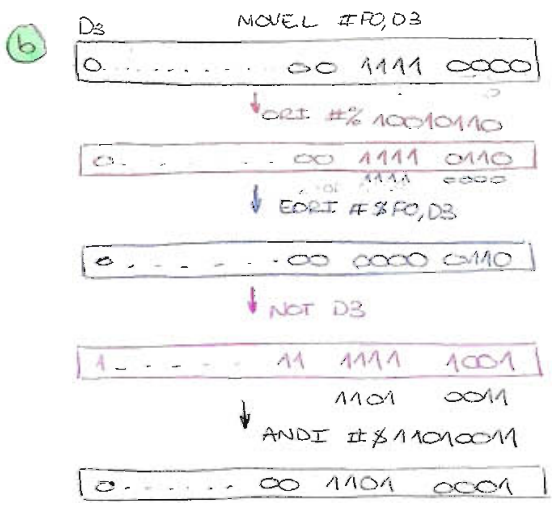
c) En este caso además se pide el valor de los bits N, Z, C y X del CCR tras la ejecución de cada instrucción

```
MOVE.L #$AFFFFFFF,D5
LSL    #3,D5
LSR    #1,D5
ASL    #2,D5
ASR    #1,D5
```

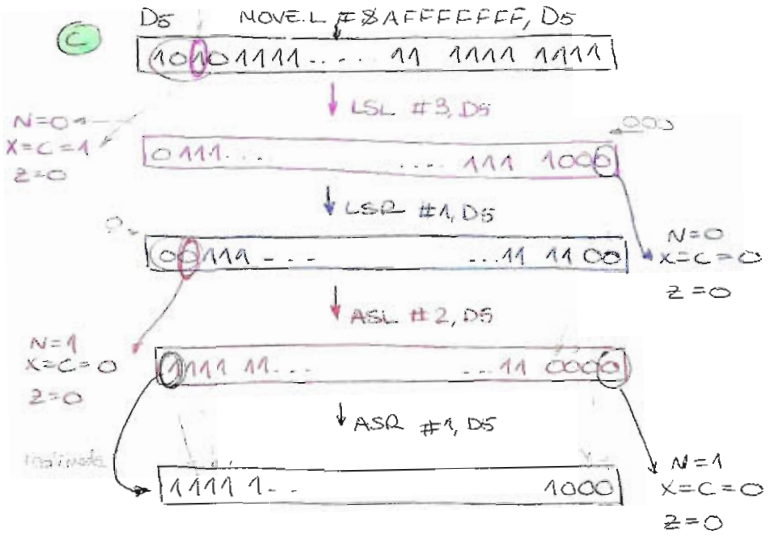
d)

```
MOVE.W #$58FA,D6
EXT.L  D6
NEG    D6
EXTB.L D6
CLR.B  D6
```

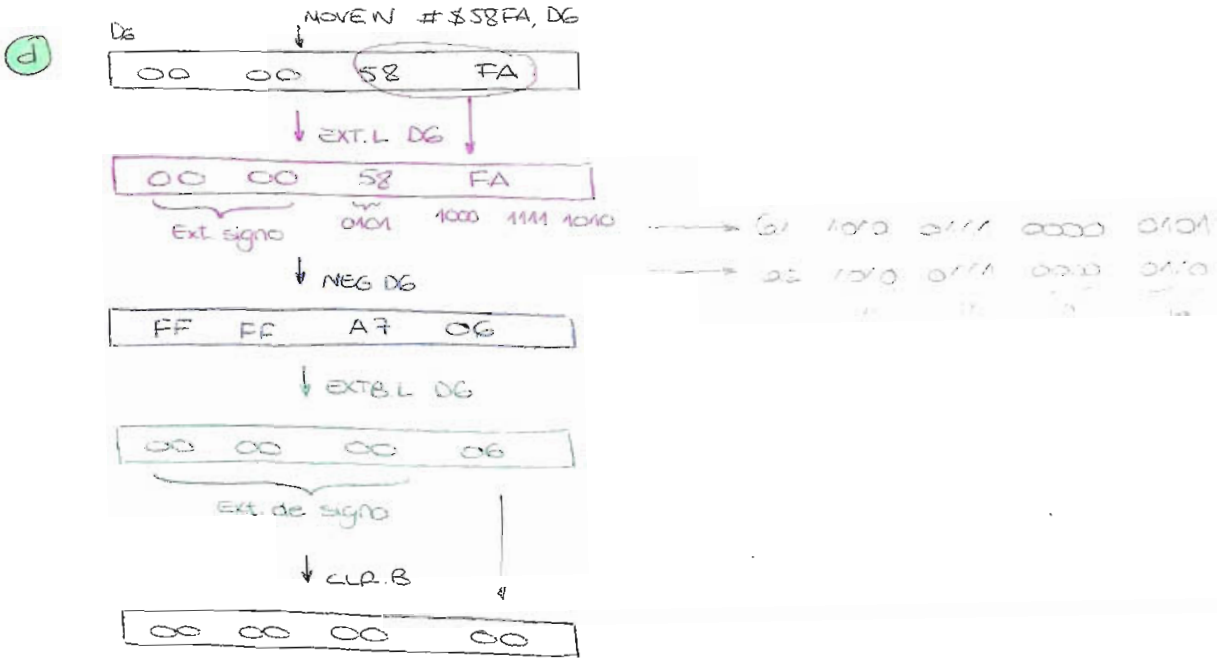




c)

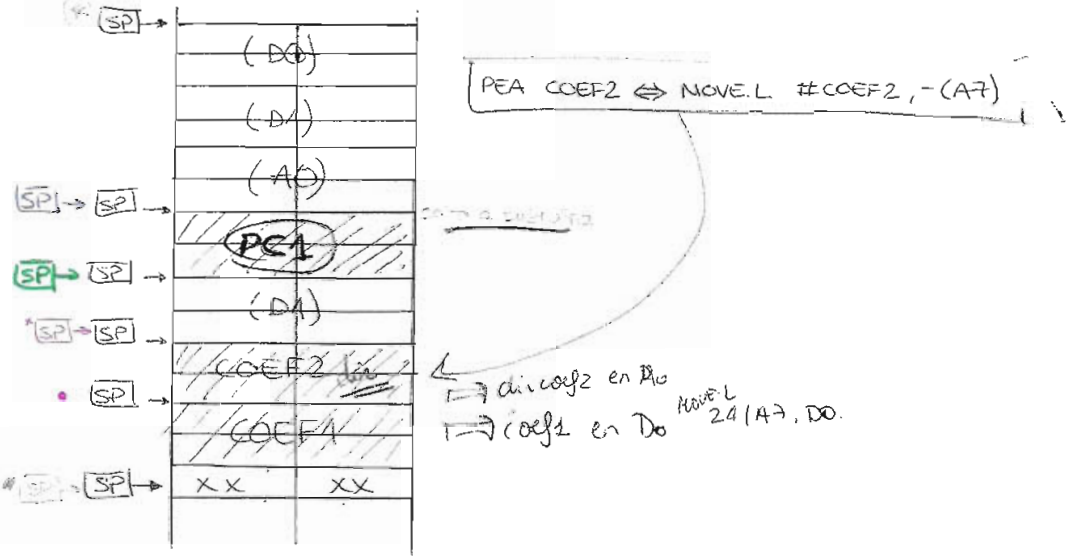

D5
MOVE.L #$AFFFFFFF, D5
1010 1111 .. .. 11 1111 1111
LSL #3, D5
N=0
X=C=1
Z=0
0111 .. .. 111 1000
LSR #1, D5
0
0011 1 .. .. 11 1100
N=0
X=C=0
Z=0
ASL #2, D5
N=1
X=C=0
Z=0
1111 11 .. .. 11 0000
ASR #1, D5
Instrucción
1111 1 1000
N=1
X=C=0
Z=0

d)


D6
MOVE.W #$58FA, D6
00 00 58 FA
EXT.L D6
00 00 58 FA
Ext. signo
0101 1000 1111 1010
NEG D6
FF FF A7 06
EXTB.L D6
00 00 00 06
Ext. de signo
CLR.B
00 00 00 00

El objetivo del ejemplo es entender el paso de parámetros a una subrutina a través de la pila. Además se puede ver cómo también se pueden devolver los resultados a través de la pila. Dibujar la pila y la posición del puntero de pila tras cada instrucción.

```
        ...
        MOVE.L  COEF1,-(A7)     * Parámetro COEF1 a pila
        PEA     COEF2           * Dirección de COEF2 a pila
        SUBQ    #4,A7           * Reserva pila para resultado
        JSR     CALC            * Llamada a subrutina CALC
        MOVE.L  (A7)+,D5        * Resultado a D5
        ADDQ    #8,A7           * Libera la pila
        ...
CALC    ADDA    #-12,A7         * Prepara A7 para el MOVEM
        MOVEM   D0-D1/A0,(A7)   * Guarda el contexto
        MOVE.L  24(A7),D0       * COEF1 en D0
        MOVEA.L 20(A7),A0       * Dirección de COEF2 en A0
        ...                     * Cálculo del resultado en D1
        MOVE.L  D1,16(A7)       * Salva resultado en pila
        MOVEM   (A7),D0-D1/A0   * Recupera el contexto
        ADDA    #12,A7          * Recupera la zona de la pila
        RTS                     * Fin de subrutina
        ...
```



Ejemplo 2.8

El objetivo de este ejemplo es entender cómo se hacen construcciones de alto nivel (tipo if-else, for ...) en ensamblador, para lo que se hace uso de las instrucciones de comparación y test y de las de salto condicional.

```
* If D3≠0 Then S1 Else S2

          TST.B   D3          * Test de D3 (a CCR)
          BEQ     ELSE        * Si es 0, ejecuta S2
          S1                  * Si no, ejecuta S1
          BRA     SALIR       * Se omite S2
ELSE      S2
SALIR

* While D6=0 do S3

REPETIR   TST.B   D6          * Test de D6 (a CCR)
          BNE     EXIT        * Si es 0, salir
          S3                  * Cuerpo del bucle
          BRA     REPETIR     * Repetir
EXIT

* For D5=N1 to D5=N2 do S4

          MOVE.L  #N1,D5      * D5 es el contador
BUCLE     S4                  * Cuerpo del bucle
          ADDQ    #1,D5       * Incrementa el contador
          CMPI    #N2+1,D5    * Comparación (a CCR)
          BNE     BUCLE       * Repetir
```

```
* Ejemplo 2 de otra manera

        TST.B  D3
        BNE    SALTO
        S2
        BRA    SALIR
SALTO   S1
SALIR   ...
```

Ejemplo 2.9

El objetivo de este ejemplo es adquirir soltura en la interpretación de programas en ensamblador del ColdFire. Hay que ser capaz de leer un fragmento de código (por ejemplo una subrutina) e interpretar cuales son los datos con los q trabaja, su función y los resultados que produce.

Diga cual es la función de los siguientes programas en ensamblador:

a)

```
                ...
BACKSPACE       EQU     $08             * Código ASCII de "backspace"
DELETE          EQU     $7F             * Código ASCII de "delete"
RETURN          EQU     $0D             * Código ASCII de "return"

                ORG     $4000           * Zona de memoria de datos
LINEABUF        DS.L    64              * Reserva longs para búfer

                ORG     $10000          * Zona de memoria de programa
                LEA     LINEABUF,A2     * Dirección a A2
                CLR.L   D0              * Borra todo D0
REPETIR         BSR     OBTEN_DATO      * Subrutina: Pone dato en D0
                CMP     #BACKSPACE,D0   * Ve si es "backspace"
                BEQ     IZQUIERDA       * Si lo es, se procesa (...)
                CMP     #DELETE,D0      * Ve si es "delete"
                BEQ     CANCELAR        * Si lo es, se procesa (...)
                CMP     #RETURN,D0      * Ve si es "return"
                BEQ     SALIR           * Si lo es, fin de línea (salir)
                MOVE.B  D0,(A2)+        * Si no, almacena D0 en búfer
                BRA     REPETIR         * Siguiente dato
                ...
                END
```

b)

```
                ...
                ORG     $25000                  * Memoria de programa
COMPARAR        LEA     CADENA1,A6              * Dirección a A6
                LEA     CADENA2,A5              * Dirección a A5
                CLR.B   D1                      * Para resultado
                CLR.L   D2                      * Para un caracter
                CLR.L   D3                      * Para el otro carácter
                MOVE.W  #(CADENA2-CADENA1),D0   * Longitud a D0
                BEQ     FINAL                   * Si vacía, acabar
BUCLE           MOVE.B  (A5)+,D2                * Carga carácter en D2
                MOVE.B  (A6)+,D3                * Carga carácter en D3
                CMP     D3,D2                   * Ve si son iguales
                BNE     FINAL                   * Si no, acabar
                SUBQ    #1,D0                   * Decrementar contador
                BNE     BUCLE                   * Si no es 0, repetir
                MOVE.B  #$FF,D1                 * Si lo es, $FF a D1
FINAL                                           * Acabar
                ...
                ORG     $80000                  * Memoria de datos
CADENA1         DC.B    "Cadena 1"
CADENA2         DC.B    "Cadena 2"
                END
```

c) ↳ Medir la longitud de una cadena de caracteres

```
                ...
RETURN          EQU     13                  * Código ASCII de "return"

                ORG     $25000              * Memoria de programa
PRINCIPAL       LEA     CADENA,A6           * Dirección a A6
                BSR     MEDIR               * Llamada a subrutina
FINAL                                       * Acabar
                ...
MEDIR           CLR.L   D1                  * Borrar todo D1
                MOVE.L  #$FFFFFFFF,D0       * D0 = -1
BUCLE           ADDQ    #1,D0               * D0 = D0+1
                MOVE.B  0(A6,D0.L),D1       * Carácter a D1
                CMPI    #RETURN,D1          * Ve si es "return"
                BNE     BUCLE               * Si no es, repetir
                RTS                         * Si es, acabar
                ...
                ORG     $80000              * Memoria de datos
CADENA          DC.B    "Soy una cadena"    * 14 bytes
                DC.B    RETURN              * 1 byte (fin de cadena)
                END
```

LEA CADENA, A6 ≠ MOVEA.L #CADENA, A6

↳ Cuenta nº de caracteres en la cadena a partir de direc. CADENA y lo devuelve en D0.

Ejercicio 4. Análisis de una rutina

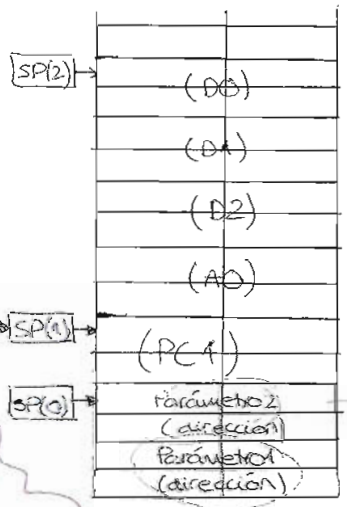
Se desea determinar la funcionalidad de la rutina CM que no incluye ningún tipo de documentación ni de comentarios.

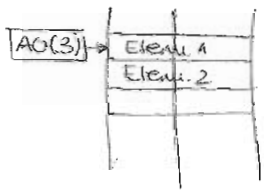
```
                org         $7000
                …
                bsr         CM
                …
CM              adda        #-16,A7
                movem       D0-D2/A0,(A7)
                clr.l       D0
                clr.l       D1
                movea.l     24(A7),A0
                move.l      #256,D2
CM_Bucle        move.w      (A0)+,D1
                ext.l       D1
                add         D1,D0
                subq        #1,D2
                bne         CM_Bucle
                asr         #8,D0
                movea.l     20(A7),A0
                move.w      D0,(A0)
                movem       (A7),D0-D2/A0
                adda        #16,A7
                rts
                …
                end
```

4.1. Explique brevemente la función de las dos instrucciones MOVEM utilizadas en el código.

4.2. Describa en pocas palabras la funcionalidad de la rutina.

1) guarda en la pila los reg. q. luego utilizamos en la subrutina xa conservar sus valores.
2) una vez q. utiliza reg. y termina subrutina, coge los valores y los lleva a los reg. xa dejarlos como estaban inicial.





Antes del rts el puntero de pila debe apuntar al valor del contador de programa, sino está mal

Es una subrutina q. calcula la media de 256 elem. word situados en memoria a partir de la direc. q. se le pasa como parámetro x la pila. Y la almacena en 1 direc. q. se le pasa como parámetro x la pila.

Ejercicio 5: Programación de una rutina

Un programa reserva un byte en la dirección HEX que contendrá valores hexadecimales en el rango $0-$F durante la ejecución. Se desea realizar una rutina llamada HEX_ASCII que convierta el contenido de HEX a código ASCII. El resultado de la conversión deberá almacenarse en la dirección ASCII. En la siguiente tabla se recogen los códigos ASCII correspondientes a los posibles valores hexadecimales en HEX.

HEX	ASCII	HEX	ASCII
$0	$30	$8	$38
$1	$31	$9	$39
$2	$32	$A	$41
$3	$33	$B	$42
$4	$34	$C	$43
$5	$35	$D	$44
$6	$36	$E	$45
$7	$37	$F	$46

5.1 Defina un algoritmo que realice la conversión de hexadecimal a código ASCII sin que sea necesario disponer de la tabla anterior.

5.2. Implemente el algoritmo anterior programando la rutina HEX_ASCII según la especificación del enunciado.

5.1

- Si HEX $\leq$ $9 $\Rightarrow$ ASCII = HEX + $30
- Si HEX > 9 $\Rightarrow$ ASCII = HEX + $37

$41 = 65

HEX = $A = 10

ASCII($A) = $A + $37 = 10 + 55 = 65

También:

Definimos N = HEX + $30

Si N $\leq$ $39 $\Rightarrow$ ASCII = N

Si N > $39 $\Rightarrow$ ASCII = N + $7

5.2

```
HEX_ASCII   CLR.L   D0
            MOVE.B  HEX, D0
            ADDI    #$30, D0      * (D0) = N
            CMPI    #$39, D0
            BLS     Salto         * Salta si N ≤ $39
            ADDI    #$7, D0       * (D0) = N + $7
Salto       MOVE.B  D0, ASCII
            RTS
```

Ejercicio 6. Preguntas cortas

6.1 Describa brevemente las diferencias entre las directivas de ensamblador DS y DC.

6.2 Empleando las instrucciones del MCF5272 y aritmética con signo, complete la secuencia de instrucciones para que sean equivalentes al pseudocódigo: while D0<D1 do D0=D0+1.

```
BUCLE:  CMP.L   D0,D1
        ____    FINAL
        ADDI    ____,D0
        BRA     BUCLE
FINAL:
```

6.3 Escriba la secuencia de instrucciones `MOVE` equivalentes a `MOVEM.L D0-D1,(A7)`

6.4 Escriba la secuencia de instrucciones `MOVE` equivalentes a:

```
ADDI    #-12,A7
MOVEM.L A0-A1/D2,(A7)
```

DS	DC
Reserva un nº de posiciones consecutivas en memoria = expresión	Reserva UNA posición de memoria y le asigna el valor inicial de expresión

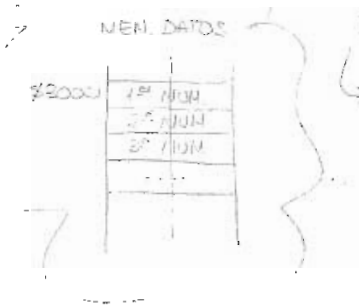
Ejercicio1. Detección de errores de programación

El siguiente fragmento de código pretende determinar el valor máximo de un conjunto de números en una tabla. Se trata de números con signo de tamaño word. Sin embargo, el código presenta dos errores y no funciona correctamente.

```
*******************************************************************************
* MAXIMO.ASM
* Determina el máximo de un conjunto de palabras con signo en una tabla
*******************************************************************************
              ORG           $2500
Principal     LEA           Tabla,A6
              MOVE.W        #Elementos,D0
              BSR           Maximo
              ...
*******************************************************************************
* Maximo - Determina el máximo de una tabla
* A6 = Dirección de comienzo de la tabla
* D0 = Tamaño de la tabla en número de palabras
* Devuelve en D1.W el valor máximo
*******************************************************************************
              ORG           $2A00
Maximo        CLR.L         D1
              CLR.L         D2
Lazo          MOVE.W        (A6)+,D2
              CMP           D2,D1
              BGE           Sigue
              MOVE.W        D2,D1
Sigue         SUBQ          #1,D0
              BNE           Lazo
              RTS

Tabla         EQU           $3000          * La tabla está en la dirección $3000
Elementos     EQU           300            * La tabla incluye 300 palabras
              END
```

1.1. Describa brevemente los dos errores presentes en el código anterior.

1.2. Justifique la necesidad de la instrucción CLR.L D2 al principio de la subrutina Maximo.



Para el 2º fallo habría q inicializar el reg D1 con el primer elemento de la tabla:

```
CLR.L   D1
MOVE.W  (A6)+, D1
CLR.L   D2
```

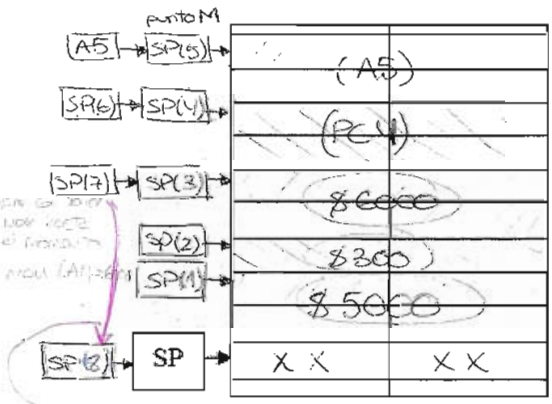
1.2. Como después de esta instr. hay un MOVE.W los bits superiores quedan como estaban, así q. los ponemos a cero con CLR.L D2. para no quedarnos con los bits superiores de otra instr

Ejercicio 2. Manejo de la pila

En el siguiente listado aparece la rutina del problema anterior con algunos cambios que no están relacionados con los errores mencionados anteriormente. En concreto, se ha modificado la llamada a la subrutina para pasar parámetros a través de la pila. Se ha señalado un punto M en el que se quiere conocer el estado que presenta la pila.

```
*************************************************************************
* MAXIMO.ASM
* Determina el máximo de un conjunto de palabras con signo en una tabla
* La subrutina escribe el valor obtenido en la posición de memoria Resultado
*************************************************************************
                ORG             $2500
Principal       PEA             Tabla
                MOVE.W          #Elementos,-(A7)
                PEA             Resultado
                BSR             Maximo
                ADDA.L          #10,A7
                …
*************************************************************************
* Maximo - Determina el máximo de una tabla
* …
*************************************************************************
                ORG             $2A00
Maximo          LINK            A5,#0
                MOVEA.L         8(A5),A1
                MOVE            12(A5),D0
                MOVEA.L         14(A5),A6
                CLR.L           D1              * Punto de ejecución M
                CLR.L           D2
Lazo            MOVE.W          (A6)+,D2
                CMP             D2,D1
                BCC             Sigue
                MOVE.W          D2,D1
Sigue           SUBQ            #1,D0
                BNE             Lazo
                MOVE            D1,(A1)
                UNLK            A5
                RTS

Tabla           EQU             $5000           * La tabla está en la dirección $5000
Elementos       EQU             300             * La tabla incluye 300 palabras
Resultado       EQU             $6000           * Dirección para el valor máximo
                END
```

2.1. Indique sobre la figura cuál es el contenido de la pila y la posición del puntero de pila cuando finaliza la ejecución de la instrucción en el punto M. Suponga que, al llegar a la etiqueta Principal, el puntero de pila apunta donde indica SP en la figura. Cada línea de la pila representa un tamaño word dividido en dos bytes.



2.2. Explique brevemente por qué se incluye la instrucción ADDA.L #10,A7 tras el salto a la subrutina Maximo.

Para vaciar la pila, ya q los datos q hay en esas posiciones eran necesarios xa la subrutina, pero no para el resto del programa.

Ejercicio3. Etiquetas y símbolos

El siguiente listado corresponde a un programa que verifica el funcionamiento de un bloque de memoria.

```
1  : 0000 0000 :                 : ********************************************
2  : 0000 0000 :                 : * VERMEM.ASM
3  : 0000 0000 :                 : * Verificación de un bloque de memoria
4  : 0000 0000 :                 : ********************************************
5  : 0000 0000 :                 :             org         $25000
6  : 0002 5000 :                 :
7  : 0002 5000 :                 : Longitud    equ         $FFFF
8  : 0002 5000 :                 : DirInic     equ         $26000
9  : 0002 5000 :                 :
10 : 0002 5000 : 303C FFFF       : Principal   move.w      #Longitud,D0
11 : 0002 5004 : 227C 0002 6000  :             movea.l     #DirInic,A1
12 : 0002 500A : 6100 0002       :             bsr         VerMem
13 : 0002 500E :                 : Final
14 : 0002 500E :                 : ********************************************
15 : 0002 500E :                 : * VerMem - Verifica un bloque de memoria
16 : 0002 500E :                 : * A1 = Dirección inicial del bloque
17 : 0002 500E :                 : * D0 = Longitud en bytes del bloque
18 : 0002 500E :                 : * Si hay fallo se llama a la subrutina Aviso
19 : 0002 500E :                 : ********************************************
20 : 0002 500E :                 :             external    Aviso
21 : 0002 500E :                 :
22 : 0002 500E : 4281            : VerMem      clr.l       D1
23 : 0002 5010 : 243C 0000 00FF  :             move.l      #$FF,D2
24 : 0002 5012 : 12BC 0000       : Verifica    move.b      #0,(A1)
25 : 0002 5018 : 4A11            :             tst.b       (A1)
26 : 0002 501A : 6600 0016       :             bne         Fallo
27 : 0002 501E : 12BC 00FF       :             move.b      #$FF,(A1)
28 : 0002 5024 : 1219            :             move.b      (A1)+,D1
29 : 0002 5026 : B441            :             cmp         D1,D2
30 : 0002 5028 : 6600 000A       :             bne         Fallo
31 : 0002 502C : 5340            :             subq        #1,D0
32 : 0002 502E : 66E6            :             bne         Verifica
33 : 0002 5030 : 6000 000C       :             bra         Salir
34 : 0002 5034 : 4EB9 0002 5040  : Fallo       jsr         Aviso
35 : 0002 503A : 5340            :             subq        #1,D0
36 : 0002 503C : 66D8            :             bne         Verifica
37 : 0002 503E : 4E75            : Salir       rts
38 : 0002 5040 :                 :
39 : 0002 5040 :                 :             end
```

3.1. Indique los valores hexadecimales de los siguientes símbolos o etiquetas:

VerMem = $ 25000E

Salir = $ 25003E

Fallo = $ 25034

Longitud = $ FFFF

3.2. En la línea 12 del listado aparece un salto a subrutina y en la línea 34 aparece otro. Explique brevemente las diferencias entre ambos.

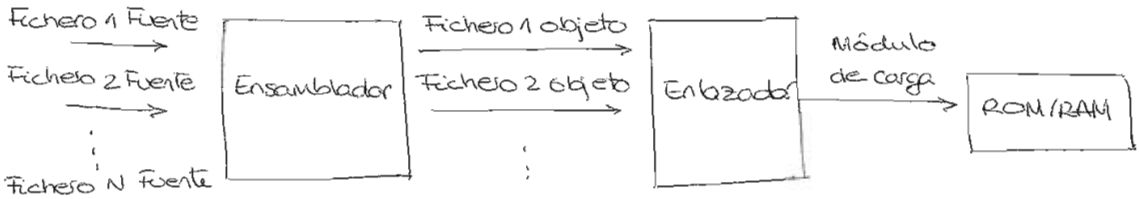
3.3. ¿Qué valor tiene Aviso? ¿De qué manera se calcula y en qué paso?

3.4. Explique brevemente el significado del código hexadecimal generado para la instrucción MOVE.L #DirInic,A1 en la línea 11.

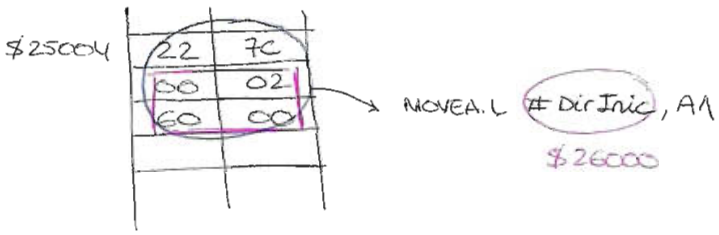
3.2 BSR: usa interna/ direccionam. interno relativo a PC, guarda el desplazam. respecto del PC desde la dirección de destino. La direc. de destino cercana al PC

JSR: usa direc. absoluta, se guarda la direc. de destino completa con 32 bits.

3.3 Aviso: subrutina definida en otro fichero (external), No se sabe q. direc. de mem. va a tener mem. Lo calcula el enlazador tras el proceso de ensamblado.



3.4



- el código de operación es: 227C (NOVEA.L con destino A1)
- operando inmediato: $ 00 02 60 00

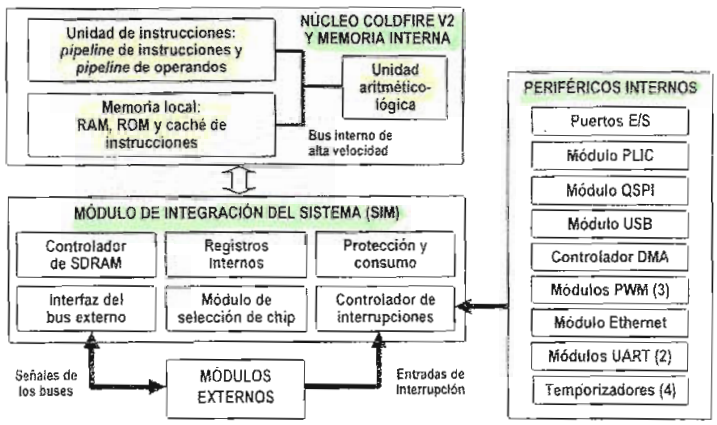
⊛ ¿Qué hace el programa?

cuando tenemos NOVEA.L #○, A es pq luego queremos acceder a mem. con un direccionam indirecto

3.1 Arquitectura del sistema
3.2 Terminales y señales externas
3.3 Configuración del sistema de memoria

3.1 Arquitectura del sistema

3.1.1 Diagrama de bloques del MCF5272



3.1.2 El núcleo ColdFire y memoria interna

El MCF5272 es un microcontrolador = núcleo (CPU) + memoria + periféricos

Características del núcleo ColdFire

- RISC (reduced instruction set computer) con instrucciones de longitud variable
- Arquitectura basada en dos pipelines desacoplados
- Bus de datos y direcciones de 32 bits dentro del chip

- Frecuencia de 66 MHz en el bus y en el procesador

Concepto genérico de pipeline

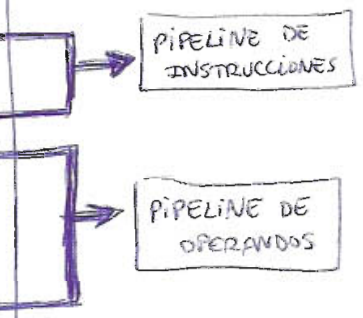
- Consideraremos que el procesamiento de cada instrucción se divide en etapas o tareas consecutivas (generar la dirección de la instrucción, obtener la instrucción, decodificar la instrucción, obtener los operandos, y operar o ejecutar)

- Un pipeline es una estructura que permite paralelizar tareas o etapas de distintas instrucciones en el tiempo

- Mientras se realiza una etapa de la instrucción i_k se puede realizar otra etapa de la instrucción i_{k+1} que no use los mismos recursos. Por eso se suelen paralelizar tareas de acceso a memoria con tareas de procesamiento (que no requieren acceder a memoria).
  - Por ejemplo: podemos traer de la memoria los operandos de ik y ejecutar dicha instrucción, a la vez que se decodifica y se obtienen las direcciones de los operandos de i_{k+1} para lo que no se necesita acceder a memoria.

Procesamiento instrucción:

1. Generar dirección de la instrucción
2. Obtener la instrucción
3. Decodificar la instrucción
4. Obtener operandos.
5. Ejecutar instrucción



THROUGHPUT = nº de instrucciones ejecutadas por segundo → viene limitado por la tarea más lenta.

- Ventajas de usar pipeline:
  - No se espera a terminar el procesamiento de i_k para empezar el de i_{k+1}
  - Se maximiza la utilización de recursos y el rendimiento
  - Aumenta el throughput o número de instrucciones ejecutadas por segundo

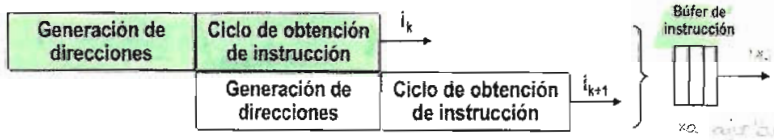
- Inconvenientes:
  - El throughput está limitado por la tarea más lenta
  - Aumenta la latencia (tiempo que tarda cada instrucción en ejecutarse)
  - Si se produce un salto condicional se invalida el contenido del pipeline y hay que volver a empezar desde la instrucción destino del salto.

El núcleo ColdFire tiene dos pipelines desacoplados por medio de un buffer

Pipeline de instrucciones del MCF5272

- Etapas o tareas:
  - Generación de direcciones: se calcula la dirección de memoria de la instrucción siguiente
  - Ciclo de obtención de instrucción: se obtiene de memoria la instrucción actual a procesar.

- Buffer de instrucción: el pipeline de instrucción puede obtener instrucciones antes de que puedan ser ejecutadas. Las instrucciones se van almacenando en una cola FIFO de 3 longs para ajustar la velocidad con la del siguiente pipeline (el de ejecución o de operandos). Aún así, si la memoria es lenta y se tarda mucho en obtener cada instrucción de la memoria, se puede retener la ejecución dando lugar a tiempos muertos en el segundo pipeline.

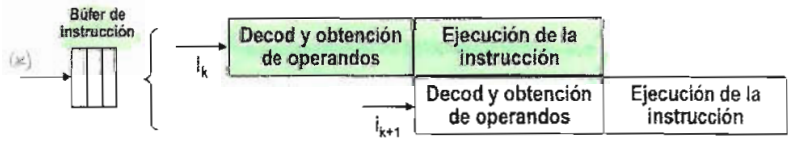


Pipeline de operandos o ejecución del MCF5272

- El funcionamiento es dependiente del tipo de instrucción (registro a registro, memoria a registro y registro a memoria).

- Etapas o tareas en el caso registro-registro (el más sencillo)
  - Decodificación y obtención de operandos (que están en registros internos de la CPU, por lo que no hace falta acceder a memoria)
  - Ejecución de la instrucción

A pesar de que el ColdFire internamente funcione con pipelines, a la hora de interpretar programas no conviene pensar en esto, a no ser que pregunten específicamente sobre ello.



Temporización de instrucciones: $C(r,w)$ → nº de accesos de lectura / escritura necesarios

- El tiempo que tarda en ejecutarse una instrucción no es fijo
  - El tiempo de acceso a memoria puede ser variable
  - El estado de los pipelines de la CPU cambia durante la ejecución
- El tiempo mínimo de ejecución de una instrucción se produce si:
  1) El pipeline de operandos funciona a pleno rendimiento
     - No tiene que esperar al pipeline de instrucciones
     - No hay retenciones debidas a la secuencia de operaciones.
  2) La memoria no provoca retardos (no hay estados de espera)
  3) Todos los operandos están correctamente alineados
     - Word en direcciones pares.
     - Long en direcciones multiplo de 4.

La memoria local

- Memoria interna (dentro del chip) del MCF5272

  - SRAM de 4 KB: RAM estática, más rápida que la dinámica pero más cara también. Por eso se utiliza para las variables y datos de programa a los que más se accede.

La ROM interna del ColdFire viene ya escrita de fábrica. No es para que pongamos nuestros programas

  - ROM de 16 KB: que contiene código de protocolos para las comunicaciones de algunos módulos. (código xa q funcionen interfaces de E/S)

Recordar la memoria oculta de FDOR.

  - Caché de instrucciones de 1 KB: contiene las últimas instrucciones ejecutadas y acelera la búsqueda de instrucciones.

La ventaja de la memoria interna es que el tiempo de acceso es mucho menor que para memorias externas. Los accesos sólo tardan un ciclo porque se usa el bus interno de alta velocidad.

3.1.3 El módulo de integración del sistema (SIM) System Internal Module

- Coordina la interacción entre la CPU y los periféricos internos y externos
- Configura el sistema según el mapa de memoria → generando señales de CS (8)
- Configura el tamaño del bus de datos para 8, 16 ó 32 bits (lo tiene q ver con .L, .W, .B)
- La configuración se realiza a través de los registros del SIM
- Componentes del SIM:

Apartado 3.3

  - Módulo de selección de chip: permite generar 8 señales de chip-select programables sin necesidad de hardware externo

Apartado 3.3 y Tema 7

  - Interfaz con memorias externas SRAM y ROM de 8, 16 y 32 bits. Es programable y permite introducir estados de espera si las memorias externas utilizadas son lentas.

Tema 7

  - Controlador de SDRAM externa (synchronous dynamic RAM) permite capacidades + grandes, + lenta.

Tema 4

  - Controlador de interrupciones: controla las interrupciones de los periféricos internos del MCF5272 y además tiene 6 entradas para interrupciones de periféricos externos.

Tema 4

  - Protección del sistema mediante Watchdog: detecta si el sistema no funciona bien mediante mecanismos hardware y software. Por ejemplo: bucle infinito.

Tema 4

  - Gestión del consumo: ahorra baterías. → sleep (CPU) / Stop (CPU + periféricos internos) [inhabilita reloj de periféricos no usados]

Se ven en el tema 5

3.1.4 Módulos de entrada salida = PERIFÉRICOS INTERNOS

- 3 Puertos de propósito general (A, B, C) de entrada/salida paralelo de 16 bits
- 2 módulos UART de comunicaciones serie.
- Interfaz con periféricos serie con colas QPSI
- Bus serie universal USB
- Módulo Ethernet

ADM en FDOR

- Controlador de interfaz de nivel físico PLIC
- Controlador de acceso directo a memoria DMA

Se ven en el Tema 6

3.1.5 Módulos de temporización y PWM

- Módulo de temporización: 4 temporizadores con interrupciones independientes
- Unidad de PWM: 3 canales independientes, que proporcionan señales periódicas rectangulares de ciclo de trabajo programable.

Esto puede caer como teoría en forma de preguntas cortas

3.1.6 Módulos de diagnóstico y depuración

JTAG: Join Test Action Group

- Módulo JTAG: para la prueba de tarjetas de circuito impreso de alta densidad. Permite por ejemplo comprobar la continuidad eléctrica de la tarjeta y tomar muestras de los terminales del MCF5272 durante su funcionamiento.

BDM: Background Debug Mode

- Modo de depuración en segundo plano (BDM):
  - Proporciona depuración a bajo nivel a través de una interfaz serie
  - ¡¡¡El procesador se detiene!!! ⇒ no permite evaluar prestaciones
  - Se puede leer y escribir en memoria y en registros y en bloques de memoria completos cuando el procesador está detenido.

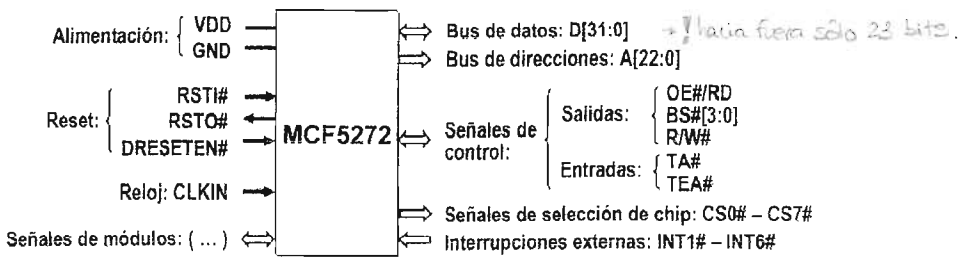
- Seguimiento en tiempo real:
  - ¡¡¡No se detiene el procesador!!!
  - Se pueden obtener trazas mediante un puerto paralelo de 8 bits donde se presentan el estado y los datos de la ejecución (según se haya configurado previamente).

- Depuración en tiempo real:
  - Basado en interrupciones de depuración en las que se guarda el contenido de variables y registros claves.
  - ¡¡¡ No se detiene el procesador!!!
  - Para programarlo se utiliza el BDM

3.2 Terminales y señales externas

Eso quiere decir que cada terminal puede tener varias funciones distintas (hasta 4 en este caso) según lo configures. Esto es para ahorrar terminales.

- El MCF5271 tiene 14x14=196 terminales (patillas) multiplexados



3.2.1 Bus de direcciones externo y señales de chip-select

¡OJO! El bus de direcciones interno tiene 32 bits y el externo 23.

- Bus de direcciones externo: A[22:0]
  - Es unidireccional triestado y tiene 23 bits (direcciona hasta 8 MB). Son los 23 bits menos significativos del bus interno de 32.

- Señales de selección de chip: CS0# - CS7#
  - Esencial para definir el mapa de memoria del sistema

- Se configura un rango de direcciones por cada CSn#
- El CSn# se activa si la dirección de 32 bits en el bus interno pertenece al rango que tiene configurado y activa la pastilla de memoria o interfaz/periférico correspondiente. Una vez hecho esto, puedes direccionar 8 MB en el dispositivo activado, con A[22:0].

Se ve en el Tema 7

- La memoria SDRAM funciona de manera diferente. Se puede configurar el CS7# para el controlador de SDRAM del módulo SIM. En ese caso A[22:0] se convierte SDA[13:0] (terminales mutliplexados). De esta forma se pueden manejar memorias de mayor tamaño.

3.2.2 Bus de datos externo D[31:0]

- Es bidireccional triestado de 32 bits
- Los terminales D[15:0] están multiplexados con el puerto C de E/S
- El tamaño se puede configurar a 32 ó 16 bits:
  - Tamaño 32 bits: hay que renunciar al puerto C, pero ¡¡son más rápidas las transferencias a periféricos y memorias de 32 bits!!.
  - Tamaño 16 bits: D[31:16] se dedican al bus de datos mientras que D[15:0] se utilizan como puerto C

No hay que confundir el tamaño del acceso que puede ser .B, .W o .L con el tamaño del bus (16 ó 32 bits). Con un bus de 16 bits puedes seguir transfiriendo una palabra .L sólo que requerirás dos pasadas y será más lento.

- El tamaño que tendrá el bus de datos se configura durante el reset dependiendo del nivel que pongamos en el terminal WSEL (multiplexado con QSPI_Dout)
  - WSEL=Nivel alto: bus de 32 bits en D[31:0] alineado con direcciones múltiplo de 4
  - WSEL=Nivel bajo: bus de 16 bits en D[31:16] alineado con direcciones pares

3.2.3 Señales de control del bus externo

Es de salida

- Habilitación de salida/Lectura: OE#/RD
  - Señal de salida que indica la dirección de la transferencia en el bus de datos
  - Se suele conectar a los terminales OE# de las memorias externas
  - Si OE#/RD=Nivel bajo ⇒ la memoria activa sus terminales de salida. La memoria "escribe" el dato en el bus de datos.
  - Si OE#/RD=Nivel alto ⇒ la memoria "lee" el dato del bus de datos. En este caso no es necesario que active sus terminales.

Es de salida

- Activación de byte: BS#[3:0]
  - Señales de salida que describen la posición de los datos en el bus (una por byte)
  - Sólo se activan las correspondientes a los bytes del bus de datos utilizados.

Observar que con bus de 32 bits:
D[31:24] ↔BS0#
D[23:16] ↔BS1#
D[15:8] ↔BS2#
D[7:0] ↔BS3#

Con bus de 16 bits:
D[31:24] ↔BS0#
D[23:16] ↔BS1#

Bus de datos de 16 bits

BS1#	BS0#	Acceso	Datos en:
1	1	Ninguno	-
1	0	Byte	D[31:24]
0	1		D[23:16]
0	0	Word	D[31:16]

Bus de datos de 32 bits

BS3#	BS2#	BS1#	BS0#	Acceso	Datos en:
1	1	1	1	Ninguno	-
1	1	1	0	Byte	D[31:24]
1	1	0	1		D[23:16]
1	0	1	1		D[15:8]
0	1	1	1		D[7:0]
1	1	0	0	Word	D[31:16]
0	0	1	1		D[15:0]
0	0	0	0	Long	D[31:0]

Es de salida

- Lectura/Escritura: R/W#
  - Señal de salida que se conecta a la entrada WE# (Habilitación de escritura) o R/W# de las memorias o periféricos externos.
  - Se programa para chip-Select por separado (apartado 3.3).

Es entrada

- Confirmación de transferencia: TA#
  - Señal de entrada que indica la terminación de un ciclo de bus externamente
  - Está multiplexada con el bit 5 del puerto B (PB5)
  - Se programa para cada chip-select por separado.
  - Sólo se utiliza con memorias lentas y periféricos asíncronos con tiempo de acceso excesivo.

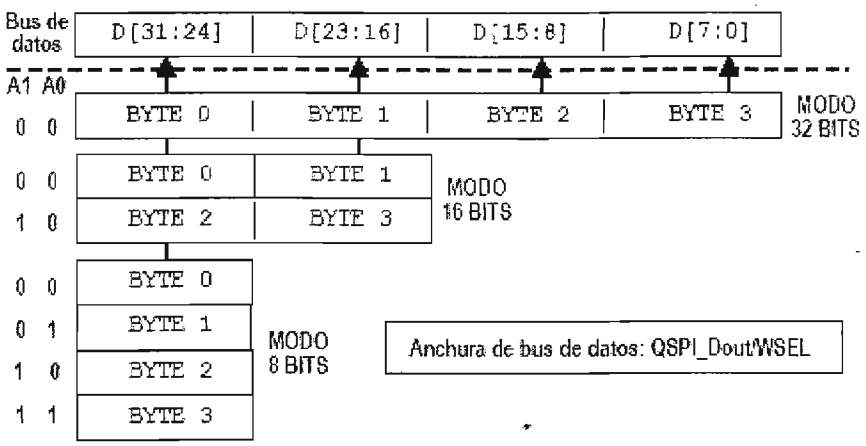
Es entrada

- Confirmación de error de transferencia: TEA#
  - Señal de entrada que indica un error en la transferencia actual del bus.
  - La activa el dispositivo esclavo externo que detecta el error para informar al MCF5272.

3.2.4 Modos de transferencia con los puertos

- Los puertos pueden ser de 32, 16 u 8 bits.
- El MCF5272 adapta el tamaño del bus y de los datos, al de los puertos:
  - Modo de 8 bits: sitúa los datos en D[31:24] que se conectan a l puerto de 8 bits
  - Modo de 16 bits: sitúa los datos en D[31:16] que se conectan al puerto de 16 bits
  - Modo de 32 bits: sitúa los datos en D[31:0] que se conectan al puerto de 32 bits

¡OJO! El modo de 32 bits sólo funciona si el bus está configurado con tamaño de 32 bits.



3.2.5 Transferencias de líneas y ráfagas

Esto puede caer como "teoría" en forma de pregunta corta

- Transferencia de línea:
  - Una línea equivale a 16 bytes (4 longs)
  - El MCF5272 está optimizado para esas transferencias ⇒ se ahorran ciclos
  - Tienen que estar alineadas con un límite de memoria de long (la dirección de inicio debe ser múltiplo de 4)

- Transferencias en ráfagas
  - Transferencia de una o varias líneas consecutivamente
  - A partir de la segunda línea la optimización es mayor
  - Se utilizan para instrucciones MOVEM, movimientos de líneas de caché y transferencias de acceso directo a memoria (DMA)

3.2.6 Señales de Reset

Es de entrada

- Entrada de Reset: RSTI#
  - Al activarla se reinician inmediatamente la CPU y los periféricos
  - Provoca que se active RSTO# durante 32K ciclos de reloj

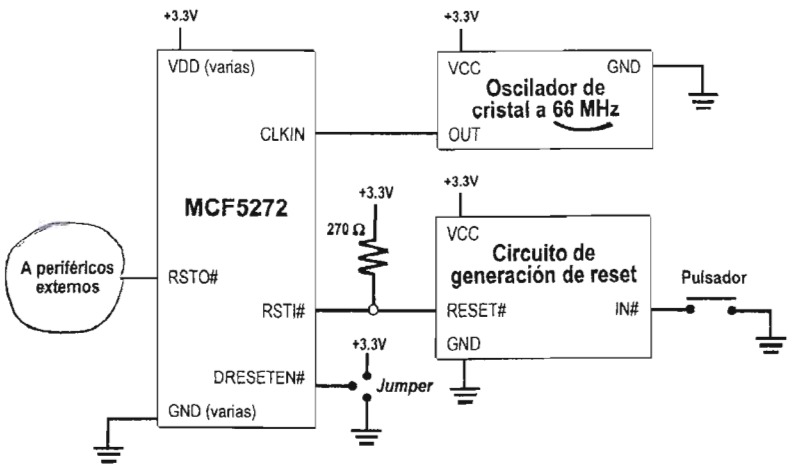
Es de entrada

- Habilitación de Reset de DRAM: DRESETEN#
  - Señal de entrada que controla el reset del controlador de SDRAM
  - Si DRESETEN# =nivel bajo (activa), cuando se activa RSTI# se reinicia el controlador de SDRAM.
  - Si DRESETEN# = nivel alto (inactiva), cuando se activa RSTI# no afecta a la SDRAM que mantiene su contenido.

Es de salida

- Salida de Reset: RSTO#
  - Se usa para reiniciar los módulos externos desde la CPU. Se conecta a las entradas de reset de estos.
  - Se activa (nivel bajo) durante 32 K ciclos de reloj al aplicar un nivel bajo en RSTI#

3.2.7 Circuitos de alimentación, reloj y reset



3.3 Configuración del sistema de memoria

3.3.1 Dispositivos internos

- Recursos definidos en el mapa de memoria. Su posición en el mapa se configura en registros internos de 32 bits. Son:

  - SIM y los módulos internos ocupan una zona de 64 KB en el mapa de memoria y se define su posición en el mismo con el registro MBAR
  -Memoria RAM de 4 KB se define su posición en el mapa de memoria con el registro RAMBAR
  - Memoria ROM de 16 KB se define su posición en el mapa de memoria con el registro ROMBAR

- Recursos no definidos en el mapa de memoria:
  - Registros de la CPU con nombres reservados: Dn, An, PC, SR, CCR. Se acceden usando su nombre como dirección

  - Registros de la CPU con direcciones de CPU:
    - ¡¡¡No están incluidos en el mapa de memoria. El número que les identifica no es una dirección de memoria sino de CPU!!!
    - ¡¡¡Se utiliza MOVEC para acceder a ellos!!!
    - Son:
      - **MBAR**, RAMBAR y ROMBAR (configuración de direcciones)
      - VBR (configuración de interrupciones)
      - CACR, ACR0, ACR1 (configuración de la caché)

¡OJO! MBAR es especial. Una vez configurado sí forma parte del mapa de memoria

Tema 4
Tema 7

Configuración de la dirección base del módulo SIM

MBAR: Module Base Address Register

Se hace mediante el registro de dirección base del SIM: MBAR

- Se accede con la instrucción MOVEC a la dirección de CPU $C0F
- La dirección que definamos como base del SIM va a ser la dirección que se le asigne a este registro dentro del mapa de memoria (ver anotación al margen en negrita de la página anterior)
- Campos del registro:

SIM: 2^{16} = 64 KB

* una vez q. se define se trata como direc de mem

- BA (base address): dirección base del SIM (los demás registros de los módulos internos se situarán a partir de ella). Sólo se definen los 16 bits más significativos de la dirección y deja los 16 menos significativos sin definir ⇒ asigna un rango de 2^{16}=64 KB al SIM y los módulos internos.

- Máscaras que habilitan o no los accesos (menos consumo)
  - SC=1 inhabilita accesos a código de supervisor
  - SD=1 inhabilita accesos a datos de supervisor
  - UC=1 inhabilita accesos a código de usuario
  - UD=1 inhabilita accesos a datos de usuario

- V=1 validación del módulo SIM → Se debe validar tras el reset, al configurar el valor correcto de MBAR

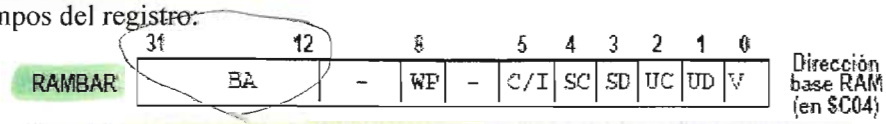
Configuración de la memoria RAM y ROM local

RAMBAR: RAM Base Address Register

RAM local: Se hace mediante el registro de dirección base de la RAM interna, RAMBAR

- Se accede con instrucción MOVEC a la dirección de CPU $C04
- Campos del registro:

RAM: 2^{12} = 4KB



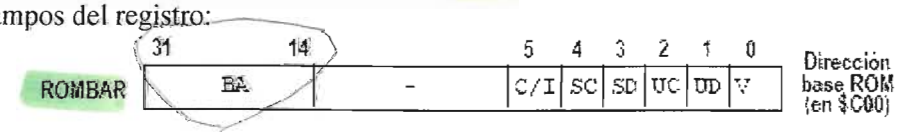
- BA: dirección base de la RAM interna. Sólo se definen los 20 bits más significativos dejando los 12 menos significativos sin definir porque es una memoria de 2^{12}=4 KB
- WP=1 protección frente a escritura
- Máscaras que habilitan o no los accesos a RAM:
  - C/I: registros de la CPU / ciclo de interrupción (¡¡casi siempre se pone a 1!!)
  - SC, SD, UC, UD: igual que en RAMBAR
- V=1 validación de la RAM → validar tras reset

ROMBAR:ROM Base Address Register

ROM local: Se hace mediante el registro de dirección base de la ROM interna, ROMBAR

- Se accede con instrucción MOVEC ala dirección de CPU $C00
- Campos del registro:

ROM: 2^{14} = 16kB



- Los mismos que RAMBAR pero sin WP

3.3.2 Dispositivos externos

En el caso de RAM y ROM externas, y de periféricos conectados directamente al bus externo, su posición en el mapa de memoria se configura mediante señales de chip-select

- Memoria RAM externa: opcional (si los 4GB RAM internos son insuficientes)
- Memoria ROM externa: obligatoria, debe contener al menos la rutina de inicialización del sistema tras el reset.
- Periféricos externos:

  - Se pueden conectar directamente al bus externo. En este caso tiene rangos de direcciones asignados en el mapa de memoria y se accede a ellos como si fuesen direcciones de memoria. Por ejemplo: un módulo de E/S adicional.

  - Es habitual conectarlos a los módulos de E/S internos del MCF5272. En ese caso no se accede directamente a los periféricos que no tienen direcciones asignadas, sino al rango de memoria asignado a los registros del módulo de E/S interno al que están conectados.

El módulo de selección de chip del MCF5272 (proporciona 8 salidas independientes de selección de chip / proporciona 4 señales BS# del bus externo.)

- Genera las señales de CS0#-CS7# cada una de la cuales tiene asociado un rango de memoria.
  - Se activan si la dirección a la que se intenta acceder está en ese rango.
  - Sólo se debe activar una señal de CSn# en todo el sistema simultáneamente

Si no, varios dispositivos podrían intentar escribir simultáneamente en el bus de datos

CS0#: se accede a la ROM de inicio del sistema tras reset
CS7#: se puede acceder a SDRAM externa

A[22:0] para A de SDA[13:0]

- Además es el encargado de generar las señales BS#[3:0]
- El CSn# se activa si la dirección de 32 bits en el bus interno pertenece al rango que tiene configurado y activa la pastilla de memoria o interfaz/periférico correspondiente. Una vez hecho esto, puedes direccionar 8 MB en el dispositivo activado, con A[22:0].
- Registros de base CSBR0-CSBR7: definen las direcciones de las zonas asociadas a cada CSn#

CSBR: Chip Select Base Register

32 bits

	31 ... 12	11	10	9	8	7 ... 0
CSBR0-CSBR7	BA	EBI		BW	SUPER	- ENABLE

  - BA: dirección base o inicial del espacio de memoria asignado a CSn#
  - EBI: modos de interfaz con el bus externo (tamaño del puerto al q se quiere acceder)
    - 00: memorias SRAM o ROM de 16/32 bits con señales de activación de byte
    - 01: SDRAM (sólo para CSBR7)
    - 10: reservado
    - 11: memorias SRAM o ROM de 8 bits sin señales de activación de byte
  - BW: anchura de bus (puerto) asociado a CSn# (menos CS0# en reset)
    - 00: long (32 bits)
    - 01: byte (8 bits)
    - 10: word (16 bits)
    - 11: línea de caché (32 bits)
  - SUPER=1 activa CSn# sólo en modo supervisor

Ver caso especial: memoria SDRAM en la página siguiente

Ver caso especial: arranque del sistema en la página siguiente

Para limitar el acceso al dispositivo externo si estás en modo usuario

  - ENABLE=1: habilita el uso de la salida CSn# (Normalmente valor 1)

CSOR: Chip Select Options Register

BAM: Base Address Mask

¡OJO! En WS a la primera cifra hexadecimal sólo le corresponde un bit (suficiente porque sólo puede ser 0 ó 1)

- Registros de opciones CSOR0-CSOR7: definen los rangos de los espacios de memoria asociados a las CSn#

	31	12	6	2	1	0
CSOR0-CSOR7	BAM	-	WS		RW	MRW

32 bits

-BAM: máscara de dirección de base. Se ponen a 1 los bits que no cambian en todo el rango asignado al CSn# Se coge dirección inicial y final del rango del CS y los bits q cambian=0 los q no cambian=1

- WS: estados de espera (ciclos de reloj) antes de finalizar el acceso. Para memorias lentas y periféricos con tiempo de acceso elevado
  - ♦ $00-$1E: número de estados de espera.
  - ♦ $1F: terminación externa. El final del acceso lo determina la activación de la entrada TA#

en realidad WS 5 bits

- RW: memoria de sólo lectura (RW=0) o sólo escritura (RW=1)
- MRW: lectura/escritura en memorias:
  - ♦ 0: memoria de lectura/escritura ⇒ valor de RW indiferente
  - ♦ 1: memoria de sólo lectura o sólo escritura, según indique RW

- Casos especiales:
  - Arranque del sistema:
    - ♦ Tras el reset se debe ejecutar una rutina de inicialización que tiene que estar en la ROM externa. La señal de CS0# está asignada a esa ROM externa y está habilitada tras el reset.
    - ♦ Si aún no están configurados los registros (lo que se hace en la rutina de reset) ¿cómo se configura el CS0# que accede a la ROM externa?
    - ♦ El campo EBI del CSBR está a 00 tras el reset (memorias SRAM o ROM de 16/32 bits)
    - ♦ El campo BW del CSBR toma valores dependiendo de los niveles que tengan los terminales BUSW1 y BUSW0 durante el reset:

Anchura de bus para CS0#

BUSW1	BUSW0	Bus CS0#
0	0	32 bits
0	1	16 bits
1	0	8 bits
1	1	-

- Uso de SDRAM: Se puede configurar el CS7# para el controlador de SDRAM del módulo SIM. En ese caso A[22:0] se convierte SDA[13:0] (terminales multiplexados). Entonces el campo EBI del CSBR7 toma el valor 01

WS = $1F

Capacidad de direccionamiento = máxima cantidad de memoria que puedo conectar al coldfire.

23 bits = 32 bits

$2^{23} = 8MB$ → Memoria externa TOTAL 8 CS x 8MB

Programación de un mapa de memoria

1) Especificar dirección de INI y FIN
2) Pasar direcciones a binario
3) Comparar con máscara 1 ¿cuales no cambian?

No cambian → 1.
Cambian → 0

No puedo especificar rango <4KB.

Ej: INI $ 0100 0000
FIN $ 017F ~~FF~~ FF

INI = 0000 0001 0000 0000 0000 0000 0000 0000
FIN = 0000 0001 0111 1111 1111 1111 1111 1111

BAM = 1111 1111 1000 0000 0000 0000 0000 0000

→ Da igual siempre pondremos 0

BAM = $ F F 8 0 0 0 0 0

Ejemplo 3.1

Teniendo en cuenta que el tiempo que tarda en ejecutarse una instrucción no es fijo debido a que el tiempo de acceso a memoria es variable (dependiendo del tipo de memoria) y el estado de los pipelines de la CPU cambia durante la ejecución, diga qué condiciones tendrán que cumplirse para que el tiempo de ejecución de una instrucción fuese el mínimo en un caso genérico.

1) El pipeline de operandos deberá trabajar a pleno rendimiento, ⇒ no haya tiempos muertos debido a q. no le lleguen instr. desde el pipeline de instr. al ritmo suficiente.

No haya tiempos muertos ⇒ Throughput pipeline instr. ≥ throughput pipeline operandos

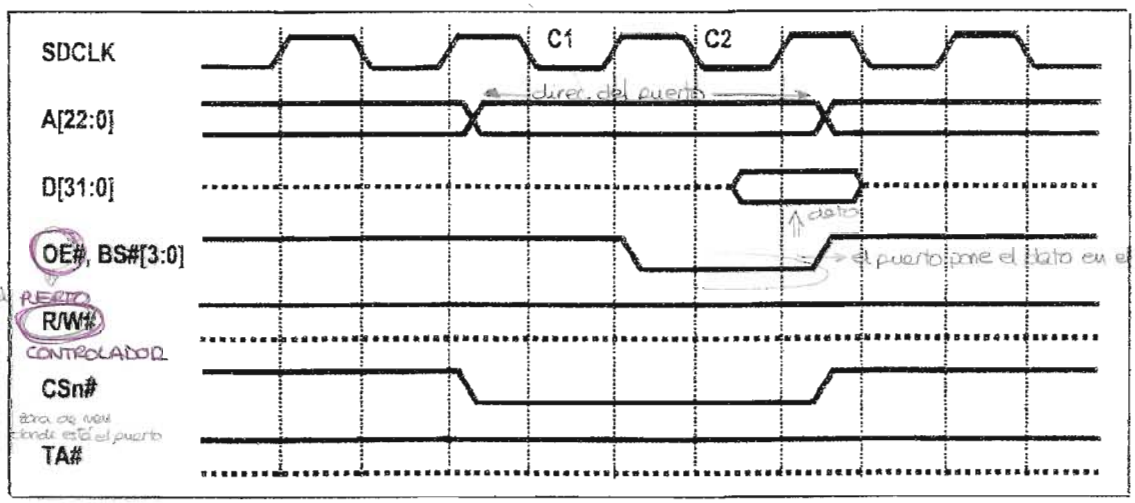
tiempo de ejec. mín.

2) operandos estén correctamente alineados en la memoria
(.L en dir. múltiplo de 4, .W en dir. par y .B en cualquier dir)

3) La memoria debe ser lo suficiente rápida xa q. la CPU no tenga q. introducir estados de espera.

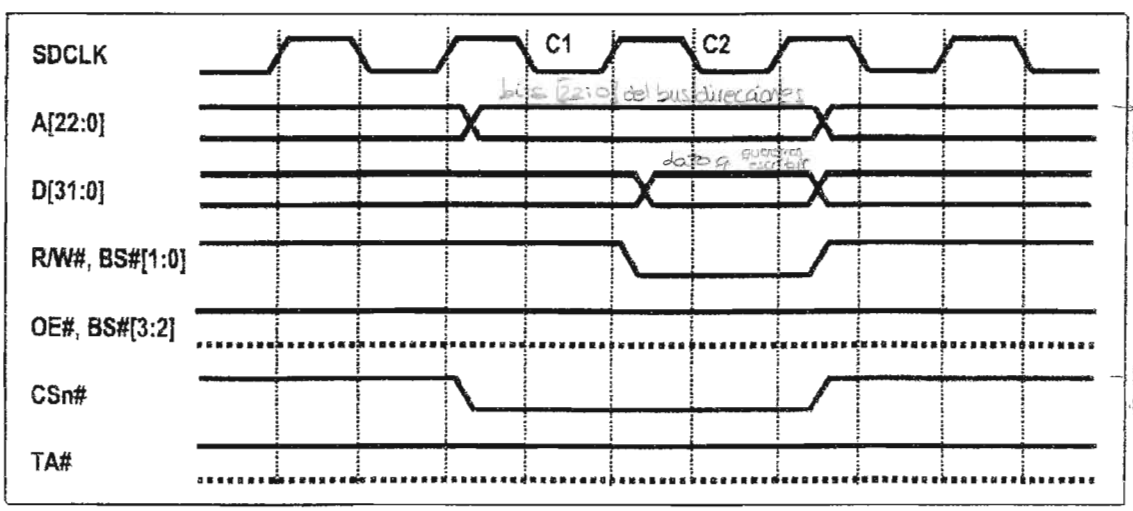
Ejemplo 3.2

a) El siguiente cronograma muestra el valor de las distintas señales implicadas en un acceso de lectura de un puerto, con las siguientes características:
- Acceso de tamaño .L
- Tamaño del bus de datos: 32 bits
- Tamaño del puerto: 32 bits
- Terminación interna



Se pide que explique el significado de los valores que toman las señales

b) El siguiente cronograma muestra el valor de las distintas señales implicadas en un acceso de escritura en un puerto, con las siguientes características:
- Acceso de tamaño .W
- Tamaño del bus de datos: 32 bits
- Tamaño del puerto: 16 bits
- Terminación interna



Se pide que explique el significado de los valores que toman las señales

Ejemplo 3.3

Se pide programar el mapa de memoria siguiente:

MAPA FUNCIONAL	MAPA FISICO	DIRECCIONES Hex	DIRECCIONES Binario
Programa y vectores interrupción	ROM externa 16 KB	00000000 00003FFF	0 ... 0 0000 0000 0000 0000 0000 0 ... 0 0000 0011 1111 1111 1111
Variables y pila	RAM local 4 KB	00004000 00004FFF	0 ... 0 0000 0100 0000 0000 0000 0 ... 0 0000 0100 1111 1111 1111
Entrada/salida	Módulo SIM 64 KB	00010000 0001FFFF	0 ... 0 0001 0000 0000 0000 0000 0 ... 0 0001 1111 1111 1111 1111
Entrada/salida	Otros dispositivos E/S 64 KB	00020000 0002FFFF	0 ... 0 0010 0000 0000 0000 0000 0 ... 0 0010 1111 1111 1111 1111
Variables menos usadas	RAM externa 256 KB	00040000 0007FFFF	0 ... 0 0100 0000 0000 0000 0000 0 ... 0 0111 1111 1111 1111 1111

sabiendo que:

- Todos los buses y puertos son de 32 bits
- Los accesos a RAM externa precisan un estado de espera (el resto no)

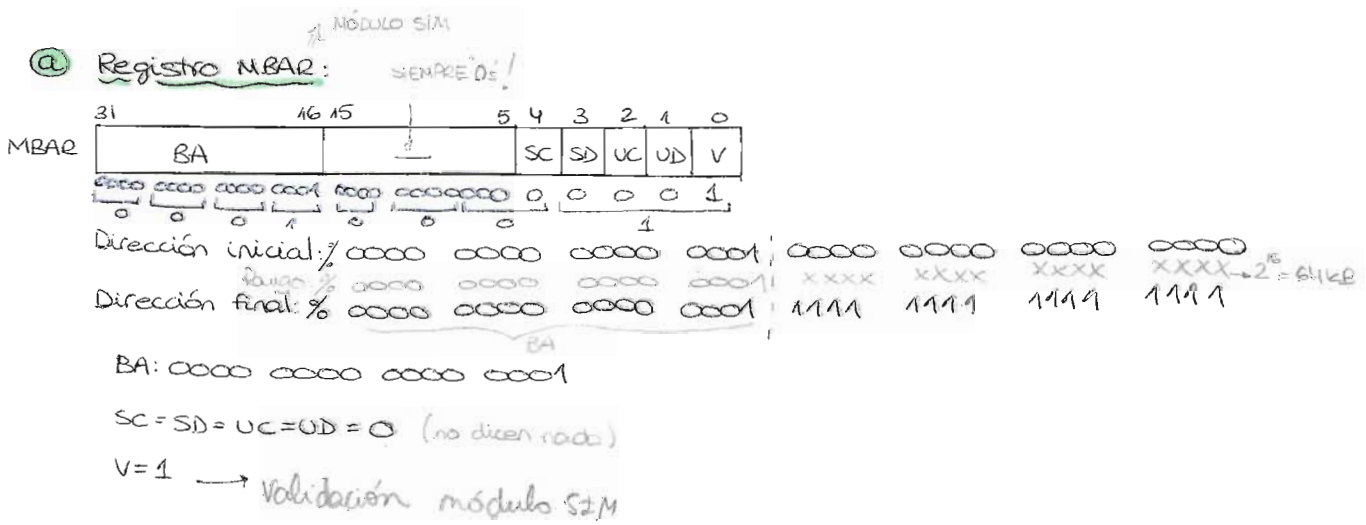
Para ello siga los siguientes pasos:

a) Calcule el valor que debe tener el registro MBAR

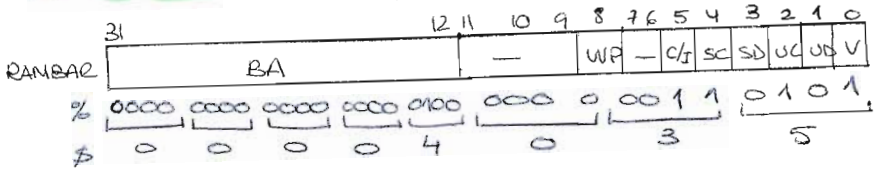
b) Configure la RAM local mediante RAMBAR

c) Configure los registros CSBRn y CSORn necesarios para generar las señales de chip-select para los módulos externos.

d) Una vez calculados los valores de los registros, escriba el código utilizaría para configurarlos.
Sugerencia: use directivas EQU para definir las direcciones que corresponden a cada registro para que el código sea más legible, sabiendo que el registro CSBR0 tiene la dirección de base del módulo SIM + $40, el CSOR0 + $44, el CSBR1 + $48 y así sucesivamente...



(b) RAMBAR (RAM local)



% 0000 0000 0000 0000 0100 000 0 00 1 1 0 1 0 1

$ 0 0 0 0 4 0 3 5

Dirección inicial: % 0000 0000 0000 0000 0100 0000 0000 0000

Rango: % 0000 0000 0000 0000 0100 xxxx xxxx xxxx

Dirección final: % 0000 0000 0000 0000 0100 1111 1111 1111

BA

WP = 0 (lectura/escritura)

C/I = 1 → casi siempre se pone a 1 (registros de la CPU / ciclo de interrupción)

SC = UC = 1 → RAM variables y pila, se inhabilita xq si nadie accede a pila (ninguno podrían ponerse a cero)

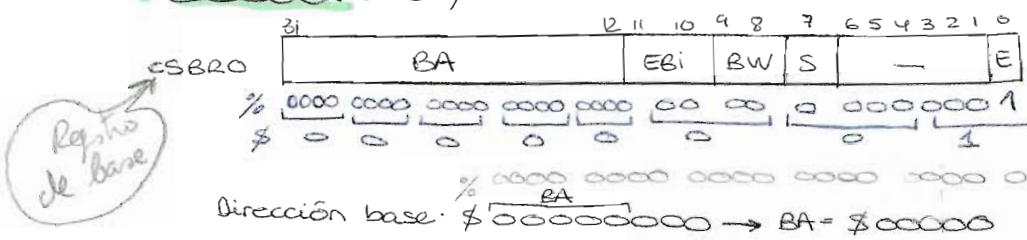
SD = UD = 0

V = 1

(c)

¿ tamaño del puerto = EBi
tamaño del bus = BW

- ROM externa (CS0#)



CSBR0 = $ 00000001

Dirección base: % BA 0000 0000 0000 0000 0000 0000 0000 0000

$ 00000000 → BA = $ 00000

} 2^{14} = 16 KB

Dirección final: % 0000 0000 0000 0000 0011 1111 1111 1111

$ 0000BFFF → BAM = $ FFFFC

1 = no cambian
0 = cambian

% 1111 1111 1111 1111 1100 0000 0000 0000

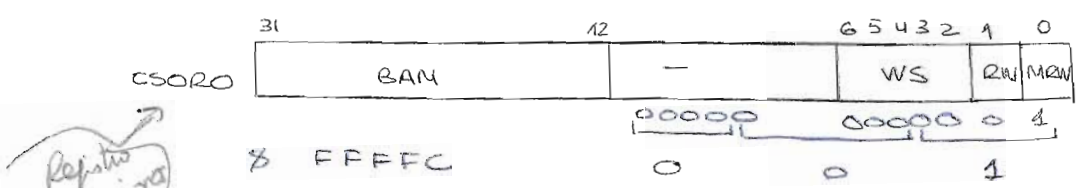
BAM (5 bytes) = $ FFFFC

Archivo puerto → EBi = % 00 (SRAM o ROM 16/32 bits) (Enunciado: puertas de 32 bits)

Archivo bus → BW = % 00 (Bus de 32 bits)

SUPER = % 0 (Activo siempre)

ENABLE = % 1 (Habilitado)



CSOR0 = $ FFFFC001

BAM = $ FFFFC

WS = % 00000

RW = % 0 (Sólo lectura)

MRW = % 1 (lo q indique RW)

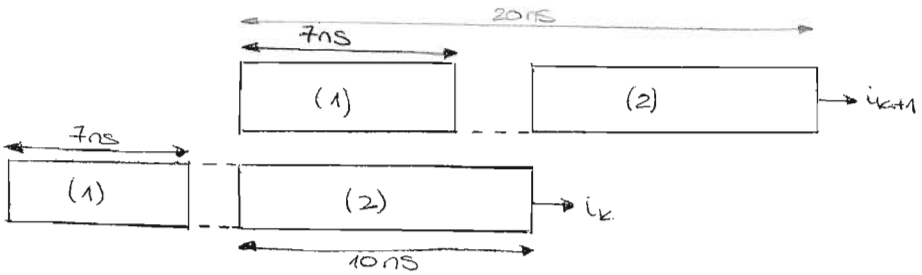
Ejercicio 7. Preguntas cortas (TEMA 3)

es ejemplo genérico, no el del Cold Fire q tiene 2 pipelines

a) Una CPU presenta un pipeline con dos etapas: (1) obtención de la instrucción y de los operandos, y (2) ejecución y almacenamiento del resultado. Si la primera etapa siempre se completa en un tiempo de 7 nanosegundos, justifique cuál es el máximo tiempo que puede tardar la segunda etapa para una velocidad de procesamiento (throughput) de 100 MIPS.

b) Sobre los módulos de diagnóstico y depuración: ¿Cuál es la principal diferencia entre la depuración en segundo plano (BDM) y la depuración en tiempo real?

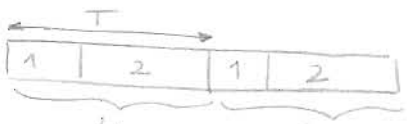
@ En un pipeline el throughput lo limita la tarea + lenta !



$V_1 = \frac{1 \text{ instr.}}{7 \text{ ns}} = \frac{1}{7} \cdot 10^9 \frac{\text{instr.}}{\text{seg}} = 142'87 \text{ MIPS}$

$T_2 = \frac{1}{V_2} = \frac{1}{10^8 \frac{\text{instr.}}{\text{seg.}}} = 10 \frac{\text{ns}}{\text{instr.}}$

Si ejecutamos las etapas seguidas sin pipeline



$V = \frac{1}{T} = \frac{1}{17 \text{ ns}} = 58'8 \text{ MIPS}$

$V_{pipeline} > V_{seguidas}$ pq al haber pipeline se paralelizan las tareas y viene determinado el throughput x la tarea + lenta

(b) BDM =
- El procesador se detiene (no permite evaluar prestaciones)
- Proporciona depuración a bajo nivel a través de una interfaz serie
- Se puede leer y escribir en memoria y en registros y en bloques de memoria completa cuando el procesador está detenido

Tiempo real =
- No se detiene el procesador.
- Basado en interrupciones de depuración en los que se guarda el contenido de variables y registros clave

Ejercicio 8. Sistema de medida de señales analógicas. (TEMA 3)

El sistema del presente ejercicio pretende proporcionar la capacidad de establecer medidas de dos señales analógicas (V0 y V1) mediante un conversor Analógico – Digital (A/D).

La medida de las dos señales se realiza mediante un único conversor, utilizando para seleccionar la señal a convertir un multiplexor analógico (MUX), controlado por una señal de selección digital registrada en el puerto PortV (véase la Figura 1).

Como se puede observar, el conversor A/D está realizado mediante un conversor D/A (Digital – Analógico) y un circuito comparador para realizar la conversión A/D mediante un pequeño algoritmo desarrollado en la rutina "Medida", por el procedimiento de "aproximaciones sucesivas".

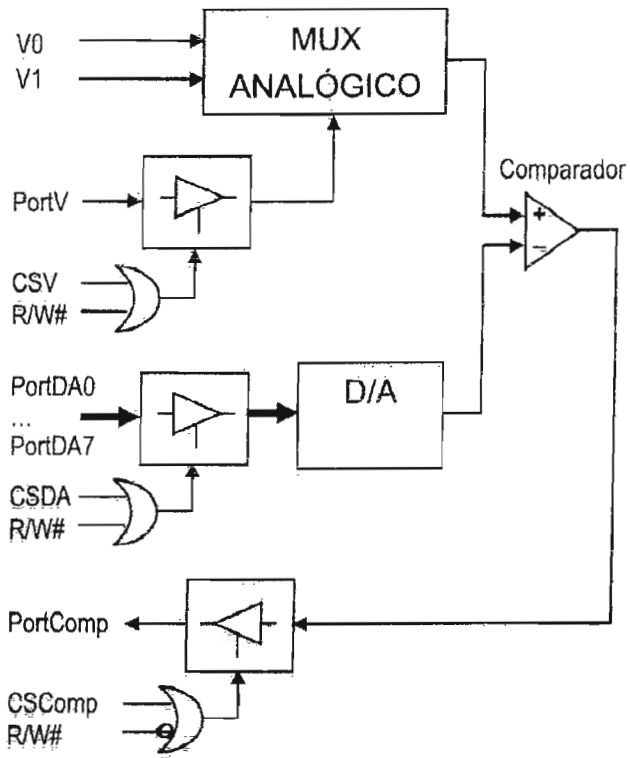


Figura 1. Esquema hardware del sistema de medida

Listado parcial del programa

```
*******************************************************************************
* Programa de medida diferencial de dos señales analógicas
* Realiza una llamada a la rutina MedidaDif, pasando como parámetro la dirección
*  de memoria donde se debe dejar el resultado de la diferencia de las señales
*  analógicas de entrada
*******************************************************************************
                ORG             $00280000       * RAM
Resultado       DS.W            3
                ORG             $00700000       * Zona de E/S
= PortDA        DS.B            1
= PortComp      DS.B            1
= PortV         DS.B            1
                ORG             $00000000       * RoM
                ; ...
                PEA             Resultado       *Punto 1
                JSR             MedidaDif       * Punto 2
                ADDQ.L          #4,A7           * Punto 29
                ; ...
```

```
*************************************************************************
* Subrutina de medida diferencial de dos señales en tensión
* Hace dos llamadas consecutivas a la rutina Medida. En la primera llamada se
*  mide la señal V0 y en la segunda la V1. El parámetro pasado por la pila indica
*  la señal a medir. La rutina Medida es una función que devuelve el resultado a
*  través de la pila. MedidaDif escribe en memoria el valor diferencia.
*************************************************************************
MedidaDif   LINK    A6,#0
            ADDA.L  #-12,A7
            MOVEM.L D0-D1/A0,(A7)
            MOVEA.L 8(A6),A0
            MOVE.W  D0,-(A7)
            MOVE.W  #0,-(A7)        ; Canal V0
            JSR     Medida
            ADDQ.L  #2,A7
            MOVE.W  D0,-(A7)
            MOVE.W  #1,-(A7)        ; Canal V1
            JSR     Medida
            ADDQ.L  #2,A7
            CLR.L   D0
            CLR.L   D1
            MOVE.W  (A7)+,D0
            MOVE.W  (A7)+,D1
            SUB.L   D1,D0
            MOVE.W  D0,(A0)
            MOVEM.L (A7),D0-D1/A0
            ADDA.L  #12,A7
            UNLK    A6
            RTS

*************************************************************************
* Subrutina Medida
* - Selecciona la entrada, una de las dos presentes a la entrada del multiplexor
*   analógico, mediante la aplicación de "0" ó "1".
* - Realiza la conversión A/D utilizando un D/A por el método de "aproximaciones
*   sucesivas" (se realizan varias exploraciones comparando una señal generada
*   por el D/A con la señal a medir).
* - Medida es realmente una función que devuelve su resultado a través de la
*   pila, igual que haría cualquier compilador
*************************************************************************
Medida      LINK    A6,#0
            ADDA.L  #-16,A7
            MOVEM.L D0-D3,(A7)
            MOVE.W  8(A6),D0
            TST.W   D0
            BNE     Mux1
            MOVE.B  #0,D1
            MOVE.B  D1,PortV
            BRA     Sigue
Mux1        MOVE.B  #$FF,D1
            MOVE.B  D1,PortV

Sigue       MOVE.L  #$7F,D0
            MOVE.L  #8,D1
            MOVE.L  #$80,D2

Lazo        MOVE.B  D0,PortDA
            MOVE.B  #2,D3
            BTST.B  D3,PortComp
            BEQ     Bajar
            EOR.L   D2,D0
Bajar       LSR.L   #1,D2
            EOR.L   D2,D0
            SUB.L   #1,D1
            BNE     Lazo

            MOVE.W  D0,10(A6)
            MOVEM.L (A7),D0-D3
            ADDA.L  #16,A7
            UNLK    A6
            RTS
            END
```

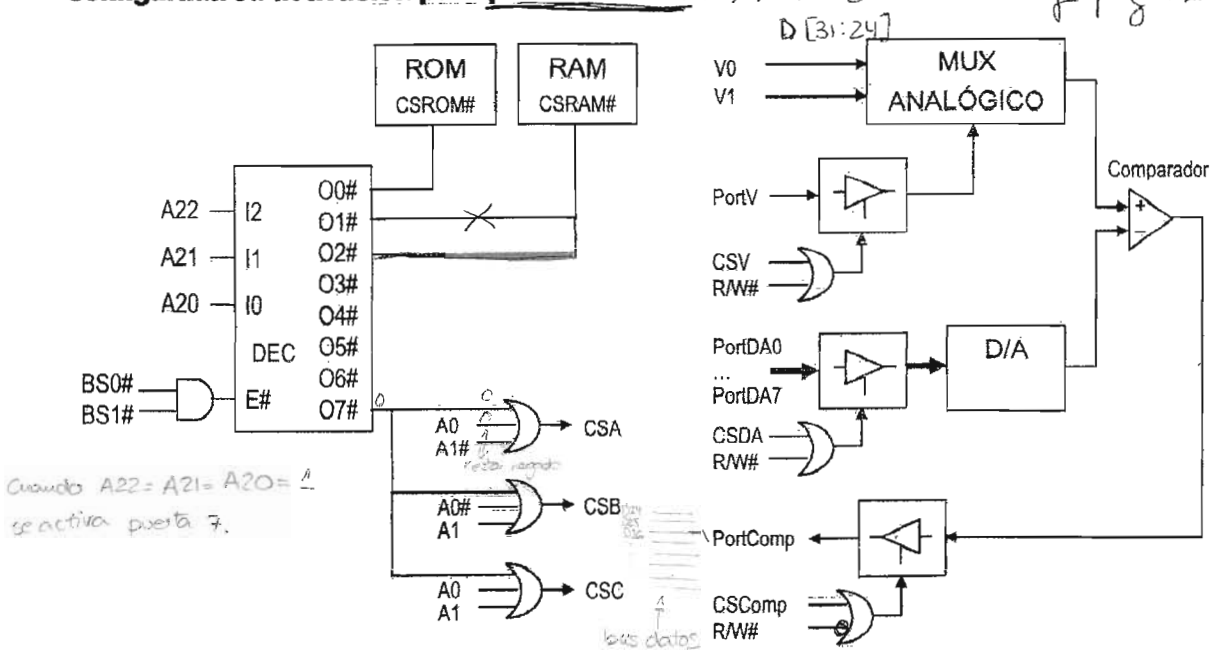
Seleccionan la señal analógica a medir

Selecciona la señal analógica a medir poniendo el valor adecuado en PortV

Algoritmo de aproximaciones sucesivas.

1. Conexionado hardware

1.1. Se supondrá, en primer lugar, que las señales CS# del ColdFire no están disponibles y, por tanto, para la generación de los CS# es necesario recurrir al hardware externo que se indica en la Figura 2. A la vista de dicha figura y del programa software, y teniendo en cuenta que las señales de selección del puerto en cuestión, CSA, CSB y CSC, son activas a nivel bajo, y que las señales PortV, PortDA0..7 y PortComp son señales a conectar al bus de datos, indique las conexiones a establecer. Utilice las tablas siguientes, asumiendo que los dispositivos de entrada/salida tienen configurada su activación para puerto de 8 bits. ⇒ Modo 8 bits (Dibujo pág T2-6)



Cuando A22 = A21 = A20 = 1 se activa puerta 7.

Figura 2. Generación de las señales de selección de puerto

¿cuándo se activa CSA (activa a nivel bajo)? A22 A21 A20 ... A1 A0 ⇒ $700002 ⇒ CSV

¿CSC? A1 A0 ⇒ $700000 ⇒ CSDA

Señal	A conectar con	Breve justificación
CSV	CSA	PortV en $700002 ⇒ $A_1=1, A_0=0$
CSDA	CSC	PortDA en $700000 ⇒ $A_1=0, A_0=0$
CSComp	CSB	PortComp en $700001 ⇒ $A_1=0, A_0=1$

Señal	A conectar con	Breve justificación
PortV	Cualquiera de D[31:24]	Puerto 8 bits ⇒ modo 8 bits ⇒ datos en D[31:24] * Como se escribe $00 ó $FF vale cualquiera de ellos
PortDA0	D[24]	Puerto 8 bits ⇒ Modo 8 bits ⇒ Datos en D[31:24] Bit menos significativo ⇒ D[24]
PortDA1	D[25]	
...		...
PortDA7	D[31]	Bit más significativo ⇒ D[31]
PortComp	D[26]	Se hace BTST sobre el bit 2 de PortComp • Comprueba el 2º bit de PortComp

* es 1 sólo bit o 1

corresponde al bit 7, 26 del bus datos (D[26])

Los puertos se conectan al bus de datos, tal y como dice el enunciado.

Al ser puerto de 8 bits ⇒ Nodo de 8 bits: D[31:24]

1.2. Sabiendo que el modo de bus es de 16 bits salvo para los terminales de entrada/salida que es de 8 bits, que la RAM externa es de 1 MB y la ROM externa de 64 KB, y que las memorias no necesitan estados de espera, configure los registros CSBRn y CSORn del ColdFire para que se generen por los terminales CS0, CS1 y CS2 del ColdFire las señales CSROM, CSRAM e ~~Y7~~ de la Figura 2 respectivamente. Incluya una justificación de cada uno de los campos de los registros.

salida del decodificador

- **ROM: CSBR0 = $ 00000201**

EXTERNA BA = $00000. Dirección base de la ROM $00000000
EBI = %00 (SRAM o ROM de 16/32 bits)
BW = %10 (Bus de 16 bits)
SUPER = %0 (Actúa siempre)
E = %1 (Habilitado)
64KB **CSOR0 = $ FFFF0001**
BAM = $FFFF0 (16 bits). Los otros 16 $\Rightarrow 2^{16} = 64kB$
WS = %0 0000 (Sin estados de espera)
RW = %0 (Solo lectura → ROM)
MRW = %1 (Según indique RW)

Hay q. deducir dirección inicial de RAM:
- vemos q RAM: ORG %00280000
- mirando el hardware: la RAM se activa cuando A22 A21 A20
10 0
hardware fig. 2)
0000 0000 0010 0000.
22 21 20
%00200000

- **RAM: CSBR1 = $ 00200201**

EXTERNA BA = $00200. Dirección base de la RAM $00200000 (ver hardware fig. 2)
EBI = %00 (SRAM o ROM de 16/32 bits)
BW = %10 (Bus de 16 bits)
SUPER = %0 (Actúa siempre)
E = %1 (Habilitada)
CSOR1 = $ FFF00000
BAM = $FFF00 (12 bits). Los otros 20 $\Rightarrow 2^{20} = 1$ MB ! → cambian 20 bits menos significativos
WS = %0 0000 (Sin estados de espera)
RW = %0 (indiferente)
MRW = %0 (lectura escritura)

dir inicial 00 2|0 0000 00
dir final 00 2F FFFF
RAM: FFF 0 0000

- **E/S: CSBR2 = $ 00700D01**

BA = $00700. Dirección base de entrada salida $00700000
RAM (5 bytes) = FFF00
EBI = %11 (SRAM o ROM de 8 bits)
BW = %01 (ancho de bus de 8 bits)
SUPER = %0 (Actúa siempre)
ENABLE = %1 (Habilitado)
CSOR2 = $ FFFFF000
BAM = $FFFFF. Tamaño mínimo de la zona: 3 bytes $\Rightarrow 2^2$
WS = %0 0000 (sin estados de espera)
RW = 0 (Indiferente)
MRW = 0 (lectura/escritura)

EA
Dir. inicial $00700000
Dir. final $00700002
RAM: $FFFFFFFC
3 bytes
byte final:
0000
0010
1100
RAM: C

1.3. ¿Qué rango de direcciones se está asignando realmente para los dispositivos externos de entrada/salida? Justifique su respuesta.

BAM = $FFFFF (20 bits). Los otros 12 $\Rightarrow 2^{12} = 4k$
→ el CS2# se activará en las direcciones de la forma
$00700XXX
12 bits = 2^{12} = 4K Bytes
El rango de direcciones será:
Direc. inicial: $00700000
Direc. final: $00700FFF (12 bits) ↕ 4KB

1.4. Suponga que además se desean utilizar las posiciones de memoria de la RAM local del ColdFire y que, para que no haya solapamientos, se va a ubicar la misma a partir de la dirección $00600000. Así mismo, la dirección base del módulo SIM será la $00500000 y no se van a aplicar máscaras de acceso. Configure los registros MBAR y RAMBAR del ColdFire justificando brevemente el contenido de cada uno de sus campos.

SIM ← MBAR = $00500001
BA = $0050. Dirección base del SIM. $00500000
SC = SD = UC = UD = 0 (Sin máscaras de acceso)
V = 1 (Habilitado)

RAM ← RAMBAR = $00600021
BA = $00600. Dirección base de la RAM: $00600000
WP = %0 (Sin protección frente a escritura)
C/I = 1
SC = SD = UC = UD = 0 (Sin máscaras de acceso)
V = 1 (Habilitado)

2. Análisis del software

→ ver dónde empieza y acaba la RAM

¡IMPORTANTE!

2.1. Si se quisiera proporcionar a la pila el tamaño máximo posible sin interferir con otras variables almacenadas en la RAM, ¿a qué valor habría que inicializarlo?

La RAM va de $00200000 a $00 2FFFFF. Hay 3 variable tipo word a partir de $00280000. Se puede comprobar que la zona mayor en entre $00200000 y $00280000. Como la pila crece hacia direcciones más bajas, habría que inicializarla en la dirección más alta (la primera ocupa), es decir, $00280000

2.2. Escriba dos formas de realizar la inicialización de la pila en $250000 en tiempo real.

¡Ojo!

1) MOVEA.L #$250000, A7
2) LEA $250000, A7

2.3. Indique para qué sirve la sentencia END. ¿Es código a ejecutar por el microprocesador?

Indica al proceso de ensamblado donde termina el código a tener en cuenta. Lo que haya a partir del END se ignora. No es código a ejecutar por el microprocesador. Es 1 directiva → no se ejecuta

2.4. En la instrucción MOVE.W D0,-(A7) que se encuentra en la rutina MedidaDif, ¿tiene sentido el valor de D0 que se almacena? Justifique su respuesta.

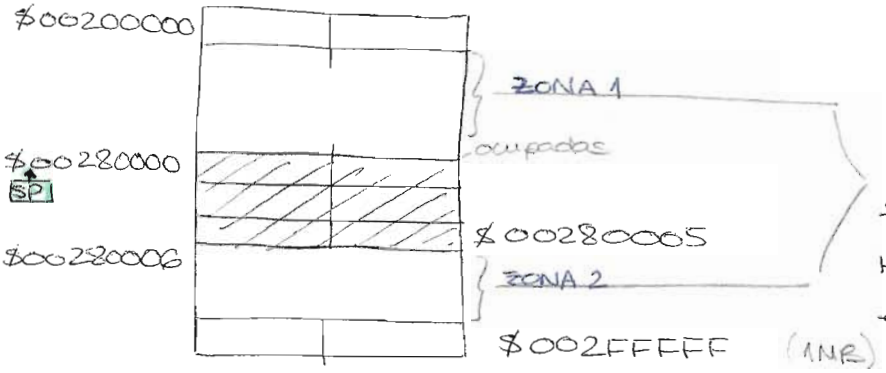
No tiene sentido. Simplemente sirve para reservar espacio en la pila (palabra .W) para el resultado de la rutina medida

Luego el valor D0.W se sobreescribe con el valor de la medida

2.5. ¿Cuántas posiciones de la memoria de la pila se van a ocupar en el caso peor (máxima ocupación)? Justifique su respuesta.

54 bytes (Ver dibujo de la pila)!

2.1



estas 2 zonas son los espacios q. quedan libres en la pila. Hay q. ver cuál de las 2 zonas es mayor.

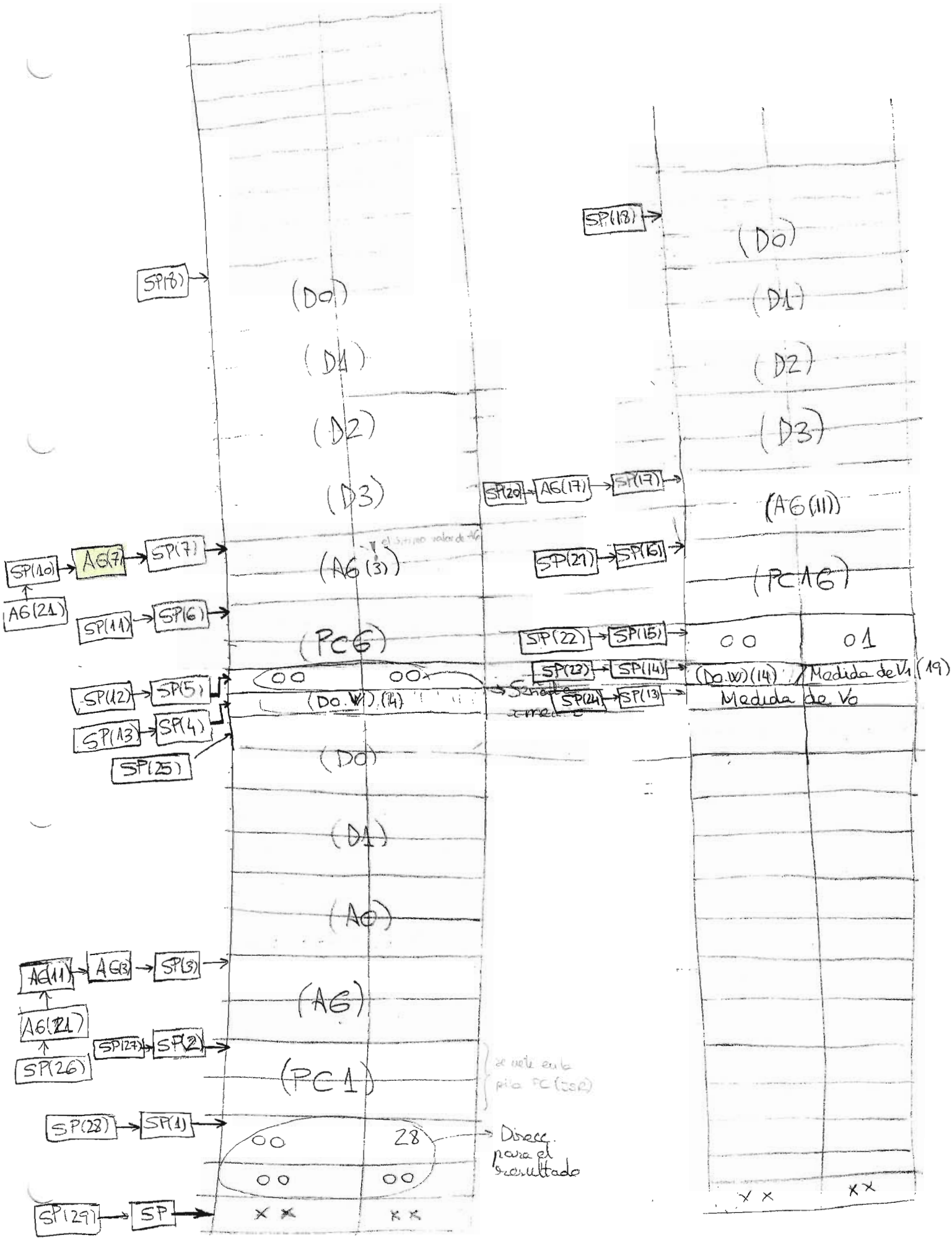
Tamaño zona = (Dir. final + 1) − Dir. inicial

- Tamaño zona1 = $00280000 − $00200000 = $00080000 = 8.16^4 = 524288 bytes
- Tamaño zona2 = $00300000 − $00280006 =

$= 3\,16^5 - (2.16^5 + 8.16^4 + 6)$ = 524282 bytes

⇒ Es mayor la ZONA 1 luego el valor inicial del puntero de pila es la 1ª posición ocupada = $00280000

PILA


SP(18)
(D0)
(D1)
(D2)
(D3)
(A6(3))
(PC6)
(A0)
(A6)
(PC1)
SP(28)
28
Direcc. para el resultado
SP(29)
SP
SP(10)
A6(7)
SP(7)
A6(21)
SP(11)
SP(6)
SP(12)
SP(5)
SP(13)
SP(4)
SP(25)
A6(11)
A6(3)
SP(3)
A6(21)
SP(26)
SP(24)
SP(2)
SP(18)
(D0)
(D1)
(D2)
(D3)
SP(20)
A6(17)
SP(17)
(A6(11))
SP(21)
SP(16)
(PC16)
SP(22)
SP(15)
00
01
SP(23)
SP(14)
(D0.W)(14)
Medida de V1 (19)
SP(24)
SP(13)
Medida de V0
XX
XX

2.6. Tras la etiqueta Sigue se hace MOVE.L #$7F,D0 en vez de MOVE.B #$7F,D0. ¿Por qué?

Porque en D0 se va a almacenar el valor de la medida de la señal analógica y luego se va a llevar a la pila con MOVE.W D0, 10(A6) De esta forma los bits 8-15 se quedan a 0

2.7. En la instrucción MOVE.W D0,(A0) al final de la rutina MedidaDif,

¿Qué contiene el registro D0? (el concepto, no el valor)

+ El valor de la medida diferencial $(V_1 - V_0)$

¿Es un número en binario natural o en complemento a 2? Justifique su respuesta.

Es un número en complemento a 2 pues $V_1 - V_0$ puede ser negativo

¿Dónde se almacena el contenido del registro D0?

En la variable resultado cuya dirección es $280000 y que está apuntada por A0

2.8. Describa la función de D0, D1 y D2 en la subrutina Medida.

D0 contiene el valor digital de la aproximación a la tensión analógica

D1 contiene el contador del bucle (8 iteraciones)

D2 contiene la máscara que se va aplicando con XOR empezando por el bit más significativo hasta llegar al menos significativo

TEMA 4: EXCEPCIONES EN EL SIST. MICROPROC.

4.1 Excepciones
4.2 Interrupciones
4.3 Protección del sistema y gestión del consumo

4.1 Excepciones

4.1.1 Definición

Internas
Externas

- Excepción: suceso externo o interno que da lugar a una alteración de la ejecución normal de un programa, pasando a ejecutarse una rutina de atención específica ¡¡en modo supervisor!!
- Aplicaciones: manejo de errores, realización de tareas especiales, comunicación eficiente con periféricos.

A cada una se le asigna un número de vector de excepción (apartado 4.1.3)

4.1.2 Tipos de excepciones

- **Internas al microcontrolador:**
  - Errores detectados internamente:
    - ♦ Error de bus: error en el proceso de lectura o escritura de una zona de memoria
    - ♦ Error de dirección: intento de ejecución de una instrucción en una dirección impar
    - ♦ Violación de privilegio: intento de ejecución en modo usuario, de una instrucción privilegiada.
    - ♦ Instrucción ilegal: código de operación inexistente.
    - ♦ División por cero.

Los errores detectados internamente y el modo traza forman parte de las llamadas "excepciones del procesador"

  - Modo traza: cuando el bit T del SR está activado, se produce una excepción tras cada instrucción.

  -Instrucciones generadoras de excepción:
    - ♦ TRAP: generación de la excepción 32 + n (Ver tabla de vectores de excepción)

	Tamaño f (bits)	Modo f	X N Z V C
TRAP f	4	#n	-----

En ARQO se utiliza una instrucción equivalente llamada BREAK que sirve para llamar al núcleo del sistema operativo

    - ♦ Instrucciones no definidas con opcode $A... ó $F...: generan las excepciones de código de línea A ó F. Se usan insertando en la zona de programa `DC.W $AXXX` ó `DC.W $FXXX`. Con ellas puedes implementar nuevas instrucciones haciendo que la rutina que atiende dicha excepción haga lo que tu quieras. Son especialmente útiles para emular hardware.

En el Coldfire no existen instrucciones cuya codificación empiece por $A ó por $F

- **Externas al microcontrolador:**

Las excepciones de activación de terminales especiales también están incluidas en las llamadas "excepciones del procesador"

  - Activación de terminales especiales. Por ejemplo el de Reset.

Excepción que ocurre al arrancar o reiniciar el sistema

  - Interrupciones por parte de los periféricos.

lista ordenada de direcciones de rutinas de atención a excepciones

VECTOR DE EXCEPCIÓN: Asocia el nº de excepción con la dirección de la rutina de atención

4.1.3 Tabla de vectores de excepción

- El vector de una excepción generalmente es una palabra de 4 bytes que contiene la dirección de la rutina de atención a dicha excepción.
- Los vectores de excepción se identifican por un número.
- Se definen todos juntos en la tabla de vectores de excepción que ocupará 256 long.
- La posición de la tabla en memoria es configurable mediante el registro VBR.

En la tabla, las excepciones se agrupan por tipos:
- Excepciones de procesador
- Excepciones de la instrucción TRAP
- Interrupciones (varios tipos)

Nº	Offset $	Tipo de Excepción
0	000	RESET: Valor inicial de A7
-	004	Valor inicial del PC
2	008	Error de bus
3	00C	Error de dirección
4	010	Instrucción ilega
5	014	División por cero
6–7	018–01C	Reservados
8	020	Violación de privilegio
9	024	Modo traza
10	028	Código de operación A (línea A)
11	02C	Código de operación F (línea F)
12	030	Interrupción de depuración
13	034	Reservado
14	038	Error de formato
15	03C	Vector no inicializado
16–23	040–05C	Reservado
24	060	Interrupción espuria
25–31	064–07C	Interrupciones autovectorizadas Niveles 1–7
32–47	080–0BC	Instrucciones TRAP #0–15
48–63	0C0–0FC	Reservado
64–255	100–3FC	Interrupciones de usuario

256 vectores 4B → 256 long. 64 tipos definidos

Las interrupciones de usuario se ven en el apartado 4.2 →

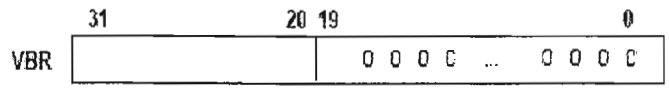
- ¡¡¡El vector de excepción "reset" (nº 0) es especial porque ocupa 8 bytes ya que incluye dos direcciones: valor inicial de A7 y dirección de la rutina de reset!!! (PC)

Configuración de la tabla de vectores de excepción

VBR = registro de la dirección base de vectores

Se realiza mediante el registro de dirección base de vectores, VBR

- Se accede con la instrucción MOVEC a la dirección de CPU $801
- Determina la dirección base en memoria de la tabla de vectores



- Tras el reset, VBR=$00000000
- La dirección de memoria en la que está el vector de excepción nº X es:

Dirección del vector de la excepción nº X = VBR + (4 x X)

- La dirección de memoria de la rutina de atención a la excepción será el contenido de dicho vector.
- ¡OJO! Aunque VBR sea distinto de $00000000, en esa dirección de memoria siempre se almacenará el vector 0 correspondiente a la excepción "reset"(8 bytes), y habrá otra copia a partir de la dirección definida por el VBR.
- Ejemplo: ¿cuál es la dirección de memoria de la rutina de atención a la excepción nº 82, dado el siguiente contenido del registro VBR y de la memoria en las posiciones indicadas?

¡OJO! Es muy importante diferenciar entre el número de vector, la dirección del vector en memoria, y el contenido del vector, que es la dirección de la rutina de atención

MEMORIA

	Pares	Impares
00F00140	C0	00
	E3	A8
00F00144	C1	62
	67	C5
00F00148	C1	70
	01	40

VBR: 00F00000

① Direc. vector 82 = $00F00000 + 4,82 = $00F00148

② Direc de la rutina de atención a la excepción 82 = $C1700140

Esto no está en la "chuleta", pero no hace falta aprendérselo de memoria. Si te fijas tiene su lógica.

4.1.4 Prioridad entre excepciones

Prioridad	Excepción	Procesamiento
0.0	Reset (E)	Máxima urgencia, no salva el contexto
1.0 1.1	Error de dirección (I) Error de bus (E)	Se suspende el proceso y se guarda el contexto
2.0 2.1 2.2	Código de líneas A y F (I) Instrucción ilegal (I) Violación de privilegio (I)	La excepción se procesa antes de ejecutarse la instrucción que la provoca
3.0	TRAP (I)	La excepción forma parte de la ejecución de la instrucción
4.0 4.1 4.2	Modo traza (I) Interrupción de depuración Interrupciones externas (E) (*)	La excepción se procesa cuando termina la ejecución de la instrucción en curso

Notas: Las interrupc. son de prioridad menor!

(E) = origen externo
(I) = origen interno
(*) = se pueden producir varias anidadas

IMP.

4.1.5 El procesamiento de excepciones

Debemos notar que tiene ciertas semejanzas con lo que ocurre cuando se llama a una subrutina con BSR o JSR, pero también hay diferencias muy importantes (Ver ejemplo 4.1)

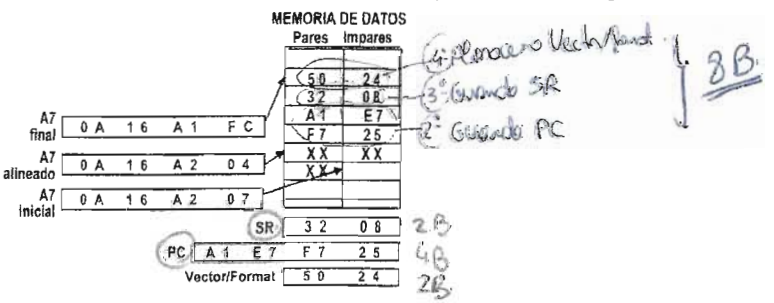
1. **Almacenamiento del contexto en la pila**
   - La pila se alinea automáticamente en límite de long (dirección múltiplo de 4), decrementando el valor de A7 en el número de bytes necesario
   - Se guarda el PC → contador de programa (dir de la instrucción en ejecución)
   - Se guarda el SR
   - Se pasa a modo supervisor (bit S=1) y se inhabilita modo traza si estaba habilitado (bit T=0)
   - Se almacena la palabra Vector/Format, que contiene el número del vector, e información acerca del alineamiento de la pila y depuración

dir X vector = VBR + 4·X

2. **Cálculo de la dirección de la rutina de atención a la excepción.** Lo hemos visto en el apartado anterior. Se lleva dicha dirección al PC lo que provoca el salto a dicha dirección.

3. **Ejecución de la rutina de atención a la excepción**
   - Finaliza con la instrucción RTE, que recupera la información de estado (SR, PC) de la pila y restaura A7 al valor anterior a la excepción (a partir de la palabra Vector/Format).

- Ejemplo de los que ocurre con la pila cuando se procesa una excepción:



Este apartado no es esencial, pero podría caer en forma de preguntas cortas como "teoría"

4.1.6 Algunas excepciones de interés

Excepción de error de bus — Errores producidos en el bus en un ciclo de lectura o escritura

- Sirve para manejar errores producidos en el bus en un ciclo de lectura o escritura:
  - Acceso a memoria o periféricos inexistentes.
  - Intento de escritura en ROM
  - Acceso en modo usuario a zonas de memoria no permitidas
  - Error de paridad en datos de la memoria, por lo que la información almacenada puede estar corrupta

(4) Subrutinas frente a Excepciones

SUBRUTINAS	EXCEPCIONES
La dirección de salto se especifica en una instrucción	La dirección de salto se especifica a través de la tabla de vectores
Guarda en la pila PC (4B)	Guarda en la pila PC, SR, vect/format (4B) (2B) (2B) (8B)
No pasa a mod. supervisor	Siempre pasa a modo supervisor (S=1)
No afecta al modo traza	Se inhabilita modo traza (T=0)
Instrucción final (RTS)	Instrucción final (RTE)
Control por programa	Ajenas al programa

- Proceso: el monitor de bus interno o un dispositivo externo activa TEA# y se genera la excepción de "error de bus"
- En algunos casos se puede producir un doble error de bus. Es un error de bus al procesar la excepción de "error de bus". Es una situación catastrófica de la que solo se sale con un reset.

Arranque y reinicio del sistema

- El arranque permite configurar y poner en funcionamiento el sistema. El reinicio permite restaurar el sistema ante fallos catastróficos.
- Las dos situaciones provocan el procesamiento de la excepción "reset" (vector 0)
  - Se inicia con la activación del terminal RSTI#
  - Se activa la señal RSTO# para reinicio software de periféricos
  - La CPU pone modo supervisor (S=1), inhabilita modo traza (T=0) y sin interrupciones (I2 I1 I0=1 1 1) en el SR.
  - Se lleva a A7 la primeros 4 bytes de la tabla de vectores.
  - Se lleva al contador de programa los siguientes 4 bytes de la tabla de vectores.
  - La mayoría de los registros de control y configuración se ponen a 0.
  - Los registros An y Dn quedan indefinidos.
  - Se inicia la ejecución del programa en la posición que indica el PC.

Una vez ajustados A7 y PC se deja de estar en la rutina de reset

Parada del sistema

- Instrucciones de parada del sistema (son privilegiadas)

	Tamaño f	Modo f	X N Z V C
HALT STOP f	- W	- #n	----- *****

- HALT: suspende la ejecución inmediatamente. Se sale de este modo desde el módulo de depuración o con reset

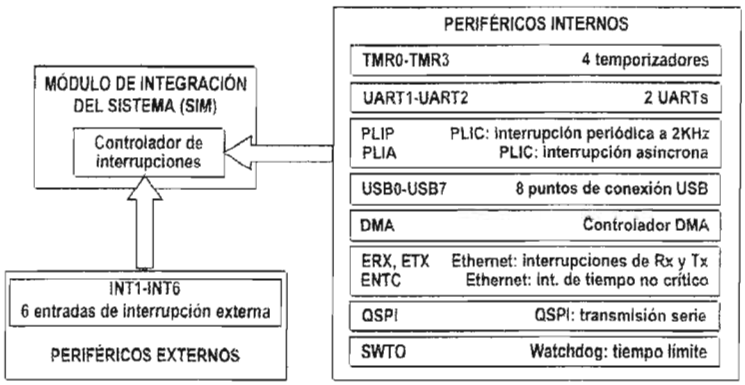
- STOP: detiene la ejecución
  - Carga n en el registro de estado
  - Carga la dirección de la siguiente instrucción en el PC
  - Entra en modo de detención
  - Se sale de este modo cuando se produce una interrupción del nivel adecuado (depende del valor de n que se ha cargado en el SR y por tanto se ha asignado un valor a I_2 I_1 I_0)

- Otras condiciones de parada son la instrucción de depuración (Apartado 3.1.6) y el doble error de bus que bloquea el procesador.

4.2 Interrupciones

4.2.1 Definición y fuentes de interrupción en el MCF5272

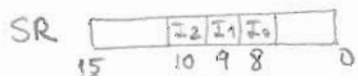
Interrupción: solicitud de servicio por parte de un dispositivo hardware interno o externo que requiere la atención de la CPU. El servicio se realiza por medio de una rutina de servicio (software). El vector de la interrupción correspondiente contendrá la dirección de comienzo de la rutina de atención.



4.2.2 Nivel de prioridad de las interrupciones, enmascaramiento y anidamiento

Hay que fijarse también en la tabla de 4.1.4, porque los demás tipos de excepciones que ahí se indican tienen mayor prioridad que las interrupciones, que aparecen al final de la tabla

- Nivel de prioridad: determina la prioridad entre interrupciones simultáneas
  - Hay 7 niveles de prioridad
  - La máscara $I_2\ I_1\ I_0$ del SR define los niveles de interrupción habilitados en un momento dado



$I_2\ I_1\ I_0 = 000$ → se aceptan todos los niveles de interrupción

$I_2\ I_1\ I_0 = 111$ → se acepta sólo N7 (no enmascarable)

AND F8FF

OR 0700

Se dice que las interrupciones de nivel 7 son *no enmascarables*

- Enmascaramiento: ante una interrupción de nivel N
  - Si N≤ $I_2\ I_1\ I_0$ ⇒ interrupción no aceptada (queda pendiente)
  - Si N> $I_2\ I_1\ I_0$ ⇒ interrupción aceptada (excepto si $I_2\ I_1\ I_0 = 111$, porque las interrupciones de nivel 7 se aceptan siempre)

MÁSCARA = Nivel mínimo de prioridad previsto del cual atiende interrupciones

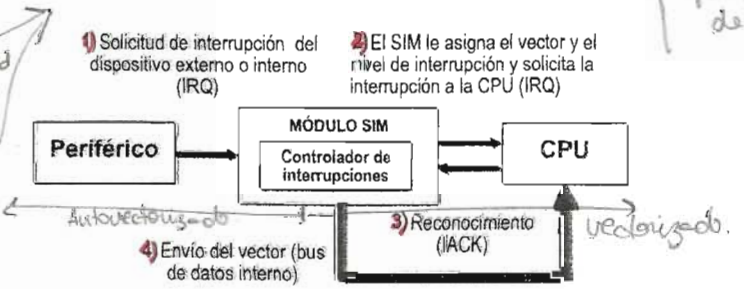
Si llega una interrupción con el mismo nivel No se atiende, tiene que estar prevista

- Anidamiento de interrupciones: cuando se atiende una interrupción, se pone su nivel de prioridad en los bits $I_2\ I_1\ I_0$ del SR, de manera que solo se atiendan las interrupciones de un nivel superior. Las de nivel inferior quedan pendientes.

4.2.3 Ciclo de reconocimiento de interrupción en el MCF5272

Señalización requerida para el reconocimiento de una interrupción

Mayor velocidad al dividir el proceso en dos partes



4.2.4 Procesamiento de las interrupciones en el MCF5272

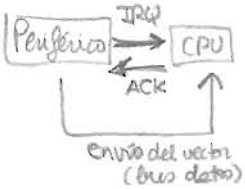
Es prácticamente igual al que se utiliza para una excepción genérica pero con algunos "detalles" importantes

1. Almacenamiento del contexto en la pila
   - La pila se alinea automáticamente en límite de long (dirección múltiplo de 4), decrementando el valor de A7 en el número de bytes necesario
   - Se guarda el PC
   - Se guarda el SR

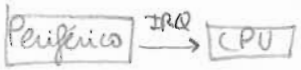
SIM = Módulo de integración del sistema

(*) Tipos de interrupciones

- **VECTORIZADAS**
  - El nº de vector de interrupción se obtiene a partir del dispositivo que interrumpe
  - Requieren un ciclo de reconocimiento de interrupción (ACK)



- **AUTOVECTORIZADAS**
  - El nº de vector de interrupción se obtiene a partir del nivel de prioridad de la interrupción
  - No hay ciclo de reconocimiento de interrupción



SONDEO (polling)	INTERRUPCIONES
El programa principal accede iterativamente al periférico (bucle de sondeo)	El programa se ejecuta normalmente mientras no se produce el suceso
La CPU no hace trabajo útil hasta que se produce el suceso.	Al producirse, el periférico solicita atención (interrupción)

- Se pasa a modo supervisor (bit S=1) y se inhabilita modo traza si estaba habilitado (bit T=0) y sin modo maestro (bit M=0)
- ¡¡¡Se copia el nivel de interrupción N en la máscara (I_2 I_1 I_0 = N)!!!

Esto es lo que permite el anidamiento de interrupciones

- Se almacena la palabra Vector/Format, que contiene el número del vector, e información acerca del alineamiento de la pila y depuración

2. Cálculo de la dirección de la rutina de atención a la excepción. Ahora el vector de interrupción viene asignado por el controlador de interrupciones.

3. Ejecución de la rutina de atención a la excepción
   - Finaliza con la instrucción RTE, que recupera la información de estado (SR, PC) de la pila (restaura por tanto también el valor anterior de I_2 I_1 I_0) y restaura A7 al valor anterior a la interrupción (a partir de la palabra Vector/Format).

4.2.5 El controlador de interrupciones del MCF5272

- Maneja y prioriza las interrupciones
- Emite el número de vector de cada interrupción
- Tiene 6 entradas de interrupción externa sensibles a flanco, INT1-INT6
- Maneja un total de 29 fuentes de interrupción (6 externas + las de perif. internos)

Siempre hay que configurarlo

Registro de vector de interrupción programable

¡OJO! El PIVR determina el número a partir del cual se van a numerar los vectores de interrupción. ¡¡¡No define una dirección de memoria!!!

	7	6	5		0
PIVR	IV			-	

- Determina el número de vector de interrupción para todas las interrupciones
- Campo IV: bits 7-5 que determinan los 3 bits más significativos del número de vector
- Los bits 4-0 se asignan automáticamente según las distintas fuentes que siempre mantienen el mismo orden
- El valor típico para este registro es 64 = $40. En ese caso los números asignados a los vectores de interrupción serían

Esta tabla se puede ver como la continuación de la que aparece en 4.1.3 donde pone "interrupciones de usuario"

Nº	Interrupción	Fuente de Interrupción
64	Int. espuria	Interrupción espuria desde fuente
65-68	INT1-INT4	Entradas de interrupción externa 1 a 4
69-72	TMR0-TMR3	4 temporizadores
73-74	UART1-UART2	2 UARTs
75-76	PLIP-PLIA	PLIC
77-84	USB0-USB7	8 puntos de conexión USB
85	DMA	Controlador DMA
86-89	Erx-Etx-ENTC	Ethernet
89	QSPI	QSPI
90-91	INT5-INT6	Entradas de interrupción externa 5 y 6
92	SWTO	Watchdog

64 = $40 es el primer vector de usuario

Los vectores se asignan a las 29 fuentes de interrupción según su offset.

Registros de control de las interrupciones ICR1-ICR4

- Definen los niveles de prioridad de las interrupciones y si están pendientes

	31	30	28	27	26	24	23	22	20	19	18	16	15	14	12	11	10	8	7	6	4	3	2	0
ICR1	INT1 PI		INT1 IPL		INT2 PI		INT2 IPL		INT3 PI		INT3 IPL		INT4 PI		INT4 IPL		TMR0 PI		TMR0 IPL		TMR1 PI		TMR1 IPL	
	TMR2 PI		TMR2 IPL		TMR3 PI		TMR3 IPL																	
ICR2	UART1 PI		UART1 IPL		UART2 PI		UART2 IPL		PLIP PI		PLIP IPL		PLIA PI		PLIA IPL		USB0 PI		USB0 IPL		USB1 PI		USB1 IPL	
	USB2 PI		USB2 IPL		USB3 PI		USB3 IPL																	
ICR3	USB4 PI		USB4 IPL		USB5 PI		USB5 IPL		USB6 PI		USB6 IPL		USB7 PI		USB7 IPL		DMA PI		DMA IPL		ERX PI		ERX IPL	
	ETX PI		ETX IPL		ENTC PI		ENTC IPL																	
ICR4	QSPI PI		QSPI IPL		INT5 PI		INT5 IPL		INT6 PI		INT6 IPL		SWTO PI		SWTO IPL		-							

n-vector = PIVR + offset.
dir-vector = VBR + n-vector · 4 ; dir-rutina = (dir-vector)

¡OJO! Para las interrupciones externas INT1-INT6 para ponerlo a 0 hay que escribir un 1

- xPI: bit de interrupción pendiente (xPI=1) o no pendiente (xPI=0). Cuando el periférico x solicite interrupción, el controlador pondrá este bit automáticamente a 1. Habrá que ponerlo a 0 en la rutina de servicio.

- xIPL: nivel de prioridad de la interrupción ⇒ 3 bits (000 inhabilita). Si se cambia, después hay que escribir un 1 en el campo xPI
- ¿Qué ocurre si dos periféricos de la misma prioridad interrumpen simultáneamente? Se ordenan según el orden de las fuentes de interrupción (orden que tienen en la tabla de la página anterior)

Registro de flanco de interrupción programable PITR

	31	30	29	28		6	5	0
PITR	INT1	INT2	INT3	INT4	-	INT5	INT6	-

- Refleja el flanco al que son sensibles las interrupciones externas
- INTn: sensible al flanco de subida (INT=1) o de bajada (INT=0)

Registro de fuente de interrupción : ISR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
INT1	INT2	INT3	INT4	TMR0	TMR1	TMR2	TMR3	UART1	UART2	PLIP	PLIA	USB0	USB1	USB2	USB3
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
USB4	USB5	USB6	USB7	DMA	ERX	ETX	ENTC	QSPI	INT5	INT6	SWTO	-			

- Valor instantáneo de las líneas de todas las fuentes de interrupción
- El terminal correspondiente podrá estar a nivel alto (bit a 0) o a nivel bajo (bit a 1)

¡OJO! Está al contrario de lo que parece lógico

4.3 Protección del sistema y gestión del consumo

4.3.1 Protección del sistema. Watchdog

- El Watchdog detecta que el sistema no está funcionando bien. Permite detectar errores en los programas (p.e. bucles infinitos) o los dispositivos (p.e. periféricos que no responden). Puede ser hardware o software.

- Watchdog hardware: dispositivo que supervisa si el ciclo de bus termina correctamente.
  - Siempre está habilitado
  - Se genera una excepción si:
    - ♦ No termina el ciclo de bus en un número programado de ciclos de reloj
    - ♦ Dicho número de ciclos se configura en un registro del SIM
    - ♦ Tras dicho número de ciclos se produce la excepción "error de bus"

- Watchdog software: el programa debe escribir periódicamente en un registro. Si no escribe en ese registro en el tiempo máximo permitido, se produce o bien una interrupción o bien una excepción "reset"

4.3.2 Módulo Watchdog software del MCF5272

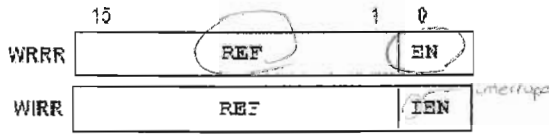
- ¿Cómo se programa el watchdog software?
  - Se selecciona la generación de interrupción o reset tras el tiempo máximo permitido
  - Se configura el valor de fin de temporización
  - Se habilita, ya que tras el reset está inhabilitado
  - Todo eso se configura a través de los registros WRRR, WIRR, WCR y WER

- Hay que tener en cuenta además:
  - ♦ Está basado en un contador de 15 bits (registro WCR)
  - ♦ El reloj del contador es el reloj del sistema dividido por 2^{15}
  - ♦ Periodo entre escrituras configurable hasta 2^{30} ciclos de reloj
  - ♦ Tras su habilitación hay que escribir un valor cualquiera en WCR

Periodo entre escrituras configurable
Valor mínimo = 2^{15}
Valor máximo = 2^{30}

Sólo se utiliza uno de los dos, dependiendo si decides usar un reset o la interrupción SWT0, cuando se produzca el fin de la temporización

Registros de referencia para reset/interrupción de watchdog WRRR/WIRR



Tras un fin de temporización sin escritura en WCR genera un reset del sistema / interrupción SWT0 *(Software Watchdog Time Out)* (Offset 28)

- EN=1 habilita el watchdog software
- REF: configura el periodo entre escrituras = (REF + 1) x 2^{15} ciclos de reloj

Con un REF de 15 bits, el valor máximo de REF será $2^{15} - 1$, y el periodo máximo entre escrituras configurable será de 2^{30} ciclos de reloj

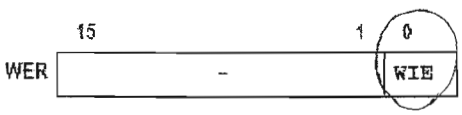
(max xa REF = $2^{15}-1$)

Registro de contador de watchdog WCR

- Valor actual del contador de watchdog
- Se reinicia cada vez que se escribe en él (da igual el valor)

Ej REF+1=7
$7 = 0111
1111
0 0 0 F
WRRR = $000F

Registro de eventos de watchdog WER



WIE = 1 atender interrupción

- WIE: evento de interrupción por watchdog
  - Se pone a 1 al generarse la interrupción (contador=referencia)
  - Hay que ponerlo a 0 al atender la interrupción (escribiendo un 1)

Aquí también se pone a 0 escribiendo un 1

4.3.3 Gestión del consumo

- El objetivo es reducir el consumo al máximo. De esta forma, en los dispositivos que funcionen de manera autónoma, las baterías podrán ser de menor tamaño y se aumentarán las horas de autonomía.
- Mecanismos para reducir el consumo:
  - Apagado selectivo de los módulos del microcontrolador, inhabilitando la señal de reloj que llega a dichos módulos.
  - Modo *standby* en memorias RAM (volátiles). No se puede acceder a memoria, pero permite mantener el contenido de la misma con tensiones de alimentación menores (del orden de 2 V) y consumo mínimo.
- El sistema sale de un modo de bajo consumo al detectarse una interrupción

Ejemplo:

$$Ref = \left(Periodo \times \frac{f_{clk}}{2^{15}}\right) - 1 = \left(40\,ms \cdot \frac{66M}{2^{15}}\right) - 1 = 79{,}57 \approx 80$$

$Ref = 80 = \$50$
$En = 1$ $\longrightarrow$ WRRR = \$00A4

4.3.4 Módulo de gestión del consumo del MCF5272

Permite:

- Modo *sleep* (dormido) de bajo consumo que detiene la CPU (el resto de módulos están activados)
- Modo *stop* (parado) de bajo consumo que detiene el sistema (no sólo la CPU)
- Apagado de módulos individuales internos

Registro de gestión de alimentación PMR

Permite programar la alimentación de módulos individuales

31	26	25	24	23	22	21	20	19	18	17	16	10	9	8	5	4	0
BDM PDN	-	ENET PDN	PLI PDN	DRAM PDN	DMA PDN	PWM PDN	QSPI PDN	TIMER PDN	GPIO PDN	USB PDN	UART1 PDN	UART0 PDN	-	USB WK	UART1 WK	UART0 WK	-	MOS	SLPEN	-

- xPDN=1 apaga módulo x (reloj inhabilitado)

Tras un reset, xPDN=0 y todos los módulos consumen

- xWK=1: despierta módulo al detectar un cambio de señal

En módulos que reciben señales desde el exterior (USB, UART)

- MOS=1: detiene el reloj principal (modo parado)

Se detiene el reloj interno en todo el procesador

- SLPEN=1: detiene el reloj de la CPU (modo dormido)

Se detiene el reloj interno sólo para la CPU

Registro de activación de bajo consumo ALPR

	15	0
ALPR		

- Para activar el bajo consumo se escribe cualquier cosa en ALPR y se ejecuta justo a continuación una instrucción STOP

Registro programable de despertar por interrupción : PIWR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
INT1	INT2	INT3	INT4	TMR0	TMR1	TMR2	TMR3	UART1	UART2	PLIP	PLIA	USB0	USB1	USB2	USB3
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
USB4	USB5	USB6	USB7	DMA	ERX	ETX	ENTC	QSPI	INT5	INT6	SWTO	-			

Este registro en realidad pertenece al controlador de interrupciones, por lo que en la "chuleta" aparece junto con los registros de dicho controlador

- Se ponen a 1 los bits correspondientes a las fuentes de interrupción que se desee que puedan despertar al sistema, tanto si está dormido como si está parado.
- El nivel de prioridad de la interrupción debe ser mayor que $I_2\ I_1\ I_0$ del SR para que pueda despertar al módulo.

¡OJO! Si estamos en modo parado sólo podrán despertarnos las fuentes de interrupción INT1-INT6 (externas) porque el resto corresponden a módulos internos que estarán apagados

Pasos a seguir para configurar el módulo de gestión del consumo

1. Configurar los bits xPDN y xWK del registro PMR
   - Modo parado: activar el bit MOS de PMR
   - Modo dormido: activar el bit SLPEN de PMR
2. Escribir cualquier dato en ALPR — MOVE.W D0, ALPR
3. Ejecutar la instrucción STOP #n justo de después de escribir en ALPR
   - Carga el valor inmediato n en el SR — Ej: STOP #2200 (nivel 2 en máscara → INT 3)
   - La interrupción debe aceptarse para abandonar el modo dormido/parado por lo que su nivel deberá ser mayor que $I_2\ I_1\ I_0$

Si estamos en modo dormido, también podrán despertarnos las interrupciones internas (las de los módulos internos del microcontrolador)

Ejemplo 4.1

Ahora que conoce los pasos que se realizan cuando se produce un salto a subrutina y cuando se atiende una interrupción, explique las principales diferencias entre ambas situaciones.

SUBRUTINAS	EXCEPCIONES
1.- Se las llama desde el programa.	1.- Son ajenas al programa
2.- La dirección de salto se especifica en la instrucción.	2.- La dirección de salto se especifica a través de la tabla de vectores.
3.- Guarda en la pila el PC	3.- Guarda en la pila el PC, el SR y Vector / Format.
4.- No afecta al modo traza.	4.- Se invabilita el modo traza
5.- No provocan el paso a modo supervisor.	5.- Siempre se pasa a modo supervisor.
6.- Instrucción final RTS	6.- Instrucción final RTE

Ejemplo 4.2

Explique las diferencias entre el mecanismo de *sondeo* para la atención de un dispositivo y el mecanismo basado en *interrupciones*. Para ellos básese en el siguiente ejemplo, referido a un teclado.

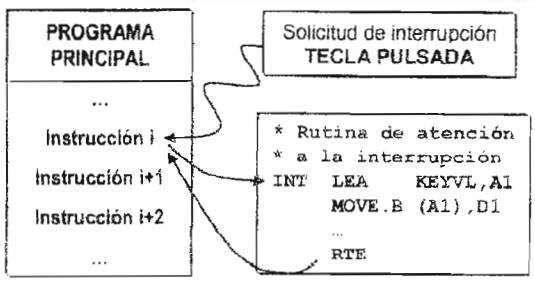
SONDEO

```
...
KEYST EQU $F00001      * Registro de estado
KEYVL EQU KEY_ST+2     * Registro de dato
...

      LEA    KEYST,A0
      LEA    KEYVL,A1
LOOP  BTST   #0,(A0)   * Cuando el bit menos
      BEQ    LOOP      * significativo ≠ 0
      MOVE.B (A1),D0   * lee el dato
...
```

comprueba el bit PR

INTERRUPCIONES

se ejecutan muchas instr. entre 2 teclas



1.- El programa principal accede iterativa/ al periférico (bucle de sondeo)

En este caso comprueba si el bit PR del reg. estado del teclado está a 1, lo q. se produce cuando se pulsa alguna tecla.

2.- La CPU no hace trabajo útil hasta q. se produce el suceso.

1.- Al producirse el evento (pulsación de una tecla), el periférico solicita atención (interrupción)

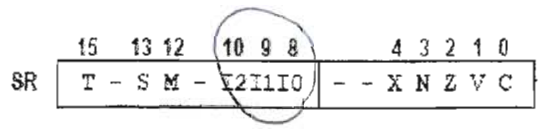
2.- El programa se ejecuta normal/ mientras no se produce el suceso.

Ejemplo 4.3

Con este ejemplo se pretende interpretar el significado del listado de código que se presenta a continuación. En dicho código, se configura una interrupción externa, para lo que tenemos la siguiente información:

- Se inyecta una señal cuadrada por INT3, y se quiere que tras 10 interrupciones, estas queden deshabilitadas.
- INT 3 se debe configurar para que sea sensible a flancos de subida y se le asignará un nivel de prioridad 5.
- VBR=$0 (casi siempre será así)
- PIVR= 64 = $40 (casi siempre será así)

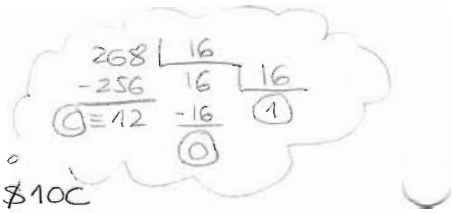
Tenga en cuenta además:



a) Calcule la dirección del vector de interrupción de INT3 en memoria. Calcule los valores que se deberán cargar inicialmente en los registros ICRx y PITR. Calcule además las máscaras que deberá aplicar al SR para permitir las interrupciones de cualquier nivel y para inhibirlas (excepto las de nivel 7 que son no enmascarables).

b) Interprete el código siguiente:

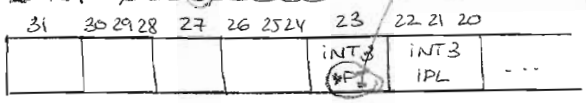
```
MBAR_MEM    EQU     $01000000       * SIM
ICR1        EQU     MBAR_MEM+$20
PITR        EQU     MBAR_MEM+$34
PIVR        EQU     MBAR_MEM+$3F

**** Tabla de vectores ****
            ORG     $0
            DC.L    FINPILA
            DC.L    PPAL
            ORG     $10C
            DC.L    PREP_INT3

**** Zona de programa ****
            ORG     $2000
PPAL        MOVEQ.L #64,D0          * Programa principal
            MOVE.B  D0,PIVR         * Inicialización de PIVR
            MOVE.L  #$00D00000,D0   * Inicialización de ICR1
            MOVE.L  D0,ICR1         * (sólo interrumpe INT3)
            MOVE.L  #$20000000,D0
            MOVE.L  D0,PITR         * Inicialización de PITR
            MOVE.W  SR,D0           * Habilitación de
            ANDI    #$F8FF,D0       * interrupciones en SR
            MOVE.W  D0,SR
            CLR.L   CONT_INT3

            (...)
            MOVEQ.L #10,D1
BUCLE       CMP     CONT_INT3,D1
            BGE     BUCLE
            MOVE.W  SR,D0           * Inhabilitación de interrupciones
            ORI     #$0700,D0
            MOVE.W  D0,SR
            MOVE.L  #$00800000,D0   * Inhabilitación de la interrupción
            MOVE.L  D0,ICR1         * de INT3 (la única activa) en ICR1
FIN         (...)

**** Rutina de atención a la interrupción INT3 ****
PREP_INT3   MOVE.L  D0,-(A7)
            MOVE.L  ICR1,D0         * Se pone a 0 el bit INT3PI de ICR1
            ORI     #$00800000,D0   * escribiendo un 1
            MOVE.L  D0,ICR1
            MOVEQ.L #1,D0           * Se incrementa del contador de
            ADD     D0,CONT_INT3    * interrupciones
            MOVE.L  (A7)+,D0
            RTE

**** Zona de variables **** (RAM)
            ORG     $4000
CONT_INT3   DC.L    0               * Contador de interrupciones
PILA        DS.B    1024            * Zona de pila
FINPILA
```



(a) Como PIVR = 64 → Nº Vector int. 3 = PIVR + 3 = 67

Dirección vector 67 = VBR + 4 × Nº Vector = 0 + 4.67 = 268 = $10C

- ICR1 = $ 00 0 00000 → bit de interrupción pendiente = 1



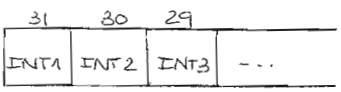
0 000 0 000 1 101

Nivel prioridad 5

el resto a 0 pq están inhabilitadas.

→ hay q. poner un 1 cuando pones un valor en los bits IPL

- PITR = $ 2000 0000



0 0 1 0

2

← Registro de flanco de interrupción programable

Sensible a ... { Flanco de subida = 1. / Flanco de bajada = 0.

- SR | T - S M - I2 I1 I0 | - - - X N Z C V |

AND % 1111 1000 1111 1111 → permitir todas las interrupciones

$ F 8 F F

OR % 0000 0111 0000 0000 → inhibir todas excepto las de nivel 7 (no enmascarable)

$ 0 7 0 0

Ejemplo 4.4

Es un ejemplo de utilización del Watchdog software con generación de reset. Se sabe que el código de un bucle tarda 20 ms como máximo en ejecutarse. Se quiere detectar si dicho código puede incluir algún bucle infinito. Para ello se programa un tiempo máximo entre escrituras en el registro WCR de 40 ms. Se pide:

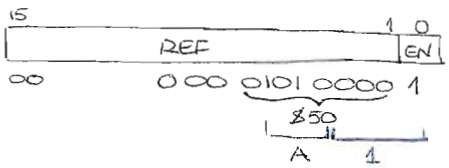
a) Configurar para un reloj de 66 MHz los registros necesarios para inicializar el módulo watchdog software.
b) Indicar como incluiría la escritura en el registro WCR dentro del código del bucle.

(a) → periodo de escritura

$T = (REF+1) \cdot 2^{15}$ ciclos de reloj $= (REF+1) \cdot 2^{15} \cdot \frac{1}{f_{CLK}}$ (T_{CLK})

$REF = \frac{T}{2^{15}} f_{CLK} - 1 = \frac{40 \cdot 10^{-3}}{2^{15}} \cdot 66 \cdot 10^{6} - 1 = 79'56$

$\Rightarrow REF = 80 = \$50$



WRRR = $00A1 (genera de reset)

¿WCR y WER no son necesarios inicializarlos

(b)

```
BUCLE...        ; se ejecuta en 20 ms
                ; - permite detectar si se ejecuta bien o está en bucle oo
MOVE.W  D0,WCR  ; * Da igual el valor q. metamos
                ; * en WCR (se reinicia igualmente)

BRA BUCLE
```

Ejemplo 4.5

Se desea pasar a una configuración de bajo consumo consistente en:
- Modo dormido (apagar reloj de la CPU)
- Apagar los módulos DRAM, DMA, PWM, QSPI y TIMER
- Sólo se debe despertar por medio de una interrupción externa INT5 de nivel 3 (suponemos que la interrupción ya ha sido configurada)

a) Calcular los valores de los registros PMR y PIWR y el que habrá que introducir en el registro de estado para configurar el módulo de bajo consumo de la manera indicada

b) Escribir el código necesario

PMR = Registro de gestión de alimentación
PIWR = Registro de fuente de interrupc

(a) { xPDN = 1 ⇒ apaga módulo
xWIC = 1 ⇒ despierta módulo

- PMR:

DRAM PDN = %1
DMA PDN = %1
PWM PDN = %1
QSPI PDN = %1
TIMER PDN = %1
MOS = %0
SLPEN = %1 (modo dormido)

} PMR = % 0000 0001 1111 0000
0000 0001 0000 =
= $ 01F00010

- PIWR:

INT5 = %1
Todos los demás a 0 } PIWR = $ 00000040

- en el SR habrá q. poner un nivel ≤ 2 para q. la INT5 de nivel 3 pueda ser atendida.

T=0, S=1, M=0 $I_2 I_1 I_0$ = 0 1 0, X=N=Z=V=C=0

⇒ SR = $ 2200

(b)

```
MOVE.L  #$00000040, D0      ! no se puede hacer move de un valor inmediato
MOVE.L  D0, PIWR            a un absoluto ⇒ D0
MOVE.L  #$01F00010, D0
MOVE.L  D0, PMR
MOVE.W  D0, ALPR    * escribe cualquier cosa en ALPR   } Deben ir seguidas!
STOP    #2200       * pone nivel 2 en la máscara       }
```

Ejercicio 9. Preguntas cortas (TEMA 4)

a) El código de atención a una excepción empieza guardando en la pila el valor de D0. ¿En qué dirección se almacena dicho valor si justo antes de la excepción A7=$00108B19?

↳ impar

b) Describa brevemente cómo se habilita y cuándo se produce la excepción del modo traza una vez habilitada.

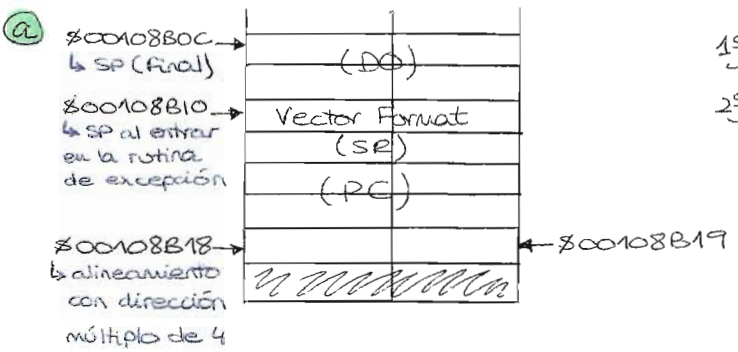
c) Explique brevemente el procesamiento de la excepción RESET.

d) Explique de qué maneras se puede producir la excepción de instrucción ilegal si los ensambladores no admiten escribir una instrucción que no sea válida.

↳ código de operación no existe

e) ¿Qué diferencias hay entre el modo dormido y el modo parado?

f) Determine la configuración del registro WRRR del *watchdog* software de un MCF5272 a 60 MHz para que se produzca un error en la ejecución de una parte del código si no se ha producido una escritura en 80 ms. Justifique el valor en cada campo del registro.



1º) se alinea el puntero (con direc. mult. 4),
2º) se escribe por encima de ese punto

b) Se habilita poniendo T=1 en el SR y una vez habilitada se produce tras cada instrucción.

c) Ver teoría

d) - Involuntariamente: por errores de direccionamiento tras instrucciones como JMP $XXXXXXXX o JMP (A0) con A0 iniciado incorrecta/ de manera q. salte a 1 zona de datos.

- Voluntariamente: se puede producir pq. el programador inserta una directiva DC.W apropiada en la zona de código

e) en modo dormido se inhabilita sólo la CPU y en modo parado se inhabilitan todos los módulos del sist. Además en modo dormido se puede despertar x interrupc. externas y tb. de los módulos internos mientras q. en el modo parado sólo se puede despertar x interrupc externas.

(f) (Igual q. ejemplo 4.4)

REF = 145'48 = 145 = $91

EN = 1

$$T = (REF+1)\, 2^{15} \cdot \frac{1}{f_{CLK}}$$

$$80ms = (REF+1)\, 2^{15} \frac{1}{60MHz}$$

$$REF = \frac{80 \cdot 10^{-3} \cdot 60 \cdot 10^{6}}{2^{15}} - 1.$$

Ejercicio 10. sistema de cronometraje automático para piscinas olímpicas (TEMA 4)

Se ha diseñado un sistema de cronometraje automático para piscinas olímpicas que permite medir con la mayor precisión posible los tiempos de paso por la meta en las pruebas de natación así como detectar las salidas falsas. Estas medidas se realizan para las 8 calles de la piscina simultáneamente. Los bloques del sistema se muestran en la Figura 1.

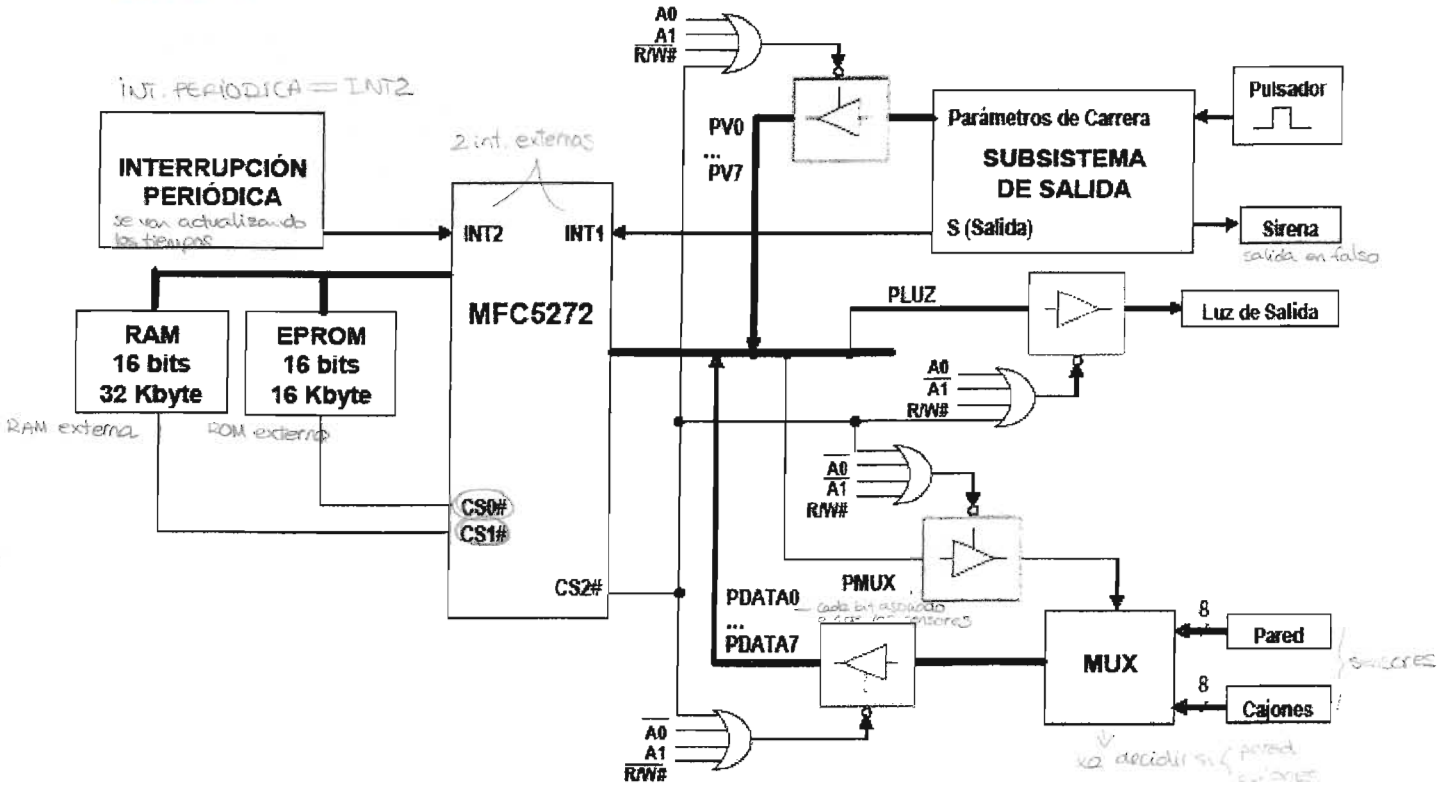


Figura 1. Diagrama de bloques del sistema

Para la realización de estas tareas el sistema consta de los siguientes dispositivos:

1. Un microcontrolador MFC5272 a 66 MHz.
2. Sensores de presión: en cada una de las calles se han instalado sensores de presión tanto en los cajones de salida, para determinar salidas falsas, como en la pared de llegada, para la medida de tiempos de carrera y la determinación de salidas falsas en las pruebas de estilo espalda. Los sensores de los cajones de salida están multiplexados con los sensores de la pared ya que no se utilizan simultáneamente. Los sensores se activan al detectar una variación de presión poniéndose a "1" durante un 1 ms.
3. Subsistema de Salida: introduce los parámetros de la carrera y determina el instante de salida.
4. La luz de salida permite al Juez verificar que no ha habido ninguna salida falsa.
5. Generador de Interrupción Periódicas. El tiempo entre interrupciones consecutivas es de 1ms y se usa el nivel de prioridad de interrupción 2. (ya medir los tiempos)

Se distinguen tres estados en todo el sistema: NoCarrera, Preparados y Carrera. NoCarrera corresponde con la situación de inicio. El juez avisa a los nadadores para que se dispongan en sus puestos ("Preparados") presionando el pulsador del subsistema de salida que activa la sirena, provocando una primera activación de la señal S, y envía información sobre la prueba a realizar. Una vez que los nadadores están en sus puestos el juez marca el inicio de la prueba

presionando de nuevo el pulsador, lo cual activa por segunda vez la sirena de aviso y la señal S. La activación de la señal S en ambos casos produce una interrupción por INT1 de nivel 5.

Todas las señales de control se conectan al bus a través de un subsistema de E/S que tiene configurada su activación para puerto de 8 bits. Así la información de carrera se transfiere por el puerto PV según la siguiente codificación: ! son puertos de 8 bits

- PV1-PV0 = Estilo: 0=Espalda, 1=Mariposa, 2=Braza, 3=Libre.
- PV5-PV2 = Longitud de la prueba: 1=100 m, 2=200 m, 4=400 m, 8=800 m, 15=1500 m. Teniendo en cuenta que la piscina olímpica tiene una longitud de 50 m, esta codificación coincide exactamente con el número de veces que cada nadador recorre la piscina ida y vuelta (número de vueltas).

Por otro lado, se utilizan otros tres puertos del subsistema de E/S: PDATA, refleja la activación de los sensores de presión. PLUZ permite activar la luz de salida (0=verde, 1=rojo) y PMUX permite controlar el multiplexor de los sensores de presión (0=cajones, 1=pared).

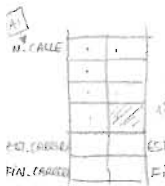
Una vez que se da como válida la salida, el cronómetro automático se pone en marcha y se almacenan los valores de tiempos parciales de cada vuelta para cada calle hasta completar el número de vueltas de la carrera determinando entonces el tiempo total. Cuando la carrera ha finalizado los tiempos de carrera para cada calle se envían a un sistema de representación, que no es objeto de este ejercicio.

Listado Parcial del Programa

```
*****************************************************************************
*       DEFINICIÓN DE CONSTANTES
*****************************************************************************
MBAR_MEM        EQU             $100000     define 1ª reg del módulo SIM
(puesto en el
ISR             EQU             MBAR_MEM+$30
PITR            EQU             MBAR_MEM+$34
PIVR            EQU             MBAR_MEM+$3F
ICR1            EQU             MBAR_MEM+$20    direcciones de los reg del
ICR2            EQU             MBAR_MEM+$24    controlador de interrupciones
ICR3            EQU             MBAR_MEM+$28
ICR4            EQU             MBAR_MEM+$2C

CSBR0           EQU             MBAR_MEM+$40
CS0R0           EQU             MBAR_MEM+$44
CSBR1           EQU             MBAR_MEM+$48    direc de mem. de los reg. de cs
CS0R1           EQU             MBAR_MEM+$4C
CSBR2           EQU             MBAR_MEM+$50
CS0R2           EQU             MBAR_MEM+$54

BASE_ES         EQU             $300000     define direc base mem. de bs E/S
PV              EQU             BASE_ES+$0
PDATA           EQU             BASE_ES+$1
PLUZ            EQU             BASE_ES+$2  direc de los puertos de E/S
PMUX            EQU             BASE_ES+$3

N_CALLES        EQU             9    nº calles

TABLA DE        ORG             $00000000 → b 1ª q hay, es el vector asociado al reset
VECTORES        DC.L            YYYYYYYY  valor crucial del puntero pila, lo normal es iniciarlo:
                DC.L            PRINCIPAL
                                                1ª vez rango asignado a la RAM => p pila
VECTOR DE       ORG             XXXXXXXX  (dir mem. del vector de interrup asociado a la INT2)
INT 2           DC.L            INT_TEMP    es 5th direc RAM+1
                                            ↳ direc de la rutina de atenc periodica asociada a la INT2
VECTOR DE       ORG             KKKKKKKK  direc del vector de interrupc de la interrupc del subsist de salida
INT 1           DC.L            INT_SAL
```

(ver diagrama bloques)

```
*************************************************************************
*       DEFINICIÓN DE VARIABLES (RAM)
*************************************************************************
                ORG             $10000
CNT_VUELTAS     DS.L            1
N_VUELTAS       DS.L            1
SAL_FALS        DS.B            1           * Detección de salidas falsas
T_CARRERA       DS.L            1
T_CALLE         DS.L            N_CALLES
N_CALLE         DS.B            N_CALLES
EST_CARRERA     DS.B            1           * 0=NoCarrera, 1=Preparados, 2=Carrera
ESTILO          DS.B            1           * 0=Espalda, 1=Mariposa, 2=Braza, 3=Libre
FIN_CARRERA     DS.B            1
FIN_CALLES      DS.B            1

*************************************************************************
*       PROGRAMA PRINCIPAL
*************************************************************************
                ORG             $400
PRINCIPAL       move.l          #$00100001,D0
                movec           D0,MBAR
                move.l          #$_________,D0
                move.l          D0,CSBR0
                move.l          #$_________,D0
                move.l          D0,CSOR0
                move.l          #$_________,D0  *Inicialización del CS de la RAM
                move.l          D0,CSBR1
                move.l          #$_________,D0
                move.l          D0,CSOR1
                move.l          #$_________,D0  *Inicialización del CS del Sub. de E/S
                move.l          D0,CSBR2
                move.l          #$_________,D0
                move.l          D0,CSOR2
                bsr             INICIO_INT

INICIO          bsr             INICIO_SW
PPAL            tst.b           FIN_CARRERA
                beq             PPAL
                bsr             DISP_TIEMP
                bra             INICIO

*** Rutina de configuración de las interrupciones
INICIO_INT      move.w          #$2700,SR
                moveq.l         #$40,D0         * Programa principal
                move.b          D0,PIVR         * Inicialización de PIVR
                move.l          #$11111111,D0   * Inicialización de ICRx
                move.l          D0,ICRx
                move.w          SR,D0           *  (*)
                andi            #$FBFF,D0       *  (*)
                move.w          D0,SR           *  (*)
                rts

*** Rutina de inicialización del SW del sistema
INICIO_SW       clr.l           CNT_VUELTAS
                clr.l           N_VUELTAS
                clr.b           EST_CARRERA
                clr.b           SAL_FALS
                clr.b           FIN_CARRERA
                clr.l           T_CARRERA
                clr.b           FIN_CALLES
                lea.l           N_CALLE,A1
                move.l          #N_CALLES,D1
BORRADO         clr.b           -1(A1,D1)
                subq            #1,D1
                bne             BORRADO
                clr.b           FLUZ            * Luz de salida verde
                rts
```



```
*** Rutina que envía los tiempos de carrera de cada calle al sistema de
*** representación.
DISP_TIEMP    ...
              rts

*** Rutina que notifica las salidas falsas a través del sistema de
*** representación.
DISP_FALLO    ...
              rts

*****************************************************************
*       ZONA DE RUTINAS DE ATENCION A INTERRUPCIONES
*****************************************************************
*** Rutina de interrupción del subsistema de E/S. salida (pulsador)
INT_SAL       adda.l      #-8,A7      ya puedo salvar reg. D0-D1
              movem.l     D0-D1,(A7)
              move.l      ICRx,D0           * Poner a 0 el bit INT1PI
              ori         #$80000000,D0     máscara y pone 1 en bit INT 1 PI y ya poniendo a 0 (se pone poniendo 1)
              move.l      D0,ICRx
              clr.l       D0
              tst.b       EST_CARRERA       * Comprobar el estado de carrera
              bne         E_PREPARADOS      si no están en estado 'EST-CARRERA', están en estado E. PREPARADOS
              clr.b       PLUZ              * luz de salida verde (lo pone a '0')
              move.w      10(A7),D0         si estamos en estado NOCARRERA:
              andi        #$F8FF,D0         pone I2I1I0 = 000 ⇒ inhabilita interrupc.
              move.w      D0,10(A7)
              move.l      #1,D0
              move.b      D0,EST_CARRERA    Pasamos al estado PREPARADOS
              move.b      PV,D0             se transfieren datos x puerto PV. lee (PV) → D0
              move.l      D0,D1
              andi        #03,D0            manipula x quedarse con los datos
              move.b      D0,ESTILO         recorda: PV1-PV0. estilo PV1-PV2 ; borra → se queda con los 2 últ. bits: PV0-PV1
              lsr.l       #2,D1             desplaza hacia dcha
              andi        #$0F,D1           x quedarse con los bits correspondientes a PV1-PV2
              move.l      D1,N_VUELTAS
              bra         SALIR

E_PREPARADOS  tst.b       SAL_FALS          se comprueba si hay salida falsa
              bne         FALLO             si es cero ⇒ todo correcto  sino FALLO (ninguna salida en falso)
              move.l      #2,D0
              move.b      D0,EST_CARRERA    estás en el estado CARRERA
              bra         SALIR

FALLO         move.b      #$FF,D0           Si hay una salida falsa se pone la luz de salida roja
              move.b      D0,PLUZ           * luz de salida rojo
              bsr         DISP_FALLO        salta a la rutina
              move.l      #0,D0             vuelves al estado 'no carrera'
              move.b      D0,EST_CARRERA
              clr.b       SAL_FALS          se reinicia la var. SAL_FALS
              move.w      10(A7),D0         ¿inhibe interrupc. periódicas
              ori         #$0300,D0         pone I2I0 = 11 nivel 2 de int ⇒ sólo int del subsist. de salida
              move.w      D0,10(A7)                                  estado no carrera ⇒ int 1
SALIR         movem.l     (A7),D0-D1        * Recuperar el contexto
              adda.l      #8,A7
              rte
                          fin de rutina de int. 1
*** Rutina de atención a la interrupción periódica (se ejecuta cada 1ms)
INT_TEMP      adda.l      #-24,A7           guardar reg
              movem.l     D0-D1/D3-D4/A0-A1,(A7)   * Salvar el contexto
    INT2      move.l      ICRx,D0                  * Poner a 0 el bit INT2PI
              ori         #$08000000,D0     se escribe en INT2 PI ya poniendo a 0
              move.l      D0,ICRx
              clr.l       D0
              move.b      EST_CARRERA,D0
              cmp         #1,D0             1 = preparados
              bne         E_CARRERA         * Comprobar el estado Preparados
              tst.b       ESTILO            si no está en el preparado, está en carrera
              beq         ESPALDA
```

```
                clr.b       PMUX            * Activar sensores de cajón si la prueba
                bra         F_SALIDAS         no es de espalda
ESPALDA         move.b      #$FF,D0           si es de espalda se
                move.b      D0,PMUX         * Activar sensores de pared
F_SALIDAS       move.b      SAL_FALS,D0
                move.b      PDATA,D1
                or          D1,D0
                move.b      D0,SAL_FALS
                bra         FIN_TEMP
E_CARRERA       move.b      #$FF,D0
                move.b      D0,PMUX         * Activar sensores de pared
                move.l      T_CARRERA,D0
                addi.l      #1,D0
                move.l      D0,T_CARRERA
                lea         T_CALLE,A0
                lea         N_CALLE,A1
                clr.l       D3
                clr.l       D4
                move.b      PDATA,D0
                move.l      #N_CALLES-1,D1
CALLE           btst.b      D1,D0
                beq         OTRA_CALLE
                move.l      T_CARRERA,D0
                move.l      D0,(A0,D1*4)
                move.b      (A1,D1),D3
                addi        #1,D3
                move.b      D3,(A1,D1)      * Número de vueltas por calle
                move.l      N_VUELTAS,D4
                cmp         D4,D3
                bne         OTRA_CALLE
                move.b      FIN_CALLES,D4
                addi        #1,D4
                move.b      D4,FIN_CALLES
OTRA_CALLE      subq        #1,D1
                bge         CALLE
                cmp         #8,D4
                bne         FIN_TEMP
                move.l      #1,D0
                move.b      D0,FIN_CARRERA
                move.w      26(A7),D0
                ori         #$0300,D0
                move.w      D0,26(A7)
FIN_TEMP        movem.l     (A7),D0-D1/D3-D4/A0-A1      * Recuperar el contexto
                adda.l      #24,A7
                rte
                end
```

1. Configuración del subsistema de memoria

1.1 Complete la siguiente tabla con el mapa de memoria que se deduce a partir de la información del enunciado y del listado.

DIRECCIONES (HEX)	DISPOSITIVO ASIGNADO
$0000 0000 - $0000 3FFF	Memoria EPROM
$0001 0000 - $0001 7FFF	Memoria RAM
$0003 0000 - $0003 0003	E/S
$0010 0000 - $0010 FFFF	SIM y módulos internos

1.2 Teniendo en cuenta el listado y las especificaciones del sistema complete los valores de la rutina INICIO_HW. Justifique cada uno de los campos de los registros.

ROM: CSBR0 = $ 00000201

BA = $ 00000 . Dirección base de la ROM $ 0000 0000 (BA)

EBI = % 00 SRAM o ROM de 16/32 bits
BW = % 10 anchura de bus de 16 bits
} tamaño del puesto

SUPER = % 0 activo siempre
ENABLE = % 1 Habilitado

CSOR0 = $ FFFFC001

BAM = $ FFFFC → 16 KB = 2^{14}
WS = % 00000 sin estados de espera
RW = 0 memoria de sólo lectura
MRW = 1 según indique RW

RAM: CSBR1 = $ 00010201

BA = $ 00010 Dirección base de la RAM $ 0001 0000
EBI = % 00 SRAM o ROM de 16/32 bits
BW = % 10 anchura de bus de 16 bits (tamaño del puesto)
SUPER = % 0 activo siempre
ENABLE = % 1 Habilitado

CSOR1 = $ FFFF8000

BAM = $ FFFF8 → 32 KB = 2^{15}
WS = % 00000 sin estados de espera
RW = 0 Indiferente
MRW = 0 lectura/escritura

E/S: CSBR2 = $ 00030D01

BA = $ 00030 Dirección base de E/S $ 0003 0000
EBI = % 11 SRAM o ROM de 8 bits (tamaño del puerto de mem)
BW = % 01 anchura de bus de 8 bits
Si no te dan el tamaño del bus, es el mismo q el del puerto
SUPER = % 0 activo siempre
ENABLE = % 1 Habilitado

CSOR2 = $ FFFFF000

BAM = $ FFFFF → 3 bytes = 2^2
WS = % 00000 sin estados de espera
RW = % 0 Indiferente
MRW = % 0 lectura / escritura

1.3 ¿Qué cambios serían necesarios si se usara la RAM local del MFC5272 en vez de la RAM externa propuesta?

Ejercicio 10. sistema de cronometraje automático para piscinas olímpicas (TEMA 4)

Se ha diseñado un sistema de cronometraje automático para piscinas olímpicas que permite medir con la mayor precisión posible los tiempos de paso por la meta en las pruebas de natación así como detectar las salidas falsas. Estas medidas se realizan para las 8 calles de la piscina simultáneamente. Los bloques del sistema se muestran en la Figura 1.

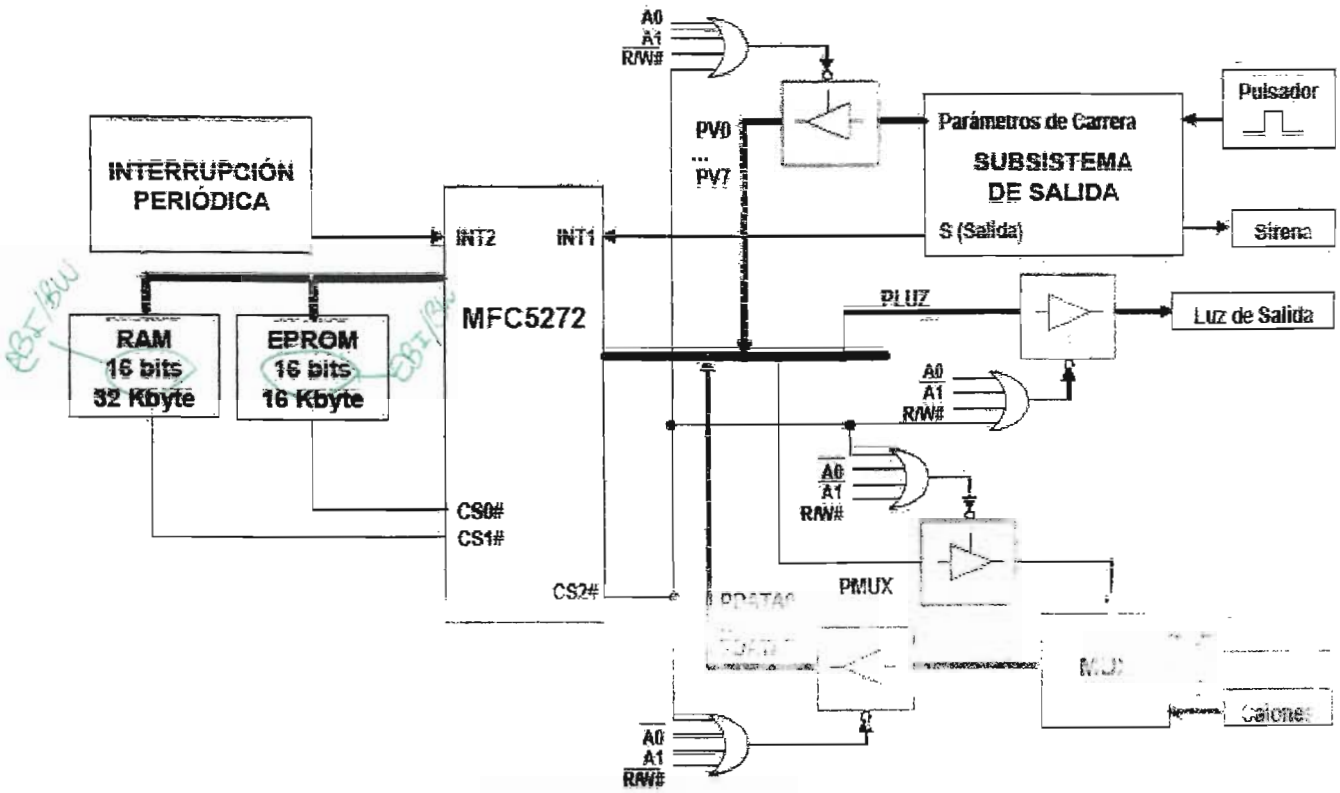


Figura 1. Diagrama de bloques del sistema

Para la realización de estas tareas el sistema consta de los siguientes dispositivos:

1. Un microcontrolador MFC5272 a 66 MHz.
2. Sensores de presión: en cada una de las calles se han instalado sensores de presión tanto en los cajones de salida, para determinar salidas falsas, como en la pared de llegada, para la medida de tiempos de carrera y la determinación de salidas falsas en las pruebas de estilo espalda. Los sensores de los cajones de salida están multiplexados con los sensores de la pared ya que no se utilizan simultáneamente. Los sensores se activan al detectar una variación de presión poniéndose a "1" durante un 1 ms.
3. Subsistema de Salida: introduce los parámetros de la carrera y determina el instante de salida.
4. La luz de salida permite al Juez verificar que no ha habido ninguna salida falsa.
5. Generador de Interrupción Periódicas. El tiempo entre interrupciones consecutivas es de 1ms y se usa el nivel de prioridad de interrupción 2. (N=2)

Se distinguen tres estados en todo el sistema: NoCarrera, Preparados y Carrera. NoCarrera corresponde con la situación de inicio. El juez avisa a los nadadores para que se dispongan en sus puestos ("Preparados") presionando el pulsador del subsistema de salida que activa la sirena, provocando una primera activación de la señal S, y envía información sobre la prueba a realizar. Una vez que los nadadores están en sus puestos el juez marca el inicio de la prueba

presionando de nuevo el pulsador, lo cual activa por segunda vez la sirena de aviso y la señal S. La activación de la señal S en ambos casos produce una interrupción por INT1 de nivel 5. (N5)

Todas las señales de control se conectan al bus a través de un subsistema de E/S que tiene configurada su activación para puerto de 8 bits. Así la información de carrera se transfiere por el puerto PV según la siguiente codificación:

- PV1-PV0 = Estilo: 0=Espalda, 1=Mariposa, 2=Braza, 3=Libre.
- PV5-PV2 = Longitud de la prueba: 1=100 m, 2=200 m, 4=400 m, 8=800 m, 15=1500 m. Teniendo en cuenta que la piscina olímpica tiene una longitud de 50 m, esta codificación coincide exactamente con el número de veces que cada nadador recorre la piscina ida y vuelta (número de vueltas).

Por otro lado, se utilizan otros tres puertos del subsistema de E/S: PDATA, refleja la activación de los sensores de presión. PLUZ permite activar la luz de salida (0=verde, 1=rojo) y PMUX permite controlar el multiplexor de los sensores de presión (0=cajones, 1=pared).

Una vez que se da como válida la salida, el cronómetro automático se pone en marcha y se almacenan los valores de tiempos parciales de cada vuelta para cada calle hasta completar el número de vueltas de la carrera determinando entonces el tiempo total. Cuando la carrera ha finalizado los tiempos de carrera para cada calle se envían a un sistema de representación, que no es objeto de este ejercicio.

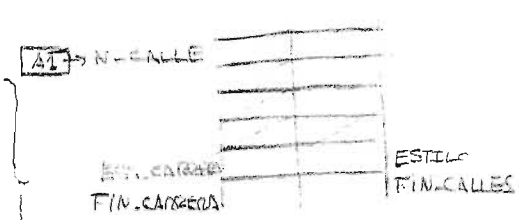
Listado Parcial del Programa

```
*****************************************************
*       DECLARACIÓN DE CONSTANTES
*****************************************************
MBAR_MEM    EQU     $100000

ISR         EQU     MBAR_MEM+$30
PITR        EQU     MBAR_MEM+$34
PIVR        EQU     MBAR_MEM+$3F
ICR1        EQU     MBAR_MEM+$20
ICR2        EQU     MBAR_MEM+$24
ICR3        EQU     MBAR_MEM+$28
ICR4        EQU     MBAR_MEM+$2C

CSBR0       EQU     MBAR_MEM+$40
CS0R0       EQU     MBAR_MEM+$44
CSBR1       EQU     MBAR_MEM+$48
CS0R1       EQU     MBAR_MEM+$4C
CSBR2       EQU     MBAR_MEM+$50
CS0R2       EQU     MBAR_MEM+$54

BASE_ES     EQU     $30000
PV          EQU     BASE_ES+$0
PDATA       EQU     BASE_ES+$1
PLUZ        EQU     BASE_ES+$2
PMUX        EQU     BASE_ES+$3

N_CALLES    EQU     8

            ORG     $00000000
            DC.L    $YYYYYYYY
            DC.L    PRINCIPAL
            ORG     $JJJJJJJJ
            DC.L    INT_TEMP
            ORG     $KKKKKKKK
            DC.L    INT_SAL
```

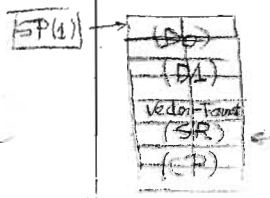
```
*************************************************************************************
*       DEFINICIÓN DE VARIABLES                    RAM
*************************************************************************************
                ORG             $10000
CNT_VUELTAS     DS.L            1
N_VUELTAS       DS.L            1
SAL_FALS        DS.B            1           * Detección de salidas falsas
T_CARRERA       DS.L            1
T_CALLE         DS.L            N_CALLES
N_CALLE         DS.B            N_CALLES    *Numero de vueltas en cada calle
EST_CARRERA     DS.B            1           * 0=NoCarrera, 1=Preparados, 2=Carrera
ESTILO          DS.B            1           * 0=Espalda, 1=Mariposa, 2=Braza, 3=Libre
FIN_CARRERA     DS.B            1
FIN_CALLES      DS.B            1

*************************************************************************************
*       PROGRAMA PRINCIPAL
*************************************************************************************
                ORG             $400
PRINCIPAL       move.l          #$00100001,D0
                movec           D0,MBAR
                move.l          #$_________,D0
                move.l          D0,CSBR0
                move.l          #$_________,D0
                move.l          D0,CSOR0
                move.l          #$_________,D0 *Inicialización del CS de la RAM
                move.l          D0,CSBR1
                move.l          #$_________,D0
                move.l          D0,CSOR1
                move.l          #$_________,D0
                move.l          D0,CSBR2
                move.l          #$_________,D0
                move.l          D0,CSOR2
                bsr             INICIO_INT

INICIO          bsr             INICIO_SW
PPAL            tst.b           FIN_CARRERA
                beq             PPAL
                bsr             DISP_TIEMP
                bra             INICIO

*** Rutina de configuración de las interrupciones
INICIO_INT      move.w          #$2700,SR       * S=1, I2 I1 I0 = 111 (Inhibe interrupc.)
                moveq.l         #$40,D0         * Programa principal
                move.b          D0,PIVR         * Inicialización de PIVR
                move.l          #$LLLLLLLL,D0   * Inicialización de ICRx
                move.l          D0,ICRx
                move.w          SR,D0           *   (*)
                andi            #$FBFF,D0       *   (*)
                move.w          D0,SR           *   (*)
                rts

*** Rutina de inicialización del SW del sistema
INICIO_SW       clr.l           CNT_VUELTAS
                clr.l           N_VUELTAS
                clr.b           EST_CARRERA
                clr.b           SAL_FALS
                clr.b           FIN_CARRERA
                clr.l           T_CARRERA
                clr.b           FIN_CALLES
                lea.l           N_CALLE,A1
                move.l          #N_CALLES,D1
BORRADO         clr.b           -1(A1,D1)
                subq            #1,D1
                bne             BORRADO
                clr.b           PLUZ            * Luz de salida verde
                rts
```

AND con %1111 1011 1111 1111 → T-SM-I2I1I0--XNZVC

Para la máscara a 3 ⇒ sólo pueden interrumpir las de nivel > 3 ⇒ Nivel 4 en este caso ⇒ Inhibe las interrupciones periódicas



```
*** Rutina que envía los tiempos de carrera de cada calle al sistema de
*** representación.
DISP_TIEMP    ...
              rts

*** Rutina que notifica las salidas falsas a través del sistema de
*** representación.
DISP_FALLO    ...
              rts

*******************************************************************************
*       ZONA DE RUTINAS DE ATENCION A INTERRUPCIONES
*******************************************************************************
*** Rutina de interrupción del subsistema de E/S.
INT_SAL       adda.l      #-8,A7            * Pundo 1
              movem.l     D0-D1,(A7)
              move.l      ICRx,D0           * Poner a 0 el bit INT1PI
              ori         #$80000000,D0     * Hay que escribir un 1!!
              move.l      D0,ICRx
              clr.l       D0
              tst.b       EST_CARRERA       * Comprobar el estado de carrera
              bne         E_PREPARADOS      * Salta si estamos en estado preparados
              clr.b       PLUZ              * Luz de salida verde
              move.w      10(A7),D0
              andi        #$F8FF,D0         * AND con I2I1I0=000 ⇒ Inhabilita interrupciones
              move.w      D0,10(A7)
              move.l      #1,D0
              move.b      D0,EST_CARRERA    * Pasamos al estado PREPARADOS
              ...
              ...
              move.b      D0,ESTILO
              lsr.l       #2,D1
              andi        #$0F,D1
              move.l      D1,N_VUELTAS
              bra         SALIR

E_PREPARADOS  tst.b       SAL_FALS
              bne         FALLO
              move.l      #2,D0
              move.b      D0,EST_CARRERA
              bra         SALIR

FALLO         move.b      #$FF,D0
              move.b      D0,PLUZ           * Luz de salida roja
              bsr         DISP_FALLO
              move.l      #0,D0
              move.b      D0,EST_CARRERA
              clr.b       SAL_FALS
              move.w      10(A7),D0
              ori         #$0300,D0
              move.w      D0,10(A7)
SALIR         movem.l     (A7),D0-D1        * Recuperar el contexto
              adda.l      #8,A7
              rte

*** Rutina de atención a la interrupción periódica
INT_TEMP      adda.l      #-24,A7
              movem.l     D0-D1/D3-D4/A0-A1,(A7)   * Salvar el contexto
              move.l      ICRx,D0                  * Poner a 0 el bit INT1PI
              ori         #$80000000,D0            (Para ponerlo a 0 se escribe un 1)
              move.l      D0,ICRx
              clr.l       D0
              move.b      EST_CARRERA,D0
              cmp         #1,D0             * Comprobar el estado Preparados
              bne         E_CARRERA         * Salta si está en estado CARRERA
              tst.b       ESTILO            * Si está en estado PREPARADO...
              beq         ESPALDA
```



Tipo de salto que puede usar para #0, duerda

Se queda con el número de vueltas (bits PV5-PV2) y lo guarda en N_VUELTAS

Paso del estado PREPARADOS al estado CARRERA: comprueba que no haya salidas falsas (SAL_FALS = $00)

Si hay una salida falsa se pone la
y se pasa al estado NO CARRERA
y se reinicia la variable SAL_FALS
y se pone I2I1I0 = %011 = 3 ⇒ inhibidas las interrupciones periódicas

```
                clr.b       PMUX            * Activar sensores de pared si la prueba NO es
                bra         F_SALIDAS         espalda
ESPALDA         move.b      #$FF,D0         Si es de espalda se
                move.b      D0,PMUX         * Activar sensores de pared pq se lee desde
F_SALIDAS       move.b      SAL_FALS,D0       la pared.
                move.b      PDATA,D1
                or          D1,D0           Comprueba si se ha activado alguna señal
                move.b      D0,SAL_FALS     de salida en falso y actualiza la variable
                bra         FIN_TEMP        SAL_FALS

E_CARRERA       move.b      #$FF,D0         *Si estamos durante la carrera se
                move.b      D0,PMUX         * Activar sensores de pared para medir
                move.l      T_CARRERA,D0      tiempos
                addi.l      #1,D0           → Suma 1 ms al tiempo de carrera
                move.l      D0,T_CARRERA
                lea         T_CALLE,A0
                lea         N_CALLE,A1
                clr.l       D3
                clr.l       D4
                move.b      PDATA,D0        Comprueba si algún nadador ha llegado a la
                move.l      #N_CALLES-1,D1  pared, calle a calle
CALLE           btst.b      D1,D0
                beq         OTRA_CALLE
Pone el tiempo  move.l      T_CARRERA,D0    Si la ha tocado almacena el tiempo que lle-
actual para     move.l      D0,(A0,D1*4)    va e incrementa el número de vueltas de esa
la calle        move.b      (A1,D1),D3      calle
Incrementa      addi        #1,D3
el número de    move.b      D3,(A1,D1)      * Número de vueltas por calle
vueltas         move.l      N_VUELTAS,D4    Lo compara con el nº de vueltas
                cmp         D3,D4
                bne         OTRA_CALLE      Si es igual es que ha terminado
                move.b      FIN_CALLES,D4     la carrera y lo anota
                addi        #1,D4
                move.b      D4,FIN_CALLES
OTRA_CALLE      subq        #1,D1
                bge         CALLE
                cmp         #8,D4           Si han acabado todos los nadadores
                bne         FIN_TEMP        pone un 1 en FIN_CARRERA
                move.l      #1,D0
                move.b      D0,FIN_CARRERA
                move.w      26(A7),D0       → Pone un 3 en I2,I1,I0 para q sólo pueda
                ori         #$0300,D0         interrumpir el subsistema de E/S
                move.w      D0,26(A7)
FIN_TEMP        movem.l     (A7),D0-D1/D3-D4/A0-A1      * Recuperar el contexto
                adda.l      #24,A7
                rte

                end
```

1. Configuración del subsistema de memoria

1.1 Complete la siguiente tabla con el mapa de memoria que se deduce a partir de la información del enunciado y del listado.

14 bits = 2^{14} = 16 kBytes

DIRECCIONES (HEX)	DISPOSITIVO ASIGNADO
$0000 0000 – $0000 3FFF	Memoria EPROM
$0001 0000 – $0001 7FFF	Memoria RAM
$0003 0000 – $0003 0003	E/S
$0010 0000 – $001 0FFFF	SIM y módulos internos

16 kByte
32 kBytes

15 bits = 2^{15} = 32 kBytes

3 bytes
64 kBytes

16 bits = 2^{16} = 64 kBytes

Mi código: Hay 3 puertos que ocupan 4B cada uno

1.2 Teniendo en cuenta el listado y las especificaciones del sistema complete los valores de la rutina INICIO_HW. Justifique cada uno de los campos de los registros.

ROM: CSBR0 = S00000201

BA = $00000 Dirección base de la ROM $0000 0000 (BA)
EBI = %00 SRAM o ROM de 16/32 bits } tamaño del puerto
BW = %10 Anchura de bus de 16 bits }
SUPER = 0 Activo siempre
ENABLE = %1 Habilitado

CSOR0 = SFFFFC001

BAM = $FFFFC 16 kB = 2^{14} ⇒ Se compararán los 32-14 = 18 bits superiores
WS = %0 0000 Sin estados de espera
RW = 0. Memoria de sólo lectura
MRW = 1. Lectura o escritura indicado por RW

RAM: CSBR1 = S00010201

BA = $00010 Dirección base de la RAM $0001 0000
EBI = %00 SRAM o ROM de 16/32 bits) (tamaño del puerto)
BW = %10 Anchura de bus de 16 bits
SUPER = % Activo siempre
[illegible]
CSOR1 = [illegible]
BAM = [illegible] 32 kB = 2^{15} Se compararán [illegible]
WS = %0 0000 Sin estados de espera
RW = 0. Indiferente
MRW = 0 lectura / escritura)

E/S: CSBR2 = S00030D01

BA = $00030. Dirección base de E/S $0003 0000
EBI = %11 SRAM o ROM de 8 bits
BW = %01 Anchura de bus de 8 bits
SUPER = %0 Activo siempre
ENABLE = %1 Habilitado

CSOR2 = S FFFFF000

BAM = $FFFFF 3 bytes = 2^2 Se compararán los 20 bits superiores (máximo)
WS = %0 0000 Sin estados de espera
RW = %0 Indiferente
MRW = % De lectura/escritura

1.3 ¿Qué cambios serían necesarios si se usara la RAM local del MFC5272 en vez de la RAM externa propuesta?

- No habría que configurar CSBR1 y CSOR1
- No sería necesario conectar el bus externo a la memoria externa
- Habría que configurar RAMBAR coherentemente con el origen de la zona de variables
- ! Podría afectar al valor inicial del puntero de pila si 4 kB < 32 kB

RAM interna = 4 kB.

2. Configuración de interrupciones

2.1 Especifique el nombre del registro de control que se debe utilizar en la configuración de las interrupciones de este sistema (ICRx) tanto en la rutina INICIO_INT como en el resto del listado. Justifique su respuesta.

ICRx = ICR1

Se usan INT1 e INT2 que se configuran en el ICR1 solamente las interrupciones externas

2.2 Determine el valor inicial de dicho registro de control (valor SLLLLLLLL) en la rutina INICIO_INT para que ambas interrupciones estén adecuadamente configuradas. Justifique el contenido de los campos de dicho registro.

31	30 29 28	27	26 25 24	
INT1 M	INT1 IPL	INT2 I2	INT2 IPL	...
1	101 Nivel 5	1	010 Nivel 2	

SLLLLLLLLL = $DA000000

INT1PI=%1 . Para poner este bit a 0 es necesario escribir un 1

INT1IPL=%101 . Nivel 5 para la interrupción del subsistema de salida

INT2IP=%1

INT2IPL=%010 . Nivel 2 para la interrupción periódica

RESTO= 0 El resto de las interrupciones están inhabilitadas

2.3 Explique qué función tienen las líneas marcadas con (*) dentro de la rutina INICIO_INT.

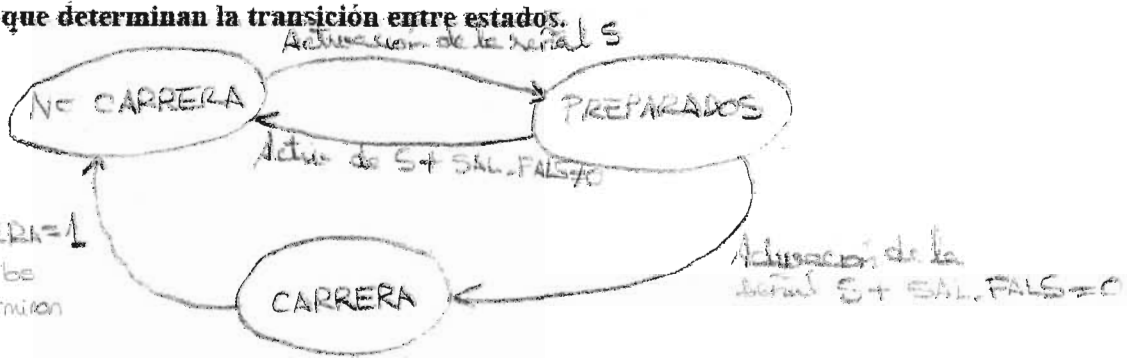
Se programa la máscara de interrupciones que modifica las interrupciones de nivel de prioridad mayor que 3 ($I_2 I_1 I_0 = 011$). Por tanto, esta instrucción habilita la interrupción del subsistema de salida pero no la interrupción periódica

FBFF = 1111 1011 1111 1111

$I_2 = 0$

3. Análisis del software

3.1 Dibuje el diagrama de estados del sistema, destacando, como parte del diagrama, las condiciones que determinan la transición entre estados.



3.2 Explique brevemente en qué parte del código se detecta la ocurrencia de salidas falsas y en qué condiciones. Justifique brevemente la respuesta.

La detección de salidas falsas se realiza en la rutina INT_SAL, en el estado PREPARADOS que verifica si se ha activado la variable SAL_FALSA. Esta variable se pone a un valor distinto de 0 en la rutina INT_TEMP si antes de que se dé la salida de la carrera se ha producido la activación de uno de los sensores de salida, para lo que es necesario que la interrupción periódica esté activa

3.3 Determine el tiempo máximo de carrera que se puede computar en el sistema actual.

T_CARRERA almacena el tiempo transcurrido en ms. Es de tamaño .L

⇒ El tiempo máximo es $(2^{32}-1)$ ms = 4294967295 ms ≈ 1193 horas.

3.4 ¿Qué ocurre si un nadador abandona la prueba? No se contempla esta posibilidad

El sistema no lo puede detectar y se quedaría indefinidamente en el estado CARRERA esperando a que ese nadador finalice la prueba. Hasta q. se resetease el sist

3.5 Determine el valor óptimo de SYYYYYYYY para el máximo aprovechamiento de la memoria. Justifique su respuesta.

Es el valor inicial del puntero de pila. Para que sea lo mayor posible será la dirección del último byte de la RAM+1 = $00013000

4. Interrupciones y excepciones

4.1 Determine los valores SJJJJJJJJ y SKKKKKKKK para que las interrupciones del sistema funcionen correctamente. Justifique su respuesta.

SJJJJJJJJ= $ 108

Como PIVR = $40 = 64 ⇒ A la interrupción INT2 (interrupc periódica) le corresponde el número de vector 66 (vector nº 2 ⇒ 64+2 = 66)

Dirección del vector = VBR + (4 × Nº vector) = 264 = $108

VBR no se inicializa así que vale 0 que es como se queda tras el reset

SKKKKKKKK= $ 104

PIVR = $40 = 64 ⇒ A la int INT1 (interrup. del subs. de salida) le corresponde el vector 65

Dirección del vecto. = VBR + (4 × Nº vector) = 260 = $104

4.2 ¿Es posible entrar en la rutina de atención a la interrupción periódica INT_TEMP en el estado NoCarrera? Justifique su respuesta.

No porque siempre que se esté en el estado NO CARRERA las interrupciones de nivel inferior a 3 están inhabilitadas. Esto es así ya cuando se inicializa el sistema y cada vez que se hace una transición al estado NO CARRERA la máscara de interrupciones del SR se pone a I2I1I0 = 011

La rutina INT_TEMP se ejecuta para atender las interrupciones periódicas q. son de nivel 2

TEMA 5: E/S EN EL SISTEMA MICROPROCESADOR .

5.2 Entrada/Salida
5.3 Entrada/Salida en paralelo
5.4 Entrada/Salida en serie

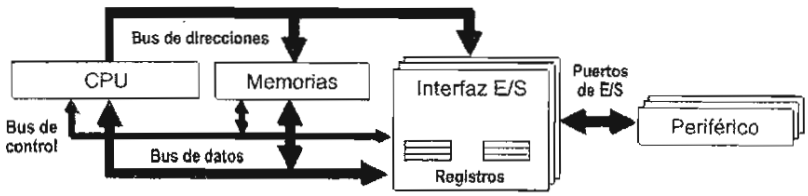
Entre CPU y periféricos hace falta una INTERFAZ

Necesidad de periféricos:
- Para que el sistema interactúe con el mundo exterior
- Para ampliar las capacidades básicas de un sistema

5.1 Entrada/Salida

• PUERTOS relacionado con el bus de datos (TEMA 3)

5.1.1 Interfaces, periféricos y puertos



- Tipos de periféricos:

Estos son particulares para un sistema concreto

- Periféricos que incluyen su propia interfaz: se conectan directamente a los buses de datos y direcciones del sistema, por lo que permiten una comunicación más rápida y su potencia/consumo están limitados por los del sistema.

Estos son multisistema

- Periféricos que usan una interfaz estándar: se conectan a una interfaz genérica conectada a los buses (más lento).
  - ♦ Si la conexión puede ser con el sistema en funcionamiento: *hot-plug*.
  - ♦ Si la alimentación puede venir a través del bus: *power-on-bus*

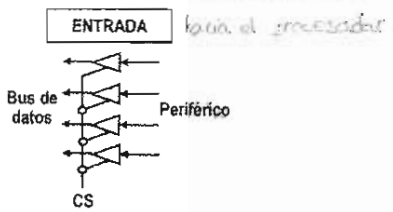
- Interfaces de E/S:
  - Al escribir en los registros del interfaz se provocan acciones en el periférico
  - Al leer los registros, se pueden consultar el estado y los datos del periférico
  - Ocupan posiciones en el mapa de memoria

- Puertos de E/S: tanto los interfaces como los periféricos tienen puertos.

La interfaz más simple que puede tener un periférico es un puerto, sin protocolo ni señales de control, pero puede haber interfaces más complejas que usen señales y protocolos.

Tipos de puertos

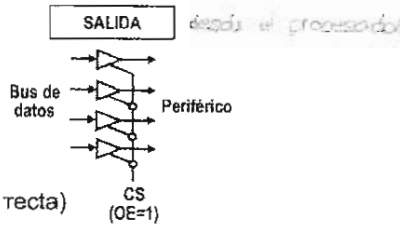
- Puerto de entrada: refleja el valor instantáneo al ser seleccionado



Por ejemplo con búfer 74HC244

Los puertos necesitan una lógica de selección que los active cuando se acceda a ellos (chip-select)

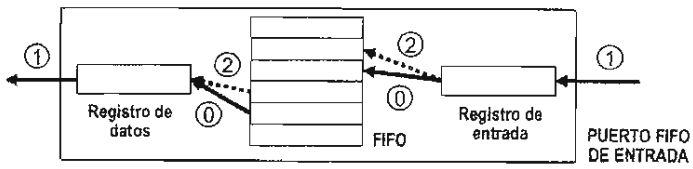
- Puerto de salida: mantiene el valor enviado



Por ejemplo con registro triestado 74HC574

• Son UNIDIRECCIONALES
Si son de entrada no se puede escribir en ellos
Si son de salida no se pueden leer

- Puerto FIFO (first in- first out):
  - Sólo hay un registro accesible directamente
  - Internamente tiene una cola FIFO de registros de datos y/o control, y , punteros de lectura y escritura
  - Cuando se produce una lectura o escritura avanzan los punteros internos



5.1.2 Estándares de codificación de símbolos

Sirven para intercambiar datos entre equipos distintos. Los equipos muchas veces usan representaciones internas diferentes. Algunos importantes son:

- ASCII:
  - Representación numérica de caracteres alfanuméricos
  - 7 bits por carácter
  - Existen extensiones a 8 bits para recoger otros símbolos, como por ejemplo algunos nacionales como la 'ñ'

- Unicode:
  - Recoge los caracteres específicos de multitud de idiomas
  - Hasta 32 bits por carácter

- BCD: codifica cada digito decimal mediante 4 bits

0 → 0000
9 → 1001

5.1.3 Transferencias masivas de datos

Las transferencias masivas pueden ser :

Típico de accesos periférico/memoria o viceversa

- Accesos de E/S entre puerto FIFO y memoria.

Por ejemplo entre un disquete y la RAM

- Transferencias entre memorias (con distintos tiempos de respuesta). ~~Se pueden hacer mediante:~~

En ambos casos se puede hacer mediante:

- Acceso controlado por la CPU:
  - ♦ En el caso puerto FIFO/memoria: leer dato de periférico/memoria y escribirlo en memoria/periférico
  - ♦ En el caso memoria/memoria: el ensamblador obliga al uso intermedio de los registros CPU, porque la instrucción move no permite transferencias directas de memoria a memoria.
  - ♦ Problemas: tasa de transferencia reducida, CPU ocupada

ADM en FDOR

- Acceso directo a memoria (DMA). Es un mecanismo para transferencias masivas entre zonas de memoria.:
  - ♦ Precisa de un controlador que pueda gobernar el bus del sistema
  - ♦ El programa configura los registros del controlador DMA: direcciones fuente y destino, número de bytes a transferir...

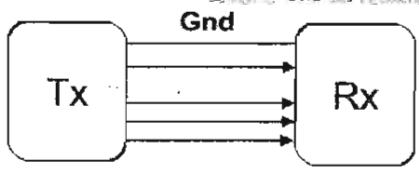
Controlador de DMA del MCF5272

No se ve con más detalle en SEDG

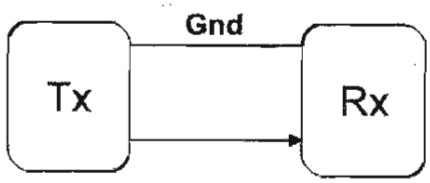
- Transferencias entre SDRAM, SRAM externa y periféricos internos
- Tamaño de los datos (byte, word, long, línea) en fuente y destino configurables
  - Tiene búferes internos para leer con un tamaño y transferir con otro
  - Se transfiere cuando se dispone de un dato completo para el destino
- Modos de dirección DMA:
  - Estático: la dirección no se modifica tras la transferencia (puertos FIFO)
  - Incremental: la dirección se incrementa tras la transferencia (memoria)
- Tiene registros para configurar:
  - Modo de dirección (estática, incremental)
  - Dirección fuente
  - Dirección destino
  - Número de bytes a transferir y restantes para finalizar
  - Interrupciones DMA (por error o fin de transferencia)

5.1.4 Introducción a los tipos de comunicación

- En paralelo: todos los bits del dato se transfieren a la vez
  - 1 línea de E/S por cada bit y líneas opcionales de control
  - Mayor velocidad: más bits y menos formato



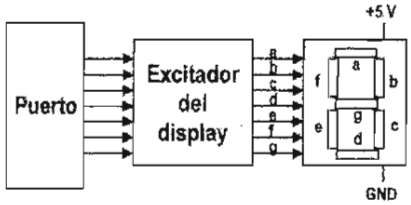
- En serie: sólo hay una línea de datos en cada sentido. Opcionalmente puede haber líneas de reloj o de control
  - Menor coste: menos HW y menos líneas
  - Mayor alcance: no hay retardos entre líneas
  - Menor ruido: no hay diafonía entre líneas



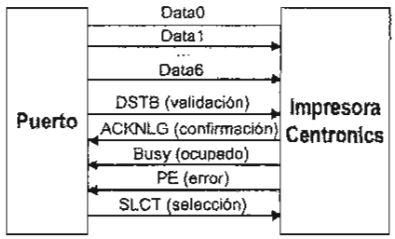
5.2 Entrada/Salida en paralelo

5.2.1 Tipos de comunicación en paralelo

- Simple: sólo hay líneas de datos. Se usa para aplicaciones simples como teclados o visualizadores sencillos



- Con protocolo: líneas de datos y de control. Se usan para aplicaciones más complejas.
  - Las líneas de control típicas son: error, validación, aceptación, ocupación, selección



La interfaz más simple que puede tener un periférico es un PUERTO.

5.2.2 E/S en paralelo del MCF5272

Como microcontrolador, el MCF5272 contiene puertos de E/S de propósito general:

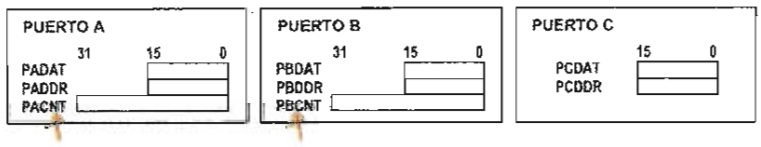
- Son de 16 bits aunque comparten los terminales con otros módulos (multiplexación)
- **Puerto A:** multiplexado parcialmente con PLIC, USB y QSPI
- **Puerto B:** multiplexado parcialmente con UART0, TOUT0 y Ethernet
- **Puerto C:** multiplexado con el bus de datos según WSEL

Recordar lo visto en el tema 3

Registros para programar los puertos de E/S

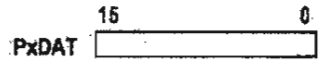
- Registro de datos del puerto de 16 bits PxDAT
- Registro de dirección del puerto de 16 bits PxDDR
- Registro de control del puerto de 32 bits PxCNT (¡OJO! sólo en puertos A y B)

No existe PCCNT

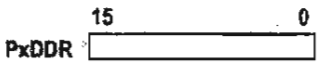


- Registro de datos del puerto **PxDAT**
  - Cada bit puede ser accedido independientemente
  - Se pueden escribir los bits de salida y leer los bits de entrada

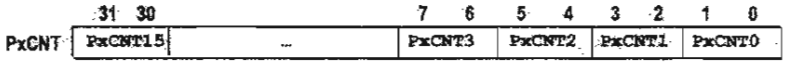
entrada leo
salida escribo



- Registro de dirección del puerto **PxDDR**
  - Indica para cada bit del puerto si es de entrada (bit=0) o salida (bit=1)



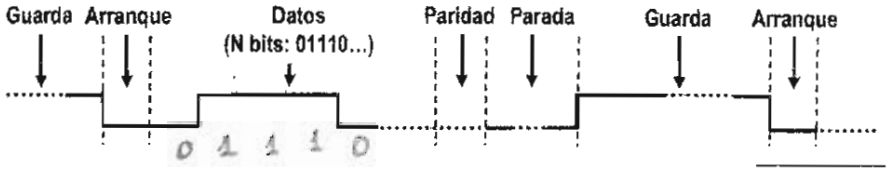
- Registro de control del puerto **PxCNT** (sólo disponible para los puertos A y B)
  - Selecciona qué función multiplexada está activa en cada bit del puerto
    - ♦ 00: función de E/S de propósito general
    - ♦ El resto de combinaciones corresponde a funciones de otros módulos



5.3 Entrada/Salida en serie

5.3.1 Comunicación en serie asíncrona

- Está orientada a símbolos
- La duración de un bit es constante (t_{bit})
- ¡¡No necesita sincronización permanente entre Tx y Rx!!
- Puede existir un tiempo de guarda mínimo entre símbolos
- Formato para transmitir un símbolo:
  - Bit de arranque: permite la sincronización del detector
  - Bits de datos: típicamente entre 5 y 8
  - Bit de paridad: ninguna, par, impar o fija
  - Bits de parada: tienen una valor fijo para detectar errores de comunicación



¡OJO! En otras asignaturas los baudios son símbolos/segundo

- Velocidad de Tx/Rx: número de bits Tx/Rx por segundo. Se mide en baudios=bits/segundo=bps

$$v = \frac{1}{t_{bit}}$$

- Variación de velocidad de Tx/Rx admisible

$$\Delta(\%) = \frac{100}{1 + 2 \cdot (\text{bits de dato} + \text{bits de formato})}$$

bits de formato=bit de arranque + bit de paridad + bits de parada

¡OJO! Si se quisiera expresar en bps habría que multiplicar por "bits de dato"

- Velocidad efectiva (expresada en símbolos por segundo). Se reduce con los tiempos de guarda.

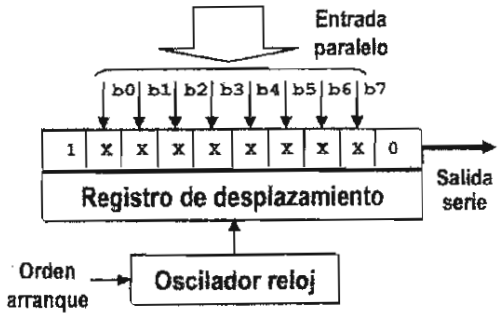
$$v_{eff} = \frac{v_{Tx/Rx}}{\text{bits de dato} + \text{bits de formato} + \text{bits de guarda}}$$

- Velocidades estandarizadas para facilitar la interconexión de dispositivos

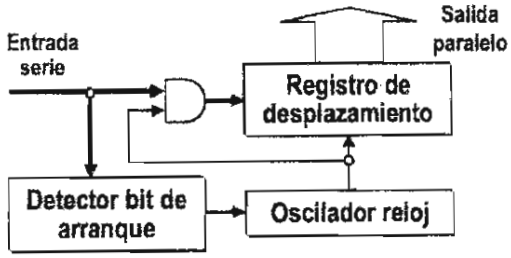
Velocidad Tx/Rx (baudios)	Tiempo de bit (µs)	Velocidad efectiva (8 bits)
110	9090,9	10
2400	416,7	218
9600	104,2	873
19200	52,1	1745
38400	26,0	3491
57600	17,4	5236
115200	8,7	10473
230400	4,3	20945
460800	2,2	41891
921600	1,1	83782

Circuitos básicos para comunicación asíncrona

- Transmisor: realiza la conversión paralelo-serie, y añade los bits de formato



- Receptor: detecta el bit de arranque tras el tiempo de guarda, muestrea los bits, y realiza la conversión serie-paralelo



APRENDER!

Errores de comunicación asíncrona

- Error de desbordamiento (*overrun*)
  - Se recibe un nuevo dato sin haber podido leer/procesar el anterior. Por ejemplo porque se está atendiendo una excepción
  - Se pierden datos
  - Se necesita incrementar el tiempo de guarda entre símbolos
  - Si hay una cola FIFO en el receptor se produce al llenarse la cola

- Error de trama (*framing*): los bits de parada no tienen el valor establecido
- Error de paridad (*parity*): los datos y la paridad recibidos no son coherentes
- Falso arranque (*false start*): el bit de arranque recibido contiene transiciones internas
- Corte de línea (*break):* hay transiciones durante la recepción de un bit

5.3.2 Comunicación en serie síncrona

Bloques de símbolos

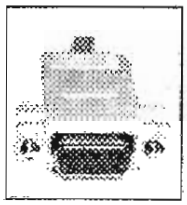
La asíncrona estaba orientada a símbolos

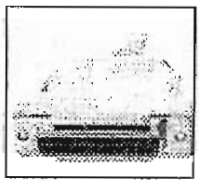
- Está orientada a bloques de símbolos
- Mayor velocidad efectiva que la asíncrona
- El Tx y el Rx tienen el mismo reloj
- Tipos:
  - Reloj compartido: se transmite por una línea independiente
  - Reloj sincronizado: si no se comparte el reloj, el receptor necesita sincronizarse mediante bytes de sincronismo al comienzo de cada trama o enviados periódicamente
  - Reloj recuperado: codificado como parte de la señal enviada

5.3.3 Estándares de comunicaciones en serie

- Constan de especificaciones mecánicas (cómo son los conectores), eléctricas (niveles de tensión, impedancias, tiempos...) y de protocolo (formato de la trama...)
- Los objetivos de estandarizar las comunicaciones en serie son:
  - Abaratar costes (economías de escala)
  - Facilitar la interconexión de equipos de fabricantes diferentes
  - Garantizar ciertos valores de parámetros clave: alcance y velocidad
  - Posibilitar estándares de conexión como *hot-plug*, *power-on.bus*, etc.

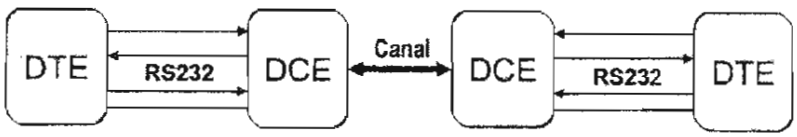
Estándar de comunicación serie asíncrona RS-232C


Conector DB-9


Conector DB-25

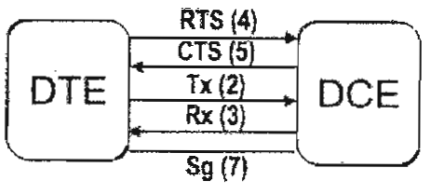
- Nomenclatura:
  - DTE: equipo terminal de datos (ordenador)
  - DCE: equipo de comunicación de datos (módem), que adapta la señal al canal disponible

- Características:
  - Comunicación asíncrona
  - Bidireccional
  - Bipolar
  - Velocidad máxima: 20 kbps
  - Alcance máximo: 15 metros

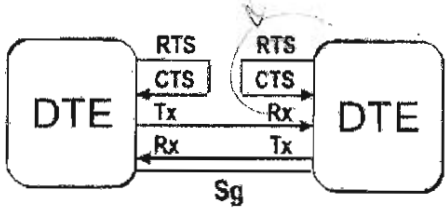


- Terminales principales:
  - Tx: para transmitir los datos
  - Rx: para recibir los datos
  - Sg: referencia común (masa)
  - RTS: el DTE indica al DCE que desea transmitir
  - CTS: el DCE indica al DTE que puede transmitir
  - También se definen otros terminales en el estándar RS-232C

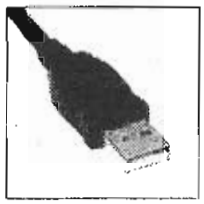
RTS y CTS son terminales para el protocolo



- Conexión directa de dos DTE (módem nulo)
  - Cable cruzado (Tx ↔ Rx)
  - Autoconexión RTS-CTS (si hay protocolo)

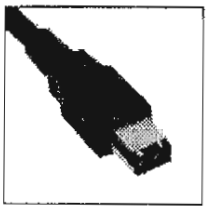


Estándares de comunicación serie síncrona



Conector para USB

- USB (*Universal Serial Bus)*
  - Comunicación síncrona maestro-esclavo
    - ♦USB 1.1: hasta 12 Mbps
    - ♦USB 2.0: hasta 480 Mbps
  - Hasta 5 metros de alcance
  - *Hot-plug* y *power-on-bus*
  - El MCF5272 incluye un módulo USB esclavo



Conector para Firewire

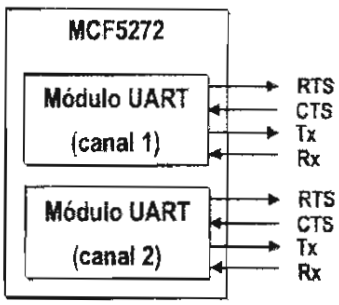
- Firewire (IEEE1394)
  - Hasta 4,5 m de alcance
  - *Hot-plug* y *power-on-bus*
  - Comunicación síncrona
    - ♦IEEE1394A: hasta 400 Mbps
    - ♦IEEE1394B: hasta 800Mbps

5.3.4 Módulos de comunicaciones serie del MCF5272

El módulo UART

En SEDG no se da más sobre la UART. En los ejercicios suele aparecer una versión simplificada y se explica en los enunciados cómo funciona (registros de configuración). Ver ejercicio 12

- 2 canales UART síncronos o asíncronos (RSC-232C)
- Permiten configuraciones en serie configurables por SW
- Puede trabajar por interrupciones o por sondeo (*polling*)
- Configurabilidad:
  - Velocidad y tipo de comunicación (duplex, semidúplex o símplex)
  - Formato de trama
  - Bits de datos (5-8)
  - Tipo de paridad: ninguna, par, impar o fija
  - Protocolo automático o manual
  - Detección de hardware de errores: desbordamiento, trama, paridad, falso comienzo y corte en la línea



El módulo USB

Recordar módulo de gestión del consumo del Tema 4

- Interfaz de comunicaciones host-periféricos
- Integrado en el MCF5272 (USB esclavo)
- Comunicaciones en serie síncronas
- Compatible con USB 1.1 (12 Mbps ó 1,5 Mbps)
- Capacidad para despertar al MCF5272 (dormido o parado)
- Permite implementar interfaces para periféricos de PC

El módulo Ethernet

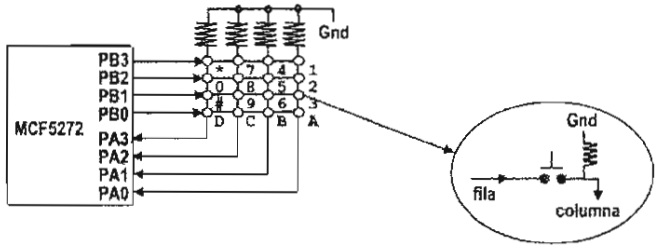
- Implementa la norma IEEE 802.3 CSMA/CD MAC
- Comunicación serie síncrona
- Semidúplex
- Transferencias por DMA con FIFO interna para compensar latencias del bus
- Interrupciones vectorizadas

El módulo QSPI

- Interfaz de comunicaciones maestro-esclavo
- Integrado en el MCF5272 (maestro)
- Comunicaciones en serie síncronas con reloj compartido
- Usa un protocolo bidireccional de Motorola
- Hay periféricos compatibles: DAC, ADC, etc.
- Por interrupciones o por sondeo
- Semidúplex

Ejemplo 5.1 Aplicación de control de un teclado matricial

El objetivo de este ejemplo es entender como funciona un teclado matricial haciendo uso de los puertos de E/S de propósito general del MCF5272. Se usa mucho en LSED



Se trata de un teclado matricial de 4x4 (16 pulsadores). Fíjate que basta usar 8 terminales para acceder a las 16 teclas. Para ello se realiza un barrido de la matriz hasta detectar una pulsación. El algoritmo que se usa en el ejemplo es el siguiente:

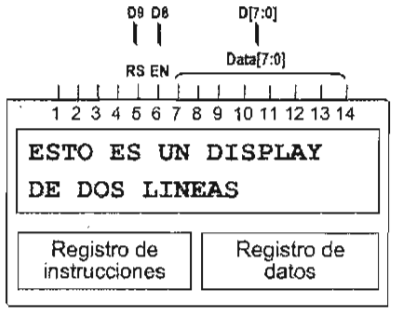
- Excitar un fila : poner un 1 en una línea de PB[3:0]
- Detectar si se ha pulsado alguna tecla en esa fila. Para ello se lee cada línea de PA[3:0]: pulsada (1) o no pulsada (0)
- El barrido finaliza al detectarse una pulsación (se ha leído un 1)

El código que configura los puertos A y B que implementa el algoritmo de barrido es el que sigue:

```
MBAR_MEM  EQU    $1000000
PACNT     EQU    MBAR+$80          * Puerto A: control
PADDR     EQU    MBAR+$84          * Puerto A: dirección
PADAT     EQU    MBAR+$86          * Puerto A: datos
PBCNT     EQU    MBAR+$88          * Puerto B: control
PBDDR     EQU    MBAR+$8C          * Puerto B: dirección
PBDAT     EQU    MBAR+$8E          * Puerto B: datos
          ...
EXCIT     DC.B   $08,$04,$02,$01   * Tabla de excitaciones
TECLAS    DC.B   "*7410852#963DCBA" * Tabla de caracteres
NO_TECL   DC.B   $FF                * Cuando no haya pulsación
          ...
INI_PORT  MOVE.L D0,-(A7)
          CLR.W  PADDR              * PA[15:0] entradas
          MOVE.W #$000F,D0          * PB[3:0] salidas y el resto entradas
          MOVE.W D0,PBDDR
          CLR.L  PACNT
          CLR.L  PBCNT
          CLR.W  PBDAT
          MOVE.L (A7)+,D0
          RTS
          ...
PPAL      BSR    INI_PORT
BUCLE     BSR    EXPLOTECL
          BRA    BUCLE
EXPLOTECL ADDA.L  #-12,A7
          MOVEM  D1-D2/A0,(A7)
          CLR.L  D0                 * D0 contendrá el número de la tecla pulsada
          MOVE.L #3,D1              * D1 es un contador de filas
          LEA    EXCIT,A0
FILA      MOVE.B (A0)+,PBDAT+1      * Envía la excitación
          MOVE.L #3,D2              * D2 es un contador decreciente de columnas
COLUMNA   BTST   D2,PADAT+1         * Comprueba el bit de la columna
          BNE    PULS               * Salta si encuentra tecla pulsada
          ADD    #1,D0              * Incrementa el contador de teclas recorridas
          SUB    #1,D2              * Decrementa
          BCC    COLUMNA            * Va a otra columna
          SUB    #1,D1              * Decrementa
          BCC    COLUMNA            * Otra fila
PULS      LEA    TECLAS,A0
          MOVE.B (A0,D0),D0         * Usa tabla
          MOVEM  (A7),D1-D2/A0
          ADDA   #12,A7
          RTS
```

Ejemplo 5.2. Aplicación de manejo de un LCD

En este ejemplo usaremos un LCD (visualizador de cristal líquido), que dispone de un controlador interno con un puerto paralelo de 8 ó 4 bits. Ese puerto tendrá una dirección dentro del mapa de memoria del MCF5272 configurable por medio de la señal de CS correspondiente.



Cada byte que se transfiere a ese puerto puede enviarse a:

- Registro de instrucciones: por ejemplo borrar pantalla
- Registro de datos: en este caso lo que se manda es un carácter

Los modos de trabajo del LCD son:

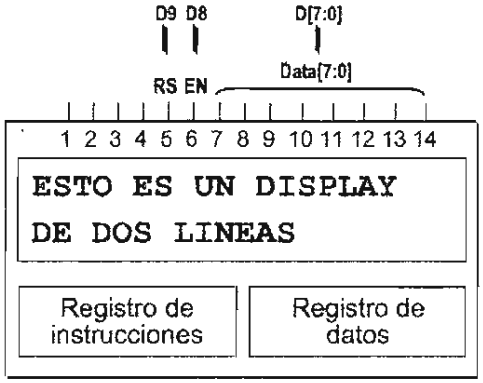
- Símplex: se espera un tiempo tras cada envío
- Semidúplex: el controlador indica si está preparado para recibir

Terminales:

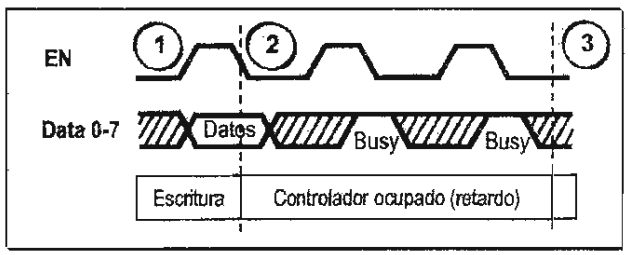
- **D[7:0]**: instrucción o dato
- **RS**: tipo de envío (0=instrucción, 1=dato)
- **EN**: habilitar (EN=1). Desde que se habilita los flancos de bajada validan el dato o la instrucción
- **Busy** (su uso es opcional): controlador ocupado, no está preparado para recibir nuevas instrucciones o datos

El protocolo que se va a implementar en el código siguiente es Símplex (por tanto sin usar Busy)

1. Envía el dato o la instrucción con flanco de subida en EN
2. Flanco de bajada en EN (valida el dato o instrucción)
3. Espera mientras el controlador está ocupado (llamando a una rutina que introduce un tiempo de espera o retardo)



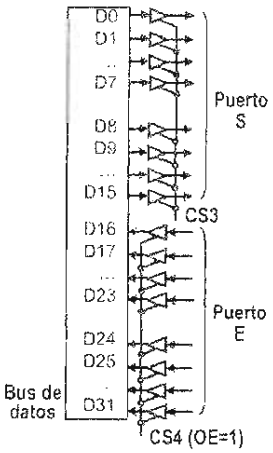

```
INIC:       MOVE.L   #$40000001,D0
            MOVE.L   D0,CSBR3
            MOVE.L   #$FFFFF078,D0
            MOVE.L   D0,CSOR3
            CLR.W    PUERTO_S       * Niveles iniciales a 0
            RTS
PPAL:       ...
            BSR      INIC
            ...
            MOVE.W   #$38,D0        * Instrucción: modo 8 bits 1 línea 5x7
            BSR      PULSO
            MOVE.W   #$06,D0        * Instrucción: cursor auto-incremental
                                    *              desplazamiento de pantalla
            BSR      PULSO
            MOVE.W   #$0F,D0        * Instrucción: activa pantalla, cursor parpadea
            BSR      PULSO
            MOVE.W   #$01,D0        * Instrucción: borrar pantalla (instrucción lenta)
            BSR      PULSO
            MOVE.B   #'A',D0
            BSR      ESCRIBIR       * Dato: escribe 'A' donde esté el cursor
PUERTO_S    EQU      $40000002              * Puerto unidireccional de 16 bits
BIT_ENABLE  EQU      $8                     * EN conectado a PUERTO_S[8]
BIT_RS      EQU      $9                     * RS conectado a PUERTO_S[9]
                                            * Dato[7:0] conectado a PUERTO_S[7:0]
PULSO:      BSET     #BIT_ENABLE,D0         * D0 auxiliar, puerto unidireccional
        (1) MOVE.W   D0,PUERTO_S
            BSR      RETARDO                * Modo símplex
            BCLR     #BIT_ENABLE,D0         * Flanco de bajada
        (2) MOVE.W   D0,PUERTO_S
            BSR      RETARDO
        (3) RTS

ESCRIBIR:   BSET     #BIT_RS,D0
            BSR      PULSO
            BCLR     #BIT_RS,D0
            MOVE.W   D0,PUERTO_S
            RTS
            ...
```



Ejemplo 5.3. Configuración de puertos distinta a la habitual !!

En general la configuración de puertos se realiza de la manera explicada en el tema 3 y dependiendo del tamaño del puerto se configuran los campos EBI y BW del CSBRn correspondiente al puerto, con el tamaño del puerto, y los datos irán en los bits del bus de datos que se indican en la figura de la página T3-6 ("Modos de transferencia con los puertos"). Sin embargo, a veces se encuentran configuraciones distintas como la de este ejemplo:

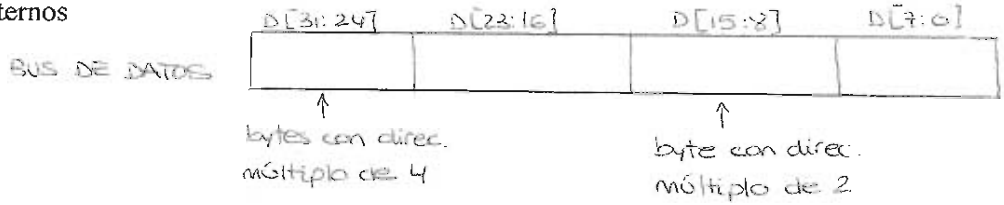
En este ejemplo se trata de configurar un puerto de salida (puerto S de la figura) y uno de entrada (puerto E de la figura), con las siguientes características:



- Puerto S de salida de 16 bits (sólo escritura):
  - Construido con 2 registros de salida de 8 bits
  - Conectados a CS3 y a D[15:0]
  - Con las direcciones $400000002 y $40000003

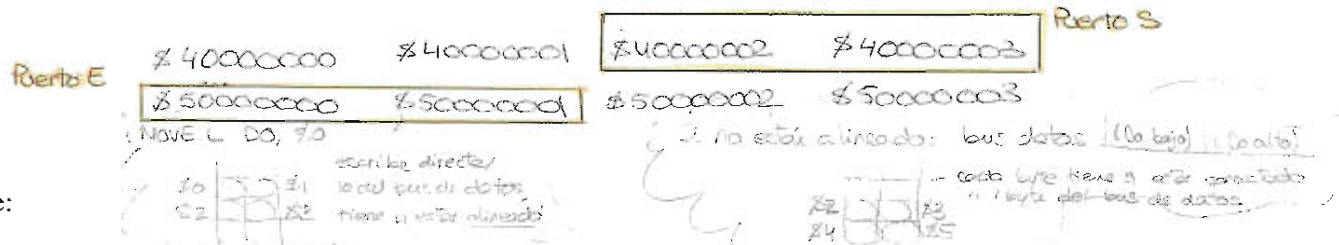
- Puerto E de entrada de 16 bits (sólo lectura):
  - Construido con dos búferes de entrada de 8 bits
  - Conectados a CS4 y a D[31:16]
  - Con las direcciones $50000000 y $500000001

Nota: correspondencia entre el bus de datos externo y las direcciones de memoria de cada byte de dispositivos externos



Ejemplo:

$0	$1	$2	$3
$4	$5	$6	$7
...			



Se pide:

a) Configurar los registros CSBRn y CSORn correspondientes teniendo en cuenta los datos del enunciado

b) Escriba un código como ejemplo de uso de los puertos E y S

bus de datos de 32 bits!

(a) • CS3 Puerto S ($4000 0002)

CSBR3 = $40000001

BA = $40000

EBI = %00 (SRAM o ROM de 32/16 bits) ! en este caso no coincide el nº de bits de EBI y EW. Por lo general, sí.

BW = %00 (long 32 bits) → OJO! Esta es la diferencia con el caso habitual.

SUPER = %0

ENABLE = %1

CSOR3 = $FFFFF003

BAM = $FFFFF

WS = %00000

RW = %1 (sólo escritura)

MRW = %1 (según indique RW)

• CS4 Puerto E ($5000 0000)

CSBR4 = $50000001

BA = $50000

EBI = %00 (SRAM o ROM de 16/32 bits) ! no coincide el nº bits entre EBI y BW

BW = %00 (long 32 bits)

SUPER = %0

ENABLE = %1

CSOR4 = $FFFFF01

BAM = $FFFFF

WS = %00000

RW = %0 (sólo lectura)

MRW = %1 (según indique RW)

(b)

```
PUERTO_S  EQU  $40000002
PUERTO_E  EQU  $50000000
          ...        ↳ pq son 16 bits
          MOVE(W) D0, PUERTO_S  } Siempre W
          MOVE(W) PUERTO_E, D0  }
```

Ejercicio 11. Reproductor MP3 con LCD

Un sistema de reproducción de MP3 emplea un visualizador LCD como interfaz de usuario. Se trata de un sistema basado en un MCF5272 a 66 MHz que incluye los bloques de la figura.

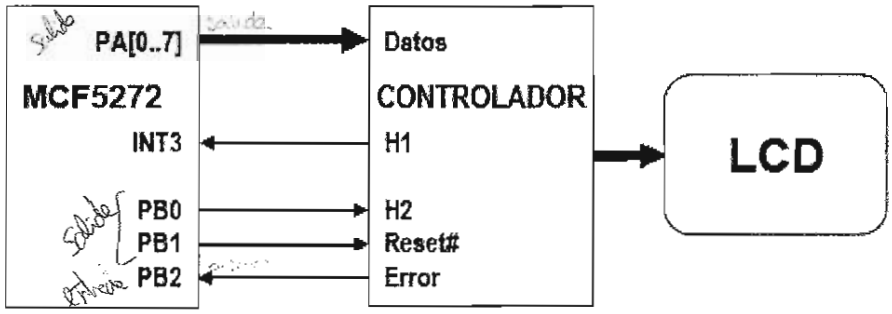
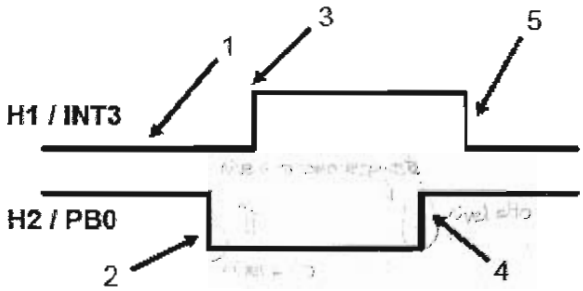


Figura 1. Diagrama de bloques del sistema de visualización

La comunicación entre el MCF5272 y el controlador del LCD se realiza a través de los puertos A y B del MCF5272 y la entrada de interrupciones INT3 como muestra la figura y se basa en la transferencia de caracteres ASCII desde el MCF5272 hacia el controlador. La comunicación implementa el protocolo asíncrono que se muestra en la figura (*interlocked handshaking*). Además, los bits 1 y 2 del puerto B se conectan a la entrada Reset (activa a nivel bajo) y a la salida Error del controlador (activa a nivel alto), respectivamente.



1 y 5. Controlador indica que está listo para recibir datos.
2. Tras la escritura en PADAT, ya hay datos para enviar.
3. Detectado que hay nuevos datos. Se empiezan a recibir.
4. Detectado que el controlador está recibiendo los datos.
5. Finaliza la recepción: controlador listo para más datos.

Figura 2. Protocolo asíncrono entre el MCF5272 y el controlador del LCD

Como se puede ver en la 0, inicialmente el controlador indica que está listo para recibir datos poniendo H1 a nivel bajo (punto 1). Tras producirse la escritura de un carácter en PADAT, el MCF5272 debe provocar un flanco de bajada en H2 para indicar al controlador que hay nuevos datos que este debe recibir (punto 2). Cuando el controlador ha detectado que hay nuevos datos y comienza el proceso de recepción, lo indica provocando un flanco de subida en H1 (punto 3). A su vez, el MCF5272 debe detectarlo y responderle con un flanco de subida en H2 (punto 4). Cuando el controlador ha terminado el proceso de recepción lo indica volviendo a la situación inicial (punto 5) y el MCF5272 puede enviar un nuevo carácter (si está disponible).

En el sistema bajo estudio, el envío de caracteres ASCII se controla mediante interrupciones de nivel 1 disparadas por flancos de bajada o de subida de INT3, conectado al terminal H1 del controlador del LCD, según muestra la 0.

Listado parcial del programa

```
*****************************************************************************
*       DEFINICIÓN DE CONSTANTES
*****************************************************************************
MBAR_MEM        EQU             $10000000           * Dirección base del SIM
MBAR_CFG        EQU             MBAR_MEM+$15        * Habilita sólo para datos
PACNT           EQU             $80                 * Puerto A: word de control
PADDR           EQU             $84                 * Puerto A: word de dirección
PADAT           EQU             $86                 * Puerto A: byte alto de datos
PADAT_L         EQU             $87                 * Puerto A: byte bajo de datos
PBCNT           EQU             $88                 * Puerto B: word de control
PBDDR           EQU             $8C                 * Puerto B: word de dirección
PBDAT           EQU             $8E                 * Puerto B: byte alto de datos
PBDAT_L         EQU             $8F                 * Puerto B: byte bajo de datos
PIVR            EQU             $3F
PITR            EQU             $34
ICR1            EQU             $20
ISR             EQU             $30

V_BASE          EQU             64
NULL            EQU             0
B_H2            EQU             ZZZZ
B_RST           EQU             ZZZZ+1
B_ERR           EQU             ZZZZ+2
ESPERA0         EQU             0
ESPERA1         EQU             1

                ORG             $0
                DC.L            $20000000+$PPPPPPPP
                DC.L            PRINCIPAL

                ORG             4*(V_BASE+$VVVVVVVV)
                DC.L            RUT_INT3

*****************************************************************************
*       DEFINICIÓN DE VARIABLES (RAM)
*****************************************************************************
                ORG             $20000000
EST_INT3        DS.B            1
DIR_ENVIO       DS.L            1
FIN_ENVIO       DS.B            1
MENSAJE         DS.B            100

*****************************************************************************
*       PROGRAMA PRINCIPAL
*****************************************************************************
                ORG             $400
PRINCIPAL:      MOVE.L          #MBAR_CFG,D0
                MOVEC           D0,MBAR
                …                                   * Inicializa los CS …
                BSR             SW_INI
                BSR             HW_INI
PPAL:           …
                LEA             MENSAJE,A1
                BSR             MENSAJE_MOSTRAR
                …
                BRA             PPAL

*****************************************************************************
*       ZONA DE SUBRUTINAS
*****************************************************************************

*** Rutinas de inicialización del sistema
SW_INI:         CLR.B           EST_INT3
                CLR.B           FIN_ENVIO
                RTS
```

```
HW_INI:         LEA.L       MBAR_MEM,A0
                CLR.L       PACNT(A0)           * Configuración del puerto A
                CLR.L       PBCNT(A0)           * Configuración del puerto B
                MOVE.W      #$XXXX,D7
                MOVE.W      D7,PADDR(A0)
                MOVE.W      #$YYYY,D7
                MOVE.W      D7,PBDDR(A0)        * Las no usadas, como entradas
                BSET.B      #B_H2,PBDAT_L(A0)
                BCLR.B      #B_RST,PBDAT_L(A0)  * Reset del LCD
                BSR         RET_2S
                BSET.B      #B_RST,PBDAT_L(A0)
                MOVEQ.L     #V_BASE,D7
                MOVE.B      D7,PIVR(A0)         * Inicialización de PIVR
                MOVE.W      #$2700,SR
                RTS

*** Rutinas de habilitación e inhabilitación de interrupciones
INT3_HABILITAR:
                MOVE.L      D7,-(A7)
                MOVE.L      #$TTTTTTTT,D7
                MOVE.L      D7,PITR+MBAR_MEM
                MOVE.W      #$0090,D7
                MOVE.W      D7,ICR1+MBAR_MEM    * Habilitar interrupciones
                MOVE.W      SR,D7
                ANDI        #$IIIIIIII,D7
                MOVE.W      D7,SR
                MOVE.L      (A7)+,D7
                RTS

INT3_INHABILITAR:
                MOVE.L      D7,-(A7)
                MOVE.W      #$0080,D7
                MOVE.W      D7,ICR1+MBAR_MEM
                MOVE.L      (A7)+,D7
                RTS

*** Rutina que habilita el envío al visualizador de la cadena de caracteres
***    A1: parámetro de entrada; contiene la dirección inicial de la
***        cadena de caracteres,
***    D0: parámetro de salida; contiene el estado del controlador
MENSAJE_MOSTRAR:
                MOVE.L      D7,-(A7)
                MOVE.L      A0,-(A7)
                LEA.L       MBAR_MEM,A0
                CLR.L       D0                  * Estado ok inicialmente
                BTST.B      #B_ERR,PBDAT_L(A0)  * Verifica línea de error
                BNE         ERROR               * Si ERR=1, no operativo
                BTST.B      #5,ISR(A0)
                BNE         ERROR
                CLR.B       FIN_ENVIO
                CLR.L       D7
                MOVE.B      (A1)+,D7
                MOVE.B      D7,PADAT_L(A0)
                CMP.L       #NULL,D7
                BEQ.S       NO_ERROR            * Cadena vacía
                MOVE.L      A1,DIR_ENVIO
                MOVE.L      #ESPERA0,D7
                MOVE.L      D7,EST_INT3
                BSR         INT3_HABILITAR
                BCLR.B      #B_H2,PBDAT_L(A0)
AL_DISP:        TST.B       FIN_ENVIO
                BEQ.S       AL_DISP
                BRA.S       NO_ERROR            * Estado ok
ERROR:          MOVE.L      #$FF,D0             * Indica estado de error en D0
NO_ERROR:       BSR         SW_INI
                MOVE.L      (A7)+,A0
                MOVE.L      (A7)+,D7
                RTS
```

$10000015 = MBAR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1

← chuleta baja 1

Listado parcial del programa

BA = $1000 porque la dirección base del SIM es $10000000

```
*****************************************************************************
*       DEFINICIÓN DE CONSTANTES
*****************************************************************************
MBAR_MEM        EQU             $10000000       * Dirección base del SIM = dirección inicial del módulo SIM
MBAR_CFG        EQU             MBAR_MEM+$15    * Habilita sólo para datos
PACNT           EQU             $80             * Puerto A: word de control
PADDR           EQU             $84             * Puerto A: word de dirección
PADAT           EQU             $86             * Puerto A: byte alto de datos
PADAT_L         EQU             $87             * Puerto A: byte bajo de datos
PBCNT           EQU             $88             * Puerto B: word de control
PBDDR           EQU             $8C             * Puerto B: word de dirección
PBDAT           EQU             $8E             * Puerto B: byte alto de datos
PBDAT_L         EQU             $8F             * Puerto B: byte bajo de datos
PIVR            EQU             $3F
PITR            EQU             $34
ICR1            EQU             $20
ISR             EQU             $30

V_BASE          EQU             64
NULL            EQU             0
B_H2            EQU             ZZZZ
B_RST           EQU             ZZZZ+1
B_ERR           EQU             ZZZZ+2
ESPERA0         EQU             0
ESPERA1         EQU             1

                ORG             $0
                DC.L            $20000000+$PPPPPPPP
                DC.L            PRINCIPAL
                ORG             4*(V_BASE+$VVVVVVVV)
                DC.L            RUT_INT3

*****************************************************************************
*       DEFINICIÓN DE VARIABLES
*****************************************************************************
                ORG             $20000000
EST_INT3        DS.B            1
DIR_ENVIO       DS.L            1
FIN_ENVIO       DS.B            1
MENSAJE         DS.B            100

*****************************************************************************
*       PROGRAMA PRINCIPAL
*****************************************************************************
                ORG             $400
PRINCIPAL:      MOVE.L          #MBAR_CFG,D0
                MOVEC           D0,MBAR
                ...
                BSR             SW_INI          * Inicializa los CS ...
                BSR             HW_INI
PPAL:           ...
                LEA             MENSAJE,A1
                BSR             MENSAJE_MOSTRAR
                ...
                BRA             PPAL

*****************************************************************************
*       ZONA DE SUBRUTINAS
*****************************************************************************

*** Rutinas de inicialización del sistema
SW_INI:         CLR.B           EST_INT3
                CLR.B           FIN_ENVIO
                RTS
```

valor que hay que cargar en el registro MBAR

valor que se carga en el MBAR

configura el puerto A

configura el puerto B

configuración de puertos de interrupciones

define por separado el byte alto y el byte bajo de los puertos de datos

ejemplo de uso en la cara de atrás

← sólo ha definido el offset con respecto a

PILA

offset que tiene el vector de interrupción de INT3 con respecto a la dirección base

dirección del vector de interrupción correspondiente a INT3 en memoria

vale 3 mirando en la chuleta porque el offset que le corresponde a INT3 es 3 con respecto a la dirección base

RAM

← si no nos dan otra pista, la RAM comienza aquí

aquí se ve que MBAR_CFG es lo que tenemos que cargar en MBAR, en este programa falta la definición de MBAR, pero debería estar

cargar la dirección donde está ese mensaje en A1 y

salta a mensaje_mostrar

① DIRECCIONES ABSOLUTAS ①

```
MBAR_MEM    EQU    $1000 0000
PACNT       EQU    MBAR_MEM + $80
```

• Ejemplo de uso:

```
MOVE.W   # $00, D0
MOVE.W   D0, PACNT
```

• Dirección efectiva = $1000 0000 + $80 = $1000 0080

⇕ equivalentes

② DIRECCIONES RELATIVAS : ventaja: ahorra instrucciones

```
MBAR_MEM    EQU    $1000 0000
PACNT       EQU    $80
```

• Ejemplo de uso:

```
LEA    MBAR_MEM, A0
MOVE.W  # $00 , PACNT(A0)
```

• Dirección efectiva = (A0) + PACNT = $1000 0000 + $80 = $10000080

La ventaja no se aprecia porque sólo hay un puerto definido pero:

demostración de la ventaja:

ahora definimos

① `PADDR EQU MBAR_MEM+$84` ⟷ ② `PADDR EQU $84`

```
MOVE.W   # $0F, D0          ⟷          MOVE.W  # $0F , PADDR(A0)
MOVE.W   D0 , PADDR
```

debajo de las instrucciones de los ejemplos de uso anteriores

```
HW_INI:     LEA.L     MBAR_MEM,A0          mete la dirección base en A0 y a partir de ahí
            CLR.L     PACNT(A0)          * Configuración del puerto A
            CLR.L     PBCNT(A0)          * Configuración del puerto B
            MOVE.W    #$XXXX,D7          PUERTA A
            MOVE.W    D7,PADDR(A0)       PUERTA B
            MOVE.W    #$YYYY,D7          valores que hay que poner en los registros para que PA Co:7] y PB0 y PB1 sean de salida y PB2 de entrada (dibujo)
            MOVE.W    D7,PBDDR(A0)       * Las no usadas, como entradas
            BSET.B    #B_H2,PBDAT_L(A0)
            BCLR.B    #B_RST,PBDAT_L(A0) * Reset del LCD
            BSR       RET_25
            BSET.B    #B_RST,PBDAT_L(A0)
            MOVEQ.L   #V_BASE,D7
            MOVE.B    D7,PIVR(A0)        * Inicialización de PIVR
            MOVE.W    #$2700,SR          carga en el registro de estado $2700
            RTS                          2≡ modo supervisor
                                         7≡ máscara 111 = 0 inhabilita todas las interrup. vectoriz excepto las de nivel 7

*** Rutinas de habilitación e inhabilitación de interrupciones
INT3_HABILITAR:
            MOVE.L    D7,-(A7)           guarda el registro D7 en la pila porque lo va a utilizar
            MOVE.L    #$TTTTTTTT,D7      carga $TTT...T en D7
            MOVE.L    D7,PITR+MBAR_MEM   guarda $T...T en MBAR_MEM + PITR
            MOVE.W    #$0090,D7
            MOVE.W    D7,ICR1+MBAR_MEM   * Habilitar interrupciones
            MOVE.W    SR,D7              carga el registro de estado en D7
            ANDI      #$IIIIIIII,D7      hace un AND con $I.... I llevará
            MOVE.W    D7,SR              ceros en los bits correspondientes a la máscara
            MOVE.L    (A7)+,D7           restaura el valor de D7 después de usarlo
            RTS

INT3_INHABILITAR:
            MOVE.L    D7,-(A7)
            MOVE.W    #$0080,D7          carga el valor $0080 en D7
            MOVE.W    D7,ICR1+MBAR_MEM   pone $0080 en ICR1 + MBAR_MEM
            MOVE.L    (A7)+,D7
            RTS

*** Rutina que habilita el envío al visualizador de la cadena de caracteres.
***   A1: parámetro de entrada; contiene la dirección inicial de la
***       cadena de caracteres,
***   D0: parámetro de salida; contiene el estado del controlador
MENSAJE_MOSTRAR:  implementa el comienzo del protocolo
            MOVE.L    D7,-(A7)
            MOVE.L    A0,-(A7)
            LEA.L     MBAR_MEM,A0          carga en A0 el valor de MBAR_MEM
            CLR.L     D0                 * Estado ok inicialmente
            BTST.B    #B_ERR,PBDAT_L(A0) * Verifica línea de error
            BNE       ERROR              * Si ERR=1, no operativo
            BTST.B    #5,ISR(A0)
            BEQ       ERROR
            CLR.B     FIN_ENVIO          pone a cero FIN_ENVIO
            CLR.L     D7
            MOVE.B    (A1)+,D7           lleva el primer byte de la zona apuntado por A1 a D7
            MOVE.B    D7,PADAT_L(A0)     A1 apunta a la zona de memoria donde está el mensaje que tiene (A1) → "D" "A" | "V" "I" | "D" " " | "B" ... | ... indica el final | ... NULL
            CMP.L     #NULL,D7           * Cadena vacía
            BEQ.S     NO_ERROR           que salir por pantalla
            MOVE.L    A1,DIR_ENVIO
            MOVE.L    #ESPERA0,D7        (DIR_ENVIO) = contiene la dirección
            MOVE.L    D7,EST_INT3        del siguiente caracter
            BSR       INT3_HABILITAR     habilita INT 3 → para poder detectar los avisos del controlador
            BCLR.B    #B_H2,PBDAT_L(A0)  pone a cero el bit de H2 para validar el cambio
AL_DISP:    TST.B     FIN_ENVIO          * Estado ok
            BEQ.S     AL_DISP            * Indica estado de error en D0
            BRA.S     NO_ERROR           mientras esté a 0 es que no ha acabado de
ERROR:      MOVE.L    #$FF,D0            escribir todos los caracteres
NO_ERROR:   BSR       SW_INI
            MOVE.L    (A7)+,A0
            MOVE.L    (A7)+,D7
            RTS
```

ESPERA0 = 0

configura los puertos A y B como puertos de entrada/salida

utiliza siempre direccionamiento indirecto con offset (lo hemos explicado en la cara de atrás de la hoja 2)

vemos en la chuleta lo que era PITR → sirve para que sea sensible a flancos de subida o flancos de bajada

```
*** Rutina que produce un retardo de 2 segundos
RET_2S:         ...
                RTS

*****************************************************************************
*       ZONA DE RUTINAS DE ATENCION A INTERRUPCIONES
*****************************************************************************

*** Rutina de Interrupción de INT3. Lee caracteres del buffer BUF_ENVIO y los
*** envía por el puerto A. Implementa el protocolo de H1-H2.
RUT_INT3:       ADD.L           #-12,A7
                MOVEM.L         D7/A0-A1,(A7)
                LEA.L           MBAR_MEM,A0
                MOVEA.L         DIR_ENVIO,A1
                MOVE.L          ICR1(A0),D7         * Pone a cero el bit de INT3PI
                ORI.L           #$00800000,D7
                MOVE.L          D7,ICR1(A0)         * Habilita futuras interrupciones
                TST.B           EST_INT3
                BNE             ESTADO0
ESTADO1:        MOVE.B          #ESPERA1,D7
                MOVE.B          D7,EST_INT3
                BSET.B          #B_H2,PBDAT_L(A0)
                MOVE.L          #$LLLLLLLL,D7
                MOVE.L          D7,PITR(A0)
                BRA             FIN_INT3
ESTADO0:        MOVE.B          #ESPERA0,D7
                MOVE.B          D7,EST_INT3
                CLR.L           D7
ENVIAR:         MOVE.B          (A1)+,D7
                CMP.L           #NULL,D7
                BEQ.S           ACABAR
                MOVE.B          D7,PADAT_L
                BCLR.B          #B_H2,PBDAT_L(A0)
                MOVE.L          #$TTTTTTTT,D7
                MOVE.L          D7,PITR(A0)
                BRA             FIN_INT3
ACABAR:         MOVE.B          #$FF,D7
                MOVE.B          D7,FIN_ENVIO
SALIR:          BSR             INT3_INHABILITAR
FIN_INT3:       MOVE.L          A1,DIR_ENVIO
                MOVEM.L         (A7),D7/A0-A1
                ADD.L           #12,A7
                RTE
```

guardo en la pila los registros que voy a utilizar y luego los restauro

→ carga en A1 la dirección del siguiente caracter a enviar

→ escribe un 1 en el bit PI de la causa de interrupción 3, al escribir un 1 pone PI a cero

→ pone a 1 el bit H2

→ pone un determinado valor en D7 y lo pone en PITR como queremos que sea sensible a flanco de bajada ⇒ se pone el bit correspondiente a cero ⇒ $LLLLLLLL = $0

salta a FIN_INT3

→ guarda un cero en EST_INT3

envía el contenido de A1 a PADAT_L y aumentamos 1 el valor de A1

ponemos a cero H2

mete $TTT...T en PITR que tiene que ser sensible a flanco de subida

vuelve a guardar la dirección del siguiente caracter en DIR_ENVIO

termina la rutina y vuelve a donde se la llama

1. Mapa de memoria

1.1. El sistema dispone de diversas memorias FLASH externas de 32 bits para almacenar el programa y la música MP3. Para seleccionarlas se emplean todos los CS del MCF5272, a razón de un CS por cada memoria. Justifique sus respuestas a las siguientes preguntas:

- **¿Cuál es el tamaño máximo de cada memoria FLASH?**

  8 Mbytes pq en el bus externo sólo disponemos de 23 terminales del bus de dirección A[22:0] que permiten direccionar dispositivos de $2^{23} = 2^3 \cdot 2^{20} = 8MB$

- **¿Cuál es la cantidad total de memoria no volátil disponible?**

  - 8 CS
    8 MB/CS
    } 64 MB en total

  - si se empleara memorias dinámicas, serán 7 CS pq no se utiliza xa Mem. Dinámica (DRAM) ⇒ 7 CS x 8 MB/CS = 56 MB (CS7)

1.2. Configure justificadamente RAMBAR para que el sistema emplee la RAM interna del MCF5272 para las variables del programa. ⇒ SC=UC=1

$RAMBAR = $ 20000035

BA = $20000. Dirección base de la RAM: $20000000 (será código)

WP = 0 Sin protección contra escritura (vamos a usarla para variables)

C/I = 1

SC=UC=1 Máscara que inhabilita los accesos a código de usuario y supervisor

SD=UD=0 (P3 es para datos)

V = 1 Se valida el uso de la RAM interna

SC = código supervisor
SD = datos supervisor
UC = código usuario
UD = datos usuario

4kB = 2^{12}

1.3. Justifique cuál sería el valor máximo posible de la constante $PPPPPPPP.

$PPPPPPPP = $ 1000

4kB = $2^2 \cdot 2^{10}$

Disponemos de 4kB de RAM interna. Para que la pila pueda tener tamaño máximo hay que inicializar A7 como la última dirección de la RAM+1 (para que el primer byte escrito en la pila, se escriba en la última dirección de la RAM)

2. Los puertos y las interrupciones

2.1. ¿Cuáles son los valores correctos de $XXXX e $YYYY en la rutina HW_INI? Justifique sus respuestas.

entrada bit = 0
salida bit = 1

$XXXX = $ 00FF

Puerto A: se configuran los 8 bits menos significativos PA[7:0] como salida. El resto no se usan, así que entrada (ver código) ⇒ PADDR = $00FF (ent sal)

$YYYY = $ 0003

Puerto B: se configuran como salidas PB0 y PB1 (H2 y Reset)

El resto como entradas ⇒ PBDDR = $0003

2.2. ¿Cuál debería ser el valor de las constantes $VVVVVVVV y $IIIIIII?

$VVVVVVVV = $ 3

V_BASE = 64 es el valor que se carga en el PIVR (número del primer vector de las interrupciones de usuario). En la chuleta se ve que INT3 tiene un offset de 3 respecto a PIVR ⇒ El vector de la interrupción INT3 será el número (VBASE + 3). La dirección de memoria donde se almacena es VBR+4(PIVR+3)

0 por defecto $wwwww

$IIIIIII = $ F8FF

Se aplica esta máscara con un AND para habilitar interrupciones de nivel 1. En el resultado tendrá que estar I2=I1=I0=0. El resto de bits de SR no se modifican

N > 000 → acepto int nivel 1.

2.3. Asigne el valor correcto a la constante $ZZZZ y justifique su respuesta con referencia al código, al texto del enunciado o a las figuras.

$ZZZZ = $ 0

B_H2, B_RST y B_ERR se utilizan con operación BSET, BCLR y BCHG por lo que representan el número de bit dentro del byte PB[7:0] a modificar

A H2 le corresponde B0 ⇒ B_H2 = $ZZZZ = $0 y así vemos que es correcto porque B_RST es B1 = 0+1 y B_ERR es B2 = 0+2 en el código

3. Análisis del software

3.1. Diga los modos de direccionamiento empleados en las siguientes instrucciones del programa, y calcule su dirección efectiva (si existe).

```
MOVE.B      D7,PITR+MBAR_MEM    * En rutina INT3_HABILITAR
```

- **Modos de direccionamiento:**
  f: Directo a registro D7
  d: Directo a memoria absoluto : PITR + MBAR_MEM (offset que tiene PITR con respecto a MBAR_MEM)
- **Direcciones efectivas:**
  f: no tiene sentido hablar de la dirección efectiva de D7
  d: $34 + $1000 0000 = $1000 0034

```
CLR.L       PACNT(A0)           * En rutina HW_INI
```

- **Modo de direccionamiento:**
  Indirecto con desplazamiento
- **Dirección efectiva:**
  (A0) + PACNT = MBAR_MEM + PACNT = $1000 0000 + $80 = $1000 0080 (offset que tiene PACNT con respecto a MBAR_MEM)
  porque previamente se había hecho LEA.L MBAR_MEM, A0

3.2. Explique para qué sirve la variable FIN_ENVIO.

Permite que la RUT_INT3 informe a la rutina MENSAJE_MOSTRAR (llamada desde el programa principal) que se ha terminado el envío de datos al controlador de LCD, y la rutina y el programa principal puedan continuar. Si el mensaje está vacío, no se envía y por tanto no se usa FIN_ENVIO

si FIN_ENVIO = $0 => no le acabado lo que hay que sacar por pantalla / FIN_ENVIO = $FF => se acabó

3.3. Justifique por qué el programa envía datos al puerto PA en 2 rutinas, MENSAJE_MOSTRAR e RUT_INT3.

El Coldfire comienza la transmisión con un primer carácter en MENSAJE_MOSTRAR, después de eso avisa al controlador (H2=0) y habilita las interrupc. de INT3. El resto de la transmisión (los caracteres restantes) se harán mediante RUT_INT3

3.4. Explique por qué se modifica varias veces el PITR y cuáles deberían ser los valores de las constantes $LLLLLLLL y $TTTTTTTT.

$LLLLLLLL = $0000 0000 INT3 sensible a flancos de bajada

$TTTTTTTT = $2000 0000 INT3 sensible a flancos de subida

Cada vez que se envía un byte al controlador hay que esperar un flanco de subida de INT3/H1. Una vez recibido y respondido a través de PB0/H2, hay que esperar a un flanco de bajada en INT3/H1 para saber que el controlador está listo para recibir de nuevo.

En realidad coge el bit 29, que al ser B corresponde al bit 5 de ese byte.

3.5. En la rutina MENSAJE_MOSTRAR se incluyen las siguientes líneas de código. Explique por qué.

```
                -
                BTST.B      #5,ISR(A0)
                BEQ         ERROR
                -
```

El protocolo exige que cuando se comienza una transmisión HI debe estar a nivel bajo (así que INT3 debe estar a nivel bajo). Para comprobarlo se mira el bit 5 del ISR que informa sobre el valor instantáneo de la línea INT3 (0 si nivel alto, 1 si nivel bajo). Si el 5 del ISR es un 0 ⇒ la línea INT3 tiene un nivel alto ⇒ ERROR (salta a subrutina ERROR)

3.6. Cada mensaje termina con un carácter nulo. Justifique si se envía o no este carácter nulo.

No se envía. Hay 2 casos:

- Si es el primer caracter de la cadena, entonces lo detecta MENSAJE_MOSTRAR y aunque se pone su código ASCII en PA[7:0], no se avisa al controlador poniendo H2 a 0 ⇒ No se envía y se salta a NO_ERROR
- Si no es el primer carácter: se detecta en RUT_INT3 (cerca de la etiqueta ENVIAR) y salta a la etiqueta ACABAR. En este caso ni siquiera se pone su código ASCII en PA[7:0]

Ejercicio 12. Sistema de adquisición remoto

Un sistema de adquisición recoge datos de múltiples sensores, los procesa, los codifica en tramas de 48 bytes y los envía por una línea serie a razón de 4 tramas/seg. La línea serie soporta una comunicación asíncrona a 19200 baudios. Cada byte se envía precedido por un bit de arranque y seguido por un bit de paridad impar y un bit de parada. El protocolo asíncrono impone un tiempo de guarda entre bytes de al menos dos tiempos de bit, 2 t_{bit}. Asimismo, el sistema de adquisición asegura un tiempo de guarda entre tramas de bytes de al menos 50 ms.

Para poder conectar este sistema de adquisición a un PC portátil sin puerto serie asíncrono, se ha diseñado un sistema que sirva de interfaz entre el PC y el sistema de adquisición. Este sistema interfaz recibe las tramas de datos, las descodifica, y las transforma en información que envía a través de una conexión USB. Se trata de un sistema basado en un MCF5272 a 66 MHz que incluye los bloques de la 0. La CPU transforma la información contenida en cada trama en un conjunto de cadenas de caracteres ASCII que se envían al PC.

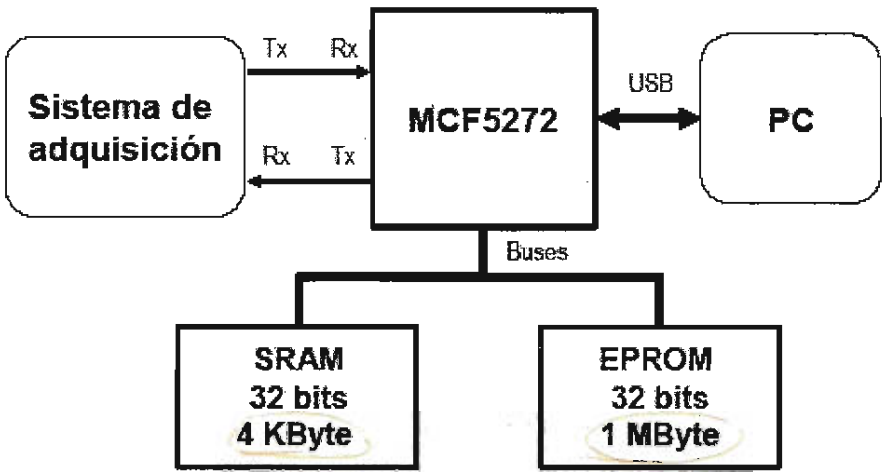


Figura 1. Diagrama de bloques del sistema completo

La CPU procesa cada trama una vez que se han recibido sus 48 bytes. Los primeros 4 bytes de cada trama describen cierta magnitud BETA tal como se indica en la Figura 2. El resto de los bytes de la trama no se analizan en el presente ejercicio, en el que además se supone que el procesamiento de una trama siempre ha finalizado cuando llega la siguiente trama. Cada byte de datos contiene 2 medios bytes (*nibbles*). Cada *nibble* contiene un dígito entre 0 y 9 (hay combinaciones de bits que son imposibles)

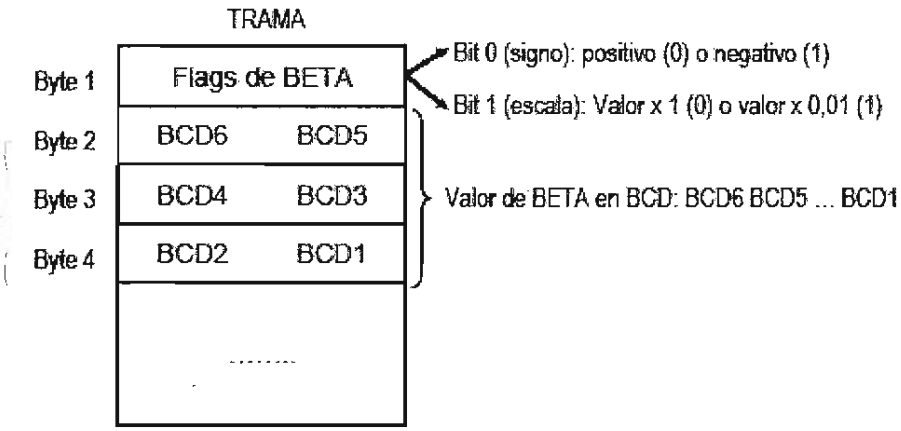
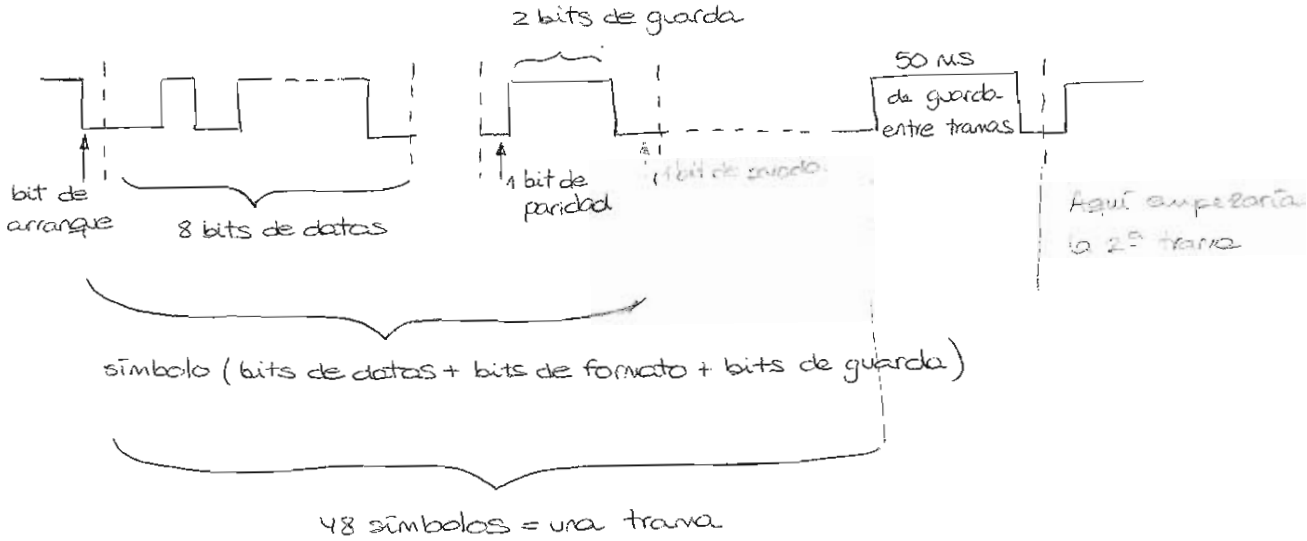


Figura 2. Bytes que describen la magnitud BETA


2 bits de guarda
50 ms
de guarda
entre tramas
bit de
arranque
8 bits de datos
1 bit de
paridad
Aquí empezaría
la 2ª trama
símbolo (bits de datos + bits de formato + bits de guarda)
48 símbolos = una trama

Los bytes de BETA se convierten en una cadena de códigos ASCII que describen el valor de BETA usando caracteres alfanuméricos. El código ASCII de cada dígito *d* se obtiene como el valor ($30+*d*). La cadena de códigos ASCII se envía al PC precedida por una cabecera y seguida del carácter NULL (código $00) que indica el final de la cadena. La estructura de la cabecera no es objeto de este ejercicio.

Interfaz serie

La UART del MCF5272 tiene la misión de enviar la información de configuración al sistema de adquisición y recibir los datos enviados por el mismo. En este ejercicio supondremos que el MCF5272 dispone de una UART1 simplificada en lugar de la estándar, que emplea el mismo vector de interrupción que la estándar pero que contiene los siguientes registros:

	7	6	5	4	3	2	1	0
XCR	-	BR		DT	ST	PR	TX	RX

- **XCR**: Registro de control

  BR: velocidad en baudios (00≡1200, 01≡4800, 10≡9600, 11≡19200)

  DT: número de bits de datos (0=7 bits; 1=8 bits)

  ST: número de bits de parada (0=1 bit: 1=2 bits)

  PR: tipo de paridad (0=par; 1=impar)

  TX: interrupción de Tx habilitada (1) o inhabilitada (0)

  RX: interrupción de Rx habilitada (1) o inhabilitada (0)

	7	6	5	4	3	2	1	0
XSR	-	-	TE	OV	FR	PR	TX	RX

- **XSR**: Registro de estado

  TE: está preparado para transmitir (TE=1)

  OV: error de desbordamiento (OV=1); al escribir un 1, se pone a cero el flag

  FR: error de trama recibido (FR=1); al escribir un 1, se pone a cero el flag

  PR: error de paridad recibido (PR=1); al escribir un 1, se pone a cero el flag

  TX: dato transmitido (TX=1); al escribir un 1, se pone a cero el flag

  RX: dato recibido (RX=1); al escribir un 1, se pone a cero el flag

	7	0
XDAT		

- **XDAT**: Registro de datos

  Cuando se escribe, se escribe en el registro interno de transmisión. Cuando se lee, se lee el registro interno de recepción

En el sistema bajo estudio, la UART1 debe generar interrupciones de nivel 2 asociadas a la recepción de bytes por la línea serie.

Listado parcial del programa

```
*************************************************************************
*       DEFINICIÓN DE CONSTANTES
*************************************************************************
MBAR_MEM        EQU             $10000000
MBAR_CFG        EQU             MBAR_MEM+$1S

PIVR            EQU             MBAR_MEM+$3F
PITR            EQU             MBAR_MEM+$34
ICR2            EQU             MBAR_MEM+$24
ICR1            EQU             MBAR_MEM+$20
XCR             EQU             MBAR_MEM+$1000
XDAT            EQU             MBAR_MEM+$1001
XSR             EQU             MBAR_MEM+$1002

LONG_TRAMA      EQU             48
MAX_CADENA      EQU             128
NULL            EQU             $00             * Código ASCII de carácter NULL
ESPACIO         EQU             $20             * Código ASCII del espacio en blanco
COMA            EQU             $2C             * Código ASCII del carácter ','
GUION           EQU             $2D             * Código ASCII del carácter '-'

                ORG             $0
                DC.L            $LLLLLLLL
                DC.L            PRINCIPAL
                DC.L            RUT_EXC
                DC.L            RUT_EXC

                ORG             $XXXXXXXX
                DC.L            INT_UART1
*************************************************************************
*       DEFINICIÓN DE VARIABLES
*************************************************************************
                ORG             $MMMMMMMM
CNT_TRAMA       DS.L            1
DIR_ENVIO       DS.L            1
FIN_TRAMA       DS.B            1
BUF_SERIE       DS.B            LONG_TRAMA
BUF_SALIDA      DS.B            MAX_CADENA
*************************************************************************
*       PROGRAMA PRINCIPAL
*************************************************************************
                ORG             $NNNNNNNN
PRINCIPAL       MOVE.L          #MBAR_CFG,D0
                MOVEC           D0,MBAR
                …                               * Inicialización de los CS …
                BSR             SW_INI
                BSR             HW_INI

PPAL            TST.B           FIN_TRAMA
                BEQ             PPAL
                BSR             PROC_TRAMA
                BSR             SW_INI
                BRA             PPAL

*************************************************************************
*       ZONA DE SUBRUTINAS
*************************************************************************

*** Rutinas de inicialización del sistema

*** Inicialización del software
SW_INI          CLR.L           CNT_TRAMA
                CLR.B           FIN_TRAMA
                RTS
```

```
*** Rutina de procesamiento de trama
PROC_TRAMA    BSR          PROC_BETA
              BSR          USB_TX
              …                                  * Llamadas a otras rutinas
              RTS

*** Rutina que habilita el envio al PC de la cadena de caracteres almacenada
*** en BUF_SALIDA
USB_TX        …
              RTS

*** Otras rutinas llamadas desde PROC_TRAMA (no incluidas)
              …

*****************************************************************************
*      ZONA DE RUTINAS DE ATENCION A INTERRUPCIONES
*****************************************************************************
*** Rutina de excepción
RUT_EXC       …                                  * Activación de una alarma
ESPERA        BRA          ESPERA
              RTE

*** Rutina de interrupción del puerto serie
INT_UART1     ADD.L        #-12,A7
              MOVEM.L      D0-D1/A0,(A7)         * Salvar el contexto
              CLR.L        D0
              BSET.B       D0,XSR
              MOVE.L       CNT_TRAMA,D0          * Cargar valor del contador
              ADDQ.L       #1,CNT_TRAMA          * Incrementar contador de bytes
              MOVE.L       CNT_TRAMA,D1
              CMP.L        #LONG_TRAMA,D1        * Verificar final de trama
              BNE          SEGUIR                * Si no lo es, seguir
              CLR.L        CNT_TRAMA             * Inicializar contador
              MOVE.B       #$FF,D1
              MOVE.B       D1,FIN_TRAMA          * Indicar fin de trama
SEGUIR        LEA          BUF_SERIE,A0          * Cargar dirección del búfer
              MOVE.B       XDAT,D1
              MOVE.B       D1,0(A0,D0)           * Cargar dato en el búfer
              MOVEM.L      (A7),D0-D1/A0         * Recuperar el contexto
              ADD.L        #12,A7
              RTE

MENSAJE_INIC  DC.B         "F:100 N:16",0        * Mensaje inicial
```

1. Mapa de memoria

1.1. Complete la siguiente tabla con el mapa de memoria que asigne las direcciones más bajas posibles a los distintos dispositivos.

DIRECCIONES (HEX)	DISPOSITIVO ASIGNADO
$00000000 – $000FFFFF	Memoria EPROM
$00100000 – $00100FFF	Memoria RAM
$10000000 – $1000FFFF	SIM y módulos internos

1.2. Configure justificadamente los registros CSxxx para poder usar CS0 y CS1 para la activación de la EPROM y la RAM (este código estaría situado en los puntos suspensivos que acompañan a la etiqueta PRINCIPAL). Considere que las memorias necesitan el número máximo de estados de espera.

EPROM

CSBR0 = $ 0000 0001
BA = $000 00. Dirección base de EPROM: $0000 0000
EBI = BW = 00 Memoria ROM de 32 bits (SIEMPRE tamaño del puerto) Lo normal es q EBI = BW
SUPER = 0 Sin máscara para supervisor
ENABLE = 1. Se habilita el uso de CS0#

CSOR0 = $FFF0 0079
BAM = $FFF00 1 MB = 2^{20} Se comparan los 32 − 20 = 12 bits superiores
WS = $1E = %1 1110. Máximo número de estados de espera (Se mira en la tabla, ¡sólo son 5 bits)
RW = 0 (sólo lectura) en la ROM
MRW = 1 (lo que indique RW)

RAM

CSBR1 = $ 0010 0001
BA = $00100. Dirección base de la RAM: $00100000
EBI = BW = 00. Memoria SRAM de 32 bits
SUPER = 0 Sin máscara para supervisor
ENABLE = 1 Se habilita el uso de CS1#

CSOR1 = $FFFF F078
BAM = $FFFFF. 4kB = 2^{12}. Se comparan los 32 − 12 = 20 bits superiores
WS = $1E = %1 1110. Máximo número de estados de espera
RW = 0. Es indiferente
MRW = 0 (lectura/escritura) ¿? son independientes

RAM externa → CS
RAM interna → RomBar

1.3. Si la inicialización del apartado anterior se incluye dentro de la subrutina HW_INIT que es llamada desde PRINCIPAL, en vez de incluirla directamente en PRINCIPAL, ¿qué sucedería?

Al no haber inicializado la activación de CS para la memoria RAM, habría un error de bus generado por el watchdog hardware al intentar guardar en la pila (RAM) la dirección de retorno de la subrutina, y habría un doble error de bus al intentar guardar en la pila los parámetros asociados a la excepción de error de bus.

1.4. Asigne justificadamente valores a las siguientes constantes del programa (asigne las direcciones *más* bajas posibles y permita que la pila pueda crecer lo máximo posible).

SMMMMMMMM = $0010 0000
Es la dirección más baja posible para la RAM, ya que la ROM empieza en la dirección $0 y tiene un tamaño de 1 MByte.

SLLLLLLLL = $ 0010 1000 valor inicial del PP.
Para tener la mayor pila posible, el valor inicial del puntero de pila debe ser la dirección siguiente al final de la RAM, pues las variables están situadas al principio de ésta.

SXXXXXXXX = $ 124
VBR = $0 (por defecto)
PIVR = 64 (en INTERR_INI)
UART1 tiene offset 9 respecto a PIVR (ver chuleta)

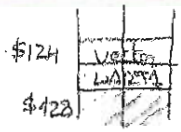
Número de vector de UART1 = PIVR + 9 = 73
Dirección del vector = VBR + 4 × Nº vector = (→ vector 9 índice dir base de la tabla de excep. normal / vale 0)
= 0 + 4 × 73 = 292 = $124

1.5. Asigne justificadamente a SNNNNNNNN (justo antes de la etiqueta PRINCIPAL) el menor valor posible para aprovechar al máximo la memoria disponible, considerando que no hay más fuentes de interrupción que las descritas en el enunciado.

Ahí es donde comienza el programa principal (ROM). Como dice que no hay más fuentes de interrupción que las del enunciado y la que tiene el mayor nº de vector es la de la UART1 ⇒ Las últimas direcciones ocupadas son las del vector de UART1. El programa se puede situar a partir de ahí ⇒ $NNNN NNNN = $124+4 = $128

Sólo vector de int UART1. P. principal justo después del vector de int

al resto de vectores de int no existen



2. Comunicaciones serie

2.1. Determine el valor de t_{bit} y calcule el máximo número de tramas completas por segundo ($N_{máx}$) que se podrían llegar a enviar desde el Sistema de Adquisición en las condiciones especificadas por el enunciado. Considere sólo las limitaciones temporales impuestas por la comunicación serie. Justifique sus respuestas.

$t_{bit} = \frac{1}{19200 \frac{bits}{sg}} = 52 \frac{\mu seg}{bit}$

Cada trama 48 "bytes"

Cada "byte" tiene { 8 bits de datos; 3 arranque, paridad, parada; 2 guarda }

Duración de cada trama con guarda:

$T_{trama} = 48 \frac{"bytes"}{trama} \times (8+3+2) \frac{bits}{"byte"} \times \frac{1}{19200} \frac{sg}{bits}$

$= 0,0825 \, sg/trama$ (+0,050 guarda)

$V = 19200 \frac{bits}{sg}$

$N_{máx} = 1/T_{trama} = 12,12 \, tramas/sg \Rightarrow 12$ tramas completas

2.2. Asigne justificadamente los valores adecuados a las constantes $YY y $ZZ usadas en la rutina HW_INI para que se cumplan las condiciones del enunciado.

$YY = $ 75 = % BR (%11) DT ST PR Tx Rx = 0 1 1 1 0 1 0 1

Corresponde a la inicialización del registro XCR

BR = %11 (19200 baudios)
DT = %1 (8 bits de datos)
ST = %0 (1 bit de parada)
PR = %1 (paridad impar)
Tx = %0 (interrup. de Tx deshabilitada)
Rx = %1 (" Rx habilitada)

$ZZ = $ 1F

Corresponde a la inicialización del registro XSR

TE = %0. Es un bit de sólo lectura así que da lo mismo lo que escribamos

OV = %1
FR = %1
PR = %1
Tx = %1
Rx = %1

} Al escribir un 1 se ponen a 0 todos estos flags

2.3. Complete la siguiente rutina que envía la cadena de caracteres MENSAJE_INIC a través de la UART del MCF5272 al sistema de adquisición.

```
PUT_STRING   LEA      MENSAJE_INIC,A0
PUT_CHAR     MOVE.B   XSR,D0
             BTST.B   #_5_,D0      * Comprueba el bit TE del XSR (preparado para transm)
             BEQ      PUT_CHAR
             MOVE.B   (A0)+,XDAT   * Si está preparado pone el carácter del mensaje
             MOVE.B   #_1_,D0        en el registro de datos de la UART para transmitirlo
             BSET.B   D0,XSR       * Pone a uno el bit TX del XSR (Dato transmitido)
             TST.B    (A0)
             BNE      PUT_CHAR     * Comprueba el fin de la cadena
             RTS
```

3. Análisis del software

3.1. Explique cuándo se ejecuta la rutina RUT_EXC y cómo puede terminar su ejecución.

Se ejecutará cuando se produzca un error de bus o de dirección, pues su dirección está en la tabla de vectores de interrupción en las posiciones correspondientes a los vectores 2 y 3. Dado que contiene un bucle infinito y las dos excepciones (error de bus y de dirección) son de alta prioridad, sólo se podrá salir con un RESET (ver tabla página T4-3)

3.2. Complete la tabla con las cadenas de caracteres alfanuméricos generadas en BUF_SALIDA desde PROC_BETA cuando los primeros 4 bytes de una trama han sido los indicados en la tabla. Represente el espacio en blanco con el símbolo "_" y no considere el carácter NULL final. (NOTA: Tenga en cuenta que se piden los caracteres alfanuméricos y no sus códigos ASCII).

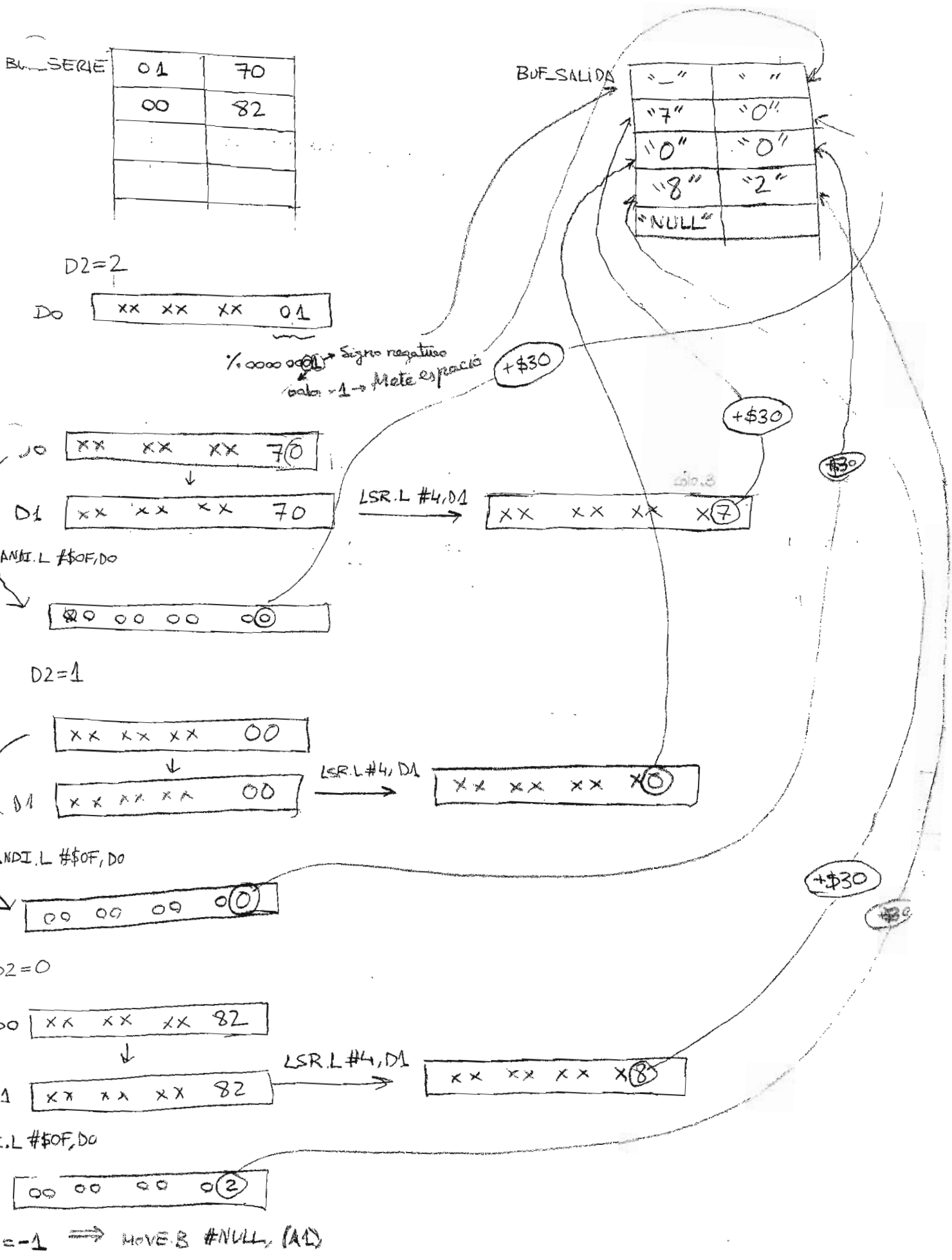
BYTES DE LA TRAMA				CADENA DE CARACTERES
Byte 1	Byte 2	Byte 3	Byte 4	
$01	$70	$00	$82	-_700682
$03	$00	$00	$45	-0000,45

3.3. El sistema de adquisición de datos necesita ser inicializado por el MCF5272, que debe enviarle un mensaje por la línea serie que contiene la frecuencia de muestreo y el número de bits. Justifique por qué este mensaje (MENSAJE_INIC) está situado en la zona del código y no en la zona de variables.

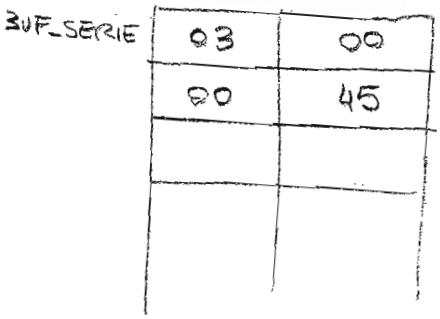
El mensaje es un DC y debe ir en ROM para que al inicializarse el sistema contenga los datos deseados. Como el sistema dispone de una única ROM, un buen sitio para situar estos datos es tras el código.

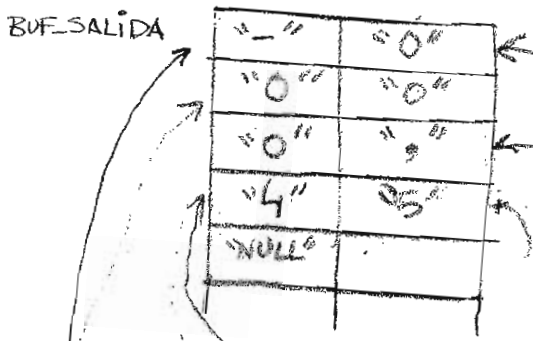
Lo q. va con normativa DC tiene que estar en ROM! no RAM es volátil

Pregunta 3.2. a)



Pregunta 3.2. b)





D2 = 2

D0 [xx xx xx 03]

%0000 0011 → signo negativo

valor x0,01 → salta a DECOD (no mete espacio)

D0 [xx xx xx 00]

↓

D1 [xx xx xx 00] —LSR.L #4,D1→ [xx xx xx x0]

+$30

ANDI.L #$0F,D0

+$30

D0 [00 00 00 00]

D2 = 1

Igual que la anterior

D2 = 0

D0 [xx xx xx 45]

↓

D1 [xx xx xx 45] —LSR.L #4,D1→ [xx xx xx x4]

Mete "," (código a partir de TST.L D2)

+$30

ANDI.L #$0F,D0

+$30

D0 [00 00 00 05]

D2 = -1 ⇒ MOVE.B #NULL, (A1)

TEMA 6: MÓDULOS DE TEMPORIZACIÓN EN EL SISTEMA MICROPROCESADOR

6.1 Temporizadores programables
6.2 Modulación por anchura de pulso

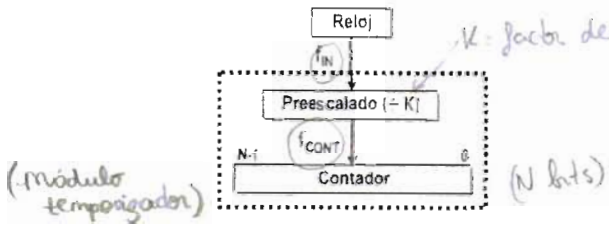
6.1 Temporizadores programables

6.1.1 Elementos de un temporizador

Un módulo temporizador es un periférico para el control temporal de eventos, basado en un contador de N bits. Sus aplicaciones más típicas son:

- Medida del periodo o la frecuencia de una señal externa
- Medida de la anchura de un pulso
- Generación de un evento tras un cierto tiempo (*timeout*)
- Generación periódica de eventos

Suele llevar un factor de preescalado, por lo que la señal de reloj en el contador es la de entrada, dividida por dicho factor de preescalado



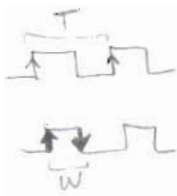
$$f_{CONT} = \frac{f_{IN}}{K}$$

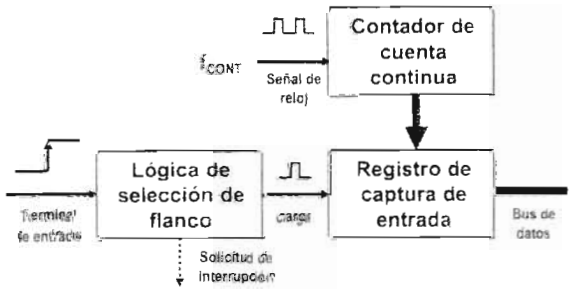
6.2.2 Captura de entrada vs. comparación de salida

Captura de entrada

Se refiere a la **detección** temporal de eventos: detectar que se ha producido un flanco en una entrada y determinar en qué momento se ha producido. Esto sirve para:

- Medida del periodo o la frecuencia de una señal externa (capturando dos flancos sucesivos de subida o dos flancos sucesivos de bajada)
- Medida de la anchura de un pulso (capturando dos flancos sucesivos de distinta polaridad)

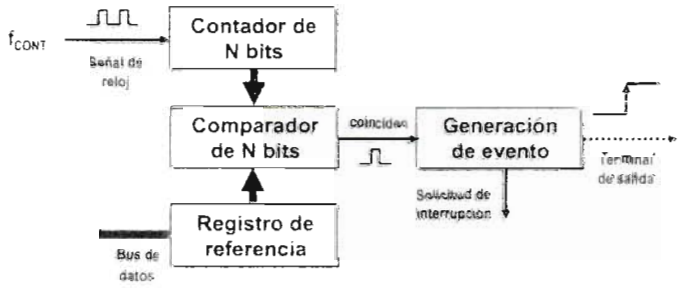




Comparación de salida

Se refiere a la **generación** temporal de un evento en un tiempo dado. Los eventos pueden ser interrupciones o flancos de cierta polaridad en una salida o ambos simultáneamente. Las aplicaciones de esto son:

- Programación de un fin de temporización o *timeout:* tras un tiempo dado se produce una interrupción o se activa un flanco en una salida, o ambas cosas a la vez.
- Generación periódica de eventos:
  - Interrupciones periódicas (en tiempo real)
  - Generación de ondas cuadradas de una frecuencia dada



6.1.3 El módulo temporizador del MCF5272

Hay 4 temporizadores

Ver Tema 4

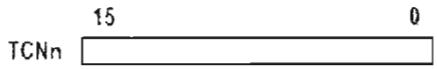
- Está formado por cuatro temporizadores independientes (TIMER0 a TIMER3)
- Además incluye el temporizador del Watchdog
- Cada temporizador incluye:
  - Contador de 16 bits TCNn
  - Fuente de reloj y preescalado programables ($f_{cont}=f_{IN}$ / K)
  - Registro de control (modo) para configuración TMRn
  - Registro de estado para eventos temporales TERn
  - Registro de captura TCAPn
  - Registro de referencia TRRn
  - Bloques (lógica de detección, comparador, lógica de salida...)

$F_{cont} = \frac{F_{in}}{K}$

Registros del módulo temporizador

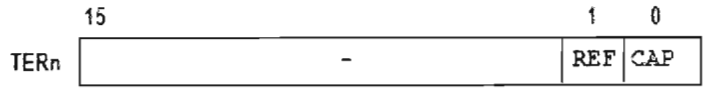
- **Contador ascendente de 16 bits TCNn**
  - Se puede leer sin afectar la cuenta
  - Si se escribe se pone a 0

Igual que el del Watchdog



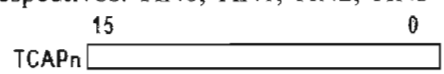
- **Registro de estado TERn**
  - CAP=1 indica la captura de un flanco de entrada
  - REF=1 indica que se ha alcanzado el valor de referencia
  - El resto de bits deben ir a 0 (reservados)

Se ponen a 0 escribiendo un 1 (escribir un 0 no afecta)

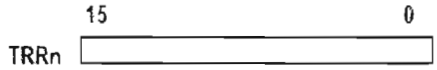


- **Registro de captura TCAPn**
  - Valor del contador al producirse una captura de entrada
  - Terminales de entrada respectivos: TIN0, TIN1, TIN2, TIN3

Es un registro de sólo lectura



- **Registro de referencia TRRn**
  - Valor de referencia para comparación de salida
  - Terminales de salida: TOUT0 y TOUT1 (sólo en TIMER0 y TIMER1)



- **Registro de control (modo) TMRn**

  - RST: habilita (RST=1) o pone a 0 (con el paso a RST=0) el temporizador

  - CLK: fuente de reloj, f_{IN}
    - 00: detiene el contador
    - 01: el reloj es el del sistema
    - 10: el reloj es el del sistema dividido por 16
    - 11: el reloj es una señal externa en TINn (sólo en TIMER0 y TIMER1)

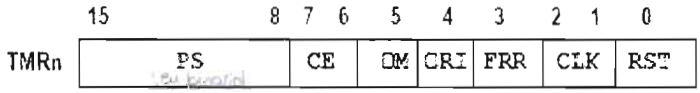
  - FRR: modo continuo/reinicio del contador
    - 0: modo continuo
    - 1: modo reinicio

  - ORI=1 habilita interrupciones en comparación de salida

  - OM: modo de salida (sólo en TIMER0 y TIMER1)
    - 0: genera un pulso activo a nivel bajo durante un ciclo en TOUTn
    - 1: conmuta el nivel de salida de TOUTn

  - CE: captura de entrada y habilitación de interrupciones
    - 00: captura e interrupciones inhabilitadas
    - 01: captura con flanco de subida + interrupción
    - 10: captura con flanco de bajada + interrupción
    - 11: captura con ambos flancos + interrupción

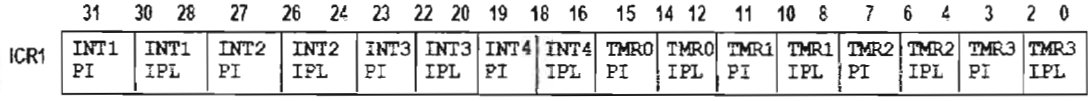
  - PS: preescalado, $f_{CONT}=f_{IN}/(PS+1)$



Interrupciones del temporizador

- El bit CE de TMRn habilita interrupciones cuando se produce captura de entrada
- El bit ORI de TMRn habilita interrupciones cuando se produce comparación de salida
- El nivel de prioridad de la interrupción se fija en el registro ICR1 del controlador de interrupciones

Fíjate que esto no es nuevo, ya lo vimos en el tema 4. sirve como repaso aplicado a un caso concreto



- TMRnIP: solicitud de interrupción pendiente desde TIMERn
  - Refleja el estado de TERn si hay interrupciones habilitadas en TMRn
  - ¡Se pone a cero automáticamente!
  - Hay que poner a 1 el bit PI para cambiar el valor del campo IPL (igual que siempre)

¡OJO! No hace falta poner TMRnIP a 0 en la rutina de atención como ocurría con INT1 IP-INT6 IP

- **Resolución:**
  - Mínimo tiempo que el temporizador es capaz de medir
  - Corresponde al ciclo del reloj más rápido ($T = \frac{1}{66 \cdot 10^6}$) (15 ns si $f_{IN} = 66$ MHz)

- **Rango:**
  - Máximo tiempo que el temporizador es capaz de medir
  - Corresponde a esta situación:

$f_{IN} = 66\text{ MHz}/16$
$K = PS + 1 = 256$
$2^{16} = 65536$
$\Big\}$ $16 \cdot 256 \cdot 65536 = 2^{28}$ ciclos del reloj del sistema.

En segundos $= \frac{2^{28}}{66 \cdot 10^6} = 4{,}067$ s.

  - Medida de tiempos más largos
    - ↳ Se genera una interrupción cada fin de cuenta del temporizador
    - ↳ Se cuenta el nº de interrupciones generadas por el programa

RESOLUCIÓN (tiempo mínimo) ↓ máx. frecuencia

CLK = %01 → $f_{IN} = f_{CLK}$.
PS = $00 → $f_{CONT} = \frac{f_{IN}}{PS+1} = f_{CLK}$

$T_{mín} = \frac{1}{f_{CLK}} = \frac{1}{66 \cdot 10^6} = 15\text{ ns}$

RANGO (tiempo máx → mín frecuencia)

CLK = %10 → $f_{IN} = f_{CLK}/16$.
PS = $FF → $f_{CONT} = \frac{66/16}{255+1}$ (MHz)

Ciclo del reloj de contador $2^N = 2^{16}$

Rango $= 2^N T_{CONT} = 2^{16} \cdot \frac{256}{66/16} = 4{,}067$ s

$$f_{CONT} = \frac{f_{CLK}/16}{PS+1}$$

Igual que siempre. Es sólo un repaso

- TMRnIPL: Nivel de prioridad de la interrupción del TIMERn
  - Para interrumpir, TMRnIPL debe ser superior a I2 I1 I0 del SR
  - Si TMRnIPL=0 no se genera interrupción

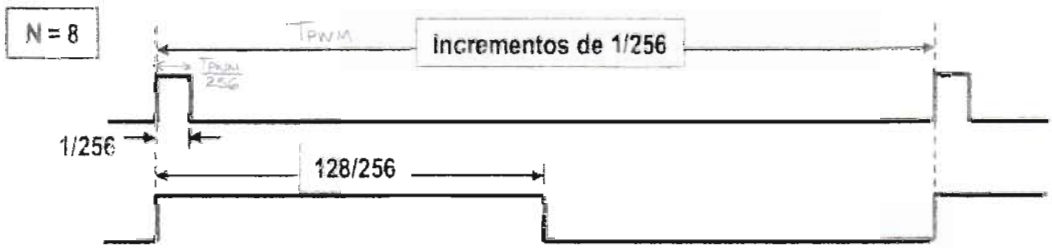
- Los vectores de interrupción tienen un offset de 5-8 con respecto al valor del PIVR

6.2 Modulación por anchura de pulso

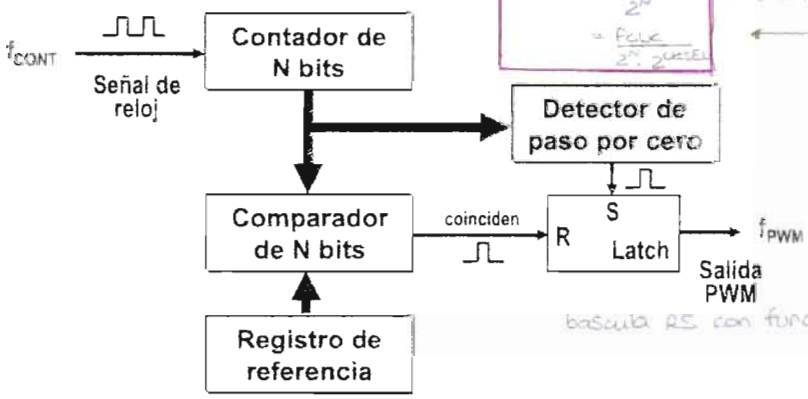
6.2.1 Concepto y bloques de un modulador PWM

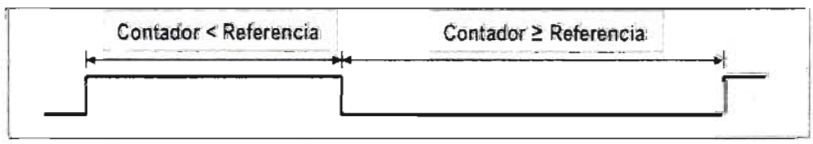
- Señal periódica rectangular con ciclo de trabajo programable
- Ciclo de trabajo: $T_{nivel\ alto}$ / T
- Base de tiempos: un contador de N bits (T corresponderá al tiempo de 2^N ciclos de contador = valor máximo del contador)
- Un contador de N bits permite 2^N ciclos de trabajo distintos
- Frecuencia de la portadora PWM: $f_{PWM} = f_{CONT} / 2^N$

Ejemplo:



El diagrama de bloques será el siguiente:





Configurando el registro de referencia podemos conseguir el ciclo de trabajo deseado

6.2.2 El módulo PWM del MCF5272

- Tres moduladores independientes (PWM0, PWM1 y PWM2)
- Cada modulador incluye:
  - Contador de cuenta continua de 8 bits
  - Señal de reloj de la CPU, fCLK
  - Unidad de preescalado de la señal de reloj
  - Registro de control, PWCRn
  - Registro de ciclo de trabajo o anchura de pulso, PWWDn
  - Bloques (comparador, detector,...) para la generación PWM

Registros del módulo PWM

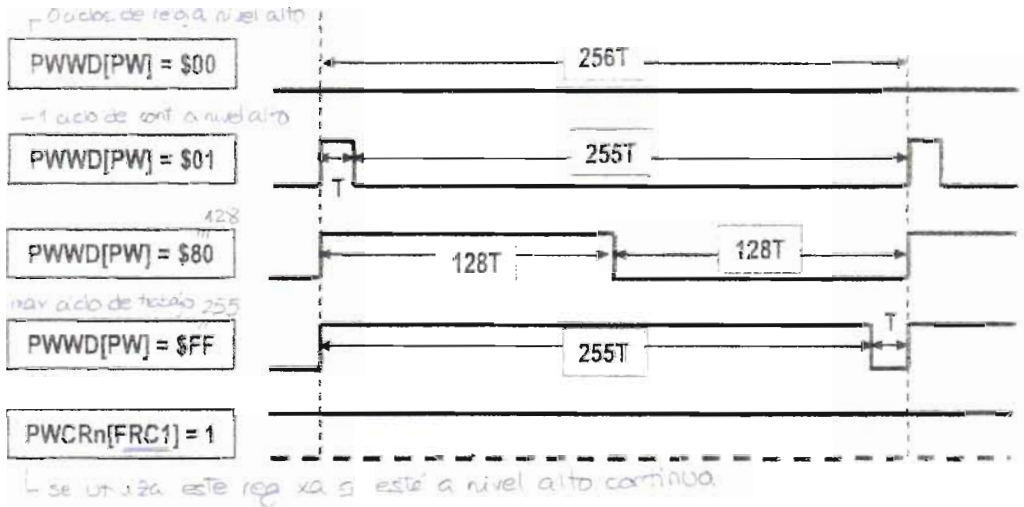
- **Registro de control PWCRn**
  - CKSEL: preescalado de la señal de reloj (4 bits)
    - $f_{CONT} = f_{CLK} / 2^{CKSEL}$
    - $f_{PWM} = f_{CONT} / 2^N = f_{CLK} / (2^8 \times 2^{CKSEL})$
  - LVL: nivel (0 ó 1) de la señal PWM con modulador inhabilitado
  - FRC1: fuerza nivel alto permanentemente (FRC1=1) la salida de la señal PWM
  - EN: habilita (EN=1) el modulador

	7	6	5	4	3	2	1	0
PWCRn	EN	FRC1	LVL	-	CKSEL			

- **Registro de anchura PWWDn**
  - Es el registro de referencia
  - Fija la anchura del pulso a nivel alto (ciclo de trabajo)
    - Si contador < PWWDn ⇒ salida PWM=1 (nivel alto)
    - Si contador ≥ PWWDn ⇒ salida PWM=0 (nivel bajo)

	7	0
PWWDn		

Controlaremos el ciclo de trabajo fijando el valor en este registro:



Ejemplo 6.1 Resolución y rango del temporizador

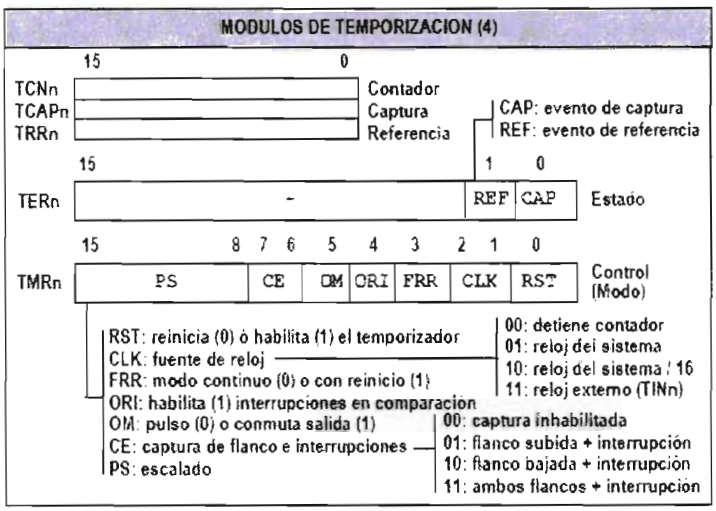
a) Calcule la resolución de los temporizadores de MCF5272 *(cuanto + grande sea la resol mejor, debe ser el reloj mayor!)*

Nota: Se entiende por resolución el mínimo tiempo que el temporizador es capaz de medir

b) Calcule el rango de los temporizadores del MCF5272

Nota: Se entiende por rango el máximo tiempo que el temporizador es capaz de medir

c) ¿Qué habría que hacer para medir tiempos mayores que el rango del temporizador?



(a) Para una resolución mejor queremos el reloj mayor. (Tmin)

↳ Corresponderá a usar el ciclo de reloj más rápido ⇒ utilizar la máxima frecuencia posible en el contador

- Campo CLK del TMRn = %01 (reloj del sistema) ⇒ $f_{IN} = f_{CLK}$ (división de reloj desactivado)
- Campo PS de TMRn = %00 ⇒ $f_{CONT} = \frac{f_{IN}}{PS+1} = \frac{f_{CLK}}{0+1} = f_{CLK}$ (preescalado desactivado)

$$\text{Resolución} = T_{min} = \frac{1}{f_{CONT\,max.}} = \frac{1}{f_{CLK}} = \frac{1}{66 \cdot 10^6} = 15\ ns$$

(b) Para poder medir el mayor tiempo posible el reloj del contador tendrá que ser lo más lento posible.

Corresponde a la situación: *(Se supone q no hay ninguna señal de reloj externa)*

- Campo CLK de TMRn = %10 ⇒ $f_{IN} = \frac{f_{CLK}}{16} = \frac{66\ MHz}{16}$
- Campo PS de TMRn = %FF ⇒ $f_{CONT} = \frac{f_{IN}}{PS+1} = \frac{f_{IN}}{255+1} = \frac{66/16}{256}\ MHz$
- Número de ciclos correspondientes al tiempo máximo: $2^{24} = 2^{16}$ ciclos de reloj de contador.

- Tiempo entre dos incrementos sucesivos del contador.

$$T_{CONT} = \frac{1}{f_{CONT}} = \frac{256}{66/16}\ \mu s = 52\ \mu s$$

- $\text{Rango} = 2^N \cdot T_{CONT} = 2^{16}\ \text{ciclos} \times \frac{256}{66/16}\ \frac{\mu s}{\text{ciclo}} = 4'067\ sg$

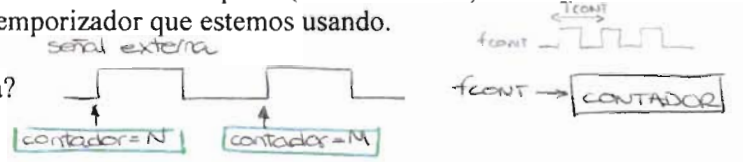
c
- Generar una interrupción cada fin de cuenta del temporizador
- Contar el número de interrupciones generadas, por programa.
- Cuando ese nº alcance el valor correspondiente, sabes que ha transcurrido el tiempo deseado.

Ejemplo 6.2. Medida del periodo o la frecuencia de una señal

Es un ejemplo de captura de entrada. La medida del periodo o frecuencia de una señal consiste en capturar dos flancos sucesivos de igual polaridad.

Si N y M son los valores del contador en los instantes de captura (N anterior a M) estos valores se almacenarán sucesivamente en el registro TCAP del temporizador que estemos usando.

a) ¿Cual es el periodo de la señal medida?



$T_{señal} = (M-N)$ ciclos de reloj de contador $\times T_{CONT}$ s/ciclo de reloj de contador $= (M-N) \times \frac{1}{f_{CONT}}$

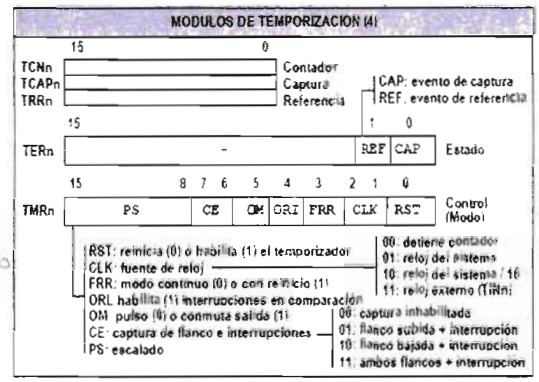
b) Observe como se podría configurar el TMR0 para la medida del periodo de una señal externa por TIN0

- TMR0=$2044
  - RST= 0. Hay que habilitarlo (RST=1) después de configurar
  - CLK= 10. La fuente de reloj es el reloj del sistema dividido por 16
  - **FRR= 0 Modo de ejecución continua**
  - ORI= 0. no se generan interrupciones de comparación de salida
  - OM= 0. Modo de salida (irrelevante porque estamos usando captura de entrada)
  - **CE= 01. Captura de flancos de subida + interrupción**
  - PS= $20. Preescalado = PS + 1 = 33

  (El valor de PS hay que ajustarlo a un valor razonable dependiendo de lo que se espera medir, para evitar desbordamientos del contador)

c) Interprete el código siguiente

```
* Constantes

MBAR_MEM   EQU $10000
TMR0       EQU $200
TMR0L      EQU $201
TRR0       EQU $204
TER0L      EQU $211
TCAP0      EQU $208

* Inicialización
       CLR.L    D0              * Borrado de D0
       CLR.L    D1              * Borrado de D1
       LEA      MBAR_MEM,A0     * Direccion referencia (MBAR)
       MOVE.W   #$2044,D2       * Configuración de TMR0 via D2
       MOVE.W   D2,TMR0(A0)     * Acceso relativo a MBAR
       BSET     #0,TER0L(A0)    * Puesta a 0 del bit de estado
       BSET     #0,TMR0L(A0)    * Arranque del contador

* Rutina de atención a interrupción
RINT   ADDA.L   #-12,A7
       MOVEM    D0-D1/A0,(A7)   * Guardar el contexto
       LEA      MBAR_MEM,A0     * Direccion referencia (MBAR)
       BSET     #0,TER0L(A0)    * Puesta a 0 del bit CAP
       MOVE.W   D0,D1           * Salvar captura anterior en D1
       MOVE.W   TCAP0,D0        * Salvar captura actual en D0
       SUB      D0,D1           * Número de ciclos en D1

       MOVEM    (A7),D0-D1/A0   * Recuperar el contexto
       ADDA     #12,A7
       RTE
```



d) Calcule la frecuencia de una señal para la que los valores de TCAP0 en las dos capturas sucesivas hayan sido M=(D0)=200 y N=(D1)=300

Paso a paso:

- Nº de ciclos de f_{CONT} entre capturas: $M-N = 300-200 = 100$ ciclos de contador
- Nº de ciclos de reloj de fIN entre capturas: $(M-N) \times (PS-1)$

Directamente:

$$T_{señal} = (M-N)\ \text{ciclos de contador} \times T_{CONT}\ \frac{\text{seg}}{\text{ciclo contador}}$$

$$f_{CONT} = \frac{f_{IN}}{PS+1} = \frac{f_{CLK}/16}{PS+1} = \frac{66.10^6/16}{33}\ Hz$$

$$T_{CONT} = \frac{1}{f_{CONT}}$$

$$T_{señal} = (M-N) \times \frac{(PS+1)}{f_{CLK}/16} = (300-200) \times \frac{32+1}{66.10^6/16}$$

$$f_{señal} = \frac{1}{T_{señal}} = 1250\ Hz$$

Ejemplo 6.3. Generación de interrupciones en tiempo real

PERIÓDICO → MODO REINICIO FRR=1.

Es un ejemplo de comparación de salida. Sirve por ejemplo para implementar las interrupciones periódicas en tiempo real como las que se necesitaban cada milisegundo en el ejercicio del sistema de cronometraje automático para piscinas olímpicas.

- La periodicidad se consigue configurando en modo **reinicio.** Cada vez que el contador alcanza el valor de referencia, se reinicia la cuenta.
- En TRRn habrá que poner el número de ciclos del reloj de contador que corresponden al tiempo deseado entre interrupciones.

TRR_n = registro de referencia = nº ciclos que corresponden al tiempo entre interrupciones.

a) ¿Cuál será el valor de la frecuencia de la interrupción?

$$f_{IRQ} = \frac{f_{CONT}}{TRR_n}$$

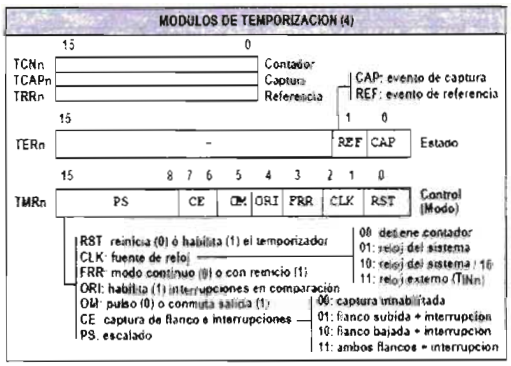
$$f_{CONT} = \frac{f_{IN}}{PS+1} = \frac{f_{CLK}/16}{PS+1}$$

T_{IRQ}

b) Observe un ejemplo de configuración del TMR0 para generar interrupciones cada 100 ms. ¿Qué valor debería tener el TRR0?

Tras llegar al valor de ref se produce interrup

- TMR0=$631C
  - RST = 0. Se habilita (RST=1) después de configurar
  - CLK = 10. La fuente de reloj es el reloj del sistema dividido por 16
  - **FRR = 1. Modo de reinicio** pq es periódica
  - **ORI = 1. Habilita interrupciones de comparación de salida**
  - OM = 0. Modo de salida (Irrelevante porque no queremos generar una señal de salida)
  - CE= 00. Captura de entrada inhabilitada
  - PS = $63. Preescalado = PS +1 = 100



(a) $f_{IRQ} = f_{CONT}\left(\frac{\text{ciclos de cont}}{sg}\right) / TRR_n\left(\frac{\text{ciclos de cont}}{\text{interrupc.}}\right) = \frac{f_{CONT}}{TRR_n}\left(\frac{\text{interrup}}{sg}\right)$

$T_{IRQ} = TRR_n\left(\frac{\text{ciclos cont}}{\text{interrup}}\right) \times T_{CONT}\left(\frac{sg}{\text{ciclo cont}}\right)$

$f_{IRQ} = \frac{1}{T_{IRQ}}$

(b) $T_{IRQ} = \frac{TRR0}{f_{CONT}} = \frac{TRR0}{\frac{f_{CLK}/16}{PS+1}} \Rightarrow \boxed{TRR0 = T_{IRQ} \cdot \frac{f_{CLK}/16}{PS+1}} = 100\cdot10^{-3}\,sg \;\; \frac{66\cdot10^{6}\,Hz/16}{100} =$

$= 4125 = \boxed{\$1015}$

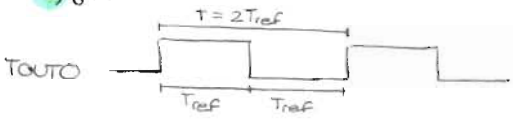
4125 ciclos de un reloj de frecuencia $f_{CONT} = \frac{f_{CLK}/16}{PS+1}$ corresponden a 100 ms

Ejemplo 6.4. Generación de ondas cuadradas (d = 50%)

Es un ejemplo de comparación de salida. Se trata de generar ondas cuadradas de frecuencia programable.

- Señal en una de las salidas del temporizador (TOUT0 ó TOUT1)
- Se cambia el nivel de salida cada vez que se alcanza TRRn, por lo que en este caso TRRn debe ser el número de ciclos del reloj del contador correspondientes a la mitad del periodo de la señal

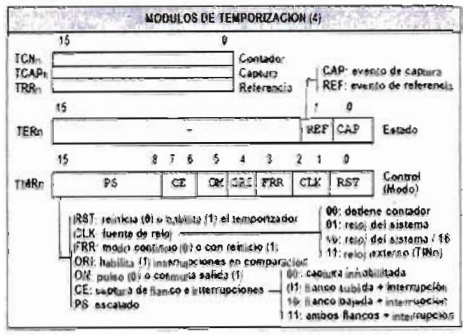
a) ¿Cuál es la frecuencia de la onda cuadrada?



Tref = tiempo correspondiente a TRRn ciclos de reloj de contador.

$$T = (2 \times TRR_n)\ \text{ciclos de cont} \times T_{CONT}\ \frac{\text{seg}}{\text{ciclo cont}} = 2 \times TRR_n \times \frac{1}{f_{CONT}} \Rightarrow \boxed{f = \frac{1}{T} = \frac{f_{CONT}}{2TRR_n}}$$

b) Observe un ejemplo de configuración del TMR1 para generar una señal cuadrada de 1 kHz en TOUT1. ¿Qué valor debería tener el TRR1?



- TMR1=$2F2C
  - RST = 0. Se habilita (RST=1) después de configurar
  - CLK = 10. La fuente de reloj es el reloj del sistema dividido por 16
  - **FRR = 1. Modo de reinicio** x ser ser onda cuadrada
  - **ORI = 0. No se generan interrupciones de comparación de salida**
  - **OM = 1. Modo de salida: se conmuta el nivel de salida** x ser onda cuadrada
  - CE= 00. Captura de entrada inhabilitada
  - PS = $2F. Preescalado = PS + 1 = 48

$$TRR1 = \frac{f_{CONT}}{2f} = \frac{f_{CLK}/16}{PS+1} = \frac{66.10^6/16}{48} = \frac{}{2.10^3} = 42'96 \simeq 43 = \$2B$$

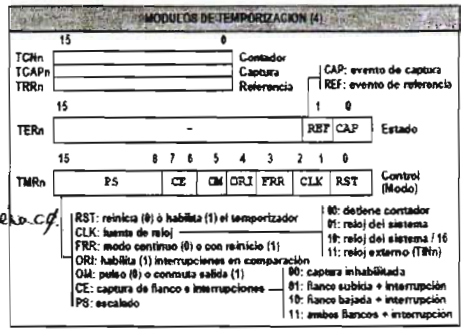
Cada vez q. transcurran 43 ciclos de reloj se conmuta el nivel de la salida.

Ejemplo 6.5. Generación de una señal de fin de temporización (timeout)

Es un ejemplo de comparación de salida.

- Timeout: tiempo máximo permitido para que se produzca un suceso
- En el instante inicial se habilita (reinicia) TCNn o se captura su valor actual
- Cuando se llega al fin de temporización (el contador alcanza el valor de referencia de TRRn) ⇒ Interrupción y/o flanco en una señal de salida

a) Observe la configuración del TMR0 para generar una interrupción que marque el fin de temporización tras 2 segundos. ¿Qué valor debería tener TRR0?



- TMR0=$8314
  - RST = 0. Se habilita (RST=1) después de configurar
  - CLK = 10. La fuente de reloj es el reloj del sistema dividido por 16
  - FRR = 0. Modo de ejecución continua (irrelevante)
  - **ORI = 1. Habilita interrupciones de comparación de salida**
  - OM = 0. Modo de salida (irrelevante)
  - CE= 00. Captura de entrada inhabilitada
  - PS = $83. Preescalado = PS+ 1= 132

TRR0 tendrá el valor correspondiente a 2sg con el configurado.

$$t_{timeout} = 2sg = TRR0 \text{ ciclos de cont} \times T_{cont} \frac{sg}{\text{ciclos de cont}} \Rightarrow \boxed{TRR0 = \frac{t_{timeout}}{T_{cont}}} = t_{timeout} \times f_{cont} =$$

$$= t_{timeout} \times \frac{f_{clk}/16}{PS+1} = 2s \frac{66\,MHz/16}{132} = 62.500 = \boxed{\$F424}$$

b) Observe el código siguiente

```
* Constantes
        ...
MBAR_MEM EQU $10000
TMR0     EQU $200
TMR0L    EQU $201
TRR0     EQU $204
TER0L    EQU $211
        ...
* Inicialización
        LEA     MBAR_MEM,A0   * Dirección referencia
        MOVE.W  #$8314,D2     * Configuración de TMR0
        MOVE.W  D2,TMR0(A0)   * Acceso relativo a MBAR
        MOVE.W  #$F424,D2     * Configuración de TRR0
        MOVE.W  D2,TRR0(A0)   * Acceso relativo a MBAR
        BSET    #1,TER0L(A0)  * Puesta a 0 del bit REF
        BSET    #0,TMR0L(A0)  * Arranque del contador
        ...
* Rutina de atención a interrupción
RINT    MOVE.L  A0,-(A7)      * Guardar el contexto
        LEA     MBAR_MEM,A0   * Dirección referencia
        BSET    #1,TER0L(A0)  * Puesta a 0 del bit REF
        BCLR.B  #0,TMR0L(A0)  * Parada del contador
        ...
        MOVE.L  (A7)+,A0      * Recuperar el contexto
        RTE
```

c) ¿Qué tendría que hacer para un timeout de 10 minutos (> rango del temporizador) partiendo de la configuración anterior?

- Hay que hacerlo contando interrupciones de programa.
- Se configura igual que en el caso anterior, pero en <u>modo reinicio</u> (fíjate q se convierte en una configuración xa int. periódicas en tiempo real).
- Como interrumpe cada 2 sg, hay que contar 300 interrupciones: $300 \text{ int} \times \frac{2s}{\text{int}} = 600sg = 10 \text{ min}$

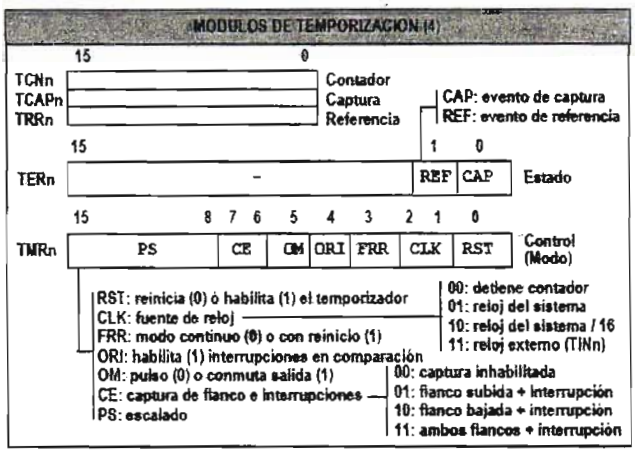
Ejemplo 6.6. Generación de un fin de temporización (timeout) tras un evento externo

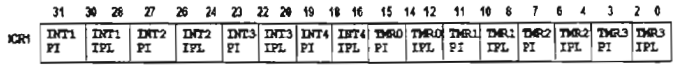
Es ejemplo de combinación de captura de entrada y comparación de salida.

- La captura del evento externo establece el instante inicial a partir del cual se empieza a medir el tiempo
- La comparación de salida genera el fin de temporización
- Se usan dos temporizadores

Diga a grandes rasgos que habría que hacer para generar una interrupción a los 2 segundos de capturar un flanco de subida en TIN0 Modo captura de entrada

* con el modo de comparación de salida





* TIMER 0: captura el flanco en TIN0 (Modo captura de entrada)
  - TMR0: captura de flancos de subida con interrupción. (CE = 01)
  - ICR1: nivel de prioridad de la interrupción (TMR0 IPL) > 0)
  - Rutina de atención: borrar bit CAP de TER0 y arrancar el TIMER 1. (RST)
    para q empiece a contar los 2 ss.

* TIMER 1: comparación de salida para generar int. tras 2sg (M. comparación de salida)
  - TRR1: nº de ciclos de reloj de fsal para 2sg
  - ICR1: nivel de prioridad de la interrupción (TIMER1 IPL)

Ejemplo 6.7. Generar PWM de aproximadamente 15 kHz con un ciclo de trabajo del 40%

Se pide configurar los registros correspondientes al módulo PWM1, con el que se generará la señal deseada:

otro ejemplo
f_{PWM} = 4 KHz
ciclo trabajo = 70%
PWWD1 = 323
PWCR1 = 386

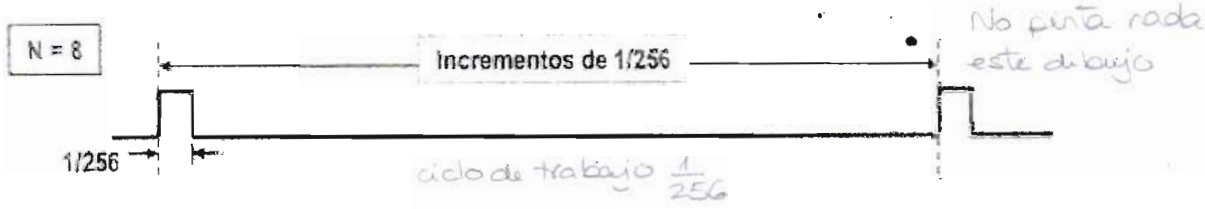
MODULOS PWM (3)

	7	6	5	4	3	2	1	0	
PWWDn	Anchura (ciclo de trabajo)								
PWCRn	EN	FRC1	LVL	-	CKSEL				Control

CKSEL: preescalado $f_{PWM} = f_{CLK} / (256 \times 2^{CKSEL})$
LVL: nivel (0 ó 1) con modulador dehabilitado
FRC1: fuerza (1) la salida PWM a nivel alto permanentemente
EN: habilita (1) el modulador

Nota: la ~~frecuencia de 15 kHz se refiere a f_{PWM}, no a la frecuencia de la señal resultante que será $f_{SEÑAL}=f_{CONT}$~~

~~La relación entre ambas ha sido vista en la teoría y es~~ $f_{PWM} = f_{CONT} / 2^N$, donde N es el número de bits del contador (en el caso del módulo PWM del MCF5272 N=8)



No pinta nada este dibujo

↳ Como queremos ciclo de trabajo del 40%, el valor que hay que meter en el registro de referencia es el 40% del nº máximo de ciclos del contador

$$PWWD1 = 2^N \frac{40}{100} = 2^8 \cdot 0'4 = 102 = \%66$$

↳ Vemos ahora el valor del registro de control PWCR1

- LVL = %0 (irrelevante)
- FCR1 = %0 (No fuerza salida a nivel alto)
- EN = %1 (habilita la señal PWM)
- CKSEL.

$$f_{PWM} = \frac{f_{CONT}}{2^N} = \frac{f_{CLK}/2^{CKSEL}}{2^N} = \frac{f_{CLK}}{256 \times 2^{CKSEL}}$$

$$2^{CKSEL} = \frac{f_{CLK}}{256 \times f_{PWM}} = \frac{66 \cdot 10^6}{256 \times 15 \cdot 10^3} = 17'19$$

Se escoge la potencia de 2 más cercana ⇒ $2^{CKSEL} = 16 \Rightarrow$ CKSEL = 4

Frecuencia PWM real: $f_{PWM} = \frac{f_{CLK}}{256 \times 2^{CKSEL}} = \frac{66 \cdot 10^6}{256 \cdot 2^4} = 16'1$ KHz

Al final queda: PWCR1 = %84

Ejemplo 6.8. Control de velocidad de un motor de continua

Para alimentar la etapa de potencia que controla un motor de continua se utiliza una señal PWM

- Si $f_{PWM} > 1$ kHz $\Rightarrow$ relación directa entre el ciclo de trabajo y la velocidad del motor

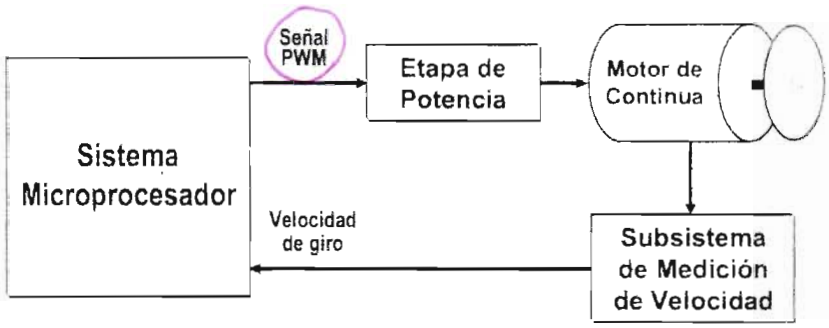
(filtro paso bajo) (conversión D/A)

Para configurar un módulo PWM del MCF5272 para este fin habría que tener en cuenta:

- Registro de control PWCRn: el preescalado (CKSL) debe garantizar $f_{PWM} > 1$ kHz
- Registro de anchura de pulso PWWDn: se va modificando dinámicamente el valor según la velocidad deseada. Si se aumenta el ciclo de trabajo, más velocidad, si se disminuye, menos velocidad del motor.

En la práctica el sistema está realimentado:

- La velocidad real depende de la carga que tenga el motor
- El valor de PWWDn se ajusta según la velocidad real
- Esta velocidad real se mide en el eje del motor



Para más velocidad → aumentar PWWDn

Ejercicio 13. Sistema de control de riego automático

En este ejercicio se desea realizar un sistema de control de riego automático. El sistema no se va a limitar a activar el riego en unos instantes concretos y predeterminados, sino que en función de la humedad de la tierra y la temperatura ambiente se determinará la cantidad de agua necesaria para el riego. El sistema está pensado para ser totalmente autónomo y funcionar sin intervención humana salvo en el caso de emergencias. Para cubrir dichas emergencias se dispone de una línea de aviso a una central para que se envíe a un operador en el caso de un problema irresoluble.

El sistema dispone de un depósito de agua principal y uno de reserva. El depósito de reserva sólo entra en funcionamiento cuando hay un problema con el principal, abriéndose una válvula que deja pasar el agua desde el depósito de reserva al principal. El depósito principal dispone de una válvula con 16 niveles de apertura (el nivel dependerá de la humedad de la tierra y la temperatura ambiente).

Para recoger la medida de la humedad de la tierra y de la temperatura ambiente provenientes de los sensores correspondientes se dispone de un único conversor A/D de 8 bits (ADC0804), del que se van a utilizar los 4 bits más significativos, y de un multiplexor analógico de 8 entradas (74HC4051) con el que se va a seleccionar alternativamente el valor de temperatura o el de humedad. Véase en la Figura 1 su conexionado.

Así mismo, se dispone de un circuito de detección de sequedad que generará una alarma cuando el sistema esté totalmente seco. La existencia de este circuito se debe a la urgencia de detectar la sequedad (no se puede esperar mucho a que se detecte el hecho en la siguiente lectura del dispositivo de medición de humedad).

En resumen, el sistema constará de los elementos de la Figura 1

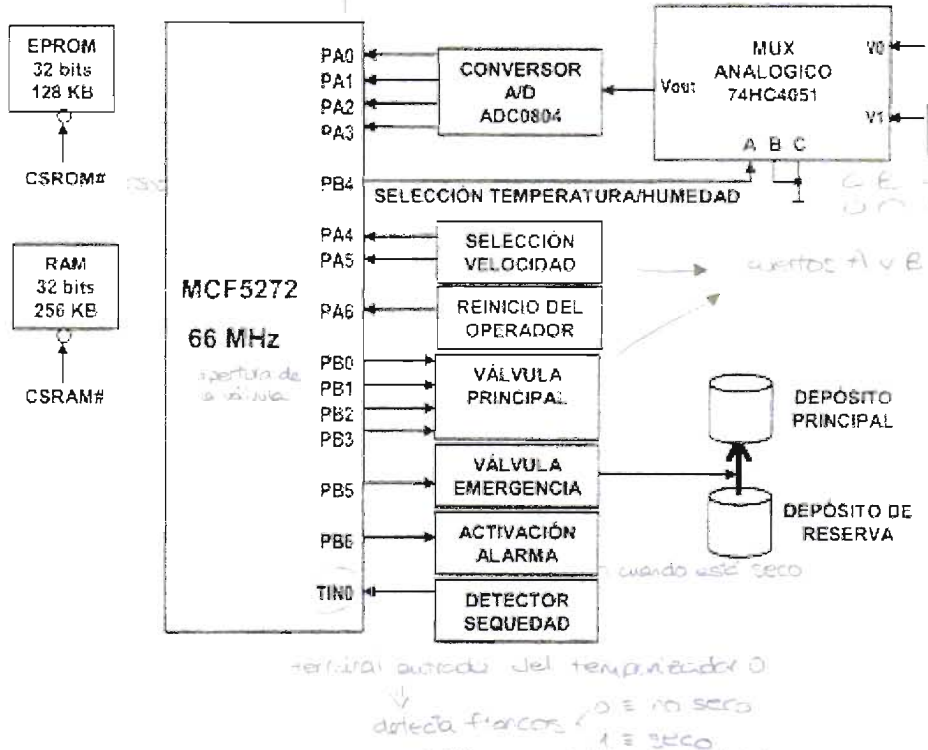


Figura 1. Diagrama de bloques del sistema

Funcionamiento detallado del sistema

Cada 30 minutos se obtienen los valores de humedad y temperatura. En función de dichos valores se calcula el grado de apertura de la válvula de salida de agua y el tiempo que debe permanecer abierta. Para decidir el grado de apertura se utilizan 4 posibles programas de rapidez, determinándose dicho programa consultando los terminales de entrada *Velocidad* que configura el operador en cada caso. Para el control de la activación cada 30 minutos se va a utilizar el temporizador 2 del ColdFire con un nivel de interrupción 4.

Para el control del tiempo que debe permanecer abierta la válvula se va a utilizar la función de comparación de salida en el temporizador 1 del ColdFire. Se va a programar de modo que interrumpa al sistema cuando haya transcurrido el tiempo programado con un nivel de interrupción 2.

Así mismo, el circuito de detección de sequedad estará conectado a la entrada de captura de entrada TIN0 del temporizador 0 del ColdFire de forma que detectamos mediante una interrupción que el sistema se ha quedado totalmente seco, momento en el que se activa la válvula de emergencia que permite el trasvase de agua desde el depósito de reserva al principal. Su nivel de interrupción es 5.

Importante: si cierto tiempo después de activarse la válvula de emergencia el sistema sigue estando seco, se generará una alarma que consistirá en activar la salida *Activa_alarma* para avisar a la central de que se debe enviar un operador para reparar el sistema.

Nota: todos los temporizadores utilizan como fuente de reloj el reloj del sistema dividido por 16.

Terminales de entrada/salida

De forma simplificada se considera que el sistema utiliza los siguientes terminales de entrada/salida, todos ellos activos a nivel alto (la Figura 1 muestra la conexión de estos terminales):

ENTRADAS.

- *Conversor A/D*: 4 terminales en los que se le presenta al sistema el nivel de temperatura o humedad actuales. Los valores están codificados entre 0 (mínima temperatura o humedad) y 15 (máxima temperatura o humedad)
- *Velocidad*: 2 terminales que indican la rapidez con la que debe producirse la salida del agua. Estos dos terminales codifican el índice del programa de rapidez (hay 4 posibles programas)
- *Reinicio del operador*: un terminal utilizado por el operador para reinicializar el sistema.
- *Detec_seco*: un terminal a través del cual el circuito de detección de sequedad señala a la CPU que el depósito principal está totalmente seco. Este terminal está preparado para funcionar como captura de entrada. El circuito genera una señal que se mantiene a cero mientras el sistema se mantenga húmedo, y a uno mientras esté seco.

SALIDAS:

- *Apertura_válvula*: 4 terminales que indican el grado de apertura que debe tener la válvula que deja pasar el agua al exterior para el riego (de 0 a 15).
- *Selec_conversor*: para seleccionar la entrada que debe utilizar el conversor A/D.
- *Válvula_emergencia*: para activar la válvula de emergencia que deja pasar el agua del depósito de reserva al principal.
- *Activa_alarma*: para activar la alarma de aviso en la central para que se mande un operador

Listado parcial del programa

```
*       Definición de constantes
*************************************************************************
MBAR_MEM        EQU             $01000000   direc base del SIM
                                    => direc indirecto con offset (=ACNT(A0))
PACNT           EQU             $80   offset          ; Puerto A: control
PADDR           EQU             $84   offset          ; Puerto A: direccion
PADAT  eje alto EQU             $86   offset          ; Puerto A: datos
PADAT_ eje bajo EQU             $87   offset+1
PBCNT           EQU             $88                   ; Puerto B: cont
PBDDR           EQU             $8C                     Puerto B: dir
PBDAT           EQU             $8E                   ; Puerto B: dat
PBDAT_L         EQU             $8F
PIVR            EQU             $3F
PICR            EQU             $34
ISR             EQU             $30
CSR1            EQU             $20
CSR2            EQU             $24
CSBRR0          EQU             $40
CSOR0           EQU             $44
CSBR1           EQU             $48
CSOR1           EQU             $4C

TMR0            EQU             $200
TRR0            EQU             $204
TCAP0           EQU             $208
TCN0            EQU             $20C
TER0_L          EQU             $211
TMR1            EQU             $220
TMR1_L          EQU             $221
TRR1            EQU             $224
TCAP1           EQU             $228
TCN1            EQU             $22C
TER1_L          EQU             $231
TMR2            EQU             $240
TMR2_L          EQU             $241
TRR2            EQU             $244
TCAP2           EQU             $248
TCN2            EQU             $24C
TER2_L          EQU             $251

CONSULTA        EQU             $00000
RESET_SEDG      EQU             $0000

                ORG             $0    vector excepc RESET
                DC.L            FIN_PILA   puntero pila
direc           "               MAIN   direc inicial prog principal
vectores de excep.  DC.L        $xxxxxxxx
restantes       DC.L            CAP_ENTRADA (temp 0)      ! al estar descendentes hay q ir
                ORG             $yyyyyyyy                   a direc xa cada temp
                DC.L            TIMER (temp 2)
                ORG             $zzzzzzzz
                DC.L            TIMEOUT (temp 1)

*************************************************************************
*       Definicion de variables   (RAM)
*************************************************************************
                ORG             $00280000   direc comienzo de la 2AM
PILA            DS.B            Tam_pila    - en este caso, a pila está al comienzo
FIN_PILA                                      de la RAM, antes de las variables.

Retardo         DS.L            1 (x milis)
Temperat        DS.L            1 (W)           ; Nivel de temperatura actual
Humedad         DS.L            1 (W)           ; Nivel de humedad actual
Plazo           DS.L            1 (W)           ; Tiempo de apertura de la válvula
Apertura        DS.L            1 (W)           ; Grado de apertura de la válvula
Seco            DS.B            1 (1)           ; El sistema se ha secado
Alarma_grave    DS.B            1 (1)           ; Hay una alarma grave en el sistema
Datos_nuevos    DS.B            1 (1)
```

reg config puerto A

puerto B

Nivel 4 TIMER - Temp. 2 - 30 min
Nivel 2 TIMEOUT - Temp. 1 - Tiempo de riego
Nivel 5 CAP. ENTRADA - Temp. 0 - Detección sequedad

```
*****************************************************************************
*       Programa principal   (ROM)
*****************************************************************************
                ORG             $400            ; a determinar apertura de la válvula
Rapidez         DC.L            $3,$7,$B,$F     ; Programar la rapidez

MAIN            MOVE.L          MBAR_MEM+$15,D0
                MOVEC           D0,MBAR
                LEA.L           MBAR_MEM,A0     ! direc indirecto con desplz.

                MOVE.L          #30030001,D0    ; direc base de la ROM
                MOVE.L          D0,CSBR0(A0)
                MOVE.L          #$FFFE3001,D0   ; ROM => tamaño de la ROM
                MOVE.L          D0,CSOR0(A0)

                MOVE.L          #$________,D0   ; Inicializ. BR de la RAM
                MOVE.L          D0,CSBR1(A0)
                MOVE.L          #$________,D0   ; RAM => CS1=
                MOVE.L          D0,CSOR1(A0)

                BSR             SW_INI
                BSR             HW_INI

* Calcular la apertura de la válvula y del tiempo que debe permanecer
* abierta en función de los valores de Humedad y Temperat.
PROGRAMA        TST.B           Alarma_grave    ; comprueba si vale 0, no hay incidencia
donde se va a   BNE             OPERADOR        ; si hay incidencia llama al OPERADOR
                TST.B           Datos_nuevos
                BEQ             PROGRAMA        ; espera a que el periodo de muestreo (T) del temporizador cambie valor de
                CLR.B           Datos_nuevos    ; datos nuevos y siga ejecutando

* Preparación de los datos de entrada
                MOVE.L          Humedad,-(A7)   ; mete en la pila humedad y T
                MOVE.L          Temperat,-(A7)

* Lectura del programa de rapidez
                MOVE.B          PADAT_I(A0),D2  ; coge byte entero de [illegible] rapidez
                AND.L           #$30,D2         ; averigua con la selección de velocidad (bits 4 y 5)
                LSR.L           #4,D2           ; pone a cero los [illegible]
                MOVE.L          D2,-(A7)        ; mete en la pila

* Llamada a la rutina que realiza los cálculos
                PEA             Plazo           ; coge las direc de plazo y apertura
                PEA             Apertura
                BSR             CALCULOS        ; llama subrutina CALCULOS
                ADDA.L          #8,A7

* Programación de la comparación de salida
                CLR.L           D3              ; contador de Tenc1, como está apagado vale 0
                MOVE.W          TCN1(A0),D0     ; Temporizador
                ADD.L           Plazo,D3        ; D3 = D3 + Dc
                MOVE.W          D3,TRR1(A0)     ; => reg de ref (Tenc1) en n ciclos de reloj de cont.
                BSET.B          #0,TMR1_I(A0)   ; pone 1 BSET => empieza a contar, es alarm cuando
                                                ; puede interrumpir al alcanzar el valor de ref.
* Programación del grado de apertura
                MOVE.L          Apertura,D0
                MOVE.B          D0,PBDAT_I(A0)  ; pre valor de Apertura en los bits de la válvula principal
                BRA             PROGRAMA        (PB0-PB2)
                                                ; ciclo [illegible] del riego. Tenc1

*****************************************************************************
*       Rutina de control de operador
*****************************************************************************
OPERADOR        BTST.B          #6,PADAT_I(A0)  ; accede byte bajo del puerto A, comprueba
                BEQ             OPERADOR        ; el bit 6, correspde a reinicio del operador.
                CLR.B           Alarma_grave    ; pone [illegible]
                CLR.B           Seco            ; [illegible]
                CLR.B           Datos_nuevos
                BSET.B          #4,TMR2_I(A0)   ; bit 4 = 0 => habilita cnt. de cuenta de salida
                BSET.B          #4,TMR1_I(A0)
                MOVE.L          #CONSULTA,D0
                MOVE.L          D0,Retardo
                BRA             PROGRAMA
```

```
*******************************************************************************
*       Rutinas de inicialización del sistema
*******************************************************************************

* Inicialización del software
SW_INI          CLR.B           Datos_nuevos
                CLR.B           Seco
                CLR.B           Alarma_grave
                MOVE.L          #CONSIGNA,D0
                MOVE.L          D0,Retardo
                RTS

* Inicialización del hardware
HW_INI          CLR.L           PACNT(A0)       ; Configuración de puertos
                CLR.L           PBCNT(A0)
                MOVE.W          #$0000,D0
                MOVE.W          D0,PADDR(A0)
                MOVE.W          #$007F,D0
                MOVE.W          D0,PBDDR(A0)
                MOVE.W          #$4C0D,D0       ; Configuración de comunicación
                MOVE.W          D0,TMR2(A0)
                MOVE.W          #$KKKK,D0       valor de referencia
                MOVE.W          D0,TRR2(A0)
                MOVE.W          #SEP14,D0
                MOVE.W          D0,TMR1(A0)
                CLR.W           TCN0(A0)
                MOVE.W          #$7CC5,D0
                MOVE.W          D0,TMR0(A0)

                BSR             INTERR_INI      inicia interrup
                RTS

* Inicialización de las interrupciones
INTERR_INI      MOVE.L          #$60,D0
                MOVE.B          D0,PIVR(A0)             ; Inicialización de PIVR
                MOVE.L          #$11111111,D0           ; Inicialización de ICR1
                MOVE.L          D0,ICR1(A0)

                CLR.L           D0
                MOVE.B          CR,D0                   ; Habilitación de
                AND             #$FBFF,D0               ; interrupciones en SR
                MOVE.W          D0,SR           ; Modo Usuario
                RTS

        Tenemos 3 interrup debido a los 3 temp., deben estar encendidos

*******************************************************************************
*       Rutina de calculos del sistema
*******************************************************************************
CALCULOS        MOVEA.L         offset1(A7),A2  ; Apertura
                MOVEA.L         offset2(A7),A3  ; Placa
                MOVE.L          offset3(A7),D2  ; Humedad
                MOVE.L          offset4(A7),D1  ; Temperat
                MOVE.L          offset5(A7),D2  ; Programa

                CLR.L           D3
                MOVE.L          #$0F,D3
                SUB.L           D2,D3   ; - humedad
                LSL.L           #9,D3   ; 2^9 (15-humedad)
                MOVE.L          D1,D4   ; temp -> D4
                LSL.L           #1,D4   ; temp. *2
                ADD.L           D3,D4   suma
                MOVE.L          D4,(A1) lo escribe en PWM
                MOVEA.L         Rapidez,A3 mete dirección de rapidez (BUCO) -> +3
                MOVE.L          0(A3,D2*4),D3   D2 puede valer 0,1,2,3 / lo escribe en PA4 y PA5 = selección de velocidad
                MULU            D1,D3   multiplico la acerada x 50, $7,3E,1F
                ...             ...     divide antes de
                MOVE.L          D3,(A2) lo mete en Apertura
                RTS
```

```
*       Rutinas de servicio de las interrupciones
*************************************************************************
*** Rutina de control de temporización - Temporizador 2
TIMER           MOVE.L          D0,-(A7)
                BSET.B          #1,TER2_L(A0) escribo 1 en bit 1 del TER (bit REF)
                SUBI.L          #1,Retardo  cuando retardo no es 0 ha pasado 30 min
                BNE             SALIR
                BSET.B          Seco
                BNE             ALAR_GRAVE
                MOVE.L          #CONSULTA,D0 si no está seco reinicia Retardo con valor 18000 (consulta)
                MOVE.L          D0,Retardo

* Selección de los valores de humedad y temperatura
                BCLR.B          #4,PBDAT_L(A0) pone a 0 bit 4 => selecciona Temperatura
                                ; Se introduce un retardo
                MOVEA.L         PADAT_L(A0),D0 lee puerto A,
                ANDI.L          #$0F,D0  se queda con los 4 bits + bajos (los otros a cero)
                MOVE.L          D0,Temperat  actualiza Temperat con la medida q obtiene
                BSET.B          #4,PBDAT_L(A0) selecciona Humedad
                                ; Se introduce un retardo
                MOVEA.L         PADAT_L(A0),D0
                ANDI.L          #$0F,D0   se queda con los 4 bits + bajos
                MOVE.L          D0,Humedad  mete la medida de la humedad
                BCLR.B          #1,D0     1 en Datos_nuevos indica al programa q ya puede utilizar
                MOVE.L          D0,Datos_nuevos  los datos nuevos introducidos.
                RTE             SALIR

* ACTIVAR la alarma grave
ALAR_GRAVE      MOVE.B          #1,D0
                MOVE.B          D0,Alarma_grave
                BSET.B          #6,PBDAT_L(A0) pone 1 bit 6 =>activa alarma
                BCLR.B          #4,TMR2_L(A0) pone 0 bit 4 de TMR2 desactiva INT,
                BCLR.B          #4,TMR1_L(A0) desactiva temp 1 y 2
SALIR           MOVE.L          (A7)+,D0
                RTE

*** Rutina de control del timeout de la válvula - Temporizador
TIMEOUT         MOVE.L          D0,-(A7)
                BSET.B          #1,TER2_L(A0) 1 en bit REF = 1
                BCLR.B          #0,TMR1_L(A0) pone a 0 bit 0 de TMR1
                MOVE.B          PBDAT_L(A0),D0 lee en D0 PBDAT
                ANDI.L          #$F0,D0  se queda con los 4 bits + altos => cierra válvula (bit 0 = 0)
                MOVE.L          D0,PBDAT_L(A0)
SALIR           MOVE.L          (A7)+,D0
                RTE             como está parado, ya no puede interrumpir el temp 2 y que lo haga

*** Rutina de control de la captura de entrada - Temporizador
CAP_ENTRADA     MOVE.L          D0,-(A7)
                BSET.B          #0,TER0_L(A0)
                TST.B           Seco
                BEQ             ACTIVA_VALV
                TST.B           Seco
                BCLR.B          #5,PBDAT_L(A0) si el problema de sequedad se soluciona, se desactiva la válvula de emergencia
                BRA             SALIR3

* ACTIVAR la válvula de emergencia
ACTIVA_VALV     BSET            #5,PBDAT_L(A0)
                MOVE.L          #1,D0
                MOVE.B          D0,Seco
                MOVE.L          #SIGUE_SECO,D0
                MOVE.L          D0,Retardo
SALIR3          MOVE.L          (A7)+,D0
                RTE

                END
```

- si se ha entrado a CAP_ENTRADA por flanco de subida SECO = 0 inicial y después se pone SECO = 1. => problema no se soluciona y se activa ALAR_GRAVE

- si se ha entrado en CAP_ENTRADA por flanco de bajada SECO = 1 inicial y después se pone a 0.

1. Configuración del subsistema de memoria

1.1. Complete la siguiente tabla con el mapa de memoria que se deduce a partir de la información del enunciado y del listado.

RANGO DE DIRECCIONES (HEX)	DISPOSITIVO ASIGNADO
$00000000-$0001FFFF	Memoria EPROM (128 KB) = 2^{17}
$00280000-$002BFFFF	Memoria RAM (256 KB) = 2^{18}
$0100 0000-$0100FFFF	Módulo SIM (64K) = 2^{16}

1.2. Teniendo en cuenta el listado, las especificaciones del sistema y que la RAM necesita 2 estados de espera, configure los registros CSBR1 y CSOR1. Justifique cada uno de los campos de los registros.

CSBR1 = $00230001

CSOR1 = $FFFC0008

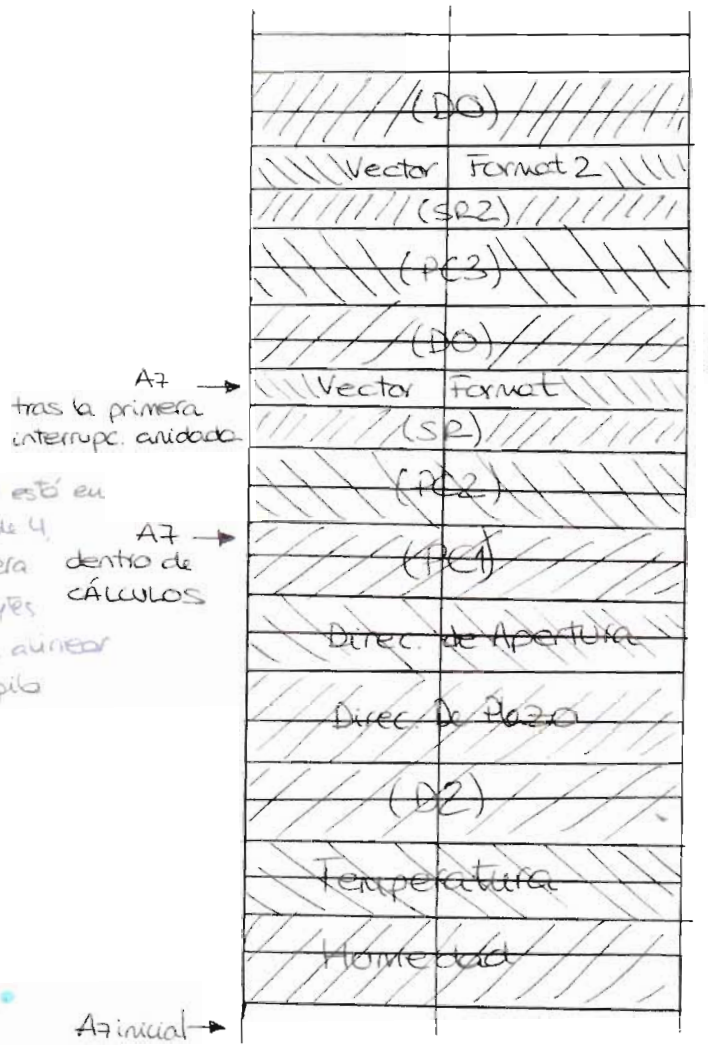
2. Control de la pila

2.1. ¿Cuál debe ser el valor mínimo de la etiqueta Tam_pila? Justifique su respuesta. Se debe tener en cuenta tanto los parámetros que se guardan en la pila como las instrucciones que la utilizan. Suponga que se pueden anidar dos interrupciones en cualquier momento de la ejecución del programa.

- Paso de parámetros desde el programa principal a la subrutina CÁLCULOS:
  - Humedad + Temperatura + (D2): 4+8 = 12 bytes
  - Direcciones de plazo y apertura: 4x2 = 8 bytes
- Dirección de retorno (PC) al salto a la subrutina (CÁLCULOS o RTI): 4 bytes
- Cada interrupción:
  - (PC) 4 byte
  - (SR) 2 byte
  - Vector/Format 2 bytes
  - (D2) 4 bytes

TOTAL: 48 bytes

En total se necesitan como mínimo 48 bytes.



(D0) se lleva a la pila en la rut. de atención de la 1ª int. anidada.

desde interrupc.

direc. de retorno desde rutina CÁLCULOS

en este caso A7 está en direc múltiplo de 4, pero si no lo fuera se dejarían bytes sueltos hasta alinear el puntero de pila

Para meter dirección: PEA PLAZO & NOVEA. #PLAZO, -(A7)

" meter contenido: MOVE.L PLAZO, -(A7)

2.2. ¿Qué deben valer offset1, offset2, offset3, offset4, offset5? Dibuje el contenido de la pila al comienzo de la subrutina CALCULOS incluyendo la posición del puntero de pila si la posición inicial de A7 en la figura es la que tiene en la etiqueta PROGRAMA.

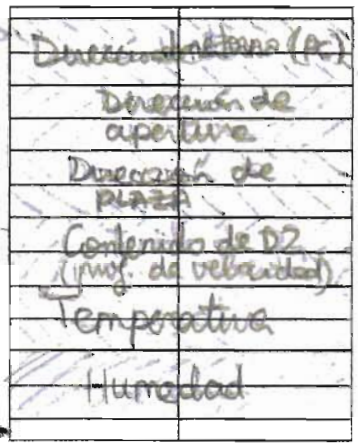
offset1 = 4 (velocidad)

offset2 = 8 (plazo)

offset3 = 20 = $14 (humedad)

offset4 = 16 = $10 (temperatura)

offset5 = 12 = $C (programa velocidad).



2.3. ¿Cuál debe ser el valor de $MM en la instrucción ADDA.L #$MM,A7 del programa principal? Justifique brevemente su respuesta.

$MM = $14 = 20

Este valor corresponde a los 5 bytes que ocupan en la pila los parámetros pasados a la subrutina con esta instrucción de acondicionamiento de la pila

3. Temporización e interrupciones

Direcciones de los vectores de interrupción

3.1. Determine los valores $XXXXXXXX, $YYYYYYYY y $ZZZZZZZZ para que las interrupciones del sistema funcionen correctamente. Justifique su respuesta. ¡¡¡ PIVR ≠ 64

VBR = 0

Direc vector nº N = VBR + (4 x N)

PIVR = offset

$XXXXXXXX = $194 ($60 + 5 x 4)

$YYYYYYYY = $19C (

$ZZZZZZZZ = $198 (

TMR2 se inicializa con 0 para $50 [illegible] Timer inicializado a 0

3.2. Determine el valor $LLLLLLLL en la rutina INTERR_INI para que todas las interrupciones estén adecuadamente configuradas. Justifique su respuesta. Valor del ICR1

$LLLLLLLL = $0000DAC0 = % 0000 0000 0000 0000 1101 1010 1100 0000

TMR0 PI = %1

TMR1 PI = %1

TMR2 PI = %1

TMR0 IPL = 5 = %101 (nivel de interrupción de temporizador 0)

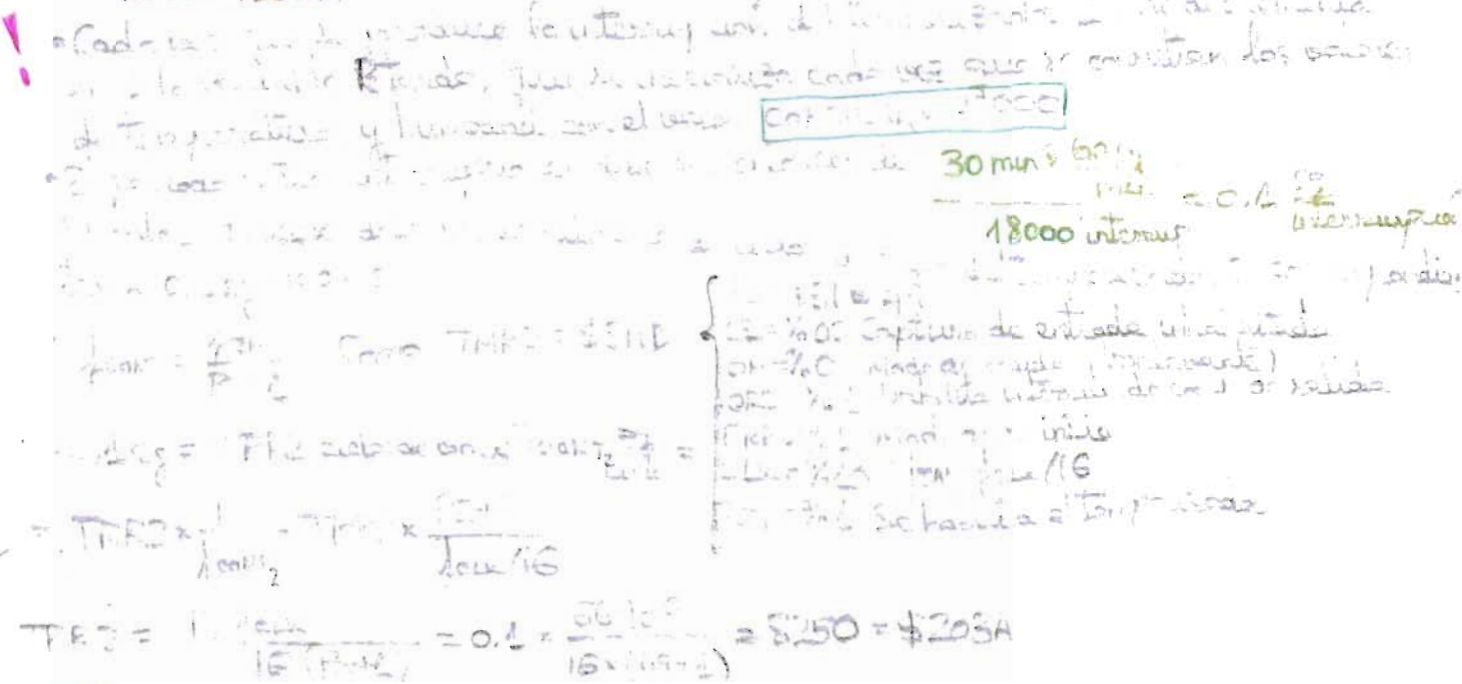
TMR1 IPL = 2 = %010 (1)

TMR2 IPL = 4 = %100 (2)

El resto de las interrupciones están inhabilitadas

3.3. ¿Qué valor debe tener $KKKK para que se examinen los valores de temperatura y humedad cada 30 minutos? Justifique detalladamente su respuesta. Valor del reg. referencia del Temp.2

$KKKK = $ 203A



3.4. ¿Cuándo se detecta que el depósito está totalmente seco?

3.5. ¿Cuánto tardará en activarse Alarma_grave si se detecta que el depósito está seco y no se consigue salir de dicho estado?

6000 interrup x 0,1 sg / interrup = 600 sg = 10 min

3.6. ¿Cómo se detecta que después de haber estado seco el depósito ya no lo está?

3.7. ¿Cuál es el propósito de la variable Datos_nuevos?

3.8. Complete los cambios habría que hacer en la rutina CAP_ENTRADA para saber el tiempo que ha tardado en dejar de estar seco el sistema. Se desea que dicho tiempo quede almacenado en D0 expresado en milisegundos. Suponga que no hay desbordamiento en ninguna operación. A continuación, justifique los cálculos realizados.

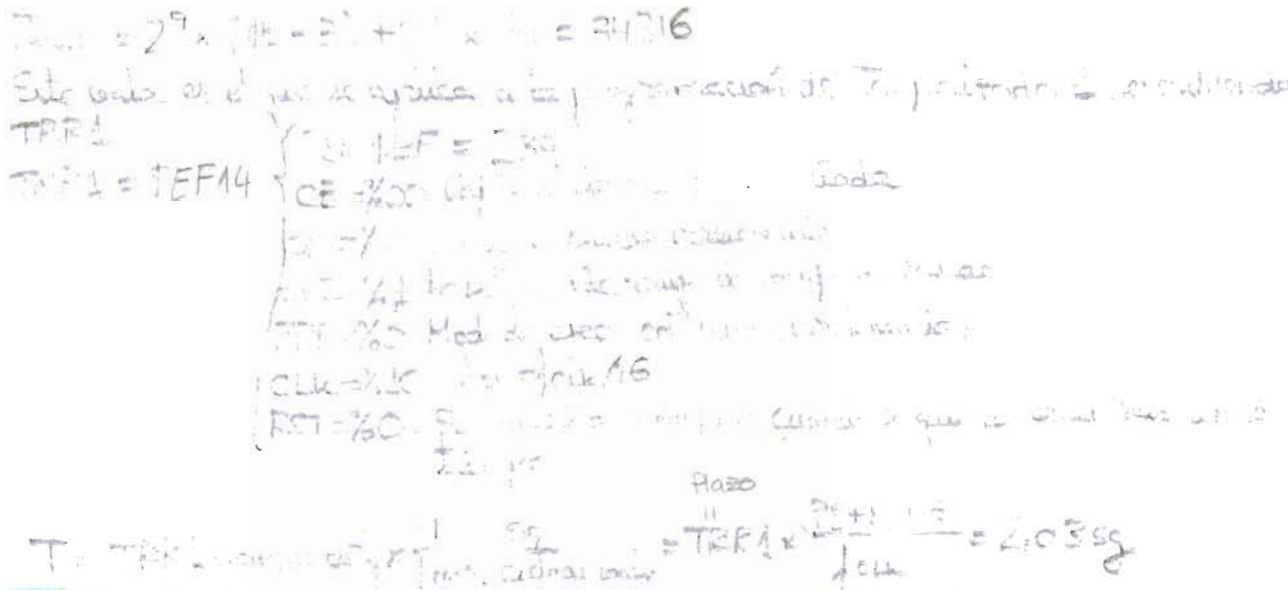
```
CAP_ENTRADA     MOVE.L      D0,-(A7)
                BSET.B      #0,TERO_L(A0)
                TST.B       Seco
                BEQ         ACTIVA_VALV
                CLR.B       Seco
                BCLR        #5,PBDAT_L(A0)
                MOVE.W      TCAP,D0
                MOVE.L      #1000,D1
                DIVU.L      D1,D0
                BRA         SALIR3

ACTIVA_VALV     BSET        #5,PBDAT_L(A0)
                CLR.W       TCN1
                MOVE.B      #1,D0
                MOVE.B      D0,Seco
                MOVE.L      #SIGUE_SECO,D0
                MOVE.L      D0,Retardo
SALIR3          MOVE.L      (A7)+,D0
                RTE
```

3.9. ¿Por qué no hace falta hacer nada si se produce un desbordamiento del contador TCN1 utilizado para el cálculo del timeout, teniendo en cuenta que el valor de la variable Plazo es siempre menor de 65535?

4. Software

4.1. ¿Cuáles son las fórmulas que se aplican para determinar el tiempo que está abierta la válvula (Plazo) y el grado de apertura de la válvula (Apertura) en función de las variables Humedad, y Temperat y el valor Programa leído en PADAT[5:4]?

4.2. **Si Humedad=$3, Temperat=$E, PADAT4=0 y PADAT5=1, ¿cuánto vale Plazo y cuánto tiempo tardaría en cerrarse la válvula principal?**



4.3. **¿En qué estado queda el sistema después de activarse Alarma_grave? ¿En qué condiciones se saldrá de dicho estado?**

Se queda bloqueado en un bucle usando en la rutina OPERACIÓN esperando a que un operador pulse el botón de "Parada de apertura"

Ejercicio 14. Sistema digital de medición basado en ultrasonidos

Una empresa ha desarrollado un sistema digital portátil de medición basado en un MCF5272 a 66 MHz. destinado a uso inmobiliario. El HW del sistema se muestra en la siguiente figura. de manera simplificada:

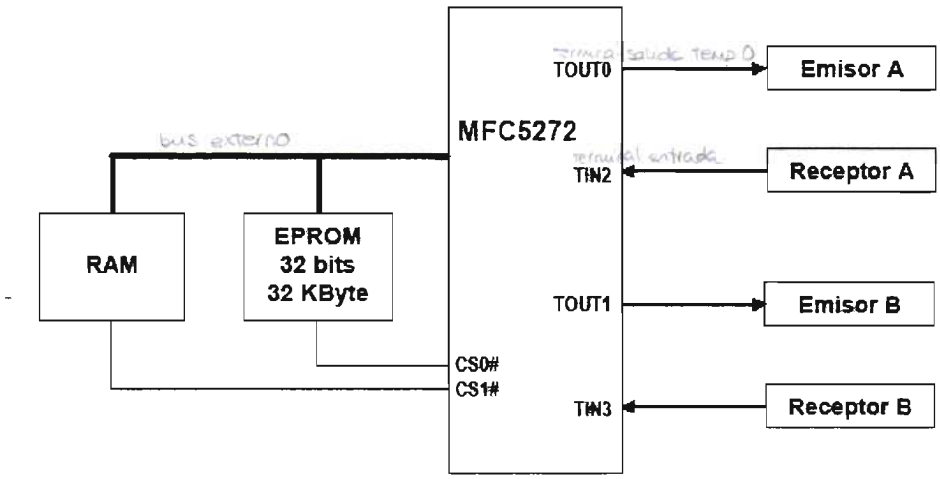


Figura 1. Diagrama de bloques del sistema

Para medir distancias. el sistema emitirá 2 tonos de ultrasonidos de 40 KHz a través de los emisores A y B: estas señales deberán rebotar en paredes opuestas cuya separación se desea medir y ser detectadas por el sistema a través de los receptores A y B respectivamente (esto es lo que se denomina una medición doble): calculando el tiempo que tarda en llegar cada eco en su viaje de ida y vuelta. el sistema puede estimar la distancia entre ambas paredes.

Como se ve en la figura 2. el sistema consta de 2 emisores y 2 detectores de ultrasonidos situados físicamente en lados opuestos del sistema de medición: la distancia total se obtendrá como la suma de las distancias estimadas por medio de ambos de sensores más el ancho del sistema de medición. A fin de incrementar la robustez del sistema ante ruidos. movimientos del medidor. etc.. se realizará un promediado del resultado de varias mediciones dobles. Para la implementación de este sistema se utilizan los temporizadores internos del MCF5272: dos de ellos para generar ondas cuadradas y los otros dos para medir tiempos mediante captura de entrada.

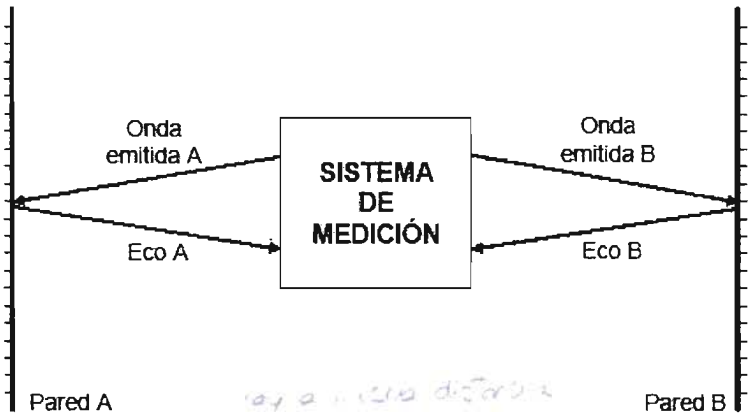


Figura 2. Esquema del proceso de medición.

Listado parcial del programa

```
*****************************************************************************************
*       DEFINICIÓN DE CONSTANTES
*****************************************************************************************
MBAR_MEM        EQU             $100000

ISR             EQU             MBAR_MEM-$30
PITR            EQU             MBAR_MEM-$34
PIVR            EQU             MBAR_MEM-$3F
ICR1            EQU             MBAR_MEM-$20
CSBR0           EQU             MBAR_MEM-$40
CSOR0           EQU             MBAR_MEM-$44
CSBR1           EQU             MBAR_MEM-$48
CSOR1           EQU             MBAR_MEM-$4C
TMR0            EQU             MBAR_MEM-$200
TRR0            EQU             MBAR_MEM-$204
TCAP0           EQU             MBAR_MEM-$208
TCN0            EQU             MBAR_MEM-$20C
TER0            EQU             MBAR_MEM-$210
TMR1            EQU             MBAR_MEM-$220
TRR1            EQU             MBAR_MEM-$224
TCAP1           EQU             MBAR_MEM-$228
TCN1            EQU             MBAR_MEM-$22C
TER1            EQU             MBAR_MEM-$230
TMR2            EQU             MBAR_MEM-$240
TRR2            EQU             MBAR_MEM-$244
TCAP2           EQU             MBAR_MEM-$248
TCN2            EQU             MBAR_MEM-$24C
TER2            EQU             MBAR_MEM-$250

                ORG             $00000000
                DC.L            FINPILA
                DC.L            PRINCIPAL

                ORG             $YYYYYYYY
                DC.L            INI_CAP2
                DC.L            INI_CAP3

*****************************************************************************************
*       PROGRAMA PRINCIPAL
*****************************************************************************************
                ORG             $400

PRINCIPAL       MOVE.L          #MBAR_MEM-$15,D0
                MOVEC           D0,MBAR
                MOVE.L          #$________,D0
                MOVE.L          D0,CSBR0
                MOVE.L          #$________,D0
                MOVE.L          D0,CSOR0
                MOVE.L          #$000100021,D0
                MOVE.L          D0,CSBR1
                MOVE.L          #$FFFF8000,D0
                MOVE.L          D0,CSOR1
BUCLE           BSR             MENU
                BSR             EMISOR_INI
                BSR             MEDIC_ESP
                BSR             MEDIC_VIS
                BRA             BUCLE

*Rutina que comienza el proceso de medición
EMISOR_INI      MOVE.W          #$WWWW,D0
                MOVE.W          D0,TMR0         * Onda cuadrada con el contador parado
                MOVE.W          D0,TMR1
                MOVE.W          #1,D0
                MOVE.W          D0,TRR0
                MOVE.W          D0,TRR1
```

```
                MOVE.W      #$3,D0          * Borrado inicial de los bits de estado
                MOVE.W      D0,TER0
                MOVE.W      D0,TER1
                MOVE.W      #$6342,D0
                MOVE.W      D0,TMR2
                MOVE.W      D0,TMR3
                MOVE.W      #$3,D0          * Borrado inicial de los bits de estado
                MOVE.W      D0,TER2
                MOVE.W      D0,TER3

                MOVEQ.L     #$40,D0
                MOVE.B      D0,PIVR
                MOVE.L      #$000000DD,D0
                MOVE.L      D0,ICR1
                MOVE.W      SR,D0
                ANDI        #$F8FF,D0
                MOVE.W      D0,SR

                CLR.B       CONT_MEDIC
                CLR.B       FIN_MEDIC
                CLR.L       CONT_A
                CLR.L       CONT_B
                MOVE.B      #1,D0
                MOVE.B      D0,CAP_A_ACT
                MOVE.B      D0,CAP_B_ACT
                CLR.L       D0
                BSET.B      D0,TMR0-1
                BSET.B      D0,TMR1+1
                BSET.B      D0,TMR2-1
                BSET.B      D0,TMR3-1
                RTS

* Rutina que espera que termine el proceso completo de mediciones
MEDIC_ESP       TST.B       FIN_MEDIC
                BEQ         MEDIC_ESP
                CLR.B       FIN_MEDIC
                                            * Cálculo promediado de la distancia total
                RTS

* Rutina de la cual se sale cuando el usuario pide que se realice una medición
MENU            ...
                RTS

* Rutina que muestra los resultados y permite guardarlos en memoria
MEDIC_VIS       ...
                RTS

*****************************************************************************
*       RUTINAS DE ATENCIÓN A LAS INTERRUPCIONES
*****************************************************************************

INT_CAP2        MOVE.L      D0,-(A7)
                MOVE.L      D1,-(A7)
                CLR.L       D0
                CLR.L       D1
                BSET.B      D1,TER2-1
                MOVE.W      ICAP2,D0
                MOVE.L      CONT_A,D1
                ADD         D0,D1           * Acumula la medición
                MOVE.L      D1,CONT_A
                CLR.L       D1
                BCLR.B      D1,TMR2+1       * Detiene y pone a cero los contadores
                BCLR.B      D1,TMR0-1
                MOVE.L      #6,D1
                BCLR.B      D1,TMR2-1       * Desactiva la captura de entrada
                CLR.B       CAP_A_ACT
                TST.B       CAP_B_ACT
                BNE         FIN_CAP2
```

```
                CLR.L           D1
                MOVE.B          CONT_MEDIC,D1
                CMP             #4,D1
                BNE             RETARDO2
                MOVE.B          #1,D1
                MOVE.B          D1,FIN_MEDIC
                BRA             FIN_CAP2
RETARDO2        ADDI            #1,D1
                MOVE.B          D1,CONT_MEDIC
                CLR.L           D1
                BSET.B          D1,TMR2-1           * Arranca el contador
ESPERA2         MOVE.W          TCN2,D1
                CMP             #33000,D1
                BNE             ESPERA2
                CLR.L           D1
                BCLR.B          D1,TMR2-1           * Detiene y pone a cero el contador
                MOVE.L          #6,D1
                BSET.B          D1,TMR2-1           * Reactiva la captura de entrada
                BSET.B          D1,TMR3-1
                MOVE.B          #1,D1
                MOVE.B          D1,CAP_A_ACT
                MOVE.B          D1,CAP_B_ACT
                CLR.L           D1
                BSET.B          D1,TMR0-1           * Arranca los contadores
                BSET.B          D1,TMR1-1
                BSET.B          D1,TMR2-1
                BSET.B          D1,TMR3-1
FIN_CAP2        MOVE.L          (A7)+,D1
                MOVE.L          (A7)+,D0
                RTE

*** Rutina como INT_CAP2, cambiando TER2 por TER3, CONT_A por CONT_B, etc.
INT_CAP3        …
                RTE

*******************************************************************************
*       DEFINICIÓN DE VARIABLES
*******************************************************************************
                ORG             $XXXXXXXX           * Zona de variables
CONT_A          DS.L (4)        1
CONT_B          DS.L (4)        1
CONT_MEDIC      DS.B (1)        1
CAP_A_ACT       DS.B (1)        1
CAP_B_ACT       DS.B (1)        -
FIN_MEDIC       DS.B (1)        1
PILA            DS.B            TAMPILA
FINPILA
                END
```

1. Configuración del subsistema de memoria

1.1 De acuerdo con el esquema HW y con el programa proporcionados, especifique cuál debería ser la dirección base de CS0# y determine la configuración de CSBR0 y CSOR0 para el correcto funcionamiento del sistema explicando justificadamente por qué.

Dirección base = S

```
                CLR.L           D1
                MOVE.B          CONT_MEDIC,D1
                CMP             #4,D1
                BNE             RETARDO2
                MOVE.B          #1,D1
                MOVE.B          D1,FIN_MEDIC
                BRA             FIN_CAP2
RETARDO2        ADDI            #1,D1
                MOVE.B          D1,CONT_MEDIC
                CLR.L           D1
                BSET.B          D1,TMR2+1
ESPERA2         MOVE.W          TCN2,D1
                CMP             #33000,D1
                BNE             ESPERA2
                CLR.L           D1
                BCLR.B          D1,TMR2+1
                MOVE.L          #6,D1
                BSET.B          D1,TMR2+1
                BSET.B          D1,TMR3+1
                MOVE.B          #1,D1
                MOVE.B          D1,CAP_A_ACT
                MOVE.B          D1,CAP_B_ACT
                CLR.L           D1
                BSET.B          D1,TMR0+1
                BSET.B          D1,TMR1+1
                BSET.B          D1,TMR2+1
                BSET.B          D1,TMR3+1
FIN_CAP2        MOVE.L          (A7)+,D1
                MOVE.L          (A7)+,D0
                RTE

*** Rutina como INT_CAP2, cambiando TER2 por TER3, CONT_A por CONT_B, etc.
INT_CAP3        ...
                RTE

*************************************************************************************
*       DEFINICIÓN DE VARIABLES (RAM)
*************************************************************************************
                ORG             $XXXXXXXX       * Zona de variables
CONT_A          DS.L            1
CONT_B          DS.L            1
CONT_MEDIC      DS.B            1
CAP_A_ACT       DS.B            1
CAP_B_ACT       DS.B            1
FIN_MEDIC       DS.B            1
PILA            DS.B            TAMPILA
FINPILA
                END
```

* Tras la primera medida CONT_MEDIC=0
" " segunda " " =1

→ Cuando CONT_MEDIC=4 (tras 5 medidas) → FIN_MEDIC=
y ya no hace más medidas para promediar

Si CONT_MEDIC < 4 espera un tiempo antes de realizar las 2 nuevas medidas A y B

* Arranca el contador

Espera 33000 ciclos de contador del temp 2

$T_{espera} = 33000 \text{ ciclos de cont} \times T_{CONT_2}\left(\frac{ss}{\text{ciclo de contador}}\right)$

* Detiene y pone a cero el contador

* Reactiva la captura de entrada
CE = %01

* Arranca los contadores

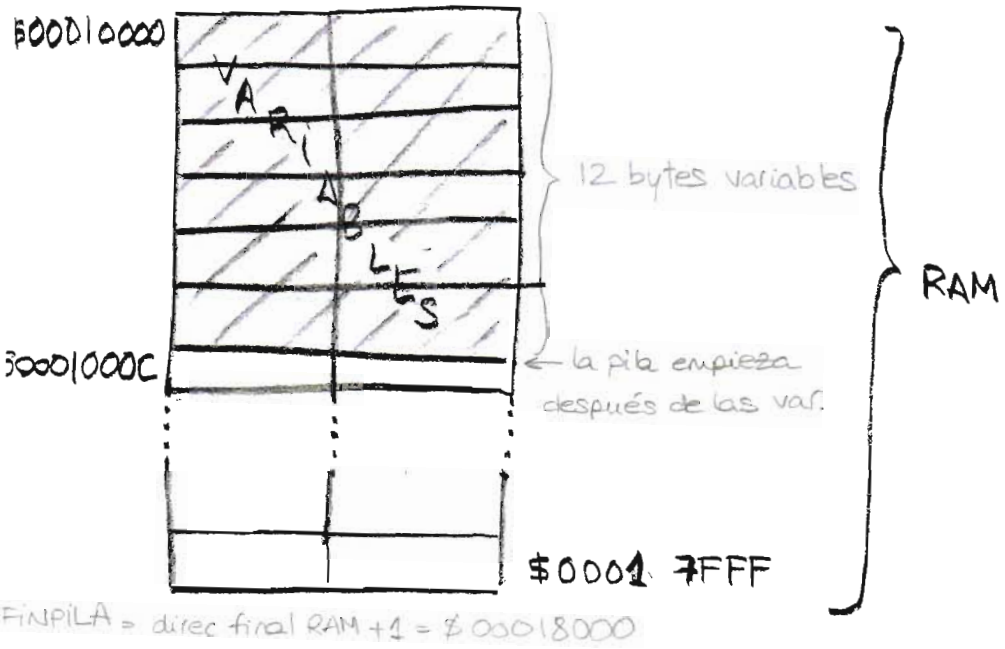
Se puede sacar con la configuración del chip select correspondiente, en este caso CSBR1.

1. Configuración del subsistema de memoria

1.1 De acuerdo con el esquema HW y con el programa proporcionados, especifique cuál debería ser la dirección base de CS0# y determine la configuración de CSBR0 y CSOR0 para el correcto funcionamiento del sistema explicando justificadamente por qué.

Dirección base = $ 0000 0000

CS0# está conectado a la memoria ROM que contiene la tabla de vectores de excepción y el programa. Como el vector de la excepción RESET que contiene el valor inicial de A7 y PC está en la dirección 0, la dirección base debe ser $0000 0000



Tampila (máxima) = Direcc. final − Dirección inicial + 1 =

= $0001 7FFF − $0001000C + $1 =

= $0001 8000 − $0001000C = $8000 − $C = $7FF4

$f_{señal} = f_{cuadrada} = \frac{f_{CONT}}{2 \cdot TRR_n}$ → $\frac{f_{CLK}/16}{PS+1}$

$\frac{1}{2 f_{señal}} = TRR_n \times \frac{(PS+1)\cdot 16}{f_{CLK}}$ Sabemos que $\begin{cases} f_{señal} = 40\ kHz\ (enunciado) \\ TRR_n = 1\ (código)\ \ n=0,1 \end{cases}$

Despejamos $PS = \frac{f_{CLK}}{2 f_{señal} \cdot TRR_n \cdot 16} - 1 = \frac{66 \cdot 10^6}{2 \cdot 40 \cdot 10^3 \cdot 1 \cdot 16} - 1 = 51,5$ ←

$PS = 51 = \$33$

CSBR0 = $0000 0001

BA=$00000. Dirección base $00000000 para la ROM

EBI=%00. Para SRAM o ROM de 16/32 bits (ver dibujo del sistema)

BW=%00. Anchura de bus de 32 bits

SUPER=%0 Activo siempre

ENABLE=%1. Memoria ROM habilitada

CSOR0 = $ FFFF8001

BAM=$FFFF8. 32 kB = 2^{15}. Se comparan los 32-15=17 bits superiores

WS=%0 0000 Sin estados de espera

R/W=%0 Memoria de sólo lectura

MRW=%1. Lectura o escritura según indique R/W

1.2 Teniendo en cuenta el SW y el HW descritos, determine justificadamente los valores de la constante $XXXXXXXX, y el valor máximo que puede tener TAMPILA con esta configuración de memoria.

XXXXXXXX = $0001 0000 (BA de CSBR1)

La inicialización de CSBR1 con el valor $00010021 indica que la dirección base de la RAM es la $10000

TAMPILA (máximo) = $7FF4

Por la configuración de CSOR1 se sabe que el tamaño de la RAM es de 32 kB = $= 2^{15} = \$8000$. Como la variables están colocadas al comienzo de la RAM, el tamaño máximo de la pila será el tamaño de la RAM menos las posiciones ocupadas por las variables (12 bytes). $8000 - $C = $7FF4

2. Configuración del módulo de mediciones

2.1 Para poder emitir el tono de ultrasonidos, cada emisor necesita recibir del temporizador una señal digital cuadrada de 40 KHz; si no se recibe esta señal, no se emite nada. Determine el valor SWWWW con el que hay que configurar los registros TMR0 y TMR1 si se desea usar sólo la comparación de salida en modo de reinicio con el reloj del sistema dividido por 16, que el contador todavía no empiece a contar, y que no haya interrupciones. Justifique detalladamente el valor del campo de PREESCALADO.

SWWWW = $ 332C

RST=%0 El contador todavía no empieza a contar

CLK=%10. La fuente es el reloj del sistema dividido por 16 ($f_{IN} = f_{CLK}/16$)

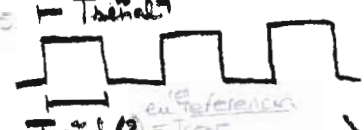
FRR=%1. Modo reinicio

ORI=%0. No se generan interrupciones de comparación de salida

OM=%1. Modo de salida: conmuta el nivel de salida TOUTn (para generar señal cuadrada)

CE=%00. Captura de entrada inhabilitada

$f_{señal} = 40 kHz \rightarrow T_{señal} = \frac{1}{f_{señal}}$



Conmuta cada $\frac{T_{señal}}{2}$

Así que $\frac{T_{señal}}{2} = TRR_n\ (ciclos\ de\ cont) \times T_{CONT}\left(\frac{s}{ciclo\ de\ cont}\right) = TRR_n \times \frac{1}{f_{CONT}}$

Sabemos que $f_{CONT} = \frac{f_{IN}}{PS+1} = \frac{f_{CLK}/16}{PS+1} \Rightarrow \frac{T_{señal}}{2} = TRR_n \times \frac{(PS+1)16}{f_{CLK}}$

2.2 Teniendo en cuenta la configuración en la rutina EMISOR_INI, determine qué dispositivos tienen habilitadas las interrupciones y con qué niveles de prioridad. Determine además el valor de SYYYYYYYY para un correcto funcionamiento del sistema. Justifique brevemente sus respuestas.

ICR1 se inicializa al valor $0000 00DD ⇒ { TIMER2 IPL = %101 = 5 ; TIMER3 IPL = %101 = 5 }

Interrupciones de TIMER2 y TIMER3 habilitadas con nivel 5

SYYYYYYYY = $0000011C

PIVR se inicializa con valor 64 = $40. Los vectores de interrupción de TIMER2 y TIMER3 tienen un offset de 7 y 8 respect. ⇒ El número de vector de TIMER2 es 64+7 = =71. $YYYY YYYY corresponde a la dirección del vector de la interrupción TIMER2

Dirección del vector 71 = VBR + (4×71) = 0 + 284 = $11C

3. Análisis del software

3.1 Analizando el programa, explique justificadamente cuántas mediciones dobles (distancia a ambas paredes) se realizan para determinar la distancia real entre dos paredes.

Se realizan 5 mediciones dobles. En la rutina INT_CAP2 o INT_CAP3 tras cada medición doble se compara CONT_MEDIC con 4 y luego se incrementa. Tras la primera medición CONT_MEDIC vale 0, luego tendrán que hacerse otras 4 mediciones dobles hasta que valga 4 ⇒ En total 5 mediciones.

3.2 ¿Cuánto tiempo espera el sistema entre el final de una doble medición y el comienzo de la siguiente? El tiempo de espera corresponde a 33000 ciclos del contador 2 (o el 3)

Es decir $T_{espera} = 33000 \times T_{CONT_2} = 33000 \times \frac{1}{f_{CONT_2}}$ y $f_{CONT2} = \frac{f_{INT_2}}{PS_2+1} = \frac{f_{CLK}}{PS_2+1}$

Así que $T_{espera} = 33000 \times \frac{PS_2+1}{f_{CLK}} = 33000 \times \frac{99+1}{66\,10^6} = 0{,}05\ sg = 50\ ms$

viendo valor de TMR2 = 36242 → reloj del sistema, se ve en el valor 36242 de TMR2 (99)

3.3 Comente qué efectos podría tener suprimir la instrucción CLR.L D0 en la rutina INT_CAP2.

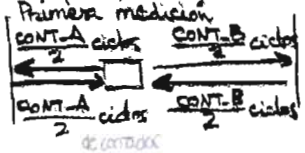
El contador de tiempos es de 16 bits, pero como hay que realizar 5 mediciones y promediar, las variables CONT_A y CONT_B tienen que ser de 32 bits. D0 se emplea para guardar el valor de TCAP2 o TCAP3, valor que se suma en modo .L con el valor de las medidas previas acumuladas ⇒ Hay que borrarlo previamente en modo .L para que lo que hubiera en los 16 bits más significativos de D0 no afecte en la suma.

3.4 Teniendo en cuenta el SW y el HW proporcionados, que la velocidad de transmisión de los ultrasonidos en el aire es igual a V m/seg, que se desprecia cualquier efecto debido al tiempo de ejecución de las instrucciones o a posibles limitaciones del HW del emisor y del receptor, y que la señal debe viajar hacia cada pared y volver, escriba y justifique la fórmula que se usaría en la rutina MEDIC_ESPERAR para calcular la distancia en metros a partir de los valores de CONT_A y CONT_B, si la frecuencia de los contadores 2 y 3 fuese de F Hz, el tamaño del medidor fuese M cm y se realizaran D mediciones dobles.

$$Distancia = V\left(\frac{m}{s}\right) \times \left(\frac{CONT\_A + CONT\_B}{2}\right)(ciclos\ de\ contador) \times \frac{1}{F}\left(\frac{s}{ciclo\ de\ contador}\right) + \frac{M}{100}(m) =$$

$$= \frac{V \times (CONT\_A + CONT\_B)}{2DF} + \frac{M}{100}$$

(2D)



con A: 5 viajes de ida y vuelta

CONTB = 5 viaje de ida y vuelta

CONTA/2D = 5 viaje de ida o vuelta ⇒ la distancia en

3.5 Explique por qué se incluyen las instrucciones TST.B CAP_B_ACT y BNE FIN_CAP2 en la rutina INT_CAP_2.

Como el cálculo de la distancia implica dos cálculos independientes, una vez se ha recibido un eco (por ejemplo, el asociado a CAP_A_ACT) hay que comprobar si ya se recibió previamente el otro (consultando CAP_B_ACT). si se recibió, es que se ha completado una doble medición y hay que comprobar si se han completado las 5 mediciones dobles que promediar; si no se recibió el otro eco, esta interrupción se acaba y hay que esperar el fin de la otra medición.

↳ 1ª parte: almacena en el valor de cuenta la medida

2ª parte: la realiza el temporizador mayor, el A está a mayor distancia del sist. de medición

Ejercicio 15. Sistema para el cálculo de estadísticas en partidos de fútbol

Se desea realizar un sistema con el que se determinen los tiempos de posesión del balón de cada uno de los equipos en un partido de fútbol. Se trata de un sistema basado en un MCF5272 a 50 MHz que incluye los bloques de la Figura 1 y en el que se van a utilizar los temporizadores del MCF5272 para llevar a cabo las medidas.

El comentarista dispone de dos pulsadores: el pulsador 1 (P1) para indicar el comienzo y el final del partido y el pulsador 2 (P2) para indicar que hay un cambio en la posesión del balón. Por simplicidad se va a considerar que el partido tiene una sola parte. P1 va conectado a la entrada INT1 del MCF5272, de forma que se genere una interrupción cada vez que se presione dicho pulsador. P2 va conectado al terminal de captura de entrada TIN0, de modo que también genere una interrupción, en este caso cada vez que haya un cambio en la posesión del esférico. El objetivo es aprovechar la funcionalidad de la captura de entrada del temporizador 0 para medir el tiempo que cada equipo controla el balón.

Al comenzar el partido, el comentarista presiona el pulsador P1, momento en el cual el sistema lee los parámetros de configuración del sistema en el Puerto A (fijados previamente por el comentarista). Estos parámetros son dos números de 5 bits que identifican a los equipos contendientes entre todos los equipos del campeonato.

- PA0-PA4: identificador del primer equipo
- PA5-PA9: identificador del segundo equipo

El programa principal envía periódicamente, cada 10 minutos, la información estadística (identificadores de los equipos seguidos del porcentaje de posesión del balón de cada equipo) por el puerto serie a un dispositivo remoto donde se visualizan los resultados. Se va a utilizar el temporizador 0 del MCF5272 configurado como reloj en tiempo real para llevar el control de dicho tiempo. La línea serie ofrece una comunicación asíncrona a 9600 baudios. Cada byte se envía precedido por un bit de arranque y seguido por un bit de paridad par y dos bits de parada.

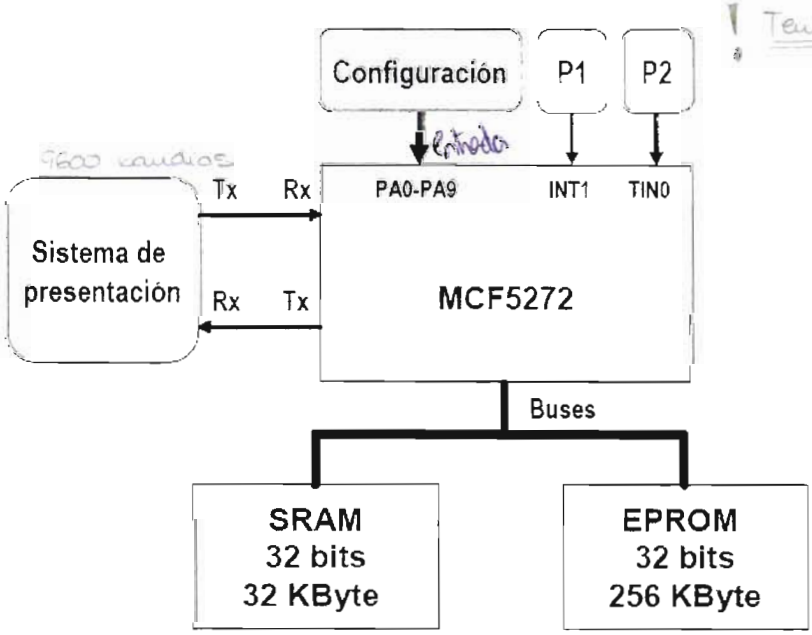


Figura 1. Diagrama de bloques del sistema

Interfaz serie

La UART1 del MCF5272 tiene la misión de enviar los datos calculados al sistema de presentación. En este ejercicio se supone que el MCF5272 dispone de una UART1 simplificada en lugar de la estándar, que emplea el mismo vector de interrupción que la estándar pero que contiene los siguientes registros:

	7	6	5	4	3	2	1	0
XCR	-	BR		DT	ST	PR	TX	RX

- **XCR**: Registro de control

  BR: velocidad en baudios (00≡1200, 01≡4800, 10≡9600, 11≡19200)
  DT: número de bits de datos (0=7 bits, 1=8 bits)
  ST: número de bits de parada (0=1 bit; 1=2 bits)
  PR: tipo de paridad (0=par; 1=impar)
  TX: interrupción de Tx habilitada (1) o inhabilitada (0)
  RX: interrupción de Rx habilitada (1) o inhabilitada (0)

	7	6	5	4	3	2	1	0
XSR	-	-	-	OV	FR	PR	TX	RX

- **XSR**: Registro de estado

  OV: error de desbordamiento (OV=1): al escribir un 1, se pone a cero el flag
  FR: error de trama recibido (FR=1): al escribir un 1, se pone a cero el flag
  PR: error de paridad recibido (PR=1): al escribir un 1, se pone a cero el flag
  TX: dato transmitido (TX=1): al escribir un 1, se pone a cero el flag
  RX: dato recibido (RX=1): al escribir un 1, se pone a cero el flag

	7							0
XDAT								

- **XDAT**: Registro de datos

  Cuando se escribe, se escribe en el registro interno de transmisión. Cuando se lee, se lee el registro interno de recepcion

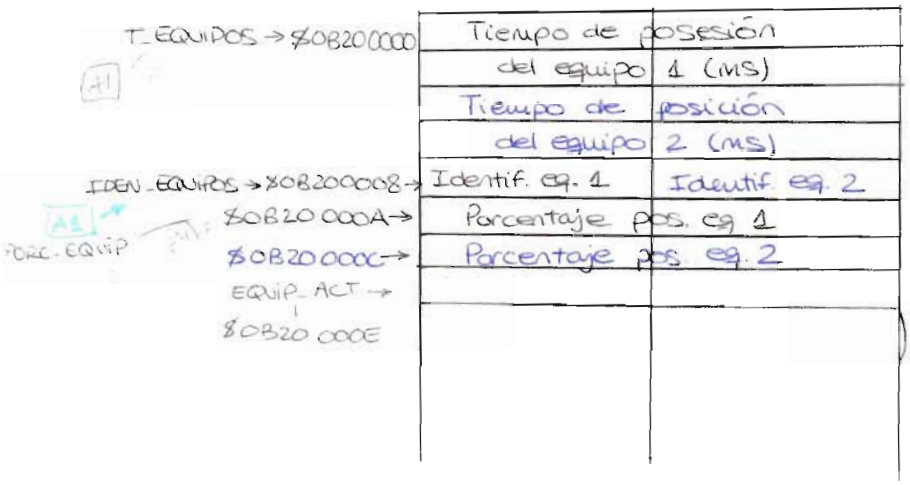
En el sistema bajo estudio, la UART1 debe generar interrupciones de nivel 2 asociadas a la transmisión de bytes por la linea serie. La interrupción asociada a INT1 es de nivel 6 y la interrupción del temporizador 0 es de nivel 3.

Listado parcial del programa

```
;*****************************************************************************
;       Tabla de vectores de excepcion
;*****************************************************************************
                ORG             0
                DC.L            $wwwwwwww
                DC.L            MAIN

                ORG             $xxxxxxxx
                DC.L            RTI_INT1
                ORG             $yyyyyyyy
                DC.L            TIMER0
                ORG             $zzzzzzzz
                DC.L            INT_UART1
```

```
*****************************************************************************
*       Definición de constantes y de variables
*****************************************************************************
MBAR_MEM        EQU             $00C00000
PACNT           EQU             $80             ; Puerto A: control
PADDR           EQU             $84             ; Puerto A: dirección
PADAT           EQU             $86             ; Puerto A: datos
PIVR            EQU             $3F
PITR            EQU             $34
ICR1            EQU             $20
ICR2            EQU             $24
CSBR0           EQU             $40
CSOR0           EQU             $44
CSBR1           EQU             $48
CSOR1           EQU             $4C
TMR0            EQU             $200
TMR0_L          EQU             $201
TRR0            EQU             $204
TCAP0           EQU             $208
TCN0            EQU             $20C
TER0_L          EQU             $211
XCR             EQU             $1000
XDAT            EQU             $1001
XSR             EQU             $1002

ENVIO           EQU             $KKKK
NUM_EQUIPOS     EQU             2

                ORG             $0B200000
T_EQUIPOS       DS.L            NUM_EQUIPOS     ; Tiempos acumulados de posesión en ms
IDEN_EQUIPOS    DS.B            NUM_EQUIPOS     ; Identificadores de los equipos
PORC_EQUIP      DS.W            NUM_EQUIPOS     ; Porcentajes de posesión
EQUIP_ACT       DS.B            1
T_ACTUAL        DS.L            1
T_ANTERIOR      DS.L            1
FIN_TIMER       DS.B            1
CONT_TRAMA      DS.L            1               ; Contador de bytes a enviar
DIR_ENVIO       DS.L            1               ; Dirección del byte a enviar
ESTADO_INT1     DS.B            1
NUM_DESBORD     DS.L            1
RETARDO         DS.L            1
*****************************************************************************
*       Programa principal
*****************************************************************************
                ORG             $400
MAIN            MOVE.L          #MBAR_MEM+$15,D0
                MOVEC           D0,MBAR
                LEA.L           MBAR_MEM,A0
                MOVE.L          #$00000001,D0
                MOVE.L          D0,CSBR0(A0)
                MOVE.L          #$_________,D0
                MOVE.L          D0,CSOR0(A0)
                MOVE.L          #$0B200001,D0   ; Inicialización del CS de la RAM
                MOVE.L          D0,CSBR1(A0)
                MOVE.L          #$_________,D0
                MOVE.L          D0,CSOR1(A0)
                BSR             SW_INI
                BSR             HW_INI

BUCLE_PPAL      TST.B           FIN_TIMER
                BEQ             BUCLE_PPAL
                CLR.B           FIN_TIMER
                PEA             T_EQUIPOS
                BSR             CALCULOS
                ADDA.L          #$MM,A7
                BSR             ENVIO_SERIE
                BRA             BUCLE_PPAL
```

Variables de la memoria 2AM



```
*******************************************************************************
*       Rutinas de inicialización del sistema
*******************************************************************************
SW_INI          CLR.B           EQUIP_ACT
                CLR.L           T_ACTUAL
                CLR.L           T_ANTERIOR
                CLR.B           FIN_TIMER
                CLR.L           CONT_TRAMA
                CLR.L           NUM_DESBORD
                CLR.B           ESTADO_INT1
                LEA             T_EQUIPOS,A1
                CLR.L           (A1)+
                CLR.L           (A1)
                MOVE.L          #ENVIO,D0
                MOVE.L          D0,RETARDO
                RTS

HW_INI          CLR.L           PACNT(A0)       ; Configuración de los puertos
                MOVE.W          #$0000,D0
                MOVE.W          D0,PADDR(A0)

                MOVE.W          #$7C5C,D0       ; Configuración del temporizador 0
                MOVE.W          D0,TMR0(A0)
                MOVE.W          #$3004,D0
                MOVE.W          D0,TRR0(A0)
                CLR.W           TCN0(A0)
                BSR             INTERR_INI
                MOVE.B          #$YY,D0         ; Configuración del puerto serie (UART)
                MOVE.B          D0,XCR(A0)
                MOVE.B          #$ZZ,D0
                MOVE.B          D0,XSR(A0)
                RTS

INTERR_INI      MOVEQ.L         #$60,D0                 ; Inicialización de PIVR
                MOVE.B          D0,PIVR(A0)
                MOVE.L          #$LLLLLLLL,D0           ; Inicialización de ICR1
                MOVE.L          D0,ICR1(A0)
                MOVE.L          #$A0000000,D0           ; Inicialización de ICR2
                MOVE.L          D0,ICR2(A0)
                MOVE.L          #$80000000,D0           ; Inicialización de PITR
                MOVE.L          D0,PITR(A0)
                CLR.L           D0
                MOVE.W          SR,D0                   ; Habilitación de las
                ANDI            #$F8FF,D0               ; interrupciones en SR
                MOVE.W          D0,SR
                RTS

*******************************************************************************
*       Rutina de cálculo del porcentaje del tiempo de posesión de cada equipo
*******************************************************************************
CALCULOS        MOVEA.L         offset1(A7),A1          ; Dirección inicial
                CLR.L           D0
                MOVE.L          (A1)+,D0
                MOVE.L          (A1)+,D1
                MOVE.L          D1,D2
                ADD.L           D0,D2
                MOVE.L          #100,D3
                MULU            D3,D0
                DIVU.L          D2,D0
                MULU            D3,D1
                DIVU.L          D2,D1
                ADDA.L          #NUM_EQUIPOS,A1
                MOVE.W          D0,(A1)+
                MOVE.W          D1,(A1)+
                RTS
```

```
*******************************************************************************
*       Rutina de envío por línea serie
*******************************************************************************
ENVIO_SERIE   MOVE.L      #(EQUIP_ACT-IDEN_EQUIPOS-1),D0
              MOVE.L      D0,CONT_TRAMA
              LEA.L       IDEN_EQUIPOS,A1
              MOVE.B      (A1)+,XDAT(A0)
              MOVE.L      A1,DIR_ENVIO

ESPERA        TST.L       CONT_TRAMA          ; ¿Ha terminado la transmisión?
              BNE         ESPERA
              RTS

*******************************************************************************
*       Rutinas de servicio a las interrupciones
*******************************************************************************
*** Rutina de control del temporizador 0
TIMER0        ADDA.L      #-16,A7
              MOVEM.L     D0-D2/A1,(A7)
              BTST.B      #0,TER0_L(A0)
              BNE         CAP_ENTRADA
              BSET.B      #1,TER0_L(A0)
              ADDQ.L      #1,NUM_DESBORD
              SUBQ.L      #1,RETARDO
              BNE         SALIR0
              MOVE.B      #1,D0
              MOVE.B      D0,FIN_TIMER
              MOVE.L      #ENVIO,D0
              MOVE.L      D0,RETARDO
              BRA         SALIR0

CAP_ENTRADA   BSET.B      #0,TER0_L(A0)
              CLR.L       D1
              MOVE.L      T_ACTUAL,D0
              MOVE.L      D0,T_ANTERIOR
              MOVE.W      TCAP0,D1
              MOVE.W      D1,T_ACTUAL
              SUB.L       D0,D1
              ...                             ; Conversión de D1 a mseg

              LEA         T_EQUIPOS,A1        ; Actualización de los tiempos
              CLR.L       D0
              MOVE.B      EQUIP_ACT,D0
              MOVE.L      0(A1,D0*4),D2
              ADD.L       D1,D2
              MOVE.L      D2,0(A1,D0*4)
              CLR.L       NUM_DESBORD
              TST.B       D0
              BEQ         EL_2
              CLR.B       EQUIP_ACT
              BRA         SALIR0
EL_2          MOVE.B      #1,D2
              MOVE.B      D2,EQUIP_ACT
SALIR0        MOVEM.L     (A7),D0-D2/A1
              ADDA.L      #16,A7
              RTE

*** Rutina de interrupción del puerto serie
INI_UART1     MOVE.L      A1,-(A7)            ; Salvar el contexto
              BSET.B      #1,XSR(A0)
              LEA.L       DIR_ENVIO,A1
              MOVE.B      (A1)+,XDAT(A0)
              MOVE.L      A1,DIR_ENVIO
              SUBQ.L      #1,CONT_TRAMA
              MOVE.L      (A7)+,A1            ; Recuperar el contexto
              RTE
```

```
*** Rutina de la interrupción INT1
RUT_INT1        MOVE.L      D0,-(A7)            ; Salvar el contexto
                MOVE.L      A1,-(A7)
                MOVE.L      ICR1(A0),D0         ; Poner a cero el bit de INT1PI
                ORI.L       #$80000000,D0
                MOVE.L      D0,ICR1(A0)
                TST.B       ESTADO_INT1
                BNE         FIN_CALCULOS
                CLR.L       D0
                BSET.B      D0,TMR0_L(A0)
                BSET.B      D0,ESTADO_INT1
                CLR.L       D0                  ; Leer identificadores
                LEA         IDEN_EQUIPOS,A1
                MOVE.W      PADAT(A0),D0
                AND.L       #$001F,D0
                MOVE.B      D0,(A1)+
                MOVE.W      PADAT(A0),D0
                AND.L       #$03E0,D0
                LSR         #5,D0
                MOVE.B      D0,(A1)+
                BRA         SALIR3
FIN_CALCULOS    MOVE.B      #1,D0
                MOVE.B      D0,FIN_TIMER        ; Al terminar, enviar resultados
                CLR.L       D0
                BCLR.B      D0,TMR0_L(A0)
                BCLR.B      D0,ESTADO_INT1
SALIR3          MOVE.L      (A7)+,A1            ; Recuperar el contexto
                MOVE.L      (A7)+,D0
                RTE
                END
```

1. Configuración del subsistema de memoria

1.1. Complete la siguiente tabla con el mapa de memoria que se deduce a partir de la información del enunciado y del listado.

RANGO DE DIRECCIONES (HEX)	DISPOSITIVO ASIGNADO
$00000000 - $0003FFFF	Memoria EPROM 256K
$0B200000 - $0B207FFF	Memoria RAM 32K
$00C00000 - $00C0FFFF	Módulo SIM 64K SIEMPRE!

1.2. Teniendo en cuenta el listado, las especificaciones del sistema y que la RAM y la ROM necesitan dos estados de espera, configure los registros CSORn. Justifique cada uno de los campos de los registros.

CSOR0 = $FFFC0009

BAM = $FFFC0. 256 kB = 2^{18}. Se comparan los 32-18 = 14 bits superiores

WS = %00010. Se usan dos estados de espera

RW = %0. Memoria de solo lectura

MRW = %1. Lectura o escritura indicado por RW

CSOR1 = $FFFF8008

BAM = $FFFF8. 32 kB = 2^{15}. Se comparan los 32-15 = 17 bits superiores

WS = %00010. Se usan dos estados de espera

RW = %0. Indiferente

MRW = %0. De lectura/escritura

2. Pila y software

2.1. Especifique justificadamente el valor de $WWWWWWWW para que el tamaño disponible para la pila sea máximo de acuerdo a las especificaciones del sistema.

$WWWWWWWW = $0B208000

Las variables están al comienzo de la RAM. Luego para que la pila sea del tamaño máximo, hay que inicializar A7 como dirección final de la RAM+1 =
= $0B207FFF + 1 = $0B208000

2.2. ¿Qué debe valer offset1 (en la subrutina CALCULOS)? ¿Cuál debe ser el valor de $MM en la instrucción ADDA.L #$MM,A7 del programa principal? Justifique su respuesta.

offset1 = 4

offset para acceder al parámetro pasado por la pila a la subrutina, que es la dirección de T_EQUIPOS

$MM = $4

A la vuelta de la llamada a CALCULOS hay que dejar el puntero de pila apuntando a la dirección original, para lo que se suma 4 que es el número de bytes ocupado por el parámetro pasado, es decir, la dirección de T_EQUIPOS

2.3. Si al terminar el partido, el contenido de la dirección $0B200000 es $001B7740 y el de la dirección $0B200004 es $000DBBA0, justifique qué valores se guardarían en las posiciones $0B20000A (word) y $0B20000C (word) en la subrutina CALCULOS.

$0B20000A = $42

$0B20000C = $21

En la palabra .L a partir de $0B200000 se almacena el tiempo de posesión del equipo 1 (ms)
" " " .L " " " $0B200004 " " " " " " " " 2 (ms)
" " " .W " " " $0B20000A " " " porcent de posesión del eq. 1
" " " .W " " " $0B20000C " " " " " " " 2

El cálculo se realiza en CALCULOS como Porcentaje equipo i = (Tiempo de posesión eq i) x100 / (Tiempo pos. eq 1 + Tiempo pos eq 2)

Como el tiempo de posesión del eq 1 es $001B7740 = 1800000 ms
y " " " " " " 2 es $000DBBA0 = 900000 ms

Porcentaje pos eq 1 = 66 = $42 y Porcentaje pos eq 2 = 33 = $21

3. Temporización e interrupciones

3.1. Determine los valores $XXXXXXXX, $YYYYYYYY y $ZZZZZZZZ para que las interrupciones del sistema funcionen correctamente. Justifique su respuesta. Direcciones de los vectores de interrupción

$XXXXXXXX = $184 = VBR + 4x($60+1) (INT1 tiene offset 1)
$YYYYYYYY = $194 = VBR + 4x($60+5) (THR0 tiene offset 5)
$ZZZZZZZZ = $1A4 = VBR + 4x($60+9) (UART1 tiene offset 9)

VBR = $0
PIVR se inicializa con $60

3.2. Determine el valor inicial del registro de control ICR1 ($LLLLLLLL) en la rutina INTERR_INI para que todas las interrupciones estén adecuadamente configuradas. Justifique su respuesta.

$LLLLLLLL = $E000B000

INT1 IPL = %110 = 6. Nivel de la interrupción INT1

TMR0 IPL = %011 = 3. Nivel de la interrupción TMR0

INT1 IP = %1. Se pone a 1 para cambiar el nivel de la interrupción INT1

TMR0 IP = %1. Se pone a 1 para cambiar el nivel de la interrupción TMR0

¡IMP!

3.3. Determine el valor de $KKKK que hace que se calculen las estadísticas de posesión cada 10 minutos. Justifique su respuesta.

$KKKK = $4B0

Cada vez que se produce la interrupción del temporizador 0 por la función de comparación de salida se decrementa en 1 la variable RETARDO, que se inicializa con el valor ENVIO ($KKKK) que es el número de interrupciones en 10 min. La configuración de dicho temporizador es:

int. period. ⇐

→ TMR0 = $705C

PS = $7C = 124

CE = %01 Captura de flanco de subida + interrupción

OM = %0 Modo de salida (irrelevante)

ORI = %1 Habilita interrupciones de comparación de salida

FRR = %1 Modo reinicio

CLK = %10. $f_{IN} = f_{CLK}/16$

RST = %0 Temporizador inhabilitado

$T_{IRQ} = TRR0 \left(\frac{\text{ciclos cont.}}{\text{int.}}\right) \times T_{CONT} \left(\frac{\text{Sg}}{\text{ciclos cont}}\right)$

$= TRR0 \times \frac{1}{f_{CONT}} \Rightarrow f_{CONT} = \frac{TRR0}{T_{IRQ}}$

$50 \cdot 10^6$ Hz OJO!

→ TRR0 = $30D4 = 12500

Frecuencia de interrup: $f_{IRQ} = \frac{f_{CONT}}{TRR0\,(\text{ciclos de cuenta})} \left(\frac{\text{ciclos de contador}}{\text{Sg}}\right) = \frac{f_{CLK}}{16(PS+1)TRR0} = 2\,\frac{\text{interrup}}{\text{Sg}}$

Así que en 10 min habrá: $10\,\text{min} \times 60\,\frac{\text{Sg}}{\text{min}} \times 2\,\frac{\text{interrup}}{\text{Sg}} = 1200 = 54B0$ interrupciones

3.4. Si el comentarista presionara el pulsador P2 mientras se está atendiendo a la rutina INT_UART1 de atención a la interrupción de la UART1 (justifique sus respuestas):

- **¿Qué rutina de interrupción se ejecutaría?**

Se ejecuta la de mayor prioridad

Al pulsar P2 se solicitaría la interrupción de captura de entrada asociada al temporizador 0. Al ser de nivel 3 mientras que la de la UART1 es de nivel 2, la interrupción sería atendida

- **¿Pueden modificarse entonces los datos que se están transmitiendo?**

los datos son los ident y los porcentajes

No porque en la rutina de atención a la captura de entrada sólo se modifican las variables T_EQUIPS y EQUIP_ACT, que no se transmiten. Al terminar la atención de la captura de entrada continuaría la transmisión de los datos, que no han sido modificados (identificadores de los equipos y sus porcentajes de posesión). Los datos se modificarán cuando se vuelva a entrar en la rutina CALCULOS

y var. EQUIP_ACT

Cada vez q. el comentarista pulsa P2 se modifican los tiempos de posesión pero no los identif y los porcentajes q son los q se transmiten

3.5. Complete justificadamente el código de la rutina de captura de entrada que convierte el valor almacenado en D1 en un tiempo medido en milisegundos.

El temporizador 0 tiene un contador de frecuencia: $f_{CONT} = \frac{f_{CLK}}{16(PS+1)} = \frac{50 \cdot 10^6}{16(124+1)} =$

$= 25000 \frac{ciclos}{sg} \Rightarrow T_{CONT} = 1/25000 = 1/25 \frac{ms}{ciclo\ de\ cont.}$ así que N ciclos de contador son $N(ciclos\ de\ contador) \times \frac{1}{25}\left(\frac{ms}{ciclo\ de\ contador}\right) = N/25$ ms. Como el número de ciclos de contador transcurridos se almacena en D1, para pasarlo a milisegundos hay que hacer D1/25. $(D_1) = ciclos\ cont. \ / \ t(ms) = (D_1) ciclos\ cont \times T_{CONT}\left(\frac{ms}{ciclos\ cont}\right) = \frac{(D_1)}{25}$

Además hay que tener en cuenta NUM_DESBORD, que contiene el número de veces que se ha producido la interrup. del temporizador 0 por comparación de salida (int. periódica) antes de producirse la captura de entrada. En 3.3 se ha visto que $f_{REQ} = 2 \frac{interrup}{sg}$, así que las interrup. periódicas se producen cada 500 ms, así que el tiempo total en ms será: D1/25 + NUM_DESBORD × 500. El código será:

```
MOVE.L #25,D2
DIVS.L D2,D1   *Con signo pues D1 puede ser neg.
MOVE.L NUM_DESBORD,D2
MOVE.L #500,D0
MULU.L D0,D2
ADD.L  D2,D1
```

4. Comunicaciones serie

4.1. Asigne justificadamente los valores adecuados a las constantes $YY y $ZZ usadas en la rutina HW_INI para que se cumplan las condiciones del enunciado.

$YY = $5A (Inicialización del registro XCR)

- BR = %10 Velocidad 9600 baudios
- DT = %1 Se consideran 8 bits de datos
- ST = %1 Se usan 2 bits de parada
- PR = %0 Se usa paridad par
- TX = %1 Interrupción de Tx habilitada
- RX = %0 Interrupción de Rx inhabilitada

$ZZ = $1F (Inicialización del registro XSR)

- OV = %1 Al escribir un 1, se pone a 0
- FR = %1 " "
- PR = %1 " "
- TX = %1 " "
- RX = %1 " "

4.2. ¿Cuántos bytes se transmiten durante la ejecución de la rutina ENVIO_SERIE? Justifique su respuesta.

Se transmiten 6 bytes en total, ya que el contador CONT_TRAMA se inicializa según la resta EQUIP_ACT − IDEN_EQUIPOS − 1 = 5 y además hay un primer byte que se envía desde la rutina ENVIO_SERIE sin decrementar dicho contador (los otros 5 bytes se envían desde la rutina de atención a la interrupción)

Ejercicio 16. Sistema de bloqueo automático de una línea ferroviaria

En las líneas ferroviarias frecuentemente se instalan dispositivos de bloqueo automático que permiten una alta circulación de trenes, sin que se produzcan alcances entre los mismos. Para ello, la vía entre dos estaciones se divide en secciones o cantones, cada uno protegido por un semáforo a su entrada y en el que sólo puede haber un tren como máximo (ver Figura 1).

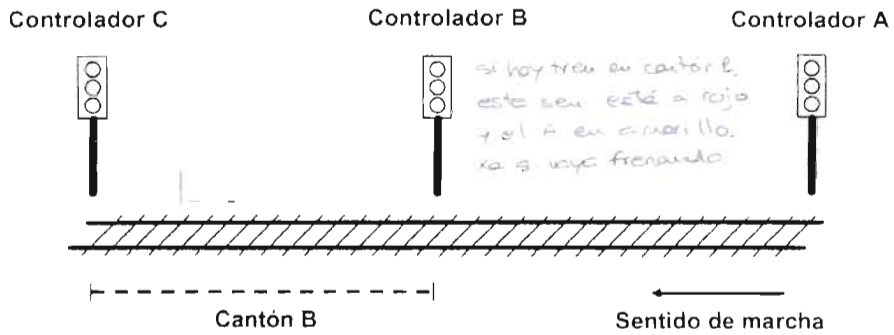
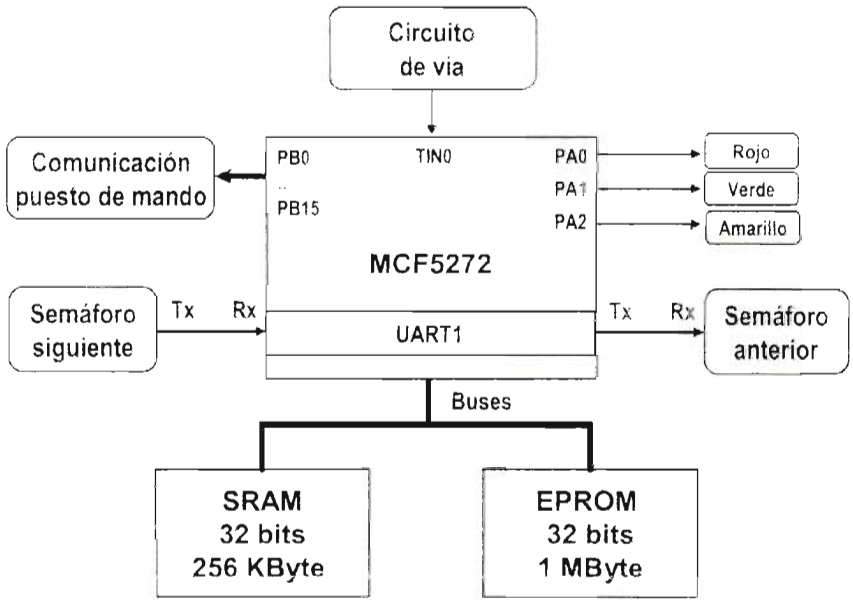


Figura 1. División de la vía en cantones

Cuando en el cantón B se encuentra un tren, el semáforo de entrada al mismo (B) estará en ROJO, no permitiendo el acceso al mismo a un segundo tren. Por otra parte, para evitar detenciones bruscas, si el semáforo B se encuentra en ROJO, el controlador del semáforo B avisará al controlador del semáforo anterior (A) y este cambiará su estado a AMARILLO ("Aviso de parada"). Así un semáforo estará en VERDE ("Vía libre") solamente cuando el cantón siguiente se encuentre vacío y el semáforo siguiente no esté en ROJO.

En este ejercicio se pretende diseñar un circuito que permita controlar uno de estos semáforos, comunicándose con dos circuitos similares, uno que controlará el semáforo anterior, y otro que controlará el siguiente. Por otra parte, cada semáforo consta de una comunicación vía radio con el puesto de mando para monitorizar el estado del sistema. Se trata de un sistema basado en un ColdFire MCF5272 a 60 MHz que incluye los bloques de la Figura 2.



El semáforo puede pasar de verde a rojo directo/.

El sistema utiliza las siguientes señales de entrada y salida, además de las líneas de comunicación serie con los semáforos anterior y siguiente:

- *Circuito de vía:* Entrada digital a través de la cual el circuito de vía informa de si el cantón correspondiente está libre u ocupado. Esta señal permanece a nivel alto mientras el tren esté en el cantón, se detecta mediante captura de entrada y está conectada a la entrada TIN0.
- *Señalización del semáforo:* Tres líneas conectadas al puerto A que activan la luz roja (R ≡ 00000001), la luz verde (V ≡ 00000010), o la luz amarilla (A ≡ 00000100).
- *Comunicación con el puesto de mando:* Salida de 16 bits por el puerto B.

Interfaz serie

La UART1 del MCF5272 se utiliza en este sistema para comunicarse con el semáforo siguiente y el anterior. En este ejercicio se supone que el MCF5272 dispone de una UART1 simplificada en lugar de la estándar, que emplea el mismo vector de interrupción que la estándar pero que contiene los siguientes registros:

	7	6	5	4	3	2	1	0
XCR	-	BR		DT	ST	PR	TX	RX

XCR: Registro de control

BR: velocidad en baudios (00≡1200, 01≡4800, 10≡9600, 11≡19200)
DT: número de bits de datos (0=7 bits; 1=8 bits)
ST: número de bits de parada (0=1 bit; 1=2 bits)
PR: tipo de paridad (0=par; 1=impar)
TX: interrupción de Tx habilitada (1) o inhabilitada (0)
RX: interrupción de Rx habilitada (1) o inhabilitada (0)

	7	6	5	4	3	2	1	0
XSR	-	-	-	OV	FR	PR	TX	RX

XSR: Registro de estado

OV: error de desbordamiento (OV=1); al escribir un 1 se pone a cero el flag
FR: error de trama recibido (FR=1); al escribir un 1, se pone a cero el flag
PR: error de paridad recibido (PR=1); al escribir un 1, se pone a cero el flag
TX: dato transmitido (TX=1); al escribir un 1, se pone a cero el flag
RX: dato recibido (RX=1); al escribir un 1, se pone a cero el flag

	7							0
XDAT								

XDAT: Registro de datos

Cuando se escribe, se escribe en el registro interno de transmisión. Cuando se lee, se lee el registro interno de recepción

En el sistema bajo estudio, la UART1 debe generar interrupciones de nivel 2 asociadas a la recepción de bytes por la línea serie. La interrupción del temporizador 0 es de nivel 4. Además, se utilizará el temporizador 1 en modo comparación de salida para generar una interrupción periódica en tiempo real (nivel 3) que permita monitorizar el sistema y enviar periódicamente el estado del semáforo al semáforo anterior. El controlador generará dos avisos diferentes que comunicará al puesto de mando. El primero de los avisos ocurrirá cuando un tren permanezca un tiempo de 6 minutos en su cantón. El otro aviso se producirá cuando pase un tiempo determinado sin que se reciba comunicación del semáforo siguiente.

UART1 – Int nivel 2 cuando se reciben bytes

Temp 0 – Int nivel 4 – captura de flancos → INT. CAP

Temp 1 – Int periódica nivel 3

Listado parcial del programa

```
*************************************************************************
*       DEFINICIÓN DE CONSTANTES
*************************************************************************
MBAR_MEM        EQU             $XXXXXXXX       direc (inicial) del módulo SIM

ISR             EQU             MBAR_MEM+$30
PITR            EQU             MBAR_MEM+$34
PIVR            EQU             MBAR_MEM+$3F    } reg. control de int
ICR1            EQU             MBAR_MEM+$20
ICR2            EQU             MBAR_MEM+$24

CSBR0           EQU             MBAR_MEM+$40
CSOR0           EQU             MBAR_MEM+$44
CSBR1           EQU             MBAR_MEM+$48    } CS
CSOR1           EQU             MBAR_MEM+$4C

PACNT           EQU             MBAR_MEM+$80
PADDR           EQU             MBAR_MEM+$84
PADAT           EQU             MBAR_MEM+$86    } puertos E/S
PBCNT           EQU             MBAR_MEM+$88
PBDDR           EQU             MBAR_MEM+$8C
PBDAT           EQU             MBAR_MEM+$8E

TMR0            EQU             MBAR_MEM+$200
TRR0            EQU             MBAR_MEM+$204
TCAP0           EQU             MBAR_MEM+$208
TCN0            EQU             MBAR_MEM+$20C
TER0            EQU             MBAR_MEM+$210   } Temporizadores
TMR1            EQU             MBAR_MEM+$220
TRR1            EQU             MBAR_MEM+$224
TCAP1           EQU             MBAR_MEM+$228
TCN1            EQU             MBAR_MEM+$22C
TER1            EQU             MBAR_MEM+$230

XCR             EQU             MBAR_MEM+$1000  }
XDAT            EQU             MBAR_MEM+$1001  } UART
XSR             EQU             MBAR_MEM+$1002  }

L_CANTON        EQU             4                       * Longitud del cantón en Km

                ORG             $00000000   vector de excepción RESET
                DC.L            PINICIAL
                DC.L            PRINCIPAL

                ORG             $YYYYYYYY   dirección de 1 vector de interrupción
                DC.L            INT_CAP     } direc de rutinas de atención a las interrupc
                DC.L            INT_TR      }

                ORG             $00000124   direc de otra rutina de atención a 1 int.
                DC.L            INT_REC     int de UART1

*************************************************************************
*       PROGRAMA PRINCIPAL
*************************************************************************
                ORG             $400    direc comienzo del SIM, se suma cierto valor xq cargar MBAR

PRINCIPAL       MOVE.L          #MBAR_MEM+$15,D0    } carga en reg MBAR
                MOVEC           D0,MBAR
                MOVE.L          #$________,D0
                MOVE.L          D0,CSBR0
                MOVE.L          #$________,D0
                MOVE.L          D0,CSOR0
                MOVE.L          #$________,D0       configurac de los CS
                MOVE.L          D0,CSBR1            se inicializan antes de entrar
                MOVE.L          #$________,D0       a las subrutinas => si no a la
                MOVE.L          D0,CSOR1            pila daría error de bus !!
                BSR             HW_INI
                BSR             SW_INI
```

```
BUCLE           BSR         ENVIO_PMANDO
                BRA         BUCLE

* Rutina que inicializa el hardware
HW_INI          CLR.L       PACNT           * Configuración de los puertos
                MOVE.W      #$0007,D0
                MOVE.W      D0,PADDR

                CLR.L       PBCNT
                MOVE.W      #$FFFF,D0
                MOVE.W      D0,PBDDR

                MOVE.W      #$WWWW,D0       * Configuración de los temporizadores
                MOVE.W      D0,TMR0
                MOVE.W      #$3,D0
                MOVE.W      D0,TER0

                MOVE.W      #$JJJJ,D0
                MOVE.W      D0,TMR1
                MOVE.W      #$61A8,D0
                MOVE.W      D0,TRR1

                MOVE.W      #$3,D0          * Borrado inicial de los bits de estado
                MOVE.W      D0,TER1

                MOVE.B      #$31,D0         * Configuración del puerto serie
                MOVE.B      D0,XCR
                MOVE.B      #$1F,D0
                MOVE.B      D0,XSR

                MOVEQ.L     #$LLLLLLLL,D0   * Config. de las interrupciones
                MOVE.B      D0,PIVR
                MOVE.L      #$ZZZZZZZZ,D0
                MOVE.I      D0,ICR1
                MOVE.L      #$A0000000,D0
                MOVE.I      D0,ICR2
                MOVE.W      SR,D0
                ANDI        #$F8FF,D0
                MOVE.W      D0,SR

                CLR.L       D0
                BSET.B      D0,TMR0+
                BSET.B      D0,TMR1+
                RTS

* Rutina que inicializa las variables del programa
SW_INI          CLR.B       CONT_REC
                CLR.L       CONT_VIA
                CLR.L       CONT_VEL
                CLR.B       VIA_ACT
                CLR.B       EST_SEMAF
                CLR.B       EST_REC
                CLR.B       ERROR
                RTS

* Rutina de envio de parametros al puesto de mando
ENVIO_PMANDO    CLR.L       D0
                MOVE.B      AVISO,D0
                LSL.L       #8,D0
                MOVE.B      EST_SEMAF,D0
                MOVE.W      D0,PBDAT
                TST.L       CONT_VEL
                BEQ         FIN_ENVIO

                BSR         CALCULO_VEL
                                            * Envio de la velocidad del
                                            * tren al puesto de mando
FIN_ENVIO       RTS

* Rutina que calcula la velocidad a la que va el tren en Km/hora
CALCULO_VEL     ...
                RTS
```

No se puede suponer nada, pq en este caso pasa todo el rato interrupciones luego leemos todas las rutinas de atención


```
*****************************************************************************
*       RUTINAS DE ATENCIÓN A LAS INTERRUPCIONES
*****************************************************************************
* Rutina de atención a la interrupción de captura de entrada - Temporizador 0
INT_CAP         ADDA.L          #-12,A7
                MOVEM.L          D0-D1/A0,(A7)
                CLR.L           D0
                CLR.L           D1
                BSET.B          D1,TERO+1       pone a 1 el bit 0 (CAP) => se produce captura de flanco
                                                corresponde a flanco de bajada, cuando mon entra en el cantón
                TST.B           VIA_ACT         mira si la vía está libre o ocupada. Sirve para decidir qué parte de código ejecutar
                BNE             PASA_LIBRE      VIA_ACT=1
                BSET.B          D1,VIA_ACT      se ejecuta si VIA_ACT=0 // pone a 1 el bit 0 de VIA_ACT
                MOVE.W          #$KKKK,D0       * Nueva configuración para
                MOVE.W          D0,TMR0         * captura de flanco de bajada
                CLR.L           CONT_VIA        (antes era el flanco de subida
                CLR.L           CONT_VEL
                BSET.B          D1,EST_SEMAF    en el sto. ← el semáforo podría estar
                BRA             FIN_CAP         EST_SEMAF %0000 0010 VERDE
                                                %0000 0100 AMARILLO
PASA_LIBRE      BCLR.B          D1,VIA_ACT      pone VIA_ACT a cero   1) al hacer BSET pone semáforo a rojo
                MOVE.L          CONT_VIA,D0
                MOVE.L          D0,CONT_VEL
                MOVE.W          #$WWWW,D0       (volver a hacer sistema sensible a flancos de subida)  * De nuevo se configura para
                MOVE.W          D0,TMR0         * captura de flanco de subida
                MOVE.L          EST_REC,D0
                CMPI            #1,D0           (para ver si semáforo siguiente está a rojo, si nuestro cantón
                BEQ             PONER_AMARILLO  esta bien => pone en amarillo)
                MOVE.B          #2,D0           si no está en rojo, lo pone en VERDE
                MOVE.B          D0,EST_SEMAF
                BRA             FIN_CAP
                                (pone actualizado los vbl, antes voy a avisarles del semáforo)

PONER_AMARILLO:
                MOVE.B          #4,D0
                MOVE.B          D0,EST_SEMAF
FIN_CAP         CLR.L           D0
                MOVE.B          EST_SEMAF,D0    extiende el semáforo
                MOVE.W          D0,PADAT        * Actualizar estado del semáforo
                LEA             XDAT,A0         pone en A0 el   * Enviar estado del semáforo al
                MOVE.B          D0,(A0)         sin de nuestro  * controlador del anterior
                MOVEM.L (A7),D0-D1/A0   mensaje al semáforo anterior (a ver consiguiendo el color del semáforo
                ADDA.L          #12,A7          siguiente y si el cantón está ocupado, poner
                RTE                             mirar su color.

* Rutina de interrupción de recepción por línea serie   (pone recibido 1 byte del semáforo siguiente
INT_REC         MOVE.L          D0,-(A7)        con su color
                MOVE.L          A0,-(A7)
                MOVE.L          #0,D0
                BSET.B          D0,XSR          pone en XSR a cero escribiendo un 1
                CLR.L           D0
                MOVE.L          D0,CONT_REC     pone var CONT_REC = 0
                LEA             XDAT,A0         pone en A0 el contenido de XDAT, es decir, el
                MOVE.B          (A0),D0         byte q se transmite
                MOVE.B          D0,EST_REC
                TST.B           VIA_ACT         si VIA_ACT=0 => vía libre
                BNE             FIN_REC
                                                VIA_ACT=0
                CMPI            #1,D0           compara si el sem siguiente = rojo y cantón libre => pone a AMARILLO
                BEQ             PONE_AMARILLO
                MOVE.B          #2,D0           si no está a rojo lo pone a verde pq el cantón está libre
                MOVE.B          D0,EST_SEMAF
                BRA             SEMAF

PONE_AMARILLO:
                MOVE.B          #4,D0
                MOVE.B          D0,EST_SEMAF    * Actualizar estado del semaforo
SEMAF           MOVE.W          D0,PADAT        (mientras el estado de nuestro semaf. al semaf anterior
                LEA             XDAT,A0         * Enviar estado del semáforo al
                MOVE.B          D0,(A0)         * controlador del anterior
FIN_REC         MOVE.L          (A7)+,A0
                MOVE.L          (A7)+,D0
                RTE
```

TERO

Para poner a 0 el bit CAP:
CLR.L D0
1) BSET.B D0,TERO+1
2) BSET.W D0,TERO
3) BSET.B D0,TERO-1

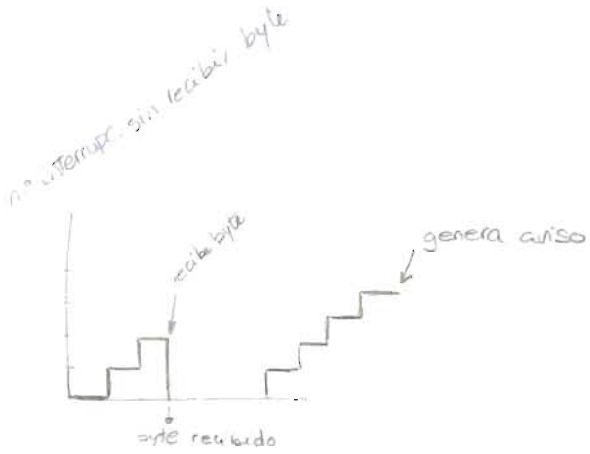
Nota: en esta 3ª forma tendría q venir:

TERO EQU $MBAR.MEN+$210
TER1 EQU $MBAR.MEN+$211

podemos entrar en la rutina si el vbl se mete en el cantón => poner semáforo ROJO. O sg el tren sale => poner semáforo VERDE

* al leer la var VIA_ACT de valor 0 o 1 se puede cambiar valor con BSET y BCLR

EST_REC es el valor del semáforo siguiente se puede ver en INT_REC



Lo normal es q se reciba al menos un byte cada TIRQ

```
* Rutina de interrupción de tiempo real - Temporizador 1
INT_TR0     MOVE.L   D0,-(A7)
            MOVE.L   D1,-(A7)
            MOVE.L   #1,D1
            BSET.B   D1,TER1+1
            CLR.L    D0
            MOVE.B   EST_SEMAF,D0
            MOVE.B   D0,XDAT
            TST.B    VIA_ACT
            BEQ      INCR_REC
vía ocupada MOVE.L   #1,D0
            MOVE.L   CONT_VIA,D1
            ADD      D0,D1
nº int period. MOVE.L D1,CONT_VIA
            CMPI     #240,D1
            BNE      INCR_REC
            CLR.L    D0
            BSET.B   D0,AVISO
INCR_REC    MOVE.L   #1,D0
            MOVE.L   CONT_REC,D1
            ADD      D0,D1
            MOVE.L   D1,CONT_REC
            CMPI     #4,D1
            BNE      FIN_TR
            MOVE.L   #1,D0
            BSET.B   D0,AVISO
FIN_TR      MOVE.L   (A7)+,D1
            MOVE.L   (A7)+,D0
            RTE

*****************************************************************
*       DEFINICIÓN DE VARIABLES
*****************************************************************
            ORG      $MMMMMMMM          * Zona de variables
CONT_VIA    DS.L     1
CONT_REC    DS.L     1
CONT_VEL    DS.L     1
VIA_ACT     DS.B     1                  * 0=Vía libre; 1=Vía ocupada
EST_SEMAF   DS.B     1                  * ROJO=1; VERDE=2; AMARILLO=4
EST_REC     DS.B     1                  * ROJO=1; VERDE=2; AMARILLO=4
AVISO       DS.B     1
FILA        DS.B     TAMFILA            se reserva TAMFILA bytes
FINFILA
            END
```

1. Configuración del subsistema de memoria

1.1. Complete la siguiente tabla con el mapa de memoria que se deduce a partir de la información del enunciado y del listado. Sitúe todos los elementos en las direcciones más bajas posibles y el modulo SIM a continuación de la memoria.

RANGO DE DIRECCIONES (HEX)	DISPOSITIVO ASIGNADO
$0000 0000 - $000F FFFF	Memoria EPROM = 1 MB = 2^{20}
$0010 0000 - $0013 FFFF	Memoria RAM = 256 KB = 2^{18}
$0014 0000 - $0014 FFFF	Módulo SIM = 64 KB = 2^{16}

1.2. Teniendo en cuenta el listado, las especificaciones del sistema y que todas las memorias del sistema necesitan el máximo número de estados de espera, configure los siguientes registros. Justifique cada uno de los campos de los registros.

la ROM siempre es CSOFF
el SIM no tiene CS

CSBR1 = $00100001
BA = $00100 Rango de la RAM: $00100000 - $0013FFFF
EBI = %00 Memoria RAM de 32 bits
BW = %00. Anchura de bus de 32 bits
SUPER = %0. Sin máscara para supervisor
ENABLE = %1. Se habilita el uso de CS1#

CSOR1 = $FFFC0078
BAM = $FFFC0. 256 kB = 2^{18}. Se comparan los 32-18 bits superiores
WS = %11110. Se usa el máximo número de estados de espera ($1E)
RW = %0. Indiferente
MRW = %0 Memoria de lectura/escritura

Direc inicial: $0010 0000
Direc. final $0013 FFFF
FFFC 0000

2. Interrupciones

2.1. Determine los valores $LLLLLLLL y $YYYYYYYY para que las interrupciones del sistema funcionen correctamente. Justifique su respuesta.

$LLLLLLLL = $40
Teniendo en cuenta que la dirección de la rutina de atención de la interrupción de la UART1 es $124, que VBR es $0 por defecto y que el offset en la tabla de vectores para la interrupción de la UART1 es 9, se puede obtener el valor de configuración $LLLL de PIVR
$124 = 292 = VBR + (PIVR + 9) × 4 ⟹ $LLLL = PIVR = 292/4 − 9 = 64 = $40

$YYYYYYYY = $114
$YYYY YYYY corresponde con la dirección de la rutina de atención a la interrupción del Temporizador 0. El offset en la tabla de vectores es 5, así que:
$YYYY YYYY = VBR + (PIVR + 5) × 4 = 0 + (64 + 5) × 4 = 276 = $114

2.2. Determine el valor inicial del registro de control ICR1 ($ZZZZZZZZ) en la rutina HW_INI para que todas las interrupciones estén adecuadamente configuradas. Justifique su respuesta.
(NOTA: UART pertenece a ICR2.)

$ZZZZZZZZ = $0000CB00
TMR0 IPL = %100 = 4. Nivel de la interrupción del temporizador 0
TMR1 IPL = %011 = 3 " " " " " " 1.
TMR0 IP = %1 Para cambiar el nivel hay que ponerlo a 1
TMR1 IP = %1 " " " " " " " "
El resto de interrupciones que configura este registro están inhabilitadas

3. Temporización

CE = %01

3.1. Justifique el valor de $WWWW para que la captura de entrada sea sensible a flanco de subida, se utilice el reloj del sistema dividido por 16 y el preescalado sea 64.

↳ CLK = %10

$WWWW = $3F44 * Valor a cargar en TMR0

PS = $3F Preescalado = PS+1 = 64

! ↳ PS+1 = 64 → PS = 63 preescalado

CE = %01. Captura de entrada habilitada (flanco subida + interrupción)

OM = %0 Modo de salida: pulso de un ciclo a nivel bajo (irrelevante) ← ser captura de entrada

ORI = %0. No habilita interrupciones de comparación de salida

FRR = %0 Modo reinicio en este caso da lo mismo ps lo usamos xa captura de flancos

CLK = %10 $f_{IN} = f_{CLK}/16$

RST = %0. Temporizador inhabilitado en este punto del código el temp todavía no estaba arrancado, lo hace luego en ESET_TMR0

3.2. Determine el valor $JJJJ que configura la interrupción periódica según el enunciado usando como fuente de reloj el reloj del sistema dividido por 16. Justifique su respuesta.

! $JJJJ = $E01C * Valor a cargar en TMR1

Si el tren está en el cantón (VIA_ACT = 1), con cada interrupción periódica del temporizador 1, se incrementa en 1 la variable CONT_VIA. Cuando alcanza el valor de 240 se genera el aviso correspondiente a que el tren lleva 6 minutos en el cantón. Así que 240 interrupciones corresponden a 6 minutos $\Rightarrow f_{IRQ} = \frac{240 \text{ interrup.}}{6 \text{ min}} = 40 \frac{\text{interrup.}}{\text{min}} =$

$= 40 \frac{\text{interrup.}}{\text{min}} \cdot \frac{1 \text{ min}}{60 \text{ sg}} = 0{,}667 \frac{\text{interrup}}{\text{sg}}$. Como la interrupción periódica se produce cada vez que el contador alcanza el valor del registro de referencia TRR1 = $61A8 = 25000 tenemos:

$T_{IRQ} = \frac{1}{f_{IRQ}} = TRR1(\text{cuentas de contador}) \times T_{CNT}\left(\frac{\text{sg}}{\text{cuenta de cont}}\right) = TRR1 \times \frac{1}{f_{CONT}} = TRR1 \times \frac{16(PS+1)}{f_{CLK}}$

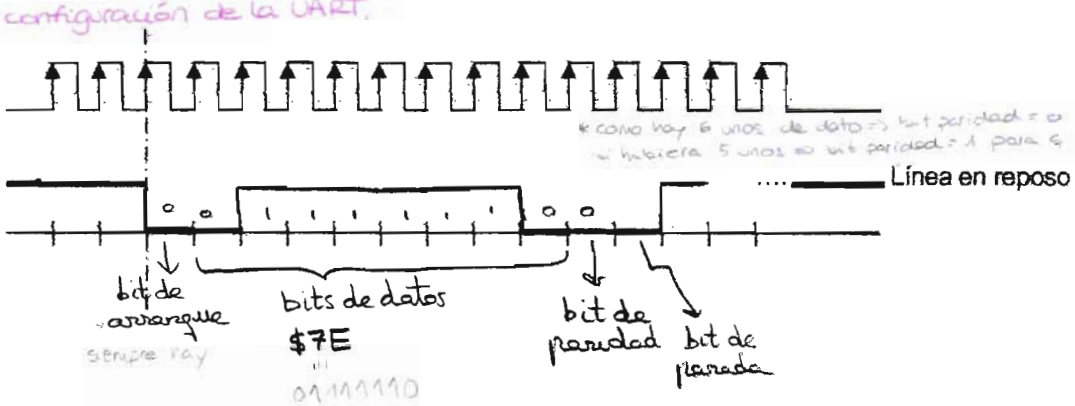
Despejamos $PS = \frac{f_{CLK}}{f_{IRQ} \times 16 \times TRR1} - 1 = \frac{60 \cdot 10^6}{\frac{40}{60} \times 16 \times 25000} - 1 = 225 - 1 = 224 = \$E0$

← Así que:

4. Comunicaciones serie

4.1. Teniendo en cuenta la configuración de la UART1 dibuje la forma de onda de la trama cuando se envía el dato $7E, especificando claramente a qué corresponde cada bit. ¿Cuál es el valor de tiempo de Tx/Rx de un bit (t_{bit})? Justifique su respuesta.

Ver configuración de la UART.



- PS = %E0 Preescalado PS+1 = $E1 = 225
- CE = %00 Captura de entrada inhabilitada
- OM = %0 Modo de salida irrelevante
- ORI = %1 Habilita interrupc. de comparac. de salida
- FRR = %1 Modo reinicio
- CLK = %10 $f_{IN} = f_{CLK}/16$
- RST = %0 Temporizador inhabilitado

$t_{bit} = 1/4800\ bps = 0,000208\ sg/bit = 208\ \mu s/bit$

Los bits de la trama y el tiempo de bit vienen determinados por la configuración del registro de control XCR = $31 Se ve en el código y mirando los registros

BR = %01. Velocidad = 4800 baudios
DT = %1. Se enviarán 8 bits de datos
ST = %0. Se usa un bit de parada
PR = %0. Paridad par → En este caso el bit de paridad será 0 pq. el número de 1's del dato es par
TX = %0. Interrupción de transmisión deshabilitada
RX = %1. Interrupción de recepción habilitada

Nota: con paridad par el número total de 1's considerando el dato + el bit de paridad, debe ser par. Con paridad impar, debe ser impar

5. Software

5.1 Explique la función de la variable EST_REC en las rutinas INT_REC e INT_CAP.

Esta variable representa el estado del semáforo siguiente y se actualiza al recibir un dato por la línea serie. Permite decidir en INT_CAP e INT_REC el color del semáforo cuando el cantón no está ocupado. Si EST_REC = 1 (Rojo) ⇒ EST_SEMAF = 4 (amarillo)
Si EST_REC = 2ó4 (Verde o amarillo) ⇒ EST-SEMAF = 2 (Verde)

5.2 Teniendo en cuenta el enunciado y el listado del programa ¿cada cuanto tiempo se debe recibir información del semáforo siguiente? ¿Cuál es el máximo tiempo permitido sin que se produzca el aviso de falta de recepción? Justifique su respuesta.

El estado del semáforo se envía al semáforo anterior en 3 circunstancias:
1. Periódicamente en la rutina de atención del temporizador 1. (INT_TR) cada $T_{IRQ} = 1,5$ sg
2. Cuando cambia el color o estado del semáforo como consecuencia del color recibido del siguiente semáforo (INT_REC)
3. Cuando se pasa de cantón ocupado a desocupado o viceversa (INT. CAP)

Cuando se recibe el estado del semáforo siguiente, se pone CONT_REC = 0 en INT_REC. Cada vez que se ejecuta INT_TR periódicamente cada $T_{IRQ} = \frac{1}{1/T_{IRQ}} = 1,5$ sg se incrementa en 1. Cuando alcanza el valor de 4 sin haber sido puesta a 0 en INT_REC debido a la recepción de un byte, se genera el aviso de falta de recepción así que esto se producirá a los: $4 \times 1,5 = 6$ sg

Se debe recibir como mínimo cada 1'5 seg que es el periodo entre interrupciones.

5.3 Suponga la siguiente situación justo antes de que se produzca la interrupción de captura del temporizador 0: DATO RECIBIDO DEL SEMAFORO SIGUIENTE = $01; VIA = Ocupada. ¿Cuál será el estado al finalizar la ejecución de la rutina de atención a la interrupción INT_CAP? Justifique su respuesta.

VIA = Libre

SEMAFORO = Amarillo

La interrupción de captura se produce o bien en el flanco de subida (paso a vía ocupada) o bien en el flanco de bajada (paso a vía libre). En la situación planteada se producirá en un flanco de bajada, ya que la vía estaba ocupada, y la vía pasará a estado libre y el semáforo se pondrá en amarillo dado que el siguiente está en rojo.

5.4 Teniendo en cuenta la variable CONT_VEL, la constante longitud del cantón L_CANTON, y que se conoce la longitud del tren L_TREN, ambas expresadas en Km, escriba la fórmula que calcula la velocidad a la que va el tren (Km/hora) al atravesar el cantón y que implementaría la rutina CALCULA_VEL. Justifique su respuesta.

(Ver dibujo) El tren recorre L_CANTON + L_TREN durante el tiempo $T\_VIA = t_2 - t_1$ así que:

$$Velocidad\ (km/h) = \frac{L\_CANTON + L\_TREN\ (km)}{T\_VIA\ (horas)}$$

T_VIA es el tiempo que permanece ocupado el cantón. Mientras está ocupado, cada vez que se atiende la interrupción periódica del temporizador 1, se incrementa en 1 el valor de la variable CONT_VEL en INT_TR, así que: (Y una vez se desocupa la vía, este valor se almacena en CONT_VEL en la rutina INT_CAP)

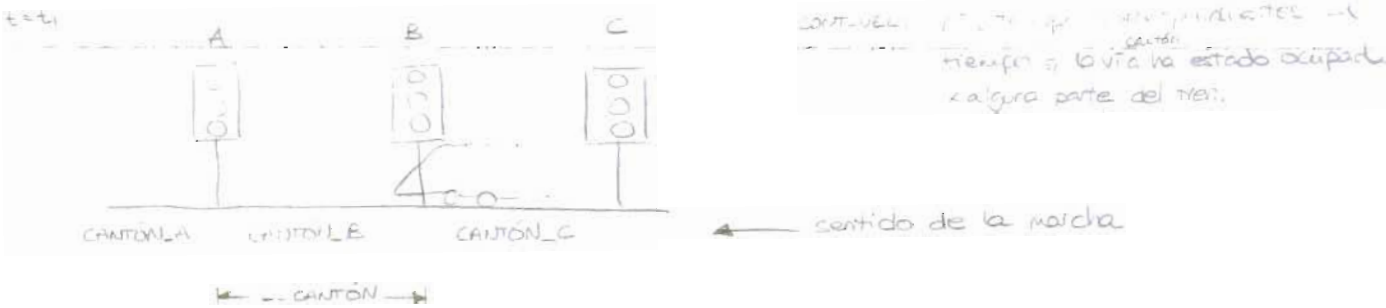
$$T\_VIA\ (sg) = CONT\_VEL\ (interrup) \times T_{IRQ}\left(\frac{sg}{interrupc}\right) = CONT\_VEL \times T_{IRQ}$$

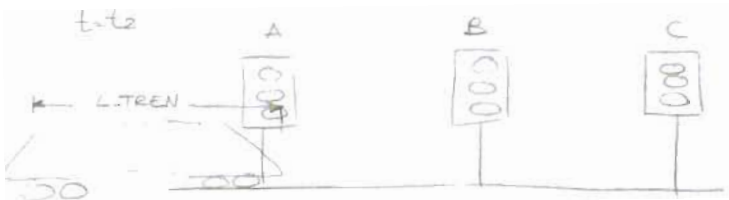
$$T\_VIA\ (h) = T\_VIA\ (sg)\ \frac{1\ hora}{3600\ sg} = \frac{CONT\_VEL \times T_{IRQ}}{3600}$$

$T_{IRQ} = 1{,}5\ sg$

Así que:

$$Velocidad\ (km/h) = \frac{L\_CANTON + L\_TREN}{CONT\_VEL \times 1{,}5} \times 3600$$





TEMA 7: MEMORIAS EN EL SISTEMA MICROPROC.

7.1 Memorias integradas
7.2 Un caso especial de memorias integradas: las memorias dinámicas

---

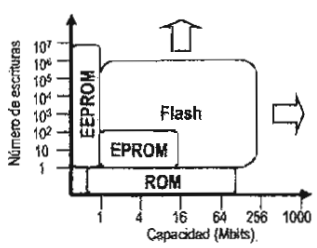
7.1 Memorias integradas

7.1.1 Tipos de memorias ROM (NO VOLÁTILES)

El nombre ROM es una reminiscencia del pasado, porque muchas de las memorias así llamadas también se pueden escribir, pero de todas formas no permiten escribir de una manera tan sencilla como una memoria RAM.

Las memorias ROM (*Read Only Memory*) son memorias no volátiles. Vemos los tipos, sus características y sus aplicaciones principales, en el siguiente cuadro:

Tipo	*Volátil*	*Características*	*Aplicaciones típicas*
ROM por máscara	No	• Se encargan y se graban durante el proceso de fabricación • Bajo precio para series grandes	• Distribución de software • Equipos fabricados en grandes series
PROM (ROM programable) *Programmable ROM*	No	• Se puede grabar posteriormente a la fabricación del chip • Programable una sola vez (fusión de fusibles) ⇒ No reprogramable	• Realización de prototipos de lógica combinacional
EPROM (ROM programable y borrable) *Erasable PROM*	No	• Programable mediante inducción de cargas • Borrable totalmente mediante exposición a rayos UV (fuera del circuito) ⇒ los chips suelen llevar una "ventanita" • Reprogramable un número pequeño de veces	• Desarrollo de prototipos de laboratorio • Pequeñas series
EEPROM (ROM programable y borrable eléctricamente) *Electrically EPROM*	No	• Borrable eléctricamente sin extraer del circuito • Son las únicas que permiten borrar células individuales • Reprogramable un número alto de veces • Capacidad pequeña • Coste alto	• Refinamiento de prototipos • Actualización de versiones
Flash	No	• Grabable muchas veces • La escritura se realiza por bloques completos (INCONVENIENTE) • Bajo coste de fabricación • Se pueden conseguir capacidades muy grandes • Características ideales para casi todas las aplicaciones	• Memoria de arranque del sistema • PC BIOS *(basic input output system)* • Controladores de disco • Tarjetas de memoria para cámaras digitales, teléfonos móviles... • Sistemas empotrados



7.1.2 Tipos de memorias RAM (VOLÁTILES)

¡OJO! Las memorias ROM también son de acceso aleatorio. Los nombres no son muy apropiados.

Las memorias RAM (*Random Access Memory*) son memorias en general volátiles que permiten lectura/escritura. Los distintos tipos y aplicaciones se ven en la siguiente tabla.

Tipo	*Volátil*	*Características*	*Aplicaciones típicas*
SRAM (RAM estática)	Sí	• Muy rápidas • Muy caras • Tamaños pequeños • La célula (bit) es un biestable • Alta integración • Tecnología CMOS: - Alta velocidad - Bajo consumo - Modo *standby* en sistemas con baterías: mantiene el contenido si Vcc ≥ 2 V (no volátil) • Tecnología ECL: - Muy alta velocidad - Alto consumo	• Memorias pequeñas • Memorias caché
DRAM (RAM dinámica)	Sí	• Bajo precio (tecnología MOS) • Se pueden conseguir grandes tamaños • La célula (bit) es una capacidad MOS • Muy alta integración (mayor que en SRAM) • Necesidad de refrescar el contenido periódicamente por las fugas de corriente en las capacidades (milisegundos) • Requieren un controlador	• Grandes bancos de memoria
VRAM (RAM de video)	Sí	• Salida serie (acceso secuencial a las posiciones)	• Controladores de vídeo

¡OJO! En estas el acceso es secuencial

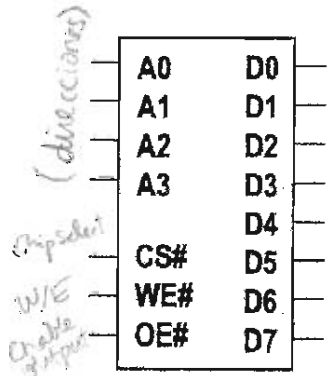
Los encapsulados que agrupan varios chips RAM reciben el nombre de:

- SIMM (bus de datos de 32 bits)
- DIMM (bus de datos de 64 bits)

7.1.3 El chip de memoria

Los terminales del chip de memoria son:

- Bus de direcciones
- Bus de datos
- Selección de chip CS#
- Lectura/escritura WE# o R/W# (sólo en RAM)
- Habilitación de salida OE# (opcional)
- Alimentación (Vcc)



Capacidad:

Nº de palabras = 2 $^{\text{Nº de teminales de dirección}}$

Nº de bits/palabra = Nº de terminales de datos

Capacidad = Nº de palabras x Nº de bits por palabra

(por ejemplo 2K x 16 bits y como 16 bits = 2 bytes ⇒ 4 kBytes)

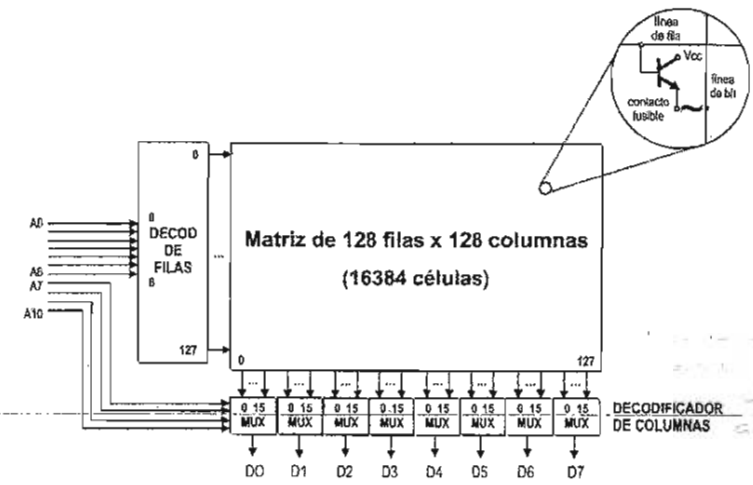
- Los chips suelen ser de 1, 2, 4, 8, 16 y 32 bits por palabra
- Se suelen combinar varios chips para formar una memoria concreta

7.1.4 Estructura de las memorias ROM

Las matrices de células normalmente son cuadradas

- Matriz de células (transistores en corte o saturación)
- Decodificador de filas y multiplexores para columnas

Los bits de una misma dirección de memoria no están contiguos en la matriz



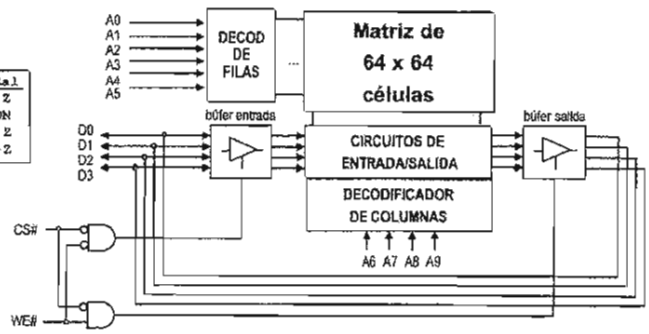
7.1.5 Estructura de las memorias SRAM

- Matriz de células (biestables)
- Decodificadores de filas y de columnas
- Circuitos de lectura/escritura
- Búferes de entrada y de salida

Si la señal de CS# está desactivada las entradas/salidas desde/hacia el bus de datos están en alta impedancia

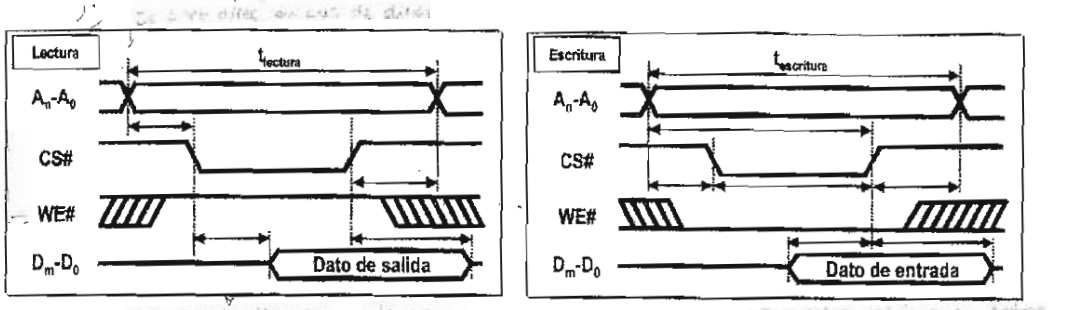
Si está activada depende del valor de la señal WE# (o R/W#)

CS#	WE#	Ent	Sal
0	0	ON	↑Z
0	1	↑Z	ON
1	0	↑Z	↑Z
1	1	↑Z	↑Z



7.1.6 Temporización en memorias

- Los accesos a memoria requieren la dirección, el dato (si es acceso de escritura), WE# (en RAM) y CS#
- La última señal en activarse es la que controla el acceso
  - Lo habitual es controlar el acceso con CS#
  - Es una secuencia de lecturas se puede controlar el acceso con la dirección (se mantienen WE# y CS# fijos)
- Tiempo de acceso a la memoria: $t_{memoria} \approx t_{lectura} \approx t_{escritura}$
- Ejemplos de cronogramas para un ciclo de lectura y para un ciclo de escritura:



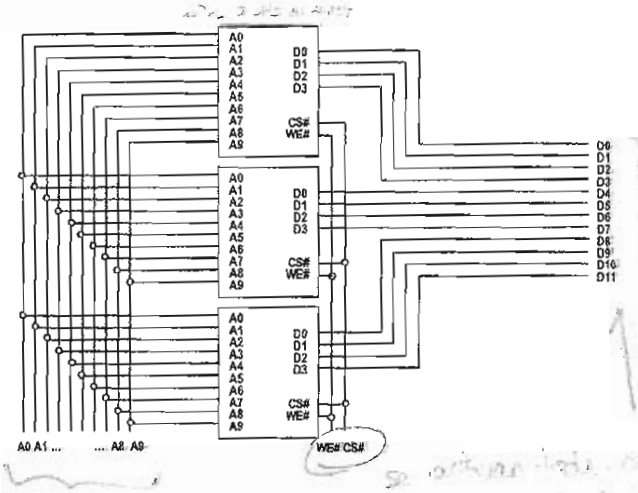
7.1.7 Expansión de memorias

Podemos querer hacer más grande el tamaño de la palabra, el número de palabras, o ambas cosas simultáneamente. Vemos unos ejemplos:

¡¡¡Norma general: nunca se pueden activar simultáneamente pastillas que accedan a los mismos bits del bus de datos!!!

Expansión del tamaño de palabra

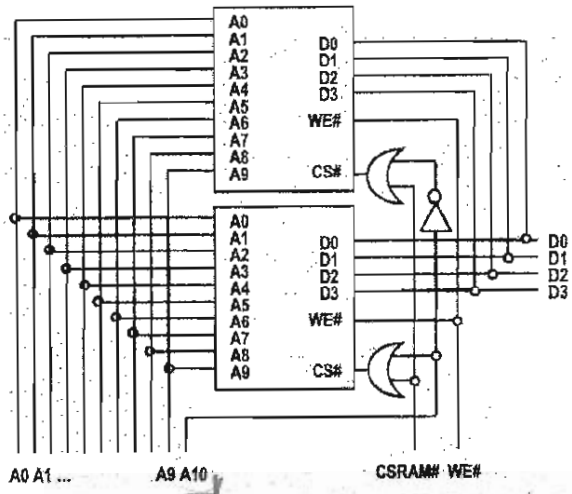
Ejemplo: construir una memoria de 1K x 12 con memorias de 1 K x 4



Clave: se activan todas las pastillas simultáneamente porque acceden a distintos bits del bus de datos, y la suma del número de terminales de datos de las pastillas usadas tiene que ser igual al número de bits de datos de la palabra deseada

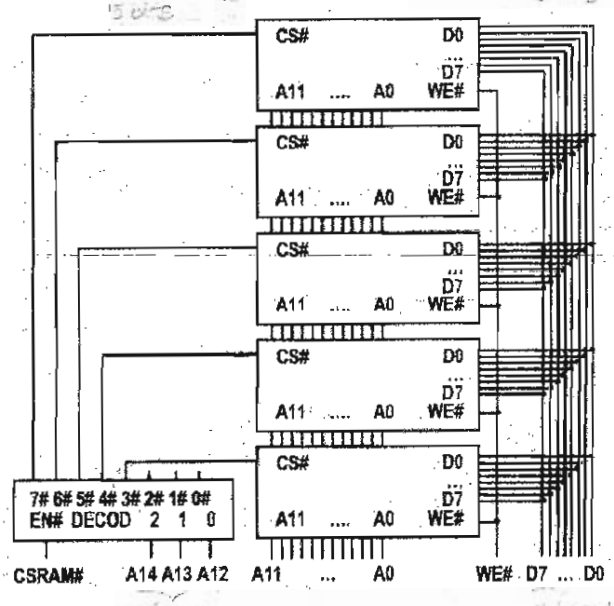
Expansión del número de palabras

Ejemplo: construir memoria de 2K x 4 con pastillas de 1 K x 4



Clave: **los bits más significativos son los que deciden que pastillas se habilitan**, mientras que los menos significativos direccionan dentro de cada pastilla

Ejemplo: construir memoria de 20 K x 8 con 4 K x 8



En este caso como se usan más de dos pastillas, se utiliza un decodificador para elegir que pastilla habilitar, pero sigue dependiendo de los bits más significativos

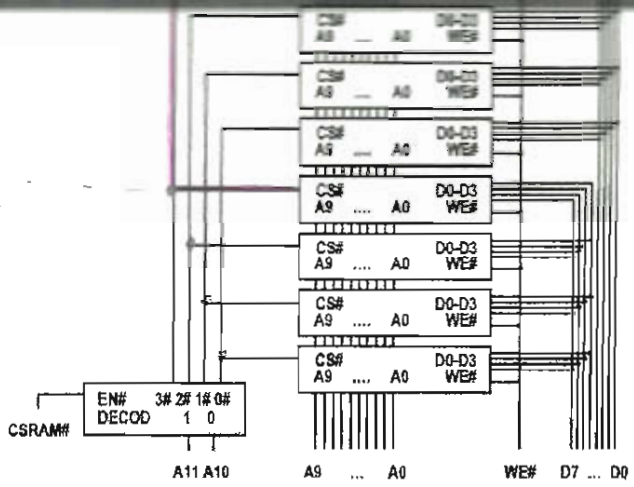
Expansión del tamaño y número de palabras

Ejemplo: construir memoria de 4K x 8 con memorias de 1K x 4

Consiste en juntar lo visto anteriormente:

- Con cada 2 pastillas de 1K x 4 formamos una de 1K x 8
- Con 4 pastillas de 1K x 8 formamos una pastilla de 4K x 8
- En total se necesitan 2 x 4 = 8 pastillas de 1 K x 4
- Las pastillas se habilitarán de dos en dos: cada vez que se quiera acceder a una palabra de 8 bits completa tendrán que activarse dos pastillas de 1 K x 4 que accederán a distintos bits del bus de datos
- Los bits más significativos se siguen encargando de decidir qué pastillas habilitar

Fijate como las pastillas que acceden a los mismos bits del bus de datos (las 4 de arriba o las 4 de abajo) nunca se habilitan a la vez porque sus entradas de CS# están conectadas a distintas salidas del decodificador



7.1.8 Jerarquía de memorias

Lo primero es entender la diferencia entre paralelismo y jerarquía de memorias

Paralelismo vs Jerarquía

- El objetivo es minimizar los tiempos de acceso a espacios de memoria grandes

El ejemplo de expansión del tamaño de palabra visto anteriormente es un ejemplo de paralelismo

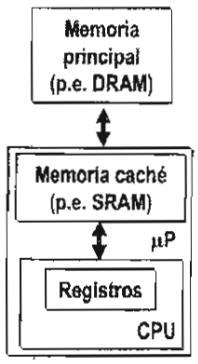
- Paralelismo:
  - El dato está distribuido entre varias memorias a las que se accede simultáneamente para obtenerlo
  - Esto es útil si el tiempo de acceso es menor en memorias con tamaños de palabra más pequeños

- Jerarquía:
  - Separación de las memorias en niveles con distintos tiempos de acceso
  - Tiempos de acceso: $t_{registro} < t_{SRAM} < t_{DRAM}$

Ahora profundizamos en el tema de la jerarquía de memorias

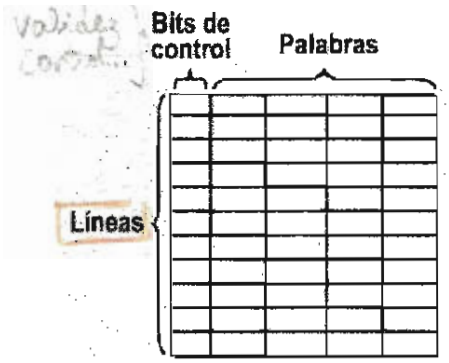
Memorias jerárquicas

- Niveles (de tiempo de acceso más rápido a tiempo de acceso más lento):
  registros ---- memoria(s) caché ---- memoria principal



Memoria caché

- La caché duplica los datos a los que se accede frecuentemente
- Se reduce el tiempo de acceso desde la CPU
- Al escribir hay que actualizar la memoria principal para mantener la coherencia de memoria
- Se suelen separar cachés de instrucciones y de datos ya que se accede de manera distinta a instrucciones que a datos y los algoritmos de organización que se usarán serán distintos



- Se organiza por líneas (bloques de palabras en direcciones consecutivas)
- Los bits de control tienen información sobre validez, dirección...
- Acierto en el acceso a la caché (*hit*): la dirección buscada está en una línea de caché
- Fallo en el acceso a la caché (*miss*):
  - La solicitud se propaga hacia la memoria principal
  - Se copia toda una línea desde la memoria principal ⇒ si hay accesos a las palabras siguientes a la solicitada, serán aciertos.
- Tasa de fallos (*miss ratio*)= fallos/accesos
  - Mide la eficiencia de la estructura
  - Interesa que sea lo más baja posible (<10-15%)
  - Depende de:
    - ° Tamaño de líneas y número de las mismas
    - ° Algoritmo de sustitución: cómo se organizan los datos en la caché
    - ° Algoritmo de invalidación: como se procesan las escrituras para mantener coherencia con la memoria principal

7.1.9 La memoria interna del MCF5272

Esto ya lo vimos en el tema 3

- SRAM: 1K x 32 (4 kBytes) : accesos en un ciclo a .B, .W, ó .L
  - Configurable mediante RAMBAR
- ROM: 4K x 32 (16 kBytes): tablas del sistema, no modificable
  - Configurable mediante ROMBAR
- Caché de instrucciones: 1 kByte en forma de 64 líneas de 128 bits cada una
  - configurable mediante CACR, ACR0 y ACR1

En SEDG no se ve nada más de la memoria caché del MCF5272

7.2 Memorias dinámicas DRAM

- Almacenamiento en capacidades C_{GS} en MOS
  - Condensador cargado ('1') o descargado ('0')
- Ventaja: muy alto nivel de integración (se pueden conseguir memorias muy grandes)
- Problema: descarga de C_{GS} por fugas de corriente
- Solución: proceso de refresco periódico
  - Leer cada célula y, si está cargada, recargarla
  - El refresco se hace por filas completas (páginas) de la matriz de células

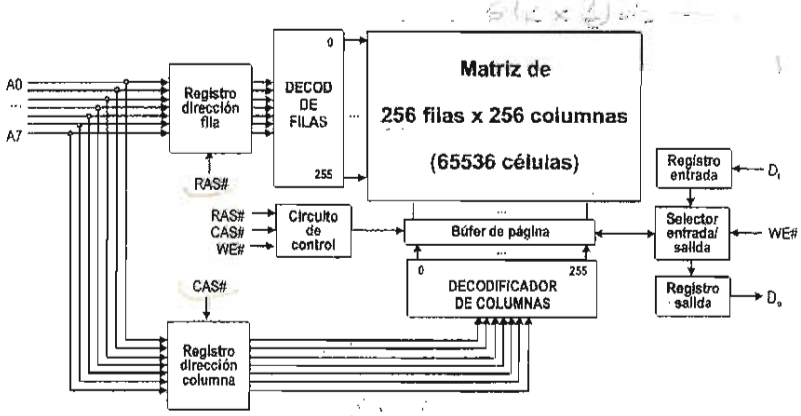
- La memoria no está disponible mientras se refresca
- Rentable sólo a partir de unos 256 K por el coste de los circuitos de refresco

- Necesidad de un controlador de DRAM:
  - Adaptan el acceso a la DRAM
  - Generan las señales que controlan el proceso de refresco

7.2.1 Estructura de un chip DRAM

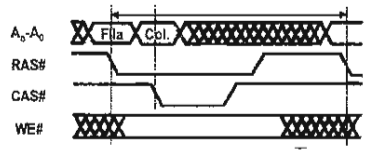
Esta es la estructura de un un chip. Fíjate que en esta figura aún no aparece el controlador de DRAM

Este es un chip DRAM de 64K x 1 bit



Con esto se reduce el número de pines y el tamaño del encapsulado permitiendo abaratar mucho los costes

- Una DRAM recibe las direcciones de N bits en 2 bloques sucesivos de N/2 bits
- Los dos bloques se cargan en registros de dirección de fila y columna:
  - RAS#: señal de carga del registro de dirección de fila
  - CAS#: señal de carga del registro de dirección de columna
- El controlador de DRAM será el encargado de separar la dirección en dos bloques de N/2 bits y generar RAS# y CAS#



Elementos de un chip DRAM

- Registros de dirección, decodificadores, matriz de células, circuito de control, circuitos de entrada/salida
- Búfer de página:
  - En él se carga la fila (página) indicada por el registro de dirección de fila
  - En él se leen o escriben las células (columnas) o se refresca la fila completa
  - Al finalizar el acceso (o refresco) la página se actualiza en la matriz de células

Bancos de memoria en un chip DRAM

- A veces, la matriz puede tener 2 ó 4 bancos (p.e. 4Mx16= 1Mx16x4 bancos)
- Así se dispone de 2 ó 4 búfers de página activos con direcciones distintas
- Ejemplo de direccionamiento: memoria de 1Mx16x4 bancos:
  - A22 y A21seleccionan el banco (estos terminales no se multiplexan). 4 bancos ⇒ necesitamos 2 bits para seleccionar
  - A20 – A1: direccionan posiciones en el banco (1M posiciones, necesitamos 20 bits)
  - A0 no se usa: palabras de 16 bits

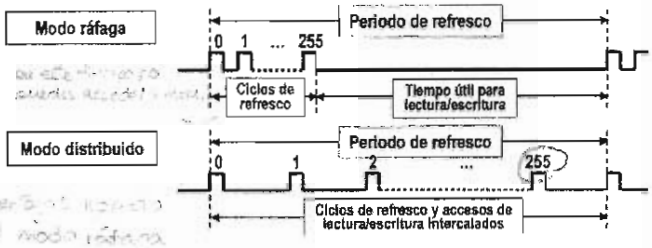
Fíjate que en el fondo 1Mx16x4 bancos= 8 Mbytes =2^{23} bytes ⇒ Se "usan" A22-A0

- Otro ejemplo de direccionamiento: memoria de 4Mx32x2 bancos
  - A24 selecciona el banco (este terminal no se multiplexa). 2 bancos ⇒ necesitamos sólo 1 bit para seleccionar
  - A23 – A2: direccionan posiciones en el banco (4M posiciones, necesitamos 22 bits)
  - A0 y A1 no se usan: palabras de 32 bits

Fíjate que en el fondo 4Mx32x2 bancos= 32 Mbytes= 2^{25} bytes ⇒ Se "usan" A24-A0

El proceso de refresco

- Periodo en el que hay que refrescar la DRAM (del orden de milisegundos)
- Ciclo de refresco: proceso de refresco de una fila (página) de la DRAM
- Modo de refresco:



- Factor de calidad de memorias DRAM: describela fracción de tiempo que la memoria no está disponible debido a los ciclos de refresco

Ver figura de arriba

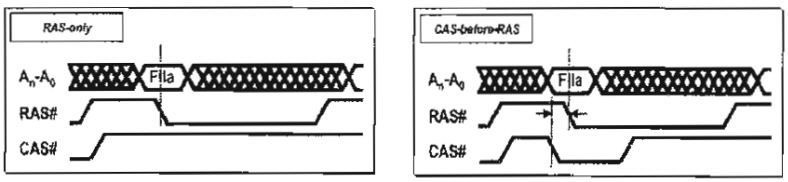
$$\text{Factor de calidad} = \frac{\text{Número de filas x Duración de un ciclo de refresco}}{\text{Periodo de refresco}}$$

- Ejemplo:
  - Periodo de refresco = 2 ms
  - DRAM de 256 filas
  - Ciclo de memoria = 200 ns

Esto se interpreta como que el proceso de refresco precisa un 2,56 % del tiempo útil de la memoria

$$\text{Factor de calidad} = \frac{256 \text{ x } 200 \cdot 10^{-9}}{2 \cdot 10^{-3}} = 0,0256 = 2,56\%$$

- Modos de inicio del ciclo de refresco. Hay varias opciones:
  - Mediante la activación de un terminal específico para ello
  - Mediante una secuencia RAS#/CAS# distinta a la normal:
    - *RAS-only*: en el ciclo sólo se activa RAS#
    - *CAS-before-RAS:* se altera el orden de activación



- Mientras se produce el proceso de refresco, la salida de datos permanece en estado de alta impedancia, y la entrada de datos y la señal WE# no afectan al proceso

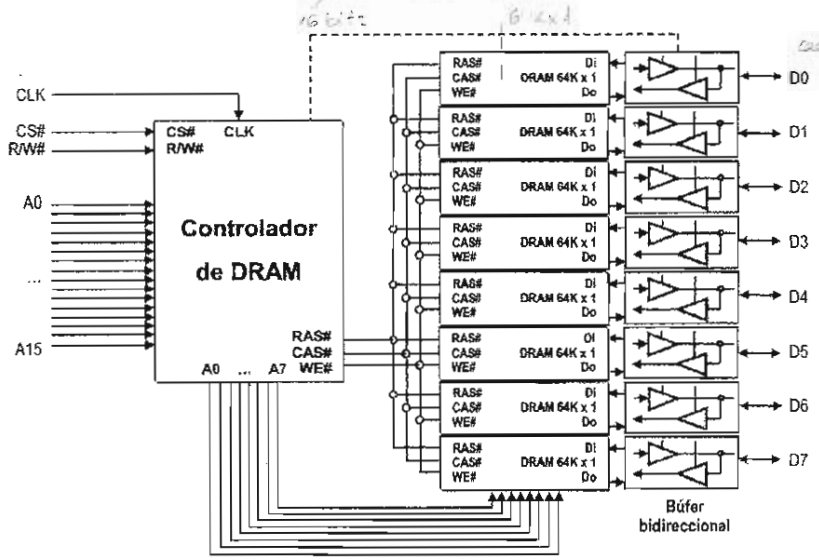
7.2.2 Controladores de DRAM

- Un subsistema de memoria DRAM está formado por un conjunto de chips de memoria DRAM para obtener una cierta capacidad y ancho de palabra
  - Cada chip puede tener varios bancos

- Para manejarlo se necesita un controlador de DRAM

- Lós bloques de un controlador de DRAM son:
  - Multiplexor de direcciones
  - Contador: necesario para direccionar las filas en el refresco
  - Circuito de control:
    - ° Recibe las señales de control del bus de sistema
    - ° Genera las señales RAS#, CAS# y WE# hacia la DRAM
    - ° Controla del proceso de refresco (de manera transparente al sistema)
    - ° Gobierna los otros bloques del controlador de DRAM

- Ejemplo de memoria DRAM de 64Kx8= 64 kbytes con controlador de DRAM



Ejercicio 17. Preguntas cortas

a) Diga al menos dos diferencias entre las memorias EEPROM y las Flash.

b) Explique las principales diferencias entre las memorias SRAM y DRAM y explique sus aplicaciones principales.

c) ¿Qué ventaja ofrece una caché de instrucciones?

d) ¿Cuántos bancos de memoria tiene una DRAM de 4 M x 8 bits si cada banco dispone de 2K páginas y 1 K columnas? Justifique su respuesta.

(a)

EEPROM

- coste alto
- capacidades pequeñas
- permite borrar células individuales

FLASH

- coste bajo
- se pueden conseguir capacidades grandes
- el borrado se realiza por bloques completos.

(b)

SRAM

- muy rápidas
- muy caras
- capacidades pequeñas
- nivel de integración menor que en las DRAM.
- la célula (bit) es un biestable
- no necesita refresco

DRAM

- más lentas que las SRAM
- más baratas que las SRAM
- grandes capacidades
- muy alto nivel de integración
- la célula (bit) es un condensador
- necesitan refresco

Aplicaciones:

- memorias pequeñas
- memorias caché
- grandes bancos de memoria

(c) • Acelera el acceso a las instrucciones que se encuentran en ella, evitando el tener que ir a buscar las instrucciones a la memoria principal (DRAM) que es más lenta.

- Se consiguen bajas tasas de fallos si se ejecutan instrucciones sucesivas en memoria (sin saltos), y en la ejecución reiterada de código (bucles).

d) • Capacidad total: $4M \times 8 \text{ bits} = 2^{22} \text{ (palabras de 8 bits)} \times \frac{1 \text{ byte}}{\text{palabra de 8 bits}} =$

$$= 2^2 \cdot 2^{20} \text{ bytes} = \boxed{4 \text{ MBytes}}$$

• Capacidad de cada banco:

$$2K \text{ filas} \times 1K \text{ columnas} = 2 \cdot 2^{10} \text{ filas} \times 2^{10} \text{ columnas} =$$

$$= 2 \cdot 2^{20} \text{ posiciones en la matriz}$$

$$2 \cdot 2^{20} \text{ posiciones} \times \frac{8 \text{ bits}}{\text{posición}} = 2 \cdot 2^{20} \, 8 = 16 \text{ Mbits} = 2 \text{ MBytes}$$

Cada banco tiene una capacidad de 2 MBytes
La capacidad total es de 4 MBytes
$\Rightarrow$ 2 bancos