eman ta zabal zazu
universidad del país vasco
euskal herriko unibertsitatea

departamento de electricidad y electrónica
elektrika eta elektronika saila

Examen de S.E.T.I.

1er curso de Ingeniería Electrónica
4 de febrero de 2005

Cuestiones. 1 + 1 puntos

1 ¿El primer teorema de Shannon es aplicable de algún modo a la compresión de ficheros en una computadora?.

2 ¿Qué es la Máquina Universal de Turing?

Ejercicio de C. 1 punto

3 Escriba en C una subrutina para calcular las raices de un polinómio de orden 2 ($ax^2+bx+c=0$) con coeficientes reales, que recibe estos coeficientes a,b,c en una estructura y devuelve las dos soluciones en otra. Tengase en cuenta que las raices pueden ser complejas y en tal caso la rutina no deberá obtener el resultado sino indicar de algún modo tal eventualidad.

Estructura de un microprocesador. 1,2+0,3 puntos

4 a) El microprocesador virtual visto en la asignatura es extremadamente simple hasta el punto de carecer de la posibilidad de programar subrutinas. Explique qué sería necesario para añadirle tal funcionalidad (es decir, sabiendo cómo funciona la llamada a subrutinas en un procesador real, indicar todo aquello de lo que carece el nuestro).

b) Una vez determinadas las necesidades indique las instrucciones que daría usted a un equipo de trabajo para modificar el hardware con objeto de proporcionar al procesador esta capacidad.

Mapeo de memoria. 1,5 puntos

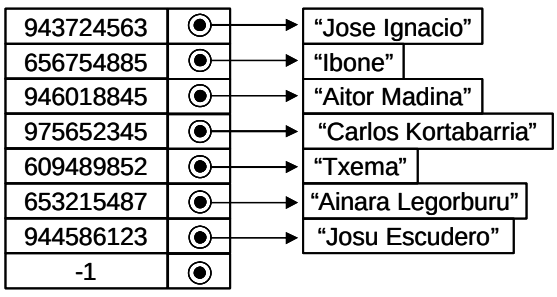
5 Diseñe el circuito necesario para que un computador con microprocesador 68000 disponga de:

a) 8 Mbytes de RAM situados a partir de 0 mediante dos pastillas estructuradas en bytes.
b) 512 Kbytes de ROM en el final del espacio direccionable mediante 4 pastillas en niblas .

No se impone ninguna restricción sobre la posibilidad de que existan copias de las pastillas en otras posiciones del espacio direccionable aparte, claro está, de las necesarias para que los distintos circuitos no se solapen. (muestre el mapa resultante)

Programación del 68000. 4 puntos (1 + 2 + 1)

6 Para representar una agenda telefónica, se ha diseñado una estructura en memoria que consta de una secuencia de pares de LongWords: la primera contiene **el número** de teléfono (podremos guardar cualquier número nacional) y la segunda contiene **la dirección** de la cadena de caracteres asociada al número telefónico. Como marcador de final de secuencia, el último número de dicha secuencia será el **-1**. Una representación simbólica de dicha estructura podría ser la de la figura:



Se pide que se diseñen las siguientes subrutinas y programa principal:

Subrutina BUSCANUM

Esta subrutina deberá tomar la dirección de una cadena de caracteres y buscar el número asociado a dicha cadena. En caso de no existir, debiera devolver -1. Para su desarrollo, podremos contar con la subrutina STRCMP que compara las cadenas apuntadas por A0 y A1, devolviendo en D0 el resultado de dicha comparación (D0==0 si son iguales). La subrutina ***podría*** tener en concreto la siguiente descripción:

```
; Subrutina:     BUSCANUM
; Descripción:   Busca el número asociado a una cadena de caracteres
; Entrada:       A0.L  dirección de la agenda
;                A1.L  dirección de la cadena a buscar
; Salida:        D0.L  número asociado o -1 si no existe
; Modifica:      (rellenar debidamente)
```

Subrutina BUSCACAD

Esta subrutina deberá tomar un número telefónico y buscarlo en la agenda, en caso de encontrarlo, deberá devolver la dirección de la cadena asociada. En caso de no existir, deberá devolver la dirección 0. La subrutina ***podría*** tener en concreto la siguiente descripción:

```
; Subrutina:     BUSCACAD
; Descripción:   Busca la cadena asociada a un número
; Entrada:       A0.L  dirección de la agenda
;                D0.L  Número a buscar
; Salida:        A1.L  dirección de la cadena o 0 si no existe
; Modifica:      (rellenar debidamente)
```

Programa principal

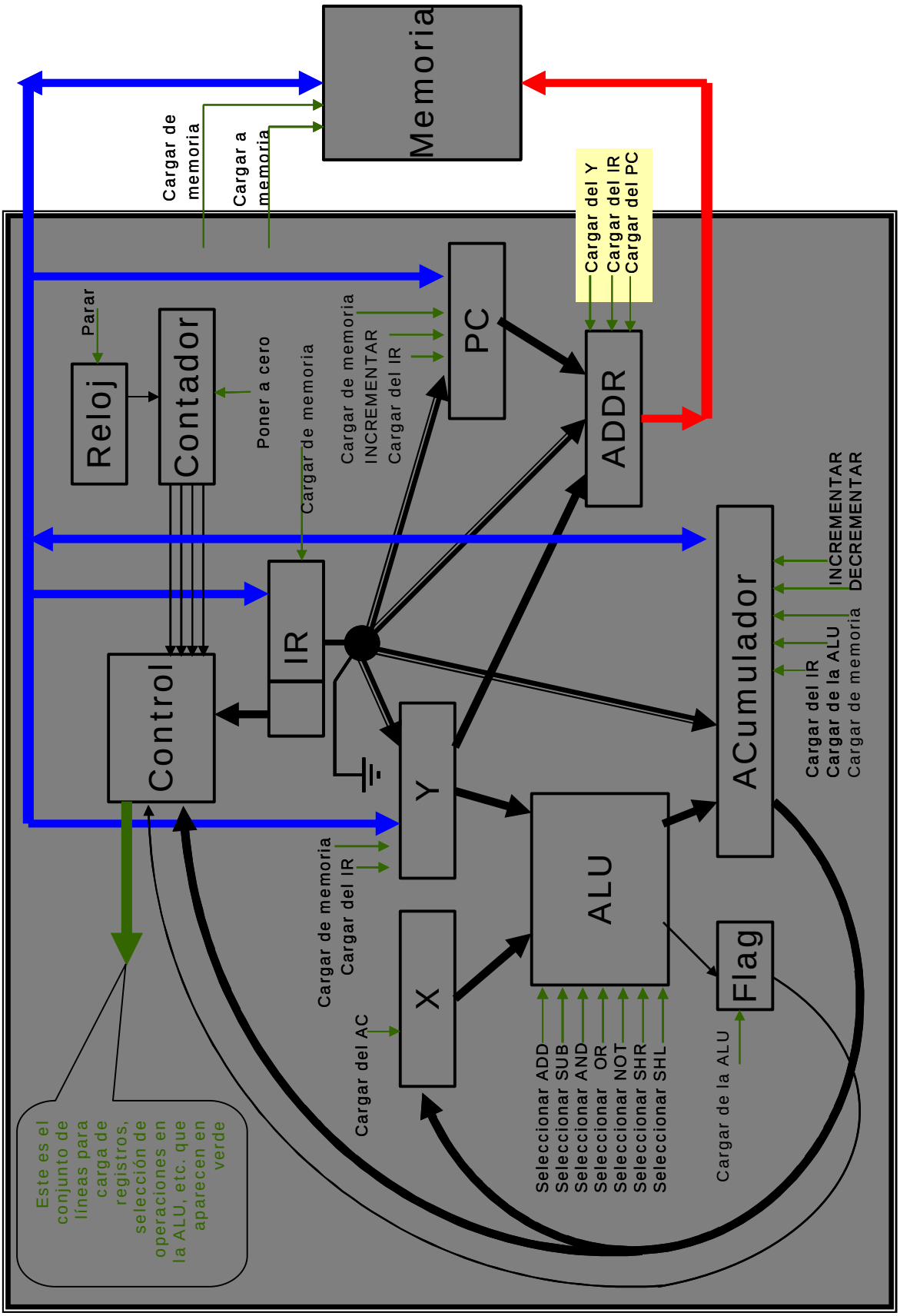
El programa principal deberá crear una agenda que contenga al menos 7 entradas y realizar dos búsquedas, una por cadena y la otra por número y guardar sus resultados en memoria. El programa principal ***podría*** tener en concreto las siguientes partes:

```
cpu     68000
include /usr/local/68k/sermones.inc

org     $1000
; Inicialización de los parámetros para la llamada a BUSCANUM
; Llamada a BUSCANUM
; Guardar el resultado
; Inicialización de los parámetros para la llamada a BUSCACAD
; Llamada a BUSCACAD
; Guardar el resultado
; Final

org     $2000
; Declaración de todas las variables/estructuras
```

.


Memoria
Cargar de memoria
Cargar a memoria
Cargar del Y
Cargar del IR
Cargar del PC
PC
ADDR
Reloj
Parar
Contador
Poner a cero
Cargar de memoria
Cargar de memoria
INCREMENTAR
Cargar del IR
IR
Control
Y
ACumulador
INCREMENTAR
DECREMENTAR
Cargar del IR
Cargar de la ALU
Cargar de memoria
Cargar de memoria
Cargar del IR
X
ALU
Flag
Cargar del AC
Seleccionar ADD
Seleccionar SUB
Seleccionar AND
Seleccionar OR
Seleccionar NOT
Seleccionar SHR
Seleccionar SHL
Cargar de la ALU
Este es el conjunto de líneas para carga de registros, selección de operaciones en la ALU, etc. que aparecen en verde

eman ta zabal zazu
universidad euskal herriko
del país vasco unibertsitatea

departamento de electricidad y electrónica
elektrika eta elektronika saila

Examen de S.E.T.I.

1er curso de Ingeniería Electrónica
4 de febrero de 2005

SOLUCIONES Y COMENTARIOS

1 El primer teorema de Shannon es aplicable a la codificación de mensajes emitidos por una fuente. Los ficheros en un ordenador pueden ser interpretados como mensajes donde los símbolos son p.ej. los bytes que los forman de modo que el alfabeto (en este caso de considerar bytes) contará con 256 símbolos. Su estructura frecuencial puede considerarse la consecuencia de una estructura interna de una fuente que hubiese emitido la secuencia en cuestión.

El teorema es aplicable en el doble sentido que supone la determinación de la existencia de un máximo para la compactación así como la indicación de que dicho límite es alcanzable (en un proceso que puede ser infinito –aunque en la práctica no lo sea-).

Nota.- muchos alumnos han indicado que no es aplicable al entender que el término "aplicable" hace referencia a ser la "solución" al problema de la compactación, algo así como ser "la fórmula" o el "algoritmo" de compactación, y esto es cierto también (lo cual no exime de explicar la relación del teorema con el problema de la compactación)

2 La Máquina Universal de Turing (MUT) es una Máquina de Turing específica (aún cuando existan infinitas MUT dependiendo de codificaciones y reglas de funcionamiento) cuyo cometido consiste en la emulación del comportamiento de una Máquina de Turing cualquiera. Su ejecución frente a un determinado problema será funcionalmente equivalente a la de la máquina que emula y para realizarla ha de disponer de -además del problema a resolver- una descripción de la máquina a emular. De este modo la MUT es un mecanismo capaz de resolver cualquier problema con solución (esto es una conjetura, la de Church-Turing) en base a que para todo problema con solución puede establecerse una Máquina de Turing específica que lo resuelva.

3

Las estructuras pueden ir en un fichero cabecera (.h) y ser genéricas como las siguientes:

```
typedef struct {double d1,d2;} DoublePar;
typedef struct {double d1,d2,d3;} DoubleTrio;
```

La primera puede utilizarse para almacenar la salida –las dos raices reales- y la segunda para aportar a la rutina los tres coeficientes del polinomio, conviniendo que d1 se corresponde con el coeficiente del término cuadrático, d2 con el del término lineal y d3 con el independiente.

De este modo la rutina puede quedar como sigue, teniendo en cuenta que su valor de retorno nos indica si hay soluciones reales o no (0 solución real devuelta por la rutina, -1 solución compleja no calculada):

```
int raices(DoubleTrio poli, DoublePar *solucion) {
       double det = poli.d2 * poli.d2 – 4 * poli.d1 * poli.d3;
       if (det<0) return -1;
       det = sqrt(det);
       solucion->d1 = (-poli.d2 + det)/(2*poli.d1);
       solucion->d2 = (-poli.d2 - det)/(2*poli.d1);
       return 0;
 }
```

4 a) Para poder ejecutar subrutinas necesitaremos en primer lugar **dos instrucciones**: una para ir a la subrutina y otra para volver de ella. La vuelta desde la subrutina requiere que se recuerde el punto de vuelta de modo que será también necesario algo al respecto. Si nos limitamos a disponer de un registro que almacene la dirección de retorno, sólo será posible disponer de un nivel de subrutina (es decir, las subrutinas no pueden llamar a su vez a subrutinas) puesto que una vez utilizado dicho registro no hay otro disponible. La solución general al problema consiste en la utilización de una estructura de pila en la memoria de modo que necesitaremos un registro que actuará como **apuntador de pila**. Cada una de las instrucciones tiene por misión cambiar el apuntador de programa y gestionar la pila: la de salto debe **apilar la dirección de retorno y poner en el PC la de la rutina**, mientras que la de retorno debe **sacar la dirección que se encuentra en la pila y situarla en el PC.**

Una vez establecidas las necesidades, veamos cómo resolver el problema. Para la inclusión de las dos nuevas instrucciones podemos adoptar diversas soluciones, que pueden ir desde eliminar dos existentes y que consideremos razonablemente prescindibles a favor de las nuevas (p.ej podríamos prescindir del incremento y del decremento ya que pueden ser llevados a cabo mediante sumas o restas), hasta establecer una codificación con más bits para dar cabida a más instrucciones. Una solución más elaborada puede ser reestructurar la codificación de las instrucciones atendiendo

al hecho de que muchas de ellas no requieren de operandos y por tanto no utilizan todos los direccionamientos (seguiremos esta vía en la segunda parte).

Lo referente a la pila se resuelve con la inclusión de un nuevo registro interno (le llamaremos SP naturalmente) conectado como sea necesario a los ya existentes en función de cómo se lleven a cabo los flujos de datos hacia y desde él, cosa que depende de cómo resolvamos los pasos de la ejecución de las instrucciones.

b) Las instrucciones al equipo podrían ser las siguientes:

- Altérese la función combinatoria que gestiona las líneas de control para considerar también la configuración **11** de los dos bits más altos en el código de instrucción (los que afectan al direccionamiento) y de este modo hágase que las instrucciones que no tienen operandos pasen a codificarse con este patrón. Las dos nuevas instrucciones (JSR y RTS) se codificarán de manera consecuente con este nuevo esquema.

operación	*dir. directo*			*dir. inmediato*			*dir. indirecto*		
Sumar al AC	000000	0	ADD	010000	16	ADD-C	100000	32	ADD-I
Restar del DC	000001	1	SUB	010001	17	SUB-C	100001	33	SUB-I
Y lógico con el AC	000010	2	AND	010010	18	AND-C	100010	34	AND-I
O lógico con el AC	000011	3	OR	010011	19	OR-C	100011	35	OR-I
	000100	4		010100	20		100100	36	
	000101	5		010101	21		100101	37	
	000110	6		010110	22		100110	38	
	000111	7		010111	23		100111	39	
Salto a subrutina	**001000**	**8**	**JSR**	011000	24		**001000**	**40**	**JSR-I**
Cargar AC de memoria	001001	9	LOD	011001	25	LOD-C	101001	41	LOD-I
Descargar AC en memoria	001010	10	STO	011010	26		101010	42	STO-I
	001011	11		011011	27		101011	44	
Saltar	001100	12	JMP	011100	28		101100	44	JMP-I
Saltar si AC=0	001101	13	JMZ	011101	29		101101	45	JMZ-I
Saltar si AC<0	001110	14	JMN	011110	30		101110	46	JMN-I
Saltar si FLAG activado	001111	15	JMF	101111	31		101111	47	JMF-I

operación	*dir. implícito*		
Incrementar el AC	**110000**	**48**	INC
Decrementar el AC	**110001**	**49**	DEC
	110010	50	
	110011	51	
NO lógico del AC	**11**0100	**52**	NOT
Desplazar a izquierda el AC	**11**0101	**53**	SRL
Desplazar a derecha el AC	**11**0110	**54**	SHR
	110111	55	
Retorno de subrutina	**111000**	**56**	**RTS**
	111001	57	
	111010	58	
Parar	**11**1011	**59**	HLT
	111100	60	
	111101	61	
	111110	62	
	111111	63	

- Añádase un nuevo registro de 10 bits (SR) con capacidad de autoincremento y autodecremento, con salida hacia el ADDR y que se cargue inicialmente –al arránque de la máquina- con el valor 0.
- Añádanse las líneas de control siguientes:
  - Incrementar el SP
  - Decrementar el SP
  - Cargar el ADDR del SP
  - Cargar Memoria del PC

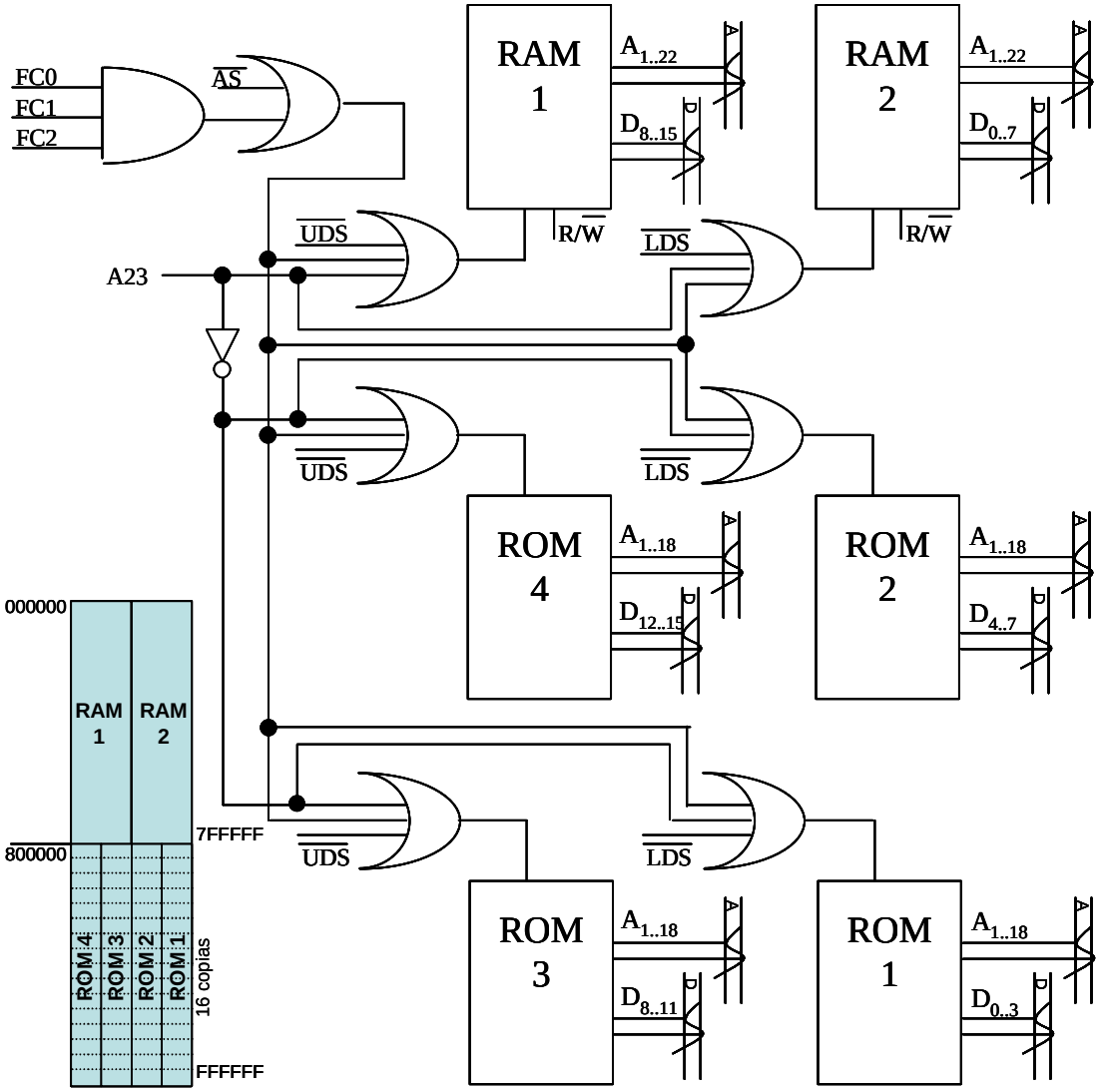
  y gestiónense mediante la función combinatoria adecuadamente para que la ejecución de las nuevas instrucciones sea la siguiente:

JSR
4- Decrementar el SP
5- Cargar el ADDR del SP
6- Cargar Memoria desde el PC
7- Cargar el PC desde el IR
8- Poner COUNT a cero

JSR-I
4- Decrementar el SP
5- Cargar el ADDR del SP
6- Cargar Memoria desde el PC
7- Cargar el ADDR desde el IR
8- Cargar PC desde Memoria
9- Poner COUNT a cero

RTS
4- Cargar ADDR del SP
5- Incrementar el SP
6- Cargar el PC de Memoria
7- poner COUNT a cero

5 La siguiente es una posible solución:



6

```
; Subrutina:     BUSCANUM
; Descripción:   Busca el número asociado a una cadena de caracteres
; Entrada:       A0.L  dirección de la agenda
;                A1.L  dirección de la cadena a buscar
; Salida:        D0.L  número asociado o -1 si no existe
; Modifica:      CCR
BUSCANUM  MOVEM.L A0/D1,-(A7)
          MOVEQ   #-1,D1        ; es más rápido comparar frente a D1
          BRA.S   While         ; mientras no lleguemos al final
Do        BSR.S   STRCMP        ;   comparamos las cadenas
          TST.L   D0            ;   si son iguales
          BEQ.S   Fuera         ;      break  (salimos del ciclo)
          ADDQ.L  #4,(A0)       ;   pasamos a la siguiente entrada
While     CMP.L   (A0)+,D1      ;
          BNE.S   Do            ; fmientras
Fuera     MOVE.L  -4(A0),D0     ; da igual cual es la razón de terminar,
          MOVEM.L (A7)+,A0/D1   ; la solución es siempre "-4(A0)" (*)
          RTS
```

(*) Aquí hacemos uso de el último elemento de la agenda tiene el número “-1”

```
; Subrutina:    BUSCACAD
; Descripción:  Busca la cadena asociada a un número
; Entrada:      A0.L  dirección de la agenda
;               D0.L  Número a buscar
; Salida:       A1.L  dirección de la cadena o 0 si no existe
; Modifica:     CCR
BUSCACAD  MOVEM.L A0/D1,-(A7)
          MOVEQ   #-1,D1      ; es más rápido comparar frente a D1
          BRA.S   While       ; mientras no lleguemos al final
Do        CMP.L   (A0),D0     ;    si los números son iguales
          BEQ.S   Fuera       ;       break  (salimos del ciclo)
          ADDQ.L  #8,(A0)     ;    pasamos a la siguiente entrada
While     CMP.L   (A0),D1     ;
          BNE.S   Do          ; fmientras
Fuera     MOVEA.L 4(A0),A1    ; da igual cual es la razón de terminar,
          MOVEM.L (A7)+,A0/D1 ; la solución es siempre “4(A0)” (*)
          RTS
```

(*) Aquí hacemos uso de que último elemento de la agenda tiene la dirección “0”

; Programa Principal

```
          cpu       68000
          include /usr/local/68k/sermones.inc

          org       $1000

          LEA       AGENDA,A0
          LEA       BUSCA1,A1
          BSR.S     BUSCANUM
          MOVE.L    D0,NUMERO

          MOVE.L    BUSCA2,D0
          BSR.S     BUSCACAD
          MOVE.L    A1,NOMBRE
          SERMON    FPROG
```

```
          org       $2000
AGENDA    DC.L      943724563
          DC.L      CADENA1
          DC.L      656754885
          DC.L      CADENA2
          DC.L      946018845
          DC.L      CADENA3
          DC.L      975652345
          DC.L      CADENA4
          DC.L      609489852
          DC.L      CADENA5
          DC.L      653215487
          DC.L      CADENA6
          DC.L      944586123
          DC.L      CADENA7
          DC.L      -1
          DC.L      0

CADENA1   DC.B      “Jose Ignacio”,0
CADENA2   DC.B      “Ibone”,0
CADENA3   DC.B      “Aitor Madina”,0
CADENA4   DC.B      “Carlos Kortabarria”,0
CADENA5   DC.B      “Txema”,0
CADENA6   DC.B      “Ainara Legorburu”,0
CADENA7   DC.B      “Jose Escudero”,0

BUSCA1    DC.B      “Pepe Gotera”,0
NUMERO    DS.L      1
BUSCA2    DC.L      9443465234
NOMBRE    DS.L      1
```