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
19 de junio de 2002

Cuestiones teórico-prácticas. 1 punto cada una

1 La demostración de que el Problema de la Parada es no computable se realiza por Reducción al Absurdo. Se supone que existe la máquina que lo resuelve (sea H) y a continuación se construye una máquina K formada como RHC donde R es una etapa inicial "replicadora" y C es una etapa final que actúa al contrario de la salida dada por H. De este modo tenemos una máquina que actúa al contrario de lo que H indica y por tanto la premisa de que H existe es falsa. ¿Por qué decimos que K actua al contrario de lo que H indica?

2 UNICODE determina una codificación de caracteres que, en principio, pretendía la utilización de 16 bits. El formato "normal" de dicha codificación es de tipo "Bloque", pero el estándar incluye otros formatos como, por ejemplo, el UTF-8. En UTF-8 se distinguen tres rangos de códigos (es decir el rango total: 0000_{hex}-$FFFF_{hex}$ se subdivide en tres con distinto tratamiento) ¿Cuales son estos rangos y cuál es el criterio para situar un carácter en ellos?. ¿En qué zona situaría los siguientes grupos de caracteres? (razónelo)

a) símbolos matemáticos
b) nuestro alfabeto en mayúsculas
c) iconos que representen utilidades en telefonía (p.ej. un teléfono descolgado)
d) el alfabeto ruso

3 Tenemos una fuente de información que emite mensajes con un vocabulario $\{S_1, S_2, S_3, S_4, S_5, S_6\}$ y sabemos que la frecuencia de estos símbolos es {1/2,1/4,1/16,1/16,1/16,1/16}. Pretendemos codificar los mensajes con el siguiente código {0,10,110,1110,1011,1101}. Calcule la entropía de la fuente y la longitud media obtenida mediante la codificación indicada. Realice los comentarios que le surgan a raiz del resultado.
NOTA.-la correspondencia entre frecuencias y códigos es la dada por el orden en que se presentan.

4 A continuación se muestra la tabla de instrucciones del procesador virtual utilizado en las clases de la asignatura. ¿Por qué la primera columna está completa y no así las otras dos?, o dicho de otro modo, ¿es lógico que todas las instrucciones tengan direcionamiento directo pero algunas carezcan de los otros? (estudiar las instrucciones por bloques, no individualmente)

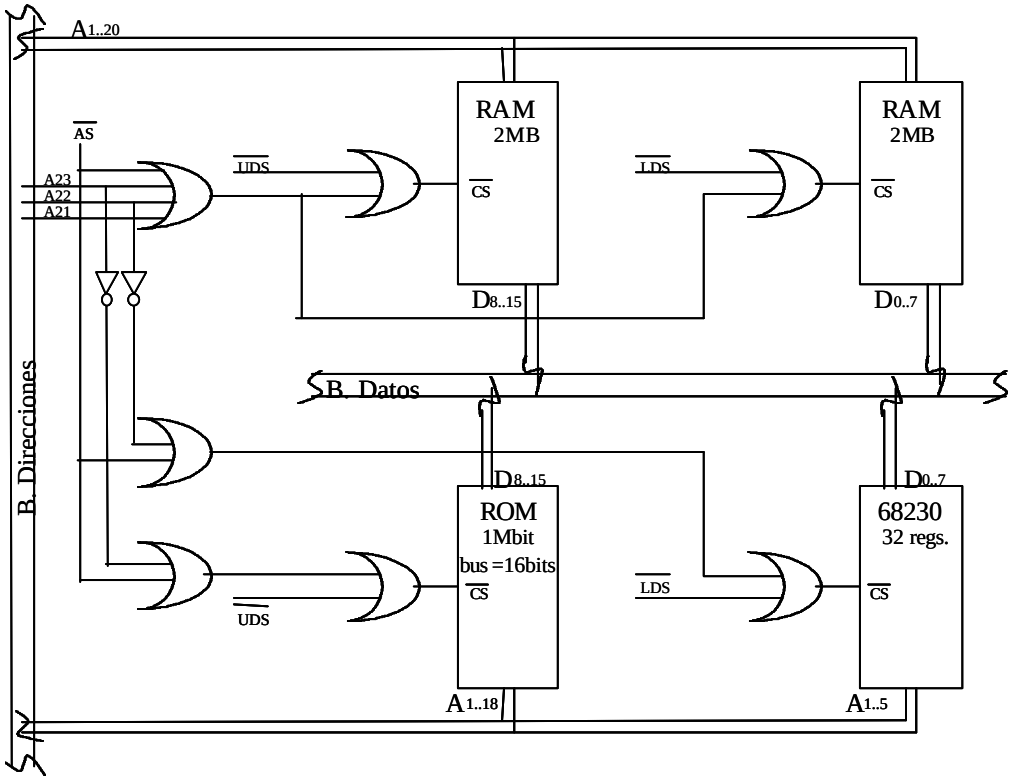
operación	*dir. directo*		*dir. inmediato*			*dir. indirecto*		
Sumar al AC	000000 0	ADD	010000	16	ADD-C	100000	32	ADD-I
Restar del AC	000001 1	SUB	010001	17	SUB-C	100001	33	SUB-I
Y lógico con el AC	000010 2	AND	010010	18	AND-C	100010	34	AND-I
O lógico con el AC	000011 3	OR	010011	19	OR-C	100011	35	OR-I
NO lógico del AC	000100 4	NOT						
Desplazar a izquierda el AC	000101 5	SHL						
Desplazar a derecha el AC	000110 6	SHR						
Incrementar el AC	000111 7	INC						
Decrementar el AC	001000 8	DEC						
Cargar AC de memoria	001001 9	LOD	011001	25	LOD-C	101001	41	LOD-I
Descargar AC en memoria	001010 10	STO				101010	42	STO-I
Parar	001011 11	HLT						
Saltar	001100 12	JMP				101100	44	JMP-I
Saltar si AC=0	001101 13	JMZ				101101	45	JMZ-I
Saltar si AC<0	001110 14	JMN				101110	46	JMN-I
Saltar si FLAG activado	001111 15	JMF				101111	47	JMF-I

Mapeo de memoria. 2 puntos

5 A un diseñador novato se le ha encomendado la realización un mapa de memoria para un sistema con 68000 con las cuatro pastillas de la figura siguiente, que es la solución que éste ha dado. Evidentemente ha mezclado algún diseño antiguo con los componentes que le han indicado sin ocuparse de hacer los ajustes pertinentes.

Comente todos los errores y situaciones conflictivas que vea y proponga cambios para llegar a un diseño viable. Indique qué mapa resulta finalmente.

NOTA.- Las características indicadas en cada circuito son las correctas.



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

6 Un polinomio de coeficientes enteros puede representarse mediante un vector de enteros formado por el orden del polinomio y los coeficientes del mismo (de menor a mayor orden):

$3x^5 + 4x^4 - 2x^3 + 8x^2 - 12$

5	-12	0	8	-2	4	3

Suponiendo que tanto el orden como los coeficientes pueden ser representados mediante bytes (el orden será un entero sin signo; los coeficientes, enteros con signo), realizar las siguientes subrutinas y programa principal:

Subrutina ORDPOL

Esta subrutina deberá tomar la dirección de dos polinomios (A0 y A1) y ordenarlos, de forma que A0 termine apuntando al polinomio de mayor orden y A1 al de menor. La subrutina podría tener en concreto la siguiente descripción:

```
; Subrutina:     ORDPOL
; Descripción:   Ordena dos polinomios de mayor a menor orden
; Entrada:       A0.L  dirección del polinomio P1
;                A1.L  dirección del polinomio P2
; Modifica:      A0, A1, ???
```

Subrutina ADDPOL

Esta subrutina deberá tomar la dirección de dos polinomios y obtener la suma de ambos en un tercer polinomio cuya dirección también tomará como entrada. Téngase en cuenta que es posible sumar polinomios de distinto orden. La subrutina podría tener en concreto la siguiente descripción:

```
; Subrutina:     ADDPOL
; Descripción:   Calcula la suma de dos polinomios
; Entrada:       A0.L  dirección del polinomio P1
;                A1.L  dirección del polinomio P2
;                A2.L  dirección donde se guarda el polinomio P1+P2
; Modifica:      ???
```

Nota: se recomienda hacer uso de la subrutina ORDPOL

Programa principal

El programa principal deberá declarar dos polinomios,

$4x^3 - 7x^2 – x$
$3x^5 + 4x^4 - 2x^3 + 8x^2 – 12$

, reservar espacio suficiente para un tercero (la suma de los dos anteriores) y realizar la suma de ambos polinomios haciendo uso de la función ADDPOL. El programa principal podría tener en concreto las siguientes partes:

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

org     $1000

; Inicialización de los parámetros de entrada antes de la llamada a subrutina

;Llamada a ADDPOL

;Final

org     $2000

; Declaración de todas las variables/estructuras.
```

Examen de S.E.T.I.

1er curso de Ingeniería Electrónica

19 de junio de 2002

SOLUCIONES y COMENTARIOS

1 Decir que existe H es decir que para todo par $[D_M,P]$, donde D_M es la descripción de una máquina M y P un problema que se le plantea a M, H nos indica si la ejecución termina o no.

Decimos que K actua al contrario de lo que H indica por lo que sucede cuando la ponemos a funcionar frente al problema consistente en su propia descripción (D_k). La máquina K comienza por replicar dicha descripción, de modo que al llegar a su segunda fase en la cinta se tiene $[D_k, D_k]$. A partir de ese momento la máquina H estará por tanto analizando la actuación de la máquina K frente al problema D_k, que es precisamente lo que se está ejecutando, y una vez lo haya hecho, por construcción la tercera fase de K actuará en contra de lo indicado por H.

2 Los tres formatos de UTF-8 utilizan 1, 2 y 3 bytes en cada caso para representar códigos de 7, 11 y 16 bits respectivamente. Por tanto es adecuado utilizar los códigos de 1 byte para los caracteres más frecuentes y los de 3 bytes para los más infrecuentes de manera que, en terminos generales, la codificación de secuencias de caracteres sea lo más corta posible. De hecho, el primer bloque (el de 7 bits) está completamente ocupado por los caracteres ASCII, lo que nos dejaría la posibilidad de situar nuevos caracteres en los bloques de 11 bits (representados por 2 bytes) y de 16 bits (representados por 3 bytes).

De todos modos la asignación de códigos en UNICODE no se determina en función de UTF-8, sino que asigna zonas considerando la probabilidad de los conjuntos desde el máximo asociado a 0000 y el mínimo asociado a FFFF. La definición de UTF-8 es consistente con esto.

NOTA.: Estas cuatro situaciones se planteaban como una simple puesta en práctica de la respuesta a las preguntas, es decir, del emplazamiento en función de la frecuencia. Como quiera que su interpretación se ha mostrado muy ambigua, se comenta aquí lo siguiente:

a) Gran número de alumnos han considerado los símbolos matemáticos como parte del ASCII. Esto no es así. El soporte que este estándar puede dar a la escritura de matemáticas se limita a los operadores de suma, resta (o negativo) y división, ademas de alguna posible coincidencia de símbolos con los de puntuación en textos. Podría parecer razonable situar este grupo en el segundo bloque, pero en realidad el espacio disponible no es siquiera suficiente para acoger a todos los alfabetos de lenguas actualmente vivas, de modo que, de hecho, los símbolos matemáticos ya se encuentran "mapeados" en posiciones del tercer bloque.

La cantidad de códigos que puede incluir cada rango es la siguiente:

Bytes	bits	capacidad	rango utilizado	número de códigos
1	7	2^7=128	0..127	128
2	11	2^{11}=2048	128..2047	1920
3	16	2^{16}=65536	2048..65535	63488

b) La interpretación del término "nuestro" ha sido también poco uniforme. En pricipio puede considerarse como "nuestro" el que se utiliza aún hoy en día con más frecuencia en informática que es en realidad el inglés, y que se encuentra en la zona de 1 byte (ASCII). Si nos referimos al alfabeto que incluye las "ñ" "ü" y vocales acentuadas, tendremos parte en la zona de 1 byte (la coincidente con el inglés) y parte en la de 2 bytes. En principio no tiene mucho sentido definir un nuevo conjunto conexo con este alfabeto despreciando las coincidencias con el inglés, ya que supondría pérdida de la ventaja en concisión y sería origen de ambigüedades. En cuanto a la posible ventaja de situar el carácter "ñ" en el lugar que le corresponde de acuerdo con la ordenación esablecida, no es generalizable a los demás caracteres no ASCII, por lo que no parece razonable la definición de un nuevo conjunto tampoco por este motivo.

c) Estos caracteres no se utilizarán en ningún caso para componer mensajes sino que podrán aparecer esporádica y puntualmente, por lo que su frecuencia de uso será muy baja frente a cualquier conjunto que constituya un alfabeto para componer mensajes.

d) No parece muy razonable (como sugieren algunos alumnos) situar el alfabeto ruso en la zona de 24 bits considerando que se utiliza poco, ya que quienes lo utilicen lo harán posiblemente con tanta o más frecuencia que otros alfabetos, de modo que habrá que proporcionarles una codificación lo más corta posible.

Puede ser razonable situar los rangos de caracteres indicados como sigue:

a)	símbolos matemáticos	zona 2 bytes*
b)	nuestro alfabeto en mayúsculas	zona 1 byte
c)	iconos que representen utilidades en telefonía (p.ej. un teléfono descolgado)	zona 3 bytes
d)	el alfabeto ruso	zona 2 bytes

3 Haciendo los cálculos, la entropía resulta 2 bits/símbolo y la longitud media 1.9375 binits/símbolo. Esto evidencia que la codificación no puede estar bien ya que el valor de la entropía nos da un límite mínimo a la longitud media de todo código. Puede comprobarse fácilmente que el código no es unívoco con un caso concreto: la secuencia 101110 puede representar dos mensajes distintos en función de que se interprete como 10-1110 ó 1011-10.

4 Las instrucciones NOT, SHL, SHR, INC, DEC y HLT no tienen operando explícito, de modo que realmente no necesitan de ningún modo de direccionamiento, y aunque estén en la columna encabezada como "direccionamiento directo", esto no es aplicable a ellas. En cierto modo, siendo estrictos, esto no es consistente, y dado que podriamos "mapear" una cuarta columna, podría considerarse mejor cambiar estos códigos como se verá más abajo. La operación STO carece de operando constante ya que no tiene sentido: el acumulador se "descarga" sobre una posición de memoria dada de un modo directo o indirecto. Lo mismo ocurre con las cuatro instrucciones de salto: no tiene sentido especificar un valor constante, sino una posición de memoria, bien directamente o indirectamente.

Aunque no se pedía en el enunciado del ejercicio, la siguiente tabla muestra una disposición más estructurada del conjunto de instrucciones de la máquina .

operación	*dir. directo*	*dir. inmediato*	*dir. indirecto*	*dir. implícito o no dir.*
Sumar al AC	000000 0 ADD	010000 16 ADD-C	100000 32 ADD-I	
Restar del AC	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	
NO lógico del AC				110100 52 NOT
Desplazar a izquierda el AC				110101 53 SHL
				110110 54 SHR
Desplazar a derecha el AC				110111 55 INC
				111000 56 DEC
Incrementar el AC	001001 9 LOD	011001 25 LOD-C	101001 41 LOD-I	
Decrementar el AC	001010 10 STO		101010 42 STO-I	
Cargar AC de memoria				111011 59 HLT
Descargar AC en memoria	001100 12 JMP		101100 44 JMP-I	
Parar	001101 13 JMZ		101101 45 JMZ-I	
Saltar	001110 14 JMN		101110 46 JMN-I	
Saltar si AC=0	001111 15 JMF		101111 47 JMF-I	
Saltar si AC<0				
Saltar si FLAG activado				

5 El primer error es el conjunto de lineas de dirección que se asigna a las memorias RAM. Al ser de 2MB cada una deben tener 21 líneas de direcciones ($2MB=2^{21}$). Por lo tanto será tambien un error incluir la linea A21 en la decodificación.

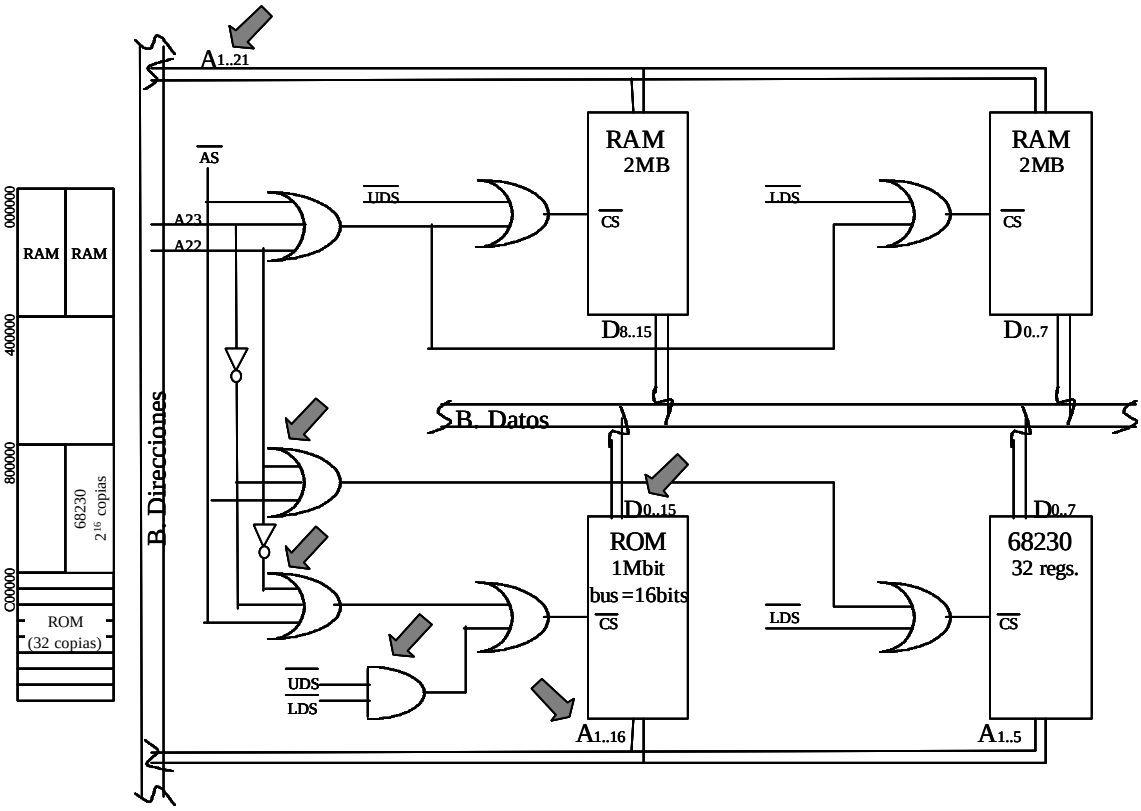
Del mismo modo el número de líneas de direcciones para la ROM es incorrecto. Como se trata de un circuito de 1Mbit estructurado en words de 16 bits, el número de elementos direccionables es $2^{20}/16=2^{16}$ y por tanto contará con 16 líneas de dirección.

Además, para este mismo circuito la asignación de líneas del bus de datos es también incorrecta. Como se indica que esta estructurado en words de 16 bits, debe conectarse a todo el bus de datos. En consecuencia también es incorrecto el uso de UDS únicamente, ya que debe poder accederse en cualquier

caso (LDS, UDS, o ambas) y por tanto hay que utilizar una combinación adecuada de LDS y UDS (ver figura).

Por último,las direcciones asignadas a ROM y 68230 tienen una zona de conflicto en el cuarto más alto del mapa de memoria (con A23 y A22 a 1 se activan ambas). Puede solucionarse el problema incluyendo el uso de A22 negada en la decodificación de la 68230, con lo que dejará de ocupar el último cuarto del espacio direccionable y por tanto no entrará en conflito con la ROM. En la figura se muestra otra solución -un poco más "elegante"- donde A23 y A22 se utilizan en ambas decodificaciones evitando el conflicto y dejando libre un cuarto del espacio direccionable.

```
  Axx->23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 [00]
RAM(1)  0  0  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  UDS
RAM(2)  0  0  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  LDS
68230   1  0  x  x  x  x  x  x  x  x  x  x  x  x  x  x  x  .  .  .  .  .  .  LDS
ROM     1  1  x  x  x  x  x  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  UDS&LDS
```



El enunciado del problema se centraba en la resolución de los errores de mapeo en el espacio de direccionamiento, por lo que la solución mostrada es suficiente. Evidentemente para llevar este diseño hasta el punto en que pueda ser tomado como soporte para la construcción de un sistema real faltan algunos aspectos como la inclusión de las señales de R/W y la negación del CS en el caso de ciclos de reconocimiento de interrupción.

Programación del 68000.

6

```
; Subrutina:    ORDPOL
; Descripción:  Ordena dos polinomios de mayor a menor orden
; Entrada:      A0.L  dirección del polinomio P1
;               A1.L  dirección del polinomio P2
; Modifica:     A0, A1
ORDPOL  MOVE.B  D0, -(A7)     ; se guarda D0 en la pila
        MOVE.B  (A0), D0
IF      CMP.B   (A1), D0      ; if orden(P1) > orden(P2)
        BHI.S   FIN           ;   then  nada
        EXG     A1, A0        ;   else  intercambiar los registros
FIN     MOVE.B  (A7)+, D0     ; recuperar D0 de la pila
        RTS


; Subrutina:    ADDPOL
; Descripción:  Calcula la suma de dos polinomios
; Entrada:      A0.L  dirección del polinomio P1
;               A1.L  dirección del polinomio P2
;               A2.L  dirección donde se guarda el polinomio P1+P2
; Modifica:
ADDPOL  MOVEM.L A0/A1/D0/D1, -(A7)      ; guardar en pila
        BSR.S   ORDPOL                  ; orden(P1)> orden(P2)
        MOVE.B  (A0), (A2)              ; orden(P3) = orden(P1)
        MOVEQ   #1, D0                  ;
        ADD.B   (A0), D0                ; D0= 1 + orden(P1)
        BRA.S   FOR                     ; for (D0= orden(P1) ; D0>=0 ; D0--)
DO      MOVE.B  1(A0,D0.W), 1(A2,D0.W)  ;   coef_P3[D0] = coef_P1[D0]
IF      CMP.B   (A1), D0                ;   if  (D0 > orden(P2))
        BHI.S   FOR                     ;     then  nada
        MOVE.B  1(A1,D0.W), D1          ;     else  coef_P3[D0] += coef_P2[D0]
        ADD.B   D1, 1(A2,D0.W)          ;
FOR     DBRA    D0, DO                  ;
        MOVEM.L (A7)+, A0/A1/D0/D1
        RTS

; PROGRAMA PRINCIPAL
        cpu     68000
        include  /usr/local/68k/sermones.inc

        org     $1000

        LEA     POL_1, A0
        LEA     POL_2, A2
        LEA     POL_3, A3
        BSR.S   ADDPOL
        sermon  fprog

        org     $2000

POL_1   DC.B    3,0,-1,-7,4
POL_2   DC.B    5,-12,0,8,-2,4,3
POL_3   DS.B
```