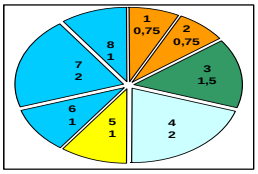
eman ta zabal zazu
universidad euskal herriko
del país vasco unibertsitatea

departamento de electricidad y electrónica
elektrika eta elektronika saila

Examen de S.E.T.I.

1er curso de Ingeniería Electrónica
5 de septiembre de 2005



Cuestiones. 0,75 + 0,75 puntos

1 AT&T cierra sus puertas. La empresa que en su día fuera la mayor del mundo, con una envergadura tres veces superior a la segunda (la General Motors), llegó a emplear a un millón de personas, pero es sin duda alguna más admirable que sus colosales dimensiones su política de inversión en I+D. En sus mejores tiempos llegó a dedicar más del 4% de sus ingresos brutos a este fin, dando lugar a una enorme contribución al avance de la ciencia y la tecnología con más de 30.000 patentes y seis premios Nobel en física. De la larga lista de gente de reconocido prestigio que llevó a cabo sus contribuciones en la AT&T, algunos nombres han aparecido en esta asignatura. Diga en una frase cual es su contribución.

Claude E. Shannon	William Shockley
Ken Thompson	John Bardeen
Dennis Ritchie	Walter Brattain
Brian Kernighan	Bjarne Stroustrup

2 El pensamiento de Alan Turing ha dado lugar básicamente a una contribucion central y algunas otras menores. Esta contribución central tiene varios aspectos o consecuencias diferenciables. Enumere, sin detallar en que consisten, estas contribuciones (central y menores) o aspectos.

Ejercicio de C. 1,5 puntos

3 Lea la explicación sobre generación de números aleatorios dada en el ejercicio 6 y escriba las mismas dos rutinas y programa principal alli pedidas, pero en este caso en lenguaje C. Ponga cuidado en la elección y manejo de tipos numéricos.

Estructura de un microprocesador. 2 puntos

4 Una diferencia entre el procesador virtual estudiado y el real (68000) consiste en que el segundo considera el contenido de las primeras posiciones de la memoria de modo especial. En particular los dos primeros elementos (longwords) se consideran como direcciones iniciales de una pila y una rutina de arranque. Comente qué alteraciones serían necesarias en el hardware y software del procesador virtual para dotarle de esta capacidad. No es necesario que lo rediseñe hasta el último detalle, sino que explique todos los aspectos a tener en cuenta con la suficiente precisión como para que su puesta en práctica sea sencilla.

Mapeo de memoria. 1 punto

5 Diseñe el circuito necesario para que un computador con microprocesador 68000 disponga de 8 Mbytes de RAM situados a partir de 0, mediante pastillas 4Mbit de ancho 1 bit. No se impone ninguna restricción sobre la posibilidad de que existan copias de las pastillas en otras posiciones del espacio direccionable aparte de que quede espacio libre para otros elementos (muestre el mapa resultante).

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

6 La generación de números aleatorios es una área en continuo desarrollo desde la decada de los 60. Una rutina de generación de números aleatorios debería ser capaz de generar (devolver) un número aleatorio cada vez que sea ejecutada. Es por ello que podemos decir que dichas rutinas generan series. Dichas series deberían tener la propiedad de ser imposible predecir cual será el siguiente número.

Sin embargo, ningún método algorítmico de generación de números aleatorios es perfecto, ya que el mero hecho de ser algorítmico lo hace predecible, y por tanto deja de ser aleatorio. Es por ello, que suele hablarse de generación de números pseudo-aleatorios. Cuanto más impredecible “parezca” el resultado del algoritmo, diremos que dicho algoritmo

cumple mejor las propiedades aleatorias que pretendemos emular. Todas las rutinas de números pseudoaleatorios se basan en definir de una u otra forma una secuencia finita de números y devolverlos de uno en uno, partiendo de un valor inicial. Dado que la secuencia es finita, la rutina genera una serie inevitablemente periódica.

Existen numerosos métodos para obtener secuencias pseudoaleatorias de números enteros. Uno de los más extendidos (no por ello más acertado) es el denominado LCG (Linear Congruential Generator), el cual obtiene un número de la serie a partir del anterior haciendo uso de la ecuación:

$$\mathbf{X_{n+1}} = (a * \mathbf{X_n} + c) \bmod m$$

Donde ***mod*** se refiere a módulo (resto de la división entera). Su representación simplificada es **LCG(a,c,m,X_0)**, donde X_0 es el valor inicial o "semilla" de partida (se entiende que estamos tratando con numeros enteros positivos o aritmética sin signo). La selección de los parámetros **a c** y **m** es crucial a la hora de generar una rutina que tenga propiedades aleatorias adecuadas, sin embargo no nos preocuparemos ahora por ello y utilizaremos el siguiente generador:

$$\mathbf{LCG(16807,13,2^{16},X_0)}$$

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

Subrutina INITSEED

Esta subrutina deberá tomar un número de 16 bits e inicializa la semilla para la generación de números aleatorios. La subrutina ***podría*** tener en concreto la siguiente descripción:

```
; Subrutina:      INITSEED
; Descripción:    Inicializa la semilla
; Entrada:        D0.W  semilla
; Modifica:       (rellenar debidamente)
```

Subrutina RANDOM

Esta subrutina devuelve el siguiente entero de 16 bits *sin signo* haciendo uso del generador LCG(16807,13,2^{16},X_0). La subrutina ***podría*** tener en concreto la siguiente descripción:

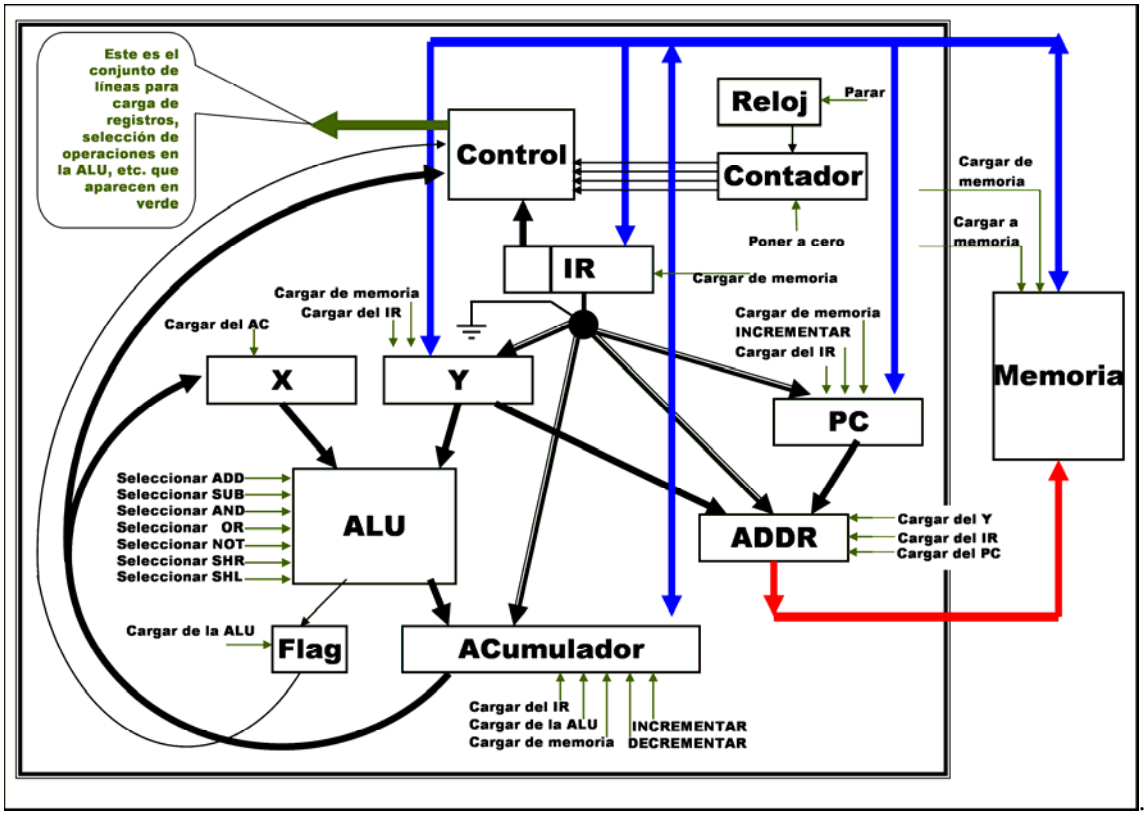
```
; Subrutina:      RANDOM
; Descripción:    Obtiene el siguiente entero de la serie LCG(16807,13,2^16,X0).
; Salida:         D0.W  siguiente entero pseudos-aleatorio
; Modifica:       (rellenar debidamente)
```

Programa principal

El programa principal deberá inicializar con el valor 1234 la semilla de la rutina generadora de números aleatorios, así como generar 1000 números y calcular la media de los mismos (si la rutina es adecuada, dicho valor debiera ser cercano a $7FFF = 32767).

El programa principal ***podría*** tener en concreto las siguientes partes:

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

org     $1000
; Llamada a INITSEED
; Ciclo de 10000 iteraciones
;   Llamada a RANDOM
;   Actualizar la media
; Guardar el resultado
; Final

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


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

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

Claude E. Shannon – La teoría de la comunicación (información)
Ken Thompson – Desarrollo de Unix
Dennis Ritchie - Desarrollo de Unix y C
Brian Kernighan – Desarrollo de C
William Shockley – El transistor
John Bardeen – El transistor
Walter Brattain – El transistor
Bjarne Stroustrup – Desarrollo de C++

2

- La computación como el manejo de símbolos
- El establecimiento de los límites de lo computable (cálculo efectivo)
- La máquina de Turing
- El test de Turing

3 Las rutinas necesitan de una variable para almacenar en todo momento el último número proporcionado (o la semilla inicialmente). Esta variable ("`anterior`") será del formato elegido para los números de la serie: unsigned short. La rutina de inicialización de la semilla se limita a asignar valor a dicha variable. La rutina generadora realiza el cálculo correspondiente con el cuidado necesario sobre el valor intermedio ("`tmp`") que puede rebasar el tamaño "short". Para ello se utiliza "long". En todo caso no se utiliza "int" puesto que su tamaño no esta determinado en diferentes máquinas.

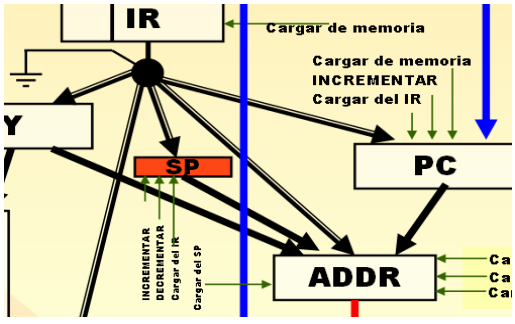
```
unsigned short anterior=0;
void initseed(unsigned short s) {anterior=s;}
unsigned short random() {
        unsigned long tmp=16807L*anterior+13;
        return anterior=tmp;        // equivalente a anterior=tmp % 65536L;

        }
```

El programa principal realiza la suma de los 1000 términos de la serie sobre una variable de tamaño largo asegurándonos de que la acumulación de valores no contará con los problemas de un formato menor.

```
unsigned long suma=0;
void main() {
        initseed(1234);
        int i=0; for (; i<1000;i++) suma+=(unsigned long)random();
        printf("La media es: %lu\n",suma/1000L);
        }
```

En esta solución se ha optado por no poner el postfijo de "unsigned" (u) a ninguna constante puesto que en todo caso son suficientemente pequeñas como para estar seguros de que su representación coincide con la "signed". Sí se pone el postfijo "long" (L) cuando es necesario (p. e. `65536L`) o incluso en ocasiones en que no lo es puesto que el compilador es capaz de determinarlo (p. e. `1000L`).

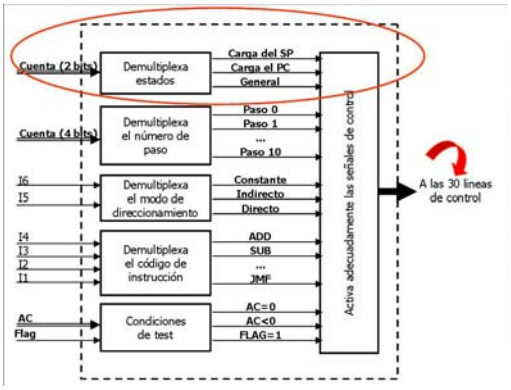
4 Lo primero que resulta evidente es que si pensamos en definir una dirección para una pila será porque queremos disponer de una pila y necesitaremos por tanto un apuntador, es decir un nuevo registro interno (SP). Este registro deberá poder ser incrementado y decrementado eficazmente por lo que habremos de dotarle de los circuitos adjuntos necesarios para disponer de estas funciones de un modo integrado (al igual que sucede con el acumulador). Por otra parte deberá poder determinar direcciones de memoria, es decir,



descargarse en el ADDR y, eventualmente, cargarse con valores dados por instrucciones, es decir, cargarse desde el IR.

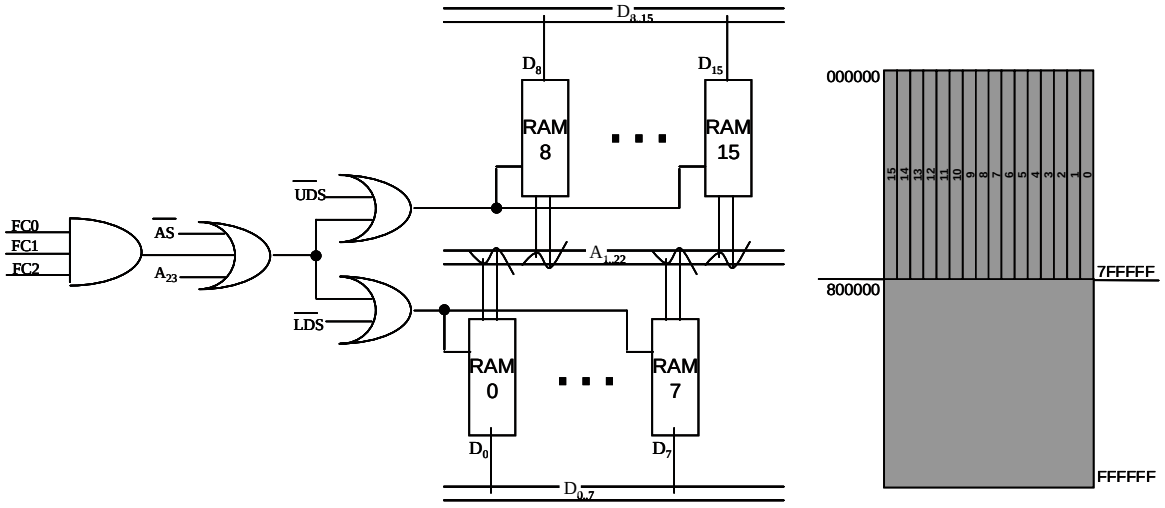
Pensemos ahora en el procedimiento de arranque: inicialmente el procesador no debe comenzar a ejecutar a partir de la dirección 0, sino que han de producirse unos ciclos previos similares a la ejecución "por la fuerza" de dos instrucciones:

1-2-3	(Fetch) Cargar ADDR desde el PC; Cargar IR desde memoria; Incrementar PC
4	Cargar SP desde IR
5	Poner COUNT a cero
1-2-3	(Fetch) Cargar ADDR desde el PC; Cargar IR desde memoria; Incrementar PC
4	Cargar PC desde IR
5	Poner COUNT a cero



Independientemente de cuales sean los bits que ocupen las posiciones de código de operación en las posiciones 0 y 1 de la memoria, habrá de ejecutarse lo anterior. Esto supone cambiar ligeramente la función combinacional general para que el paso 4 (el primero de la ejecución tras en fetch) actúe distinto en tres situaciones: la primera (cargar SP desde IR), la segunda (Cargar PC desde IR) y la general (según la función actual). Podemos llevar esto a cabo insertando un contador de 2 bits que pase por los estados 0,1,2,2,2,2,2,… con la misma señal utilizada para poner a cero el contador de estados internos.

5 La siguiente es una posible solución:



6

```
; Subrutina:      INITSEED
; Descripción:    Inicializa la semilla
; Entrada:        D0.W  semilla
; Modifica:       CCR

INITSEED   MOVE.W D0,LAST_RAND
           RTS

; Subrutina:      RANDOM
; Descripción:    Obtiene el siguiente entero de la serie LCG(16807,13,2^16,X0).
; Salida:  D0.W  siguiente entero pseudos-aleatorio
; Modifica:       CCR, D0.L

A          EQU     16807
C          EQU     13        ; X(i-1)=( A*X(i)+C ) mod 2^^16

RANDOM     MOVE.W  LAST_RAND,D0
           MULU    #A,D0
           ADDI.W  #C,D0
           MOVE.W  D0,LAST_RAND
           RTS

;(*) La variable LAST_RAND la guardamos junto a la subrutina
LAST_RAND  DC.W    0



; Programa Principal
           cpu     68000
           incluye /usr/local/68k/sermones.inc

           org     $1000
N          EQU     1000
SEMILLA    EQU     1234

           MOVE.W  #SEMILLA,D0      ; inicializamos la semilla
           BSR.S   INIT_SEED
           CLR.L   D1               ; inicializamos el sumatorio
           MOVE.W  #N,D2            ; ciclo for 1000 iteraciones
           BRA.S   for
do         BRS.S   RANDOM           ; obtenemos un número aleatorio
           ANDI.L  #$0000FFFF,D0    ; lo extendemos a 32 bits
           ADD.L   D0,D1            ; lo añadimos al sumatorio
for        DBRA    D2,do
           DIVU    #N,D1            ; dividimos el sumatorio por 1000
           MOVE.W  D1,MEDIA         ; guardamos el resultado
           SERMON  FPROG

           org     $2000
MEDIA      DS.W    1
```