2º Grado Informática
Estructura de Computadores
19 Septiembre 2012

Nombre:	
DNI:	**Grupo:**

Examen Test (3.0p)

Todas las preguntas son de elección simple sobre 4 alternativas.
Cada respuesta vale 3/30 si es correcta, 0 si está en blanco o claramente tachada, -1/30 si es errónea.
Anotar las respuestas (a, b, c o d) en la siguiente tabla.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

1) ¿Cuál de las siguientes direcciones no está alineada a *double* (8-byte)?

a) $1110110101110100)_2$

b) $1110110101101000)_2$

c) $1110110101110000)_2$

d) Todas están alineadas a double

2) ¿Cómo se devuelve en ensamblador x86-64 Linux gcc el valor de retorno de una función long int al terminar ésta?

a) La instrucción RET lo almacena en un registro especial de retorno.

b) Por convención se guarda en %eax.

c) Se almacena en pila justo encima de los argumentos de la función.

d) Ninguna de esas formas es la correcta

3) ¿Cuál afirmación es FALSA al comparar las arquitecturas x86 y x86-64?

a) El tamaño de un double es el mismo.

b) El tamaño de un puntero es el mismo.

c) El tamaño de un entero (int) es el mismo.

d) El tamaño de las posiciones de memoria es el mismo.

4) Considerar las siguientes declaraciones de estructuras en una máquina Linux de 64-bit.

```
struct RECORD {
int   value2;
short ref_count;
char  tag[10];
};
```

```
struct NODE {
long value;
struct RECORD record;
char string[8];
};
```

También se declara una variable global `my_node` como sigue:

```
struct NODE my_node;
```

Si la dirección de `my node` es 0x600940, ¿cuál es el valor de `&my node.record.tag[1]` ?

a) 0x60094a

b) 0x60094e

c) 0x60094f

d) Ninguna de las anteriores

5) En la pregunta anterior, ¿cuál es el tamaño de `my_node` en bytes?

a) 32

b) 40

c) 28

d) Ninguno de los anteriores

6) Respecto a la convención de llamada usada en Linux/gcc

a) Una subrutina que modifique algún registro debe restaurar su valor anterior antes de retornar.

b) Hay registros que pueden ser modificados libremente por las subrutinas, y otros que deben ser tratados como lo anteriormente dicho: si se modifican se deben restaurar. Y también hay registros especiales.

c) Hay registros modificables, otros que deben ser restaurados, y las subrutinas anidadas deben respetar los registros modificables que están en uso por otras subrutinas.

d) Todos los registros pueden ser modificados libremente por todas las subrutinas.

7) Respecto a direccionamiento a memoria en ensamblador IA-32 (sintaxis AT&T), de la forma D(Rb, Ri, S), sólo una de las siguientes afirmaciones es FALSA. ¿Cuál?

a) El desplazamiento D puede ser una constante literal (1, 2 ó 4 bytes).
b) EBP no se puede usar como registro base.
c) ESP no se puede usar como registro índice.
d) El factor de escala S puede ser 1, 2, 4, 8.

---

8) Las siguientes afirmaciones sugieren que el tamaño de varios tipos de datos en C (usando el compilador gcc) son iguales tanto en IA-32 como en x86-64. Sólo una de ellas es FALSA. ¿Cuál?

a) El tamaño de un `int` es 4 bytes.
b) El tamaño de un puntero es 4 bytes.
c) El tamaño de un `double` es 8 bytes.
d) El tamaño de un `short` es 2 bytes.

---

9) Estudiando el listado de una función C presuntamente compilada con gcc en modo 64bit (x86-64), nos dicen que la instrucción `movl (%rdi), %eax`, carga en el registro EAX el valor adonde apunta el primer argumento.

a) Está mal, porque EAX no se puede usar en modo 64bit, debería ser RAX.
b) Está mal, porque EAX no se carga con ningún valor.
c) Está mal, porque el primer argumento de una función C no se pasa en RDI.
d) Está bien, y pone a cero los 32 bits más significativos de RAX.

---

10) Se ha declarado en un programa C la variable `int val[5]={1,5,2,1,3}`. ¿Cuál de las siguientes afirmaciones es FALSA?

a) val[1] == 1
b) &val[3] == val+3
c) sizeof(val) == 20.
d) Todas son ciertas.

---

11) ¿Qué tipo de operaciones de E/S consume menos tiempo del procesador?
a) E/S programada
b) E/S mediante interrupciones
c) E/S mediante DMA
d) Todos consumen el mismo tiempo del procesador

---

12) ¿Cuál de las siguientes funciones no corresponde a un módulo de E/S?
a) Comunicación con el microprocesador
b) Comunicación con el dispositivo
c) Almacenamiento de programas
d) Almacenamiento temporal de datos

---

13) ¿En qué pareja de registros están el dato/instrucción que se leerá o escribirá en memoria, y la propia dirección de memoria?
a) MAR y ACUMULADOR
b) IR y ACUMULADOR
c) MBR y MAR
d) MBR y PC

---

14) ¿Con cuál de los siguientes dispositivos tendría sentido utilizar E/S programada sin consulta de estado?
a) Salida a un display de 7 segmentos
b) Entrada desde un disco duro
c) Salida a una impresora
d) Entrada desde un escáner

---

15) ¿Qué tipo de sincronización es más conveniente en el caso de tener dispositivos con distintos requisitos de temporización?
a) Síncrona
b) Asíncrona
c) No se pueden conectar dispositivos con distintos requisitos de temporización
d) No es necesario sincronizar el procesador con los dispositivos de E/S

---

16) En la ejecución de una instrucción...
a) el registro de instrucción se va incrementando para apuntar a la siguiente instrucción
b) la ALU realiza las operaciones aritméticas y lógicas
c) la UC activa las señales de control que envía por el bus de direcciones
d) siempre se altera el registro de estado

---

17) Sea un formato de microinstrucción que incluye dos campos independientes de 10 bits cada uno. Si se rediseña de modo que se solapen los dos campos, ¿cuántos bits se ahorran en cada microinstrucción?
a) 1
b) 9
c) 8
d) 4

---

18) Un computador tiene una memoria de control de 16000 palabras de 250 bits, de las que 447 son diferentes. ¿Cuántos bits ahorramos usando nanoprogramación en lugar de microprogramación?
a) 3744250
b) 259206
c) 287935
d) ninguno de los resultados anteriores es exacto

---

19) ¿Cómo actúa el indicador Z del registro de indicadores de estado?
a) Se pone a 1 cuando el resultado es negativo.
b) Se pone a 0 cuando el resultado es negativo.
c) Se pone a 1 cuando el resultado de una operación es 0.
d) Se pone a 1 cuando el resultado es positivo.

---

20) Supongamos dos CPU con idéntica anchura tanto en el bus de direcciones como en el de datos. Si una de ellas emplea E/S independiente y la otra mapeada en memoria, ¿cuál podrá acceder en general a una mayor cantidad de memoria?
a) La CPU con E/S independiente.
b) La CPU con E/S mapeada en memoria.
c) Ambas podrán acceder a la misma cantidad de memoria.
d) Depende de la técnica de E/S utilizada.

21) Una posible codificación en microinstrucciones de la instrucción CALL X es:

a) SP=SP-1 ; m[SP]=PC ; PC=PC+1
b) SP=PC-1 ; m[SP]=PC ; PC=X
c) PC=X ; SP=SP-1 ; m[SP]=PC
d) SP=SP-1 ; m[SP]=PC ; PC=X

22) Respecto al sistema de Entrada / Salida, ¿cuál de las siguientes afirmaciones es errónea?

a) Un módulo de E/S se encarga de la comunicación con el procesador.
b) Un protocolo sirve para "ponerse de acuerdo" en cosas como velocidad, paridad, nº de bits, etc.
c) La mayoría de los periféricos trabajan a velocidad muy superior al procesador; por eso es necesario sincronizar.
d) El procesador se comunica con el periférico por medio del controlador y de software de E/S.

23) ¿A qué tipo de localidad de memoria hace referencia la siguiente afirmación: "si se referencia un elemento, los elementos cercanos a él serán referenciados pronto"?

a) Localidad espacial
b) Localidad secuencial
c) Localidad temporal
d) Ninguna de las respuestas anteriores es correcta

24) ¿A qué tipo de memoria caché corresponde la siguiente afirmación: "permite que cualquier dirección se pueda almacenar en cualquier marco de bloque de caché"?

a) Con correspondencia directa
b) Totalmente asociativa
c) Asociativa por conjuntos
d) Ninguna de las anteriores

25) ¿Cuál de las siguientes afirmaciones acerca de las memorias RAM dinámicas es cierta?

a) Los datos permanecen en cada celda indefinidamente
b) Las celdas de almacenamiento son complejas
c) Las operaciones de lectura no son destructivas
d) Las operaciones de escritura sirven como operaciones de refresco

26) Cada celda de un chip de memoria DRAM de 1M x 1, organizada en una matriz de 512 filas x 2048 columnas, necesita ser refrescada cada 16 ms. ¿Cada cuánto tiempo ha de realizarse una operación de refresco en el chip?

a) 31,25 microsegundos
b) 61 nanosegundos
c) 8192 milisegundos
d) 7,8125 microsegundos

27) Un sistema no segmentado tarda 20 ns en procesar una tarea. La misma tarea puede ser procesada en un cauce (pipeline) de 4 segmentos con un ciclo de reloj de 5 ns. Cuando se procesan muchas tareas, la ganancia máxima de velocidad que se obtiene se aproxima a:

a) 5
b) 4
c) 0,25
d) 20

28) El primer nivel de una jerarquía de memoria tiene una tasa de aciertos del 75% y las peticiones de memoria tardan 12 ns en completarse si dicha posición se encuentra en ese nivel y 100 ns si no es así. ¿Cuál es el tiempo medio de acceso de la jerarquía?

a) 25 ns
b) 34 ns
c) 88 ns
d) 112 ns

29) En una jerarquía de memoria, a medida que nos alejamos de la CPU:

a) el tamaño de la unidad de transferencia entre dos niveles aumenta
b) el tiempo de transferencia disminuye
c) el tamaño de la memoria disminuye
d) el coste por byte aumenta

30) Si se necesitan 60 ns para escribir una palabra de datos de caché en memoria principal y cada bloque de caché tiene 8 palabras, ¿cuántas veces seguidas se tiene que escribir en un mismo bloque para que una caché de postescritura sea más eficiente que una de escritura inmediata?

a) Más de 8 veces.
b) La caché de postescritura no puede ser más eficiente que la de escritura inmediata.
c) La caché de postescritura siempre será más eficiente que la de escritura inmediata.
d) Depende de la tasa de aciertos.

2º Grado Informática
Estructura de Computadores
19 Septiembre 2012

Nombre:	
DNI:	**Grupo:**

Examen de Problemas (3.0p)

1. **Acceso a arrays** (0.5 puntos). Considerar el código C mostrado abajo, donde `H` y `J` son constantes declaradas mediante `#define`.

```
int array1[H][J];
int array2[J][H];

void copy_array(int x, int y) {
       array2[y][x] = array1[x][y];
}
```

Suponer que ese código C genera el siguiente código ensamblador x86-64:

```
# A la entrada:
#   %edi = x
#   %esi = y
#
copy_array:
       movslq  %esi,%rsi
       movslq  %edi,%rdi
       movq    %rsi, %rdx
       salq    $3, %rdx
       subq    %rsi, %rdx
       addq    %rdi, %rdx
       leaq   (%rdi,%rdi,4), %rax
       addq    %rsi, %rax
       movl    array1(,%rax,4), %eax
       movl    %eax, array2(,%rdx,4)
       ret
```

¿Cuáles son los valores de `H` y `J`?

`H = 7`

`J = 5`

2. **Representación y acceso a estructuras** (0.5 puntos). En el siguiente código C, las declaraciones de los tipos `type1_t` y `type2_t` se han hecho mediante `typedef`'s, y la constante `CNT` se ha declarado mediante `#define`.

```
typedef struct {
       type1_t y[CNT];
       type2_t x;
} a_struct;

void p1(int i, a_struct *ap) {
       ap->y[i] = ap->x;
}
```

La compilación del código para IA32 produce el siguiente código ensamblador:

```
# i en 8(%ebp), ap en 12(%ebp)
# movsbw = move signed byte to word (movw = move word)
p1:
        pushl   %ebp
        movl    %esp, %ebp
        movl    12(%ebp), %eax
        movsbw  28(%eax),%cx
        movl    8(%ebp), %edx
        movw    %cx, (%eax,%edx,2)
        popl    %ebp
        ret
```

Indicar una combinación de valores para los dos tipos y `CNT` que pueda producir el código ensamblador mostrado arriba:

```
type1_t:
type2_t:
CNT:
```

3. Memoria cache (0.5 puntos). Los parámetros que definen la memoria de un computador son los siguientes:

- Tamaño de la memoria principal: 32 K palabras.
- Tamaño de la memoria cache: 4 K palabras.
- Tamaño de bloque: 64 palabras.

Dibujar el formato de una dirección de memoria desde el punto de vista del sistema cache, determinando el tamaño de cada campo, y anotar los cálculos realizados para obtener dichos tamaños, para las siguientes políticas de colocación:

A. Totalmente asociativa.

B. Por correspondencia directa.

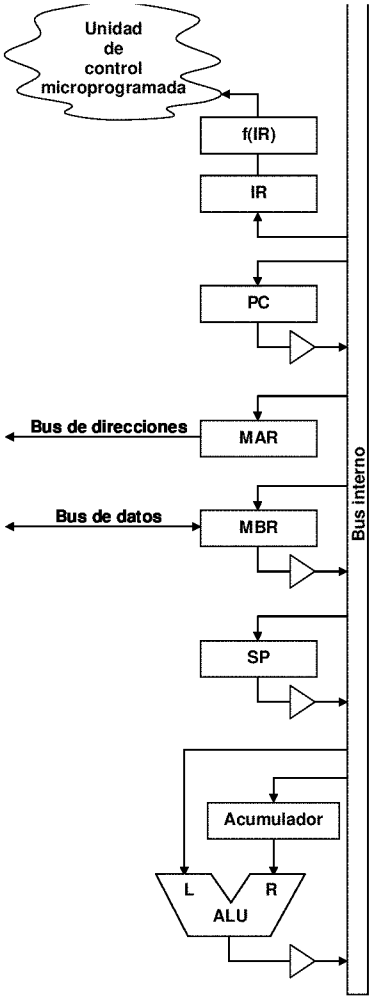
C. Asociativa por conjuntos con 16 bloques por conjunto.

4. **Diseño del sistema de memoria** (0.5 puntos). Disponemos de una CPU con buses de datos y direcciones de 16bit. Diseñar un sistema de memoria para la misma a partir de módulos SRAM de 16Kx8 y ROM de 8Kx4. La memoria ROM debe ocupar las direcciones 0x0000 a 0x3FFF y la SRAM 0x4000 a 0xFFFF. Se valorará la simplicidad del diseño.

5. **Entrada/Salida** (0.5 puntos). Dibuje un módulo sencillo de E/S capaz de realizar la interfaz entre los buses de un procesador y una impresora usando direccionamiento en memoria (no independiente). El módulo contará con un decodificador que activará una señal SEL cuando la dirección de memoria sea 10111100 o 10111101. La primera de estas dos direcciones corresponderá a dos registros, uno de estado y otro de control (A0 = 0), y la segunda al registro de datos (A0 = 1). Los registros de estado y control pueden tener una única dirección asociada porque el primero sólo se lee (RD) y el segundo sólo se escribe (WR) desde el bus.

6. Unidad de control microprogramada (0.5 puntos). Sea la ruta de datos de la figura, donde la ALU puede realizar las siguientes operaciones: R, L, R+1, R-1, L+1, L-1, R+L y R-L. Todos los registros son de 8 bits.

- Indique las señales de control necesarias para esta arquitectura.
- Diseñe un formato de microinstrucción
- Indique la secuencia de señales de control necesaria para leer de memoria una instrucción máquina y saltar a su fase de ejecución.



2º Grado Informática
Estructura de Computadores
19 Septiembre 2012

Nombre:	
DNI:	**Grupo:**

Examen de Prácticas (4.0p)

1. **Funcionamiento de la Pila.** (1 punto). Responder a las siguientes preguntas sobre la convención de pila usada en Linux x86 (con gcc en modo 32-bit):

A. ¿Cómo modifica la pila la instrucción CALL?
B. ¿Cómo modifica la pila la instrucción LEAVE?
C. ¿Cómo modifica la pila la instrucción RET?
D. ¿Cómo se pasan los argumentos a una función?
E. ¿Cómo se devuelven los valores de retorno a la función invocante?
F. Dibujar una region de pila detallando cómo llamaría un programa a la función

```
printf("%s %d elementos.\n", "Se insertaron", 4);
```

El primer argumento de `printf` (el *string* de formato) está almacenado en `0xbeefbabe` y el segundo *string* en `0xbabecafe`. Dibujar la zona de pila modificada/creada durante esta llamada a función, al menos desde el área de preparación de argumentos hasta el tope de pila alcanzado justo después de ejecutar la instrucción CALL. Como `printf` es una función `libc`, también se puede indicar el siguiente valor que se insertará en la pila.

2. Funciones aritméticas sencillas. (1 punto). A continuación se muestran **tres** pequeñas funciones aritméticas en C con el código ensamblador correspondiente. Cada tramo ensamblador contiene como máximo un error. Si hay error, marcarlo con un círculo y dar una **breve** explicación de por qué está mal en el espacio bajo el código. Si no hay error, decir sencillamente que no lo hay. Notar que el error (si existe) está en la traducción C→ensamblador, no en la lógica o funcionamiento del código C.

```
int twice(int x) {
  return (x + x);
}
08048334 <twice>:
 8048334:       55                      push   %ebp
 8048335:       89 e5                   mov    %esp,%ebp
 8048337:       8b 45 08                mov    0x8(%ebp),%eax
 804833a:       01 c0                   add    %eax,%eax
 804833c:       c3                      ret
 804833d:       c9                      leave
```

```
int second(char *str) {
  return str[1];
}
0804833e <second>:
 804833e:       55                      push   %ebp
 804833f:       89 e5                   mov    %esp,%ebp
 8048341:       8b 45 08                mov    0x8(%ebp),%eax
 8048344:       0f be 40 01             movsbl 0x1(%eax),%eax
 8048348:       c9                      leave
 8048349:       c3                      ret
```

```
int constant() {
  return 2;
}
0804834a <constant>:
 804834a:       55                      push   %ebp
 804834b:       89 e5                   mov    %esp,%ebp
 804834d:       b8 02 00 00 00          mov    0x2,%eax
 8048352:       5d                      pop    %ebp
 8048353:       c3                      ret
```

---

3. Programación mixta C-asm. (1 punto). A continuación se muestra un listado C con compilación condicional, pudiéndose recompilar un bucle *while* o una sentencia *asm inline* según el valor del símbolo ASM. Recordar que el mnemotécnico de Intel `cdq` (*convert double to quad*) equivale a AT&T `cltd`, y que en lenguaje C una expresión de asignación vale el valor asignado y `%` es la operación módulo.

Se ha ocultado parte del código C del bucle *while*. Sabiendo que la sentencia *asm* realiza el mismo cálculo que el bucle *while*:

A. Completar el código del bucle *while* sobre el enunciado

B. Indicar el resultado que se imprimiría en pantalla, debajo del listado

C. Describir brevemente en lenguaje natural (español) qué calcula el programa. Si se conoce el nombre del algoritmo, indicarlo también

```
#include <stdio.h>
#define ASM 1

int main(void){
    int A=21, B=9, r;

#if ! ASM
    while ( _ = _ % _){             // Completar!
       _ = _;
       _ = _;
    }
#else

    asm( "movl   %[B],%%ecx  \n"
         "movl   %[A],%%eax  \n"
".bucle:  cdq                \n"    // AT&T cltd
         "idivl       %%ecx  \n"
         "testl %%edx,%%edx  \n"    // while (_=_%_)
         "jz     .fin        \n"
         "movl  %%ecx,%%eax  \n"    // _=_;
         "movl  %%edx,%%ecx  \n"    // _=_;
         "jmp    .bucle      \n"
".fin:                       \n"
         "movl %%eax, %[A]   \n"
         "movl %%ecx, %[B]   \n"
         "movl %%edx, %[r]   \n"

         :[A] "+m" (A),
          [B] "+m" (B),
          [r] "+m" (r)
         :
         :"%eax", "%ecx", "%edx"
    );
#endif

    printf("resultado: %d\n", B);
}
```

```
resultado:
```

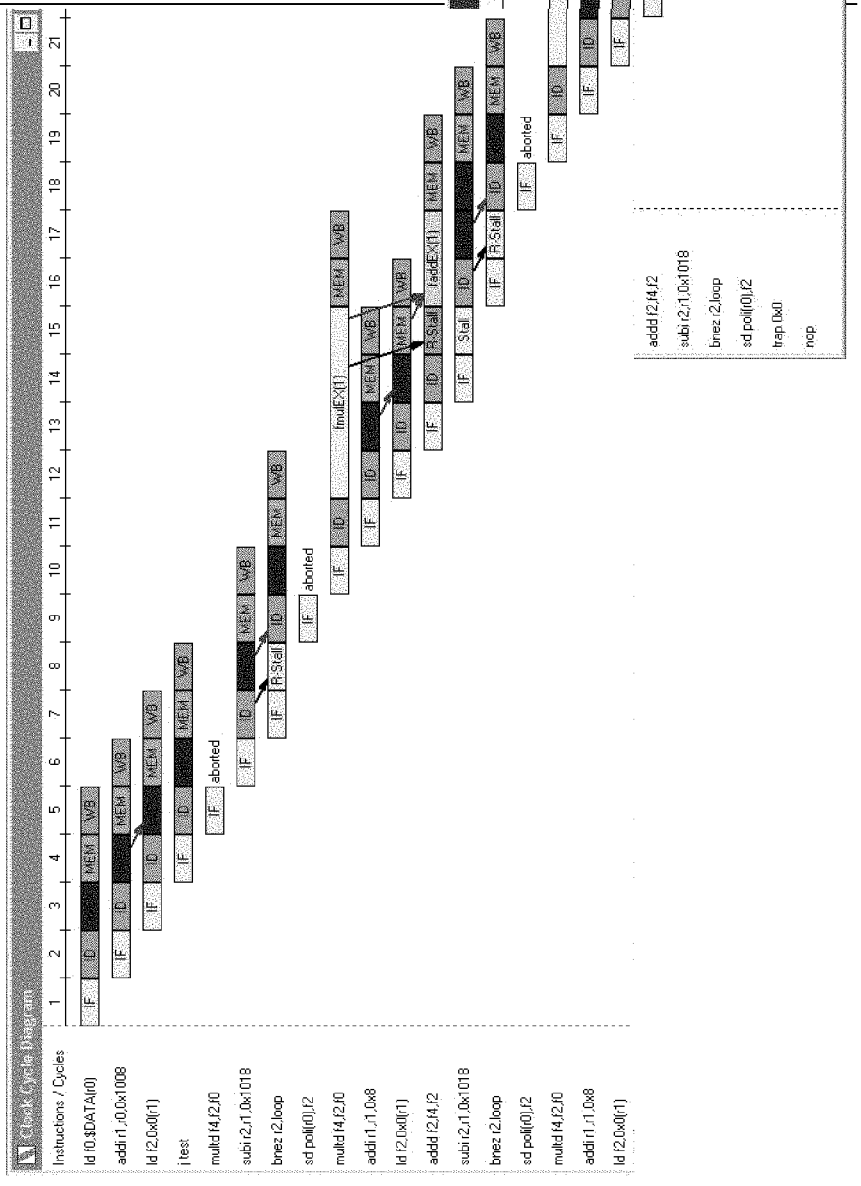
Descripción:

4 . Segmentación de cauce. (1 punto). A continuación se muestra un listado WinDLX de un programa que tarda en ejecutarse 32 ciclos con una unidad de multiplicación de 4 ciclos, como se muestra en la figura más abajo. Por motivos de legibilidad (y de espacio) ha sido necesario poner la figura de forma apaisada.

```
            .data
x:          .double 2
coefs:      .double 3,4,5
poli:       .double 0

            .text
start:
            ld      f0,x
            add     r1,r0,coefs
            ld      f2,(r1)
            j       test
loop:
            multd   f4,f2,f0
            add     r1,r1,8
            ld      f2,(r1)
            addd    f2,f4,f2
test:
            sub     r2,r1,poli-8
            bnez    r2,loop

            sd      poli,f2
            trap     0
;
; 32 ciclos con fMUL=4cyc
```



A. Indicar qué bloqueos presenta el programa: en qué número de ciclo, en qué instrucción, cuántos ciclos dura, por qué motivo, tipo de bloqueo. Si se repite varias veces, basta indicar los #ciclo adicionales en los que se repite

B. Indicar qué resultado calcula el programa: valor concreto y dónde se guarda

C. Describir brevemente en lenguaje natural (español) qué calcula el programa. Si se conoce el nombre del algoritmo, indicarlo también

D. Basta con mover de sitio una instrucción, y además repetirla en otro sitio, para que desaparezcan todas las dependencias. Reescribir el listado, mostrando cómo quedaría tras esa modificación (resaltar las partes modificadas)

2º Grado Informática
Estructura de Computadores
19 Septiembre 2012

Examen Test (3.0p)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
a	d	b	c	a	b	b	b	d	a	c	c	c	a	b	b	b	a	c	a	d	c	a	b	d	a	b	b	a	d

Examen de Prácticas (4.0p)

1. Funcionamiento de la Pila. (1 punto). (en gris respuestas adicionales opcionales)

A. CALL guarda en pila (push) la dirección de retorno (%eip) (y salta %eip <- función invocada)

B. LEAVE (deshace el marco de pila (mov %ebp, %esp) y…) recupera de pila el marco anterior (pop %ebp)

C. RET recupera de pila (pop) la dirección de retorno (%eip anteriormente salvado)

D. Args se meten en pila en orden inverso (…, push arg3, push arg2, push arg1)

E. Valor retorno se devuelve en el registro %eax (y %edx, si 64-bit)

F.

marco pila invocante
3er arg 0x4
2º arg 0xbabecafe
1er arg 0xbeefbabe
dirección retorno
antiguo EBP
marco pila de printf

Pila crece hacia abajo ↓ (posiciones inferiores)

2. Funciones aritméticas sencillas. (1 punto).

A. **twice** –ret está mal, antes de leave (se retorna a dirección basura %ebp antiguo, leave nunca llega a ejecutarse).

B. **second** – no hay error.

C. **constant** – 0x2 debería ser $0x2 (inmediato). 0x2 (direccionamiento directo) intenta accede a la posición de memoria 0x2 (que seguramente causará segmentation fault).

3. Programación mixta C-asm. (1 punto).

A. while (r = A % B) { A = B; B = r; }

B. resultado= **3**

C. Descripción: calcula el máximo común divisor (mcd) de A y B (algoritmo de Euclides).
- En cada iteración se calcula el resto de la división A/B en r (r=A%B). En la siguiente iteración se repite el cálculo, usando B y r como nuevos valores de A y B (porque mcd(A,B)=mcd(B,A%B)). Eventualmente r=0 porque A será múltiplo de B, y ese B es el resultado (mcd).

4. Segmentación de cauce. (1 punto).

A. Bloqueos:

#ciclo	instrucción	retardo	tipo de bloqueo-motivo
8,17,26	bnez r2,loop	1	RAW de R2 con instrucción anterior: sub r2,r1,poli-8
15,24	addd f2,f4,f2	1	RAW de D4 con instrucción -3 antes: multd f4,f2,f0 hay otra dependencia con instr. anterior: ld f2,(r1), pero no causa bloqueo
(otros bloqueos no son relevantes para ahorrar ciclos con la reordenación que se pide)			

B. Resultado: Se guarda en la variable poli.
Poli= 25(3x4 + 4x2 + 5)

C. Calcula coefs[0] x^2 + coefs[1] x + coefs[2]. Si coefs contiene los coeficientes de un polinomio desde Cn-1 a C0, este programa evalúa el polinomio en x. Lo hace sacando factores comunes x todo lo posible: ax^2+bx+c = ((ax)+b)x+c.

D. Como los bloqueos están entre sub-bnez y entre multd-addd (también hay dependencia ld-addd, ver 2 flechas verdes), se puede adelantar sub entre ld y addd (así se rellena el ciclo del bloqueo), pero entonces hay que repetirla antes de "j test" (o antes de ld f2, pero siempre después de add r1, que es donde se fija el valor de r1) para que se siga calculando bien la condición de salto en la primera iteración

Original

```
start:
        ld      f0,x
        add     r1,r0,coefs
        ld      f2,(r1)

        j       test
loop:
        multd   f4,f2,f0
        add     r1,r1,8
        ld      f2,(r1)

        addd    f2,f4,f2
test:
        sub     r2,r1,poli-8
        bnez    r2,loop

        sd      poli,f2
        trap    0
```

Modificado

```
start:
        ld      f0,x
        add     r1,r0,coefs
        ld      f2,(r1)
        sub     r2,r1,poli-8
        j       test
loop:
        multd   f4,f2,f0
        add     r1,r1,8
        ld      f2,(r1)
        sub     r2,r1,poli-8
        addd    f2,f4,f2
test:

        bnez    r2,loop

        sd      poli,f2
        trap    0
```

Examen de Problemas (3.0p)

1. Acceso a arrays (0.5 puntos).

```
H = 7
J = 5
Posibles comentarios que podrían añadirse al código para demostrar su comprensión
copy_array:
      movslq  %esi,%rsi             # índice y
      movslq  %edi,%rdi             # índice x
      movq    %rsi, %rdx            # rdx = y
      salq    $3, %rdx              #       8y
      subq    %rsi, %rdx            #             - y
      addq    %rdi, %rdx            #             + x -> rdx = 7y+x
      leaq   (%rdi,%rdi,4), %rax    # rax = 5x
      addq    %rsi, %rax            #             + y -> rax = 5x+y
      movl    array1(,%rax,4), %eax # &array1[x][y] = array1+ 5x+y -> J = 5
      movl    %eax, array2(,%rdx,4) # &array2[y][x] = array2+ 7y+x -> H = 7
      ret
```

2. Representación y acceso a estructuras (0.5 puntos).

```
type1_t:  [unsigned] short (tamaño 2B)
type2_t:  char
CNT:      14
Posibles comentarios que podrían añadirse al código para demostrar su comprensión
p1:
      pushl   %ebp
      movl    %esp, %ebp
```

```
movl    12(%ebp), %eax          # eax = ap
movsbw  28(%eax),%cx            # cx = ap->x, pasar de 1B con signo a 2B (para y[i]=x)
                                #      => type2_t = 1B con signo = char
                                #      => y[CNT] ocupa 28B, saltarlo para llegar a x
movl    8(%ebp), %edx           # edx = i, notar cómo siguiente instr. indexa *2
movw    %cx, (%eax,%edx,2)      # ap->y[i]=x => elementos y[] son de tamaño 2B
popl    %ebp                    #      => type1_t = cualquiera de 2B
ret                             #      => 28/2 = 14 = CNT
```

3. **Memoria cache** (0.5 puntos).
Bloques de 64 palabras = 2^6 pal/blq
-> 6 bits para direccionar la palabra dentro de cada bloque

MP 32K = 2^5 2^10 = 2^15 palabras
-> 15 bits para direccionar en memoria
-> 2^15 pal / 2^6 pal/blq = 2^9 bloques en MP (512 bloques)

Cache 4K = 2^2 2^10 = 2^12 palabras de cache
-> 2^12 pal / 2^6 pal/blq = 2^6 marcos en cache (64 marcos)

Totalmente Asociativa:
Cualquier bloque puede ir a cualquier marco, se identifica bloque por etiqueta

15 bits	
15-6 = 9 bits	6 bits
etiqueta	palabra

Correspondencia directa:
Bloques sucesivos van a marcos sucesivos. Al acabarse la cache, vuelta a empezar
-> 6 bits inferiores palabra, 6 bits siguientes marco, resto etiqueta

15 bits		
15-6-6=3	6 bits	6 bits
etiqueta	marco (bloque)	palabra

Asociativa 16 vías:
Bloques sucesivos van a conjuntos sucesivos, pero cada conj.tiene varios marcos (vías)
16 = 2^4 blq/conj, cache de 2^6 marcos
-> 2^6 marcos / 2^4 blq/conj = 2^2 conjuntos -> 2 bits para direccionar conjunto
-> 6 bits inferiores palabra, 2 bits siguientes conjunto, resto etiqueta

15 bits		
15-2-6= 7 bits	2bits	6 bits
etiqueta	conj.	palabra

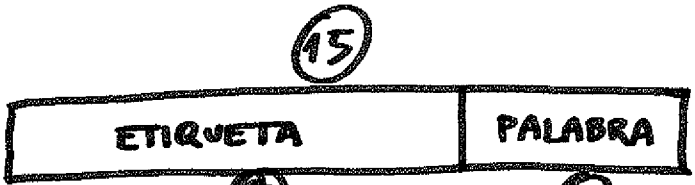
4. **Diseño del sistema de memoria** (0.5 puntos).
Ver imágenes adjuntas

5. **Entrada/Salida** (0.5 puntos).
Ver imágenes adjuntas

6. **Unidad de control microprogramada** (0.5 puntos).
Ver imágenes adjuntas

3. Memoria caché (0,5 puntos)

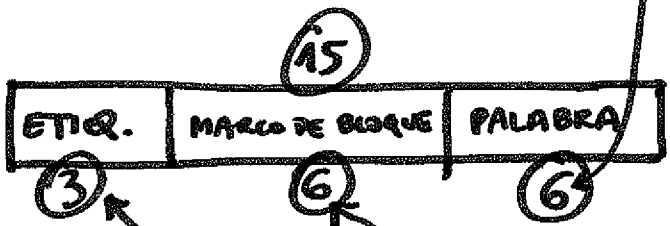
A. Totalmente asociativa



M.P.: $\frac{32K = 2^{15} \text{ palabras}}{2^6 \text{ palabras/bloque}} = 2^9 \text{ bloques}$

$64 = 2^6$ palabras/bloque

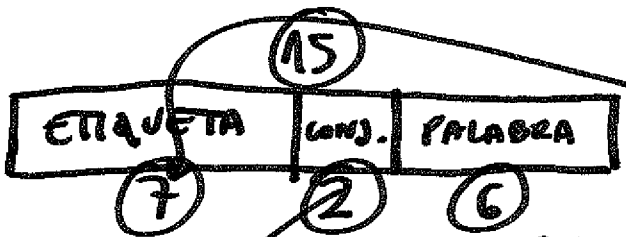
B. Por correspondencia directa



Caché: $\frac{4K = 2^{12} \text{ palabras}}{2^6 \text{ palabras/bloque}} = 2^6$ bloques

$\frac{2^9 \text{ bloques}}{2^6 \text{ bloques}} = 2^3$ bloques de mem. principal por cada marco de bloque de caché

C. Asociativa por conjuntos con 16 bloques por conjunto



$\frac{2^6 \text{ bloques}}{2^4 \frac{\text{bloques}}{\text{conj}}} = 2^2$ conjuntos

$\frac{2^9 \text{ bloques}}{2^2 \text{ conj.}} = 2^7$ bloques de mem. principal por cada conjunto

4. Diseño del sistema de memoria (0,5 puntos)