

Nombre:

DNI:

Grupo:

Examen Test (3.0p)

Tipo B

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

- ¿Qué parámetro es más importante para comparar la velocidad de dos ordenadores diferentes?
 - La frecuencia de reloj del procesador.
 - El número medio de ciclos de reloj por instrucción.
 - La arquitectura del procesador.
 - El resultado de la ejecución de un conjunto de programas de prueba.
- ¿Cuál es la característica tecnológica principal de la segunda generación de computadores?
 - Los circuitos integrados
 - Los transistores
 - La gran integración de los circuitos (VLSI)
 - Las válvulas
- ¿Cuál de las siguientes no es una característica de los computadores RISC?
 - La decodificación de las instrucciones debe ser simple: un computador RISC debería emplear un único formato de instrucción
 - Para acelerar el computador RISC se emplean técnicas de pipelining.
 - Las funciones que realizan los computadores RISC deben ser lo más complejas y potentes que sea posible.
 - Un computador RISC no debe emplear microprogramación.
- ¿Cuál de las siguientes instrucciones no modifica necesariamente la secuencia de ejecución del programa?
 - JMP dir
 - JNE dir
 - CALL dir
 - RET
- ¿Cuál de las siguientes parejas de nemotécnicos de ensamblador corresponden a la misma instrucción máquina?
 - CMP (comparar), SUB (restar).
 - JC (saltar si acarreo), JL (saltar si menor, para números con signo).
 - JZ (saltar si cero), JE (saltar si igual).
 - SAR (desplazamiento aritmético a la derecha) / SHR (desplazamiento lógico a la derecha).
- ¿Cuál de los siguientes grupos de instrucciones sólo modifican los indicadores de estado sin almacenar el resultado de la operación?
 - AND, OR, XOR
 - ADC, SBB
 - CMP, TEST
 - IMUL, IDIV
- ¿Cuál de las siguientes características es típica de la microprogramación horizontal?
 - Muchos campos solapados.
 - Ninguna o escasa codificación.
 - Microinstrucciones cortas.
 - Escasa capacidad para expresar paralelismo entre microoperaciones.
- ¿En qué pareja de registros están el dato/instrucción que se leerá o escribirá en memoria, y la dirección de memoria?
 - MAR y ACUMULADOR
 - IR y ACUMULADOR
 - MBR y MAR
 - MBR y PC
- ¿Qué circuito suele utilizarse para traducir el código de operación de una instrucción máquina a dirección de comienzo en la memoria de control del microprograma correspondiente?
 - Una memoria.
 - Un contador.
 - Un multiplexor.
 - Un registro.
- ¿Con cuál de los siguientes dispositivos tendría sentido utilizar E/S programada sin consulta de estado?
 - Salida a un display de 7 segmentos
 - Entrada desde un disco duro
 - Salida a una impresora
 - Con ningún dispositivo tiene sentido

11. ¿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

12. ¿Cuál de los siguientes grupos de instrucciones podrá pertenecer a un procesador con E/S mapeada en memoria?

- a. IN, LOAD, OUT
- b. IN, LOAD, MOV
- c. LOAD, MOV, STORE
- d. Ninguno de los anteriores

13. ¿Cuántas señales de control se necesitan como mínimo para implementar un sistema de gestión de interrupciones?

- a. 2
- b. 3
- c. 4
- d. 5

14. ¿Cuántos bits hacen falta como mínimo para implementar tres niveles de inhibición de interrupciones (general, nivel y máscara) en un sistema con cuatro niveles de interrupción?

- a. 4
- b. 5
- c. 6
- d. 7

15. ¿En qué tipo de transferencias es necesario establecer un período de tiempo máximo después del cual se considera que ha fallado?

- a. En las transferencias sincronicas
- b. En las transferencias asincronicas
- c. En ambas
- d. En ninguna

16. ¿Qué técnica de E/S requiere menos atención por parte del procesador?

- a. E/S programada
- b. E/S mediante acceso directo a memoria
- c. E/S mediante interrupciones
- d. Todas requieren la misma atención

17. ¿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

18. ¿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

19. ¿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

20. 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

21. ¿Cuál de las siguientes direcciones está alineada a double (8-byte)?

- a. 1110110101110111)₂
- b. 1110110101110100)₂
- c. 1110110101110000)₂
- d. Ninguna de ellas

22. ¿Cómo se devuelve en ensamblador x86 Linux gcc el valor de retorno de una función 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 del (%ebp) del invocado
- d. Se almacena en pila justo encima de los argumentos de la función

23. ¿Cuál afirmación es FALSA en arquitecturas x86-64?

- a. El tamaño de un double es 64 bits
- b. El tamaño de los registros es 64 bits
- c. El tamaño de un puntero es 64 bits
- d. El tamaño de las posiciones de memoria es 64 bits

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

```
struct RECORD {          struct NODE {
    long value2;           double value;
    int ref_count;         struct RECORD record;
    char tag[4];           char string[8];
};                          };
```

También se declara una variable global my_node como sigue:

```
struct NODE my_node;
```

Si la dirección de my_node es 0x601040, ¿cuál es el valor de smy_node.record.tag[1] ?

- a. 0x601050
- b. 0x601054
- c. 0x601055
- d. Ninguna de las anteriores

25. En la pregunta anterior, ¿cuál es el tamaño de my_node en bytes?

- a. 32

- b. 40
 - c. 28
 - d. Ninguno de los anteriores
-

26. ¿Cuál de las siguientes instrucciones x86 se puede usar para sumar dos registros y guardar el resultado sin sobrescribir ninguno de los registros originales?

- a. mov
 - b. lea
 - c. add
 - d. Ninguna de ellas
-

27. Respecto a los registros enteros en arquitectura IA-32

- a. Hay 8, y en cada uno puede accederse a todos los 32 bits (p.ej. EAX), a los 16 bits menos significativos (p.ej. AX) ó a los 8 LSBs (p.ej. AL)
 - b. Hay 8 de cada tamaño (32, 16, 8 bits), aunque no todos los registros tienen versión en 8 y 16 bits
 - c. No hay distintos tamaños, son sólo registros de 32 bits, como corresponde a dicha arquitectura
 - d. Son de 32bits en las aplicaciones de 32bit, y de 64bits en las aplicaciones de 64bit
-

28. 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. Los registros base e índice (Rb y Ri) pueden ser cualesquiera de los 8 registros enteros (EAX...ESP)
 - c. El factor de escala S puede ser 1, 2, 4, 8
 - d. El desplazamiento D también puede ser el nombre de una variable (que se traduce por su dirección, de 4bytes)
-

29. 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 `unsigned` es 4 bytes
 - c. El tamaño de un `long` es 4 bytes
 - d. El tamaño de un `short` es 2 bytes
-

30. Estudiando el listado de una función C presuntamente compilada con gcc en modo 64bit (x86-64), nos dicen que la primera instrucción, `movl %eax, (%rdi)`, carga en 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
-

Nombre:	
DNI:	Grupo:

Examen de Problemas (3.0p)

Tipo A

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
    leaq      (%rsi,%rsi,8), %rdx
    addq      %rdi, %rdx
    movq      %rdi, %rax
    salq      $4, %rax
    subq      %rdi, %rax
    addq      %rsi, %rax
    movl      array1(,%rax,4), %eax
    movl      %eax, array2(,%rdx,4)
    ret
```

¿Cuáles son los valores de H y J?

H =

J =

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)
# movsbl = move signed byte to long
p1:
    pushl   %ebp
    movl    %esp, %ebp
    movl    12(%ebp), %eax
    movsbl  28(%eax), %ecx
    movl    8(%ebp), %edx
    movl    %ecx, (%eax,%edx,4)
    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.

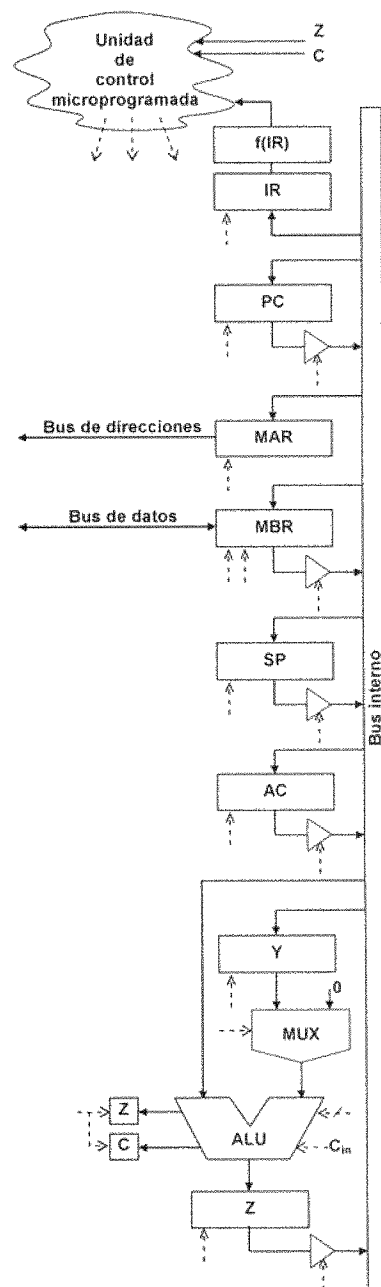
Dibujar el formato de una dirección de memoria desde el punto de vista del sistema cache, determinando el tamaño de cada campo, y anotar los cálculos realizados para obtener dichos tamaños, para las siguientes políticas de colocación:

- 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). Disponemos de un microprocesador de 8 bits (bus de datos de 8 bits y bus de direcciones de 16 bits) con E/S independiente. Diseñar un sistema de E/S que permita acceder a los siguientes puertos: puerto 0x0240 de entrada y 0x0241 de salida. Utilizar lógica de decodificación distribuida. No emplear decodificadores.

6. **Unidad de control microprogramada** (0.5 puntos). La figura de la derecha muestra el camino de datos de un procesador. Todas las instrucciones del procesador ocupan una palabra. Escriba en pseudocódigo la parte del microprograma correspondiente a la captación y decodificación de instrucción.



Nombre:	
DNI:	Grupo:

Examen de Prácticas (4.0p)**Tipo B**

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("%d %s fueron insertados.\n", 5, "elementos");
```

El primer argumento de `printf` (el *string* de formato) está almacenado en `0xbeefbabe` y el *string* "elementos" 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 squareNumber(int x) {
    return (x * x);
}
08048344 <squareNumber>:
8048344:    55                push    %ebp
8048345:    89 e5             mov     %esp,%ebp
8048347:    8b 45 04           mov     0x4(%ebp),%eax
804834a:    0f af c0           imul    %eax,%eax
804834d:    5d                pop     %ebp
804834e:    c3               ret
```

```
int fourth(char *str) {
    return str[3];
}
0804834f <fourth>:
804834f:    55                push    %ebp
8048350:    89 e5             mov     %esp,%ebp
8048352:    8b 45 08           mov     0x8(%ebp),%eax
8048355:    83 c0 03           add     0x3,%eax
8048358:    0f be 00           movsbl  (%eax),%eax
804835b:    5d                pop     %ebp
804835c:    c3               ret
```

```
int unrandomNumber() {
    return 4;
}
0804835e <unrandomNumber>:
804835e:    55                push    %ebp
804835f:    89 e5             mov     %esp,%ebp
8048361:    a1 04 00 00 00     mov     $0x4,%eax
8048366:    5d                pop     %ebp
8048367:    c9               leave
8048368:    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 *cldq*, 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*:

- Completar el código del bucle *while* sobre el enunciado
- Indicar el resultado que se imprimiría en pantalla, debajo del listado
- 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=20, B=12, r;

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

        asm( "movl    %[B],%%ecx \n"
            "movl    %[A],%%eax \n"
        ".bucle:  cdq      \n"           // AT&T cldq
            "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 5
coefs:  .double 4,3,2
poli:   .double 0

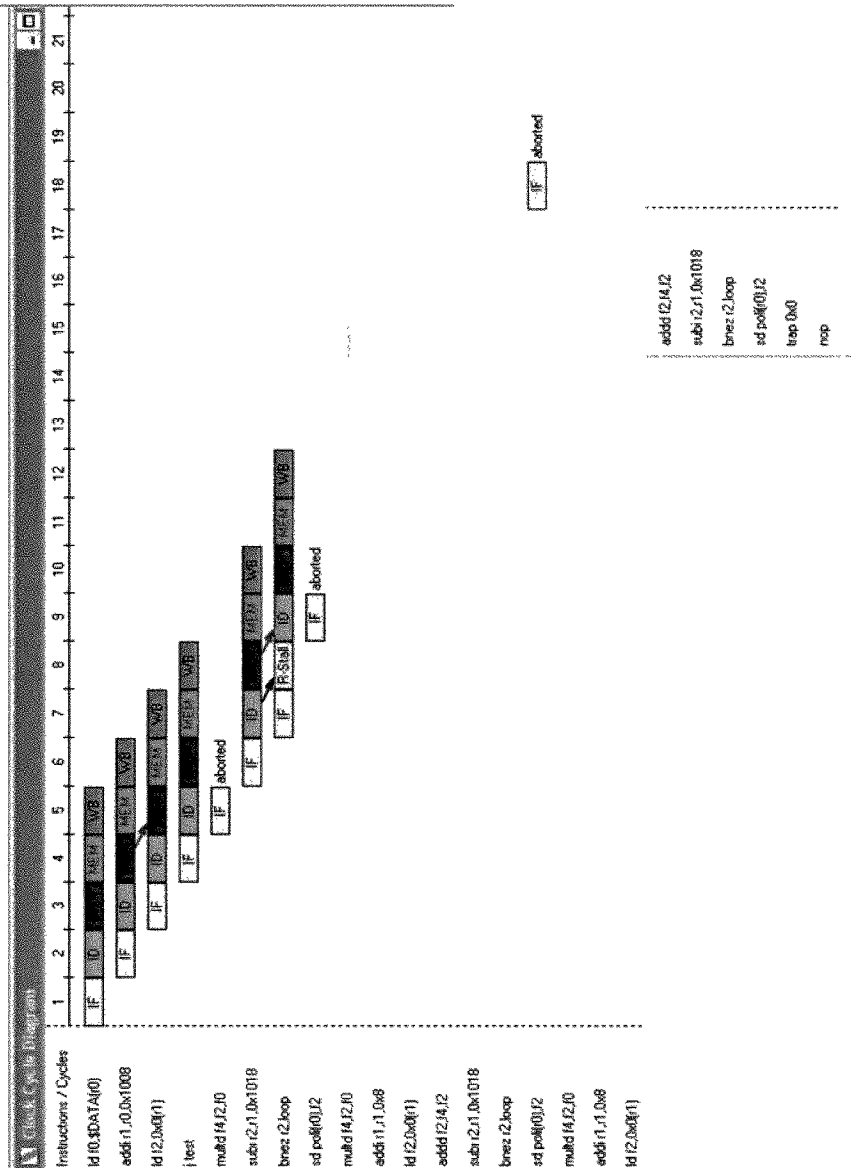
.text
start:
    ld      f0,x
    add     r1,r0,coefs
    ld      f2,(r1)
    j       test

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)