Nombre y apellido: _____ Padrón: _____

1. Responda las siguientes preguntas sobre arquitectura segmentada.

   a) Enumere las etapas de la implementación secuencial de Y86-64 y describa el propósito de cada una, incluyendo las acciones que se ejecutan durante las mismas.

   b) Elija la opción que corresponda: "Tras la etapa *decode*, es posible determinar el siguiente valor del *program counter*": [ ] A veces. [ ] Siempre. [ ] Nunca.

   Si la respuesta es "A veces", describa un ejemplo de un caso en que no se pueda. Si la respuesta es "Siempre" o "Nunca", justifique.

   c) Responda verdadero o falso conforme a la arquitectura Y86-64 (o, de manera equivalente, conforme a x86/64; pues para ambas preguntas el comportamiento es el mismo en las dos arquitecturas). **Justifique brevemente la respuesta.**

   - Si en el tope de la pila está el valor V, tras ejecutar la instrucción `popq %rsp` el registro `%rsp` tiene como valor V+8.
   - Tras ejecutar `pushq %rsp`, el valor en el tope de la pila es mayor que el valor en `%rsp`.

2. Un grupo de arquitectos diseñando una nueva implementación del *datapath* del procesador determinó los siguientes retardos en el hardware a utilizar:

instruction memory	140 ps
decode	70 ps
register fetch	130 ps
ALU	130 ps
data memory	190 ps
register write-back	110 ps

   Además de los 20 ps de cada *pipeline register*.

   a) ¿Cuál es el valor mínimo del ciclo de clock para una implementación secuencial? ¿Cuál es la frecuencia máxima del procesador?

   b) ¿Cuál es el valor del clock más rápido que se puede obtener con un *datapath* segmentado en 5 etapas? ¿Cuánto más rápido es que la implementación secuencial (como cociente de las frecuencias del clock)? *Ayuda: al combinar las etapas, los circuitos van en el orden listados. Por ejemplo, no se pueden unir en una etapa el* register write-back *y el* decode*, sin incluir en la misma etapa* register fetch*,* ALU *y* data memory*.*

   c) ¿Cuál es el valor del clock más rápido que se puede obtener con un *datapath* segmentado en 9 etapas? ¿Cuánto más rápido es que la implementación secuencial (como cociente de las frecuencias del clock)? *Ayuda: asuma que cualquiera de los circuitos dados se puede separar en k-partes iguales.* ***versión alternativa: en vez de k-partes, k sólo puede ser 2, 3 o 4.***

3. Clasifique los distintos tipos y subtipos de riesgos por dependencia en una arquitectura paralela, y explique qué estrategias pueden usarse (o no) para solventar cada uno de ellos.

4. Para la siguiente secuencia de instrucciones, ejecutada en un *datapath* de 5 etapas (el visto en clase):

```
1 popq    %rsi
2 subq    %rsi, %rax
3 irmovq  $0x16, %rdx
4 mrmovq  8(%rsp), %rbx
5 addq    %rbx, %rax
6 addq    %rdx, %rsi
7 xorq    %rax, %rdx
```

   a) Indentifique y marque todas las dependencias de datos del tipo *read-after-write (RAW)*: para ello, marque con un circulo los operandos dependientes y únalos con una flecha. Identifique asimismo las dependencias de datos de tipo *Load/Use*, y los riesgos de control.

b) Complete el cuadro siguiente con el estado que tendría el *pipeline* si el mismo cuenta con *data forwarding* para salvar las dependencias que marcó en el punto anterior, y registros de pipeline con capacidad de demorar la ejecución e insertar burbujas si fuese necesario.

Instrucción	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
`popq %rsi`	F	D															
		F															

Nombre y apellido: _____ Padrón: _____

1) En el siguiente ejercicio se pide hacer un seguimiento de pedidos de datos hechos por la CPU, utilizando memoria virtual. El sistema cuenta con una TLB, paginado de 1 nivel (1-level page table) y cache L1 (no dispone de cache L2 ni L3).

Las direcciones virtuales son de 16 bits y las direcciones físicas de 13 bits. La TLB y la cache se muestran **completas**, mientras únicamente se muestran las primeras 32 entradas de la tabla de paginación. El *page size* es de 512 bytes.

Index	Tag	PPN	V
0	09	4	1
	12	2	1
	10	0	1
	08	5	1
	05	7	1
	13	1	0
	10	3	0
	18	3	0
1	04	1	0
	0C	1	0
	12	0	1
	08	1	0
	06	7	0
	03	1	1
	07	5	0
	02	2	1

(a) TLB

VPN	PPN	V	VPN	PPN	V
00	6	1	10	0	1
01	5	0	11	5	0
02	3	1	12	2	1
03	4	1	13	4	0
04	2	0	14	6	0
05	7	1	15	2	0
06	1	0	16	4	0
07	3	0	17	6	0
08	5	1	18	1	1
09	4	0	19	2	0
0A	3	0	1A	5	0
0B	2	0	1B	2	0
0C	5	0	1C	6	0
0D	6	0	1D	2	0
0E	1	1	1E	3	0
0F	0	0	1F	1	0

(b) Page Table

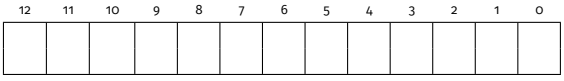
Index	Tag	V	Byte 0	Byte 1	Byte 2	Byte 3	Tag	V	Byte 0	Byte 1	Byte 2	Byte 3
0	19	1	99	11	23	11	00	0	99	11	23	11
1	15	0	4F	22	EC	11	2F	1	55	59	0B	41
2	1B	1	00	02	04	08	0B	1	01	03	05	07
3	06	0	84	06	B2	9C	12	0	84	06	B2	9C
4	07	0	43	6D	8F	09	05	0	43	6D	8F	09
5	0D	1	36	32	00	78	1E	1	A1	B2	C4	DE
6	11	0	A2	37	68	31	00	1	BB	77	33	00
7	16	1	11	C2	11	33	0E	1	00	C0	0F	00

(c) Cache L1

Cuadro 1: Tablas de memoria

a) En los esquemas de las direcciones físicas (2b) y virtuales (2a) indique claramente los bits utilizados para el *offset* de la página física, el número de página física, el número de página virtual, el *offset* de la página virtual, el *offset* de cache, el índice de cache, el *tag* de cache, el *tag* de TLB, y el índice de TLB.

(a)



(b)

b) Complete la siguiente tabla. La columna VA indica las direcciones virtuales que pide la CPU:

VA	VPN	VPO	TLBi	TLBt	PA	Co	Ci	Ct	¿TLB hit?	¿Cache hit?	¿Page fault?	Dato
1DDE												
37E4												
4B6A												

2) Se tiene una cache de 2048 bytes con mapeo directo y bloques de 32 bytes.
Dadas las siguientes definiciones:

```
struct color {
    int c;
    int m;
    int y;
    int k;
};

struct color cuadrado[16][16];
int i, j;
```

y asumiendo:

- El `sizeof` corresponde a una arquitectura x86 (32 bits).
- `cuadrado` comienza en la dirección 0.
- La memoria cache está inicialmente vacía.
- Los únicos accesos a memoria son al arreglo, las variables `i` y `j` se almacenan en registros.

a) ¿Qué porcentaje de escrituras en la cache va a fallar el siguiente código?

```
for (i=0; i < 16; i++) {
    for (j=0; j < 16; j++) {
        cuadrado[i][j].c = 0;
        cuadrado[i][j].m = 0;
        cuadrado[i][j].y = 1;
        cuadrado[i][j].k = 0;
    }
}
```

b) ¿Qué porcentaje de escrituras en la cache va a fallar el siguiente código?

```
for (i=0; i < 16; i++) {
    for (j=0; j < 16; j++) {
        cuadrado[j][i].c = 0;
        cuadrado[j][i].m = 0;
        cuadrado[j][i].y = 1;
        cuadrado[j][i].k = 0;
    }
}
```

c) ¿Qué porcentaje de escrituras en la cache va a fallar el siguiente código?

```
for (i=0; i < 16; i++) {
    for (j=0; j < 16; j++) {
        cuadrado[j][i].y = 1;
    }
}
for (i=0; i < 16; i++) {
    for (j=0; j < 16; j++) {
        cuadrado[j][i].c = 0;
        cuadrado[j][i].m = 0;
        cuadrado[j][i].k = 0;
    }
}
```

3) Responda las siguientes preguntas sobre arquitectura de hardware.
a) Enumere las etapas de la implementación secuencial de Y86-64 y describa el propósito de cada una, incluyendo las acciones que se ejecutan durante las mismas.
b) Elija la opción que corresponda: "Tras la etapa *decode*, es posible determinar el siguiente valor del *program counter*": [ ] A veces. [ ] Siempre. [ ]Nunca.

Si la respuesta es "A veces", describa un ejemplo de un caso en que no se pueda. Si la respuesta es "Siempre" o "Nunca", justifique.

---

c) Responda verdadero o falso conforme a la arquitectura Y86-64 (o, de manera equivalente, conforme a x86/64; pues para ambas preguntas el comportamiento es el mismo en las dos arquitecturas). **Justifique brevemente la respuesta.**

- Si en el tope de la pila está el valor V, tras ejecutar la instrucción `popq %rsp` el registro `%rsp` tiene como valor V+8.

- Tras ejecutar `pushq %rsp`, el valor en el tope de la pila es mayor que el valor en `%rsp`.

---

4) a) Dadas las siguientes definiciones

```
typedef struct {
    long l;
    char c;
    int i;
} a;
```

```
typedef struct {
    int i;
    long l;
    char c;
    int j;
} b;
```

```
typedef struct {
    int i;
    long l;
    char c;
} d;
```

dé el valor de las siguientes expresiones:

`sizeof(a):`		`sizeof(b):`		`sizeof(d):`	
`offsetof(a, l):`		`offsetof(b, i):`		`offsetof(d, i):`	
`offsetof(a, c):`		`offsetof(b, l):`		`offsetof(d, l):`	
`offsetof(a, i):`		`offsetof(b, c):`		`offsetof(d, c):`	
		`offsetof(b, j):`			

Nota: La macro `offsetof` está definida en `stddef.h` y simplemente devuelve el *offset* de un miembro respecto al comienzo del struct (esto es, la cantidad de bytes desde el comienzo del struct, hasta el comienzo del campo). **b)** Considere la siguiente representación en punto flotante de 10 bits, basada en el formato de la IEEE:

- El bit más significativo es un bit de signo (s),
- le siguen 3 bits de exponente (e), y
- los últimos 6 bits son la fracción (f).

Se cumplen las mismas reglas que en el estándar IEEE para definir números normalizados, denormalizados, infinitos, NaN y representación del 0. Complete la siguiente tabla, donde:

binario es la representación en 10 bits

M es el significando. Puede ser un número x o x/y, donde x e y son enteros. Ejemplo: 0, 3/256.

E es el exponente (¡distinto de e!).

Si tiene que redondear, hágalo según las reglas de redondeo al par más cercano (*Round-To-Even*).

Valor	binario	M	E
−15,375			
−4,8125			
9,78125			
10,8125			

c) A partir de las siguientes dos funciones en C,

```
short f(short a, short b) {
    short result = ________
    while (________) {
        result = ________;
        a      = ________;
    }
    return result;
}
```

```
long g(long a, long b) {
    long result = ________;  ;
    while (________) {
        result = ________;
        b      = ________;
    }
    return result;
}
```

se obtuvo el código assembly que se lista a continuación:

```
f:
    movl $0, %eax
    jmp .L2
.L3:
    leaq (,%rsi,%rdi), %rdx
    addq %rdx, %rax
    subq $1, %rdi
.L2:
    cmpq %rsi, %rdi
    jg .L3
    ret
```

```
g:
    testq %rsi, %rsi
    jle .L8
    movq %rsi, %rax
.L7:
    imulq %rdi, %rax
    subq %rdi, %rsi
    testq %rsi, %rsi
    jg .L7
    ret
.L8:
    movq %rsi, %rax
    ret
```

Complete las expresiones incompletas en el código C de forma tal que, si se compila del mismo modo, se obtiene un código assembly similar al mostrado.

Nombre y apellido: _____ Padrón: _____

Arquitectura

1) Dada la siguiente secuencia de instrucciones, y asumiendo que se ejecuta en un *datapath* segmentado en cinco etapas.

```
1 popq    %rsi
2 addq    %rsi, %rax
3 leaq    4(%rax), %rdx
4 movq    4(%rdx), %rdi
5 addq    %rdi, %rax
6 subq    %rdx, %rax
7 testq   %rax, %rax
```

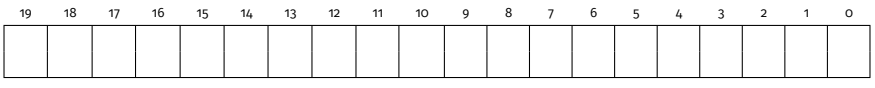
a) Asumiendo que no hay *forwarding* ni detección de riesgos, inserte NOPs para asegurar la correcta ejecución del código.
b) Reogranice el código original para minimizar la cantidad de NOPs necesarios, si es posible, para el procesador del inciso anterior.
c) Asumiendo que hay *forwarding* pero no detección de riesgos, inserte NOPs para asegurar la correcta ejecución del código.
d) Reogranice o reemplace el código original para minimizar la cantidad de NOPs necesarios, si es posible, para el procesador del inciso anterior.
e) Si el procesador dispone de *data-forwarding* pero no tiene detección de riesgos ¿qué hace el código provisto?

Memoria Virtual y Cache

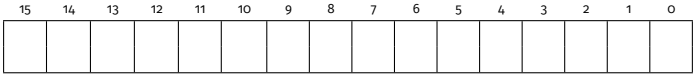
2) En el siguiente ejercicio se pide hacer un seguimiento de pedidos de datos hechos por la CPU, utilizando memoria virtual. El sistema cuenta con una TLB, paginado de 1 nivel (1-level page table) y cache L1 (no dispone de cache L2 ni L3).

Las direcciones virtuales son de 20 bits y las direcciones físicas de 16 bits. La TLB es asociativa de 8 vías con un total de 32 entradas (se muestran 16 en la tabla). La memoria cache se muestra completa. Se muestran las primeras 32 entradas de la tabla de paginación. El *page size* es de 4096 bytes. Los accesos a memoria se realizan de a bytes.

a) En los esquemas de las direcciones físicas (2b) y virtuales (2a) indique claramente los bits utilizados para el *offset* de la página física, el número de página física, el número de página virtual, el *offset* de la página virtual, el *offset* de cache, el índice de cache, el *tag* de cache, el *tag* de TLB, y el índice de TLB.



(a)



(b)

Cuadro 2: Esquemáticos de (a) direcciones virtuales, y (b) direcciones físicas

b) Complete la siguiente tabla. La columna VA indica las direcciones virtuales que pide la CPU:

Index	Tag	PPN	V
0	03	B	1
	07	6	0
	28	3	1
	01	F	0
1	31	0	1
	12	3	0
	07	E	1
	0B	1	1
2	2A	A	0
	11	1	0
	1F	8	1
	07	5	1
3	07	3	1
	3F	F	0
	10	D	0
	32	0	0

(a) TLB

VPN	PPN	V	VPN	PPN	V
00	6	1	10	0	1
01	5	0	11	5	0
02	3	1	12	2	1
03	4	1	13	4	0
04	2	0	14	6	0
05	7	1	15	2	0
06	1	0	16	4	0
07	3	0	17	6	0
08	1	1	18	1	1
09	4	0	19	2	0
0A	3	0	1A	5	0
0B	2	0	1B	2	0
0C	5	0	1C	6	0
0D	6	0	1D	2	0
0E	1	1	1E	3	0
0F	0	0	1F	1	0

(b) Page Table

Index	Tag	V	Byte 0	Byte 1	Byte 2	Byte 3	Tag	V	Byte 0	Byte 1	Byte 2	Byte 3
0	19	1	99	11	23	11	00	0	99	11	23	11
1	15	0	4F	22	EC	11	2F	1	55	59	0B	41
2	1B	1	00	02	04	08	0B	1	01	03	05	07
3	06	0	84	06	B2	9C	12	0	84	06	B2	9C
4	C7	1	43	6D	8F	09	CD	1	8F	09	84	06
5	0D	1	36	32	00	78	1E	1	A1	B2	C4	DE
6	11	0	A2	37	68	31	00	1	BB	77	33	00
7	16	1	11	C2	11	33	F2	1	00	C0	0F	00

(c) Cache L1

Cuadro 1: Tablas de memoria

VA	VPN	VPO	TLBi	TLBt	PA	Co	Ci	Ct	¿TLB hit?	¿Cache hit?	¿Page fault?	Dato
59E5D												
088F3												
1B5B4												

c) Justifique si la eliminación de la TLB permitiría el acceso a algún dato al que estando la TLB presente no se puede acceder. Si es así, y en el inciso anterior no pudo acceder a algún o algunos datos, indique el o los datos que obtendría.

Lenguaje de máquina

3) Se pide implementar, en assembly x86_64 y siguiendo la convención de llamadas éstandar de la arquitectura, las siguientes tres funciones:

a)

```
// Calcula el tamaño de una cadena de C (terminada en '\0').
// Pre-condición: s != NULL.
size_t my_strlen(const char *s);
```

b)

```
// Compara dos cadenas por valores ASCII. Se devuelve 0 si las
// cadenas son iguales, -1 si 'a' es menor, 1 si 'a' es mayor.
// Una cadena es menor a otra cuando la primera posición en la
// que difieren o bien no existe, o bien es numéricamente inferior.
// Nota: no se permite invocar a otras funciones, incluida strlen.
// Pre-condición: a != NULL, b != NULL.
int my_strcmp(const char *a, const char *b);
```

c)

```
// Calcula si un arreglo de cadenas está ordenado de manera
// ascendente. (No se recibe la longitud del arreglo, pero el
// último elemento es siempre NULL.)
// Ejemplos de ordenamiento ascendente:
//     {"ab", "c", NULL}, {"ab", "bb", "bb", "bc", NULL}
// Nota: el valor 'true' se devuelve como 1 y 'false' como 0.
// Pre-condición: argv != NULL.
bool arr_ordenado(char **argv);
```

Nombre y apellido: _____ Padrón: _____

1) a) ¿Qué se entiende por riesgo por dependencia en los datos? En una arquitectura secuencial ¿qué riesgos están presentes y cómo se evitan estos?

---

b) ¿Qué se entiende por riesgo estructural?

---

c) ¿Cuál es la mínima cantidad de ciclos necesaria para ejecutar completamente *n* instrucciones en una CPU segmentada en *k* etapas?

---

d) Dados los siguientes retardos correspondientes a cada bloque correspondiente al camino de datos de una arquitectura:

F	D	E	M	W
250 ps	350 ps	150 ps	300 ps	200 ps

¿Cuáles son los frecuencias de reloj máximas en un procesador de arquitectura segmentada de cinco etapas y uno de arquitectura no segmentada? Considerar que la escritura en registros de pipeline demora 10 ps.

---

2) Considere el siguiente bucle, ejecutado en un procesador con un *datapath* de 5 etapas (el visto en clase):

```
1         irmovq  $16, %rcx
2 loop:   mrmovq  0(%rdx), %rax
3         mrmovq  8(%rdx), %rbx
4         addq    %rbx, %rax
5         subq    %rcx, %rdx
6         xorq    %rax, %rdx
7         jne     loop
```

a) Complete el cuadro siguiente con la evolución del estado del *pipeline*, únicamente para dos iteraciones, si el mismo cuenta con *data forwarding*, un predictor de saltos ideal, y registros de pipeline con capacidad de demorar la ejecución e insertar burbujas si fuese necesario. Indique claramente si se insertan burbujas y donde. *Aclaración: se sabe que al menos iterará dos veces.*

Instrucción	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
`irmovq $16,%rcx`	F	D															
		F															

3) Se tiene una caché "16-way" inicialmente vacía, con 4 sets y una capacidad total de 128 bytes. Las direcciones de memoria son de 7 bits, y la unidad de direccionamiento es un byte. La política de desalojo es LRU (*least recently used*).

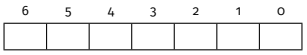
Dados los contenidos de la memoria principal que se muestran en la tabla adjunta, y una serie de operaciones a aplicar, se pedirá indicar cómo se ve afectada la caché tras cada operación.

Aclaración: Para hacer determinística la asignación de líneas en la cache, cada vez que sea posible elegir entre más de una línea, se deberá elegir la de menor índice numérico. (Esto quiere decir, por ejemplo, que en una cache vacía la primera operación siempre usará una línea con índice 0, en el set que corresponda.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	5B	1F	3E	A6	87	26	44	B3	78	8B	5A	AB	F6	5C	ED	B1
1	CF	A4	A5	B6	3C	99	3D	B7	2F	AE	8A	C4	E8	38	42	1C
2	BA	69	EB	83	E7	B4	D4	79	C1	14	47	06	93	22	05	9A
3	95	41	54	C3	33	43	01	16	9D	7F	7E	18	55	6B	89	B2
4	F5	1A	67	37	BE	45	35	7A	64	D6	21	EF	6A	59	92	02
5	6C	C5	FB	B0	C0	57	A1	DB	4A	D7	FE	76	0B	96	20	FF
6	27	49	5E	AD	70	91	24	68	51	8E	EC	63	58	3A	3F	E6
7	15	85	77	74	81	D2	B5	25	D8	AF	8C	3B	7D	11	FD	1E

Cuadro 1: Contenido de la memoria principal.

a) Se pide, en primer lugar, indicar en el esquema de direcciones físicas los bits utilizados para el *offset* (CO), *set* (CI) y *tag* (CT).



(a)

b) Se pide completar, para cada dirección y tipo de acceso, **y en secuencia temporal ordenada**, en qué set y línea se alojará cada valor leído (o escrito).

Aclaración: En la columna "byte/bloque" se debe anotar, en el caso de *cache hit* de lectura, el byte leído; en caso contrario (*cache miss* o escritura), se debe escribir la línea completa.

Recordatorio: La caché está inicialmente vacía, y la política de desalojo es LRU.

Dir (hex)	Op (R/W)	Set	Tag	¿Hit?	¿Desalojo?	Línea	Byte/Bloq.
1E	R						
0C	R						
3E	R						
1F	R						
3A	R						
2D	R						
4F	R						
68	R						
6E	R						
3B	R						
5F	R						
69	W (9F)						
1F	W (2D)						
3E	R						
1E	R						
2E	W (00)						
0D	W (63)						

4) A continuación se muestra el código C, incompleto, de una función y el código assembly x86-64 generado al compilar utilizando `gcc -std=c99 -Og -S`. A partir del código assembly, complete el código C.

```
long fun(const int * v,
         unsigned long l,
         unsigned long r,
         int t) {
    long p = ____________________;
    int m = ____________________;

    if (m == t)
        return ___________________;

    if (t < m)
        r = ____________________;
    else
        l = ____________________;

    if (l == r)
        return ___________________;

    return ___(v, ___, ___, ___);
}
```

```
fun:
.LFB10:
    leaq    (%rsi,%rdx), %rax
    shrq    $1, %rax
    movl    (%rdi,%rax,4), %r8d
    cmpl    %ecx, %r8d
    je      .L8
    cmpl    %ecx, %r8d
    jg      .L3
    leaq    1(%rax), %rsi
    movq    %rdx, %rax
.L3:
    cmpq    %rax, %rsi
    je      .L5
    subq    $8, %rsp
    movq    %rax, %rdx
    call    fun
    jmp     .L2
.L5:
    movq    $-1, %rax
    ret
.L2:
    addq    $8, %rsp
.L8:
    ret
```

Nombre y apellido: _____ Padrón: _____

1) En el siguiente ejercicio se pide hacer un seguimiento de pedidos de datos hechos por la CPU, utilizando memoria virtual. El sistema cuenta con una TLB, paginado de 1 nivel (1-level page table) y cache L1 (no dispone de cache L2 ni L3).

Las direcciones virtuales son de 20 bits y las direcciones físicas de 16 bits. La TLB es asociativa de 8 vías con un total de 32 entradas (se muestran 16 en la tabla). La memoria cache se muestra completa. Se muestran las primeras 32 entradas de la tabla de paginación. El *page size* es de 4096 bytes. Los accesos a memoria se realizan de a bytes.

Index	Tag	PPN	V
0	03	B	1
	07	6	0
	28	3	1
	01	F	0
1	31	0	1
	12	3	0
	07	E	1
	0B	1	1
2	2A	A	0
	11	1	0
	1F	8	1
	07	5	1
3	07	3	1
	3F	F	0
	10	D	0
	32	0	0

(a) TLB

VPN	PPN	V	VPN	PPN	V
00	6	1	10	0	1
01	5	0	11	5	0
02	3	1	12	2	1
03	4	1	13	4	0
04	2	0	14	6	0
05	7	1	15	2	0
06	1	0	16	4	0
07	3	0	17	6	0
08	1	1	18	1	1
09	4	0	19	2	0
0A	3	0	1A	5	0
0B	2	0	1B	2	0
0C	5	0	1C	6	0
0D	6	0	1D	2	0
0E	1	1	1E	3	0
0F	0	0	1F	1	0

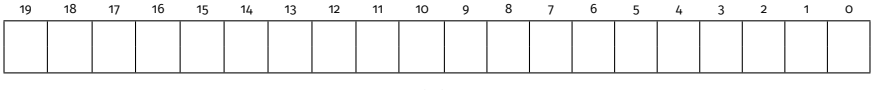
(b) Page Table

Index	Tag	V	Byte 0	Byte 1	Byte 2	Byte 3	Tag	V	Byte 0	Byte 1	Byte 2	Byte 3
0	19	1	99	11	23	11	00	0	99	11	23	11
1	15	0	4F	22	EC	11	2F	1	55	59	0B	41
2	1B	1	00	02	04	08	0B	1	01	03	05	07
3	06	0	84	06	B2	9C	12	0	84	06	B2	9C
4	C7	1	43	6D	8F	09	CD	1	8F	09	84	06
5	0D	1	36	32	00	78	1E	1	A1	B2	C4	DE
6	11	0	A2	37	68	31	00	1	BB	77	33	00
7	16	1	11	C2	11	33	F2	1	00	C0	0F	00

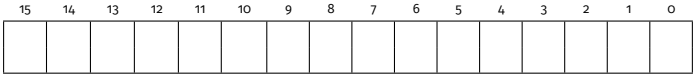
(c) Cache L1

Cuadro 1: Tablas de memoria

a) En los esquemas de las direcciones físicas (2b) y virtuales (2a) indique claramente los bits utilizados para el *offset* de la página física, el número de página física, el número de página virtual, el *offset* de la página virtual, el *offset* de cache, el índice de cache, el *tag* de cache, el *tag* de TLB, y el índice de TLB.



(a)



(b)

Cuadro 2: Esquemáticos de (a) direcciones virtuales, y (b) direcciones físicas

b) Complete la siguiente tabla. La columna VA indica las direcciones virtuales que pide la CPU:

VA	VPN	VPO	TLBi	TLBt	PA	Co	Ci	Ct	¿TLB hit?	¿Cache hit?	¿Page fault?	Dato
59E5D												
088F3												
1B5B4												

c) Justifique si la eliminación de la TLB permitiría el acceso a algún dato al que estando la TLB presente no se puede acceder. Si es así, y en el inciso anterior no pudo acceder a algún o algunos datos, indique el o los datos que obtendría.

2) A continuación se muestra el código C, incompleto, de una función y el código assembly x86-64 generado al compilar utilizando `gcc -std=c99 -Og -S`. A partir del código assembly, complete el código C.

```
long fun(const int * v,
         unsigned long l,
         unsigned long r,
         int t) {
    long p = ____________________;
    int m = ____________________;

    if (m == t)
        return ___________________;

    if (t < m)
        r = ____________________;
    else
        l = ____________________;

    if (l == r)
        return ___________________;

    return ___(v, ___, ___, ___);
}
```

```
fun:
.LFB10:
    leaq    (%rsi,%rdx), %rax
    shrq    $1, %rax
    movl    (%rdi,%rax,4), %r8d
    cmpl    %ecx, %r8d
    je      .L8
    cmpl    %ecx, %r8d
    jg      .L3
    leaq    1(%rax), %rsi
    movq    %rdx, %rax
.L3:
    cmpq    %rax, %rsi
    je      .L5
    subq    $8, %rsp
    movq    %rax, %rdx
    call    fun
    jmp     .L2
.L5:
    movq    $-1, %rax
    ret
.L2:
    addq    $8, %rsp
.L8:
    ret
```

3)

1. Las siguientes instrucciones poseen 2 riesgos *load/use*, entre las instrucciones 1. y 2., y entre las instrucciones 4. y 5. En la arquitectura PIPE, uno de ellos se puede salvar utilizando *load forwarding*. Justifique por qué puede salvarse uno y no el otro. *Ayuda:* destaque las diferencias entre las instrucciones.

```
1. mrmovq   0(%rcx),%rdx
2. pushq    %rdx
3. nop
4. popq     %rdx
5. rmmovq   %rax,0(%rdx)
```

2. En la implementación PIPE con *data forwarding*, inserción de burbujas/stalling, e *instruction squashing* se ejecuta el siguiente código Y86-64. Dibuje el diagrama de tiempos correspondiente. La predicción de saltos es saltar siempre.

```
0x000:      xorq %rax,%rax
0x002:      jne target
0x00b:      irmovq $1, %rax
0x015:      halt
0x016:  target:
0x016:      irmovq $2, %rdx
0x020:      irmovq $3, %rbx
0x02a:      halt
```

	Instrucción	1	2	3	4	5	6	7	8	9	10	11	12
0x000:	xorg %rax,%rax	F											

Nombre y apellido: _____ Padrón: _____

1) a) Dada las siguientes definiciones

```
struct a {      |  struct b {      |  struct c {       |  union d {
  int x;        |    int v[3];     |    short v[9];    |    char *m;
  void *p;      |    long x;       |    int *x;        |    struct {
};              |    void *p;      |    char c;        |      int n;
                |  };              |  };               |      char x;
                |                  |                   |    } s;
                |                  |                   |  };
```

dé el valor de las siguientes expresiones:

`sizeof(struct a):` _____ `sizeof(struct c):` _____

`sizeof(struct b):` _____ `sizeof(union u):` _____

b) Dadas las siguientes cuatro funciones:

```
int  get_a_x(struct a *s) { return s->x; }
long get_b_x(struct b *s) { return s->x; }
int* get_c_x(struct c *s) { return s->x; }
char get_d_x(union d *s)  { return s->x; }
```

se pide indicar su implementación en lenguaje *assembly* (no es necesario incluir la instrucción `ret`).

`get_a_x:` _____ `get_c_x:` _____

`get_b_x:` _____ `get_d_x:` _____

c) Dadas las siguientes definiciones

```
long v[] = {20, 30, 40, 50};
char *s  = "hola";
char p[] = "mundo";
```

dé el valor de las siguientes expresiones:

`sizeof(v):` _____ `sizeof(s):` _____

`sizeof(*v):` _____ `sizeof(p):` _____

`sizeof(&v[0]):` _____

d) Indique el valor de las siguientes expresiones (con las variables definidas antes):

`*(s + 3):` _____ `strlen(&p[1]):` _____

`&v[3] - &v[1]:` _____ `strlen(&p[5]):` _____

`&p[3] - &p[1]:` _____ `p[4] - p[2]:` _____

2) Considere la siguiente representación en punto flotante de 7 bits, basada en el formato de la IEEE:

- El bit más significativo es un bit de signo (s),
- le siguen 3 bits de exponente (e), y
- los últimos 3 bits son la fracción (f).

Se cumplen las mismas reglas que en el estándar IEEE para definir números normalizados, denormalizados, infinitos, NaN y representación del 0. Complete la siguiente tabla, donde:

binario es la representación en 7 bits

M es el significando. Puede ser un número x o x/y, donde x e y son enteros. Ejemplo: 0, 3/256.

E es el exponente (¡distinto de e!).

valor es el valor numérico representado.

Si tiene que redondear, hágalo según las reglas de redondeo al par más cercano (*Round-To-Even*).

Descripción	**binario**	**M**	**E**	**valor**
Menos cero				$-0{,}0$
—	1100101			
Denormalizado menor (positivo)				
Normalizado mayor (negativo)				
Uno				1,0
—				13,3
—				$-18{,}25$
Infinito positivo				$+\infty$
Not-A-Number				NaN

3) Dadas las siguientes implementaciones de funciones en C y el código en assembly, responda:

a) ¿A qué función corresponde el código assembly? _____

b) Indique los valores de las constantes H: _____ y K _____

c) La función `abs( )` fue reemplazada por el compilador directamente en el código.

a) Indique las líneas del código assembly donde se ve la función: desde _____ hasta _____ (inclusive)

b) Indique breve y concisamente qué hace la función: _____

a)
```
int ff1(int m[][K], int n) {
  int aux = m[0][0];
  unsigned long i, j;
  if (n > H) n = H;
  for (i = 0; i < n; i++) {
    for (j = 0; j < K; j++) {
      int z = m[i][j];
      if (z > aux)
        aux = z;
    }
  }
  return aux;
}
```

b)
```
int ff3(int m[][K], int n) {
  int aux = m[0][0];
  unsigned long i, j;
  if (n > H) n = H;
  for (i = 0; i < n; i++) {
    for (j = 0; j < K; j++) {
      int z = m[i][j];
      if (z > aux)
        aux = z;
    }
  }
  return abs(aux);
}
```

c)
```
int ff2(int m[][K], int n) {
  int aux = m[0][0];
  unsigned long i, j;
  if (n > H) n = H;
  for (i = 0; i < n; i++) {
    for (j = 0; j < K; j++) {
      int z = abs(m[i][j]);
      if (z > aux)
        aux = z;
    }
  }
  return aux;
}
```

d)
```
int ff6(int m[][K], int n) {
  int aux = m[0][0];
  unsigned long i, j;
  if (n > H) n = H;
  for (i = 0; i < n; i++) {
    for (j = 0; j < K; j++) {
      int z = m[i][j];
      if (z < aux)
        aux = z;
    }
  }
  return abs(aux);
}
```

e)
```
int ff4(int m[][K], int n) {
  int aux = m[0][0];
  unsigned long i, j;
  if (n > H) n = H;
  for (i = 0; i < n; i++) {
    for (j = 0; j < K; j++) {
      int z = m[i][j];
      if (z < aux)
        aux = z;
    }
  }
  return aux;
}
```

f)
```
int ff5(int m[][K], int n) {
  int aux = m[0][0];
  unsigned long i, j;
  if (n > H) n = H;
  for (i = 0; i < n; i++) {
    for (j = 0; j < K; j++) {
      int z = abs(m[i][j]);
      if (z < aux)
        aux = z;
    }
  }
  return aux;
}
```

assembly

```
1         .file   "ffi.c"
2         .text
3         .globl   ffi
4         .type   ffi, @function
5   ffi:
6   .LFB0:
7         .cfi_startproc
8         movl    (%rdi), %eax
9         cmpl    $35, %esi
10        movl    $34, %edx
11        cmovge    %edx, %esi
12        movl    $0, %r8d
13        jmp    .L3
14  .L5:
15        leaq    (%r8,%r8,2), %rdx
16        leaq    (%r8,%rdx,4), %rdx
17        leaq    (%rdi,%rdx,4), %rdx
18        movl    (%rdx,%rcx,4), %edx
19        movl    %edx, %r9d
20        sarl    $31, %r9d
```

```
        xorl        %r9d, %edx
        subl        %r9d, %edx
        cmpl        %edx, %eax
        cmovg        %edx, %eax
        addq        $1, %rcx
        jmp        .L6
.L9:
        movl        $0, %ecx
.L6:
        cmpq        $12, %rcx
        jbe        .L5
        addq        $1, %r8
.L3:
        movslq        %esi, %rdx
        cmpq        %rdx, %r8
        jb        .L9
        rep ret
        .cfi_endproc
.LFE0:
        .size        ffi, .-ffi
        .ident        "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.11) 5.4.0 20160609"
        .section        .note.GNU-stack,"",@progbits
```

Nombre y apellido: _____ Padrón: _____

1) Indique si las siguientes afirmaciones son verdaderas o falsas, y justifique brevemente la respuesta.
a) Dada una *direct-mapped cache* ($E = 1$), y otra *fully associative cache* ($S = 1$), ambas con la misma capacidad total (C) y tamaño de bloque (B); entonces, para una misma secuencia de operaciones, el número de desalojos en la segunda no puede ser superior al número de desalojos en la primera, pero sí inferior.

b) En el paginado con dos niveles de la arquitectura x86/32 (PDX=10 bits, PTX=10 bits), las consultas a la TLB se realizan exclusivamente mediante los 10 bits del PTX (*page table index*).

c) Dedicar una caché distinta para las instrucciones (*fetch*) y los datos (*read*/*write*) mejora el rendimiento frente a una única caché unificada (compartida).

d) Dada una función que suma todos los elementos de una matriz, el impacto positivo de la caché en el tiempo de ejecución se debe a la localidad temporal de las operaciones.

2) Se tiene una caché "4-way" inicialmente vacía, con 8 sets y una capacidad total de 64 bytes. Las direcciones de memoria son de 7 bits, y la unidad de direccionamiento es un byte. La política de desalojo es LRU (*least recently used*).

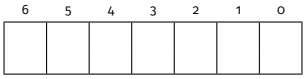
Dados los contenidos de la memoria principal que se muestran en la tabla adjunta, y una serie de operaciones a aplicar, se pedirá indicar cómo se ve afectada la caché tras cada operación.

Aclaración: Para hacer determinística la asignación de líneas en la cache, cada vez que sea posible elegir entre más de una línea, se deberá elegir la de menor índice numérico. (Esto quiere decir, por ejemplo, que en una cache vacía la primera operación siempre usará una línea con índice 0, en el set que corresponda.)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	5B	1F	3E	A6	87	26	44	B3	78	8B	5A	AB	F6	5C	ED	B1
1	CF	A4	A5	B6	3C	99	3D	B7	2F	AE	8A	C4	E8	38	42	1C
2	BA	69	EB	83	E7	B4	D4	79	C1	14	47	06	93	22	05	9A
3	95	41	54	C3	33	43	01	16	9D	7F	7E	18	55	6B	89	B2
4	F5	1A	67	37	BE	45	35	7A	64	D6	21	EF	6A	59	92	02
5	6C	C5	FB	B0	C0	57	A1	DB	4A	D7	FE	76	0B	96	20	FF
6	27	49	5E	AD	70	91	24	68	51	8E	EC	63	58	3A	3F	E6
7	15	85	77	74	81	D2	B5	25	D8	AF	8C	3B	7D	11	FD	1E

Cuadro 1: Contenido de la memoria principal.

a) Se pide, en primer lugar, indicar en el esquema de direcciones físicas los bits utilizados para el *offset* (CO), *set* (CI) y *tag* (CT).



(a)

b) Se pide completar, para cada dirección y tipo de acceso, **y en secuencia temporal ordenada**, en qué set y línea se alojará cada valor leído (o escrito).

Aclaración: En la columna "byte/bloque" se debe escribir, en el caso de *cache hit* de lectura, el byte leído; en caso contrario (*cache miss* o escritura), se debe escribir la la línea completa (los 2^b bytes).

Recordatorio: La caché está inicialmente vacía, y la política de desalojo es LRU.

Dir (hex)	Op (R/W)	Set	Tag	¿Hit?	¿Desalojo?	Línea	Byte/Bloq.
1E	R						
0C	R						
3E	R						
1F	R						
3A	R						
2D	R						
4F	R						
68	R						
6E	R						
3B	R						
5F	R						
69	W (9F)						
1F	W (2D)						
3E	R						
1E	R						

3) En el siguiente ejercicio se pide hacer un seguimiento de pedidos de datos hechos por la CPU, utilizando memoria virtual. El sistema cuenta con una TLB, paginado de 1 nivel (1-level page table) y cache L1 (no dispone de cache L2 ni L3).

Las direcciones virtuales son de 17 bits y las direcciones físicas de 14 bits. La TLB y la cache se muestran **completas**, mientras únicamente se muestran 48 entradas (ordenadas) de la tabla de paginación. El *page size* es de 128 bytes.

a) En los esquemas de las direcciones físicas (4b) y virtuales (4a) indique claramente los bits utilizados para el *offset* de la física, el número de página física, el número de página virtual, el *offset* de la página virtual, el *offset* de cache, el índice de cache, el *tag* de cache, el *tag* de TLB, y el índice de TLB.

b) Complete la siguiente tabla. La columna VA indica las direcciones virtuales que pide la CPU:

Advertencia: si una tabla (3a, 3b, 3c) se encuentra incompleta, es posible que una dirección lleve a necesitar un dato no presente en la tabla. En dicho caso se debe indicar y procesar la siguiente dirección virtual.

Index	Tag	PPN	V	Tag	PPN	V
0	ED	5E	1	CF	1D	0
	C4	5A	1	79	69	1
	2C	0E	1	89	0E	0
	57	2C	1	5E	65	1
1	65	2F	0	4E	4C	1
	AB	1E	0	B3	74	0
	94	72	0	92	5D	1
	F4	46	0	A5	26	0
2	52	26	0	34	31	1
	AF	10	1	BF	4B	1
	01	74	1	A0	67	0
	BA	1D	1	DF	0E	0
3	C3	5B	0	51	42	0
	6F	0C	1	D1	06	0
	F9	44	0	8D	24	1
	1E	07	1	2E	0A	1

(a) TLB

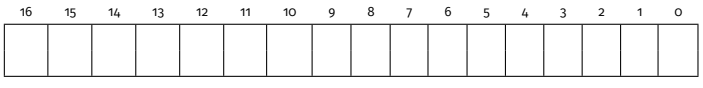
VPN	PPN	V	VPN	PPN	V	VPN	PPN	V
016	0D	1	169	2C	1	2FA	6A	0
031	2B	0	178	69	0	314	6C	0
056	09	1	198	48	1	329	51	1
069	61	1	1AD	26	0	32B	4E	0
08C	4A	0	1D9	41	0	35F	37	1
093	53	1	1F8	11	0	37D	2A	1
098	6D	0	212	50	0	390	66	0
0B3	1E	0	22D	62	0	3A7	0C	0
0B6	40	1	250	49	1	3B9	75	1
0C0	46	0	276	3E	0	3C4	38	0
0E2	36	0	28A	78	0	3D3	72	1
105	7C	0	28D	76	0	3D8	3B	1
113	13	0	293	4F	0	3E8	2D	0
146	31	0	2A4	3C	0	3EA	73	0
153	68	1	2A5	18	0	3ED	0A	0
159	43	0	2CC	39	0	3F9	5F	1

(b) Page Table

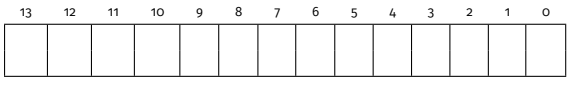
I	T	V	B7	B6	B5	B4	B3	B2	B1	B0	T	V	B7	B6	B5	B4	B3	B2	B1	B0
0	32	1	93	DC	44	05	8E	08	CD	15	E4	1	BF	FC	49	5E	BA	BD	FB	5A
	60	1	D7	74	45	E3	61	20	3F	23	BD	1	AE	F5	9D	79	01	15	49	60
1	7C	1	60	CD	CE	3E	AE	BE	38	CD	C6	0	46	6A	D2	56	FD	91	73	DD
	14	0	C5	40	D4	04	32	86	83	62	59	1	67	20	AD	8D	D8	92	8E	1A
2	6A	1	E4	4E	8A	71	2A	06	4D	03	E3	1	1D	55	A5	1B	DB	8B	0D	C3
	65	1	1E	4A	66	7E	13	B8	0A	CB	FE	0	2B	15	4E	9F	0B	6B	81	DA
3	68	1	4B	28	D1	8B	60	C1	00	02	8B	1	D8	C0	BB	AC	07	34	B9	C7
	EC	1	46	D3	04	6C	17	25	60	7D	BA	1	DD	34	CD	BF	80	CF	33	19
4	11	0	75	EA	C4	B1	1F	BF	88	A6	7F	1	3C	92	CA	D3	D3	40	32	16
	B5	1	C4	75	D1	63	9D	8F	9C	7C	93	1	45	25	58	48	4B	54	FA	76
5	02	0	89	22	41	A7	FE	6C	C1	E6	73	0	12	5F	0D	FE	A4	9F	20	01
	F3	0	65	87	9E	20	08	14	14	2A	BB	1	BE	1C	EA	6C	6D	02	EE	EE
6	46	0	F7	A2	0A	52	91	F4	6C	60	00	0	59	A9	0C	72	76	BC	1E	FB
	74	1	FA	13	A6	1D	CE	44	23	EE	EF	1	52	5F	54	12	E3	3F	1E	25
7	FF	1	6C	E6	D1	7F	0E	31	E0	1B	21	1	A5	FA	8D	AE	32	A8	50	1B
	6E	1	C0	C2	F9	1F	F4	A6	01	AA	E8	1	43	AB	44	CE	A6	3C	35	31

(c) Cache L1

Cuadro 3: Tablas de memoria



(a)



(b)

Cuadro 4: Esquemáticos de (a) direcciones virtuales, y (b) direcciones físicas

VA	VPN	VPO	TLBi	TLBt	PA	Co	Ci	Ct	¿TLB hit?	¿Cache hit?	¿Page fault?	Dato
1AFBD												
0D6D1												
15F7A												
0B4C9												
12E02												

Nombre y apellido: _____ Padrón: _____

1) En el siguiente ejercicio se pide hacer un seguimiento de pedidos de datos hechos por la CPU, utilizando memoria virtual. El sistema cuenta con una TLB, paginado de 1 nivel (1-level page table) y cache L1 (no dispone de cache L2 ni L3).

Las direcciones virtuales son de 16 bits y las direcciones físicas de 13 bits. La TLB y la cache se muestran **completas**, mientras únicamente se muestran las primeras 32 entradas de la tabla de paginación. El *page size* es de 512 bytes.

Index	Tag	PPN	V
0	09	4	1
	12	2	1
	10	0	1
	08	5	1
	05	7	1
	13	1	0
	10	3	0
	18	3	0
1	04	1	0
	0C	1	0
	12	0	1
	08	1	0
	06	7	0
	03	1	1
	07	5	0
	02	2	1

(a) TLB

VPN	PPN	V	VPN	PPN	V
00	6	1	10	0	1
01	5	0	11	5	0
02	3	1	12	2	1
03	4	1	13	4	0
04	2	0	14	6	0
05	7	1	15	2	0
06	1	0	16	4	0
07	3	0	17	6	0
08	5	1	18	1	1
09	4	0	19	2	0
0A	3	0	1A	5	0
0B	2	0	1B	2	0
0C	5	0	1C	6	0
0D	6	0	1D	2	0
0E	1	1	1E	3	0
0F	0	0	1F	1	0

(b) Page Table

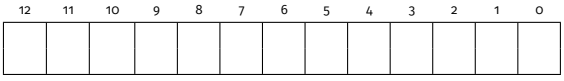
Index	Tag	V	Byte 0	Byte 1	Byte 2	Byte 3	Tag	V	Byte 0	Byte 1	Byte 2	Byte 3
0	19	1	99	11	23	11	00	0	99	11	23	11
1	15	0	4F	22	EC	11	2F	1	55	59	0B	41
2	1B	1	00	02	04	08	0B	1	01	03	05	07
3	06	0	84	06	B2	9C	12	0	84	06	B2	9C
4	07	0	43	6D	8F	09	05	0	43	6D	8F	09
5	0D	1	36	32	00	78	1E	1	A1	B2	C4	DE
6	11	0	A2	37	68	31	00	1	BB	77	33	00
7	16	1	11	C2	11	33	0E	1	00	C0	0F	00

(c) Cache L1

Cuadro 1: Tablas de memoria

a) En los esquemas de las direcciones físicas (2b) y virtuales (2a) indique claramente los bits utilizados para el *offset* de la página física, el número de página física, el número de página virtual, el *offset* de la página virtual, el *offset* de cache, el índice de cache, el *tag* de cache, el *tag* de TLB, y el índice de TLB.

(a)



(b)

b) Complete la siguiente tabla. La columna VA indica las direcciones virtuales que pide la CPU:

VA	VPN	VPO	TLBi	TLBt	PA	Co	Ci	Ct	¿TLB hit?	¿Cache hit?	¿Page fault?	Dato
1DDE												
37E4												
4B6A												

2) Dadas las funciones que se muestran a continuación ¿cuál de ellas corresponde con el código assembly x86 mostrado?

```
int fun1(int a, int b) {
    if (a < b)
        return a;
    else
        return b;
}
```

```
int fun2(int a, int b) {
    if (b < a)
        return b;
    else
        return a;
}
```

```
int fun3(int a, int b) {
    unsigned ua = (unsigned) a;
    if (ua < b)
        return b;
    else
        return ua;
}
```

```
    pushl   %ebp
    movl    %esp, %ebp
    movl    8(%ebp), %edx
    movl    12(%ebp), %eax
    cmpl    %eax, %edx
    jge     .L9
    movl    %edx, %eax
.L9
    movl    %ebp, %esp
    popl    %ebp
    ret
```

3) Para la siguientes instrucciones, ejecutadas en la implementación PIPE completa vista en clase (5-stage pipeline con data forwarding), grafique el diagrama de tiempos completo indicando claramente, si fuese necesaria, la inserción de burbujas y las instrucciones donde se aprovecha la existencia de data forwarding. Describa el estado de los registros de pipeline en el ciclo 8.

```
irmovq  $128, %rdx
irmovq  $3, %rcx
rmmovq  %rcx, 0(%rdx)
irmovq  $10, %rbx
mrmovq  0(%rdx), %rax
addq    %rbx, %rax
halt
```

Instrucción	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
`irmovq $128,%rdx`	F																

Nombre y apellido: _____ Padrón: _____

1) a) Dada las siguientes definiciones

```
struct a {   |  struct b {    |  union u {
  int x;     |    char s[3];  |    char *m;
  void *p;   |    int i;      |    struct {
};           |    short v[5]; |      int n;
             |    void *p;    |      char x;
             |    unsigned u; |    } s;
             |  };            |  };
```

dé el valor de las siguientes expresiones:

`sizeof(struct a):` _____ `sizeof(struct b):` _____ `sizeof(union u):` _____

b) Dadas las siguientes cuatro funciones:

```
int  get_a_x(struct a *s) { return s->x; }
long get_b_v3(struct b *s) { return s->v[3]; }
int  get_u_x(union u *s)  { return s->s.n; }
```

se pide indicar su implementación en lenguaje *assembly* (no es necesario incluir la instrucción `ret`).

`get_a_x:` _____ `get_b_v3:` _____ `get_u_x:` _____

c) Dadas las siguientes definiciones

```
long v[] = {20, 30, 40, 50};
char *s  = "hola";
```

dé el valor de las siguientes expresiones:

`sizeof(v):` _____ `sizeof(&v[0]):` _____

`sizeof(*v):` _____ `sizeof(s):` _____

2) Considere la siguiente representación en punto flotante de 7 bits, basada en el formato de la IEEE:

- El bit más significativo es un bit de signo (s),
- le siguen 3 bits de exponente (e), y
- los últimos 3 bits son la fracción (f).

Se cumplen las mismas reglas que en el estándar IEEE para definir números normalizados, denormalizados, infinitos, NaN y representación del 0. Complete la siguiente tabla, donde:

binario es la representación en 7 bits

M es el significando. Puede ser un número x o x/y, donde x e y son enteros. Ejemplo: 0, 3/256.

E es el exponente (¡distinto de e!).

valor es el valor numérico representado.

Si tiene que redondear, hágalo según las reglas de redondeo al par más cercano (*Round-To-Even*).

Descripción	**binario**	**M**	**E**	**valor**
Menos cero				$-0{,}0$
—	1100101			
Denormalizado menor (positivo)				
Normalizado mayor (negativo)				
Uno				1,0
—				13,3
—				$-18{,}25$
Infinito positivo				$+\infty$
Not-A-Number				NaN

3) Un grupo de arquitectos diseñando una nueva implementación del *datapath* del procesador determinó los siguientes retardos en el hardware a utilizar:

instruction memory	140 ps
decode	70 ps
register fetch	130 ps
ALU	130 ps
data memory	190 ps
register write-back	110 ps

Además de los 20 ps de cada *pipeline register*.

a) ¿Cuál es el valor mínimo del ciclo de clock para una implementación secuencial? ¿Cuál es la frecuencia máxima del procesador?

b) ¿Cuál es el valor del clock más rápido que se puede obtener con un *datapath* segmentado en 5 etapas? ¿Cuánto más rápido es que la implementación secuencial (como cociente de las frecuencias del clock)? *Ayuda: al combinar las etapas, los circuitos van en el orden listados. Por ejemplo, no se pueden unir en una etapa el* register write-back *y el* decode*, sin incluir en la misma etapa* register fetch*,* ALU *y* data memory*.*

c) ¿Cuál es el valor del clock más rápido que se puede obtener con un *datapath* segmentado en 9 etapas? ¿Cuánto más rápido es que la implementación secuencial (como cociente de las frecuencias del clock)? *Ayuda: asuma que cualquiera de los circuitos dados se puede separar en k-partes iguales.*

4) Se pide implementar, en assembly x86_64 y siguiendo la convención de llamadas éstandar de la arquitectura, las siguientes tres funciones:

a)

```
// Calcula el tamaño de una cadena de C (terminada en '\0').
// Pre-condición: s != NULL.
size_t my_strlen(const char *s);
```

b)

```
// Compara dos cadenas por valores ASCII. Se devuelve 0 si las
// cadenas son iguales, -1 si 'a' es menor, 1 si 'a' es mayor.
// Una cadena es menor a otra cuando la primera posición en la
// que difieren o bien no existe, o bien es numéricamente inferior.
// Nota: no se permite invocar a otras funciones, incluida strlen.
// Pre-condición: a != NULL, b != NULL.
int my_strcmp(const char *a, const char *b);
```

c)

```
// Calcula si un arreglo de cadenas está ordenado de manera
// ascendente. (No se recibe la longitud del arreglo, pero el
// último elemento es siempre NULL.)
// Ejemplos de ordenamiento ascendente:
//     {"ab", "c", NULL}, {"ab", "bb", "bb", "bc", NULL}
// Nota: el valor 'true' se devuelve como 1 y 'false' como 0.
// Pre-condición: argv != NULL.
bool arr_ordenado(char **argv);
```

Nombre y apellido: _____ Padrón: _____

Arquitectura

1) Dado el siguiente pseudo-*dump* de código **Y86-64**:

```
0x000: 30f20900000000000000 |     irmovq $9, %rdx
0x00a: 30f31500000000000000 |     irmovq $21, %rbx
0x014: 6123                 |     subq   %rdx, %rbx
0x016: 30f48000000000000000 |     irmovq $128,%rsp
0x020: 4043640000000000000 |     rmmovq %rsp, 100(%rbx)
0x02a: 00                   |     halt
```

a) Complete la siguiente tabla donde se describe el procesamiento de la instrucción `irmovq $128, %rsp`, en una arquitectura secuencial. *Tache las líneas que no corresponden para la instrucción dada.*

	`irmovq $128,%rsp`
Fetch	`icode:ifun` ← M_1[_____] = _____
	`rA:rB` ← M_1[_____] = _____
	`valC` ← M_8[_____] = _____
	`valP` ← _____
Decode	`valA` ← `R`[_____] = _____
	`valB` ← `R`[_____] = _____
Execute	`valE` ← _____ = _____
	`Establecer CC`
Memory	M_8[_____] ← _____ = _____
	_____ ← M_8[_____] = _____
Write Back	`R`[_____] ← _____ = _____
PC Update	`PC` ← _____ = _____

b) Complete el cuadro siguiente suponiendo una arquitectura segmentada de 5 etapas (`F`, `D`, `E`, `M`, `W`) donde en el *pipeline* se resuelven correctamente los riesgos.

Instrucción	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
`irmovq $9,%rdx`	F	D															
		F															

c) Suponga una arquitectura segmentada que no puede hacer frente a ningún tipo de riesgo. Introduzca los nops necesarios para que código Y86-64 anterior se ejecute y arroje los resultados correctos.

Memoria Virtual y Cache

2) a) En una computadora típica, con un CPU con TLB, caches L1, L2, y L3, memoria DRAM, y disco rígido ¿cuándo sucede un *page-fault*?

b) ¿Dónde se encuentra alojada la tabla de paginado (*page table*) y de qué propiedad se beneficia para resultar en una solución útil en la traducción de direcciones?

c) ¿Qué evento dispara un *page miss* y quién lo resuelve?

3) Indique si las siguientes afirmaciones son verdaderas o falsas, y justifique brevemente la respuesta.

a) Dada una *direct-mapped cache* ($E = 1$), y otra *fully associative cache* ($S = 1$), ambas con la misma capacidad total (C) y tamaño de bloque (B); entonces, para una misma secuencia de operaciones, el número de desalojos en la segunda no puede ser superior al número de desalojos en la primera, pero sí inferior.

b) En el paginado con dos niveles de la arquitectura x86/32 (PDX=10 bits, PTX=10 bits), las consultas a la TLB se realizan exclusivamente mediante los 10 bits del PTX (*page table index*).

c) Dedicar una caché distinta para las instrucciones (*fetch*) y los datos (*read*/*write*) mejora el rendimiento frente a una única caché unificada (compartida).

d) Dada una función que suma todos los elementos de una matriz, el impacto positivo de la caché en el tiempo de ejecución se debe a la localidad temporal de las operaciones.

4) La siguiente tabla muestra los parámetros de diferentes memorias cache donde: m es el número de bits de la dirección física, C es el tamaño de la cache (cantidad de bytes de datos), B es el tamaño del bloque en bytes, y E es la cantidad de líneas por set. Para cada cache, indique la cantidad de sets (S) en la cache, bits correspondientes al cache tag (t), bits correspondientes al set index (s), y bits correspondientes al block offset (b).

Cache	*m*	*C*	*B*	*E*	*S*	*t*	*s*	*b*
1.	32	1024	4	4				
2.	32	1024	4	256				
3.	32	1024	8	1				
4.	32	1024	8	128				
5.	32	1024	32	1				
6.	32	1024	32	4				

Lenguaje de máquina

5) La siguiente función `main( )` en assembly x86 de 32 bits imprime los argumentos de un programa (sin incluir `argv[0]`) cuando su longitud es múltiplo de 8. Sin embargo, el código tiene varios errores: sobreescribe registros callee-saved, la comprobacion de la longitud esta mal implementada, y el programa termina con un segmentation fault.

Se pide:

a) Una versión corregida del código para x86 de 32 bits que subsane estos errores. (Nota: no se puede cambiar "ret" por "_exit".)

b) Una versión del código corregido que funcione en x86 de 64 bits, respetando las diferencias en la convención de llamadas entre las dos arquitecturas.

```
.globl main
main:
    xorl %ebx, %ebx
    movl 4(%esp), %esi
    movl 8(%esp), %edi
    jmp .loop_cond

.loop:
    movl (%edi, %ebx, 4), %eax

    call strlen
    cmpl $8, %eax
    jne .loop_cond

    push %eax
    call puts
    subl $4, %esp
.loop_cond:
    incl %ebx
    cmpl %ebx, %esi
    jae .loop

    movl $0, %eax
    ret
```

Nombre y apellido: _____ Padrón: _____

1. *a*) Dada las siguientes definiciones

```
struct a {          struct b {          struct c {          union u {          union v {
  int v[4];           int v[4];           short v[9];         int i;              char *m;
  void *p;            char c;             void *p;            char *s;            struct {
};                    void *p;          };                  };                    int x;
                    };                                                              void *p;
                                                                                  } s;
                                                                                };
```

dé el valor de las siguientes expresiones:

`sizeof(struct a):` _____ `sizeof(union u):` _____

`sizeof(struct b):` _____ `sizeof(union v):` _____

`sizeof(struct c):` _____

b) Dada las siguientes definiciones

```
int v[] = {100, 200, 300, 400};
char *s = "hola";
char arr[] = {'h', 'o', 'l', 'a', '\0'};
char ars[] = "hola";
```

dé el valor de las siguientes expresiones:

`sizeof(v);` _____ `sizeof(arr);` _____

`sizeof(*v);` _____ `sizeof(arr[1]);` _____

`sizeof(s);` _____ `sizeof(ars);` _____

`sizeof(*s);` _____ `sizeof(&ars[2]);` _____

c) Indique el valor de las siguientes expresiones (con las variables definidas antes)

`&s[3] - s:` _____ `strlen(s + 2):` _____

`&v[3] - &v[1]:` _____ `strlen(&arr[1]):` _____

`(v + 2) - v:` _____ `strlen(&ars[4]):` _____

d) [+0.5 pts] Dé el valor de las siguientes expresiones:

`strlen((char *)(v + 1))` _____ `strlen((char *)(v + 2))` _____

e) [+0.5 pts] Dé el valor de las siguientes expresiones:

`strlen(((char *) v) + 6)` _____ `strlen(((char *) v) + 9)` _____

2. *a*) Considere la siguiente representación en punto flotante de 9 bits, basada en el formato de la IEEE:
   - El bit más significativo es un bit de signo (`s`),
   - le siguen 4 bits de exponente (`e`), y
   - los últimos 4 bits son la fracción (`f`).

   Se cumplen las mismas reglas que en el estándar IEEE para definir números normalizados, denormalizados, infinitos, NaN y representación del 0. Complete la siguiente tabla, donde:

   binario es la representación en 9 bits
   M es el significando. Puede ser un número x o x/y, donde x e y son enteros. Ejemplo: 0, 3/256.
   E es el exponente (¡distinto de `e`!).
   valor es el valor numérico representado.

Descripción	binario	M	E	valor
Menos cero				$-0{,}0$
—	001000101			
Denormalizado menor (positivo)				
Normalizado mayor (negativo)				
Uno				1,0
—				13,3
Infinito positivo				$+\infty$
Not-A-Number				NaN

b) Dadas las definiciones `int a, b;` y teniendo en cuenta que los enteros signados se representan en complemento a dos. Además, las constantes `INT_MIN` e `INT_MAX` representan los valores mínimo y máximo que puede tener un entero de 32 bits, y la constante `W = 31`. Una las descripciones de la izquierda con las expresiones de la derecha.

1. ∼a (complemento a uno)

   _____

2. a

   _____

3. a * 7

   _____

4. (a < 0) ? 1 : -1

   _____

a. `~(~a | (b ^ (INT_MIN + INT_MAX)))`

b. `((a ^ b) & ~b) | (~(a ^ b) & b)`

c. `1 + (a << 3) + ~a`

d. `(a << 4) + (a << 2) + (a << 1)`

e. `((a < 0) ? (a + 3) : a) >> 2`

f. `a ^ (INT_MIN + INT_MAX)`

g. `~((a | (~a + 1)) >> W) & 1`

h. `~((a >> W) << 1)`

i. `a >> 2`

3. A partir del siguiente código C,

```
int mglobal[FILAS][COLUMNAS];
long int funcion(size_t f, size_t c, int * v) {
    size_t i, j;
    *v = ____________;
    for (i = ____________; ____________; ____________) {
        mglobal[i][i] = ____________;
        for (j = ____________; ____________; ____________) {
            mglobal[i][i] += ____________;
        }
        *v += ____________;
    }
    return (*v > sizeof(mglobal)) ? ____________ : -1;
}
```

se obtuvo el código assembly que se lista a continuación:

```
funcion:
    movl    $0, %r8d
    movl    $-7, (%rdx)
    jmp .L2
.L5:
    leaq    (%r8,%r8,2), %rax
    salq    $5, %rax
    movl    $-1, mglobal(%rax)
    leaq    1(%r8), %r11
    movq    %r11, %r9
    jmp .L3
.L4:
    leaq    (%r8,%r8), %rax
    leaq    (%rax,%r8), %rcx
    salq    $3, %rcx
    subq    %r8, %rcx
    addq    %r9, %rcx
    addq    %r8, %rax
    salq    $5, %rax
    movl    mglobal(%rax), %r10d
    addl    mglobal(,%rcx,4), %r10d
    movl    %r10d, mglobal(%rax)
    addq    $1, %r9
.L3:
    cmpq    %rsi, %r9
    jb  .L4
    leaq    (%r8,%r8,2), %rax
    salq    $5, %rax
    movl    mglobal(%rax), %eax
    addl    %eax, (%rdx)
    movq    %r11, %r8
.L2:
    cmpq    %rdi, %r8
    jb  .L5
    movl    (%rdx), %edx
    movslq  %edx, %rax
    cmpl    $1289, %edx
    movq    $-1, %rdx
    cmovb   %rdx, %rax
    ret
```

a) Complete las expresiones incompletas en el código C de forma tal que, si se compila del mismo modo, se obtiene un código assembly similar al mostrado.

b) A partir del código assembly, encuentre el valor de las constantes `FILAS` y `COLUMNAS` definidas en el código C.