

# examen-bp4.pdf



**BrokenQuagga**



**Arquitectura de Computadores**



**2º Grado en Ingeniería Informática**

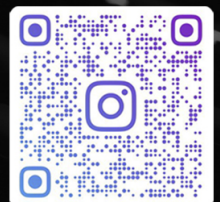


**Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación  
Universidad de Granada**



**LA PRIMERA RESIDENCIA GAMING  
EN EL MUNDO ABRE EN MADRID**

**ESCANEA Y PARTICIPA EN EL  
SORTEO DE UN ALIENWARE**



**[gamingresidences.com](http://gamingresidences.com)**

**[info@gamingresidences.com](mailto:info@gamingresidences.com)**



EMMA STONE

Disney

Cruella

EN  
CINES

O DISFRÚTALA  
EN

Disney+

A TRAVÉS DE  
ACCESO PREMIUM  
Con coste adicional

©2021 Disney Enterprises, Inc. Requiere de una suscripción activa a Disney+ y un único pago adicional por el Acceso Premium a Cruella. Disfrútala en aquellos cines donde esté disponible y con Acceso Premium antes de que esté disponible para todos los suscriptores a Disney+.





28/5/2021

SWAD: plataforma de apoyo a la docencia / UGR /

## Examen BP4 - Grupo A2



Universidad de Granada - Grado en Ingeniería Informática  
Arquitectura de Computadores



- 1** ¿Cuál de los siguientes códigos dirías que tiene menor tiempo de ejecución?  
Elección única Usuario Profesores
- ☐ a) Todas las opciones tienen el mismo tiempo de ejecución.
  - ☐ b) 

```
switch ( queue ) {
    case 0 : letter = 'W';
        break;
    case 1 : letter = 'S';
        break;
    case 2 : letter = 'U';
        break;
}
```
  - ☐ c) 

```
static char *classes = "WSU";
letter = classes [queue];
```
  - ☐ d) 

```
if ( queue == 0)
    letter = 'W';
else if ( queue == 1)
    letter = 'S';
else
    letter = 'U';
```

- 2** ¿Cuál de las siguientes alternativas es mejor para implementar  $ebx = (A < B) ? C1 : C2$  teniendo en cuenta que: se repite su ejecución en el programa 100 veces, todas las instrucciones suponen siempre 1 ciclo (entonces el Código 2 supone siempre 6 ciclos y el Código 1 supone siempre 4 ciclos si no se salta en la instrucción de salto condicional), la predicción del salto condicional es siempre No Saltar, la predicción falla 2 de cada 10 ejecuciones del salto condicional y suponiendo que la penalización por predicción incorrecta es siempre 8 ciclos?

Código 1:

```
cmpl    %ebx, %eax
jge     .L1
```

WUOLAH

```

    movl    %ecx, %ebx // ecx=C1
    jmp     .L2
.L1:
    movl    %edx, %ebx // edx=C2
.L2:

```

Código 2:

```

xorl    %ebx, %ebx
cmpl    %ebx, %eax
setlge  %b1
decl    %ebx
andl    %ecx, %ebx // ecx=C1-C2
addl    %edx, %ebx // edx=C2

```

Usuario Profesores

- ☐ a) Código 1 en este caso
- ☐ b) Hay un empate entre los dos
- ☐ c) Código 2 en este caso
- ☐ d) Falta información

**3**

¿Cuál cree que es la implementación óptima del siguiente algoritmo?

Elección única

```

int f(int n)
{
    int s = 0;
    for (int i = 0; i < n; ++i)
        s += i % 5 + 1;
    return s;
}

```

Usuario Profesores

- ☐ a) ninguna otra respuesta es correcta
- ☐ b) f(int):
 

```

xorl    %ecx, %ecx
xorl    %r8d, %r8d
movl    $5, %esi
.L3:
    cmpl    %edi, %ecx
    jge     .L1
    movl    %ecx, %eax
    incl    %ecx
    cltd
    idivl   %esi
    leal    1(%r8,%rdx), %r8d
    jmp     .L3
.L1:
    movl    %r8d, %eax
    ret

```
- ☐ c) f(int):
 

```

leal    (%rdi,%rdi,2), %eax
ret

```
- ☐ d) f(int):
 

```

leal    0(%rdi,4), %eax
ret

```

**4**

¿Qué código cree que calculará de forma correcta y en menor tiempo el producto de dos matrices en un sistema multiprocesador? Suponga matrices cuadradas, c inicializada a cero y N muy grande.

Elección única

```

int a[N][N], b[N][N], c[N][N];

```


Usuario Profesores


- ☐ a) for (int i=0; i<N; ++i)


```

for (int j=0; j<N; ++j)
    for (int k=0; k<N; ++k)
        c[i][j] += a[i][k] * b[k][j];

```

 b) for (int i=0; i<N; ++i)  
 #pragma omp parallel for  
 for (int j=0; j<N; ++j)  
 for (int k=0; k<N; ++k)  
 #pragma omp critical  
 c[i][j] += a[i][k] \* b[k][j];

 c) for (int i=0; i<N; ++i)  
 #pragma omp parallel for  
 for (int j=0; j<N; ++j)  
 for (int k=0; k<N; ++k)  
 c[i][j] += a[i][k] \* b[k][j];

•  d) for (int i=0; i<N; ++i)  
 #pragma omp parallel for  
 for (int j=0; j<N; ++j)  
 for (int k=0; k<N; ++k)  
 #pragma omp atomic  
 c[i][j] += a[i][k] \* b[k][j];

## 5 Indique qué opción se ejecutará más rápido dados


Elección única


```


const int n = 1000000;
int a[n], b[n];


```

Usuario Profesores

•  a) int tmp0=0, tmp1=0, tmp2=0, tmp3=0;  
 for (i=0 ; i<n ; i+=4) {  
 tmp0 += a[i] \* b[i];  
 tmp1 += a[i+1] \* b[i+1];  
 tmp2 += a[i+2] \* b[i+2];  
 tmp3 += a[i+3] \* b[i+3];  
 }  
 \*p = tmp0 + tmp1 + tmp2 + tmp3;

 b) for (i=0 ; i<n ; ++i) {  
 \*p += a[i] \* b[i];  
 }





 c) for (i=0 ; i<n ; i++) {  
 \*p = \*p + a[i] \* b[i];  
 }

 d) for (i=0 ; i<n ; i+=4) {  
 \*p += a[i] \* b[i];  
 \*p += a[i+1] \* b[i+1];  
 \*p += a[i+2] \* b[i+2];  
 \*p += a[i+3] \* b[i+3];  
 }

## 6 ¿Cuál de las siguientes ventajas aporta la técnica de desenrollado de bucle?

Elección única

Usuario Profesores

-  a) aumenta la velocidad de ejecución
-  b) elimina todas las instrucciones de salto del bucle
-  c) reduce el tamaño del código
-  d) ninguna otra respuesta es correcta

## 7 ¿Cuál de las siguientes versiones de una función que multiplica un entero por 6 cree que se obtendrá al compilar con optimización en espacio (-Os)?

Elección única

```

int f(int x)
{

```

28/5/2021

SWAD: plataforma de apoyo a la docencia / UGR /

```
return x * o;
}
```

Usuario Profesores

- ☐ a) ninguna otra respuesta es correcta
- ☐ b) 0x401106 <+0>: lea (%rdi,%rdi,2),%eax  
0x401109 <+3>: add %eax,%eax  
0x40110b <+5>: retq
- ☒ c) 0x401116 <+0>: imul \$0x6,%edi,%eax  
0x401119 <+3>: retq
- ☐ d) 0x401106 <+0>: push %rbp  
0x401107 <+1>: mov %rsp,%rbp  
0x40110a <+4>: mov %edi,-0x4(%rbp)  
0x40110d <+7>: mov -0x4(%rbp),%edx  
0x401110 <+10>: mov %edx,%eax  
0x401112 <+12>: add %eax,%eax  
0x401114 <+14>: add %edx,%eax  
0x401116 <+16>: add %eax,%eax  
0x401118 <+18>: pop %rbp  
0x401119 <+19>: retq

8

Elección única

¿Cuál de las siguientes formas de implementar el mismo algoritmo cree más rápida?


```
const int N = 5000, REP = 40000;
int R[REP + 1];
struct S { int a, b; } s[N];
Usuario Profesores
```

- ☐ a) for ( int ii = 1; ii <= REP ; ++ ii )  
{  
int X1 = 0 , X2 = 0;  
for ( int i = 0; i < N ; ++ i )  
X1 += 2 \* s [ i ]. a + ii;  
for ( int i = 0; i < N ; ++ i )  
X2 += 3 \* s [ i ]. b - ii;  
if ( X1 < X2 )  
R [ ii ] = X1;  
else  
R [ ii ] = X2;  
}
- ☐ b) struct { int x1 , x2; } x[N];  
  
for ( int i = 0; i < N ; ++ i )  
{  
x [ i ]. x1 = 2 \* s [ i ]. a ;  
x [ i ]. x2 = 3 \* s [ i ]. b ;  
}
- ☒ c) for ( int ii = 1; ii <= REP ; ++ ii )  
{  
int x1 = 0 , x2 = 0;  
for ( int i = 0; i < N ; ++ i )  
{  
x1 += x [ i ]. x1 + ii ;  
x2 += x [ i ]. x2 - ii ;  
}  
R [ ii ] = std :: min ( x1 , x2 ) ;  
}
- ☐ d) for ( int ii = 1; ii <= REP ; ++ ii )  
{  
int x1 = 0 , x2 = 0;  
for ( int i = 0; i < N ; ++ i )  
{  
x1 += x [ i ]. x1 + ii ;  
x2 += x [ i ]. x2 - ii ;  
}  
R [ ii ] = std :: min ( x1 , x2 ) ;  
}

```

for ( int i = 0; i < N ; ++ i )
{
    x1 += 2 * s [ i ]. a + ii;
    x2 += 3 * s [ i ]. b - ii;
}
R [ ii ] = std :: min ( x1 , x2 );
}

```

-  d) 

```

int sa = 0 , sb = 0;
for (int i = 0; i < N ; ++ i)
{
    sa += s [i]. a;
    sb += s [i]. b;
}
sa *= 2;
sb *= 3;
for (int ii = 1; ii <= REP ; ++ ii)
    R[ii] = std :: min (sa + N * ii , sb - N * ii);

```

**9**

Elección única


¿Cómo cree que implementará el compilador una función que multiplica un entero por 11 al compilar con optimización máxima (-O3)?

```

int function_f(int x)
{
    return x * 11;
}


```

Usuario Profesores

-  a) 


```

0x401120 <+0>: lea    (%rdi,%rdi,4),%eax
0x401123 <+3>: lea    (%rdi,%rax,2),%eax
0x401126 <+6>: retq

```
-  b) 


```

0x401106 <+0>: push   %rbp
0x401107 <+1>: mov    %rsp,%rbp
0x40110a <+4>: mov    %edi,-0x4(%rbp)
0x40110d <+7>: mov    -0x4(%rbp),%edx
0x401110 <+10>: mov    %edx,%eax
0x401112 <+12>: shl    $0x2,%eax
0x401115 <+15>: add    %edx,%eax
0x401117 <+17>: add    %eax,%eax
0x401119 <+19>: add    %edx,%eax
0x40111b <+21>: pop    %rbp
0x40111c <+22>: retq

```
-  c) 

```

0x401116 <+0>: imul   $0xb,%edi,%eax
0x401119 <+3>: retq





```
-  d) ninguna otra respuesta es correcta

**10**

Elección única

Sin indicarle un núcleo concreto, ¿cómo ordenaría las instrucciones con enteros en orden creciente de tiempo de ejecución?

Usuario Profesores

-  a) Multiplicación, división y desplazamiento de bits
-  b) Desplazamiento de bits, división y multiplicación
-  c) Desplazamientos de bits, multiplicación y división
-  d) División, desplazamiento de bits y multiplicación