Cognoms: . . . . . Nom: . . . . . .

1er Control Arquitectura de Computadors **Curs 2013-2014 Q2**

Problema 1. (5 puntos)

Ferrari nos ha encargado producir un sistema de control de inyección para el motor del F1 de Fernando Alonso. Dicho sistema tiene un procesador RISC similar al MIPS donde todas las instrucciones tienen 4 bytes de tamaño. El sistema de inyección ejecuta el siguiente bucle infinito:

```
while (true) do {
    leer_muestras_sensores();
    procesar_muestras();
    modificar_parametros_inyección();
}
```

El tiempo de dicho bucle esta totalmente dominado por **`procesar_muestras()`** y consideraremos despreciable el tiempo usado por el resto del código. Se desea poder procesar una muestra cada milisegundo y sabemos que cada ejecución de **`procesar_muestras()`** ejecuta 1 millon de instrucciones dinámicas y realiza 300.000 operaciones de punto flotante.

a) **Calcula** el rendimiento mínimo en MIPS y en MFLOPS que deberia tener el procesador

MIPS = 10^6 / (10^-3 * 10^6) = 1.000
MFLOPS= 3*10^5 / (10^-3 * 10^6) = 300

b) **Calcula** el ancho de banda mínimo que deberia poder sostener la cache de instrucciones

Ancho de banda = (4B * 10^6instr) / 10^-3 s = 4*10^9 B/s

La siguiente tabla muestra la distribución de instrucciones por tipos y el CPI medio de cada tipo para la rutina **`procesar_muestras()`**

	punto flotante	**enteras**	**memoria**
% de instrucciones	50%	25%	25%
CPI	2	1	5

c) **Calcula** la frecuencia mínima a la que deberia funcionar el procesador

Teje = N*CPI*Tc --> (Tc = 1/f) --> f = N*CPI / Teje =

= [10^6*(2*0.5+1*0.25+5*0.25)] / 10^-3 = 2.5*10^9 Hz

La corriente de fugas de dicho procesador es despreciable. Sin embargo, la potencia dinámica consumida se considera excesiva (recuerda: potencia dinámica = CV^2F). Nuestros arquitectos han sugerido añadir instrucciones SIMD al repertorio de instrucciones. La carga capacitiva del nuevo procesador SIMD ha aumentado un 20%. En el código recompilado para el nuevo repertorio de instrucciones el número de instrucciones de punto flotante se ha reducido a la mitad.

d) **Calcula** a que frecuencia mínima debería operar el nuevo procesador SIMD

Nuevo #instr = 10^6 - (10^6*0.5 / 2) = 750.000 instr

Teje = N * CPI * 1/f --> f = N*CPI / Teje =
= 750.000*(2*0.25+1*0.25+5*0.25)*100/75 / 10^-3 = 2 GHz

Operando a esta nueva frecuencia se puede reducir el voltaje del procesador en un 10%.

e) **Calcula** la ganancia en potencia (porcentaje de mejora) del procesador con instrucciones SIMD respecto el original.

Ganancia = Pold / Pnew = (C * V^2 * 2.5*10^9) / 1.2*C * (0.9*V)^2 * 2*10^9 =

= 2.5 / 1.2 * 0.9^2 * 2 = 1.286

Cognoms: . . . . . Nom: . . . . . .

1er Control Arquitectura de Computadors **Curs 2013-2014 Q2**

Problema 2. (5 puntos)

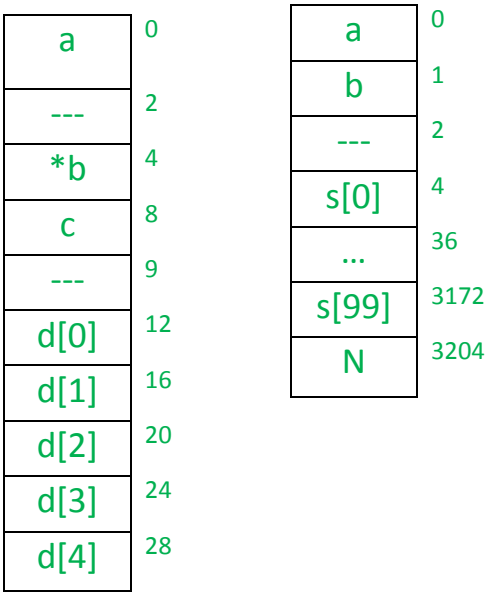
Dado el siguiente código escrito en C, que compilamos para un sistema linux de 32 bits:

```
typedef struct {
  short a;
  int *b;
  char c;
  int d[5];
} s1;
```

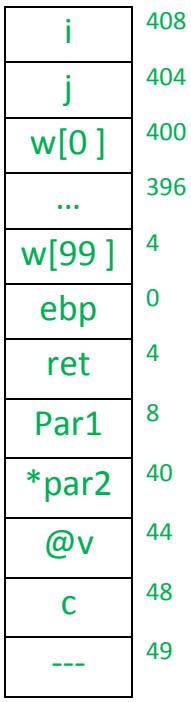
```
typedef struct {
  char a, b;
  s1  v[100];
  int N;
  } s2;
```

```
void ExaBA(S1 par1, S2 *par2, S1 v[100], char c) {
  int i,j;
  int w[100];
  ...
}
```

a) **Dibuja** como quedarían almacenadas en memoria las estructuras **s1** y **s2**, indicando claramente los deplazamientos respecto al inicio, el tamaño de todos los campos y el tamaño de los structs.



b) **Dibuja** el bloque de activación de la rutina `ExaBa`.



Dado el siguiente código escrito en C, que compilamos para un sistema linux de 32 bits:

```
void Code1(char v[], int N){
  int i;
  for (i=0; i<N; i++)
    if (v[i] != 'A')
      v[i] = Code2(v[i]);
}
```

```
char Code2(char c){
  return c ^ 0xC1;
}
```

Nota:
Las funciones usan el %al para retornar caracteres.

c) **Traduce** las subrutinas **Code1** y **Code2** a ensamblador del x86:

```
Code1:
        Pushl %ebp
        Movl %esp, %ebp
        Subl $4, %esp
        Pushl %esi
        Movl $0, %esi

For:    cmpl 12(%ebp), %esi
        jge fifor

if:     cmpb $'A', 8(%ebp,%esi,4)
        je fiif
        pushl 8(%ebp,%esi,4)
        call Code2
        addl $4, %ebp
        movl %eax, 8(%ebp,%esi,4)

fiif:   incl %esi
        jmp for

fifor:  popl %esi
        movl %ebp, %esp
        popl %ebp
        ret
```

```
Code2:
        Pushl %ebp
        Movl %esp, %ebp

        Xorl %eax, %eax
        Movb 8(%ebp), %al
        Xorb $0xC1, %al

        movl %ebp, %esp
        popl %ebp
        ret
```