Nombre: _____

Nota: _____ / _____

Lab 2a Castellano

Parte 1

Alumno1
Nombre ____ Apellido1 ____ Apellido2 ____

Alumno2
Nombre ____ Apellido1 ____ Apellido2 ____

EJECUCIÓN DEL CÓDIGO ORIGINAL **apxpy.s**

a) Configuración: riesgos de control *stalls* y riesgos de datos *stalls*.

Contenido vector *z*: 90-91-92-93...105 (16 elementos)
Análisis del tiempo de ejecución del programa original:

Instrucciones	Stalls	Ciclos totales	CPI
166	178	348	2,1

(5 + 10 (16) +1) (348-4-166) (178+4+166) 348/166

b) Configuración: riesgos de control *predict not taken* y riesgos de datos *stalls*.
Indicar cuántas instrucciones se abortan cuando se ejecuta un salto Efectivo: 3 instrucciones
Análisis del tiempo de ejecución del programa:

Instrucciones	Stalls	Ciclos totales	CPI	Mejora sobre el original
166	**175**	**345**	**2,08**	**2,1 / 2,08 = 1,0096**
(5 + 10*16 +1	**(345 - 166 - 4)**	**(166 + 4 + 175)**		

c) Configuración: riesgos de control*predictnottaken* y riesgos de datos anticipación.

Análisis del tiempo de ejecución del programa:

Instrucciones	Stalls	Ciclos totales	CPI	Mejora sobre el original
166	**77**	**247**	**1,49**	**2,1 / 1,49 = 1,4094**

Las instrucciones son las mismas (es el mismo programa), en cuanto a ciclos de parada, nos comemos Dos por instrucciones normales en cada pasada del bucle (16 veces) y 3 del salto (15/16 veces)

OPTIMIZACIÓN DEL CÓDIGO apxpy.s

a) Configuración: *predict not taken* y anticipación:

Análisis del tiempo de ejecución del programa:

Instrucciones	Stalls	Ciclos totales	CPI	Mejora sobre el original
166	45	215	1,3	2,1 / 1,3 = 1,615

Simplemente moviendo la segunda load del bucle una posición arriba arreglamos los Ciclos de parada y ahora solo nos penaliza el salto que al ser predict not taken predice Que saltará y se equivoca todas las veces menos la última (15 veces) cancelando 3 Instrucciones cada vez (3 * 15) = 45 ciclos de parada .
Los ciclos totales son instrucciones + stalls + 4 (son los ciclos de carga (tienen que pasar 4 ciclos Para poder obtener un resultado por ciclo

b) Configuración: *delayslot 1* y anticipación.

Análisis del tiempo de ejecución del programa:

Instrucciones	Stalls	Ciclos totales	CPI	Mejora sobre el original
166	0	170	1,02	2,1 / 1,02 = 2,06

Hemos colocado dadd r3,r3,#8 tras el salto para rellenar el delay slot y hemos puesto la dadd r2,r2 #8 entre la seq y el salto

Ya no tenemos ningún ciclo de parada, ciclos = instrucciones + 4 (ciclos iniciales que hacen falta para tener un resultado por ciclo)

(ver comentario)

c) Configuración: *delayslot 3* y anticipación.

Análisis del tiempo de ejecución del programa:

Instrucciones	Stalls	Ciclos totales	CPI	Mejora sobre el original
166	0	170	1,02	2,1 / 1,02 = 2,06

En esta ocasión coo necesitamos rellenar con tres instrucciones útiles el delay Slot , hemos puesto tras el salto :

```
dadd r14,r12,r14
sd r14,0(r3)
dadd r3,r3,#8
```

PROGRAMA ORDENACIÓN DE UN VECTOR (ordena.s)

a) Configuración: *predict not taken* y anticipación:

Análisis del tiempo de ejecución del programa original:

Instrucciones	Stalls	Ciclos totales	CPI
454	177	635	1,4

Hay un ciclo de parada tras cada load y tres ciclos de parada tras cada salto

b) Configuración: *delayslot 1* y anticipación:

Análisis del tiempo de ejecución del programa:

Instrucciones	Stalls	Ciclos totales	CPI	Mejora sobre el original
464	**144**	**612**	**1,32**	1,4/1,32

```
        .data
a:      .dword  9,8,7,6,5,4,3,2,1,0
afin:

        .text
start:  dadd r1,r0,a
        dadd r5,r0,afin
        dsub r4,r5,#8

        ld r6,0(r1)

loopi:

        dadd r2,r1,#8
        ld r7,0(r2)
loopj:

        sgt r3,r6,r7
        beqz r3, endif
then:
        sd r7,0(r1)
        sd r6,0(r2)
        dadd r6,r7,r0
endif:
        dadd r2,r2,#8
        seq r3,r2,r5
        beqz r3,loopj
        ld r7,0(r2)
        dadd r1,r1,#8
        seq r3,r1,r4
        beqz r3,loopi
        ld r6,0(r1)

        trap #0
```