Práctica
Ingeniería de Computadores II

Nombre: Jorge Martinez Pazos

DNI: 39510046W

Correo: jmartinez5741@alumno.uned.es

Centro Asociado: Vigo (Pontevedra)

Contenido

ENUNCIADO: 3
CÓDIGO ESCALAR: 4
REORGANIZACIÓN DE CÓDIGO: 4
DETENCIONES: 7
CÓDIGO VECTORIAL 9
REORGANIZACIÓN DE CÓDIGO: 9
DETENCIONES: 12
ESTADÍSTICAS ESCALAR / VECTORIAL 14
ESCALAR: 14
VECTORIAL: 15
PRUEBAS CON N: 16
VECTORIAL: 16
ESCALAR: 21
APÉNDICE A: 26
APÉNDICE B: 28
CONCLUSIÓN: 31

ENUNCIADO:

Considere el siguiente código en el que se utilizan valores en coma flotante de 8 bytes (doble precisión):

```
for (k=n; k>0; k--)
     x[k]:= b[k];
     for (i=k+1; i<=n; i++)
        x[k]:=x[k]-x[i]*U[k,i];
     end for;
end for;
```

El bucle codifica el algoritmo de sustitución hacia atrás para resolver un sistema de ecuaciones lineales **Ux=b** en el que **U** es una matriz triangular superior de dimensión *n* x *n* en la que los coeficientes de la diagonal principal son 1, y **x** y **b** son vectores columna de dimensión *n x* 1. La siguiente figura sirve para ilustrar el aspecto de las tres matrices que conforman el sistema de ecuaciones:

$$\begin{pmatrix} 1 & u_{12} & u_{13} & & u_{1n} \\ 0 & 1 & u_{23} & \cdots & u_{2n} \\ 0 & 0 & 1 & & u_{3n} \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{pmatrix}$$

Utilizando el simulador del procesador DLXV, denominado WinDLXV y disponible en el curso virtual, se pide que:

a) Programe el código anterior en un fichero denominado BACK_ SUBSTITUTION _ESC.S. Recuerde que, aunque la matriz sea triangular superior, los coeficientes de la diagonal principal tienen que ser 1 para poder aplicar el algoritmo, por lo que, si no lo son, deberá añadir el código necesario para ello. Utilice las directivas del ensamblador con el objeto de reservar espacio de memoria para las tres matrices `U`, `x` y `b`, y cargue las direcciones de comienzo en los registros `R1`, `R2` y `R3`, respectivamente. En el manual de usuario del procesador dispone de información sobre el uso y funciones de las directivas. ¡¡¡¡¡Y si tiene dudas, acuda al curso virtual para formularlas!!!!!

b) Programe el código del apartado (a) recurriendo a instrucciones vectoriales. El nombre del fichero debe ser BACK_SUBSTITUTION_VEC.S.

c) En base a los dos códigos que ha desarrollado en los apartados previos, argumente cuál es la mejor opción para usar teniendo en cuenta el tamaño del sistema de ecuaciones a resolver. Para ello analice a qué tiende el valor del CPI en función del valor de *n* en ambos casos, es decir, calcule

$$\lim_{n \to \infty} CPI$$

y dibuje una gráfica en la que se aprecie le evolución de este indicador del rendimiento.

Algunas observaciones para tener en cuenta al realizar la práctica:

- En los apartados (a) y (b) debe comentar en la memoria las detenciones que se producen y las razones de ello.
- Todas las ejecuciones y cálculos se efectuarán con el adelantamiento de resultados (*forwarding*) entre etapas habilitado y las segmentaciones de las unidades funcionales habilitadas.
- Utilice las latencias de las unidades funcionales en coma flotante que, por defecto, trae el simulador.
- Cualquier decisión que tome sobre el desarrollo de la práctica (código adicional, configuración del simulador, etc.) debe argumentarse adecuadamente en la memoria.

CÓDIGO ESCALAR:

El código fuente se encuentra en el apéndice A.

REORGANIZACIÓN DE CÓDIGO:

A la hora de escribir el código es vital organizarlo correctamente, pues pequeños cambios pueden llevar a mejoras significativas en el rendimiento del programa, disminución de ciclos, cargas o parones.

Estadísticas

Ejecución

246 Ciclos
159 Instrucciones
1.547 Ciclos de reloj por instrucción (CPI)

Configuración hardware

Memoria
20 Kbytes, distribuida en 64 bancos

Direcciones de comienzo por defecto

0x00000100 Código
0x00001000 Datos

Unidades funcionales escalares

Unidad funcional	Segmentación	Latencia
Suma/Resta FP	SI	4
Multiplicación FP	SI	7
División FP	NO	25

Unidades funcionales vectoriales

Unidad funcional	Segmentación	Latencia
Suma/Resta FP	SI	7
Multiplicación FP	SI	8
División FP	NO	21
Carga/Almac. Vectorial	SI	13

Cola de instrucciones vectoriales
5 Longitud

Registros vectoriales
2 Puertos de lectura
1 Puertos de escritura

Características

Adelanto de resultados activado
Tratamiento de los saltos: predicción de no tomar
Encadenamiento vectorial no activado

Saltos

15 Saltos efectivos
10 Saltos no efectivos
25 Saltos totales

Instrucciones de carga/almacenamiento

24 Cargas escalares
0 Cargas vectoriales
10 Almacenamientos escalares
0 Almacenamientos vectoriales

Instrucciones de punto flotante

6 Sumas/Restas
6 Multiplicaciones
0 Divisiones

Instrucciones vectoriales

0 Sumas/Restas de vectores
0 Multiplicaciones de vectores
0 Divisiones de vectores

Traps

1 Traps

Parones

104 RAW (lectura después de escritura)
0 WAW (escritura después de escritura)
0 WAR (escritura después de lectura)
0 Estructurales

Estado de la memoria

	Tamaño	Dirección de comienzo
Código	0140 Bytes	0x00000100
Datos	0196 Bytes	0x00001000

Estadísticas

Ejecución

236 Ciclos
159 Instrucciones
1.484 Ciclos de reloj por instrucción (CPI)

Configuración hardware

Memoria
20 Kbytes, distribuida en 64 bancos

Direcciones de comienzo por defecto

0x00000100 Código
0x00001000 Datos

Unidades funcionales escalares

Unidad funcional	Segmentación	Latencia
Suma/Resta FP	SI	4
Multiplicación FP	SI	7
División FP	NO	25

Unidades funcionales vectoriales

Unidad funcional	Segmentación	Latencia
Suma/Resta FP	SI	7
Multiplicación FP	SI	8
División FP	NO	21
Carga/Almac. Vectorial	SI	13

Cola de instrucciones vectoriales
5 Longitud

Registros vectoriales
2 Puertos de lectura
1 Puertos de escritura

Características

Adelanto de resultados activado
Tratamiento de los saltos: predicción de no tomar
Encadenamiento vectorial no activado

Saltos

15 Saltos efectivos
10 Saltos no efectivos
25 Saltos totales

Instrucciones de carga/almacenamiento

24 Cargas escalares
0 Cargas vectoriales
10 Almacenamientos escalares
0 Almacenamientos vectoriales

Instrucciones de punto flotante

6 Sumas/Restas
6 Multiplicaciones
0 Divisiones

Instrucciones vectoriales

0 Sumas/Restas de vectores
0 Multiplicaciones de vectores
0 Divisiones de vectores

Traps

1 Traps

Parones

88 RAW (lectura después de escritura)
0 WAW (escritura después de escritura)
0 WAR (escritura después de lectura)
0 Estructurales

Estado de la memoria

	Tamaño	Dirección de comienzo
Código	0140 Bytes	0x00000100
Datos	0196 Bytes	0x00001000

```
iniciobucle:
                BEQZ r5, findebucle     ; si k = 0, salto al final

                subi r17, r5, 1         ; k-1 porque bits van de 0 a 31
                mult r8, r17, r7        ; k * 8 para desplazamiento
                add r9, r3, r8          ; puntero b[k]
                add r10, r2, r8         ; puntero x[k]
                ld f0, 0(r9)            ; carga b[k] en f0
                sd 0(r10), f0           ; x[k] = b[k]

                addi r11, r5, 1 ; i = k + 1
```

```
iniciobucle:
                BEQZ r5, findebucle     ; si k = 0, salto al final

                subi r17, r5, 1         ; k-1 porque bits van de 0 a 31
                mult r8, r17, r7        ; k * 8 para desplazamiento

                add r9, r3, r8          ; puntero b[k]
                add r10, r2, r8         ; puntero x[k]
                ld f0, 0(r9)            ; carga b[k] en f0

                addi r11, r5, 1 ; i = k + 1
                sd 0(r10), f0           ; x[k] = b[k]
```

```
iniciobuclei:
                SLE r12, r11, r6        ; si i <= n, continuar
                BEQZ r12, findebuclei

                subi r18, r11, 1        ; i-1 porque bits van de 0 a 31
                mult r13, r17, r6       ; k * n
                add r13, r13, r18       ; k * n + i
                mult r13, r13, r7       ; (k * n + i) * 8
                add r14, r1, r13        ; puntero U[k, i]
                ld f2, 0(r14)           ; carga U[k, i]

                mult r15, r18, r7       ; i * 8
                add r16, r2, r15        ; puntero x[i]
                ld f4, 0(r16)           ; carga x[i]

                ld f6, 0(r10)           ; carga x[k]

                multd f8, f4, f2        ; x[i] * U[k, i]
                subd f6, f6, f8         ; x[k] = x[k] - (x[i] * U[k, i])
                sd 0(r10), f6           ; guardar x[k]

                addi r11, r11, 1        ; i++
                J iniciobuclei
```

```
iniciobuclei:
                SLE r12, r11, r6        ; si i <= n, continuar
                BEQZ r12, findebuclei

                subi r18, r11, 1        ; i-1 porque bits van de 0 a 31

                mult r13, r17, r6       ; k * n
                mult r15, r18, r7       ; i * 8
                add r13, r13, r18       ; k * n + i
                add r16, r2, r15        ; puntero x[i]
                mult r13, r13, r7       ; (k * n + i) * 8
                ld f4, 0(r16)           ; carga x[i]
                add r14, r1, r13        ; puntero U[k, i]
                ld f2, 0(r14)           ; carga U[k, i]

                ld f6, 0(r10)           ; carga x[k]

                multd f8, f4, f2        ; x[i] * U[k, i]
                addi r11, r11, 1        ; i++
                subd f6, f6, f8         ; x[k] = x[k] - (x[i] * U[k, i])
                sd 0(r10), f6           ; guardar x[k]

                J iniciobuclei
```

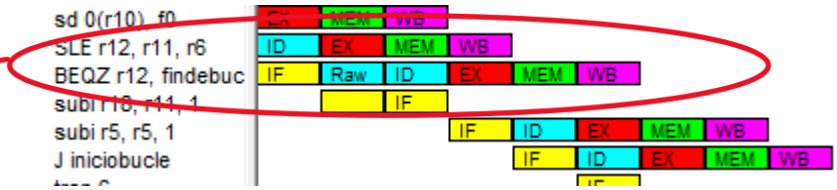
Basándonos en la diferencia de instrucciones, estos pequeños cambios han mejorado el programa:

(246/236) – 1 = 4,2%

Reduciendo la cantidad de ciclos por instrucción y la cantidad de detenciones.

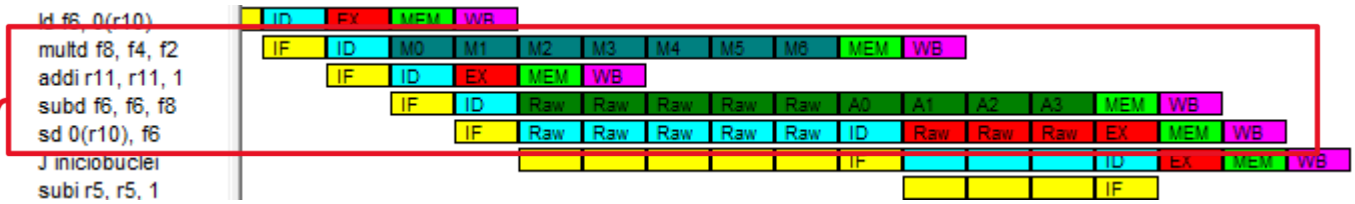
DETENCIONES:

Al comenzar el bucle i, hay una parada RAW por cada iteración, lo que serían en total, y teniendo en cuenta que n = 4, 1 + 2 + 3 + 4 = 10 RAW




```
iniciobuclei:
                SLE r12, r11, r6        ; si i <= n, continuar
                BEQZ r12, findebuclei

                subi r18, r11, 1        ; i-1 porque bits van de 0 a 31

                mult r13, r17, r6       ; k * n
                mult r15, r18, r7       ; i * 8
                add r13, r13, r18       ; k * n + i
                add r16, r2, r15        ; puntero x[i]
                mult r13, r13, r7       ; (k * n + i) * 8
                ld f4, 0(r16)           ; carga x[i]
                add r14, r1, r13        ; puntero U[k, i]
                ld f2, 0(r14)           ; carga U[k, i]

                ld f6, 0(r10)           ; carga x[k]

                multd f8, f4, f2        ; x[i] * U[k, i]
                addi r11, r11, 1        ; i++
                subd f6, f6, f8         ; x[k] = x[k] - (x[i] * U[k, i])
                sd 0(r10), f6           ; guardar x[k]
```

Más adelante, en el propio bucle i, se producen 13 errores RAW por cada iteración del bucle que se realiza de manera completa. 1 + 2 + 3 = 6 * 13 = 78 RAW.




```
iniciobuclei:
                SLE r12, r11, r6        ; si i <= n, continuar
                BEQZ r12, findebuclei

                subi r18, r11, 1        ; i-1 porque bits van de 0 a 31

                mult r13, r17, r6       ; k * n
                mult r15, r18, r7       ; i * 8
                add r13, r13, r18       ; k * n + i
                add r16, r2, r15        ; puntero x[i]
                mult r13, r13, r7       ; (k * n + i) * 8
                ld f4, 0(r16)           ; carga x[i]
                add r14, r1, r13        ; puntero U[k, i]
                ld f2, 0(r14)           ; carga U[k, i]

                ld f6, 0(r10)           ; carga x[k]

                multd f8, f4, f2        ; x[i] * U[k, i]
                addi r11, r11, 1        ; i++
                subd f6, f6, f8         ; x[k] = x[k] - (x[i] * U[k, i])
                sd 0(r10), f6           ; guardar x[k]
```

Estas detenciones RAW se podrían reducir si se usan instrucciones NOP para dar tiempo al procesador a terminar las demás instrucciones, pero solo se sustituirían las detenciones RAW por detenciones estructurales.

En total se producen 10 + 78 = 88 paradas RAW en la ejecución del programa.

CÓDIGO VECTORIAL

El código fuente se encuentra en el apéndice B.

REORGANIZACIÓN DE CÓDIGO:

Estadísticas

Ejecución

437 Ciclos
188 Instrucciones
2.324 Ciclos de reloj por instrucción (CPI)

Configuración hardware

Memoria
20 Kbytes, distribuida en 64 bancos

Direcciones de comienzo por defecto

0x00000100 Código
0x00001000 Datos

Unidades funcionales escalares

Unidad funcional	Segmentación	Latencia
Suma/Resta FP	SI	4
Multiplicación FP	SI	7
División FP	NO	25

Unidades funcionales vectoriales

Unidad funcional	Segmentación	Latencia
Suma/Resta FP	SI	7
Multiplicación FP	SI	8
División FP	NO	21
Carga/Almac. Vectorial	SI	13

Cola de instrucciones vectoriales
5 Longitud

Registros vectoriales
2 Puertos de lectura
1 Puertos de escritura

Características

Adelanto de resultados activado
Tratamiento de los saltos: predicción de no tomar
Encadenamiento vectorial no activado

Saltos

15 Saltos efectivos
10 Saltos no efectivos
25 Saltos totales

Instrucciones de carga/almacenamiento

12 Cargas escalares
12 Cargas vectoriales
4 Almacenamientos escalares
8 Almacenamientos vectoriales

Instrucciones de punto flotante

10 Sumas/Restas
0 Multiplicaciones
0 Divisiones

Instrucciones vectoriales

4 Sumas/Restas de vectores
4 Multiplicaciones de vectores
0 Divisiones de vectores

Traps

1 Traps

Parones

360 RAW (lectura después de escritura)
42 WAW (escritura después de escritura)
0 WAR (escritura después de lectura)
4 Estructurales

Estado de la memoria

	Tamaño	Dirección de comienzo
Código	0192 Bytes	0x00000100
Datos	0196 Bytes	0x00001000

Estadísticas

Ejecución

393 Ciclos
190 Instrucciones
2.068 Ciclos de reloj por instrucción (CPI)

Configuración hardware

Memoria
20 Kbytes, distribuida en 64 bancos

Direcciones de comienzo por defecto

0x00000100 Código
0x00001000 Datos

Unidades funcionales escalares

Unidad funcional	Segmentación	Latencia
Suma/Resta FP	SI	4
Multiplicación FP	SI	7
División FP	NO	25

Unidades funcionales vectoriales

Unidad funcional	Segmentación	Latencia
Suma/Resta FP	SI	7
Multiplicación FP	SI	8
División FP	NO	21
Carga/Almac. Vectorial	SI	13

Cola de instrucciones vectoriales
5 Longitud

Registros vectoriales
2 Puertos de lectura
1 Puertos de escritura

Características

Adelanto de resultados activado
Tratamiento de los saltos: predicción de no tomar
Encadenamiento vectorial no activado

Saltos

16 Saltos efectivos
13 Saltos no efectivos
29 Saltos totales

Instrucciones de carga/almacenamiento

12 Cargas escalares
12 Cargas vectoriales
4 Almacenamientos escalares
8 Almacenamientos vectoriales

Instrucciones de punto flotante

10 Sumas/Restas
0 Multiplicaciones
0 Divisiones

Instrucciones vectoriales

4 Sumas/Restas de vectores
4 Multiplicaciones de vectores
0 Divisiones de vectores

Traps

1 Traps

Parones

236 RAW (lectura después de escritura)
50 WAW (escritura después de escritura)
0 WAR (escritura después de lectura)
10 Estructurales

Estado de la memoria

	Tamaño	Dirección de comienzo
Código	0196 Bytes	0x00000100
Datos	0196 Bytes	0x00001000

```
; Implementacion vectorial eliminando el bucle interno
sub r19, r6, r5          ; Cantidad de elementos a procesar (n - k)
movi2s vlr, r19          ; Cargar vlr con el numero de elementos

mult r13, r17, r6        ; k * n
subi r12, r11, 1         ; i = i - 1
add r13, r13, r12        ; k * n + i
mult r16, r12, r7        ; i * 8 para desplazamiento
mult r13, r13, r7        ; (k * n + i) * 8 para direcciones
add r14, r2, r16         ; Direccion de x[i]
add r15, r1, r13         ; Direccion de U[k,i]

lv v1, 0(r14)            ; Cargar vector x[i]
lv v2, 0(r15)            ; Cargar vector U[k,i]

multv v3, v1, v2         ; Vector v3 = x[i] * U[k,i]
sv 0(r20),v3             ; guardo v3 en direccion r20

add r22,r0,r5            ; inicializo contador k
addi r25,r0,0
movi2fp f8,r25
subd f8,f8,f8
```

```
; Implementacion vectorial eliminando el bucle interno
sub r19, r6, r5          ; Cantidad de elementos a procesar (n - k)
movi2s vlr, r19          ; Cargar vlr con el numero de elementos

lv v4, 0(r10)            ; Cargar x[k] en v4

mult r13, r17, r6        ; k * n
subi r12, r11, 1         ; i = i - 1
add r13, r13, r12        ; k * n + i
mult r16, r12, r7        ; i * 8 para desplazamiento
mult r13, r13, r7        ; (k * n + i) * 8 para direcciones
add r14, r2, r16         ; Direccion de x[i]
add r15, r1, r13         ; Direccion de U[k,i]

lv v1, 0(r14)            ; Cargar vector x[i]
lv v2, 0(r15)            ; Cargar vector U[k,i]

multv v3, v1, v2         ; Vector v3 = x[i] * U[k,i]
add r22,r0,r5            ; inicializo contador k
addi r25,r0,0            ; le doy un 0 a r25
movi2fp f8,r25           ; le paso el 0 a r25
sv 0(r20),v3             ; guardo v3 en direccion r20
subd f8,f8,f8            ; por si acaso, elimino f8
```

```
bucle_suma:
                sne r24,r22,r6
                beqz r24,finsuma

                ld f2, 0(r20)           ; cargo el valor de v3 en f2
                addi r20,r20,8          ; aumento el puntero

                addd f8,f8,f2           ; guardo la suma de ambos

                addi r22,r22,1          ; aumento k
                j bucle_suma
finsuma:

        addi r25,r0,1
        movi2s vlr,r25
        lv v4, 0(r10)           ; Cargar x[k] en v4
        subvs v5,v4,f8         ; x[k] = x[k] - (x[i] * U[k,i])
        sv 0(r10), v5           ; Guardar x[k]

        subi r5, r5, 1          ; k--
        j iniciobucle

findebucle:
        trap 6                  ; Finalizar ejecucion
```

```
bucle_suma:
                sne r24,r22,r6
                beqz r24,finsuma

                ld f2, 0(r20)           ; cargo el valor de v3 en f2
                addi r20,r20,8          ; aumento el puntero
                addi r22,r22,1          ; aumento k
                addd f8,f8,f2           ; guardo la suma de ambos

                j bucle_suma
finsuma:

        addi r25,r0,1
        movi2s vlr,r25          ; cambio el vlr

        subvs v5,v4,f8         ; x[k] = x[k] - (x[i] * U[k,i])
        subi r5, r5, 1          ; k--
        sv 0(r10), v5           ; Guardar x[k]

        j iniciobucle

findebucle:
        trap 6                  ; Finalizar ejecucion
```

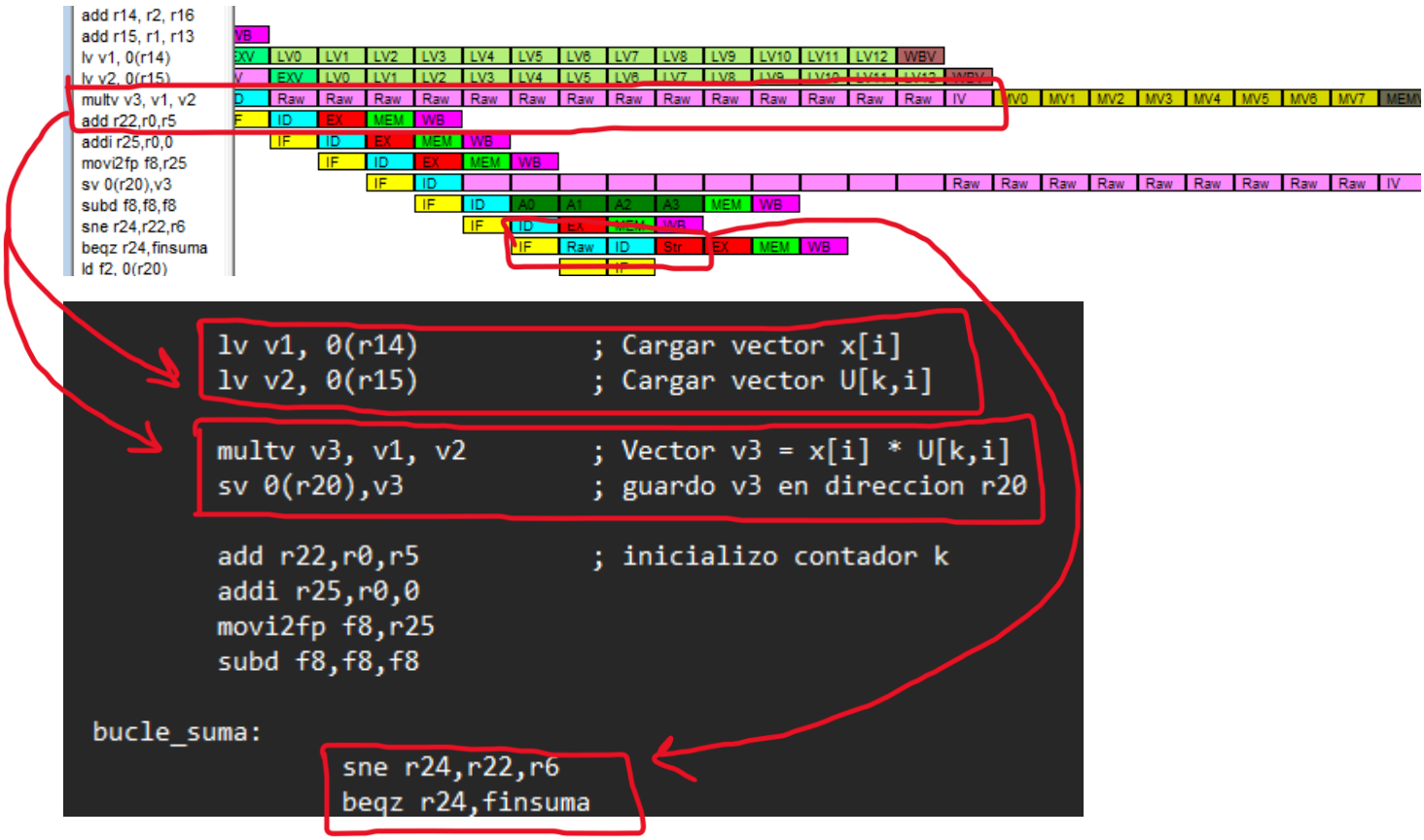
Basándonos en la diferencia de instrucciones, estos pequeños cambios han mejorado el programa:

(437/393) – 1 = 11.19%

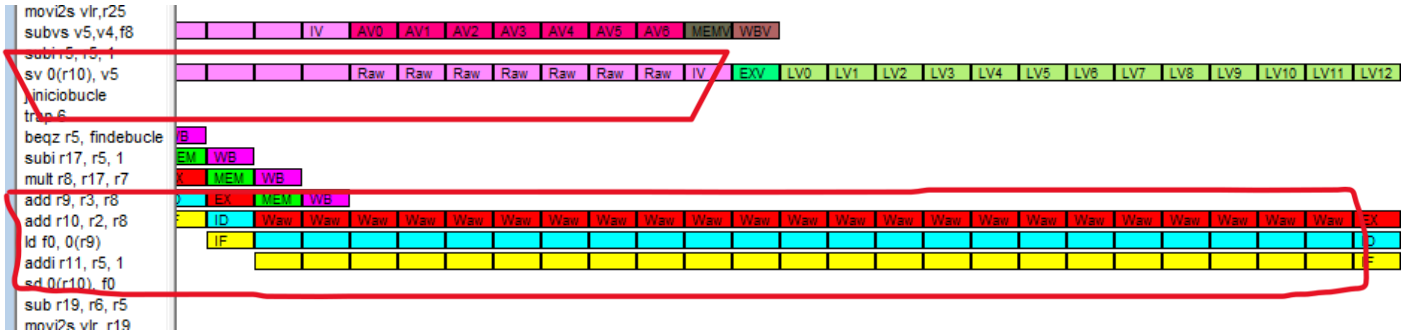
Reduciendo la cantidad de ciclos por instrucción y la cantidad de detenciones.

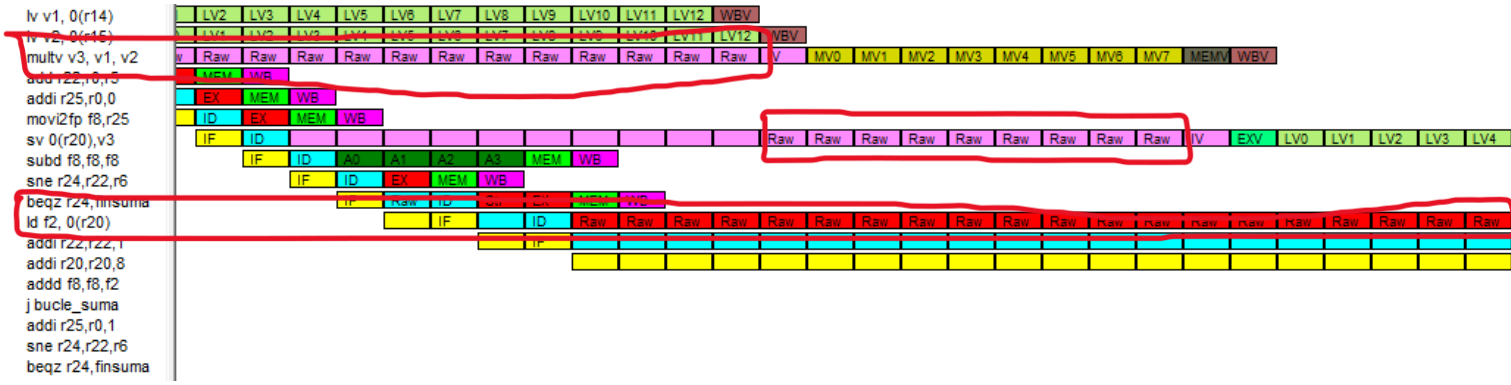
DETENCIONES:

En las operaciones vectoriales se producen múltiples RAW por la cercanía inevitable que hay entre estas. A mayores se pueden observar una detención RAW y una detención estructural al comprobar la condición del bucle sumatorio. La operación multv provoca 14 detenciones RAW, y la operación sv 9, mas la RAW y la estructural de la condición. N = 4, 14 + 9 = 23 * 4 = 92 RAW, mientras que el bucle se realiza, 1 + 2 +3 +4 = 10 x RAW + 10 x Str



Lo mismo ocurre un poco mas adelante en la ejecución, donde el arrastre de las operaciones vectoriales provoca largas esperas por registros. La operación add provoca 24 WAW en su primera interacción, y 13 en cada una de las siguientes, llegando a las 50 WAW marcadas.





Vuelven a aparecer las RAW de operaciones vectoriales, y se presentan detenciones en las operaciones del bucle de suma, RAW, provocadas por el arrastre de las operaciones vectoriales. Esto provoca 27 RAW, 31 RAW en su próxima interacción y otras 35 RAW en su ultima interacción, lo que suma un total de 93 detenciones RAW. En total, 92 + 93 + 10 = 195 RAW + 18 RAW de pequeños accesos a registros esparcidos por el programa + 23 RAW de operaciones sv, suman un total de 236 RAW, de acuerdo con el resultado del programa.

```
addi r25,r0,1
movi2s vlr,r25
lv v4, 0(r10)          ; Cargar x[k] en v4
subvs v5,v4,f8         ; x[k] = x[k] - (x[i] * U[k,i])
sv 0(r10), v5          ; Guardar x[k]

subi r5, r5, 1         ; k--
j iniciobucle
```

```
add r9, r3, r8         ; Direccion de b[k]
add r10, r2, r8        ; Direccion de x[k]
ld f0, 0(r9)           ; Cargar b[k] en f0

addi r11, r5, 1        ; i = k + 1
sd 0(r10), f0          ; x[k] = b[k]
```

ESTADÍSTICAS ESCALAR / VECTORIAL

ESCALAR:

Estadísticas

```
Ejecución
236 Ciclos
159 Instrucciones
1.484 Ciclos de reloj por instrucción (CPI)
Configuración hardware
   Memoria
   20 Kbytes, distribuida en 64 bancos
   Direcciones de comienzo por defecto
   0x00000100  Código
   0x00001000  Datos
   Unidades funcionales escalares
   Unidad funcional        Segmentación    Latencia
   Suma/Resta FP                SI            4
   Multiplicación FP            SI            7
   División FP                  NO            25
   Unidades funcionales vectoriales
   Unidad funcional        Segmentación    Latencia
   Suma/Resta FP                SI            7
   Multiplicación FP            SI            8
   División FP                  NO            21
   Carga/Almac. Vectorial       SI            13
   Cola de instrucciones vectoriales
   5 Longitud
   Registros vectoriales
   2 Puertos de lectura
   1 Puertos de escritura
   Características
   Adelanto de resultados activado
   Tratamiento de los saltos: predicción de no tomar
   Encadenamiento vectorial no activado
Saltos
15 Saltos efectivos
10 Saltos no efectivos
25 Saltos totales
Instrucciones de carga/almacenamiento
24 Cargas escalares
0 Cargas vectoriales
10 Almacenamientos escalares
0 Almacenamientos vectoriales
Instrucciones de punto flotante
6 Sumas/Restas
6 Multiplicaciones
0 Divisiones
Instrucciones vectoriales
0 Sumas/Restas de vectores
0 Multiplicaciones de vectores
0 Divisiones de vectores
Traps
1 Traps
Parones
88 RAW (lectura después de escritura)
0 WAW (escritura después de escritura)
0 WAR (escritura después de lectura)
0 Estructurales
Estado de la memoria
                    Tamaño        Dirección de comienzo
   Código          0140 Bytes          0x00000100
   Datos           0196 Bytes          0x00001000
```

Se producen 236 ciclos y 159 instrucciones. Se nos muestra que son 1.484 CPI, que se pueden obtener dividiendo ambas cantidades:

236 / 159 = 1.48427 CPI

VECTORIAL:

Estadísticas

```
Ejecución

390 Ciclos
188 Instrucciones
2.074 Ciclos de reloj por instrucción (CPI)

Configuración hardware

   Memoria
   20 Kbytes, distribuida en 64 bancos

   Direcciones de comienzo por defecto

   0x00000100  Código
   0x00001000  Datos

   Unidades funcionales escalares

   Unidad funcional       Segmentación    Latencia
   Suma/Resta FP               SI            4
   Multiplicación FP           SI            7
   División FP                 NO            25

   Unidades funcionales vectoriales

   Unidad funcional       Segmentación    Latencia
   Suma/Resta FP               SI            7
   Multiplicación FP           SI            8
   División FP                 NO            21
   Carga/Almac. Vectorial      SI            13

   Cola de instrucciones vectoriales
   5 Longitud

   Registros vectoriales
   2 Puertos de lectura
   1 Puertos de escritura

   Características

   Adelanto de resultados activado
   Tratamiento de los saltos: predicción de no tomar
   Encadenamiento vectorial no activado

Saltos

15 Saltos efectivos
10 Saltos no efectivos
25 Saltos totales

Instrucciones de carga/almacenamiento

12 Cargas escalares
12 Cargas vectoriales
4 Almacenamientos escalares
8 Almacenamientos vectoriales

Instrucciones de punto flotante

10 Sumas/Restas
0 Multiplicaciones
0 Divisiones

Instrucciones vectoriales

4 Sumas/Restas de vectores
4 Multiplicaciones de vectores
0 Divisiones de vectores

Traps

1 Traps

Parones

237 RAW (lectura después de escritura)
49 WAW (escritura después de escritura)
0 WAR (escritura después de lectura)
10 Estructurales

Estado de la memoria

                    Tamaño        Dirección de comienzo

   Código          0192 Bytes          0x00000100
   Datos           0196 Bytes          0x00001000
```

Se producen 393 ciclos y 190 instrucciones. Se nos muestra que son 2.068 CPI, que se pueden obtener dividiendo ambas cantidades:

393 / 190 = 2.06842 CPI

Parámetros	*Escalar*	*Vectorial*
Ciclos	236	393
Instrucciones	159	190
CPI	1.48427	2.06842

En este caso, la CPI es mayor en la vectorial, lo que indica que se tarda más en ejecutar cada instrucción.

PRUEBAS CON N:

VECTORIAL:

N = 4

Estadísticas

Ejecución

393 Ciclos
190 Instrucciones
2.068 Ciclos de reloj por instrucción (CPI)

La solución general: $X=\begin{pmatrix} 7 \\ -27 \\ -9 \\ 6 \end{pmatrix}$

0x00001080	0000007.00000000	-000027.00000000	-000009.00000000	0000006.00000000

Editar...

F:\Program Files (x86)\WinDLXV\programas\BACK_SUBSTITUTION_VEC.s

```
;*******************************************************
;**********************PEC_IC_II**********************
;*******************************************************

                .data

        .align 3                ; Alinear para accesos mas rapidos
U:      .double 1, 2, -2, 4
        .double 0, 1, -5, -3
        .double 0, 0, 1, 1
        .double 0, 0, 0, 1      ; Matriz U (triangular superior)

x:      .space 32               ; Vector x (4*8 bytes = 32)

b:      .double -5, 0, -3, 6    ; Vector b

n:      .word 4                 ; Tamaño del sistema de ecuaciones
```

N = 5

Estadísticas

Ejecución

526 Ciclos
253 Instrucciones
2.079 Ciclos de reloj por instrucción (CPI)

La solución general: $X=\begin{pmatrix} 378 \\ -188 \\ -23 \\ -22 \\ 7 \end{pmatrix}$

0x000010c0	0000001.00000000	0000378.00000000	-000188.00000000	-000023.00000000
0x000010e0	-000022.00000000	0000007.00000000	-000005.00000000	0000000.00000000

Editar...

F:\Program Files (x86)\WinDLXV\programas\BACK_SUBSTITUTION_VEC.s

```
;*****************************************************
;**********************PEC_IC_II**********************
;*****************************************************

                .data


U:      .double 1, 2, -2, 4, 5
        .double 0, 1, -5, -3, 1
        .double 0, 0, 1, 1, 6
        .double 0, 0, 0, 1, 4      ; Matriz U (triangular superior)
        .double 0, 0, 0, 0, 1

x:      .space 40               ; Vector x (4*8 bytes = 32)

b:      .double -5, 0, -3, 6, 7    ; Vector b

n:      .word 5                 ; Tamaño del sistema de ecuaciones
```

N = 3

Estadísticas

Ejecución

274 Ciclos
134 Instrucciones
2.045 Ciclos de reloj por instrucción (CPI)

La solución general: $X=\begin{pmatrix} 19 \\ -15 \\ -3 \end{pmatrix}$

0x00001040	0000001.00000000	0000019.00000000	-000015.00000000	-000003.00000000

Editar...

F:\Program Files (x86)\WinDLXV\programas\BACK_SUBSTITUTION_VEC.s

```
;*****************************************************
;**********************PEC_IC_II**********************
;*****************************************************

                .data


U:      .double 1, 2, -2
        .double 0, 1, -5
        .double 0, 0, 1

x:      .space 24               ; Vector x (4*8 bytes = 32)

b:      .double -5, 0, -3       ; Vector b

n:      .word 3                 ; Tamaño del sistema de ecuaciones
```

	CPI	Ciclos	Instrucciones
3	2.045	274	134
4	2.068	393	190
5	2.079	526	253

Se puede apreciar que las diferencias entre los ciclos, tomando distintas n, no son constantes, por lo que el crecimiento de la ecuación no puede ser lineal, de manera que, por descarte, seguramente sea cuadrático. Resultando en ecuaciones del tipo:

Ciclos (n) = $a * n^2 + b * n + c$

Instrucciones (n) = $d * n^2 + e * n + f$

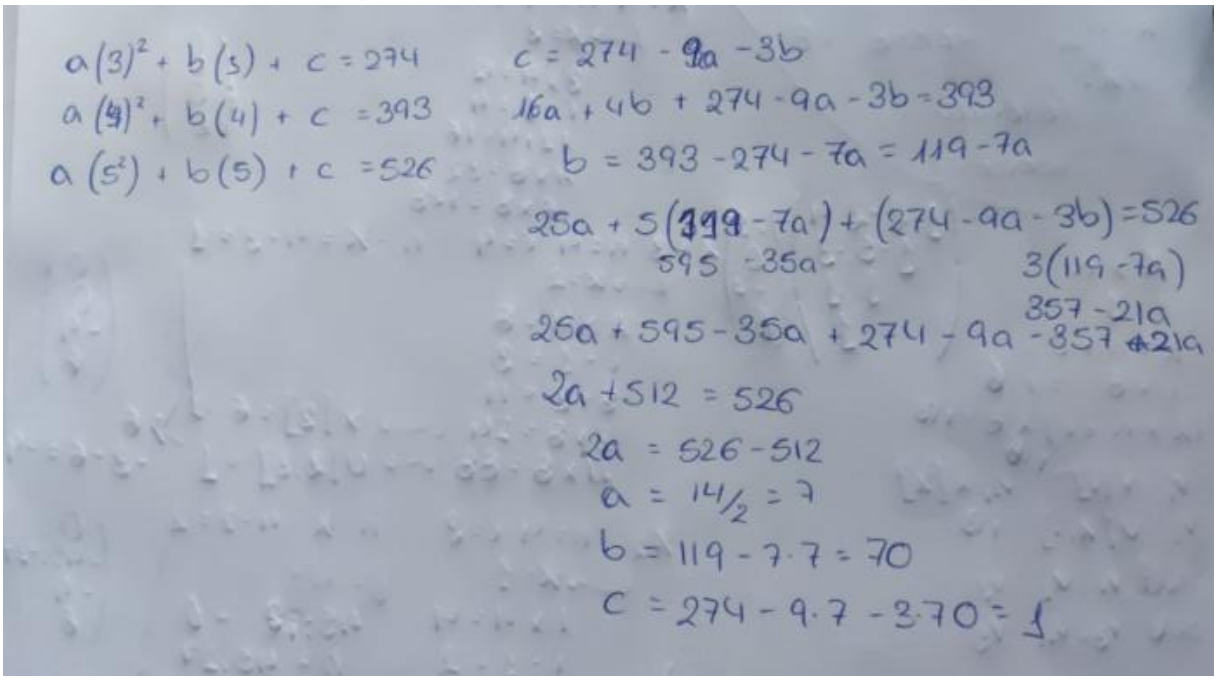
Para solucionar este sistema, simplemente le aportamos valores a n teniendo en cuenta los valores que ya conocemos:

$$a * 3^2 + b * 3 + c = 274$$

$$a * 4^2 + b * 4 + c = 393$$

$$a * 5^2 + b * 5 + c = 526$$

Realizando los cálculos en una hoja aparte obtenemos: $\mathrm{Ciclos(n)} = 7\mathrm{n}^2 + 70\mathrm{n} + 1$



Siguiendo los mismos pasos para las instrucciones:

$$d * 3^2 + e * 3 + f = 134$$

$$d * 4^2 + e * 4 + f = 190$$

$$d * 5^2 + e * 5 + f = 253$$

Obtenemos: Instrucciones(n)=3.5n^2+31.5n+8

$a(3)^2 + b(3) + c = 134$ $\quad c = 134 - 9a - 3b$

$a(4)^2 + b(4) + c = 190$ $\quad b = 190 - 134 - 7a = 56 - 7a$

$a(5)^2 + b(5) + c = 253$ $\quad 25a + 280 - 35a + 134 - 9a - 168 + 21a$

$2a = 253 - 246$

$a = 7/2 = 3'5$

$b = 31.5$

$c = 8$

De modo que, teniendo en cuenta que el CPI se define como:

$$CPI(n) = \frac{Ciclos(n)}{Instrucciones(n)}$$

Sustituyendo obtenemos:

$$\lim_{n \to \infty} CPI(n) = \frac{7\mathrm{n}^2 + 70\mathrm{n} + 1}{3.5\mathrm{n}^2 + 31.5\mathrm{n} + 8}$$

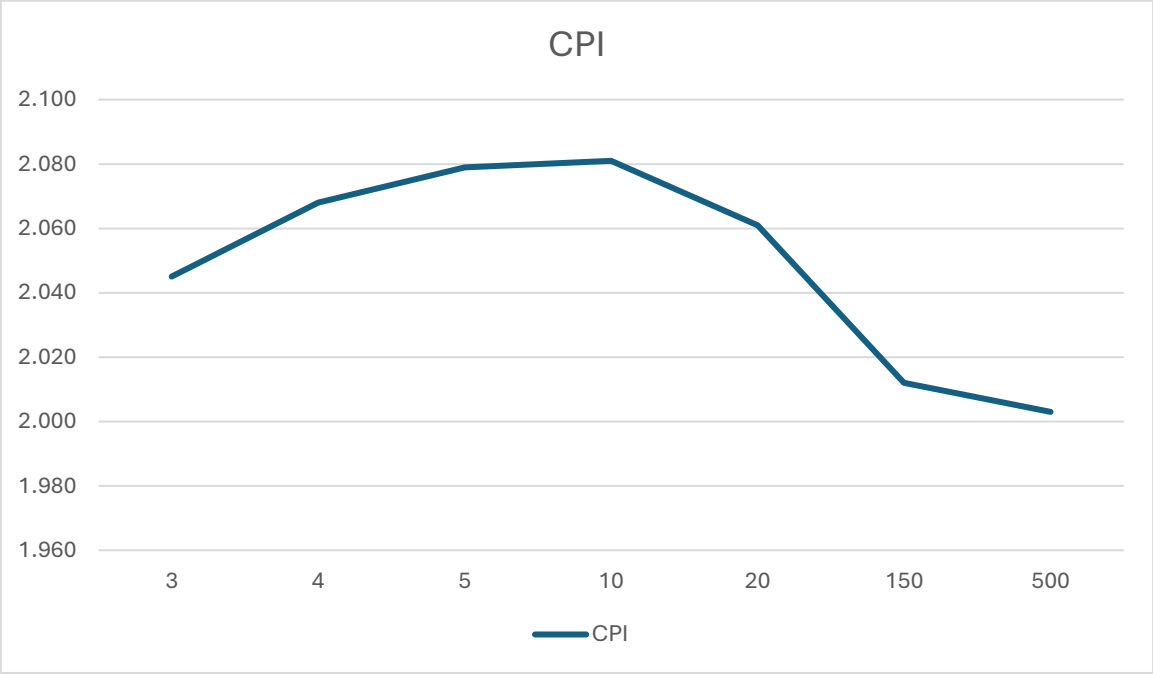
Como es una ecuación cuadrática, podemos despreciar los demás elementos distintos de n^2, quedándonos con:

$$\frac{7}{3.5} = 2$$

De esta forma, se comprueba que el CPI tenderá a 2 cuando $n \to \infty$. Insinuando que la diferencia entra las instrucciones y los ciclos será cada vez menor, pues que los casos de los que partimos, el CPI ya es mayor que 2 de por sí. Aportando una serie de valores aleatorios a N:

	CPI	Ciclos	Instrucciones
3	2.045	274	134
4	2.068	393	190
5	2.079	526	253
10	2.081	1401	673
20	2.061	4201	2038
150	2.012	168001	83483
500	2.003	1785001	890758

Cogiendo los valores de N y CPI para representarlos:



- Si en la gráfica se mostrasen los ciclos y las instrucciones, debido a sus altos valores, el valor de CPI apenas sería visible.

ESCALAR:

N = 4

Estadísticas

Ejecución

236 Ciclos
159 Instrucciones
1.484 Ciclos de reloj por instrucción (CPI)

La solución general: $X=\begin{pmatrix} 7 \\ -27 \\ -9 \\ 6 \end{pmatrix}$

0x00001080	0000007.00000000	-000027.00000000	-000009.00000000	0000006.00000000

Editar...

F:\Program Files (x86)\WinDLXV\programas\BACK_SUBSTITUTION_ESC.s

```
;*******************************************************
;**********************PEC_IC_II*********************
;*******************************************************

                .data

                .align 3                ; Alinear par
U:              .double 1,2, -2, 4      ; Filas matri
                .double 0,1, -5, -3
                .double 0,0, 1, 1
                .double 0,0, 0, 1

x:              .space  32              ; Fila matriz

b:              .double -5,0,-3,6       ; Fila matriz

n:              .word 4         ; Tamaño problema
```

N = 5

Estadísticas

Ejecución

359 Ciclos
239 Instrucciones
1.502 Ciclos de reloj por instrucción (CPI)

La solución general: $X=\begin{pmatrix} 378 \\ -188 \\ -23 \\ -22 \\ 7 \end{pmatrix}$

0x000010c0	0000001.00000000	0000378.00000000	-000188.00000000	-000023.00000000
0x000010e0	-000022.00000000	0000007.00000000	-000005.00000000	0000000.00000000

Editar...

F:\Program Files (x86)\WinDLXV\programas\BACK_SUBSTITUTION_ESC.s

```
;*****************************************************
;**********************PEC_IC_II**********************
;*****************************************************

                .data

                .align 3                ; Alinear par
U:              .double 1,2, -2, 4, 5   ; Filas matri
                .double 0,1, -5, -3, 1
                .double 0,0, 1, 1, 6
                .double 0,0, 0, 1, 4
                .double 0,0, 0, 0, 1

x:              .space  40              ; Fila matriz

b:              .double -5,0,-3,6, 7    ; Fila matriz

n:              .word 5         ; Tamaño problema
```

N = 3

Estadísticas

Ejecución

140 Ciclos
96 Instrucciones
1.458 Ciclos de reloj por instrucción (CPI)

La solución general: $X=\begin{pmatrix} 19 \\ -15 \\ -3 \end{pmatrix}$

0x00001040	0000001.00000000	0000019.00000000	-000015.00000000	-000003.00000000

Editar...

F:\Program Files (x86)\WinDLXV\programas\BACK_SUBSTITUTION_ESC.s

```
;*****************************************************
;**********************PEC_IC_II**********************
;*****************************************************

                .data

                .align 3                ; Alinear para a
U:              .double 1,2, -2         ; Filas matriz U
                .double 0,1, -5
                .double 0,0, 1

x:              .space  24              ; Fila matriz X

b:              .double -5,0,-3 ; Fila matriz b

n:              .word 3         ; Tamaño problema
```

Valor de N	CPI	Ciclos	Instrucciones
3	1.458	140	96
4	1.484	236	159
5	1.502	359	239

Se puede apreciar que las diferencias entre los ciclos, al igual que en el código vectorial, tomando distintas n, no son constantes, por lo que el crecimiento de la ecuación no puede ser lineal, de manera que, será cuadrático. Resultando en ecuaciones del tipo:

Ciclos (n) = $a * n^2 + b * n + c$

Instrucciones (n) = $d * n^2 + e * n + f$

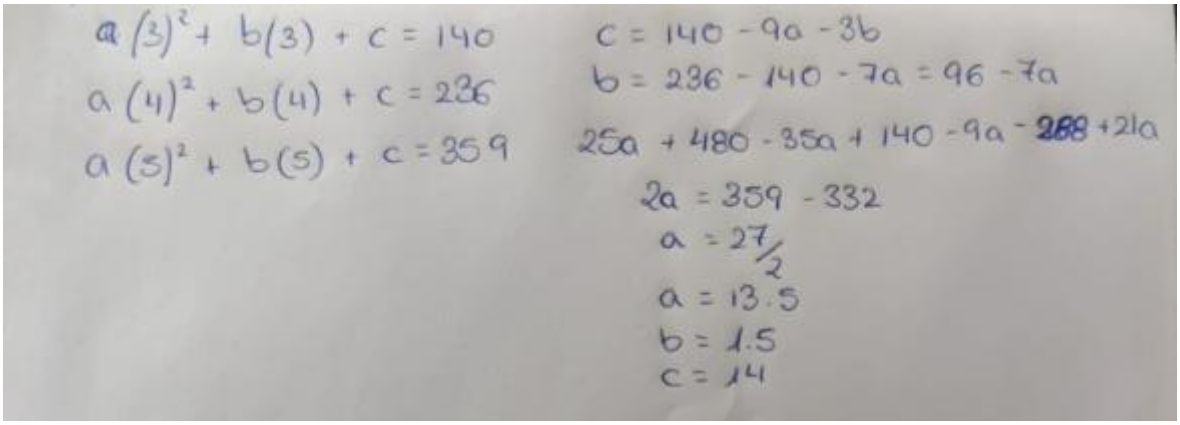
Para solucionar este sistema, al igual que antes, simplemente le aportamos valores a n teniendo en cuenta los valores que ya conocemos:

$$a * 3^2 + b * 3 + c = 140$$

$$a * 4^2 + b * 4 + c = 236$$

$$a * 5^2 + b * 5 + c = 359$$

Realizando los cálculos en una hoja aparte obtenemos: $\mathrm{Ciclos(n)} = 13.5\mathrm{n}^2 + 1.5\mathrm{n} + 14$



Siguiendo los mismos pasos para las instrucciones:

$$d * 3^2 + e * 3 + f = 96$$

$$d * 4^2 + e * 4 + f = 159$$

$$d * 5^2 + e * 5 + f = 239$$

Obtenemos: Instrucciones(n)=8.5n^2+3.5n+9

$$a(3)^2 + b(3) + c = 96$$
$$a(4)^2 + b(4) + c = 159$$
$$a(5)^2 + b(5) + c = 239$$

$$c = 96 - 9a - 3b$$
$$b = 159 - 96 - 7a = 63 - 7a$$
$$25a + 315 - 35a + 96 - 9a - 189 + 21a$$
$$2a = 239 - 222$$
$$a = 17/2$$
$$a = 8.5$$
$$b = 3.5$$
$$c = 9$$

De modo que, teniendo en cuenta que el CPI se define como:

$$CPI(n) = \frac{Ciclos(n)}{Instrucciones(n)}$$

Sustituyendo obtenemos:

$$\lim_{n \to \infty} CPI(n) = \frac{13.5\mathrm{n}^2 + 1.5\mathrm{n} + 14}{8.5\mathrm{n}^2 + 3.5\mathrm{n} + 9}$$

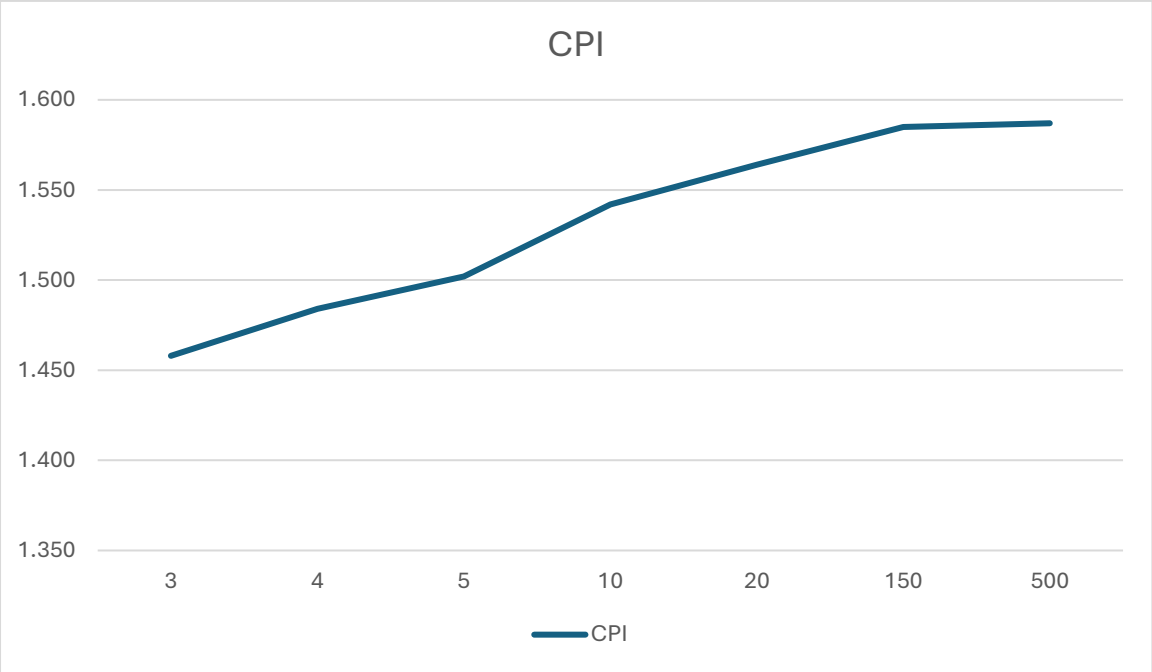
Como es una ecuación cuadrática, podemos despreciar los demás elementos distintos de n^2, quedándonos con:

$$\frac{13.5}{8.5} = 1.5882 \ldots$$

De esta forma, se comprueba que el CPI tenderá a 1.5882 cuando $n \to \infty$. Insinuando que la diferencia entra las instrucciones y los ciclos aumentará un poco conforme aumente n, pero se terminará estabilizando, ya que partimos, en valores bajos de n, de un CPI cercano a 1.5. Aportando una serie de valores aleatorios a N:

	CPI	Ciclos	Instrucciones
3	1.458	140	96
4	1.484	236	159
5	1.502	359	239
10	1.5425	1379	894
20	1.5648	5444	3479
150	1.5850	303989	191784
500	1.5872	3375764	2126759

Cogiendo los valores de N y CPI para representarlos:



Teniendo ambos resultados, del código vectorial y del código escalar, y comparándolos, se puede decir que **es más óptimo el código vectorial.** A pesar de que, en valores bajos de N, el código escalar emplea muchas menos instrucciones y ciclos, cuanto más aumenta N la cantidad de ciclos se dispara. Podemos ver cómo alcanza y supera los 3 millones de ciclos con N = 500, mientras que, con el código vectorial, a pesar de que emplea más ciclos e instrucciones con valores bajos de N, conforme este aumenta, no se llega a disparar tanto, alcanzando 1 millón 700 mil ciclos con N = 500, "casi" la mitad que en el caso escalar. Para ser exactos un 52.87% menos de ciclos. Ocurre lo mismo con las instrucciones.

APÉNDICE A:

```
;****************************************************
;**********************PEC_IC_II**********************
;****************************************************
                .data
                .align 3                ; Alinear para accesos más rápidos
U:              .double 1,2, -2, 4      ; Filas matriz U
                .double 0,1, -5, -3
                .double 0,0, 1, 1
                .double 0,0, 0, 1
x:              .space   32             ; Fila matriz X (4*8 bytes = 32)
b:              .double -5,0,-3,6       ; Fila matriz b
n:              .word 4                 ; Tamaño problema


                .text
                .global main
main:
                addi r1, r0, U      ; puntero de U
                addi r2, r0, x      ; puntero de x
                addi r3, r0, b      ; puntero de b
                addi r4, r0, n      ; puntero de n
                lw r5, 0(r4)        ; guardo valor de k = n
                lw r6, 0(r4)        ; cargo n en r6
                addi r7, r0, 8      ; constante 8 para multiplicaciones
iniciobucle:
                BEQZ r5, findebucle     ; si k = 0, salto al final
                subi r17, r5, 1         ; k-1 porque bits van de 0 a 31
                mult r8, r17, r7    ; k * 8 para desplazamiento
                add r9, r3, r8          ; puntero b[k]
                add r10, r2, r8         ; puntero x[k]
                ld f0, 0(r9)            ; carga b[k] en f0
                addi r11, r5, 1     ; i = k + 1
                sd 0(r10), f0           ; x[k] = b[k]
```

```
iniciobuclei:
        SLE r12, r11, r6   ; si i <= n, continuar
        BEQZ r12, findebuclei
        subi r18, r11, 1   ; i-1 porque bits van de 0 a 31
        mult r13, r17, r6 ; k * n
        mult r15, r18, r7 ; i * 8
        add r13, r13, r18; k * n + i
        add r16, r2, r15   ; puntero x[i]
        mult r13, r13, r7 ; (k * n + i) * 8
        ld f4, 0(r16)              ; carga x[i]
        add r14, r1, r13   ; puntero U[k, i]
        ld f2, 0(r14)              ; carga U[k, i]
        ld f6, 0(r10)              ; carga x[k]
        multd f8, f4, f2   ; x[i] * U[k, i]
        addi r11, r11, 1   ; i++
        subd f6, f6, f8            ; x[k] = x[k] - (x[i] * U[k, i])
        sd 0(r10), f6              ; guardar x[k]
        J iniciobuclei
findebuclei:
        subi r5, r5, 1             ; k--
        J iniciobucle
findebucle:
        trap 6                     ; fin de programa
```

APÉNDICE B:

```
;****************************************************
;**********************PEC_IC_II**********************
;****************************************************
                    .data
                    .align 3            ; Alinear para accesos mas rapidos
U:    .double 1, 2, -2, 4               ; Filas matriz U
      .double 0, 1, -5, -3
      .double 0, 0, 1, 1
      .double 0, 0, 0, 1

x:    .space 32                         ; Fila matriz X (4*8 bytes = 32)
b:    .double -5, 0, -3, 6              ; Fila matriz b
n:    .word 4                           ; Tamaño problema

      .text
      .global main
main:
      addi r1, r0, U                    ; Puntero a  U
      addi r2, r0, x                    ; Puntero a x
      addi r3, r0, b                    ; Puntero a b
      addi r4, r0, n                    ; Puntero a n
      lw r5, 0(r4)                      ; guardo valor de k = n
      lw r6, 0(r4)                      ; cargo n en r6
      addi r7, r0, 8                    ; constante 8 para multiplicaciones
      mult r18, r7, r6                  ; 4 * 8 para desplazamientos (vlr = 4)
iniciobucle:
      beqz r5, findebucle               ; si k = 0, salto al final
      subi r17, r5, 1                   ; k - 1
      mult r8, r17, r7                  ; Desplazamiento de k * 8 bytes
      add r9, r3, r8                    ; puntero b[k]
      add r10, r2, r8                   ; puntero x[k]
      ld f0, 0(r9)                      ; Cargar b[k] en f0

      addi r11, r5, 1                   ; i = k + 1
      sd 0(r10), f0                     ; x[k] = b[k]
```

```
; Implementacion vectorial eliminando el bucle interno
sub r19, r6, r5          ; Cantidad de elementos a procesar (n - k)
movi2s vlr, r19          ; Cargar vlr con el numero de elementos
lv v4, 0(r10)            ; Cargar x[k] en v4
mult r13, r17, r6        ; k * n
subi r12, r11, 1         ; i = i - 1
add r13, r13, r12        ; k * n + i
mult r16, r12, r7        ; i * 8 para desplazamiento
mult r13, r13, r7        ; (k * n + i) * 8 para direcciones
add r14, r2, r16         ; Direccion de x[i]
add r15, r1, r13         ; Direccion de U[k,i]
lv v1, 0(r14)            ; Cargar vector x[i]
lv v2, 0(r15)            ; Cargar vector U[k,i]
multv v3, v1, v2         ; Vector v3 = x[i] * U[k,i]
add r22,r0,r5            ; inicializo contador k
addi r25,r0,0            ; le doy un 0 a r25
movi2fp f8,r25           ; le paso el 0 a r25
sv 0(r20),v3             ; guardo v3 en direccion r20
subd f8,f8,f8            ; por si acaso, elimino f8

bucle_suma:
        sne r24,r22,r6
        beqz r24,finsuma
        ld f2, 0(r20)        ; cargo el valor de v3 en f2
        addi r20,r20,8       ; aumento el puntero
        addi r22,r22,1       ; aumento k
        addd f8,f8,f2        ; guardo la suma de ambos

        j bucle_suma
finsuma:
    beqz r19,saltar
    addi r25,r0,1
    movi2s vlr,r25       ; cambio el vlr
saltar:
    subvs v5,v4,f8   ; x[k] = x[k] - (x[i] * U[k,i])
```

```
        subi r5, r5, 1      ; k--
        sv 0(r10), v5       ; Guardar x[k]
        j iniciobucle
findebucle:
    trap 6              ; Finalizar ejecucion
```

CONCLUSIÓN:

Me da pena haberlo hecho post-examen, por recomendación, porque he aprendido bastantes cosas sobre ensamblador y procesadores que no tenía muy claras de cara al examen. Siempre es gratificante enfrentarse a un problema tan complejo y conseguir sacarlo por cuenta propia tras muchas horas entre líneas de código, a pesar de que el código vectorial no tuve mucho tiempo para profundizar en él y sé que con otro enfoque quizás hubiese tenido un mejor rendimiento.

El único problema, considero que está en la introducción al simulador y al “lenguaje”, estaría bien algún video corto sobre como empezar a programar en ensamblador con conceptos básicos pero que permitan salir del apuro, punteros, memoria, índices...