INGENIERÍA DE COMPUTADORES II

- PRUEBA DE EVALUACIÓN A DISTANCIA -

Alumno: Misrraim Suárez Pérez (44721968D)
E-mail: misrraimsp@gmail.com
Centro Asociado: Las Palmas de Gran Canaria

Cuestión a)

El código correspondiente a este apartado se encuentra adjunto en el Apéndice A. Se ha usado la directiva *.align 2* para garantizar que lo datos definidos sean colocados en memoria empezando siempre por una dirección múltiplo de 4 (2^2).
El tamaño de los vectores se fija estáticamente por simplicidad. El valor fijado es 200 elementos.

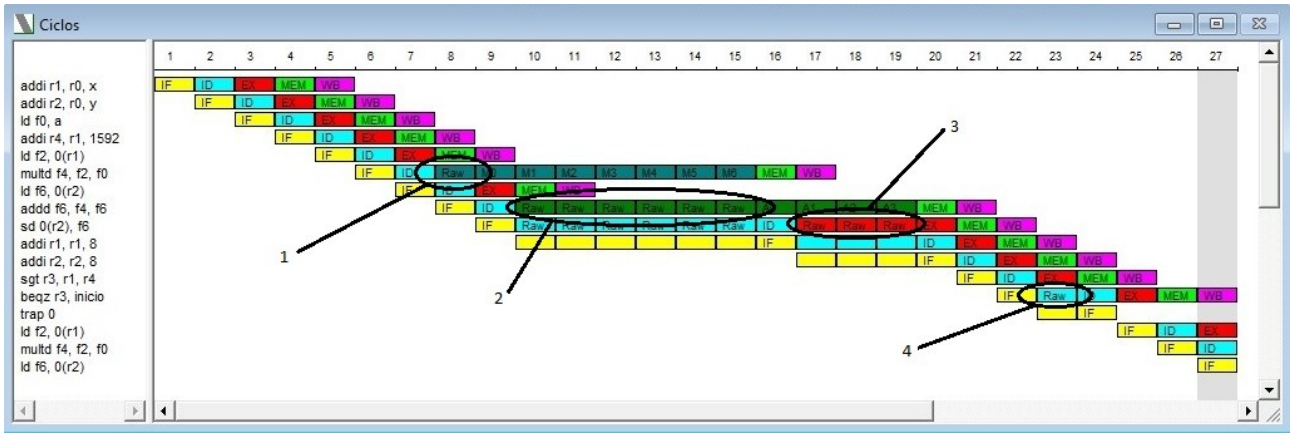

Figura a.1

Se ha detectado **3400** detenciones a causa de dependencias RAW. A continuación se pasa a explicar sus causas en detalle, con apoyo de la Figura a.1:

(1) `multd f4, f2, f0` espera un ciclo por el operando `f2`, que está siendo cargado desde memoria por `ld f2, 0(r1)`.

(2) `addd f6, f4, f6` espera seis ciclos por el operando `f4`, que está siendo calculado en la unidad de multiplicación en coma flotante por `multd f4, f2, f0`.

(3) `sd 0(r2), f6` espera tres ciclos por el operando `f6`, que está siendo calculado en la unidad de suma en coma flotante por `addd f6, f4, f6`.

(4) `beqz r3, inicio` espera un ciclo por el operando `r3`, que está siendo calculado en la unidad entera por `sgt r3, r1, r4`.

Cuestión b)

El código correspondiente a este apartado se encuentra adjunto en el Apéndice B.

El tamaño de los vectores se fija estáticamente por simplicidad. El valor fijado es 200 elementos.

Se ha usado la directiva *.align 2* para garantizar que lo datos definidos sean colocados en memoria empezando siempre por una dirección múltiplo de 4 (2^2).

Se ha desenrollado el bucle del apartado a) tres veces. Además, si el tamaño de los vectores no fuese múltiplo de tres, se gestionaría la cantidad sobrante por separado.

Se puede dividir el código funcionalmente en tres partes:

i. Bajo la etiqueta `main` se encuentra la parte de inicialización de valores constantes necesarios y la lógica para determinar si el numero de elementos es múltiplo de tres.

ii. Bajo la etiqueta `thinloop` está el código para manejar las componentes restantes que quedan fuera del bucle desenrollado. Se trata de un bucle que itera decrementando el valor `r9` (este registro se inicializa con el valor 200 mod 3) hasta que se haga nulo.

iii. Bajo la etiqueta `thickloop` está el código para manejar el bucle desenrollado.

Se ha detectado **233** detenciones a causa de dependencias RAW, y **132** debido a causas estructurales. A continuación se pasan a explicar en detalle, con apoyo de las figuras b.1 y b.2:

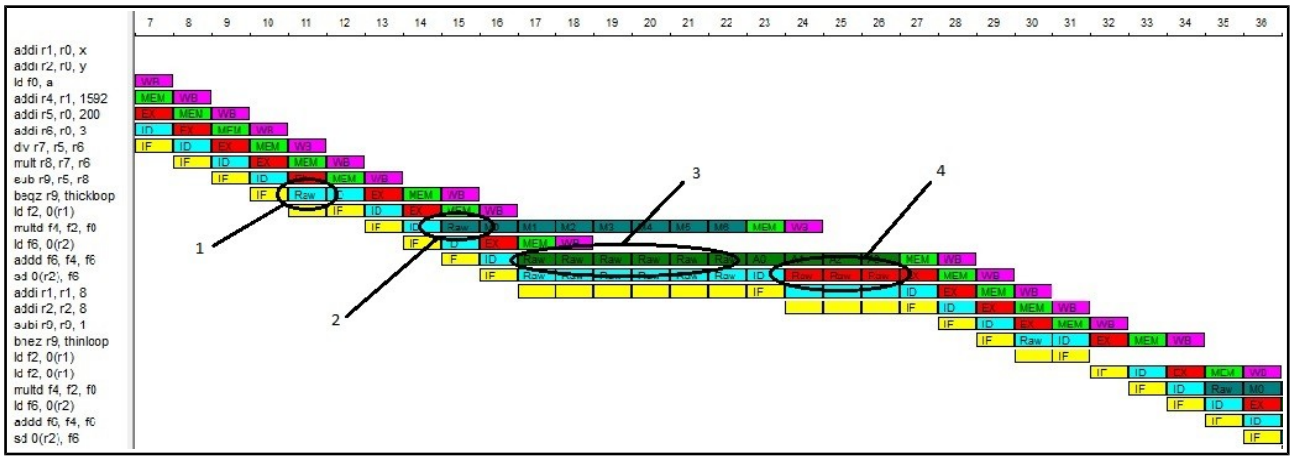

Figura b.1

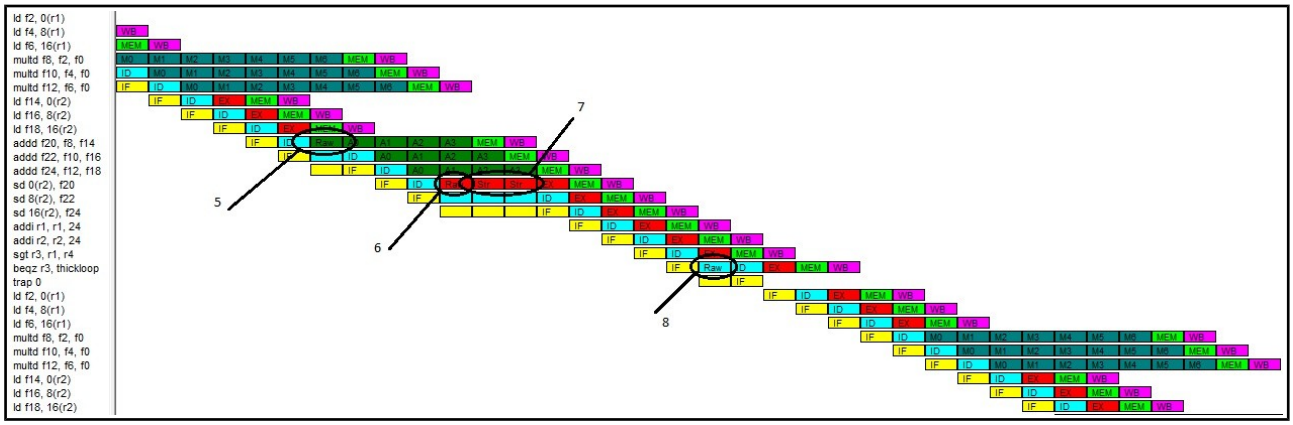

Figura b.2

(1) `beqz r9, thickloop` espera un ciclo por el operando `r9`, que está siendo calculado en la unidad entera por `sub r9, r5, r8`.

(2) `multd f4, f2, f0` espera un ciclo por el operando `f2`, que está siendo cargado desde memoria por `ld f2, 0(r1)`.

(3) `addd f6, f4, f6` espera seis ciclos por el operando `f4`, que está siendo calculado en la unidad de multiplicación en coma flotante por `multd f4, f2, f0`.

(4) `sd 0(r2), f6` espera tres ciclos por el operando `f6`, que está siendo calculado en la unidad de suma en coma flotante por `addd f6, f4, f6`.

(5) `addd f20, f8, f14` espera un ciclo por el operando `f8`, que está siendo calculado en la unidad de multiplicación en coma flotante por `multd f8, f2, f0`.

(6) `sd 0(r2), f20` espera un ciclo por el operando `f20`, que está siendo calculado en la unidad de suma en coma flotante por `addd f20, f8, f14`.

(7) `sd 0(r2), f20` espera dos ciclos para acceder a la etapa MEM (detención estructural), hasta que deja de ser usada por `addd f24, f12 f18`.

(8) `beqz r3, thickloop` *espera un ciclo por el operando* `r9`*, que está siendo calculado en la unidad entera por* `sgt r3, r1, r4`*.*

Cuestión c)

El código correspondiente a este apartado se encuentra adjunto en el Apéndice C.

El tamaño de los vectores se fija estáticamente por simplicidad. El valor fijado es 200 elementos.

Se ha usado la directiva *.align 2* para garantizar que lo datos definidos sean colocados en memoria empezando siempre por una dirección múltiplo de 4 (2^2).

Se ha vectorizado el código del apartado a) usando la técnica de seccionamiento de bucles, teniendo en cuenta que MVL (maximum vector length) = 64.

Se puede dividir el código funcionalmente en 5 partes (ver Figura c.1):

i. Bajo la etiqueta `main` se encuentra la parte de inicialización de valores constantes necesarios y la lógica para determinar si el numero de elementos es múltiplo de MVL. En caso afirmativo se salta a `loopsetup`, si no se continúa hacia `prologue`.

ii. Bajo la etiqueta `prologue` está el código para manejar vectorialmente las componentes que quedan fuera de los grupos de MVL elementos. Al final de esta etapa se comprueba si el número de elementos del vector es menor que MVL. En caso afirmativo se salta a `end`, si no se continúa hacia `loopsetup`.

iii. Bajo la etiqueta `loopsetup` está el código para configurar el bucle vectorial principal, que sólo debe ser ejecutado una vez.

iv. Bajo la etiqueta `loop` está el código para el cálculo vectorial.

v. Bajo la etiqueta `end` se termina la ejecución.

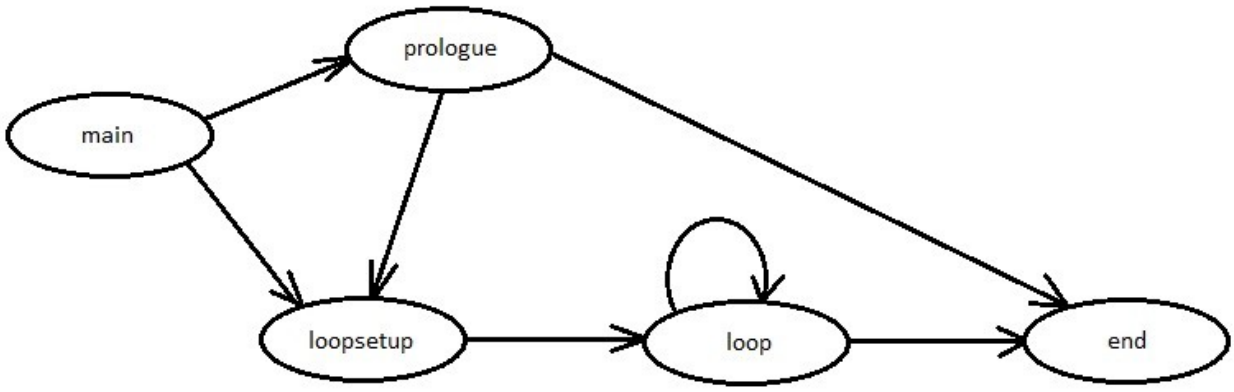

Figura c.1

Se ha detectado **1238** detenciones a causa de dependencias RAW, y **468** debido a dependencias WAW. A continuación se pasan a explicar en detalle, con apoyo de la figura c.2:

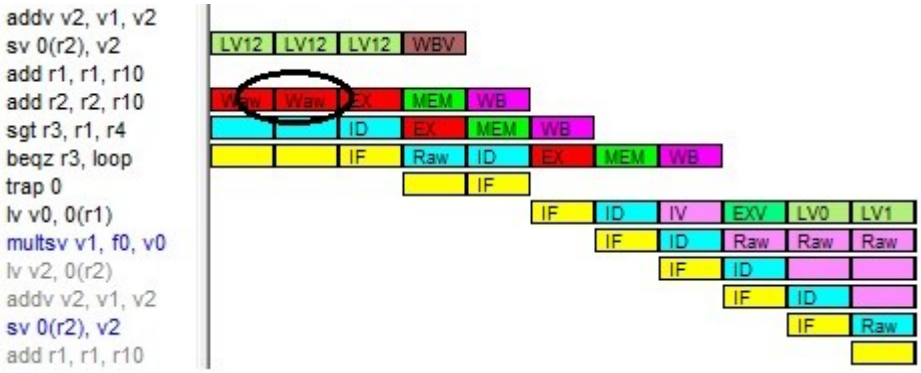

Figura c.2

(1) Las dependencias RAW no presentan novedad en su origen respecto a las ya explicadas para los apartados anteriores. La única diferencia es en las instrucciones que la generan.

(2) Las dependencias WAW aparecen por primera vez en este apartado. `add r2, r2, r10` intenta escribir en `r2`, pero no puede porque está siendo usado por la instrucción de almacenamiento vectorial `sv 0(r2), v2`.

Cuestión d)

Se ha obtenido, para vectores de 200 elementos, el siguiente consumo en ciclos de reloj:

(1) código escalar del Apéndice A: **4005 ciclos**
(2) código escalar del Apéndice B: **1701 ciclos**
(3) código vectorial del Apéndice C, sin encadenamiento vectorial: **1026 ciclos**
(4) código vectorial del Apéndice C, con encadenamiento vectorial: **777 ciclos**

Cuestión e)

El CPI medio se obtiene del ratio entre el número de ciclos y el número de instrucciones. Para los mismos casos que el apartado anterior, se ha obtenido:

(1) código escalar del Apéndice A: **2,221 cpi**
(2) código escalar del Apéndice B: **1,328 cpi**
(3) código vectorial del Apéndice C, sin encadenamiento vectorial: **20,12 cpi**
(4) código vectorial del Apéndice C, con encadenamiento vectorial: **15,23 cpi**

Cuestión f)

La aceleración (speedup) respecto al código del Apéndice A se calcula como:

$$S_p = \frac{T_{cpu-original}}{T_{cpu-mejorado}} = \frac{Ciclos_{cpu-original}}{Ciclos_{cpu-mejorado}}$$

Los valores obtenidos han sido:

(1) Apéndice B: $S_p = \frac{4005}{1701} = 2{,}35$ **135% más rápido**

(2) Apéndice C, sin encadenamiento vectorial: $S_p = \frac{4005}{1026} = 3{,}9$ **290% más rápido**

(3) Apéndice C, con encadenamiento vectorial: $S_p = \frac{4005}{777} = 5{,}22$ **422% más rápido**

Conclusiones

La principal conclusión obtenida del desarrollo de este trabajo práctico es que programar en el ensamblador de un procesador ayuda enormemente a comprender, a tener consciencia, de múltiples conceptos relacionados con las máquinas computadoras. Ha sido una ayuda muy buena para comprender lo que pasa “detrás del telón”.
También ayuda a valorar las herramientas de alto nivel, como abstractoras de tareas y agilizadoras del trabajo.
Por supuesto otro factor a favor es la propia preparación de la asignatura, en tanto que nos acerca a la posición en la que estaremos en condiciones de aprobar el examen presencial.
En mi caso particular nunca había tenido contacto con ningún ensamblador, y este trabajo me ha permitido adquirir una minúscula experiencia en este ámbito, lo cual es muy de agradecer.

APÉNDICE A

```
                .data
                .align 2
x:              .double 1.00, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07, 1.08, 1.09
                .double 1.10, 1.11, 1.12, 1.13, 1.14, 1.15, 1.16, 1.17, 1.18, 1.19
                .double 1.20, 1.21, 1.22, 1.23, 1.24, 1.25, 1.26, 1.27, 1.28, 1.29
                .double 1.30, 1.31, 1.32, 1.33, 1.34, 1.35, 1.36, 1.37, 1.38, 1.39
                .double 1.40, 1.41, 1.42, 1.43, 1.44, 1.45, 1.46, 1.47, 1.48, 1.49
                .double 1.50, 1.51, 1.52, 1.53, 1.54, 1.55, 1.56, 1.57, 1.58, 1.59
                .double 1.60, 1.61, 1.62, 1.63, 1.64, 1.65, 1.66, 1.67, 1.68, 1.69
                .double 1.70, 1.71, 1.72, 1.73, 1.74, 1.75, 1.76, 1.77, 1.78, 1.79
                .double 1.80, 1.81, 1.82, 1.83, 1.84, 1.85, 1.86, 1.87, 1.88, 1.89
                .double 1.90, 1.91, 1.92, 1.93, 1.94, 1.95, 1.96, 1.97, 1.98, 1.99
                .double 2.00, 2.01, 2.02, 2.03, 2.04, 2.05, 2.06, 2.07, 2.08, 2.09
                .double 2.10, 2.11, 2.12, 2.13, 2.14, 2.15, 2.16, 2.17, 2.18, 2.19
                .double 2.20, 2.21, 2.22, 2.23, 2.24, 2.25, 2.26, 2.27, 2.28, 2.29
                .double 2.30, 2.31, 2.32, 2.33, 2.34, 2.35, 2.36, 2.37, 2.38, 2.39
                .double 2.40, 2.41, 2.42, 2.43, 2.44, 2.45, 2.46, 2.47, 2.48, 2.49
                .double 2.50, 2.51, 2.52, 2.53, 2.54, 2.55, 2.56, 2.57, 2.58, 2.59
                .double 2.60, 2.61, 2.62, 2.63, 2.64, 2.65, 2.66, 2.67, 2.68, 2.69
                .double 2.70, 2.71, 2.72, 2.73, 2.74, 2.75, 2.76, 2.77, 2.78, 2.79
                .double 2.80, 2.81, 2.82, 2.83, 2.84, 2.85, 2.86, 2.87, 2.88, 2.89
                .double 2.90, 2.91, 2.92, 2.93, 2.94, 2.95, 2.96, 2.97, 2.98, 2.99
y:              .double 1.00, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07, 1.08, 1.09
                .double 1.10, 1.11, 1.12, 1.13, 1.14, 1.15, 1.16, 1.17, 1.18, 1.19
                .double 1.20, 1.21, 1.22, 1.23, 1.24, 1.25, 1.26, 1.27, 1.28, 1.29
                .double 1.30, 1.31, 1.32, 1.33, 1.34, 1.35, 1.36, 1.37, 1.38, 1.39
                .double 1.40, 1.41, 1.42, 1.43, 1.44, 1.45, 1.46, 1.47, 1.48, 1.49
                .double 1.50, 1.51, 1.52, 1.53, 1.54, 1.55, 1.56, 1.57, 1.58, 1.59
                .double 1.60, 1.61, 1.62, 1.63, 1.64, 1.65, 1.66, 1.67, 1.68, 1.69
                .double 1.70, 1.71, 1.72, 1.73, 1.74, 1.75, 1.76, 1.77, 1.78, 1.79
                .double 1.80, 1.81, 1.82, 1.83, 1.84, 1.85, 1.86, 1.87, 1.88, 1.89
                .double 1.90, 1.91, 1.92, 1.93, 1.94, 1.95, 1.96, 1.97, 1.98, 1.99
                .double 2.00, 2.01, 2.02, 2.03, 2.04, 2.05, 2.06, 2.07, 2.08, 2.09
                .double 2.10, 2.11, 2.12, 2.13, 2.14, 2.15, 2.16, 2.17, 2.18, 2.19
                .double 2.20, 2.21, 2.22, 2.23, 2.24, 2.25, 2.26, 2.27, 2.28, 2.29
                .double 2.30, 2.31, 2.32, 2.33, 2.34, 2.35, 2.36, 2.37, 2.38, 2.39
                .double 2.40, 2.41, 2.42, 2.43, 2.44, 2.45, 2.46, 2.47, 2.48, 2.49
                .double 2.50, 2.51, 2.52, 2.53, 2.54, 2.55, 2.56, 2.57, 2.58, 2.59
                .double 2.60, 2.61, 2.62, 2.63, 2.64, 2.65, 2.66, 2.67, 2.68, 2.69
                .double 2.70, 2.71, 2.72, 2.73, 2.74, 2.75, 2.76, 2.77, 2.78, 2.79
                .double 2.80, 2.81, 2.82, 2.83, 2.84, 2.85, 2.86, 2.87, 2.88, 2.89
                .double 2.90, 2.91, 2.92, 2.93, 2.94, 2.95, 2.96, 2.97, 2.98, 2.99
a:              .double 3.14159265
                .text
main:           addi r1, r0, x          ; r1 ← X(0) addr
                addi r2, r0, y          ; r2 ← Y(0) addr
                ld f0, a                ; f0 ← a
                addi r4, r1, 1592       ; r4 ← X(199) addr ; [1592 = 199 * 8]
inicio:         ld f2, 0(r1)            ; f2 ← X(i)
                multd f4, f2, f0        ; f4 ← a * X(i)
                ld f6, 0(r2)            ; f6 ← Y(i)
                addd f6, f4, f6         ; f6 ← a * X(i) + Y(i)
                sd 0(r2), f6            ; Y(i) ← f6
                addi r1, r1, 8          ; r1 ← X(i+1) addr
                addi r2, r2, 8          ; r1 ← Y(i+1) addr
                sgt r3, r1, r4          ; r3 ← (r1 > r4)?
                beqz r3, inicio         ; PC ← inicio if r3==0
                trap 0                  ; end
```

APÉNDICE B

```
                .data
                .align 2
x:              .double 1.00, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07, 1.08, 1.09
                .double 1.10, 1.11, 1.12, 1.13, 1.14, 1.15, 1.16, 1.17, 1.18, 1.19
                .double 1.20, 1.21, 1.22, 1.23, 1.24, 1.25, 1.26, 1.27, 1.28, 1.29
                .double 1.30, 1.31, 1.32, 1.33, 1.34, 1.35, 1.36, 1.37, 1.38, 1.39
                .double 1.40, 1.41, 1.42, 1.43, 1.44, 1.45, 1.46, 1.47, 1.48, 1.49
                .double 1.50, 1.51, 1.52, 1.53, 1.54, 1.55, 1.56, 1.57, 1.58, 1.59
                .double 1.60, 1.61, 1.62, 1.63, 1.64, 1.65, 1.66, 1.67, 1.68, 1.69
                .double 1.70, 1.71, 1.72, 1.73, 1.74, 1.75, 1.76, 1.77, 1.78, 1.79
                .double 1.80, 1.81, 1.82, 1.83, 1.84, 1.85, 1.86, 1.87, 1.88, 1.89
                .double 1.90, 1.91, 1.92, 1.93, 1.94, 1.95, 1.96, 1.97, 1.98, 1.99
                .double 2.00, 2.01, 2.02, 2.03, 2.04, 2.05, 2.06, 2.07, 2.08, 2.09
                .double 2.10, 2.11, 2.12, 2.13, 2.14, 2.15, 2.16, 2.17, 2.18, 2.19
                .double 2.20, 2.21, 2.22, 2.23, 2.24, 2.25, 2.26, 2.27, 2.28, 2.29
                .double 2.30, 2.31, 2.32, 2.33, 2.34, 2.35, 2.36, 2.37, 2.38, 2.39
                .double 2.40, 2.41, 2.42, 2.43, 2.44, 2.45, 2.46, 2.47, 2.48, 2.49
                .double 2.50, 2.51, 2.52, 2.53, 2.54, 2.55, 2.56, 2.57, 2.58, 2.59
                .double 2.60, 2.61, 2.62, 2.63, 2.64, 2.65, 2.66, 2.67, 2.68, 2.69
                .double 2.70, 2.71, 2.72, 2.73, 2.74, 2.75, 2.76, 2.77, 2.78, 2.79
                .double 2.80, 2.81, 2.82, 2.83, 2.84, 2.85, 2.86, 2.87, 2.88, 2.89
                .double 2.90, 2.91, 2.92, 2.93, 2.94, 2.95, 2.96, 2.97, 2.98, 2.99
y:              .double 1.00, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07, 1.08, 1.09
                .double 1.10, 1.11, 1.12, 1.13, 1.14, 1.15, 1.16, 1.17, 1.18, 1.19
                .double 1.20, 1.21, 1.22, 1.23, 1.24, 1.25, 1.26, 1.27, 1.28, 1.29
                .double 1.30, 1.31, 1.32, 1.33, 1.34, 1.35, 1.36, 1.37, 1.38, 1.39
                .double 1.40, 1.41, 1.42, 1.43, 1.44, 1.45, 1.46, 1.47, 1.48, 1.49
                .double 1.50, 1.51, 1.52, 1.53, 1.54, 1.55, 1.56, 1.57, 1.58, 1.59
                .double 1.60, 1.61, 1.62, 1.63, 1.64, 1.65, 1.66, 1.67, 1.68, 1.69
                .double 1.70, 1.71, 1.72, 1.73, 1.74, 1.75, 1.76, 1.77, 1.78, 1.79
                .double 1.80, 1.81, 1.82, 1.83, 1.84, 1.85, 1.86, 1.87, 1.88, 1.89
                .double 1.90, 1.91, 1.92, 1.93, 1.94, 1.95, 1.96, 1.97, 1.98, 1.99
                .double 2.00, 2.01, 2.02, 2.03, 2.04, 2.05, 2.06, 2.07, 2.08, 2.09
                .double 2.10, 2.11, 2.12, 2.13, 2.14, 2.15, 2.16, 2.17, 2.18, 2.19
                .double 2.20, 2.21, 2.22, 2.23, 2.24, 2.25, 2.26, 2.27, 2.28, 2.29
                .double 2.30, 2.31, 2.32, 2.33, 2.34, 2.35, 2.36, 2.37, 2.38, 2.39
                .double 2.40, 2.41, 2.42, 2.43, 2.44, 2.45, 2.46, 2.47, 2.48, 2.49
                .double 2.50, 2.51, 2.52, 2.53, 2.54, 2.55, 2.56, 2.57, 2.58, 2.59
                .double 2.60, 2.61, 2.62, 2.63, 2.64, 2.65, 2.66, 2.67, 2.68, 2.69
                .double 2.70, 2.71, 2.72, 2.73, 2.74, 2.75, 2.76, 2.77, 2.78, 2.79
                .double 2.80, 2.81, 2.82, 2.83, 2.84, 2.85, 2.86, 2.87, 2.88, 2.89
                .double 2.90, 2.91, 2.92, 2.93, 2.94, 2.95, 2.96, 2.97, 2.98, 2.99
a:              .double 3.14159265
                .text
main:           addi r1, r0, x          ; r1 ← X(0) addr
                addi r2, r0, y          ; r2 ← Y(0) addr
                ld f0, a                ; f0 ← a
                addi r4, r1, 1592       ; r4 ← X(199) addr ; [1592 = 199 * 8]
                addi r5, r0, 200        ; r5 ← VL (vector length)
                addi r6, r0, 3
                div r7, r5, r6
                mult r8, r7, r6
                sub r9, r5, r8          ; r9 ← VL mod 3
                beqz r9, thickloop
thinloop:       ld f2, 0(r1)            ; f2 ← X(i)
                multd f4, f2, f0        ; f4 ← a * X(i)
                ld f6, 0(r2)            ; f6 ← Y(i)
                addd f6, f4, f6         ; f6 ← a * X(i) + Y(i)
                sd 0(r2), f6            ; Y(i) ← f6
                addi r1, r1, 8          ; r1 ← X(i+1) addr
                addi r2, r2, 8          ; r2 ← Y(i+1) addr
                subi r9, r9, 1
                bnez r9, thinloop       ; PC ← thinloop if r9==0
thickloop:      ld f2, 0(r1)            ; f2 ← X(i)
                ld f4, 8(r1)            ; f4 ← X(i+1)
                ld f6, 16(r1)           ; f6 ← X(i+2)
                multd f8, f2, f0        ; f8 ← a * X(i)
                multd f10, f4, f0       ; f10 ← a * X(i+1)
                multd f12, f6, f0       ; f12 ← a * X(i+2)
                ld f14, 0(r2)           ; f14 ← Y(i)
                ld f16, 8(r2)           ; f16 ← Y(i+1)
                ld f18, 16(r2)          ; f18 ← Y(i+2)
                addd f20, f8, f14       ; f20 ← a * X(i) + Y(i)
                addd f22, f10, f16      ; f22 ← a * X(i+1) + Y(i+1)
                addd f24, f12, f18      ; f24 ← a * X(i+2) + Y(i+2)
                sd 0(r2), f20           ; Y(i) ← f20
```

```
sd 8(r2), f22          ; Y(i+1) ← f22
sd 16(r2), f24         ; Y(i+2) ← f24
addi r1, r1, 24        ; r1 ← X(i+3) addr
addi r2, r2, 24        ; r2 ← Y(i+3) addr
sgt r3, r1, r4         ; r3 ← (r1 > r4)?
beqz r3, thickloop     ; PC ← thickloop if r3==0
trap 0                 ; end
```

APÉNDICE C

```
                .data
                .align 2
x:              .double 1.00, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07, 1.08, 1.09
                .double 1.10, 1.11, 1.12, 1.13, 1.14, 1.15, 1.16, 1.17, 1.18, 1.19
                .double 1.20, 1.21, 1.22, 1.23, 1.24, 1.25, 1.26, 1.27, 1.28, 1.29
                .double 1.30, 1.31, 1.32, 1.33, 1.34, 1.35, 1.36, 1.37, 1.38, 1.39
                .double 1.40, 1.41, 1.42, 1.43, 1.44, 1.45, 1.46, 1.47, 1.48, 1.49
                .double 1.50, 1.51, 1.52, 1.53, 1.54, 1.55, 1.56, 1.57, 1.58, 1.59
                .double 1.60, 1.61, 1.62, 1.63, 1.64, 1.65, 1.66, 1.67, 1.68, 1.69
                .double 1.70, 1.71, 1.72, 1.73, 1.74, 1.75, 1.76, 1.77, 1.78, 1.79
                .double 1.80, 1.81, 1.82, 1.83, 1.84, 1.85, 1.86, 1.87, 1.88, 1.89
                .double 1.90, 1.91, 1.92, 1.93, 1.94, 1.95, 1.96, 1.97, 1.98, 1.99
                .double 2.00, 2.01, 2.02, 2.03, 2.04, 2.05, 2.06, 2.07, 2.08, 2.09
                .double 2.10, 2.11, 2.12, 2.13, 2.14, 2.15, 2.16, 2.17, 2.18, 2.19
                .double 2.20, 2.21, 2.22, 2.23, 2.24, 2.25, 2.26, 2.27, 2.28, 2.29
                .double 2.30, 2.31, 2.32, 2.33, 2.34, 2.35, 2.36, 2.37, 2.38, 2.39
                .double 2.40, 2.41, 2.42, 2.43, 2.44, 2.45, 2.46, 2.47, 2.48, 2.49
                .double 2.50, 2.51, 2.52, 2.53, 2.54, 2.55, 2.56, 2.57, 2.58, 2.59
                .double 2.60, 2.61, 2.62, 2.63, 2.64, 2.65, 2.66, 2.67, 2.68, 2.69
                .double 2.70, 2.71, 2.72, 2.73, 2.74, 2.75, 2.76, 2.77, 2.78, 2.79
                .double 2.80, 2.81, 2.82, 2.83, 2.84, 2.85, 2.86, 2.87, 2.88, 2.89
                .double 2.90, 2.91, 2.92, 2.93, 2.94, 2.95, 2.96, 2.97, 2.98, 2.99
y:              .double 1.00, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07, 1.08, 1.09
                .double 1.10, 1.11, 1.12, 1.13, 1.14, 1.15, 1.16, 1.17, 1.18, 1.19
                .double 1.20, 1.21, 1.22, 1.23, 1.24, 1.25, 1.26, 1.27, 1.28, 1.29
                .double 1.30, 1.31, 1.32, 1.33, 1.34, 1.35, 1.36, 1.37, 1.38, 1.39
                .double 1.40, 1.41, 1.42, 1.43, 1.44, 1.45, 1.46, 1.47, 1.48, 1.49
                .double 1.50, 1.51, 1.52, 1.53, 1.54, 1.55, 1.56, 1.57, 1.58, 1.59
                .double 1.60, 1.61, 1.62, 1.63, 1.64, 1.65, 1.66, 1.67, 1.68, 1.69
                .double 1.70, 1.71, 1.72, 1.73, 1.74, 1.75, 1.76, 1.77, 1.78, 1.79
                .double 1.80, 1.81, 1.82, 1.83, 1.84, 1.85, 1.86, 1.87, 1.88, 1.89
                .double 1.90, 1.91, 1.92, 1.93, 1.94, 1.95, 1.96, 1.97, 1.98, 1.99
                .double 2.00, 2.01, 2.02, 2.03, 2.04, 2.05, 2.06, 2.07, 2.08, 2.09
                .double 2.10, 2.11, 2.12, 2.13, 2.14, 2.15, 2.16, 2.17, 2.18, 2.19
                .double 2.20, 2.21, 2.22, 2.23, 2.24, 2.25, 2.26, 2.27, 2.28, 2.29
                .double 2.30, 2.31, 2.32, 2.33, 2.34, 2.35, 2.36, 2.37, 2.38, 2.39
                .double 2.40, 2.41, 2.42, 2.43, 2.44, 2.45, 2.46, 2.47, 2.48, 2.49
                .double 2.50, 2.51, 2.52, 2.53, 2.54, 2.55, 2.56, 2.57, 2.58, 2.59
                .double 2.60, 2.61, 2.62, 2.63, 2.64, 2.65, 2.66, 2.67, 2.68, 2.69
                .double 2.70, 2.71, 2.72, 2.73, 2.74, 2.75, 2.76, 2.77, 2.78, 2.79
                .double 2.80, 2.81, 2.82, 2.83, 2.84, 2.85, 2.86, 2.87, 2.88, 2.89
                .double 2.90, 2.91, 2.92, 2.93, 2.94, 2.95, 2.96, 2.97, 2.98, 2.99
a:              .double 3.14159265
                .text
main:           cvm                     ; reset mask
                addi r1, r0, x          ; r1 ← X(0) addr
                addi r2, r0, y          ; r2 ← Y(0) addr
                ld f0, a                ; f0 ← a
                addi r4, r1, 1592       ; r4 ← X(199) addr ; [1592 = 199 * 8]
                addi r5, r0, 200        ; r5 ← VL (vector length)
                addi r6, r0, 64         ; r6 ← MVL (maximum vector length)
                addi r11, r0, 8         ; r11 ← byte length
                div r7, r5, r6          ;
                mult r8, r7, r6         ;
                sub r9, r5, r8          ; r9 ← VL mod MVL
                beqz r9, loopsetup      ; PC ← loopsetup if (VL mod MVL)==0
prologue:       movi2s vlr, r9          ; vlr ← VL mod MVL
                lv v0, 0(r1)            ; v0 ← [X(0),...,X((VL mod MVL) - 1)]
                multsv v1, f0, v0       ; v1 ← [a * X(0),...,a * X((VL mod MVL) - 1)]
                lv v2, 0(r2)            ; v2 ← [Y(0),...,Y((VL mod MVL) - 1)]
                addv v2, v1, v2         ; v2 ← [a*X(0)+Y(0),...,a*X((VLmodMVL)-1)+Y((VLmodMVL)-1)]
                sv 0(r2), v2            ; [Y(0),...,Y((VL mod MVL) - 1)] ← v2
                sgt r3, r6, r5          ; r3 ← (VL < MVL)?
                bnez r3, end            ; PC ← end if VL < MVL
                mult r10, r9, r11       ; r10 ← (VL mod MVL) * 8 [bytes]
                add r1, r1, r10         ; r1 ← X(VL mod MVL) addr
                add r2, r2, r10         ; r2 ← Y(VL mod MVL) addr
loopsetup:      movi2s vlr, r6          ; VLR ← MVL
                mult r10, r6, r11       ; r10 ← MVL * 8 [bytes]
loop:           lv v0, 0(r1)            ; v0 ← [X(i),...,X(i + MVL - 1)]
                multsv v1, f0, v0       ; v1 ← [a * X(i),...,a * X((i + MVL - 1)]
                lv v2, 0(r2)            ; v2 ← [Y(i),...,Y((i + MVL - 1)]
                addv v2, v1, v2         ; v2 ← [a*X(i)+Y(i),...,a*X(i+MVL-1)+Y(i+MVL-1)]
                sv 0(r2), v2            ; [Y(i),...,Y(i + MVL - 1)] ← v2
                add r1, r1, r10         ; r1 ← X(i + MVL) addr
                add r2, r2, r10         ; r2 ← Y(i + MVL) addr
```

```
                sgt r3, r1, r4          ; r3 ← (X(i + MVL) addr > X(199) addr)?
                beqz r3, loop           ; PC ← loop if r3==0
end:            trap 0                  ; end
```