Práctica 2.

Determinación de la microarquitectura de la memoria cache a partir de la evaluación de prestaciones de un computador real

El principal objetivo de esta práctica consiste en manejar distintos elementos de la arquitectura del Computador Básico que se configura en la placa DE0-Nano y que se encargan de implementar varios niveles de lo que denominamos Jerarquía de Memoria. Estos niveles e implementaciones de la jerarquía de memoria en DE0-Nano son los siguientes:

- El nivel de la memoria principal que se puede implementar con DE0-Nano utilizando dos tecnologías electrónicas distintas:
    - Con circuitos electrónicos de tipo SDRAM que se encuentran en el exterior del chip donde se encuentra el procesador (ver Figura 1).
    - Con circuitos electrónicos de tipo SRAM. En DE0-Nano existe un dispositivo SRAM que se utiliza en esta práctica para implementar la memoria principal (ver Figura 1):
        - La *memoria on-chip* (on-chip memory) que es una memoria externa al procesador pero que está en el mismo chip donde se encuentra el procesador que se denomina FPGA (Field Programmable Gate Array).
- El nivel de la memoria cache:
    - Este nivel se implementa con circuitos electrónicos de tipo SRAM que se encuentran en el mismo chip donde se encuentra el procesador (FPGA, ver Figura 1).

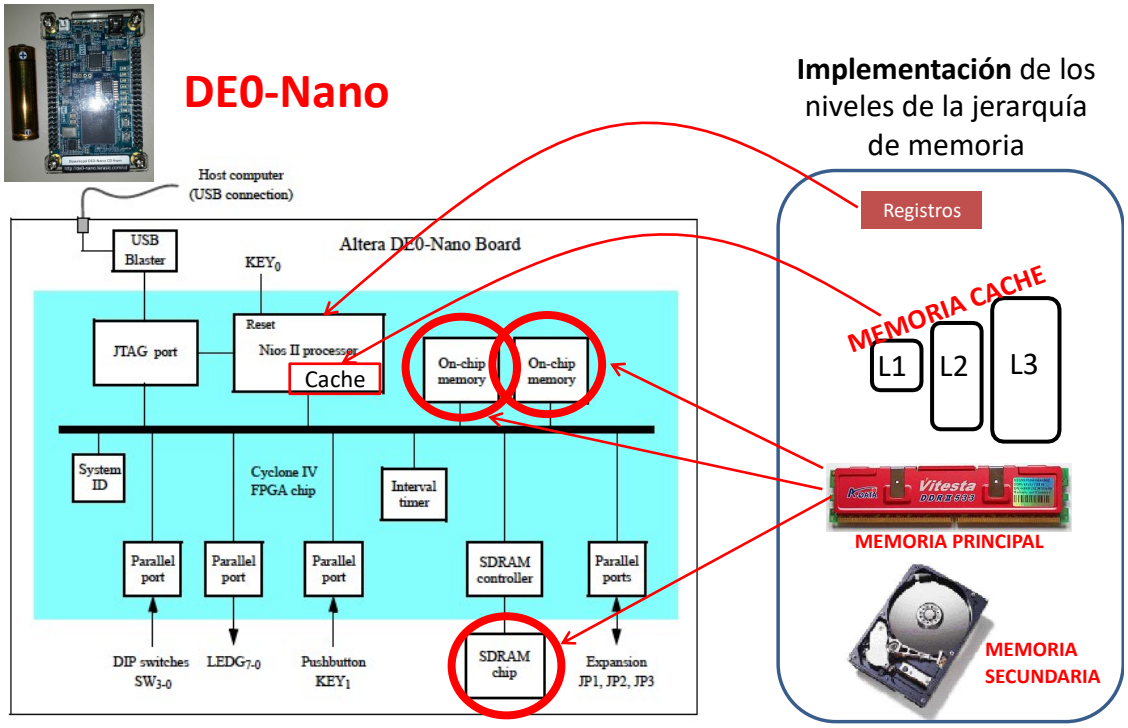

Figura 1. Implementación de los niveles de la Jerarquía de Memoria de la arquitectura del Computador Básico de la placa DE0-Nano

Adicionalmente, en esta práctica se utilizarán dos versiones hardware del procesador NIOS II que se denominan **Nios II/e** (economy) y **Nios II/f** (fast), respectivamente. Cada tipo de procesador y memoria causan que el tiempo de ejecución de los programas sea distinto.

En esta práctica se realizará la medida del tiempo de ejecución de un mismo programa denominado Fibonacci cuando es ejecutado con distintas configuraciones de la arquitectura del Computador Básico en la placa DE0-Nano. Estas configuraciones se distinguen en el tipo de procesador configurado: Nios II/e o Nios II/f, así como en el tipo de implementación activada para los niveles de la jerarquía de memoria: cache, memoria principal (SRAM o SDRAM).

La metodología de prácticas consiste en realizar cuatro actividades con la placa DE0-Nano y el entorno software Altera Monitor Program (AMP) de Altera. En los ejercicios prácticos se solicitará razonar los resultados que has obtenido experimentalmente.

Actividad 1. Acceso a la memoria SDRAM con el procesador Nios II/e (economy)

El objetivo de esta primera actividad consiste en **medir el tiempo real de cómputo** del programa Fibonacci utilizando la memoria SDRAM de la placa DE0-Nano, además del procesador Nios II/e, y el temporizador Timer que es un dispositivo de entrada/salida que acompaña a los componentes hardware mencionados (ver Figura 1).

Durante la ejecución del programa Fibonacci, el procesador Nios II cuenta el número de intervalos de 33 ms que han pasado. Al finalizar el programa Fibonacci, el terminal que se encuentra en la aplicación AMP muestra el número de intervalos completos de 33 ms que han pasado, como se puede observar en la Figura 2.

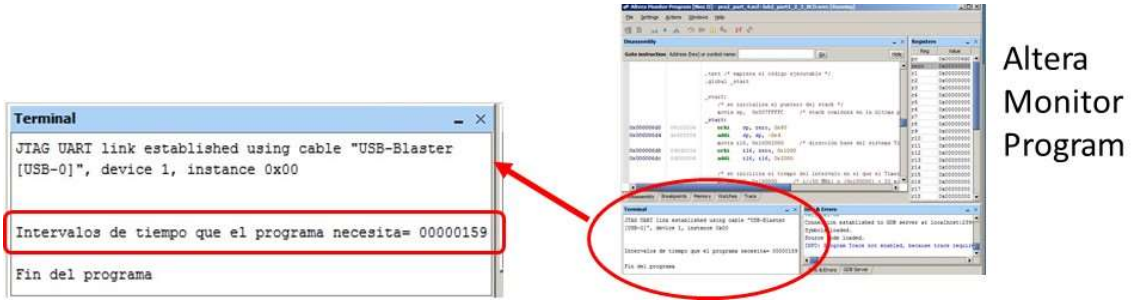

Figura 2. Visualización del resultado final de la ejecución del programa utilizado para medir el tiempo de ejecución de los programas. El valor del número de intervalos de 33 ms aparece en el terminal de la aplicación AMP.

Los programas ensamblador involucrados en esta actividad de la práctica son:

- `lab2_part1_2_3_main.s`: programa principal (ver Anexo 1).
- `lab2_part1_2_3_fibo.s`: rutina benchmark de cómputo de la que se pretende medir su tiempo de cómputo (ver Anexo 2).
- `lab2_part1_2_3_interrupts.s`: rutina que se ejecuta cuando se activa la interrupción del Timer (ver Anexo 3).
- `lab2_part1_2_3_excepciones.s`: rutina que se llama desde lab2_part1_2_3_interrupts.s y que gestiona el contador de tiempo de ejecución (ver Anexo 4).
- `lab2_part1_2_3_JPEG.s`: rutina que se llama desde lab2_part1_2_3_main.s para mostrar en el terminal de AMP el número de intervalos que han pasado hasta la finalización de la ejecución del programa benchmark (ver Anexo 5).
- `lab2_part1_2_3_BCD.s`: rutina que se llama desde lab2_part1_2_3_JTAG.s para transformar un código binario en BCD (ver Anexo 6).
- `lab2_part1_2_3_div.s`: rutina que se llama desde lab2_part1_2_3_BCD.s para realizar la división entera (ver Anexo 7).

El flujo del programa benchmark que usaremos para medir el tiempo de ejecución se muestra en la Figura 3. Como se puede observar, la aplicación informática activa el sistema de interrupciones del Timer en el programa principal (ver `lab2_part1_2_3_main.s`).

La rutina de servicio de interrupciones permite mantener un contador de eventos en una posición de memoria denominada CONTADOR (ver fichero `lab2_part1_2_3_excepciones.s`). Cada evento consiste en la indicación que ha pasado un intervalo de 33 ms desde que terminó el anterior intervalo de tiempo. Por tanto, CONTADOR registra el número de intervalos de 33 ms que han pasado desde que se activa el Timer.

De forma concurrente a la medida de tiempo que realiza el Timer, el programa principal ejecuta el bucle Fibonacci un número de veces que es indicado por la constante ITERACIONES (ver fichero `lab2_part1_2_3_main.s`). Cuando termina este bucle, se consulta el contenido de la posición de memoria CONTADOR y su valor se visualiza en el terminal de la aplicación AMP (ver fichero `lab2_part1_2_3_JTAG.s`). Para este último proceso se transforma el código binario que representa al valor de CONTADOR en un número codificado en BCD y posteriormente en un código ASCII. En la transformación a código BCD es necesario realizar una o varias divisiones, para lo cual se incluye una rutina de división ya que el procesador Nios II/e no dispone del hardware para realizar divisiones (ver fichero `lab2_part1_2_3_JTAG.s`).

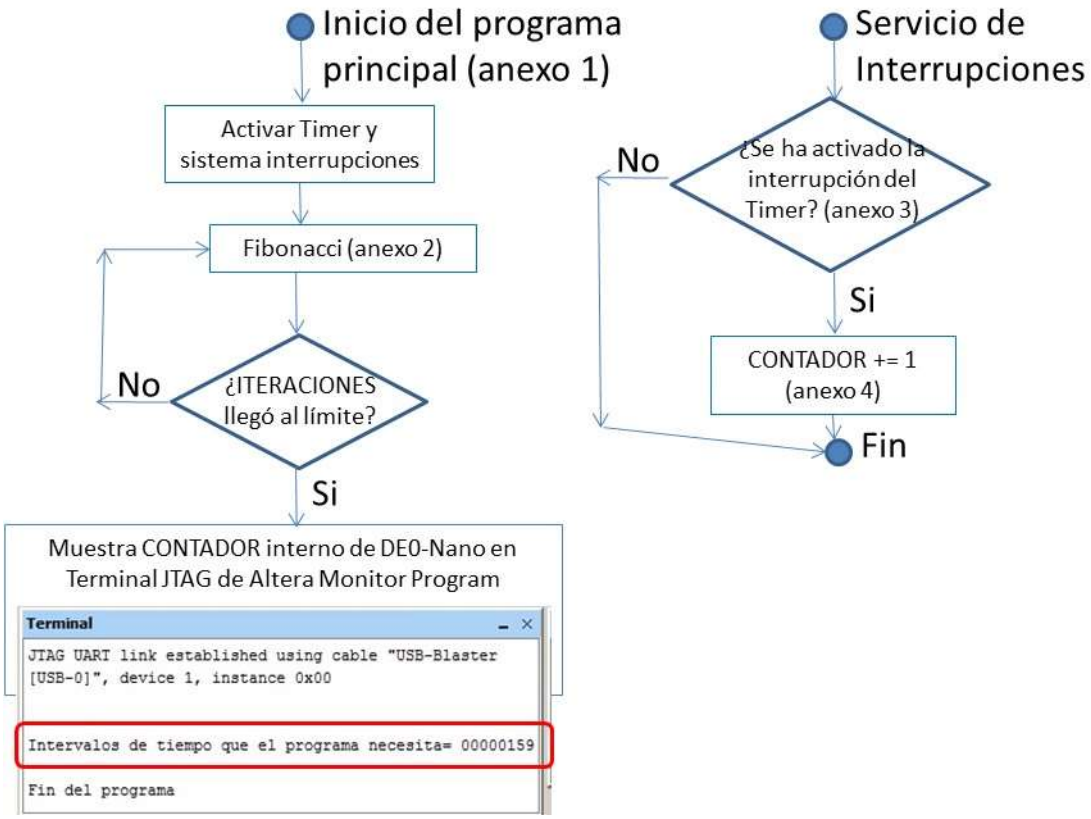

Figura 3. Diagrama de flujo del programa de la Práctica 2.

Metodología de prácticas

Iniciar un nuevo proyecto en el entorno software Altera Monitor Program (AMP) de la misma forma que en la práctica anterior, seleccionando la configuración *DE0-Nano Basic Computer*, e incluyendo los siete ficheros en ensamblador (<`*.s`>) indicados al principio de esta actividad práctica. Adicionalmente, especificar en el proyecto AMP que los programas y datos se almacenen a partir de la dirección del espacio de direccionamiento en memoria `0x400` (`_start`). Esto se hace de la siguiente forma dentro de AMP:

- Settings > System settings > Memory settings >

  .text – memory device = SDRAM/s1; start offset in device (hex) = 400

  .data – memory device = SDRAM/s1; start offset in device (hex) = 400.

A continuación, cargar la configuración en la placa DE0-Nano:

- Actions > Donwload System > Download

Seguidamente, compilar los programas y cargarlos en la memoria:

- Actions > Compile & Load

Finalmente, ejecutar el programa hasta el final:

- Actions > Continue

A continuación, anotar en la Tabla 1 el valor del tiempo de ejecución que se muestra en el terminal de AMP. Este tiempo coincide con el número de intervalos de 33 ms que han pasado durante la ejecución del programa Fibonacci. El tiempo de ejecución se comparará con el que se obtenga en las siguientes dos actividades de esta práctica. Fijarse que el programa `lab2_part1_2_3_main.s` tiene declarada e inicializada una constante: ITERACIONES = 500000.

Cambiar ITERACIONES de 500.000 a 100.000 en el fichero `lab2_part1_2_3_main.s`, compilar de nuevo y cargar el programa en la placa. A continuación, ejecutar el programa y apuntar el correspondiente valor que se muestra por el terminal de AMP.

Pregunta 1

¿Este nuevo resultado de prestaciones temporales es razonable? Razónalo y justifica la respuesta.

Actividad 2. Acceso a la memoria "on-chip" con el procesador Nios II/e

El objetivo de esta segunda actividad consiste en **medir el tiempo real de cómputo** del programa Fibonacci utilizando la memoria SRAM que se encuentra dentro del chip

(denominada "on-chip") donde también se encuentra el procesador Nios II/e. La diferencia con la Actividad 1 consiste en que se utiliza una memoria SRAM que se ha fabricado con otra tecnología y además se encuentra más cerca del procesador físicamente, en vez de utilizar la memoria SDRAM externa al procesador y que se encuentra soldada en la placa DE0-Nano.

Metodología de prácticas

Seleccionar el anterior proyecto de AMP de la Actividad 1 y el espacio de direccionamiento "on-chip" de la siguiente forma:

- Settings > System settings > Memory settings >

  .text – memory device = onchip memory/s1 (0x9000000 – 0x9001FFF)

  .text – start offset in device (hex) = 400

  .data – memory device = onchip memory/s2 (0x9000000 – 0x9001FFF)

  .data – start offset in device (hex) = 400.

En el código fuente del programa (fichero `lab2_part1_2_3_main.s`) se debe mantener el número de iteraciones a 500000.

Compilar de nuevo los ficheros ensamblador del proyecto AMP y cargar el ejecutable en la placa DE0-Nano. A continuación, ejecutar el programa y anotar en la Tabla 1 el valor del tiempo de ejecución que se muestra en el terminal de AMP.

Pregunta 2

¿Este nuevo resultado de prestaciones temporales es razonable? Razónalo y justifica la respuesta utilizando el diagrama de la Figura 1 que se encuentra al principio de este documento.

Tabla 1. Medidas de prestaciones

Versión de procesador + tecnología memoria	**Tiempo de ejecución**	**Speed-up**
Nios II/e + memoria SDRAM (Actividad 1)		1X
Nios II/e + memoria on-chip (Actividad 2)		
Nios II/f + memoria SDRAM (Actividad 3)		
Nios II/f + memoria on-chip (Actividad 3)		

Actividad 3. Acceso a la memoria cache del procesador Nios II/f (fast)

El objetivo de esta actividad consiste en **medir el tiempo real de cómputo** del programa Fibonacci utilizando el procesador Nios II/f. Esta versión del procesador Nios II proporciona mayor nivel de prestaciones temporales ya que implementa varias características que no tiene el procesador de las dos actividades anteriores. Nios II/f dispone de: una ruta de datos segmentada de seis etapas, memoria cache de datos e instrucciones de primer nivel, y predicción dinámica de saltos.

Metodología de prácticas

Para esta actividad práctica, se procederá a cambiar la configuración de la placa DE0-Nano de la forma que se indica a continuación.

Seleccionar el proyecto de AMP de la Actividad 2:

- Settings > System Settings >

Selecciona una configuración personalizada:

- Select a system: > `<Custom System>` >

Cambiar los ficheros de descripción del computador (`nios_system.sopcinfo`) y el de configuración de la FPGA de la placa Altera (`DE0_Nano_Basic_Computer.sof`) según se puede observar en la Figura 4.

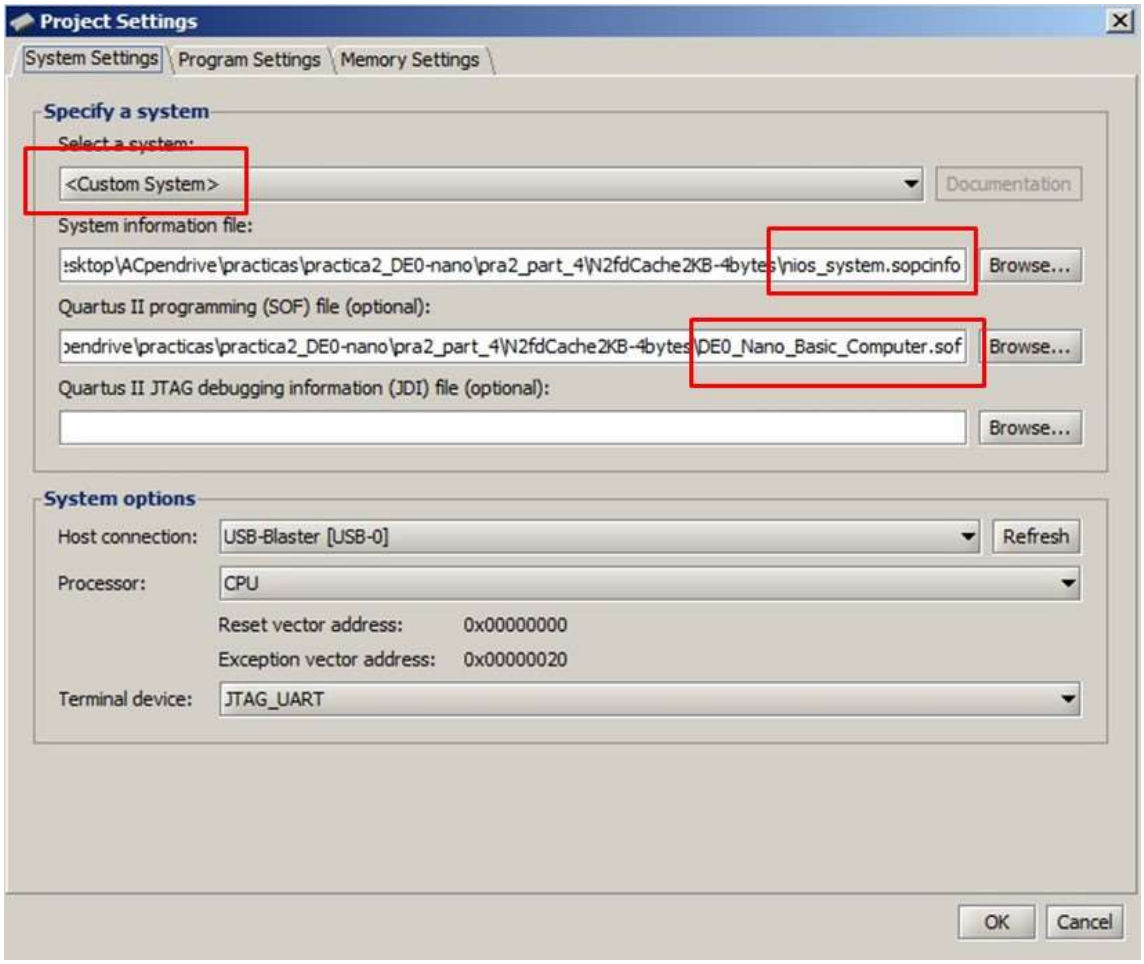

Figura 4. Parte del proyecto AMP donde se selecciona una arquitectura de computador basada en el procesador Nios II / f.

Tabla 2. Ficheros de configuración de la placa DE0-Nano

Placa	**Fichero `SOPCINFO`**	**Fichero `SOF`**
DE0-Nano	`nios_system.sopcinfo`	`DE0_Nano_Basic_Computer.sof`

Nota: estos ficheros de configuración están disponibles en el fichero comprimido lab2.rar (ver "Material de la Práctica 2" en la página Moodle)

Adicionalmente, se debe seleccionar cualquier tipo de espacio de direccionamiento de las dos alternativas que se pueden elegir (on-chip, SDRAM), con la restricción de que la parte de código (`.text`) y la de datos (`.data`) comience a partir de la dirección 0x400.

- Settings > System settings > memory settings >

  .text – start offset in device (hex) = 400

  .data – start offset in device (hex) = 400.

Anotar en la Tabla 1 el número de intervalos de 33 ms (Tiempo de ejecución) que aparece en el terminal de AMP al final de cada ejecución del programa. Calcular el Speed-up que se obtiene con las distintas versiones del procesador Nios II utilizando como benchmark el programa Fibonacci y considerando como sistema base (Speed-up = 1X) al procesador Nios II/e cuyo espacio de direccionamiento se encuentra asignado a la memoria externa SDRAM de la placa DE0-Nano. Finalmente, responder a las siguientes preguntas.

Pregunta 3

¿Cuál es la razón por la que las distintas versiones del procesador Nios II/e proporciona tal variación de prestaciones?

Pregunta 4

¿Cuál es la razón por la que las dos versiones del procesador Nios II/f coinciden en tiempo de ejecución?

Pregunta 5

¿Cuál es la razón por la que la versión Nios II/e proporciona peores prestaciones que Nios II/f?

Actividad 4. Descubrimiento de la arquitectura interna de la memoria cache de datos

Para descubrir la capacidad de almacenamiento y el tamaño de bloque de la arquitectura de la memoria cache de datos del procesador Nios II/f utilizaremos un programa sencillo que recorra un vector de bytes (`V`). Los pasos a seguir en esta actividad práctica son los siguientes:

- Reducir el número de iteraciones del programa principal (ITERACIONES) de 500000 a 50000 (fichero: `lab2_part1_2_3_main.s`). En cada ITERACIONES se hace una llamada a la subrutina FIBONACCI. El código fuente de esta subrutina se encuentra en el fichero: `lab2_part1_2_3_fibo.s`.
- Modificar de la forma que se muestra a continuación el código de la subrutina Fibonacci para que se limite simplemente a recorrer un vector `V` de bytes (fichero: `lab2_part1_2_3_fibo.s`):

```
        ...
        movi r4, 0
        movi r5, X
LOOP:   bge  r4, r5, END
        ldb  r0, V(r4)
        addi r4, r4, P
        br   LOOP
END:
        ...
.data
V:
.skip 65536
...
```

  Observar que en el código anterior existen dos parámetros a los que se necesita dar valores: **`X`** y **`P`**. **`X`** representa al número de elementos del vector `V` que se van a usar para acceder a ellos con una pauta **`P`**. La pauta **`P`** es el número de elementos del vector `V` que se encuentran en memoria entre dos sucesivos accesos con la instrucción *`ldb`*.

  Se define un nuevo parámetro, **E**, que representa al número de elementos del vector realmente accedidos con la instrucción `ldb`. De esta forma, **`X`** = **`P`** × **E**.

- A continuación, se rellenará la Tabla 3 con valores de tiempo de ejecución obtenidos de la misma forma que en las actividades anteriores, midiendo el tiempo de ejecución del programa benchmark. Este tiempo es en su mayor parte debido al que se necesita para recorrer parte de los elementos del vector `V` con una pauta de salto (**`P`**):
  - de uno en uno (**`P`** = 1: *`addi r4, r4, 1`*)
  - de dos en dos (**`P`** = 2: *`addi r4, r4, 2`*)
  - de cuatro en cuatro (**`P`** = 4: *`addi r4, r4, 4`*)
  - de ocho en ocho (**`P`** = 8: *`addi r4, r4, 8`*)
  - de diez y seis en diez y seis (**`P`** = 16: *`addi r4, r4, 16`*)
  - de treinta y dos en treinta y dos (**`P`** = 32: *`addi r4, r4, 32`*)

El número de elementos de `V` que son necesarios para realizar los **E** accesos será **X** para cada una de estas pautas. Por ejemplo, para la primera columna de la Tabla 3, los valores de **X**, **E** y **P** son los siguientes:

- **X** = 128 para **E** = 128, **P** = 1 (`movi r5, 128`)
- **X** = 256 para **E** = 128, **P** = 2 (`movi r5, 256`)
- **X** = 512 para **E** = 128, **P** = 4 (`movi r5, 512`)
- **X** = 1024 para **E** = 128, **P** = 8 (`movi r5, 1024`)
- **X** = 2048 para **E** = 128, **P** = 16 (`movi r5, 2048`)
- **X** = 4096 para **E** = 128, **P** = 32 (`movi r5, 4096`)

Observar que, en todos los casos de **X**, el número de iteraciones del bucle del programa es **E**=128. Por tanto, **X** = **E** x **P**. El número de accesos a memoria y de instrucciones `ldb` ejecutadas es también 128.

- Rellenar la Tabla 3 con los datos obtenidos en el punto anterior y dibujar una gráfica (ver Figura 5). Para rellenar la Tabla 3 seguir el siguiente procedimiento:

1) Ejecutar AMP
2) New project

CAMBIA-CACHE:

3) Settings > System Settings > Select a system > `Custom System`
4) Settings > System Settings > System information file + browse > `nios_system.sopcinfo`
5) Settings > System Settings > Quartus II Programming File + browse > `DE0_Nano_Basic_Computer.sof`
6) Settings > Memory Settings > Memory device > `SDRAM`
7) Actions > Download System

CAMBIA-DATOS:

8) Modificar fichero del programa: `lab2_part1_2_3_fibo.s` para establecer nuevos valores de **X** y **P**
9) Compile
10) Load
11) Ejecutar el programa y esperar a que salga el tiempo en el terminal de AMP y apuntarlo en la Tabla 3

SEGUIR CAMBIA-DATOS

SEGUIR CAMBIA-CACHE

Tabla 3. Tabla que se utiliza en la Actividad 4 para recoger las medidas de tiempos de ejecución.

	E: Número de bytes del vector V realmente accedidos					
P: Pauta de salto	128	256	512	1024	2048	4096
P = 1: de 1 en 1						
P = 2: de 2 en 2						
P = 4: de 4 en 4						
P = 8: de 8 en 8						

- En base a los datos obtenidos, deducir (descubrir) la capacidad total de almacenamiento de la memoria cache de datos y el tamaño del bloque de cache, razonando y justificando todas las respuestas. Ayuda: Acordarse de que en clase de teoría hemos explicado y calculado el concepto de tiempo de acceso promedio a la memoria:

$$AMAT = t_{acierto} + FrecuenciaFallos \times Penalización$$

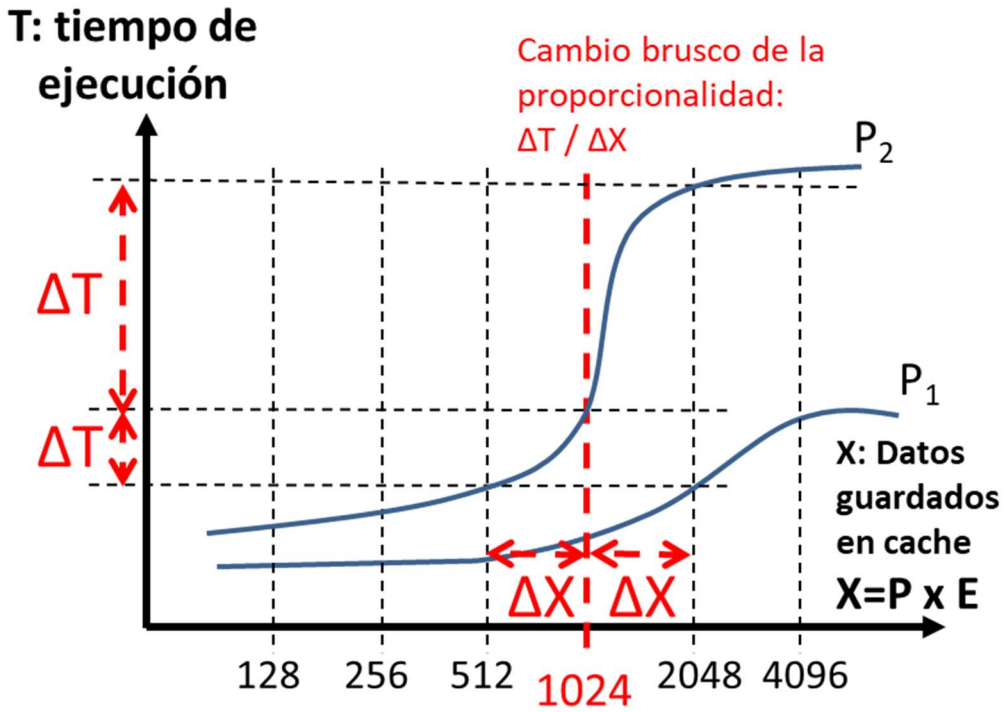

Figura 5. Gráfica donde se representa la relación entre el tiempo de ejecución del programa y el número de elementos almacenados en la cache (X) del vector V.

Bibliografía complementaria

- Altera, Basic Computer System for the Altera DE0-Nano Board, Altera Corporation - University Program, 2012; ftp://ftp.intel.com/pub/fpgaup/pub/Intel_Material/12.1/Computer_Systems/DE0-Nano/DE0-Nano_Basic_Computer.pdf
- Intel, Introduction to Altera Nios II soft processor, Intel Corporation - University Program, 2019; https://ftp.intel.com/Public/Pub/fpgaup/pub/Teaching_Materials/current/Tutorials/Nios2_introduction.pdf
- Altera, Using the SDRAM Memory on Altera's DE2 Board with Verilog Design, Altera Corporation - University Program, 2009 ; http://people.ece.cornell.edu/land/courses/ece5760/DE2/tut_DE2_sdram_verilog.pdf
- Altera, Nios II Processor Reference Handbook, Altera Corporation, 2014; https://www.intel.com/content/dam/altera-www/global/ja_JP/pdfs/literature/hb/nios2/n2cpu_nii5v1.pdf

Anexo 1.

```
/*********************************************************************************
* lab2_part1_2_3_main.s
*
* Programa principal de la Práctica 2 de AC
*
* Inicializa el controlador Timer de DE0-Nano
* Inicializa y activa el sistema de interrupciones del procesador Nios II
* Ejecuta un bucle Fibonacci y muestra el número de intervalos de 33 ms en el terminal de Altera Monitor Program
*
* Subrutinas: PRINT_JTAG (lab2_part1_2_3_JTAG.s), FIBONACCI (lab2_part1_2_3_fibo.s),
*
*********************************************************************************/
.equ ITERACIONES, 500000
.text /* empieza el código ejecutable */
.global _start

_start:
        /* se inicializa el puntero del stack */
        movia sp,  0x007FFFFC   /* stack comienza en la última posición de memoria de la SDRAM */
        movia r16, 0x10002000   /* dirección base del sistema Timer interno */

        /* se inicciliza el tiempo del intervalo en el que el Timer genera una interrupción para análisis de prestaciones*/
        movia r12, 0x190000     /* 1/(50 MHz) x (0x190000) = 33 milisegundos */
        sthio r12, 8(r16)       /* guarda la mitad interior de la palabra del valor inicial del Timer */
        srli  r12, r12, 16      /* desplaza el valor 16 bits a la derecha */
        sthio r12, 0xC(r16)     /* guarda la mitad superior de la palabra del valor inicial del Timer */

        /* se inicializa el Timer, habilitando sus interrupciones */
        movi  r15, 0b0111       /* START = 1, CONT = 1, ITO = 1 */
        sthio r15, 4(r16)

        /* se habilita el sistema de interrupciones del procesador Nios II */
        movi  r7, 0b011   /* se inicializa la máscara de bits de interrupciones para el nivel 0 (interval */
        wrctl ienable, r7  /* Timer) y nivel 1 (pushbuttons) */
        movi  r7, 1
        wrctl status, r7    /* se activan las interrupciones del Nios II */

        movia r14, ITERACIONES      /* inicializa el contador de iteraciones Fibonacci */
        addi  r17, r0, 0            /* inicializa el contador de intervalos del programa "r17" */

LOOP:   beq   r14, r0, END          /* se ejecuta el bucle Fibonacci */
        call  FIBONACCI
        addi  r14, r14, -1
        br    LOOP

END:    movi  r7, 0
        wrctl status, r7        /* se desactiva el procesamiento de interrupciones en el Nios II */

        call PRINT_JTAG         /* se muestra el número de intervalos de 33 ms en el terminal de AMP */

IDLE:   br   IDLE               /* termina el programa principal */

.data
.global CONTADOR
CONTADOR:
        .skip 4                 /* posición de memoria que guarda el contador de intervalos del Timer */

.end
```

Anexo 2.

```
/*****************************************************************************
* lab2_part1_2_3_fibo.s
*
* Subrutina: Ejecuta el cómputo de la Serie Fibonacci para 8 números
*
* Llamada desde: lab2_part1_2_3_main.s
*
*****************************************************************************/
.text
.global FIBONACCI
FIBONACCI:
        subi sp, sp, 24             /* reserva de espacio para el Stack */
        stw r4, 0(sp)
        stw r5, 4(sp)
        stw r6, 8(sp)
        stw r7, 12(sp)
        stw r8, 16(sp)
        stw r9, 20(sp)

        movia   r4, N               /* r4 apunta N */
        ldw     r5, (r4)            /* r5 es el contador inicializado con N */
        addi    r6, r4, 4           /* r6 apunta al primer números Fibonacci */
        ldw     r7, (r6)            /* r7 contiene el primer número Fibonacci */
        addi    r6, r4, 8           /* r6 apunta al primer números Fibonacci */
        ldw     r8, (r6)            /* r7 contiene el segundo número Fibonacci */
        addi    r6, r4, 0x0C        /* r6 apunta al primer número Fibonacci resultado */
        stw     r7, (r6)            /* Guarda el primer número Fibonacci */
        addi    r6, r4, 0x10        /* r6 apunta al segundo número Fibonacci resultado */
        stw     r8, (r6)            /* Guarda el segundo número Fibonacci  */
        subi    r5, r5, 2           /* Decrementa el contador en 2 números ya guardados */
LOOP:
        beq     r5, r0, STOP        /* Termina cuando r5 = 0 */
        subi    r5, r5, 1           /* Decrement the counter */
        addi    r6, r6, 4           /* Increment the list pointer          */
        add     r9, r7, r8          /* suma dos número precedentes */
        stw     r9, (r6)            /* guarda el resultado */
        mov     r7, r8
        mov     r8, r9
        br      LOOP
STOP:
        ldw r4, 0(sp)
        ldw r5, 4(sp)
        ldw r6, 8(sp)
        ldw r7, 12(sp)
        ldw r8, 16(sp)
        ldw r9, 20(sp)
        addi sp, sp, 24             /* libera el stack reservado */
        ret
.data
N:
        .word 8                     /* Números Fibonacci */
NUMBERS:
        .word   0, 1                /* Primeros 2 números */
RESULT:
        .skip   32                  /* Espacio para 8 números de 4 bytes */
.end
```

Anexo 3.

```
/*********************************************************************************
* subrutina: lab2_part1_2_3_interrupts.s
*
* El programa AMP (Altera Monitoro Program) sitúa automáticamente la sección ".reset"
* en la dirección de memoria del reset que se especifica en la configuración del NIOS II
* que se determina con SOPC Builder.
* "ax" se necesita para indicar que esta sección se reserva y ejecuta
*/

.section .reset, "ax"
movia r2, _start
jmp r2 /* salta al programa principal */

/*********************************************************************************
* El programa AMP (Altera Monitor Program) sitúa automáticamente la sección ".exceptions"
* en la dirección de memoria del reset que se especifica en la configuración del NIOS II
* que se determina con SOPC Builder.
* "ax" se necesita para indicar que esta sección se reserva y ejecuta
*
* Subrutinas: INTERVAL_TIMER_ISR (lab2_part1_2_3_excepciones.s)
*/

.section .exceptions, "ax"
.global EXCEPTION_HANDLER

EXCEPTION_HANDLER:
        subi sp, sp, 16     /* reserva el Stack */
        stw et, 0(sp)
        rdctl et, ctl4
        beq et, r0, SKIP_EA_DEC /* interrupción no es externa */
        subi ea, ea, 4              /* debe decrementarse ea en 1 instrucción */

/* para interrupciones externas, de forma tal que */
/* la instrucción interrumpida se ejecutará después de eret (Exception RETurn) */
SKIP_EA_DEC:
        stw ea, 4(sp)               /* guardar registros en el Stack */
        stw ra, 8(sp)               /* se requiere si se ha usado un call */
        stw r22, 12(sp)
        rdctl et, ctl4
        bne et, r0, CHECK_LEVEL_0 /* la excepción es una interrupción externa */

NOT_EI: /* excepción para instrucciones no implementadas o TRAPs */
        br END_ISR

CHECK_LEVEL_0: /* Timer dispone de interrupciones de Level 0 */
        call INTERVAL_TIMER_ISR
        br END_ISR

END_ISR:
        ldw et, 0(sp)               /* restaurar valores previos de registros */
        ldw ea, 4(sp)
        ldw ra, 8(sp)
        ldw r22, 12(sp)
        addi sp, sp, 16
eret
.end
```

Anexo 4.

```
/*****************************************************************************
* lab2_part1_2_3_excepciones.s
*
* Subrutina que aumenta un contador de intervalos del Timer
*
* LLamada desde: lab2_part1_2_3_interrupts.s
*
*****************************************************************************/

.extern CONTADOR
.global INTERVAL_TIMER_ISR
INTERVAL_TIMER_ISR:
        subi sp, sp, 8              /* reserva de espacio en el stack */
        stw r10, 0(sp)
        stw r11, 4(sp)

        movia r10, 0x10002000   /* direccion base del Timer */
        sthio r0, 0(r10)            /* inicializa a 0 la interrupción */

        movia r10, CONTADOR  /* dirección base del contador de intervalos del Timer */
        ldw r11, 0(r10)
        addi r11, r11, 1            /* suma el contador de intervalos Timer */
        stw r11, 0(r10)

        ldw r10, 0(sp)
        ldw r11, 4(sp)
        addi sp, sp, 8              /* libera el stack */

        ret
.end
```

Anexo 5.

```
/*****************************************************************
* fichero: lab2_part1_2_3_JTAG.s
* AC - Practica 2 - Eejrcicio 1
* Subrutinas relacionadas con la muestra de un caracter en la terminal
* parametros de entrada:
*                               r10 = valor ascii del caracter a procesar
* parametros de salida: ninguno
*****************************************************************/

.extern   CONTADOR /* variable definida en prog principal */

/*
Subrutina: PRINT_JTAG
Muestra en terminal JTAG de AMP el contenido de la posición de memoria externa CONTADOR
*/
.global PRINT_JTAG
PRINT_JTAG:

        subi      sp, sp, 24          /* gestion de pila */
        stw       r2,  4(sp)
        stw       r3,  8(sp)
        stw               r4, 12(sp)
        stw          r10, 16(sp)
        stw          r17, 20(sp)
        stw       ra, 24(sp)

        movia     r3, TEXTO
        call      ESCRIBE_TEXTO_JTAG          /* escribe en terminal JTAG texto fijo */

        movia     r17, CONTADOR          /* direccion base del contador de intervalos del Timer */
        ldw         r4, 0(r17)
        call      BCD                         /* r4= valor binario, r2= valor BCD */

        call      ESCRIBE_VALOR_JTAG          /* escribe en terminal JTAG valor BCD */

        movia     r3, TEXTO_FIN
        call      ESCRIBE_TEXTO_JTAG          /* escribe en terminal JTAG texto fijo */

        ldw         r2,  4(sp)                /* gestion de pila */
        ldw         r3,  8(sp)
        ldw         r4, 12(sp)
        ldw       r10, 16(sp)
        ldw       r17, 20(sp)
        ldw         ra, 24(sp)
        addi        sp, sp, 24

        ret

/*
Subrutina: ESCRIBE_TEXTO_JTAG
escribe una cadena de caracteres por terminal JTAG
        parametros:         r3, puntero de la string
*/
.global ESCRIBE_TEXTO_JTAG
ESCRIBE_TEXTO_JTAG:
        subi        sp, sp, 12
        stw   r3,  4(sp)
        stw       r10,  8(sp)
```

```
        stw   ra, 12(sp)

BUC:
        ldb     r10, 0(r3)              /* carga 1 byte desde dirección de la cadena de caracteres */
        beq     r10, r0, CON    /* si lee un 0, significa que ha llegado al final de la cadena y sale bucle
*/
        call    ESCRIBIR_JTAG           /* subrutina que muestra el byte por JTAG-UART */
        addi    r3, r3, 1               /* siguiente byte */
        br      BUC                     /* cierra el bucle */
CON:
        ldw     r3,  4(sp)
        ldw      r10,  8(sp)
        ldw     ra, 12(sp)
        addi    sp, sp, 12

        ret

/*
Subrutina: ESCRIBE_VALOR_JTAG
escribe un valor en BCD por terminal JTAG
        parametros:     r2, valor BCD
*/
.global ESCRIBE_VALOR_JTAG
ESCRIBE_VALOR_JTAG:
        subi    sp, sp, 16
        stw     r2,  4(sp)
        stw     r4,  8(sp)
        stw             r10, 12(sp)
        stw     ra, 16(sp)

        addi    r4, r0, 8               /* contador de 8 nibles BCD */
VALOR:
        andhi   r10, r2, 0xf000 /* extrae 4 bits BCD más significativos*/
        srli    r10, r10, 28    /* resultado r10 se desplaza a dcha 28 bits */
        addi    r10, r10, 0x30  /* suma 0x30: BCD -> ASCII */
        call    ESCRIBIR_JTAG           /* muesra ASCII */
        subi    r4, r4, 1               /* contador de nible -- */
        slli    r2, r2, 4               /* siguiente nible BCD */
        bne     r4, r0, VALOR

        ldw     r2,  4(sp)
        ldw     r4,  8(sp)
        ldw     r10, 12(sp)
        ldw     ra, 16(sp)
        addi    sp, sp, 16

        ret

.global ESCRIBIR_JTAG
ESCRIBIR_JTAG:
        subi     sp, sp, 12     /* los registros usados se guardan en la pila */
        stw      r3,  4(sp)
        stw     r22,  8(sp)
        stw      ra, 12(sp)

        movia   r22, 0x10001000         /* direccion base de puerto JTAG */

otraVEZ:        /* encuesta: comprobar si hay espacio para escribir */
        ldwio   r3, 4(r22)              /* lee registro del puerto JTAG-UART */
        andhi   r3, r3, 0xffff  /* se seleccionan los 16 bits más significativos */
```

```
        beq     r3, r0, otraVEZ    /* ¿WSPACE=0? */

WRT:    /* envia el caracter escribiendo en JTAG-UART */
        stwio   r10, 0(r22)

FIN:    ldw       r3, 4(sp)                /* recuperar los registros de la pila y retornar */
        ldw      r22, 8(sp)
        ldw       ra, 12(sp)
        addi      sp, sp, 12

        ret

/*
* Zona de datos
*/
TEXTO:
.asciz   "\n\nIntervalos de tiempo que el programa necesita= "
TEXTO_FIN:
.asciz   "\n\nFin del programa "

.end
```

Anexo 6.

```
/*****************************************************************************
* lab2_part1_2_3_BCD.s
*
* Subrutina: transforma código binario en BCD
*
* LLamada desde: lab2_part1_2_3_JTAG.s
* Subrutina: DIV (lab2_part1_2_3_div.s)
*
* argumentos: r4= valor binario
* resultados: r2= valor BCD
*
*****************************************************************************/

.text
.global BCD
BCD:
        subi sp, sp, 24     /* reserva de memoria en el Stack */
        stw r3, 0(sp)
        stw r4, 4(sp)
        stw r5, 8(sp)
        stw r6, 12(sp)
        stw r10, 16(sp)
        stw r31, 20(sp)     /* por posible llamada anidada */

        beq r4, r0, END   /* si binario == 0 goto END */

        addi r5, r0, 10     /* r5 = 10 para dividir BCD */

        add r6, r0, r0      /* i = 0 */
        add r10, r0, r0     /* r10 = 0 */

LOOP2: bge r0, r4, END    /* while valor binario > 0 */

        call DIV            /* llama a division con r4 = dividendo, r5 = divisor; devuelve r3= cociente, r2= resto */
        sll r2, r2, r6      /* desplaza el resultado 4 bits a la izquierda excepto el primer número */
        or r10, r10, r2     /* acumula el resultado en r10 */
        addi r6, r6, 4      /* actualiza r6 += 4 */

        bgt r5, r3, END   /* si cociente < 10 goto END */
        add r4, r3, r0      /* r4 = cociente anterior */

        jmpi LOOP2          /* si cociente >= 10 goto LOOP2 */

END:    sll r3, r3, r6      /* desplaza el cociente final varios 4 bits a la izquierda */
        or r10, r10, r3     /* acumula el resultado en r10 */
        add r2, r10, r0     /* pone el resultado en el registro de salida r2 */

        ldw r3, 0(sp)
        ldw r4, 4(sp)
        ldw r5, 8(sp)
        ldw r6, 12(sp)
        ldw r10, 16(sp)
        ldw r31, 20(sp)
        addi sp, sp, 24     /* libera el stack reservado */

        ret
.end
```

Anexo 7.

```
/*****************************************************************************
* lab2_part1_2_3_div.s
*
* División entera para el NIOS II que se requiere cuando el procesador no dispone
* del hardware de un divisor
*
* Referencia:
* http://stackoverflow.com/questions/938038/assembly-mod-algorithm-on-processor-with-no-division-operator
*
* Llamada desde: lab2_part1_2_3_JTAG.s
*
* argumentos: r4= dividendo, r5= divisor
* resultados: r2= resto, r3= cociente
*
*****************************************************************************/

.text
.global DIV
DIV:
        subi sp, sp, 16     /* reservar espacio en el Stack */
        stw r6, 0(sp)
        stw r7, 4(sp)
        stw r10, 8(sp)
        stw r11, 12(sp)

        beq r5, r0, END             /* si divisor == 0 goto END */

EMPIEZA:add r2, r4, r0              /* resto = dividendo */
        add r6, r5, r0              /* r6 = next_multiple = divisor */
        add r3, r0, r0              /* cociente = 0 */

LOOP:   add r7, r6, r0              /* r7 = multiple = next_multiple */
        slli r6, r7, 1              /* next_multiple = left_shift(multiple,1) */

        sub r10, r2, r6             /* r10 = resto - next_multiple */
        sub r11, r6, r7             /* r11 = next_multiple - multiple */

        blt r10, r0, LOOP2          /* si r10 < 0 goto LOOP2 */
        bgt r11, r0, LOOP           /* si r11 > 0 goto LOOP */

LOOP2: bgt r5, r7, END              /* while divisor <= multiple */
        slli r3, r3, 1              /* cociente << 1 */
        bgt r7, r2, DESPLAZA        /* si multiple <= resto */
        sub r2, r2, r7              /* then resto = resto - multiple */
        addi r3, r3, 1              /*          cociente += 1 */

DESPLAZA:
        srli r7, r7, 1              /* multiple = right_shift(multiple, 1) */
        jmpi LOOP2

END:    ldw r6, 0(sp)
        ldw r7, 4(sp)
        ldw r10, 8(sp)
        ldw r11, 12(sp)
        addi sp, sp, 16             /* libera el stack reservado */

        ret
.end
```