Práctica 3. Utilización de la Memoria de un Computador

El principal objetivo de esta práctica consiste en manejar distintos elementos de la estructura del Computador Básico que se configura en la placa DE2 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 DE2 son:

- El nivel de la memoria principal que se puede implementar con DE2 utilizando dos tecnologías electrónicas distintas:
  - Con circuitos electrónicos de tipo SDRAM que se encuentran en el exterior del chips donde se encuentra el procesador.
  - Con circuitos electrónicos de tipo SRAM. En DE2 existen dos dispositivos distintos SRAM que se utilizan en esta práctica para implementar la memoria principal:
    - 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.
    - Los *chips SRAM* (SRAM chip) que es una memoria tanto externa al procesador como externa al chip donde se encuentra el procesador.
- El nivel de la memoria cache :
  - Con circuitos electrónicos de tipo SRAM que se encuentran en el mismo chip donde se encuentra el procesador (on-chip).

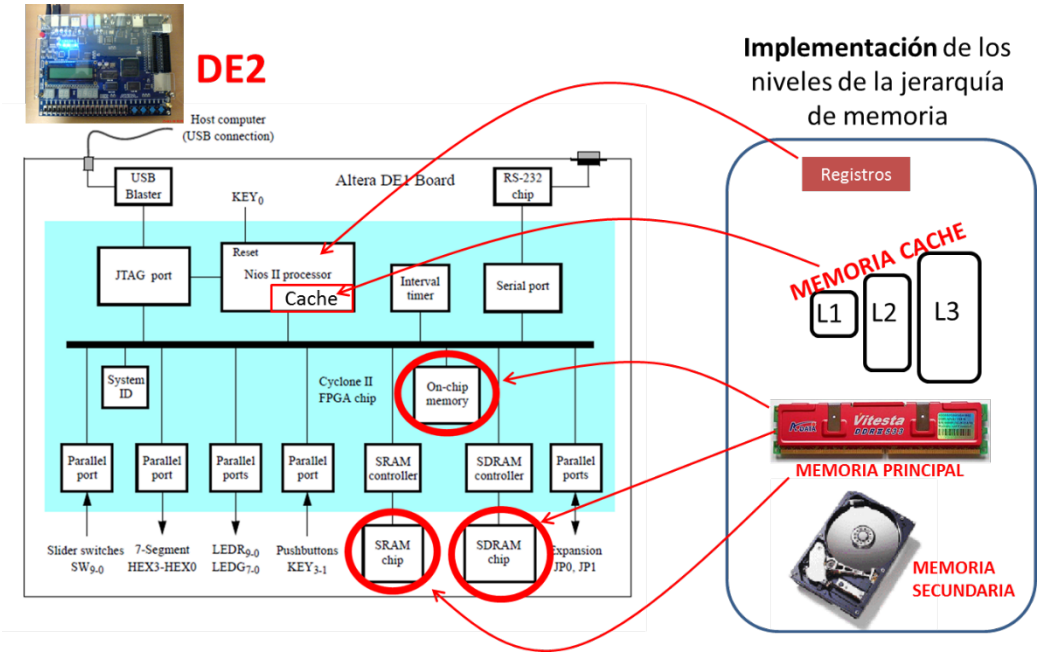

Figura 1. Implementación de los niveles de la Jerarquía de Memoria de la estructura del Computador Básico de la placa DE2

Adicionalmente, se utilizarán dos versiones hardware del procesador NIOS II y que se denominan **NIOS II/e** (economy) y **NIOS II/f** (fast) respectivamente. Cada tipo de procesador y memoria proporciona un nivel de prestaciones en tiempo de ejecución de los programas que es distinto. En esta práctica se realizará la evaluación del tiempo de ejecución de un mismo programa denominado Fibonacci cuando es ejecutado con distintas configuraciones de la estructura del Computador Básico en DE2. 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. La metodología de prácticas consiste en realizar cuatro actividades con la placa DE2 y el entorno software 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 actividad consiste en **medir el tiempo real de cómputo** del programa Fibonacci utilizando la memoria SDRAM de la placa DE2, el procesador NIOS II/e, y el temporizador Timer que se utilizó en la Práctica 2. 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 display de 7-segmentos muestra el número de intervalos completos de 33 ms que han pasado, como se puede observar en la Figura 2.

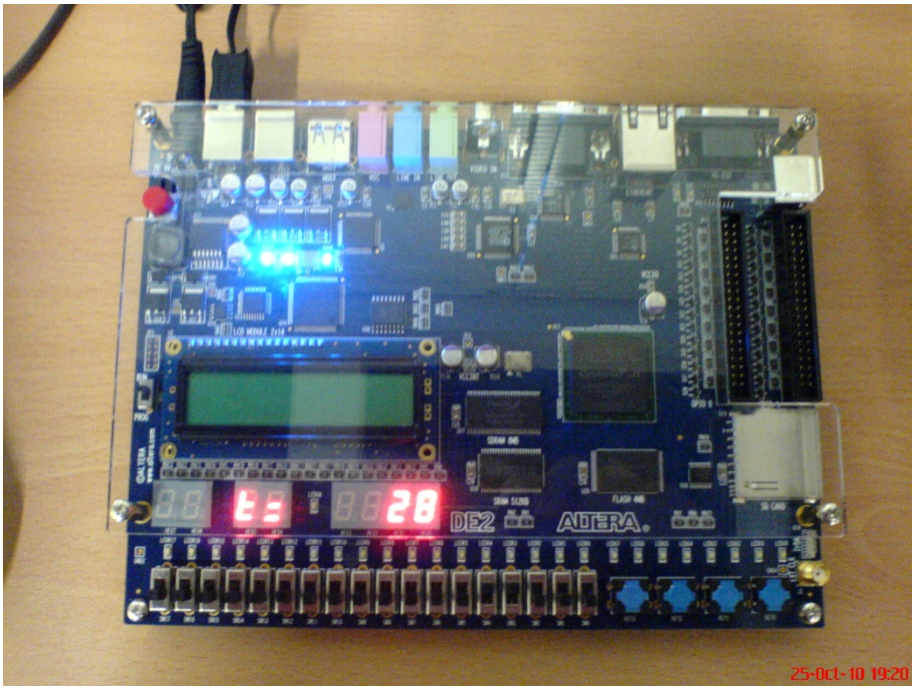

Figura 2. Resultado final de la ejecución del programa utilizado para medir el tiempo de ejecución de los programas.

Los programas ensamblador involucrados en la práctica son:

- `lab3_part1_main.s`: programa principal (ver Anexo 1).
- `lab3_part1_fibo.s`: rutina benchmark de cómputo de la que se pretende medir su tiempo de cómputo (ver Anexo 2).
- l`ab3_part1_interrupts.s`: rutina que se ejecuta cuando se activa la interrupción del Timer (ver Anexo 3).
- `lab3_part1_excepciones.s`: rutina que se llama desde lab3_part1_interrupts.s y que gestiona el contador de tiempo de ejecución (ver Anexo 4).
- `lab3_part1_print.s`: rutina que se llama desde lab3_part1_main.s para mostrar en el display 7-segmentos el número de intervalos que han pasado hasta la finalización de la ejecución del programa benchmark (ver Anexo 5).
- `lab3_part1_BCD.s`: rutina que se llama desde lab3_part1_print.s para transformar un código binario en BCD (ver Anexo 6).
- `lab3_div.s`: rutina que se llama desde lab3_part1_BCD.s para realizar la división entera (ver Anexo 7).

El flujo del programa 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 `lab3_part1_main.s`).

La rutina de servicio de interrupciones permite mantener un contador de eventos en una posición de memoria denominada CONTADOR (ver fichero `lab3_part1_excepciones.s`). De forma concurrente al contaje de tiempo por parte del Timer, el programa principal ejecuta el bucle Fibonacci un número de veces que es indicado por la constante ITERACIONES (ver fichero `lab3_part1_main.s`). Cuando termina este bucle, se consulta el contenido de la posición de memoria CONTADOR y su valor se visualiza por el display 7-segmentos HEX de la placa DE2 (ver fichero `lab3_ part1_print.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 de 7-segmentos. 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 `lab3_part1_print.s`).

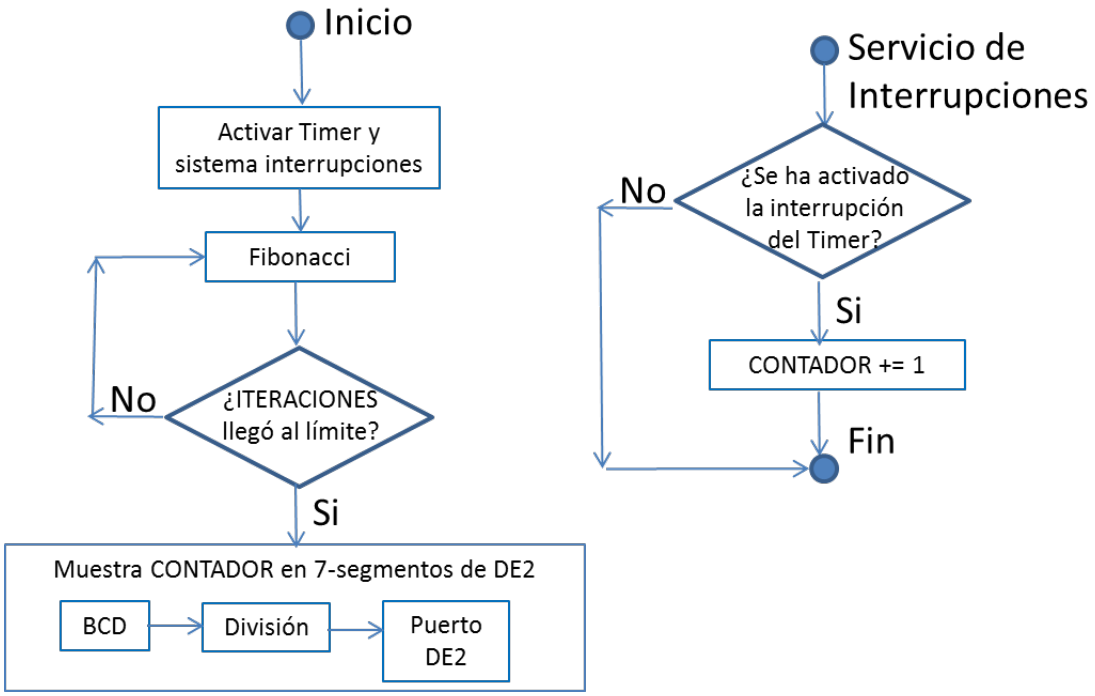

Figura 3. Diagrama de flujo del programa de la Práctica 3.

Metodología de prácticas

Iniciar un nuevo proyecto en el entorno software Altera Monitor Program (AMP) de la misma forma que en las prácticas anteriores, seleccionando la configuración Basic Computer, e incluyendo los siete ficheros en ensamblador 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; start offset in device (hex) = 400

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

A continuación, cargar la configuración en la placa ED2:

- 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 la parte derecha del display HEX constituido por cuatro 7-segmentos. 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 `lab3_part1_main.s` tiene declarada e inicializada una constante: ITERACIONES = 500000.

Cambiar ITERACIONES a 100000 en el fichero `lab3_part1_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 HEX.

Pregunta

¿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 (economy)

El objetivo de esta actividad consiste en **medir el tiempo real de cómputo** del programa Fibonacci utilizando la memoria que se encuentra dentro del chip donde también se encuentra el procesador NIOS II/e (denominada "on-chip") en vez de utilizar la memoria SDRAM externa al procesador y que se encuentra en la placa DE2.

Metodología de prácticas

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

- Settings > System settings > Memory settings >

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

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

  .data – memory device = onchip memory (0x9000000 – 0x9001 FFF)

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

En el código fuente del programa (fichero `lab3_part1_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 DE2. A continuación, ejecutar el programa y anotar en la Tabla 1 el valor del tiempo de ejecución que se muestra en el display HEX de 7-segmentos.

Desconectar la monitorización de la placa DE2:

- Actions > Disconnect

y posteriormente seleccionar el espacio de direccionamiento “avalon_sram_slave” para la sección de datos (.data section) del programa:

- Settings > System settings > Memory settings > …

A continuación, realizar los mismos pasos indicados en esta segunda actividad para la memoria on-chip y anotar en la Tabla 1 el valor del tiempo de ejecución que se muestra por el display HEX de 7-segmentos. Adicionalmente, calcula el Speed-up de las distintas configuraciones hardware “procesador + tecnología de memoria” que se han usado para obtener los tiempos de ejecución a anótalo en la Tabla 1.

Pregunta

¿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		1X
NIOS II/e + memoria SRAM avalon		
NIOS II/e + memoria on-chip		
NIOS II/f + memoria SDRAM		
NIOS II/f + memoria SRAM avalon		
NIOS II/f + memoria on-chip		

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, que corresponde a la versión del procesador NIOS II que proporciona mayor nivel de prestaciones temporales ya que implementa una ruta de datos segmentada de seis etapas, memoria cache de datos e instrucciones de primer

nivel, y predicción dinámica de saltos. Para ello, se procederá a cambiar la configuración de la placa DE2 de la forma que se indica a continuación.

Metodología de prácticas

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 (`.sopcinfo`) y el de configuración de la FPGA de la placa Altera (`.sof`) según se puede observar en la Figura 4. Estos dos ficheros son distintos dependiendo del tipo de placa DE2 que se esté utilizando (DE2 o DE2-115). Consultar la Tabla 2 para saber exactamente qué ficheros son los que hay que seleccionar.

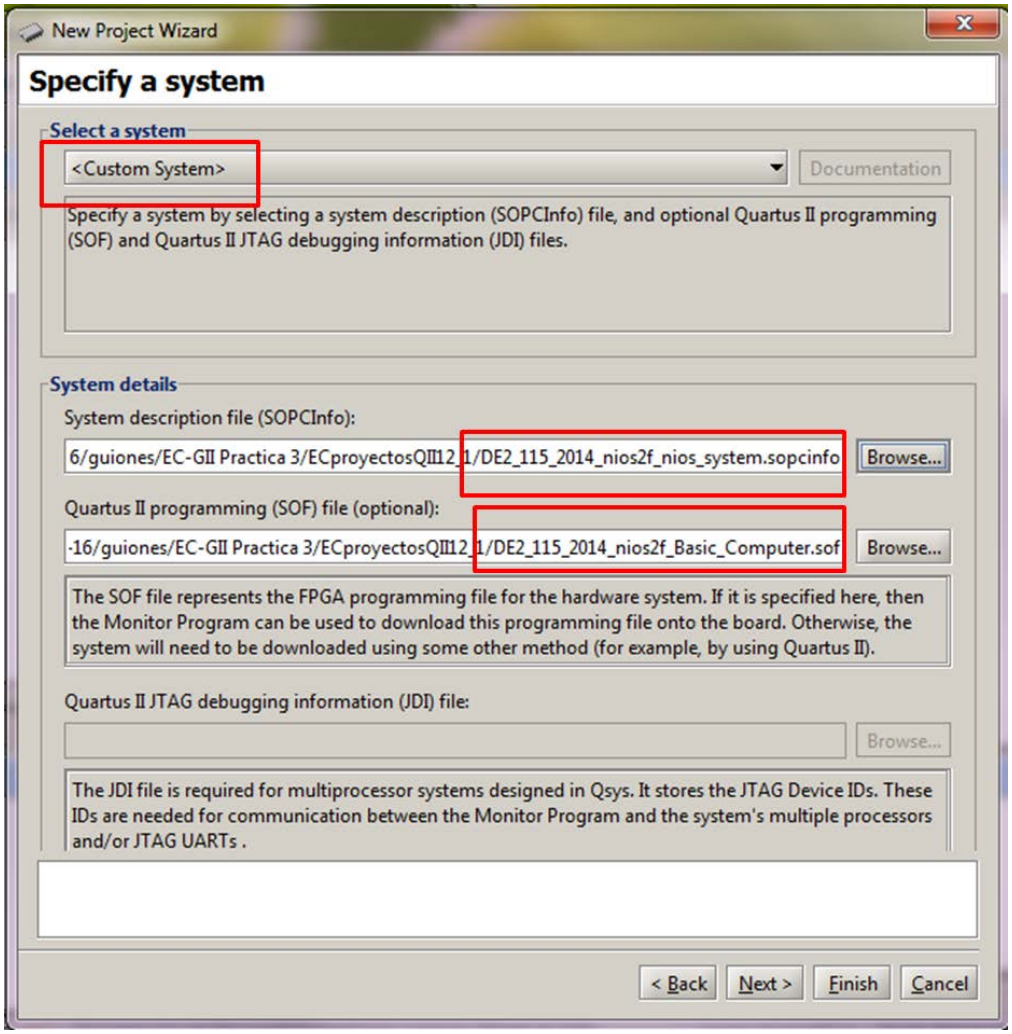

Figura 4. Parte del proyecto AMP donde se selecciona una estructura de computador basada en el procesador NIOS II / f.

Tabla 2. Ficheros de configuración de la placa DE2

Placa	Fichero SOPCINFO	Fichero SOF
DE2	DE2_2014_nios_f_system.sopcinfo	DE2_2014_nios_f_Basic_Computer.sof
DE2-115	DE2_115_2014_nios_f_system.sopcinfo	DE2_115_2014_nios_f_Basic_Computer.sof

Nota: estos ficheros de configuración están disponibles en el fichero comprimido lab3.rar (ver "Material de la Práctica 3" en la página Moodle)

Adicionalmente, se debe seleccionar cualquier tipo de espacio de direccionamiento de las tres alternativas que se pueden elegir con la restricción de que la parte de código y la de datos comience a partir de la dirección 0x400: on-chip, SRAM avalon, SDRAM.

- 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 display HEX de 7-segmentos 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 DE2. Finalmente, responder a las siguientes preguntas.

Pregunta 1

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

Pregunta 2

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

Pregunta 3

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

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

Para evaluar la memoria cache del procesador NIOS II/f utilizaremos un programa sencillo que recorra un vector de bytes. Los pasos a seguir son los siguientes:

- Reducir el número de iteraciones del programa principal de 500000 a 50000 (fichero `lab3_part1_main.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 `lab3_part1_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*.
- De la misma forma que en las actividades anteriores, medir el tiempo de ejecución del programa que es invertido en recorrer los elementos de un vector 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), y de ocho en ocho (**P** = 8). El número de elementos realmente accedidos (**E**) del vector V será para cada una de estas pautas: **E** = {128, 256, 512, 1024, 2048, 4096}. Para asegurar este número de elementos recorridos se modificará la instrucción: *movi r5, **X***; donde ***X*** representará al número que resulta de la multiplicación de la pauta y el número de elementos recorridos del vector V: **X = P × E**. Por ejemplo, para 128 elementos recorridos (**E** = 128) de dos en dos (**P** = 2), el valor de ***X*** es 256 (***X*** = 2 × 128); y la instrucción afectada por ***X*** que habrá que modificar es: *movi r5, **256***.
- Rellenar una tabla 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 > System information file + browse > `<file>.sopcinfo`

4) Settings > System Settings > Quartus II Programming File + browse > `<file>.sof`
5) Actions > Download System

CAMBIA-DATOS:

6) Modificar fichero del programa: `lab3_part1_fibo.s` para establecer **X** y **P**
7) Compile
8) Load
9) Ejecutar el programa y esperar a que salga el tiempo en el display 7-segmentos y apuntarlo en la Tabla 3

SEGUIR CAMBIA-DATOS

SEGUIR CAMBIA-CACHE

Tabla 3. Medidas de tiempos de ejecución

	Número de elementos recorridos del vector (E)					
Pauta de salto (P)	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: $AMAT = t_{acierto} + FrecuenciaFallos \times Penalización$.

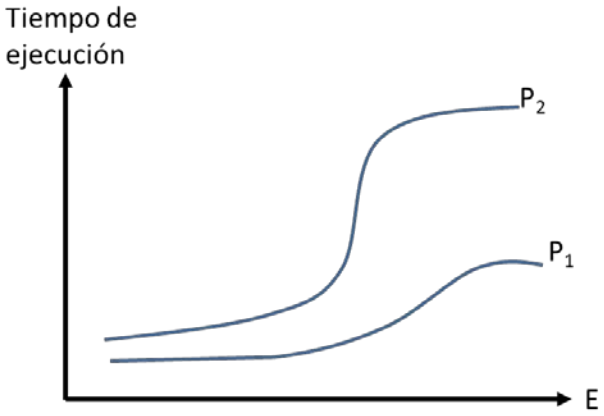

Figura 5. Gráfica donde se representa la relación entre el tiempo de ejecución del programa y el número de elementos accedidos del vector V.

Bibliografía complementaria

- Altera, Basic Computer System for the Altera DE2 Board, Altera Corporation - University Program, 2010; ftp://ftp.altera.com/up/pub/Altera_Material/9.0/Computer_Systems/DE2/DE2_Basic_Computer.pdf, accedido: 5/9/15
- Altera, Introduction to Altera NIOS II soft processor, Altera Corporation - University Program, 2012; ftp://ftp.altera.com/up/pub/Altera_Material/12.0/Tutorials/Nios2_introduction.pdf, accedido: 5/9/15
- 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, accedido: 5/9/15
- Altera, NIOS II Processor Reference Handbook, Altera Corporation, 2015; https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/nios2/n2cpu_nii5v1.pdf, accedido: 5/9/15

Anexo 1.

```
/**********************************************************************************
* lab3_part1_main.s
*
* Programa principal de la Práctica 3 de EC
*
* Inicializa el sistema Timer de DE2
* 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 HEX de la placa DE2
*
* Subrutinas: PRINT_HEX (lab3_part1_print.s), FIBONACCI (lab3_part1_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 inicializa 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_HEX          /* se muestra el número de intervalos de 33 ms en el display HEX de DE2 */

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.

```
/*********************************************************************************
* lab3_part1_fibo.s
*
* Subrutina: Ejecuta el cómputo de la Serie Fibonacci para 8 números
*
* Llamada desde: lab3_part1_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: lab3_part1_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 (lab3_part1_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.

```
/*****************************************************************************
* lab3_part1_excepciones.s
*
* Subrutina que aumenta un contador de intervalos del Timer
*
* LLamada desde: lab3_part1_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.

```
/*****************************************************************************
* lab3_part1_print.s
*
* Subrutina: Muestra en el display HEX de DE2 el contenido de la posición de memoria externa CONTADOR
*
* Llamada desde: lab3_part1_main.s
* Subrutina: BCD (lsb3_part1_BCD.s)
*
*****************************************************************************/

.extern CONTADOR
.text
.global PRINT_HEX
PRINT_HEX:
        subi sp,  sp, 28            /* reserva de espacio en el Stack */
        stw  r17, 0(sp)
        stw  r18, 4(sp)
        stw  r19, 8(sp)
        stw  r20, 12(sp)
        stw  r21, 16(sp)
        stw  r22, 20(sp)
        stw  r23, 24(sp)
        stw  r4,  28(sp)
        stw  r2,  32(sp)

        add  r23, r0, r0            /* inicializa r23 = 0 */

        movia r20, 0x10000020   /* dirección base del periférico HEX3_HEX0 */
        movia r21, 0x10000030   /* dirección base del periférico HEX7_HEX4 */

        movia r17, CODIGO_ESTATICO /* inicializa el puntero de datos */
        ldw   r18, 0(r17)           /* carga el código 7-segmentos correspondiente */
        stwio r18, 0(r21)           /* envía R18 que tiene el código "t=" a HEX7 ... HEX4 */

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

LOOP:   beq   r2, r0, END           /* si valor BCD=0, goto END */
        andi  r17, r2, 0xF          /* extrae los 4 bits menos significativos */

        slli  r18, r17, 2           /* multiplica por 4 para calcular el desplazamiento de palabras en la zona de datos*/
        movia r17, CODIGOS_HEX      /* inicializa el puntero de datos */
        add   r18, r18, r17         /* suma el puntero de datos con el desplazamiento correspondiente al número a
mostrar en HEX */
        ldw   r19, 0(r18)           /* carga el código 7-segmentos correspondiente al valor de r18 */
        sll   r19, r19, r23         /* desplaza el código 8 bits x orden cifra decimal */
        or    r22, r22, r19         /* acumula en r22 */
        stwio r22, 0(r20)           /* envía R22 a HEX3 ... HEX0 */

        srli  r2, r2, 4             /* desplaza a la derecha 4 bits el valor BCD */
        addi  r23, r23, 8           /* actualiza r23 en 8 porque HEX utiliza 8 bits para cada valor */
        jmpi  LOOP:

END:
        ldw r17, 0(sp)
        ldw r18, 4(sp)
        ldw r19, 8(sp)
```

```
        ldw r20, 12(sp)
        ldw r21, 16(sp)
        ldw r22, 20(sp)
        ldw r23, 24(sp)
        addi sp, sp, 28             /* libera el stack reservado */

        ret
.data
CODIGOS_HEX:
/* códigos 7-segmentos de los primeros 10 números en binario
(ver DE2_Basic_Computer.pdf, pp.4)
        0: 0111111 = 0x3f
        1: 0000110 = 0x06
        2: 1011011 = 0x5B
        3: 1001111 = 0x4F
        4: 1100110 = 0x66
        5: 1101101 = 0x6D
        6: 1111101 = 0x7D
        7: 0000111 = 0x07
        8: 1111111 = 0x7F
        9: 1100111 = 0x67
*/
.word 0x3f,0x6,0x5b,0x4f,0x66,0x6d,0x7d,0x7,0x7f,0x67

CODIGO_ESTATICO: /* t=, t: 0111 0000, =: 0100 1000 (ver DE2_Basic_Computer.pdf, pp.4)*/
.word 0x7848

.end
```

Anexo 6.

```
/*****************************************************************************
* lab3_part1_BCD.s
*
* Subrutina: transforma código binario en BCD
*
* LLamada desde: lab3_part1_print.s
* Subrutina: DIV (lab3_part1_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.

```
/*******************************************************************************
* lab3_part1_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: lab3_part1_print.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
```