Computer Architecture - Lab Assignment 2

Performance evaluation of the memory hierarchy of a computer and reverse engineering of the data cache memory

The main objective of this practice consists of manipulate different elements of the architecture of the Basic Computer that it is configured in the DE0-Nano board and they are responsible for implementing various levels of what we call Memory Hierarchy. These levels and implementations of the memory hierarchy on DE0-Nano are the following:

- The level of the main memory can be implemented with DE0-Nano using two different electronic technologies:
  - With electronic circuits of type SDRAM that are located outside of the chip where is the processor (See Fig. 1).
  - With electronic circuits of type SRAM. In DE0-Nano exists one device SRAM that is used in this practice to implement the main memory (See Fig. 1). The on-chip memory is an external memory to the processor but that it is in the same chip where is located the processor which is called FPGA (Field Programmable Gate Array).
- The level of the cache memory:
  - This level is implemented with electronics circuits of type SRAM that are in the same chip where is located the processor (FPGA, see Fig. 1).

In addition, in this practice it will be use two hardware versions of the NIOS V processor that are called **Nios V/m** and **Nios V/g**, respectively. Each type of processor and memory cause that the times of the executions of programs be different.

In this practice it will be carry out the measure of time of execution of a same program called Fibonacci when it is executed with different configurations of the architecture of the Basic Computer in the DE0-Nano Board. These configurations are distinguished in the type of processor configured: Nios V/m or Nios V/g, as well as in the type of the implementation activated for the levels of the memory hierarchy: cache, main memory (SRAM or SDRAM).

The methodology of hands-on exercises consists of four activities with the DE0-Nano board and the software environment Nios V Command Shell. It will be requested to think about the results that you have experimentally obtained.

Part I. Access to SDRAM memory with Nios V/m processor

The objective of this first activity consists of measure the real time of computation of the Fibonacci program using the SDRAM memory of the DE0-Nano board, also of the Nios V/m, and the Timer which is an input/output device that joins to the hardware components mentioned (see Fig. 1).

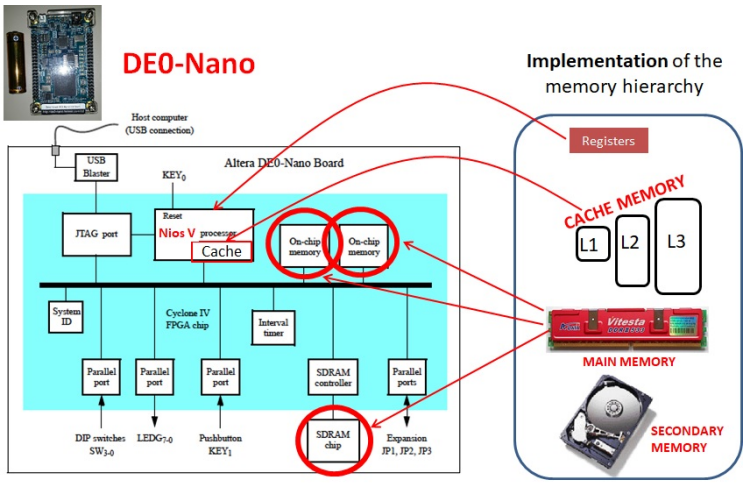

Figure 1: Implementation of the levels of the memory hierarchy integrated in the architecture of the Basic Computer of the DE0-Nano board.

During the execution of the Fibonacci program, the Nios V processor counts the number of intervals of 1321.20 ms that have passed. When the Fibonacci program finishes, the Nios V Command Shell terminal shows the number of complete intervals of 1321.20 ms that have passed, as you can see in Fig. 2.

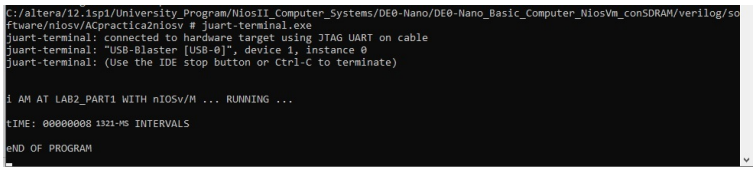

Figure 2: Visualization of the final result of the execution of the program used to measure the time of execution of the programs. The value of the number of 1321.20 *ms* intervals that is shown in the terminal.

A benchmark program will be used in all sections of this hands-on experience to measure execution time. The assembly source codes involved in this Part I of the lab assignment are in the following files:

- `lab2_part1_2_3_main.s`: main program.
- `lab2_part1_2_3_fibo.s`: benchmark routine of which aims to measure the time of computation.
- `excepcionTimer.s`: routine which is executed when the timer interruption has fired.
- `contador.s`: routine which is called from `excepcionTimer.s` that manages the counter of 1321.20 *ms* intervals.
- `escribir_JTAG.s`: routine which is called from `lab2_part1_2_3_main.s` to show in the terminal the number of intervals that have passed until the end of the execution of the benchmark program.
- `BCD.s`: routine which is called from `escribir_JTAG.s` to transform a binary code in BCD format.
- `DIV.s`: routine that is called from `BCD.s` to perform the integer division.

The flow of the benchmark program that we will use to measure the time of execution is show in Fig. 3. As we can see, the computer application activates the interrupts system of the timer in the main program (see `lab2_part1_2_3_main.s`).

The interrupt routine allows to maintain a counter of events in one position of memory called COUNTER (see excepcionTimer.s file). Each event consist of the indication that one interval of 1321.20 *ms* has passed since the previous interval of time. So, COUNTER records the numbers of 1321.20 *ms* intervals that have passed since the timer was activated.

Concurrently with the time measurement performed by the Timer, the main program executes the Fibonacci´s loop a number of times that is indicated by the constant called ITERATIONS (see lab2_part1_2_3_main.s

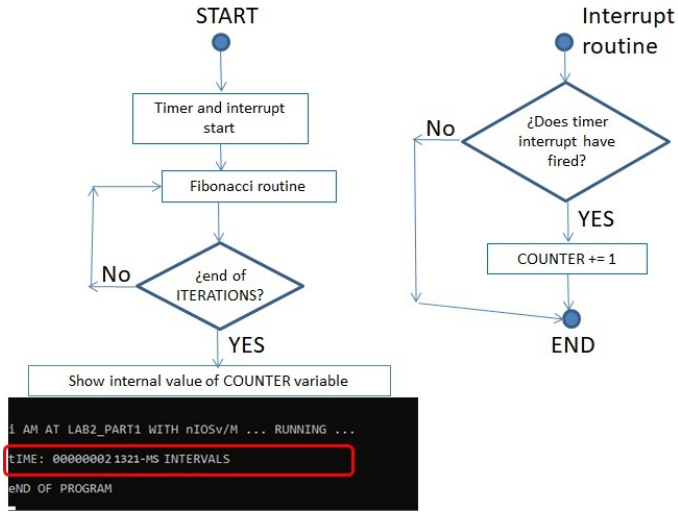

Figure 3: Flow diagram of the benchmark for Parts I, II and III.

file). When the loop finishes, the content of the position of the memory that saves COUNTER is consulted and its value is shown in the terminal (see JTAG.s file). For this last process, the binary code is transformed. This binary code represents the value of COUNTER in a number encoded in BCD and, then, in an ASCII code. In the transformation to BCD code, it is necessary to make one o various divisions, for which one routine is included for division because the Nios V/m processor has not the hardware to make division (see JTAG.s file).

Perform the following steps:

1. Turn on the power to the Terasic DE0-Nano board.
2. Open the Nios V Command Shell.

   To run an application program it is necessary to create a new BSP project.
3. Create a new BSP directory called for example: `lab2_bsp`.
4. Select a predesigned *System-on-Chip* File: `nios_system_22jul24.sopcinfo`. This file was obtained by using the *Platform Designer* tool. It integrates a Nios V/m soft processor, a 8 KB on-chip SRAM memory, a 32 KB SDRAM memory, in addition to an I/O controller for a timer that is available in the board.
5. Execute the following command in the Nios V Command Shell Window.

   ```
   $ cd lab2_bsp
   $ sh
   $ niosv-bsp.exe -c -t=hal -s=nios_system_22jul24.sopcinfo settings.bsp
   ```

   The output provides two files called: `settings.bsp` and `linker.x`.
6. Now, create a new directory for the RISC-V assembler program written in RISC-V assembly language and its building, for example: `lab2_bin`.
7. Copy the files `lab2_part1_2_3_main.s`, `lab2_part1_2_3_fibo.s`, `excepcionTimer.s`, `contador.s`, `escribir_JTAG.s`, `BCD.s`, `DIV.s` to this directory.
8. The source file `lab2_part1_2_3_main.s` contains the application program. This file specifies the starting point in the selected application program. The default symbol is start, which is used in the selected sample program.
9. Execute the following command in the Nios V Command Shell window.

```
$ cd lab2_bin
$ riscv32-unknown-elf-as.exe lab2_part1_2_3_main.s -alsg -o
lab2_part1_2_3_main.s.obj > lab2_part1_2_3_main.s.log
$ riscv32-unknown-elf-as.exe excepcionTimer.s -alsg -o excepcionTimer.s.obj >
excepcionTimer.s.log
$ riscv32-unknown-elf-as.exe escribir_jtag.s -alsg -o escribir_jtag.s.obj
> escribir_jtag.s.log $ riscv32-unknown-elf-as.exe contador.s -alsg -o
contador.s.obj > contador.s.log
$ riscv32-unknown-elf-as.exe DIV.s -alsg -o DIV.s.obj > DIV.s.log
$ riscv32-unknown-elf-as.exe BCD.s -alsg -o BCD.s.obj > BCD.s.log
$ riscv32-unknown-elf-as.exe lab2_part1_2_3_fibo.s -alsg -o
lab2_part1_2_3_fibo.s.obj > lab2_part1_2_3_fibo.s.log
```

The outputs are files called: `lab2_part1_2_3_main.s.obj`, `lab2_part1_2_3_fibo.s.obj`, `excepcionTimer.s.obj`, `contador.s.obj`, `escribir_JTAG.s.obj`, `BCD.s.obj`, `DIV.s.obj`.

10. Execute the following command for linking in the Nios V Command Shell window.

```
$ riscv32-unknown-elf-ld.exe -g -T ../lab2_bsp/linker.x -nostdlib
-e _start -u _start --defsym __alt_stack_pointer=0x08001F00 --defsym
__alt_stack_base=0x08002000 --defsym __alt_heap_limit=0x8002000 --defsym
__alt_heap_start=0x8002000 -o lab2_part1_2_3_main.elf lab2_part1_2_3_main.s.obj
lab2_part1_2_3_fibo.s.obj excepcionTimer.s.obj contador.s.obj
escribir_JTAG.s.obj BCD.s.obj DIV.s.obj
$ niosv-stack-report.exe -p riscv32-unknown-elf- lab2_part1_2_3_main.elf
```

The output is a file called: `lab2_part1_2_3_main.elf`.

11. Execute the following commands in the Nios V Command Shell window.

```
$ riscv32-unknown-elf-objdump.exe -Sdtx lab2_part1_2_3_main.elf >
lab2_part1_2_3_main.elf.objdump
```

The output is a file called: `lab2_part1_2_3_main.elf.objdump`.

Now, it is needed to download the soft SoC system associated with this program onto the DE0-Nano board. Make sure that the power to the DE0-Nano board is turned on.

12. Execute the following command in the Nios V Command Shell window.

```
$ jtagconfig.exe
```

The following message must appear in the Nios V Command Shell window. In the case the message does not show, repeat again the command.

```
1) USB-Blaster [USB-0]
020F30DD 10CL025(Y|Z)/EP3C25/EP4CE22
```

13. Execute the following command in the Nios V Command Shell window.

```
$ quartus_pgm.exe -c 1 -m JTAG -o "p;DE0_Nano_Basic_Computer_22jul24.sof@1"
```

Note the change in the state of the blue LEDs on the DE0-Nano board that correspond to LOAD and GOOD, which will blink as the circuit is being downloaded.

14. Having downloaded the `sof` configuration DE0-Nano Basic Computer into the FPGA chip on the DE0-Nano board, we can now load and run programs on this SoC computer. In the Nios V Command Shell window, execute the following command.

```
$ niosv-download.exe -g lab2_part1_2_3_main.elf
```

This command also run the program.

15. Observe the messages displayed on the Nios V Command Shell using the following command.

```
$ juart_terminal.exe
```

16. Register the value displayed for the number of 1321.20 *ms* time intervals that were needed to execute the benchmark. This measure is related with the soft configuration: *Nios V/m + SDRAM memory.* Fill in the Table 1 using this value. This execution time will be compared with the time that will be obtained in the next two parts of this lab assignment. Stop the execution of the sample program by typing "CTRL+c".

Table 1: Runtime measurements.

Soft processor + memory technology	**Execution time**	**Speed-up**
Nios V/m + SDRAM memory (Part I)		1x
Nios V/m + on-chip memory (Part II)		
Nios V/g + SDRAM memory (Part III)		
Nios V/g + on-chip memory (Part III)		

Question 1. Note that the file lab2_part1_2_3_main.s has declared and initialized a constant: ITERATIONS = 1,000,000. Complete the following steps and respond to the question shown at the end of this box.

- Change ITERATIONS to 4,000,000 in the lab2_part1_2_3_main.s file.
- Compile again the program
- Download the *.elf file into the board.
- Register the value displayed on the Nios V terminal.
- Obtain the speed-up.

Is this value a reasonable speed-up? Explain and justify your answer.

Part II. Access to SRAM on-chip memory with Nios V/m processor

The goal of this second part consists in measuring the execution time of the Fibonacci program using the SRAM on-chip memory that is located inside the FPGA chip, where the NIOS V/m soft processor is also integrate. The difference between this part and Part I is that the SRAM memory is used. SRAM memory was built using another technology. Additionally, the on-chip memory is located physically nearer the processor. This is a different situation with respect to using the external SDRAM memory of DE0-Nano board. Follow the following steps.

1. Change again the value for ITERATIONS in file `lab2_part1_2_3_main.s` to 1,000,000.
2. Change the immediate value for the `li` instruction at the begining of the `lab2_part1_2_3_main.s` file. This is the start address of the exception routine.

   lab2_part1_2_3_main.s

   ```
   li s2, 0x08000000
   ```

3. Modify the `linker.x` file changing the string `new_sdram_controller_0` with the string `Onchip_memory_SRAM` at the end of SECTION: `.exceptions`, `.text`, `.rodata`, `.rwdata` and `.bss`.
4. Call this new file for example: `linker_SRAM.x`.
5. Now, compile the file `lab2_part1_2_3_main.s` as explained in Part I.

   ```
   $ riscv32-unknown-elf-as.exe lab2_part1_2_3_main.s -alsg -o
   lab2_part1_2_3_main.s.obj > lab2_part1_2_3_main.s.log
   ```

6. Link the program as follows.

   ```
   $ riscv32-unknown-elf-ld.exe -g -T ../lab2_bsp/linker_SRAM.x -nostdlib
   -e _start -u _start --defsym __alt_stack_pointer=0x08001F00 --defsym
   __alt_stack_base=0x08002000 --defsym __alt_heap_limit=0x8002000 --defsym
   __alt_heap_start=0x8002000 -o lab2_part1_2_3_main.elf lab2_part1_2_3_main.s.obj
   lab2_part1_2_3_fibo.s.obj excepcionTimer.s.obj contador.s.obj
   escribir_JTAG.s.obj BCD.s.obj DIV.s.obj
   $ niosv-stack-report.exe -p riscv32-unknown-elf- lab2_part1_2_3_main.elf
   ```

7. Now, download and execute the program. Additionally, display the results of the program using the terminal software.

   ```
   $ niosv-download.exe -g lab2_part1_2_3_main.elf
   $ juart_terminal.exe
   ```

8. Register the value displayed for the number of 1321.20 *ms* time intervals that were needed to execute the benchmark. This measure is related with the soft configuration: *Nios V/m + SRAM memory*. Fill in the Table 1 using this value. This execution time will be compared with the time that will be obtained in the previous and next parts I and III of this lab assignment. Stop the execution of the sample program by typing "CTRL+c".

   Question 2.
   Is this new value a reasonable speed-up repect to the results obtained in Part I? Explain and justify your answer using Fig. 1.

Part III. Access to memory hierarchy with Nios V/g processor

The goal of this third part consists in measuring the execution time of the Fibonacci program using the Nios V/g soft processor. Both technologies for main memory will be used: external SDRAM memory and SRAM on-chip memory. The difference between this part and Parts I and II is that the model of Nios V/g processor used. A new configuration file will be used: `DE0_Nano_Basic_Computer_23jul24.sof`.

First of all, the FPGA circuit of DE0-Nano board should be reconfigured.

1. Execute the following command in the Nios V Command Shell window.

```
$ quartus_pgm.exe -c 1 -m JTAG -o "p;DE0_Nano_Basic_Computer_23jul24.sof@1"
```

Follow the following steps when using the external SDRAM memory.

2. Change the immediate value for the `li` instruction at the beginning of the `lab2_part1_2_3_main.s` file. This is the start address of the exception routine when the SDRAM memory is used. Additionally, change the output message sent to terminal.

lab2_part1_2_3_main.s

```
li s2, 0x0

TEXTO_MAIN:
.asciz "\nI am at lab2_part3 with NiosV/g ...  running ...  \n"
```

3. Now, compile the file `lab2_part1_2_3_main.s` as explained in Part I.

```
$ riscv32-unknown-elf-as.exe lab2_part1_2_3_main.s -alsg -o
lab2_part1_2_3_main.s.obj > lab2_part1_2_3_main.s.log
```

4. Link the program using the `linker.x` file as follows.

```
$ riscv32-unknown-elf-ld.exe -g -T ../lab2_bsp/linker.x -nostdlib
-e _start -u _start --defsym __alt_stack_pointer=0x08001F00 --defsym
__alt_stack_base=0x08002000 --defsym __alt_heap_limit=0x8002000 --defsym
__alt_heap_start=0x8002000 -o lab2_part1_2_3_main.elf lab2_part1_2_3_main.s.obj
lab2_part1_2_3_fibo.s.obj excepcionTimer.s.obj contador.s.obj
escribir_JTAG.s.obj BCD.s.obj DIV.s.obj
$ niosv-stack-report.exe -p riscv32-unknown-elf- lab2_part1_2_3_main.elf
```

5. Now, download and execute the program. Additionally, display the results of the program using the terminal software.

```
$ niosv-download.exe -g lab2_part1_2_3_main.elf
$ juart_terminal.exe
```

6. Register the value displayed for the number of 1321.20 *ms* time intervals that were needed to execute the benchmark. This measure is related with the soft configuration: *Nios V/g + SDRAM memory*. Fill in the Table 1 using this value. This execution time will be compared with the times that were obtained in Parts I and II of this lab assignment. Stop the execution of the sample program by typing "CTRL+c".

Follow the following steps when using the external on-chip SRAM memory.

7. Change the immediate value for the `li` instruction at the beginning of the `lab2_part1_2_3_main.s` file. This is the start address of the exception routine when the on-chip SRAM memory is used. Additionally, change the output message sent to terminal.

lab2_part1_2_3_main.s

```
li s2, 0x08000000

TEXTO_MAIN:
.asciz "\nI am at lab2_part3 with NiosV/g ...  running ...  \n"
```

8. Now, compile the file `lab2_part1_2_3_main.s` as explained in Part I.

```
$ riscv32-unknown-elf-as.exe lab2_part1_2_3_main.s -alsg -o
lab2_part1_2_3_main.s.obj > lab2_part1_2_3_main.s.log
```

9. Link the program using the `linker_SRAM.x` file that was generated in Part II as follows.

```
$ riscv32-unknown-elf-ld.exe -g -T ../lab2_bsp/linker_SRAM.x -nostdlib
-e _start -u _start --defsym __alt_stack_pointer=0x08001F00 --defsym
__alt_stack_base=0x08002000 --defsym __alt_heap_limit=0x8002000 --defsym
__alt_heap_start=0x8002000 -o lab2_part1_2_3_main.elf lab2_part1_2_3_main.s.obj
lab2_part1_2_3_fibo.s.obj excepcionTimer.s.obj contador.s.obj
escribir_JTAG.s.obj BCD.s.obj DIV.s.obj
$ niosv-stack-report.exe -p riscv32-unknown-elf- lab2_part1_2_3_main.elf
```

10. Now, download and execute the program. Additionally, display the results of the program using the terminal software.

```
$ niosv-download.exe -g lab2_part1_2_3_main.elf
$ juart_terminal.exe
```

11. Register the value displayed for the number of 1321.20 *ms* time intervals that were needed to execute the benchmark. This measure is related with the soft configuration: *Nios V/g + SRAM memory*. Fill in the Table 1 using this value. Calculate the speed-up of each soft SoC configuration using as reference the SoC configuration based on Nios V/m and external SDRAM memory, which is takes as SpeedUp = 1. This execution time will be compared with the times that were obtained in Parts I and II of this lab assignment. Stop the execution of the sample program by typing "CTRL+c".

12. Now, please, answer the following questions.

Question 3. Why do different versions of soft SoC systems based on Nios V/m processor provide such values of speed-up? Explain and justify your answer.

Question 4. Why do the executions times of the two versiones of soft SoC systems based on Nios V/g processor provide the same execution time? Explain and justify your answer.

Question 5. Why does the soft SoC systems based on Nios V/m processor provide lower performance than the SoC system based on Nios V/g? Explain and justify your answer.

Part IV. Reverse engineering of the data cache memory

In this section, the memory size and block size of data cache memory of a Nios V/g processor will be discovered. A simple program traversing a vector of bytes (`V`) is used to achieve this goal.

The steps to follow in this hands-on activity are described in the following paragraphs.

- Take the files for the source code of the benchmark used in Parts I, II and III. Reduce the number of iterations (`ITERATIONS`) of the main program from 1000000 to 50000 (file: `lab2_part1_2_3_main.s`).
- Modify as shown below the source code of Fibonacci subroutine (`lab2_part1_2_3_fibo.s`). This modification simply traverses a `V` vector of bytes. The `lw` and `sw` instructions to manage the stack should be also modified.

Modification of `lab2_part1_2_3_fibo.s` file for Part IV

```
...
add s4, zero, zero /* s4 is a counter */
la s5, X /* s5 points to X: 1024, 2048, 4096, 8192, 16384, 32768, 65536 */
la s6, V /* s6 points to V vector */
LOOP:
bge s4, s5, STOP /* loop ends when s5 >= s4 */
add s7, s6, s4 /* next memory address is obtained */
lb zero, 0(s7) /* V vector is accessed but data is not saved in register */
addi s4, s4, P /* P, data stride, increments V pointer by:  1,2,4,...,64 */
j LOOP
...
.data
V:
.skip 65536
...
```

Note that in the code above there are two parameters that need values to be assigned: `X` and `P`. `X` represents the number of elements of the `V` vector that are accessed with `P` data stride. `P` represents the number of elements of the `V` vector that are in memory between two successive memory accesses using the `lb` instruction.

A new parameter called E is defined. It represents the number of elements of the `V` vector actually accessed with the `lb` instruction. The following equation is valid, $X = P \times E$. This equation calculates the number of vector elements that are copied to data cache memory.

Table 2: Runtime measurements for discovering the microarchitecture of data cache. E is the number of memory accesses. P is data stride.

	E=1024	**E=2048**	**E=4096**	**E=8192**	**E=16384**
P=1					
P=2					
P=4					
P=8					
P=16					
P=32					
P=64					

- Now, Table 2 will be filled in with values of execution times obtained in the same way as in the previous activities, measuring the execution times of the benchmark program. The configuration file for the FPGA is named: `DE0_Nano_Basic_Computer_23jul24.sof`, which integrates a Nios V/g soft processor and a SDRAM memory controller. A `Makefile` file has been prepared to automatize this process.

```
# USB connection to DE0-Nano board is tested
$ jtagconfig.exe
# compile and link:  lab2_part1_2_3_main.elf is generated
$ make
# configure the FPGA
$ make configure
# save the *.elf program in main memory
$ make download
# shows results on screen
$ juart_terminal.exe
```

The execution times are mostly due to the `lb` instruction that accesses to part of the elements of the `V` vector with `P` data stride. For each measure, the program should be executed after modifying the value

for `X` and the value for `P` assuring that $X = P \times E$. The value for `X` is introduced in the instruction: `la s5, X; X=1024, 2048, 4096, 8192, 16384, 32768, 65536`. The value for `P` is introduced in the instruction: `addi s4, s4, P; P= 1,2,4,8,16,32,64`.

Question 6. Using the values in Table 2, discover the memory size and block size of Nios V/g's data cache.
Note that when `X` is doubled, i.e., `E` is doubled for the same `P`, the execution time usually is doubled. However, for each `P`, there is one transition from `X` to 2×`X` that this is not true. This case determines the cache size.
Additionally, for the same column of Table 2, the execution time increases as `P` increases. However, after a given value for `P`, the execution time remains constant when the data cache is overflowed. This case determines the block size.