Lab Assignment 3:

Performance evaluation of pipelined processors

Computer Architecture (40969)
School of Computer Science (EII)
University of Las Palmas de Gran Canaria

Main goals of Lab 3

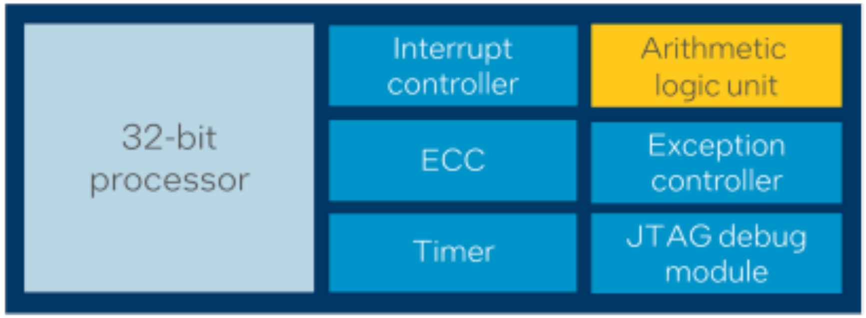
- Performance evaluation of the Nios V/g pipelined processor and comparison with the Nios V/m pipelined processor: CPI, operations/sec.
- Analyze the effect on Nios V/g performance of the machine instruction reordering software technique.
- Propose the realization of a theoretical exercise in which a possible change in the Nios V/g microarchitecture is evaluated.

Scheduling: 4 weeks

• S1: Section 1;	2 hours
• S2: Section 2;	2 hours
• S3: Sections 3 y 4;	2 hours
• S4: Test;	1.5 hours

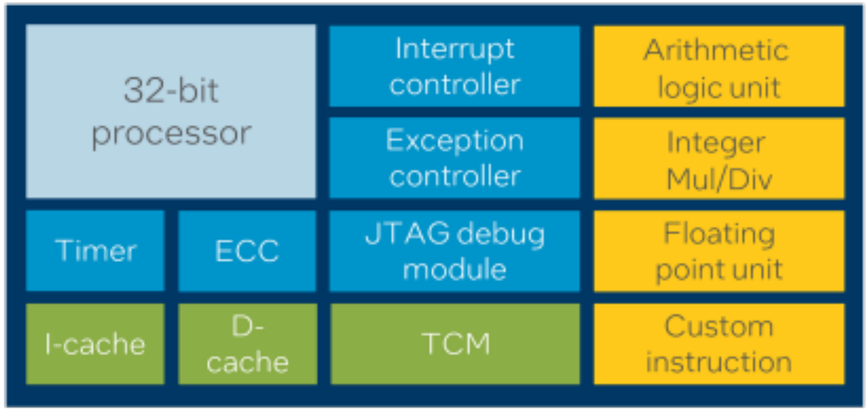
Soft processors: Nios V/{m,g}

Nios V/m



Multicycle: mínimum 4 clk/inst
Pipelined: 5 pipes
WITHOUT instruction cache
WITHOUT data cache
f = 50 MHz
Configuration file of the pipelined version for the DE0-Nano board: “DE0_NanoBasic_Computer_22jul24.sof ”

Nios V/g



Pipelined : 5 pipes
WITH instruction cache
WITH data cache
WITH dynamic branch prediction
f = 50 MHz
Configuration file for the DE0-Nano board: “DE0_NanoBasic_Computer_23jul24.sof”

Instruction phases for the "multicycle" Nios V/m

Table 21. Processor Non-pipelined Stages

Stage	Denotation	Function
F	Instruction fetch	Pre-decode for register file read
D	Instruction decode	• Decode the instruction • Register file read data available • Hazard resolution and data forwarding
E	Instruction execute	• ALU operations • Memory address calculation • Branch resolution • CSR read/write
M	Memory	• Memory and multicycle operations • Register file write • Next PC logic • Branch redirection

- Issue: 1 instruction / cycle
- Commitment: 1 instruction / cycle
- Base CPI = 4 cycles
- Stalls: multicycle operations, data dependences

Pipes for the Nios V/{m,g} cores

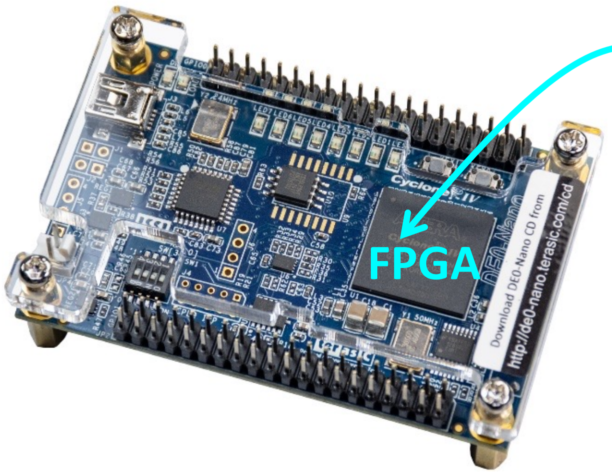
Table 63. Processor Pipeline Stages

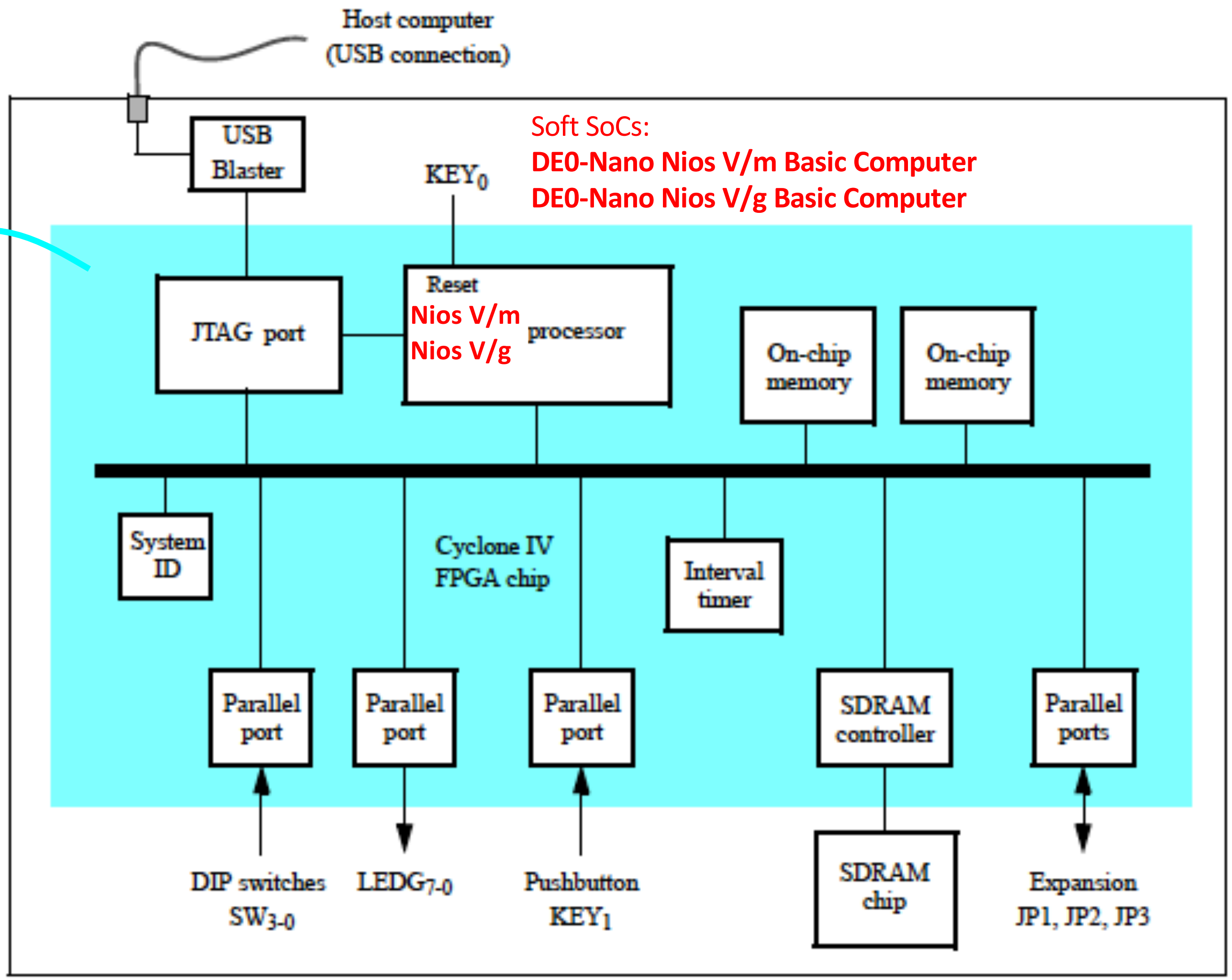
Stage	Denotation	Function
F	Instruction fetch	• PC+4 calculation • Next instruction fetch • Pre-decode for register file read
D	Instruction decode	• Decode the instruction • Register file read data available • Hazard resolution and data forwarding
E	Instruction execute	• ALU operations • Memory address calculation • Branch resolution • CSR read/write
M	Memory	• Memory and multicycle operations • Register file write • Branch redirection
W	Write back	• Facilitates data dependency resolution by providing general-purpose register value.

- Issue: 1 instruction / cycle
- Commitment: 1 instruction / cycle
- Base CPI = 1 cycle
- Stalls: multicycle operations, data dependences

Soft SoCs

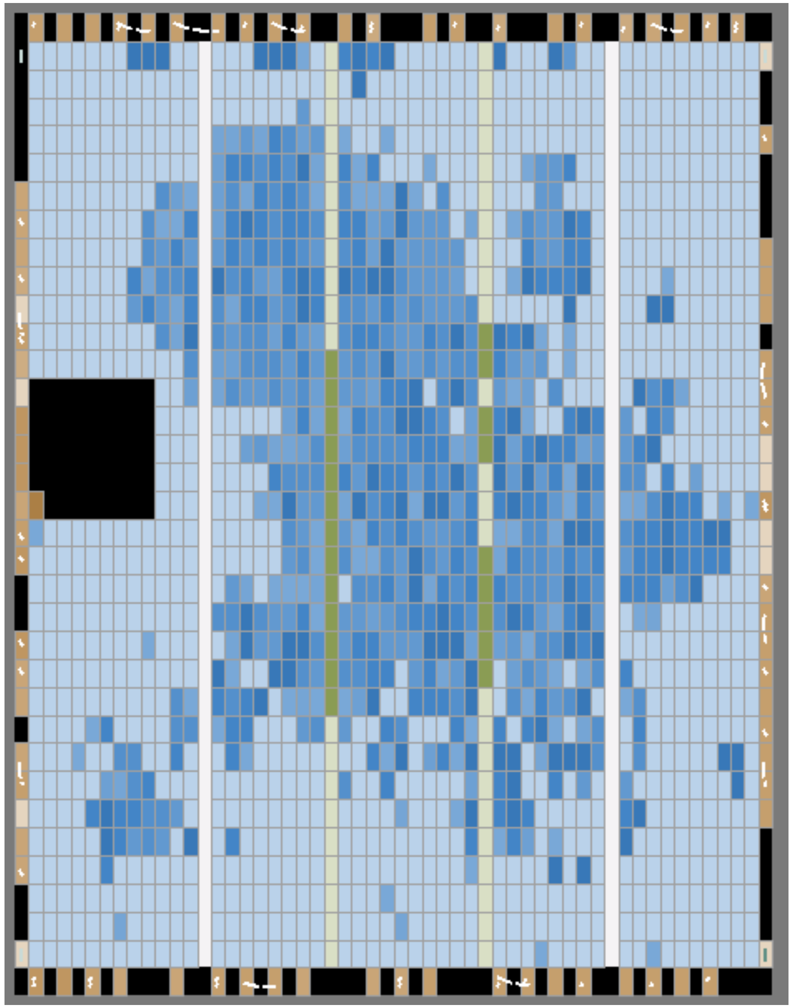
Terasic DE0-Nano board





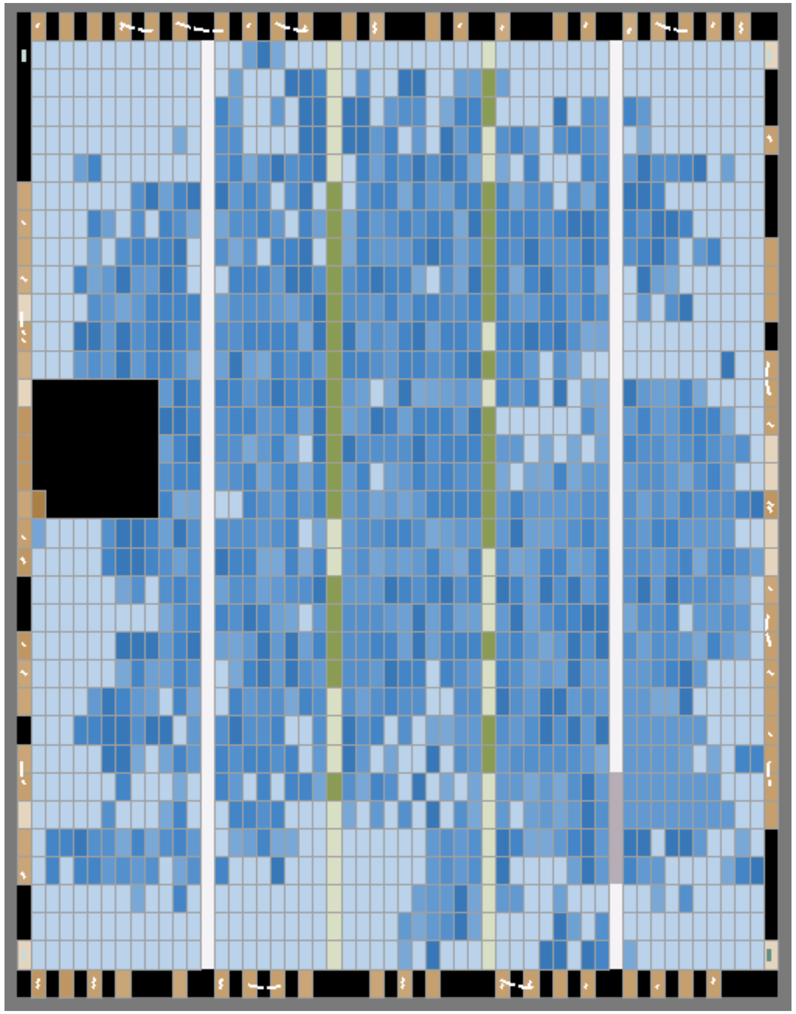
FPGA: Intel Cyclone IV - EP4CE22F17C6N

DE0-Nano Nios V/{m,g} Basic Computers



Chip Layout, Nios V/m + SDRAM

Logic elements: 36 %, max. freq.: 124.47 MHz



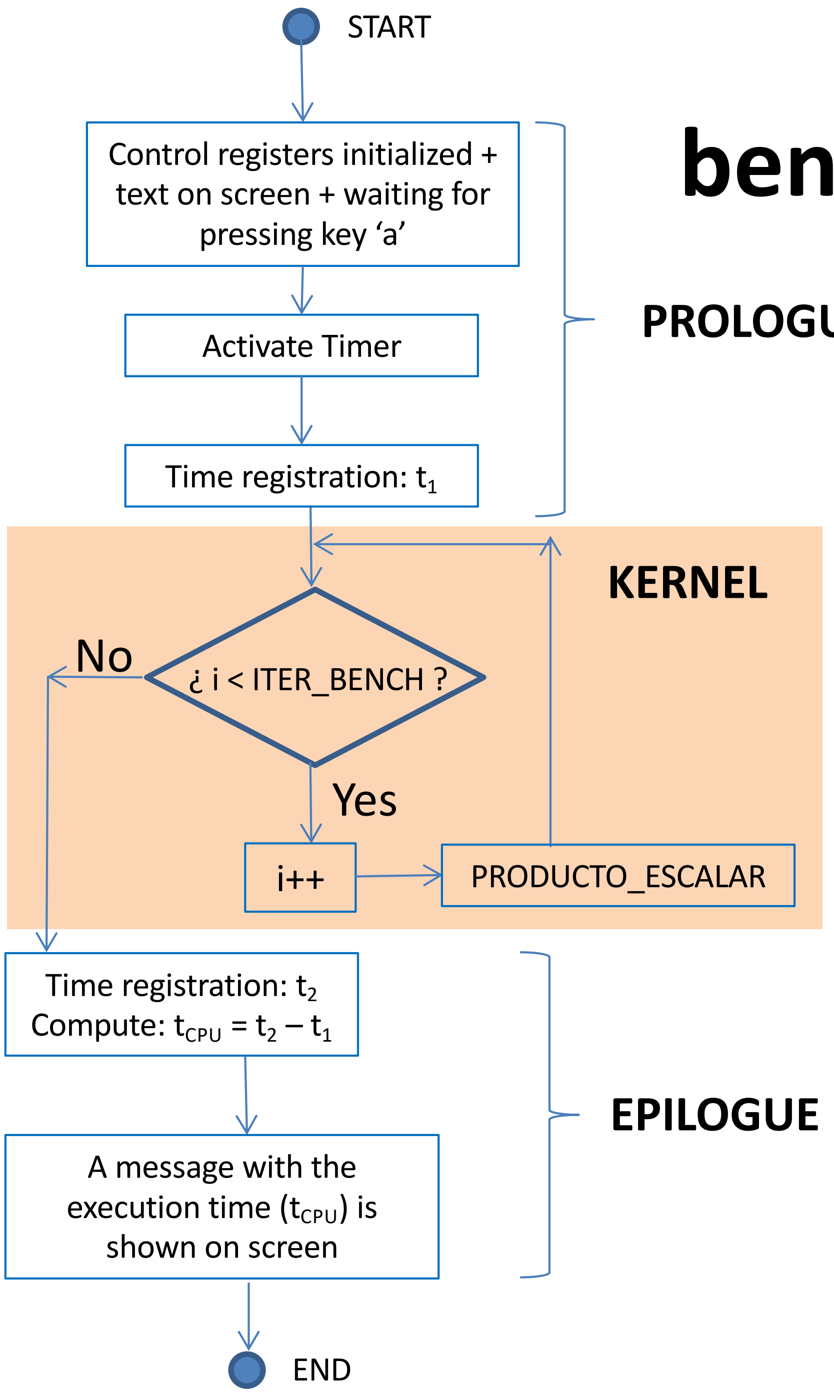
Chip Layout, Nios V/g + SDRAM

Logic elements: 57 %, max. freq.: 122.13 MHz

Part 1: Analysis of the usage of instructions types in a benchmark program and the CPI of Nios V/{m,g} processors

General Description. A synthetic benchmark program named *benchNIOSV2024_dotProduct* is used **to analyze the instruction usage of the 32-bit RISC-V instruction set**. This program performs the scalar product of two vectors repeatedly, as many times as specified by the program constant ITER_BENCH. Additionally, the CPI of the Nios V/{m,g} processors is calculated.

PART 1: benchNIOSV2024_dotProduct



Parte 1: Objetives

- <u>Objective 1:</u> Classify the instructions by counting the number of times each instruction is executed in the PRODUCT_ESCALAR subroutine of the source code located in the file: `producto_escalar.s`. A flowchart of the main activities performed in the benchmark is shown in the transparency above.
- <u>Objective 2:</u> Next, calculate the total number of instructions executed and the percentage of each type of instruction (ALU, MEMORY, JUMP, OTHER) by filling in Table 1.
- <u>Objective 3:</u> Record the total number of clock cycles in which the benchmark is executed, both for the Nios V/m and Nios V/g processors, and calculate the CPI of the program. **Important:** Note that the non-kernel portion of the benchmark does not contribute a significant number of instructions to the CPI calculation.

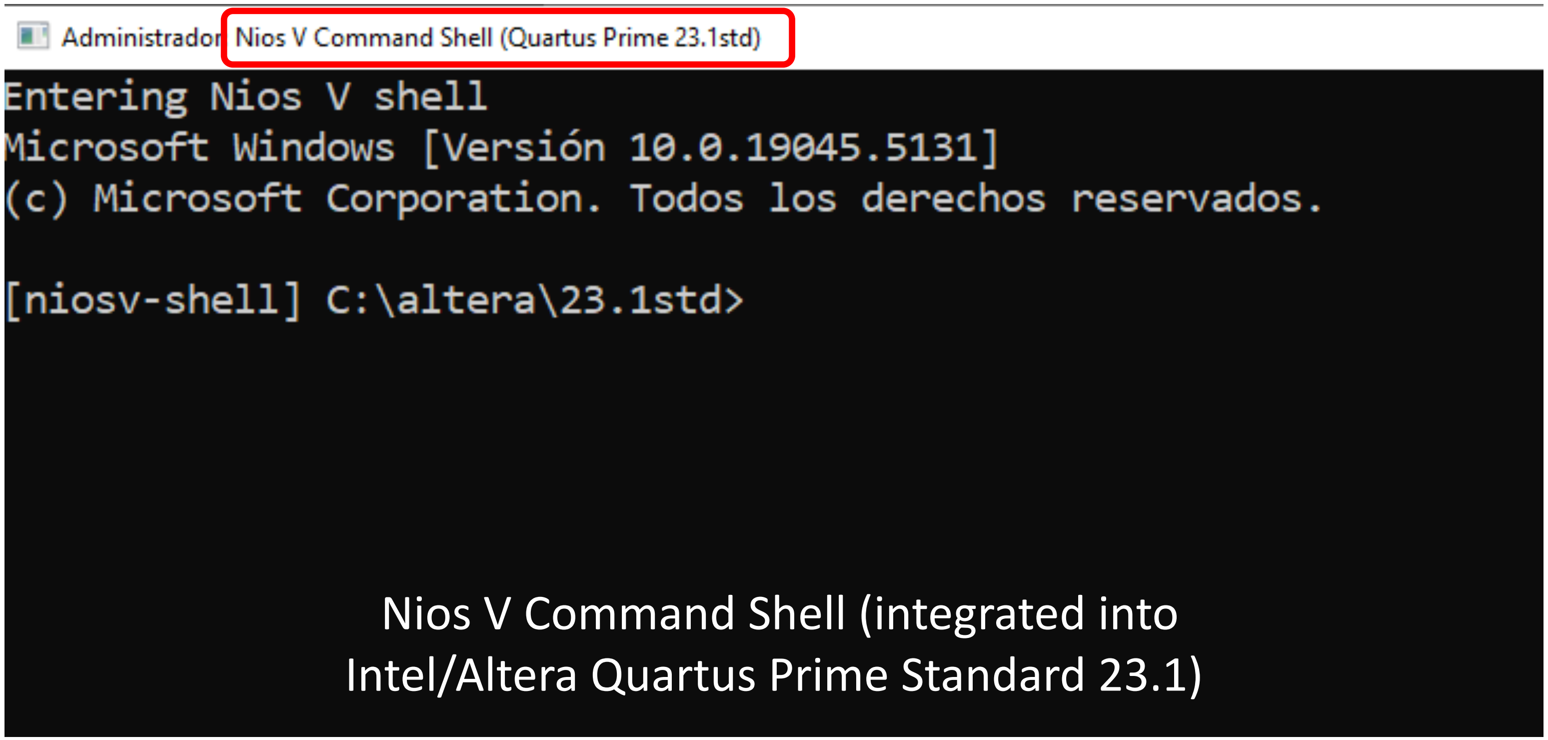
Part 1: hands-on exercises using Nios V/m

1. Create a new directory, for example: part1 and make cd. Execute the following command in the Nios V Command Shell Window.
2. Copy the files: `benchNIOSV2024_dotProduct.s`, `productoEscalar.s`, `escribir_jtag.s`, `DIV.s`, `BCD.s`, `Makefile` in this directory
3. Compile and link the benchmark program: `benchNIOSII_dotProduct`.
4. Configure the FPGA in the DE0-Nano board using the file `DE0_NanoBasic_Computer_22jul24.sof` into the board DE0-Nano
5. Execute step by step to count the type of instruction.

Part 1: hands-on exercises using Nios V/g

The same steps as for Nios V/m except the configuration file for the DE0-Nano board:

`DE0_NanoBasic_Computer_23jul24.sof`



Nios V Command Shell (integrated into Intel/Altera Quartus Prime Standard 23.1)

Nios V Command Shell

- $ cd <project directory>
- $ sh
- $ make help --> help for Makefile
- $ jtagconfig.exe
- (modify Makefile: `SOFfile`, `LINKfile`)
- $ make --> compila & link, output: "`benchNIOSV2024_dotProduct.elf`"
- $ make configure
- $ make download
- $ juart-terminal.exe

```
C:/altera/12.1sp1/University_Program/NiosII_Computer_Systems/DE0-Nano/DE0-Nano_Basic_Computer_Ni
osVm_conSDRAM/verilog/software/niosv/ACpractica3niosv/Part1 # juart-terminal.exe
juart-terminal: connected to hardware target using JTAG UART on cable
juart-terminal: "USB-Blaster [USB-0]", device 1, instance 0
juart-terminal: (Use the IDE stop button or Ctrl-C to terminate)


pRESS KEY A TO START THE BENCHMARK: A
... RUNNING ...
 cYCLE COUNTER (CLK@50mhZ): 0000000021225909
 nUMBER OF ITERATIONS DONE: 00005000
bye!
```

Parte 1: Table 1

ALU instructions	Number of executions	MEMORY instructions	Number of executions	BRANCH & JUMP instructions	Number of executions	OTHER instructions	Number of executions
`addi`		`lw`		`beq`		`nop`	
...		...		...		...	
...		...		...		...	
Total ALU instructions		**Total MEMORY instructions**		**Total BRANCH & JUMP instructions**		**Total OTHER instructions**	
N (total number of executed instructions)							
% ALU		**% MEMORY**		**% BRANCH & JUMPS**		**% OTHER**	
Cicles Nios V/m				**Cicles Nios V/g**			
Total CPI of program Nios V/m				**Total CPI of program Nios V/g**			

Questions

Question 1.
What type of program is `benchNIOSV2024_dotProduct.elf`: arithmetic, memory, or branch/jump? Justify and argue your answer.

Question 2.
Taking the following average number of cycles per instruction for the Nios V/m and Nios V/g soft processors: 1 cycle (ALU ops), 1 cycle (memory ops), 2 cycles (jump and branch ops), calculate the theoretical CPI of the program when both processors execute the benchmark program. Justify and argue your answer.

Question 3.
What are the differences you found out in the values obtained for the CPIs of Nios V/m and Nios V/g processors? What are the causes for these differences? Justify and argue your answer.

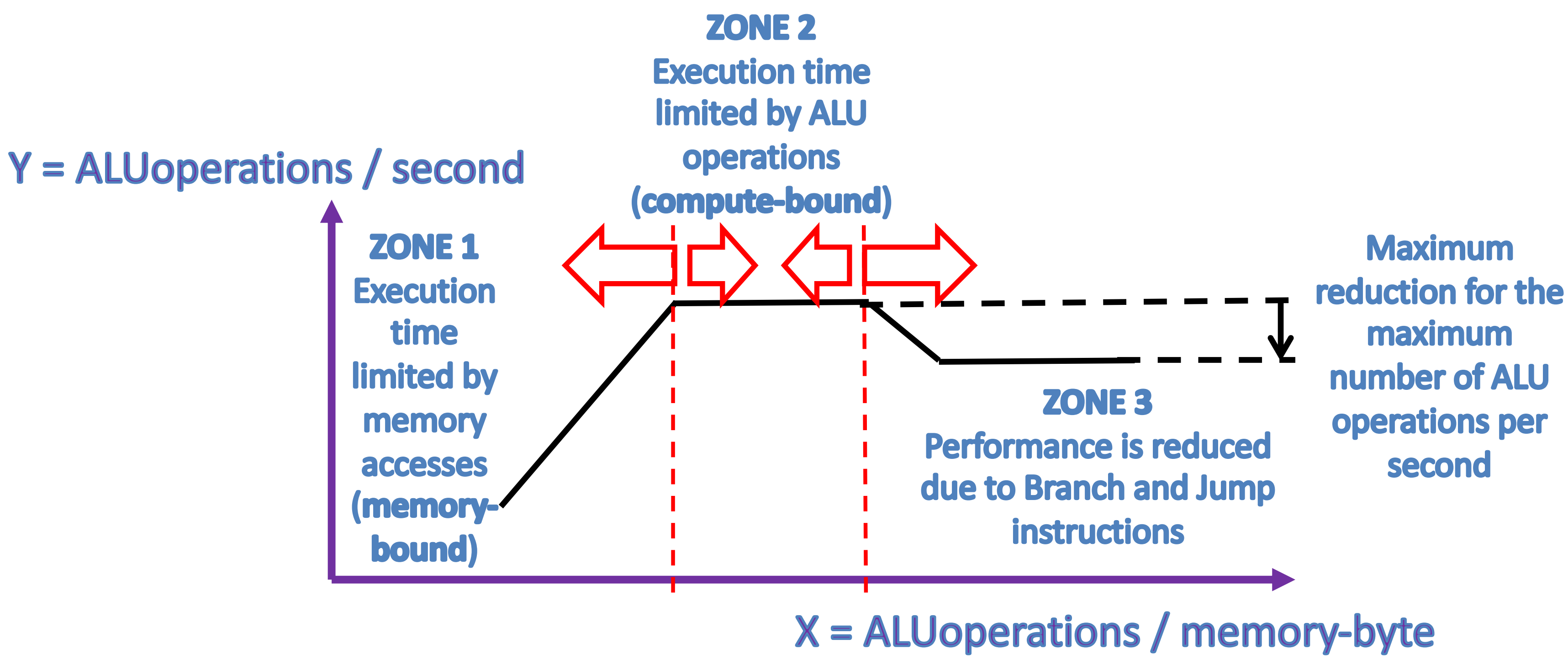
Part 2. Analysing the limitations of the 'operationsALU/second' ratio of a benchmark program on the Nios V/m and Nios V/g pipelined processors

General Description. This part evaluates the limits of each Nios V/{m,g} soft processor in terms of the number of ALU operations it can perform in each unit of time. In particular, the limits measured in 'ALU operations/second' caused by the ALU functional unit of the processors, the memory hierarchy of the SoC embedded computer and the need to execute jump instructions are analysed.

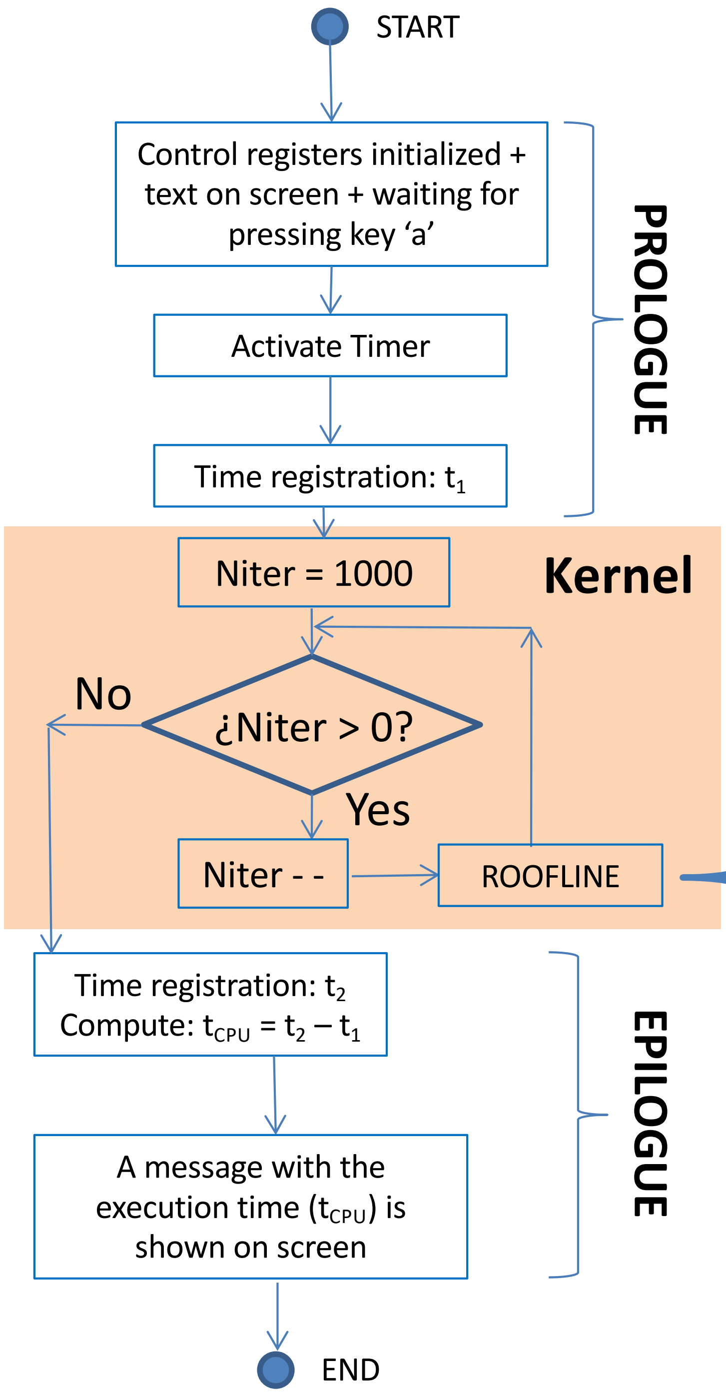
Parte 2: Objetive

- Obtain the curve ‘ALU operations/second “ versus ”ALU operations/byteMEMORY’ (see Figure “roofline”). This curve represents the performance level of the Nios V processor measured in number of ALU operations performed per unit time.

“roofline” curve: “ALUoperations/second” vs. “ALUoperations/memory-byte”



PARTE 2: benchNIOSV2024_roofline.s




```
lw              t4, 0(t4)  /* r4 = Niter, number of iterations */
LOOP:
/* Begin: ZONE 1 for memory accesses */
                lw              t5, 0(t2) /* carga A */
/* End: Zone for memory accesses */

/* Begin: ZONE 2 for ALU operations*/
                add             t5, t5, t5
/* End: Zone for ALU operations */

/* 2 ALU instructions, they must not be commented */
                addi            t6, t6, 1   /* contador_iteraciones_realizadas++ */
                subi            t4, t4, 1   /* Niter-- */

/* Begin: ZONE 3 for an internal loop with multiple branch instructions */
                addi            t1, zero,NiterInternas /* NiterInternas= 1,5,20,47,... */
bucleInterno:
                add             t5, t5, t5
                addi            t1, t1, -1
                bgt             t1, zero, bucleInterno
/* End: ZONE 3 for an internal loop that forces multiple branch instructions */

/* End of LOOP, this instruction must not be commented */
                bgt             zero, t4, LOOP
```

Part 2: Table 2

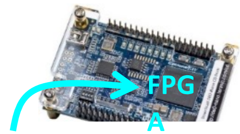
(one for Nios V/m and another for Nios V/g)

ID	kernel: roofline.s Number of executed instructions per iteration			**X coordinate** ALUoperations/ memory-byte (ALU / 4 * lw/sw)	Iterations (N_{iter})	N (executed instructions, N_{iter} * [lw/sw+ALU+br/j])	Nios V/{m,g}, frequency (f)= 50 MHz			
	lw/sw	**ALU**	**br/j**				cycles	t_{CPU} (sec=cycles / f)	**Y coordinate** ALUoperations/sec (N_{iter} * ALU / t_{CPU})	CPI (cycles / N)
1	4	3	1		1000					
2	3	3	1		1000					
3	2	3	1		1000					
4	1	3	1		1000					
5	1	4	1		1000					
6	1	5	1		1000					
7	1	7	1		1000					
8	1	11	1		1000					
9	1	15	1		1000					
10	1	19	1		1000					
11	1	23	1		1000					
12	1	27	1		1000					
13	1	31	1		1000					
14	1	35	1		1000					
15	1	39	1		1000					
16	1	47	1		1000					
17	1	50	2		1000					
18	1	58	6		1000					
19	1	88	21		1000					
20	1	142	48		1000					
21	1	848	401		1000					
22	1	1048	501		1000					

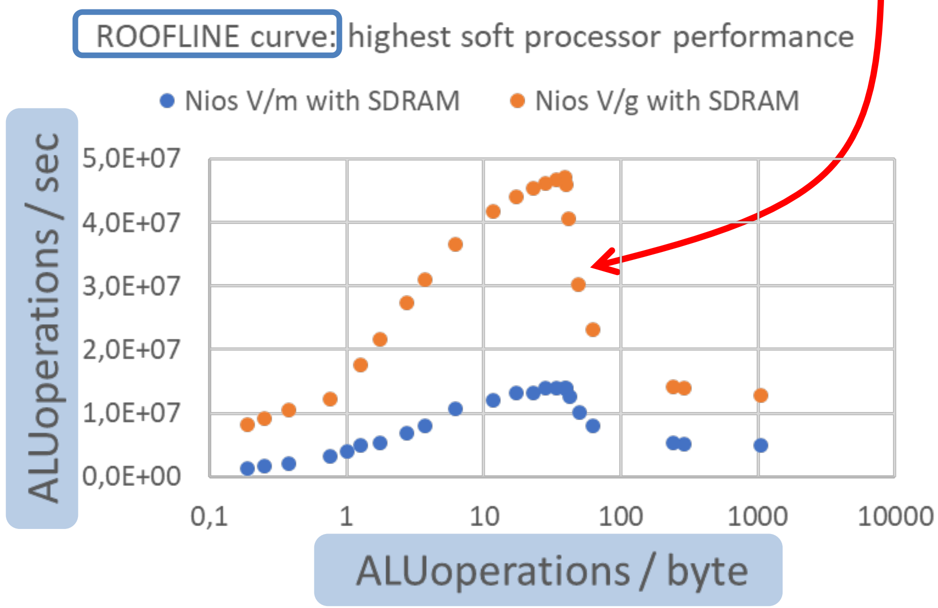
Lab 3. Performance evaluation of pipelined processors: ALUops/sec, Cycles-per-instruction

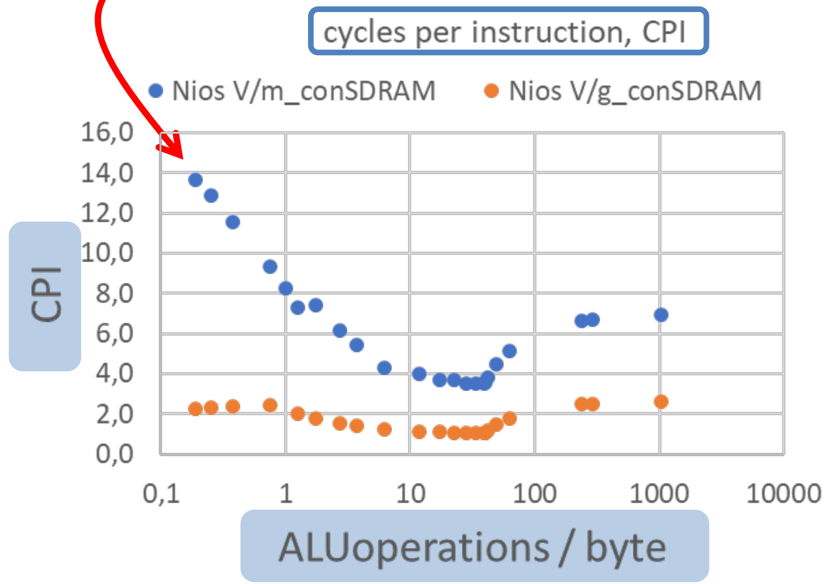
DE0-Nano Nios V/g Basic Computer

Nios V/g vs. Nios V/m



DE0-Nano Nios V/m Basic Computer





Questions

Question 4.
What number of ALU operations per memory-byte the processors Nios V/{m, g} are "compute-bound", i.e., the program is limited by ALU operations? Which is the maximum number of ALU operations per second for the Nios V/m processor? What is the maximum number of ALU operations per second achieved by the Nios V/g processor? Justify and argue your answer.

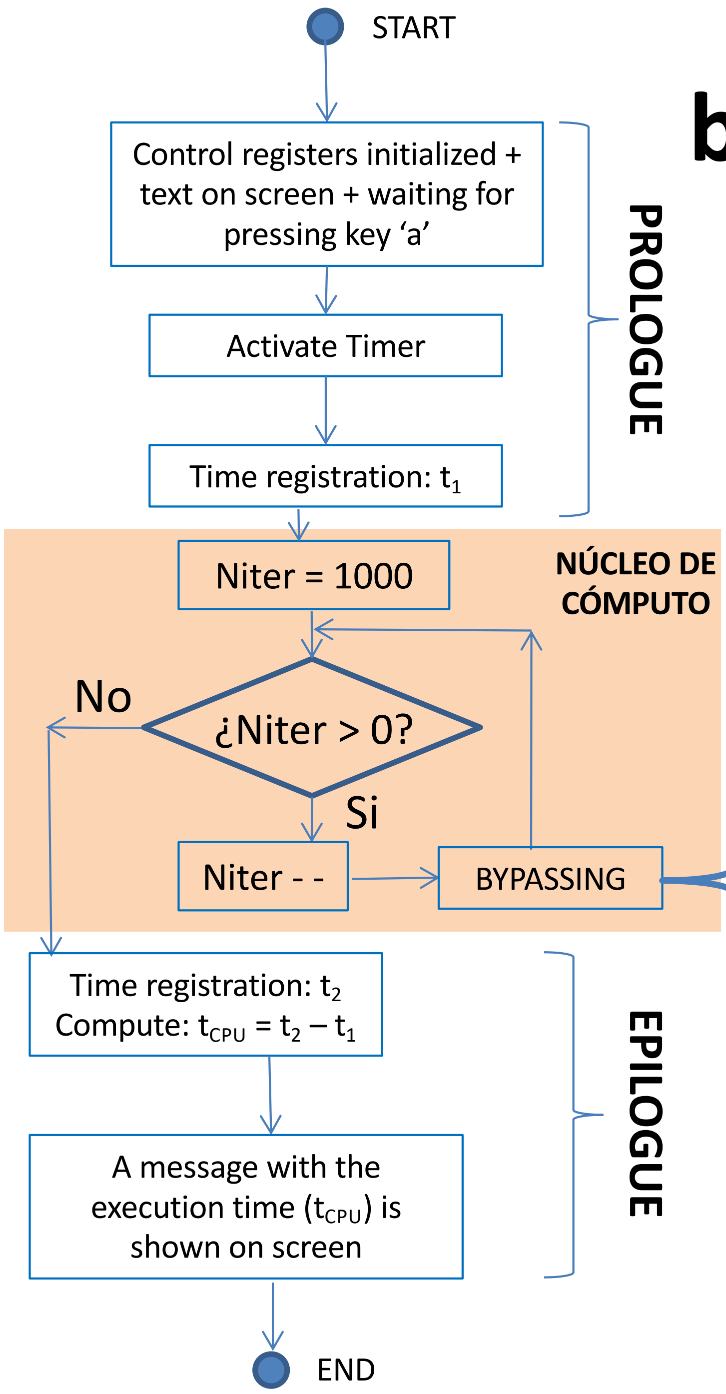
Question 5.
Which is the maximum percentage of performance reduction for the Nios V/{m,g} processors when the branch instructions are executed?

Question 6.
Is the benchmark program used in Part 1, (`benchNIOSV2024_dotProduct` memory-bound or compute-bound? Justify and argue your answer.

Part 3. Influence of instruction reordering on performance of Nios V/m y Nios V/g pipelined processors

General description. A new synthetic benchmark program called *benchNIOSV2024_bypassing* is used to evaluate the effect of true data (RAW) dependencies between load instructions and ALU instructions. This benchmark program is similar to Parts 1 and 2 of this lab assignment except that the computation kernel has been modified and is now located in a file named `bypassing.s`. Next, we propose to apply the instruction reordering technique to reduce the execution time of the benchmark program using the Nios V/m and Nios V/g pipelined processors.

PART 3: benchNIOSV2024_reordering.s



VERSION 1

```
LOOP:
        lw      s6, 0(s2)       /* load A variable */

/* ZONE: data dependence */
        /* addition WITH data dependence to lw s6, 0(s2) */
        add     s6, s6, s6

/* 2 more ALU instructions, they must not be commented */
        addi    s7, s7,  1      /* Niter++ */
        addi    s4, s4, -1      /* N-- */
/* end of LOOP*/
        bne     s4, zero, LOOP
```

VERSION 2

```
LOOP:
        lw      s6, 0(s2)       /* load A variable */

/* ZONE: data dependence */
        /* addition WITHOUT data dependence to lw s6, 0(s2) */
        add     s6, s4, s4

/* 2 more ALU instructions, they must not be commented */
        addi    s7, s7,  1      /* Niter++ */
        addi    s4, s4, -1      /* N-- */
/* end of LOOP*/
        bne     s4, zero, LOOP
```

Part 3: VERSION 3

- Reorder the location of instruction in the LOOP of Version 1 to reduce execution time (t_{CPU})
- Register execution time (t_{CPU}) for Nios V/m and Nios V/g processors
- Calculate speed-up
- Calculate total CPI
- Calculate the CPI due to stall cycles from data dependences between `lw -> add` instructions

Benchmark output

```
rEORDERING RISC-V INSTRUCTIONS

pRESS KEY A TO START THE BENCHMARK: A
... RUNNING ...
 cYCLE COUNTER (CLK@50mhZ): 0000000000013144
 nUMBER OF ITERATIONS DONE: 00000001
bye!
```

PART 4: Designing a new pipelined processor

Original Processor 5 pipes

```
lw    r1, inmediato(r2)
```

Inmediate = 0

Inmediate != 0

New Processor 4 pipes

Inmediate = 0:

```
lw    r1, r2
```

Inmediate != 0:

```
addi  r2,r2,inmediato
lw    r1, r2
```

Calculate the additional percentage of executed instructions needed for the new processor focussing only on the PRODUCTO_ESCALAR subroutine that was coded in Part 1