



## **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

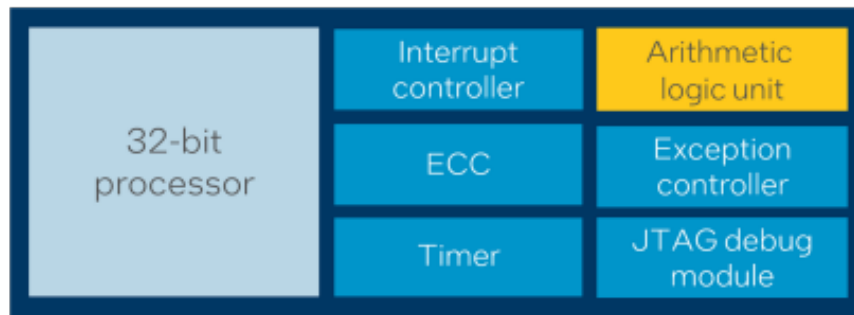
- Performance evaluation of the Nios V/g pipelined processor and comparison with the Nios V/m pipelined processor: CPI, operations/sec.
- Analyze the effect on Nios V/g performance of the machine instruction reordering software technique.
- Propose the realization of a theoretical exercise in which a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

# Scheduling: 4 weeks

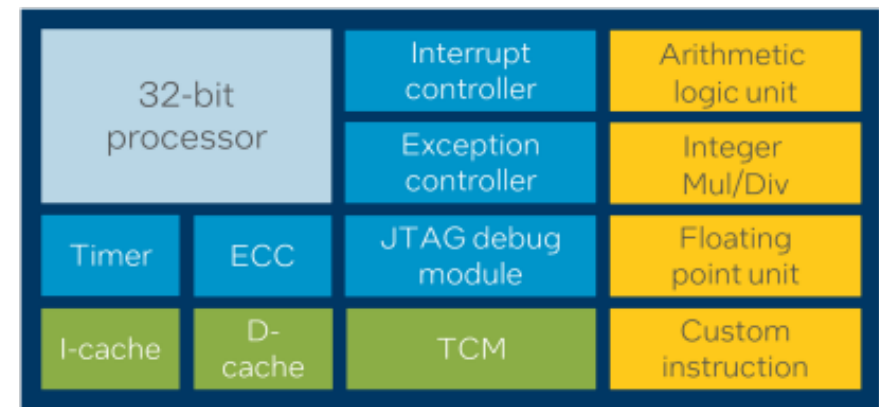
- S1: Section 1; 2 hours
- S2: Section 2; 2 hours
- S3: Sections 3 y 4; 2 hours
- S4: Test; 1.5 hours

# Soft processors: Nios V/{m,g}

Nios V/**m**



Nios V/**g**



**Multicycle**: minimum 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"  
 "

**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	<ul style="list-style-type: none"><li>• Decode the instruction</li><li>• Register file read data available</li><li>• Hazard resolution and data forwarding</li></ul>
E	Instruction execute	<ul style="list-style-type: none"><li>• ALU operations</li><li>• Memory address calculation</li><li>• Branch resolution</li><li>• CSR read/write</li></ul>
M	Memory	<ul style="list-style-type: none"><li>• Memory and multicycle operations</li><li>• Register file write</li><li>• Next PC logic</li><li>• Branch redirection</li></ul>

- 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

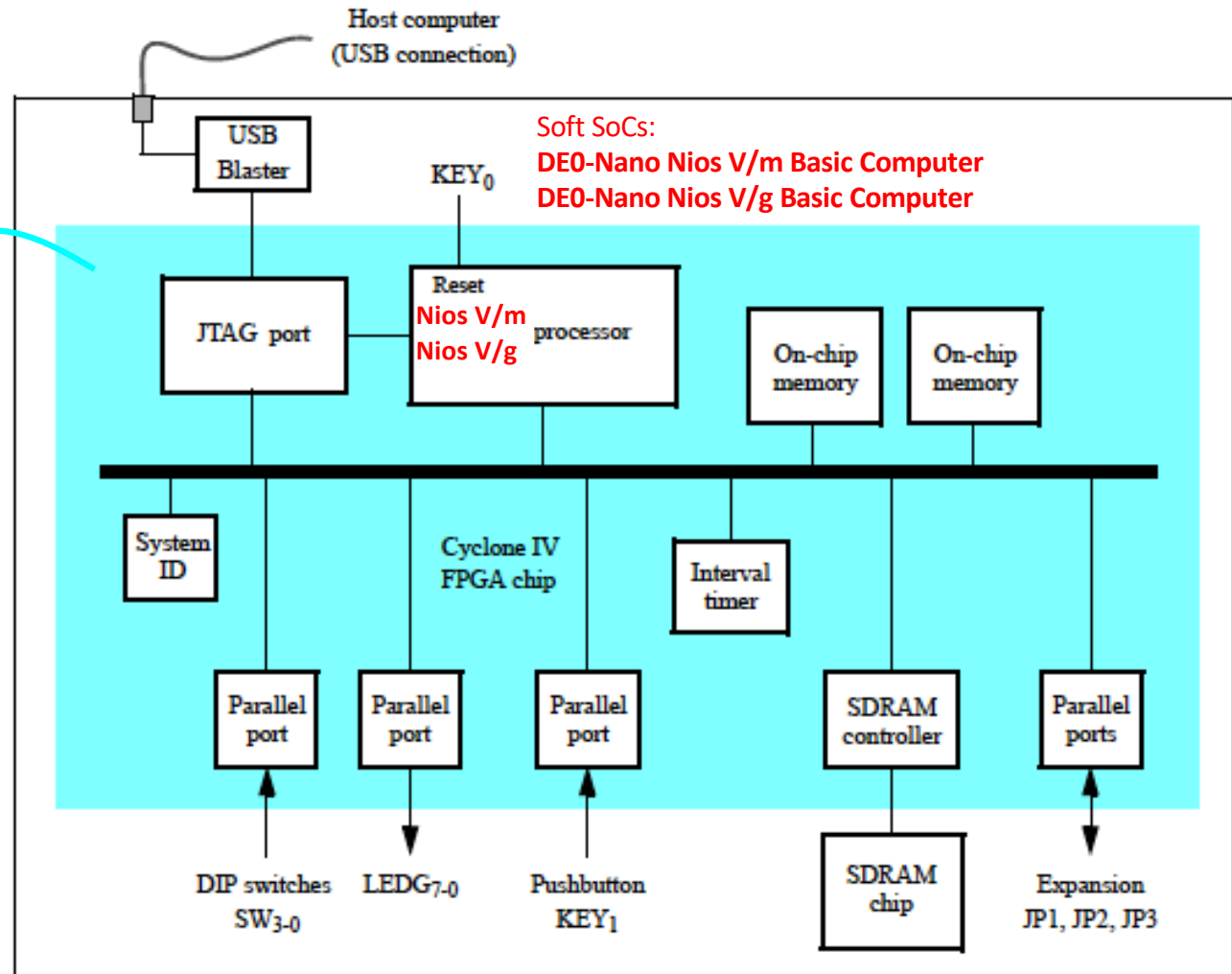
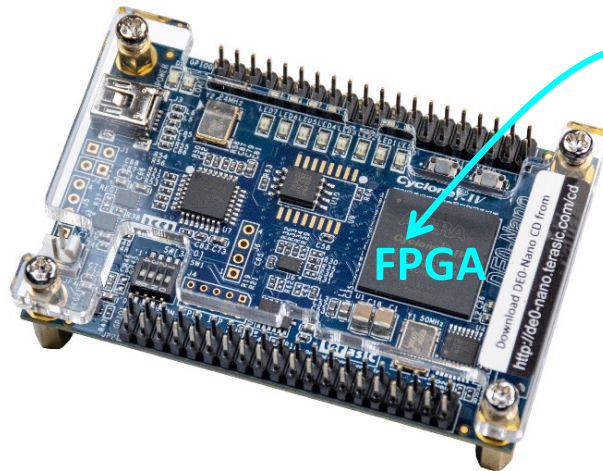
Table 63. Processor Pipeline Stages

Stage	Denotation	Function
F	Instruction fetch	<ul style="list-style-type: none"><li>• PC+4 calculation</li><li>• Next instruction fetch</li><li>• Pre-decode for register file read</li></ul>
D	Instruction decode	<ul style="list-style-type: none"><li>• Decode the instruction</li><li>• Register file read data available</li><li>• Hazard resolution and data forwarding</li></ul>
E	Instruction execute	<ul style="list-style-type: none"><li>• ALU operations</li><li>• Memory address calculation</li><li>• Branch resolution</li><li>• CSR read/write</li></ul>
M	Memory	<ul style="list-style-type: none"><li>• Memory and multicycle operations</li><li>• Register file write</li><li>• Branch redirection</li></ul>
W	Write back	<ul style="list-style-type: none"><li>• Facilitates data dependency resolution by providing general-purpose register value.</li></ul>

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

# Soft SoCs

Terasic DE0-Nano board



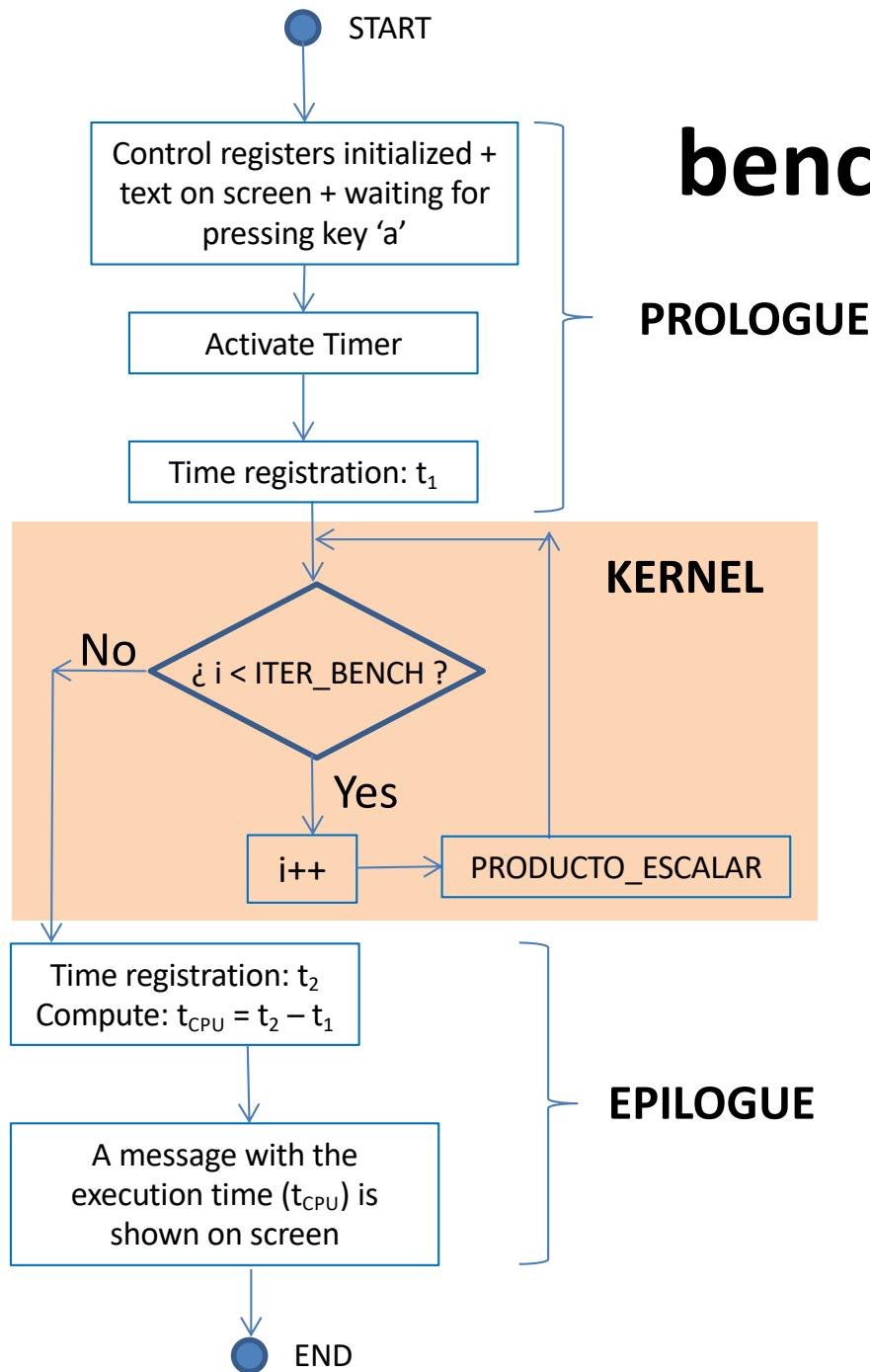
# Part 1: Analysis of the usage of instructions types in a benchmark program and the CPI of Nios V/{m,g} processors

**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: Objectives

- Objective 1: 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.
- Objective 2: 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.
- Objective 3: 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.

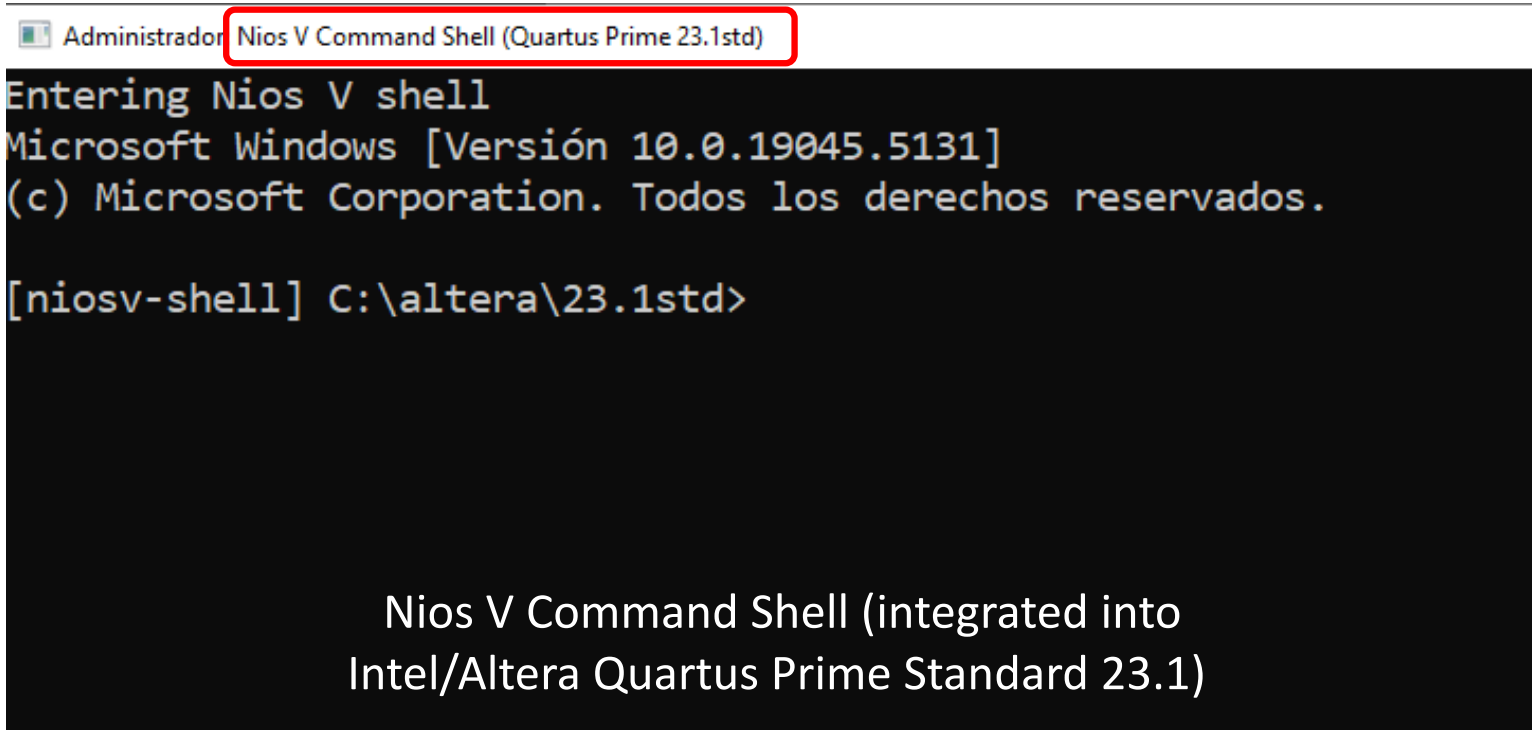
# Part 1: hands-on exercises using Nios V/m

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



The screenshot shows a Windows command prompt window titled "Administrador Nios V Command Shell (Quartus Prime 23.1std)". The window has a black background with white text. The text inside the window reads: "Entering Nios V shell", "Microsoft Windows [Versión 10.0.19045.5131]", "(c) Microsoft Corporation. Todos los derechos reservados.", and "[niosv-shell] C:\altera\23.1std>".


```
Administrador Nios V Command Shell (Quartus Prime 23.1std)
Entering Nios V shell
Microsoft Windows [Versión 10.0.19045.5131]
(c) Microsoft Corporation. Todos los derechos reservados.

[niosv-shell] C:\altera\23.1std>
```

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	
...		...		...		...	
...		...		...		...	
<b>Total ALU instructions</b>		<b>Total MEMORY instructions</b>		<b>Total BRANCH &amp; JUMP instructions</b>		<b>Total OTHER instructions</b>	
<b>N</b> (total number of executed instructions)							
<b>% ALU</b>		<b>% MEMORY</b>		<b>% BRANCH &amp; JUMPS</b>		<b>% OTHER</b>	
<b>Cicles</b> <b>Nios V/m</b>				<b>Cicles</b> <b>Nios V/g</b>			
<b>Total CPI of program</b> <b>Nios V/m</b>				<b>Total CPI of program</b> <b>Nios V/g</b>			

# 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**

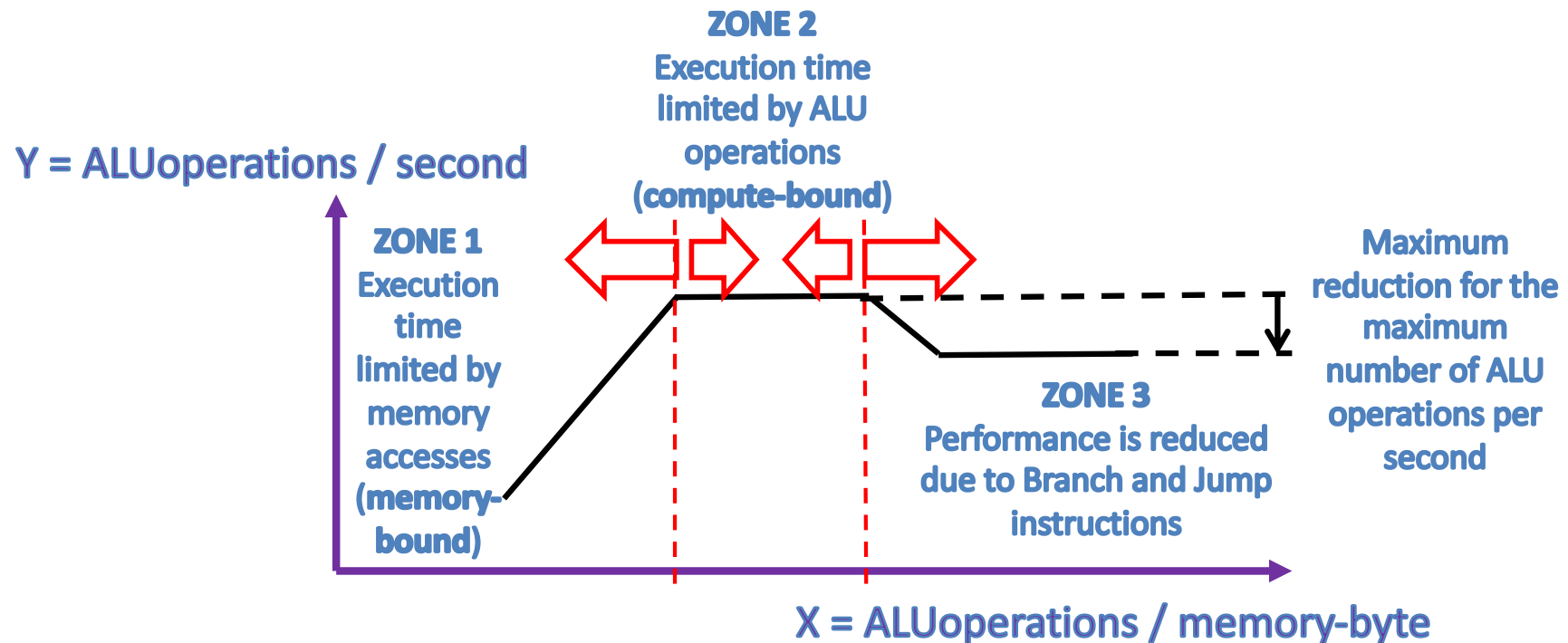
General Description. This part evaluates the limits of each Nios V/{m,g} soft processor in terms of the number of ALU operations it can perform in each unit of time. In particular, the limits measured in 'ALU operations/second' caused by the ALU functional unit of the processors, the memory hierarchy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



## Parte 2: Objective

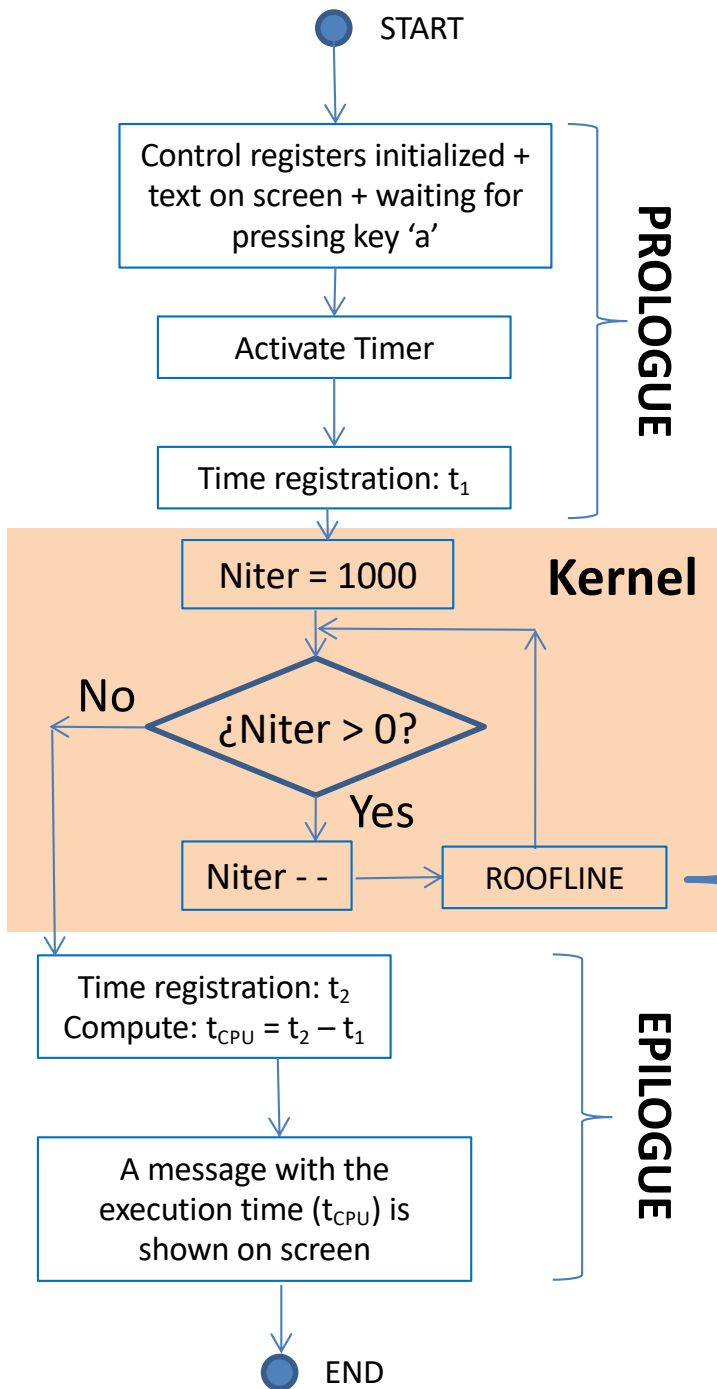
- Obtain the curve 'ALU operations/second " versus "ALU operations/byteMEMORY' (see Figure "roofline"). This curve represents the performance level of the Nios V processor measured in number of ALU operations performed per unit time.

# “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
  
```

ID	kernel: roofline.s			X coordinate ALUoperations/ memory-byte (ALU / 4 * lw/sw)	Iterations (N <sub>iter</sub> )	N (executed instructions, N <sub>iter</sub> * [lw/sw+ALU+ br/j])	Nios V/{m,g}, frequency (f)= 50 MHz			
	Number of executed instructions per iteration						cycles	t <sub>CPU</sub> (sec=cycles / f)	Y coordinate ALUoperations/sec (N <sub>iter</sub> * ALU / t <sub>CPU</sub> )	CPI (cycles / N)
	lw/sw	ALU	br/j							
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					

**Part 2:**  
**Table 2**  
(one for Nios V/m  
and another for  
Nios V/g)

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



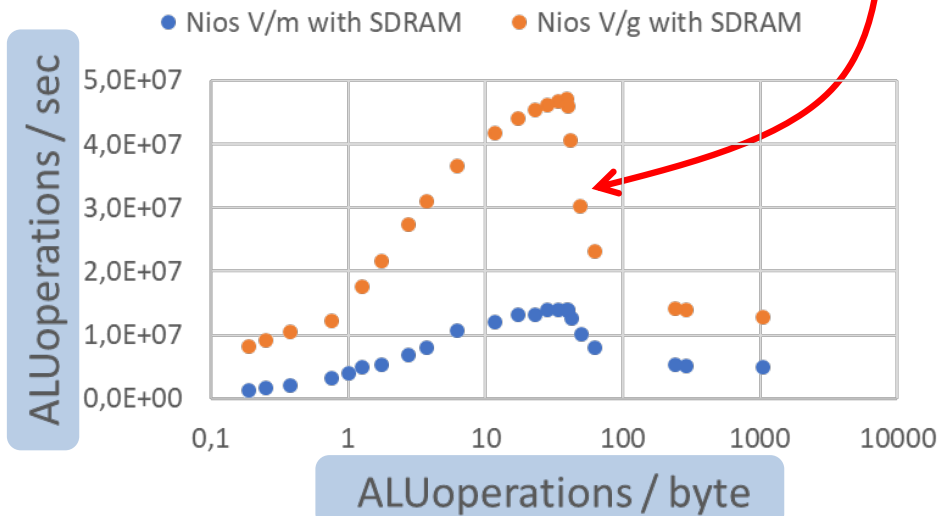
Nios V/g

vs.

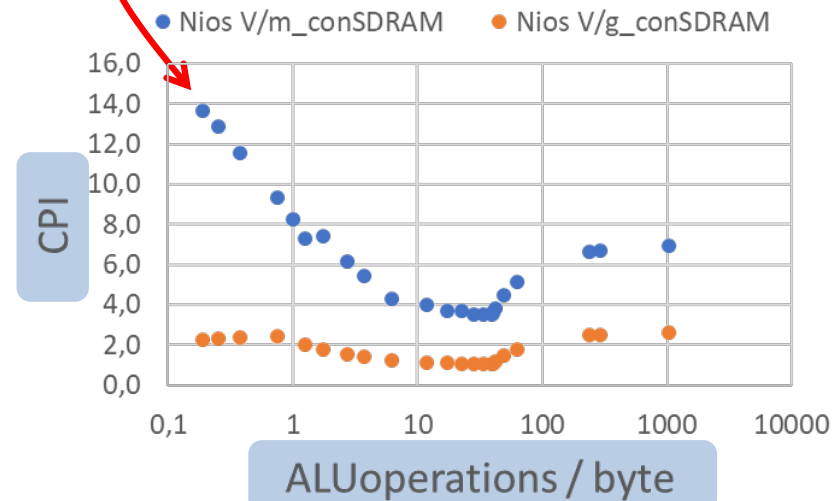
Nios V/m



ROOFLINE curve: highest soft processor performance



cycles per instruction, CPI



# 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.

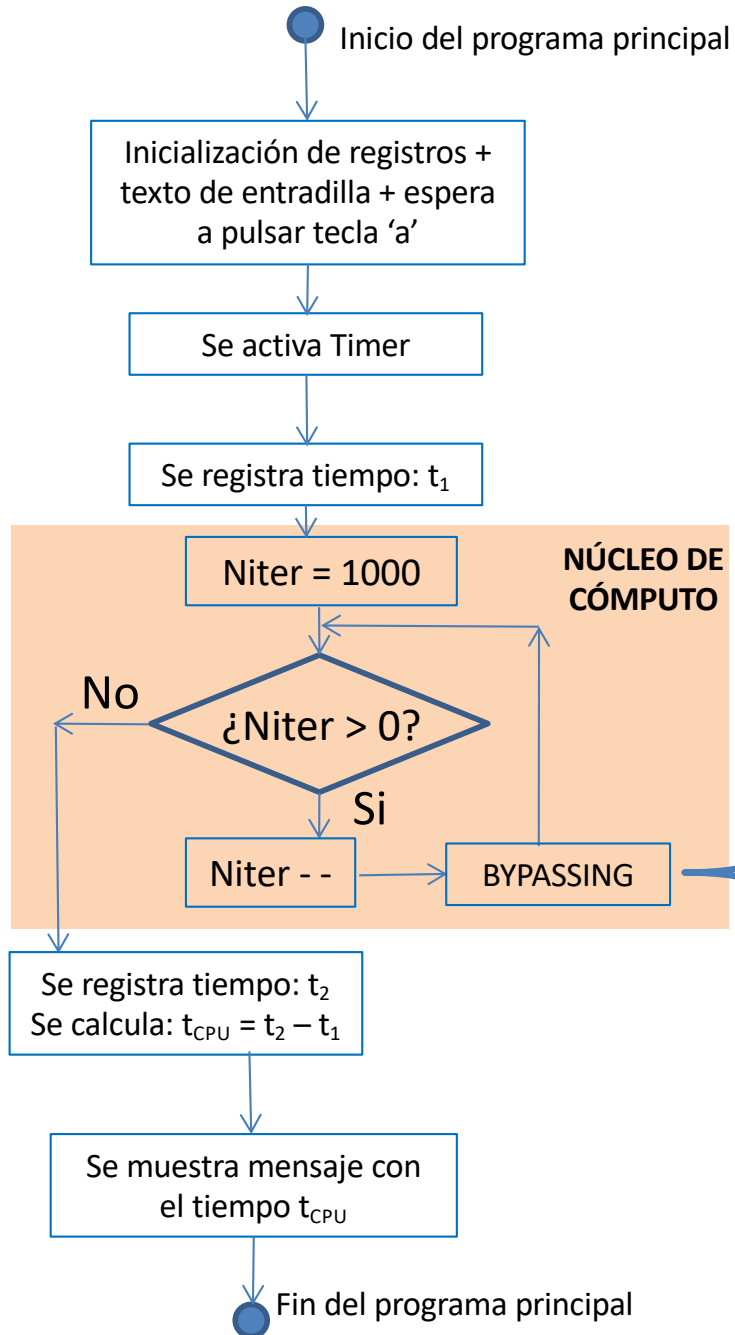
# Parte 3. Análisis de los efectos que ocasiona la reordenación de instrucciones en procesadores segmentados Nios II/f

Descripción general. Se utilizará un nuevo programa benchmark sintético que denominamos *benchNIOsII2021\_Parte3* para evaluar el efecto de las dependencias de datos verdaderas RAW entre instrucciones de carga e instrucciones ALU. Este programa benchmark es similar al de las partes 1 y 2 de esta práctica excepto que se modifica el núcleo de cómputo que ahora se encuentra en un fichero llamado `bypassing.s`.

Posteriormente, se propone aplicar la técnica de reordenación de instrucciones para reducir el tiempo de ejecución del programa benchmark usando el procesador Nios II/f.

# PARTE 3:

## OSII2021\_Parte3.s



# VERSIÓN 1

LOOP:

```
ldw      r6, 0(r2) /* carga A */
```

```
/* ZONA de dependencia de datos */
```

```
/* suma CON dependencia de datos con ldw r6, 0(r2) */
```

```
add    r6, r6, r6
```

```
/* 2 instrucciones mas de tipo ALU, nunca se comentan */
```

```
addi    r7, r7, 1 /* Niter++ */
```

```
subi    r4, r4, 1 /* N-- */
```

```
/* fin del bucle LOOP*/
```

```
bgt      r4, r0, LOOP
```

## VERSIÓN 2

LOOP:

```
ldw      r6, 0(r2) /* carga A */
```

```
/* ZONA de dependencia de datos */
```

```
/* suma SIN dependencia de datos con ldw r6, 0(r2) */
```

```
add    r6, r4, r4
```

```
/* 2 instrucciones mas de tipo ALU, nunca se comentan */
```

```
addi    r7, r7, 1 /* Niter++ */
```

```
subi    r4, r4, 1 /* N-- */
```

```
/* fin del bucle LOOP*/
```

```
bgt      r4, r0, LOOP
```



## Ejercicio Parte 3: **VERSIÓN 3**

- Reordenar las instrucciones del bucle `LOOP` de la Versión 1 para reducir el tiempo de cómputo total del programa ( $t_{CPU}$ )
- Medir el tiempo de ejecución ( $t_{CPU}$ ) en Nios II/f
- Medir el speed-up
- Medir el CPI total
- Medir el CPI de penalización ocasionado por la dependencia verdadera RAW de la pareja de instrucciones `ldw -> add`

# PARTE 4: NUEVO DISEÑO DE PROCESADOR SEGMENTADO

Procesador original 6 segmentos

Nuevo Procesador 5 segmentos

```
ldw  r1, inmediato(r2)
      |
      | inmediato=0
      v
ldw  r1, r2
      |
      | inmediato!=0
      v
addi  r2, r2, inmediato
ldw  r1, r2
```

¿Cuál es el porcentaje adicional de instrucciones que se tienen que ejecutar en el nuevo procesador suponiendo que se ejecuta la subrutina PRODUCTO\_ESCALAR de la Parte 1 de esta práctica?