Lab Assignment 2:

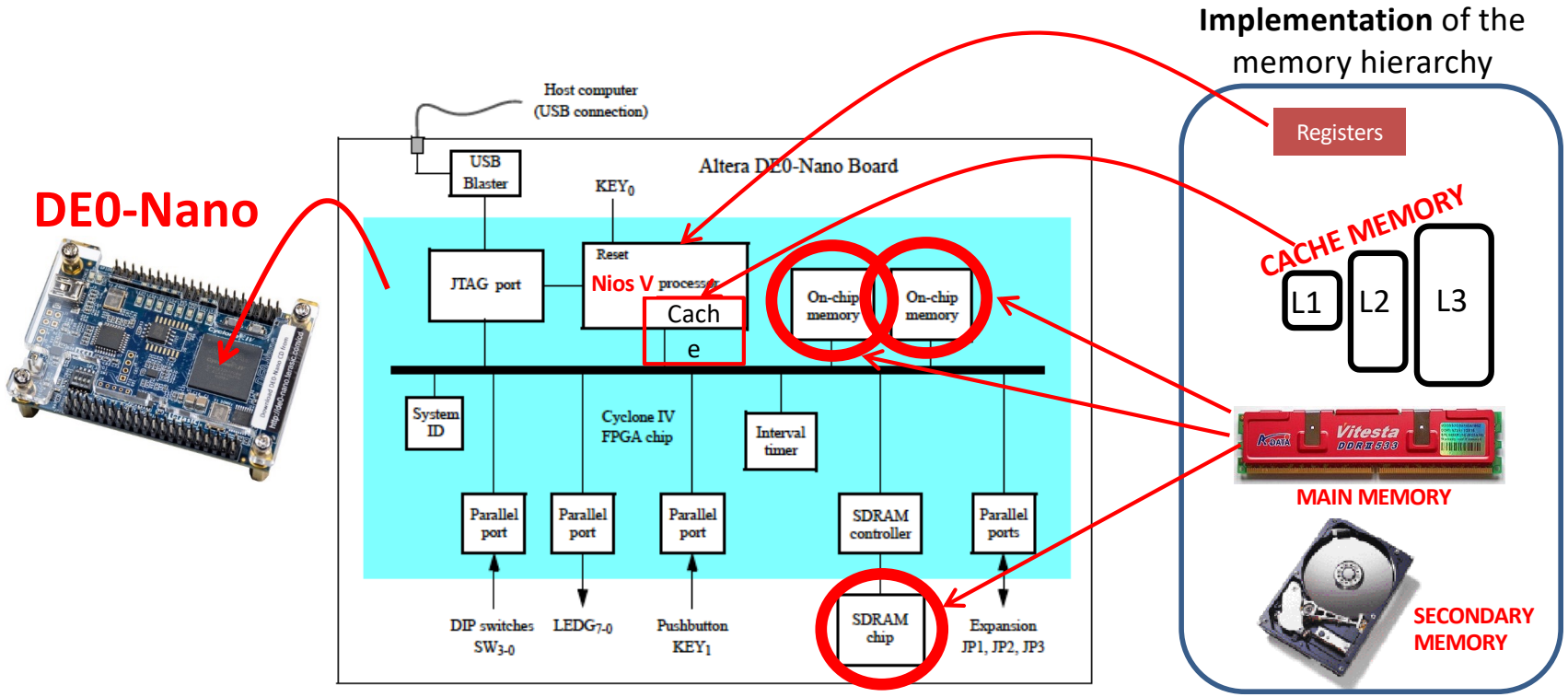
Performance evaluation of the memory hierarchy of a computer and reverse engineering of the data cache memory

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

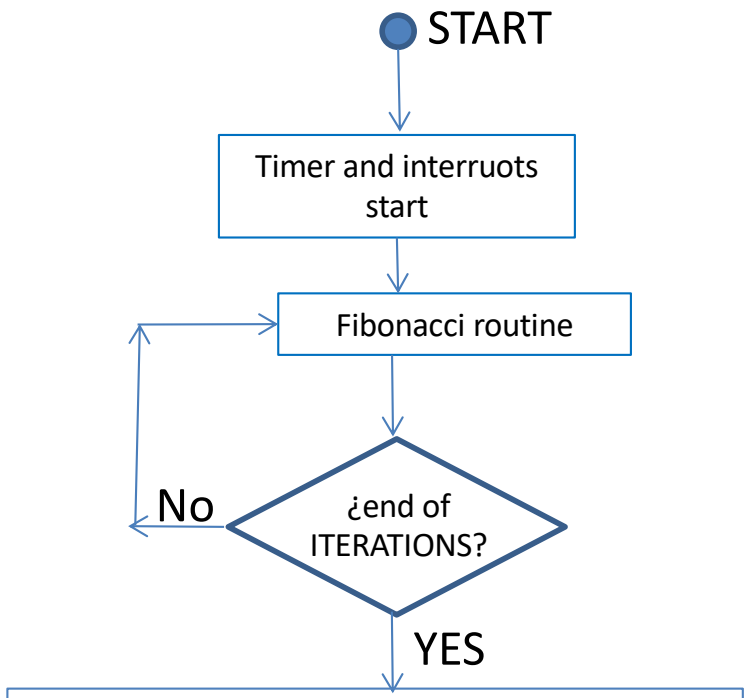
Scheduling: 4 weeks

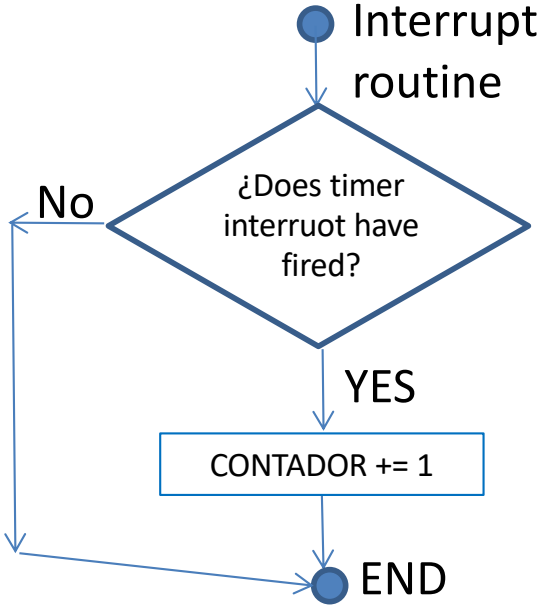
- Session 1: Activities 1,2,3; 2 hours
- S2: Activity 4 ; 2 hours
- S3: Reverse engineering for the data cache of Nios V/g; 2 hours
- S4: Examn; 1,5 hours

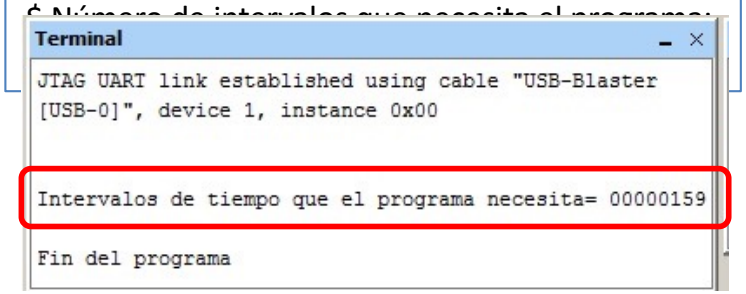
Implementing the Memory Hierarchy Levels of the Basic Computer Structure of the DE0-Nano board



Activity 1: benchmark







Activity 1: measuring execution time

Soft processor model + memory technology	Execution time	Speed-up
Nios V/m + SDRAM memory (Activity 1)		1X
Nios V/m + on-chip memory (Activity 2)		
Nios V/g + SDRAM memory (Activity 3)		
Nios V/g + on-chip memory (Activity 3)		

Activity 4: discovering data cache microarchitecture

Source code for the main program:

`lab2_part1_2_3_main.s`

```
                ...
                movia r14, ITERACIONES    /* inicializa el contador de iteraciones LOOP, cada una de ellas ejecuta un bucle 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
```

Source code for the routine FIBONACCI:

`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
...
```

Activity 4

Modify source code in the file:
`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
...
```

X: data size accessed by program :
X = P × E

E: number of accesses to main memory

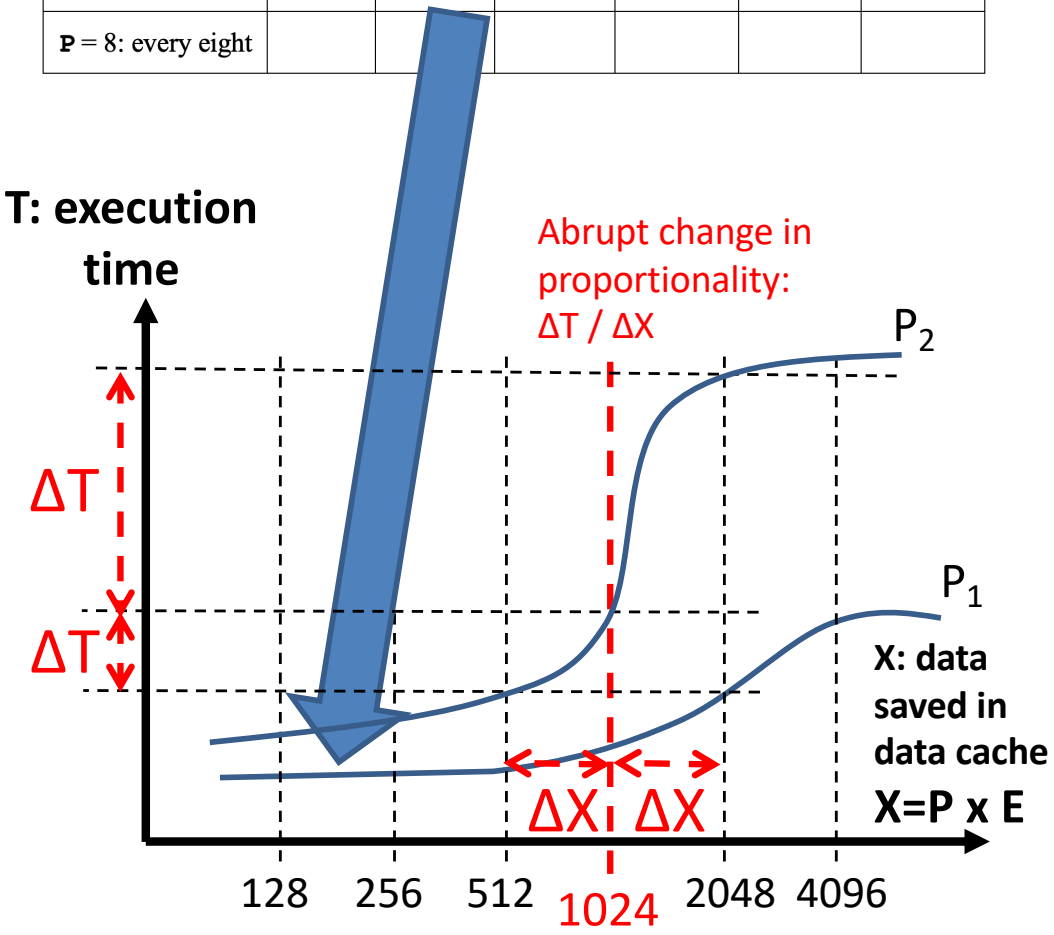
Table 3. Table used in Activity 4 to collect run time measurements.

	E: number of V vector bytes accessed actually					
P: data access pattern	128	256	512	1024	2048	4096
P = 1: everyone						
P = 2: every two						
P = 4: every four						
P = 8: every eight						

Activity 4

Table 3. Table used in Activity 4 to collect run time measurements.

	E: number of V vector bytes accessed actually					
P: data access pattern	128	256	512	1024	2048	4096
P = 1: everyone						
P = 2: every two						
P = 4: every four						
P = 8: every eight						



Method to fill in Table 3

1) Ejecutar AMP
2) New project

CAMBIA-CACHE:

3) Settings > System Settings > "Custom System" + browse > `<file>.sopcinfo` (System information file)
4) Settings > System Settings > "Custom System" + browse > `<file>.sof` (Quartus II Programming File)
5) Memory Settings > Memory device > `SDRAM`
6) Actions > Download System

CAMBIA-DATOS:

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

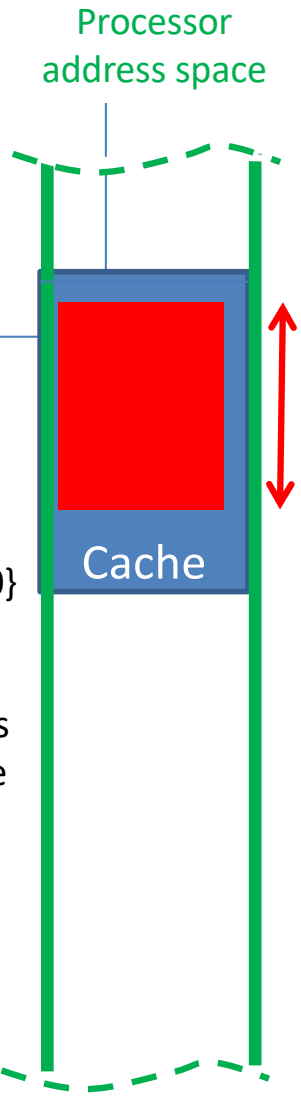
SEGUIR CAMBIA-DATOS

SEGUIR CAMBIA-CACHE

¿Data cache size?: P=1

```
        ...
        movi r4, 0
        movi r5, X
LOOP:   bge  r4, r5, END
        ldb  r0, V(r4)
        addi r4, r4, 1
        br   LOOP
END:
        ...
.data
V:
.skip 65536
...
```

- **X sí** cabe en la cache de datos.
- Sólo existen fallos en ITERACION=1.
- En ITERACION={2,…,50000} los accesos aciertan en la cache.
- Tiempo de ejecución (T) es proporcional al número de accesos a la memoria (E).

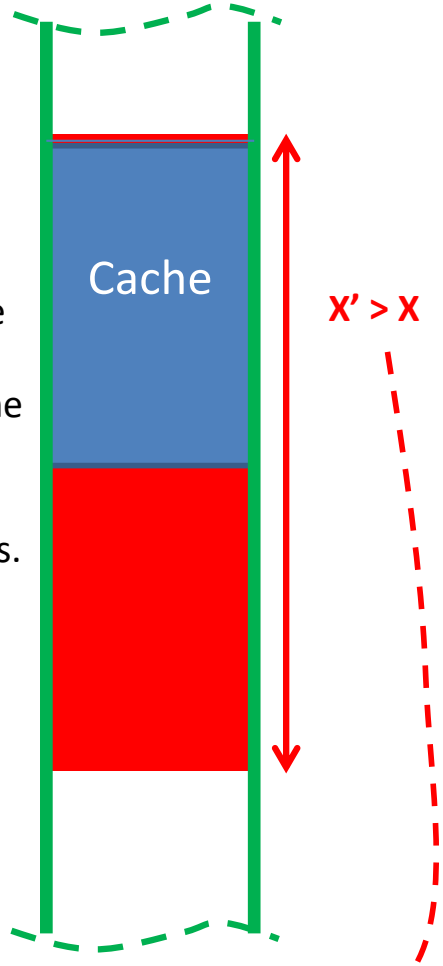


Case Tyoe-1: **NO-overflow**

X: volúmenes de datos que maneja el programa; los bytes del vector V accedidos desde el programa caben en la memoria cache de datos.

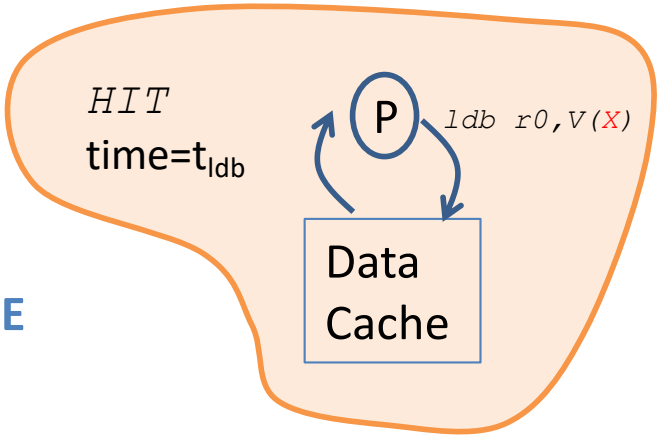
Case Tyte-2: **OVERFLOW**

- **X' no** cabe en la cache de datos.
- Muchos fallos en cache de datos por remplazamientos en todas las ITERACIONes.
- Tiempo de ejecución NO tiene la misma proporción que en X inferiores.



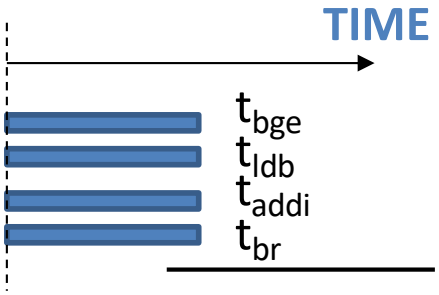
- **CLAVE:** encontrar X' para P=1 que hace que el tiempo de ejecución no guarda la proporción con el número de accesos E que en X inferiores; la capacidad es X'/2

Execution time **WITHOUT** cache misses after the first LOOP ITERATION




```
        ...
        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
...
```

ITERation



$t_{ITER} = \sum_i t_i$

Case Type-1:
NO-cache overflow for all iterations except the first

$t_{ldb}\ (average) = t_{hit}$

$$X \rightarrow 2X$$

$$\sum^{2X} t_{ITER} \approx 2 \times \sum^{X} t_{ITER}$$

Double the number of ITERations (2X) should cause the program to take twice as long

Execution time **WITH** cache misses

Case Type-2:

Overflow occurs in all ITERations

```
        ...
        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
...
```

ITERtion

TIME

$t_{bge'} = t_{bge}$

$t_{addi'} = t_{addi}$

$t_{br'} = t_{br}$

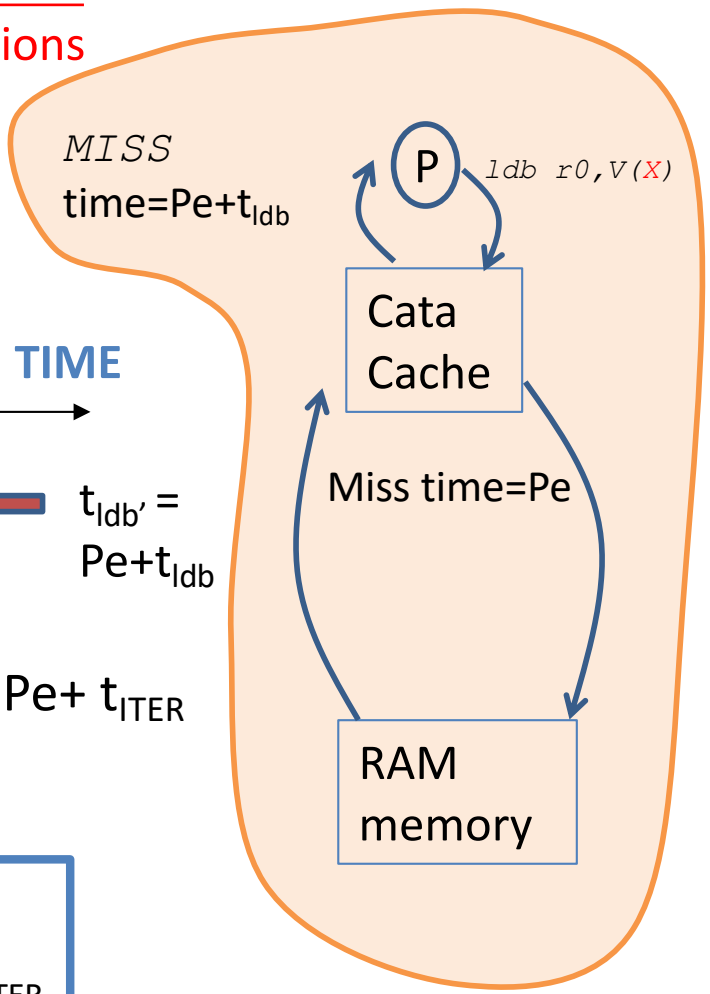
$t_{ldb'} = Pe+t_{ldb}$

$t'_{ITER} = \sum_{i'} t_{i'} = Pe+\sum_i t_i = Pe+ t_{ITER}$

$t_{ldb'}(AMAT) = t_{hit} + MissRate \times Penalization$

$X \rightarrow 2X$

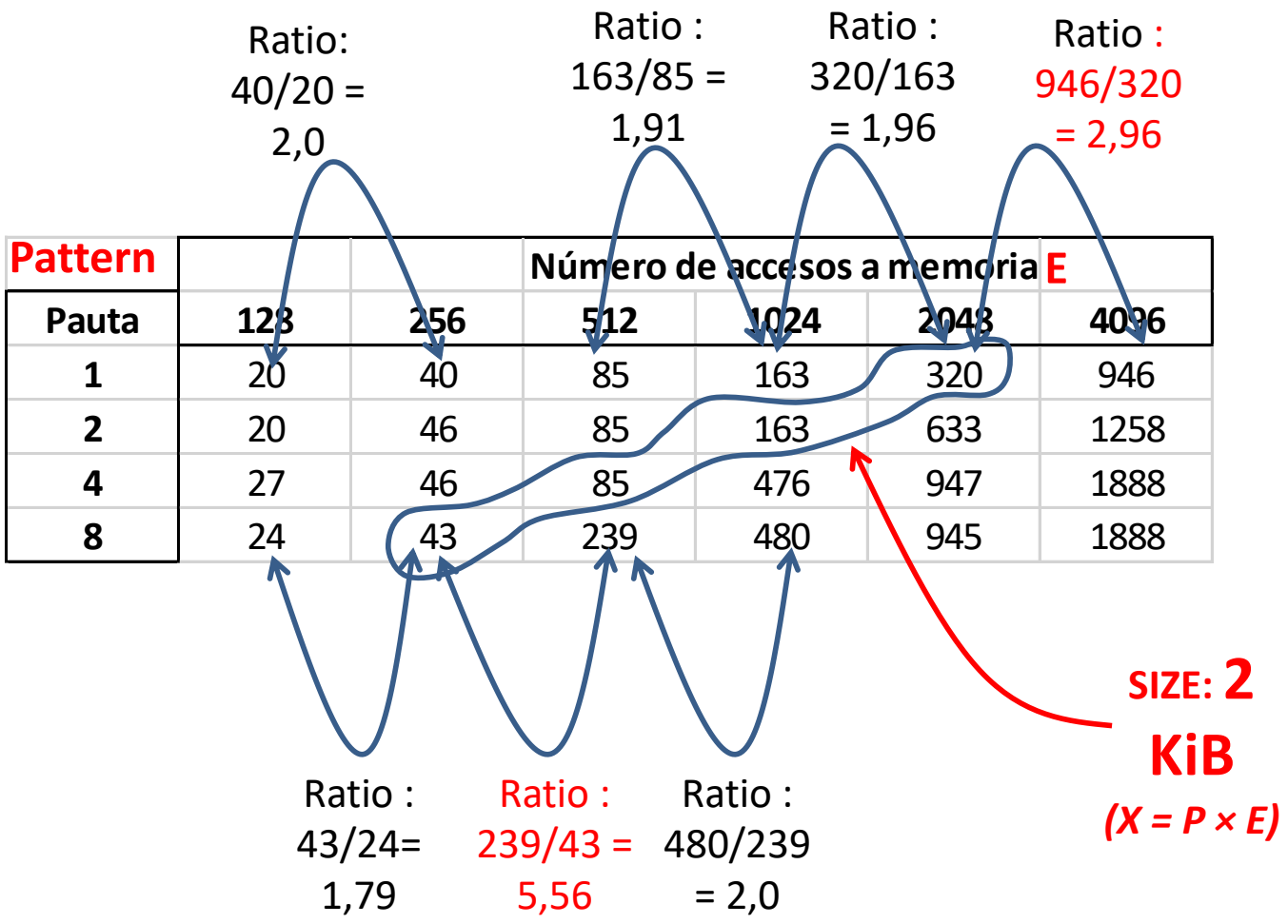
$\sum^{2X} t_{ITER} > 2 \times \sum^{X} t_{ITER}$



Double the number of ITERations (2X) should cause the program to take longer than **twice** the time with half the number of ITERations (X).

¿How do you discover cache size?: P=1

Board
DE0-Nano



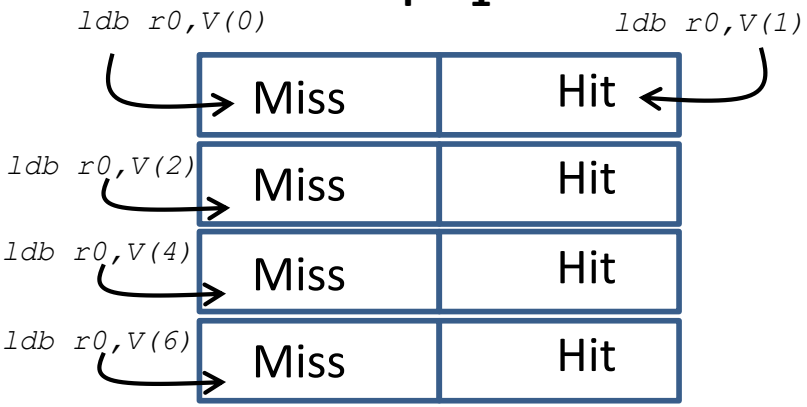
¿How do you discover block size?

```
        ...
        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
...
```

- Assume the case where the block size is 2 bytes.
- Assume that X is large enough so that the cache memory is always overflowing.

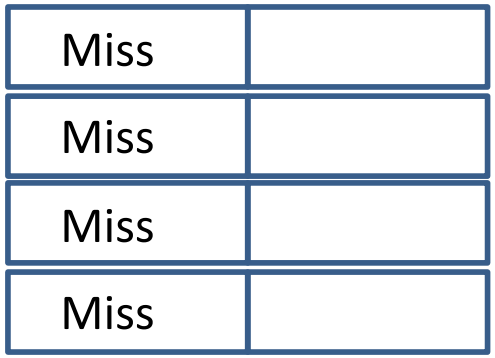
BLOCK:
2 bytes

P = 1



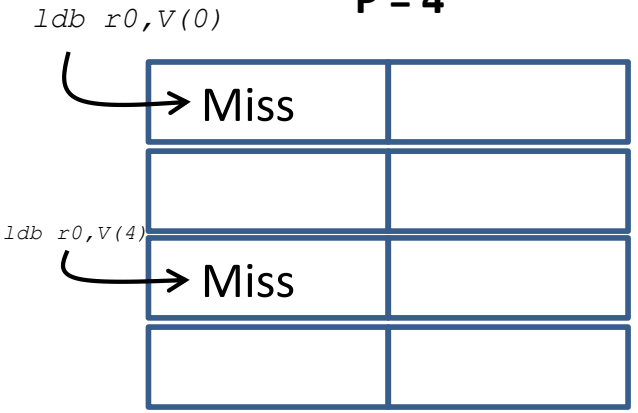
Some accesses hit in the cache → **Execution time <u>lower than cases</u> P=2,4**

P = 2



All accesses miss in the cache → **Execution time <u>larger tan case</u> P=1**

P = 4



All accesses miss in the cache → **Execution time <u>similar to case</u> P=2**

¿How do you discover block size?

Look at the same column of accesses and make sure that the cache is overflowing to ensure that at least capacity misses occur

Board DE0-Nano

Pattern	Número de accesos a memoria E					
	128	**256**	**512**	**1024**	**2048**	**4096**
1	20	40	85	163	320	946
2	20	46	85	163	633	1258
4	27	46	85	476	947	1888
8	24	43	239	480	945	1888

BLOCK:
4 B

Ratio:
480/476
= 1,0

Ratio:
476/163
= 2,9