ARCOS Group

uc3m | Universidad Carlos III de Madrid

L4: The processor (2/2)

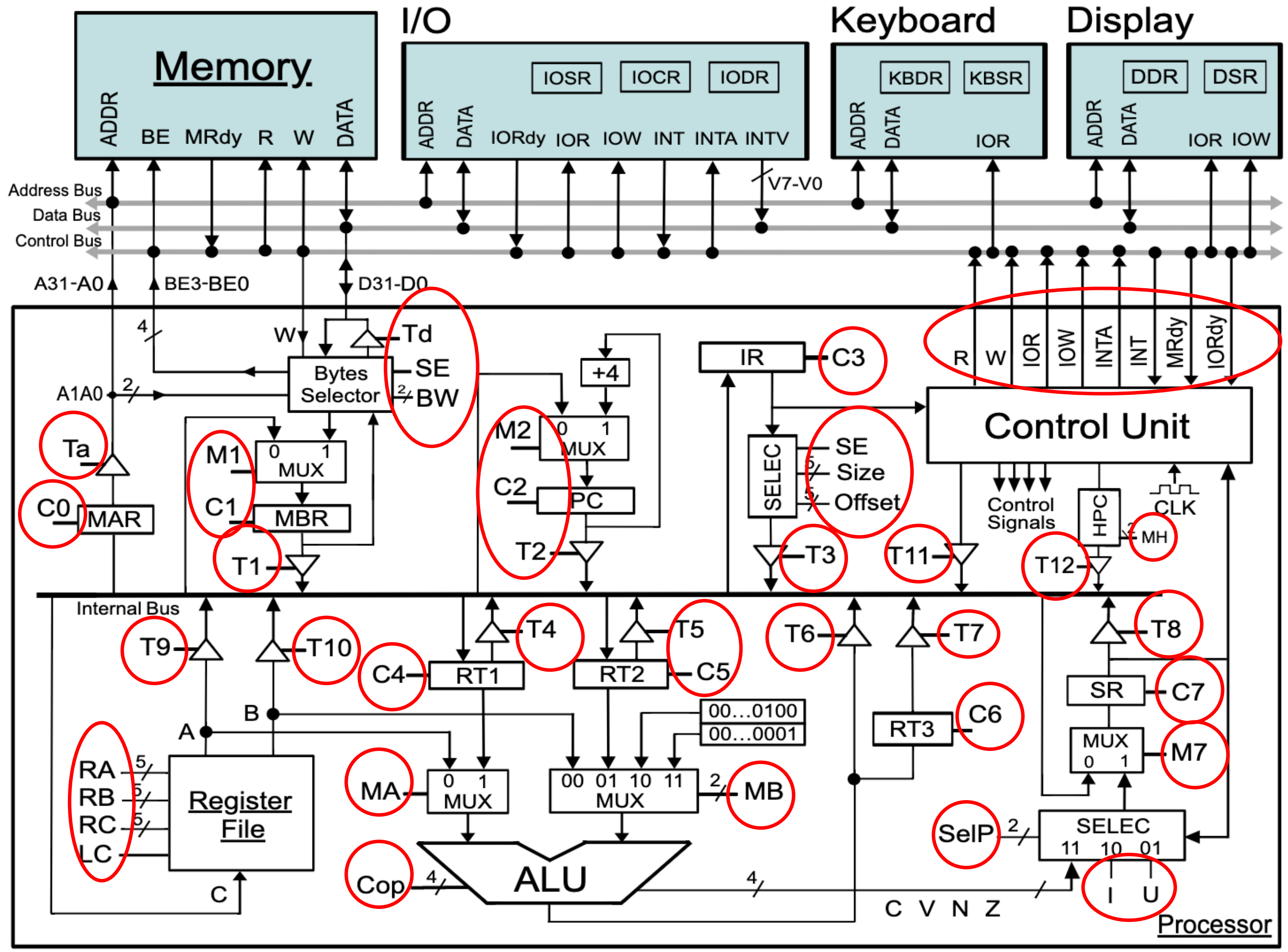
Computer Structure

Bachelor in Computer Science and Engineering
Bachelor in Applied Mathematics and Computing
Dual Bachelor in Computer Science and Engineering and Business Administration

Contents

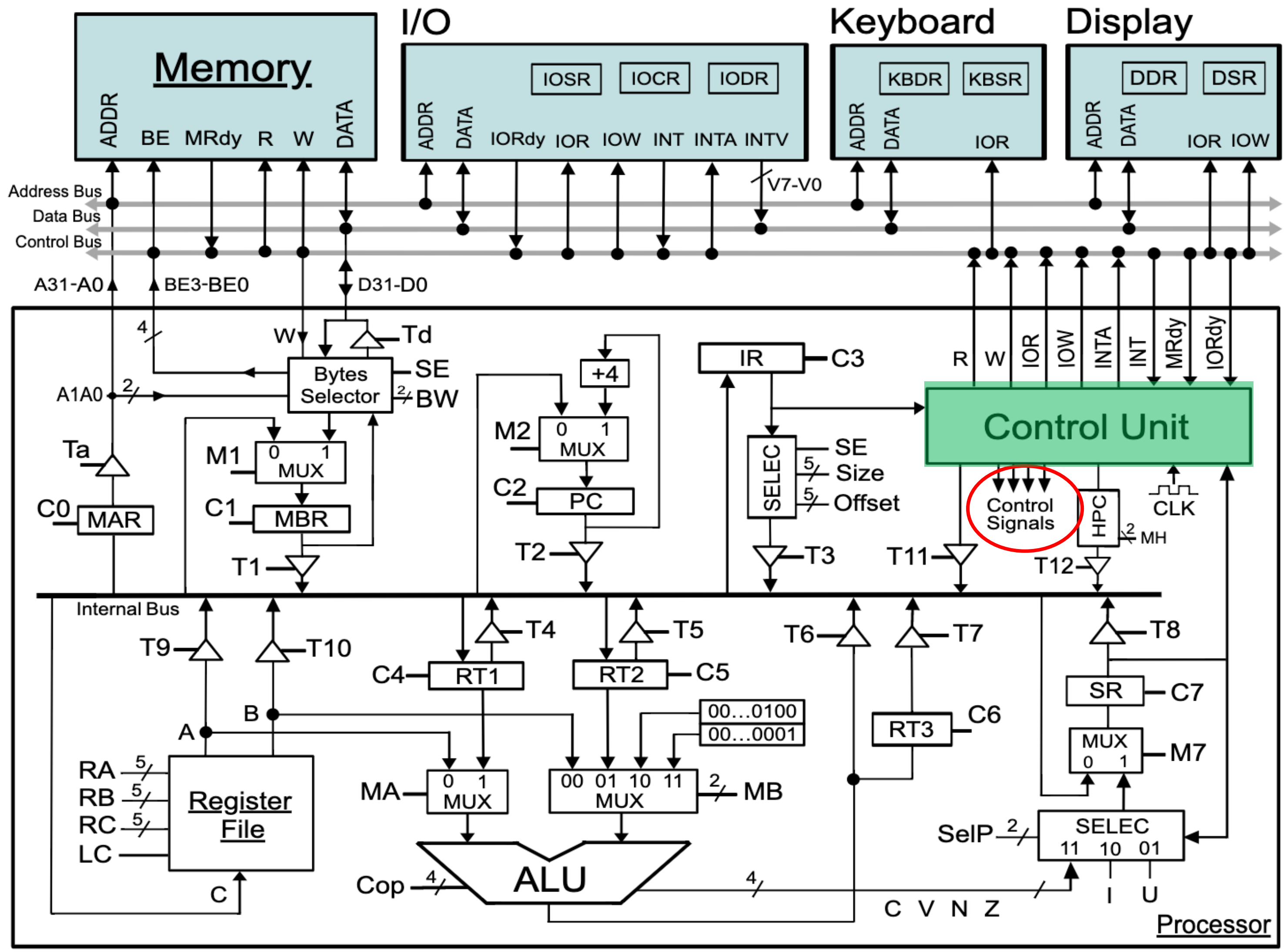
Control signals

Reminder

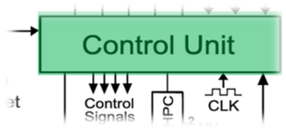


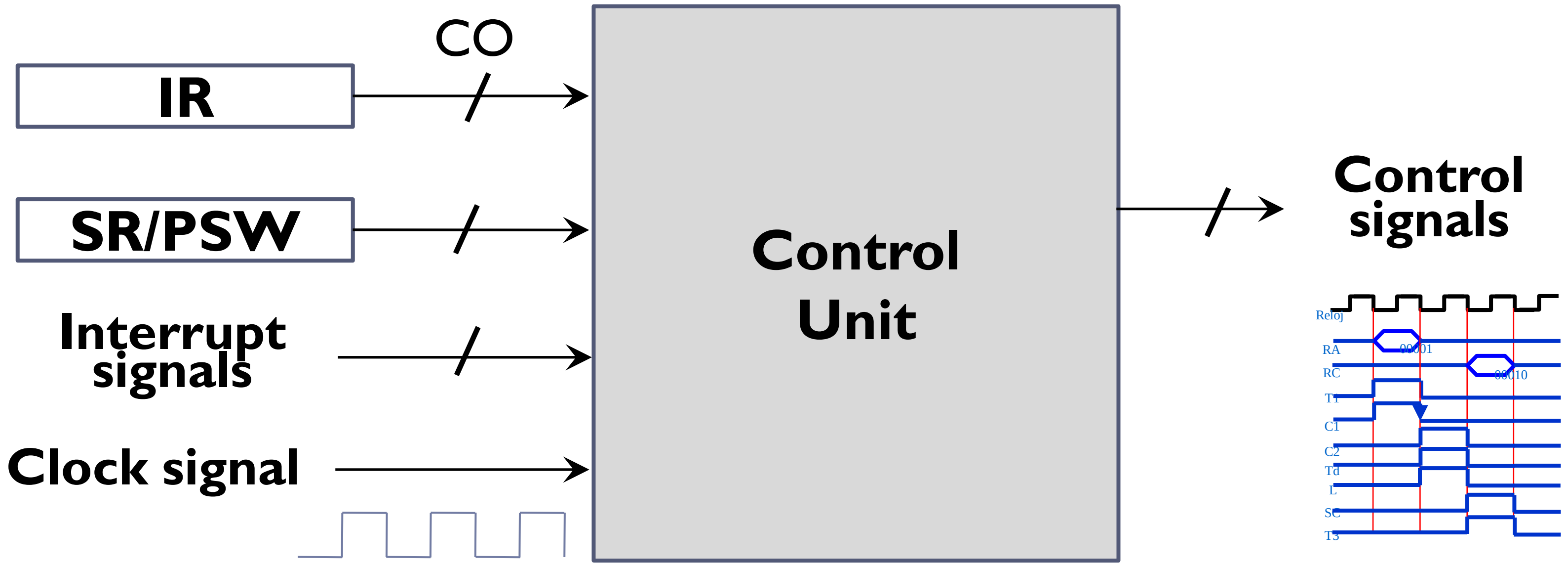
Control unit

Reminder



Control unit





- Every **control signal** is **function** of the values **of:**
  - The content of the **IR**
  - The content of **SR**
  - The **period of time (clock)**

Control unit design

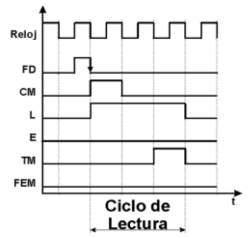
- For each machine instruction:
  1. Define the behavior using RTL (register transfer language) for every clock cycle
  2. Translate the behavior to values of each control signal at each clock cycle
  3. Design a circuit that generates the value of each control signal at each clock cycle

Instruction *mv R0 R1*

Sequence of **elementary operations**

1. IR <- [PC]
2. PC++
3. decode
4. *R0* <- *R1*

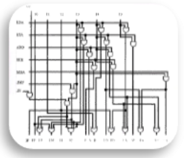
Sequence of **control signals** for each elementary operation



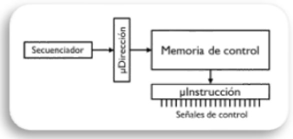
Circuit that generates signals:

a) Hardwired control
b) Microprogrammed control

a)



b)



Example

- Design of a control unit for a set of 4 machine instructions.

- Instructions to consider:
  - `add Rd, Rf:      Rd <- Rd + Rf`
  - `lw Rd, dir:      Rd <- MP[dir]`
  - `sw Rf, dir:      MP[dir] <- Rf`
  - `bz R, dir:       if (R==0) PC<- dir`

Control unit design

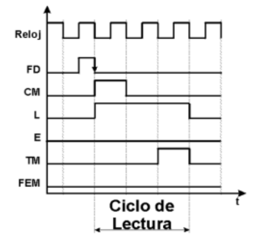
- For each machine instruction:
  1. Define the behavior using RTL (register transfer language) for every clock cycle
  2. Translate the behavior to values of each control signal at each clock cycle
  3. Design a circuit that generates the value of each control signal at each clock cycle

Instruction *mv R0 R1*

Sequence of **elementary operations**

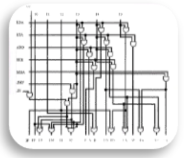
1. IR <- [PC]
2. PC++
3. decode
4. *R0* <- *R1*

Sequence of **control signals** for each elementary operation

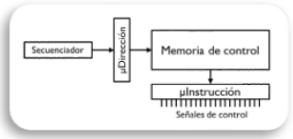


Circuit that generates signals:
a) Hardwired control
b) Microprogrammed control

a)

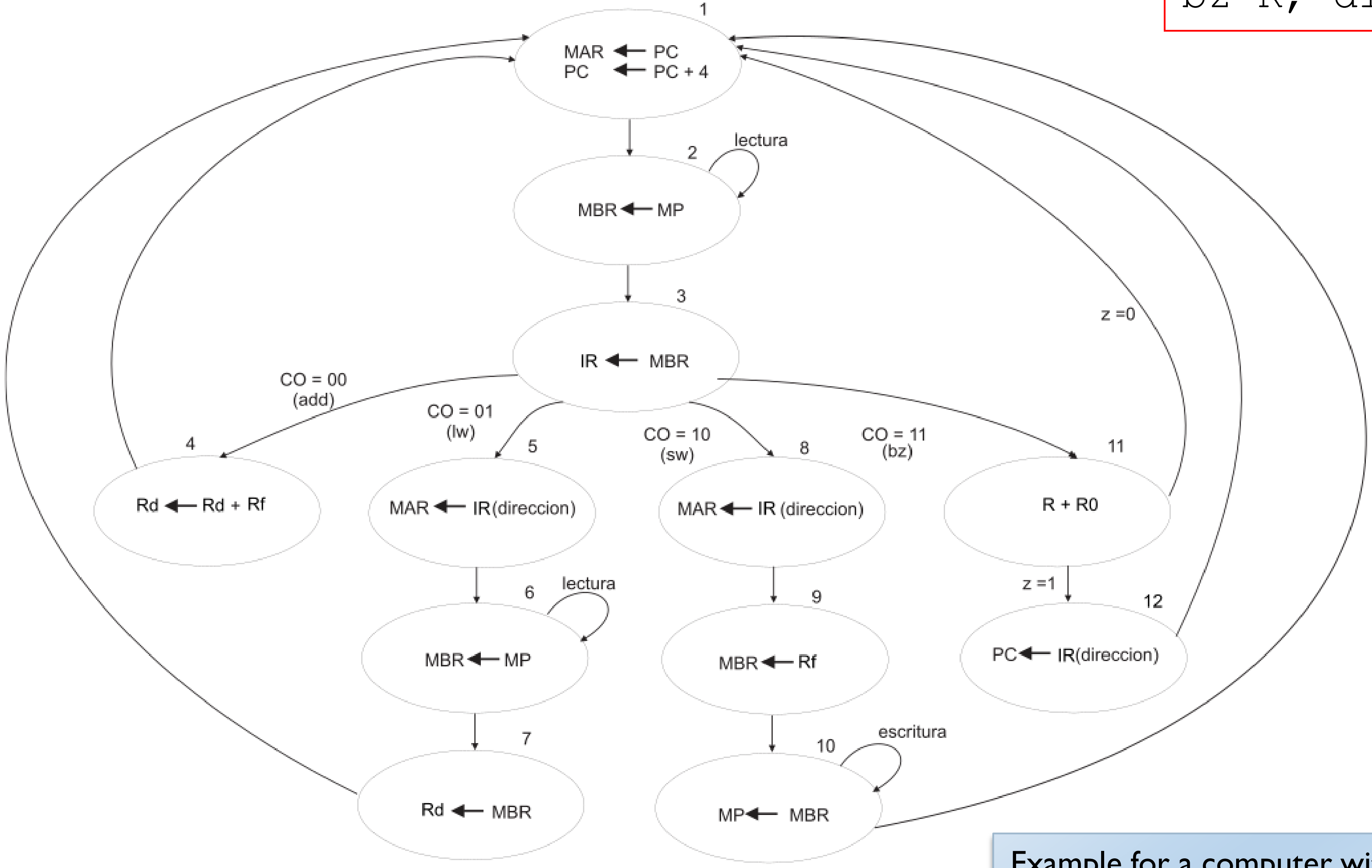


b)



State machine for the example

```
add rd, rf
lw rd, dir
sw Rf, dir
bz R, dir
```



Example for a computer with only 4 machine instructions

Control unit design

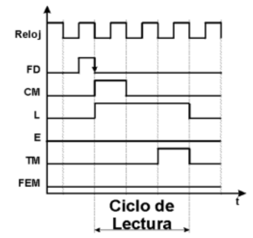
- For each machine instruction:
  1. Define the behavior using RTL (register transfer language) for every clock cycle
  2. Translate the behavior to values of each control signal at each clock cycle
  3. Design a circuit that generates the value of each control signal at each clock cycle

Instruction *mv R0 R1*

Sequence of **elementary operations**

1. IR <- [PC]
2. PC++
3. decode
4. *R0* <- *R1*

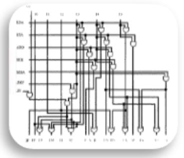
Sequence of **control signals** for each elementary operation



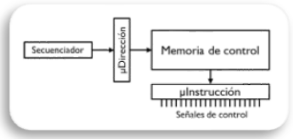
Circuit that generates signals:

a) Hardwired control
b) Microprogrammed control

a)

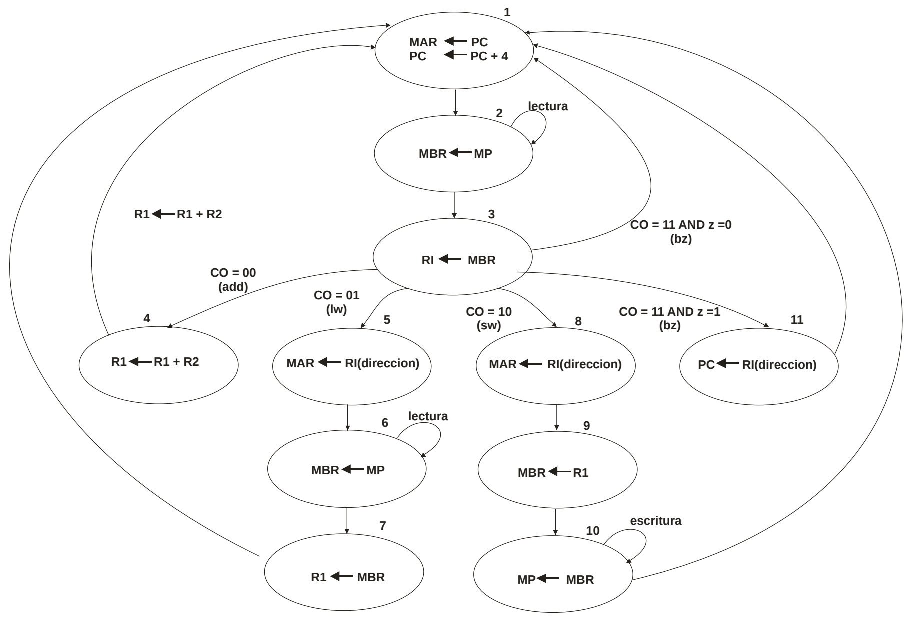


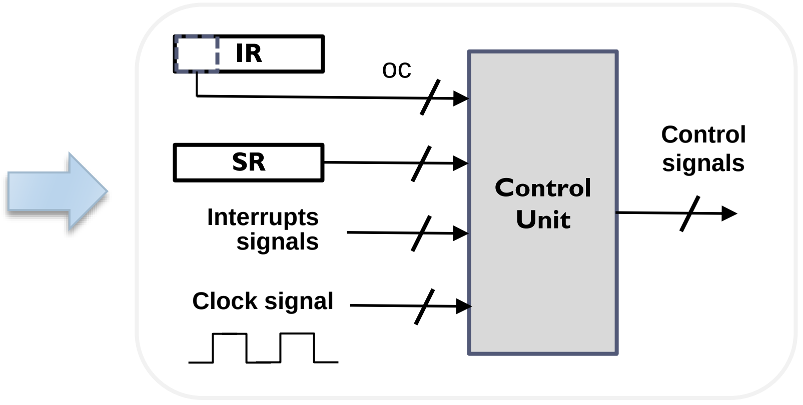
b)



Control techniques

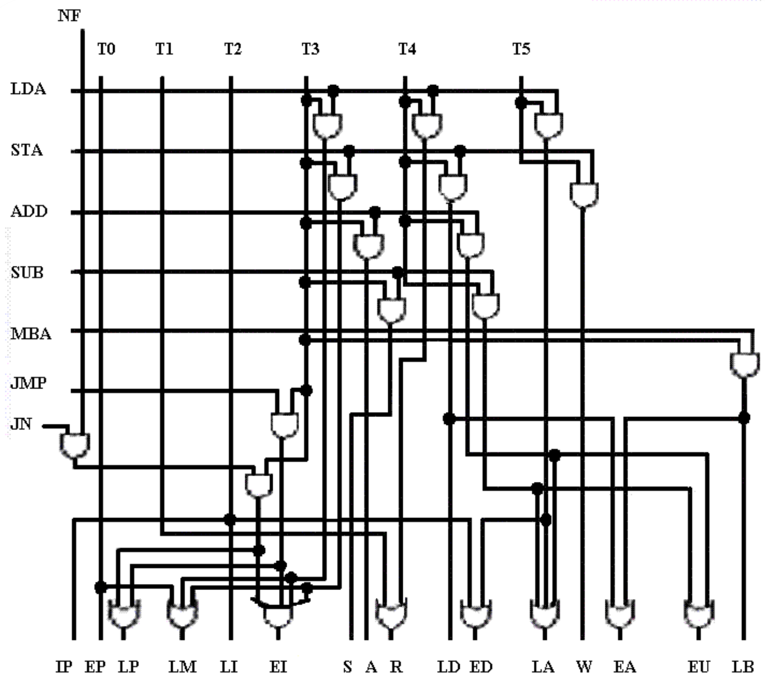
- **Two techniques** to design and build the control unit:
  a) **Relay logic**
  b) **Programmable logic (microprogrammed)**





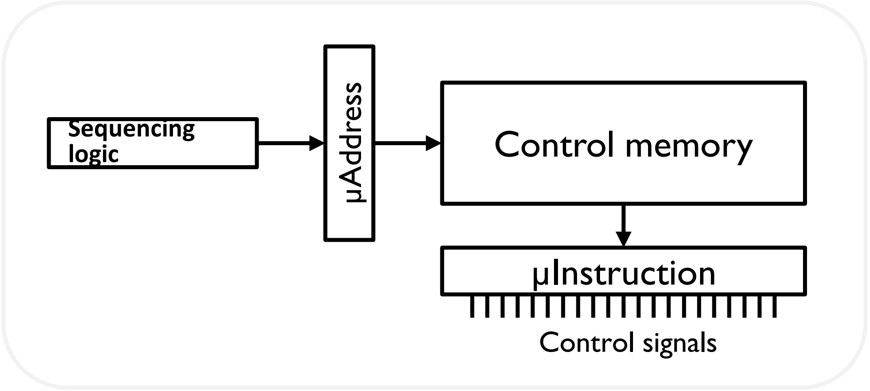
Control Unit: relay logic

- Construction by means of logic gates, following logic design methods.
- Characteristics:
  - Laborious and costly circuit design and tuning.
  - Difficult to modify:
    - Complete redesign.
  - Very fast (used in RISC computers).



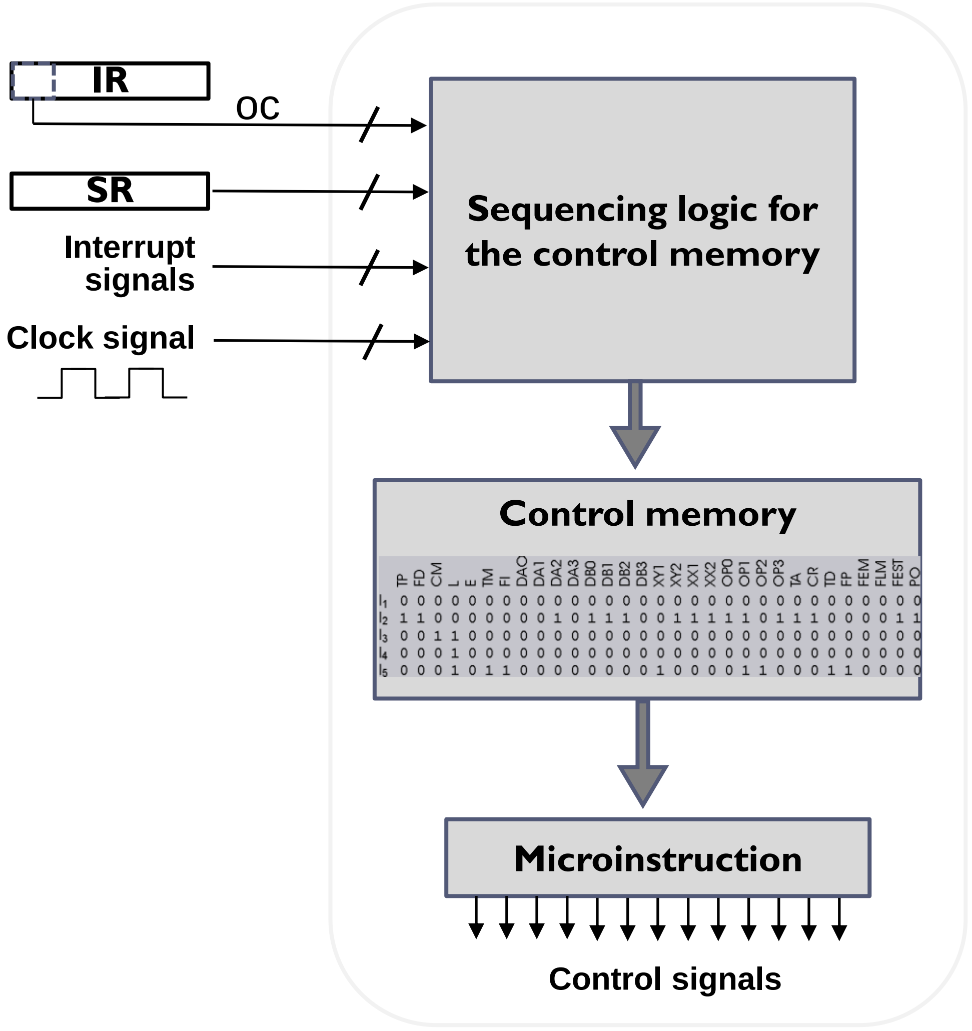
Control Unit: programmable logic
microprogramming

- Basic idea:
  Use a memory (control store)
  to store the signals of each cycle of each instruction..
- Characteristics:
  - Easy modification
    - Upgrade, expansion, etc.
    - E.g.: Certain consoles, routers, etc.
  - Easy to have complex instructions
    - E.g.: Diagnostic routines, etc.
  - Easy to have several sets of instructions
    - Other computers can be emulated.
  - Simple HW ⇒ hard microcode



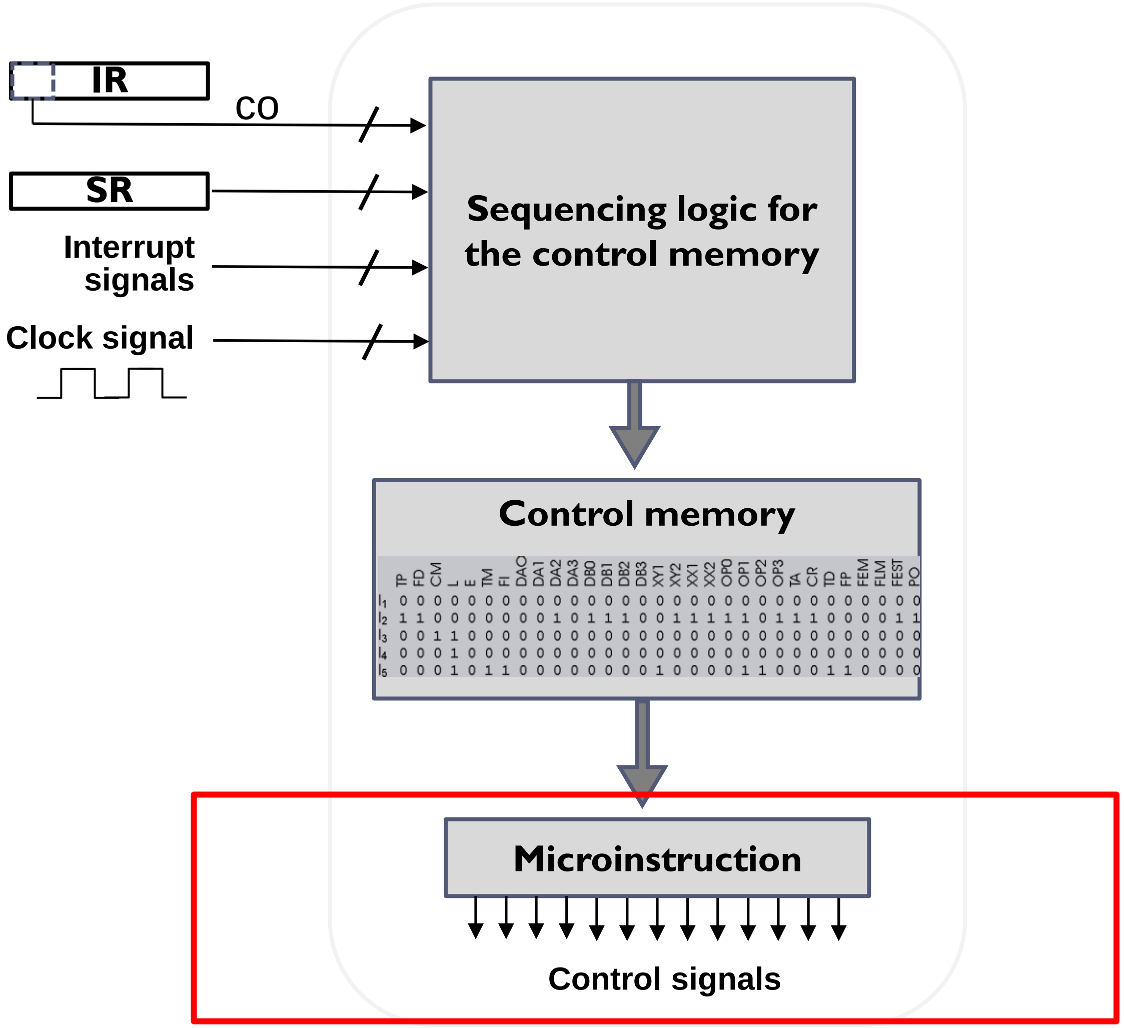
Contents

General structure of a microprogrammed control unit



	TP	FD	CM	L	E	TM	FI	DAO	DA1	DA2	DA3	DB0	DB1	DB2	DB3	XY1	XY2	XX1	XX2	OP0	OP1	OP2	OP3	TA	CR	TD	FP	FEM	FLM	FEST	PO
I_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I_2	1	1	0	0	0	0	0	0	0	1	0	1	1	1	0	0	1	1	1	1	1	0	1	1	1	0	0	0	0	1	1
I_3	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I_4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I_5	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0

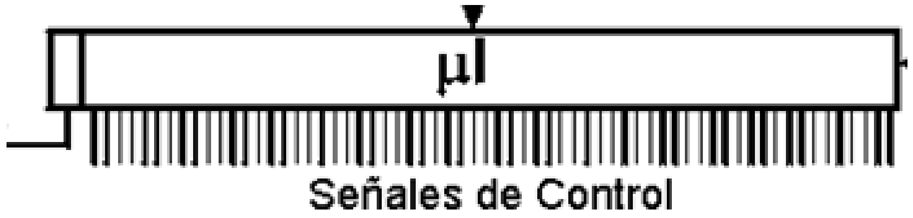
General structure of a microprogrammed control unit



	TP	FD	CM	L	E	TM	FI	DAO	DA1	DA2	DA3	DB0	DB1	DB2	DB3	XY1	XY2	XX1	XX2	OP0	OP1	OP2	OP3	TA	CR	TD	FP	FEM	FLM	FEST	PO
I_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I_2	1	1	0	0	0	0	0	0	0	1	0	1	1	1	0	0	1	1	1	1	1	0	1	1	1	0	0	0	0	1	1
I_3	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I_4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I_5	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0

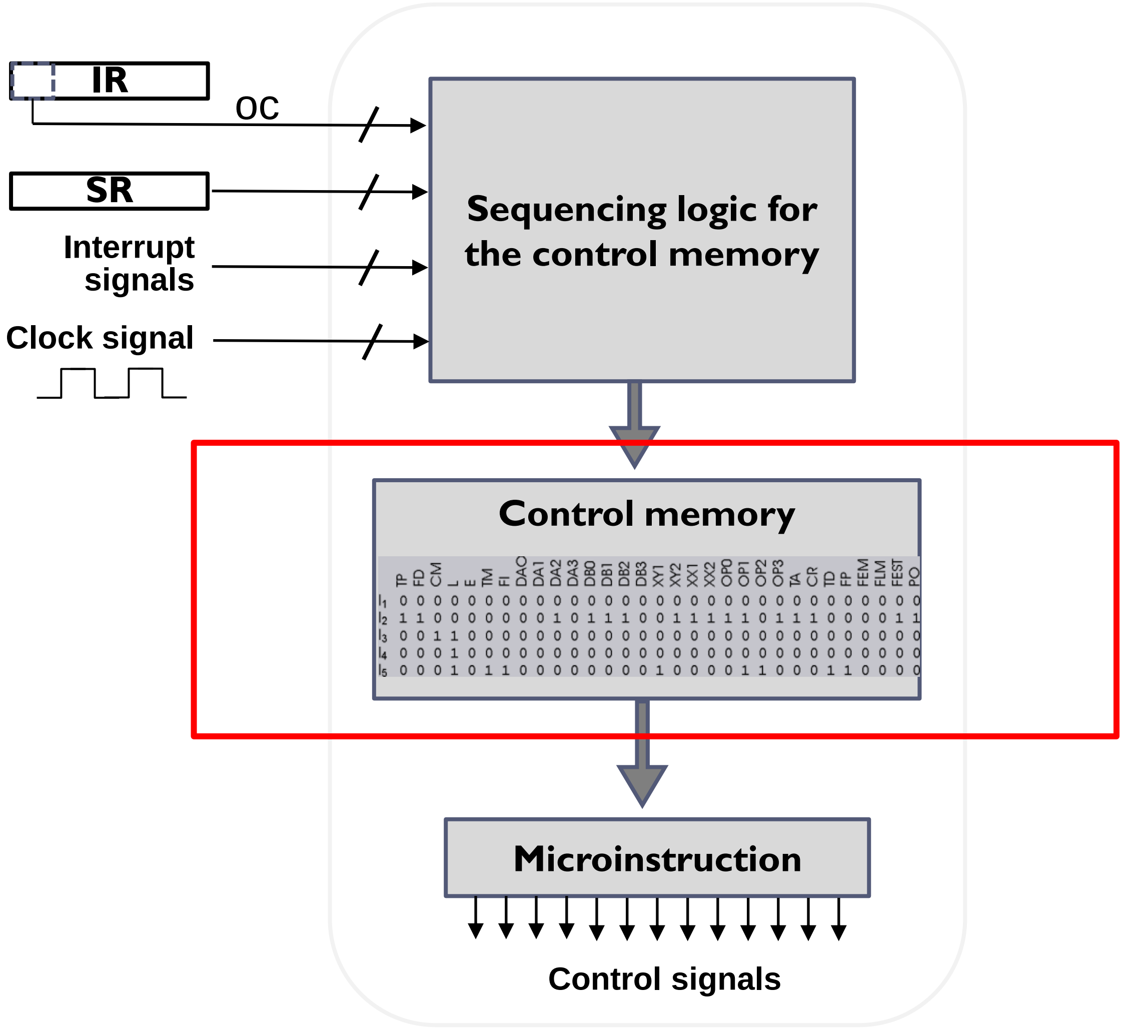
Microinstruction format

- **Microinstruction format**:
  specifies the number of bits and the meaning of each bit.



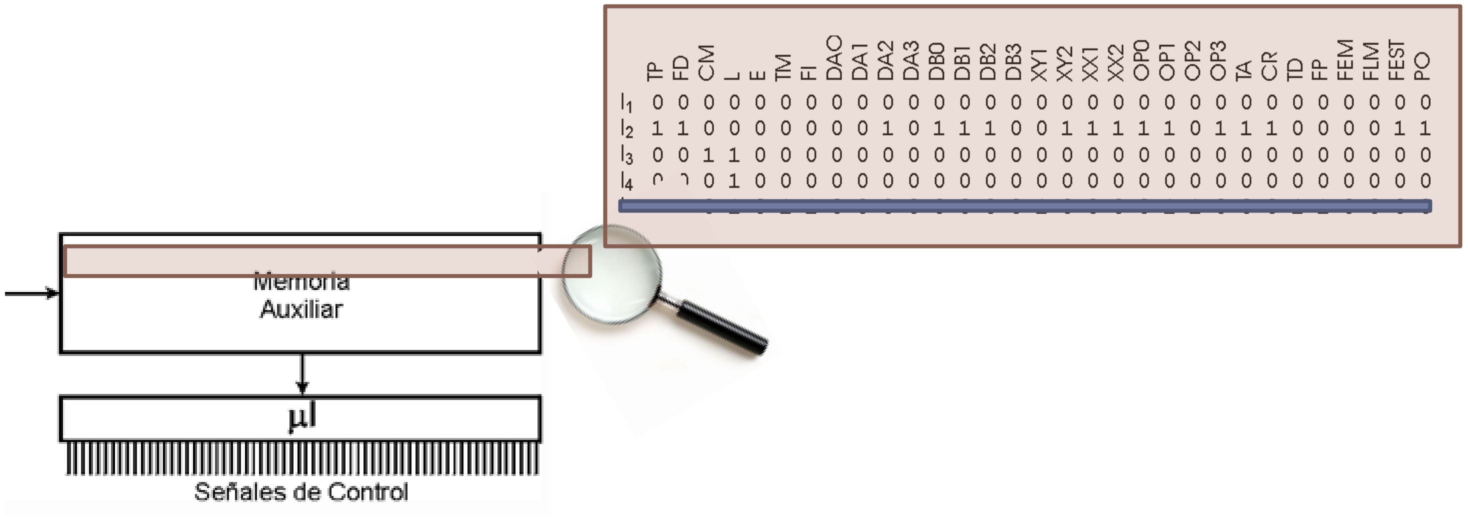
- Signals grouped into **fields**:
  - Tristate bus signals
  - ALU signals
  - Registers file signals
  - Main memory signals
  - Multiplexor signals

General structure of a microprogrammed control unit



	TP	FD	CM	L	E	TM	FI	DAO	DA1	DA2	DA3	DB0	DB1	DB2	DB3	XY1	XY2	XX1	XX2	OP0	OP1	OP2	OP3	TA	CR	TD	FP	FEM	FLM	FEST	PO
I_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I_2	1	1	0	0	0	0	0	0	0	1	0	1	1	1	0	0	1	1	1	1	1	0	1	1	1	0	0	0	0	1	1
I_3	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I_4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I_5	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0

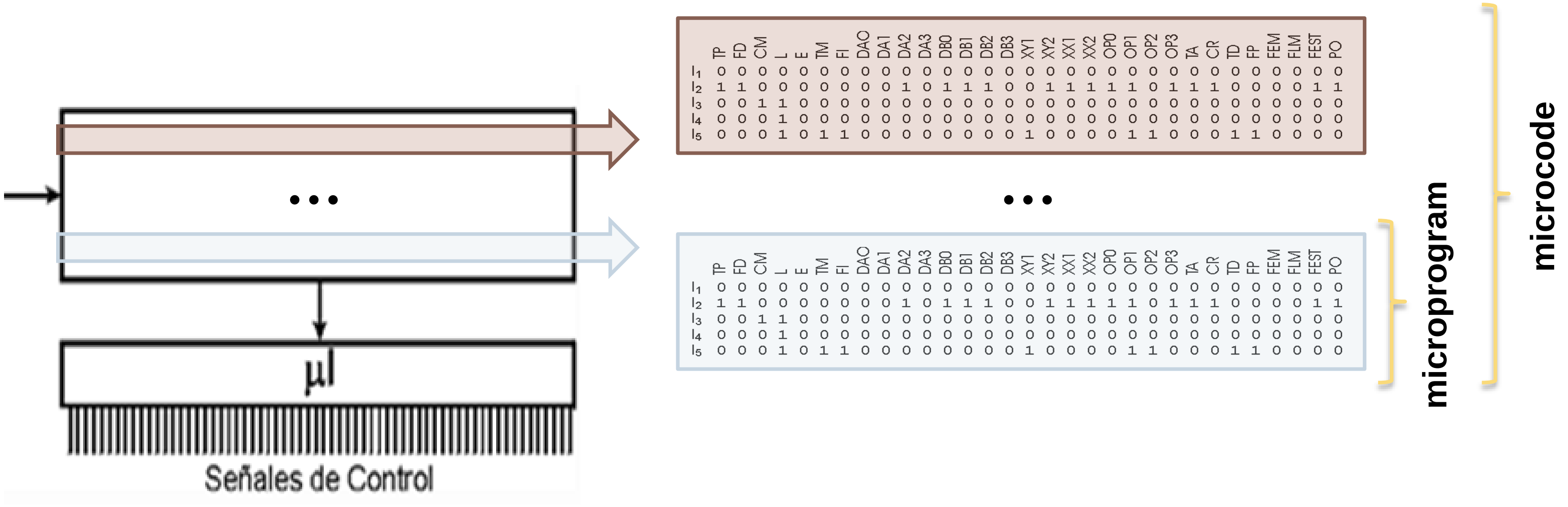
Microprogrammed control unit.
microinstructions



	TP	FD	CM	L	E	TM	FI	DA0	DA1	DA2	DA3	DB0	DB1	DB2	DB3	XY1	XY2	XX1	XX2	OP0	OP1	OP2	OP3	TA	CR	TD	FP	FEM	FLM	FEST	PO
I_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I_2	1	1	0	0	0	0	0	0	0	1	0	1	1	1	0	0	1	1	1	1	1	0	1	1	1	0	0	0	0	1	1
I_3	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I_4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- **Microinstruction:** To each word defining the value of each control signal in a cycle of an instruction/fetch+IAC
- The **microinstructions…**
  - Are a list of 1's and 0's representing the state of each control signal during a period of one instruction.
  - Have one bit for each control signal.

Microprogrammed control unit.
microprogram and microcode



	TP	FD	CM	L	E	TM	FI	DAO	DA1	DA2	DA3	DB0	DB1	DB2	DB3	XY1	XY2	XX1	XX2	OP0	OP1	OP2	OP3	TA	CR	TD	FP	FEM	FLM	FEST	PO
I_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I_2	1	1	0	0	0	0	0	0	0	1	0	1	1	1	0	0	1	1	1	1	1	0	1	1	1	0	0	0	0	1	1
I_3	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I_4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I_5	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0

…

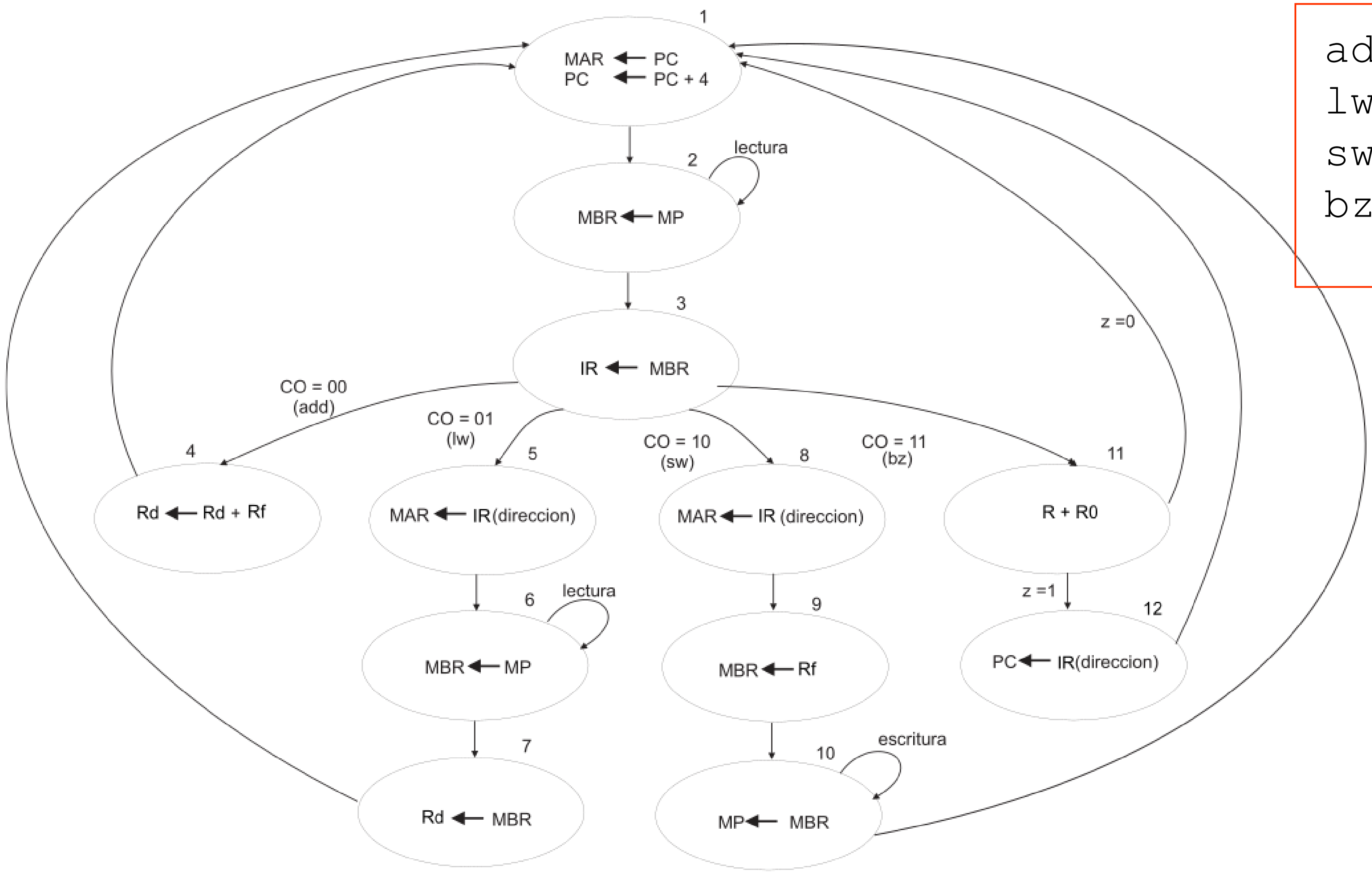
	TP	FD	CM	L	E	TM	FI	DAO	DA1	DA2	DA3	DB0	DB1	DB2	DB3	XY1	XY2	XX1	XX2	OP0	OP1	OP2	OP3	TA	CR	TD	FP	FEM	FLM	FEST	PO
I_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I_2	1	1	0	0	0	0	0	0	0	1	0	1	1	1	0	0	1	1	1	1	1	0	1	1	1	0	0	0	0	1	1
I_3	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I_4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I_5	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0

- **microprogram**: ordered list of microinstructions, which represent the chronogram of a machine instruction.
- **microcode**: set of microprograms of a machine.

Example: State machine

Example for a computer with only 4 machine instructions

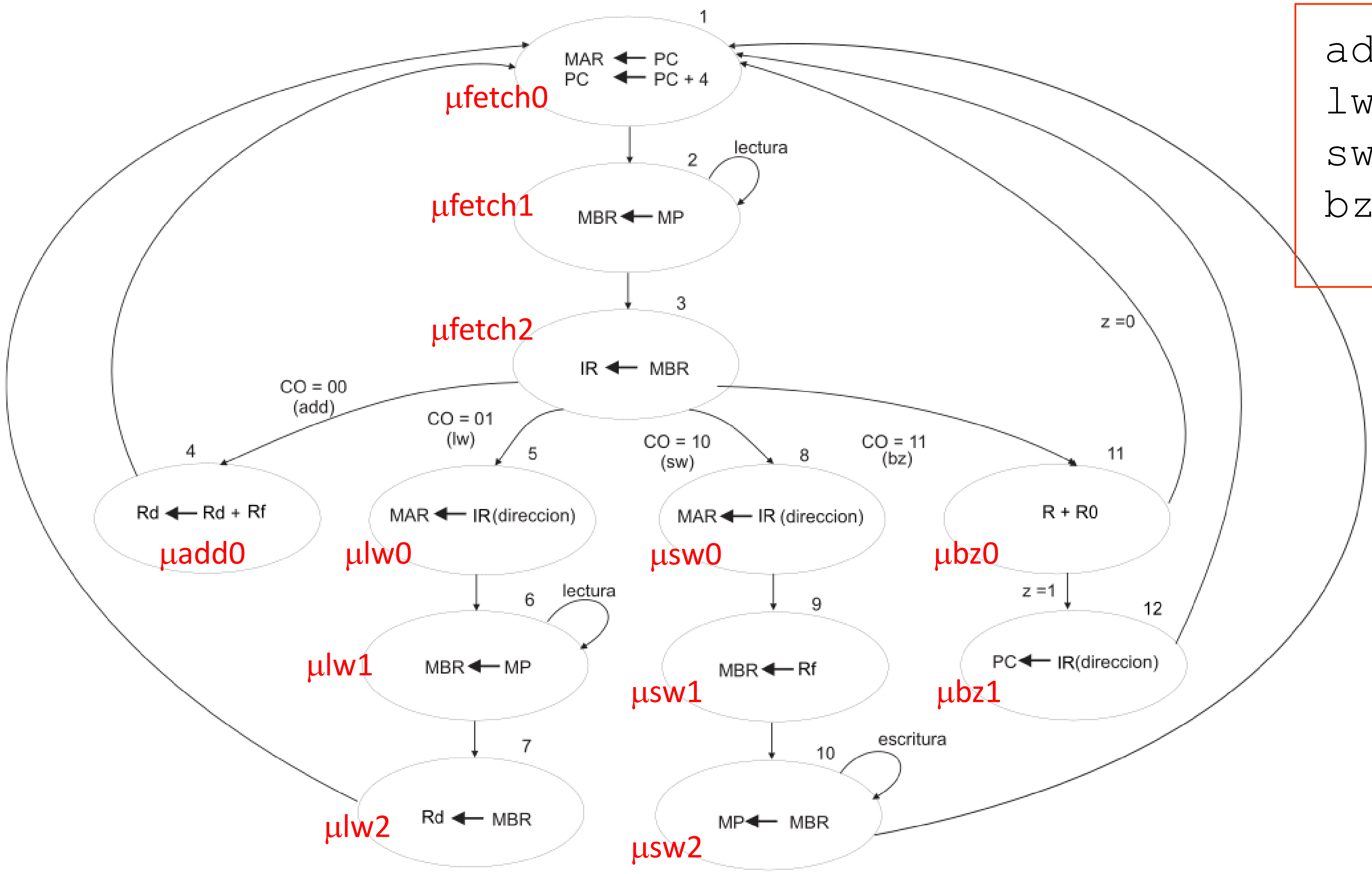
```
add rd, rf
lw rd, dir
sw Rf, dir
bz R, dir
```



Example: associated microinstructions

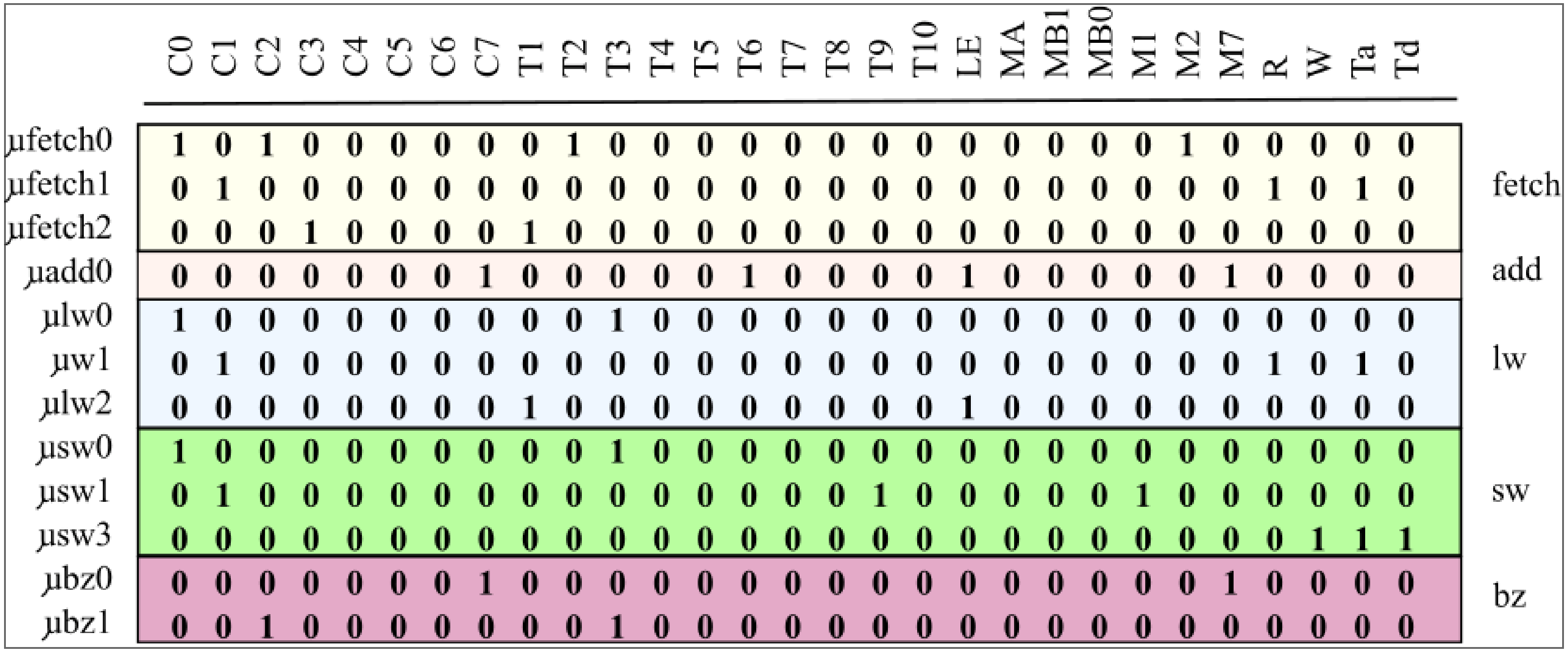
Example for a computer with only 4 machine instructions

```
add rd, rf
lw rd, dir
sw Rf, dir
bz R, dir
```



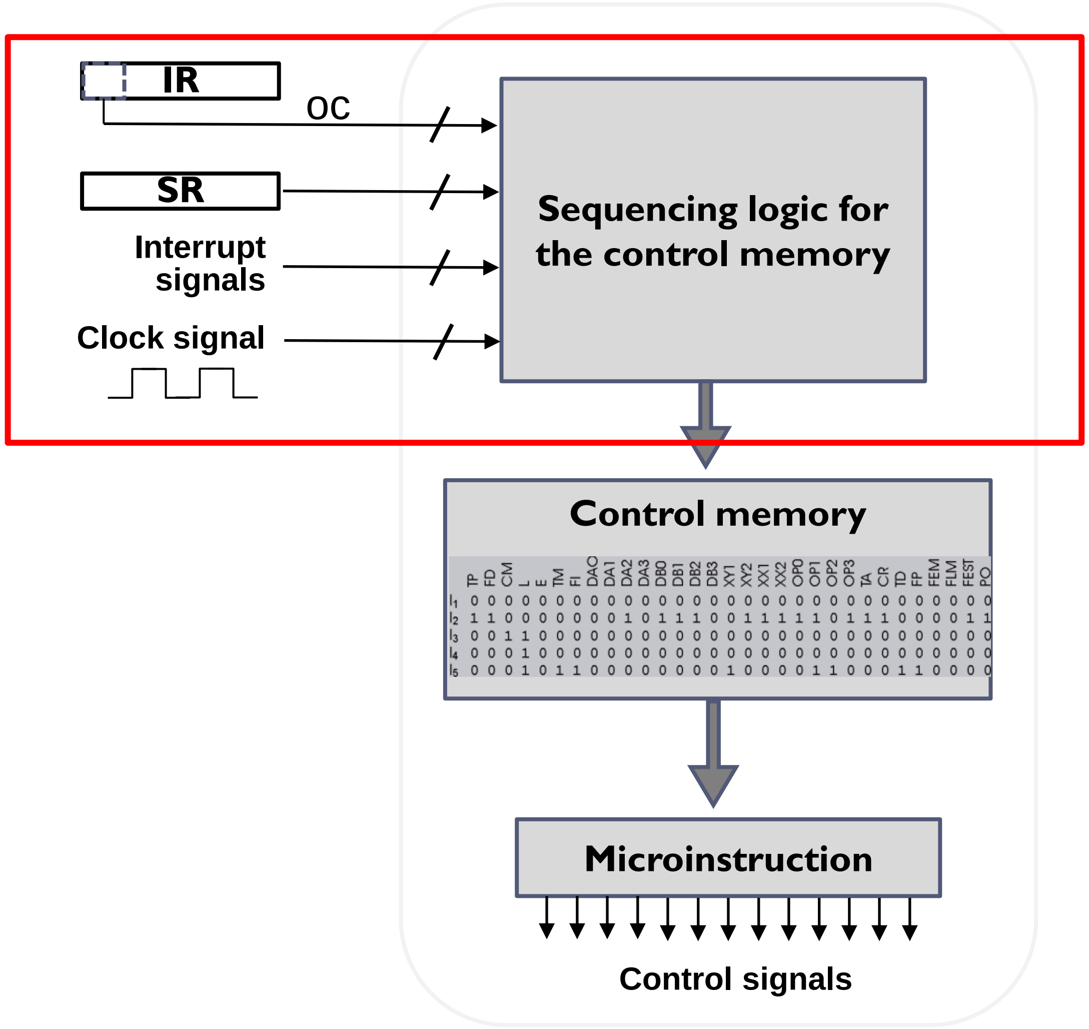
Example: microcode

```
add r1, r2
lw r1, dir
bz dir
sw r1
```



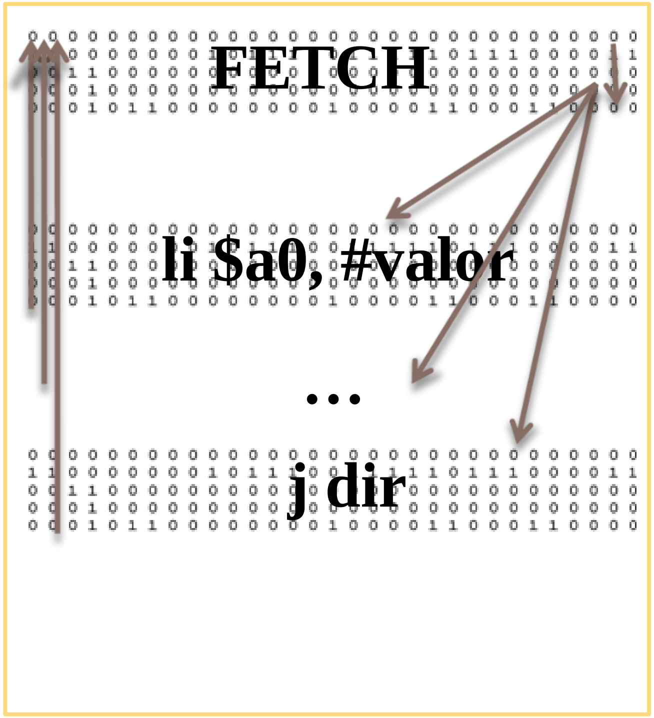
	C0	C1	C2	C3	C4	C5	C6	C7	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	LE	MA	MB1	MB0	M1	M2	M7	R	W	Ta	Td	
µfetch0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	
µfetch1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	fetch
µfetch2	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
µadd0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	add
µlw0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
µw1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	lw
µlw2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	
µsw0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
µsw1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	sw
µsw3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	
µbz0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	bz
µbz1	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

General structure of a microprogrammed control unit



	TP	FD	CM	L	E	TM	FI	DAO	DA1	DA2	DA3	DB0	DB1	DB2	DB3	XY1	XY2	XX1	XX2	OP0	OP1	OP2	OP3	TA	CR	TD	FP	FEM	FLM	FEST	PO
I_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I_2	1	1	0	0	0	0	0	0	0	1	0	1	1	1	0	0	1	1	1	1	1	0	1	1	1	0	0	0	0	1	1
I_3	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I_4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I_5	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1	0	0	0	0

Contents of the control memory

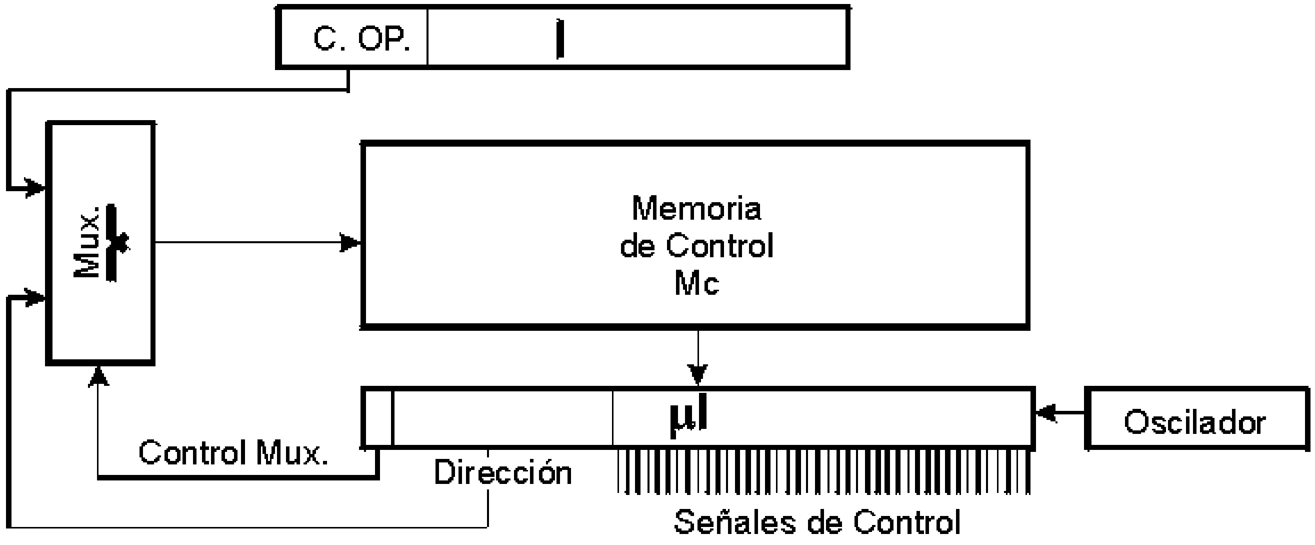


- FETCH: get next instruction
  - IAC: interrupt acknowledge cycle.
  - IR<- Mem[PC], PC++, jump-to-O.C.
- Microprograms:
  one for every machine instruction
  - fetch rest of operands (if any)
    - Updates PC on multi-word instructions
  - Execute the instruction
  - Jump to FETCH

Microprogrammed control unit structure

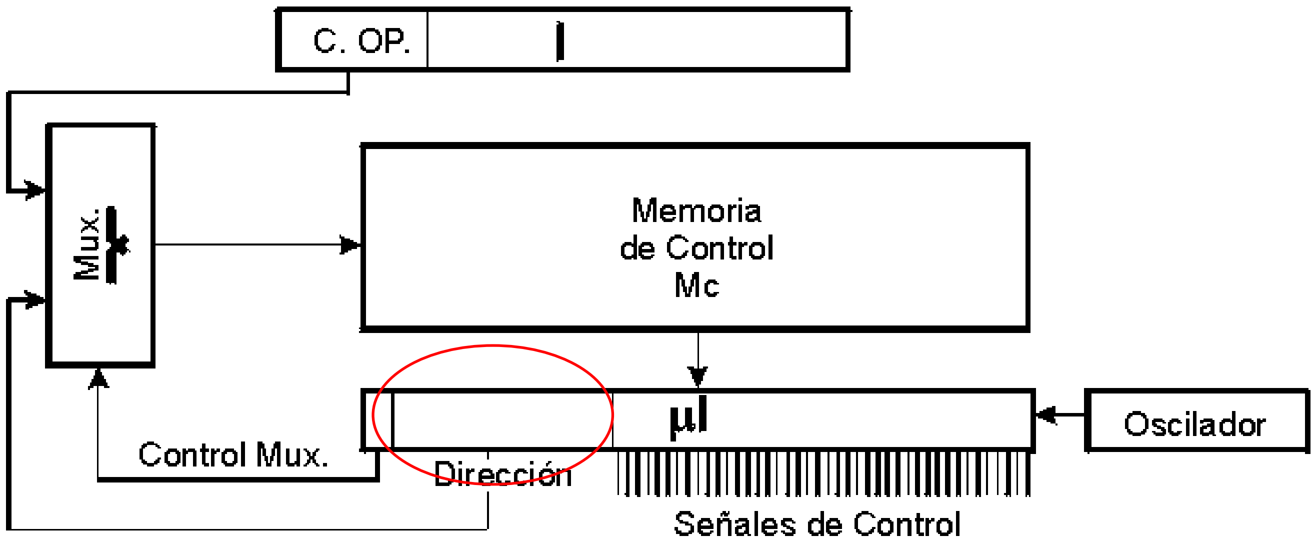
- Three basic conditions:
  1. Sufficient control memory to store all microprograms corresponding to all instructions.
  2. Procedure for associating each instruction with its microprogram
     - Procedure that converts the instruction operation code to the control memory address where your microprogram starts..
  3. Sequencing mechanism to read successive microinstructions, and to branch to another microprogram when the current one is finished.
- Two alternatives:
  1. Explicit sequencing.
  2. Implicit sequencing.

Microprogrammed C.U. structure with **explicit** sequencing



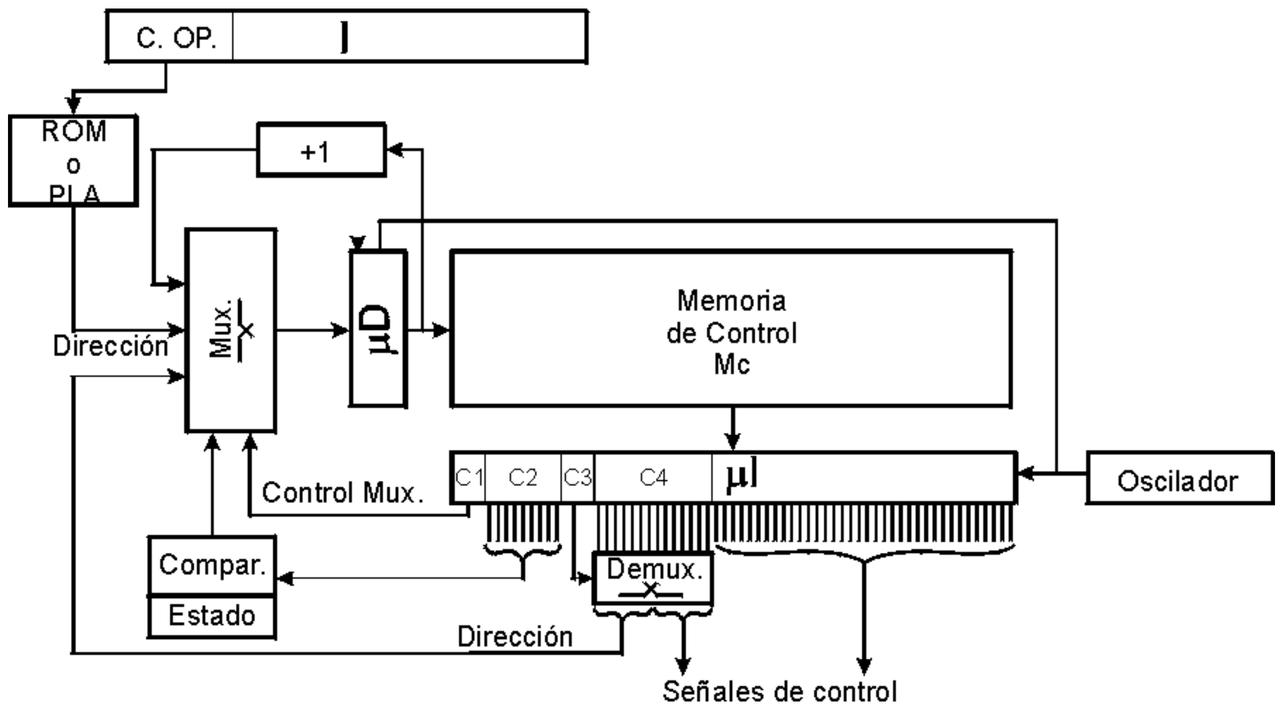
- Control memory stores all µprograms, where each µinstruction provides the next µinstruction µaddress
- The OC represents the µAddress of the first µinstruction associated with the machine instruction.

Microprogrammed C.U. structure with **explicit** sequencing



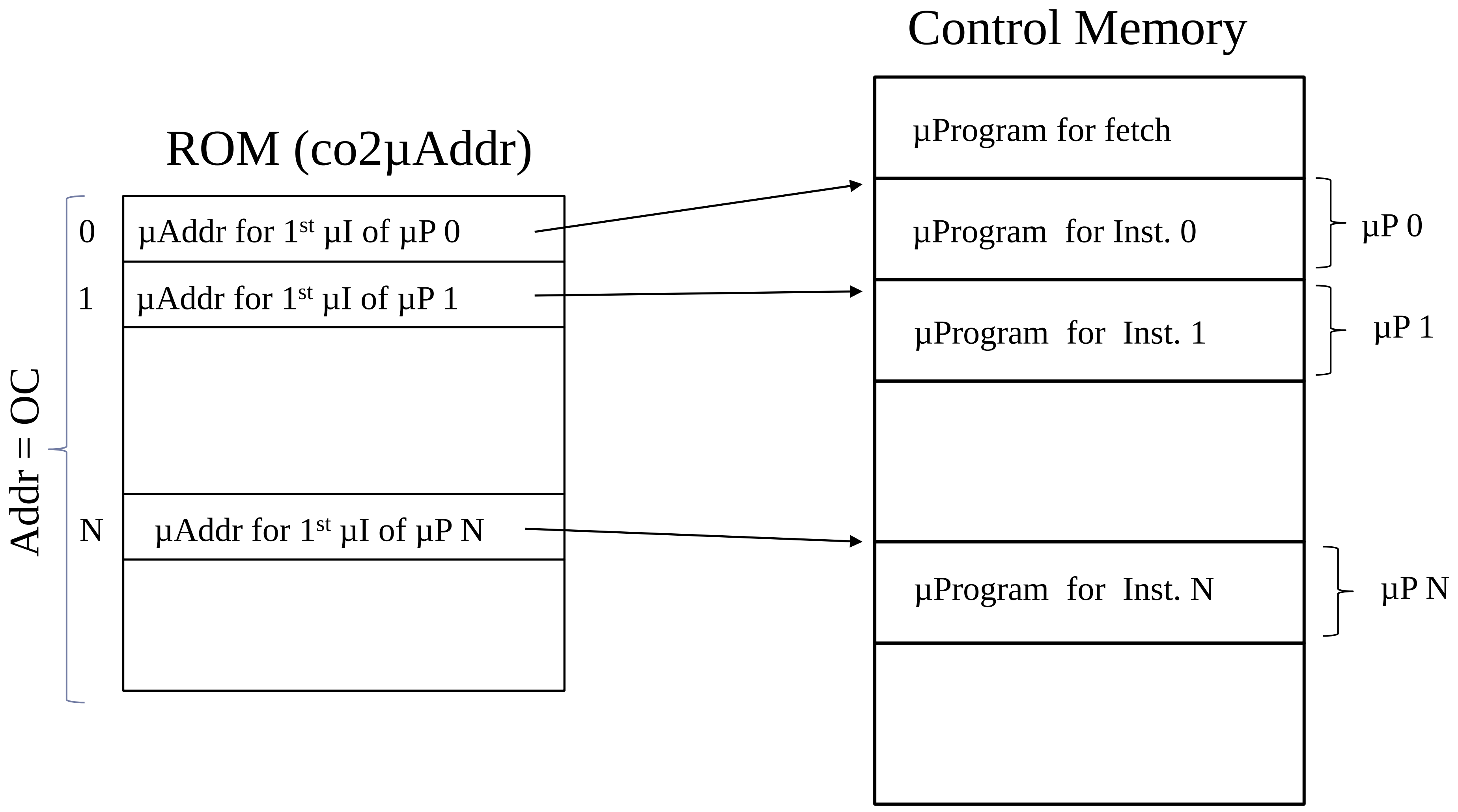
- Control memory stores all µprograms, where each µinstruction provides the next µinstruction µaddress
- Problem: large amount of control memory for instruction sequencing, required stores the next µaddress

Microprogrammed C.U. structure with **implicit** sequencing

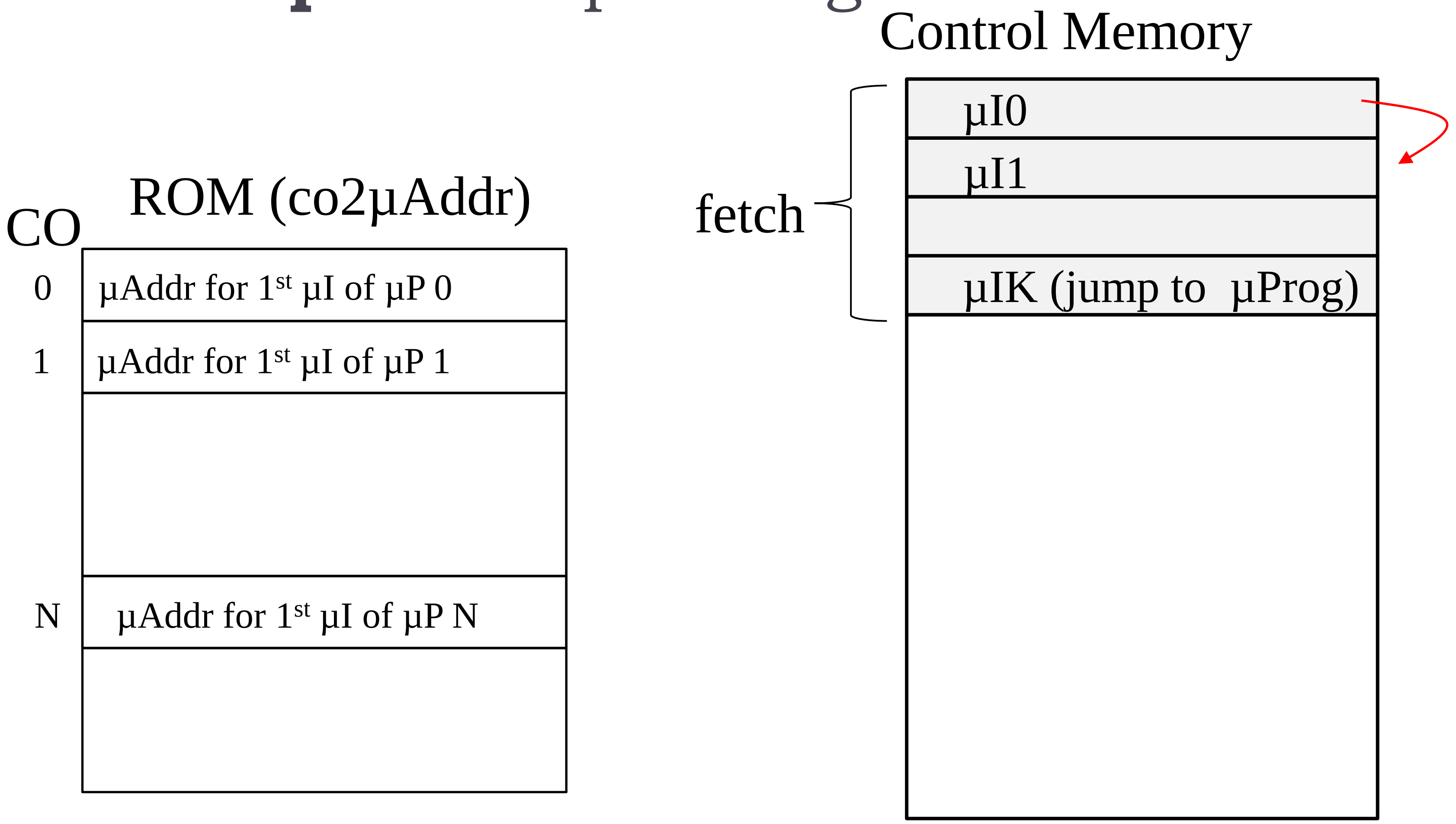


- Control memory stores all microprograms consecutively in the control memory.
- The ROM/PLA associates each instruction with its microprogram (first μaddress, μconditional μinstruction (+1), μconditional μbifurcations or μloops).
- Next μinstruction (+1), conditional μbifurcations or μloops

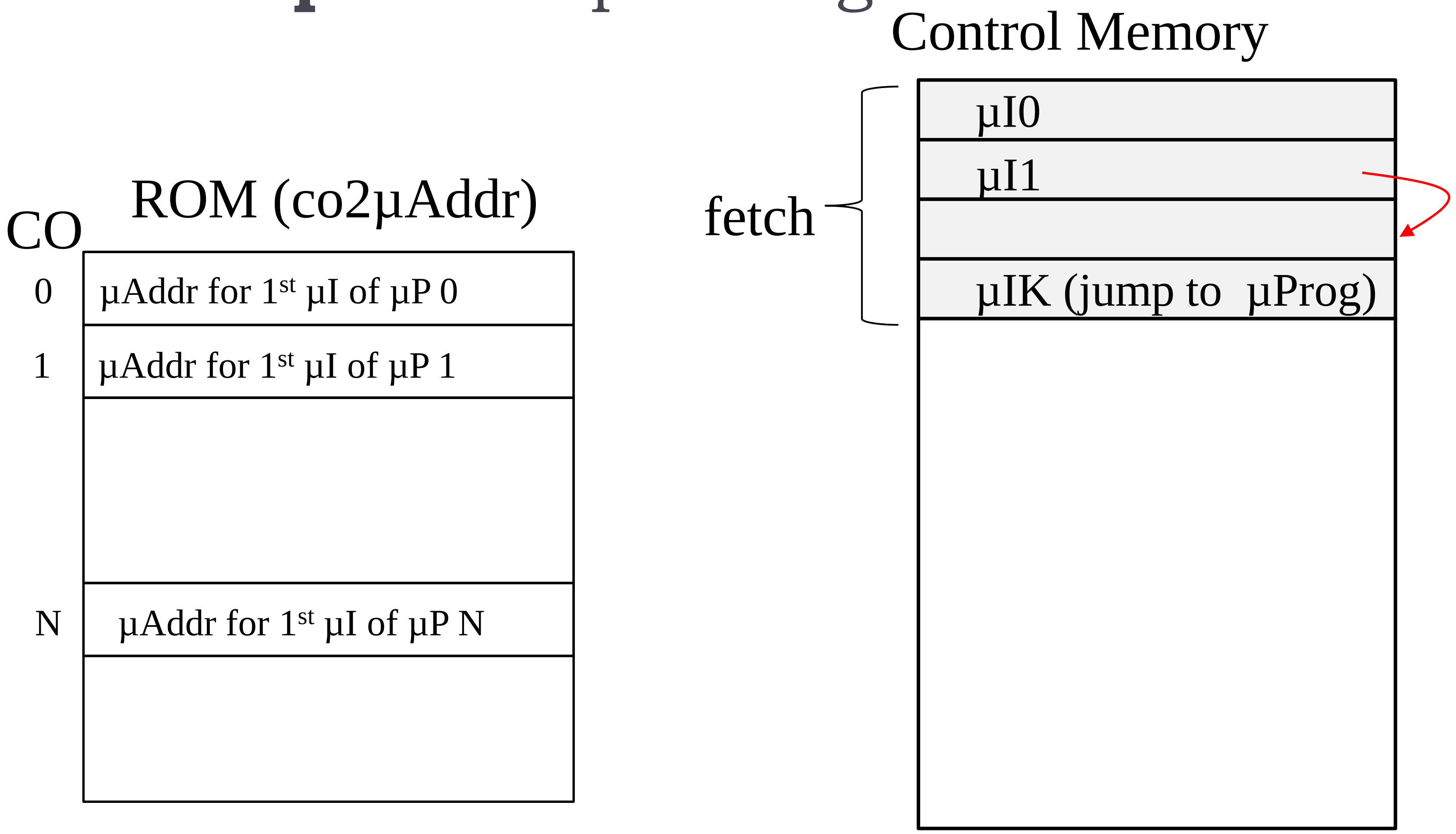
Example of Control Unit with **implicit** sequencing



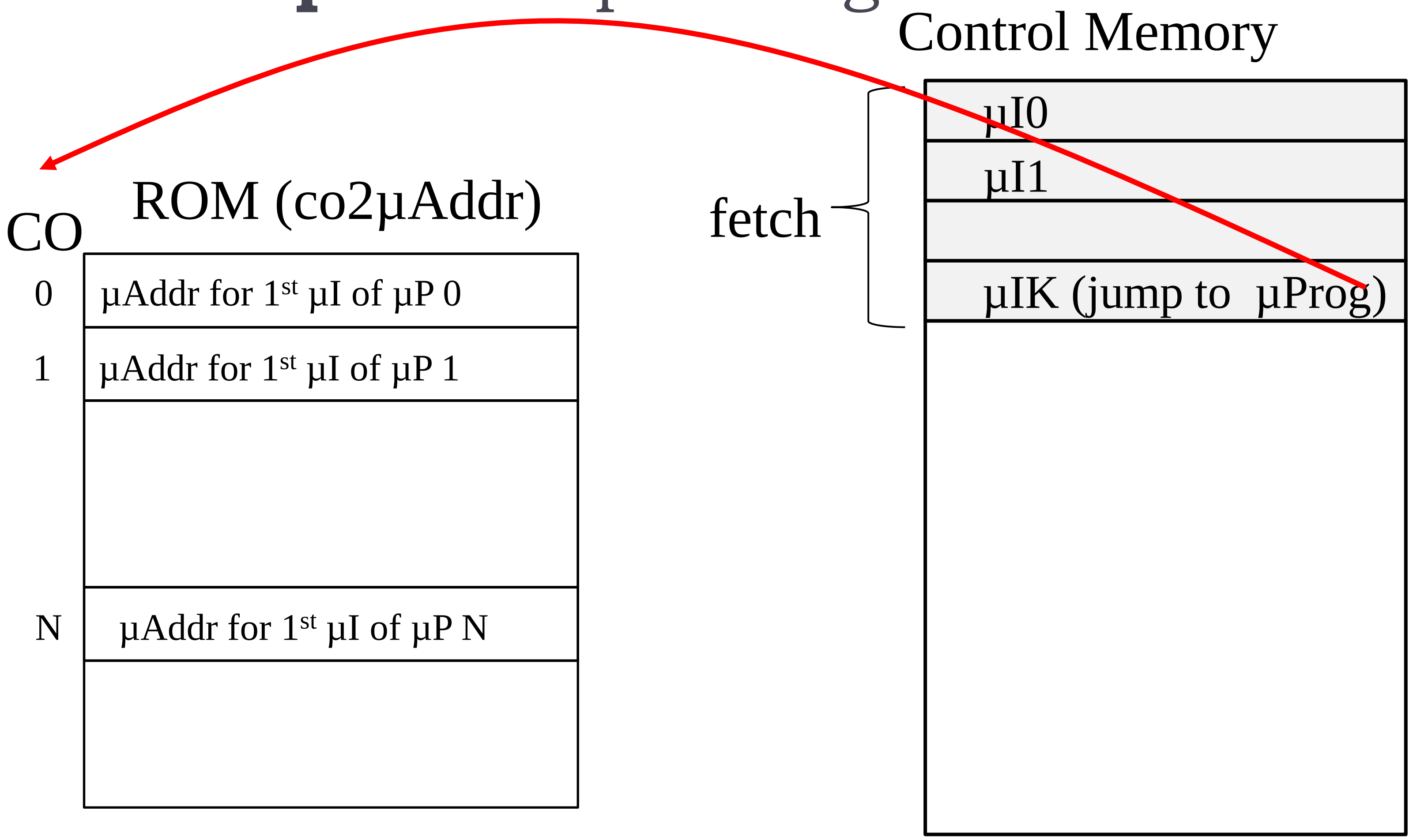
Example of Control Unit with **implicit** sequencing



Example of Control Unit with **implicit** sequencing

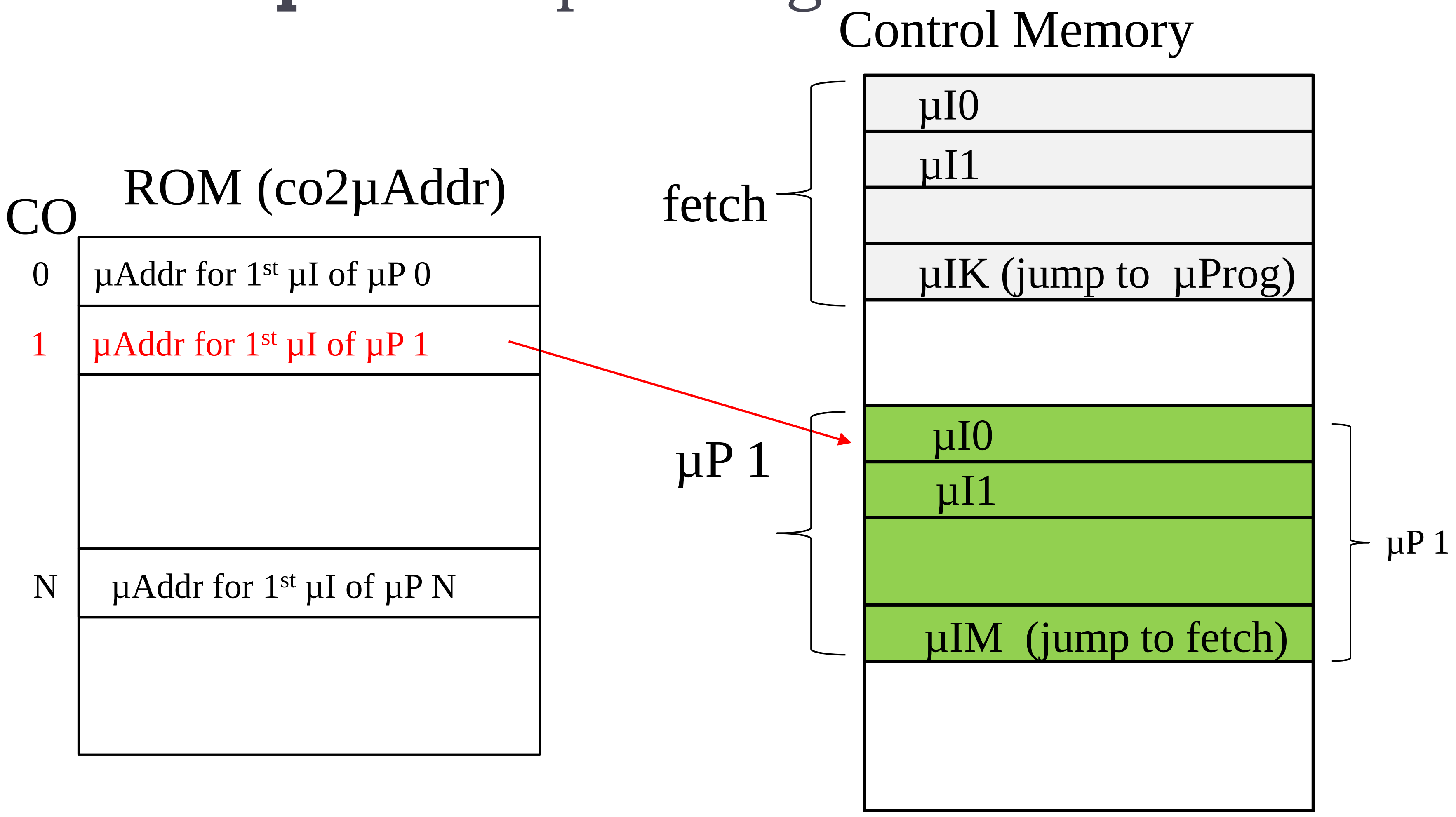


Example of Control Unit with **implicit** sequencing

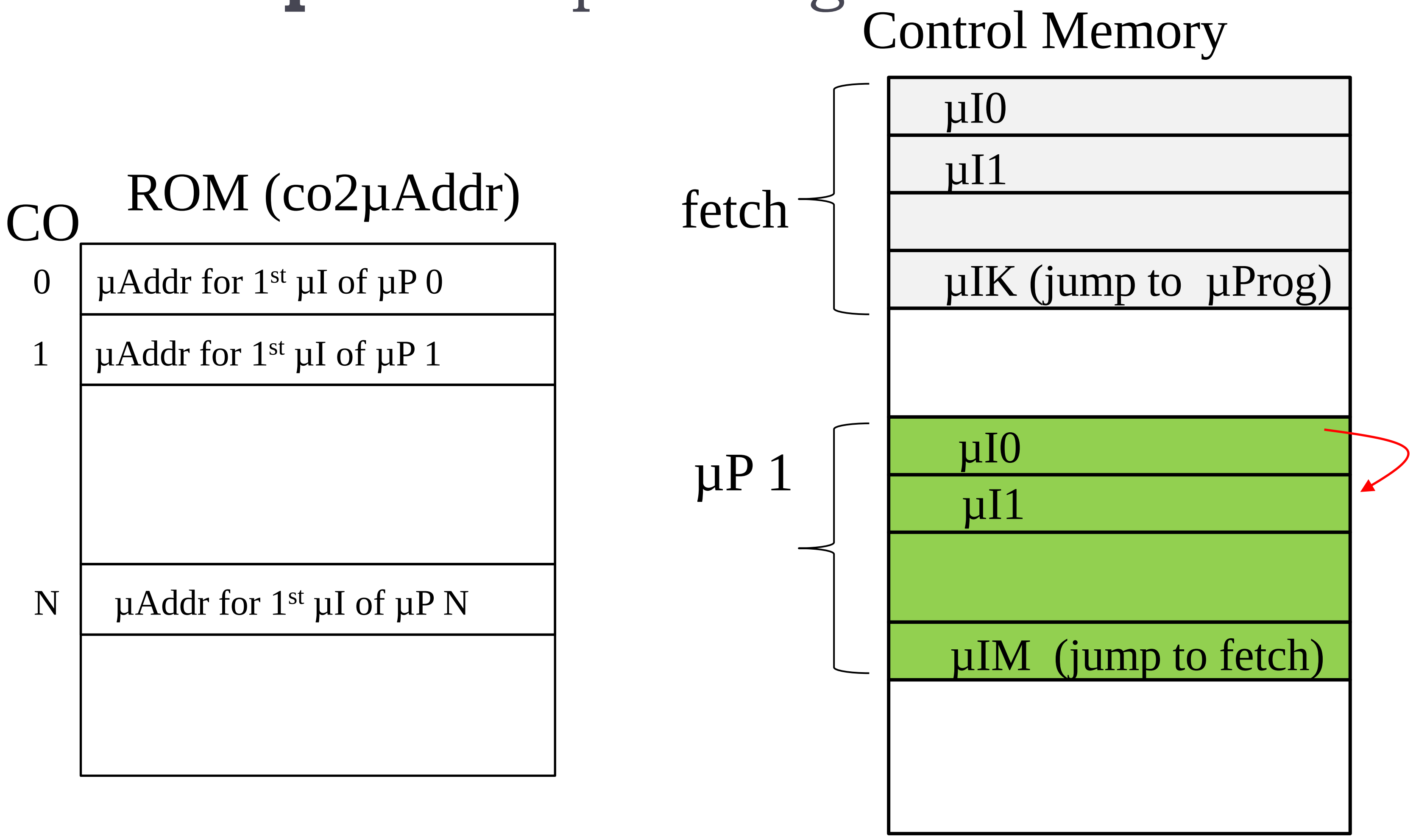


The Operation Code (OC) is at the Instruction Register (IR)

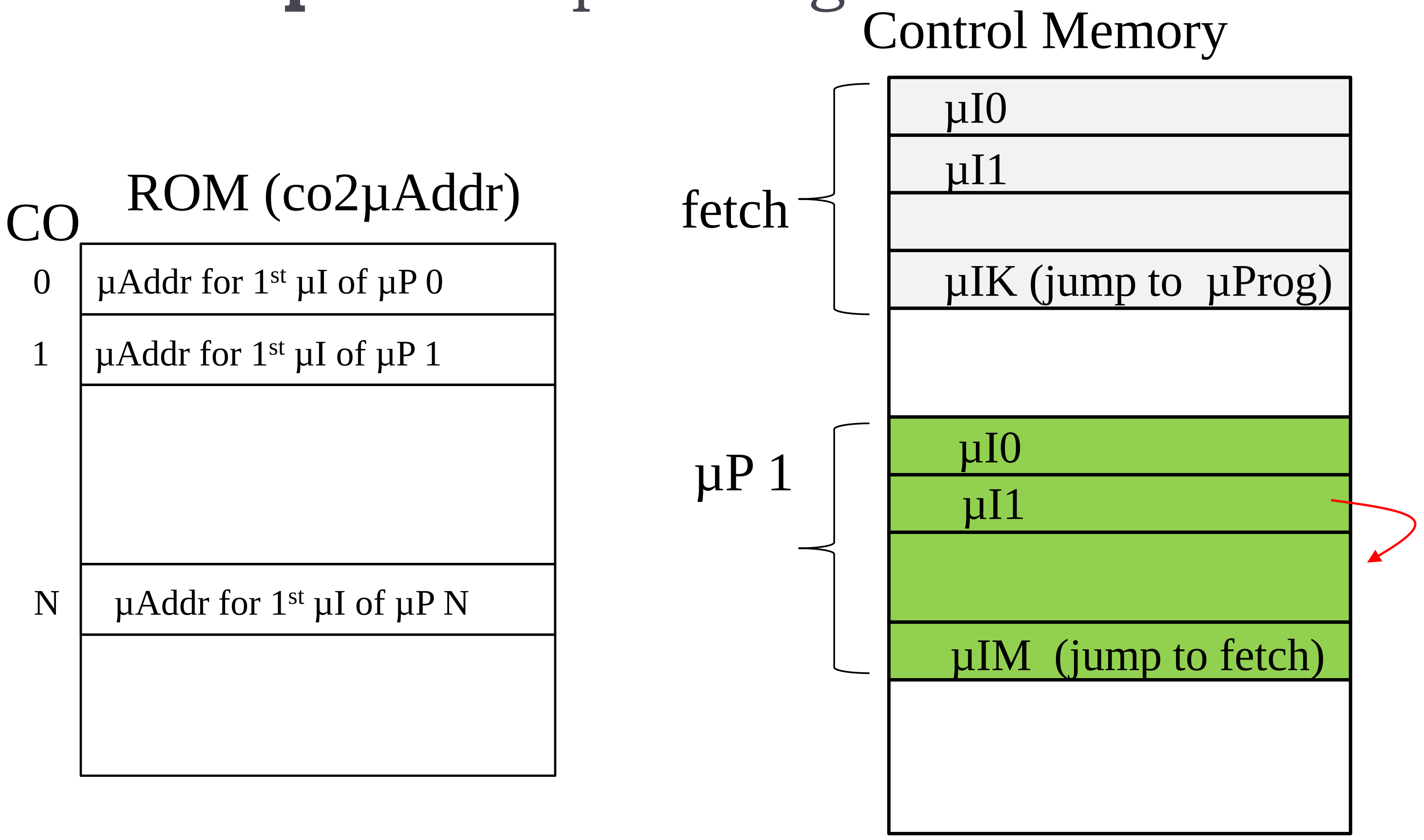
Example of Control Unit with **implicit** sequencing



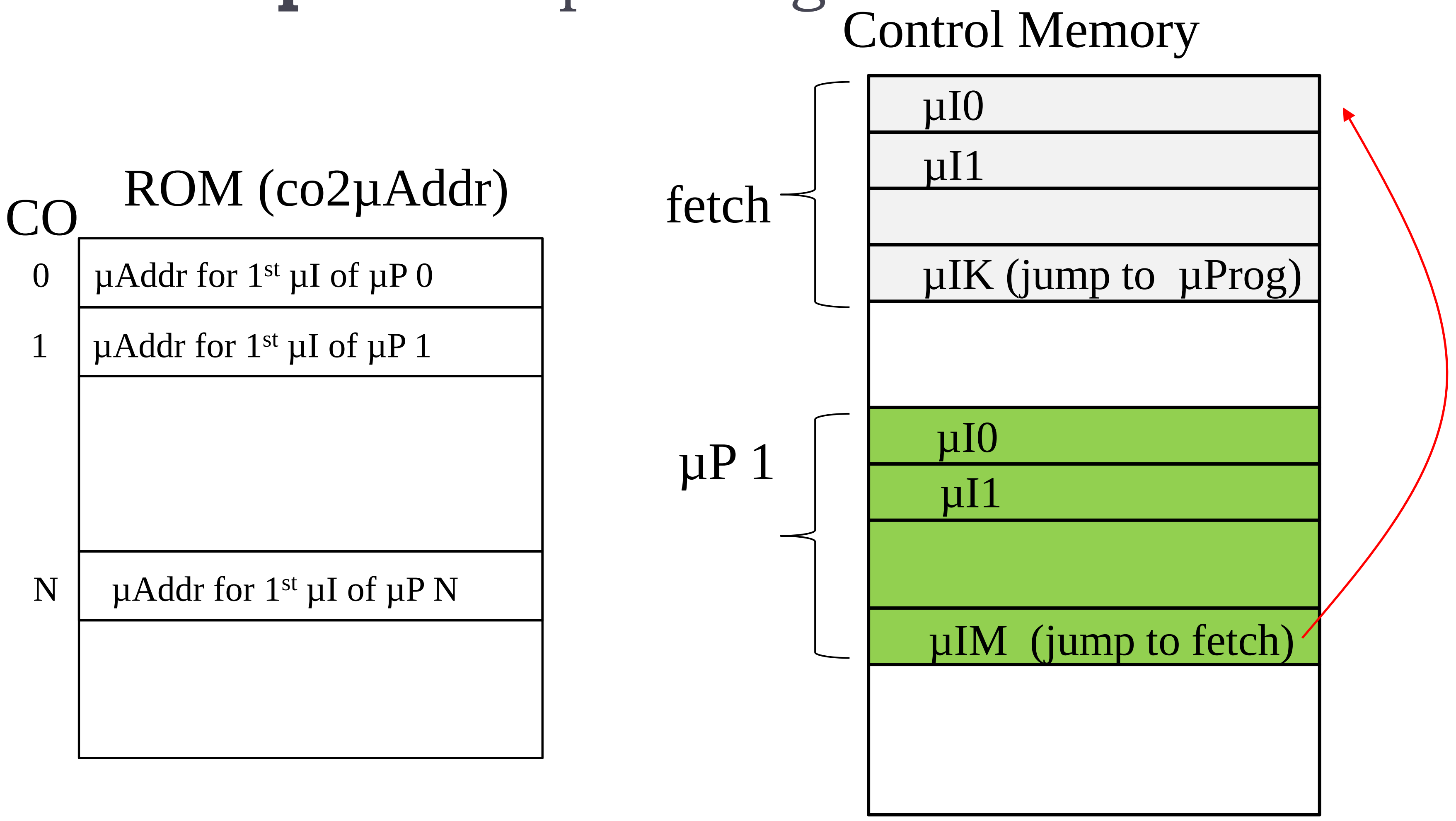
Example of Control Unit with **implicit** sequencing



Example of Control Unit with **implicit** sequencing

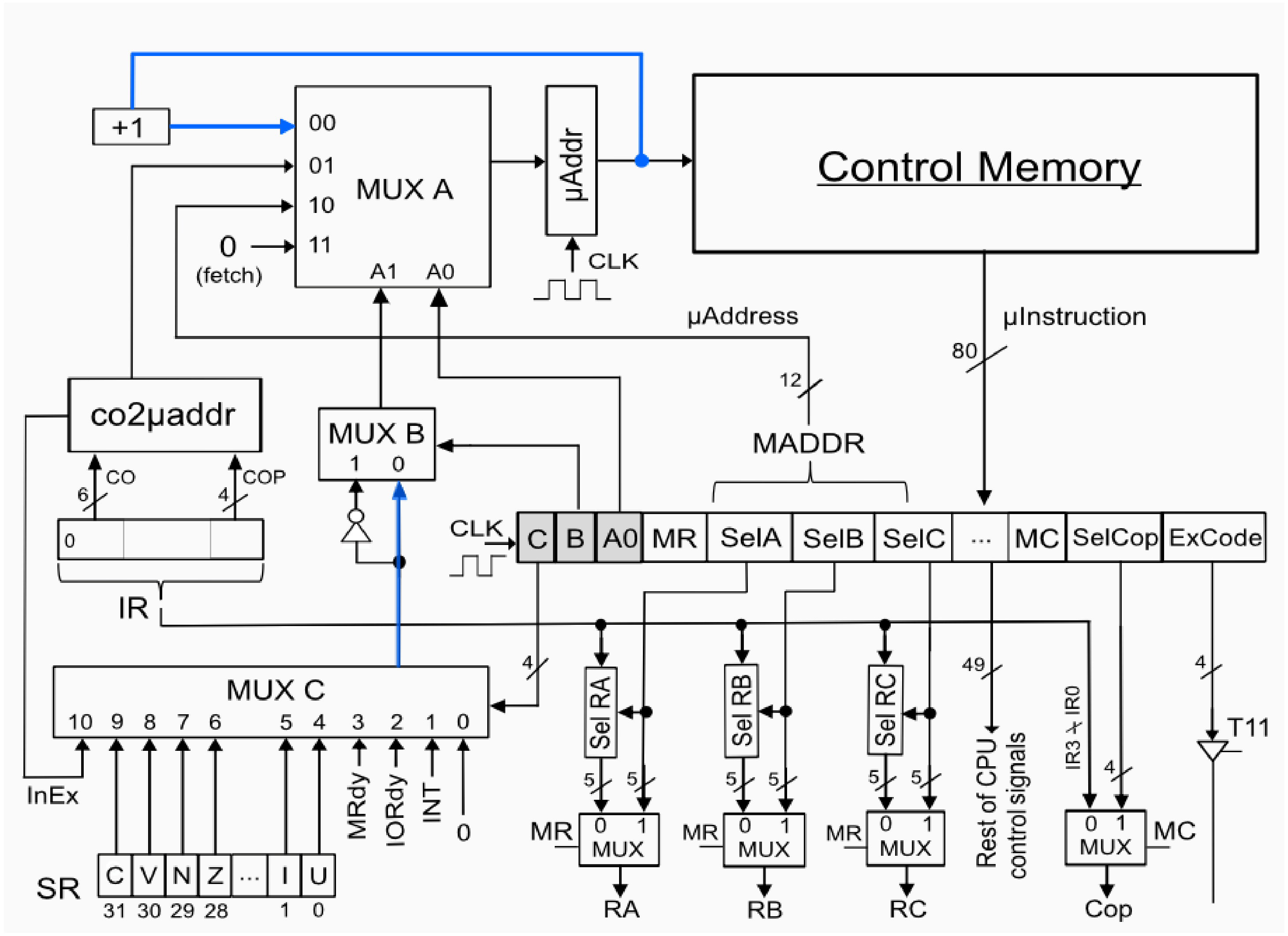


Example of Control Unit with **implicit** sequencing

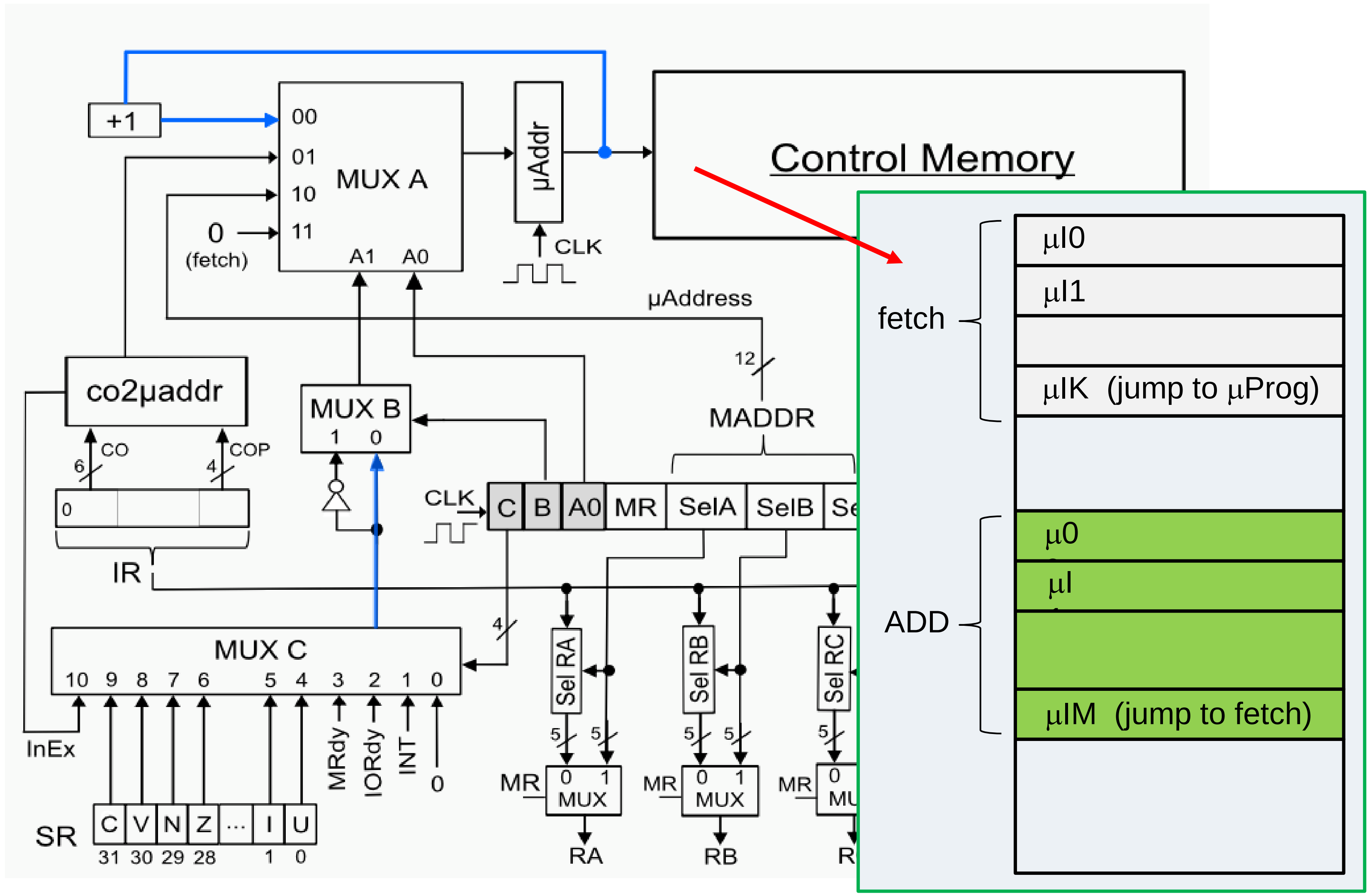


Contents

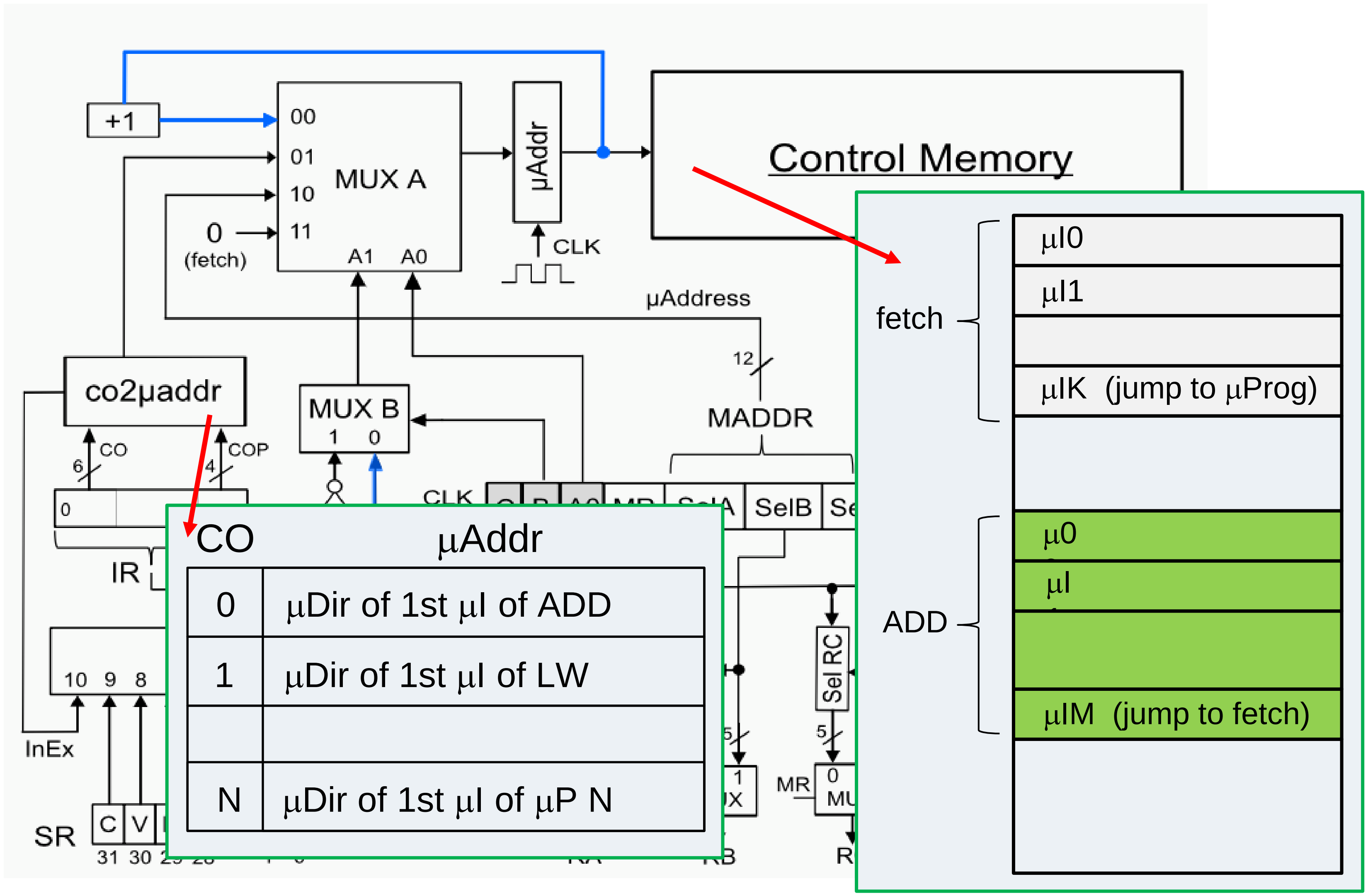
Control Unit in WepSIM



Control Unit in WepSIM



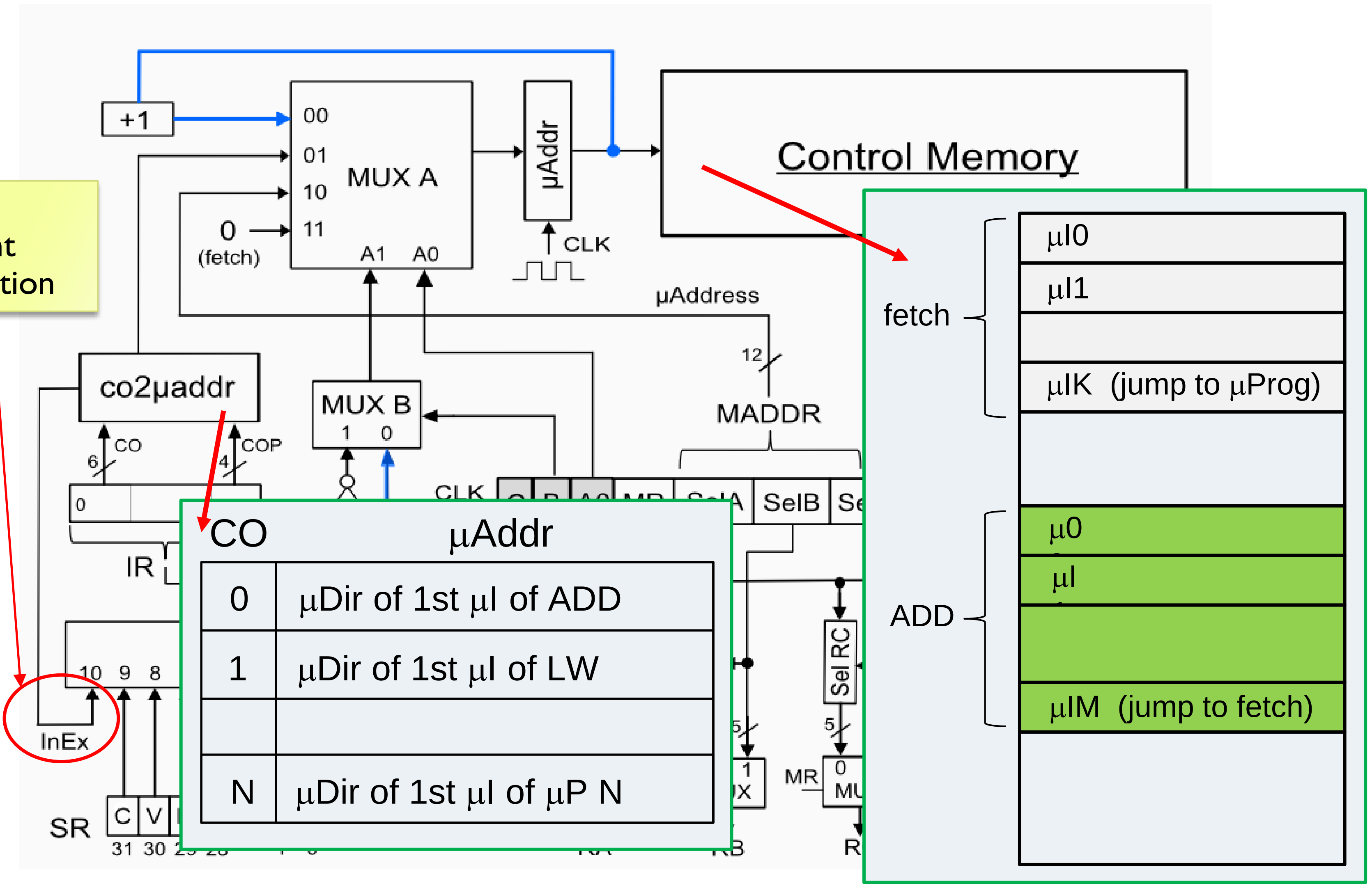
Control Unit in WepSIM



CO	μAddr
0	μDir of 1st μI of ADD
1	μDir of 1st μI of LW
N	μDir of 1st μI of μP N

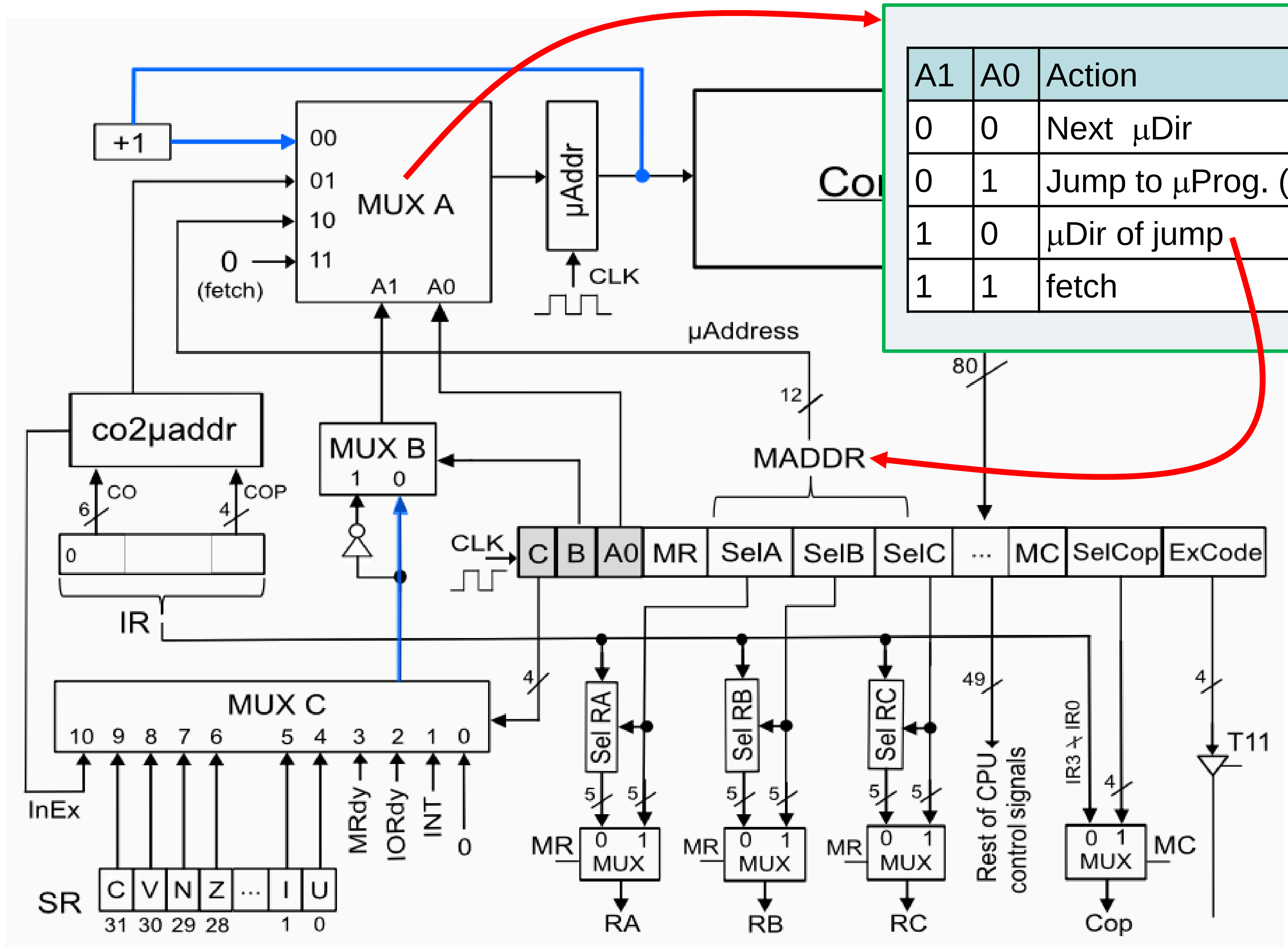
Control Unit in WepSIM

Non-existent instruction



CO	μAddr
0	μDir of 1st μI of ADD
1	μDir of 1st μI of LW
N	μDir of 1st μI of μP N

Control Unit in WepSIM



A1	A0	Action
0	0	Next μDir
0	1	Jump to μProg. (ROM)
1	0	μDir of jump
1	1	fetch

Examples of more frequent jumps

Elemental operations with CU

- **Jump to address 000100011100 (12 bits) if Z = 1. Otherwise jump to the next one.**

Elemental operation	Signals
If (Z) μPC=000100011100	A0=0, B=0, C=0110_2, mADDR=000100011100_2

- **Salto incondicional a la dirección 000100011111**

Elemental operation	Signals
μPC=000100011111	A0=0, B=1, C=0000_2, mADDR=000100011111_2

- **Jump to first μaddress of the μprogram related to OC**

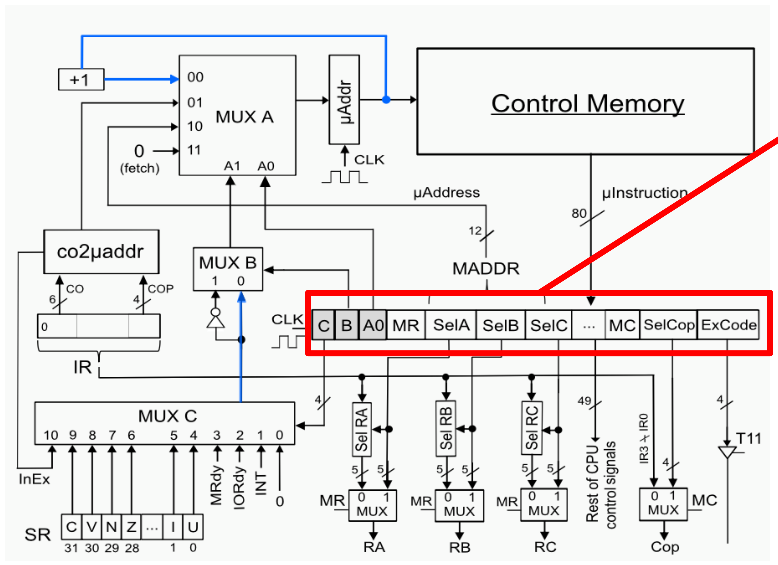
Elemental operation	Signals
Jump to OC	A0=1, B=0, C=0000_2

Control Unit in WepSIM

A0	B	C3	C2	C1	C0	Acción
0	0	0	0	0	0	Siguiente µDirección
0	1	0	0	0	0	Salto incondicional a MADDR
0	0	0	0	0	1	Salto condicional a MADDR si INT = 1 (*)
0	1	0	0	1	0	Salto condicional a MADDR si IORdy = 0 (*)
0	1	0	0	1	1	Salto condicional a MADDR si MRdy = 0 (*)
0	0	0	1	0	0	Salto condicional a MADDR si U = 1 (*)
0	0	0	1	0	1	Salto condicional a MADDR si I = 1 (*)
0	0	0	1	1	0	Salto condicional a MADDR si Z = 1 (*)
0	0	0	1	1	1	Salto condicional a MADDR si N = 1 (*)
0	0	1	0	0	0	Salto condicional a MADDR si O = 1 (*)
1	0	0	0	0	0	Salto a µProg. (ROM c02µaddr)
1	1	0	0	0	0	Salto a fetch (µDir = 0)

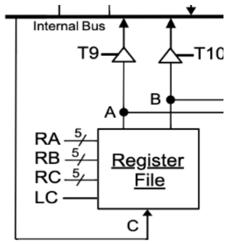
- (*) If the condition is not satisfied → Next µAddress
- Remaining entries → indefinite behaviour

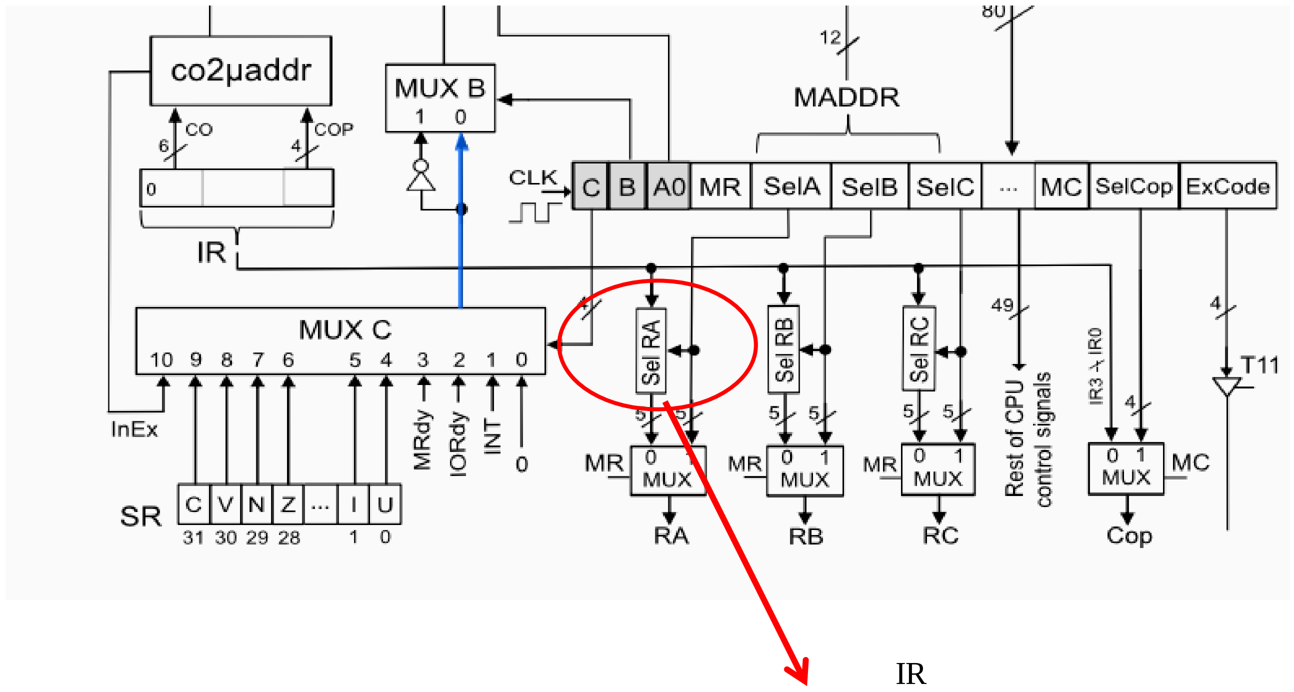
Microinstruction format



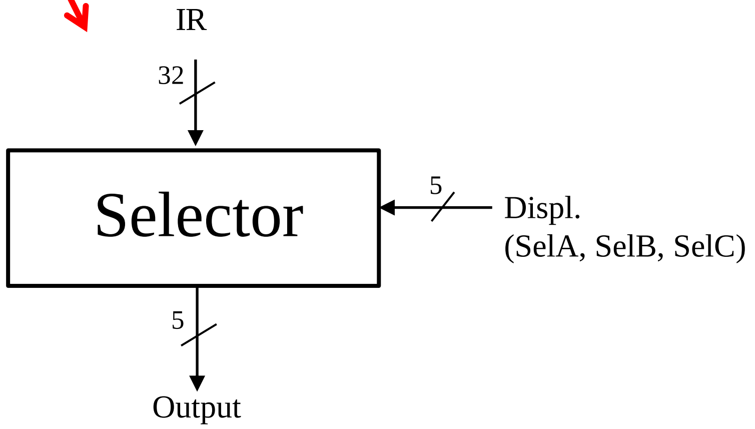
C0 .. C7	Load register
Ta, Td	Tristate buffers to bus
T1..T10	Tristate buffers
M1, M2, M7, MA, MB	Multiplexors
SelP	State register selector
LC	Load in Register File
SE	Sign extensión
Size, Offset	Selector of IR register
BW	Size of memory Access
R, W	Main memory operation
IOR, IOW	I/O operation
INTA	INT selector
I	Enables interuptions
U	User/kernel modes

Register file selector

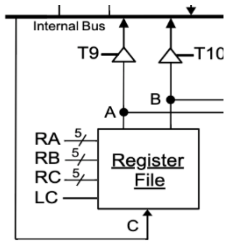


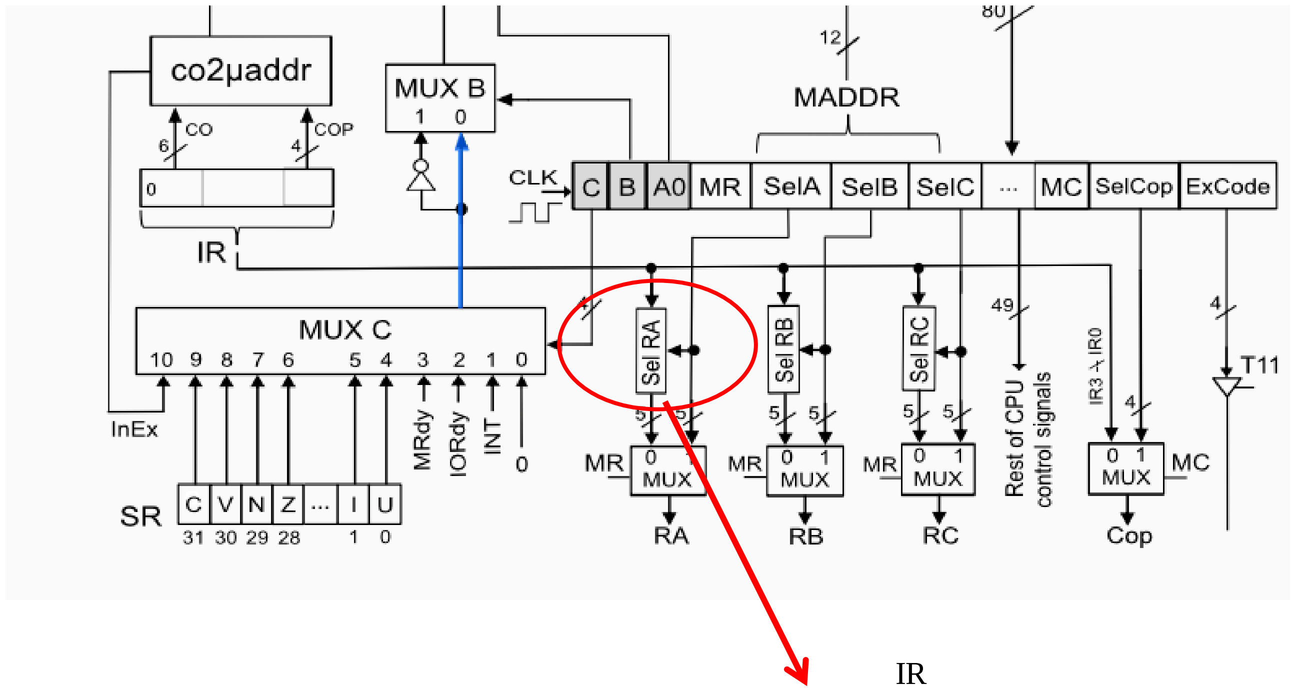


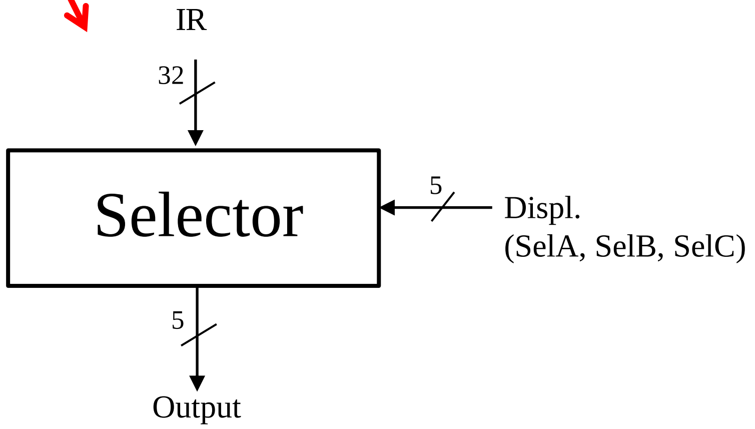
Select 5 bits within 32-bit starting from the position indicated in *Displ* (lower bit)



Register file selector



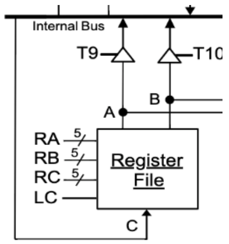


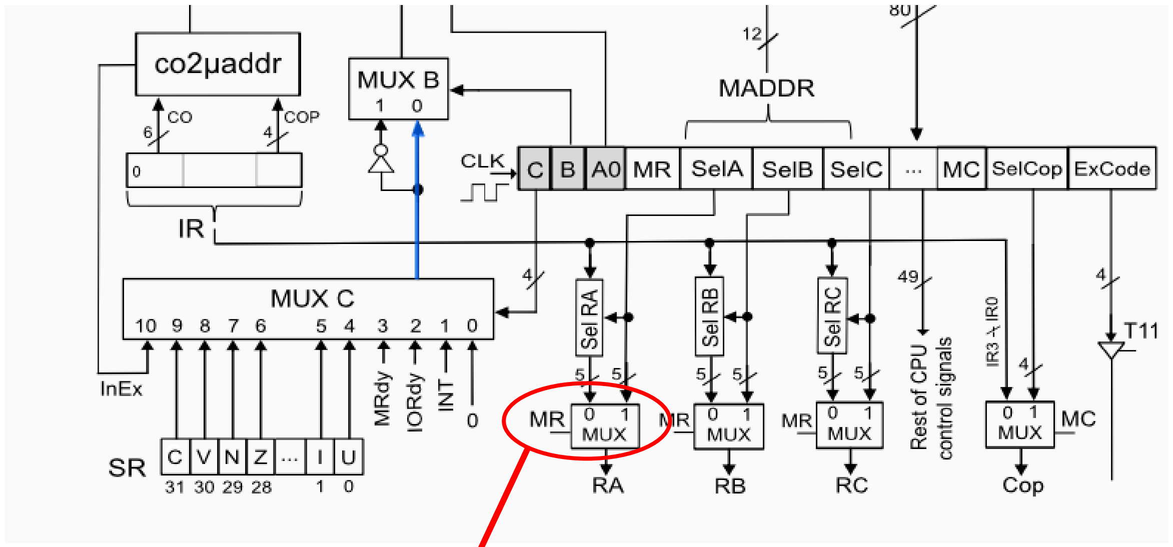


IR: $D_{31}D_{30}D_{29}D_{28}D_{27}D_{26}D_{25}$ ……. $D_4D_3D_2D_1D_0$

If Displ = 11011	→	Output =	$D_{31}D_{30}D_{29}D_{28}D_{27}$
If Displ = 00000	→	Output =	$D_4D_3D_2D_1D_0$
If Displ = 10011	→	Output =	$D_{23}D_{22}D_{21}D_{20}D_{19}$
If Displ = 01011	→	Output =	$D_{15}D_{14}D_{13}D_{12}D_{11}$

Register file selector

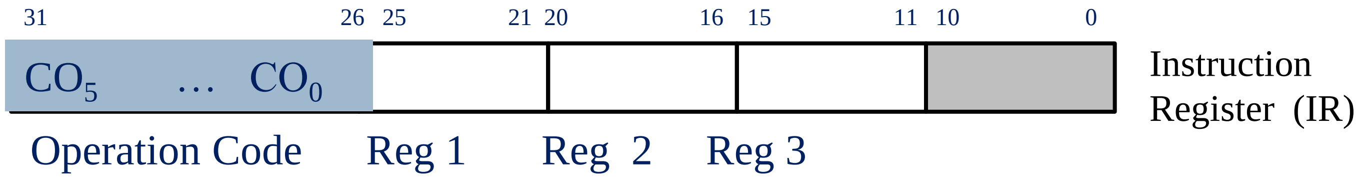


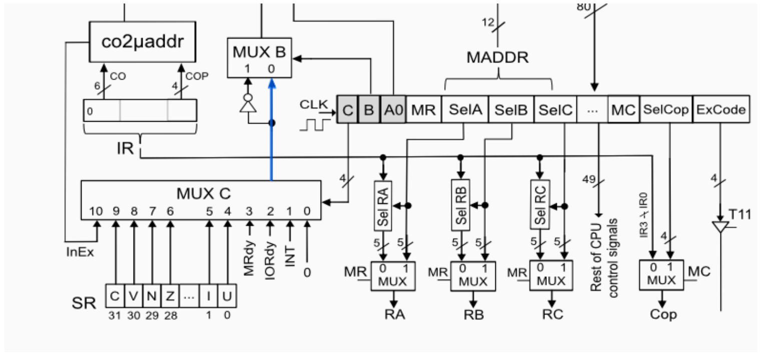


- If MR = 1, RA is obtained directly from the µInstruction
- If MR = 0, RA is obtained from a field of the instruction (in IR)

Register file selector

- If the format of an instruction stored in IR is:

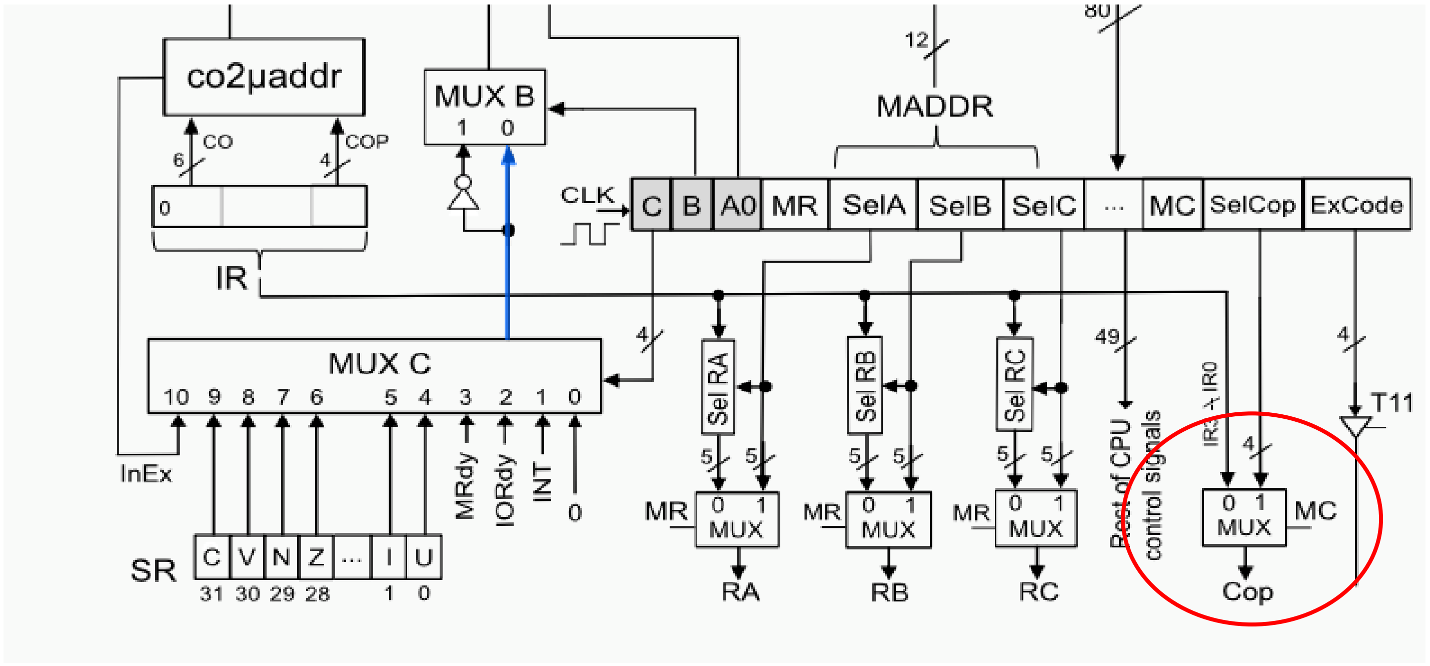




$\boxed{MR = 0}$

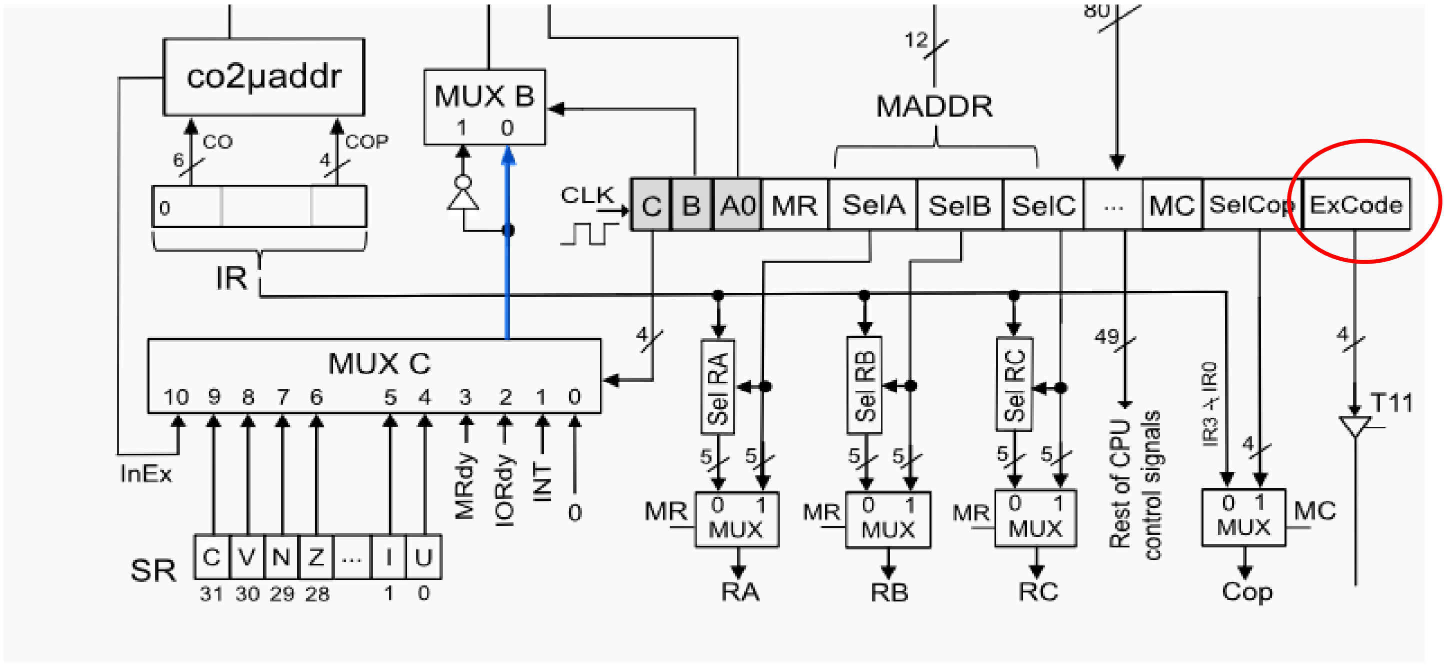
- If you want to select the field with the Reg 2 in port B of the register file → SelB = 10000 (RB is obtained from bits 20…16 of IR)
- If you want to select the field with the Reg 3 in port A of the register file → SelA = 01011 (RA is obtained from bits 15…11 of IR)
- If you want to select the field with the Reg 1 in port C of the register file → SelC = 10101 (RC is obtained from bits 25…21 of IR)

Selection of the ALU operation code



- If **MC = 1**, the operation code of the ALU is obtained directly from the microinstruction (**SelCop**)
- If **MC = 0**, the operation code of the ALU is obtained from the **last four bits** stored in the **instruction register** (IR)

Exception codes



- ExCode:
  - Allows to have an immediate value of any 4 bits,
  - Especially useful for generating the interrupt vector to be used when an exception occurs in the instruction.

Contents

Example

- Instruction for microprogramming with WepSIM*:

Instruction	Operation code	Meaning
ADD Rd, Rf1, Rf2	000000	Rd ← Rf1+ Rf2
LI R, value	000001	R ← value
LW R, addr	000010	R ← MP[addr]
SW R, addr	000011	MP[addr] ← R
BEQ Rf1, Rf2, off1	000100	if (Rf1 == Rf2) PC ← PC +off1
J addr	000101	PC ← addr
HALT	000110	HALT (infinite loop)

* Memory answer in one cycle

Design with the WepSIM control unit

- For each machine instruction:
  1. Define the behavior using RTL (register transfer language) for every clock cycle

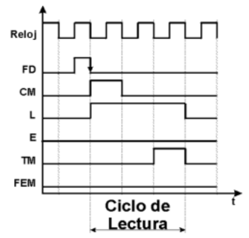
Instruction *mv R0 R1*

Sequence of **elementary operations**

1. IR <- [PC]
2. PC++
3. decode
4. *R0* <- *R1*

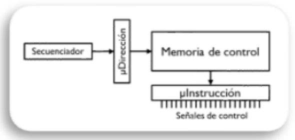
  2. Translate the behavior to values of each control signal at each clock cycle

Sequence of **control signals** for each elementary operation



  3. Design a circuit that generates the value of each control signal at each clock cycle

Circuit that generates signals: Microprogrammed control



Microprogrammed instructions

- FETCH

Cycle	Elemental Op.		
0	MAR ← PC		
1	MBR ← MP		
	PC ← PC + 4		
2	IR ← MBR		
3	Decodificación		

Design with the WepSIM control unit

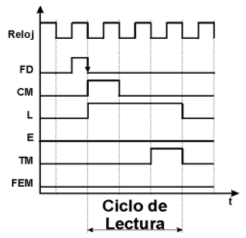
- For each machine instruction:
  1. Define the behavior using RTL (register transfer language) for every clock cycle
  2. Translate the behavior to values of each control signal at each clock cycle
  3. Design a circuit that generates the value of each control signal at each clock cycle

Instruction *mv R0 R1*

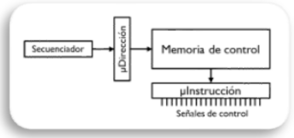
Sequence of **elementary operations**

1. IR <- [PC]
2. PC++
3. decode
4. *R0* <- *R1*

Sequence of **control signals** for each elementary operation



Circuit that generates signals: Microprogrammed control



Microprogrammed instructions

WepSIM: FETCH example

- FETCH

Cycle	Elemental Op.	Activated signals (rest to 0)	C	B	A0
0	MAR ← PC	T2, C0	0000	0	0
1	MBR ← MP	Ta, R, BW=11, C1, M1	0000	0	0
	PC ← PC + 4	M2, C2	0000	0	0
2	IR ← MBR	T1, C3	0000	0	0
3	Decode		0000	0	1

Design with the WepSIM control unit

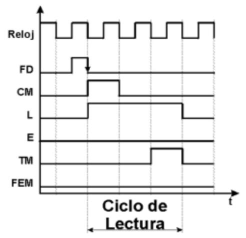
- For each machine instruction:
  1. Define the behavior using RTL (register transfer language) for every clock cycle
  2. Translate the behavior to values of each control signal at each clock cycle
  3. Design a circuit that generates the value of each control signal at each clock cycle

Instruction *mv R0 R1*

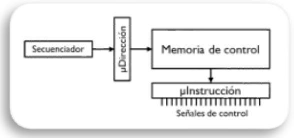
Sequence of **elementary operations**

1. IR <- [PC]
2. PC++
3. decode
4. *R0* <- *R1*

Sequence of **control signals** for each elementary operation



Circuit that generates signals: Microprogrammed control



Microprograms in WepSIM

C.	E.O.	Activated signals
0	MAR ← PC	T2, C0
1	MBR ← MP, PC ← PC + 4	Ta, R, BW=11, C1, M1, M2, C2
2	IR ← MBR	T1, C3
3	Decod.	A0=1, B=0, C=0

Skeleton

<List of microcodes>

<Register section>

<Pseudoinstrucions>

```
begin
{
    fetch: (T2, C0=1),
           (Ta, R, BW=11, C1, M1),
           (M2, C2, T1, C3),
           (A0, B=0, C=0)
}

registers {
        0=(zero,  x0),    1=(ra, x1),     2=(sp, x2)(stack_pointer),
        3=(gp,    x3),    4=(tp, x4),     5=(t0, x5),
        6=(t1,    x6),    7=(t2, x7),     8=(s0,   x8),
        9=(s1,    x9),   10=(a0,  x10),  11=(a1,  x11),
       12=(a2,   x12),   13=(a3,  x13),  14=(a4,  x14),
       15=(a5,   x15),   16=(a6,  x16),  17=(a7,  x17),
       18=(s2,   x18),   19=(s3,  x19),  20=(s4,  x20),
       21=(s5,   x21),   22=(s6,  x22),  23=(s7,  x23),
       24=(s8,   x24),   25=(s9,  x25),  26=(s10, x26),
       27=(s11,  x27),   28=(t3,  x28),  29=(t4,  x29),
       30=(t5,   x30),   31=(t6,  x31)
}
```

Microprogrammed instructions

- ADD Rd, Rf1, Rf2

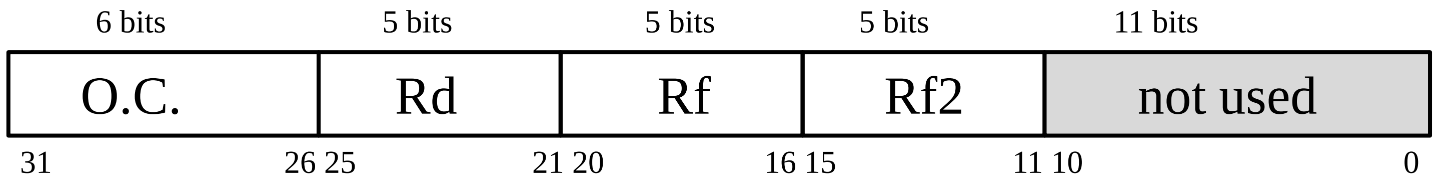
Cycle	Elemental Op.	Activated signals (rest to 0)	C B A0
0	Rd ← Rf1 + Rf2	SelA=10000 (16), SelB=01011 (11), MC=0, T6, SelP=11, C7, M7, SelC=10101 (21), LC	0000 1 1

6 bits	5 bits	5 bits	5 bits	7 bits	4 bits
000000	Rd	Rf1	Rf2	not used	1010
31 26	25 21	20 16	15 11	10 4	3 0

Microprogrammed instructions

- ADD Rd, Rf1, Rf2

Cycle	Elemental Op.	Activated signals (rest to 0)	C B A0
0	Rd ← Rf1 + Rf2	SelA=10000 (16), SelB=01011 (11), SelCop=1010, MC=1, SelP=11, C7, M7, T6, SelC=10101 (21), LC	0000 1 1



Defining instructions in WepSIM

WepSIM: ADD examples

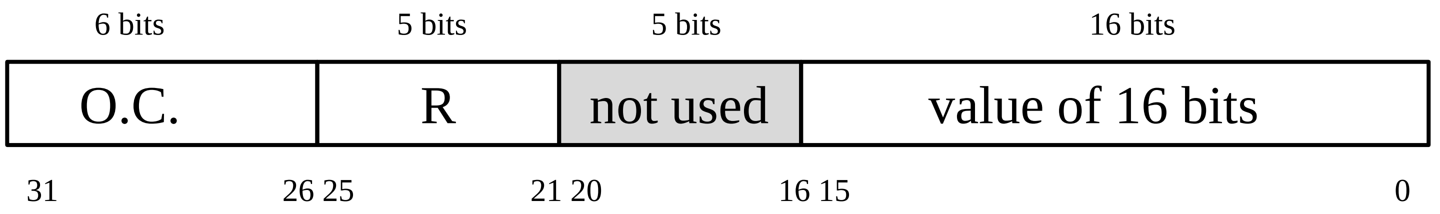
```
ADD R1,R2, R3 {
        co=100000,
        nwords=1,
        R1=reg(25,21),
        R2=reg(20,16),
        R3=reg(15,11),
        {
            (SelA=01011, SelB=10000,
             SelCop=1010, MC,  T6,  SelP=11, M7,C7,
             SelC=10101, LC,         A0=1, B=1, C=0)
        }
}
```

Microprogrammed instructions

- LI R, value

Cycle	Elemental Op.	Activated signals (rest to 0)	C B A0

0	R ← IR (value)	LC
		SelC = 10101 (21)
		T3,
		Size = 10000
		Offset= 00000
		SE=1
			0000 1 1



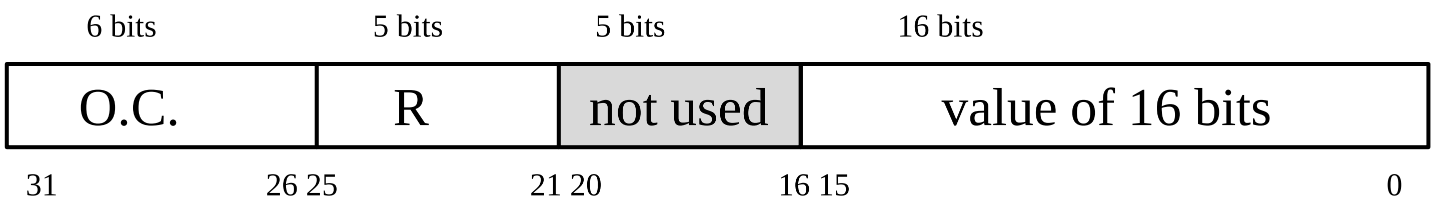
Microprogrammed instructions

- LW R addr   # sync memory, 1 clock cycle

Cycle	Elemental Op.	Activated signals (rest to 0)	C B A0

0	MAR ← IR (addr)	T3, C0
		Size = 10000,
		Offset= 00000
			0000 0 0

1	MBR ← MP[MAR]	Ta, R, BW =11, C1, M1	0000 0 0
2	R ← MBR	T1, LC, SelC = 10101	0000 1 1



Microprogrammed instructions

- LW R addr # async memory (MRdy=1 for ready)

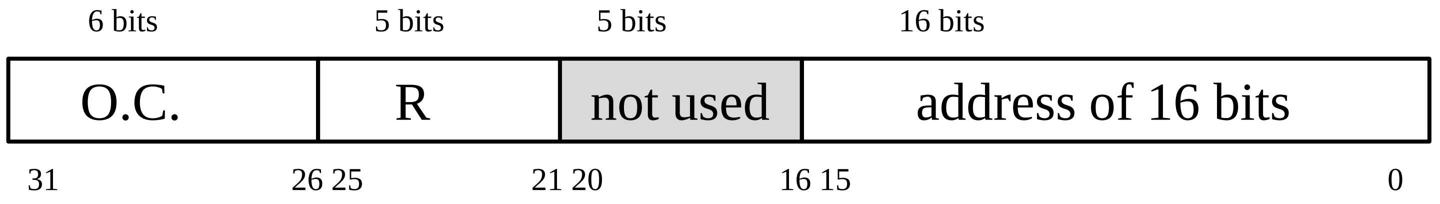
Ciclo	Op. Elemental	Señales activadas (resto a 0)	C	B	A0
0	MAR ← IR (addr)	T3, C0 Size = 10000, Offset= 00000	0000	0	0
1	while (!MRdy) MBR ← MP[MAR]	Ta, R, BW =11, C1, M1, MADDR=μAdd of this μinstruction	0011	1	0
2	R ← MBR	T1, LC, SelC = 10101	0000	1	1

This microinstruction is beening executed while MRdy==0

Microprogrammed instructions

- SW R addr   # sync memory, 1 clock cycle

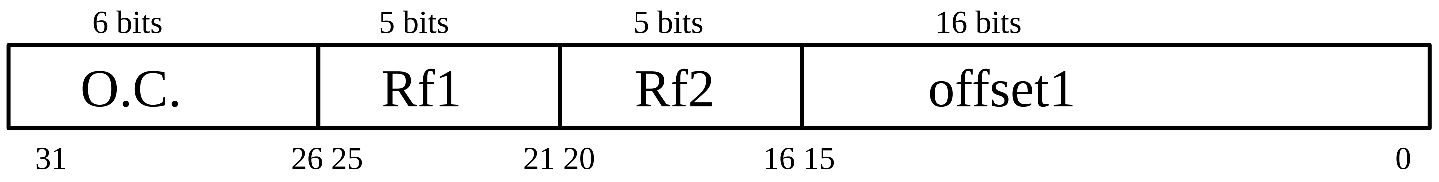
Ciclo	Op. Elemental	Señales activadas (resto a 0)	C	B	A0
0	MBR ← R	T9, C1, SelA=10101	0000	0	0
1	MAR ← IR(addr)	T3, C0, Size = 10000, offset= 00000	0000	0	0
2	MP[addr] ← MBR	Td, Ta, BW = 11, W	0000	1	1



Microprogrammed instructions

- BEQ Rf1, Rf2, offset1

Cycle	Elemental Op.	Activated signals (rest to 0)	C B A0
0	Rf1 - Rf2	SelA=10101, SelB=10000, SelCop=1011, MC, C7, M7, SelP=11	0000 0 0
1	If (Z == 0) goto fetch else next	MADDR = 0	0110 1 0
2	RT1 ←PC	T2, C4	0000 0 0
3	RT2 ← IR(offset1)	Size = 10000, Offset = 00000, T3,C5	0000 0 0
4	PC ← RT1 +RT2	MA, MB=01, SelCop=1010, MC,T6, C2	0000 1 1



Defining instructions in WepSIM

WepSIM: BEQ example

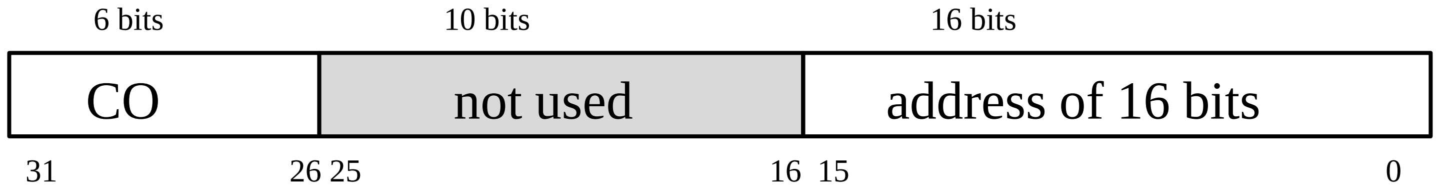
label, represents a µaddress

```
BEQ R1, R2, desp {
          co=000100,
          nwords=1,
          R1=reg(25,21),
          R2=reg(20,16),
          desp=address(15,0)rel,
          {
               (T8, C5),
               (SELA=10101, SELB=10000, MC=1, SELCOP=1011, SELP=11, M7, C7),
               (A0=0, B=1, C=110, MADDR=bck2ftch),
               (T5, M7=0, C7),
               (T2, C4),
               (SE=1, OFFSET=0, SIZE=10000,T3, C5),
               (MA=1, MB=1, MC=1, SELCOP=1010,T6, C2,A0=1, B=1, C=0),
  bck2ftch:    (T5, M7=0, C7),
               (A0=1, B=1, C=0)
         }
}
```

Microprogrammed instructions

- J addr

Cycle	Elemental Op.	Activated signals (rest to 0)	C B A0
0	PC ← IR (addr)	C2, T3, size = 10000, offset= 00000	0000 1 1



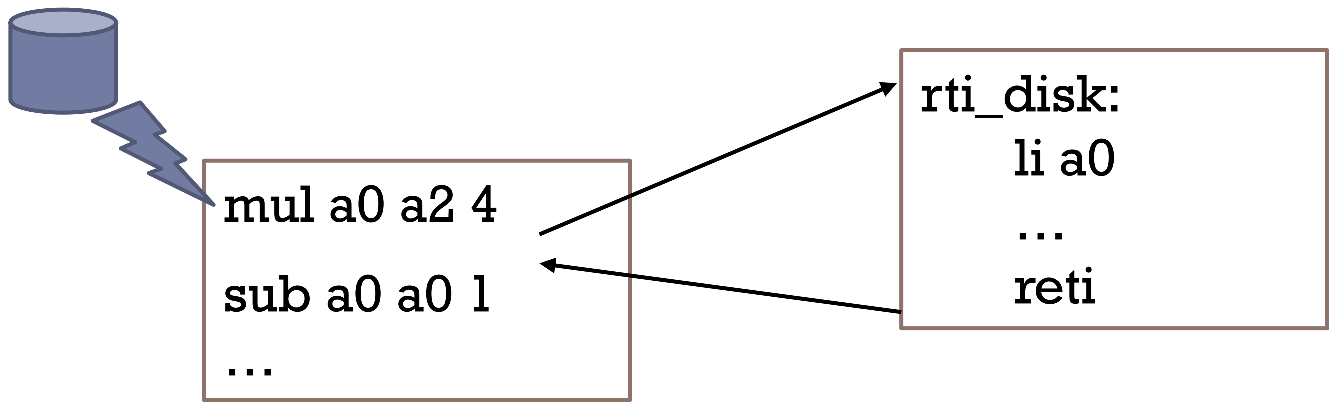
Contenido

Execution modes

- It is indicated by a bit in the status register (U)
- At least 2 modes:
  - User Mode
    - The processor cannot execute privileged instructions (e.g.: I/O instructions, interrupt enable instructions, ...)
    - If a user process executes a privileged instruction, an interruption (exception) occurs
  - Kernel Mode
    - Reserved to the operating system
    - The processor can execute the entire repertoire of instructions

Contenido

Interrupts: panoramic view



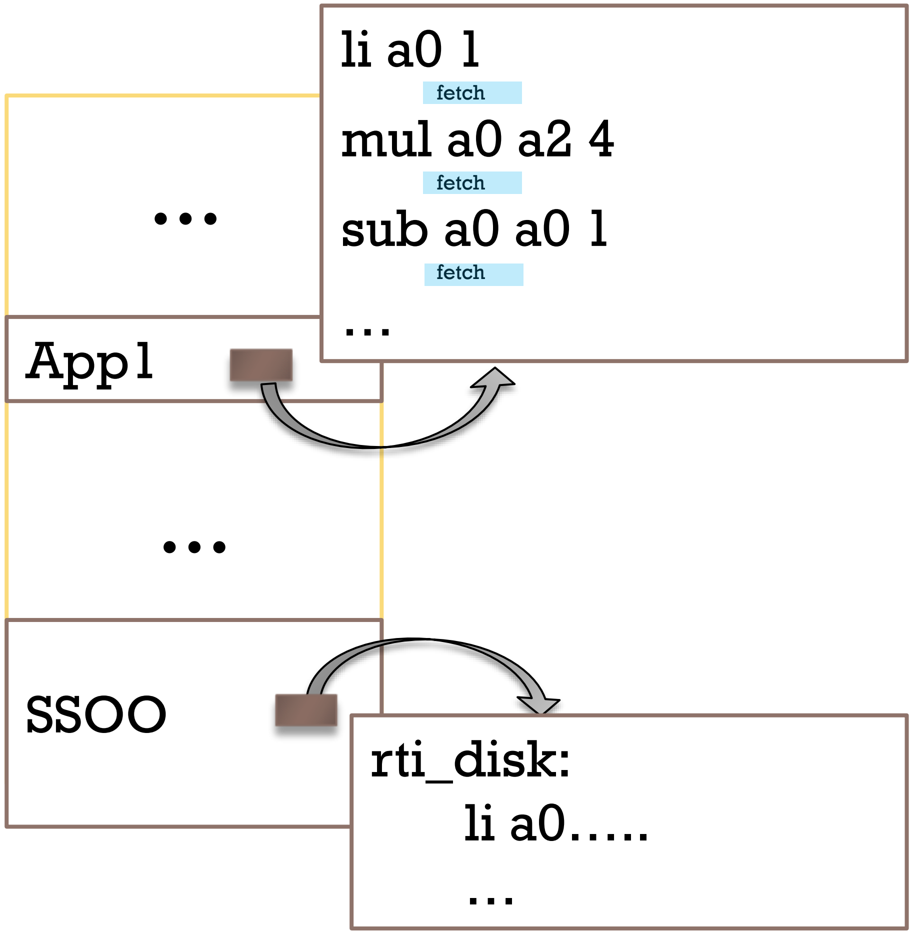
- Condition detected by the Control Unit that breaks the normal execution sequence:
  - The current program is stopped.
  - The execution is transferred to another program that attend the interruption (Interrupt Service Routine a.k.a. ISR)
  - When the ISR ends, the execution of the interrupted program is resumed.
- Example of causes:
  - When a peripheral requests the attention of the processor,
  - When an error occurs in the execution of the instruction, Etc.

Classification of interruptions

- Asynchronous
  - Excepciones hardware asíncronas
    - Faults or errors in hardware not related to current instruction: printer without paper, power failure, etc.
  - External interruptions
    - When a peripheral (or CPU) requests the attention of the CPU: Peripherals, clock interruption, etc.
- Synchronous
  - Synchronous hardware exceptions
    - When an error occurs in the execution of the instruction: Division by zero, access to an illegal memory position, etc.
  - System calls
    - Special machine instructions that generate an interruption to activate the operating system (request an operating system service)

Asynchronous Hardware Exceptions and External Interrupts

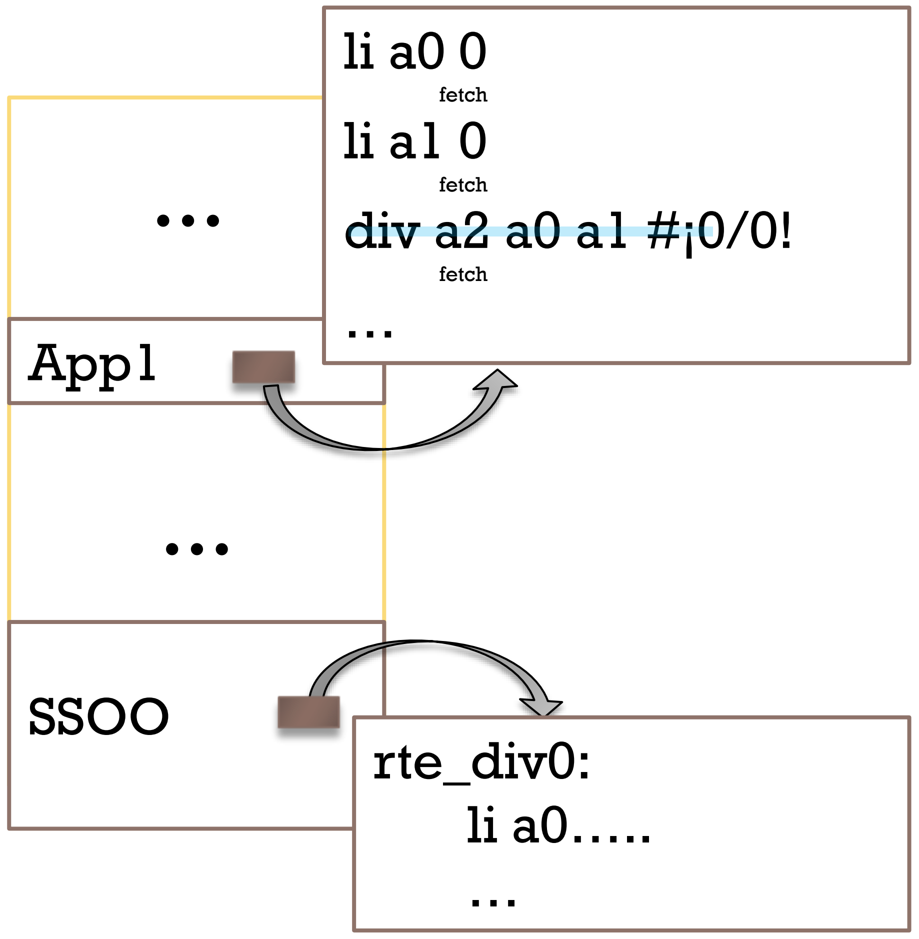
WepSIM: int, syscall, exception…



- They cause an unscheduled sequence break
  - **Before doing the fetch cycle, first see if there is any pending interruption, and if so…**
  - …Bifurcation to subroutine of the O.S. that treats it
- It then restores the status and returns control to the interrupted program.
- **Asynchronous cause to the execution of the current program**
  - Peripheral care
  - Etc.

Synchronous Hardware Exceptions and System Calls

WepSIM: int, syscall, exception…



- They cause an unscheduled sequence break
  - **Within the microprogram of the ongoing instruction…**
  - …Bifurcation to subroutine of the O.S. that treats it
- It restores the status and returns control to the interrupted program **or ends its execution**
- **Synchronous cause to the execution of the current program**
  - Division between zero
  - Etc.

Interrupt Acknowledge Cycle (IAC)

WepSIM: int, syscall, exception…

- It is a microcode before the fetch cycle
  - It handles the asynchronous interrupts
- General structure of the IAC:
  1. Checks if an interruption signal is activated.
  2. If it is activated:
     1. Saves the program counter and status register
        - Equivalent to "push pc, push sr"
     2. Switches from user mode to kernel mode
        - Equivalent to "SR.U = 0"
     3. Obtains the address of the Interrupt Service Routine (ISR)
        - Equivalent to "isr_addr = Vector_interrupts[id_interrupt]"
     4. Store the address obtained in the program counter (this way the following instruction will be the first one for the treatment routine)
        - Equivalent to "PC = isr_addr"

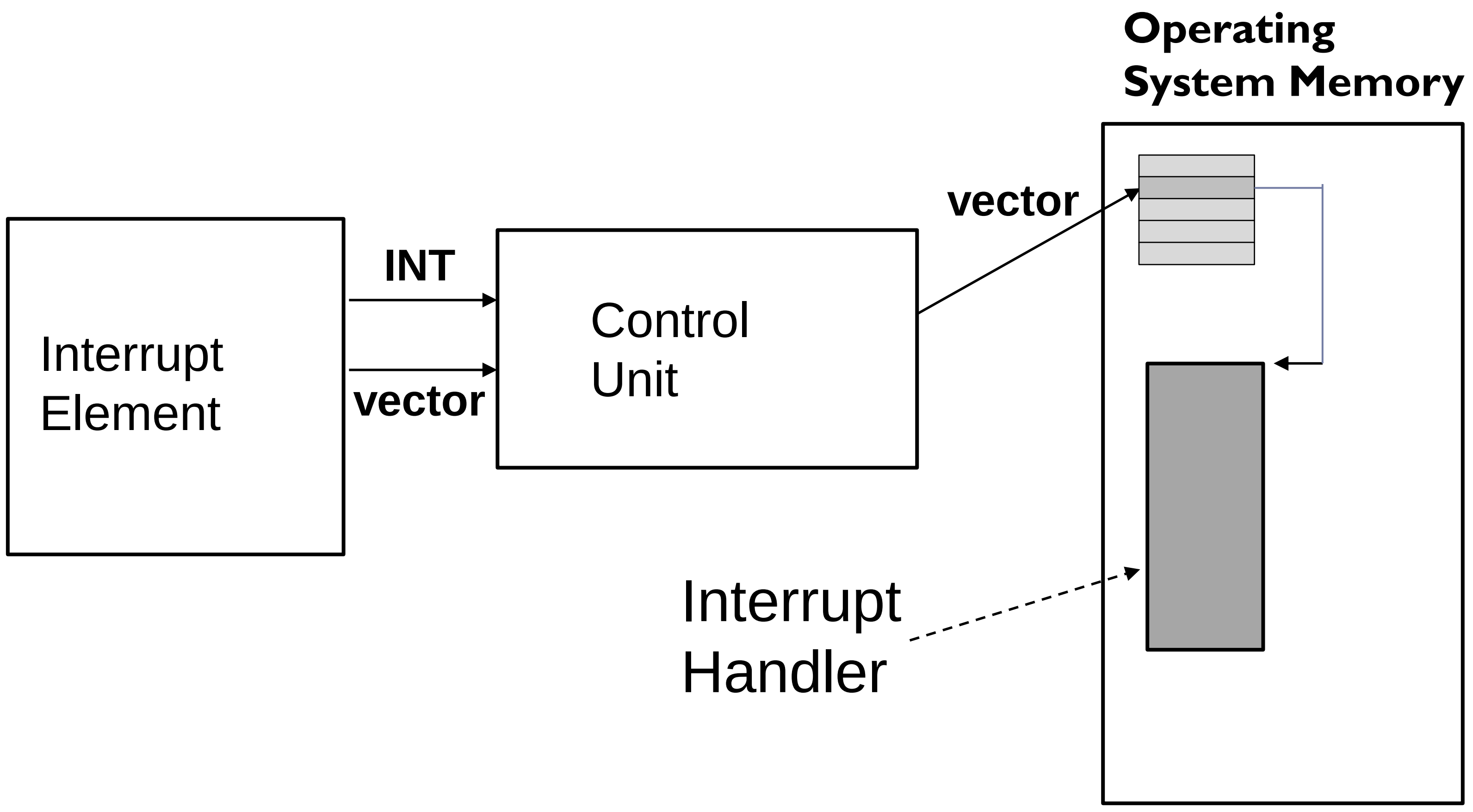
Interrupt Service Routine (ISR)

WepSIM: int, syscall, exception…

- It is part of the operating system code
  - There is one ISR for each interruption that may occur
- General structure of the ISR:

1. Saves the rest of the processor registers (if required)
2. Service the interrupt
3. Restores processor registers saved in (2)
4. Executes a special machine instruction: RETI
   - Resets the status register of the interrupted program (by setting the processor mode back to user mode).
   - Resets the program counter (so that the next instruction is that of the interrupted program).

Vector interrupts

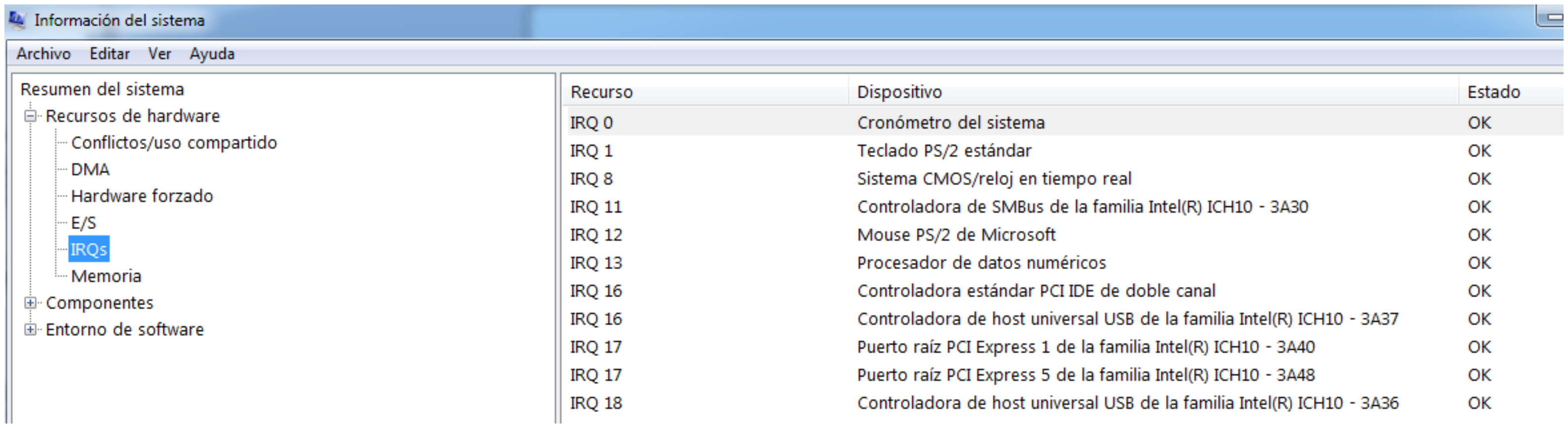


Vector interrupts

- A table of memory addresses of the processing routines associated with each interrupt is used:
  - The interrupting element supplies the interrupt vector.
  - This vector is an index in a table (stored at main memory) containing the address of the interrupt handler routine.
  - The Control Unit reads the content of this entry and loads the value into the PC register.
- Each operating system fills this table with the addresses of each of the treatment routines (there are dependent on each operating system) at boot time.

Interrupts in a PC

- Windows



Información del sistema

Archivo Editar Ver Ayuda

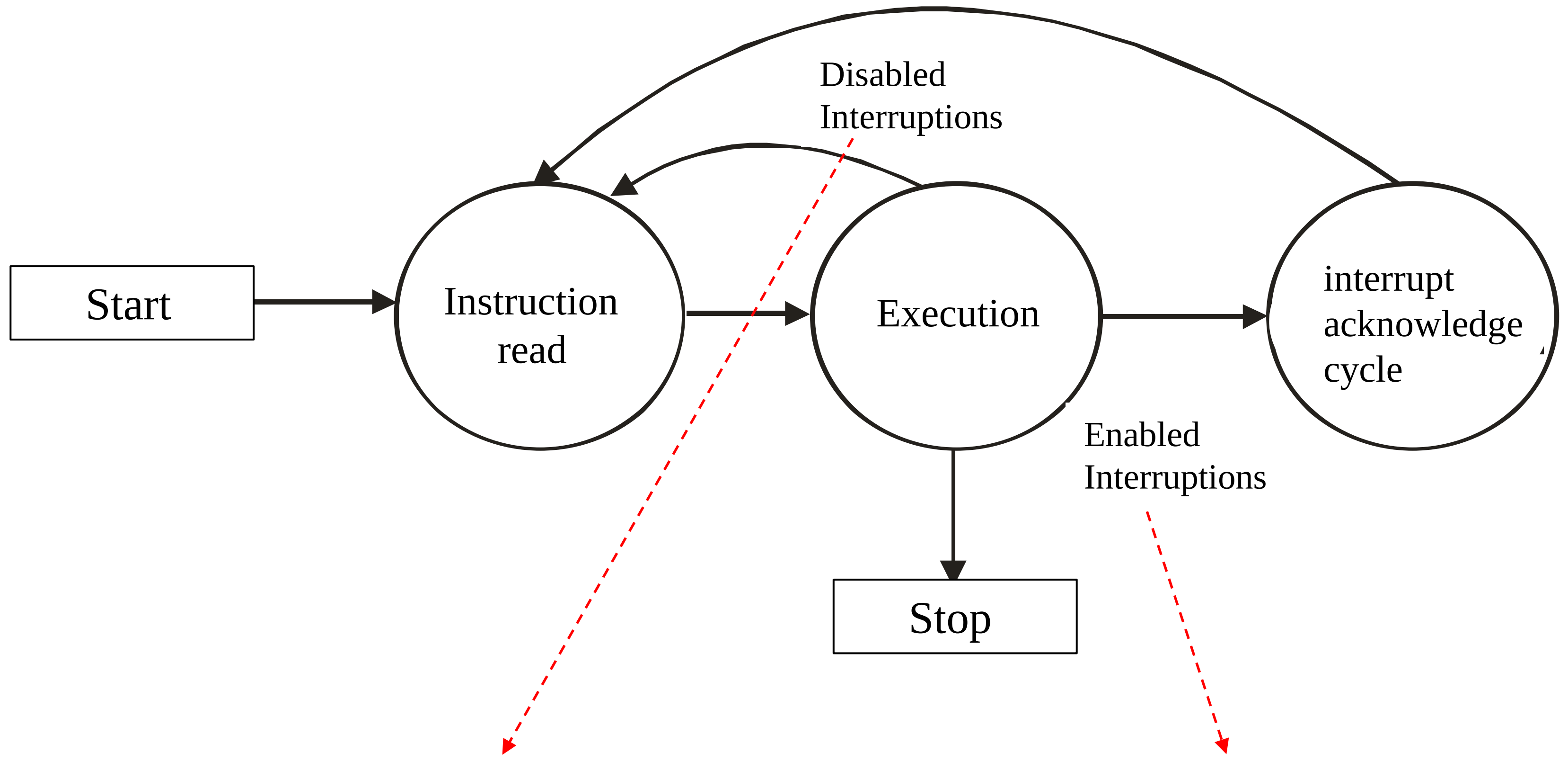
- Resumen del sistema
  - Recursos de hardware
    - Conflictos/uso compartido
    - DMA
    - Hardware forzado
    - E/S
    - IRQs
    - Memoria
  - Componentes
  - Entorno de software

Recurso	Dispositivo	Estado
IRQ 0	Cronómetro del sistema	OK
IRQ 1	Teclado PS/2 estándar	OK
IRQ 8	Sistema CMOS/reloj en tiempo real	OK
IRQ 11	Controladora de SMBus de la familia Intel(R) ICH10 - 3A30	OK
IRQ 12	Mouse PS/2 de Microsoft	OK
IRQ 13	Procesador de datos numéricos	OK
IRQ 16	Controladora estándar PCI IDE de doble canal	OK
IRQ 16	Controladora de host universal USB de la familia Intel(R) ICH10 - 3A37	OK
IRQ 17	Puerto raíz PCI Express 1 de la familia Intel(R) ICH10 - 3A40	OK
IRQ 17	Puerto raíz PCI Express 5 de la familia Intel(R) ICH10 - 3A48	OK
IRQ 18	Controladora de host universal USB de la familia Intel(R) ICH10 - 3A36	OK

- Linux

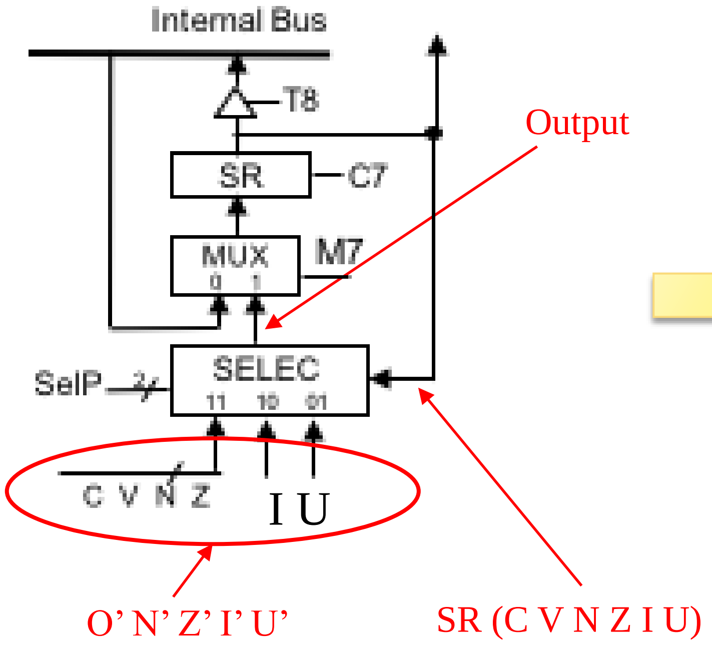
```
cloud9@lab.inf:~$ cat /proc/interrupts
           CPU0
  0:         33   IO-APIC   2-edge      timer
  1:        171   IO-APIC   1-edge      i8042
  6:          3   IO-APIC   6-edge      floppy
  8:          1   IO-APIC   8-edge      rtc0
  9:          0   IO-APIC   9-fasteoi   acpi
 11:         36   IO-APIC  11-fasteoi   virtio3, uhci_hcd:usb1
 12:         15   IO-APIC  12-edge      i8042
 14:          0   IO-APIC  14-edge      ata_piix
 15:     289039   IO-APIC  15-edge      ata_piix
...
NMI:          0   Non-maskable interrupts
LOC:    5397142   Local timer interrupts
SPU:          0   Spurious interrupts
PMI:          0   Performance monitoring interrupts
IWI:          0   IRQ work interrupts
...
```

Activation of the status register



It is indicated by a bit located in the status register (I)

Activation of the status register



SELEC operation:

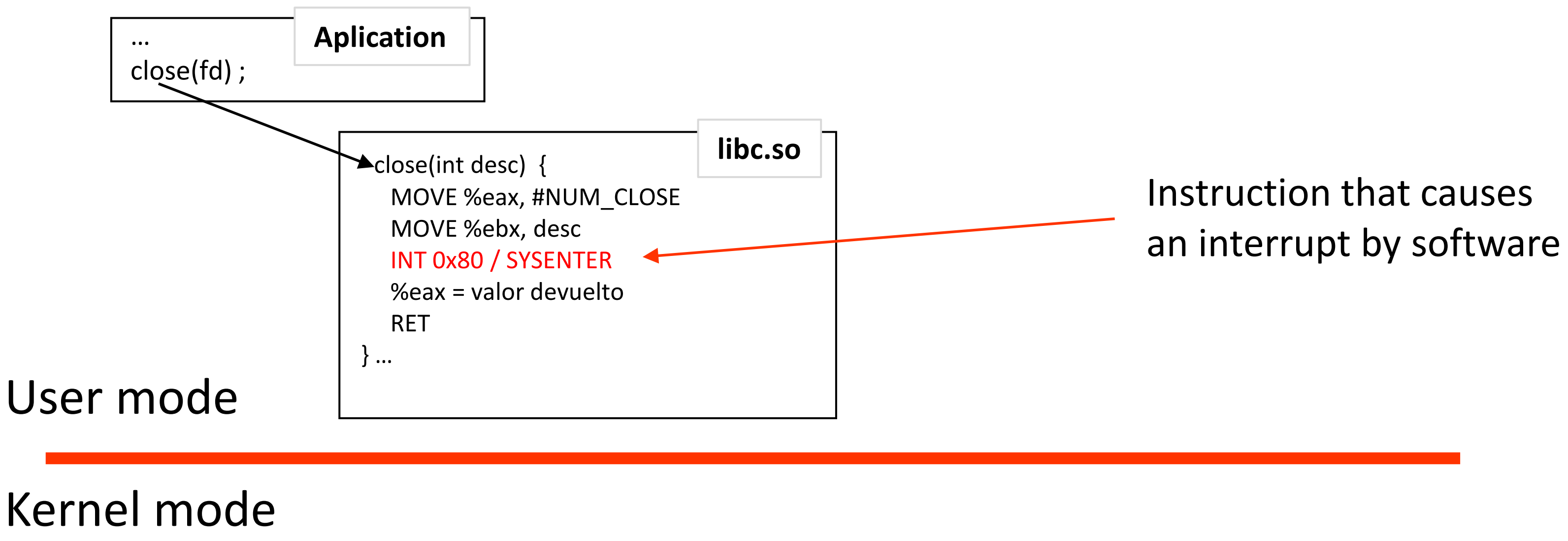
```
if  (SelP1 = 1  AND  SelP0 == 1)
    Output = C' V' N' Z' I U

if  (SelP1 == 1 AND  SelP0 ==0)
    Output = C V N Z I' U

if (SelP1 == 0  AND  SelP0 == 1)
    Output = C V N Z I U'
```

Interrupts by Software. System calls and operating systems

WepSIM: syscall example

- The system call mechanism is the one that allows user programs to request the services offered by the operating system
  - Load programs into memory for execution
  - Access to peripheral devices
  - Etc.
- Similar to the system calls offered by the CREATOR simulator
  - WepSIM examples show how system calls are internally implemented.

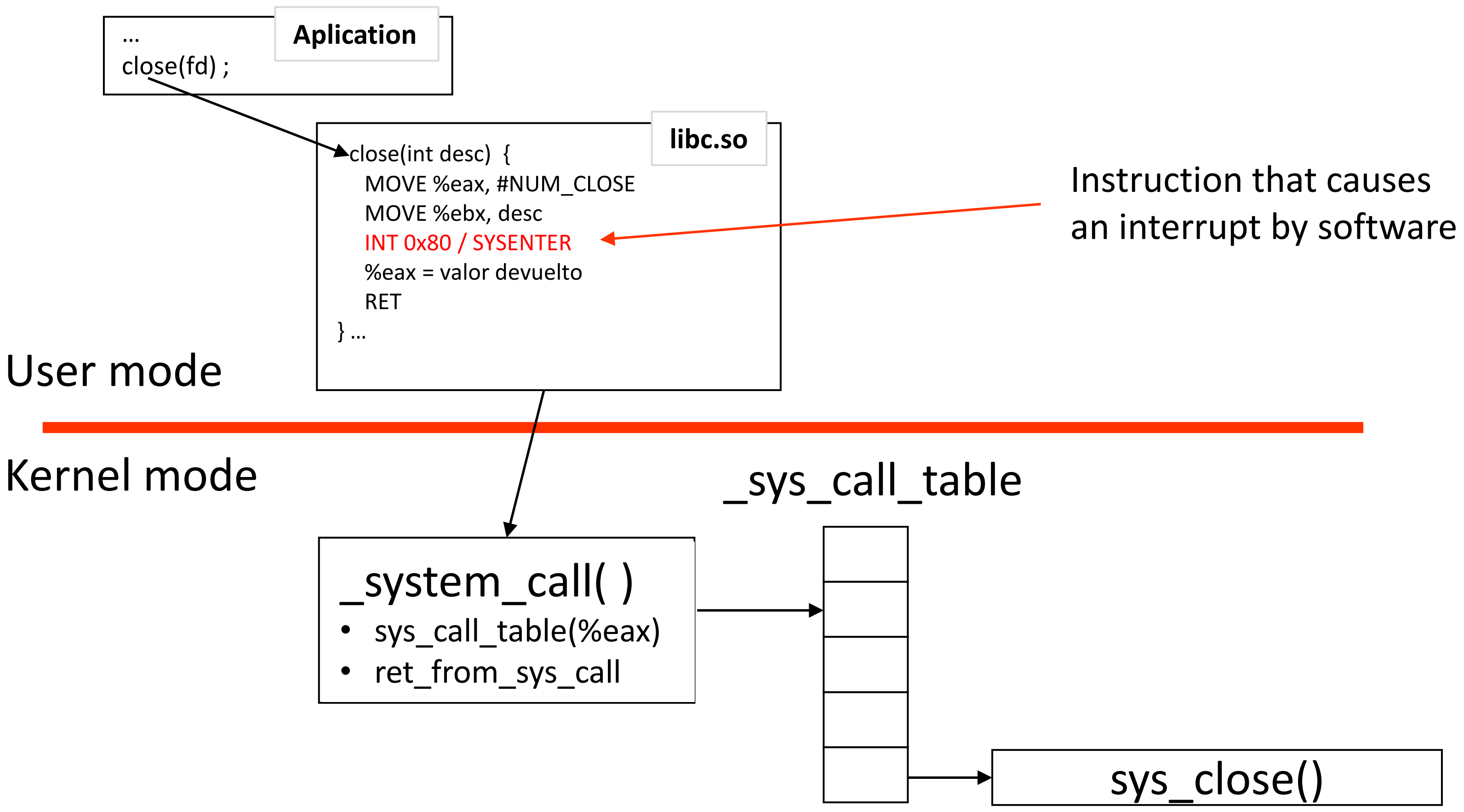
Interrupts by Software.
System calls (example: Linux)



```
…
close(fd) ;
```

```
close(int desc)  {
   MOVE %eax, #NUM_CLOSE
   MOVE %ebx, desc
   INT 0x80 / SYSENTER
   %eax = valor devuelto
   RET
} …
```

Interrupts by Software. System calls (example: Linux)



```
…
close(fd) ;
```

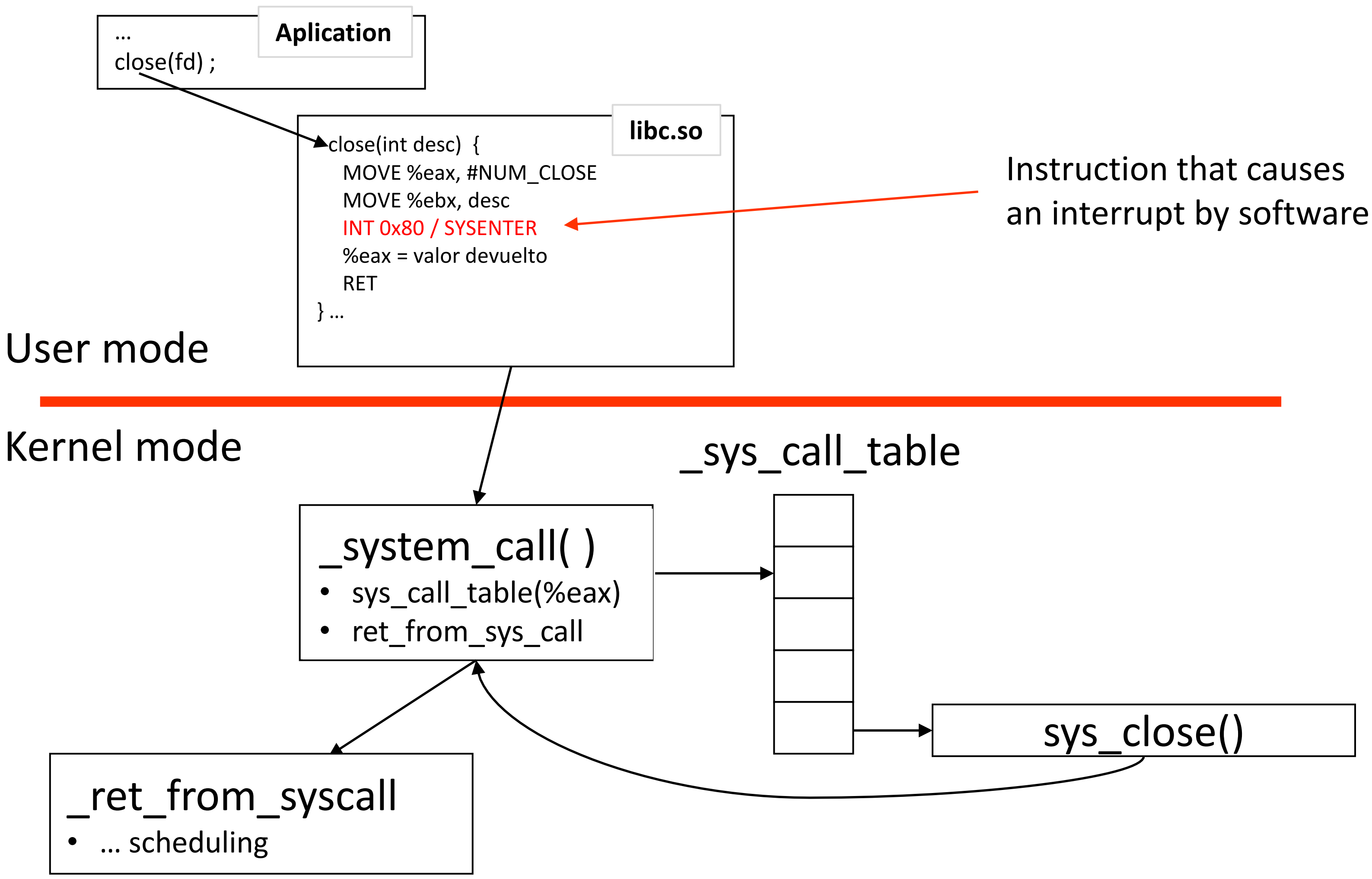
```
close(int desc)  {
   MOVE %eax, #NUM_CLOSE
   MOVE %ebx, desc
   INT 0x80 / SYSENTER
   %eax = valor devuelto
   RET
} …
```

_system_call( )

- sys_call_table(%eax)
- ret_from_sys_call

sys_close()

Interrupts by Software. System calls (example: Linux)



Clock interrupts and the operating system

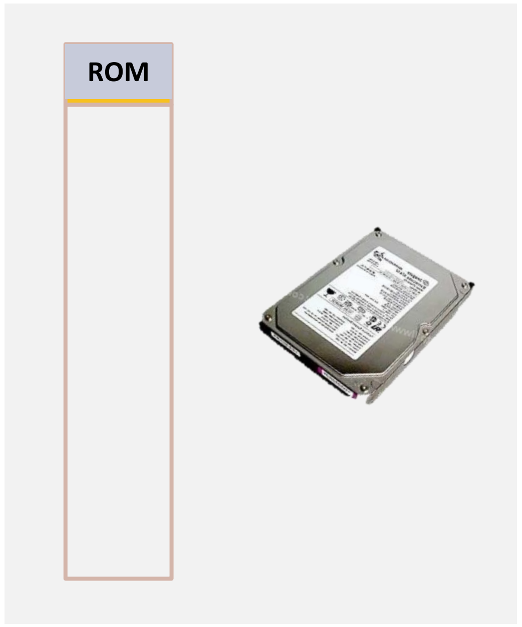
WepSIM: example clock int.

- The signal that governs the execution of machine instructions is divided by a frequency divider to generate an external interruption every certain time interval (a few milliseconds)
- These clock interruptions or ticks are periodic interruptions that allow the operating system to come in and run periodically, preventing a user program from monopolizing the CPU
  - Allows to alternate the execution of various programs on a system given the appearance of simultaneous execution
  - Each time a clock interruption arrives, the program is suspended and the operating system that runs the scheduler is skipped to decide the next program to run

Contenido

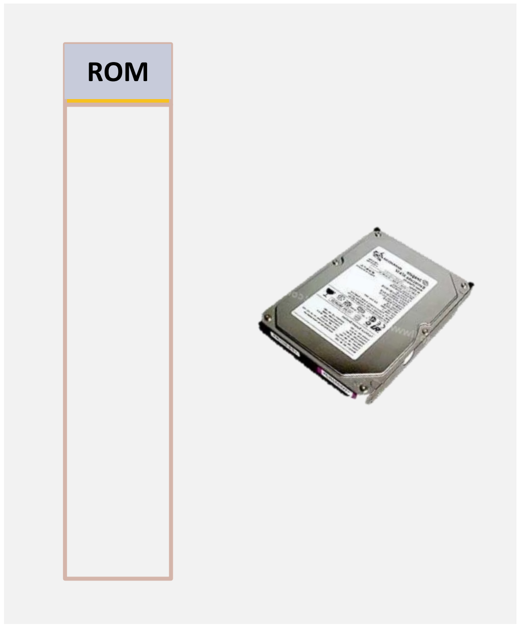
Computer booting

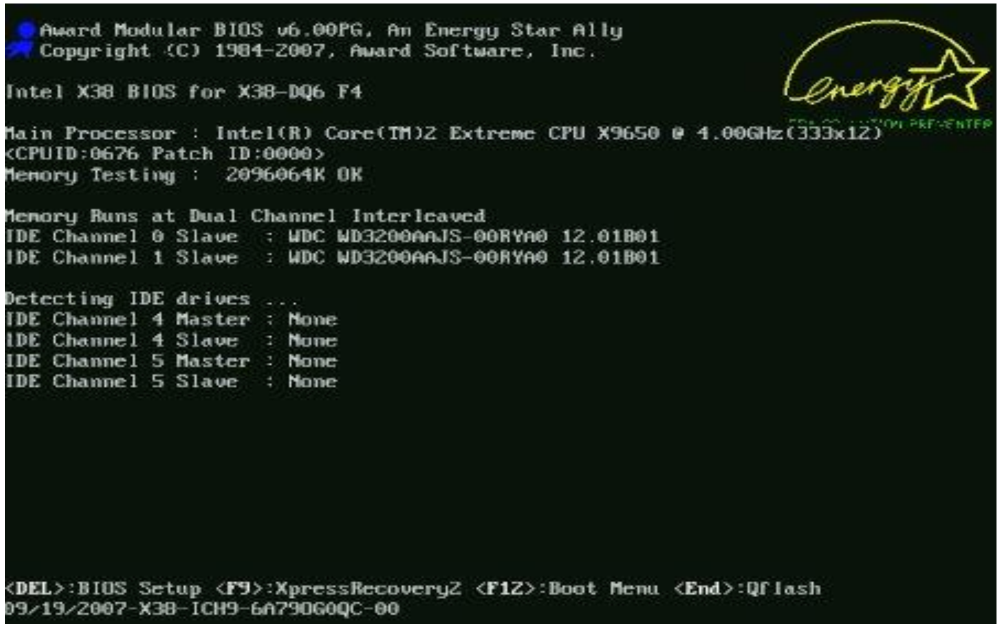
- The *Reset* loads the predefined values in registers:
  - PC ← initial address of the initialization program (in ROM memory)



Computer booting

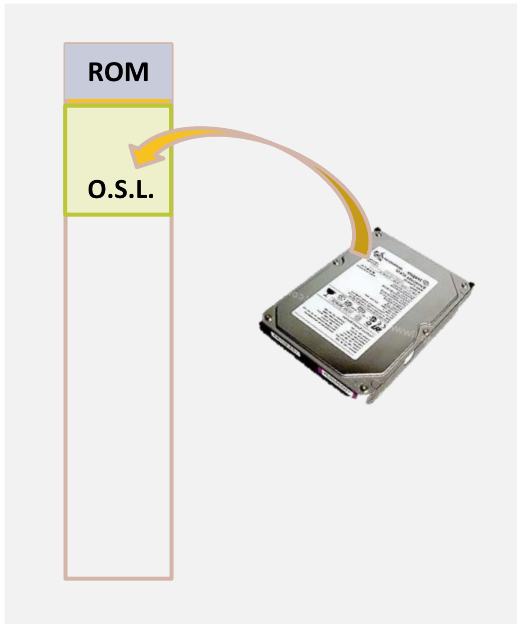
- The *Reset* loads the predefined values in registers:
  - PC ← initial address of the initialization program (in ROM memory)
- The initialization program is executed:
  - System test (POST)





```
Award Modular BIOS v6.00PG, An Energy Star Ally
Copyright (C) 1984-2007, Award Software, Inc.

Intel X38 BIOS for X38-DQ6 F4

Main Processor : Intel(R) Core(TM)2 Extreme CPU X9650 @ 4.00GHz(333x12)
<CPUID:0676 Patch ID:0000>
Memory Testing :  2096064K OK

Memory Runs at Dual Channel Interleaved
IDE Channel 0 Slave   : WDC WD3200AAJS-00RYA0 12.01B01
IDE Channel 1 Slave   : WDC WD3200AAJS-00RYA0 12.01B01

Detecting IDE drives ...
IDE Channel 4 Master : None
IDE Channel 4 Slave  : None
IDE Channel 5 Master : None
IDE Channel 5 Slave  : None






<DEL>:BIOS Setup <F9>:XpressRecovery2 <F12>:Boot Menu <End>:Qflash
09/19/2007-X38-ICH9-6A79OG0QC-00
```

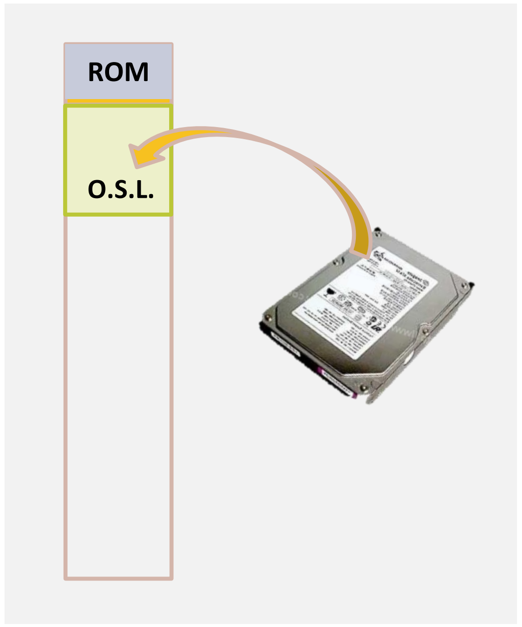
Computer booting

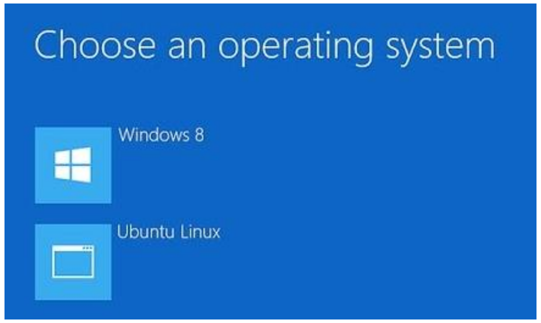


- The *Reset* loads the predefined values in registers:
  - PC ← initial address of the initialization program (in ROM memory)
- The initialization program is executed:
  - System test (POST)
  - Load into memory the operating system loader (MBR)

Computer booting

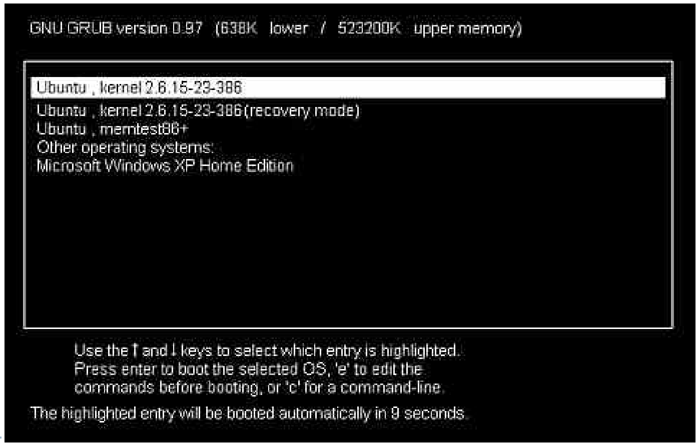
- The *Reset* loads the predefined values in registers:
  - PC ← initial address of the initialization program (in ROM memory)
- The initialization program is executed:
  - System test (POST)
  - Load into memory the operating system loader (MBR)
- The Operating System Loader is executed:
  - Sets boot options



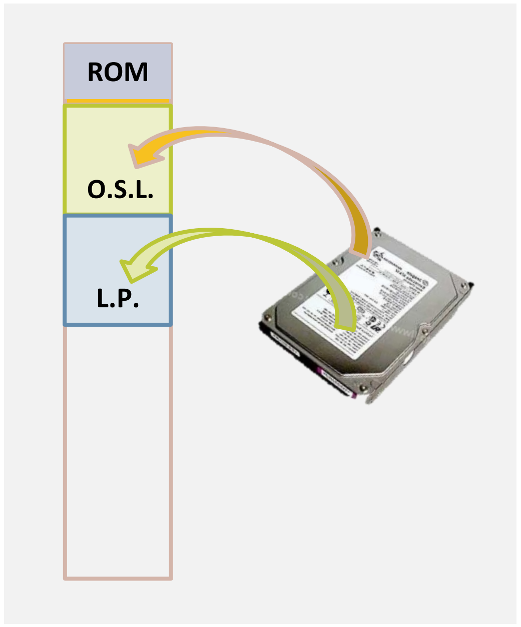

Choose an operating system

Windows 8

Ubuntu Linux


```
GNU GRUB version 0.97  (638K  lower  /  523200K  upper memory)

Ubuntu , kernel 2.6.15-23-386
Ubuntu , kernel 2.6.15-23-386(recovery mode)
Ubuntu , memtest86+
Other operating systems:
Microsoft Windows XP Home Edition

    Use the ↑ and ↓ keys to select which entry is highlighted.
    Press enter to boot the selected OS, 'e' to edit the
    commands before booting, or 'c' for a command-line.
The highlighted entry will be booted automatically in 9 seconds.
```

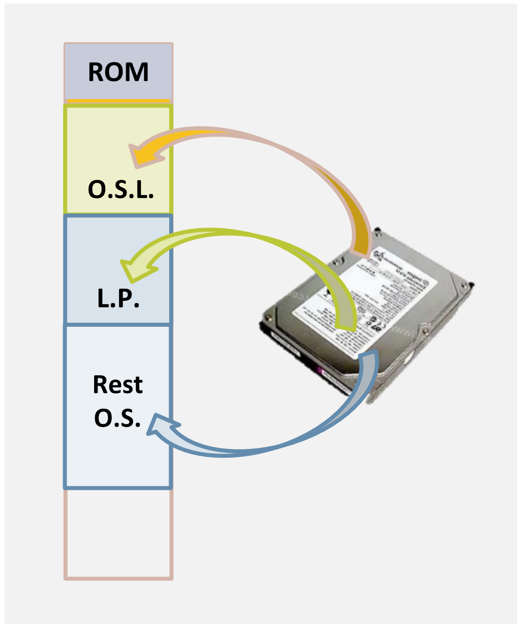
Computer booting

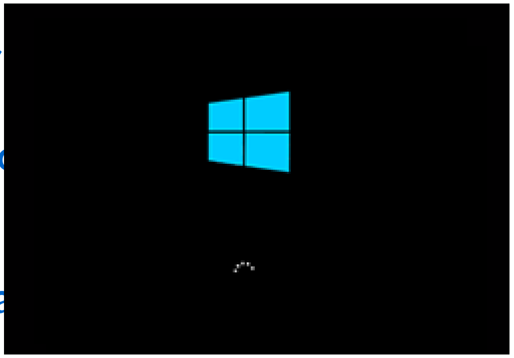


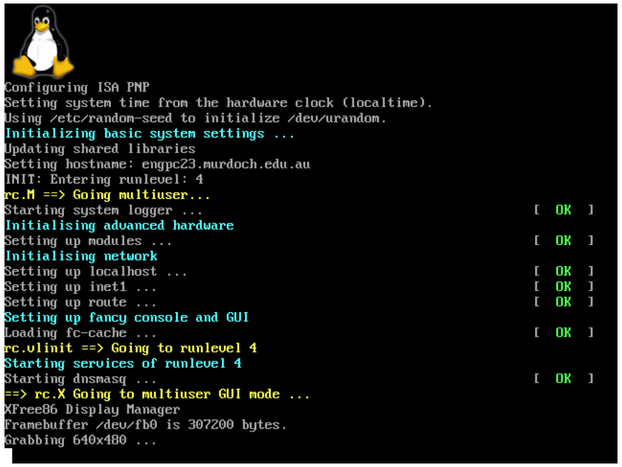
- The *Reset* loads the predefined values in registers:
  - PC ← initial address of the initialization program (in ROM memory)
- The initialization program is executed:
  - System test (POST)
  - Load into memory the operating system loader (MBR)
- The Operating System Loader is executed:
  - Sets boot options
  - Loads the loading program

Computer booting

- The *Reset* loads the predefined values in registers:
  - PC ← initial address of the initialization program (in ROM memory)
- The initialization program is executed:
  - System test (POST)
  - Load into memory the operating system loader
- The Operating System Loa
  - Sets boot options
  - Loads the loading progra
- The Loading Program is executed:
  - Sets the initial state of the O.S.
  - Loads the O.S. and executed it.



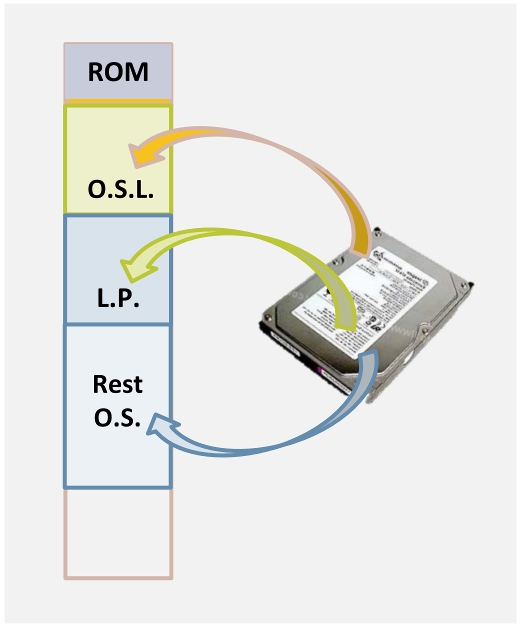




```
Configuring ISA PNP
Setting system time from the hardware clock (localtime).
Using /etc/random-seed to initialize /dev/urandom.
Initializing basic system settings ...
Updating shared libraries
Setting hostname: engpc23.murdoch.edu.au
INIT: Entering runlevel: 4
rc.M ==> Going multiuser...
Starting system logger ...                                          [  OK  ]
Initialising advanced hardware
Setting up modules ...                                              [  OK  ]
Initialising network
Setting up localhost ...                                            [  OK  ]
Setting up inet1 ...                                                [  OK  ]
Setting up route ...                                                [  OK  ]
Setting up fancy console and GUI
Loading fc-cache ...                                                [  OK  ]
rc.vlinit ==> Going to runlevel 4
Starting services of runlevel 4
Starting dnsmasq ...                                                [  OK  ]
==> rc.X Going to multiuser GUI mode ...
XFree86 Display Manager
Framebuffer /dev/fb0 is 307200 bytes.
Grabbing 640x480 ...
```

Computer booting
summary

- The *Reset* loads the predefined values in registers:
  - PC ← initial address of the initialization program (in ROM memory)
- The initialization program is executed:
  - System test (POST)
  - Load into memory the operating system loader (MBR)
- The Operating System Loader is executed:
  - Sets boot options
  - Loads the loading program
- The Loading Program is executed:
  - Sets the initial state of the O.S.
  - Loads the O.S. and executed it.



Contenido

Program execution time

Iron law of processor performance

$$\boxed{Time_{execution} = IN \times CPI \times t_{cycle\_CPU} + IN \times AMI \times t_{cycle\_mem}}$$

IN	is the number of instructions of the program
CPI	is the average number of clock cycles to execute an instruction
t_{cycle_CPI}	is the cycle clock duration
AMI	is the average number of memory access per instruction
t_{cycle_mem}	is the time needed for a memory access

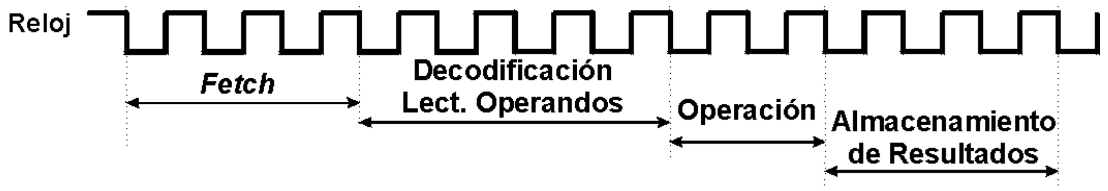
Factors affecting execution time

	NI	CPI	t_{cycle_CPI}	AMI	t_{cycle_mem}
Program	✓			✓	
Compiler	✓	✓		✓	
Instruction set	✓	✓	✓	✓	
Organization		✓	✓		✓
Technology			✓		✓

Instruction level parallelism

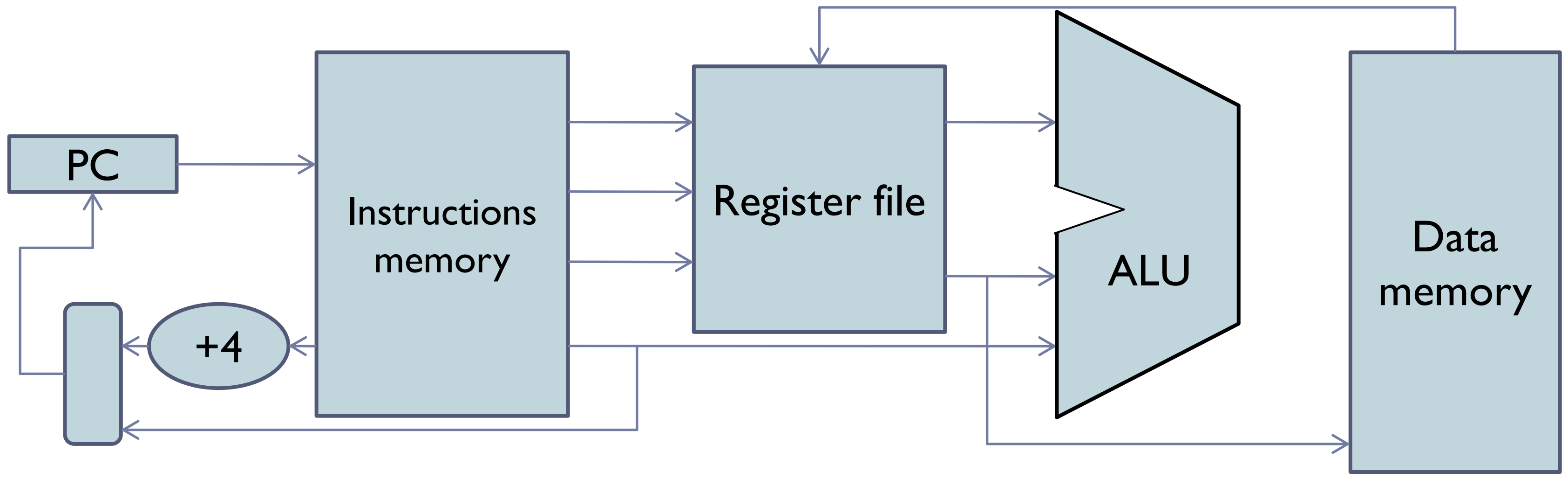
- Concurrent execution of several machine instructions
- Combination of elements working in parallel:
  - Pipelined processor: use pipelines in which multiple instructions are overlapped in execution
  - Superscalar processor: multiple independent instruction pipelines are used. Each pipeline can handle multiple instructions at a time
  - Multicore processor: several processors or cores in the same chip

Segmentation of instructions

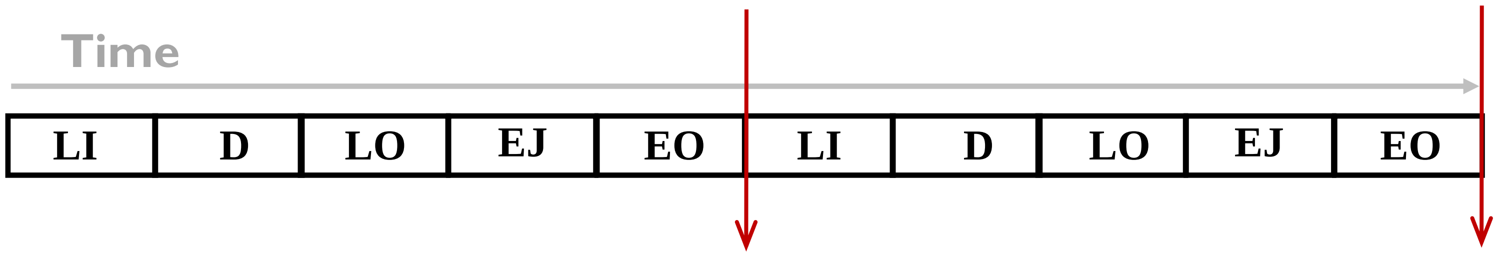


- Stages in the execution of instructions:
  - IF: Instruction fetch
  - D: Decoding
  - RO: Read operands
  - EX: Execution
  - WO: Write operands

Model of processor based on datapath (without internal bus)

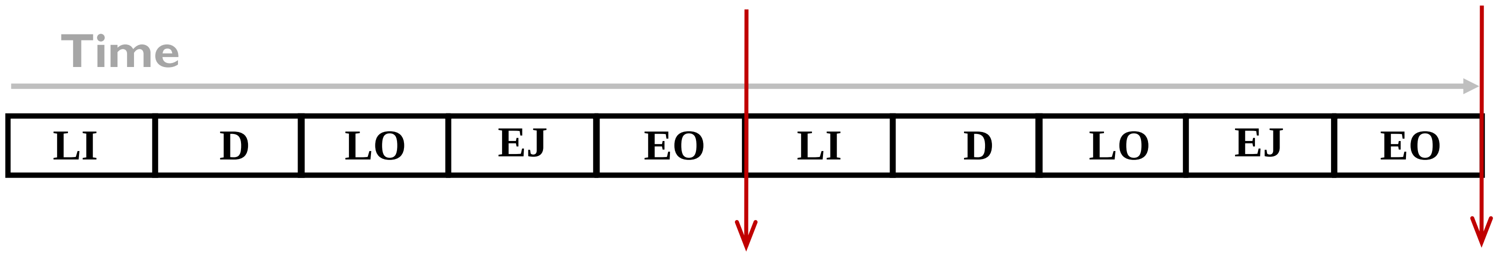


Segmentation of instructions
without pipeline



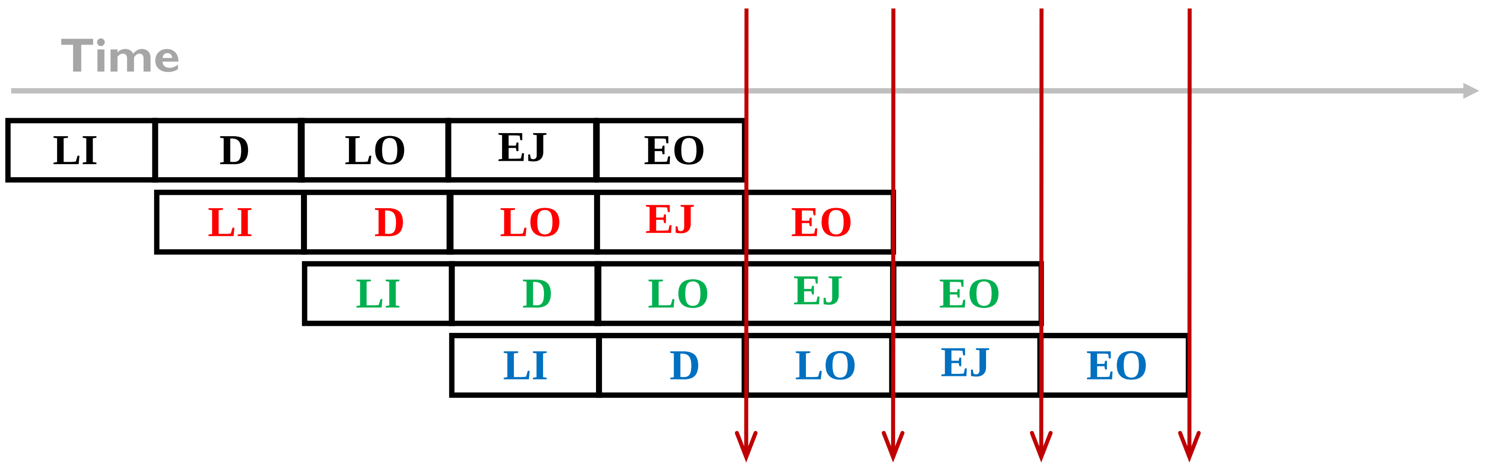
- Stages in the execution of instructions:
  - IF: Instruction fetch
  - D: Decoding
  - RO: Read operands
  - EX: Execution
  - WO: Write operands

Segmentation of instructions
without pipeline



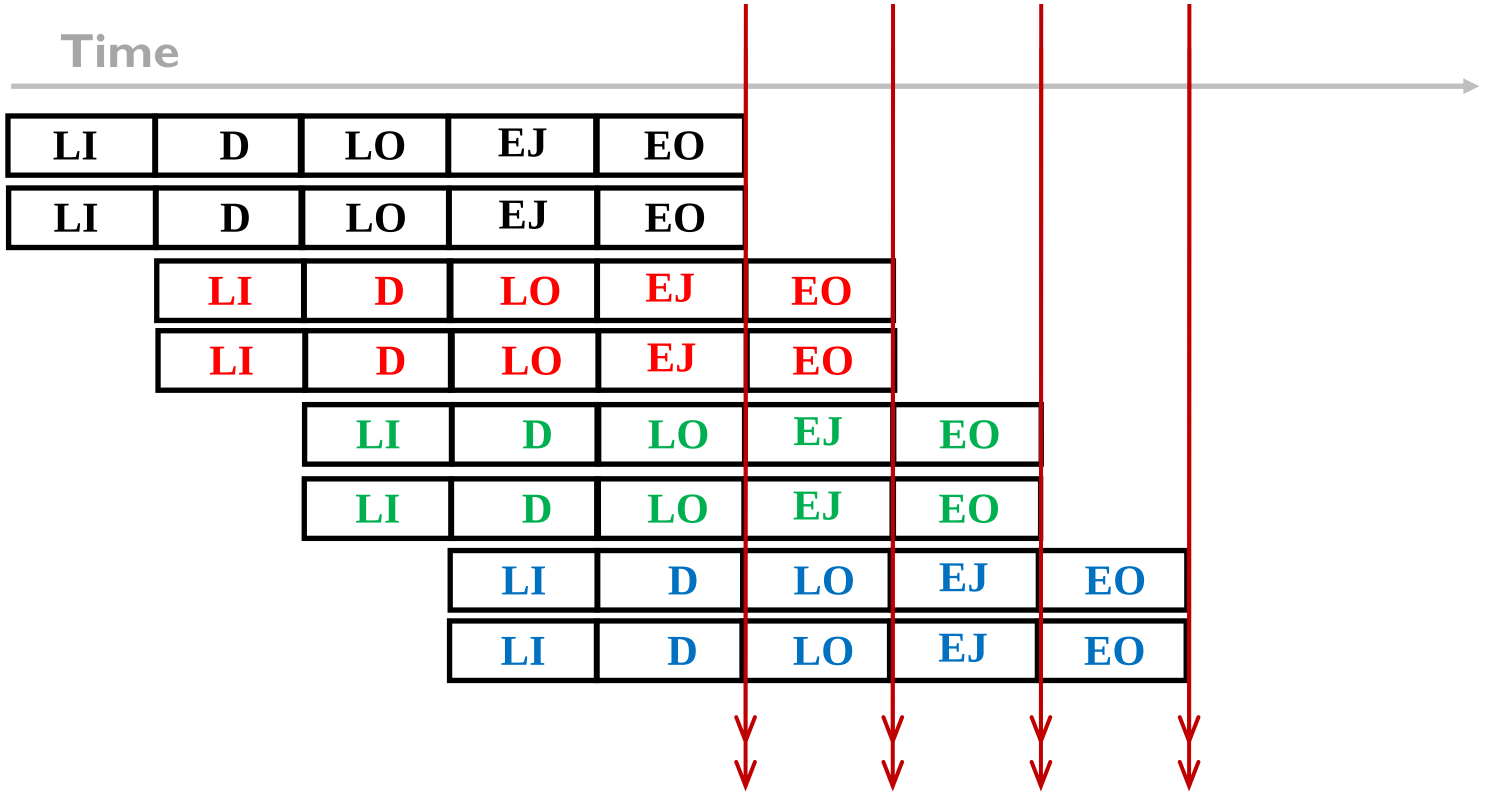
- If each phase takes N clock cycles, then:
  - One instruction takes 5*N clock cycles to be executed
  - 1/5 of instruction is issued every N clock cycles

Segmentation of instructions
with pipeline



- If each phase takes N clock cycles, then:
  - One instruction takes 5*N clock cycles to be executed
  - One instruction is issued every N clock cycles

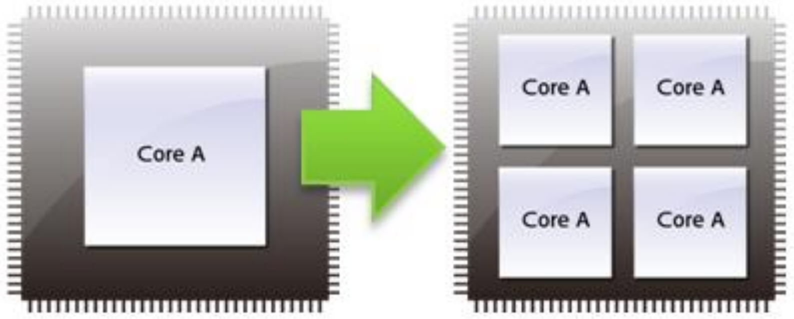
Superescalar

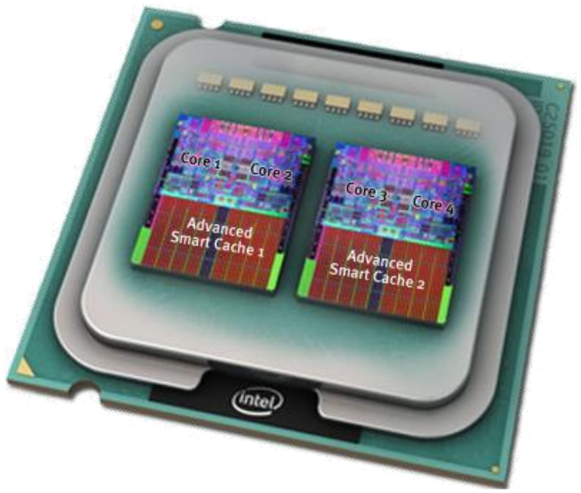


- Pipeline with several functional units in parallel

Multicore

- Multiples processors in the same chip





ARCOS Group

uc3m | Universidad Carlos III de Madrid

L4: The processor (2/2)
Computer Structure

Bachelor in Computer Science and Engineering
Bachelor in Applied Mathematics and Computing
Dual Bachelor in Computer Science and Engineering and Business Administration