

ARCOS Group

**uc3m** | Universidad **Carlos III** de Madrid

## Lesson 4 (II) The processor

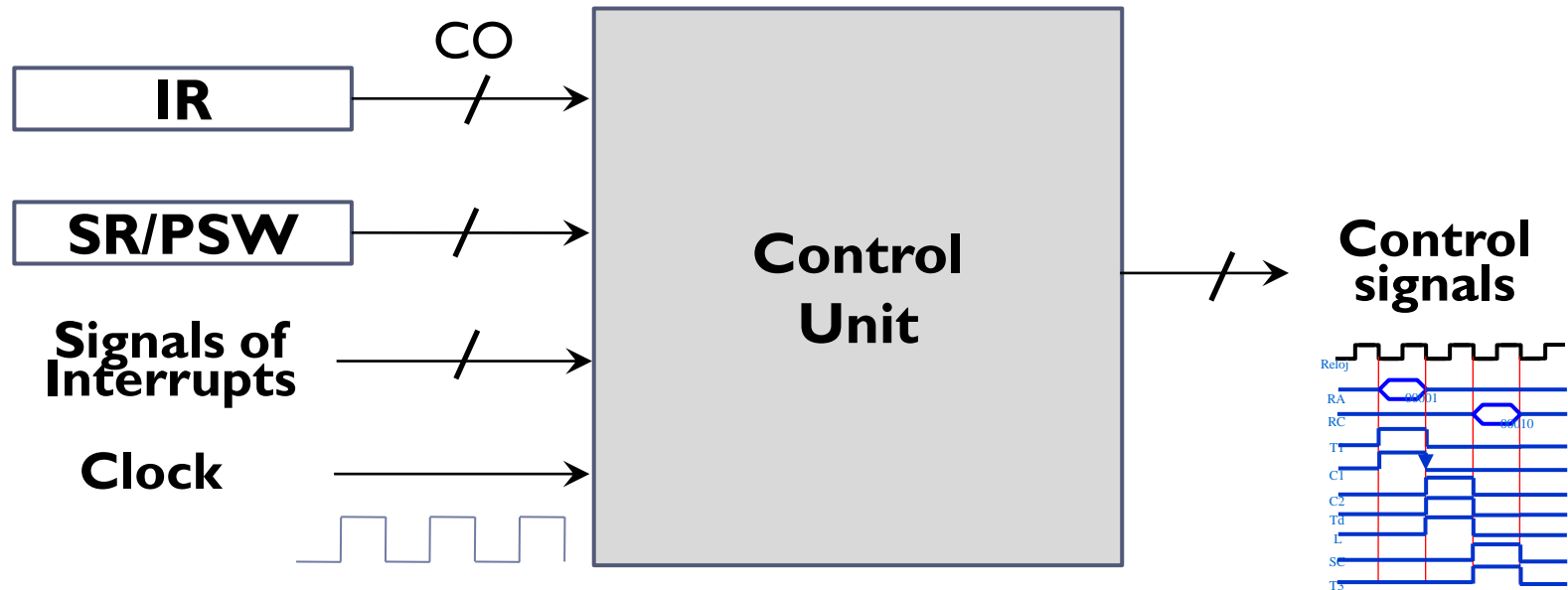
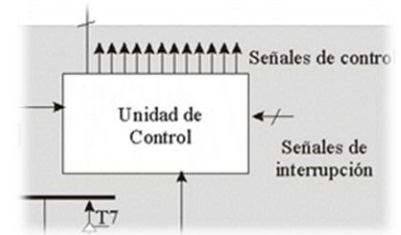
Computer Structure  
Bachelor in Computer Science and Engineering



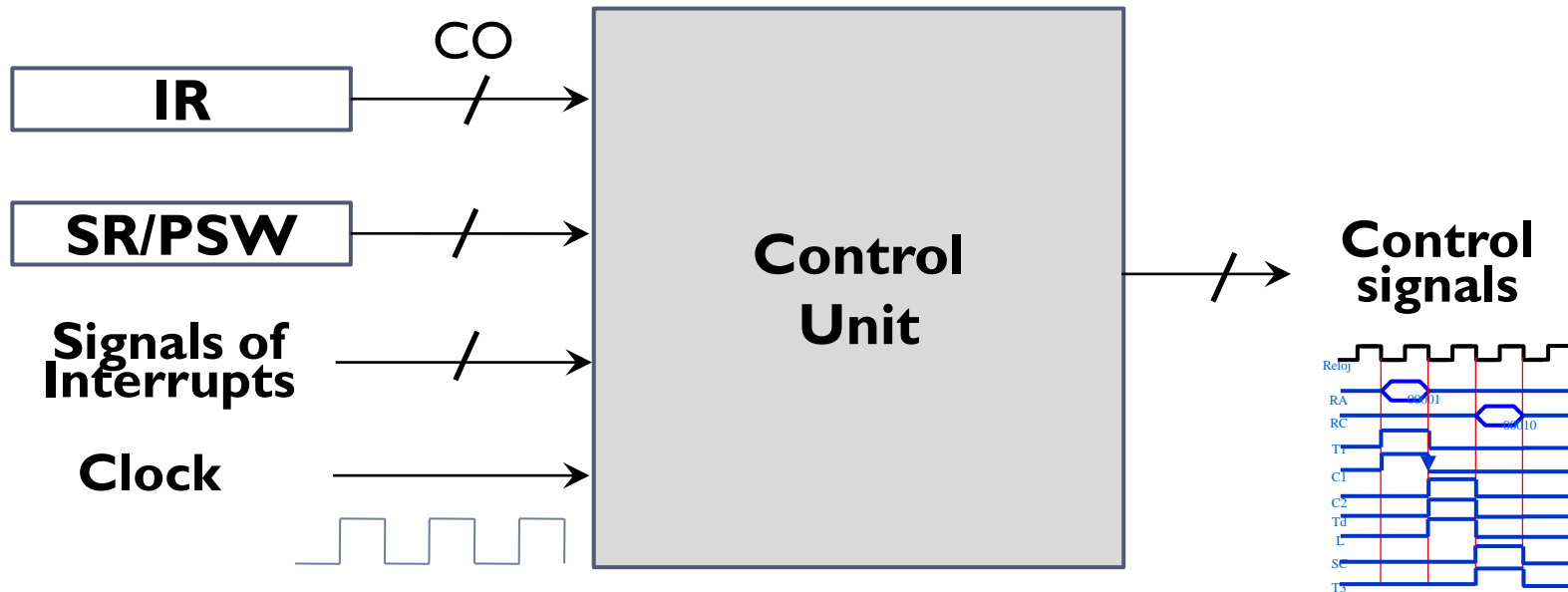
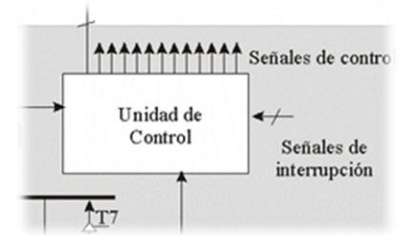
# Contents

1. Computer elements
2. Processor organization
3. Control unit
4. Execution of instructions
5. Control unit design
6. Execution modes
7. Interrupts
8. Computer startup
9. Performance and parallelism

# Control Unit



# 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

# Control unit design

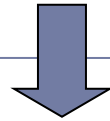
## ► For each machine instruction:

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

# Control unit design

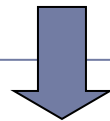
Instruction

*mv R0 R1*

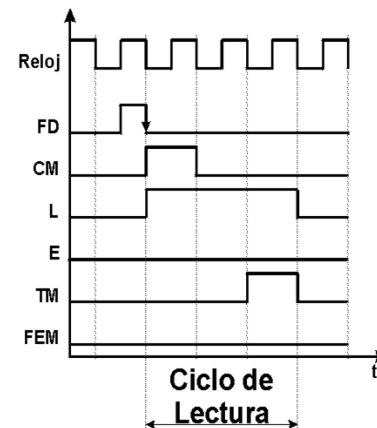


Sequence of **elementary operations**

- $RI \leftarrow [PC]$
- $PC++$
- decoding
- $R0 \leftarrow R1$



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



# Control unit design

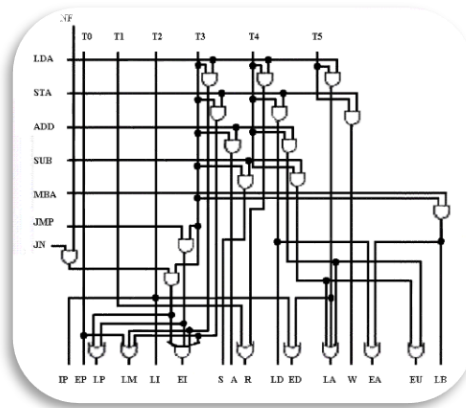
► For each machine instruction:

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

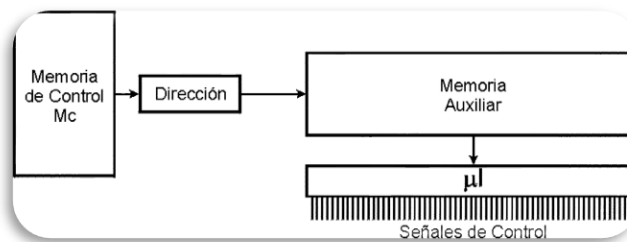


# Control techniques

## ► Hardwired (relay logic) control unit



## ► Microprogrammed control unit



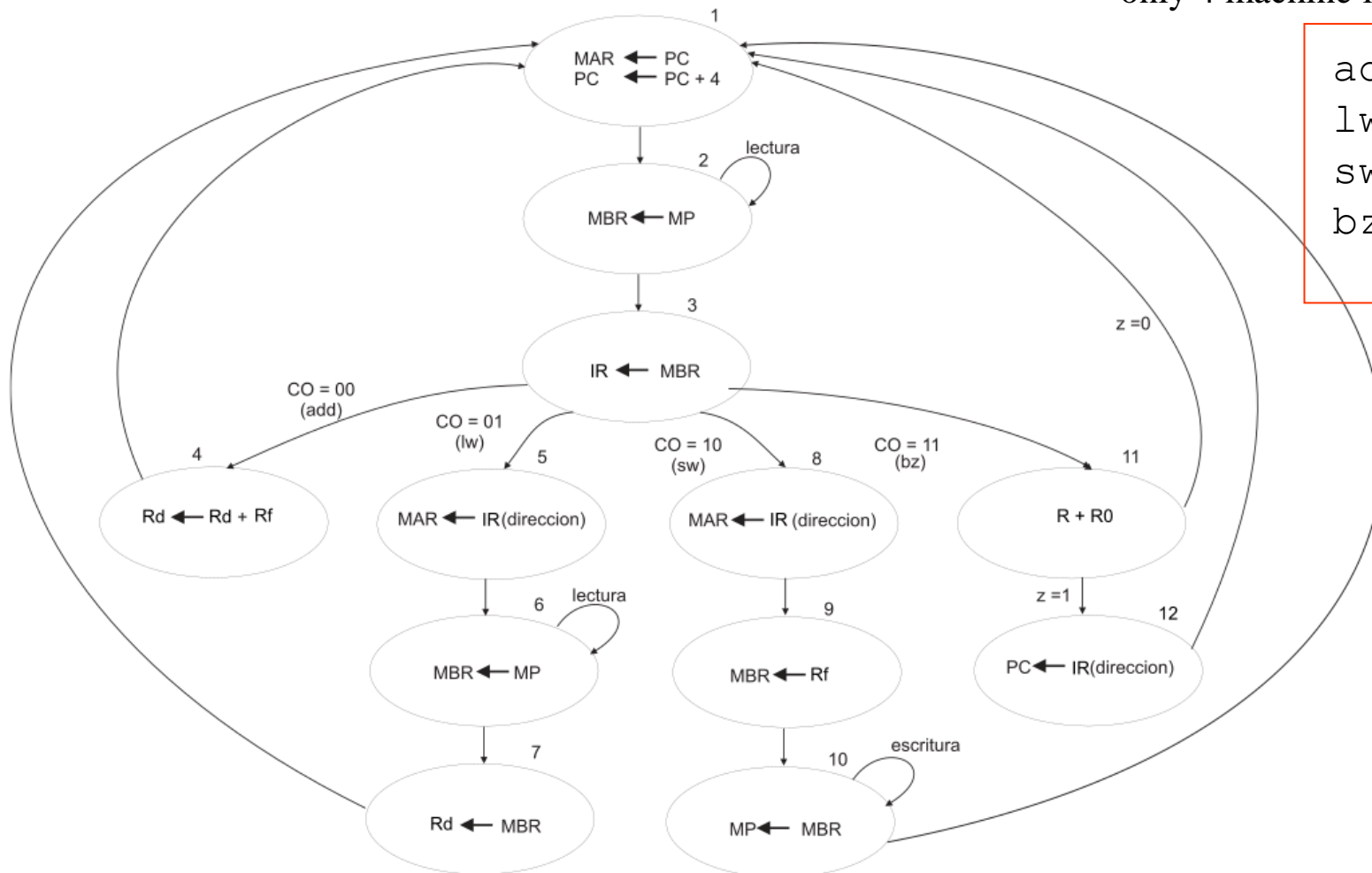
# Example

- ▶ 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`

# State machine for the example

Example for a computer with only 4 machine instructions

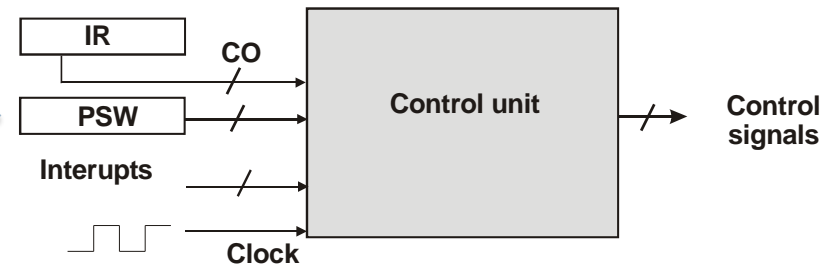
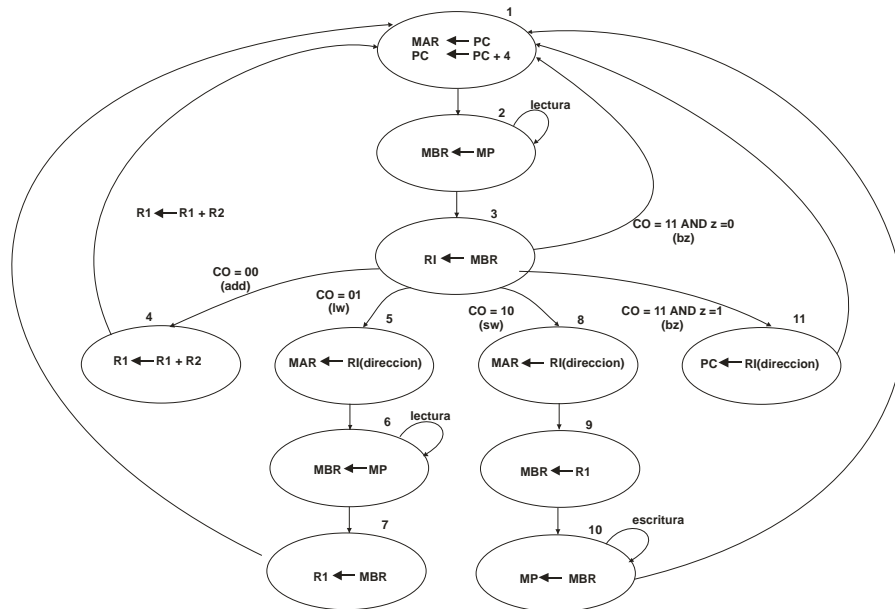
```
add rd, rf
lw rd, dir
sw Rf, dir
bz R, dir
```



# Control techniques

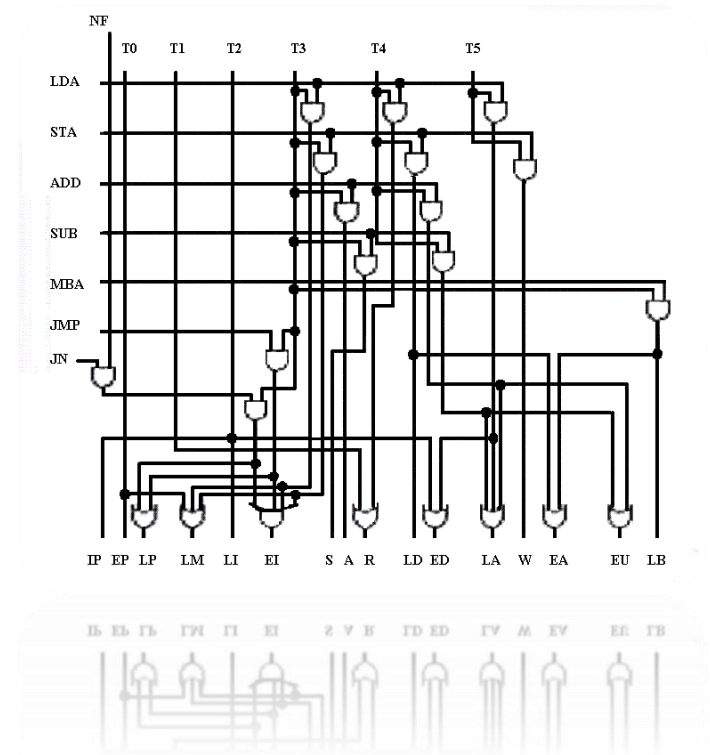
## ► Two techniques to design and build the control unit:

- a) Relay logic
- b) Programmable logic (microprogrammed)



# Control Unit: relay logic

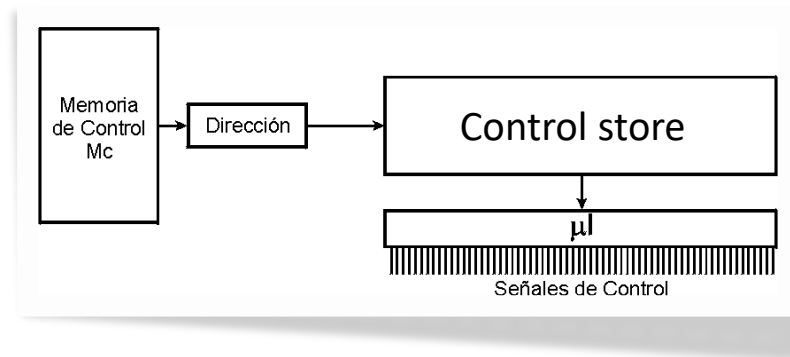
- ▶ Construction by means of logic gates, following logic design methods.
- ▶ Characteristics:
  - ▶ Laborious and costly circuit design and tuning.
  - ▶ Difficult to modify:
    - ▶ Complete redesign.
  - ▶ Very fast (used in RISC computers).



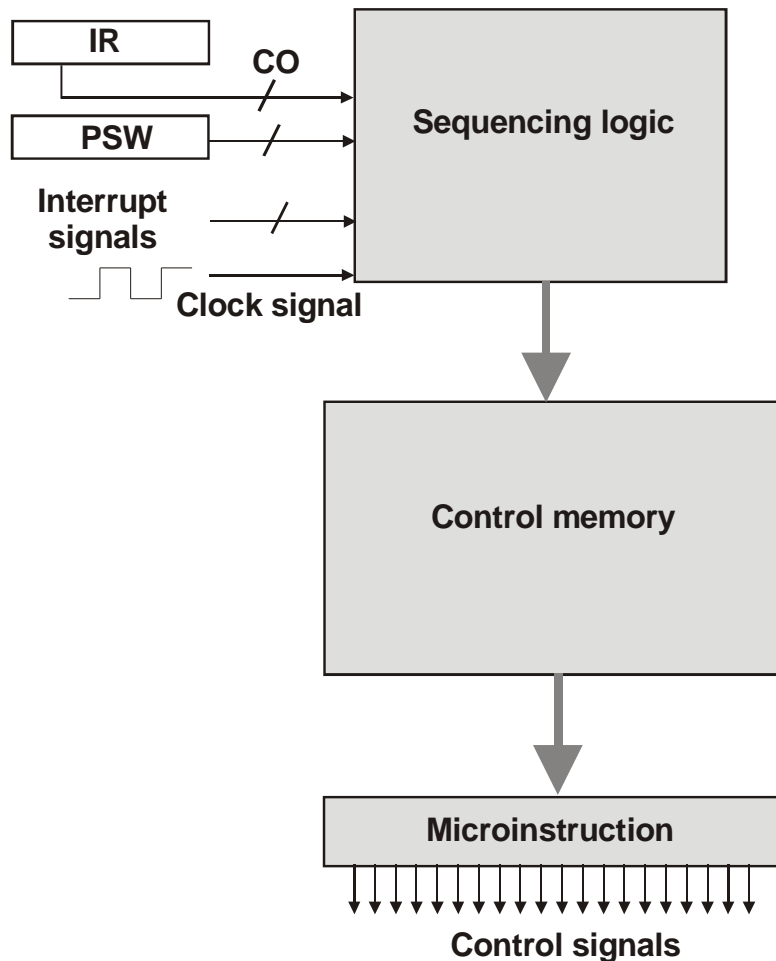
# Control Unit: programmable logic

## 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  $\Rightarrow$  hard microcode



# General structure of a microprogrammed control unit



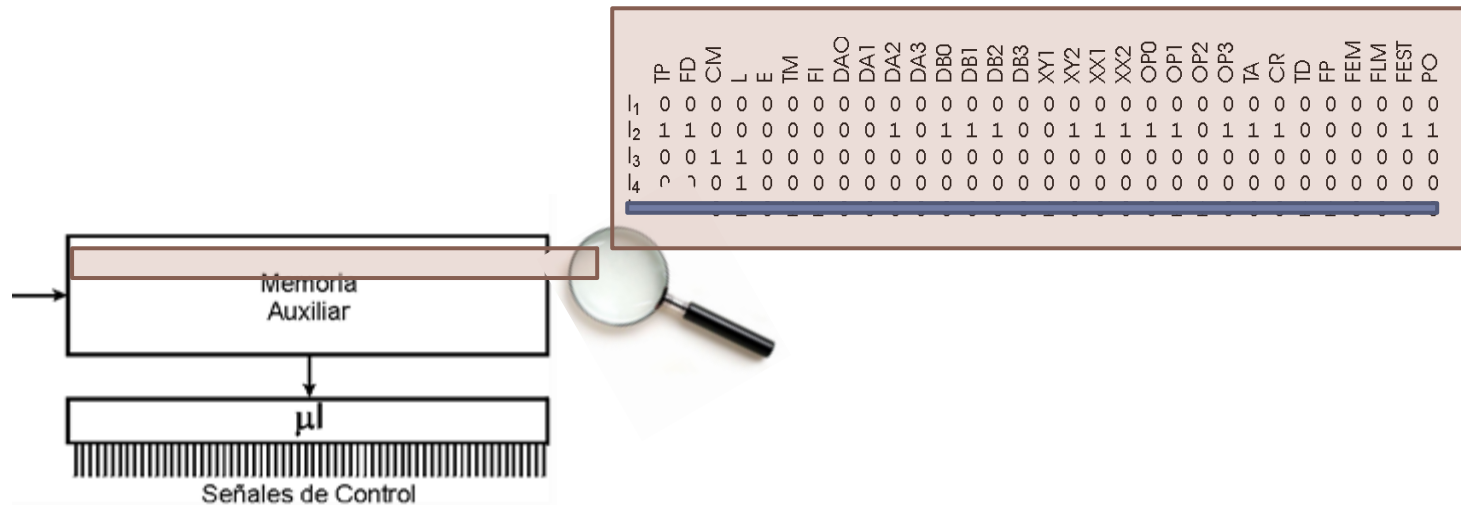
C1	C2	C3	C4	R5	C6	C7	C8	C9	C10	C11	Td	Ta	T1	T2	T3	T4	T5	T6	T7	T8	RA4	RA3	RA2	Ra1
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

RB4	Rb3	RB2	RB1	RB0	RC4	RC3	RC2	RC1	RC0	SC	L	E	Cop3	Cop2	Cop1	Cop0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

# Microprogrammed control unit.

## Microinstructions

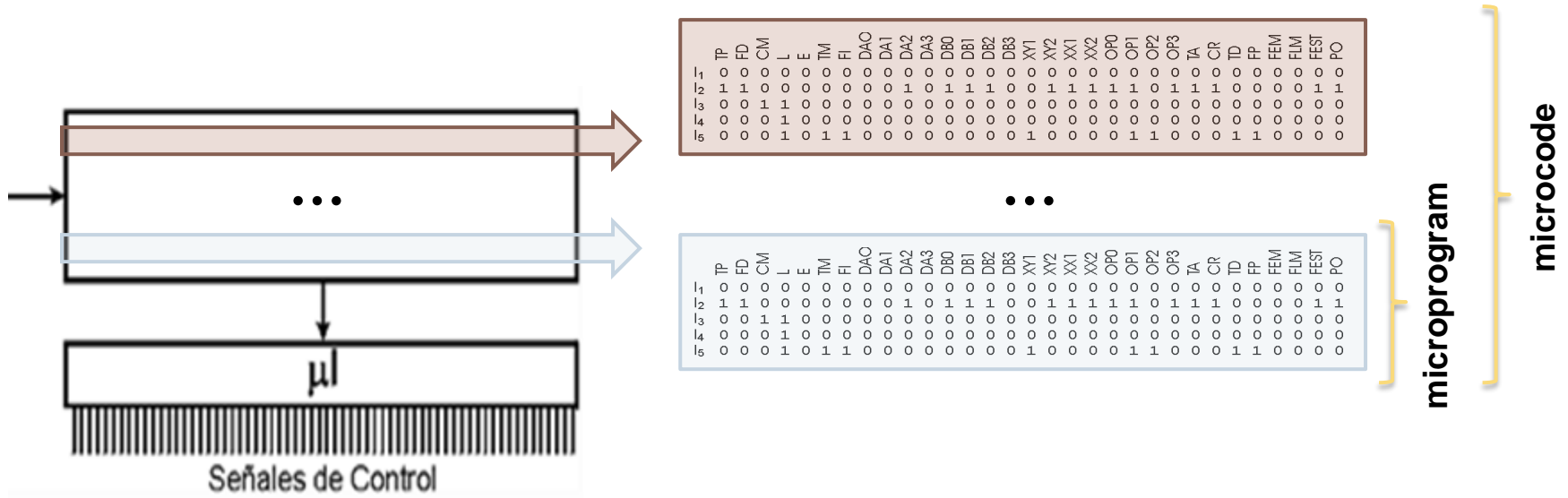


- ▶ **Microinstruction:** To each word defining the value of each control signal in a cycle of an instruction/fetch+IAC
- ▶ **The microinstructions**
  - ▶ have one bit for each control signal.
  - ▶ A list of 1's and 0's representing the state of each control signal during a period of one instruction.



# Microprogrammed control unit.

## microprogram and microcode



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

# Contents of the control memory



- ▶ **FETCH**: get next instruction
  - ▶ IAC: interrupt acknowledge cycle.
  - ▶  $IR \leftarrow \text{Mem}[\text{PC}], \text{PC}++, \text{jump-to-O.C.}$
- ▶ **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

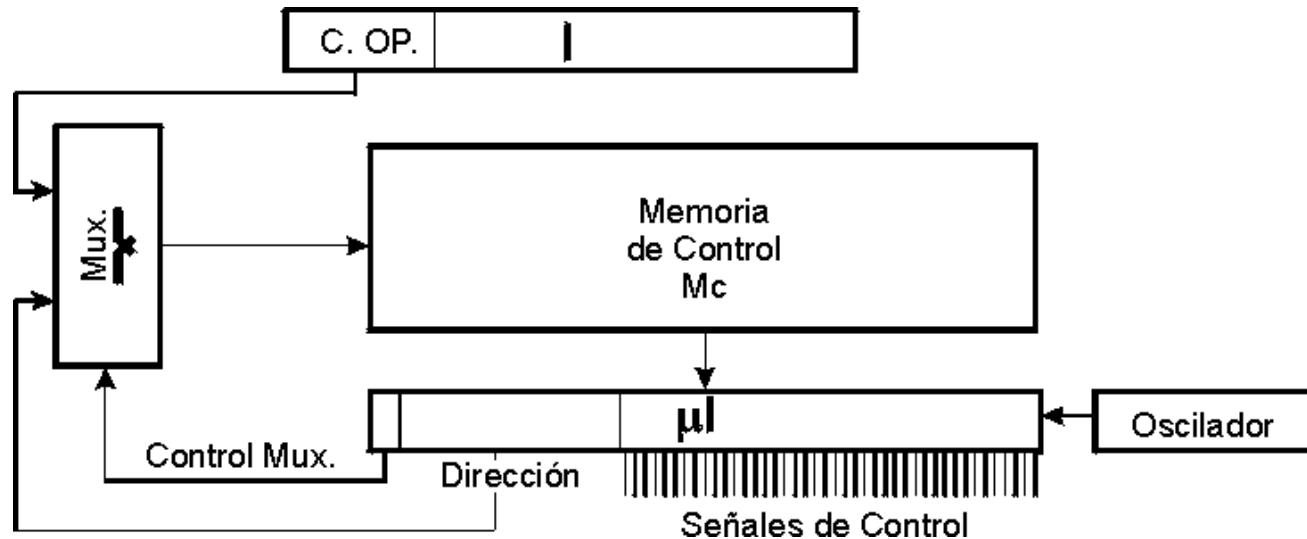
## ▶ Three basic conditions:

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:

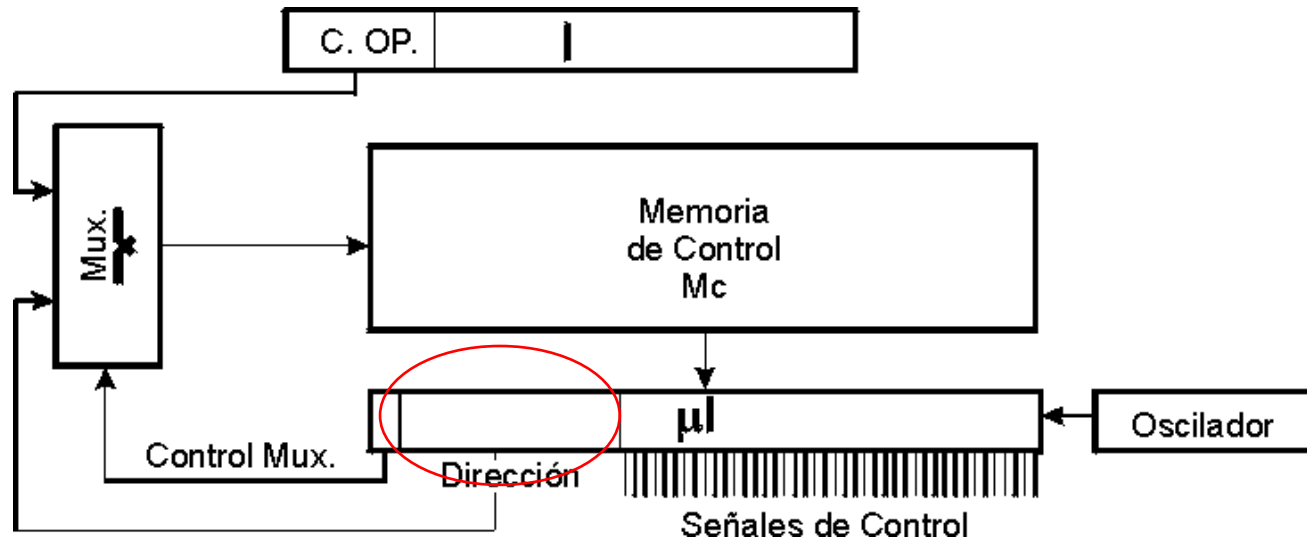
1. Explicit sequencing.
2. Implicit sequencing.

# Microprogrammed C.U. structure with explicit sequencing



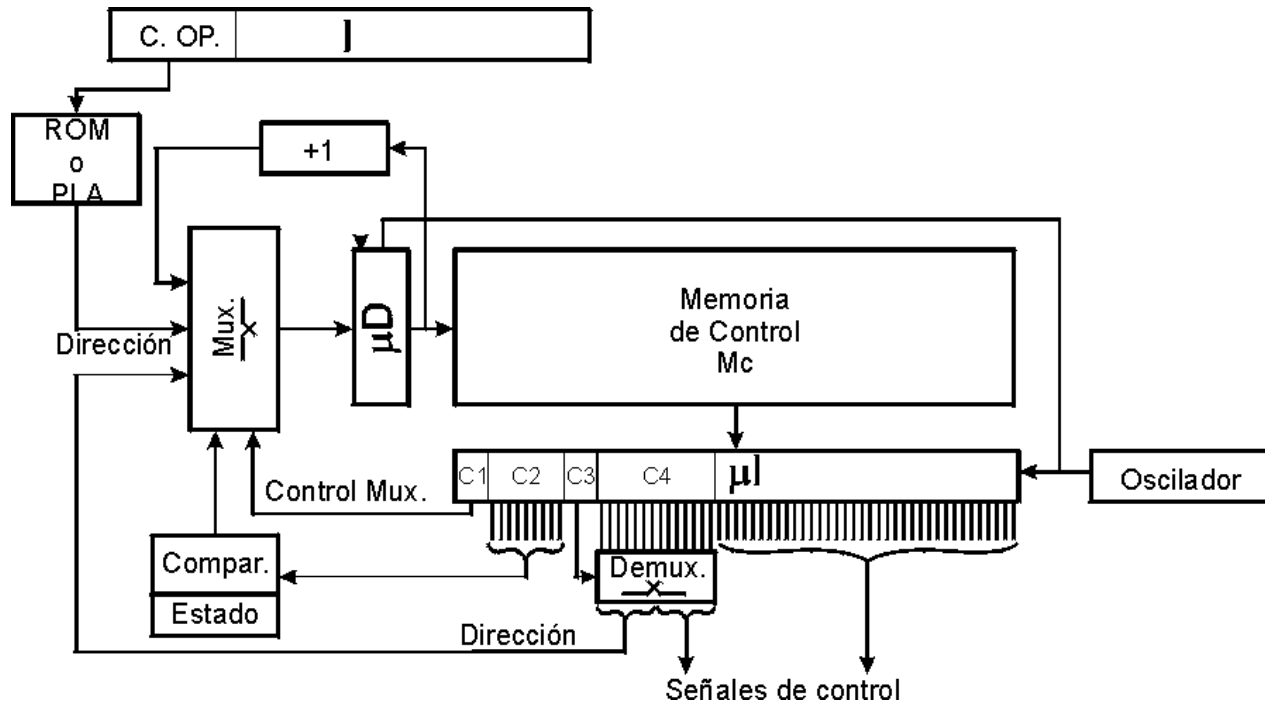
- ▶ Control memory stores all  $\mu$ programs, where each  $\mu$ instruction provides the next  $\mu$ instruction  $\mu$ address
- ▶ The OC represents the  $\mu$ Address of the first  $\mu$ instruction associated with the machine instruction.

# Microprogrammed C.U. structure with explicit sequencing



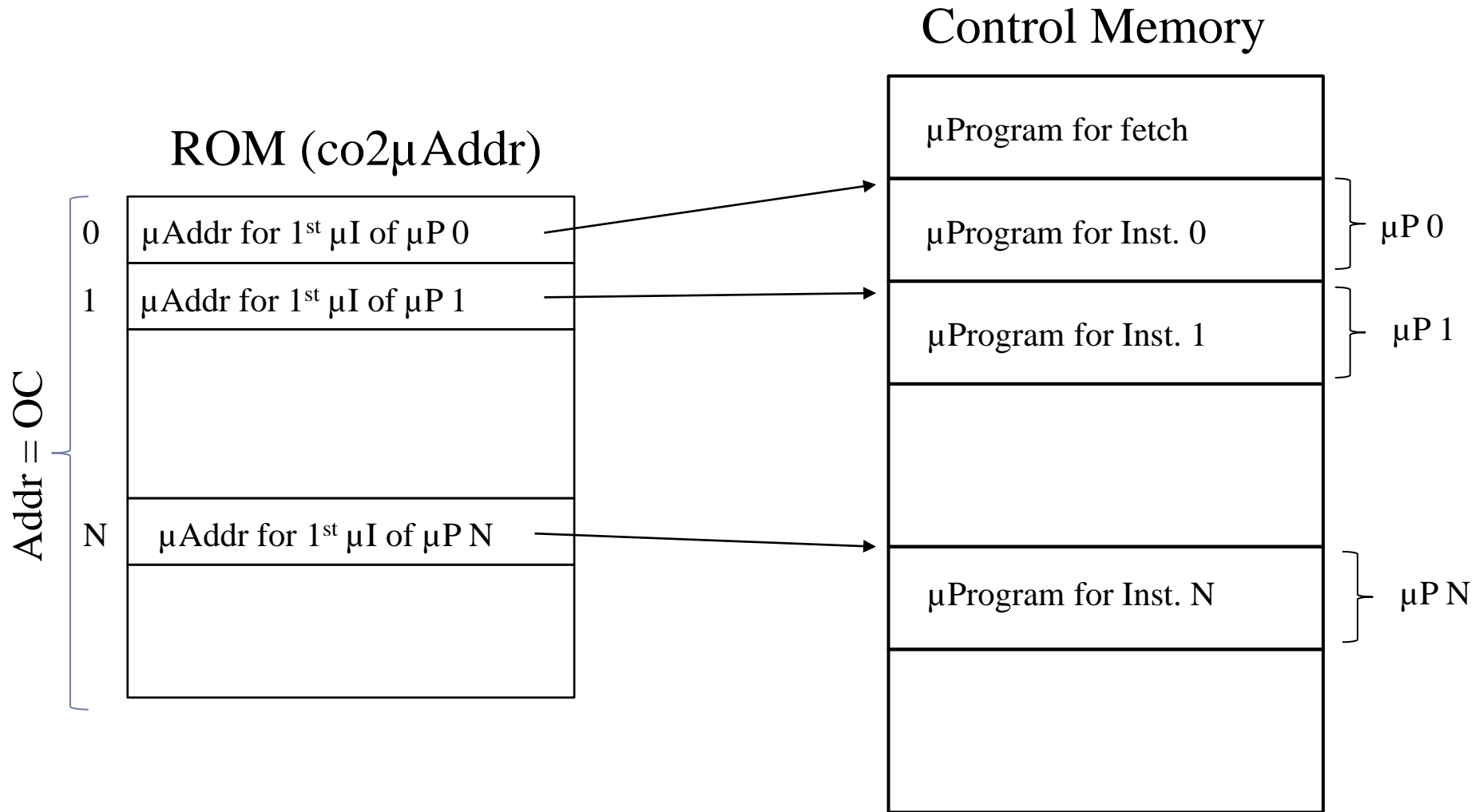
- ▶ Control memory stores all  $\mu$ programs, where each  $\mu$ instruction provides the next  $\mu$ instruction  $\mu$ address
- ▶ **Problem:** large amount of control memory for instruction sequencing, required stores the next  $\mu$ address

# Microprogrammed C.U. structure with implicit sequencing

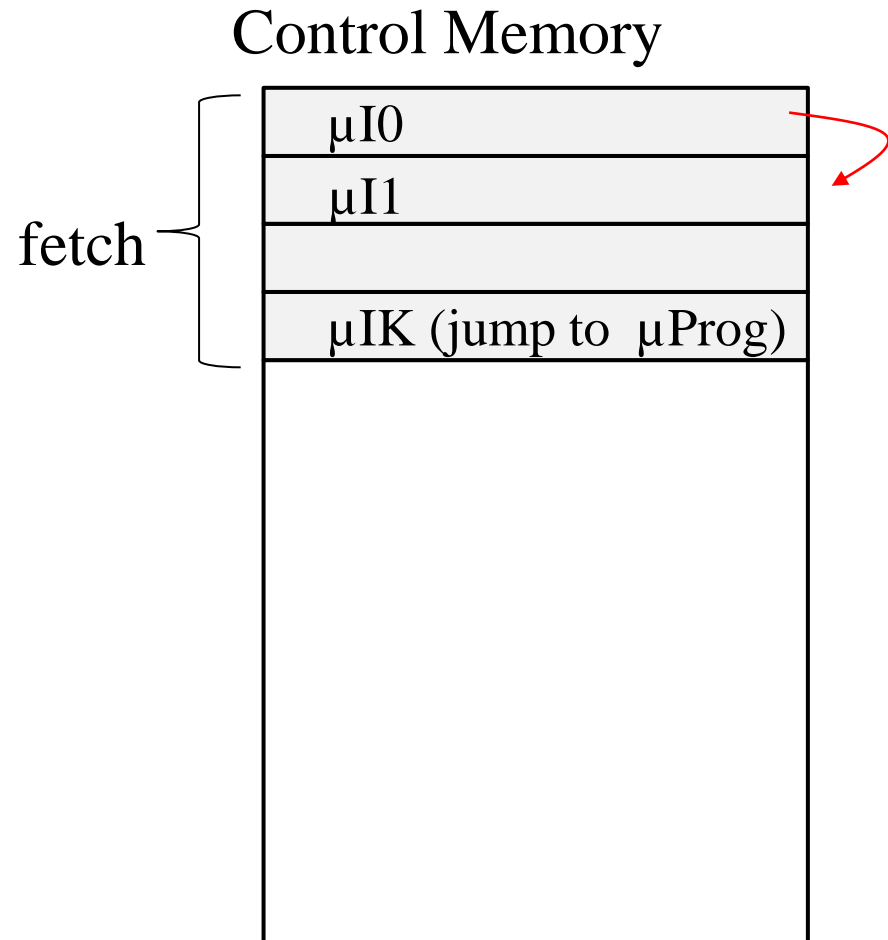
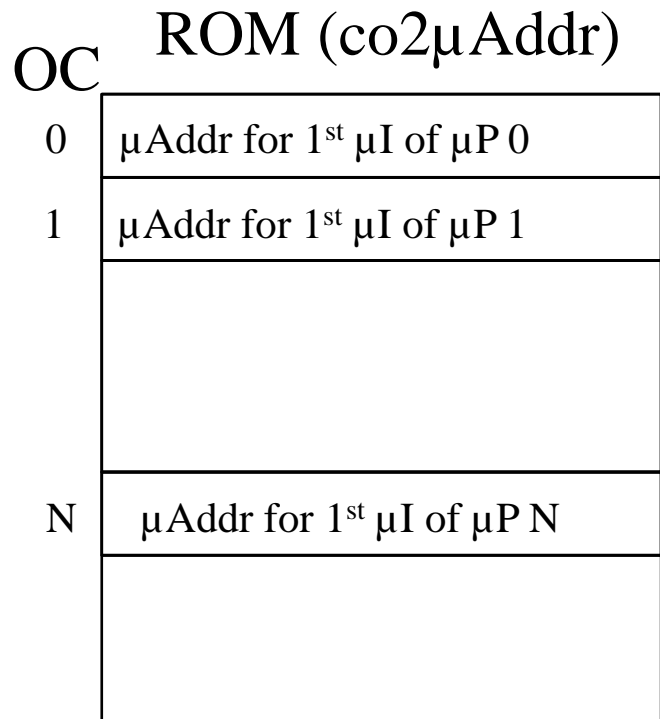


- ▶ Control memory stores all microprograms consecutively in the control memory.
- ▶ The ROM/PLA associates each instruction with its microprogram (first  $\mu$ address,  $\mu$ conditional  $\mu$ instruction (+1),  $\mu$ conditional  $\mu$ bifurcations or  $\mu$ loops).
- ▶ Next  $\mu$ instruction (+1), conditional  $\mu$ bifurcations or  $\mu$ loops

# Example of Control Unit with implicit sequencing

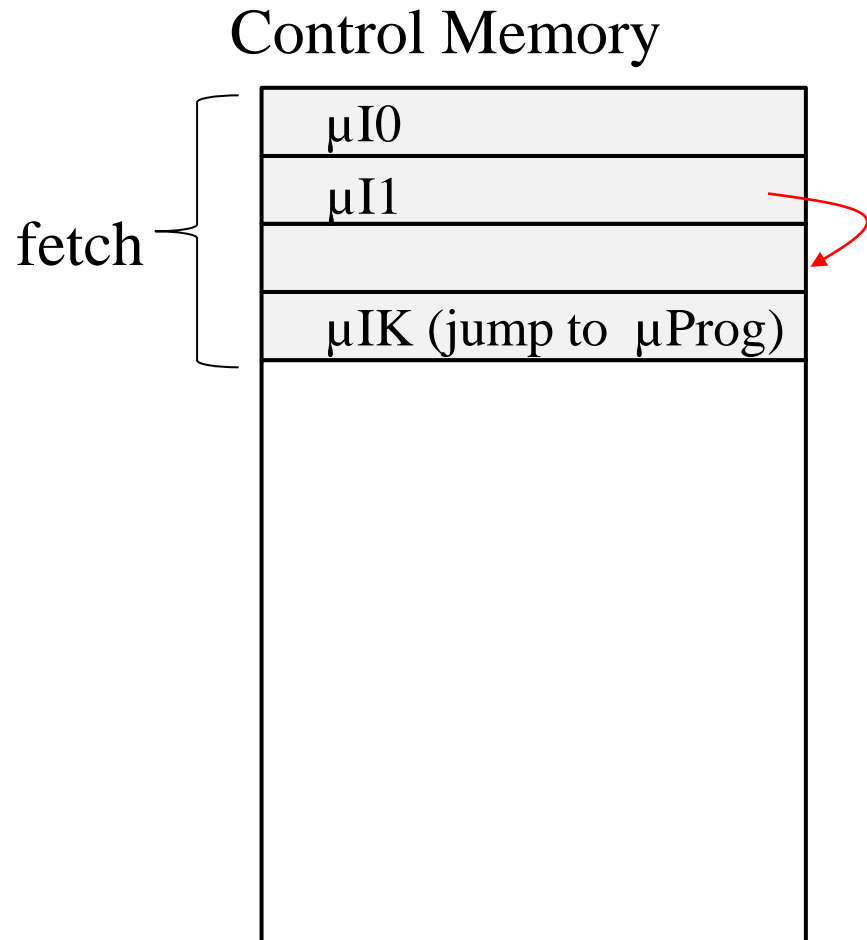
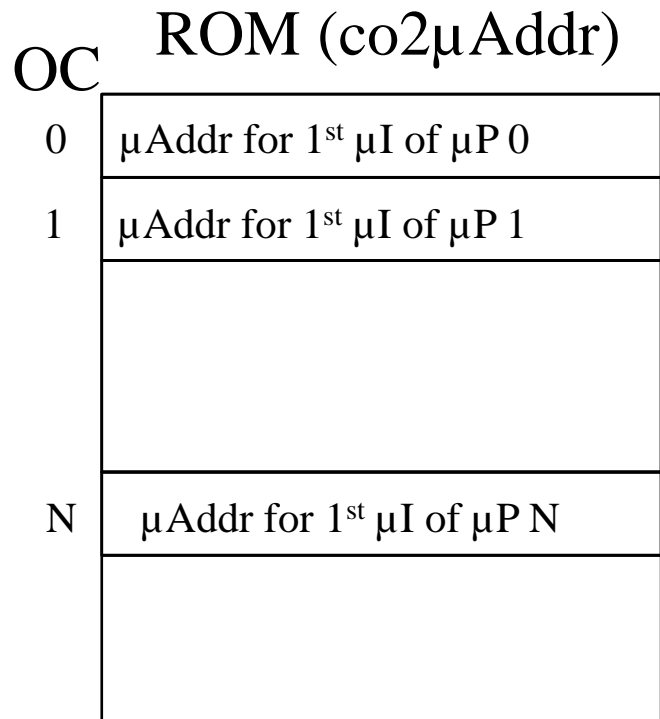


## Example of Control Unit with implicit sequencing

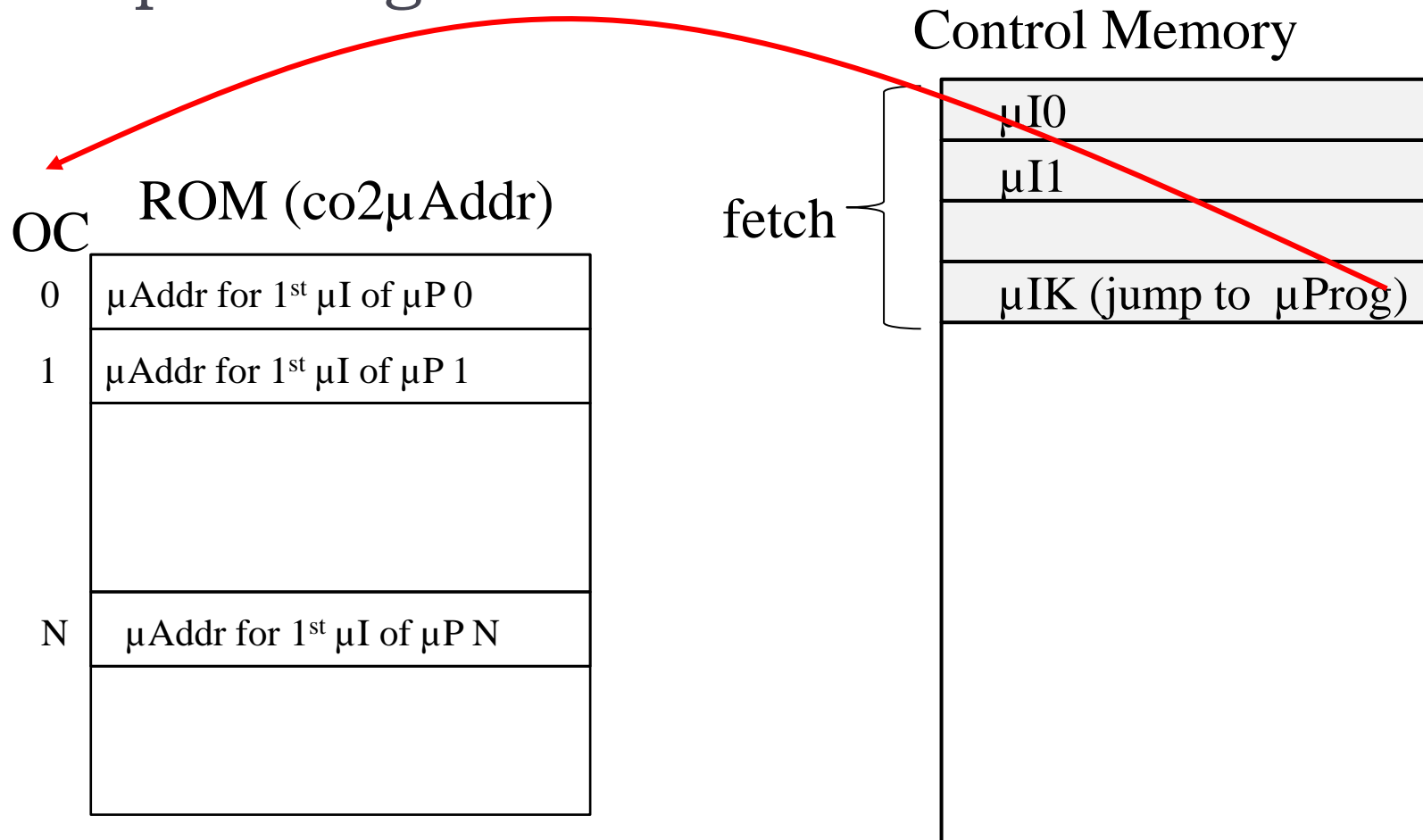




## Example of Control Unit with implicit sequencing

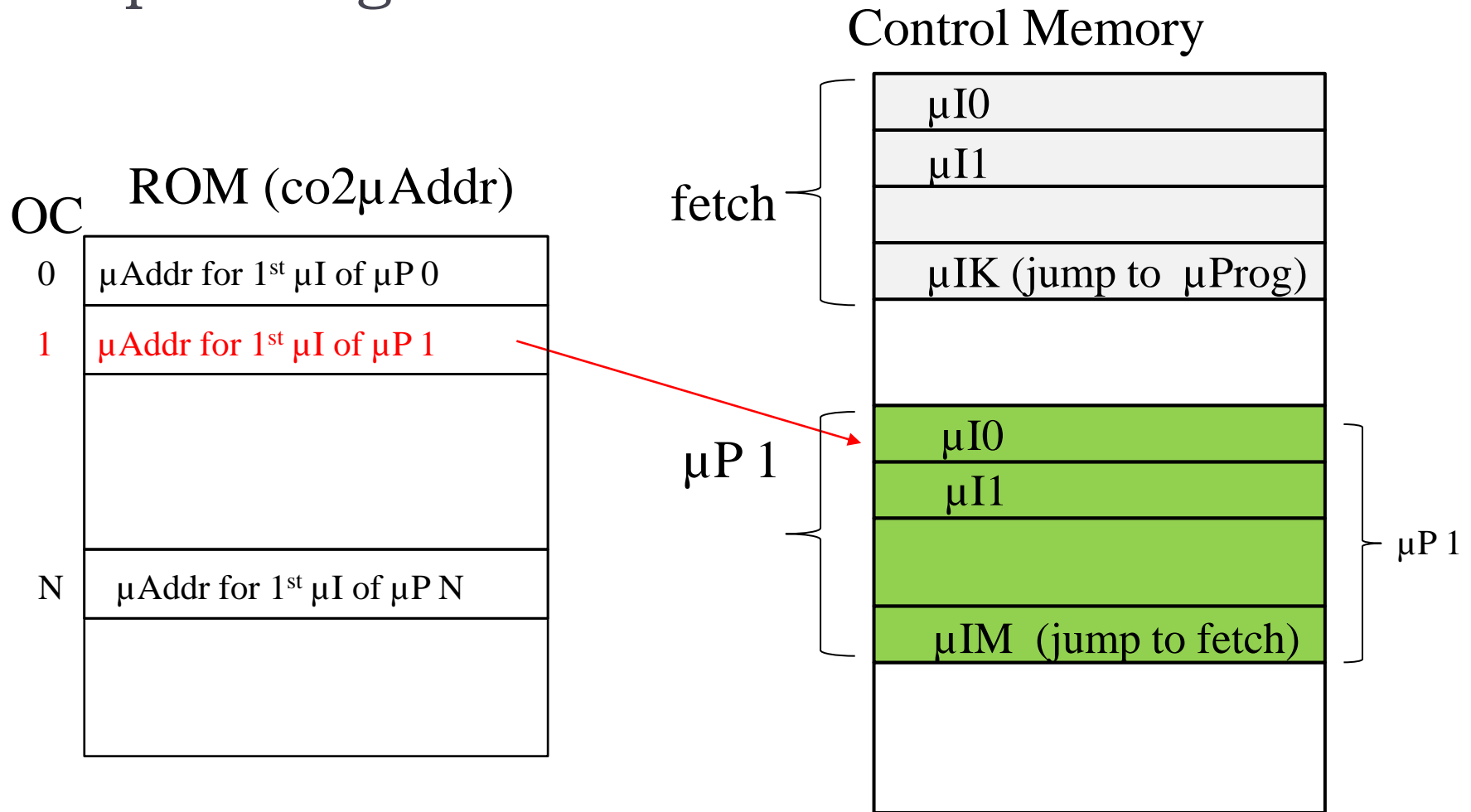


# Example of Control Unit with implicit sequencing

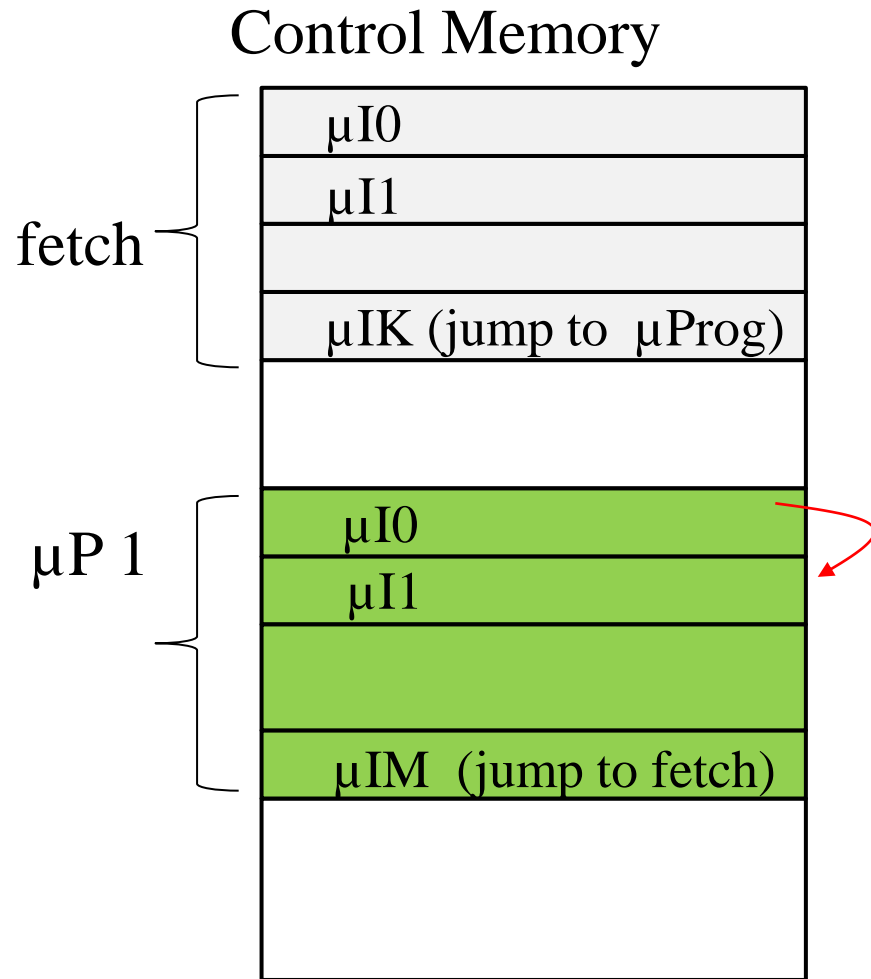
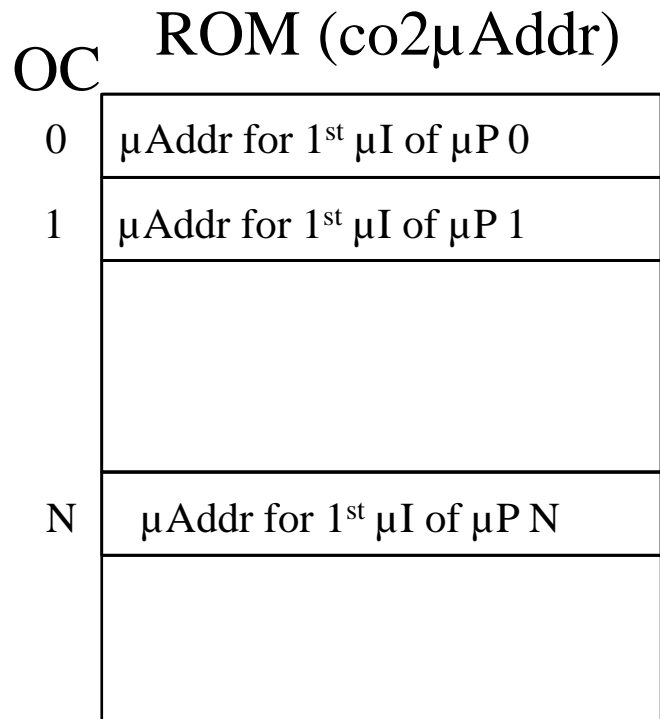


The Operation Code (OC) is at the Instruction Register (IR)

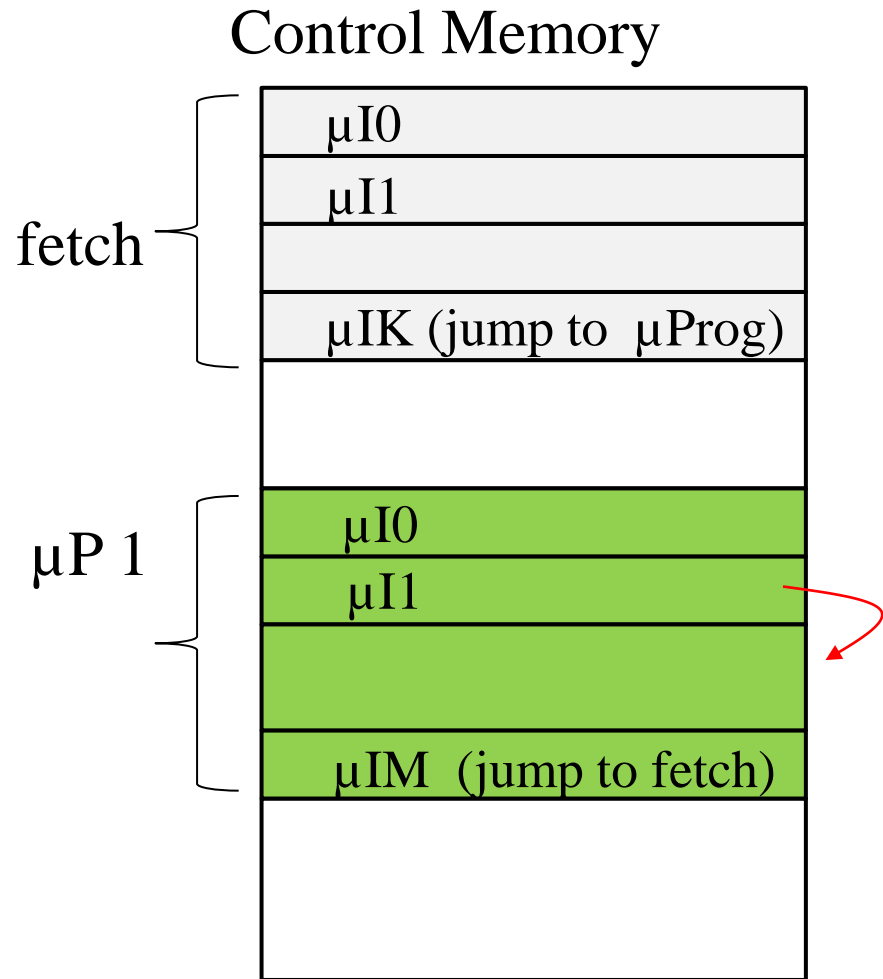
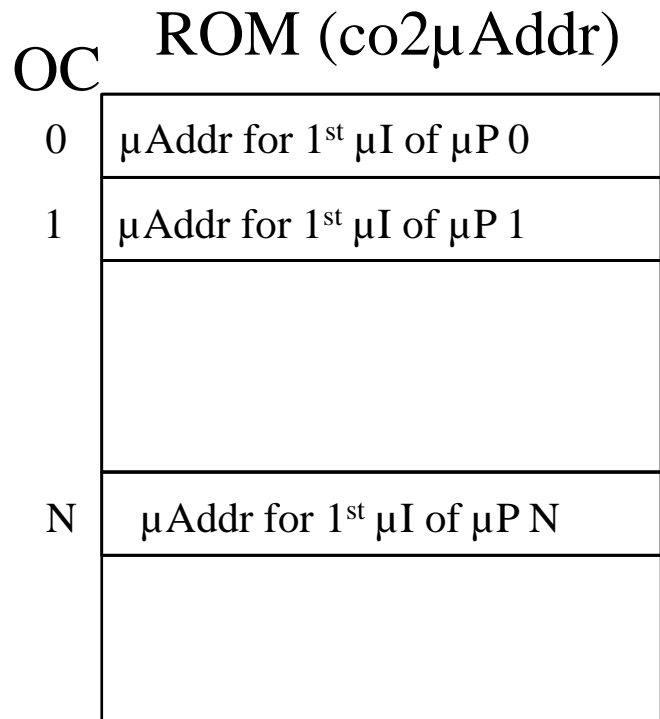
# Example of Control Unit with implicit sequencing



# Example of Control Unit with implicit sequencing

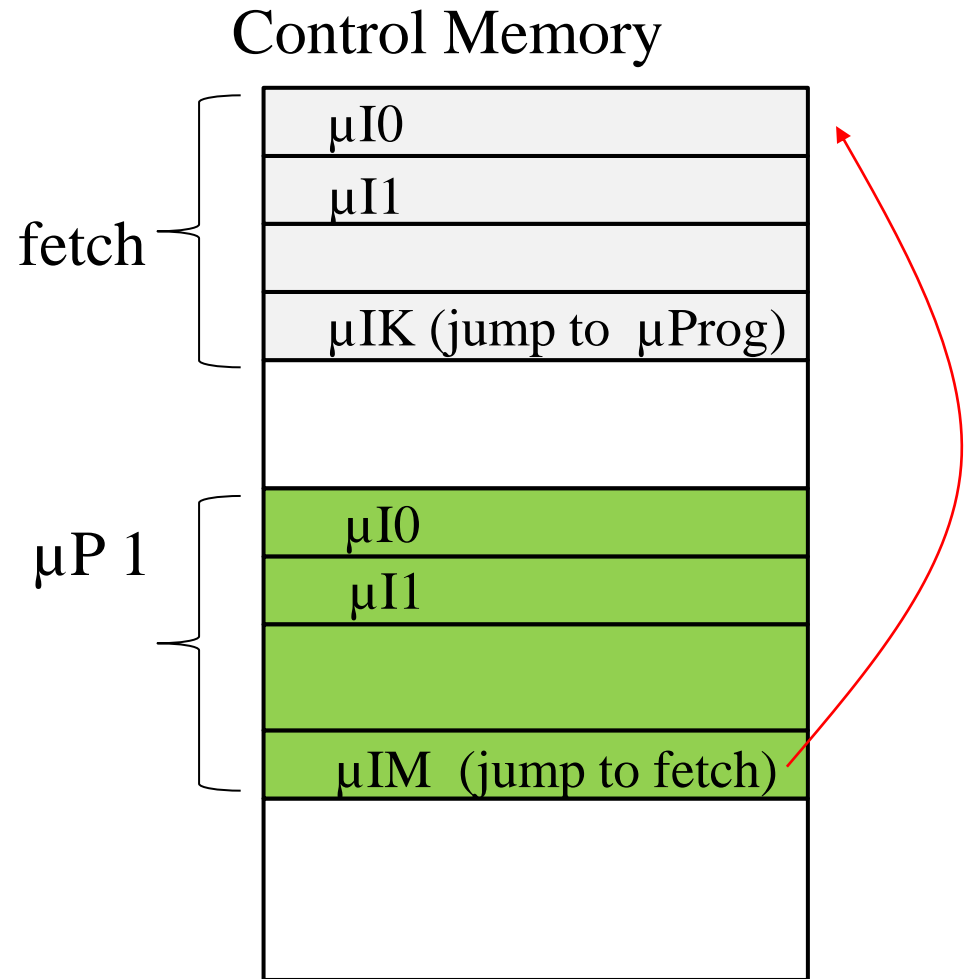


# Example of Control Unit with implicit sequencing



# Example of Control Unit with implicit sequencing

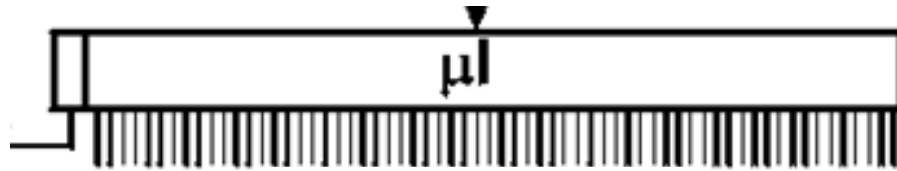
OC	ROM (co2 $\mu$ Addr)
0	$\mu$ Addr for 1 <sup>st</sup> $\mu$ I of $\mu$ P 0
1	$\mu$ Addr for 1 <sup>st</sup> $\mu$ I of $\mu$ P 1
N	$\mu$ Addr for 1 <sup>st</sup> $\mu$ I of $\mu$ P N



# Microinstruction format

- ▶ **Microinstruction format:**

specifies the number of bits and the meaning of each bit.



Control signals

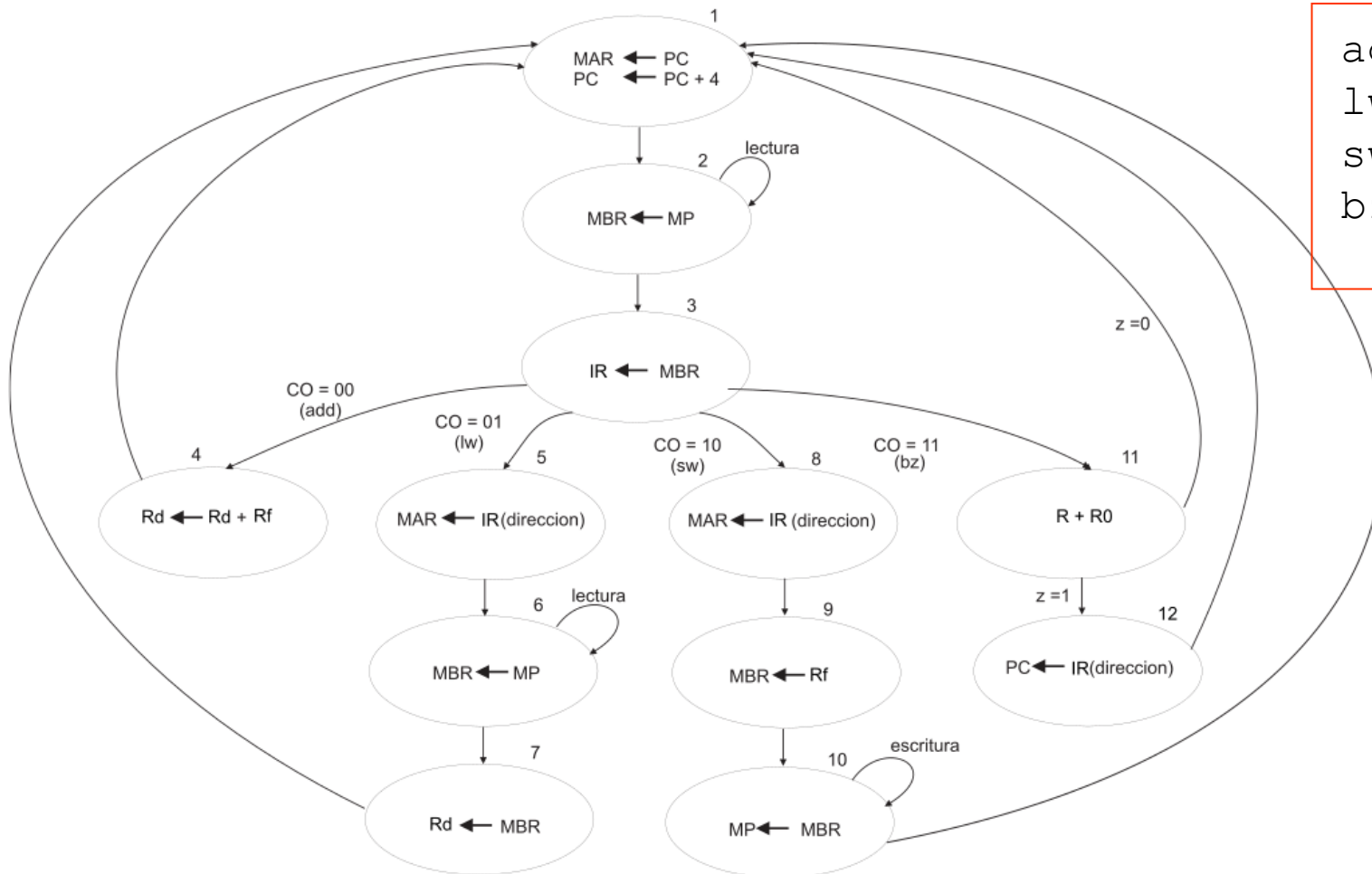
- ▶ **Signals grouped into fields:**

- ▶ Tristate bus signals
- ▶ ALU signals
- ▶ Registers file signals
- ▶ Main memory signals
- ▶ Multiplexor signals

# Example of state machine

Example for a computer with only 4 machine instructions

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

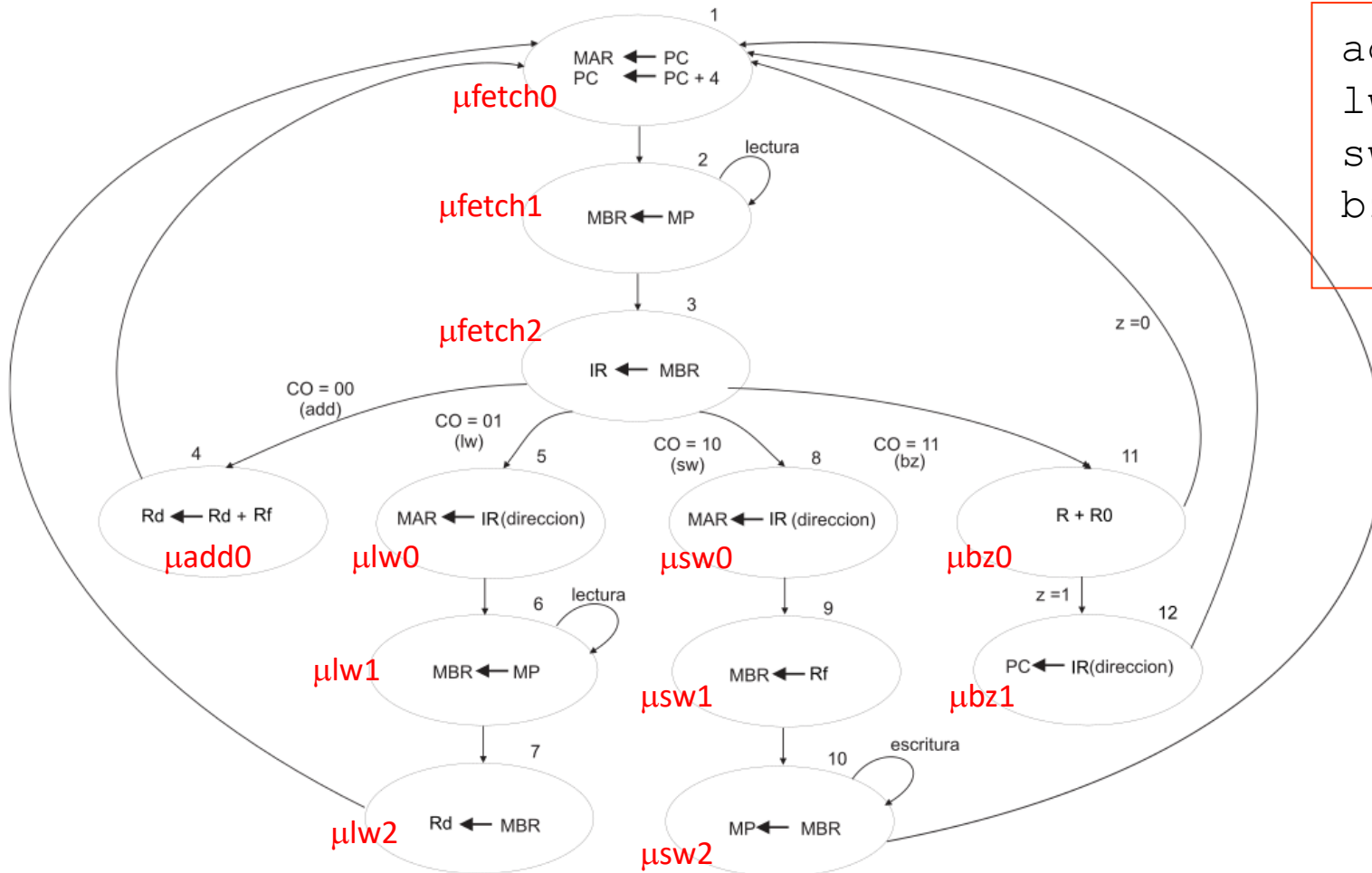




# Microinstructions for the example

Example for a computer with only 4 machine instructions

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

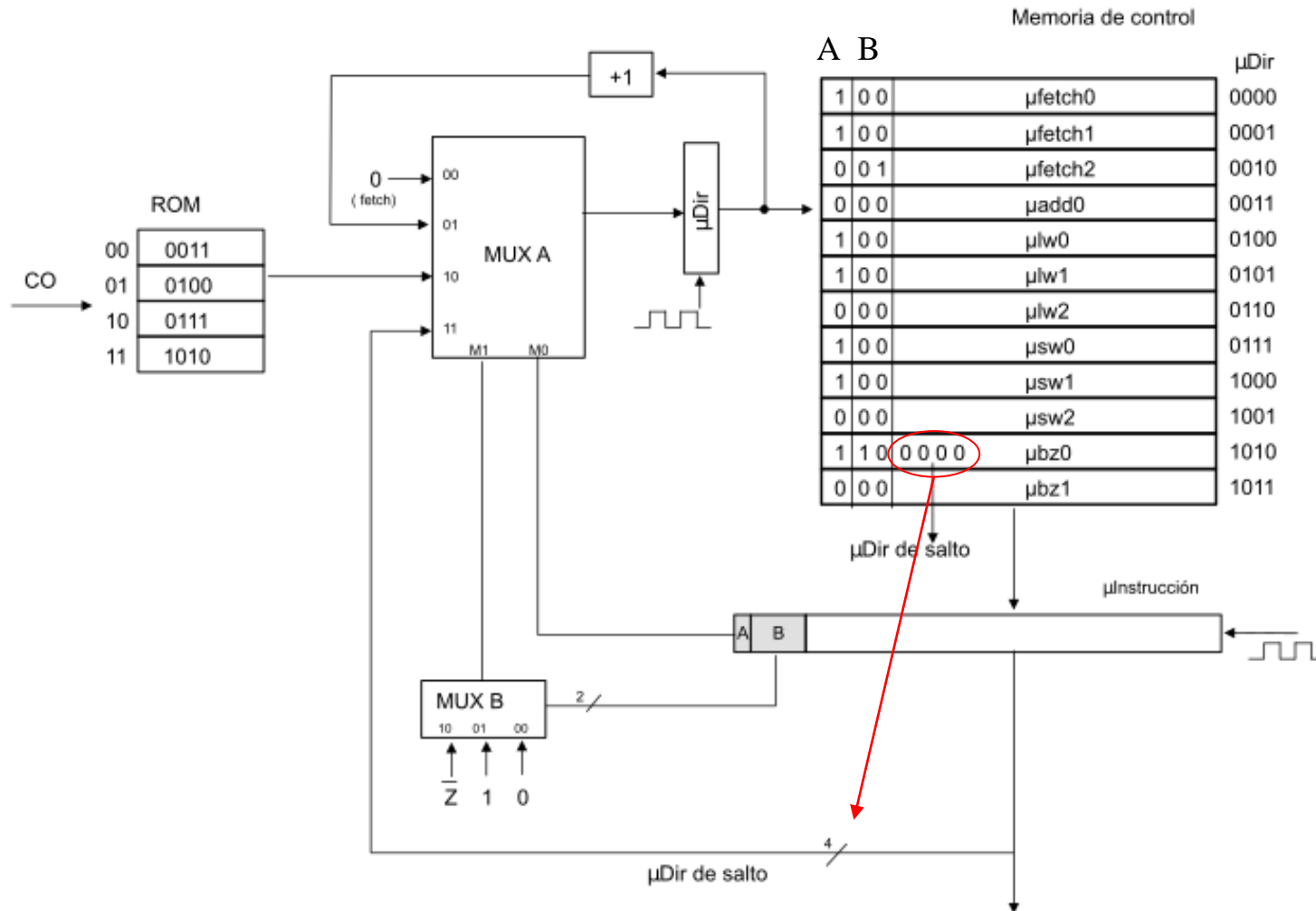


# Microcode for the example

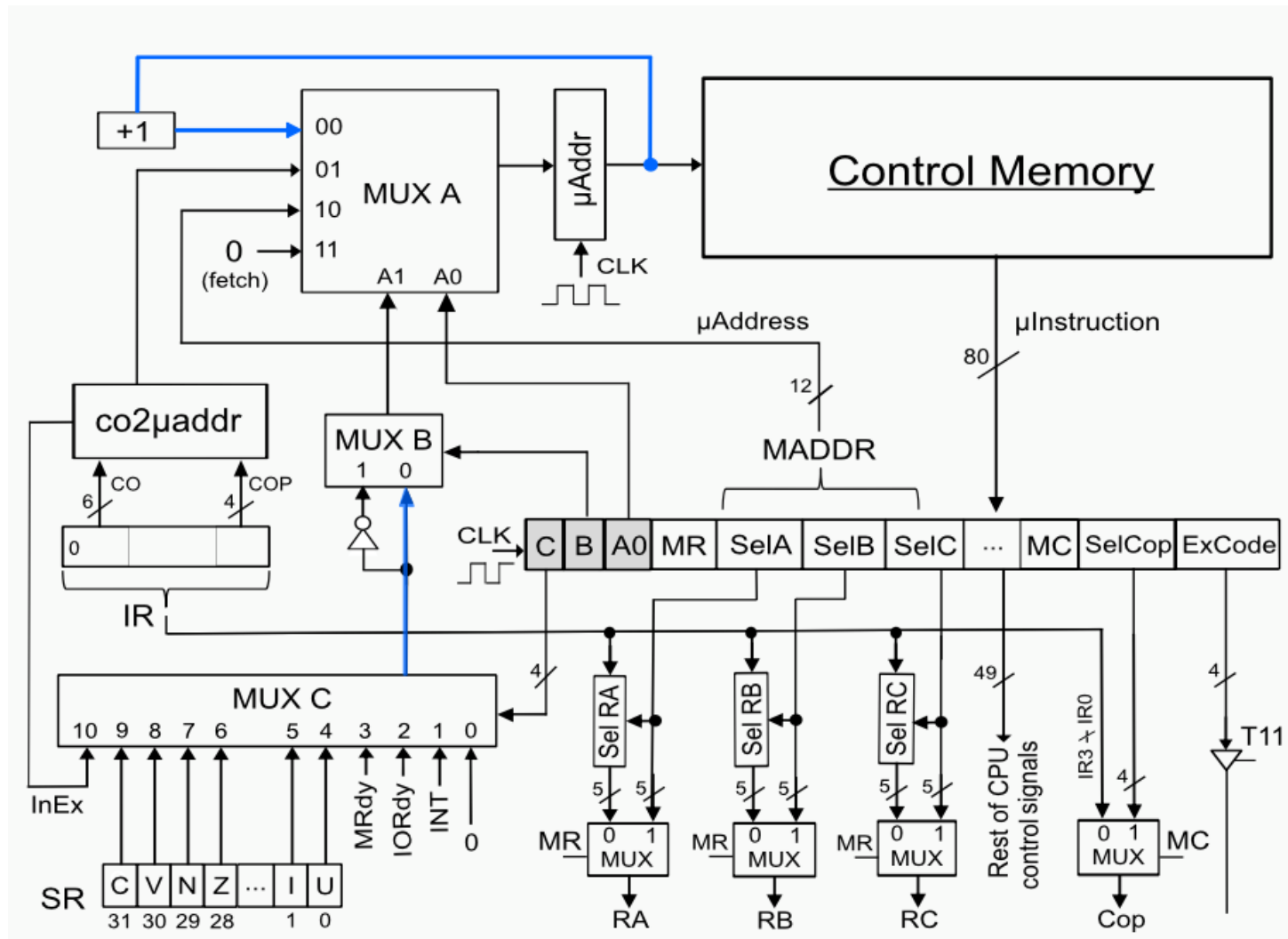
```
add r1, r2
lw r1, dir
bz dir
sw r1
```

	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	0	fetch	
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			
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	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	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	0	lw	
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			
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	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	0	sw	
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	0		
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	1	0	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			

# Microprogrammed control unit for the example

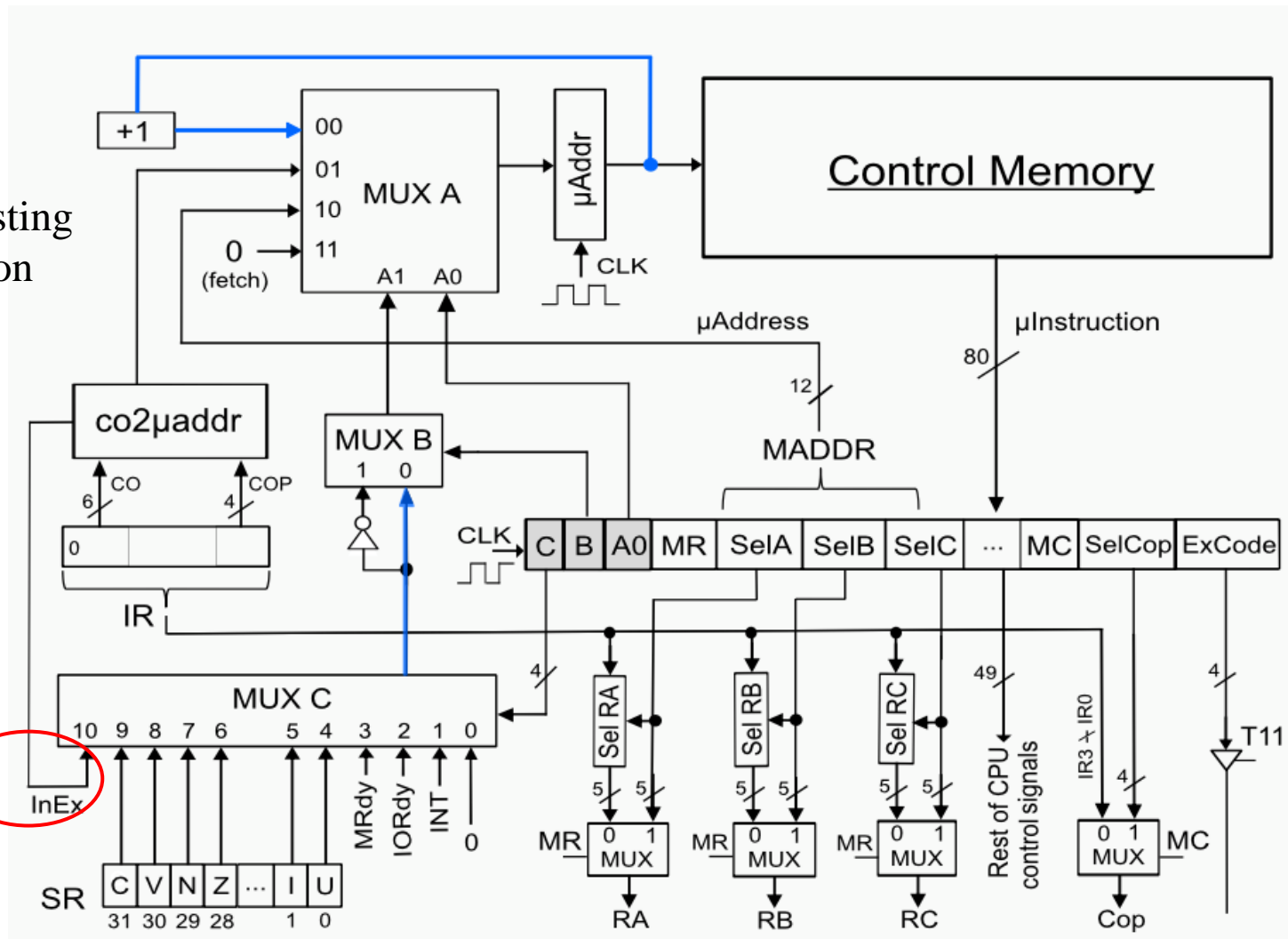


# Control Unit in WepSIM

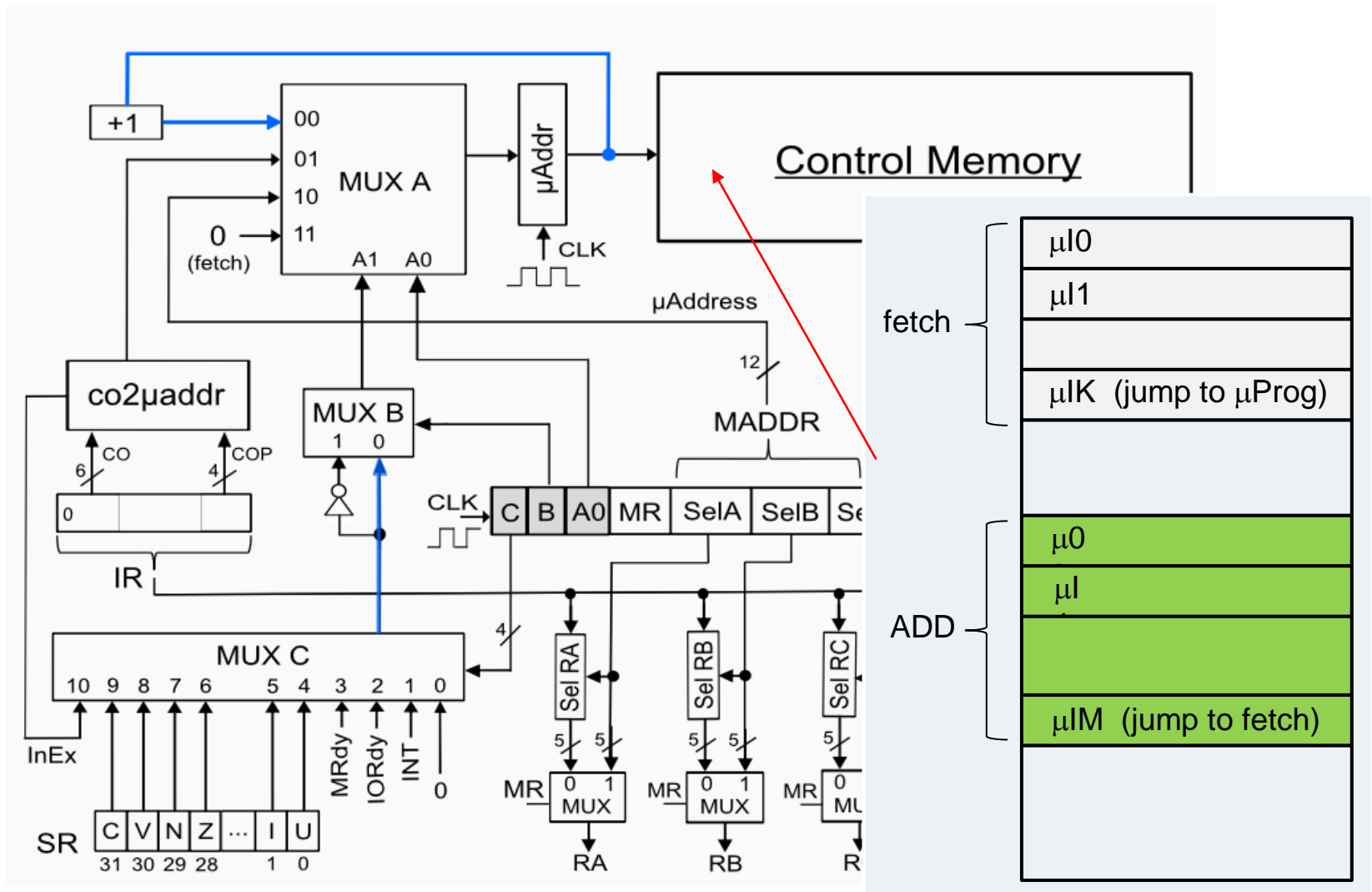


# Control Unit in WepSIM

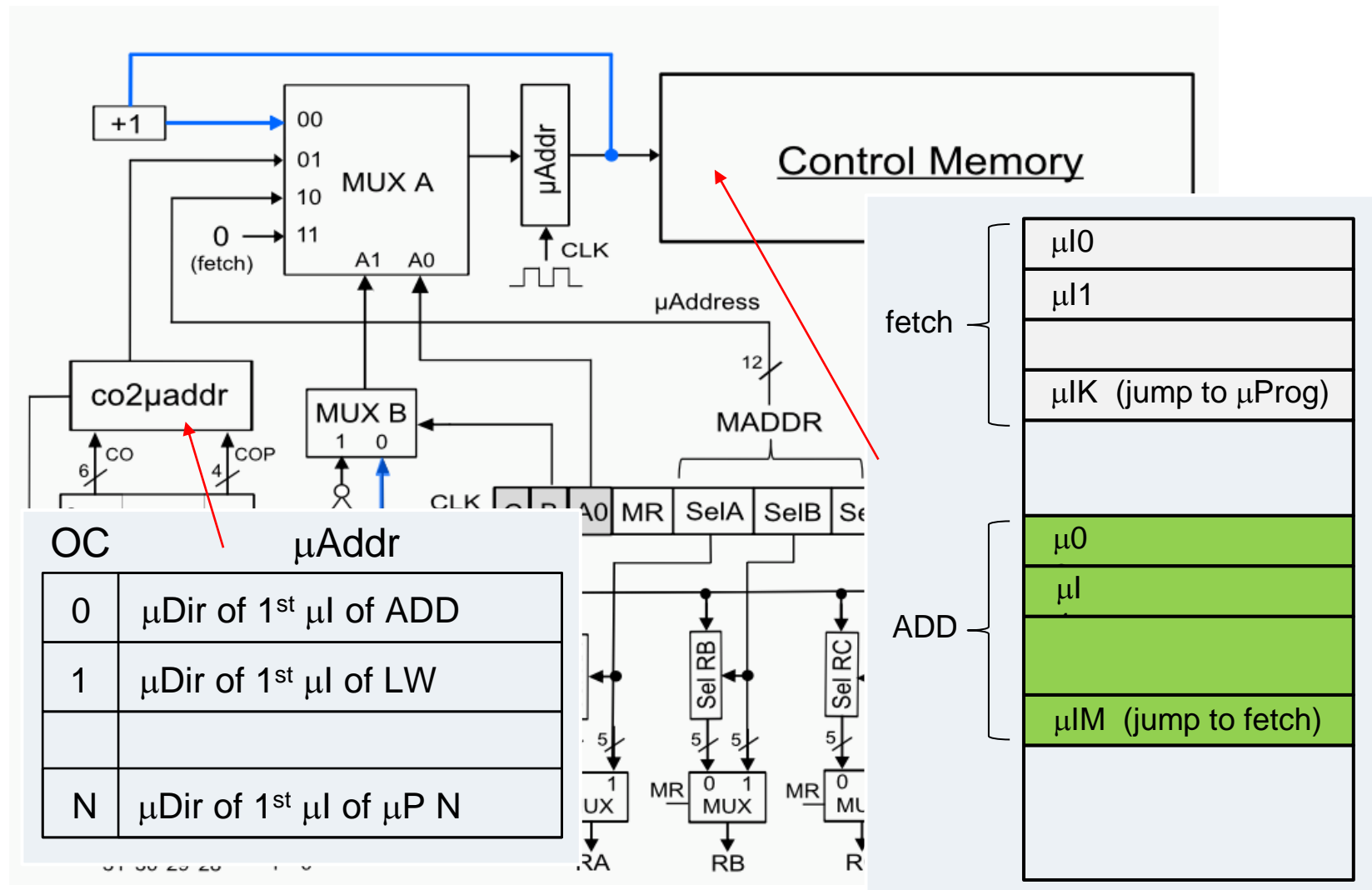
Non-existing instruction



# Control Unit in WepSIM

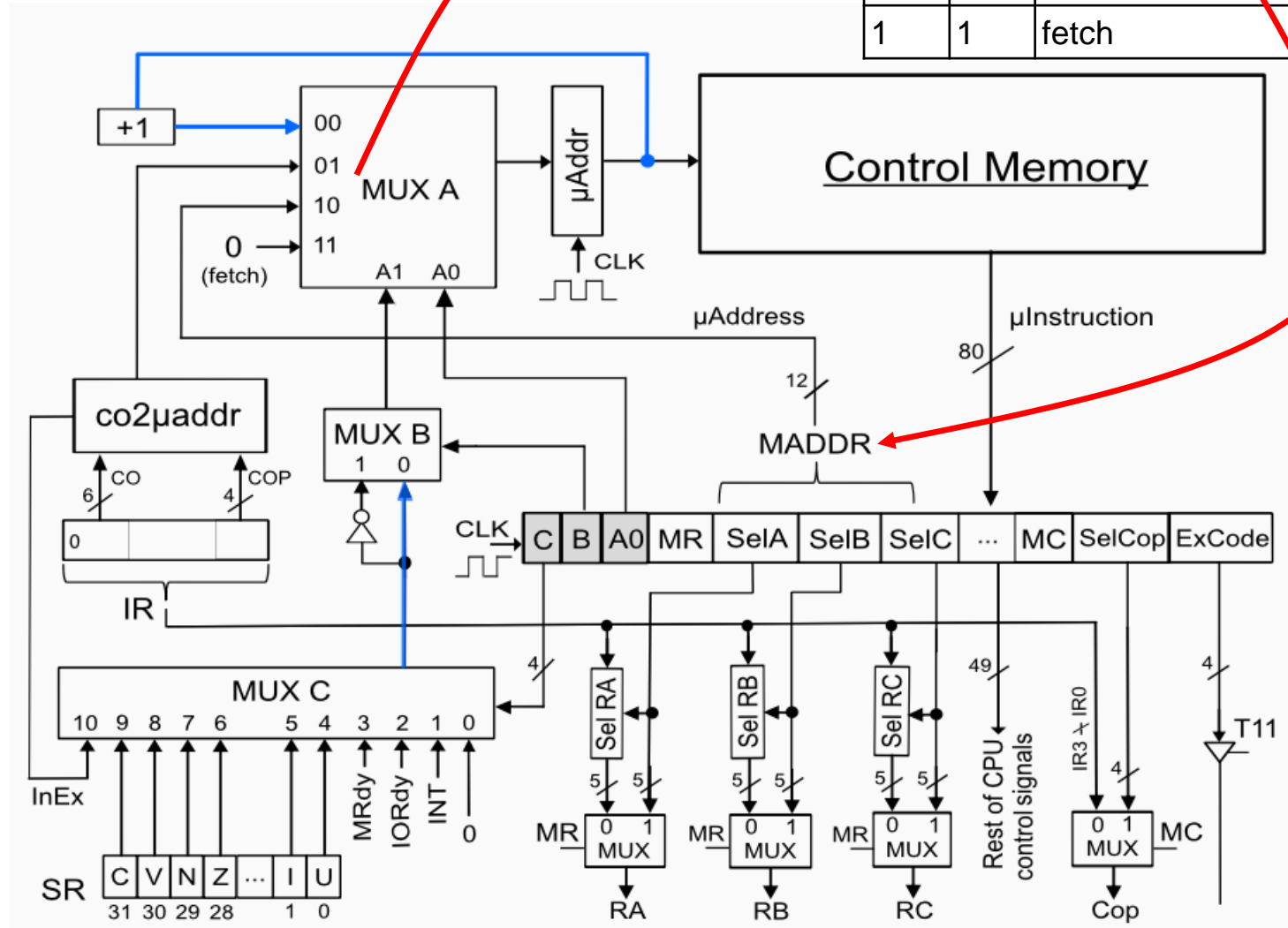


# Control Unit in WepSIM



# Control Unit in WepSIM

A1	A0	Action
0	0	Next $\mu$ Dir
0	1	Jump to $\mu$ Prog. (ROM)
1	0	$\mu$ Dir of jump
1	1	fetch





# Control Unit in WepSIM

A0	B	C3	C2	C1	C0	Action
0	0	0	0	0	0	Next $\mu$ Address
0	1	0	0	0	0	Unconditional jump to MADDR
0	0	0	0	0	1	Conditional jump to MADDR if INT = 1 (*)
0	1	0	0	1	0	Conditional jump to MADDR if IORdy = 0 (*)
0	1	0	0	1	1	Conditional jump to MADDR if MRdy = 0 (*)
0	0	0	1	0	0	Conditional jump to MADDR if U = 1 (*)
0	0	0	1	0	1	Conditional jump to MADDR if I = 1 (*)
0	0	0	1	1	0	Conditional jump to MADDR if Z = 1 (*)
0	0	0	1	1	1	Conditional jump to MADDR if N = 1 (*)
0	0	1	0	0	0	Conditional jump to MADDR if O = 1 (*)
1	0	0	0	0	0	Jump to $\mu$ Prog. (ROM c02 $\mu$ addr)
1	1	0	0	0	0	Jump to fetch ( $\mu$ Dir = 0)

- ▶ (\*) If the condition is not satisfied  $\rightarrow$  Next  $\mu$ Address
- ▶ Rest of entries  $\rightarrow$  undefined behavior

# Example

## Elemental operations with CU

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

Elemental operation	Signals
If (Z) $\mu\text{PC}=000100011100$	$A0=0, B=0, C=0110_2, \text{mADDR}=000100011100_2$

- ▶ **Unconditional jump to address 000100011111**

Elemental operation	Signals
$\mu\text{PC}=000100011111$	$A0=0, B=1, C=0000_2, \text{mADDR}=000100011111_2$

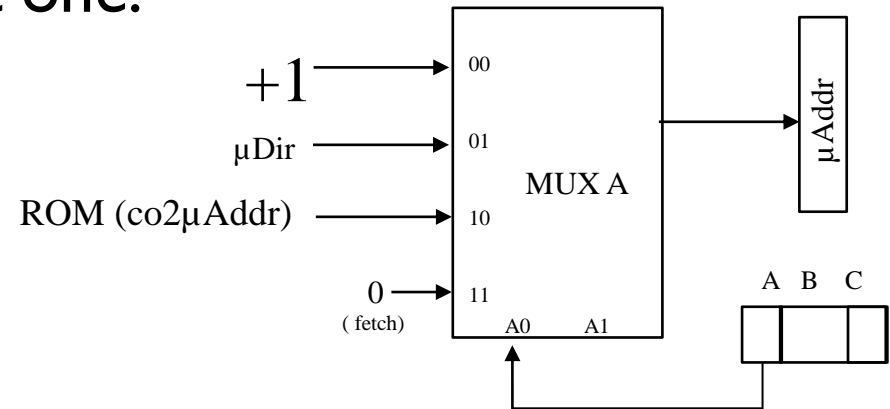
- ▶ **Jump to first  $\mu$ address of the  $\mu$ program related to OC**

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

# Example

- ▶ Jump to the  $\mu$ Address 000100011100 (12 bits) if  $Z = 1$ . Otherwise, jump to the next one:

- ▶  $A0 = 0$
- ▶  $B = 0$
- ▶  $C = 0110$
- ▶  $\mu$ Addr = 000100011100

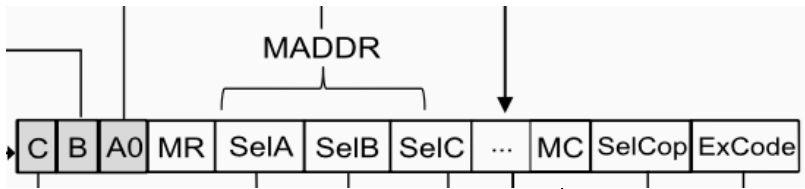


- ▶ Unconditional jump to  $\mu$ Address 000100011111

- ▶  $A0 = 0$
- ▶  $B = 1$
- ▶  $C = 0000$
- ▶  $\mu$ Addr = 000100011111

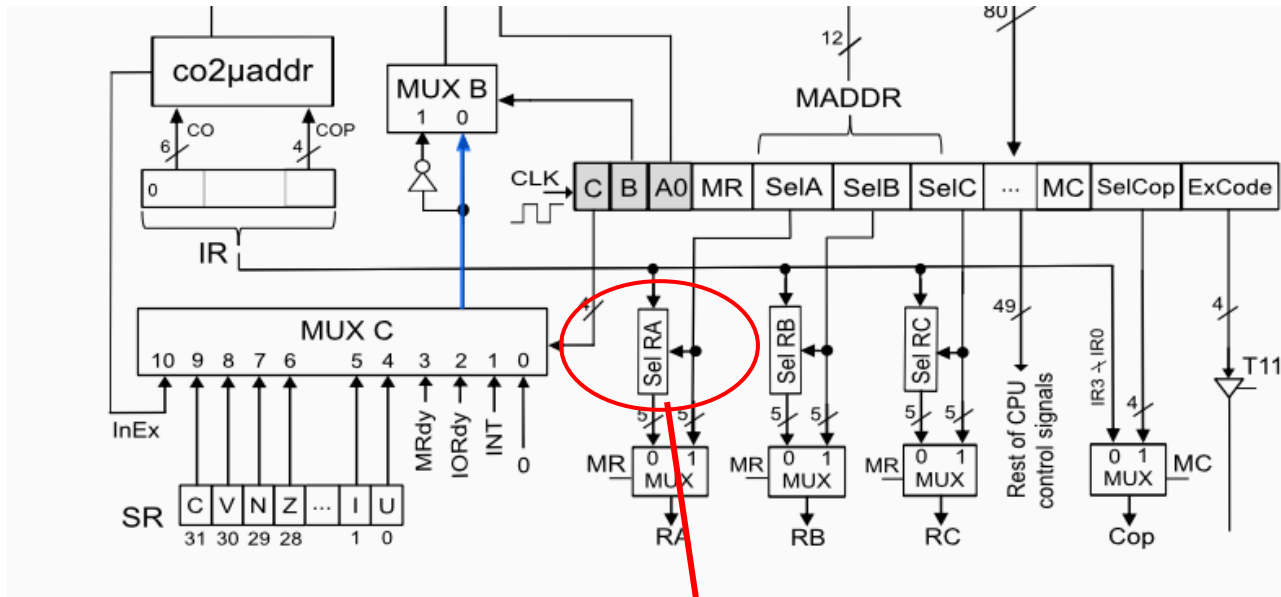
$\mu$ Address encoded  
in bits 72-61  
of the  $\mu$ Instruction

# Microinstruction format

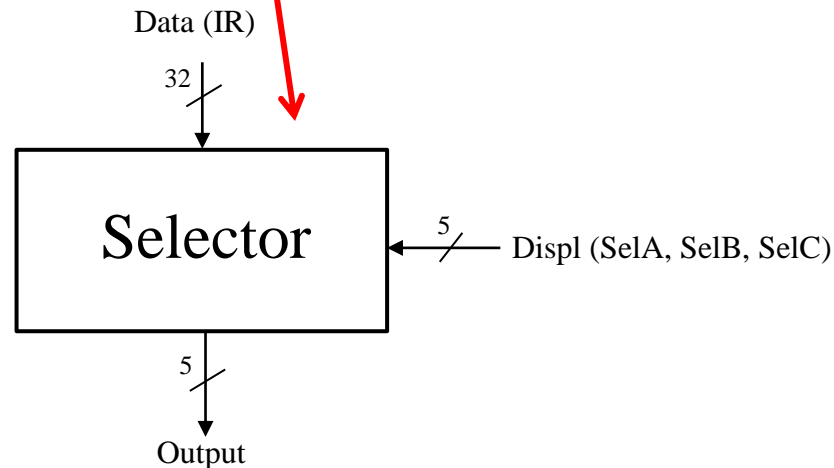


C0 .. C7	Load register
Ta, Td	Tristate buffers to bus
T1..T10	Tristate buffers
M1, M2, M7, MA, MB	Multiplexors
SelP	State register selector
LC	Load in Register File
SE	Sign extensión
Size, Offset	Selector of IR register
BW	Size of memory Access
R, W	Main memory operation
IOR, IOW	I/O operation
INTA	INT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

# Register file selector

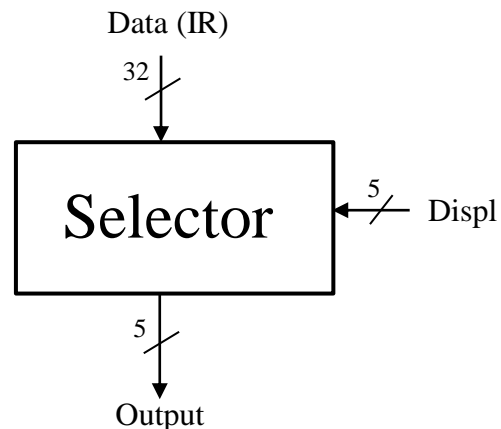
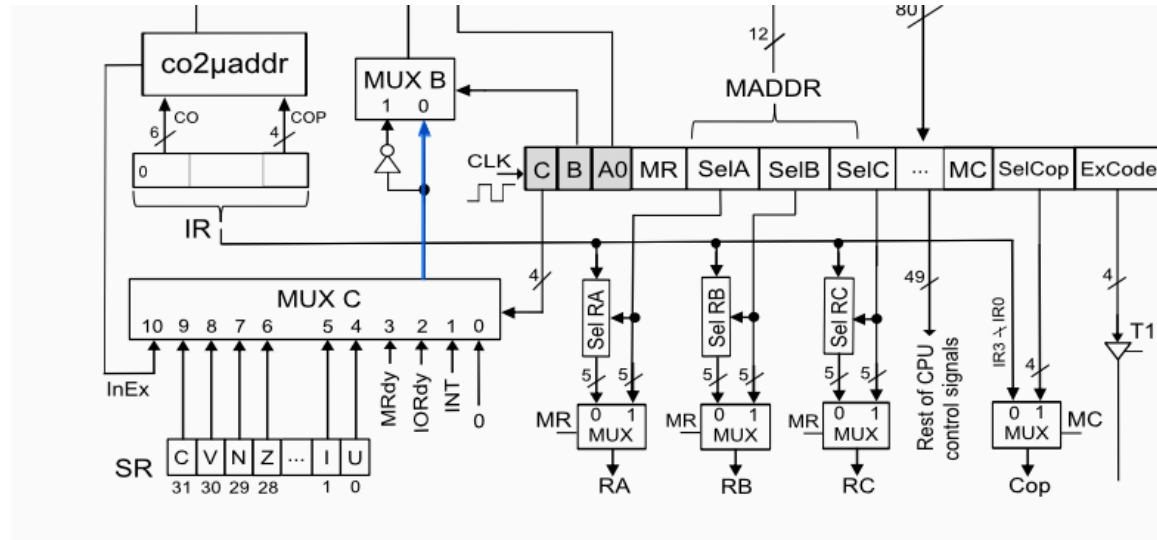


Select 5 bits within 32-bit starting from the position indicated in *Displ* (lower bit)



# Register file selector

## Example



RI:  $D_{31}D_{30}D_{29}D_{28}D_{27}D_{26}D_{25} \dots D_4D_3D_2D_1D_0$

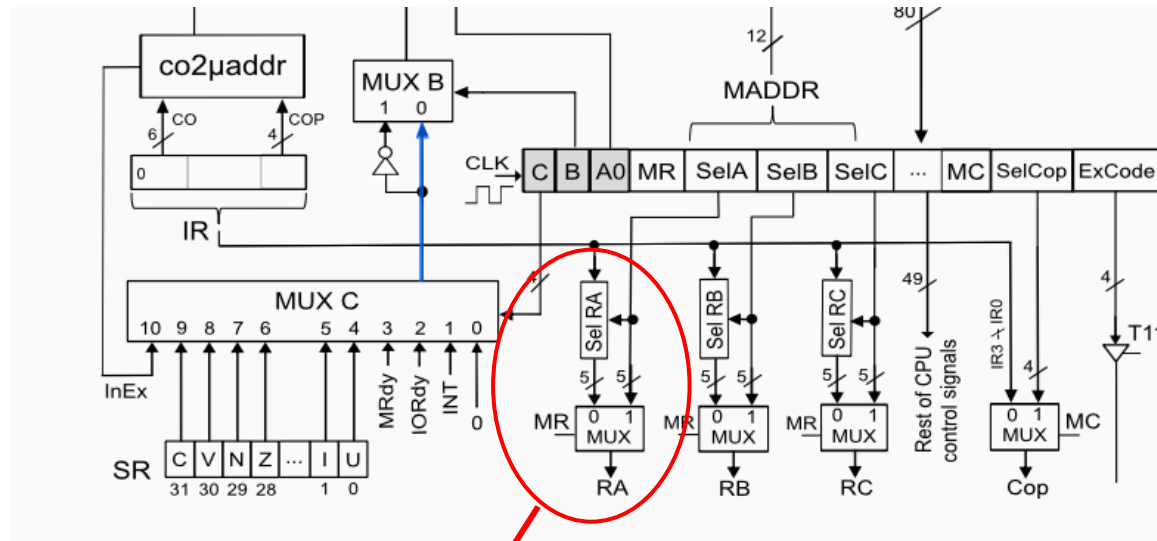
If Displ = 11011  $\rightarrow$  Output =  $D_{31}D_{30}D_{29}D_{28}D_{27}$

If Displ = 00000  $\rightarrow$  Output =  $D_4D_3D_2D_1D_0$

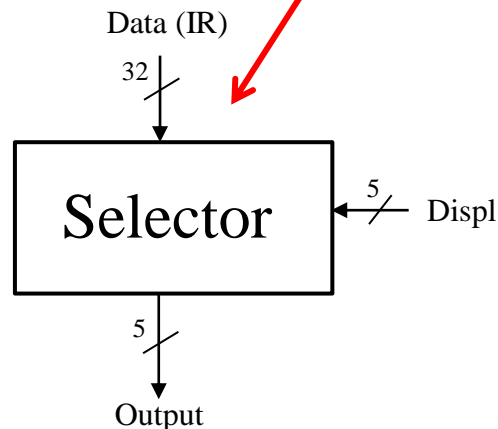
If Displ = 10011  $\rightarrow$  Output =  $D_{23}D_{22}D_{21}D_{20}D_{19}$

If Displ = 01011  $\rightarrow$  Output =  $D_{15}D_{14}D_{13}D_{12}D_{11}$

# Register file selector



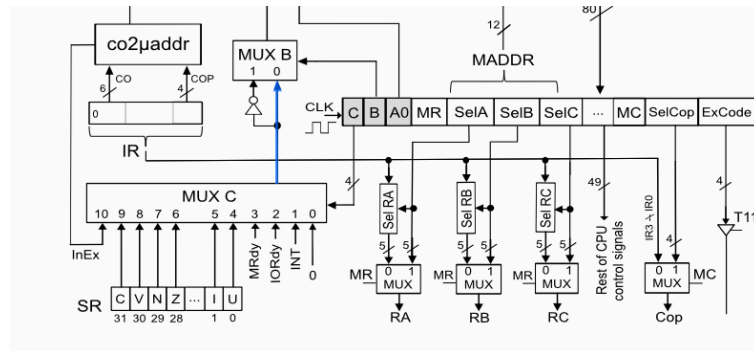
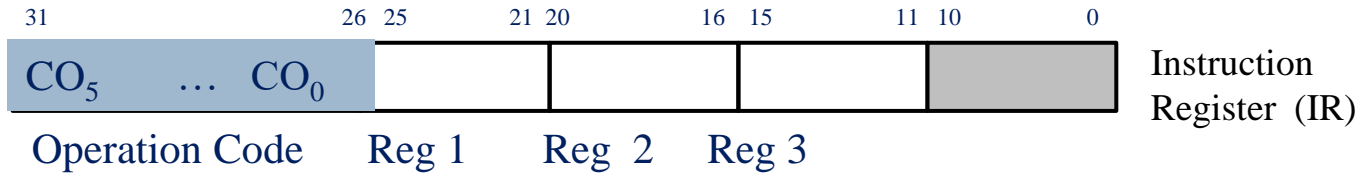
Select 5 bits within 32-bit starting from the position indicated in *Displ* (lower bit)



- If  $MR = 1$ ,  $RA$  is obtained from the  $\mu$ Instruction
- If  $MR = 0$ ,  $RA$  is obtained from a field of IR

## Register file selector

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

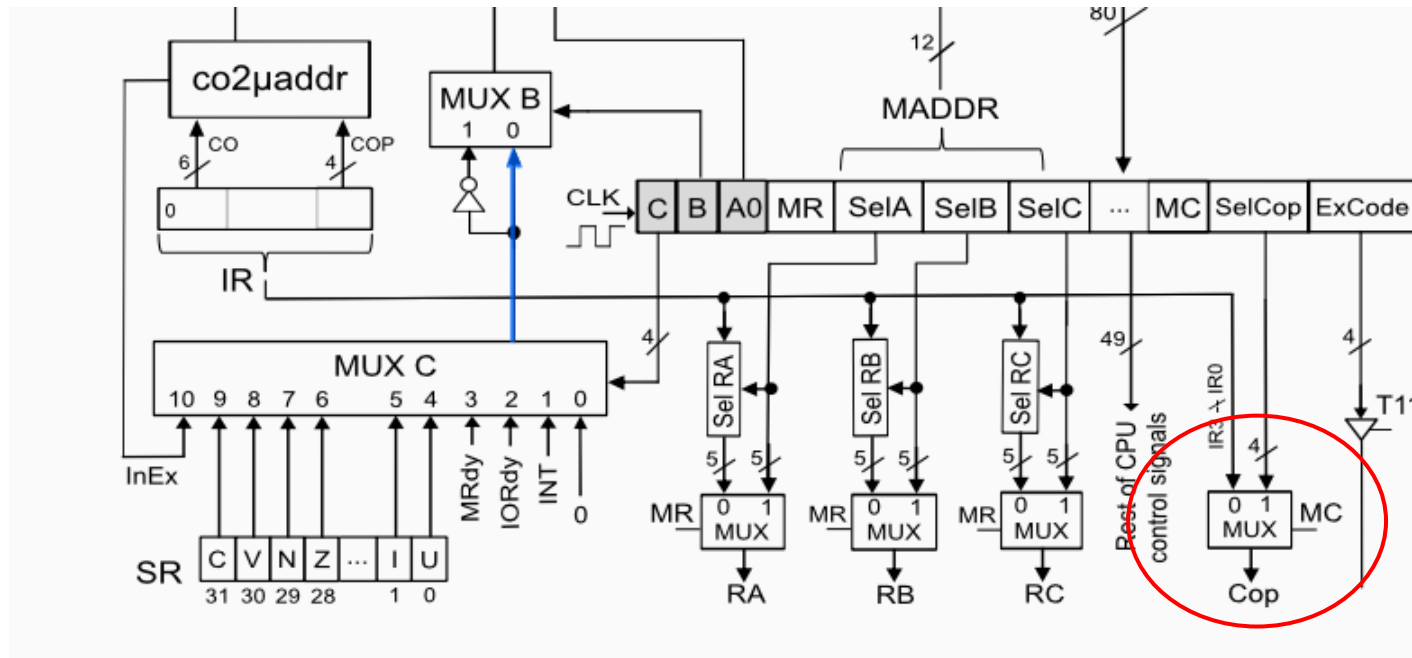


$$\mathbf{MR} = \mathbf{0}$$

- ▶ If you want to select the field with the Reg 2 in gate B of the record 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 record file → SelA = 01011 (RA is obtained from bits 15...11 of IR)
- ▶ If you want to select the field with the Reg 1 in gate C of the record file → SelC = 10101 (RC is obtained from bits 25...21 of IR)



# Selection of the ALU operation



- 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



# Examples

## ► Instruction to microprogramming with WepSIM\*:

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

\* Memoria de un ciclo

# Microprogrammed instructions

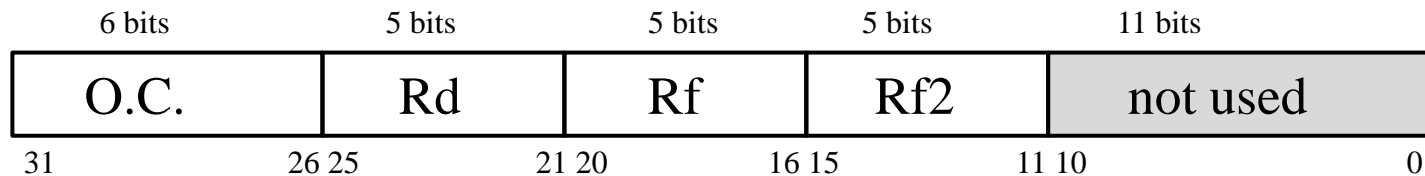
## ► FETCH

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

# Microprogrammed instructions

## ► ADD Rd, Rf1, Rf2

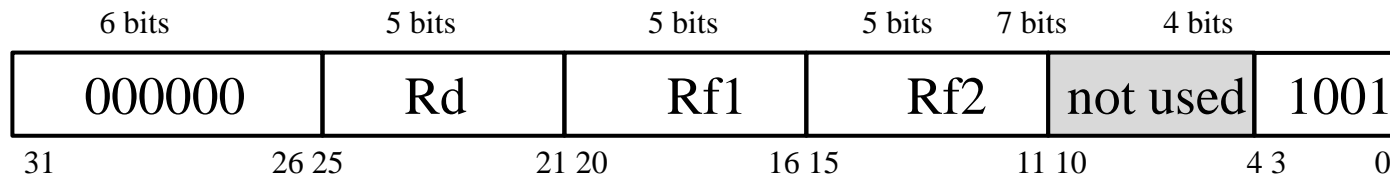
Cycle	Elemental Op.	Activated signals (rest to 0)	C	B	A0
0	$Rd \leftarrow Rf1 + Rf2$	SelCop = 1010, MC SelP=11, C7, M7 T6, LC SelA = 10000 (16) SelB = 01011 (11) SelC = 10101 (21)	0000	1	1



# Microprogrammed instructions (another)

## ► ADD Rd, Rf1, Rf2

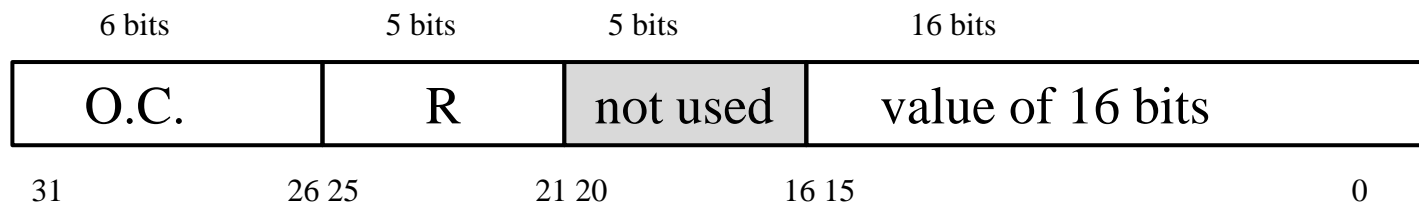
Cycle	Elemental Op.	Activated signals (rest to 0)	C	B	A0
0	$Rd \leftarrow Rf1 + Rf2$	SelCop = 1010, MC SelP=11, C7, M7 T6, LC SelA = 10000 (16) SelB = 01011 (11) SelC = 10101 (21)	0000	1	1



# Microprogrammed instructions

## ► LI R, value

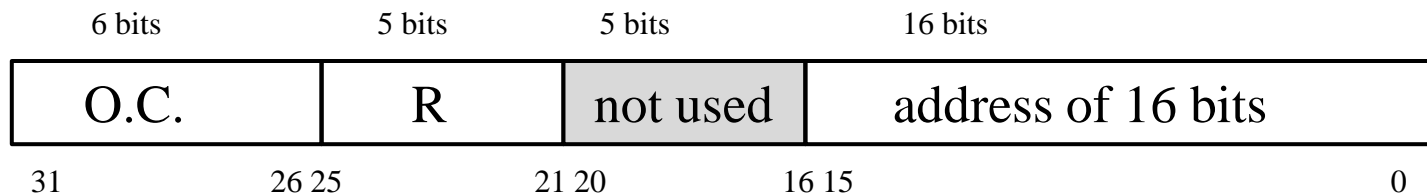
Cycle	Elemental Op.	Activated signals (rest to 0)	C	B	A0
0	$R \leftarrow IR \text{ (value)}$	LC SelC = 10101 (21) T3, Size = 10000 Offset= 00000 SE=1	0000	1	1



# Microprogrammed instructions

- LW R addr      # sync memory, 1 clock cycle

Cycle	Elemental Op.	Activated signals (rest to 0)	C	B	A0
0	MAR $\leftarrow$ IR (dir)	T3, C0 Size = 10000, Offset= 00000	0000	0	0
1	MBR $\leftarrow$ MP[MAR]	Ta, R, BW = 11, CI, MI	0000	0	0
2	R $\leftarrow$ MBR	T1, LC, SelC = 10101	0000	1	1





# Microprogrammed instructions

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

Cycle	Elemental Op.	Activated signals (rest to 0)	C	B	A0
0	MAR $\leftarrow$ IR (dir)	T3, C0 Size = 10000, Offset= 00000	0000	0	0
1	while (!MRdy) MBR $\leftarrow$ MP[MAR]	Ta, R, BW = 11, C1, M1, MADDR= $\mu$ Add of this microinstruction	0011	1	0
2	R $\leftarrow$ MBR	T1, LC, SelC = 10101	0000	1	1

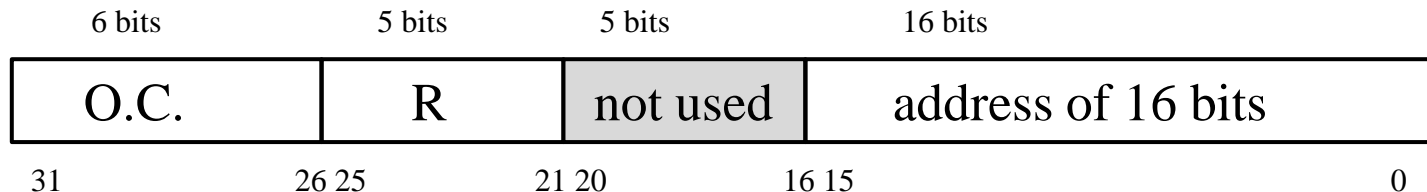
This microinstruction is beening executed while MRdy==0



# Microprogrammed instructions

- SW R addr # sync memory, 1 clock cycle

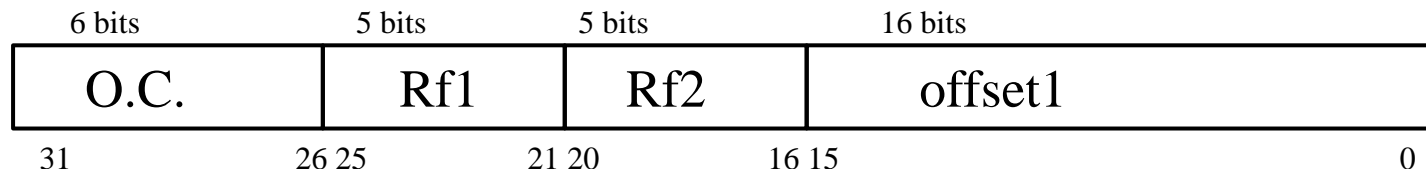
Cycle	Elemental Op.	Activated signals (rest to 0)	C	B	A0
0	$MBR \leftarrow R$	$T9, CI, SelA=10101$	0000	0	0
1	$MAR \leftarrow IR(addr)$	$T3, C0,$ Size = 10000, offset= 00000	0000	0	0
2	$MP[addr] \leftarrow MBR$	$Td, Ta, BW = 11, W$	0000	1	1



# Microprogrammed instructions

## ► BEQ Rf1, Rf2, offset1

Cycle	Elemental Op.	Activated signals (rest to 0)	C	B	A0
0	Rf1 - Rf2	SelCop = 1011, MC, C7, M7 SelP = 11, SelA = 10101 SelB = 10000	0000	0	0
11	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	SelCop = 1010, MC, MA, MB=01, T6, C2,	0000	1	1



# Microprogrammed instructions

## ► J dir

Cycle	Elemental Op.	Activated signals (rest to 0)	C	B	A0
0	PC $\leftarrow$ IR (dir)	C2,T3, size = 10000, offset= 00000	0000	I	I



# Defining instructions in WepSIM

<List of implemented instructions>

<Register file specification>

<Pseudo instructions>

# Defining instructions in WepSIM

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

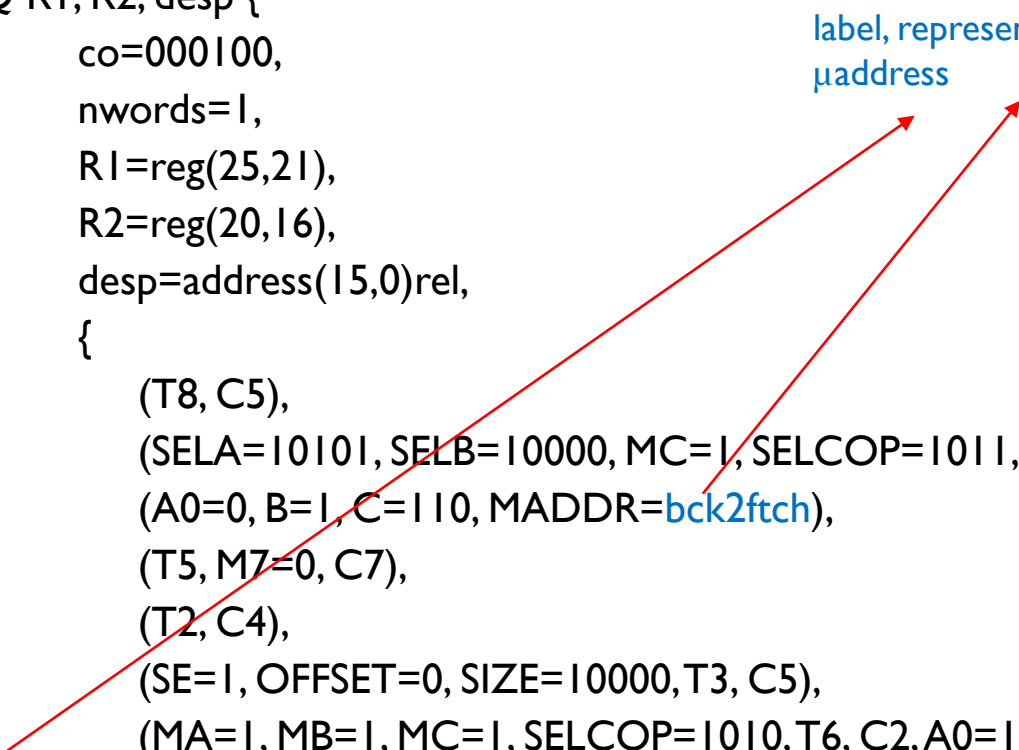
# Defining instructions in WepSIM

```
ADD R1,R2,R3 {  
    co=000000,  
    nwords=1,  
    R1=reg(25,21),  
    R2=reg(20,16),  
    R3=reg(15,11),  
    {  
        (SelCop=1010, MC, SelP=11, M7,C7,T6, LC,  
        SelA=01011, SelB=10000, SelC=10101,  
        A0=1, B=1, C=0)  
    }  
}
```

# Defining instructions in WepSIM

```
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)  
    }  
}
```

label, represents a  $\mu$ address





# Register specification

```
registers {  
    0=$zero, 1=$at,  
    2=$v0, 3=$v1,  
    4=$a0, 5=$a1,  
    6=$a2, 7=$a3,  
    8=$t0, 9=$t1,  
    10=$t2, 11=$t3,  
    12=$t4, 13=$t5,  
    14=$t6, 15=$t7,  
    16=$s0, 17=$s1,  
    18=$s2, 19=$s3,  
    20=$s4, 21=$s5,  
    22=$s6, 23=$s7,  
    24=$t8, 25=$t9,  
    26=$k0, 27=$k1,  
    28=$gp, 29=$sp (stack_pointer),  
    30=$fp, 31=$ra  
}
```

ARCOS Group

**uc3m** | Universidad **Carlos III** de Madrid

## Lesson 4 (II) The processor

Computer Structure  
Bachelor in Computer Science and Engineering

