

ARCOS Group

uc3m | Universidad **Carlos III** de Madrid

Lesson 4 (I)

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. Execution modes
6. Interrupts
7. Control unit design
8. Computer startup
9. Performance and parallelism

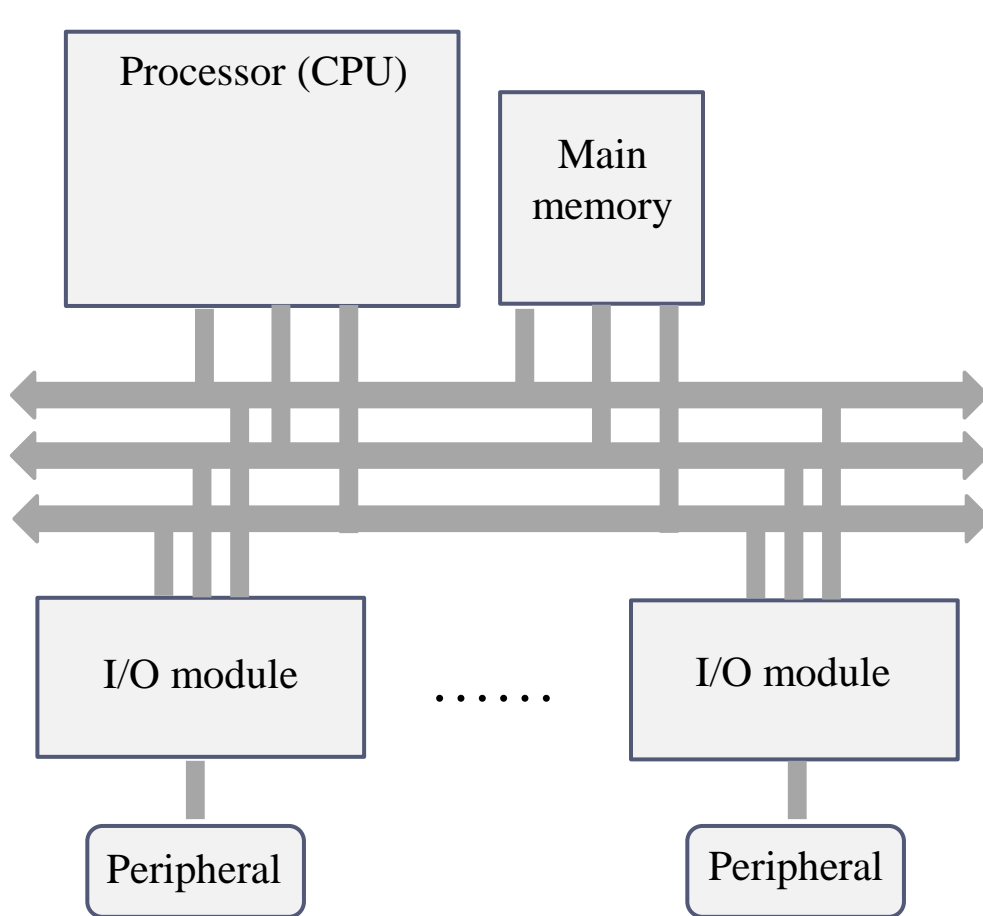
Contents

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

- 1) Motivation and goals
- 2) Basic functionality of the control unit
- 3) Control signals and elemental operations
- 4) Introduction of the elemental processor

Computer components

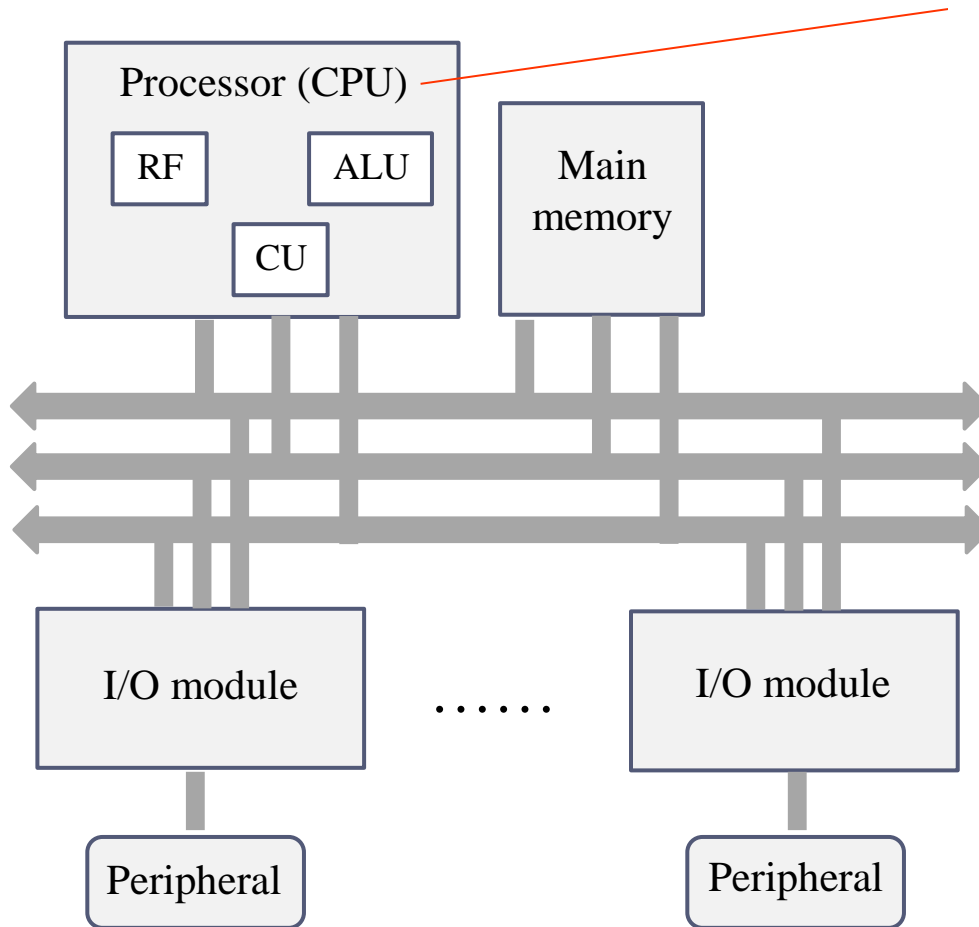
review



- ▶ Processor
- ▶ Main memory
- ▶ I/O module
- ▶ Peripheral

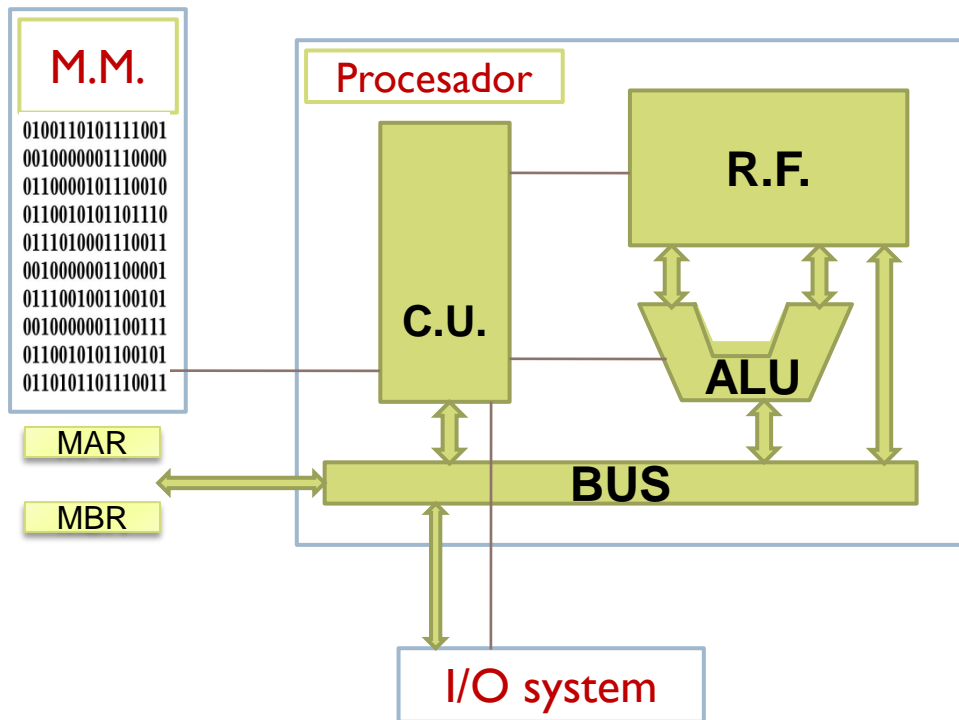
Processor components

review



- ▶ Register file
- ▶ Arithmetic-logic unit
- ▶ Control unit
- ▶ Cache memory

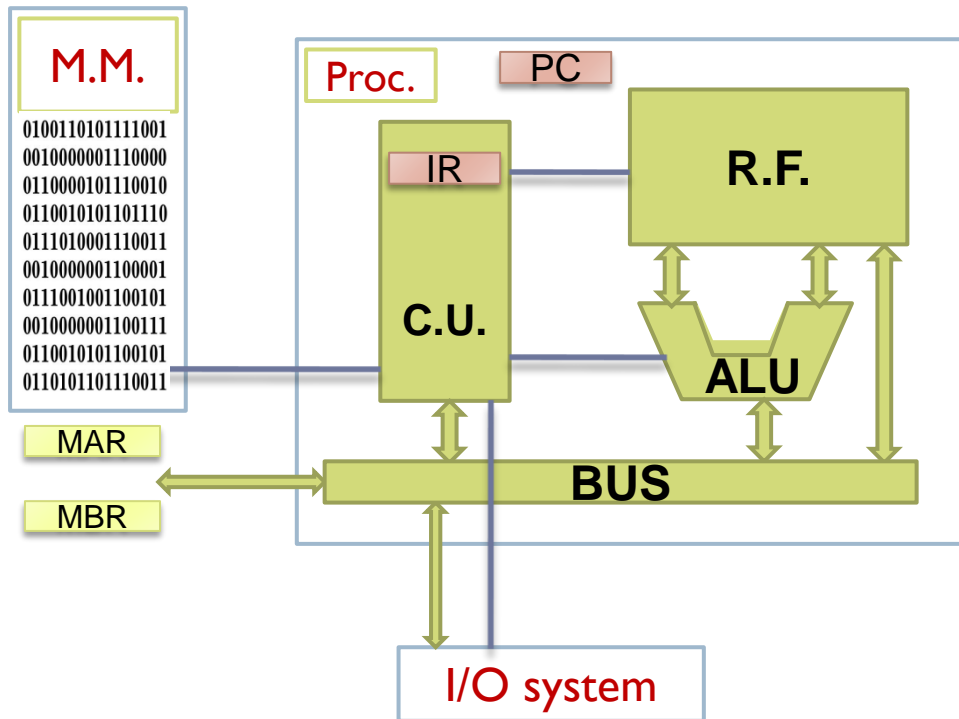
Main motivation



- In lesson 3, we studied **what** processor execute: assembly programming.
- In lesson 4 we are going to study **how** the instructions are executed in the computer.

How C.U. works:

Execute machine instructions



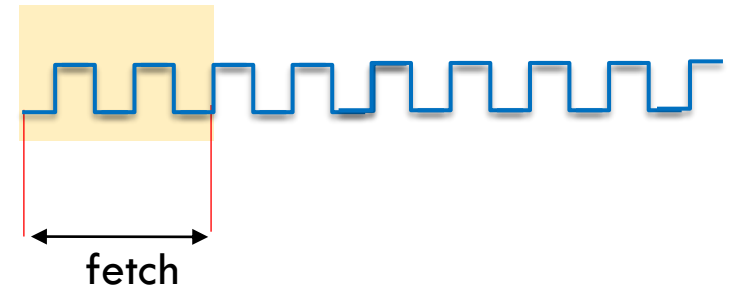
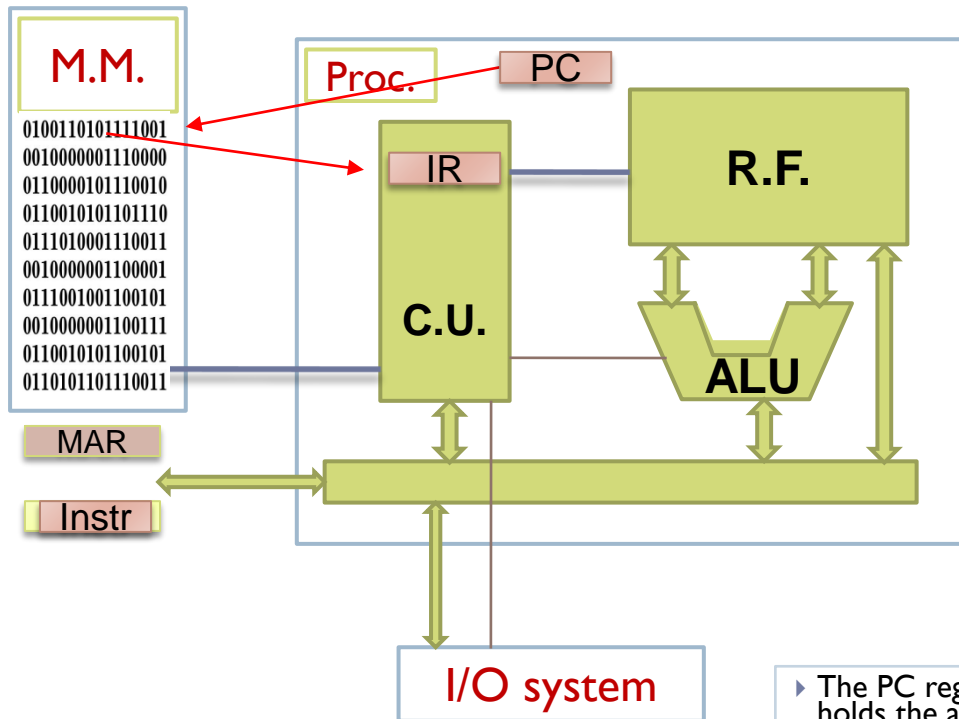
- Each element of the computer has inputs, outputs and control signals.
- At each clock cycle, the Control Unit (C.U.) sends the control signals via the control bus wires.
- Control signals indicate what value to output:
 - Move from an input to an output: $S=E_x$
 - Transform an input: $S=f(E)$

Contents

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

- 1) Motivation and goals
- 2) Basic functionality of the control unit
- 3) Control signals and elemental operations
- 4) Introduction of the elemental processor

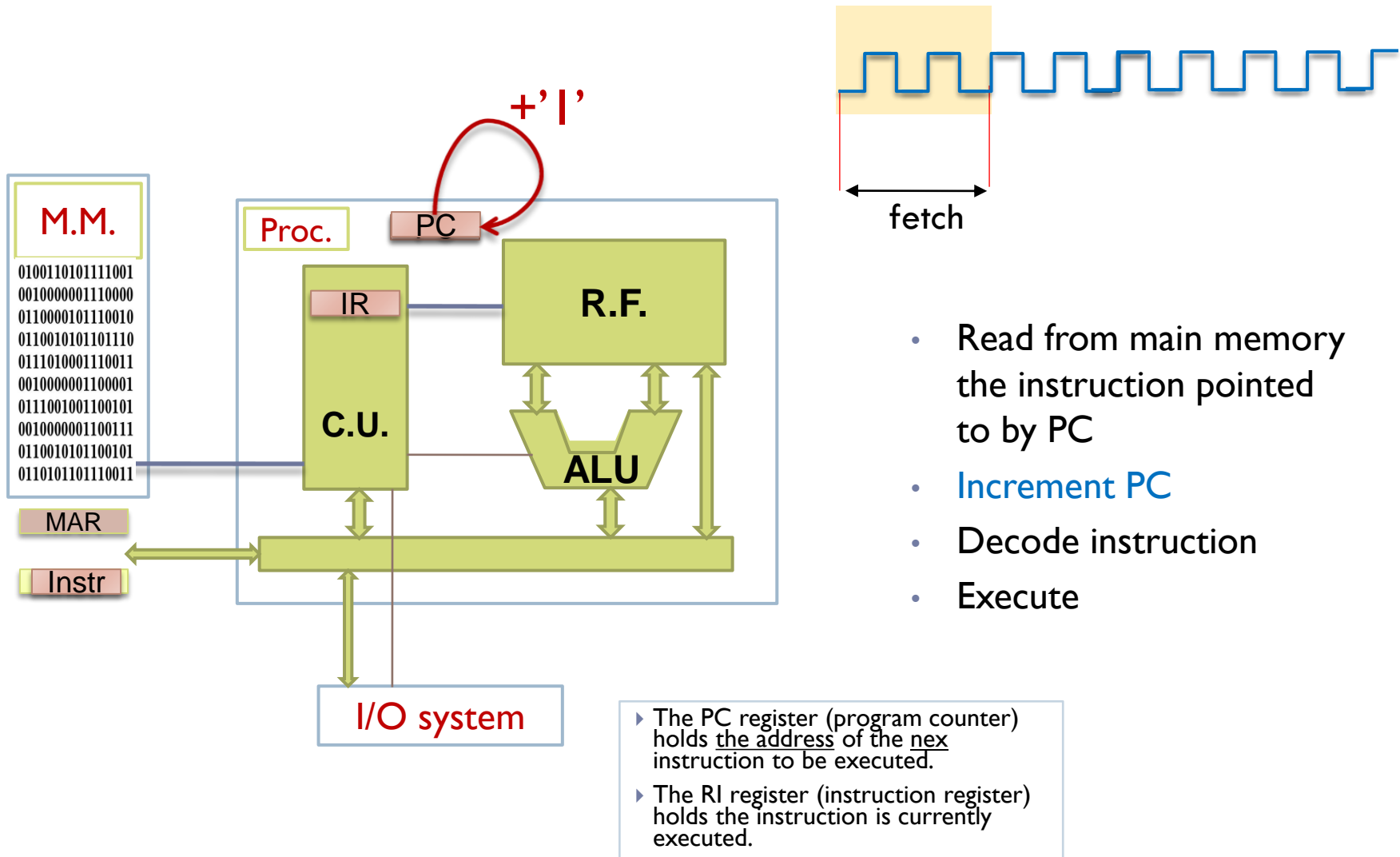
How C.U. works...



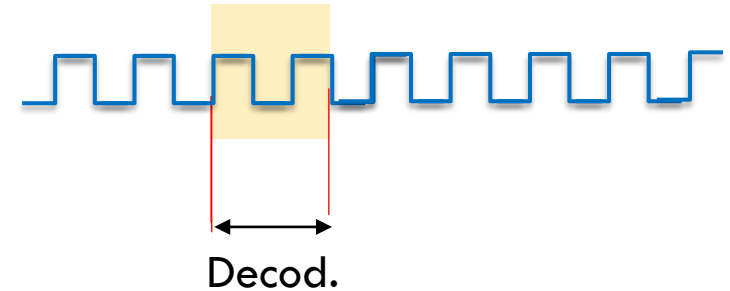
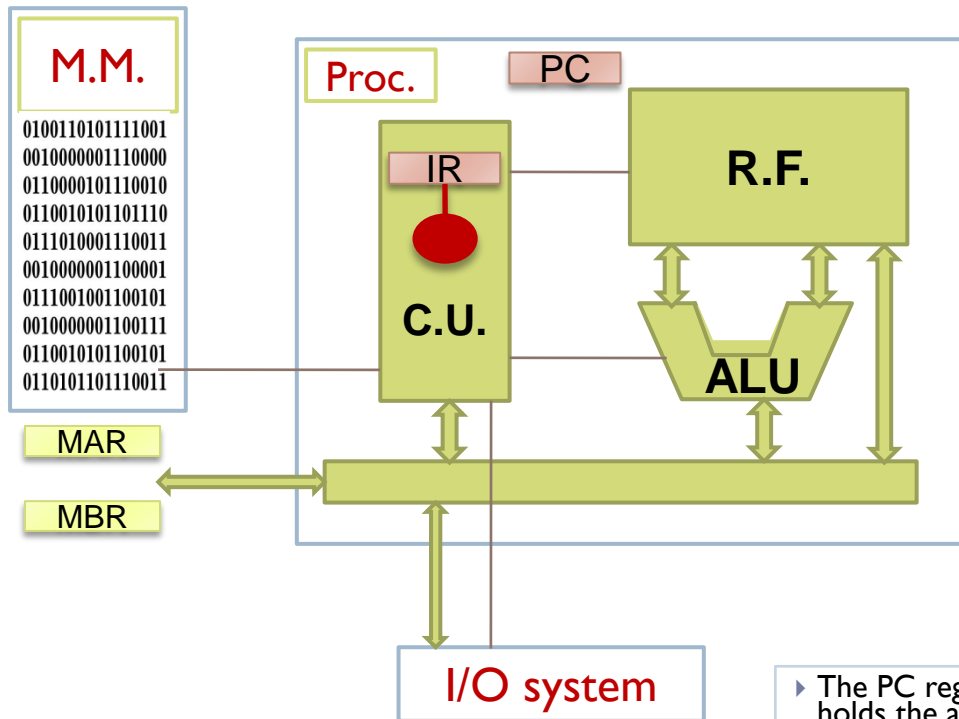
- Read from main memory the instruction pointed to by PC
- Increment PC
- Decode instruction
- Execute

- ▶ The PC register (program counter) holds the address of the next instruction to be executed.
- ▶ The RI register (instruction register) holds the instruction is currently executed.

How C.U. works...



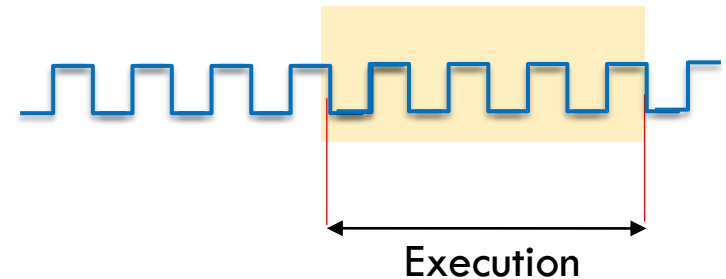
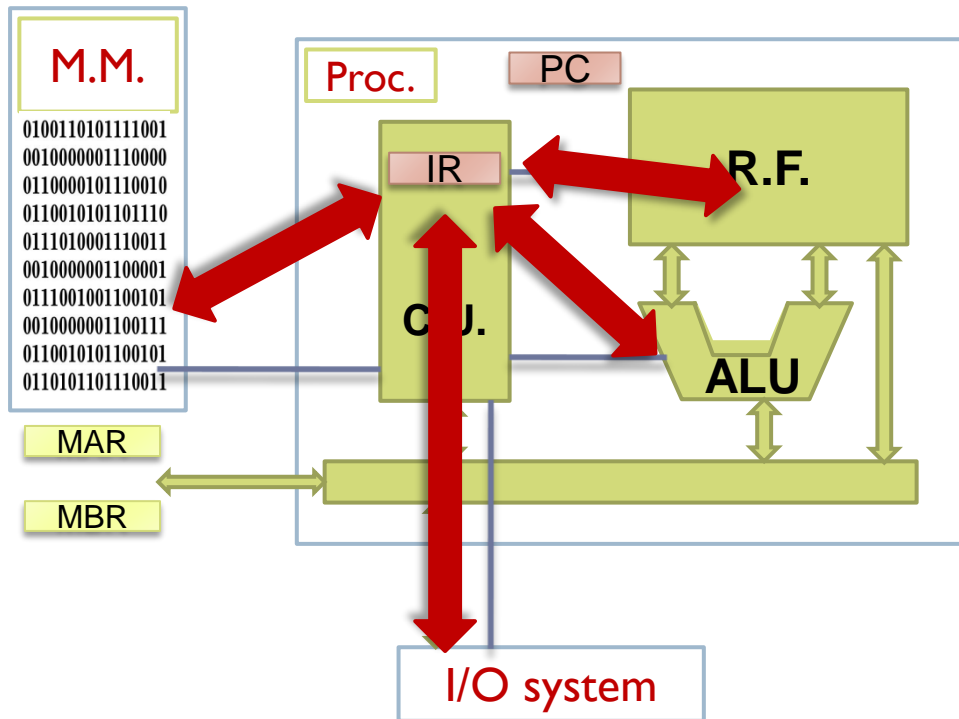
How C.U. works...



- Read from main memory the instruction pointed to by PC
- Increment PC
- **Decode instruction**
- Execute

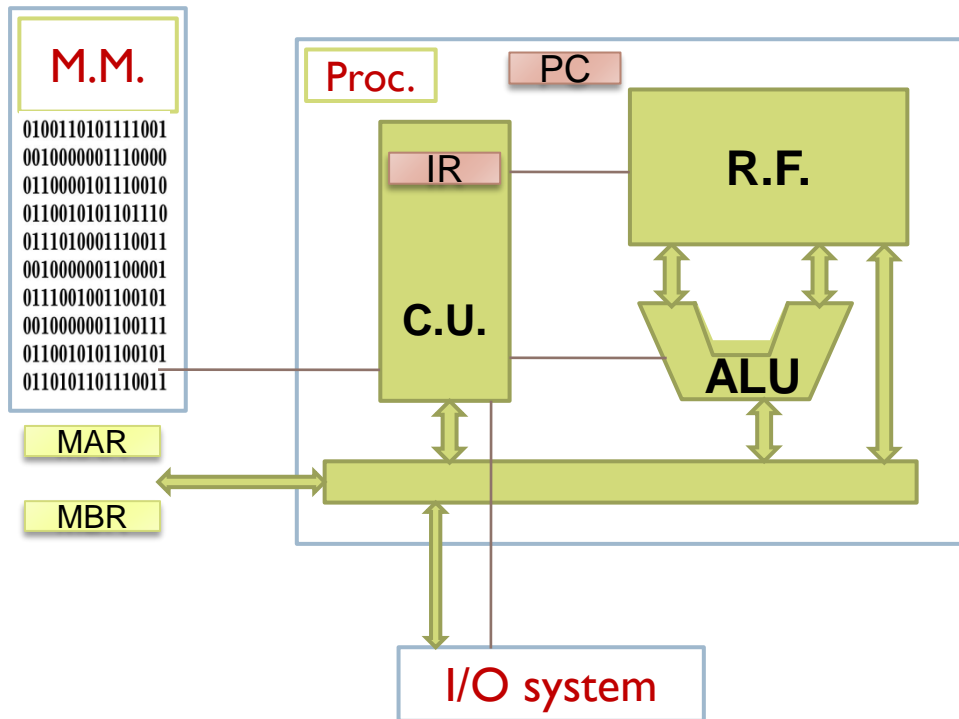
- ▶ The PC register (program counter) holds the address of the next instruction to be executed.
- ▶ The RI register (instruction register) holds the instruction is currently executed.

How C.U. works...



- Read from main memory the instruction pointed to by PC
- Increment PC
- Decode instruction
- **Execute**

Other functions of the C.U.



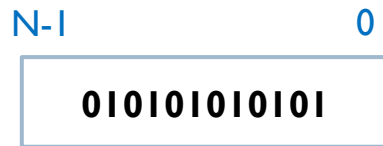
- Resolving anomalous situations
 - Illegal instructions
 - Illegal memory accesses
 - ...
- Attend to interruptions
- Control the communication with the peripherals.

Contents

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

- 1) Motivation and goals
- 2) Basic functionality of the control unit
- 3) Control signals and elemental operations
- 4) Introduction of the elemental processor

Register and bus



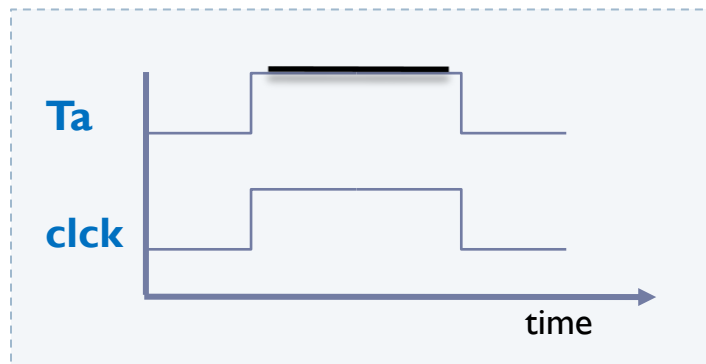
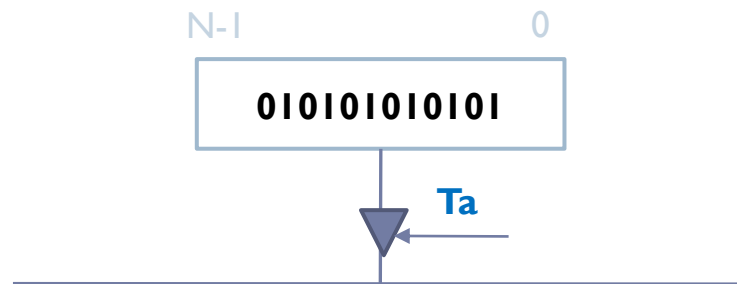
▶ Register

- ▶ Let us store a list of bits

▶ Bus

- ▶ Let us to transfer a list of bit between two elements connected though the bus

Signals: output tristate



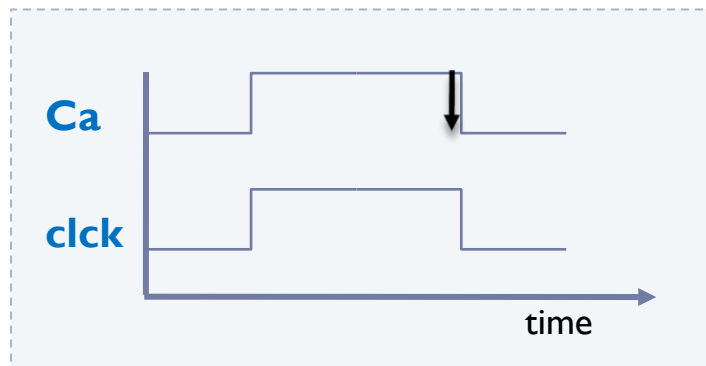
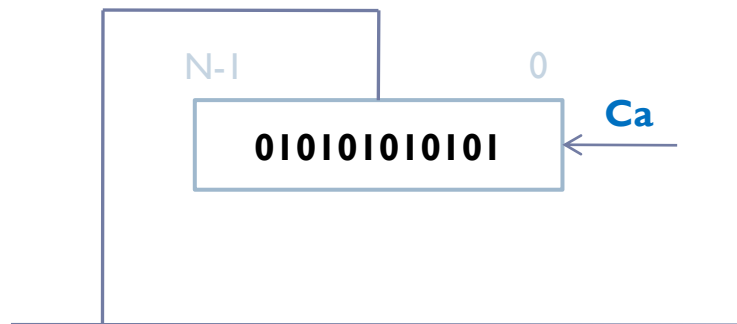
► Tri-state

- In the middle of the elements and the bus.
- Allows to send data to the bus.

► IMPORTANT

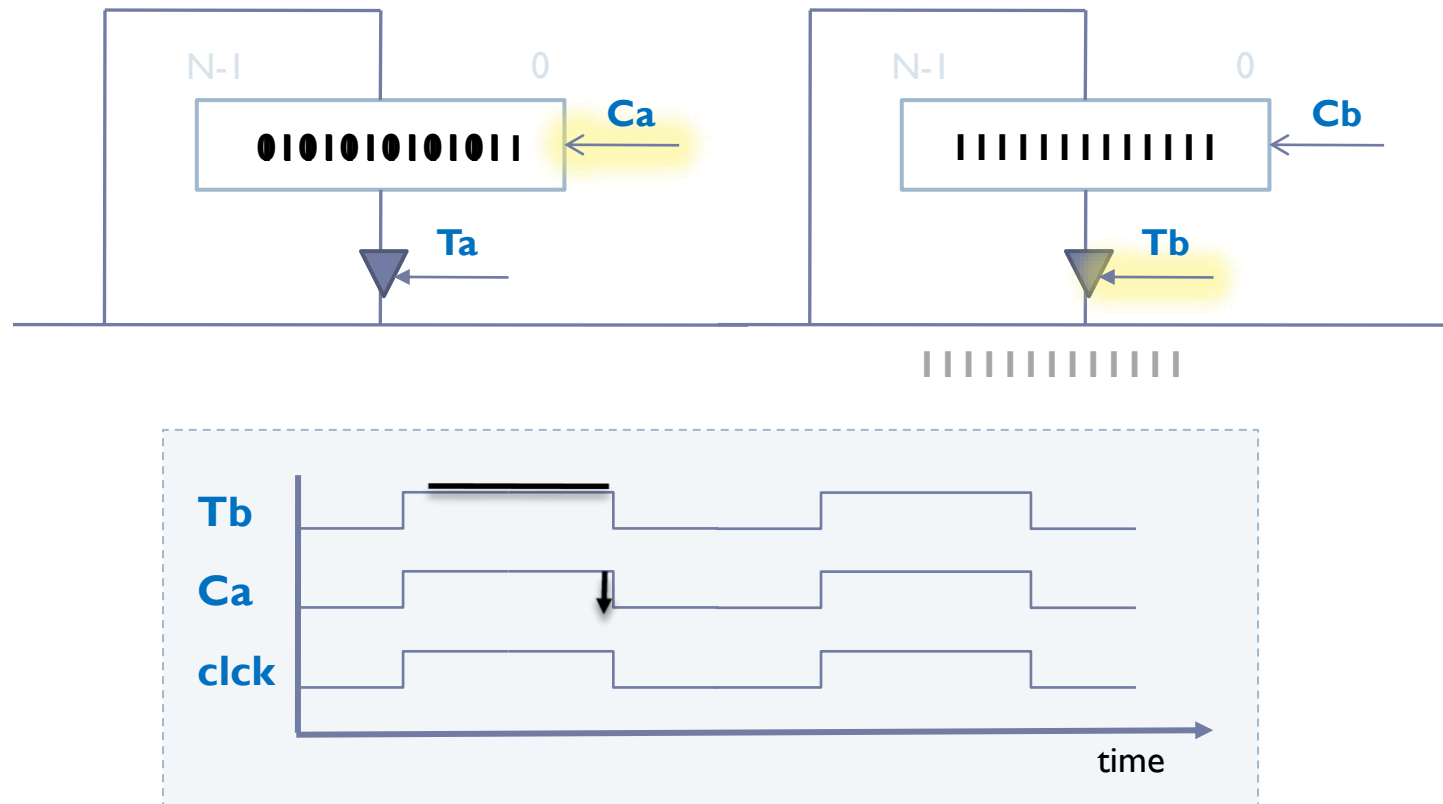
- Two or more tri-states cannot be activated on the same bus at the same time.

Signals: load in register

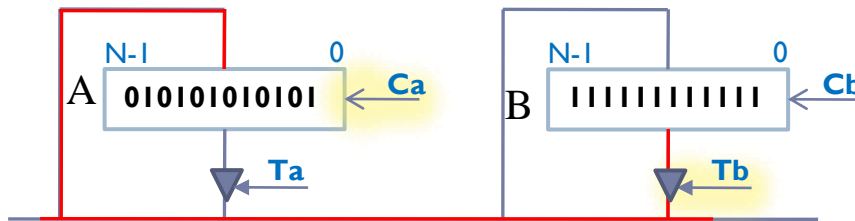


- ▶ Load in register
 - ▶ Let store the input value at the clock falling edge
 - ▶ During the clock level the register keeps the inner (old) value.
 - ▶ At the end of the clock cycle (falling edge) is when the inner value is updated
- ▶ **IMPORTANT**
 - ▶ Therefore, in the following cycle, the new value will be seen at the output

Sequence of signals



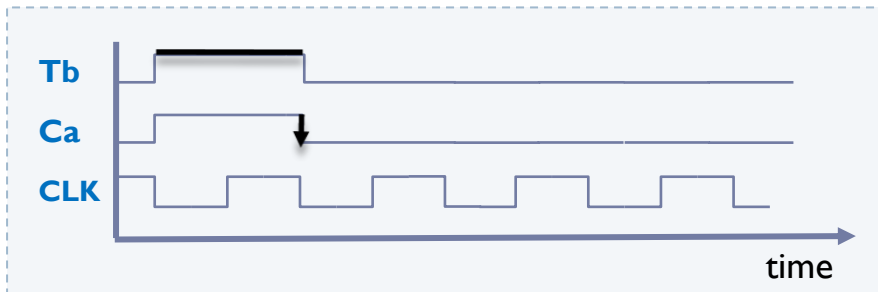
Example of *transfer* elemental operation



▶ Elementary transfer operation:

- ▶ Source storage element
- ▶ Target storage element
- ▶ A path is established

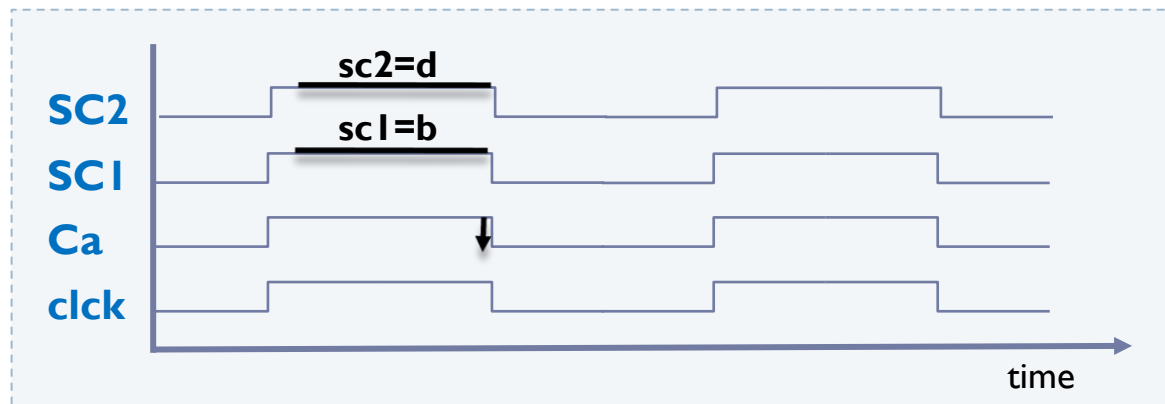
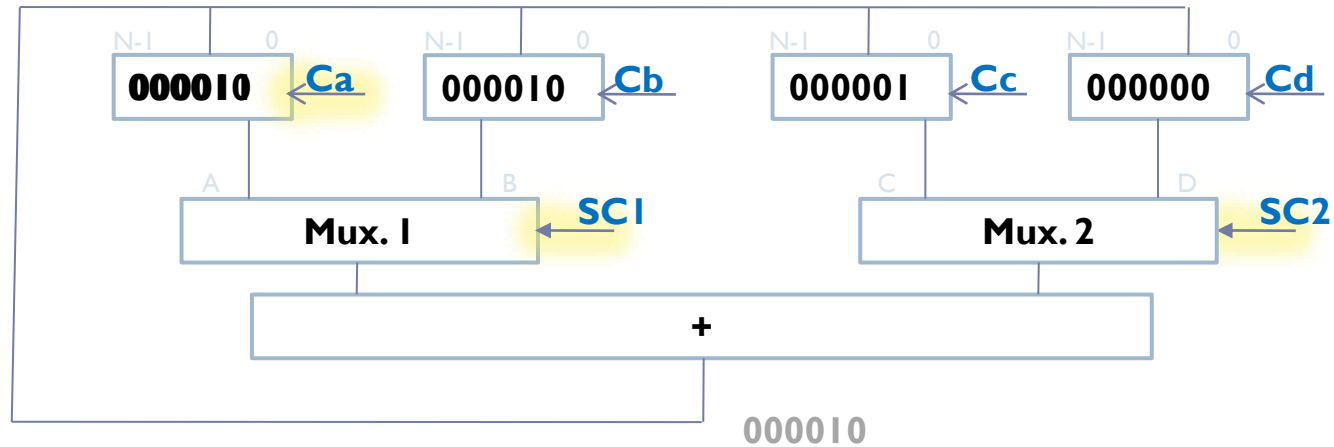
xx: $A \leftarrow B$ [T_b, C_a]



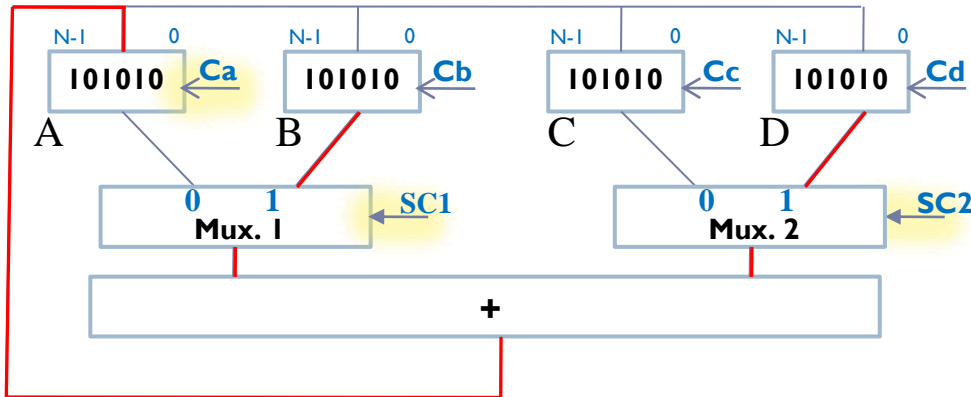
▶ IMPORTANT

- ▶ Establish the path between origin and destination in the same cycle
- ▶ In the same cycle NOT:
 - ▶ Traverse a register
 - ▶ carry two values to a bus at the same time.

Sequence of signals



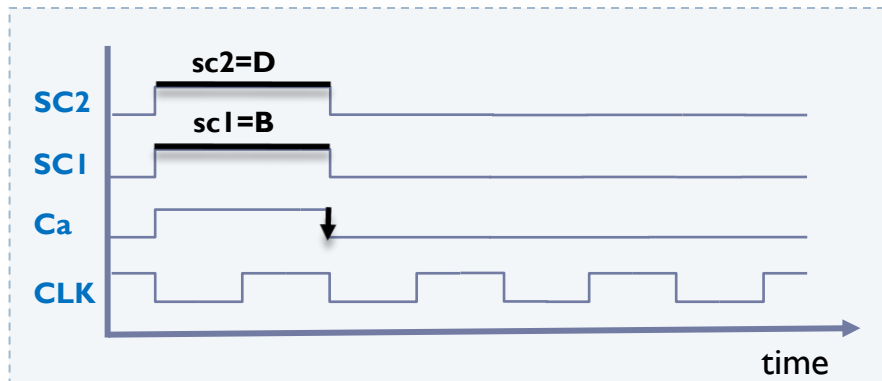
Example of *process* elemental operation



Elementary processing operation:

- ▶ Source element(s)
- ▶ Target element
- ▶ Transformation operation on the path

yy: $A \leftarrow B + D$ [SC1=b, SC2=d, Ca]



IMPORTANT

- ▶ Establish the path between origin and destination in the same cycle
- ▶ In the same cycle NOT:
 - ▶ Traverse a register
 - ▶ carry two values to a bus at the same time.

RT Language and Elementary Operations

- ▶ RT Language:

- ▶ Register transfer level language.
- ▶ It specifies what happens in the computer by elementary operations.

- ▶ Elementary operations:

- ▶ Transfer operations

- ▶ $MAR \leftarrow PC$



$Reg \leftarrow Reg$

- ▶ Processing operations

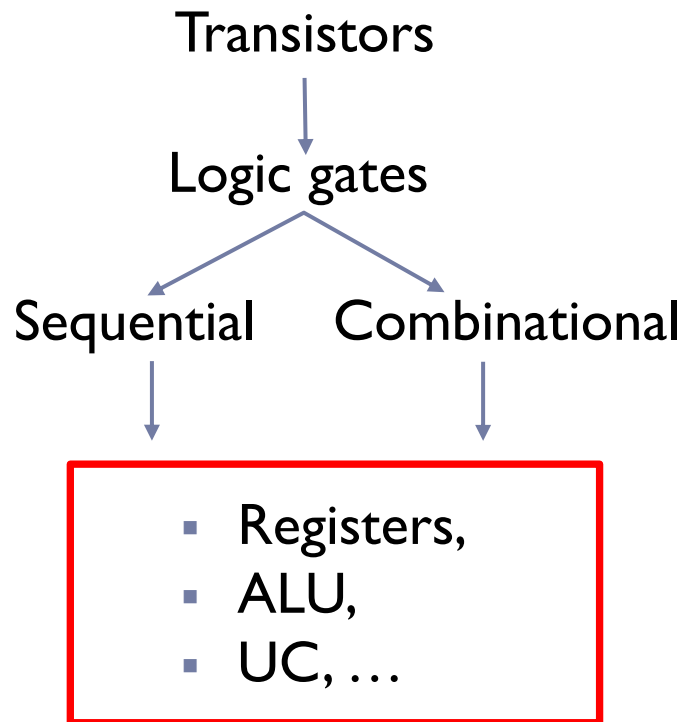
- ▶ $R1 \leftarrow R2 + R2$



$Reg \leftarrow \varphi(Reg, Reg)$

Review all components...

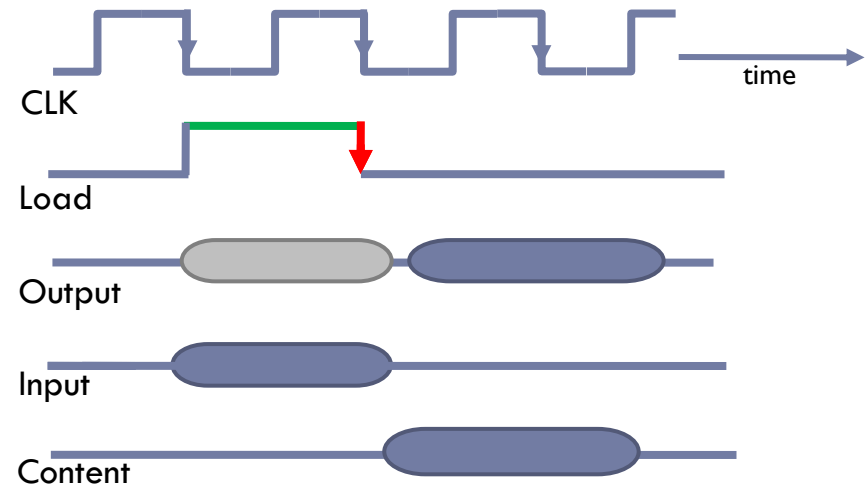
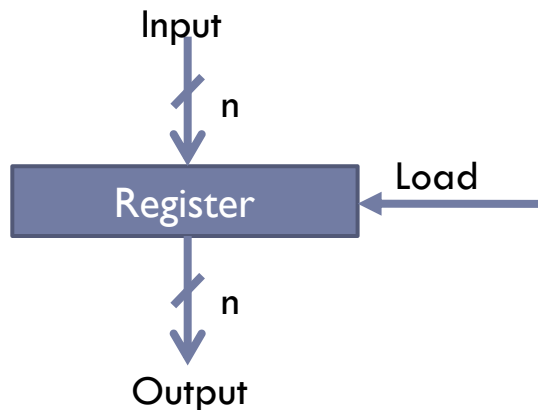
- ▶ Binary system based on 0 y 1
- ▶ Building blocks:



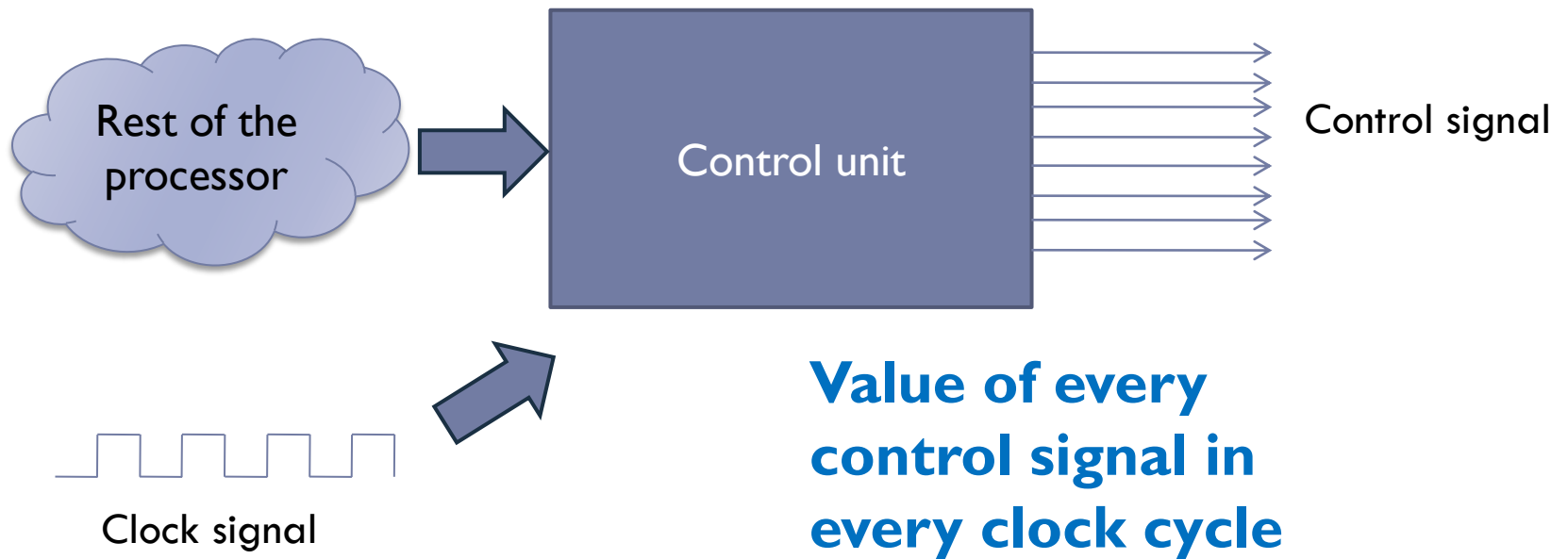
Review all components...

Registers

- ▶ Element storing n bits at a time
 - ▶ Output: 1
 - ▶ During **the level**, the output is the value stored in the register.
 - ▶ Input: 1
 - ▶ Possible new value to be stored
 - ▶ Control: 1 or 2
 - ▶ Load: in the **falling edge** the possible new value is stored
 - ▶ Reset: there may be a signal to set the register to zero



Control Unit (UC)



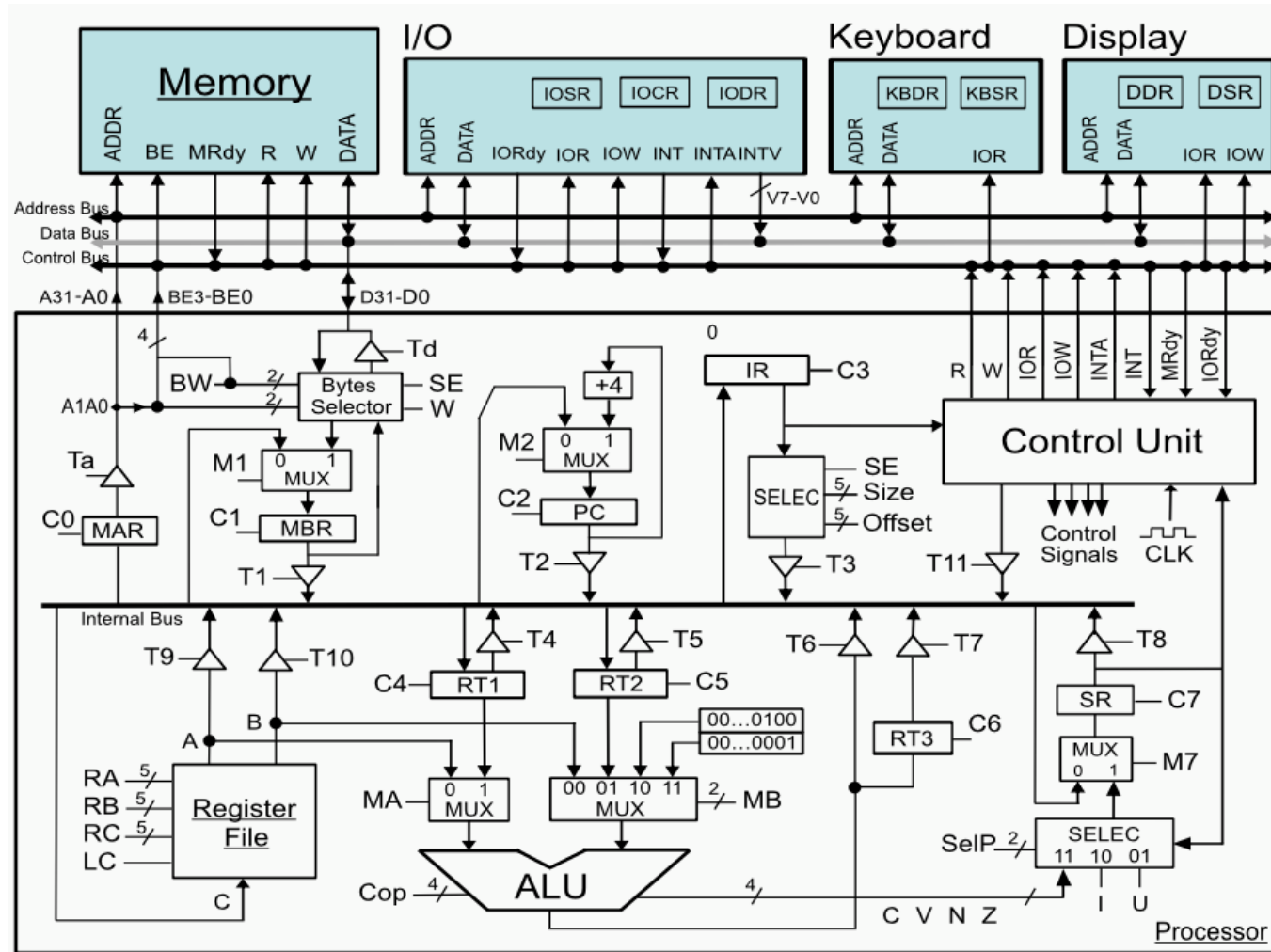
Contents

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

- 1) Motivation and goals
- 2) Basic functionality of the control unit
- 3) Control signals and elemental operations
- 4) Introduction of the elemental processor

Structure of an elementary computer and WepSIM Simulator

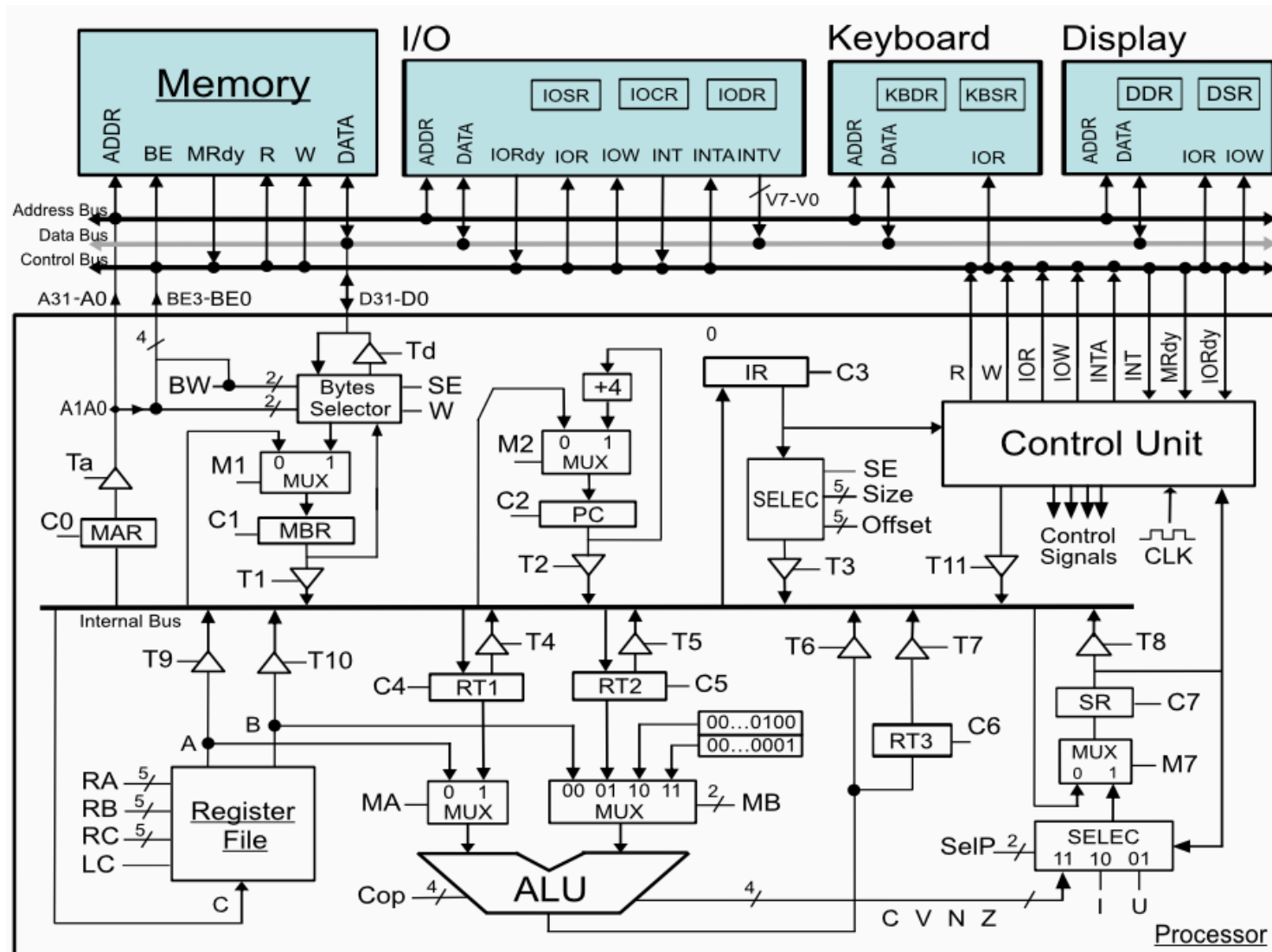
<https://wepsim.github.io/wepsim/>



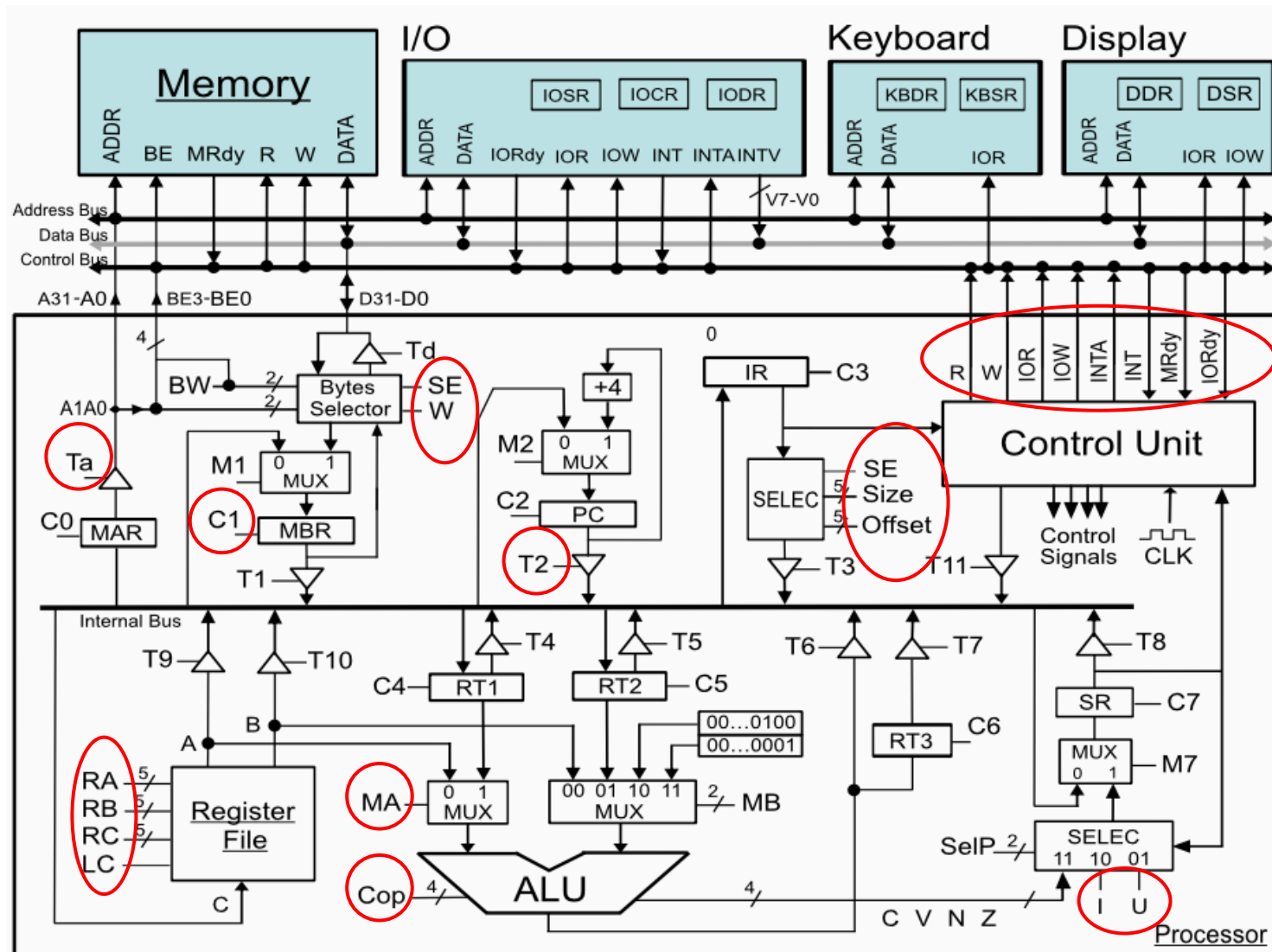
Main features

- ▶ Main features of the elemental processor (EP)
 - ▶ 32 bits computer
 - ▶ Main memory:
 - ▶ Addressed by bytes
 - ▶ A clock cycle for reading and writing operations
 - ▶ Different types of registers available:
 - ▶ Register file of 32 registers visible to programmers (R0...R31)
 - Similar to MIPS: R0 = 0 y SP = R29
 - ▶ Registers not visible to programmers (RT1, RT2 and RT3)
 - Possible use for intermediate calculations within an instruction
 - ▶ Control registers (PC, IR, MAR, MBR) and state register (SR)
 - MAR, MBR, PC, SR, IR
- ▶ WepSIM simulates the E.P.:
 - ▶ <https://wepsim.github.io/wepsim/>

Structure of an elementary computer

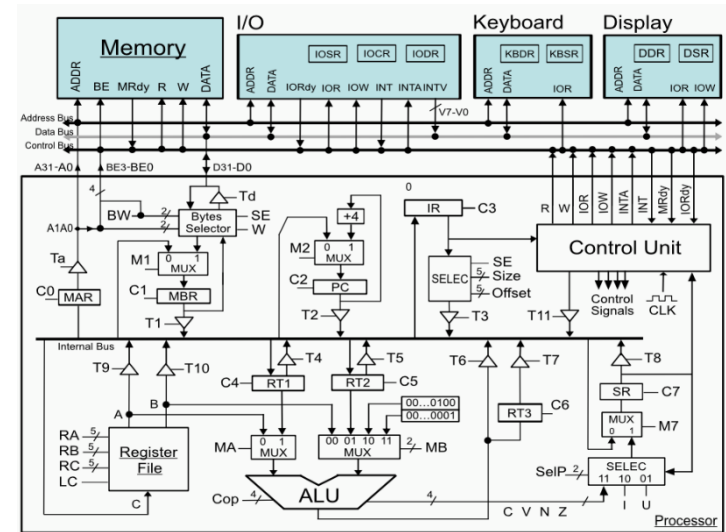


Control signals



Control signals

- ▶ Memory access signals
- ▶ Load signals in registers
- ▶ Tri-state gate control signals
- ▶ MUX selection signals
- ▶ Register file control signals
- ▶ Other selection signals

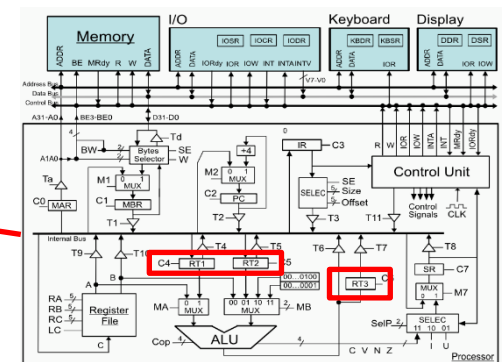
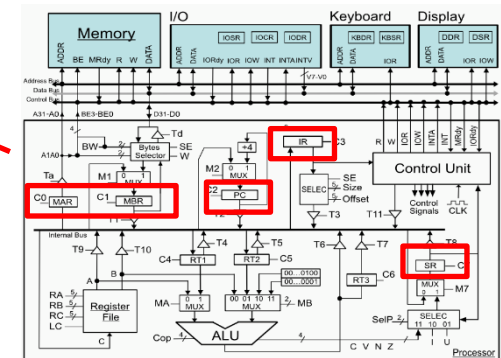
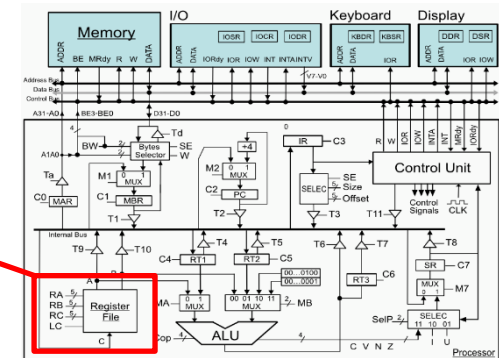


General nomenclature:

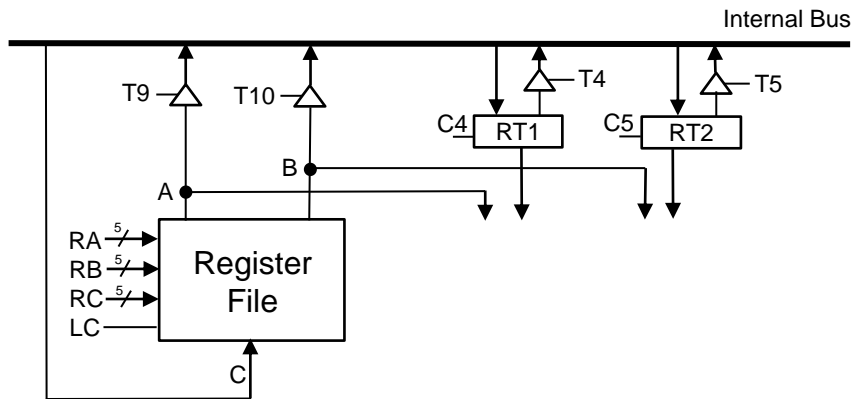
- Mx: Selection in multiplexor
- Tx: Tri-state activation signal
- Cx: Register load signal
- Ry: Register file selection

Registers

- ▶ Registers visible to programmers
 - ▶ Register file's registers (~MIPS: \$t0, \$t1, etc.)
- ▶ Control and status registers:
 - ▶ PC: program counter
 - ▶ IR: instruction register
 - ▶ SP: stack pointer (in the register file)
 - ▶ MAR: memory address register
 - ▶ MBR: memory data register
 - ▶ SR: status register
- ▶ Registers not visible to the user:
 - ▶ RT1, RT2 and RT3 (internal temporal reg.)



Control signals



Nomenclature:

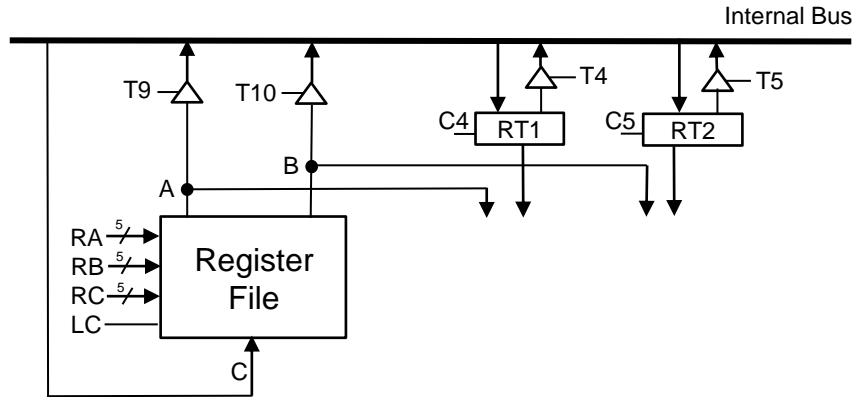
- **R_y**: Register file selection
- **M_x**: Selection in multiplexer
- **T_x**: Tri-state activation signal
- **C_x**: Register load signal

▶ Register file, RT1 and RT2

- ▶ RA – register output by A
- ▶ RB – register output by B
- ▶ RC – input C to the RC register
- ▶ LC – activates writing for RC
- ▶ T9 - copy A to the internal bus
- ▶ T10 - copy B to the internal bus
- ▶ C4 - from the internal bus to RT1
- ▶ T4 - RT1 output to internal bus
- ▶ C5 - from the internal bus to RT2
- ▶ T5 - RT2 output to internal bus

Example

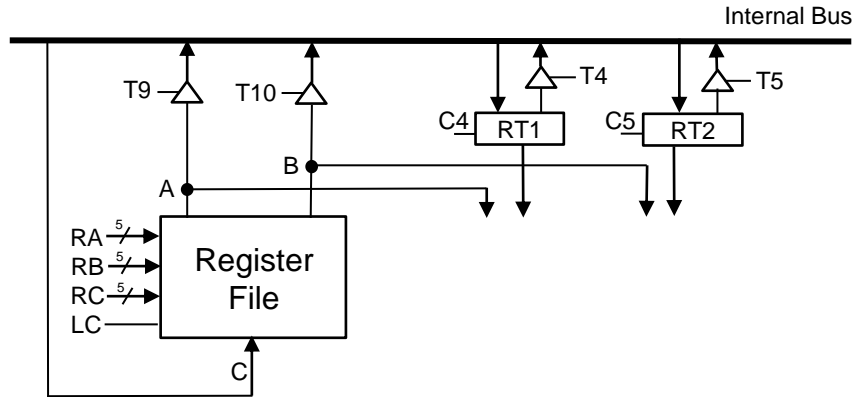
elemental operations in registers



► **SWAP R1 R2**

Example

elemental operations in registers

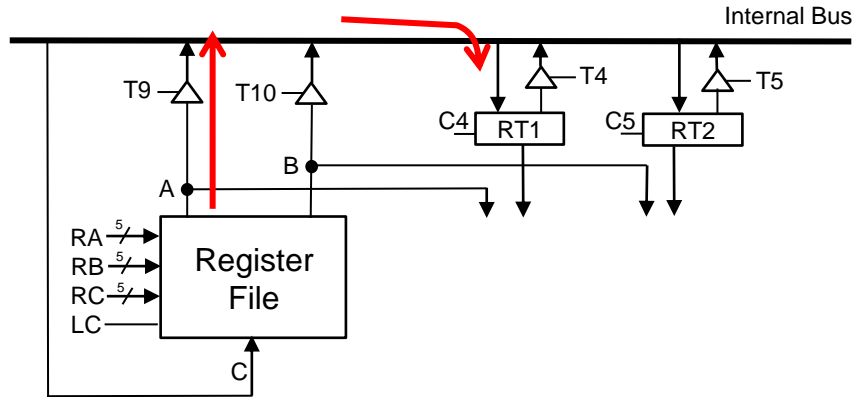


► **SWAP R1 R2**

Elemental Op.	Signals

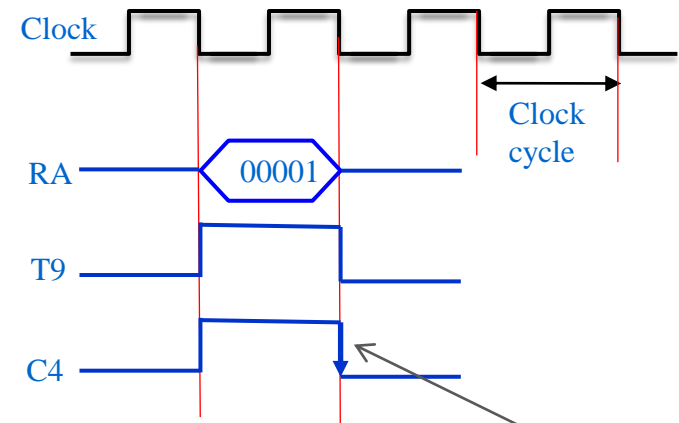
Example

elemental operations in registers



► SWAP R1 R2

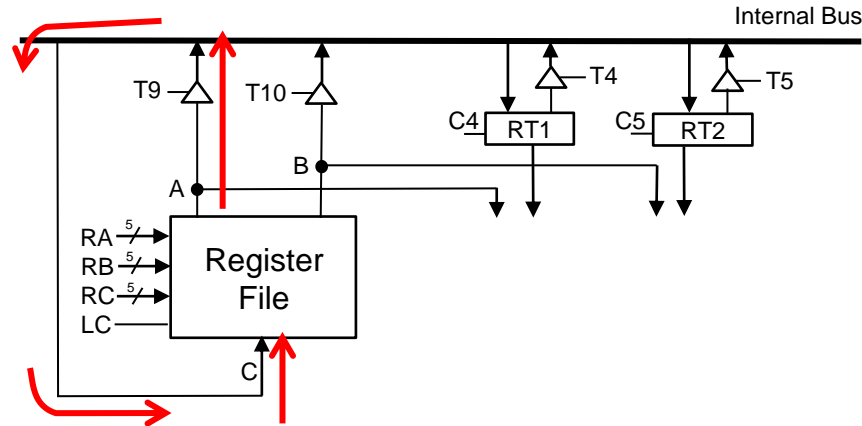
Elemental Op.	Signals
$RT1 \leftarrow R1$	$RA=00001, T9, C4$



The data is loaded on RT1 on the falling edge.
It will be available on RT1 during the **next** cycle.

Example

elemental operations in registers

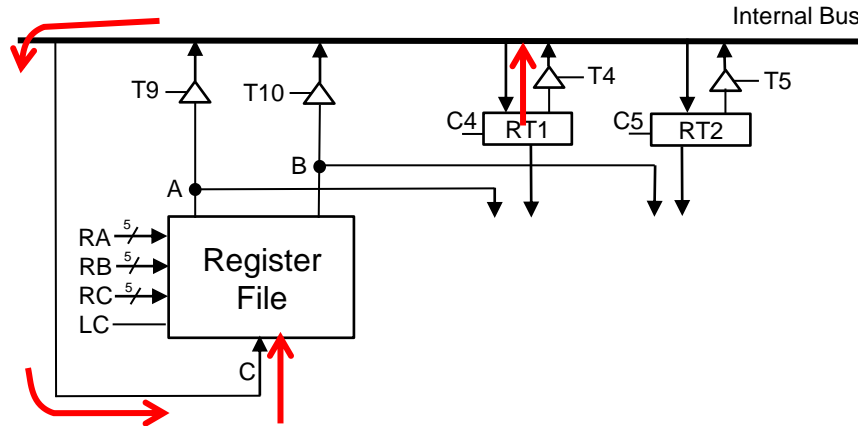


► **SWAP R1 R2**

Elemental Op.	Signals
$RT1 \leftarrow R1$	RA=00001, T9, C4
$R1 \leftarrow R2$	RA=2 (00010), T9, RC=1, LC

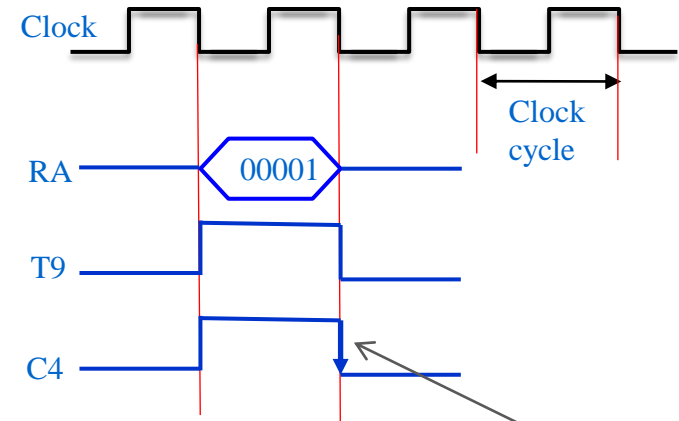
Example

elemental operations in registers



► SWAP R1 R2

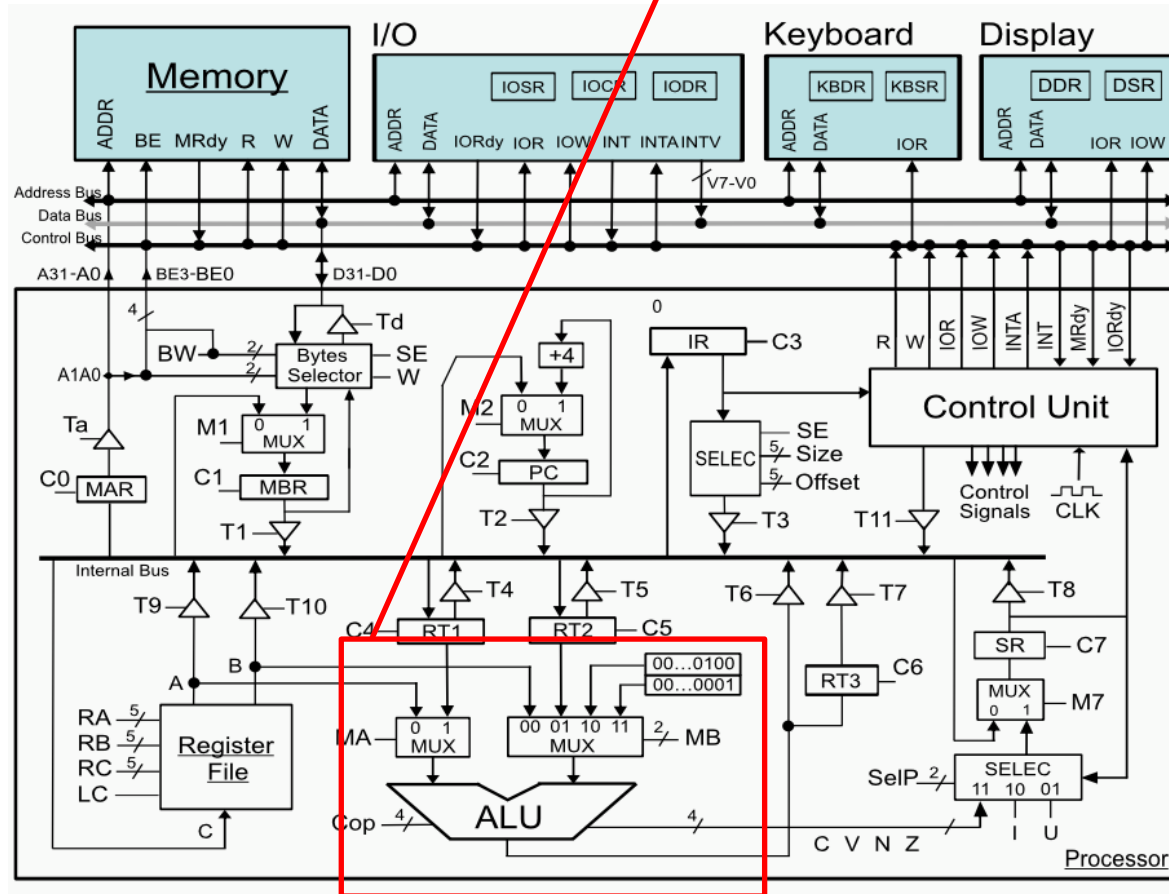
Elemental Op.	Signals
$RT1 \leftarrow R1$	RA=00001, T9, C4
$R1 \leftarrow R2$	RA=2 (00010), T9, RC=1, LC
$R2 \leftarrow RT1$	T4, RC=2 (00010), LC



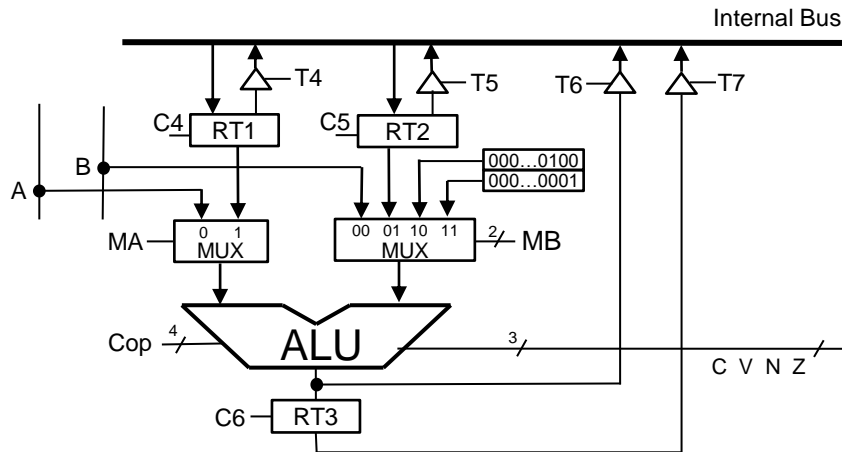
The data is loaded on RT1 on the falling edge.
It will be available on RT1 during the **next** cycle.

Structure of an elementary computer

arithmetic logic unit (ALU)



Control Signals

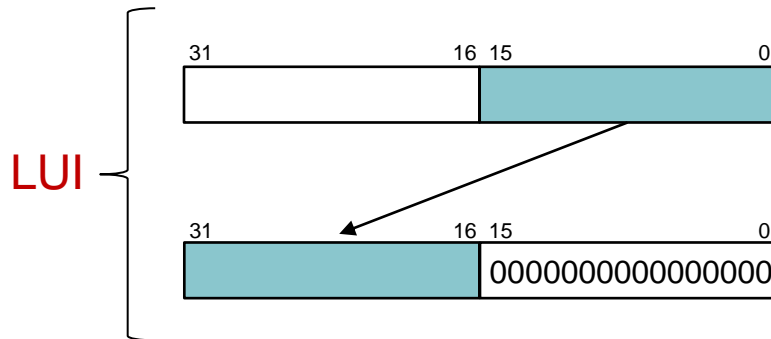
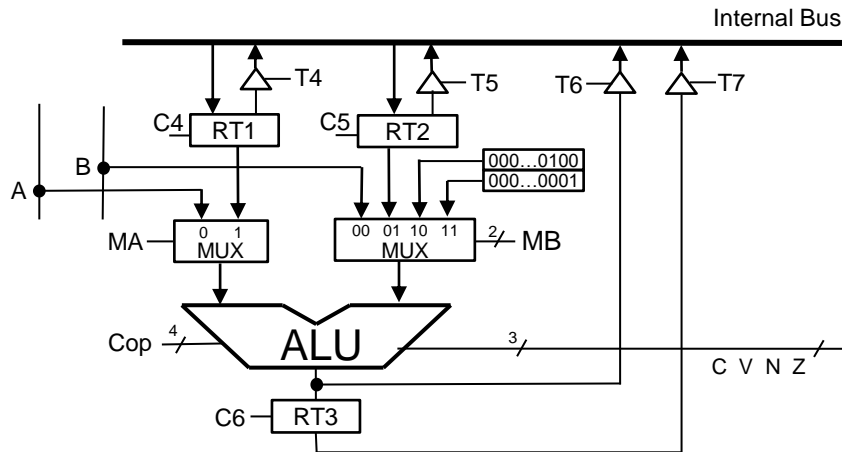


▶ ALU

- ▶ MA - selection of operand A
- ▶ MB - selection of operand B
- ▶ Cop - operation code

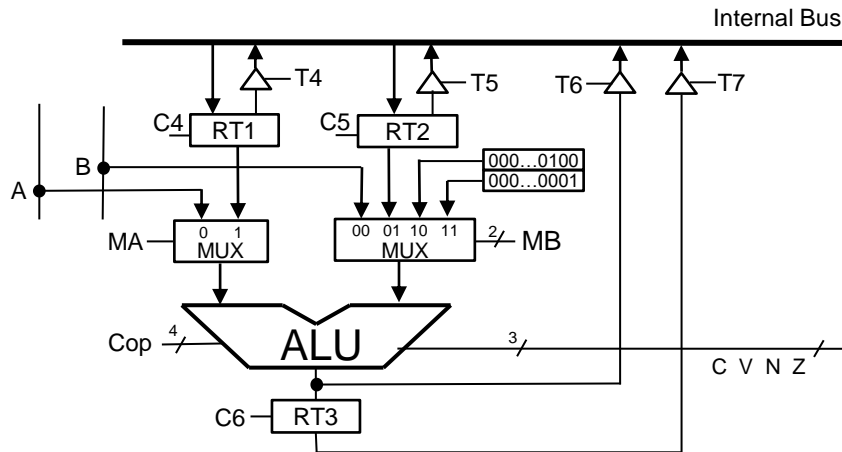
Cop (Cop ₃ -Cop ₀)	Operation
0000	NOP
0001	A and B
0010	A or B
0011	not (A)
0100	A xor B
0101	Shift Right Logical (A) B= number of bits to shift
0110	Shift Right Arithmetic (A) B= number of bits to shift
0111	Shift left (A) B= number of bits to shift
1000	Rotate Right (A) B= number of bits to rotate
1001	Rotate Left (A) B= number of bits to rotate
1010	A + B
1011	A - B
1100	A * B (with overflow)
1101	A / B (integer division)
1110	A % B (integer division)
1111	LUI (A)

Control Signals



Cop (Cop ₃ -Cop ₀)	Operation
0000	NOP
0001	A and B
0010	A or B
0011	not (A)
0100	A xor B
0101	Shift Right Logical (A) B= number of bits to shift
0110	Shift Right Arithmetic (A) B= number of bits to shift
0111	Shift left (A) B= number of bits to shift
1000	Rotate Right (A) B= number of bits to rotate
1001	Rotate Left (A) B= number of bits to rotate
1010	A + B
1011	A - B
1100	A * B (with overflow)
1101	A / B (integer division)
1110	A % B (integer division)
1111	LUI (A)

Control Signals

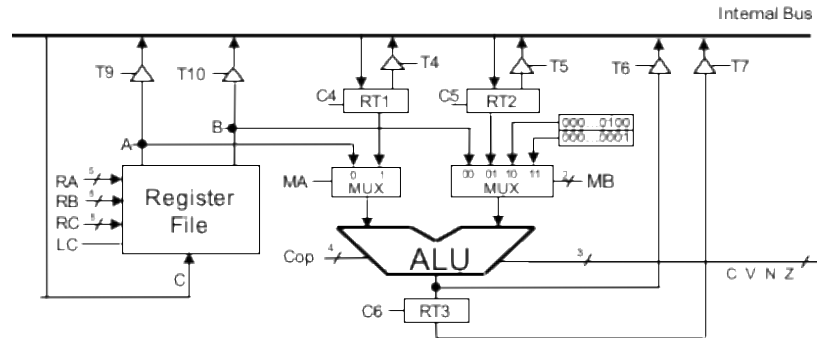


Result	C	V	N	Z
Positive result (0 is considered +)	0	0	0	0
Result == 0	0	0	0	1
Negative result	0	0	1	0
Overflow	0	1	0	0
Division by zero	0	1	0	1
Carrying at bit 32	1	0	0	0

Cop (Cop ₃ -Cop ₀)	Operation
0000	NOP
0001	A and B
0010	A or B
0011	not (A)
0100	A xor B
0101	Shift Right Logical (A) B= number of bits to shift
0110	Shift Right Arithmetic (A) B= number of bits to shift
0111	Shift left (A) B= number of bits to shift
1000	Rotate Right (A) B= number of bits to rotate
1001	Rotate Left (A) B= number of bits to rotate
1010	A + B
1011	A - B
1100	A * B (with overflow)
1101	A / B (integer division)
1110	A % B (integer division)
1111	LUI (A)

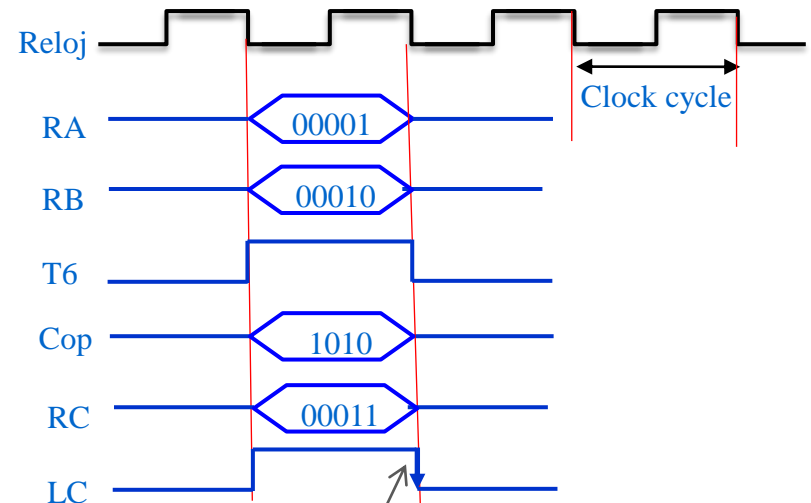
Example

elemental operations in ALU



► ADD R3 R1 R2

Elem. Op.	Signals
$R3 \leftarrow R1 + R2$	RA=R1, RB=R2, Cop=+, T6, RC=R3, LC=1



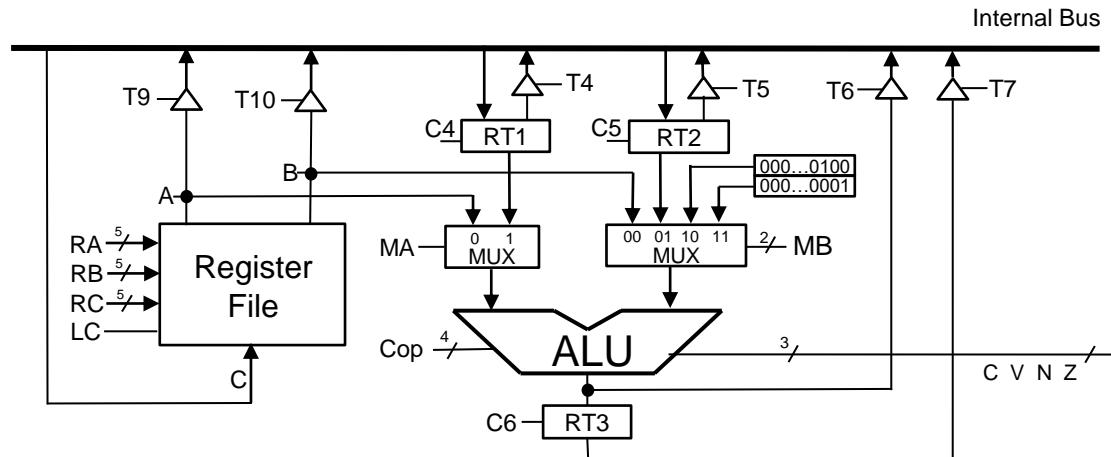
Rest of signals at 0.

The load is performed on R3 on the falling edge.

The data is available in register R3 for the next cycle.

Example

elemental operations in ALU



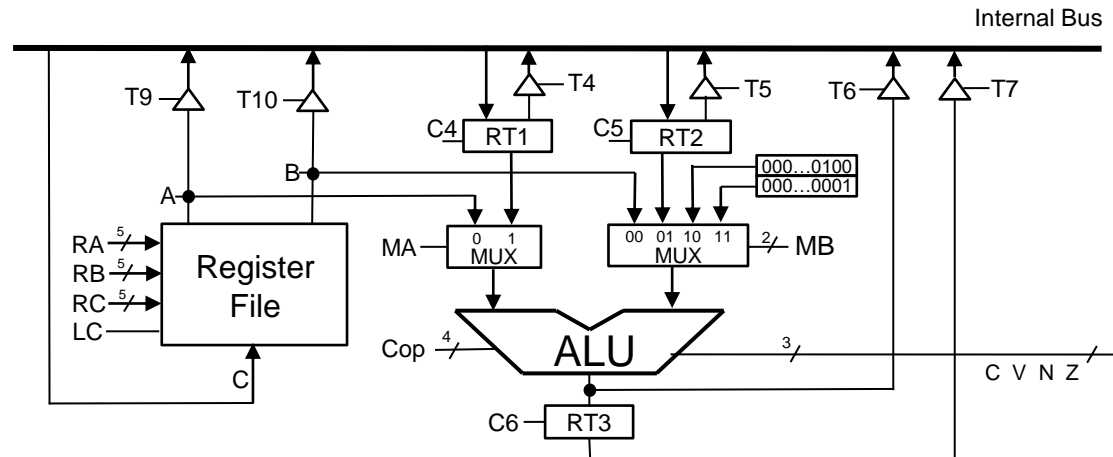
► SWAP R1 R2

Elem. Op.	Signals
$RT1 \leftarrow R1$	RA=1, T9, C4
$R1 \leftarrow R2$	RA=2, T9, RC=1, LC
$R2 \leftarrow RT1$	T4, RC=2, LC

► SWAP R1, R2 without R_{tmp}

Example

elemental operations in ALU



► SWAP R1 R2

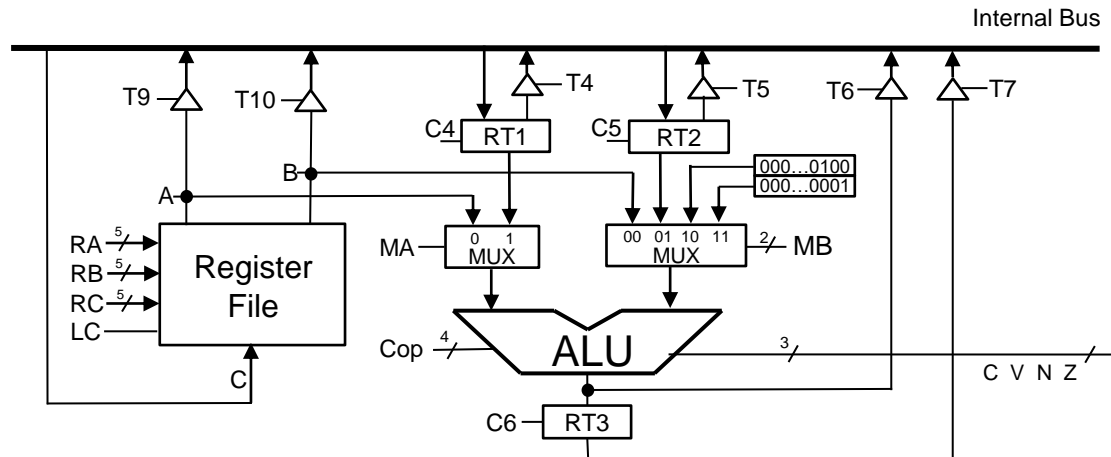
Elem. Op.	Signals
$RT1 \leftarrow R1$	RA=1, T9, C4
$R1 \leftarrow R2$	RA=2, T9, RC=1, LC
$R2 \leftarrow RT1$	T4, RC=2, LC

► SWAP R1, R2 without R_{tmp}

Elem. Op.	
$R1 \leftarrow R1 \wedge R2$	$R1 \leftarrow (R1 \wedge R2)$
$R2 \leftarrow R1 \wedge R2$	$R2 \leftarrow (R1 \wedge R2) \wedge R2$
$R1 \leftarrow R1 \wedge R2$	$R1 \leftarrow (R1 \wedge R2) \wedge R1$

Example

elemental operations in ALU



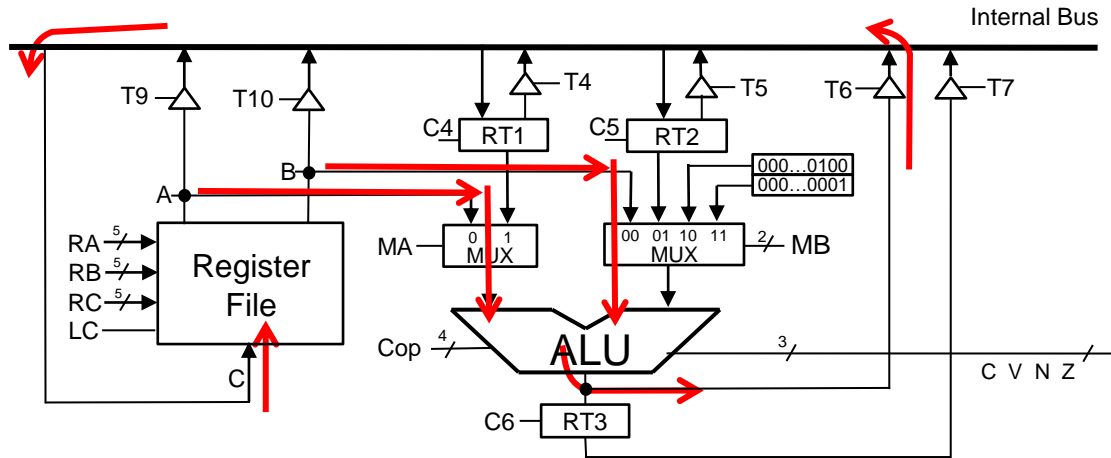
► SWAP R1 R2

Elem. Op.	Signals
$RT1 \leftarrow R1$	RA=1, T9, C4
$R1 \leftarrow R2$	RA=2, T9, RC=1, LC
$R2 \leftarrow RT1$	T4, RC=2, LC

► SWAP R1, R2 without R_{tmp}

Elem. Op.	Signals
$R1 \leftarrow R1 \wedge R2$	RA=1, RB=2, Cop= \wedge , T6, RC=1, LC
$R2 \leftarrow R1 \wedge R2$	RA=1, RB=2, Cop= \wedge , T6, RC=2, LC
$R1 \leftarrow R1 \wedge R2$	RA=1, RB=2, Cop= \wedge , T6, RC=1, LC

elemental operations in ALU



► **SWAP R1 R2**

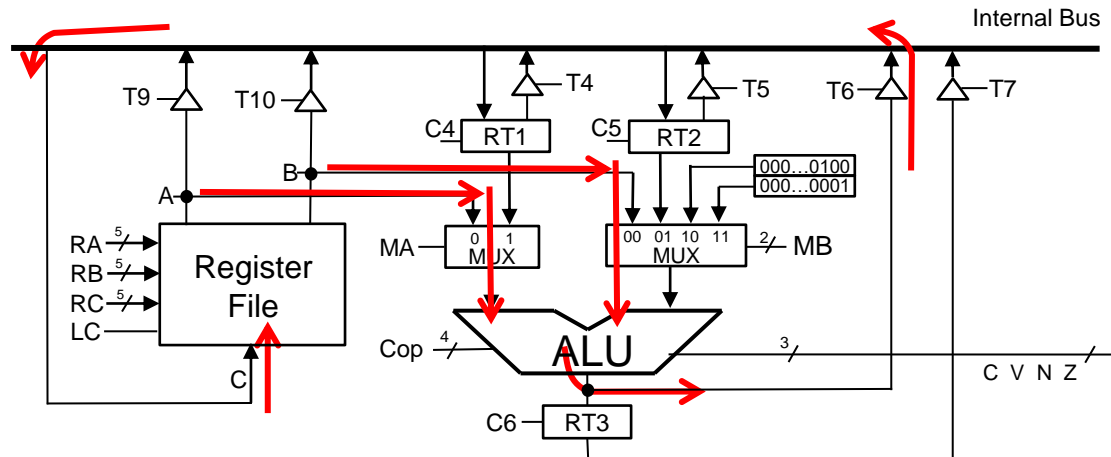
Elem. Op.	Signals
$RT1 \leftarrow R1$	RA=1, T9, C4
$R1 \leftarrow R2$	RA=2, T9, RC=1, LC
$R2 \leftarrow RT1$	T4, RC=2, LC

► SWAP R1, R2 without R_{tmp}

Elem. Op.	Signals
$R1 \leftarrow R1 \wedge R2$	RA=1, RB=2, Cop= \wedge , T6, RC=1, LC
$R2 \leftarrow R1 \wedge R2$	RA=1, RB=2, Cop= \wedge , T6, RC=2, LC
$R1 \leftarrow R1 \wedge R2$	RA=1, RB=2, Cop= \wedge , T6, RC=1, LC

Example

elemental operations in ALU



► SWAP R1 R2

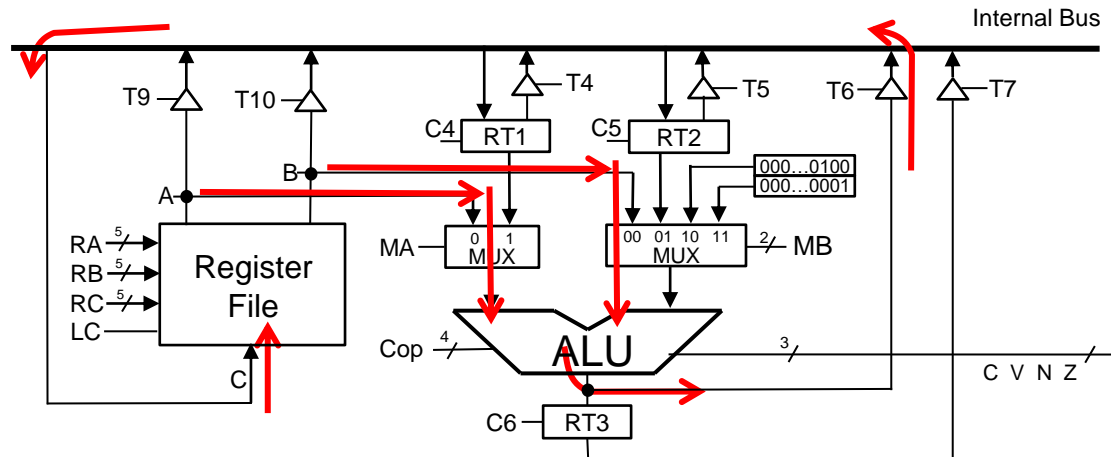
Elem. Op.	Signals
$RT1 \leftarrow R1$	RA=1, T9, C4
$R1 \leftarrow R2$	RA=2, T9, RC=1, LC
$R2 \leftarrow RT1$	T4, RC=2, LC

► SWAP R1, R2 without R_{tmp}

Elem. Op.	Signals
$R1 \leftarrow R1 \wedge R2$	RA=1, RB=2, Cop=^, T6, RC=1, LC
$R2 \leftarrow R1 \wedge R2$	RA=1, RB=2, Cop=^, T6, RC=2, LC
$R1 \leftarrow R1 \wedge R2$	RA=1, RB=2, Cop=^, T6, RC=1, LC

Example

elemental operations in ALU



► SWAP R1 R2

Elem. Op.	Signals
$RT1 \leftarrow R1$	RA=1, T9, C4
$R1 \leftarrow R2$	RA=2, T9, RC=1, LC
$R2 \leftarrow RT1$	T4, RC=2, LC

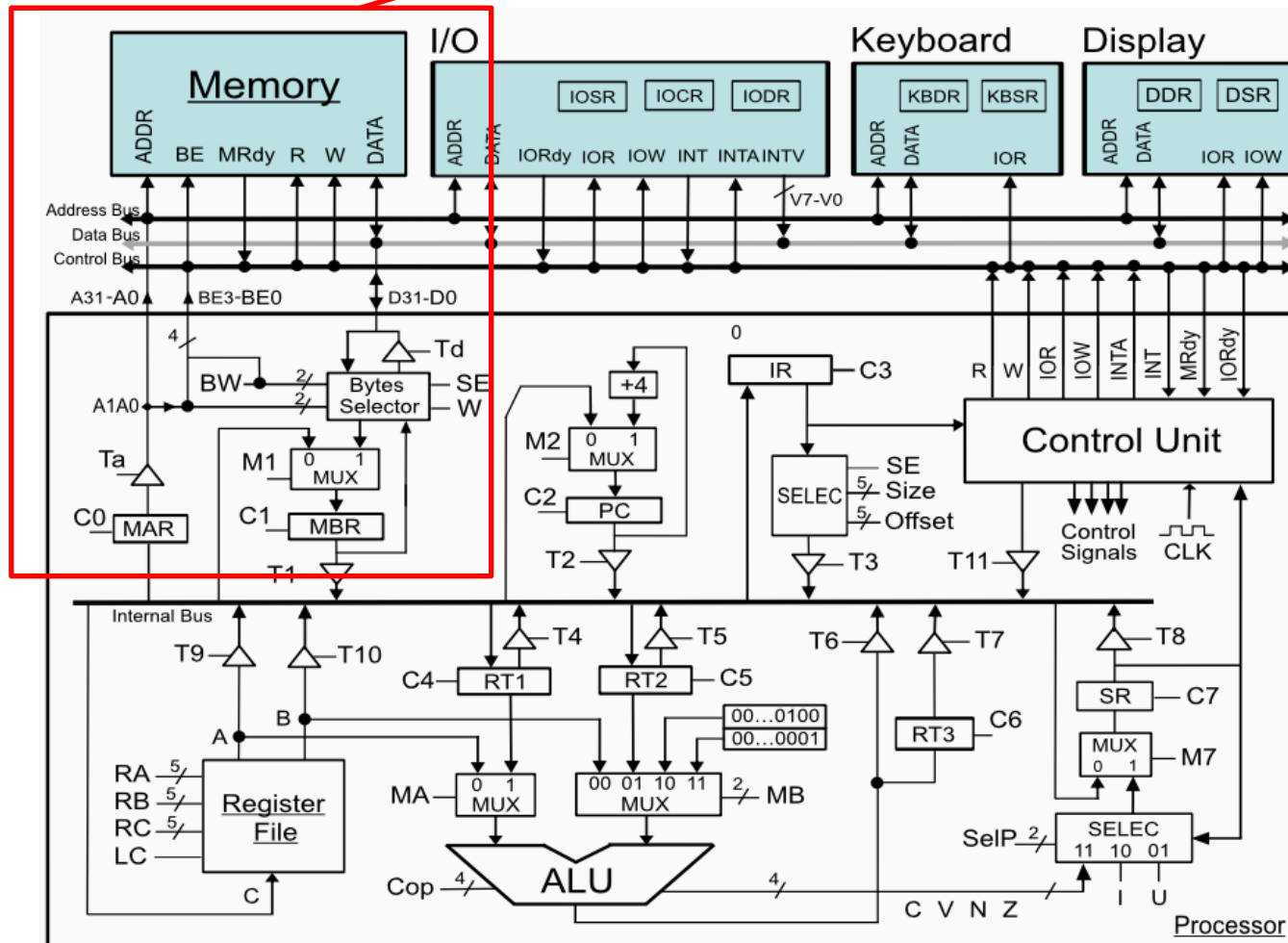
► SWAP R1, R2 without R_{tmp}

Elem. Op.	Signals
$R1 \leftarrow R1 \wedge R2$	RA=1, RB=2, Cop=^, T6, RC=1, LC
$R2 \leftarrow R1 \wedge R2$	RA=1, RB=2, Cop=^, T6, RC=2, LC
$R1 \leftarrow R1 \wedge R2$	RA=1, RB=2, Cop=^, T6, RC=1, LC

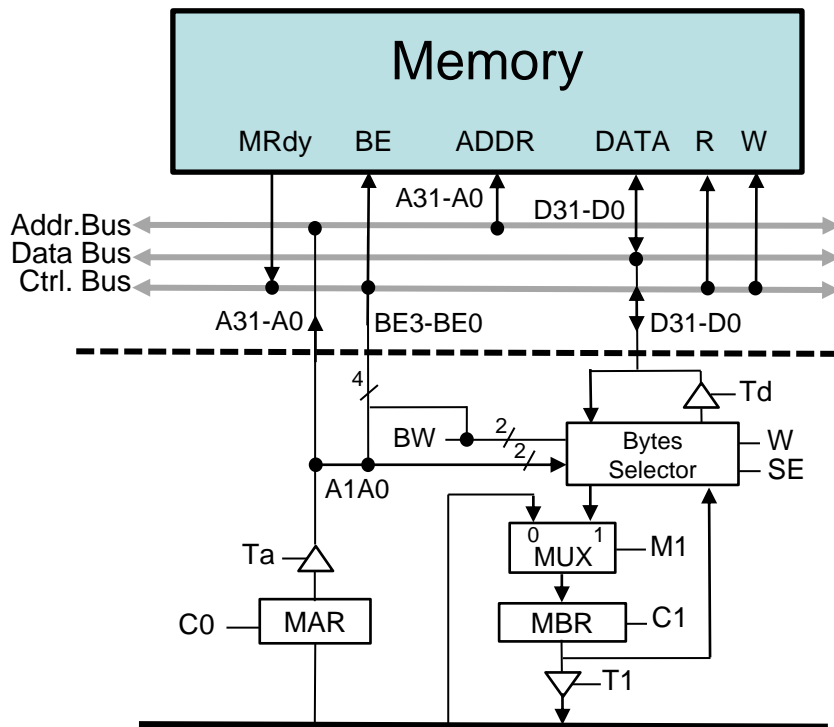
Structure of an elementary computer

Main memory,

address register and data register



Control Signals



Nomenclature:

- MAR -> Address register
- MBR -> Data register

► Main Memory

- R – Read
- W – Write
- $BE3-BE0 = A1A0 + BW$
 - Access size (byte, word, half word)
- C0 – from internal bus to MAR
- C1 – from data bus to MBR
- Ta - output of MAR to the address bus
- Td - MBR output to data bus
- T1 - MBR output to internal bus
- M1 - selection for MBR: memory or internal bus

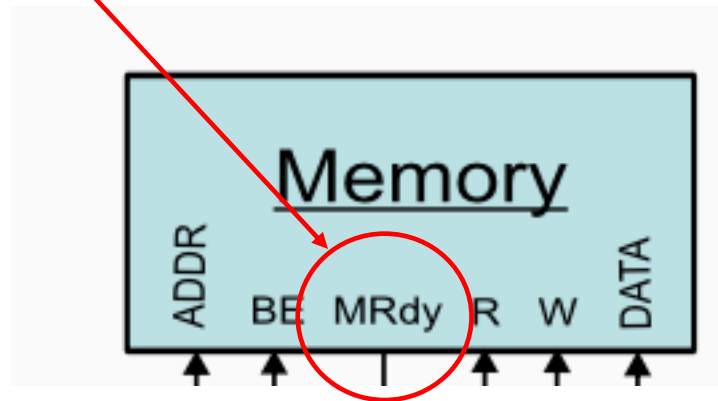
Memory access

- ▶ **Synchronous:**

- ▶ Memory requires a certain number of cycles for all operations.

- ▶ **Asynchronous:**

- ▶ Non fixed number of clock cycles for memory operations.
 - ▶ The memory indicates when the operation ends



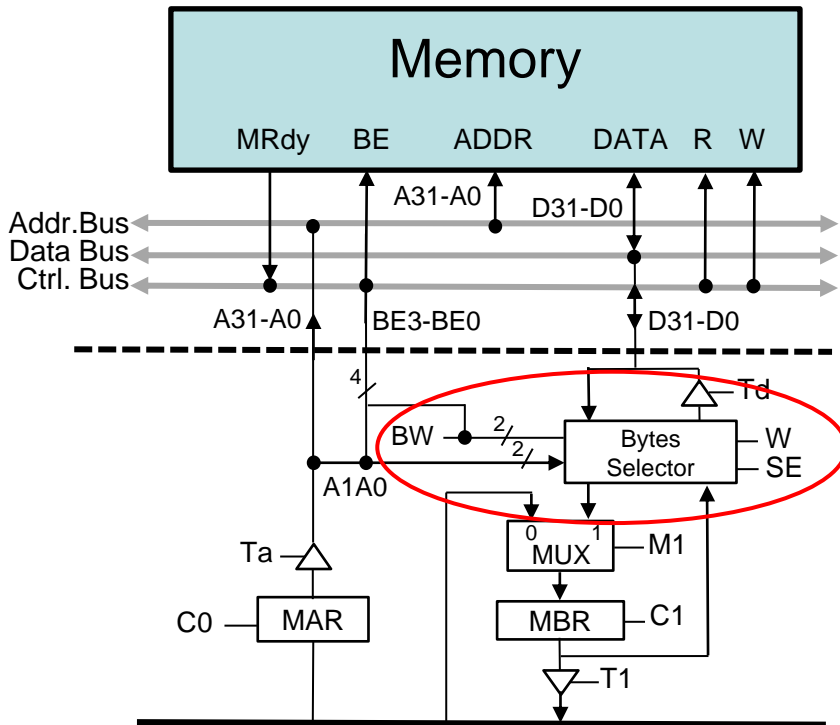
BE (Byte-Enable) signals for reading

Bytes in memory				Bytes selection				Output to bus			
D31-D24	D23-D16	D15-D8	D7-D0	BE3	BE2	BE1	BE0	D31-D24	D23-D16	D15-D8	D7-D0
Byte 3	Byte 2	Byte 1	Byte 0	0	0	0	0	---	---	---	Byte 0
Byte 3	Byte 2	Byte 1	Byte 0	0	0	0	1	---	---	Byte 1	---
Byte 3	Byte 2	Byte 1	Byte 0	0	0	1	0	--	Byte 2	---	---
Byte 3	Byte 2	Byte 1	Byte 0	0	0	1	1	Byte 3	---	---	---
Byte 3	Byte 2	Byte 1	Byte 0	0	1	0	X	---	---	Byte 1	Byte 0
Byte 3	Byte 2	Byte 1	Byte 0	0	1	1	X	Byte 3	Byte 2	---	---
Byte 3	Byte 2	Byte 1	Byte 0	1	1	X	X	Byte 3	Byte 2	Byte 1	Byte 0

BE (Byte-Enable) signals for writing

Bytes in memory				Bytes selection				Output to bus			
D31-D24	D23-D16	D15-D8	D7-D0	BE3	BE2	BE1	BE0	D31-D24	D23-D16	D15-D8	D7-D0
Byte 3	Byte 2	Byte 1	Byte 0	0	0	0	0	---	---	---	Byte 0
Byte 3	Byte 2	Byte 1	Byte 0	0	0	0	1	---	---	Byte 1	---
Byte 3	Byte 2	Byte 1	Byte 0	0	0	1	0	--	Byte 2	---	---
Byte 3	Byte 2	Byte 1	Byte 0	0	0	1	1	Byte 3	---	---	---
Byte 3	Byte 2	Byte 1	Byte 0	0	1	0	X	---	---	Byte 1	Byte 0
Byte 3	Byte 2	Byte 1	Byte 0	0	1	1	X	Byte 3	Byte 2	---	---
Byte 3	Byte 2	Byte 1	Byte 0	1	1	X	X	Byte 3	Byte 2	Byte 1	Byte 0

Memory Access size



Nomenclature:

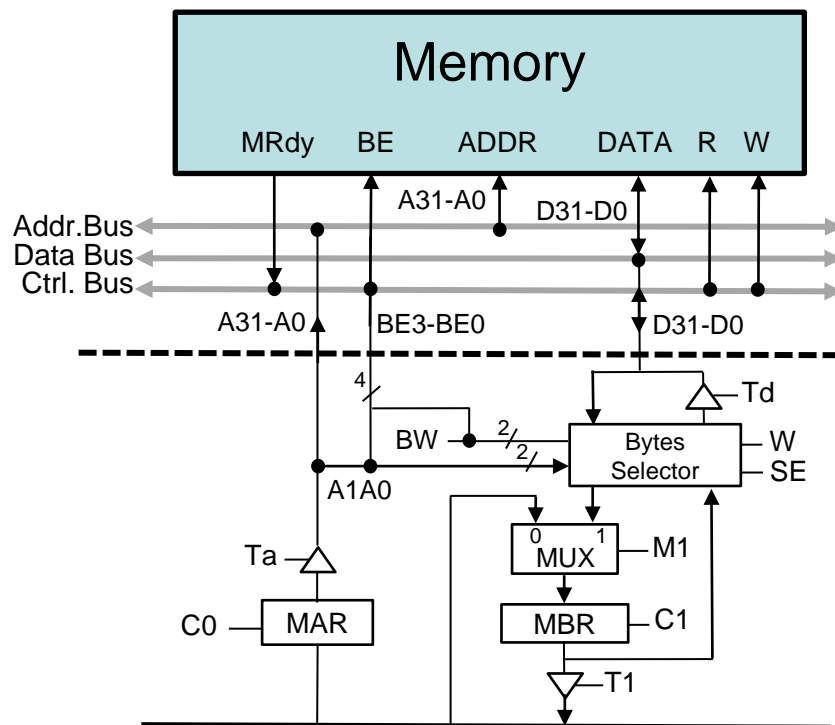
- MAR -> Addresss register
- MBR -> Data register

- ▶ Byte Selector: selects which bytes are stored in MBR while reading and copy to the bus on writes.
- ▶ BW=0: access to bytes
- ▶ BW=01: access to half a word
- ▶ BW =11: word access
- ▶ SE: sign extension
 - ▶ 0: does not extend the sign in smaller accesses of a word
 - ▶ 1: extends the sign in smaller word accesses

Example

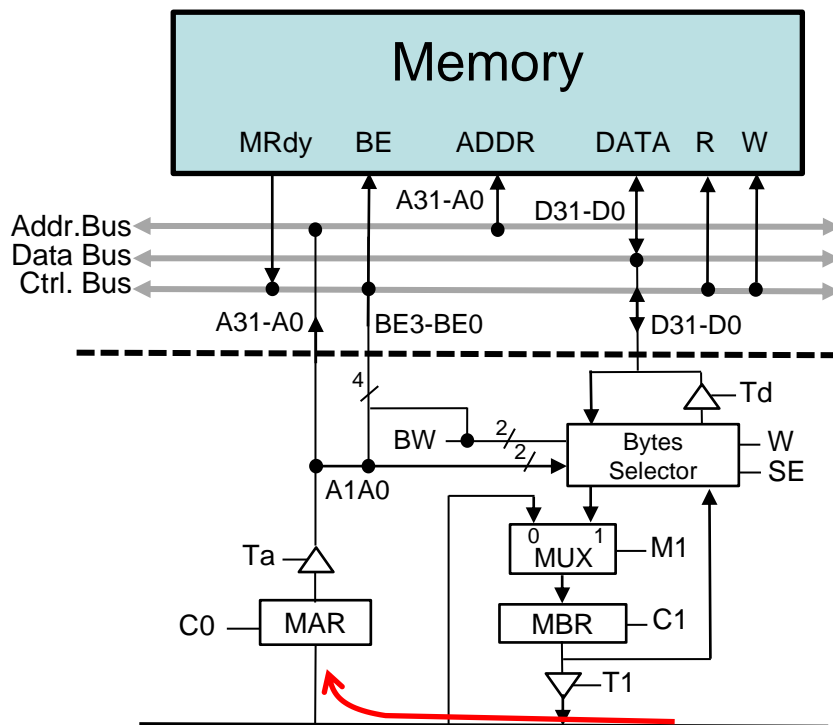
elemental operations in main memory

► Reading a word



access to 1 cycle synchronous main memory

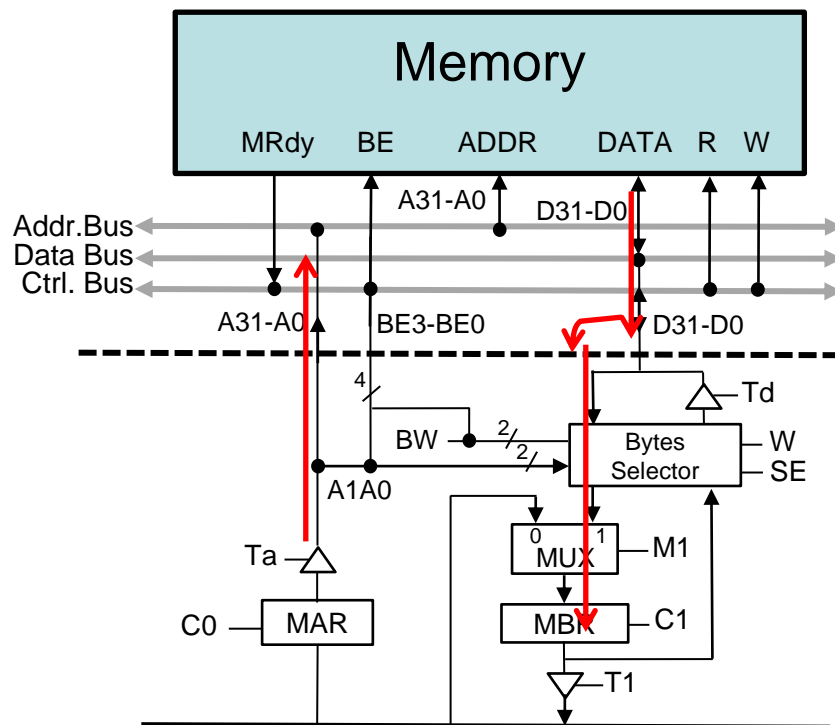
► **Read**



Elem. Op.	Signals
MAR ← <address>	..., C0

access to 1 cycle synchronous main memory

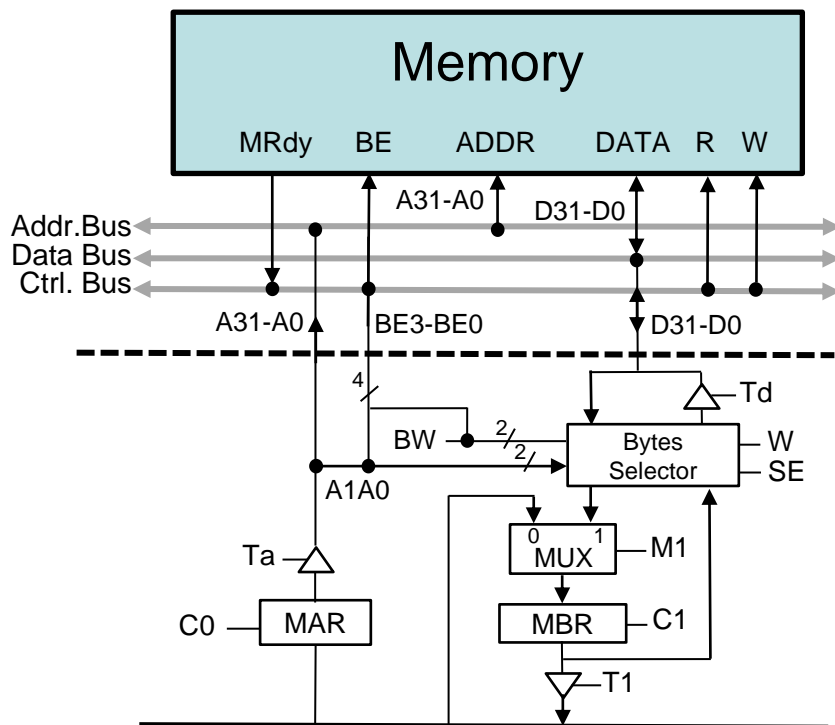
► **Read**



Elem. Op.	Signals
MAR \leftarrow <address>	..., C0
MBR \leftarrow MP[MAR]	Ta, R, M1, C1, BW=11

Example

access to 1 cycle synchronous main memory



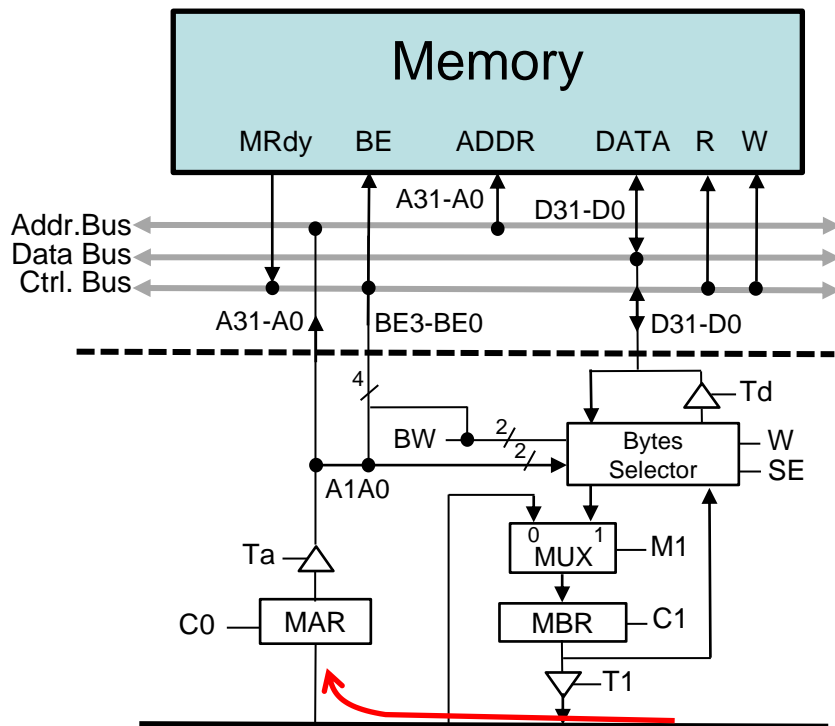
► Read

Elem. Op.	Signals
$MAR \leftarrow \langle \text{address} \rangle$..., C0
$MBR \leftarrow MP[MAR]$	Ta, R, M1, C1, BW=11

► Writing a word

Example

access to 1 cycle synchronous main memory



Read

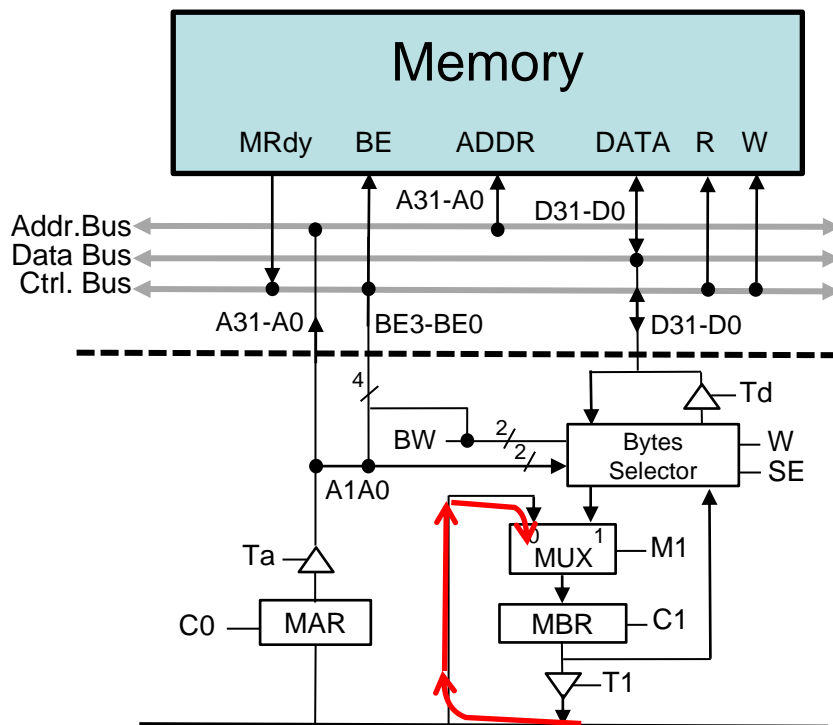
Elem. Op.	Signals
MAR \leftarrow <address>	..., C0
MBR \leftarrow MP[MAR]	Ta, R, M1, C1

Write

Elem. Op.	Signals
MAR \leftarrow <address>	..., C0

Example

access to 1 cycle synchronous main memory



Read

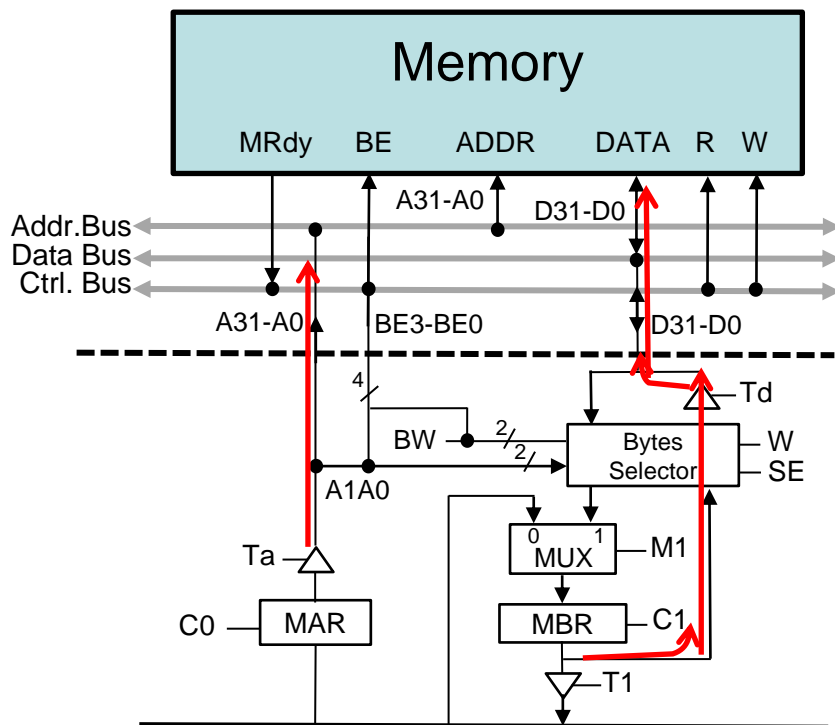
Elem. Op.	Signals
MAR \leftarrow <address>	..., C0
MBR \leftarrow MP[MAR]	Ta, R, M1, C1

Write

Elem. Op.	Signals
MAR \leftarrow <address>	..., C0
MBR \leftarrow <data>	..., C1

Example

access to 1 cycle synchronous main memory



Read

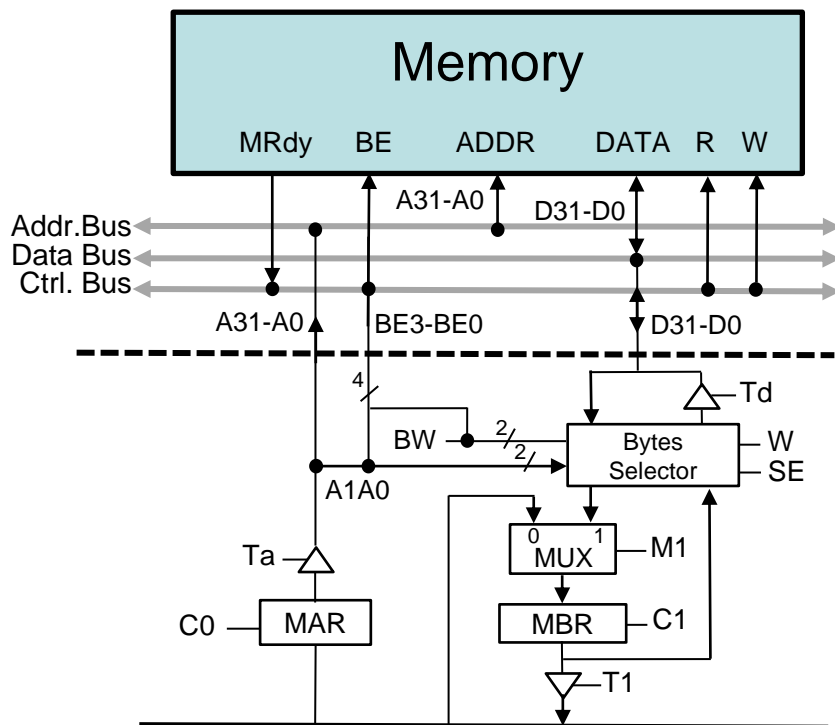
Elem. Op.	Signals
MAR \leftarrow <address>	..., C0
MBR \leftarrow MP[MAR]	Ta, R, M1, C1

Write

Elem. Op.	Signals
MAR \leftarrow <address>	..., C0
MBR \leftarrow <data>	..., C1
Writing cycle	Ta, Td, W, BW=11

Example

access to 1 cycle synchronous main memory



Read

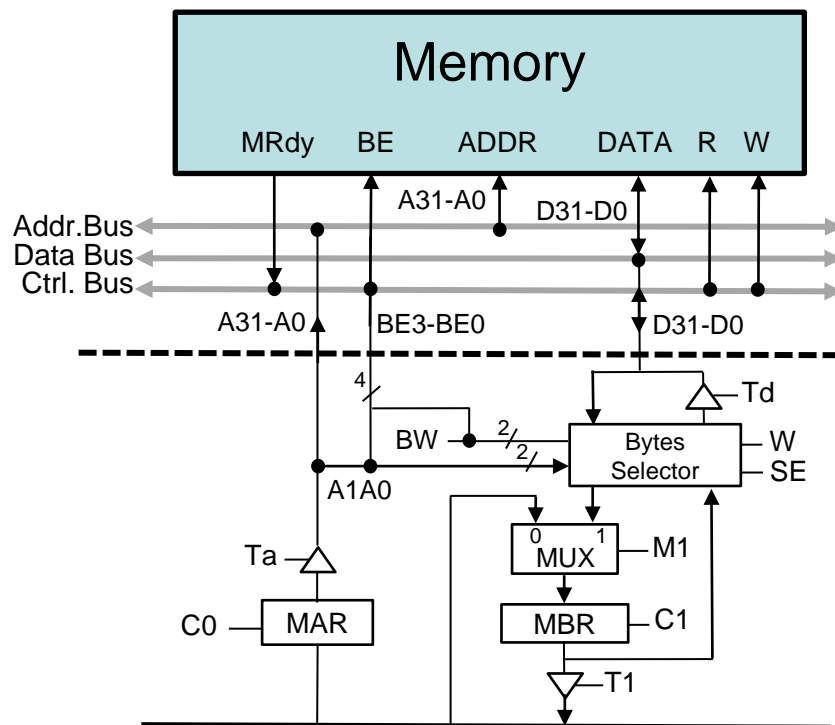
Elem. Op.	Signals
MAR \leftarrow <address>	..., C0
MBR \leftarrow MP[MAR]	Ta, R, M1, C1

Write

Elem. Op.	Signals
MAR \leftarrow <address>	..., C0
MBR \leftarrow <data>	..., C1
Writing cycle	Ta, Td, W, BW=11

access to **2** cycle synchronous main memory

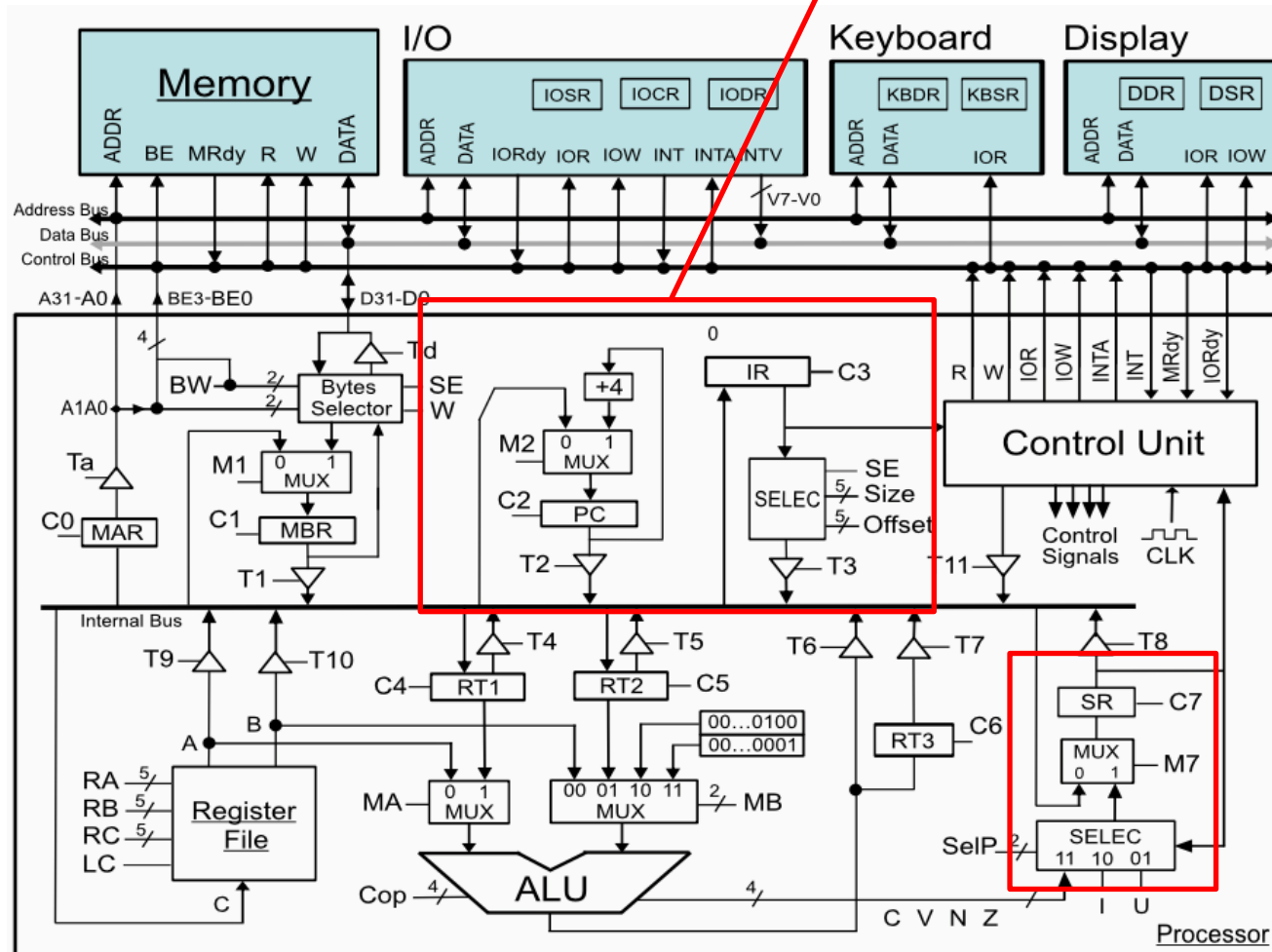
▶ Reading a word



Elem. Op.	Signals
MAR \leftarrow <address>	..., C0
Reading cycle	Ta, R,
Reading cycle MBR \leftarrow MP[MAR]	Ta, R, M1, C1, BW=11

Structure of an elementary computer

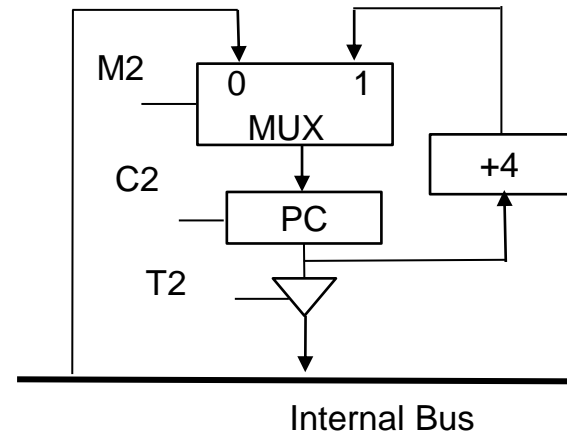
SR, PC and IR registers



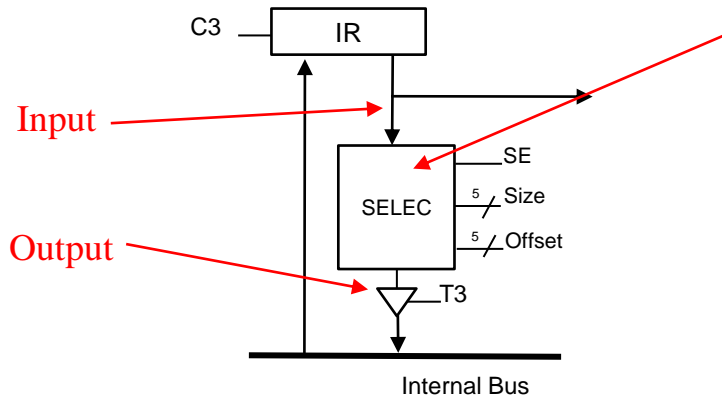
Program Counter

▶ Program Counter (PC):

- ▶ C2, M2
 - ▶ $PC \leftarrow PC + 4$
- ▶ C2 – from internal bus to PC
- ▶ T2 – from PC to internal bus



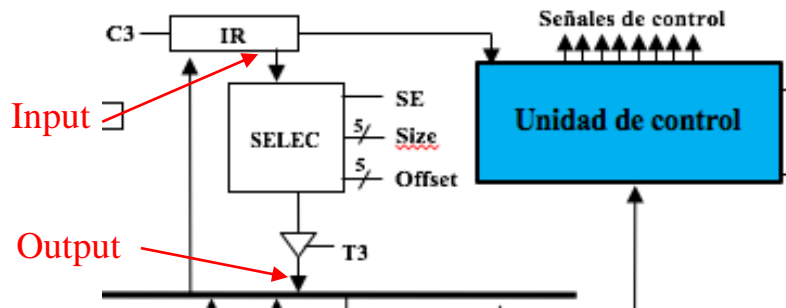
Instruction register



- ▶ C3 - from internal bus to IR
- ▶ SELEC: Transfer IR content to the bus
 - ▶ Size: Size
 - ▶ Offset: displacement
 - ▶ Start bit (less significant)
 - ▶ SE: sign extension

Selector circuit

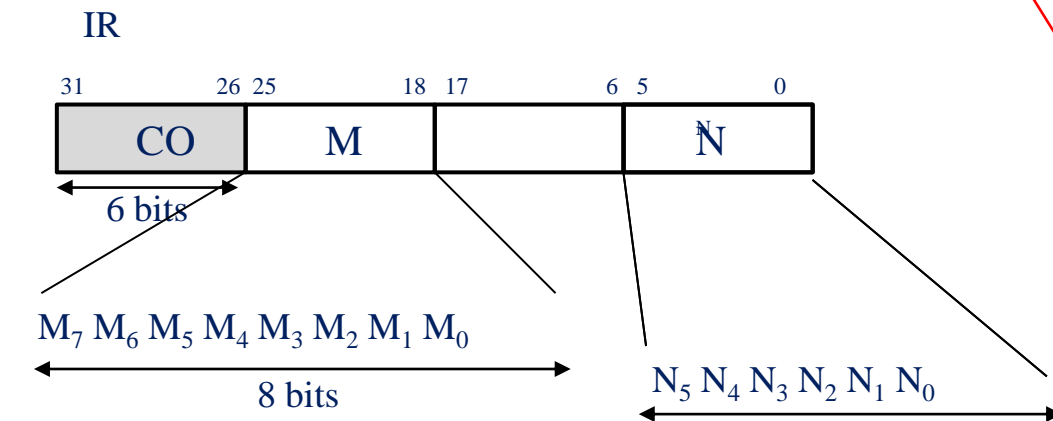
Selection without sign extension (SE = 0)



Size	Offset	Output			
01000	10010	31	24 23	16 15	8 7 0
		0	0	0	$M_7..M_0$
00110	00000	31	24 23	16 15	8 7 0
		0	0	0	$00N_5...N_0$

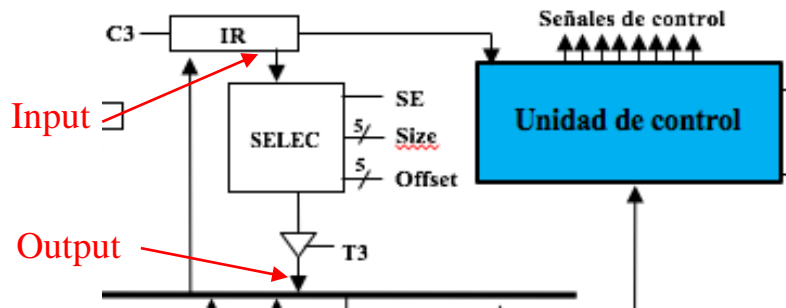
Start bit
(less significant)

Size in bits



Selector circuit

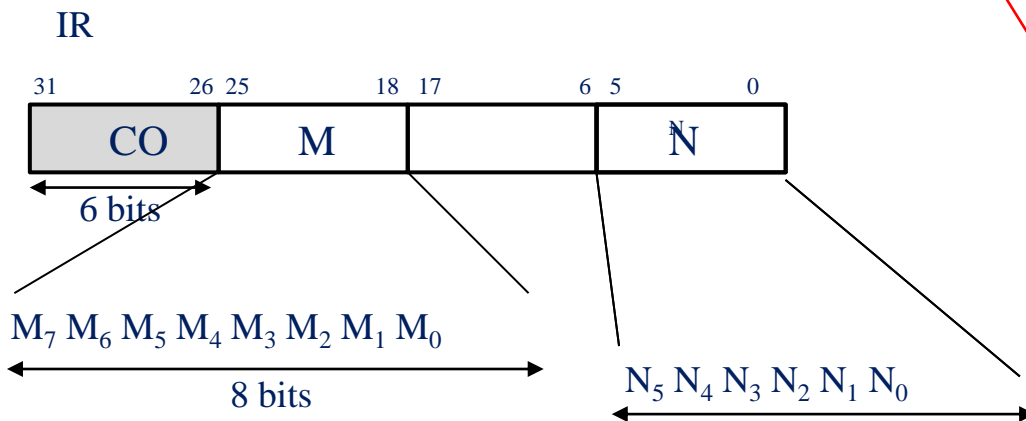
Selection without sign extension (SE = 1)



Size	Offset	Output			
01000	10010	31	24 23	16 15	8 7 0
		$M_7..M_7$ $M_7..M_7$ $M_7..M_7$ $M_7..M_0$			
00110	00000	31	24 23	16 15	8 7 0
		$N_5..N_5$ $N_5..N_5$ $N_5..N_5$ $N_5N_5N_5..N_0$			

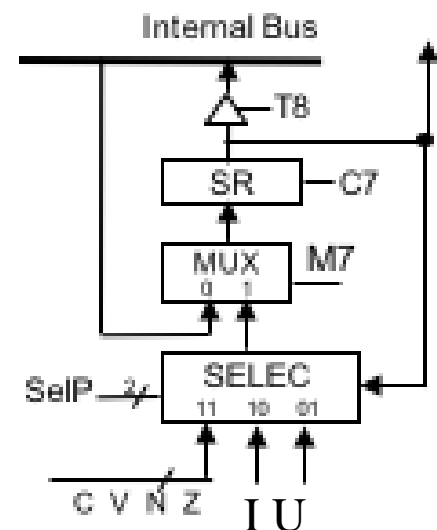
Start bit
(less significant)

Size in bits

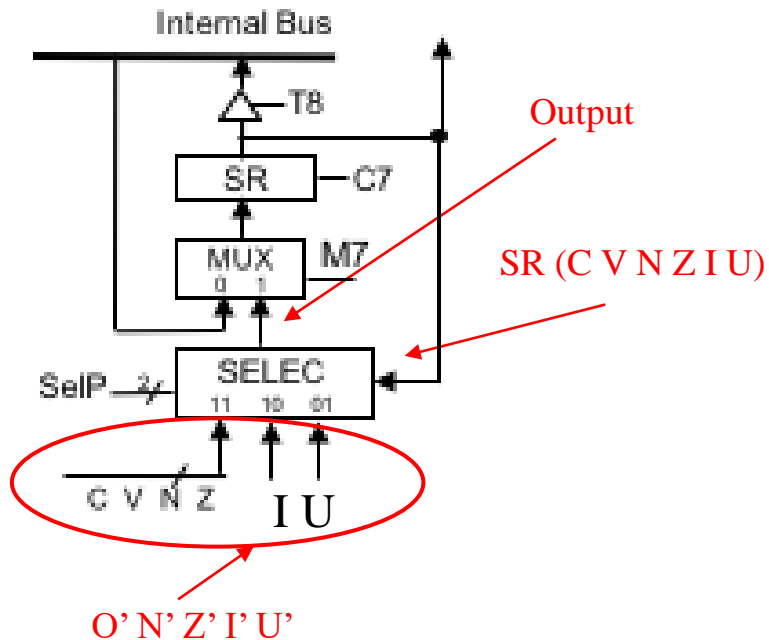


Status register

- ▶ Stores information (status bits) about the status of the program being executed on the processor:
 - ▶ Result of the last operation in the ALU: C, V, N, Z
 - ▶ If the processor is running in kernel mode or user mode (U)
 - ▶ Whether interruptions are enabled or not (I)
- ▶ Associated control signals:
 - ▶ C7 – from internal bus to SR
 - ▶ SelP, M7 – flags from ALU, I, o U to SR
 - ▶ T8 – from SR to internal bus



Status register



SELEC Operation:

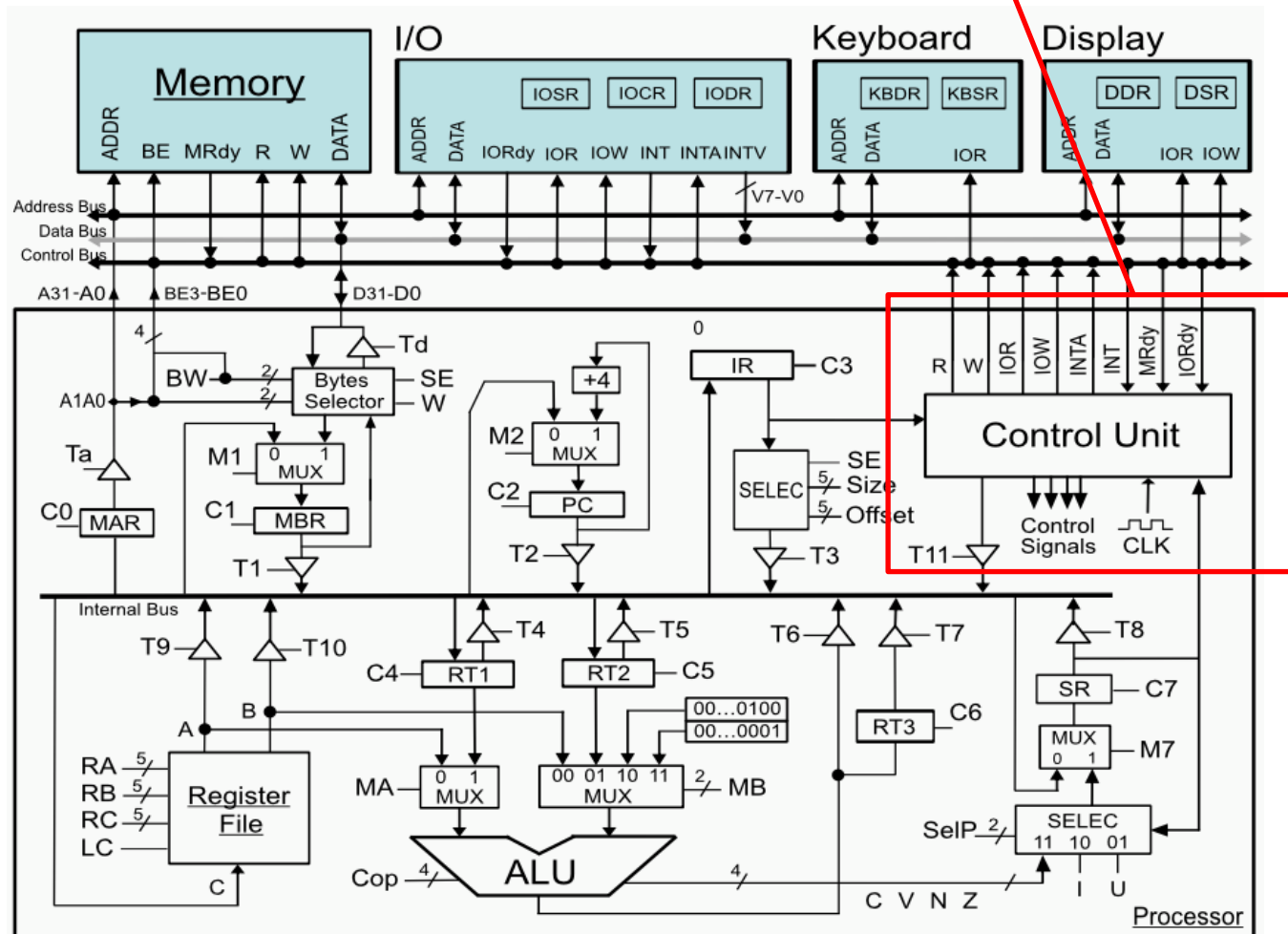
if (SelP1 = 1 AND SelP0 == 1)
Output = $C'V'N'Z'I'U$

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

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

Structure of an elementary computer

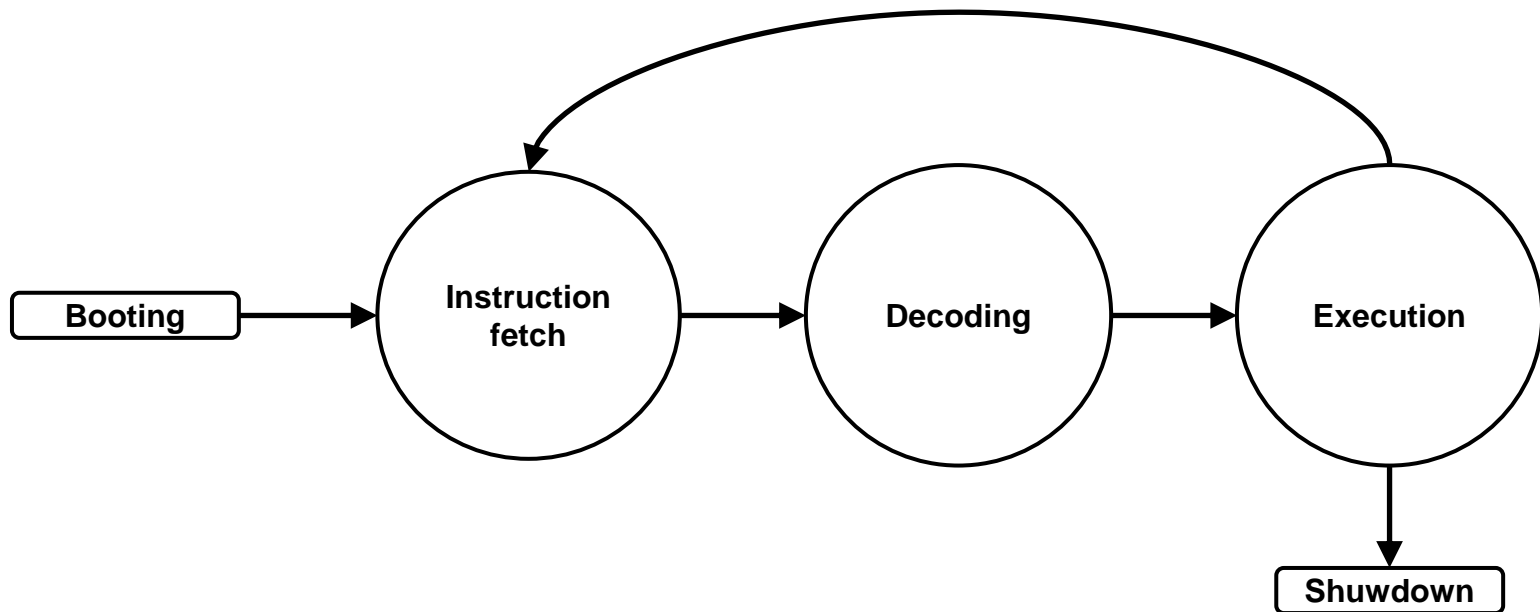
Control Unit (C.U.)



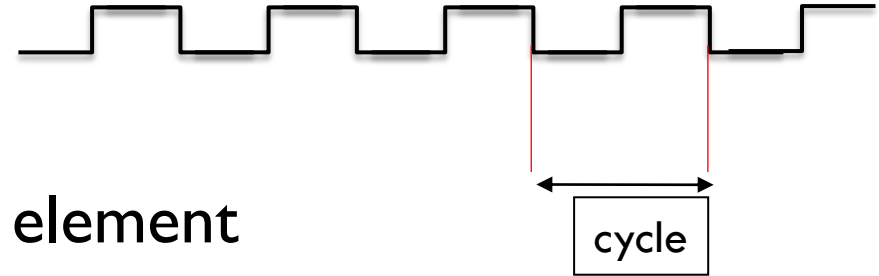
Control unit

Phases of execution of an instruction

- ▶ Basic functions:
 - ▶ Reading instructions from memory
 - ▶ Decoding
 - ▶ Execution of instructions



Clock

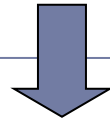


- ▶ A computer is a synchronous element
- ▶ Controls the operation
- ▶ The clock times the operations:
 - ▶ In a clock cycle one or more elementary operations are executed as long as there is no conflict
 - ▶ The necessary control signals are kept active during the cycle
- ▶ In the same cycle you can perform
 - ▶ $MAR \leftarrow PC$ y $RT3 \leftarrow RT2 + RT1$
- ▶ In the same cycle it **is not possible** to perform
 - ▶ $MAR \leftarrow PC$ y $RI \leftarrow RT3$ why?

Description of the Control Unit activity

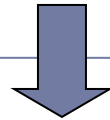
Instruction

mv R0 R1

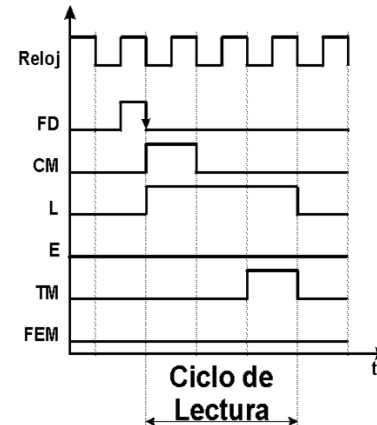


Sequence of **elementary operations**

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

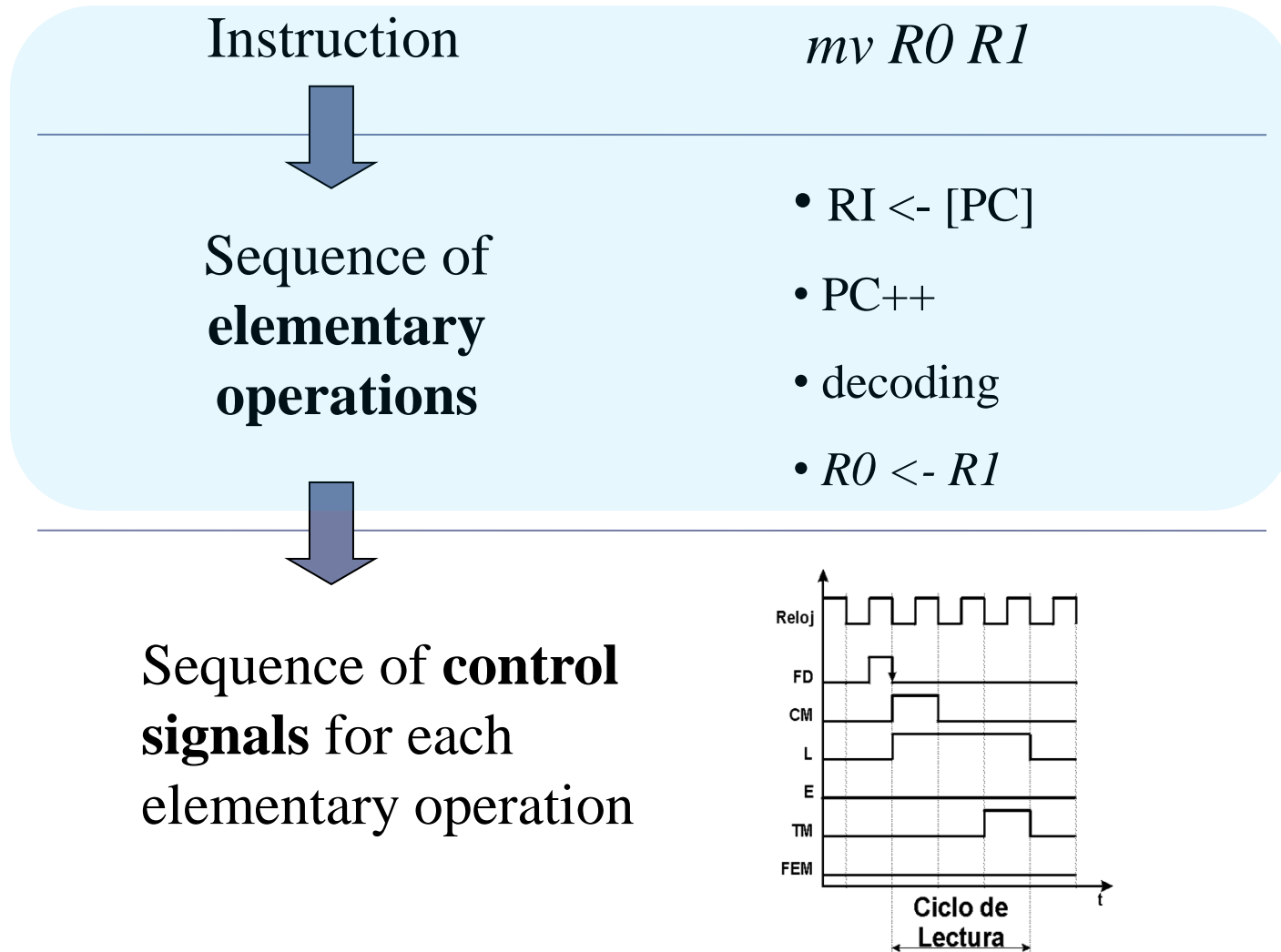


Sequence of **control signals** for each elementary operation



+ level of
hw. details

Description of the Control Unit activity



+ level of
hw. details

Instruction execution phases

▶ Instruction Reading or fetch

- ▶ Read the instruction stored in the memory address indicated by PC and take it to IR.
- ▶ PC is updated to point to the next instruction

▶ Decoding

- ▶ Analysis of the instruction in RI to determine:
 - ▶ The operation to be performed.
 - ▶ Address to be applied.
 - ▶ Control signals to be activated

▶ Execution

- ▶ Generation of the control signals in each clock cycle.

Fetch

Cycle	Elem. Op.
C1	$MAR \leftarrow PC$
C2	$PC \leftarrow PC + 4$
C3	$MBR \leftarrow MP$
C4	$IR \leftarrow MBR$



Cycle	Elem. Op.
C1	$MAR \leftarrow PC$
C2	$PC \leftarrow PC + 4,$ $MBR \leftarrow MP$
C3	$IR \leftarrow MBR$

Possibility of simultaneous operations

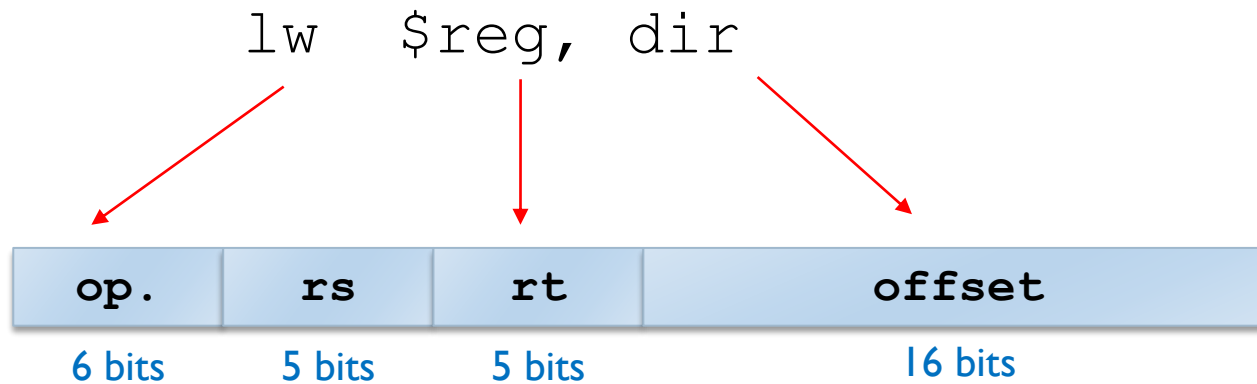
Fetch Cycle Control Signals

- Specification of the active control signals in each clock cycle
 - Can be generated from the RT level.

Cycle	Elem. Op.	Control Signals
C1	$MAR \leftarrow PC$	T2, C0
C2	$PC \leftarrow PC + 4,$ $MBR \leftarrow MP$	C2, M2 Ta, R, C1, M1, BW=11
C3	$IR \leftarrow MBR$	T1, C3

Example

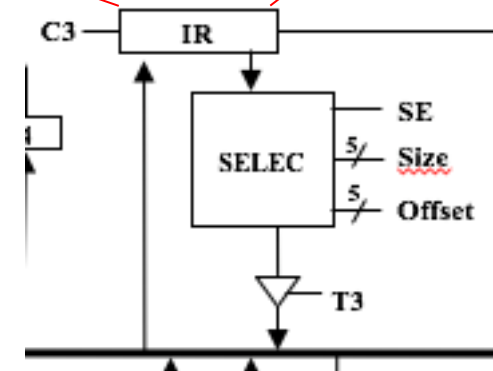
► `lw $reg, dir`



Execution of `lw $reg, dir`



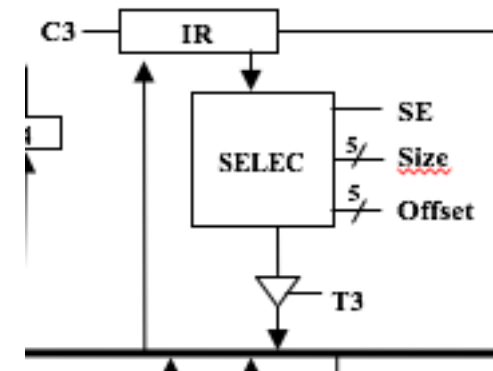
Cycle	Elem. Op.	Control Signals
C1	$MAR \leftarrow PC$	T2, C0
C2	$PC \leftarrow PC + 4,$ $MBR \leftarrow MP$	C2, M2 Ta, R, CI, MI, BW=11
C3	$IR \leftarrow MBR$	T1, C3
C4		
C5		
C6		
C7		



Execution of `lw $reg, dir`



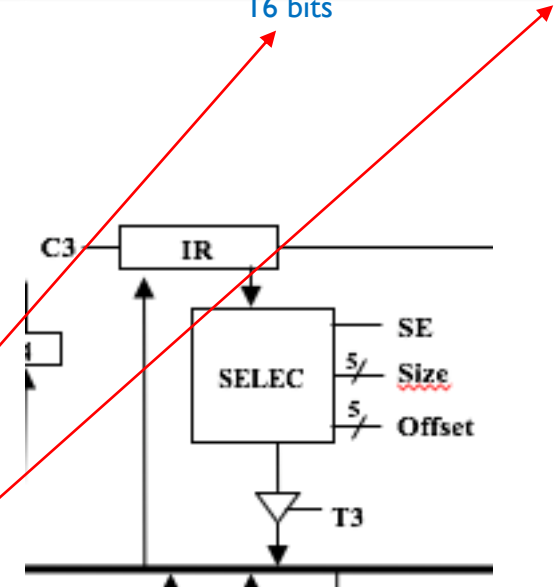
Cycle	Elem. Op.	Control Signals
C1	$MAR \leftarrow PC$	T2, C0
C2	$PC \leftarrow PC + 4,$ $MBR \leftarrow MP$	C2, M2 Ta, R, CI, MI, BW=II
C3	$IR \leftarrow MBR$	T1, C3
C4	Decoding	
C5		
C6		
C7		



Execution of `lw $reg, dir`



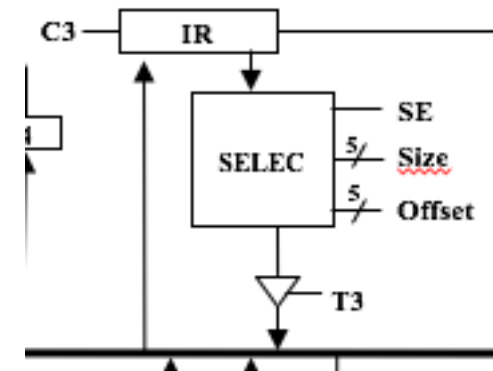
Cycle	Elem. Op.	Control Signals
C1	$MAR \leftarrow PC$	T2, C0
C2	$PC \leftarrow PC + 4,$ $MBR \leftarrow MP$	C2, M2 Ta, R, CI, MI, BW=11
C3	$IR \leftarrow MBR$	T1, C3
C4	Decoding	
C5	$MAR \leftarrow RI(dir)$	C0, T3, Size = 10000 Offset = 00000
C6		
C7		



Execution of `lw $reg, dir`



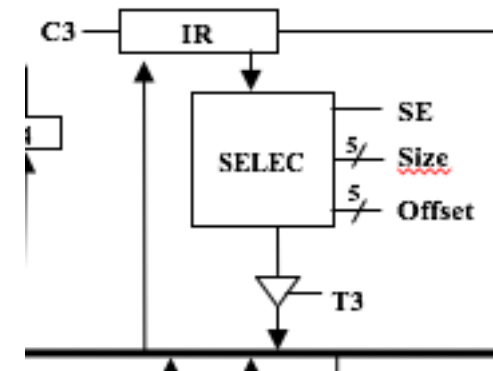
Cycle	Elem. Op.	Control Signals
C1	$MAR \leftarrow PC$	T2, C0
C2	$PC \leftarrow PC + 4,$ $MBR \leftarrow MP$	C2, M2 Ta, R, CI, MI, BW=11
C3	$IR \leftarrow MBR$	T1, C3
C4	Decoding	
C5	$MAR \leftarrow RI(dir)$	C0, T3, Size = 10000 Offset = 00000
C6	$MBR \leftarrow MP$	Ta, R, CI, MI, BW=11
C7		



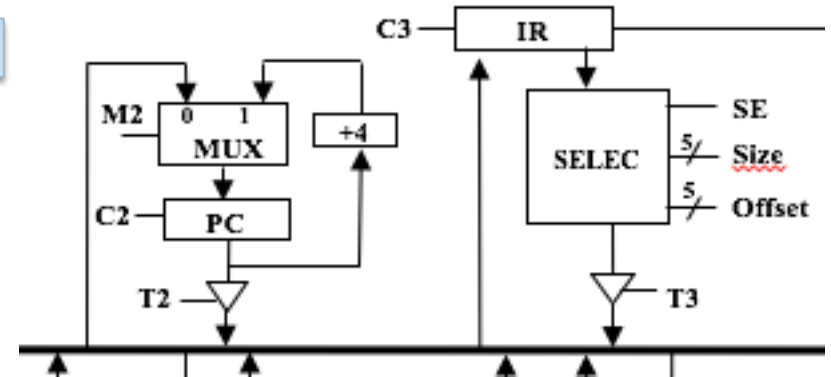
Execution of `lw $reg, dir`



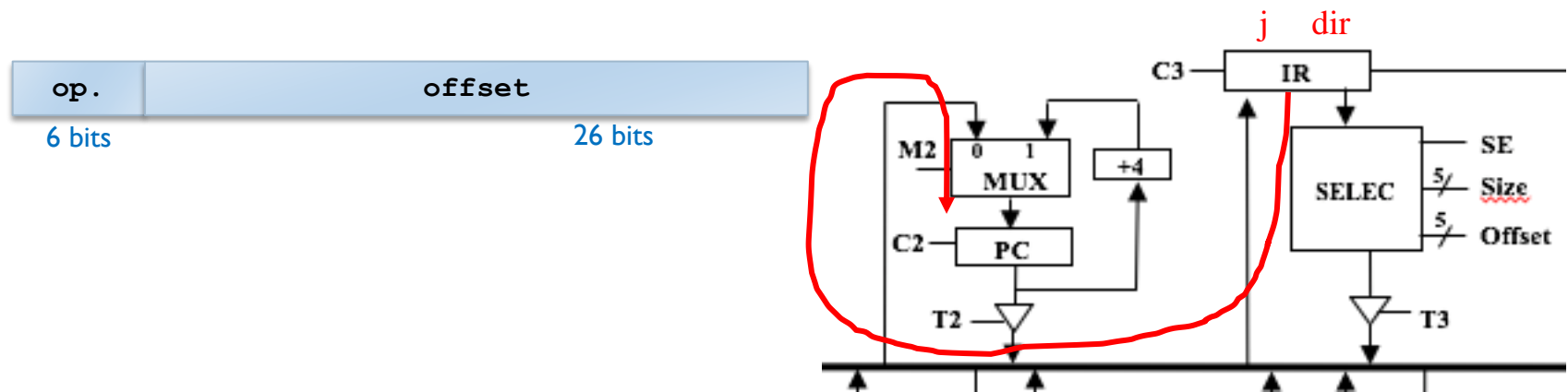
Cycle	Elem. Op.	Control Signals
C1	$MAR \leftarrow PC$	T2, C0
C2	$PC \leftarrow PC + 4,$ $MBR \leftarrow MP$	C2, M2 Ta, R, CI, MI, BW=11
C3	$IR \leftarrow MBR$	T1, C3
C4	Decoding	
C5	$MAR \leftarrow RI(dir)$	C0, T3, Size = 10000 Offset = 00000
C6	$MBR \leftarrow MP$	Ta, R, CI, MI, BW=11
C7	$\\$reg \leftarrow MBR$	T1, RC=id \$reg, LC



Execution of j dir



Execution of j dir



Cycle	Elem. Op.	Control Signals
C1	$MAR \leftarrow PC$	T2, C0
C2	$PC \leftarrow PC + 4,$ $MBR \leftarrow MP$	C2, M1 Ta, R, C1, M1, BW=11
C3	$IR \leftarrow MBR$	T1, C3
C4	Decoding	
C5	$PC \leftarrow RI(dir)$	C2, T3, Size = 11010 (26) Offset = 00000

Exercises

► Instructions that fit in one word:

- `sw $reg, dir`
- `add $rd, $ro1, $ro2`
- `addi $rd, $ro1, inm`
- `lw $reg1, desp($reg2)`
- `j dir`
- `jr $reg`
- `beq $ro1, $ro2, desp`

beqz \$reg, desplaz

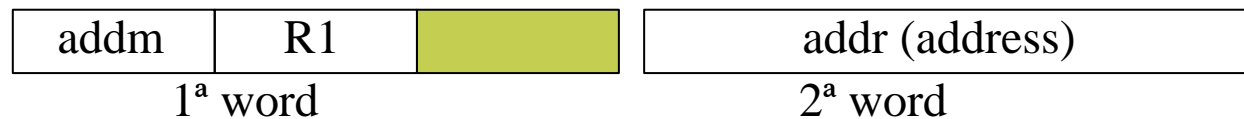
Cycle	Elem. Op.
C1	$MAR \leftarrow PC$
C2	$PC \leftarrow PC + 4,$ $MBR \leftarrow MP$
C3	$IR \leftarrow MBR$
C4	Decoding
C5	$\$reg + \0
C6	Si $SR.Z == 0$ jump to fetch
C7	$RT2 \leftarrow PC$
C8	$RT1 \leftarrow IR(\text{desplaz})$
C9	$RT1 \leftarrow RT1 * 4$
C10	$PC \leftarrow RT1 + RT2$

Si $\$reg == 0$
 $PC \leftarrow PC + \text{desp} * 4$

Instructions that take up several words

Example : `addm R1, addr` $R1 \leftarrow R1 + MP[addr]$

Format:



Cycle	Elem. Op.
C1	$MAR \leftarrow PC$
C2	$PC \leftarrow PC + 4,$ $MBR \leftarrow MP$
C3	$IR \leftarrow MBR$
C4	Decoding
C5	$MAR \leftarrow PC$

Cycle	Elem. Op.
C6	$MBR \leftarrow MP,$ $PC \leftarrow PC + 4$
C7	$MAR \leftarrow MBR$
C8	$MBR \leftarrow MP$
C9	$RTI \leftarrow MBR$
C10	$RI \leftarrow RI + RTI$

Example

ADD (R_2) R_3 (R_4)

A. Fetch + Decod.

- 1.- $MAR \leftarrow PC$
- 2.- $RI \leftarrow \text{Memory}(MAR)$
- 3.- $PC \leftarrow PC + "4"$
- 4.- Decoding

B. Fetch operands.

- 5.- $MAR \leftarrow R_4$
- 6.- $MBR \leftarrow \text{Memory}(MAR)$
- 7.- $RTI \leftarrow MBR$

C. Execution

- 8.- $MBR \leftarrow R_3 + RTI$

D. Store results

- 9.- $MAR \leftarrow R_2$
- 10.- $\text{Memory}(MAR) \leftarrow MBR$

Warnings

remember don'ts, everything else is yes...

1. **It is not possible to go through a register in the clock cycle**
2. **It is not possible to take two or more values to a bus at the same time**
3. **It is not possible to set a datapath if the circuitry does not enable it.**

Contents

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

Modes of execution

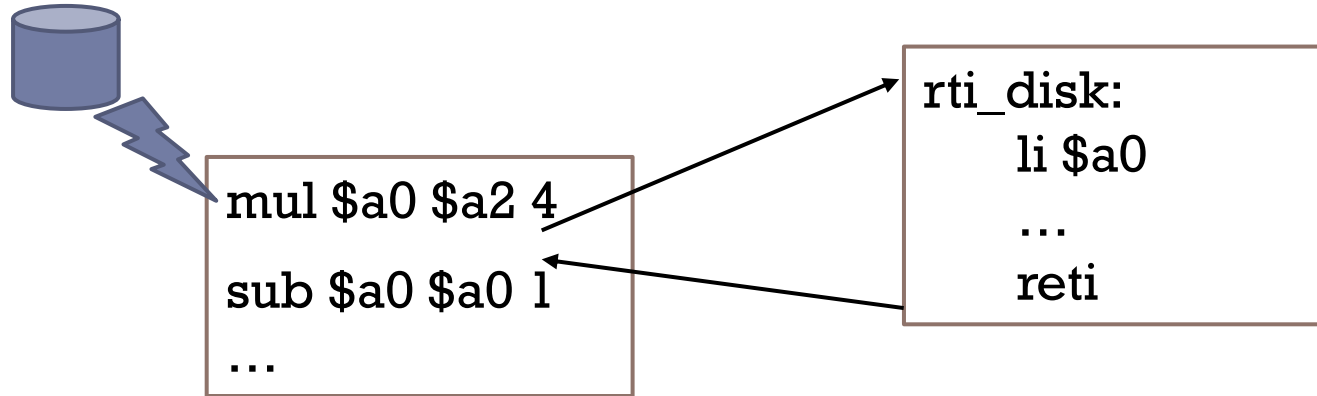
▶ User Mode

- ▶ 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
- ▶ It is indicated by a bit in the status register (U)

Interrupts



- ▶ Signal that arrives at the C.U. and breaks the normal execution sequence: the current program is stopped and another one is executed to attend the interruption.
- ▶ Example of causes:
 - ▶ When a peripheral requests the attention of the processor
 - ▶ When an error occurs in the execution of the instruction, ...

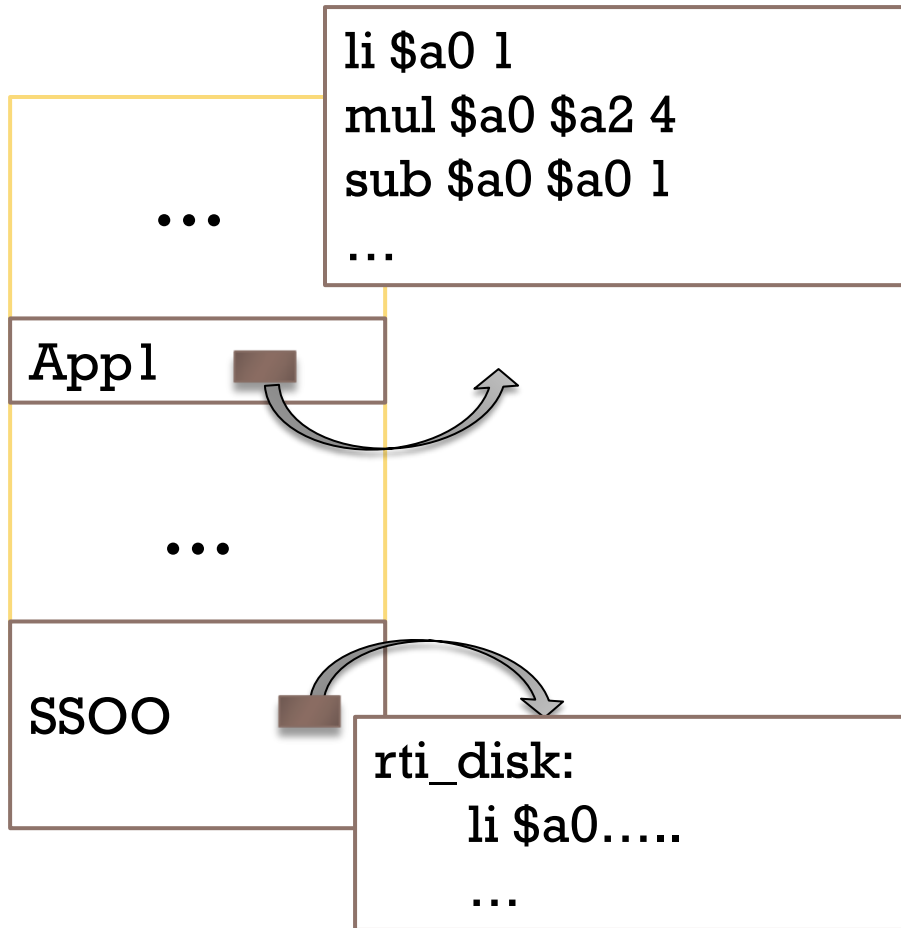
Interrupts

- ▶ Signal that reaches the control unit and breaks the normal execution sequence
- ▶ Causes:
 - ▶ When an error occurs in the execution of the instruction (division by zero, ...)
 - ▶ Execution of an illegal instruction
 - ▶ Accessing an illegal memory location
 - ▶ When a peripheral requests the attention of the processor
 - ▶ The clock. Clock Interrupts
- ▶ When an interruption is generated, the current program is stopped, and the execution is transferred to another program that serves the interruption

Classification of interruptions

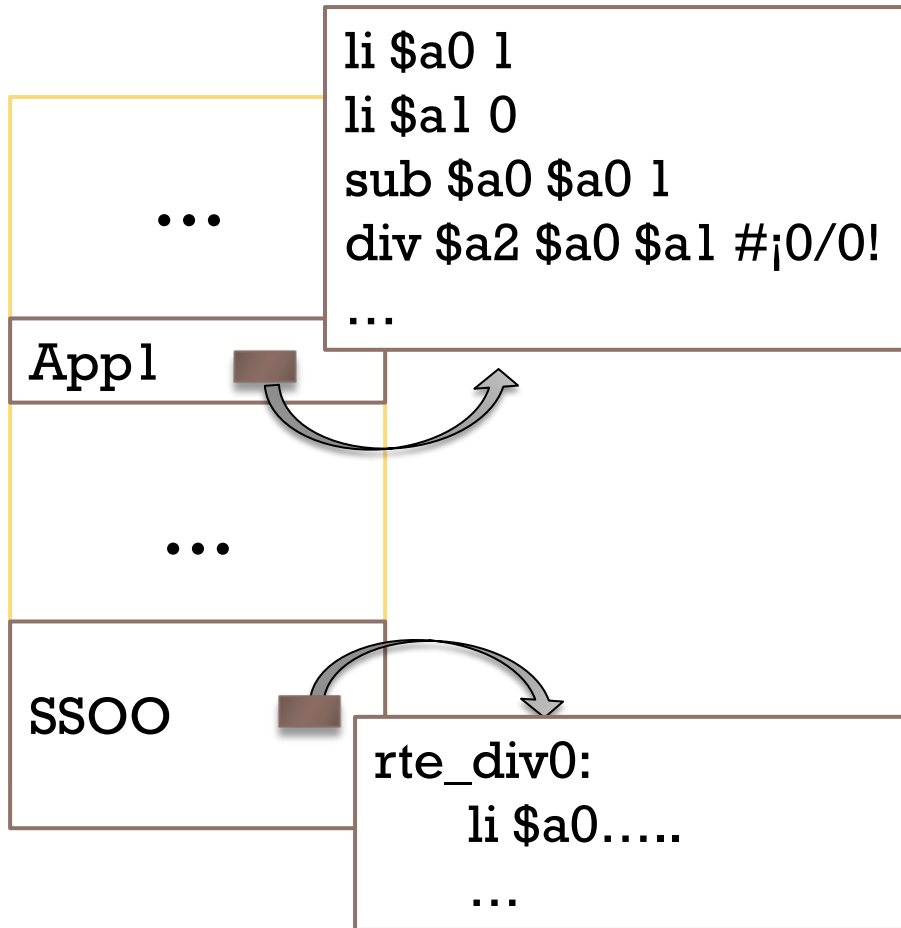
- ▶ **Synchronous hardware exceptions**
 - ▶ Division by zero, access to an illegal memory position,
- ▶ **Asynchronous hardware exceptions**
 - ▶ Faults or errors in the HW
- ▶ **External interruptions**
 - ▶ Peripherals, clock interruption
- ▶ **Calls to the system**
 - ▶ Special machine instructions that generate an interruption to activate the operating system

Asynchronous Hardware Exceptions and External Interrupts



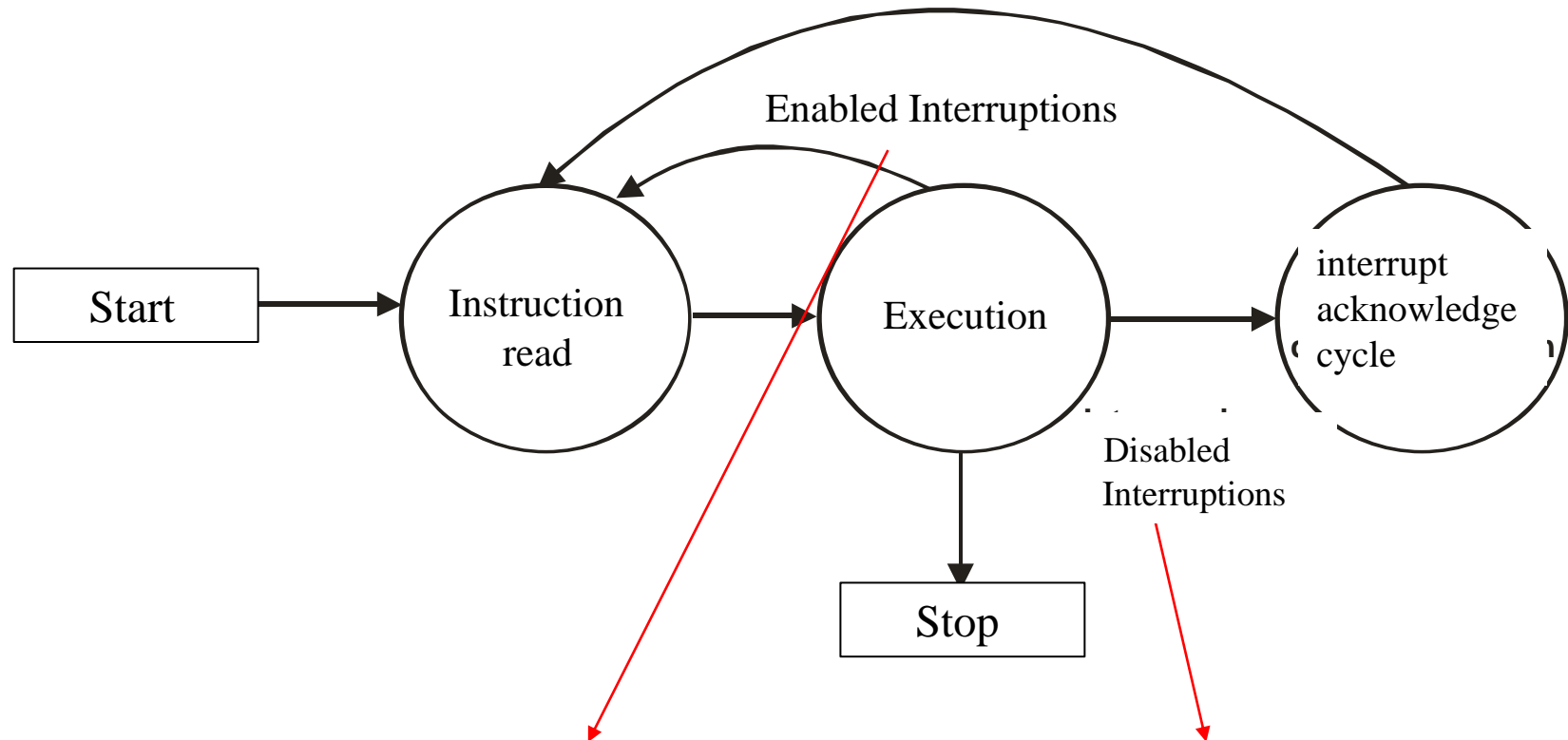
- ▶ They cause an unscheduled sequence break
 - ▶ **At the end of the microprogram of the instruction in progress see if there is any pending interruption, and if so...**
 - ▶ ...Bifurcation to subroutine of the O.S. that treats it
- ▶ It then restores the status and returns control to the interrupted program.
- **Asynchronous cause to the execution of the current program**
 - ▶ Peripheral care
 - ▶ Etc.

Synchronous hardware exceptions



- ▶ They cause an unscheduled sequence break
 - ▶ **Within the microprogram of the ongoing instruction...**
 - ▶ ...Bifurcation to subroutine of the O.S. that treats it
- ▶ Subsequently, 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.

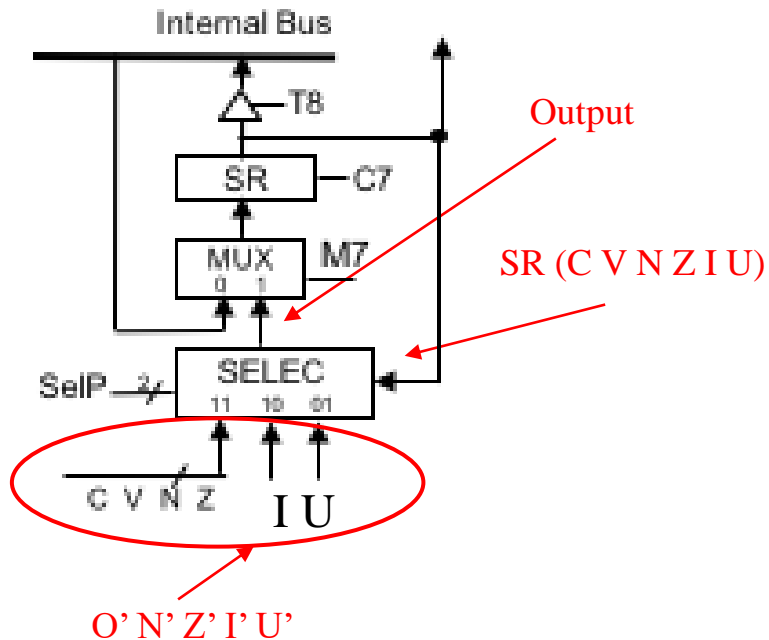
Interrupt handle



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'**

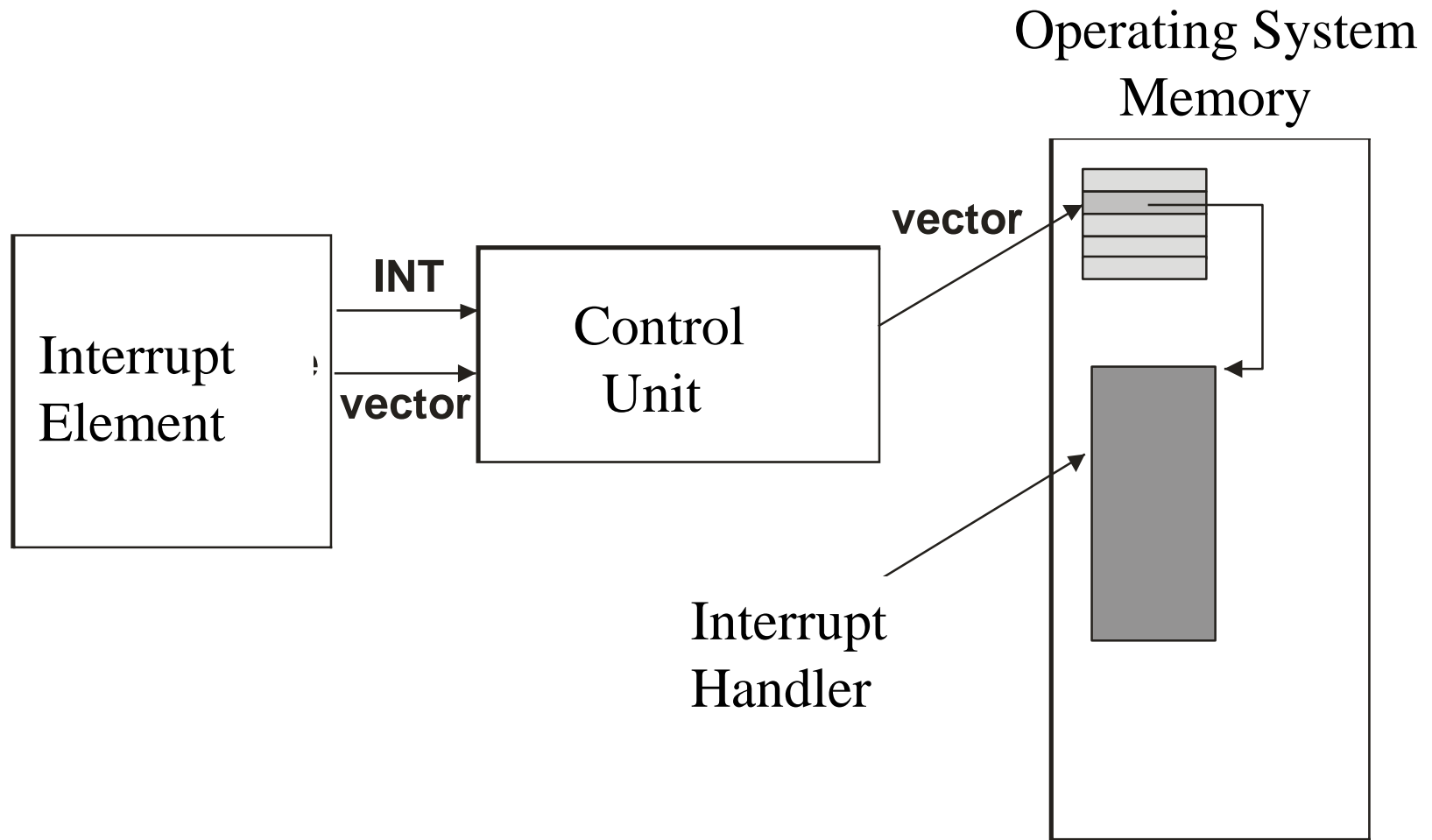
Interrupt acknowledge cycle

- ▶ During this cycle the Control Unit performs the following steps:
 - ▶ Checks if an interruption signal is activated.
 - ▶ If it is activated:
 - ▶ Saves the program counter and status register
 - ▶ Switches from user mode to core mode
 - ▶ Obtains the address of the interruption treatment routine
 - ▶ Store the address obtained in the program counter (this way the following instruction will be the one for the treatment routine)

Interrupt service routine (ISR)

- ▶ 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).

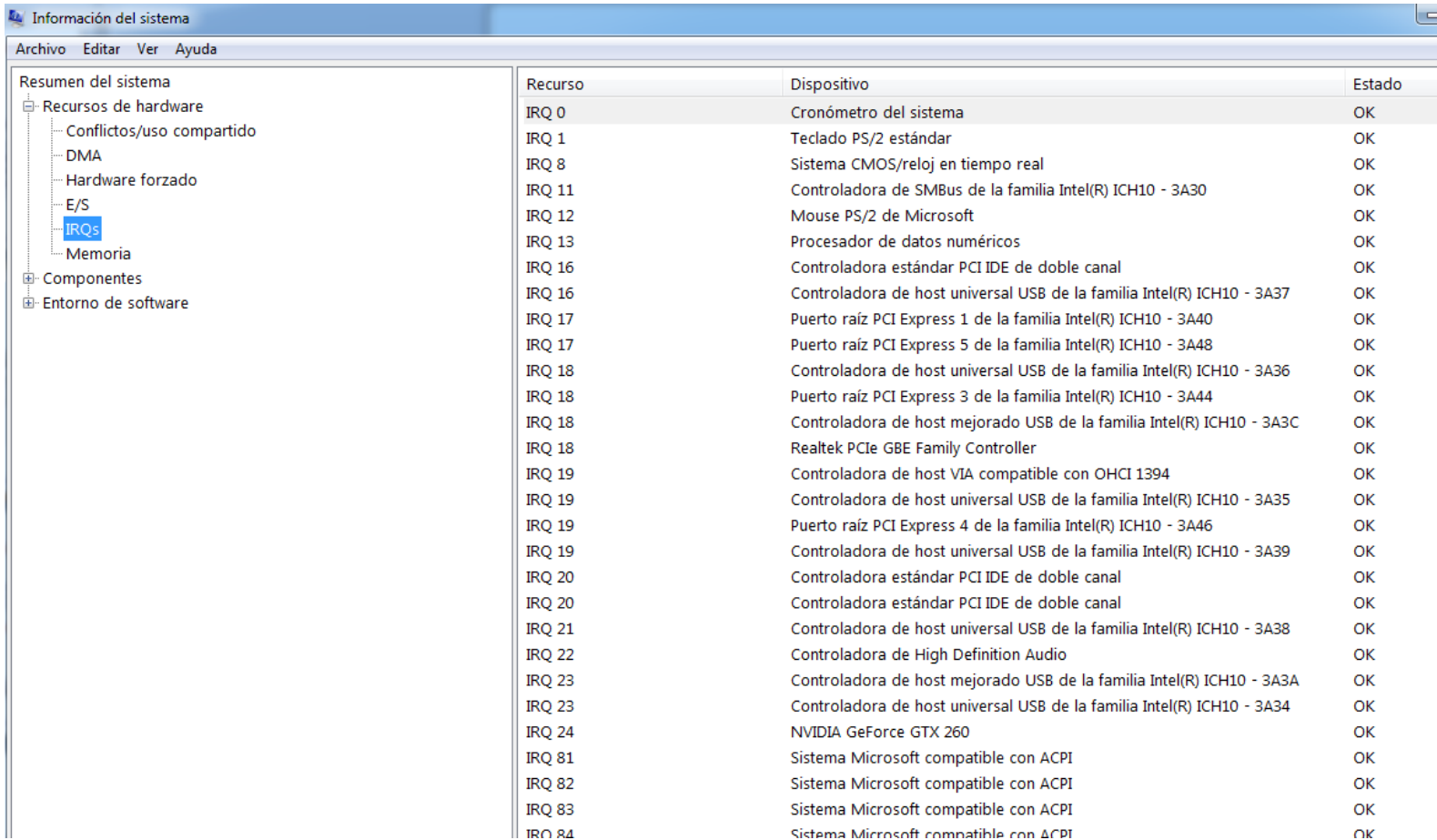
Vector interrupts



Vector interrupts

- ▶ The interrupting element supplies the **interrupt vector**
- ▶ This vector is an index in a table containing the address of the interrupt handler routine.
- ▶ The Control Unit reads the content of this entry and loads the value into the PC
- ▶ Each operating system fills this table with the addresses of each of the treatment routines, which are dependent on each operating system.

Interrupts in Windows



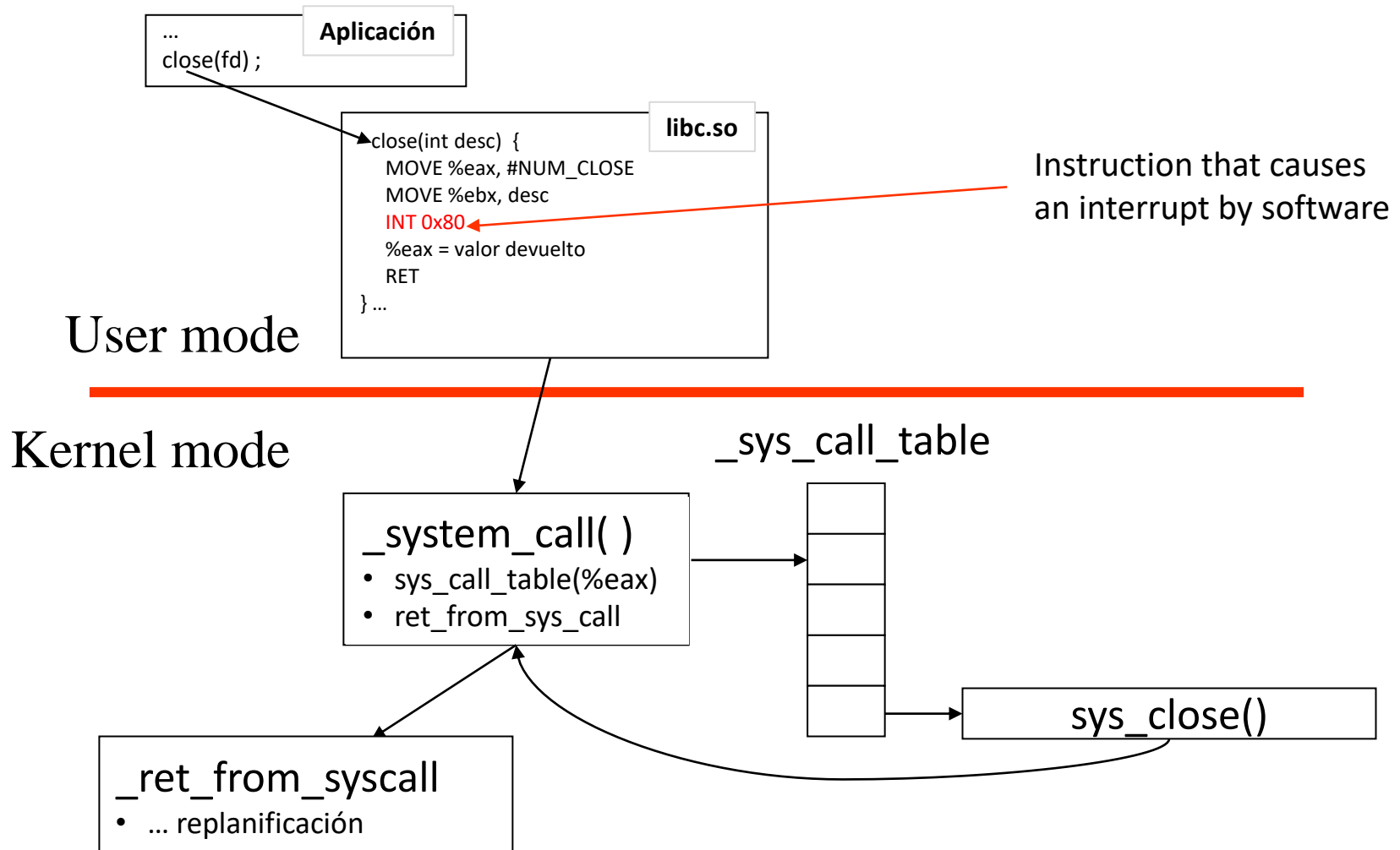
Información del sistema			
Archivo Editar Ver Ayuda			
Resumen del sistema		Recurso	Dispositivo Estado
Recursos de hardware		IRQ 0	Cronómetro del sistema OK
Conflictos/uso compartido		IRQ 1	Teclado PS/2 estándar OK
DMA		IRQ 8	Sistema CMOS/reloj en tiempo real OK
Hardware forzado		IRQ 11	Controladora de SMBus de la familia Intel(R) ICH10 - 3A30 OK
E/S		IRQ 12	Mouse PS/2 de Microsoft OK
IRQs		IRQ 13	Procesador de datos numéricos OK
Memoria		IRQ 16	Controladora estándar PCI IDE de doble canal OK
Componentes		IRQ 16	Controladora de host universal USB de la familia Intel(R) ICH10 - 3A37 OK
Entorno de software		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
		IRQ 18	Puerto raíz PCI Express 3 de la familia Intel(R) ICH10 - 3A44 OK
		IRQ 18	Controladora de host mejorado USB de la familia Intel(R) ICH10 - 3A3C OK
		IRQ 18	Realtek PCIe GBE Family Controller OK
		IRQ 19	Controladora de host VIA compatible con OHCI 1394 OK
		IRQ 19	Controladora de host universal USB de la familia Intel(R) ICH10 - 3A35 OK
		IRQ 19	Puerto raíz PCI Express 4 de la familia Intel(R) ICH10 - 3A46 OK
		IRQ 19	Controladora de host universal USB de la familia Intel(R) ICH10 - 3A39 OK
		IRQ 20	Controladora estándar PCI IDE de doble canal OK
		IRQ 20	Controladora estándar PCI IDE de doble canal OK
		IRQ 21	Controladora de host universal USB de la familia Intel(R) ICH10 - 3A38 OK
		IRQ 22	Controladora de High Definition Audio OK
		IRQ 23	Controladora de host mejorado USB de la familia Intel(R) ICH10 - 3A3A OK
		IRQ 23	Controladora de host universal USB de la familia Intel(R) ICH10 - 3A34 OK
		IRQ 24	NVIDIA GeForce GTX 260 OK
		IRQ 81	Sistema Microsoft compatible con ACPI OK
		IRQ 82	Sistema Microsoft compatible con ACPI OK
		IRQ 83	Sistema Microsoft compatible con ACPI OK
		IRQ 84	Sistema Microsoft compatible con ACPI OK

Software Interrupts. System calls and operating systems

- ▶ 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.

Software interrupts

System calls (example: Linux)



Clock interrupts and operating system

- ▶ 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