

ARCOS Group

uc3m | Universidad **Carlos III** de Madrid

L4: The processor (1)

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

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

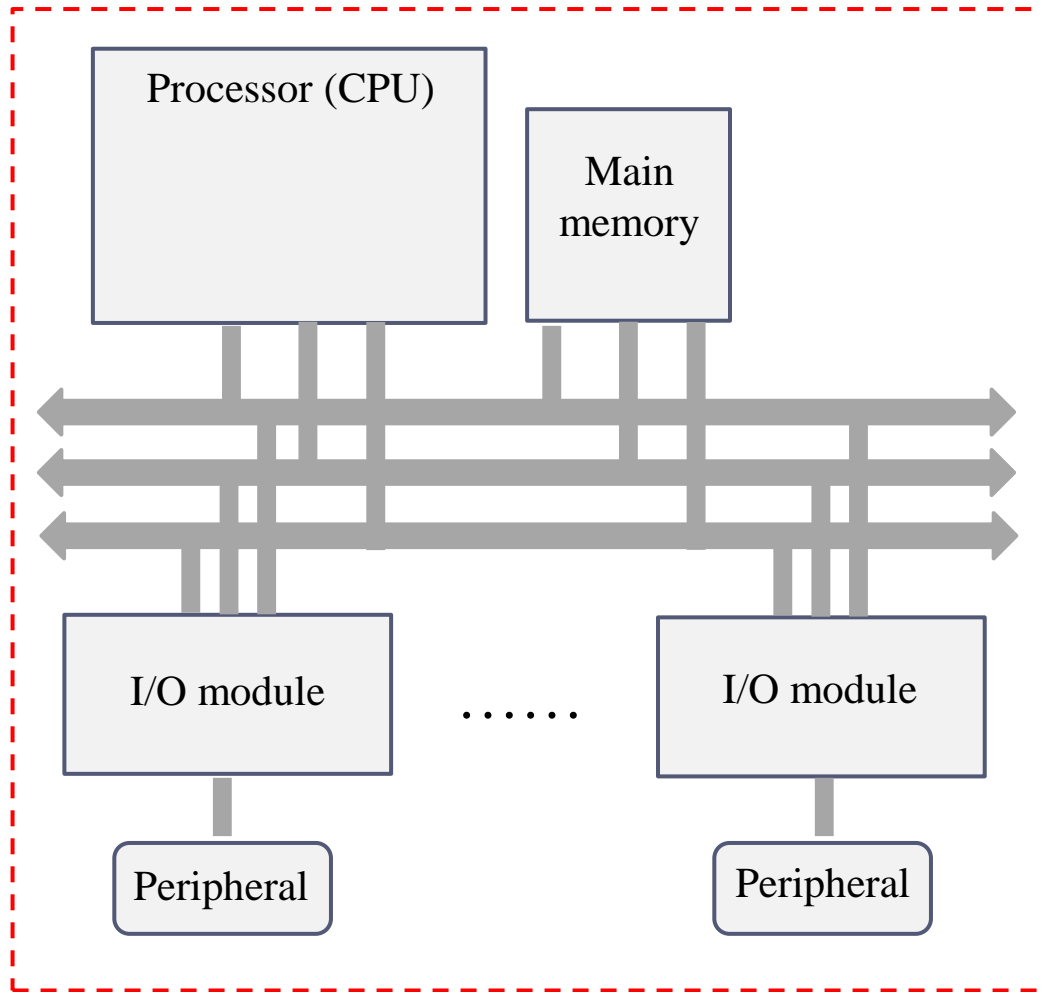
Contents

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

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

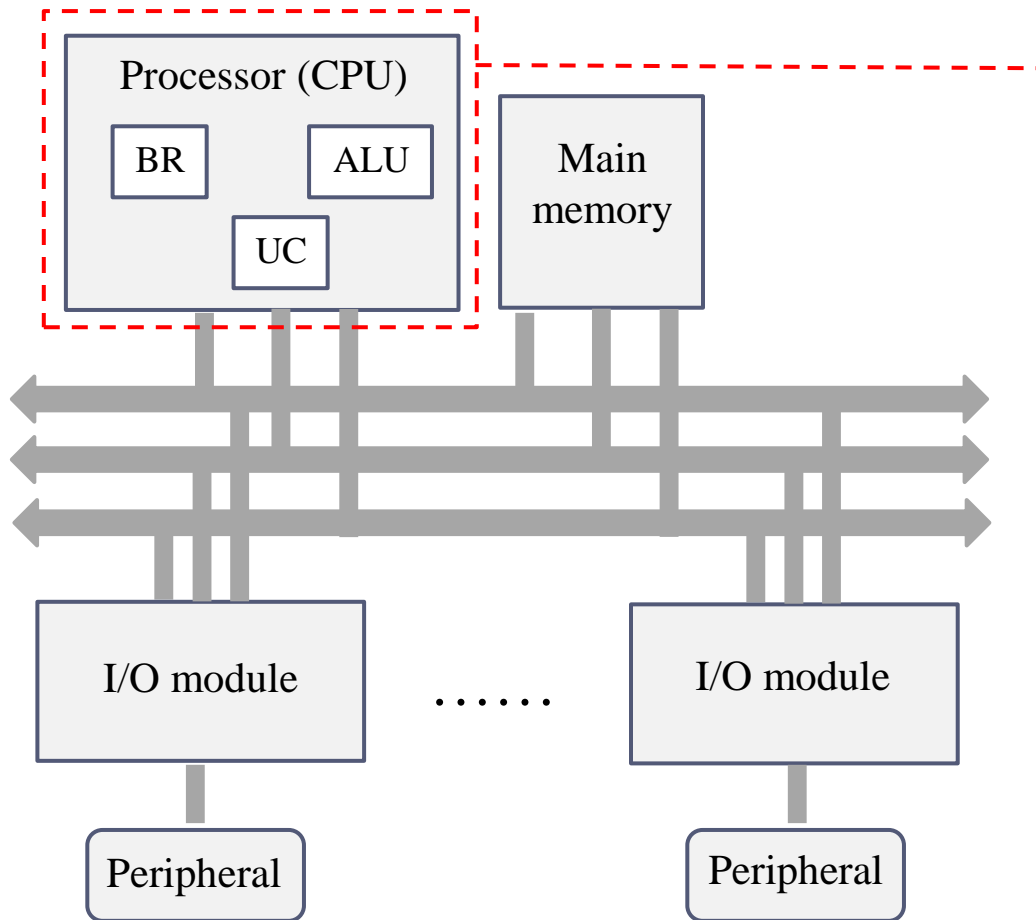
Computer components

Reminder



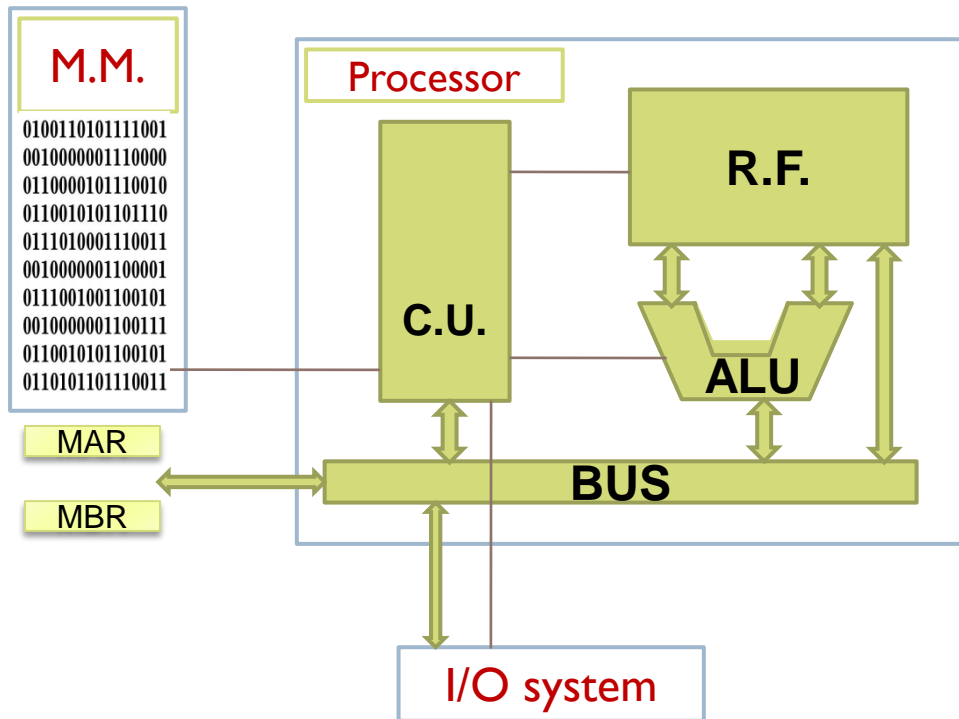
- ▶ Processor
- ▶ Main memory
- ▶ I/O module
 - ▶ Peripheral

Processor components



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

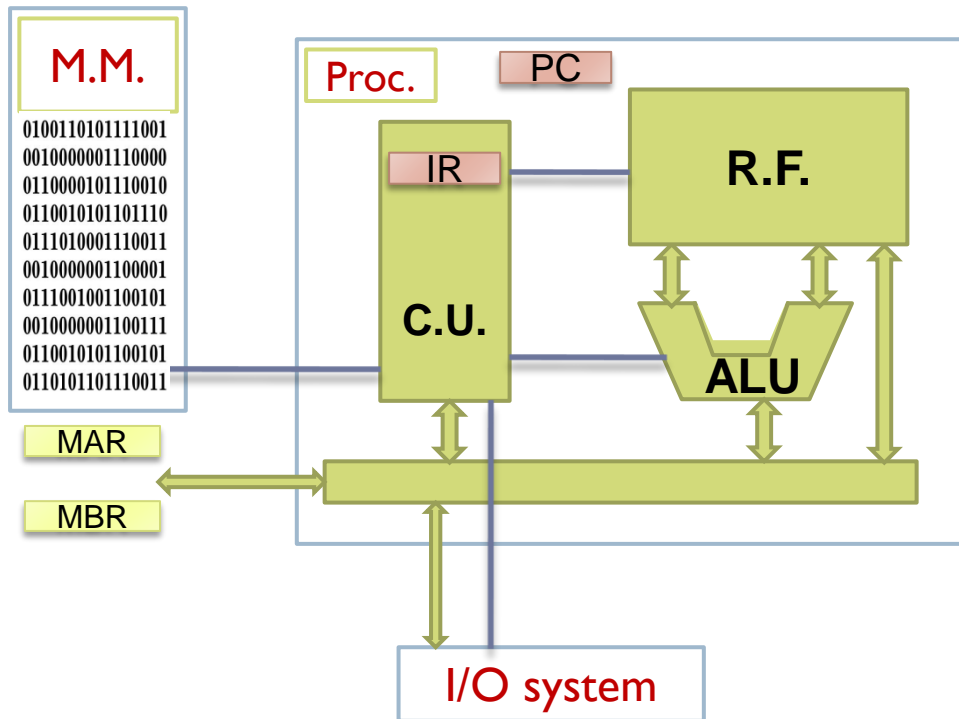
Introduction: motivation



- 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



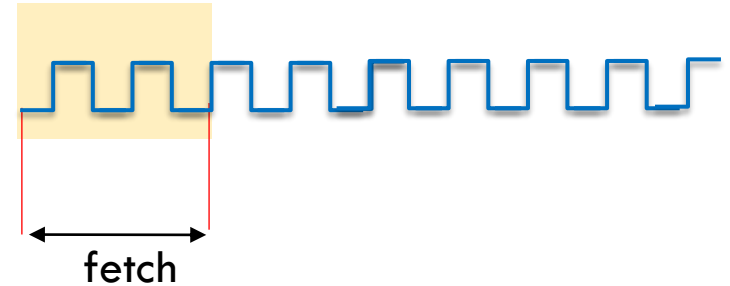
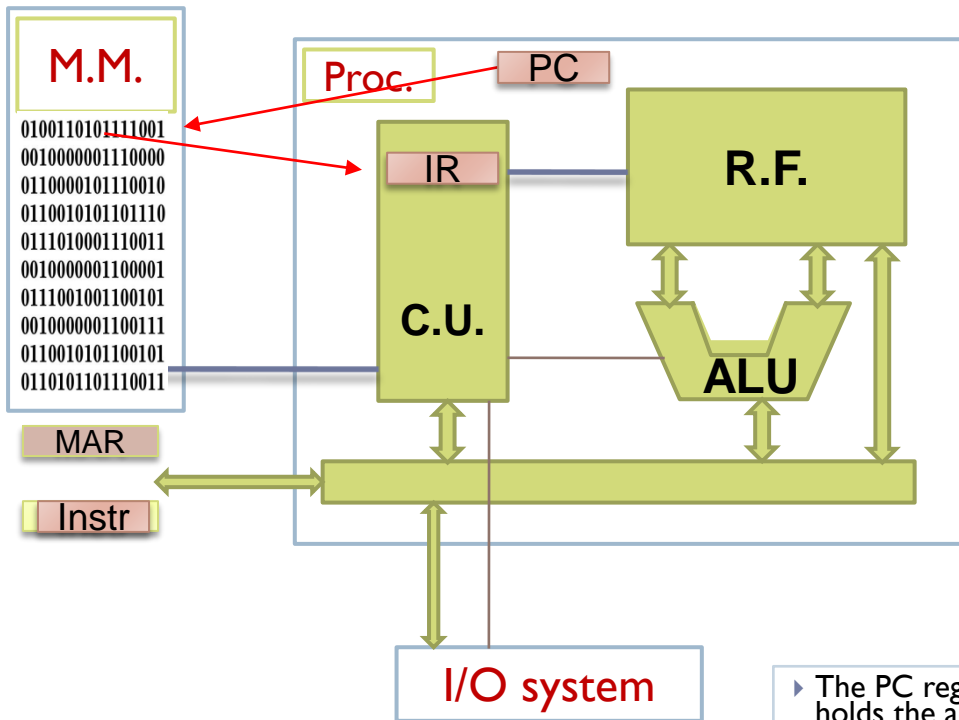
- Each element of the computer has inputs, outputs and control signals.
- At each clock cycle, the Control Unit (C.U.) sends the control signals via the control bus wires.
- Control signals indicate what value to output:
 - Move from an input to an output: $S = E_x$
 - Transform an input: $S = f(E)$

Contents

1. Computer elements
2. Processor organization
3. Control unit
4. Execution of instructions
5. Control unit design
6. Execution modes
7. Interrupts
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

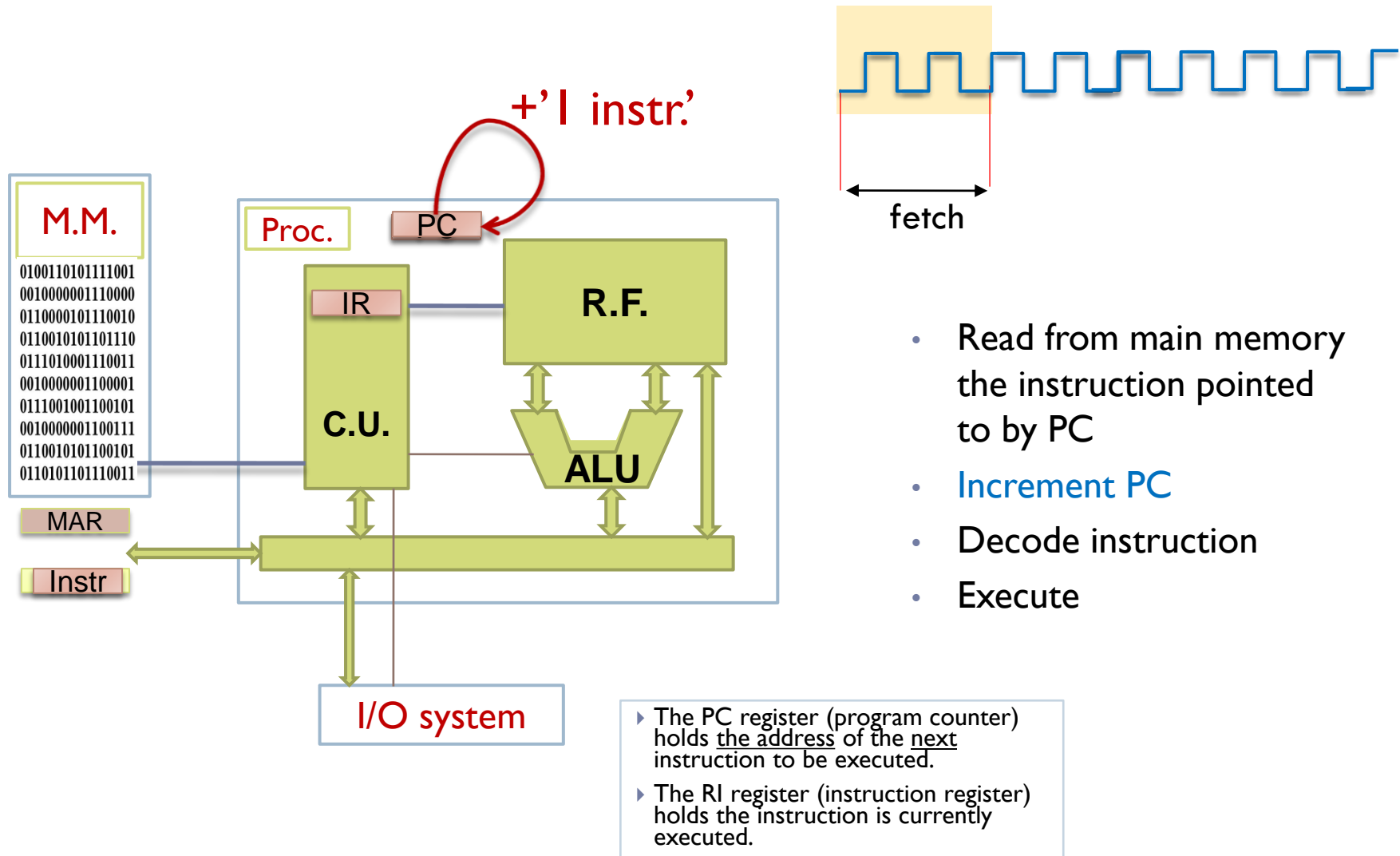
How C.U. works...



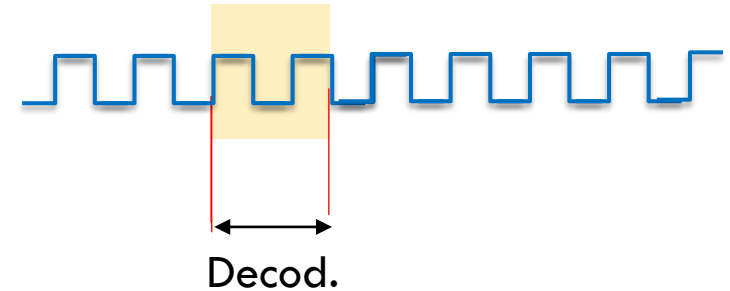
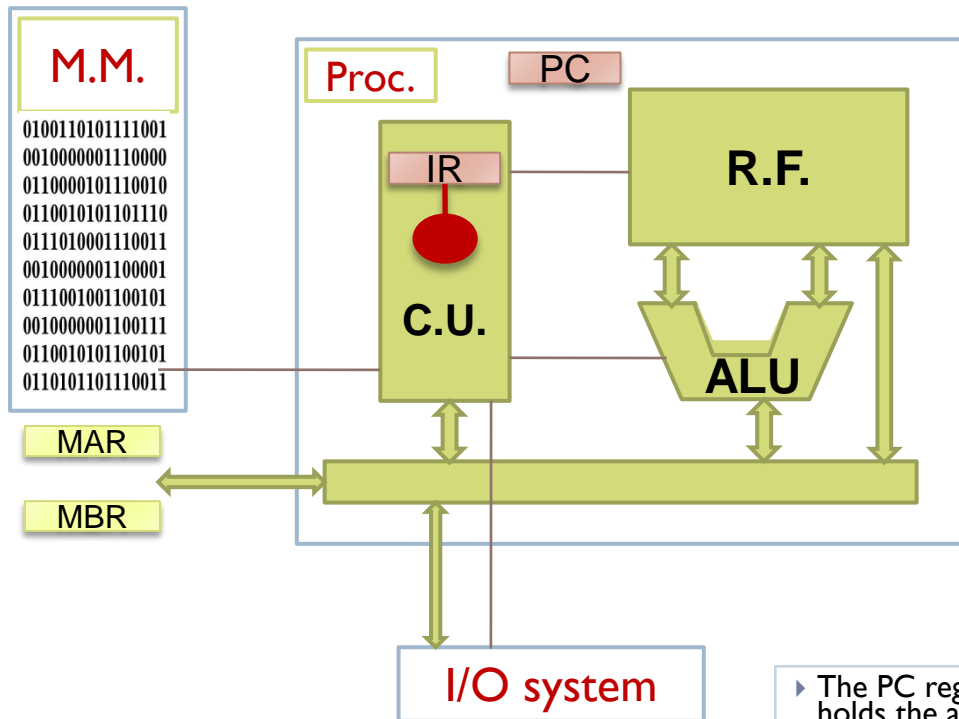
- Read from main memory the instruction pointed to by PC
- Increment PC
- Decode instruction
- Execute

- ▶ 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...



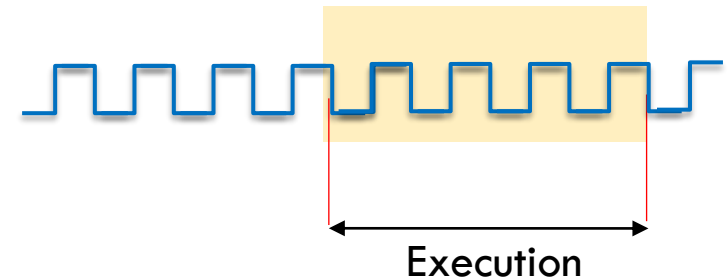
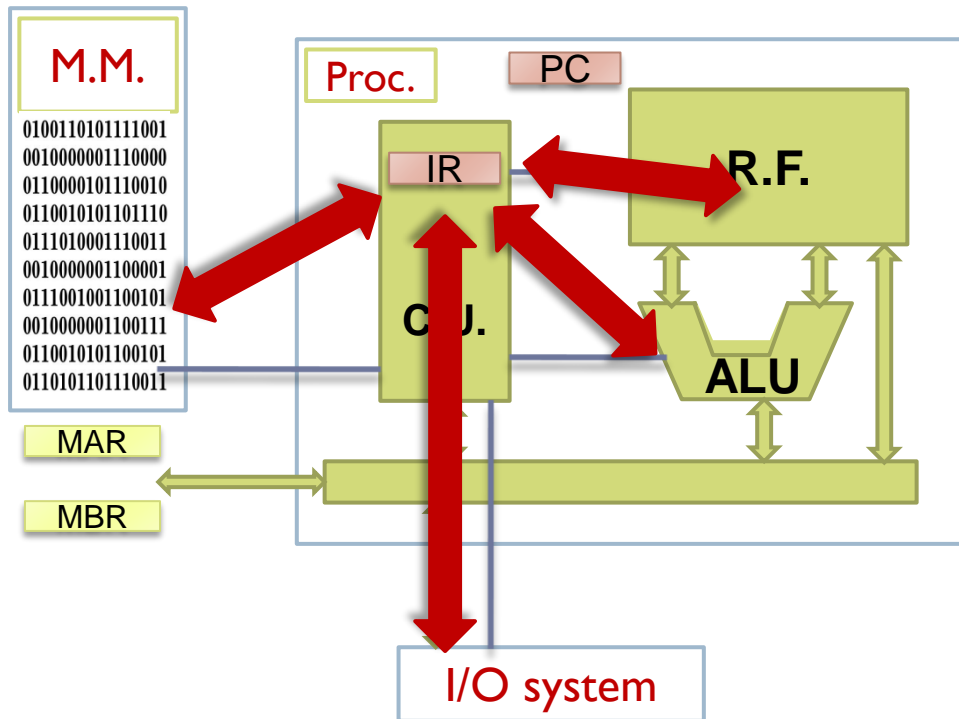
How C.U. works...



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

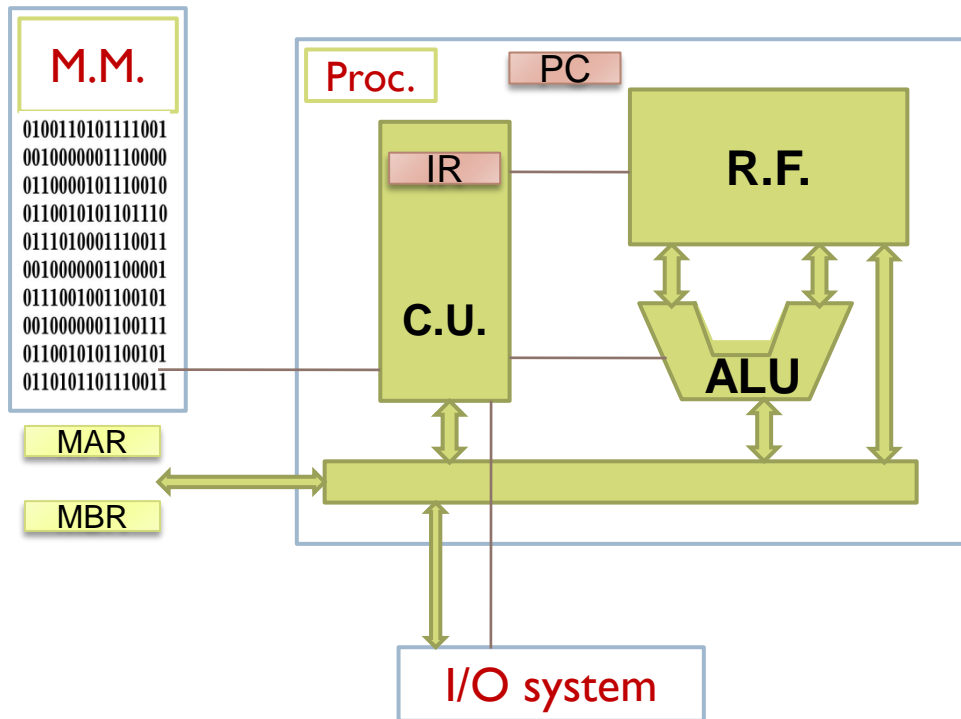
- ▶ The PC register (program counter) holds the address of the next instruction to be executed.
- ▶ The RI register (instruction register) holds the instruction is currently executed.

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.



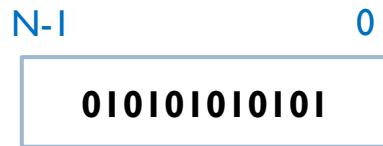
- Resolving anomalous situations
 - Illegal instructions
 - Illegal memory accesses
 - ...
- Attend to interruptions
- Control the communication with the peripherals.

Contents

1. Computer elements
2. Processor organization
3. Control unit
4. Execution of instructions
5. Control unit design
6. Execution modes
7. Interrupts
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



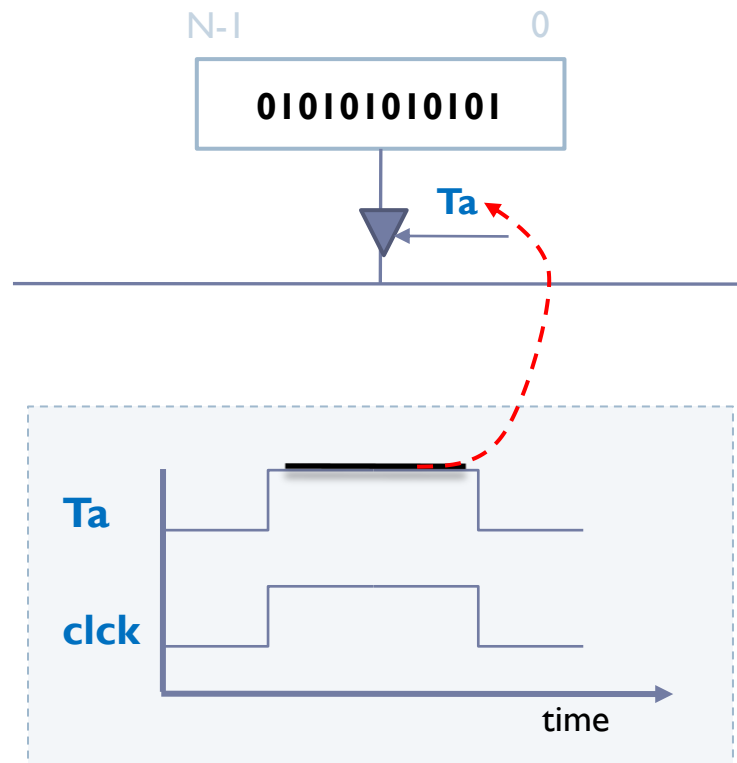
▶ Register

- ▶ 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



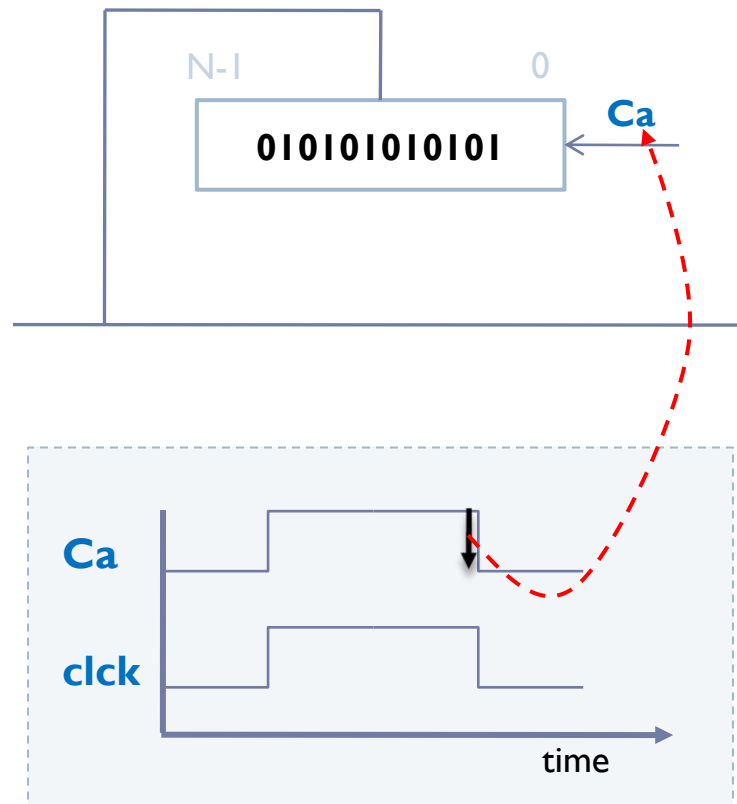
► Tri-state

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

► IMPORTANT

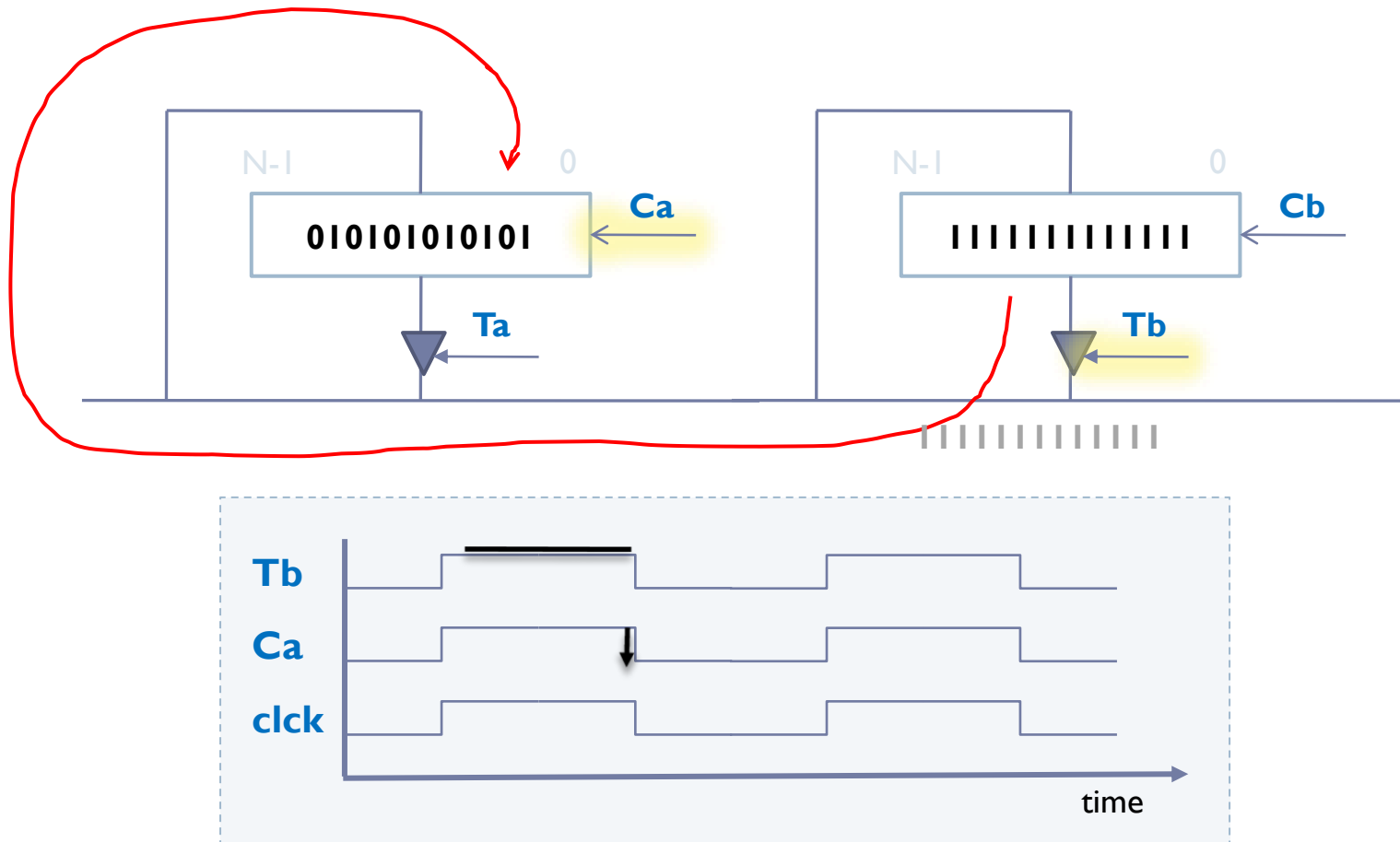
- Two or more tri-states cannot be activated on the same bus at the same time.

Signals: load in register

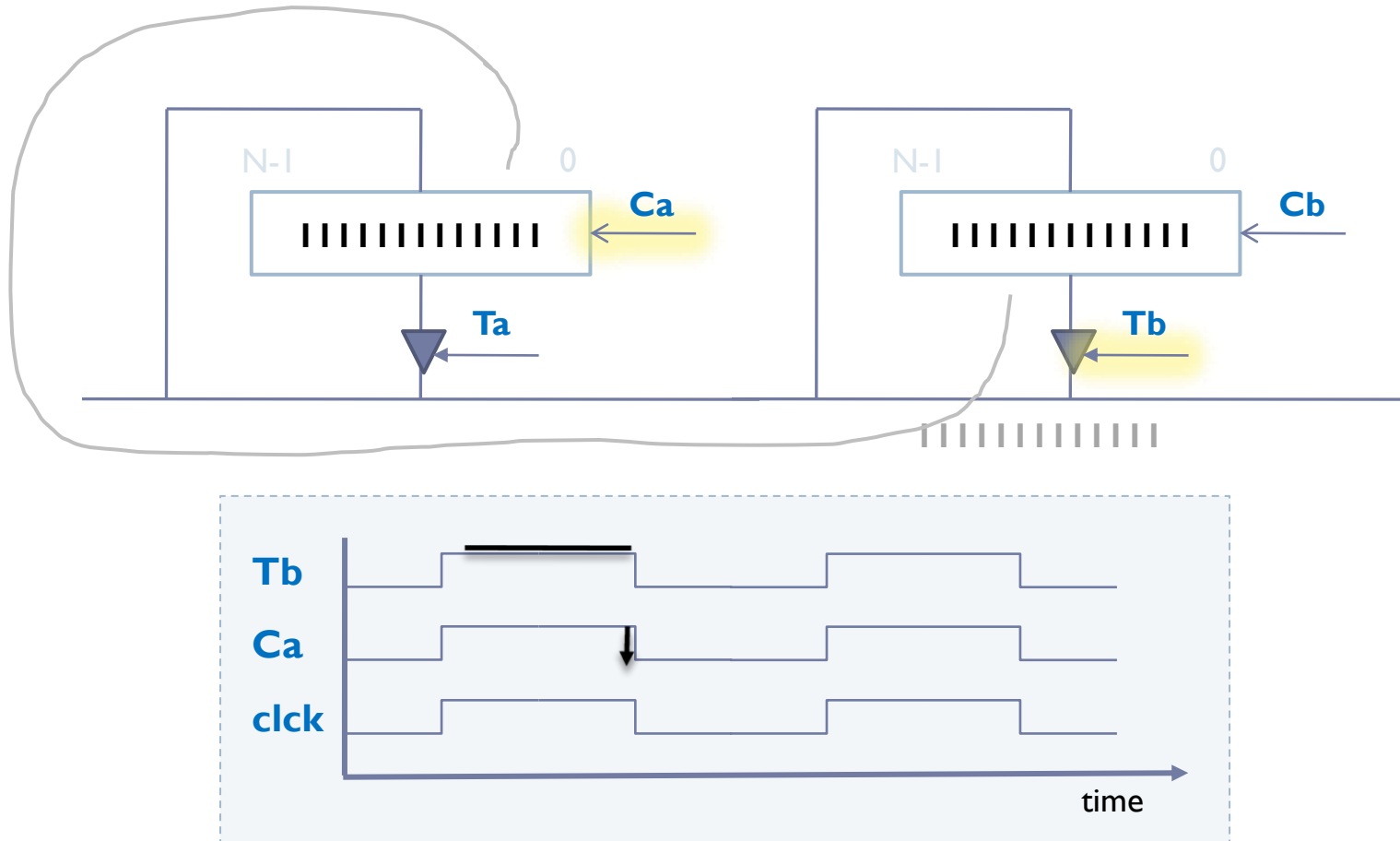


- ▶ Load in register
 - ▶ Let store the input value at the clock falling edge
 - ▶ During the clock level the register keeps the inner (old) value.
 - ▶ At the end of the clock cycle (falling edge) is when the inner value is updated ↓
- ▶ **IMPORTANT**
 - ▶ Therefore, in the following cycle, the new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

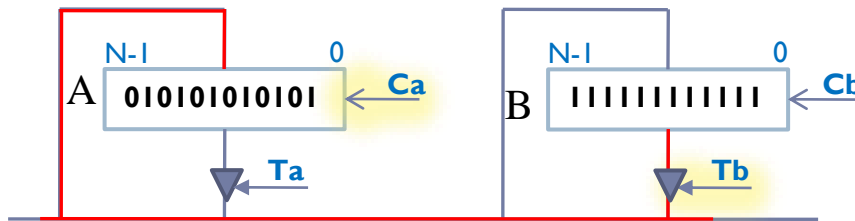
Sequence of signals



Sequence of signals



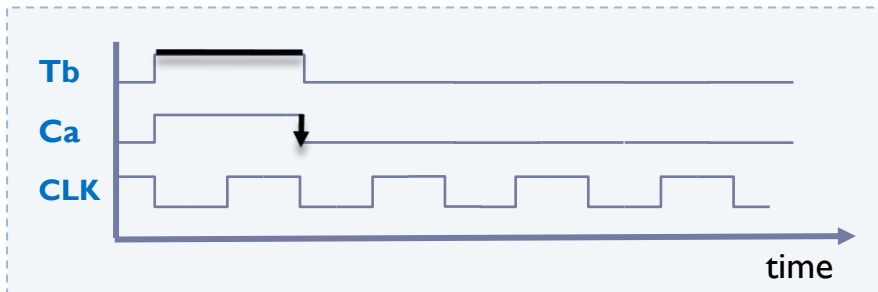
Example of *transfer* elemental operation



▶ Elementary transfer operation:

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

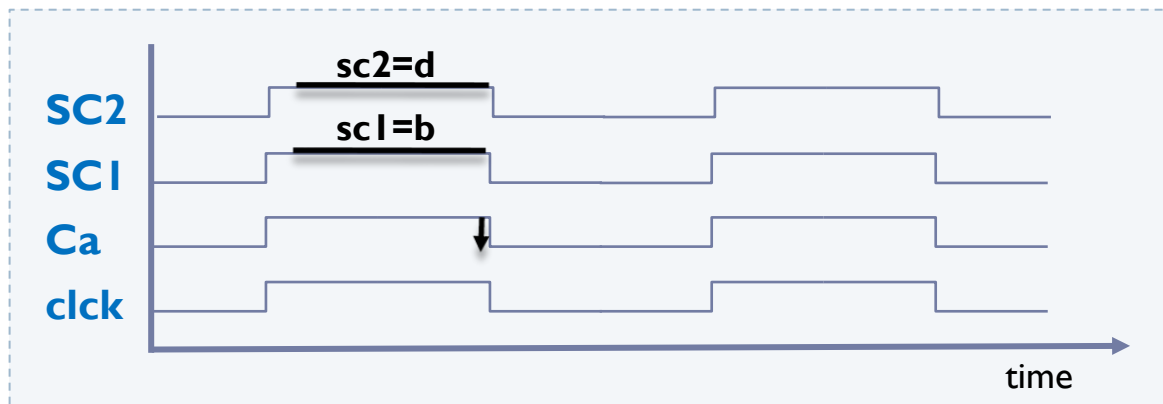
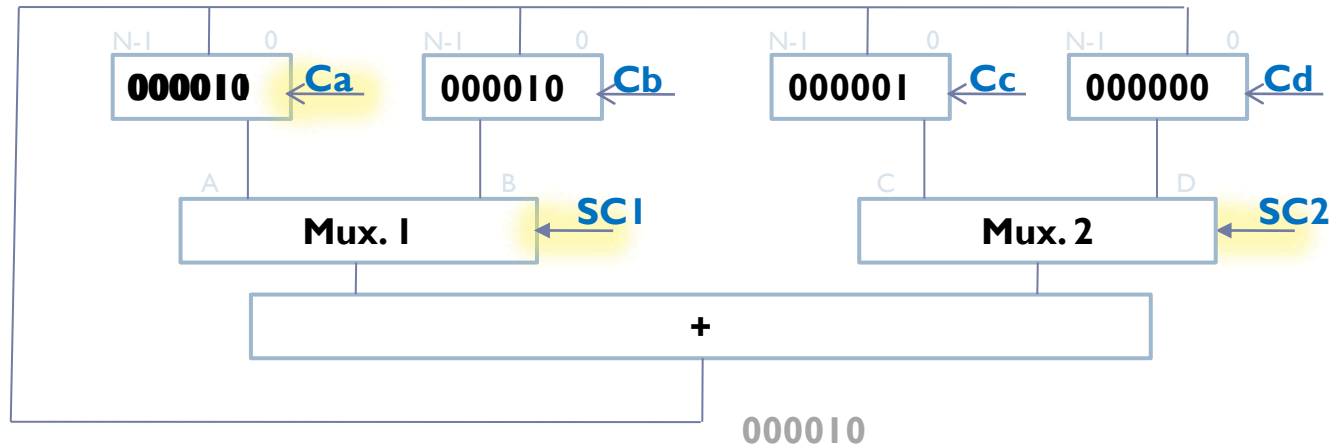
xx: $A \leftarrow B$ [T_b , C_a]



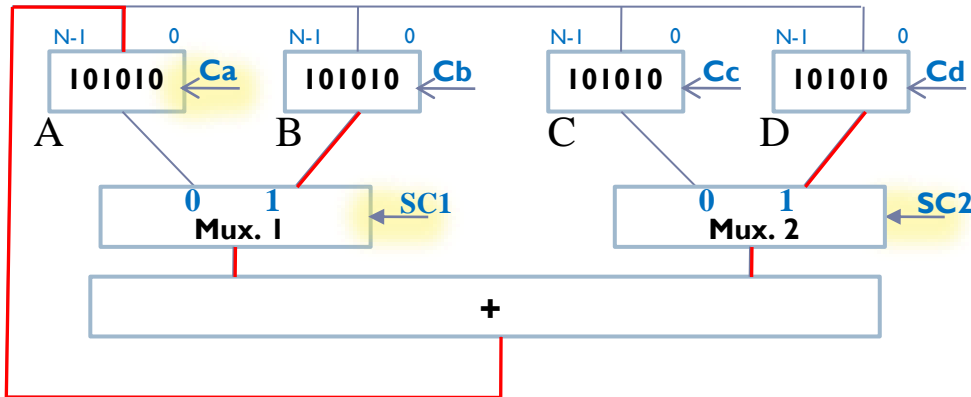
▶ IMPORTANT

- ▶ 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.
 - ▶ Use a non-existing circuitry

Sequence of signals



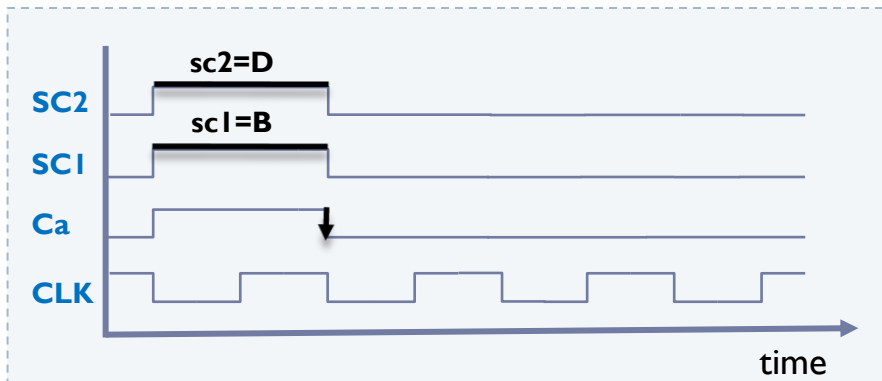
Example of *process* elemental operation



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.
 - ▶ Use a non-existing circuitry

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

- ▶ $R1 \leftarrow R2 + R2$



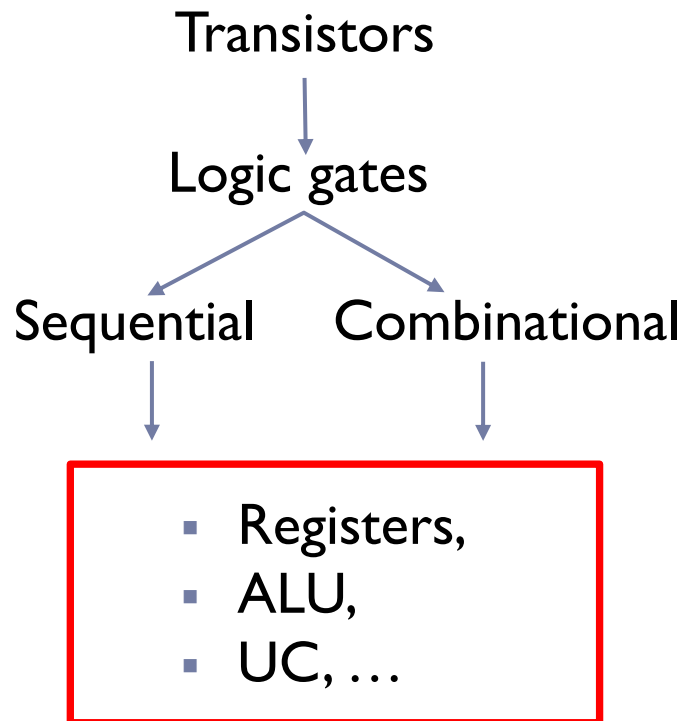
$Reg \leftarrow \varphi(Reg, Reg)$

Review all components...

From transistors...to elemental operations

Reminder

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

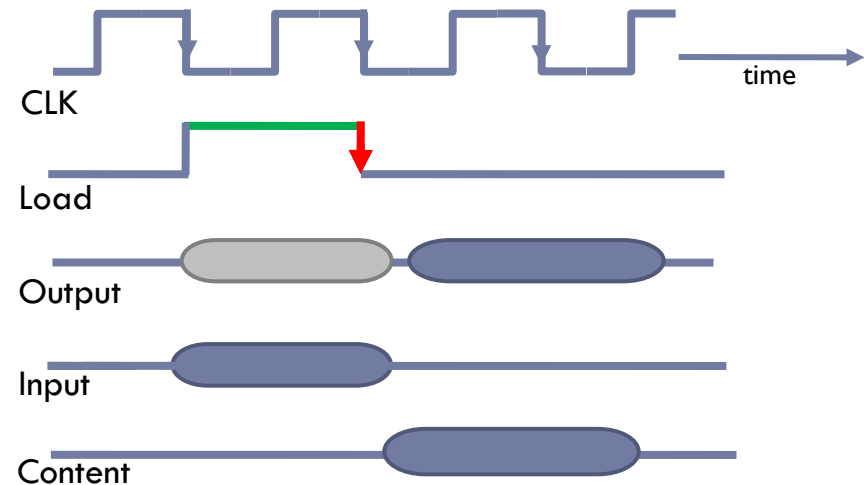
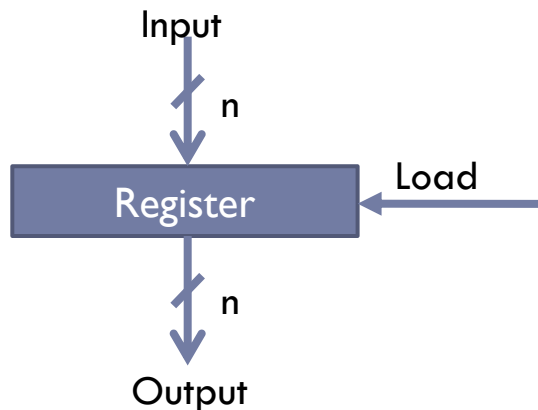


Review all components...

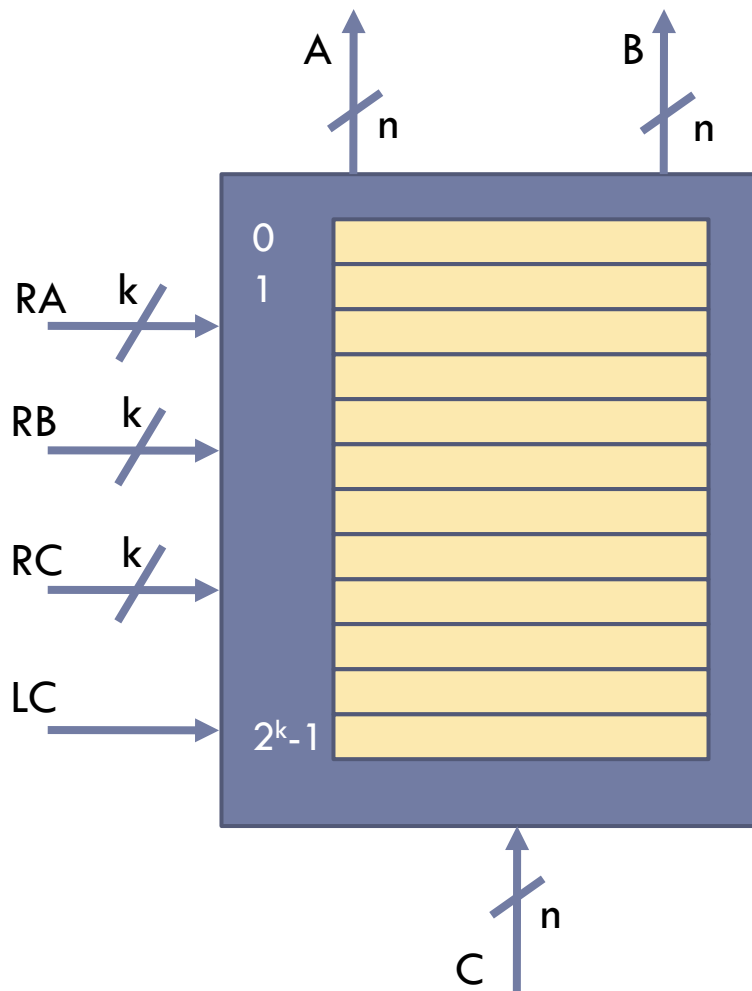
Registers

Reminder

- ▶ 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



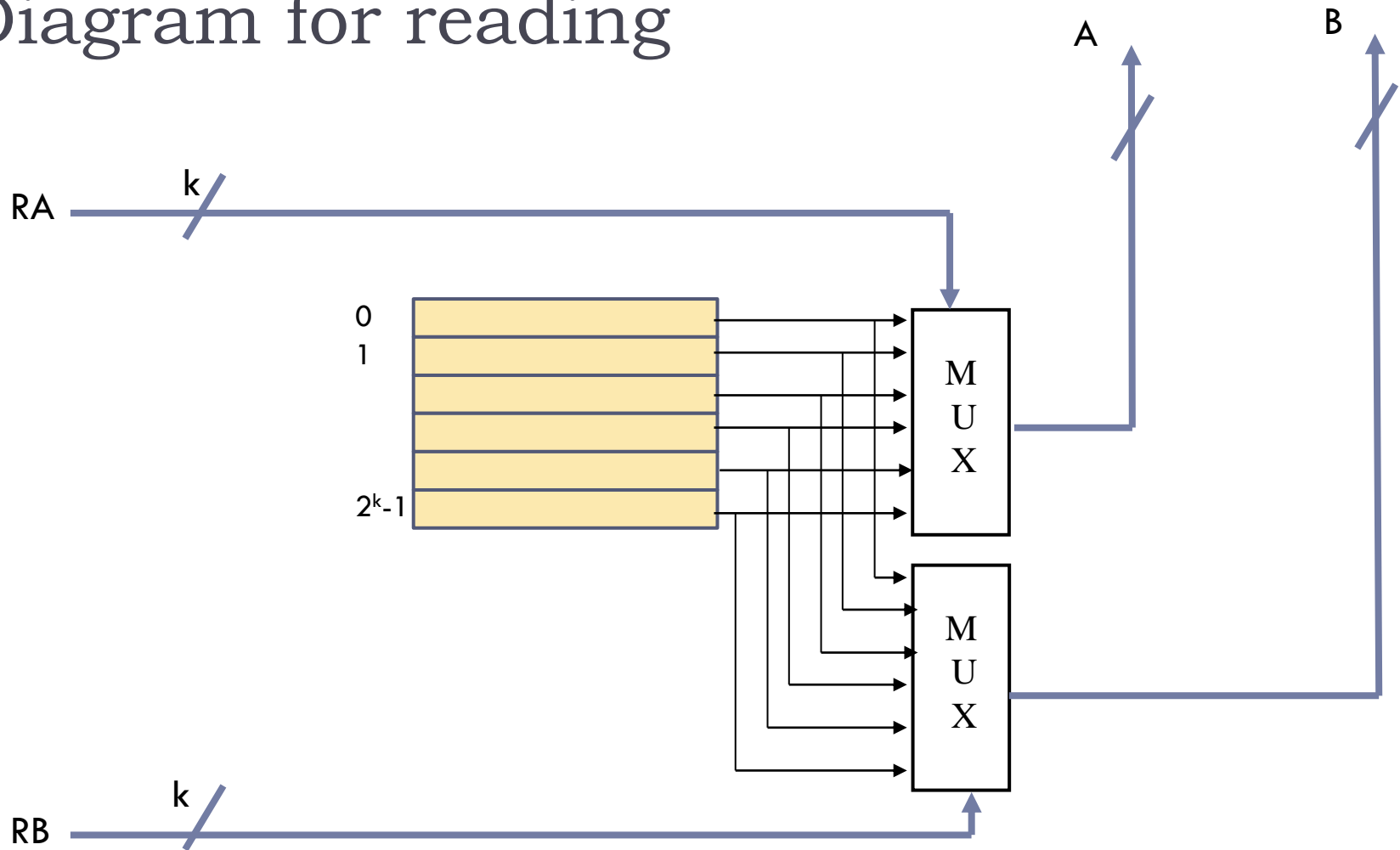
Register File (RF)



- ▶ Grouping of registers.
- ▶ Typically, the number of registers is a power of 2.
 - ▶ n registers $\rightarrow \log_2 n$ bits for select each register
 - ▶ k selection bits $\rightarrow 2^k$ registers
 - ▶ **E.g.: with 32 registers, $k=5$**
- ▶ Fundamental storage element.
 - ▶ Very fast access.

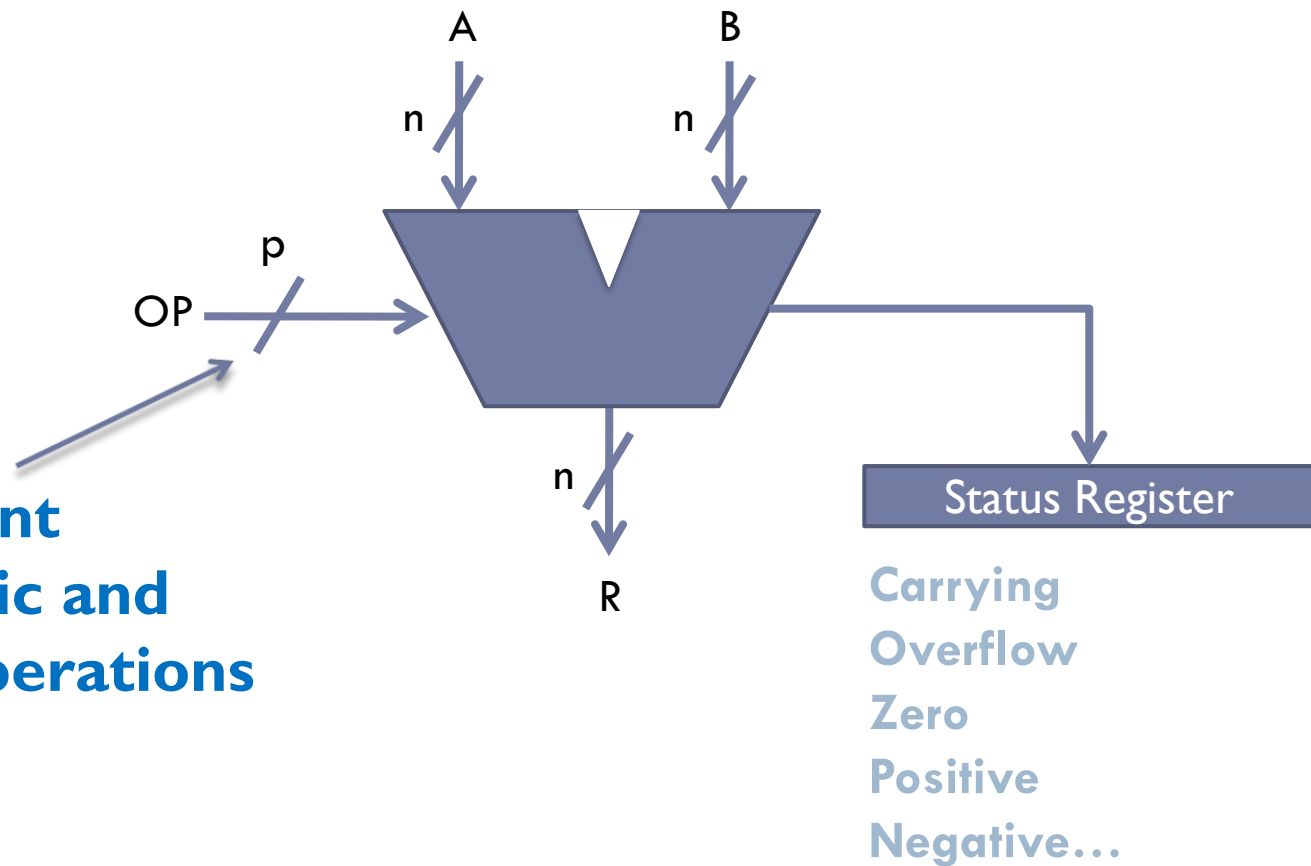
What value does RA have to have in order to get the contents of register 14 out of A?

Diagram for reading

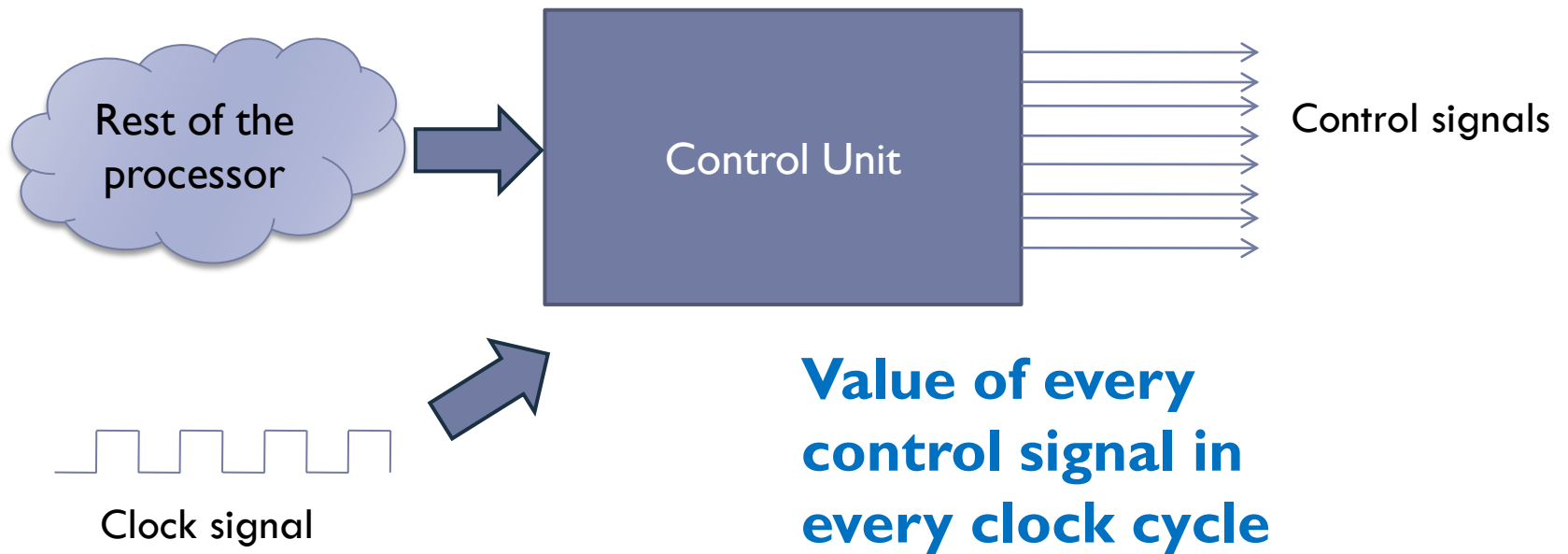


What value does RA have to have in order to get the contents of register 14 out of A ?

Arithmetic logic unit (ALU)



Control Unit (UC)



Contents

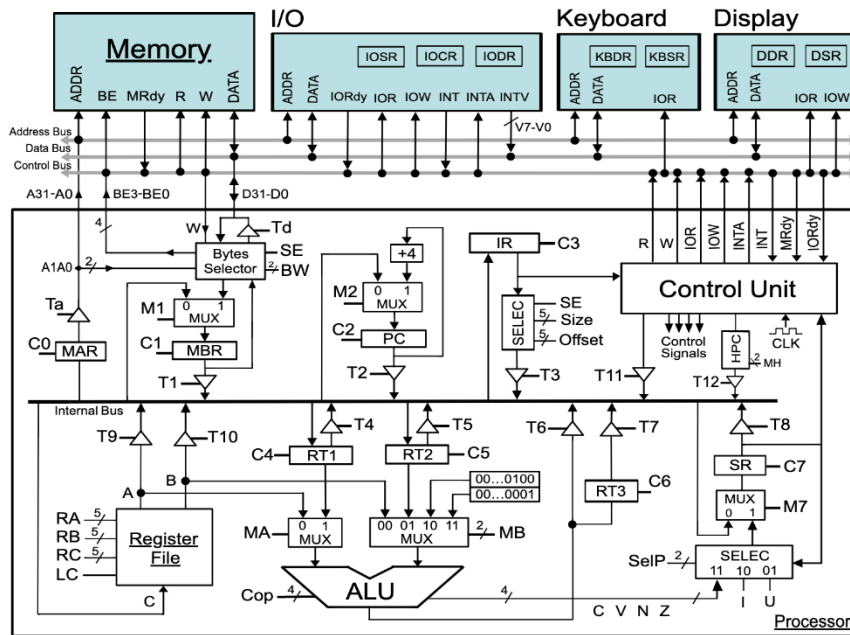
1. Computer elements
2. Processor organization
3. Control unit
4. Execution of instructions
5. Control unit design
6. Execution modes
7. Interrupts
8. Computer startup
9. Performance and parallelism

- 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 & WepSIM Simulator

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

► Elemental Processor (E.P.):

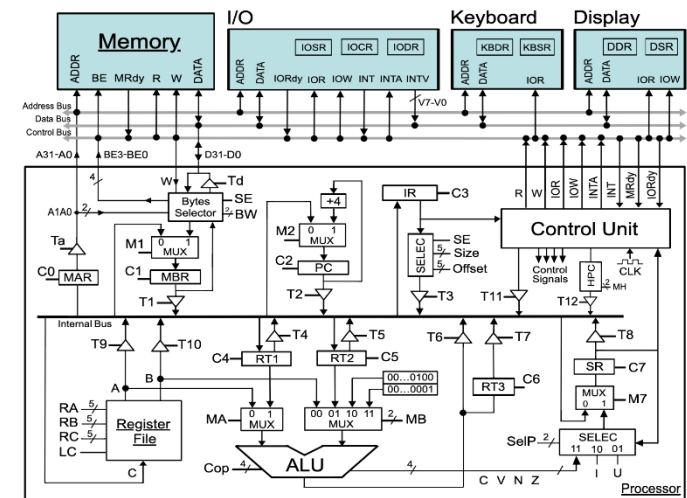


► WepSIM simulates the E.P.:

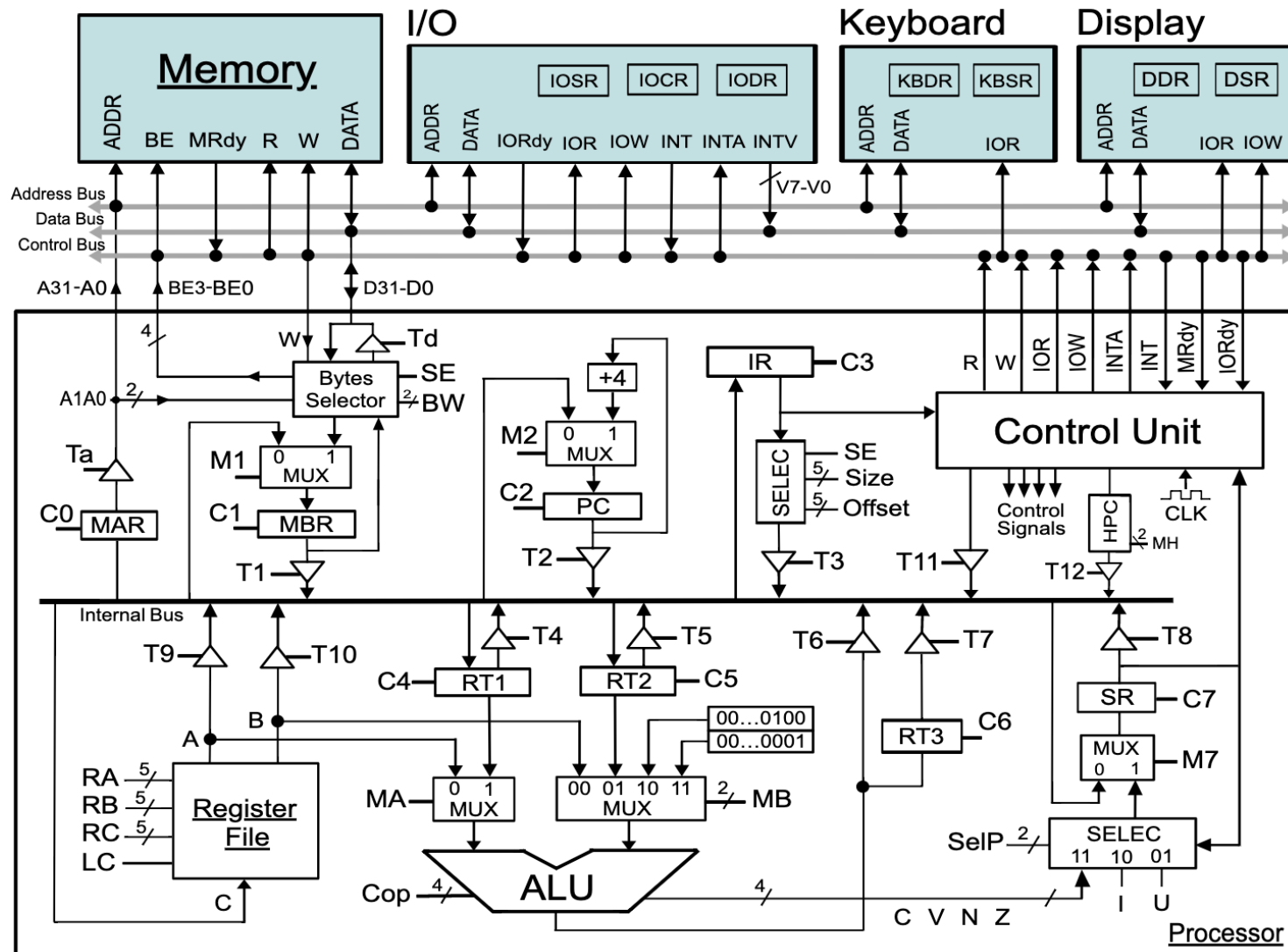
► <https://wepsim.github.io/wepsim/>

Main features

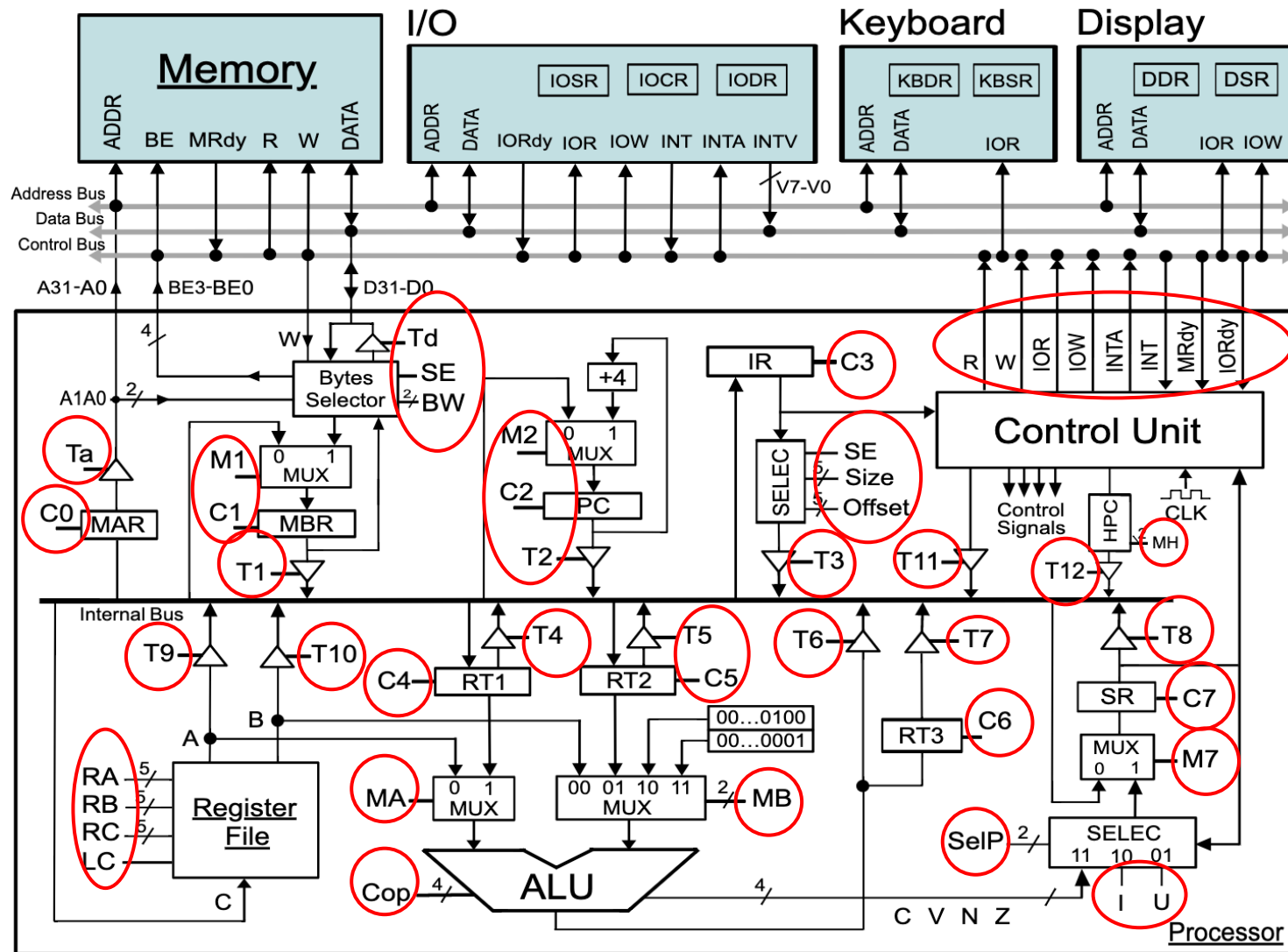
- ▶ **Elemental Processor (EP):**
 - ▶ 32-bits computer
 - ▶ Main memory:
 - ▶ Addressed by bytes
 - ▶ Reading and writing operations: 1 clock cycle
 - ▶ Register file with 32 registers (R0...R31) of 32-bits each
 - ▶ For RISC-V: R0 = 0 y SP = R2
 - ▶ For MIPS: R0 = 0 y SP = R29
 - ▶ Control registers (PC, IR, ...) status register (SR) and not visible to programmers (RT1...RT3) registers
- ▶ WepSIM simulator implements the E.P.:
 - ▶ <https://wepsim.github.io/wepsim/>



Structure of an elementary computer



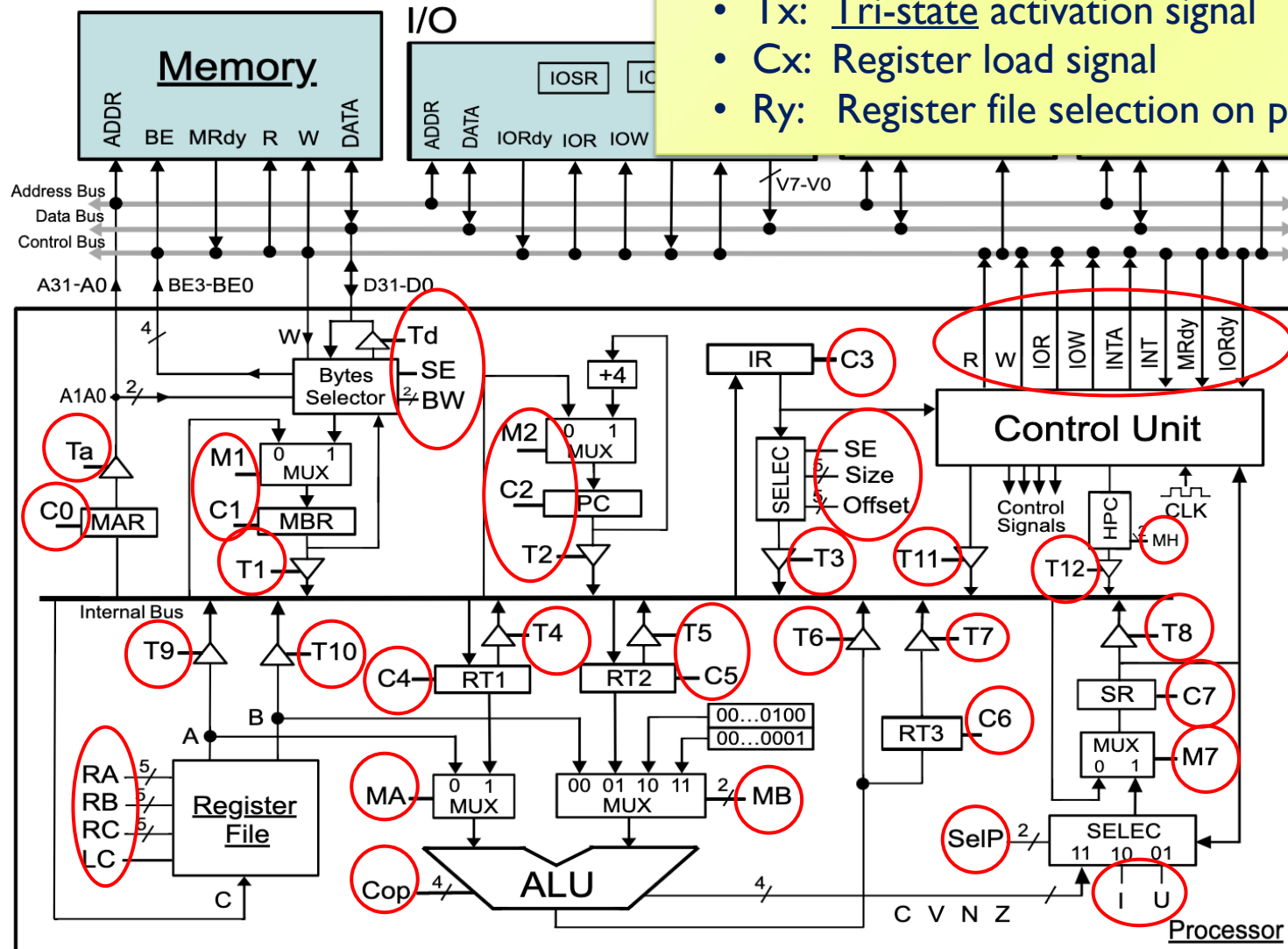
Control signals



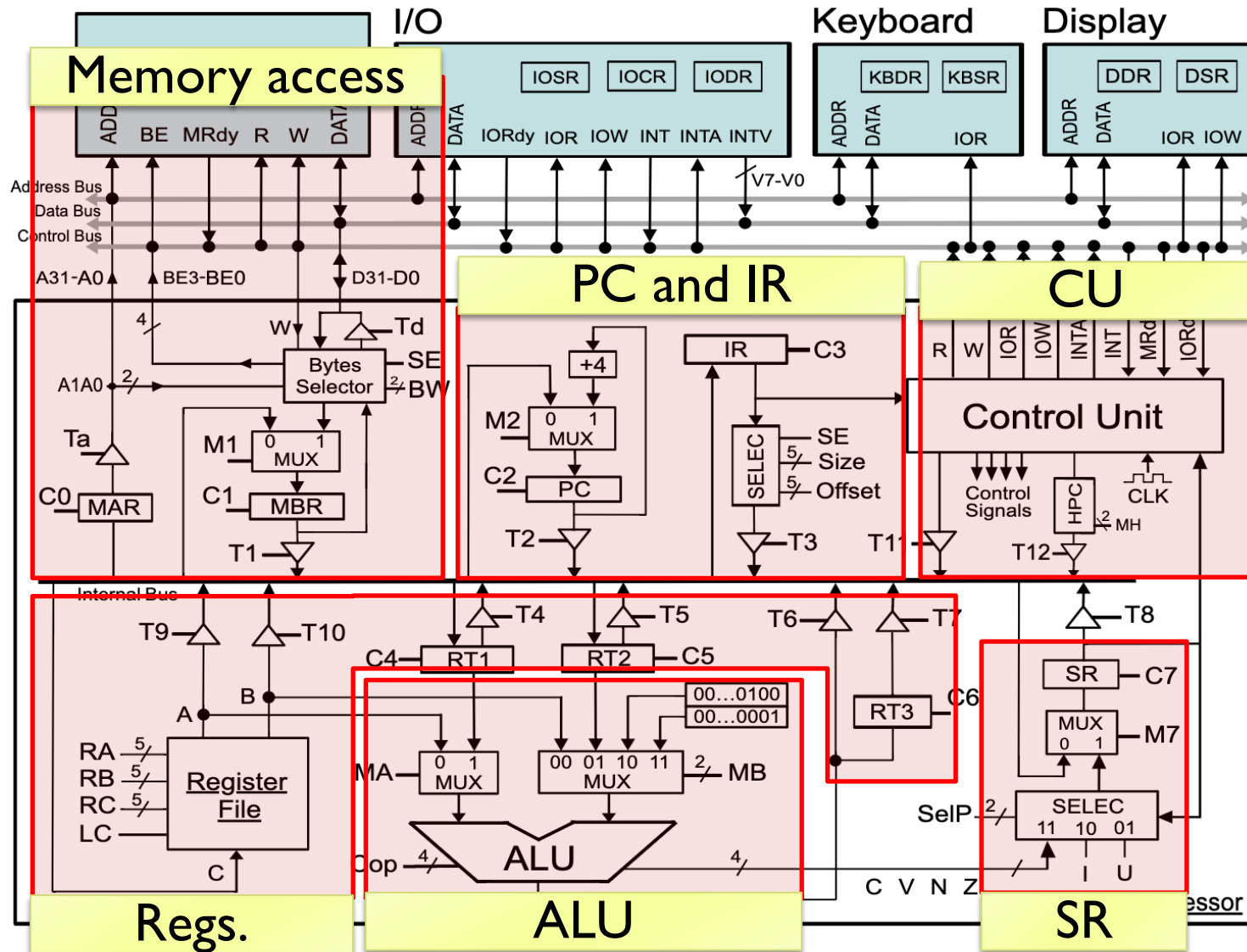
Control signals

General nomenclature:

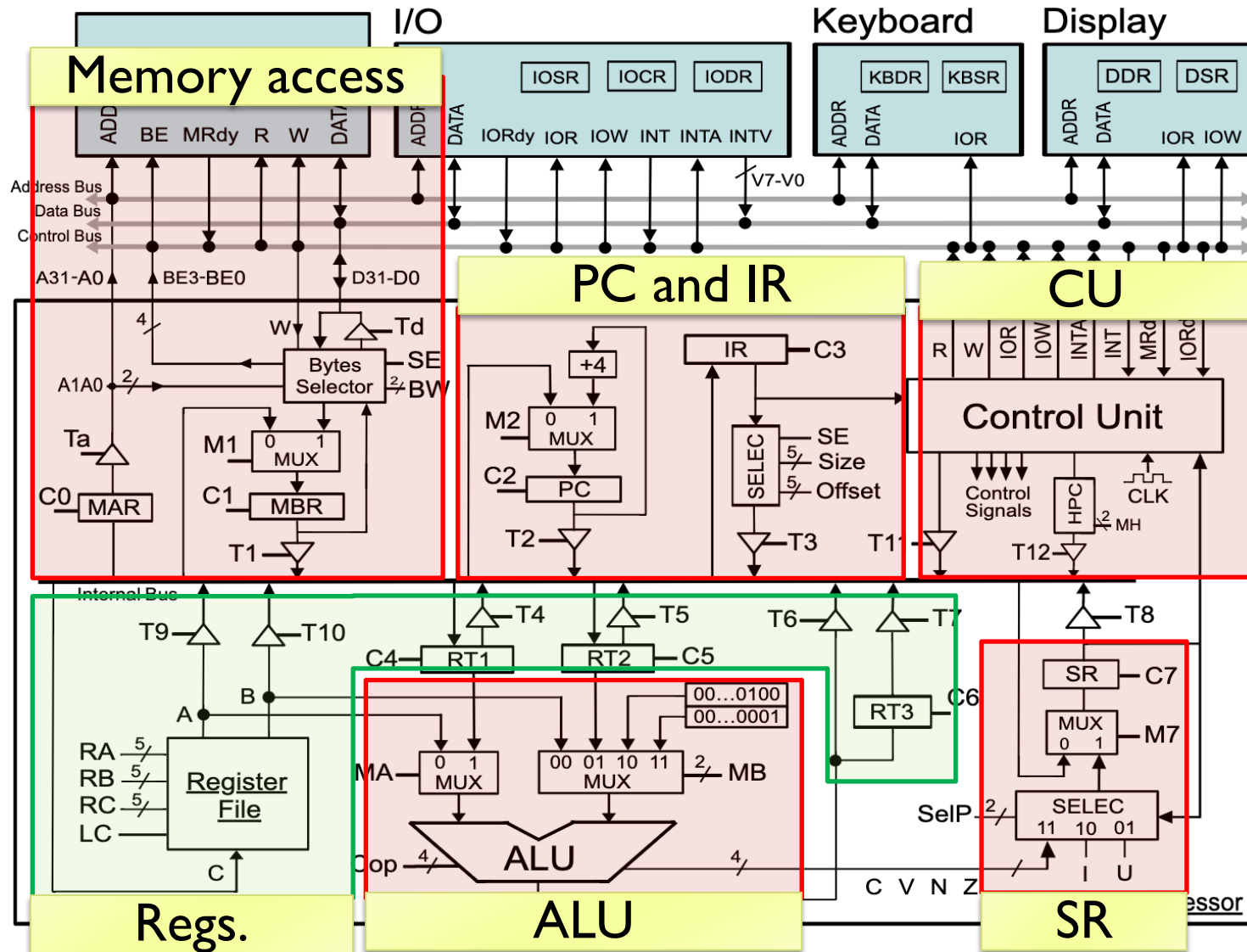
- Mx: Selection in multiplexor
- Tx: Tri-state activation signal
- Cx: Register load signal
- Ry: Register file selection on point y



Elemental Processor: control signals

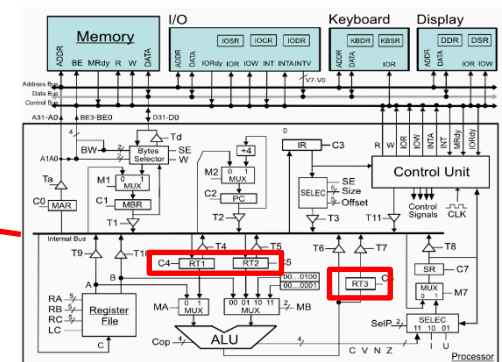
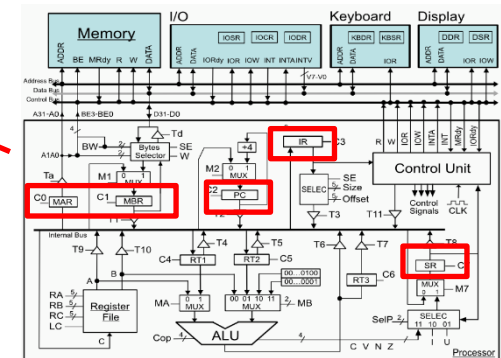
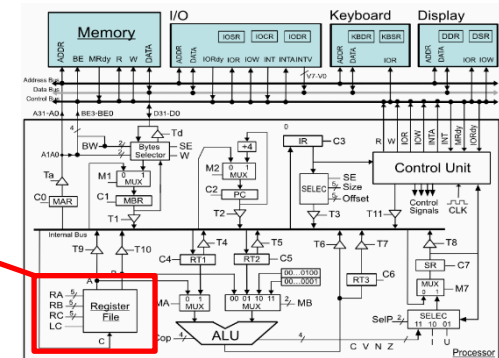


Elemental Processor: control signals



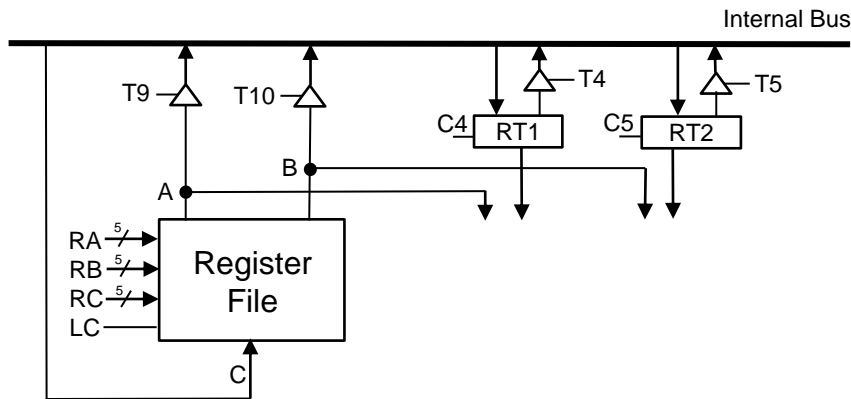
Registers

- ▶ Registers visible to programmers:
 - ▶ Register file's registers (E.g. : 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 users:
 - ▶ RT1, RT2 and RT3 (internal temporal reg.)



Control signals

Registers



Nomenclature:

- Ry: Register file selection for A/B/C
- Tx: Tri-state activation signal
- Cx: Register load signal

▶ Register file

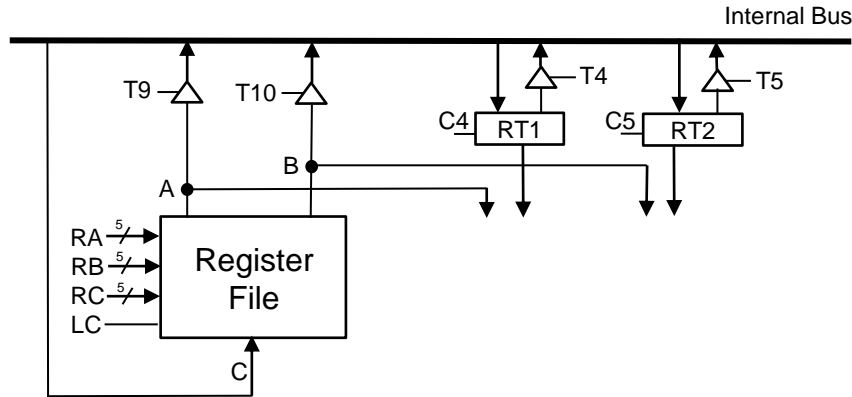
- ▶ RA – register output by A
- ▶ RB – register output by B
- ▶ RC – input C to the RC register
- ▶ LC – activates writing for RC
- ▶ T9 - copy A to the internal bus
- ▶ T10 - copy B to the internal bus

▶ RT1 and RT2

- ▶ 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

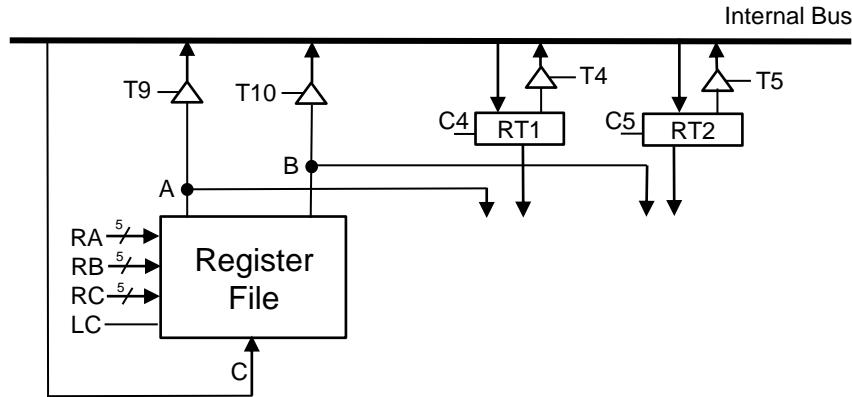
elemental operations in registers



► **SWAP R1 R2**

Example

elemental operations in registers

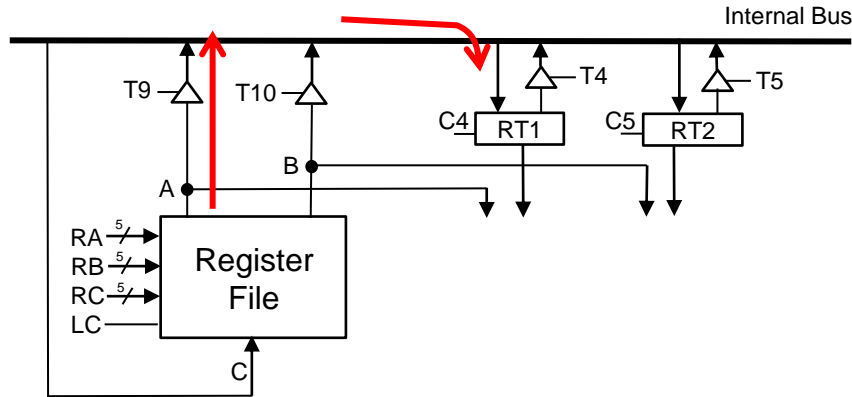


► **SWAP R1 R2**

Elemental Op.	Signals

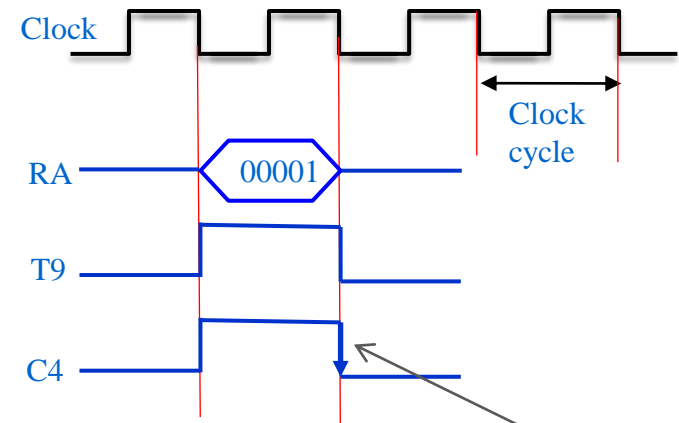
Example

elemental operations in registers



► SWAP R1 R2

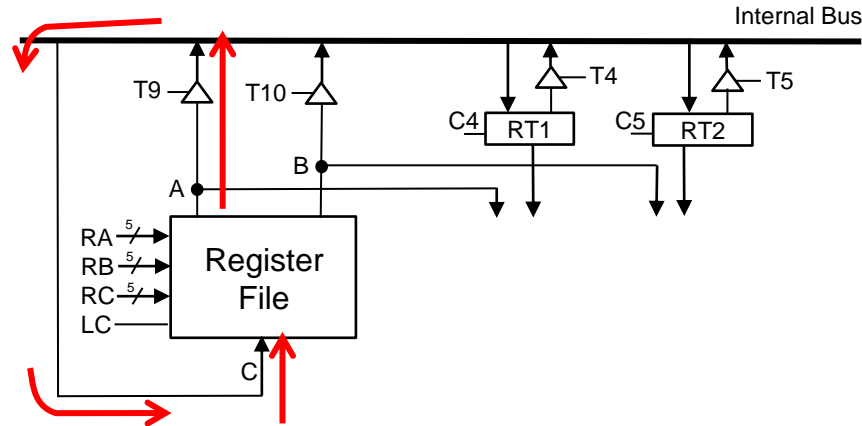
Elemental Op.	Signals
$RT1 \leftarrow R1$	$RA=00001, T9, C4$



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

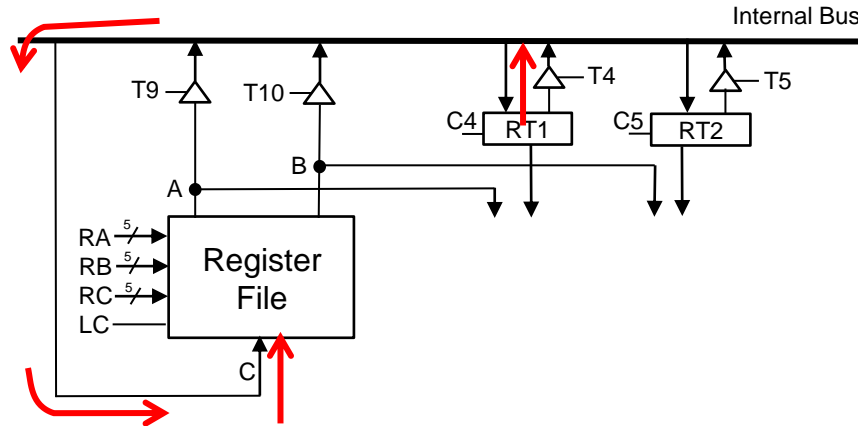


► SWAP R1 R2

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

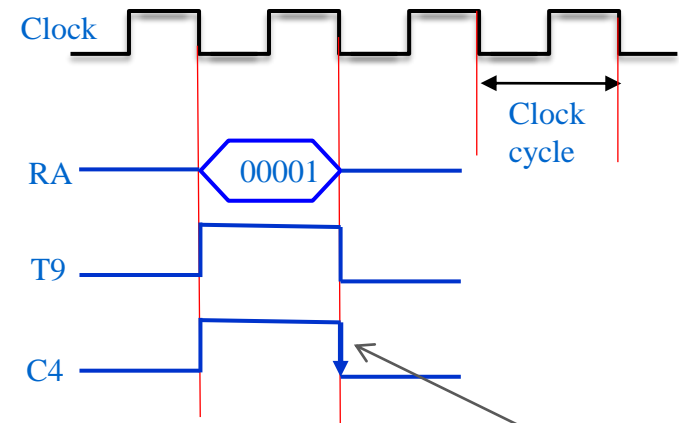
Example

elemental operations in registers



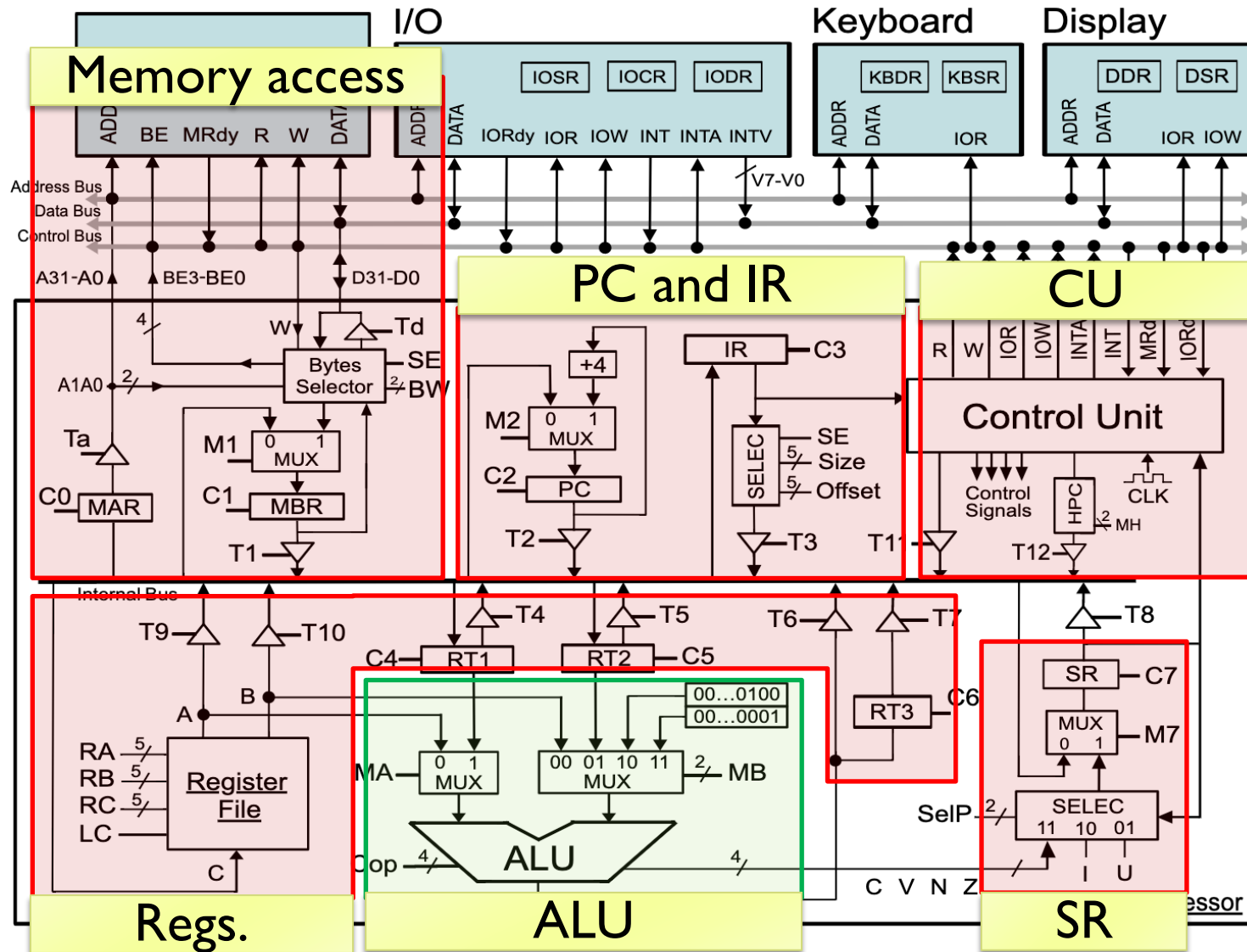
► SWAP R1 R2

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

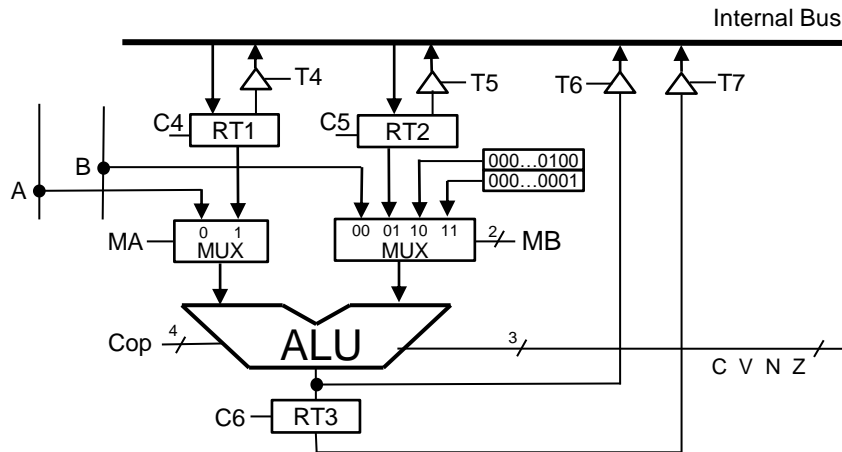


The data is loaded on RT1 on the falling edge.
It will be available on RT1 during the **next** cycle.

Elemental Processor: control signals



Control Signals

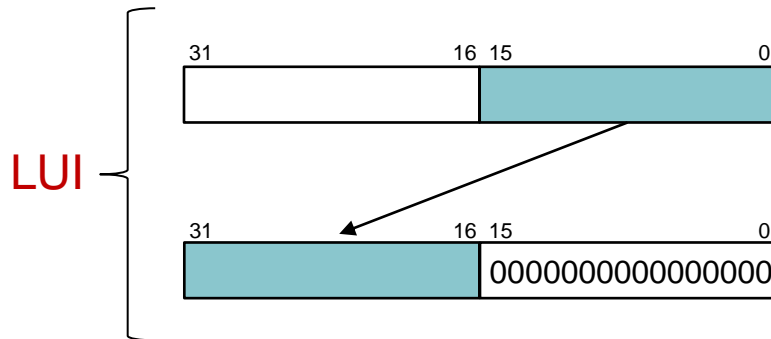
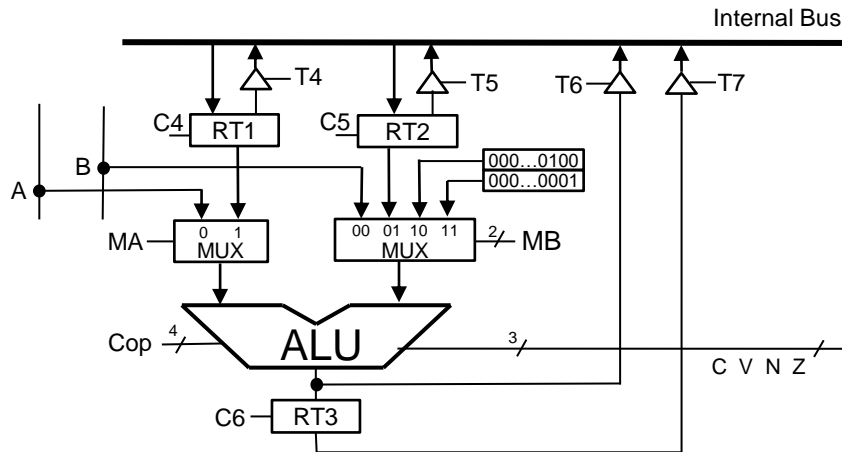


▶ ALU

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

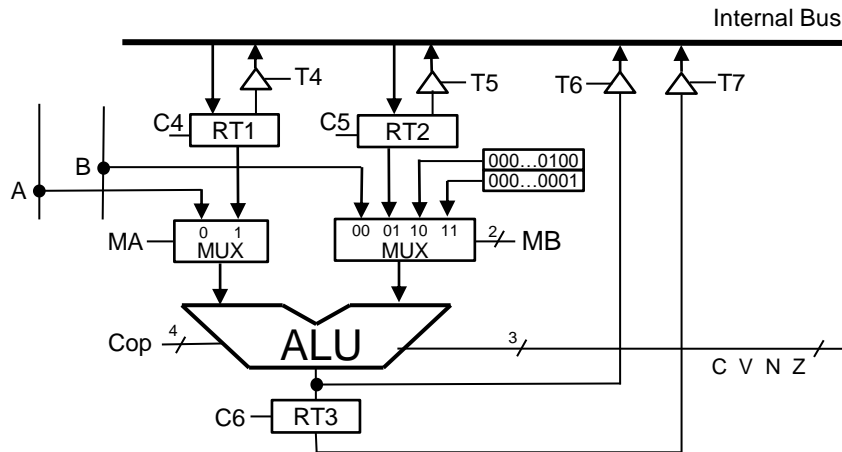
Cop (Cop ₃ -Cop ₀)	Operation
0000	NOP
0001	A and B
0010	A or B
0011	not (A)
0100	A xor B
0101	Shift Right Logical (A) B= number of bits to shift
0110	Shift Right Arithmetic (A) B= number of bits to shift
0111	Shift left (A) B= number of bits to shift
1000	Rotate Right (A) B= number of bits to rotate
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

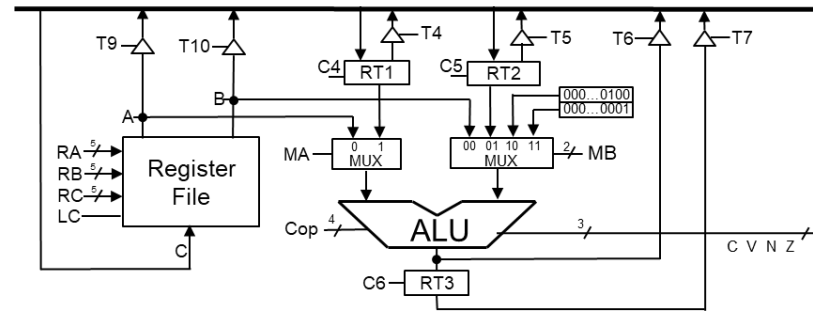


Result	C	V	N	Z
Positive result (0 is considered +)	0	0	0	0
Result == 0	0	0	0	1
Negative result	0	0	1	0
Overflow	0	1	0	0
Division by zero	0	1	0	1
Carrying at bit 32	1	0	0	0

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

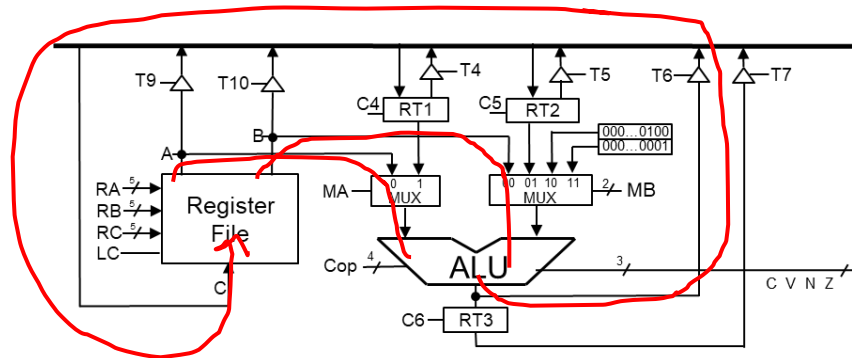


► ADD R3 R1 R2

Elem. Op.	Signals

Example

elemental operations in ALU

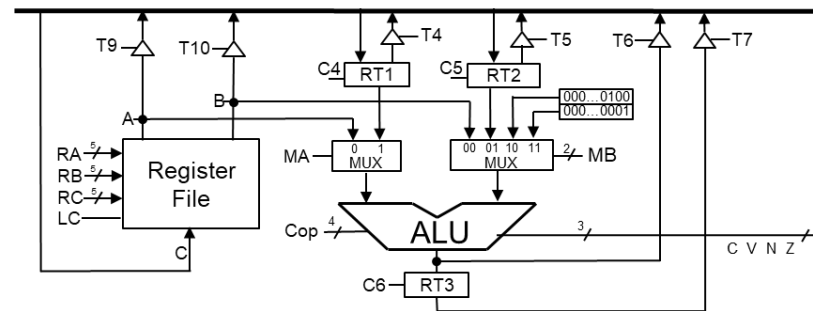


► ADD R3 R1 R2

Elem. Op.	Signals

Example

elemental operations in ALU

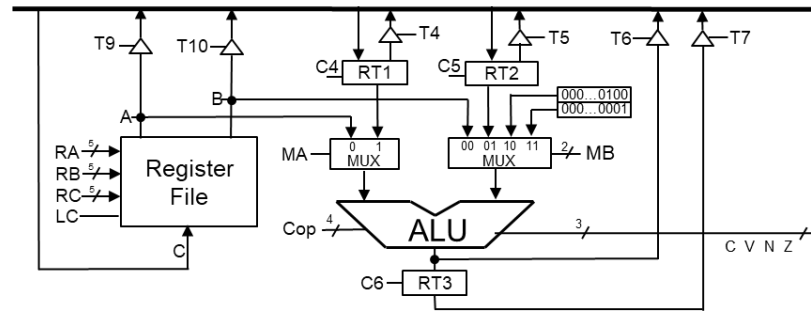


► ADD R3 R1 R2

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

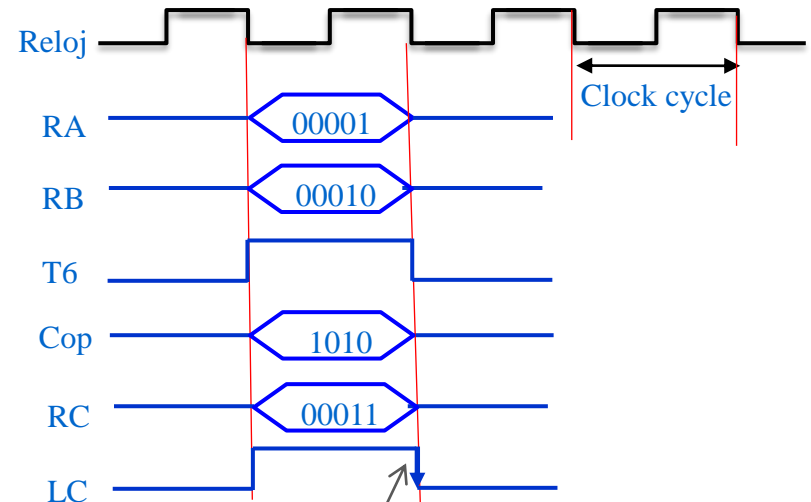
Example

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



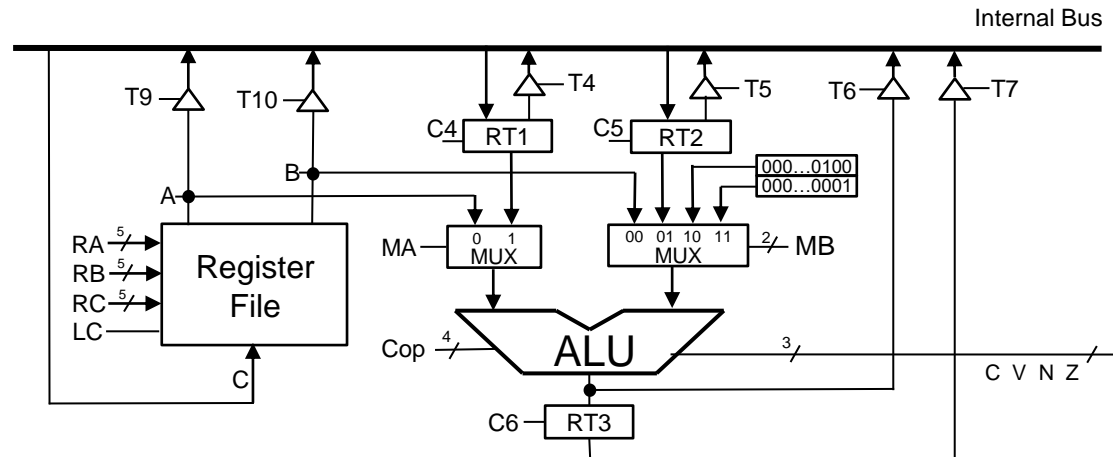
Rest of signals at 0.

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

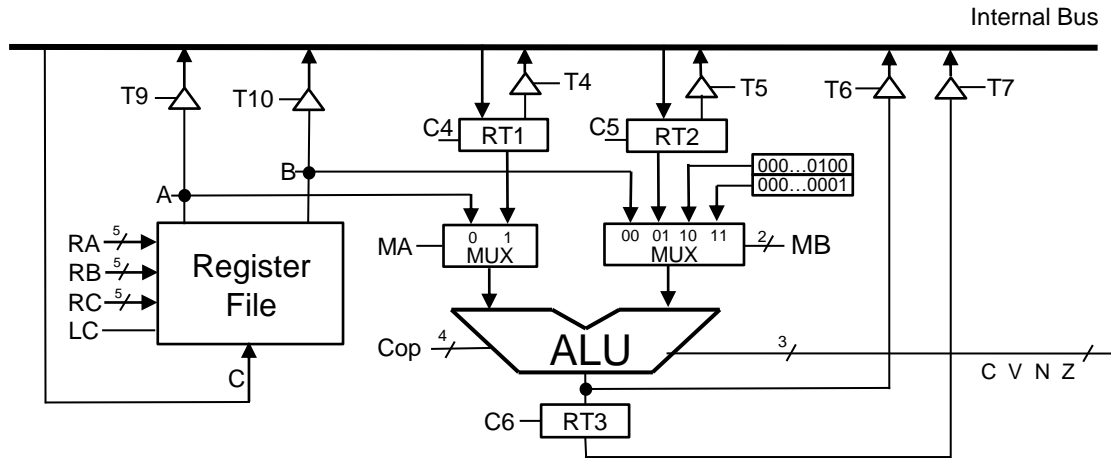


► SWAP R1 R2

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}

elemental operations in ALU



► **SWAP R1 R2**

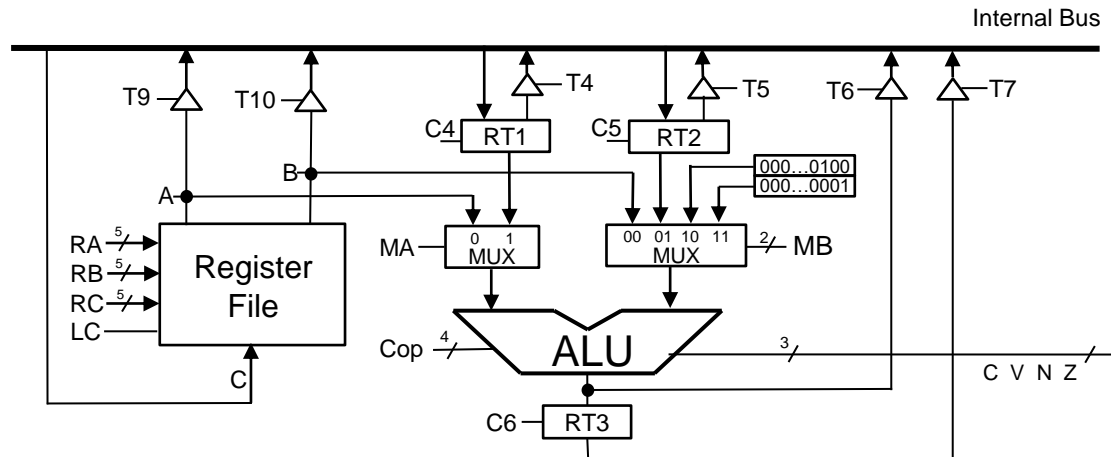
Elem. Op.	Signals
$RT1 \leftarrow R1$	RA=1, T9, C4
$R1 \leftarrow R2$	RA=2, T9, RC=1, LC
$R2 \leftarrow RT1$	T4, RC=2, LC

► 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

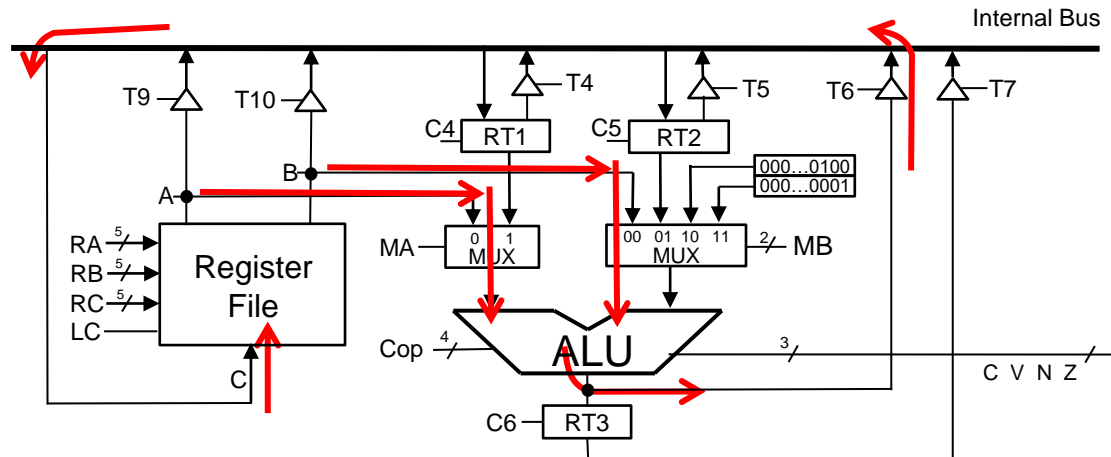
Elem. Op.	Signals
$RT1 \leftarrow R1$	RA=1, T9, C4
$R1 \leftarrow R2$	RA=2, T9, RC=1, LC
$R2 \leftarrow RT1$	T4, RC=2, LC

► 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

Example

elemental operations in ALU



► SWAP R1 R2

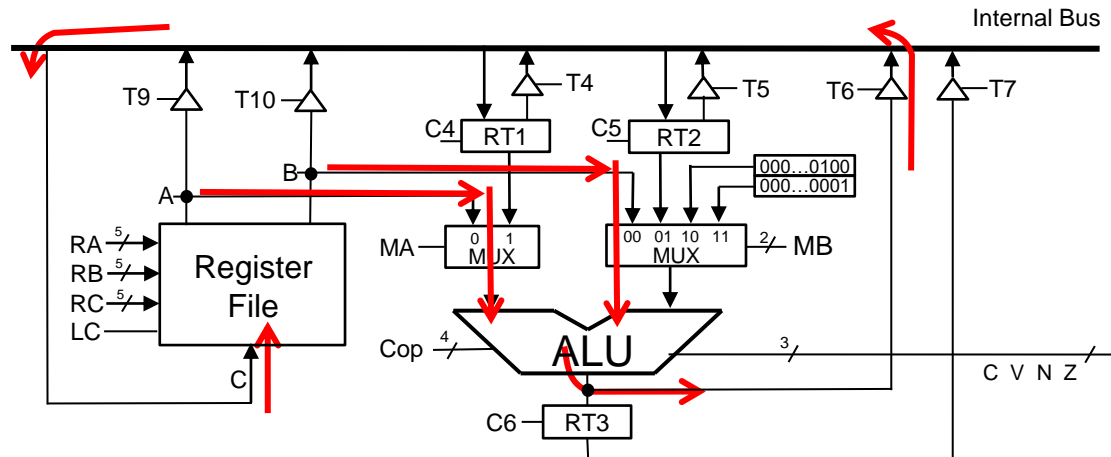
Elem. Op.	Signals
$RT1 \leftarrow R1$	RA=1, T9, C4
$R1 \leftarrow R2$	RA=2, T9, RC=1, LC
$R2 \leftarrow RT1$	T4, RC=2, LC

► 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

Example

elemental operations in ALU



► SWAP R1 R2

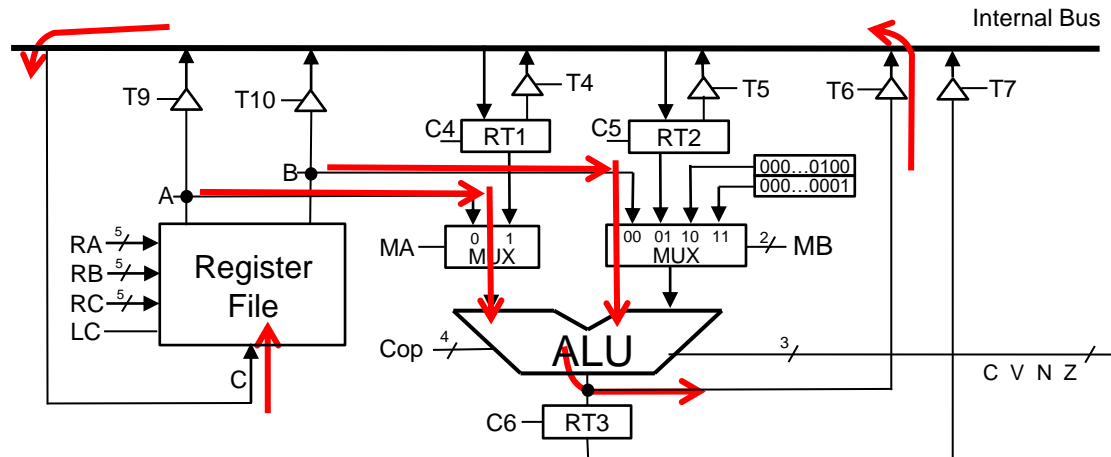
Elem. Op.	Signals
$RT1 \leftarrow R1$	RA=1, T9, C4
$R1 \leftarrow R2$	RA=2, T9, RC=1, LC
$R2 \leftarrow RT1$	T4, RC=2, LC

► 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



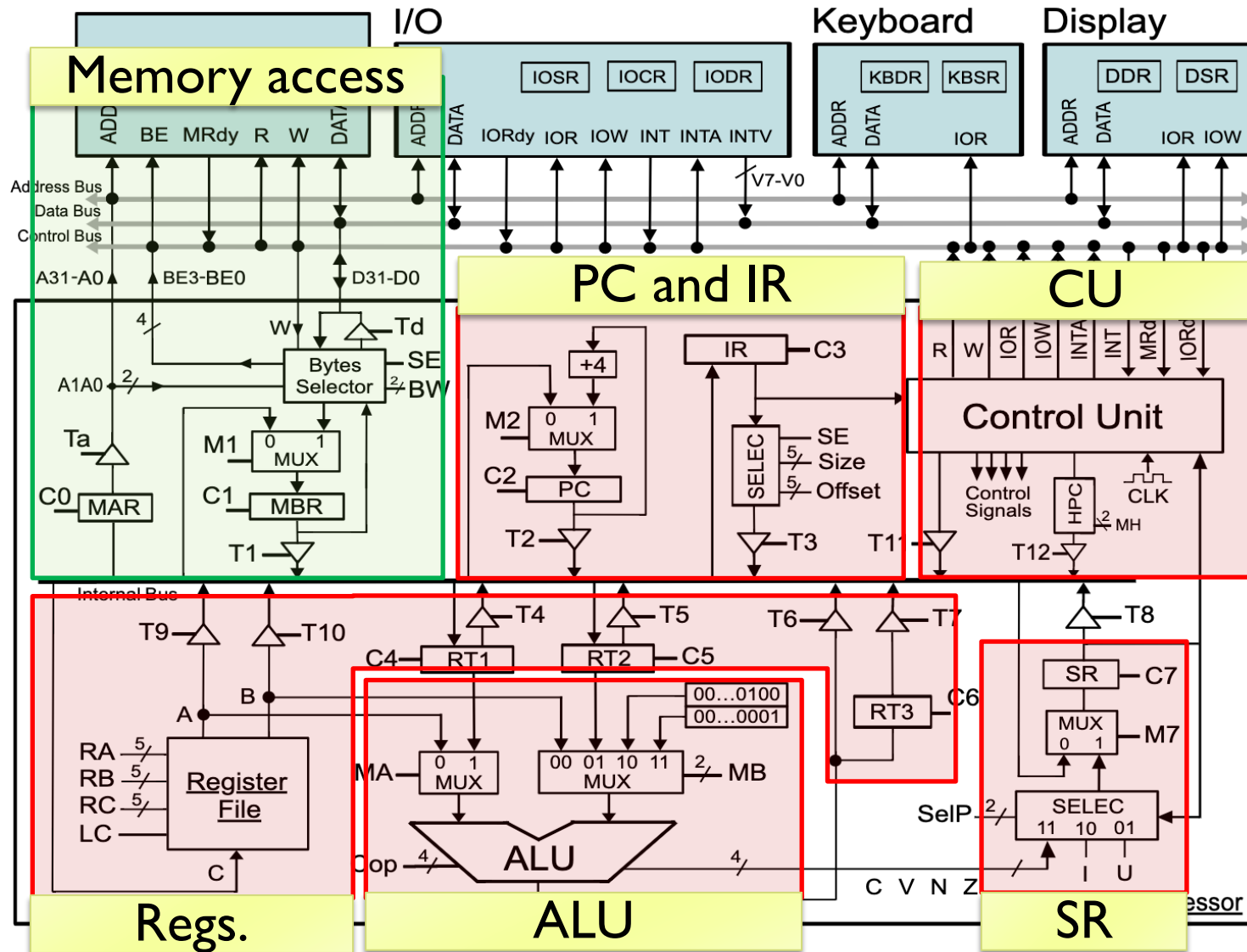
► SWAP R1 R2

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= \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

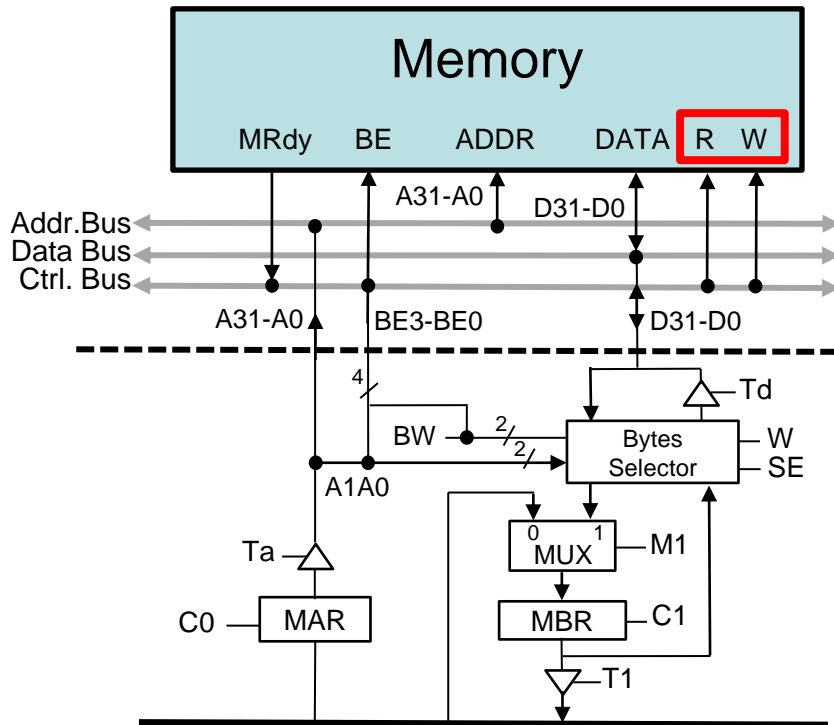
Elemental Processor: control signals



Control Signals

► Main Memory

- R – Read
- W – Write



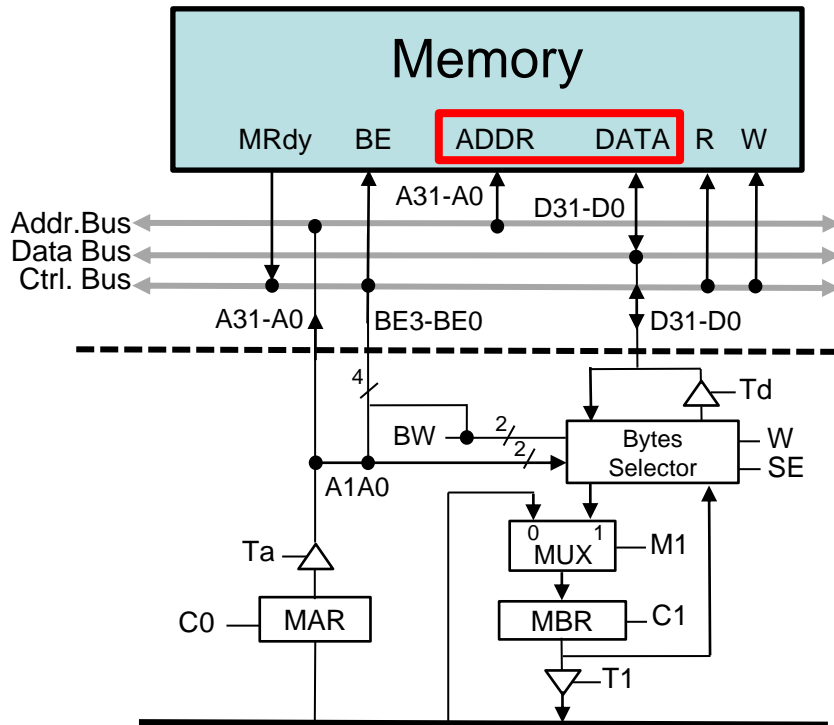
Nomenclature:

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

Control Signals

▶ Main Memory

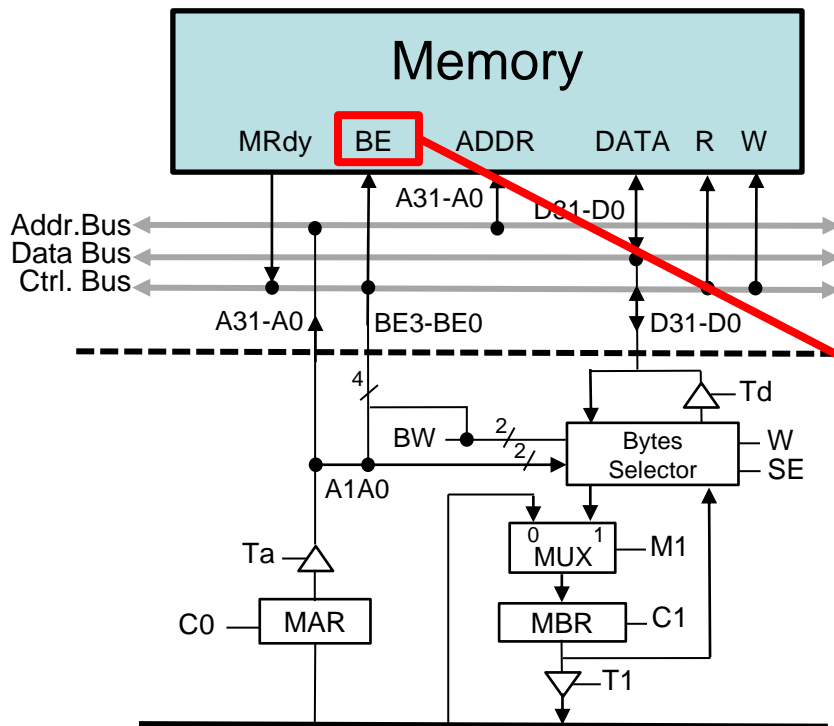
- ▶ R – Read
- ▶ W – Write
- ▶ DATA – data from/to memory
- ▶ ADDR – address



Nomenclature:

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

Control Signals



Nomenclature:

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

▶ Main Memory

- ▶ R – Read
- ▶ W – Write
- ▶ DATA – data from/to memory
- ▶ ADDR – address
- ▶ $BE3-BE0 = A1A0 + BW$
 - ▶ Access size (byte, word, half word)

▶ BW: byte selector

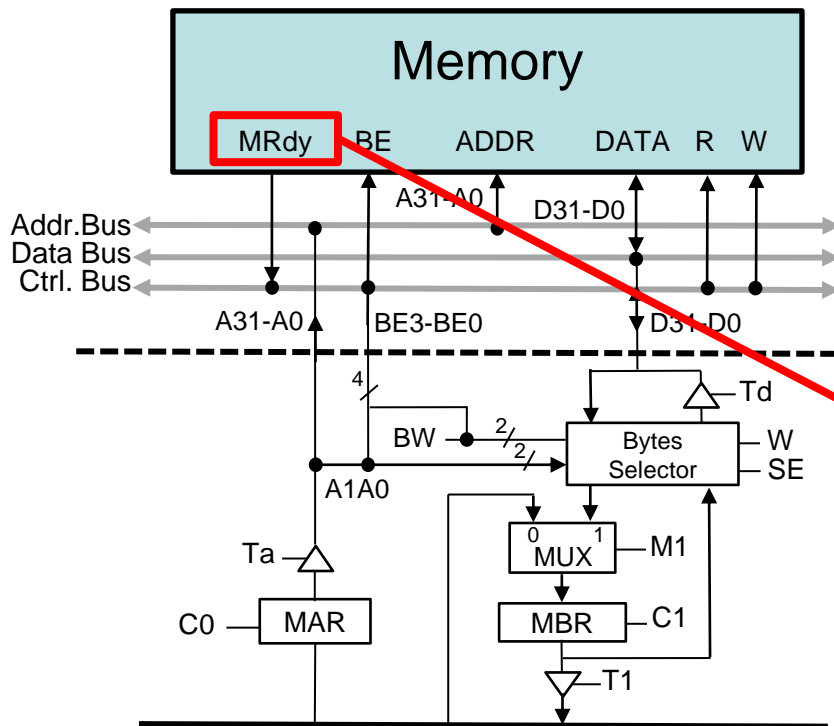
It selects which bytes are, stored in MBR while reading and copy to the bus on writes.

- ▶ **BW=0:** access to **byte**
- ▶ **BW=01:** access to **half 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

Control Signals



Nomenclature:

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

▶ Main Memory

- ▶ R – Read
- ▶ W – Write
- ▶ DATA – data from/to memory
- ▶ ADDR – address
- ▶ BE3-BE0 = A[3:0] + BW
 - ▶ Access size (byte, word, half word)
- ▶ MRdy – operation ended
[only in asynchronous]

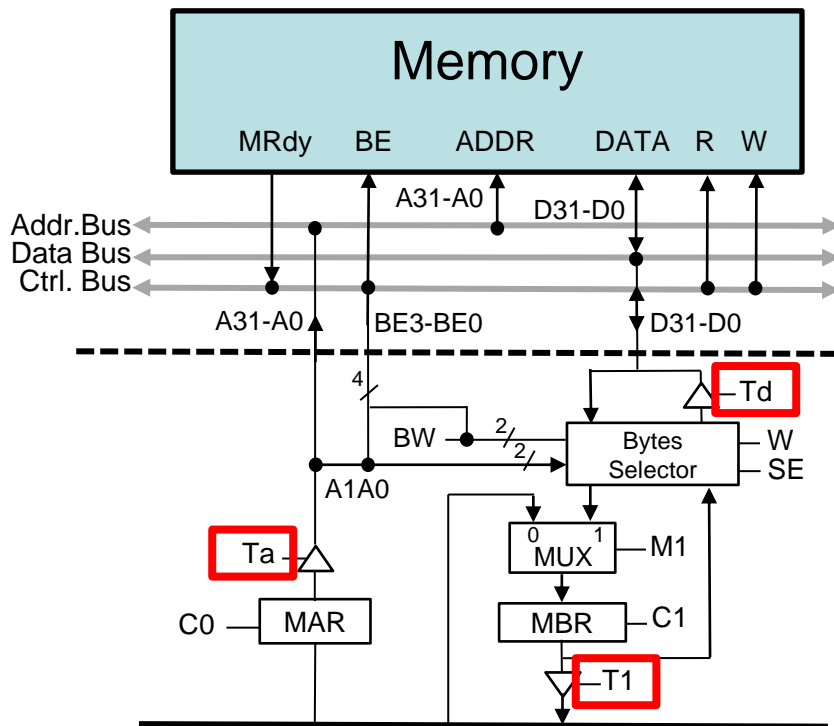
► Synchronous:

- ▶ 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

Control Signals



Nomenclature:

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

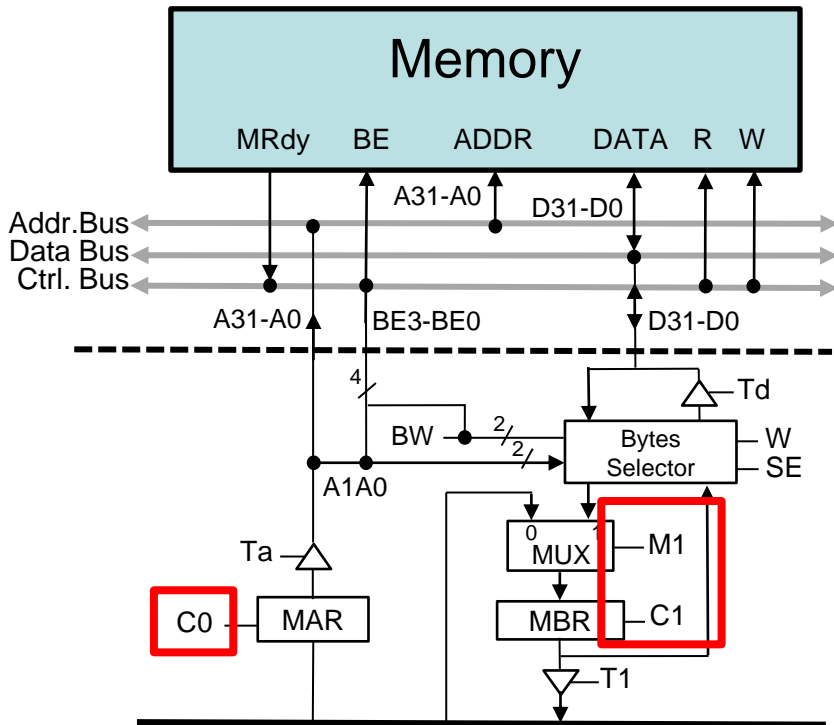
▶ Main Memory

- ▶ R – Read
- ▶ W – Write
- ▶ DATA – data from/to memory
- ▶ ADDR – address
- ▶ $BE3-BE0 = A1A0 + BW$
 - ▶ Access size (byte, word, half word)
- ▶ MRdy – operation ended
[only in asynchronous]

▶ MAR & MBR

- ▶ Ta – output of MAR to the address bus
- ▶ Td – MBR output to data bus
- ▶ T1 – MBR output to internal bus

Control Signals



Nomenclature:

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

▶ Main Memory

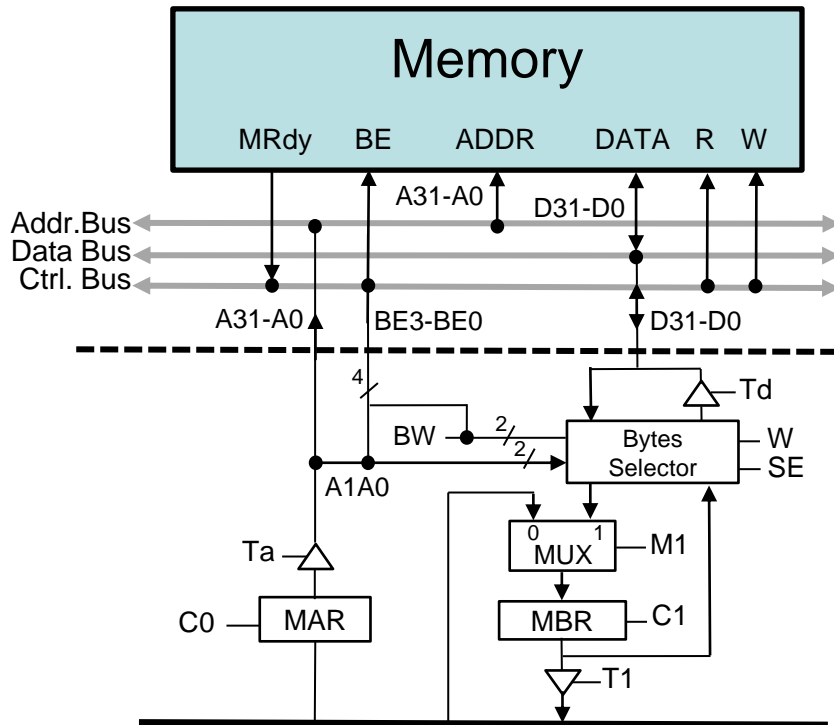
- ▶ R – Read
- ▶ W – Write
- ▶ DATA – data from/to memory
- ▶ ADDR – address
- ▶ $BE3-BE0 = A[3:A0] + BW$
 - ▶ Access size (byte, word, half word)
- ▶ MRdy – operation ended
[only in asynchronous]

▶ MAR & MBR

- ▶ Ta – output of MAR to the address bus
- ▶ Td – MBR output to data bus
- ▶ TI – MBR output to internal bus
- ▶ MI – selection for MBR: memory or internal bus
- ▶ CI – from data bus to MBR
- ▶ C0 – from internal bus to MAR

Control Signals

summary



Nomenclature:

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

▶ Main Memory

- ▶ R – Read
- ▶ W – Write
- ▶ DATA – data from/to memory
- ▶ ADDR – address
- ▶ $BE3-BE0 = A1A0 + BW$
 - ▶ Access size (byte, word, half word)
- ▶ MRdy – operation ended
[only in asynchronous]

▶ MAR & 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
- ▶ C1 – from data bus to MBR
- ▶ C0 – from internal bus to MAR

BE (Byte-Enable) signals

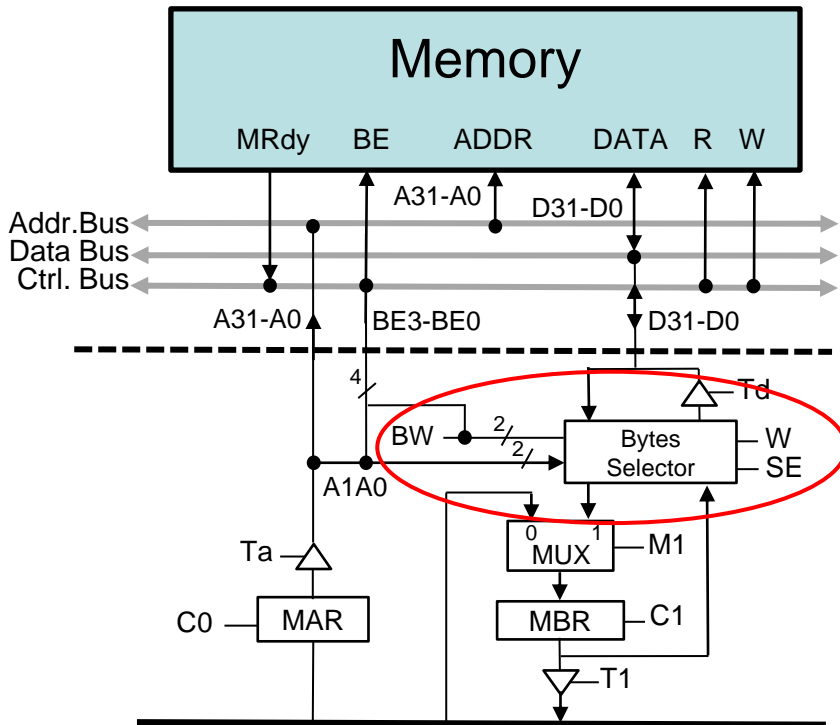
► For reading:

Bytes in memory				Byte-Enable				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

► For writing:

Data in bus				Byte-Enable				Bytes written in memory			
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:

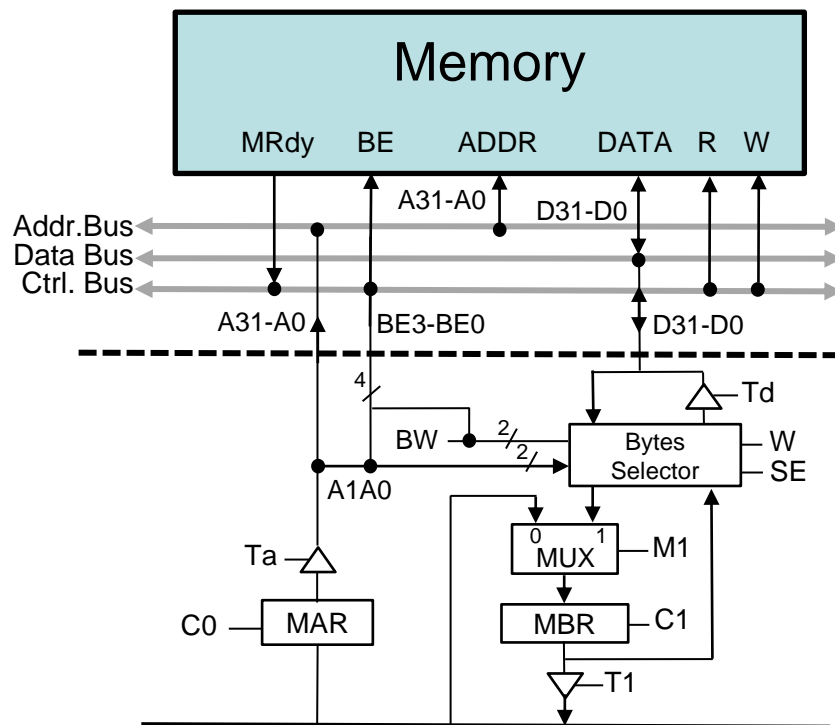
- MAR -> Address register
- MBR -> Data register

- ▶ Byte Selector: selects which bytes are stored in MBR while reading and copy to the bus on writes.
 - ▶ **BW=0**: access to **byte**
 - ▶ **BW=01**: access to **half 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

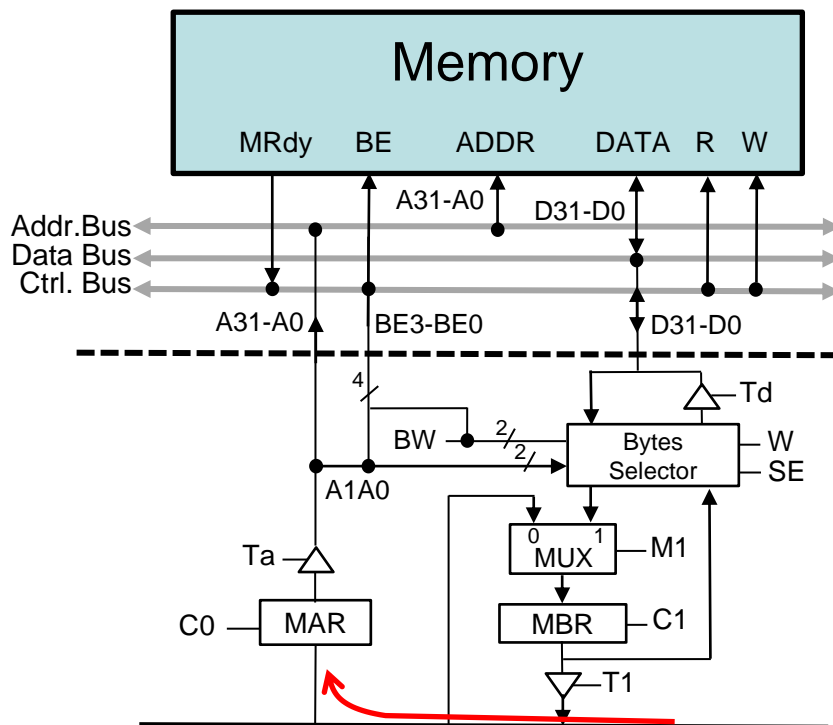
► Reading a word



Example

access to 1 cycle synchronous main memory

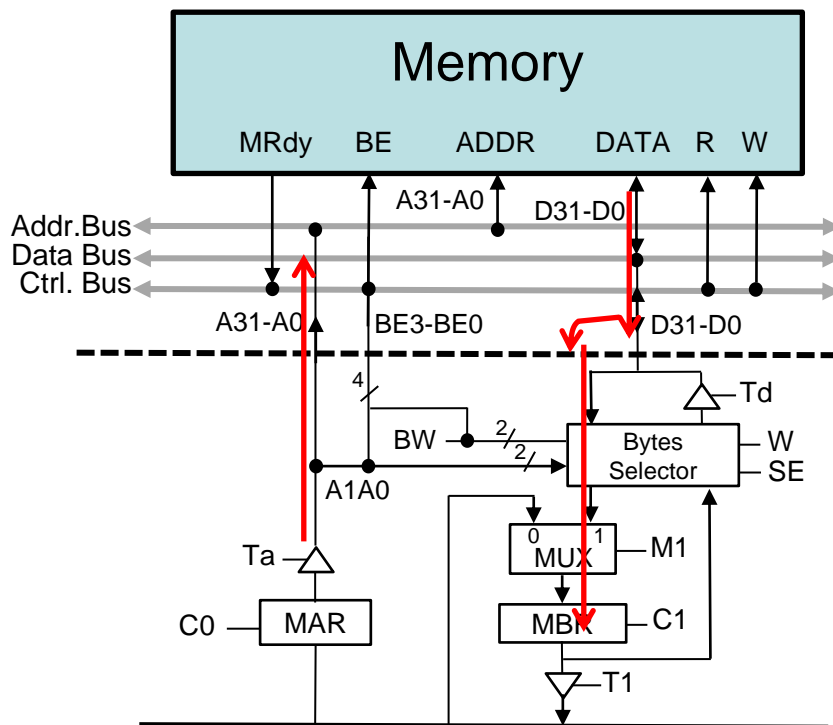
► Read



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

access to 1 cycle synchronous main memory

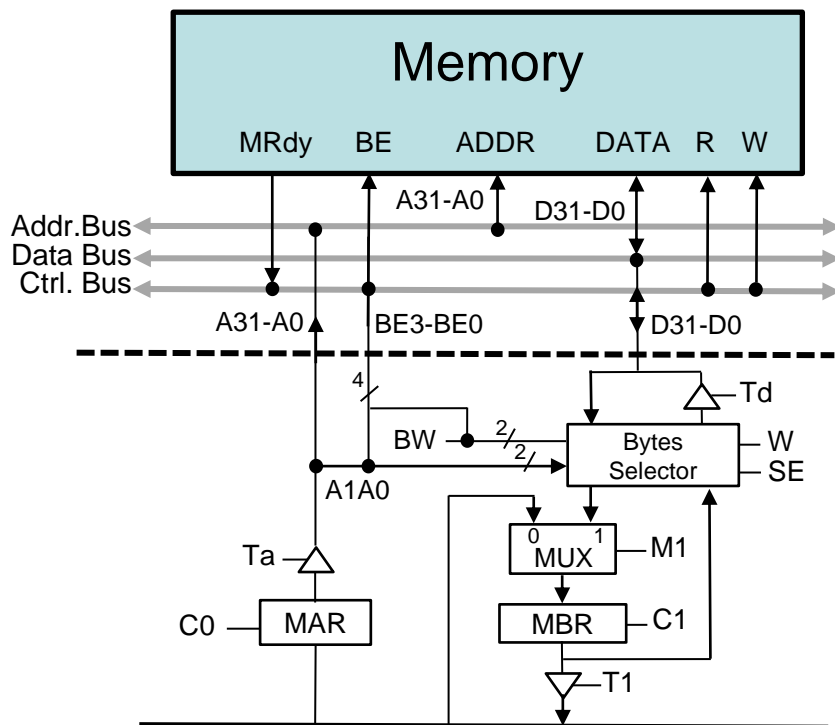
► **Read**



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



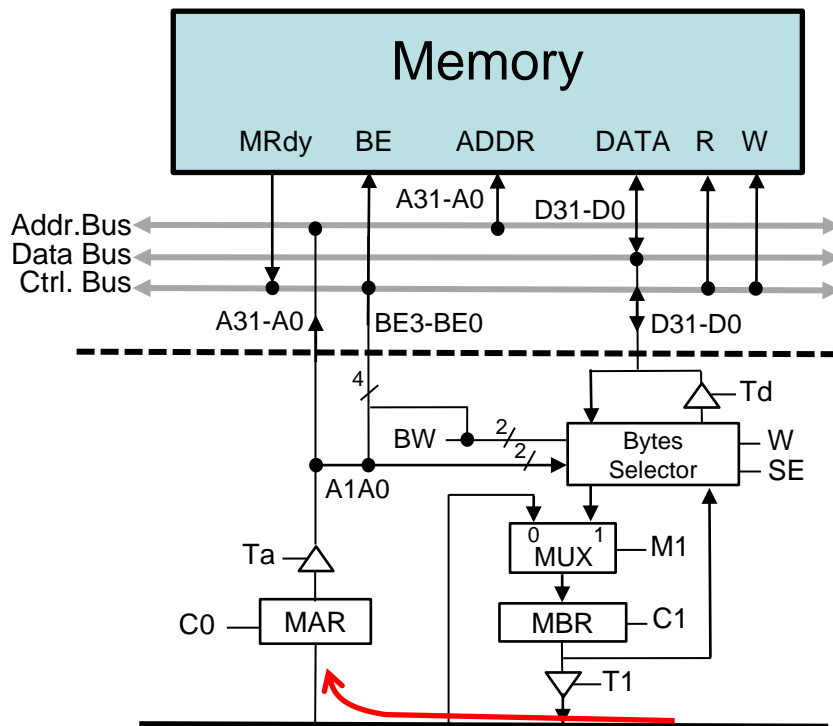
► Read

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



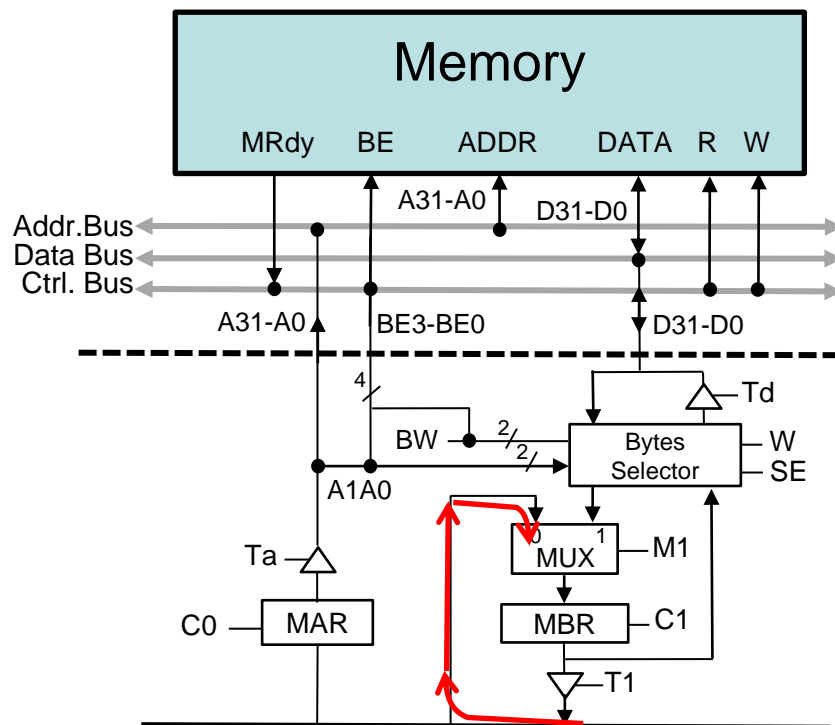
Read

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

Write

Elem. Op.	Signals
$MAR \leftarrow \langle \text{address} \rangle$..., C0

access to 1 cycle synchronous main memory



► **Read**

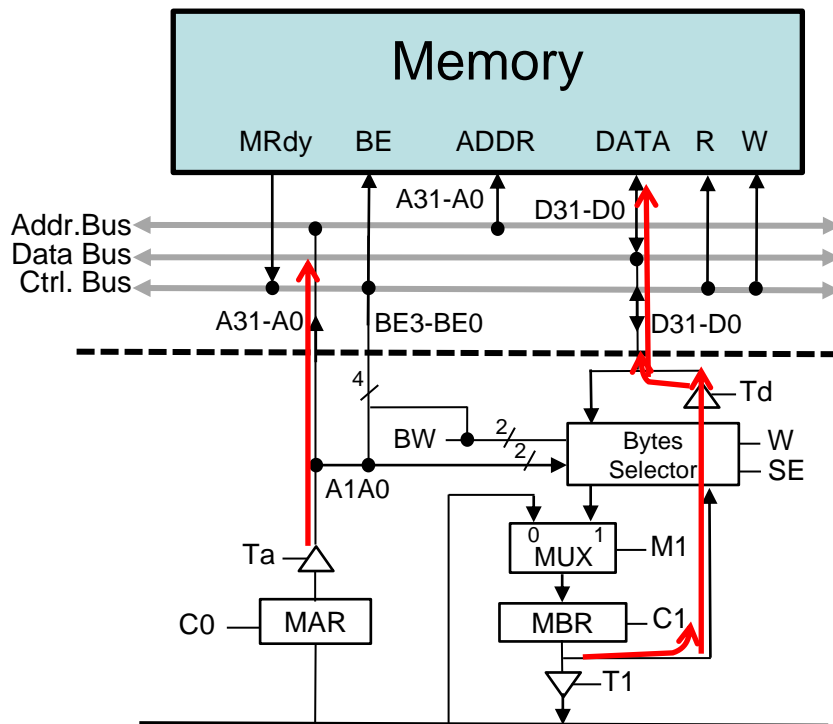
Elem. Op.	Signals
MAR \leftarrow <address>	..., C0
MBR \leftarrow MP[MAR]	Ta, R, M1, C1, BW=11

► Write

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

Example

access to 1 cycle synchronous main memory



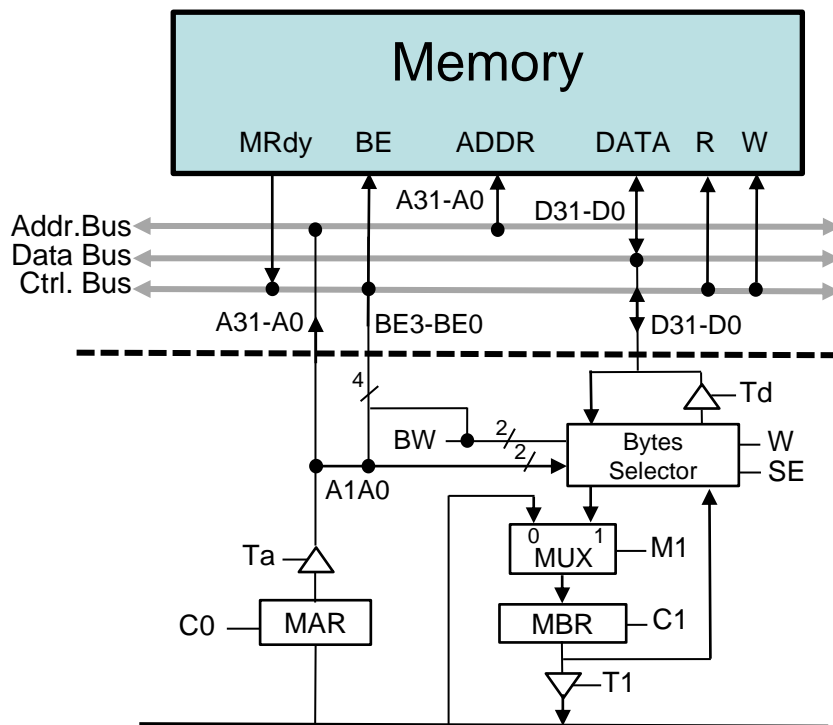
Read

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

Write

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

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

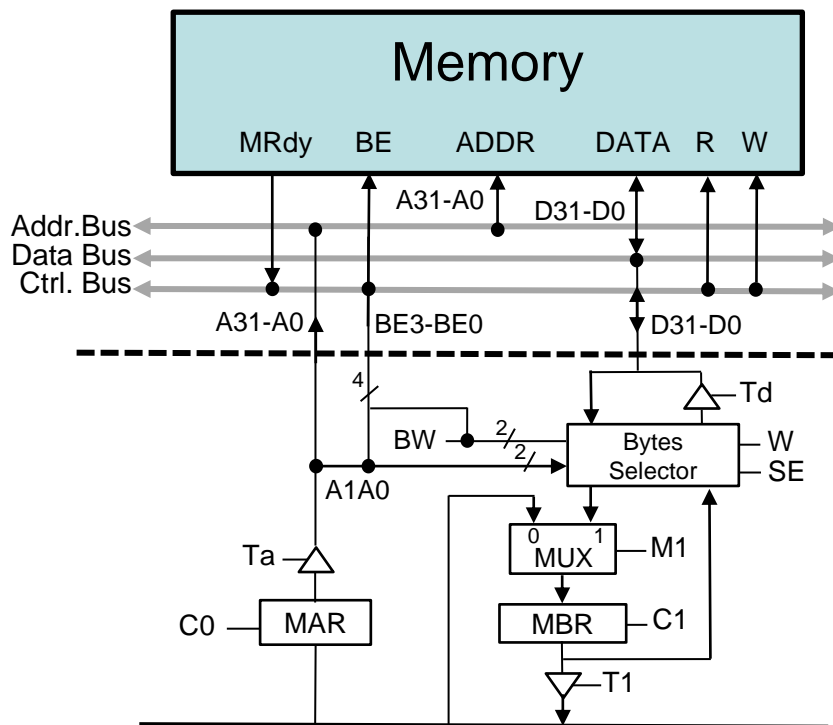
► Write

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

Example

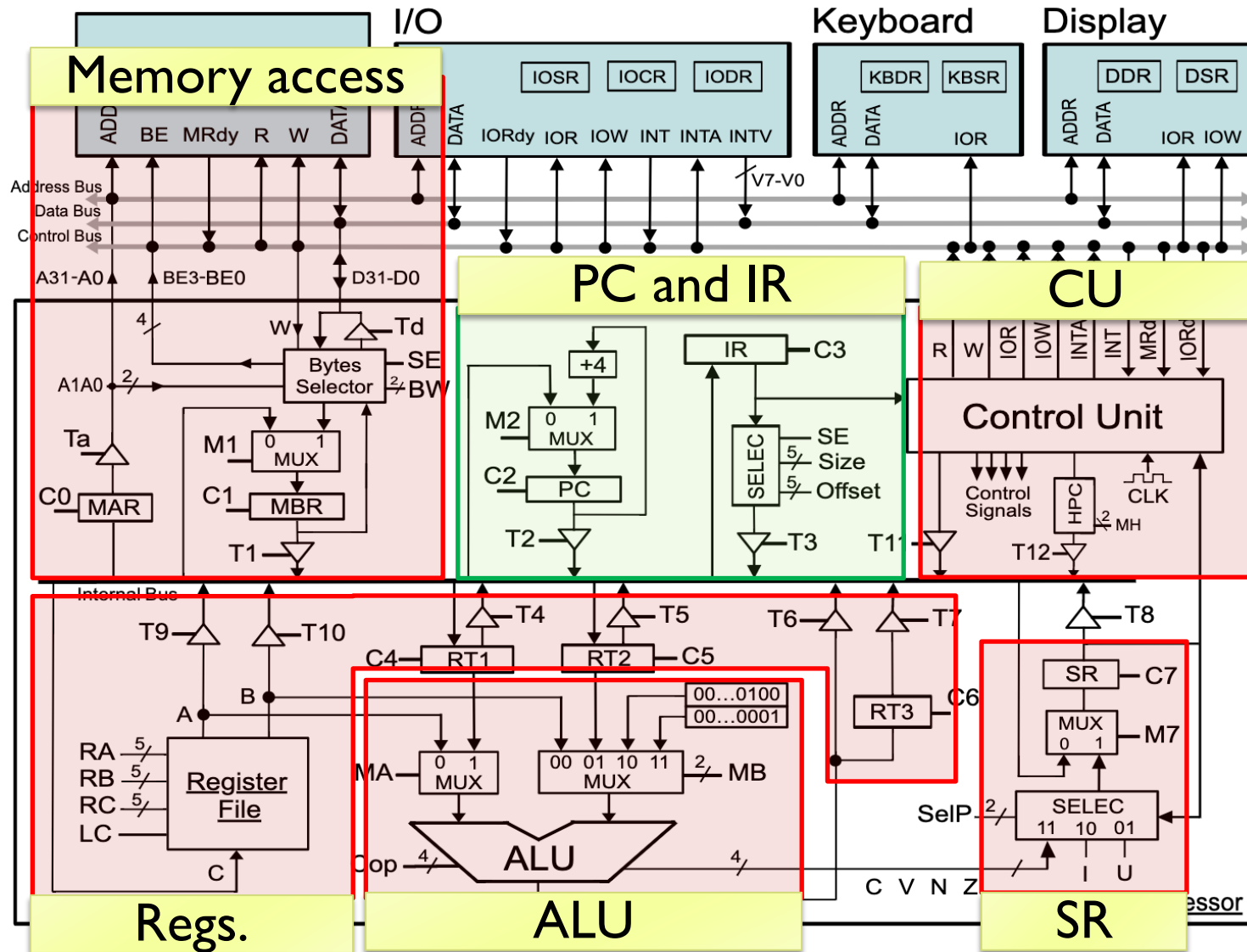
access to **2** cycle synchronous main memory

► 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

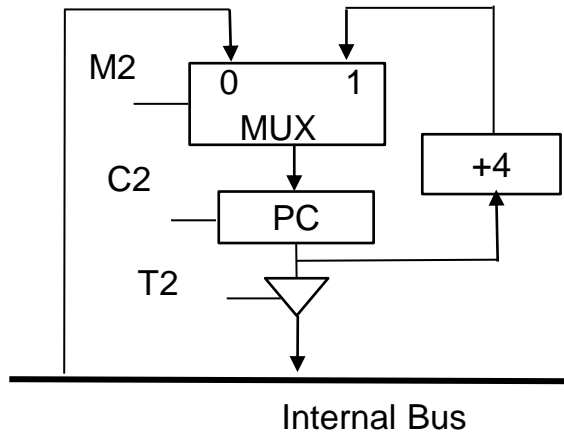
Elemental Processor: control signals



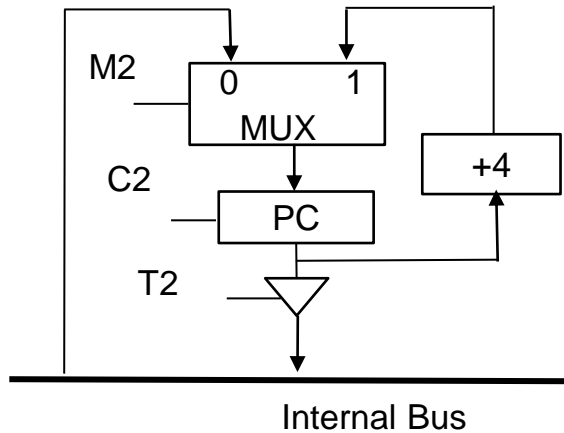
PC: Program Counter

► PC

- C2 – load value into PC
- T2 – from PC to internal bus
- M2 – internal bus or PC+ 4



PC Mux: IB/PC+4

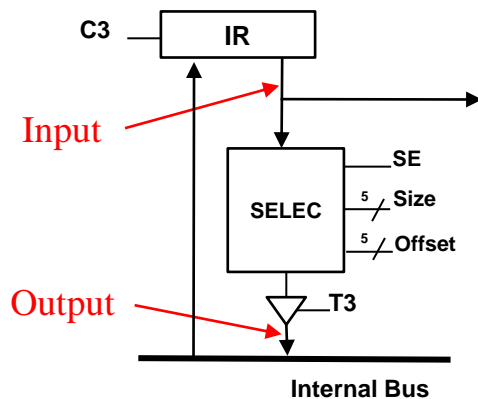


► PC

- C2 – load value into PC
- T2 – from PC to internal bus
- M2 – internal bus or PC+ 4

- C2, M2
 - $PC \leftarrow PC + 4$
- C2, M2=0
 - $PC \leftarrow \text{<internal bus>}$

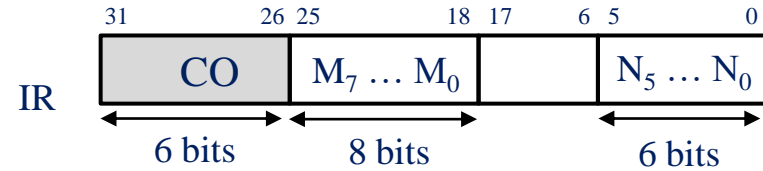
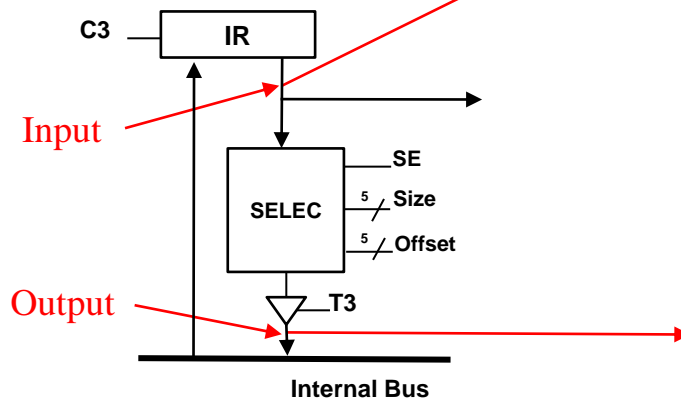
IR: Instruction register



► IR:

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

SELEC: selector circuit



(SE = 0) Selection without sign extension

Size	Offset	Output
01000	10010	
00110	00000	

(SE = 1) Selection with sign extension

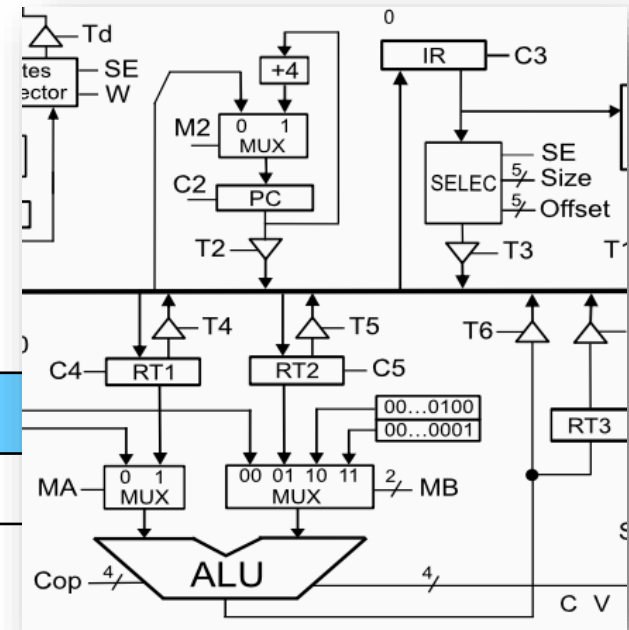
Size	Offset	Output
01000	10010	
00110	00000	

Size in bits

Start bit (less significant)

Execution of j addr

op.	address
6 bits	26 bits



Cycle	Elem. Op.	Control Signals

Execution of j addr

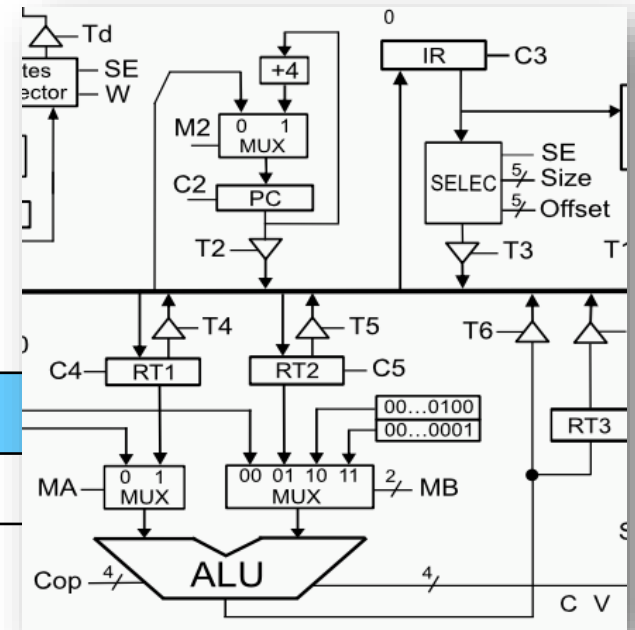
TIP

General phases:

- Fetch + Decode
- Fetch operands
- Execution
- Store results

Cycle	Elem. Op.	Control Signals

op.	address
6 bits	26 bits



Execution of j addr

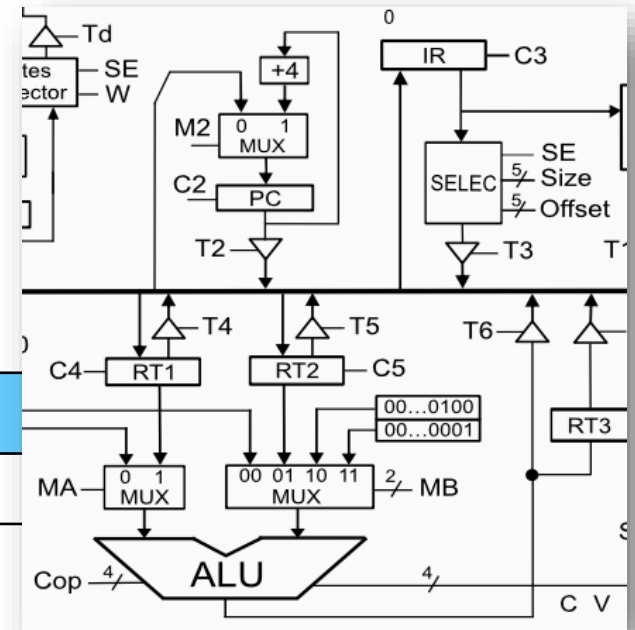
TIP

Not possible in the same clock cycle:

1. To passthrough a register (~~C4, MA=1~~)
2. To send several values to a bus (~~T4, T5~~)
3. To set a datapath if the circuitry does not enable it (~~IDB → RT3~~)

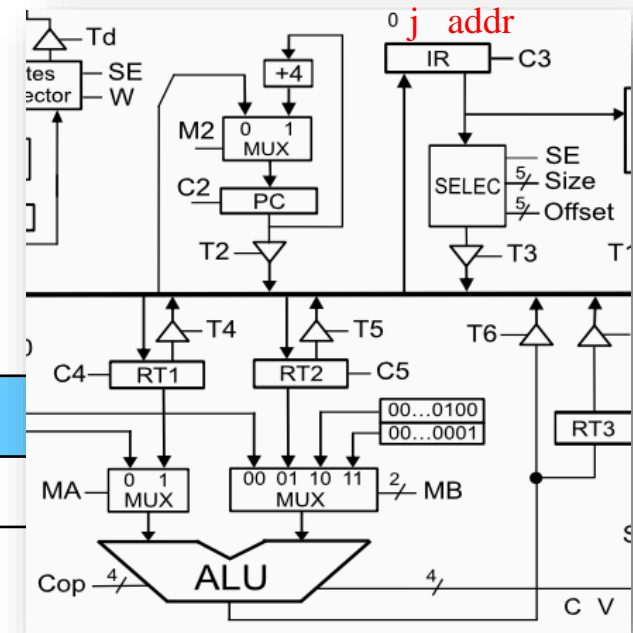
Cycle	Elem. Op.	Control Signals

op.	address
6 bits	26 bits



Execution of j addr

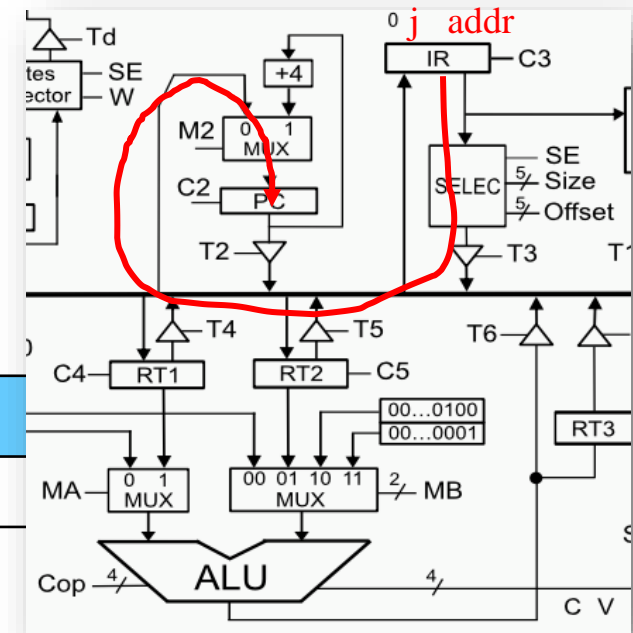
op.	address
6 bits	26 bits



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	A0, B=0, C=0

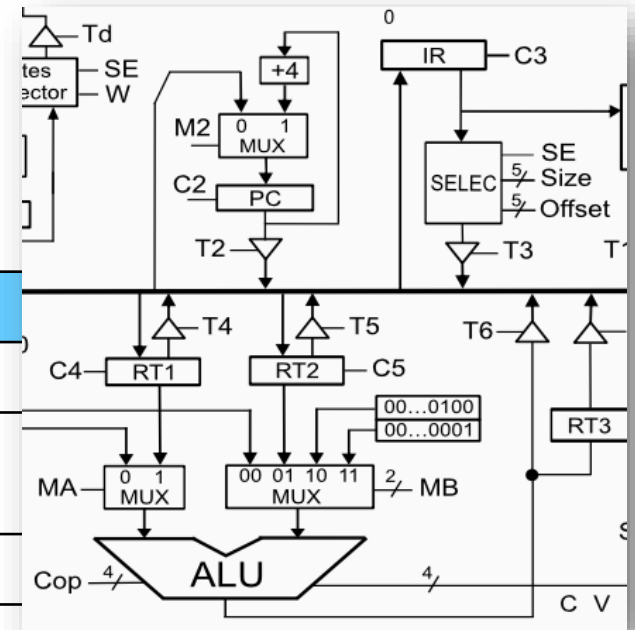
Execution of j addr

op.	address
6 bits	26 bits



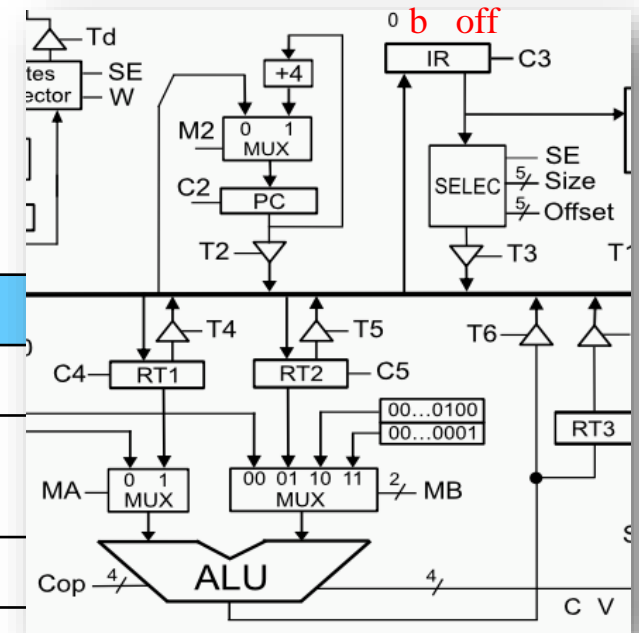
Cycle	Elem. Op.	Control Signals
C1	$MAR \leftarrow PC$	T2, C0
C2	$PC \leftarrow PC + 4,$ $MBR \leftarrow MP$	C2, M1 Ta, R, C1, M1, BW=11
C3	$IR \leftarrow MBR$	T1, C3
C4	Decoding	A0, B=0, C=0
C5	$PC \leftarrow RI(dir)$	Size = 11010 (26), Offset = 00000, SE=0, C2, T3
C6	Salto a fetch	A0, B=1, C=0

Execution of b offset



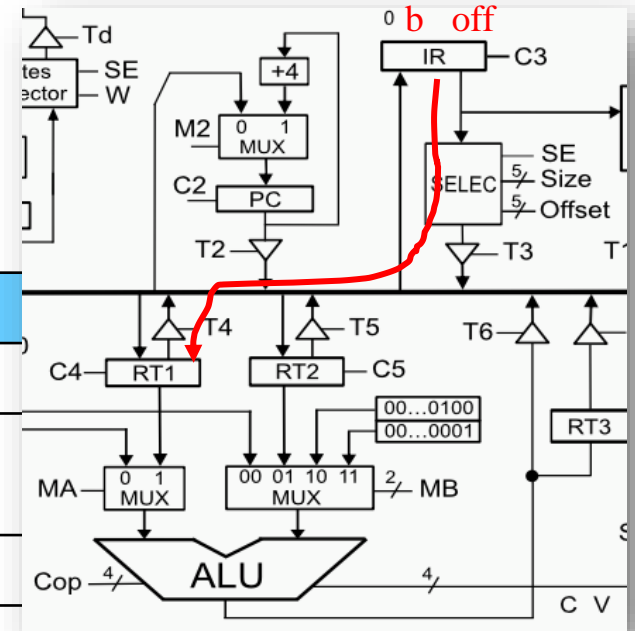
Cycle	Elem. Op.	Control Signals

Execution of b offset



Cycle	Elem. Op.	Control Signals
C1	$MAR \leftarrow PC$	T2, C0
C2	$PC \leftarrow PC + 4,$ $MBR \leftarrow MP$	C2, MI Ta, R, CI, MI, BW=11
C3	$IR \leftarrow MBR$	T1, C3
C4	Decoding	A0, B=0, C=0

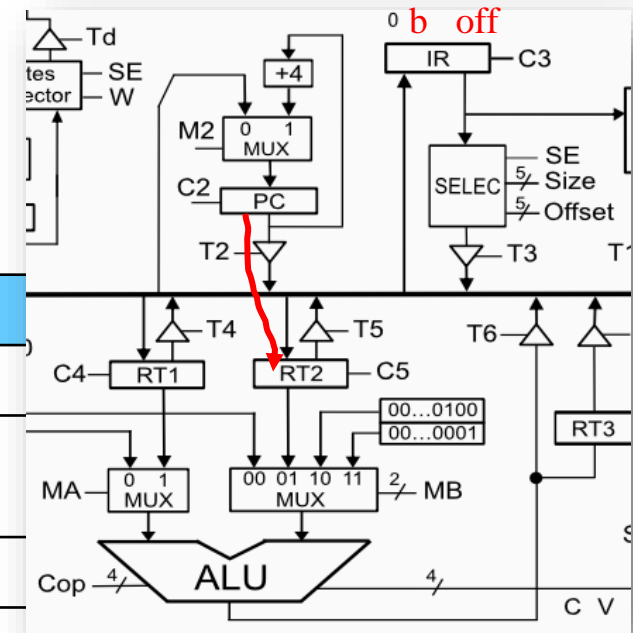
Execution of b offset



Cycle	Elem. Op.	Control Signals
C1	$MAR \leftarrow PC$	T2, C0
C2	$PC \leftarrow PC + 4,$ $MBR \leftarrow MP$	C2, MI Ta, R, CI, MI, BW=11
C3	$IR \leftarrow MBR$	T1, C3
C4	Decoding	A0, B=0, C=0
C5	$RTI \leftarrow RI(off)$	Size=10000 (16), Offset=0, SE=1, T3, C4

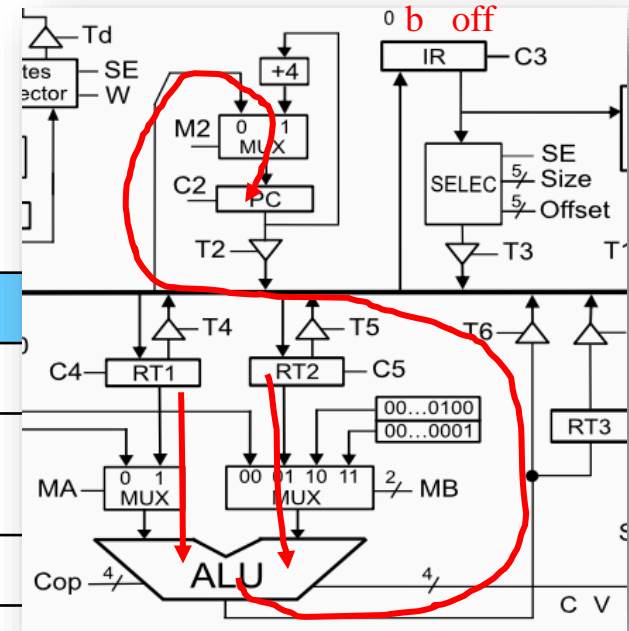
The diagram illustrates the ALU and register file for Cycle 5. The ALU has two 4-bit inputs from multiplexers (MUX) and a 4-bit output. The MUXes are controlled by signals T4, T5, and T6. The ALU output is connected to the RTI register. The register file contains registers RT1, RT2, and RT3. The ALU output is also connected to the RTI register. The ALU output is connected to the RTI register.

Execution of b offset



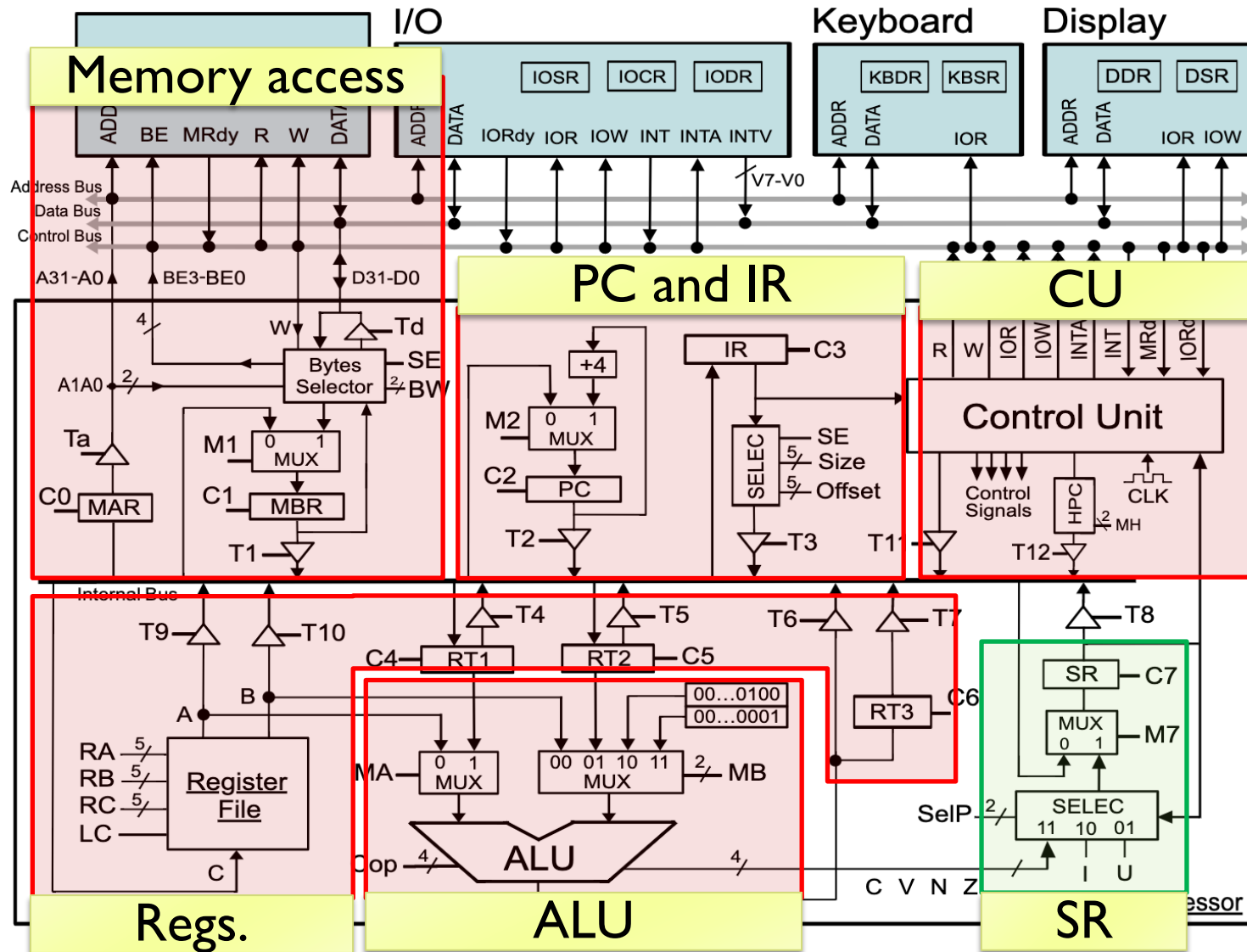
Cycle	Elem. Op.	Control Signals
C1	$MAR \leftarrow PC$	T2, C0
C2	$PC \leftarrow PC + 4,$ $MBR \leftarrow MP$	C2, MI Ta, R, CI, MI, BW=11
C3	$IR \leftarrow MBR$	T1, C3
C4	Decoding	A0, B=0, C=0
C5	$RT1 \leftarrow RI(off)$	Size=10000 (16), Offset=0, SE=1, T3, C4
C6	$RT2 \leftarrow PC$	T2, C5

Execution of b offset

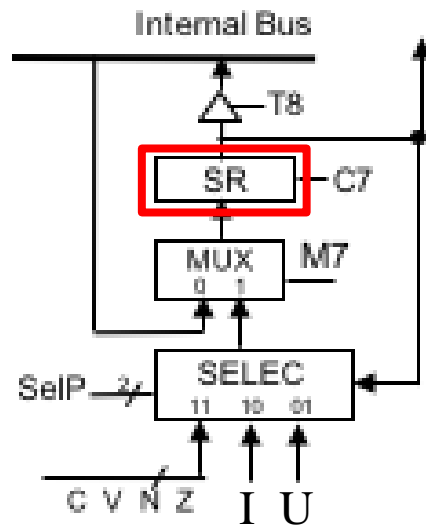


Cycle	Elem. Op.	Control Signals
C1	$MAR \leftarrow PC$	T2, C0
C2	$PC \leftarrow PC + 4,$ $MBR \leftarrow MP$	C2, M1 Ta, R, C1, M1, BW=11
C3	$IR \leftarrow MBR$	T1, C3
C4	Decoding	A0, B=0, C=0
C5	$RT1 \leftarrow RI(off)$	Size=10000 (16), Offset=0, SE=1, T3, C4
C6	$RT2 \leftarrow PC$	T2, C5
C7	$PC \leftarrow RT1 + RT2$	MA=1, MB=1, MC=1, SELCOP=+, T6, M2=0, C2
C6	Salto a fetch	A0, B=1, C=0

Elemental Processor: control signals

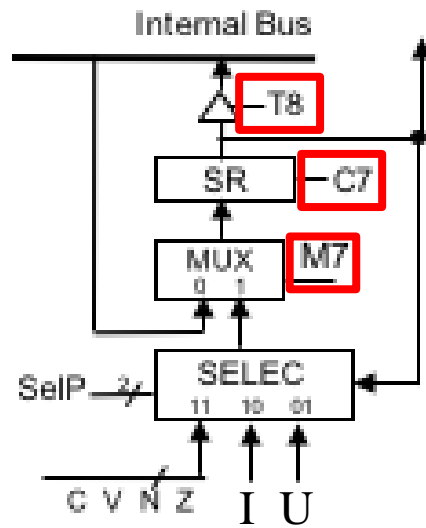


SR: Status Register



- ▶ Stores information (status bits) about the **status of the program being executed on the processor.**
- ▶ Typical status bits:
 - ▶ C, V, N, Z:
Result from **last operation in ALU**
 - ▶ U:
CPU running in **kernel or user mode**
 - ▶ I:
Interruptions are **enabled or not**

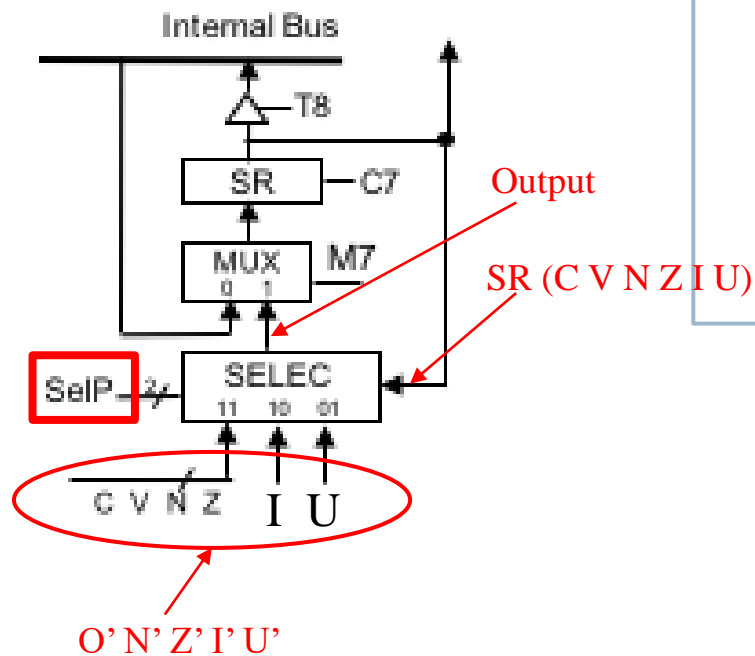
SR: Status Register



► SR

- T8 – from SR to internal data bus
- C7 – from internal data bus to SR
- M7 – flags from IDB or SELEC

SR: Status Register



► SR

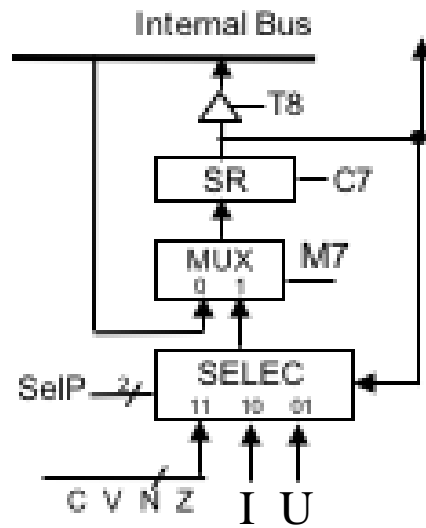
- T8 – from SR to internal data bus
- C7 – from internal data bus to SR
- M7 – flags from IDB or SELEC
- SelP – update flags: ALU / I / U

if (SelP == 11)
 Output = C' V' N' Z' I U

if (SelP == 10)
 Output = C V N Z I' U

if (SelP == 01)
 Output = C V N Z I U'

SR: Save/Restore + Update from ALU

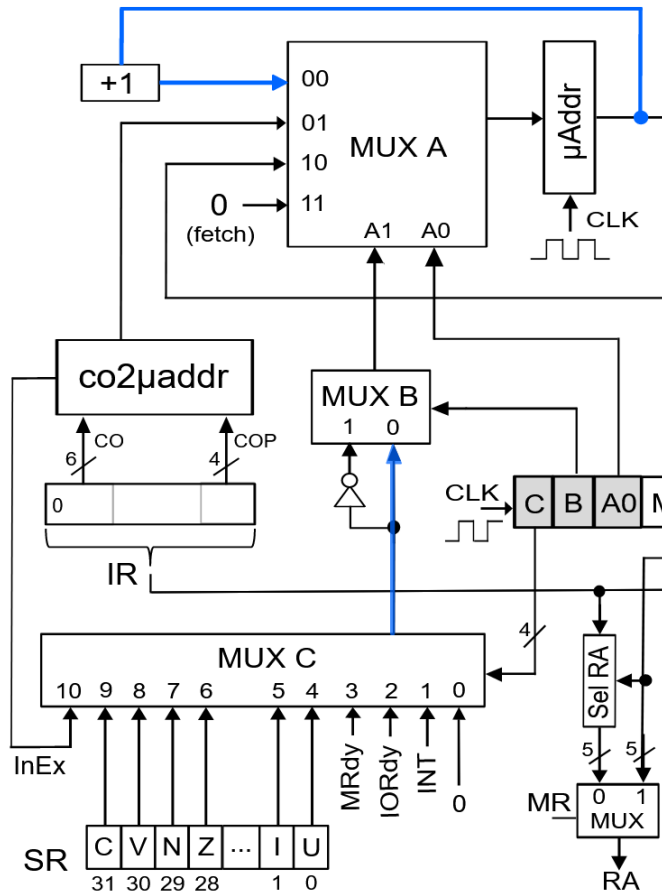


► SR

- T8 – from SR to internal data bus
- C7 – from internal data bus to SR
- M7 – flags from IDB or SELEC
- SelP – update flags: ALU / I / U

- T8, C4
 - $RTI \leftarrow SR$
- T4, M7=0, C7
 - $SR \leftarrow RTI$
- SelP=11, M7, C7
 - $SR \leftarrow \langle \text{ALU flags} \rangle$

SR: check some flag...



SR

- C – condition to select
- B – condition or not condition
- A0 – 1: fetch/co2maddr
0: from maddr/maddr+1

- $C=[4...9], B=0, A0=0$
 - If $C \text{ maddr} \leftarrow MADDR$
else $\text{maddr} \leftarrow \text{maddr}+1$
- $C=0, B=1, A0=0$
 - $\text{maddr} \leftarrow MADDR$

beq r1, r2, offset

Cycle	Elem. Op.
C1	$MAR \leftarrow PC$
C2	$PC \leftarrow PC + 4,$ $MBR \leftarrow MP$
C3	$IR \leftarrow MBR$
C4	Decode
C12	Jump to fetch

Si $\$r1 == \$r2$
 $PC \leftarrow PC + \text{offset}$

beq r1, r2, offset

Cycle	Elem. Op.
C1	$MAR \leftarrow PC$
C2	$PC \leftarrow PC + 4,$ $MBR \leftarrow MP$
C3	$IR \leftarrow MBR$
C4	Decode
C5	$MBR \leftarrow SR$
C6	$\$r1 - \$r2$
C7	Si $SR.Z \neq 0$ jump to C11
C11	$SR \leftarrow MBR$
C12	Jump to fetch

Si $\$r1 == \$r2$
 $PC \leftarrow PC + \text{offset}$

beq r1, r2, offset

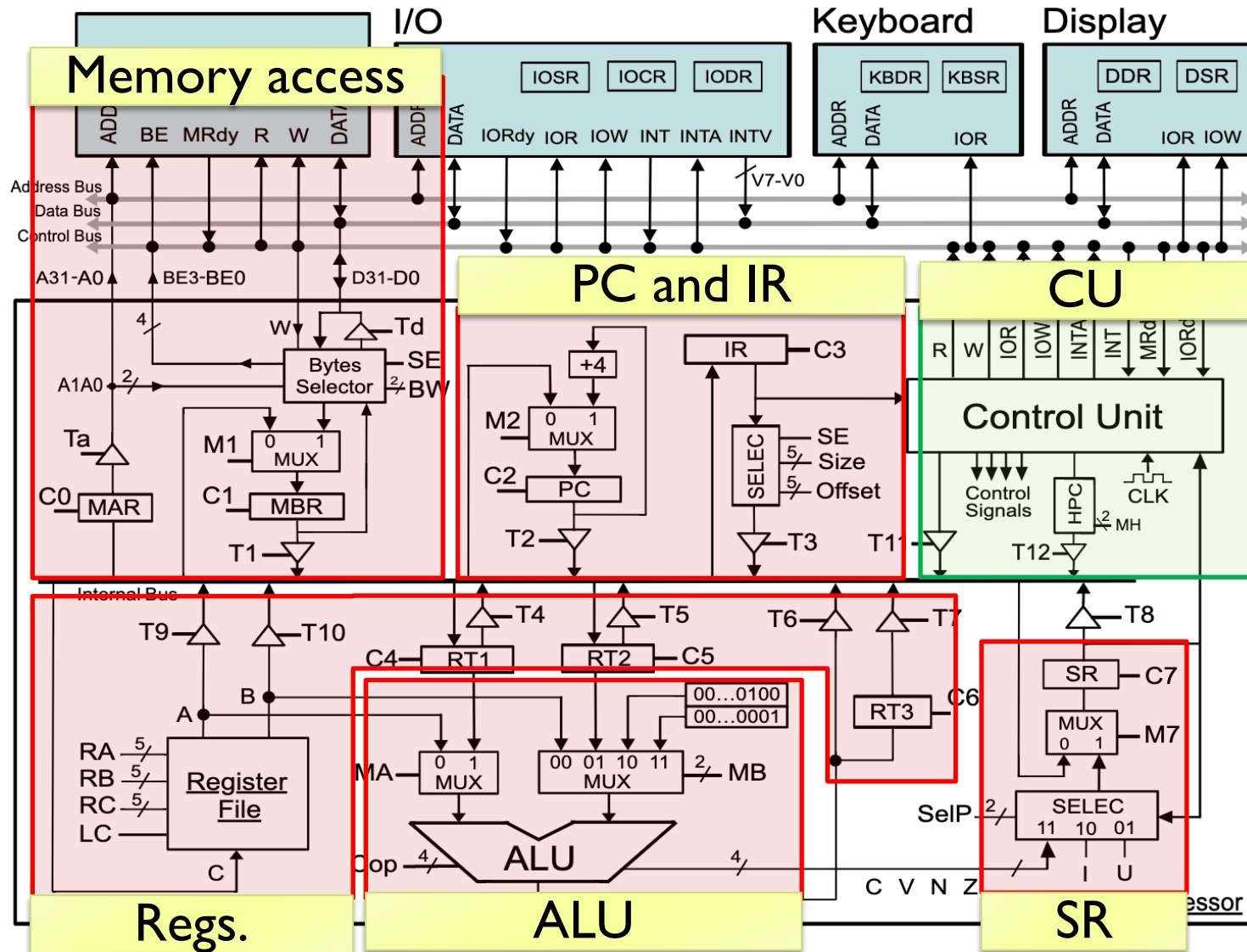
Cycle	Elem. Op.
C1	$MAR \leftarrow PC$
C2	$PC \leftarrow PC + 4,$ $MBR \leftarrow MP$
C3	$IR \leftarrow MBR$
C4	Decode
C5	$MBR \leftarrow SR$
C6	$\$r1 - \$r2$
C7	Si $SR.Z \neq 0$ jump to C11
C8	$RT1 \leftarrow PC$
C9	$RT2 \leftarrow IR(\text{offset})$
C10	$PC \leftarrow RT1 + RT2$
C11	$SR \leftarrow MBR$
C12	Jump to fetch

Si $\$r1 == \$r2$
 $PC \leftarrow PC + \text{offset}$

beq r1, r2, offset

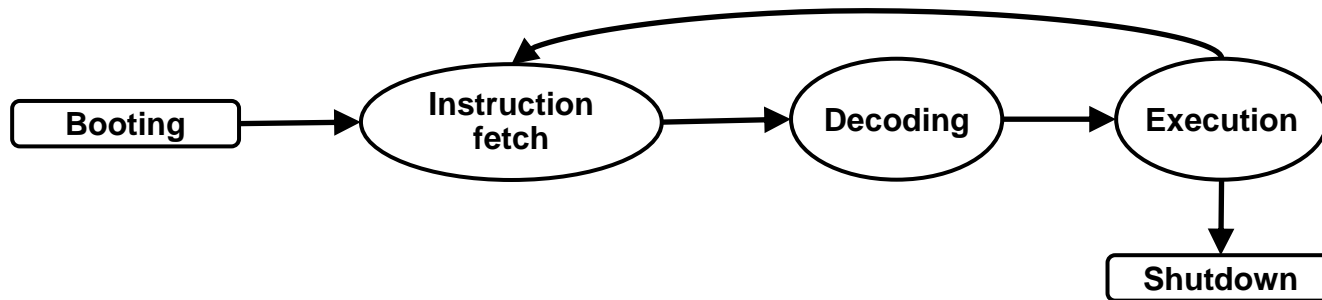
Cycle	Elem. Op.	Control Signals
C1	$MAR \leftarrow PC$	T2, C0
C2	$PC \leftarrow PC + 4,$ $MBR \leftarrow MP$	C2, M1, Ta, R, C1, M1, BW=11
C3	$IR \leftarrow MBR$	T1, C3
C4	Decode	A0, B=0, C=0
C5	$MBR \leftarrow SR$	T8, C1
C6	$\$r1 - \$r2$	SELA=10101, SELB=10000, MC=1, SELCOP=1011, SELP=11, M7, C7
C7	Si SR.Z != 0 jump to C11	A0=0, B=1, C=110, MADDR=beq11
C8	$RT1 \leftarrow PC$	T2, C4
C9	$RT2 \leftarrow IR(\text{offset})$	Size=10000, Offset=0, SE=1, T3, C5
C10	$PC \leftarrow RT1 + RT2$	MA=1, MB=1, MC=1, SELCOP=+, T6, C2
C11	$SR \leftarrow MBR$	T1, M7=0, C7
C12	Jump to fetch	A0, B=1, C=0

Elemental Processor: control signals



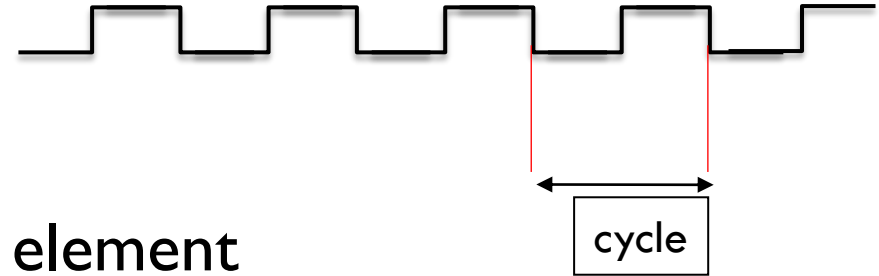
Control unit:

Phases of execution of an instruction



- ▶ **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 IR to determine:
 - ▶ The operation to be performed.
 - ▶ Control signals to be activated
- ▶ **Execution**
 - ▶ Generation of the control signals in each clock cycle.

Clock

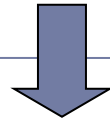


- ▶ A computer is a synchronous element
 - ▶ Clock controls the operation
- ▶ The clock regulates the operations in a given time:
 - ▶ In a clock cycle one or more elementary operations are executed as long as there is no conflict
 - ▶ In the same cycle you can perform
 - $MAR \leftarrow PC$ and $RT3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

Description of the Control Unit activity

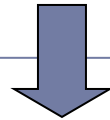
Instruction

mv R0 R1

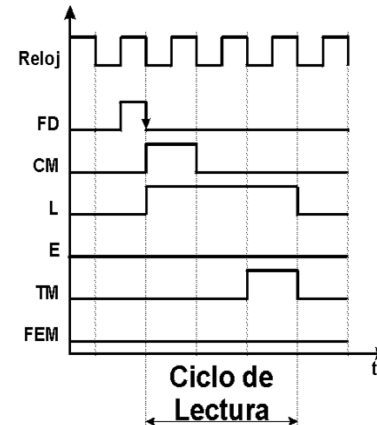


Sequence of **elementary operations**

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

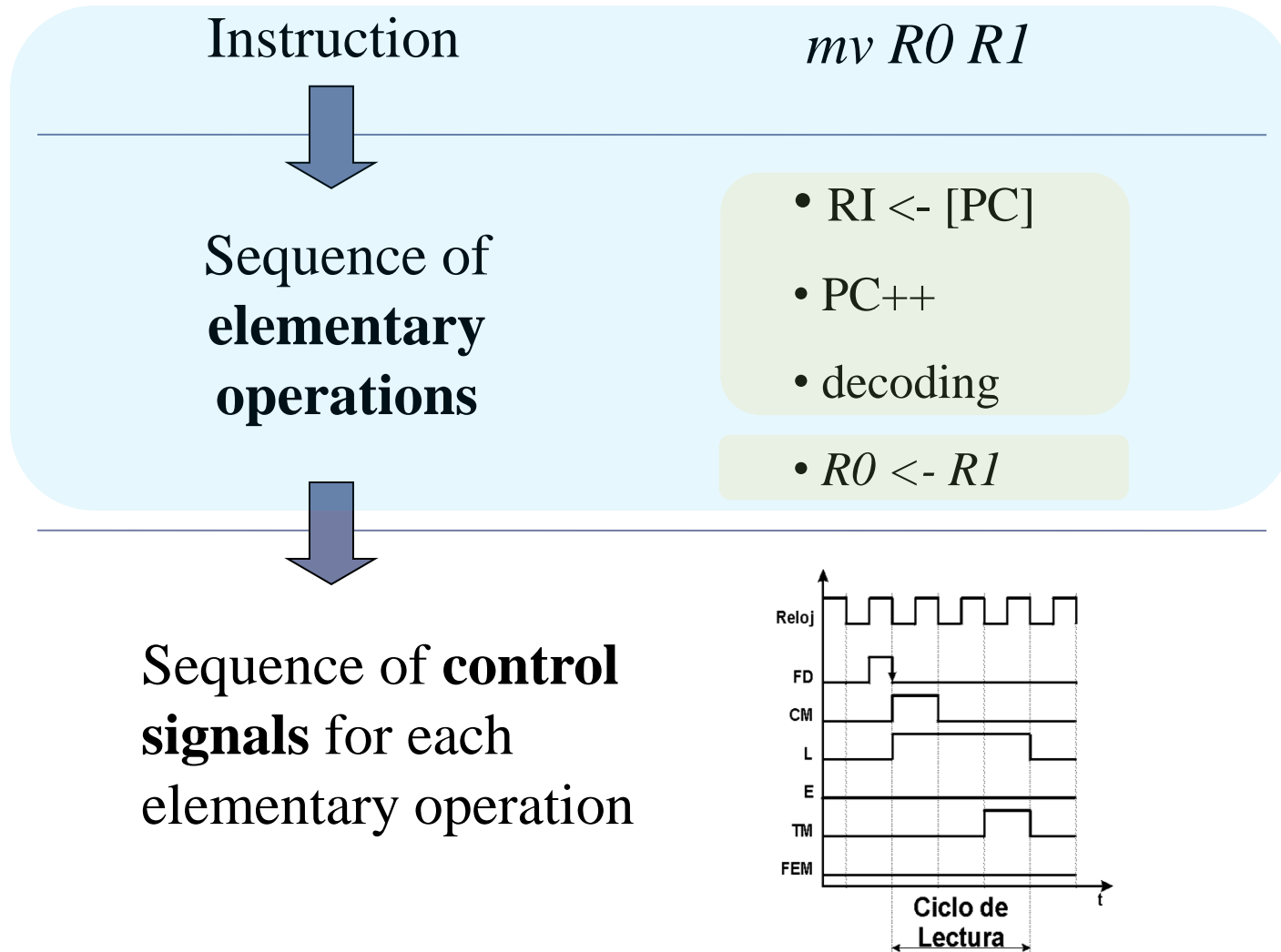


Sequence of **control signals** for each elementary operation



+ level of
hw. details

Description of the Control Unit activity



+ level of
hw. details

Fetch (Elemental Operations)

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

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



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

Possibility of simultaneous operations

Description of the Control Unit activity

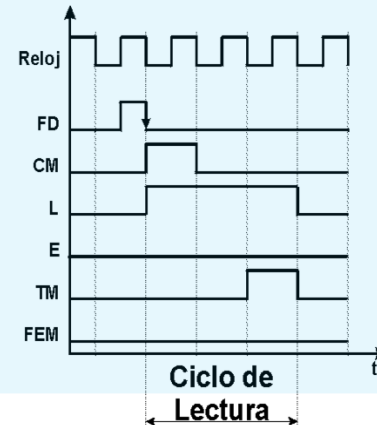
Instruction

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

Fetch (Control Signals)

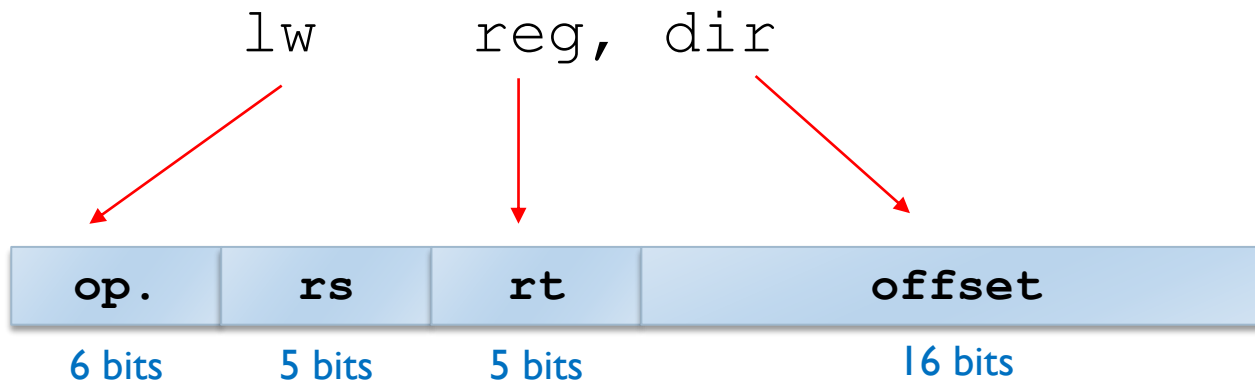
- $RI \leftarrow [PC]$
- $PC++$
- decoding

- ▶ 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

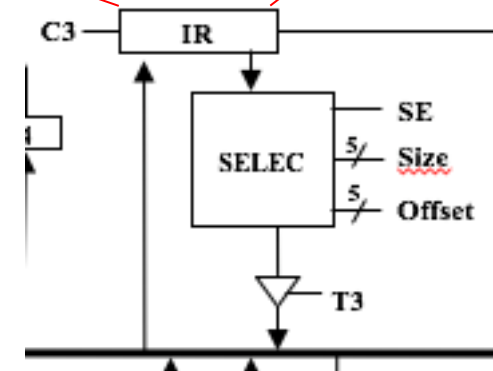
► `lw reg, dir`



Execution of lw reg, dir



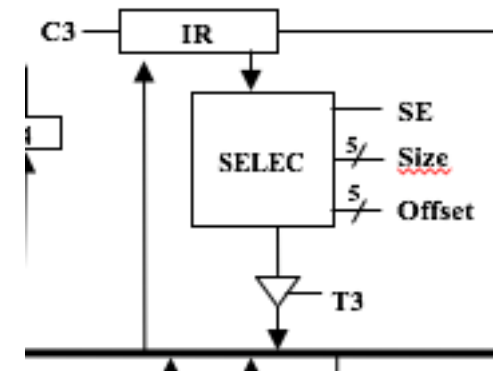
Cycle	Elem. Op.	Control Signals
C1	$MAR \leftarrow PC$	T2, C0
C2	$PC \leftarrow PC + 4,$ $MBR \leftarrow MP$	C2, M2 Ta, R, CI, MI, BW=11
C3	$IR \leftarrow MBR$	T1, C3
C4		
C5		
C6		
C7		



Execution of `lw reg, dir`



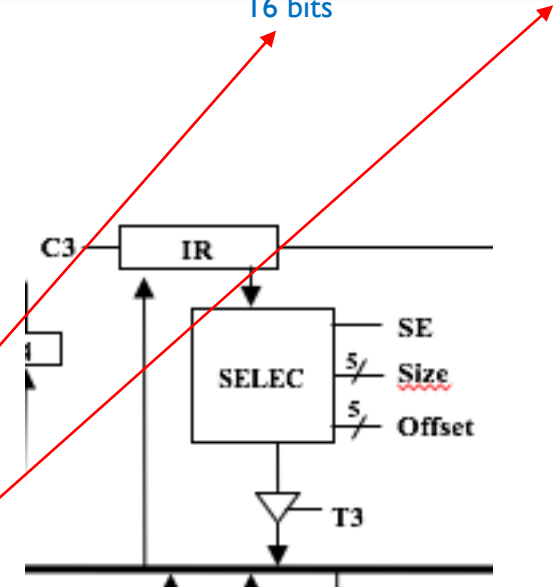
Cycle	Elem. Op.	Control Signals
C1	$MAR \leftarrow PC$	T2, C0
C2	$PC \leftarrow PC + 4,$ $MBR \leftarrow MP$	C2, M2 Ta, R, CI, MI, BW=11
C3	$IR \leftarrow MBR$	T1, C3
C4	Decoding	A0, B=0, C=0
C5		
C6		
C7		



Execution of `lw reg, dir`



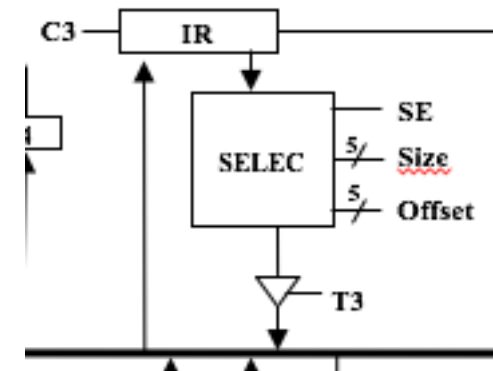
Cycle	Elem. Op.	Control Signals
C1	$MAR \leftarrow PC$	T2, C0
C2	$PC \leftarrow PC + 4,$ $MBR \leftarrow MP$	C2, M2 Ta, R, CI, MI, BW=11
C3	$IR \leftarrow MBR$	T1, C3
C4	Decoding	A0, B=0, C=0
C5	$MAR \leftarrow RI(dir)$	C0, T3, Size = 10000 Offset = 00000
C6		
C7		



Execution of `lw reg, dir`



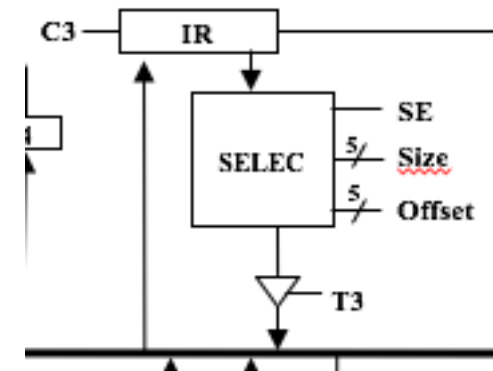
Cycle	Elem. Op.	Control Signals
C1	$MAR \leftarrow PC$	T2, C0
C2	$PC \leftarrow PC + 4,$ $MBR \leftarrow MP$	C2, M2 Ta, R, CI, MI, BW=11
C3	$IR \leftarrow MBR$	T1, C3
C4	Decoding	A0, B=0, C=0
C5	$MAR \leftarrow RI(dir)$	C0, T3, Size = 10000 Offset = 00000
C6	$MBR \leftarrow MP$	Ta, R, CI, MI, BW=11
C7		



Execution of `lw reg, dir`



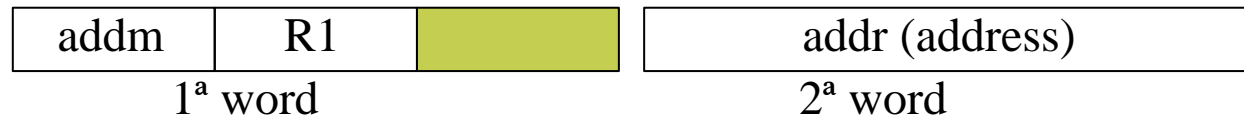
Cycle	Elem. Op.	Control Signals
C1	$MAR \leftarrow PC$	T2, C0
C2	$PC \leftarrow PC + 4,$ $MBR \leftarrow MP$	C2, M2 Ta, R, CI, MI, BW=11
C3	$IR \leftarrow MBR$	T1, C3
C4	Decoding	A0, B=0, C=0
C5	$MAR \leftarrow RI(dir)$	C0, T3, Size = 10000 Offset = 00000
C6	$MBR \leftarrow MP$	Ta, R, CI, MI, BW=11
C7	$\$reg \leftarrow MBR$	T1, RC=id reg, LC



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	$RI \leftarrow RI + RTI$

Simple tips (1 / 2): general phases in an instruction...

A. Fetch + Decod.

B. Fetch operands.

C. Execution

D. Store results

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.- $MBR \leftarrow R_3 + RTI$

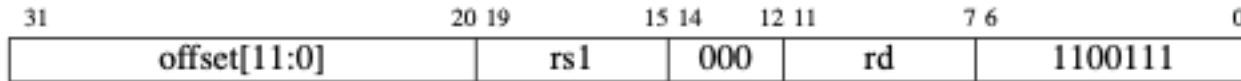
D. Store results

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

Simple tips (2/2):
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.**

jr rs1



Cycle	Elemental Op.	Control signals
C1	$MAR \leftarrow PC$	T2, C0
C2	$PC \leftarrow PC + 4,$ $MBR \leftarrow MP$	C2, MI Ta, R, CI, MI, BW=II
C3	$IR \leftarrow MBR$	T1, C3
C4	Decode	
C5	$PC \leftarrow rs1$	C2, RA=nI de RSI, T9, C2

beqz reg, offset

Cycle	Op. Elemental
C1	$MAR \leftarrow PC$
C2	$PC \leftarrow PC + 4,$ $MBR \leftarrow MP$
C3	$IR \leftarrow MBR$
C4	Decode
C5	$\$reg + \0
C6	Si $SR.Z == 0$ jump to fetch
C7	$RT2 \leftarrow PC$
C8	$RT1 \leftarrow IR(offset)$
C9	$PC \leftarrow RT1 + RT2$

Si $reg == 0$
 $PC \leftarrow PC + offset$

► Instructions that fit in one word:

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

ARCOS Group

uc3m | Universidad **Carlos III** de Madrid

L4: The processor (1)

Computer Structure

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

