

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

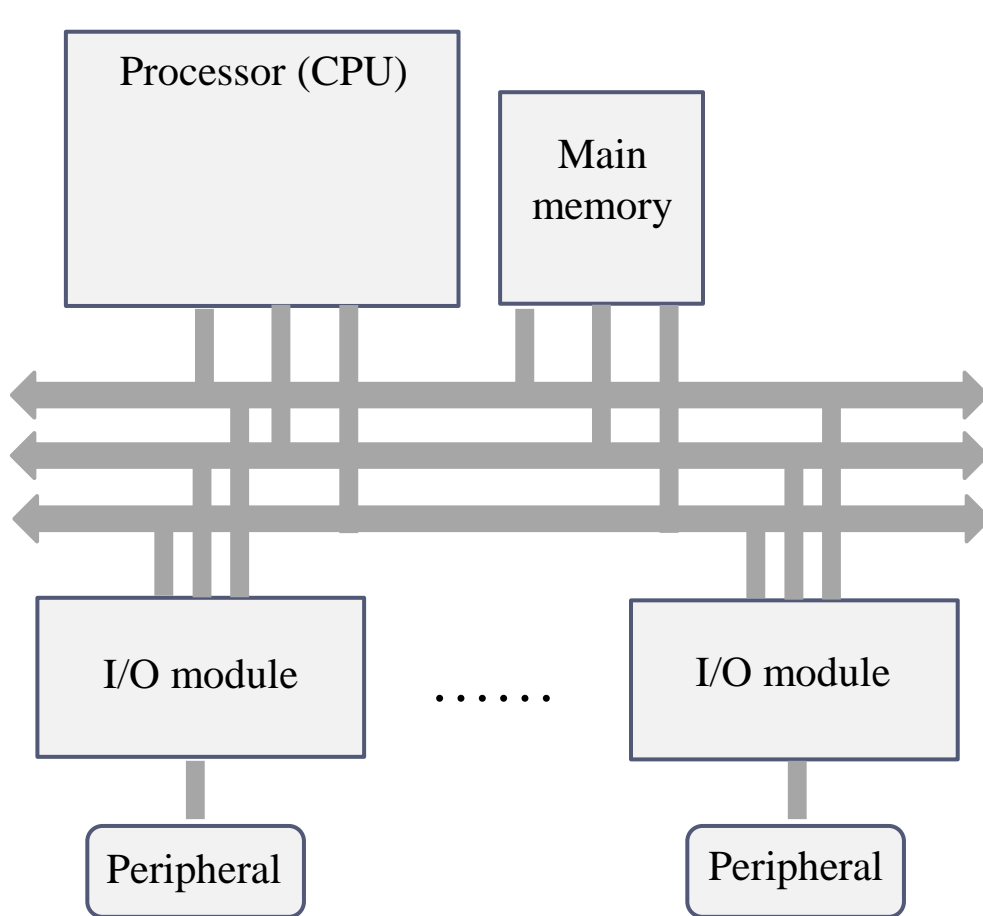
Contents

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

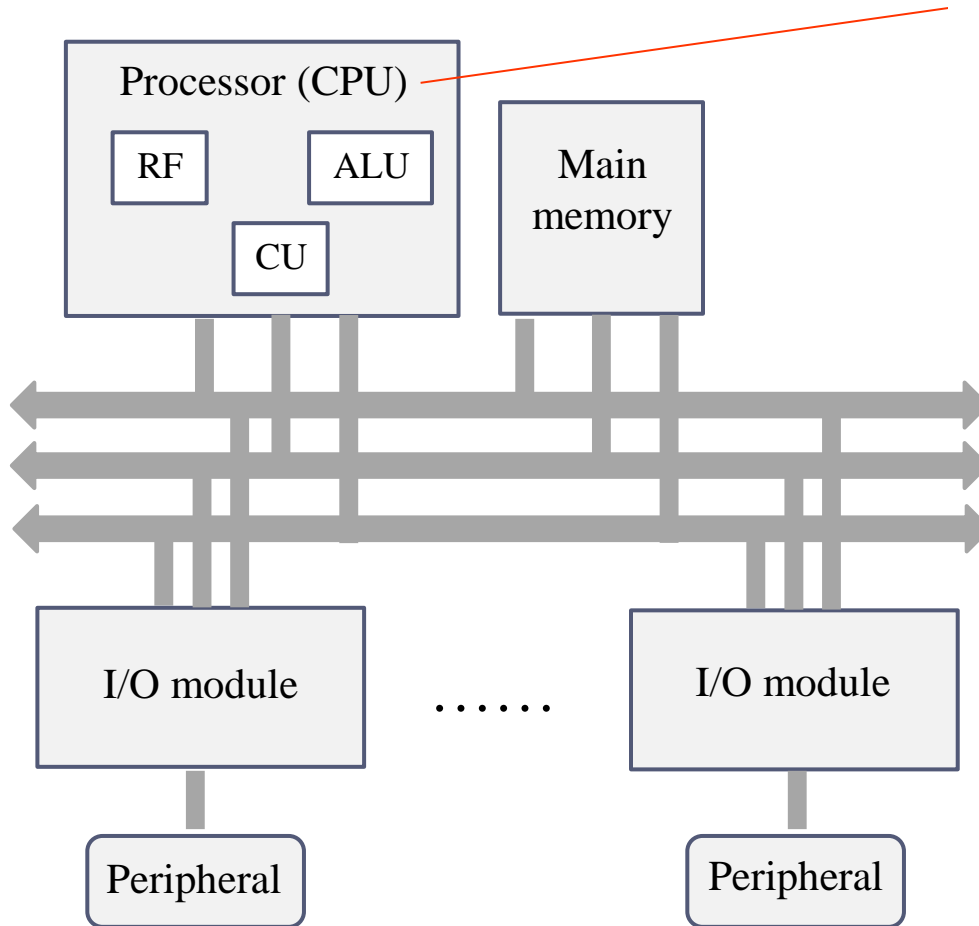
review



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

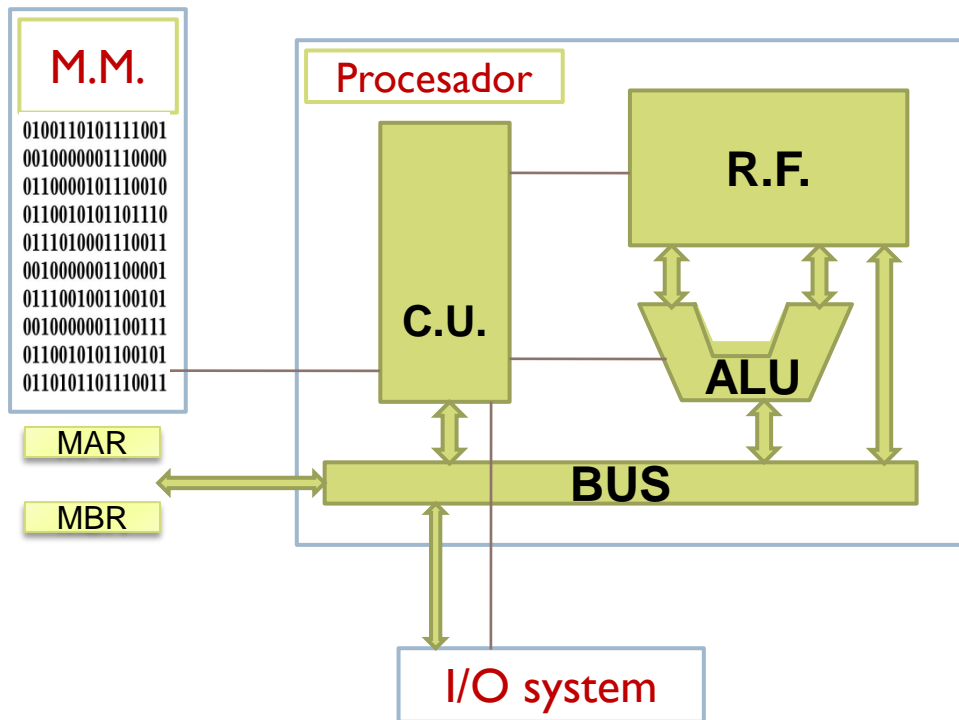
Processor components

review



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

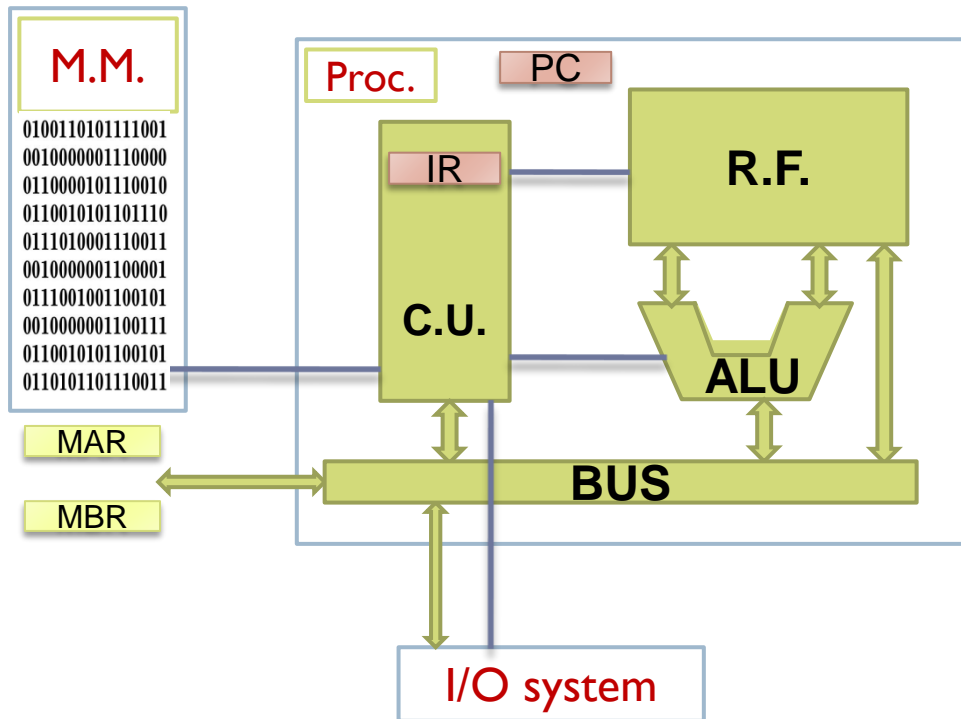
Main motivation



- 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



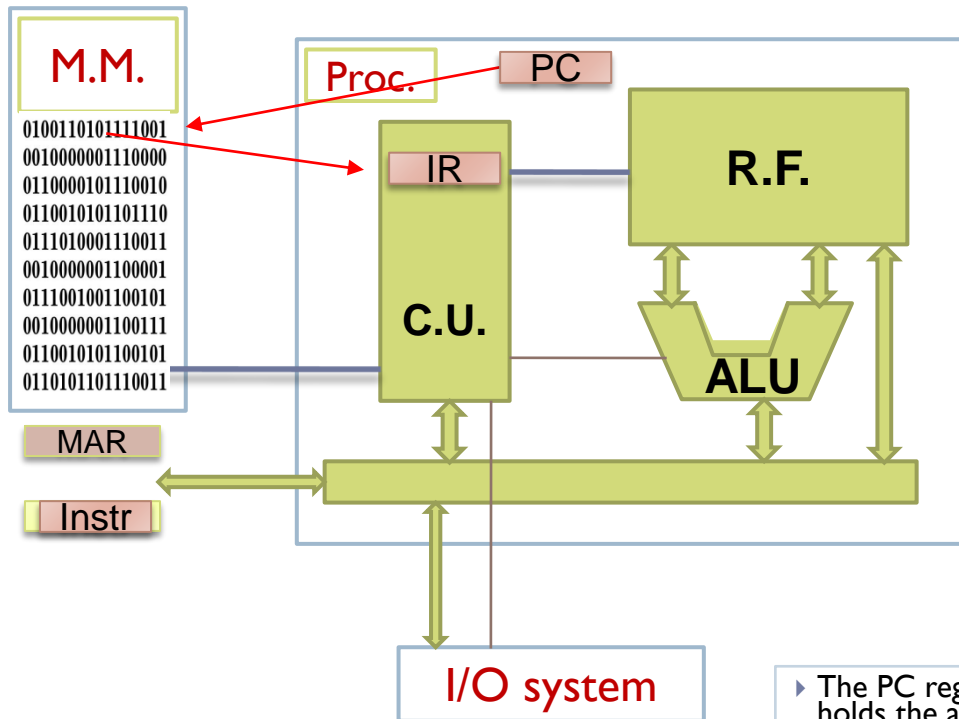
- Each element of the computer has inputs, outputs and control signals.
- At each clock cycle, the Control Unit (C.U.) sends the control signals via the control bus wires.
- Control signals indicate what value to output:
 - Move from an input to an output: $S = Ex$
 - Transform an input: $S = f(E)$

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

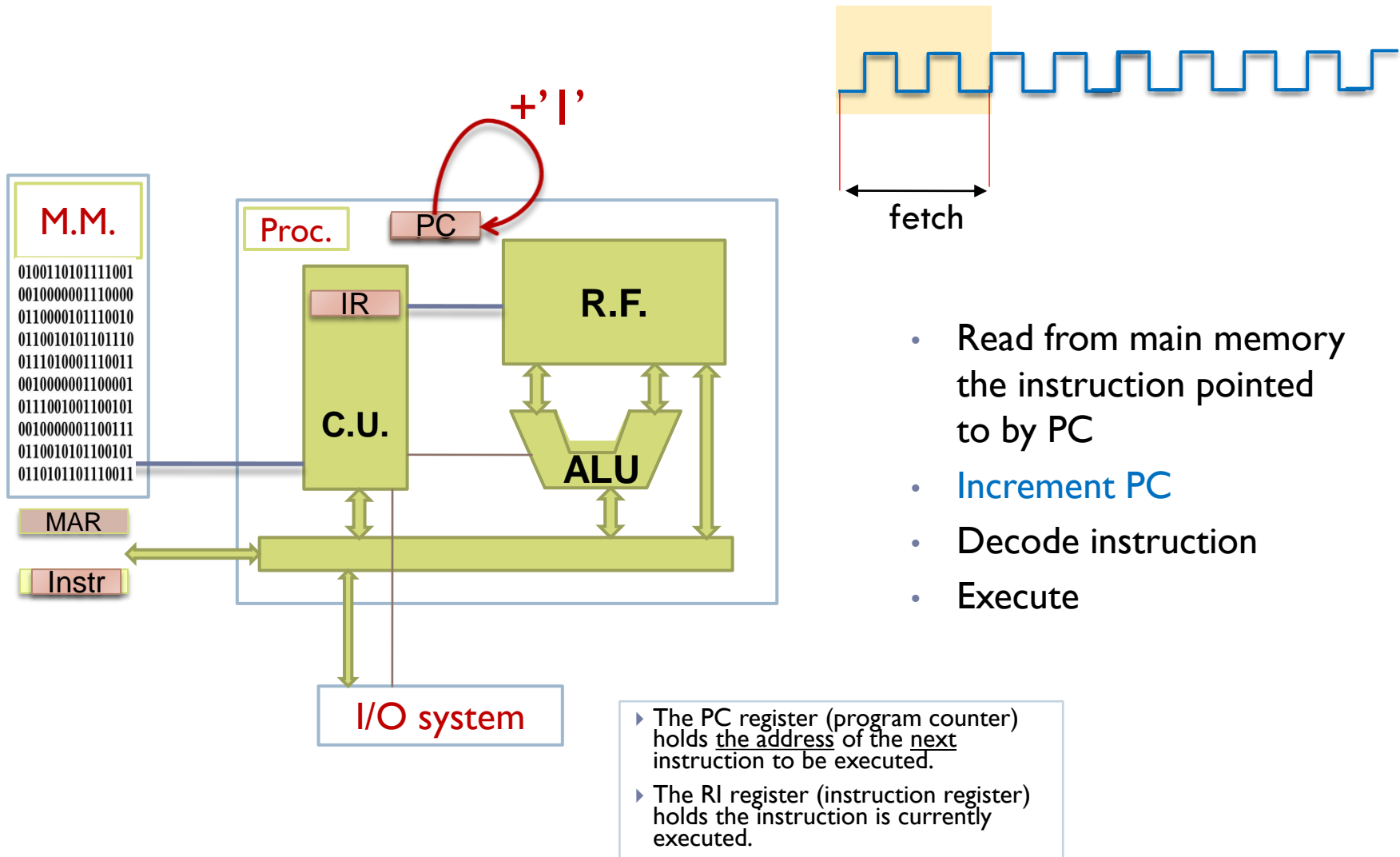
How C.U. works...



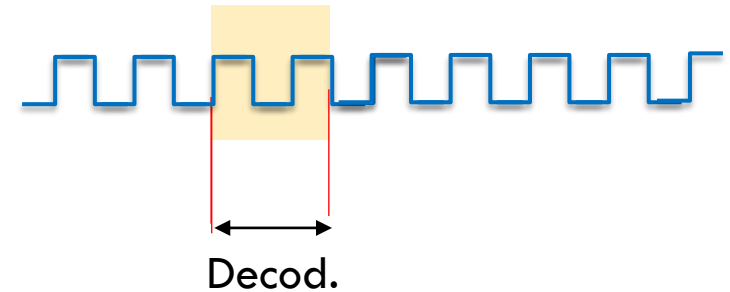
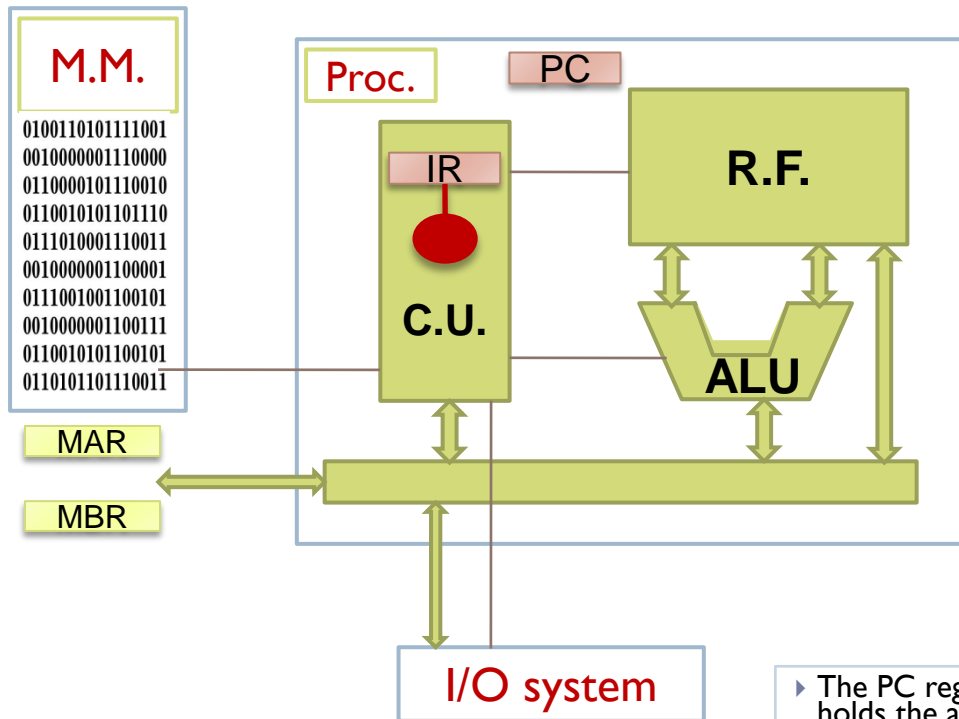
- Read from main memory the instruction pointed to by PC
- Increment PC
- Decode instruction
- Execute

- ▶ 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...



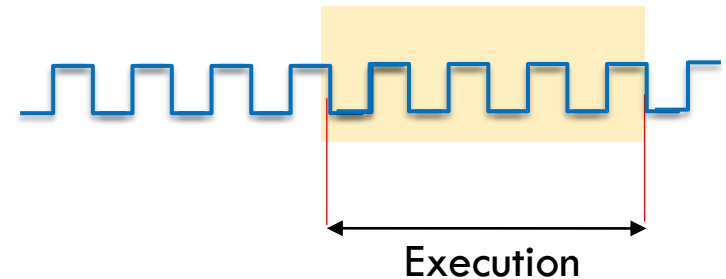
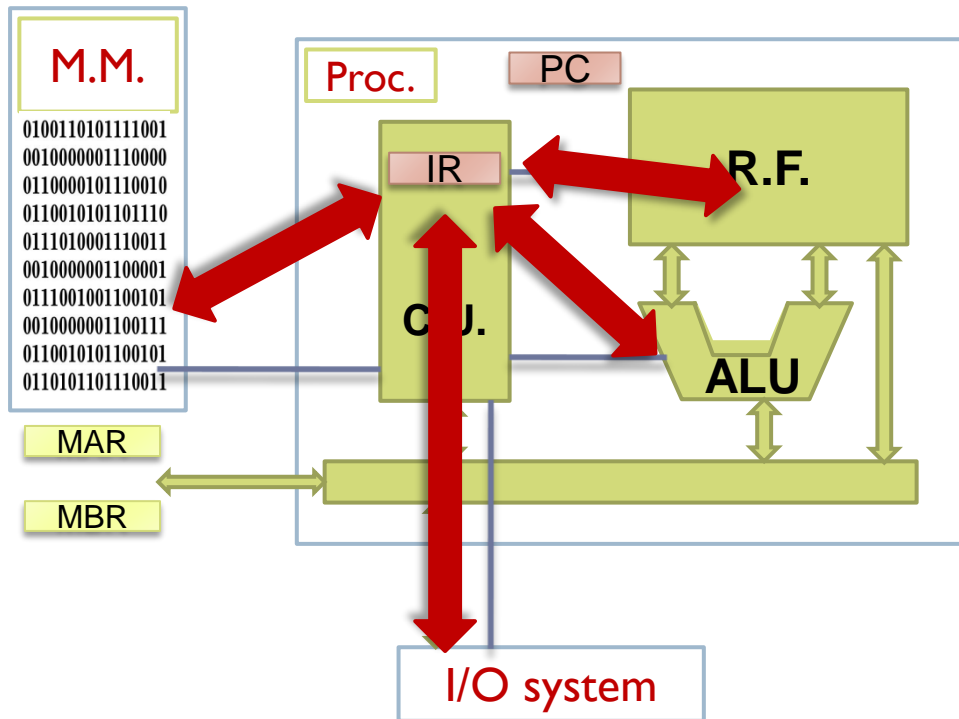
How C.U. works...



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

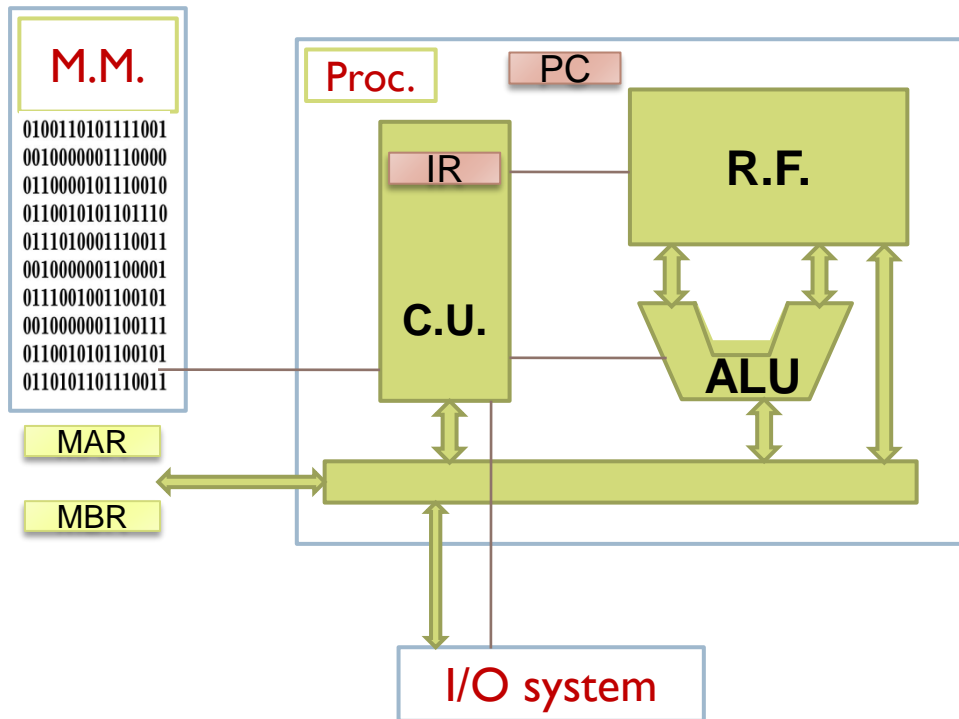
- ▶ The PC register (program counter) holds the address of the next instruction to be executed.
- ▶ The RI register (instruction register) holds the instruction is currently executed.

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.



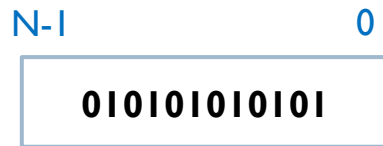
- Resolving anomalous situations
 - Illegal instructions
 - Illegal memory accesses
 - ...
- Attend to interruptions
- Control the communication with the peripherals.

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



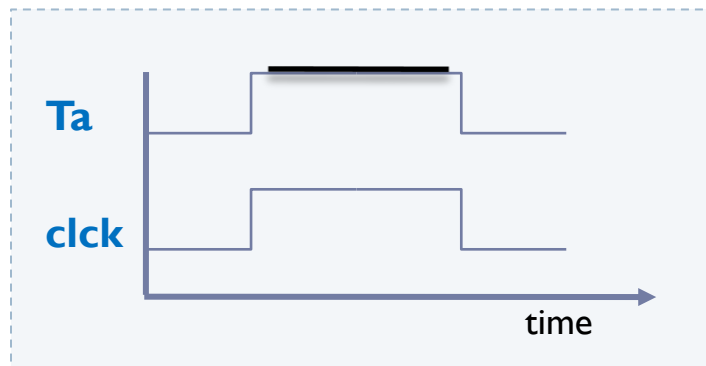
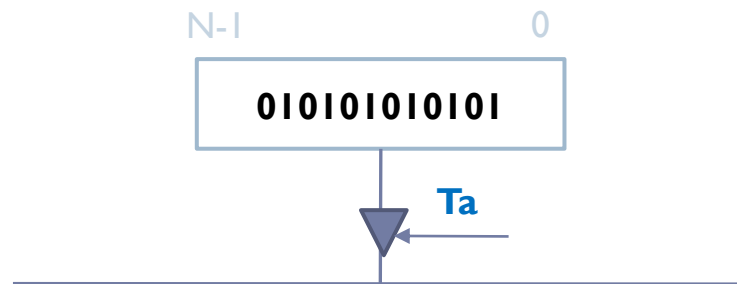
▶ Register

- ▶ 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



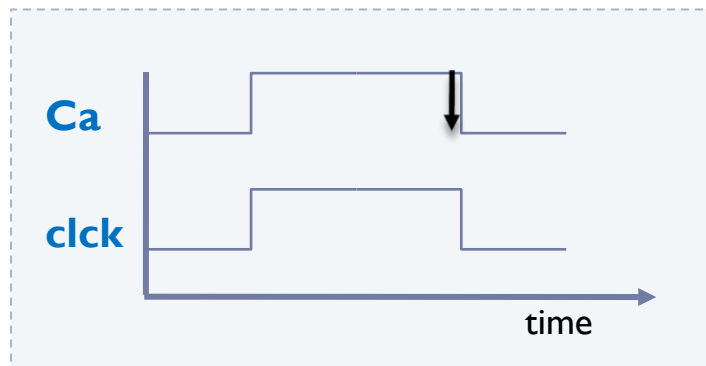
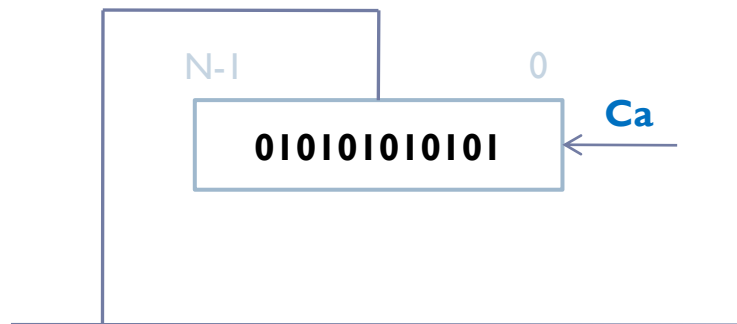
► Tri-state

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

► IMPORTANT

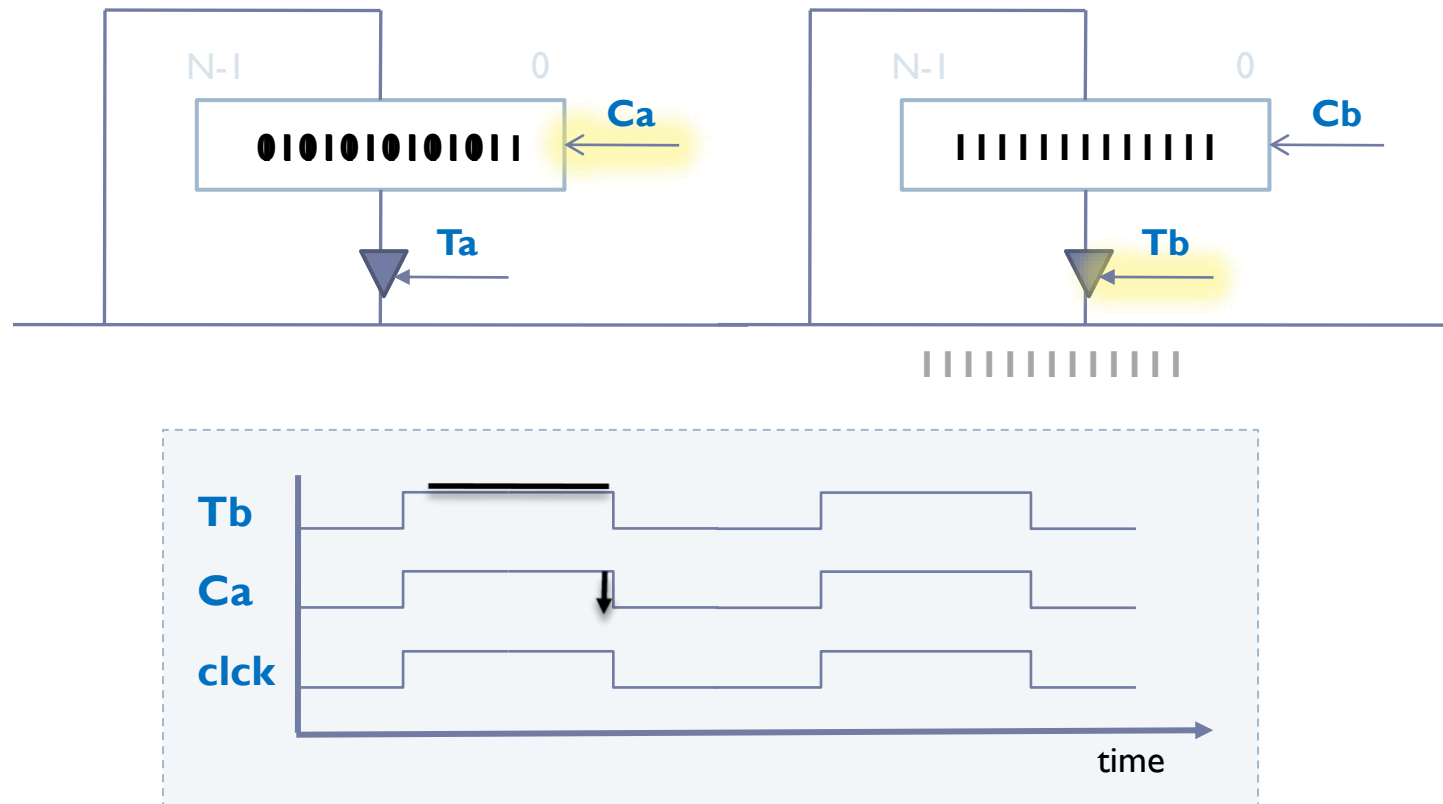
- Two or more tri-states cannot be activated on the same bus at the same time.

Signals: load in register

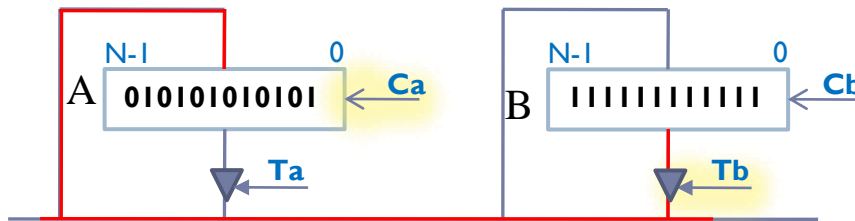


- ▶ Load in register
 - ▶ Let store the input value at the clock falling edge
 - ▶ During the clock level the register keeps the inner (old) value.
 - ▶ At the end of the clock cycle (falling edge) is when the inner value is updated ↓
- ▶ **IMPORTANT**
 - ▶ Therefore, in the following cycle, the new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

Sequence of signals



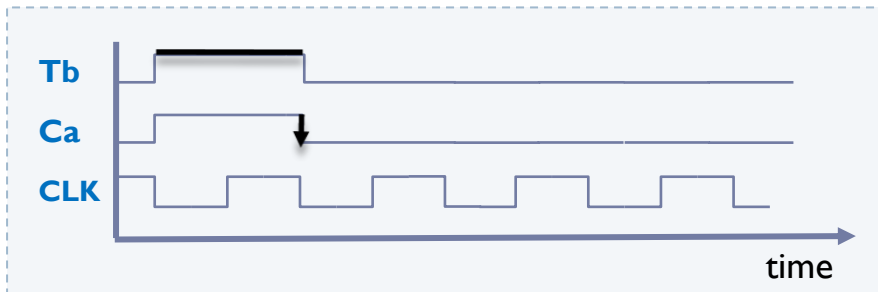
Example of *transfer* elemental operation



▶ Elementary transfer operation:

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

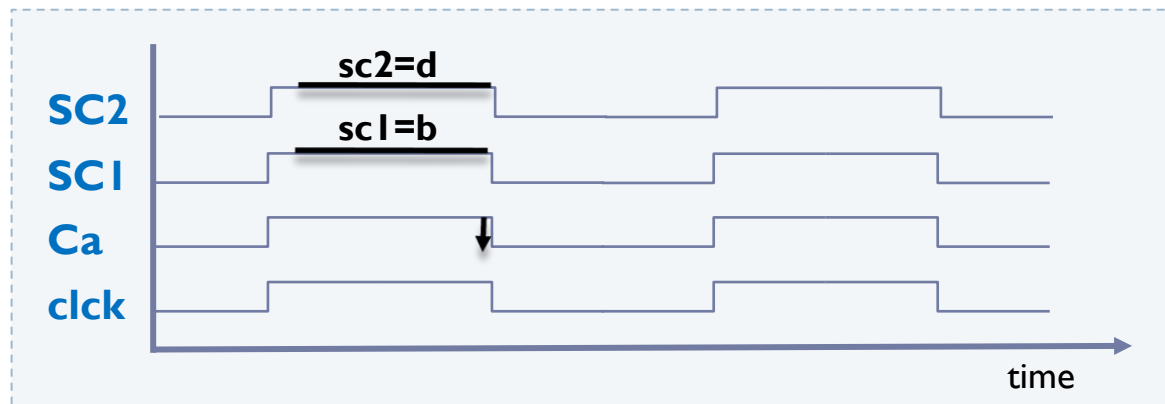
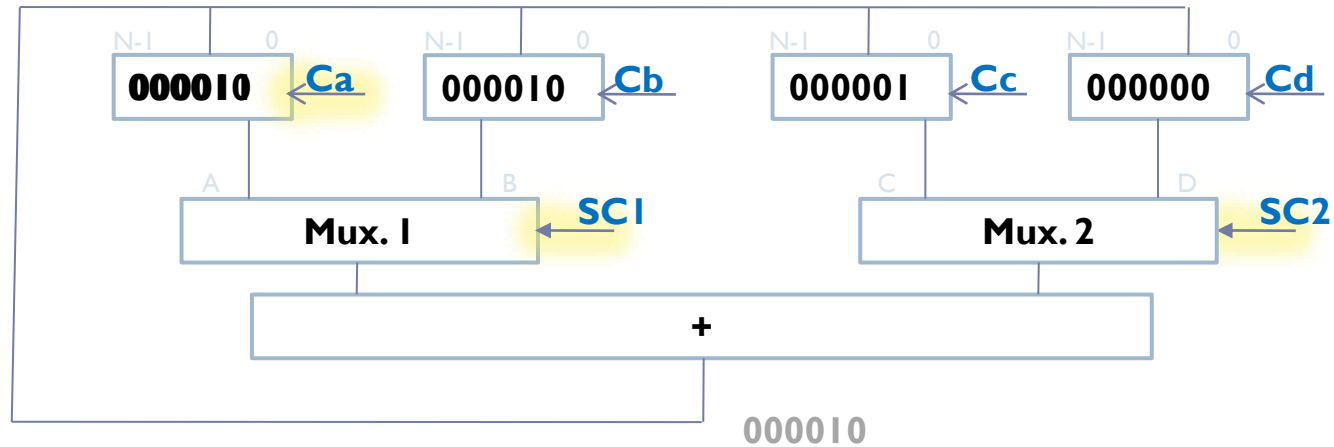
xx: $A \leftarrow B$ [Tb, Ca]



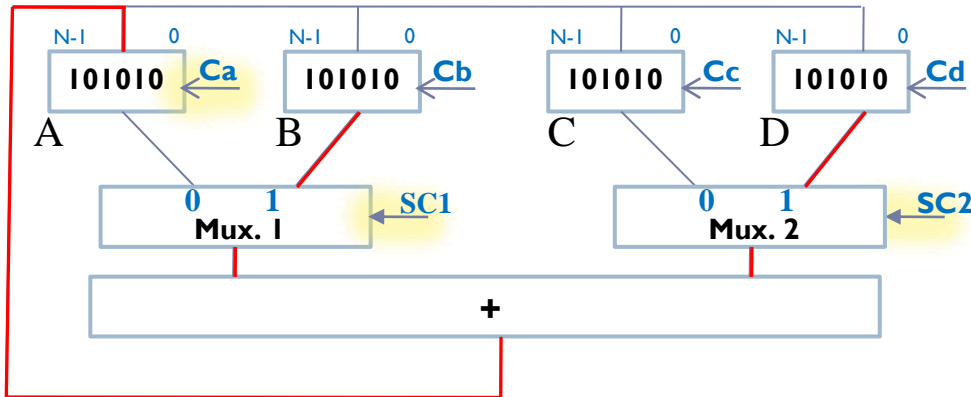
▶ IMPORTANT

- ▶ 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.

Sequence of signals



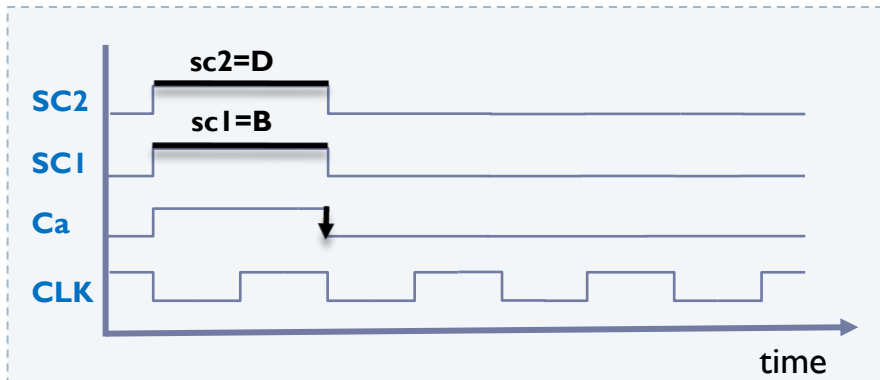
Example of *process* elemental operation



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

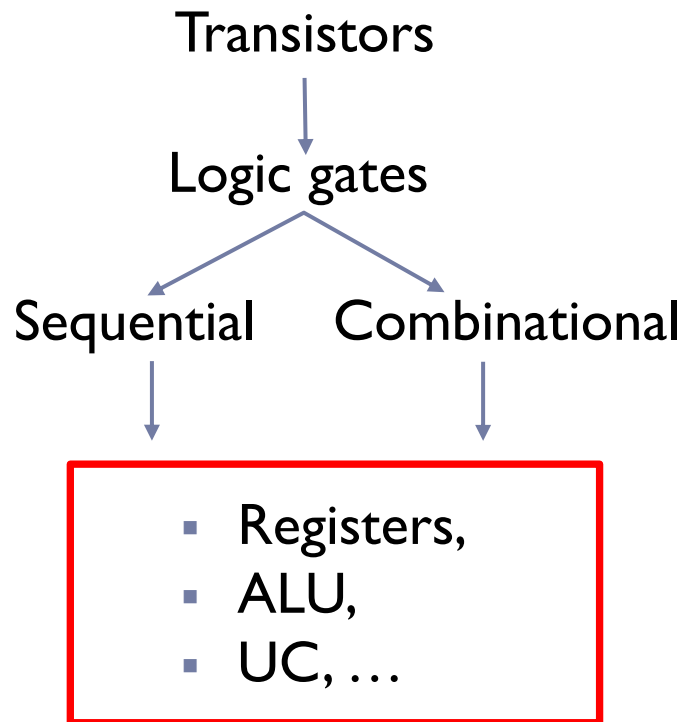
- ▶ $R1 \leftarrow R2 + R2$



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

Review all components...

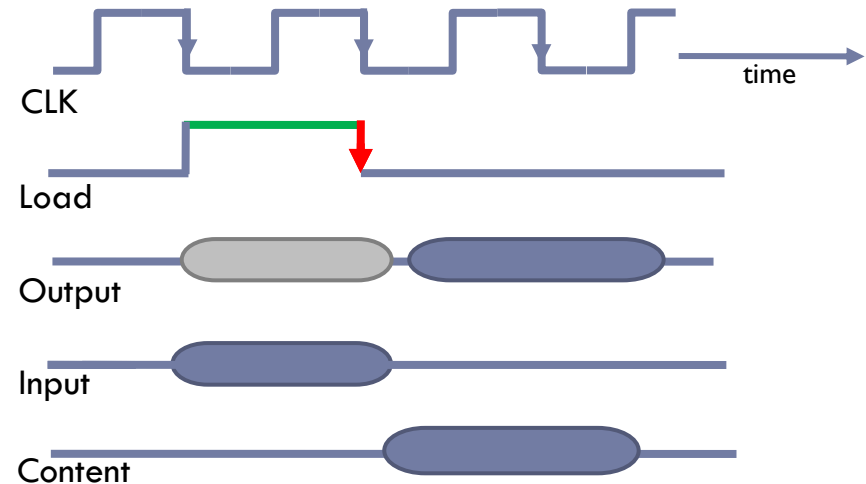
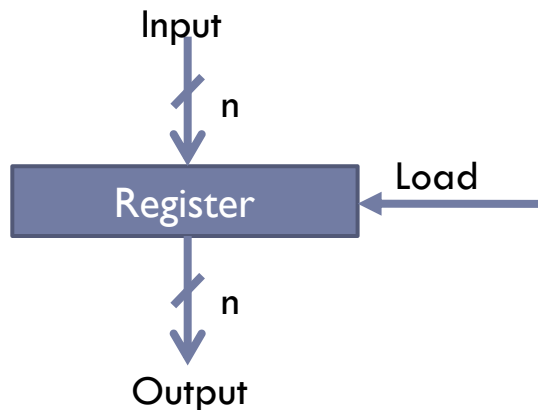
- ▶ Binary system based on 0 y 1
- ▶ Building blocks:



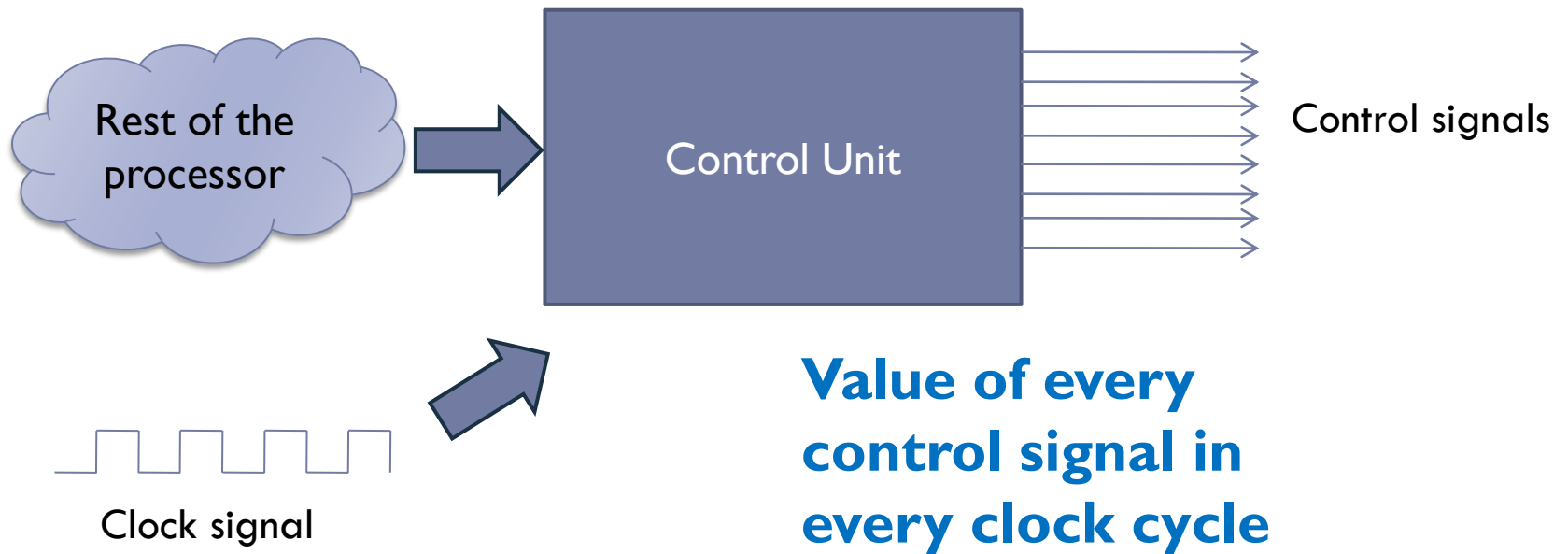
Review all components...

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)



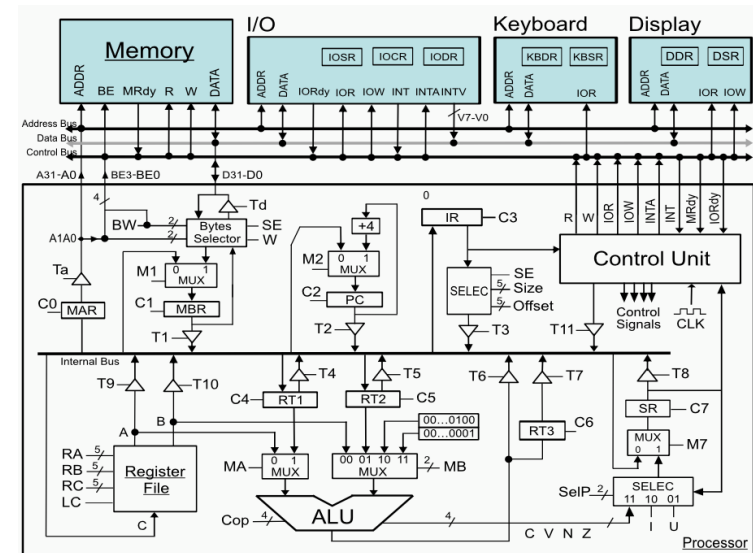
Contents

1. Computer elements
2. Processor organization
3. Control unit
4. Execution of instructions
5. 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 and WepSIM Simulator

► Elemental Processor (E.P.):



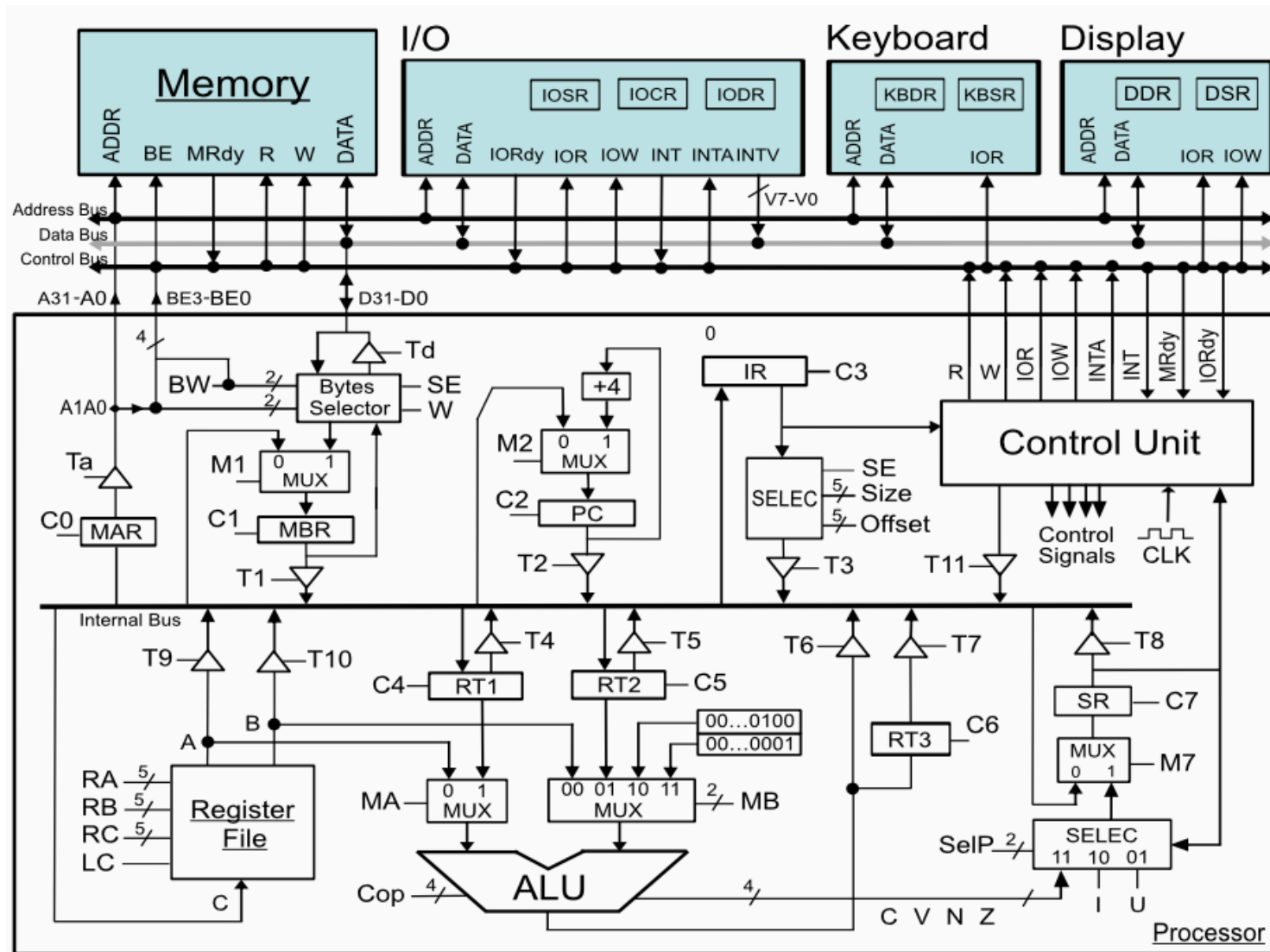
► WepSIM simulates the E.P.:

► <https://wepsim.github.io/wepsim/>

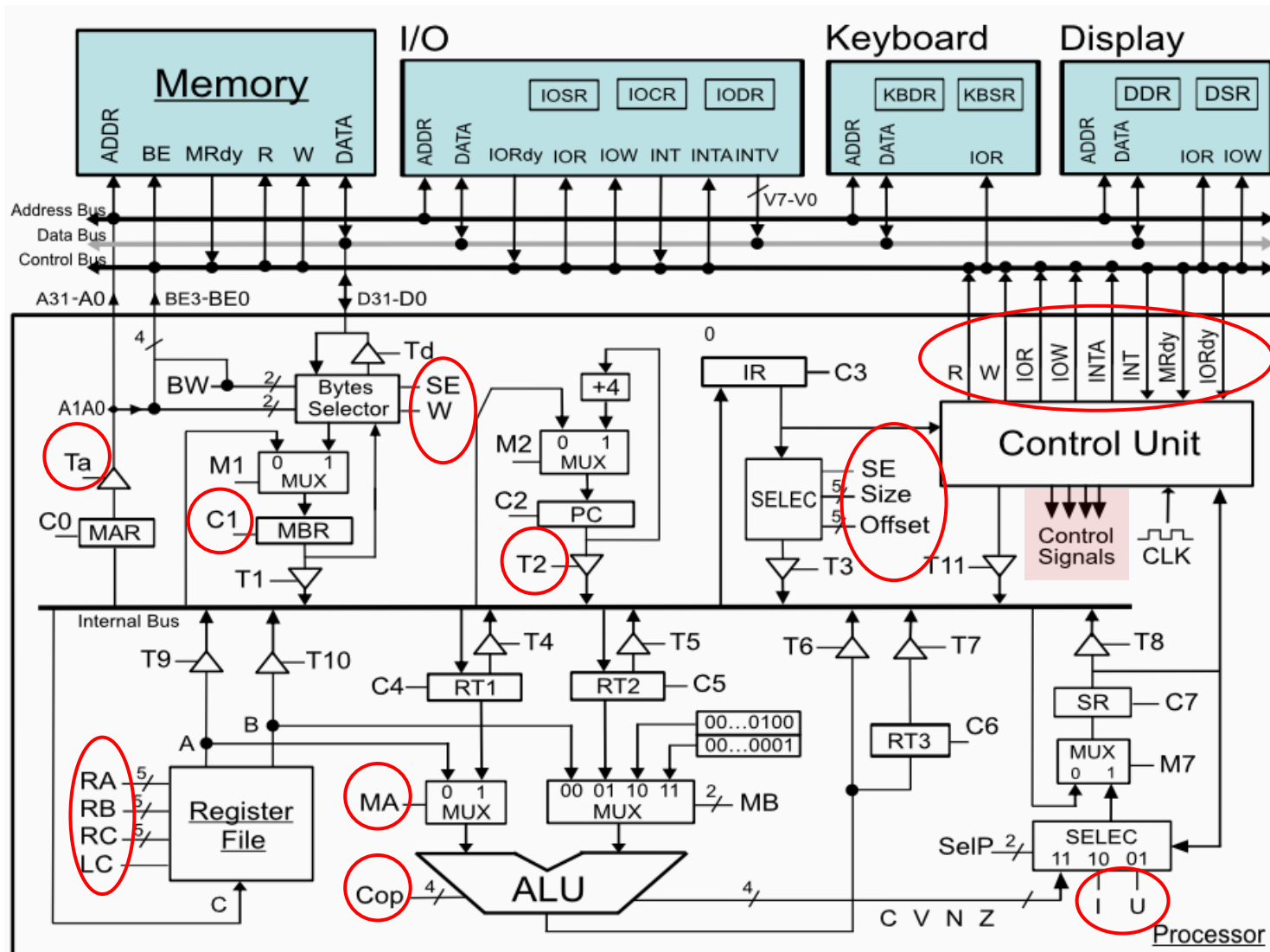
Main features of the elemental processor

- ▶ 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 and 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

Structure of an elementary computer



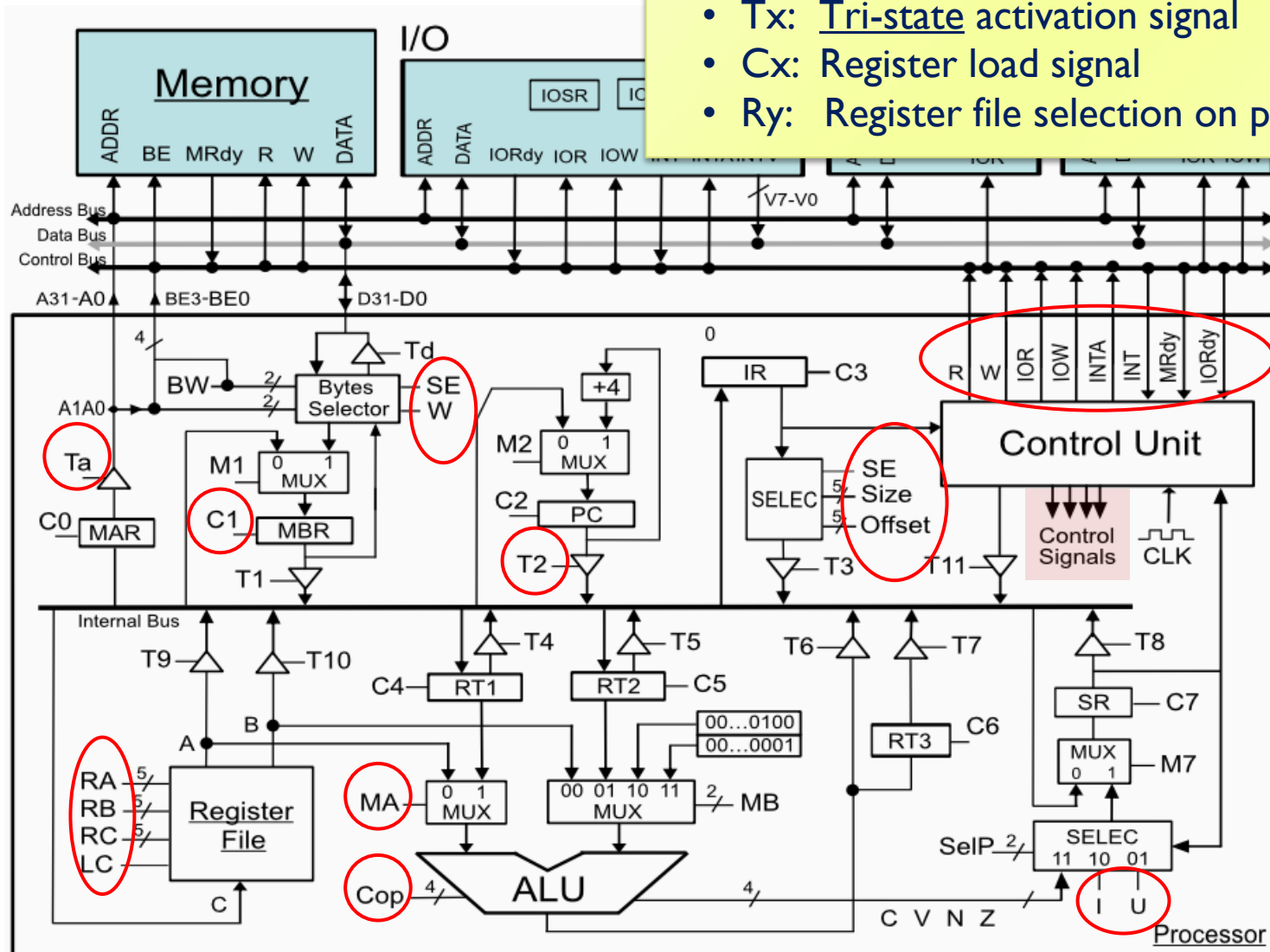
Control signals



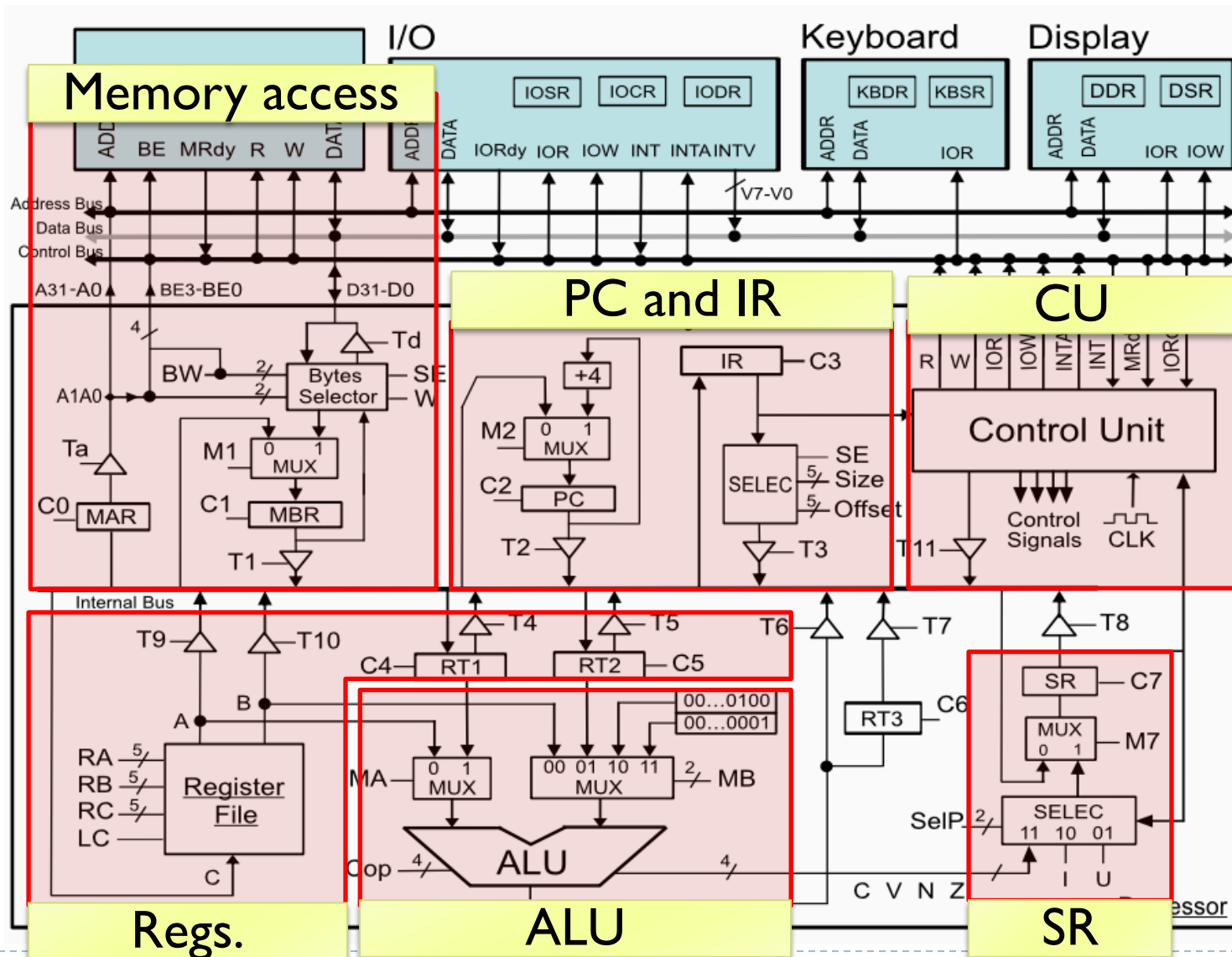
Control signals

General nomenclature:

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

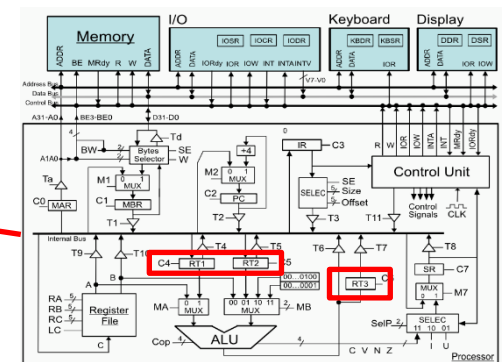
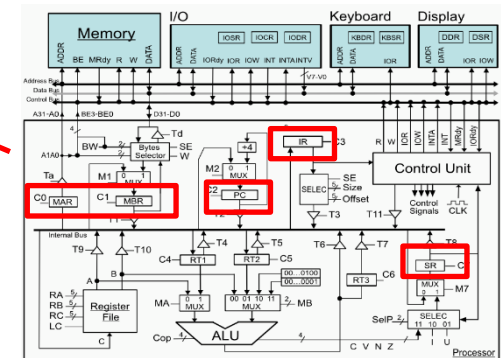
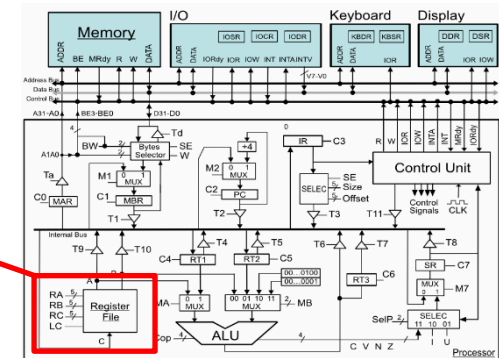


Control signals



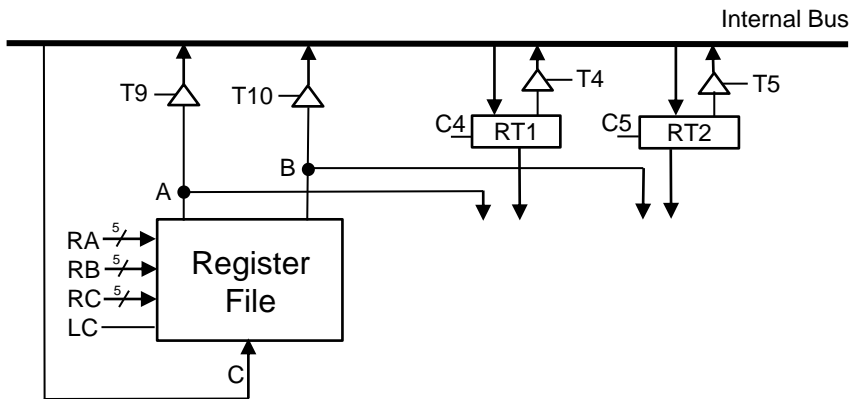
Registers

- ▶ 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.)



Structure of an elementary computer

Registers



Nomenclature:

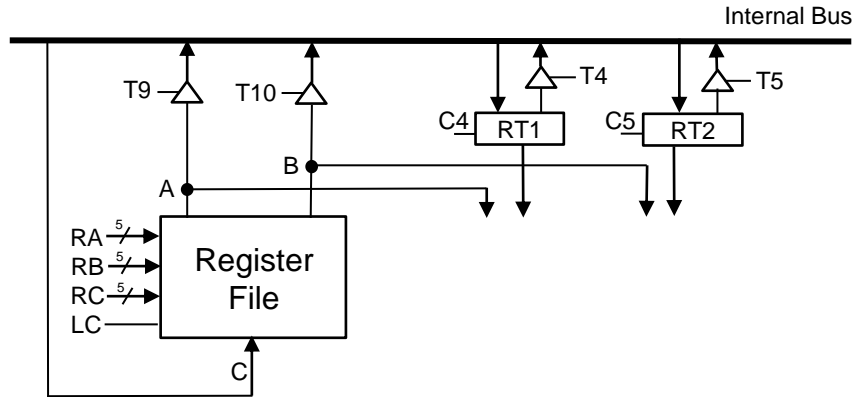
- Ry: Register file selection
- Mx: Selection in multiplexer
- Tx: Tri-state activation signal
- Cx: Register load signal

▶ 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

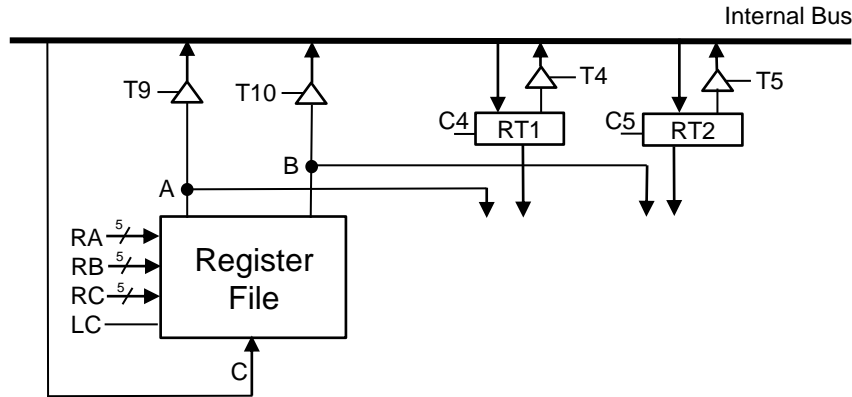
elemental operations in registers



► **SWAP R1 R2**

Example

elemental operations in registers

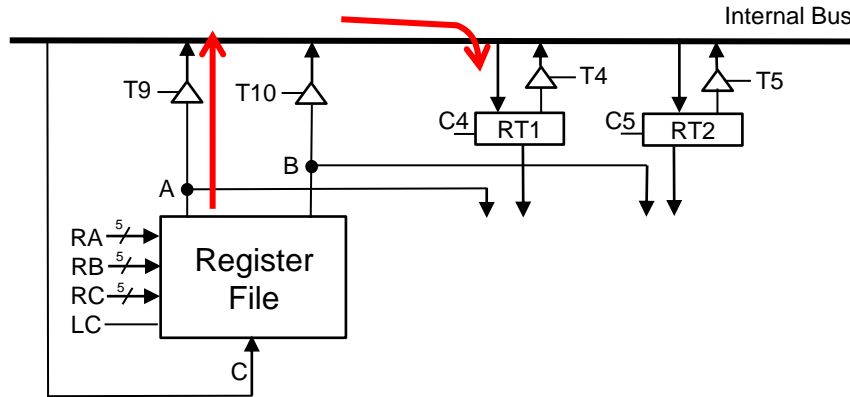


► **SWAP R1 R2**

| Elemental Op. | Signals |
|---------------|---------|
| | |
| | |
| | |

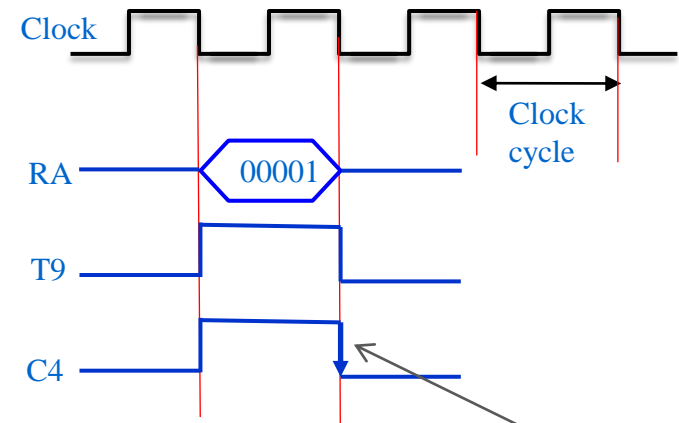
Example

elemental operations in registers



► **SWAP R1 R2**

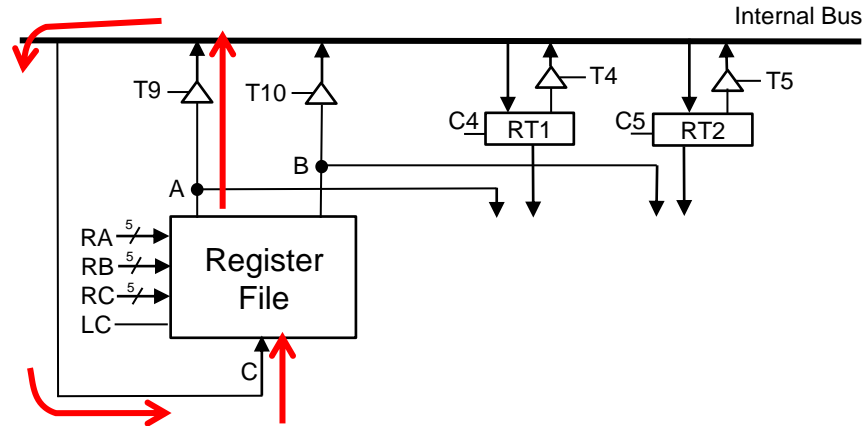
| Elemental Op. | Signals |
|---------------------|--------------------|
| $RT1 \leftarrow R1$ | $RA=00001, T9, C4$ |
| | |
| | |



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

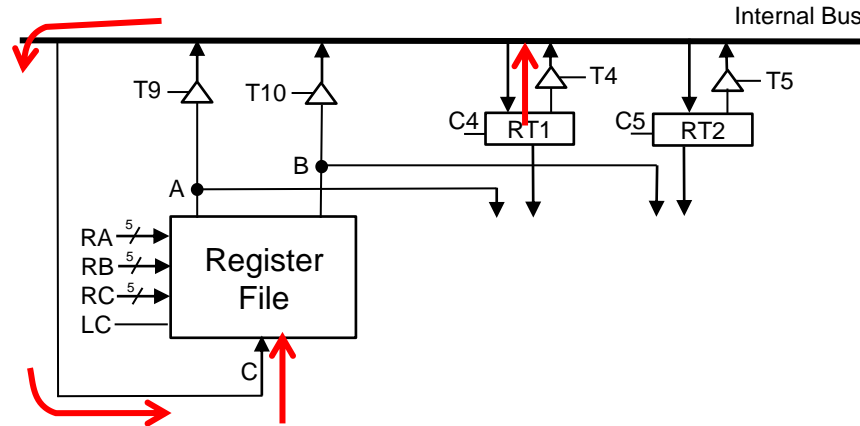


► **SWAP R1 R2**

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

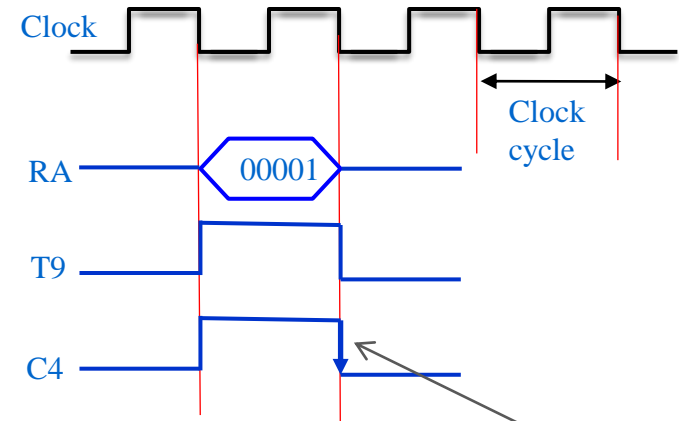
Example

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 |

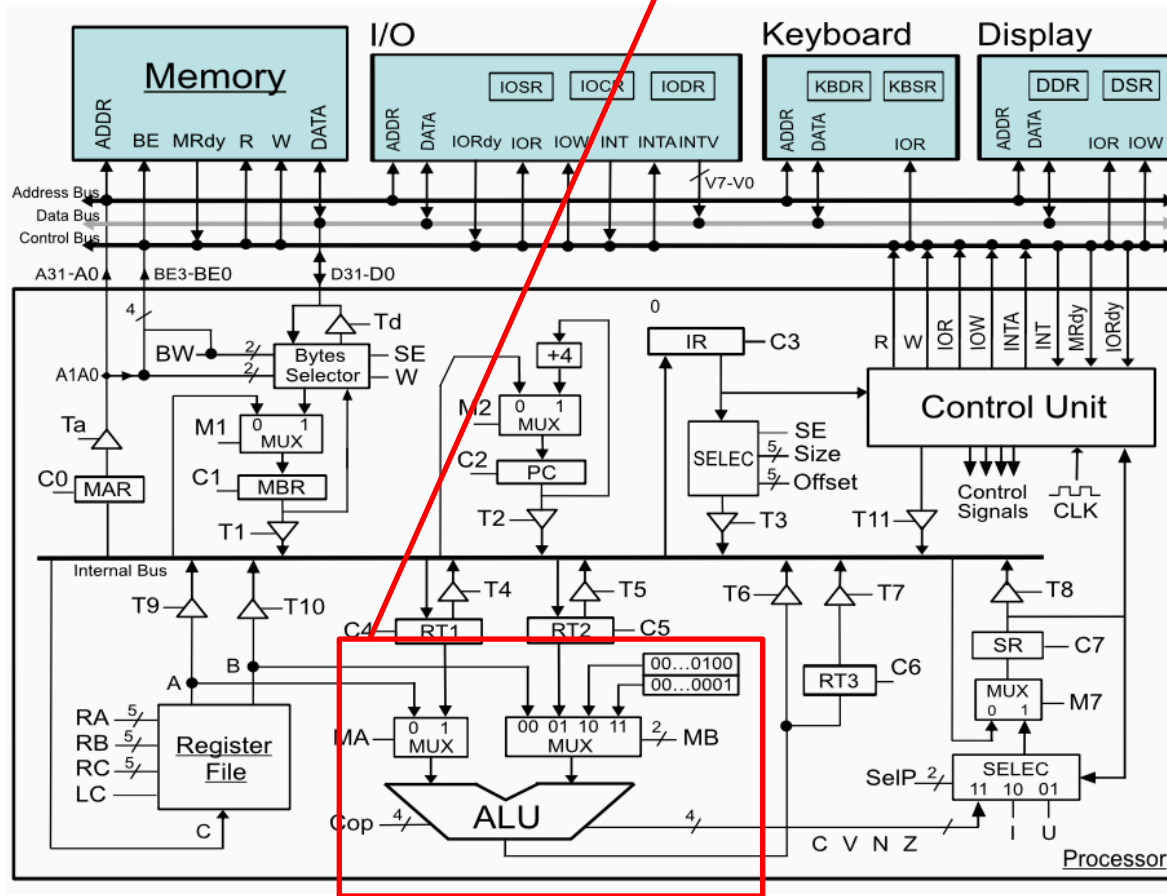


The data is loaded on RT1 on the falling edge.
It will be available on RT1 during the **next** cycle.

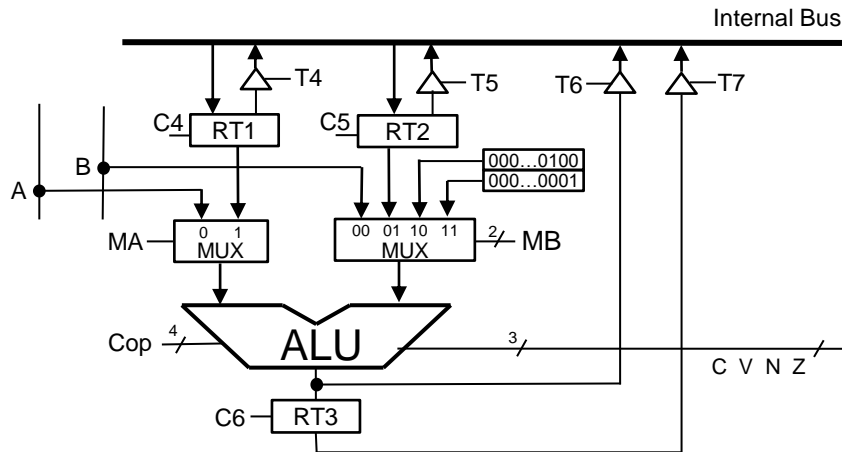
Structure of an elementary computer

ALU

arithmetic logic unit (ALU)



Control Signals

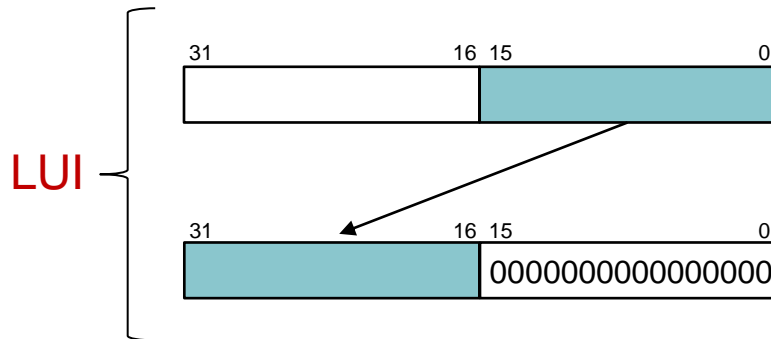
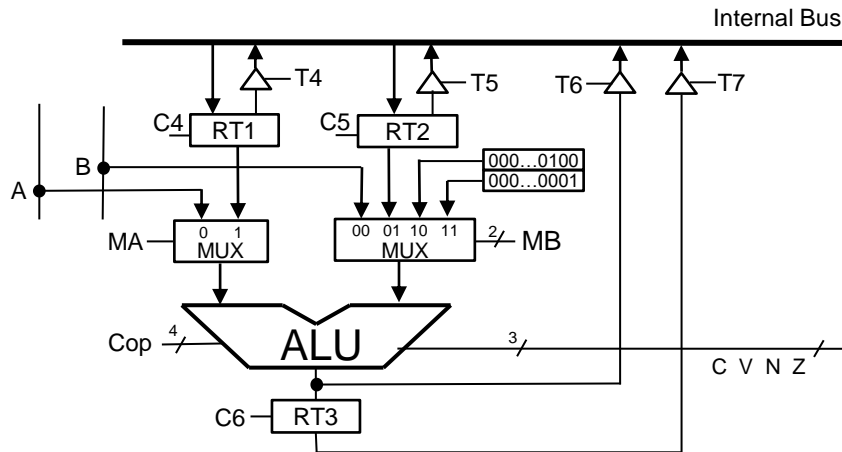


▶ ALU

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

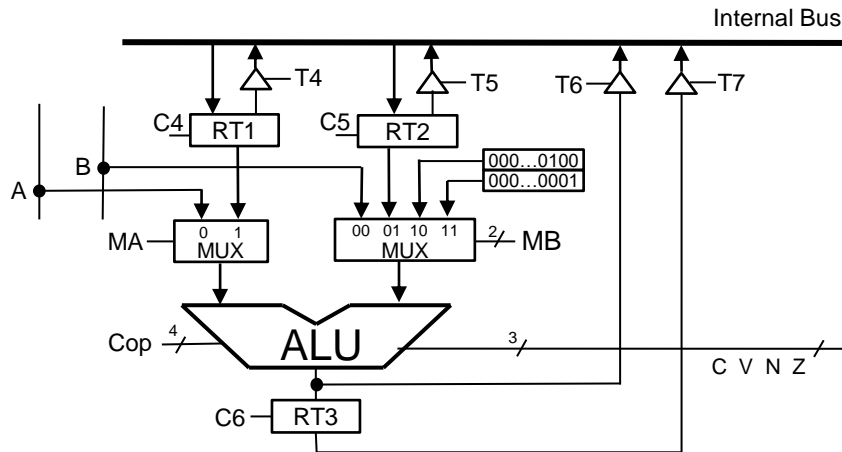
| Cop (Cop ₃ -Cop ₀) | Operation |
|--|--|
| 0000 | NOP |
| 0001 | A and B |
| 0010 | A or B |
| 0011 | not (A) |
| 0100 | A xor B |
| 0101 | Shift Right Logical (A) B= number of bits to shift |
| 0110 | Shift Right Arithmetic (A) B= number of bits to shift |
| 0111 | Shift left (A) B= number of bits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

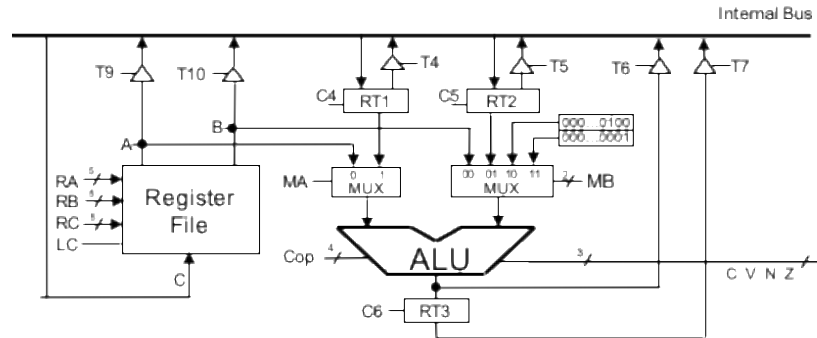


| Result | C | V | N | Z |
|-------------------------------------|---|---|---|---|
| Positive result (0 is considered +) | 0 | 0 | 0 | 0 |
| Result == 0 | 0 | 0 | 0 | 1 |
| Negative result | 0 | 0 | 1 | 0 |
| Overflow | 0 | 1 | 0 | 0 |
| Division by zero | 0 | 1 | 0 | 1 |
| Carrying at bit 32 | 1 | 0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

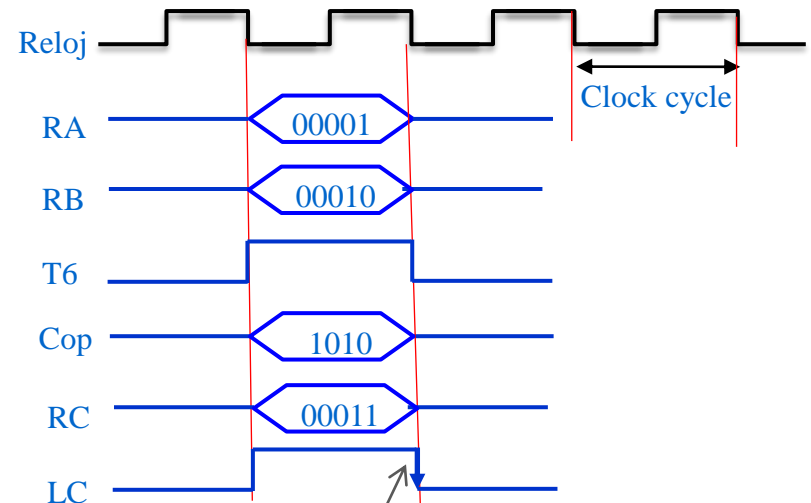
Example

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 |

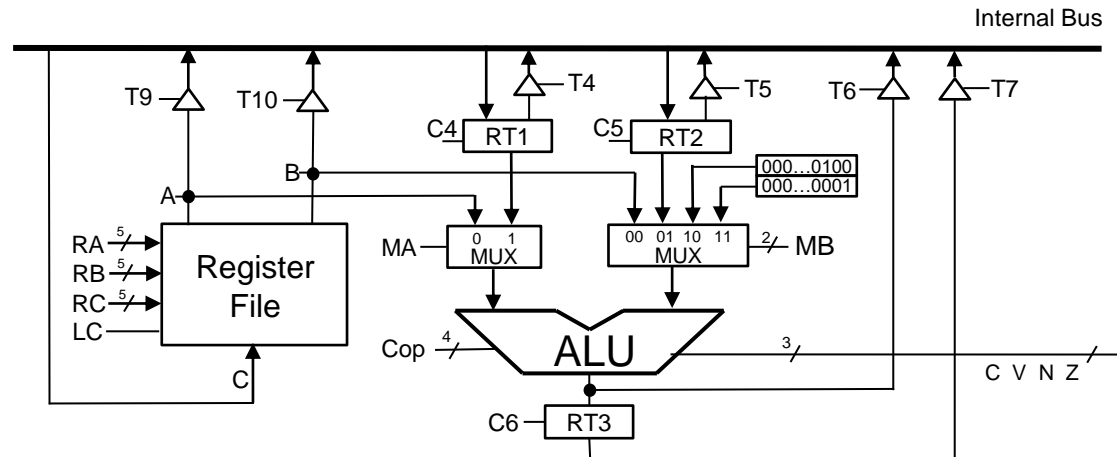


Rest of signals at 0.

The load is performed on R3 on the falling edge.

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

elemental operations in ALU

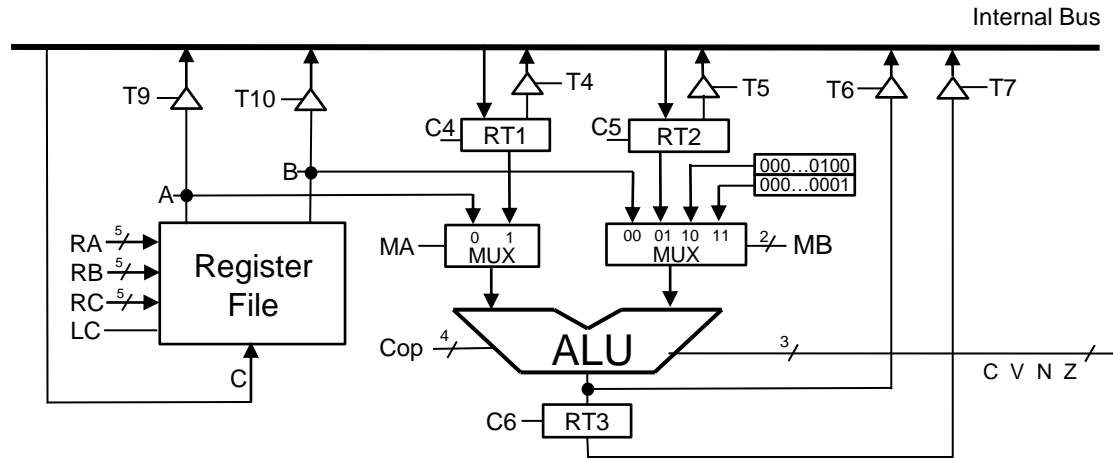


► **SWAP R1 R2**

| 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**

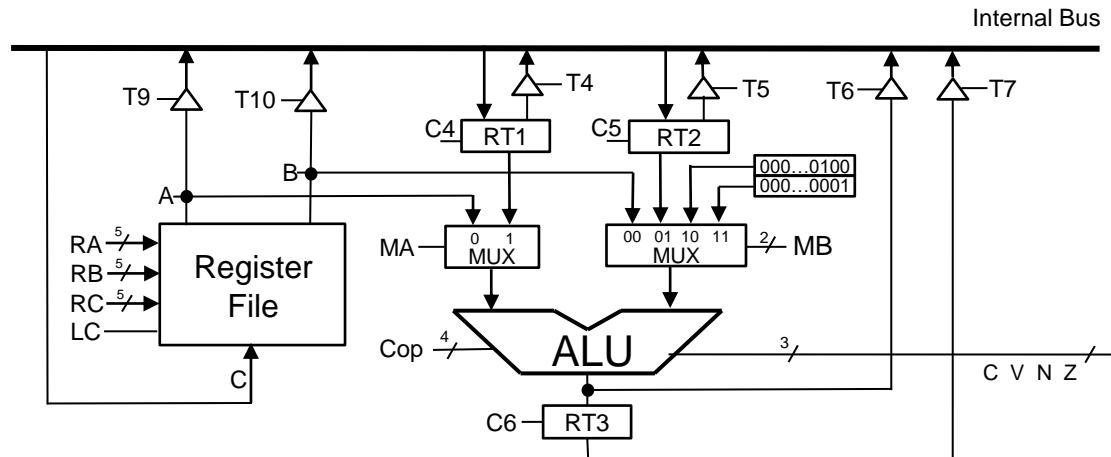
| Elem. Op. | Signals |
|---------------------|--------------------|
| $RT1 \leftarrow R1$ | RA=1, T9, C4 |
| $R1 \leftarrow R2$ | RA=2, T9, RC=1, LC |
| $R2 \leftarrow RT1$ | T4, RC=2, LC |

► 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

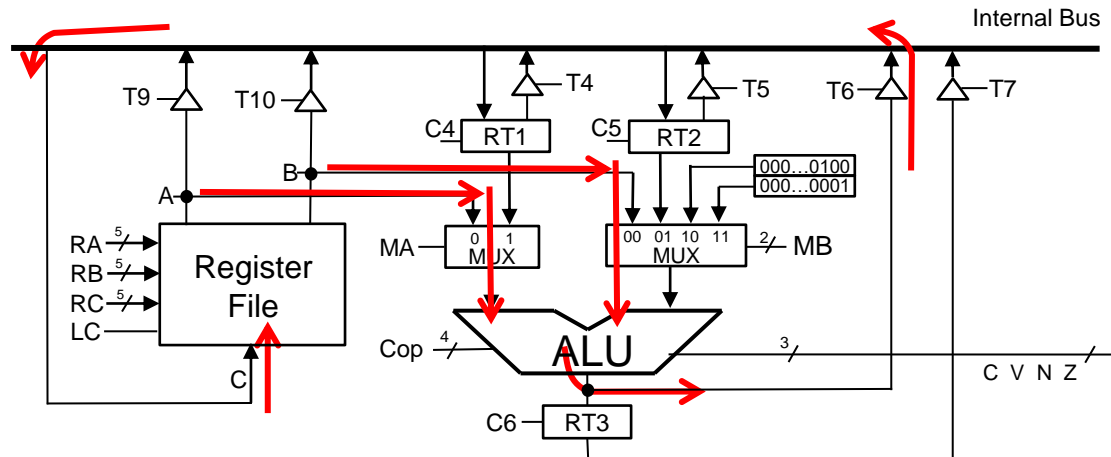
| Elem. Op. | Signals |
|---------------------|--------------------|
| $RT1 \leftarrow R1$ | RA=1, T9, C4 |
| $R1 \leftarrow R2$ | RA=2, T9, RC=1, LC |
| $R2 \leftarrow RT1$ | T4, RC=2, LC |

► 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

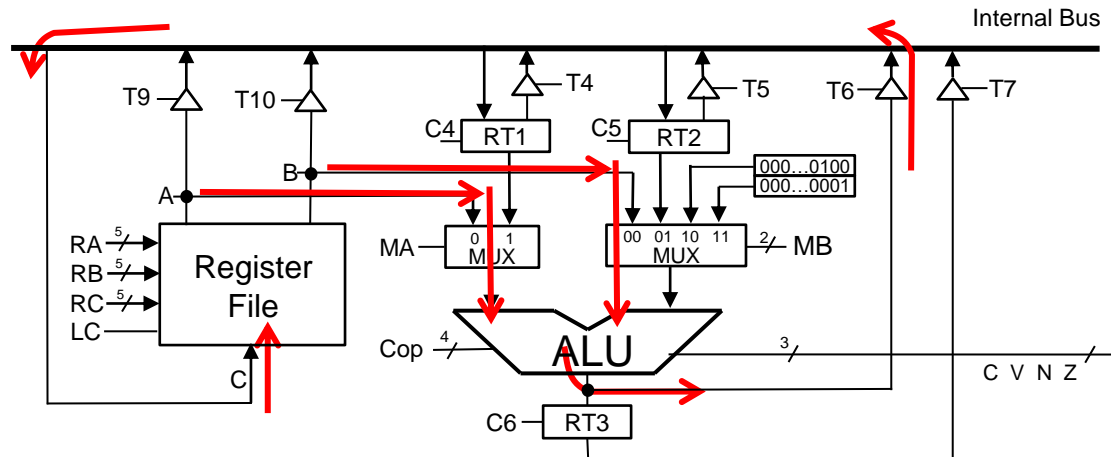
| Elem. Op. | Signals |
|---------------------|--------------------|
| $RT1 \leftarrow R1$ | RA=1, T9, C4 |
| $R1 \leftarrow R2$ | RA=2, T9, RC=1, LC |
| $R2 \leftarrow RT1$ | T4, RC=2, LC |

► SWAP R1, R2 without R_{tmp}

| Elem. Op. | Signals |
|------------------------------|---------------------------------|
| $R1 \leftarrow R1 \wedge R2$ | RA=1, RB=2, Cop=^, 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

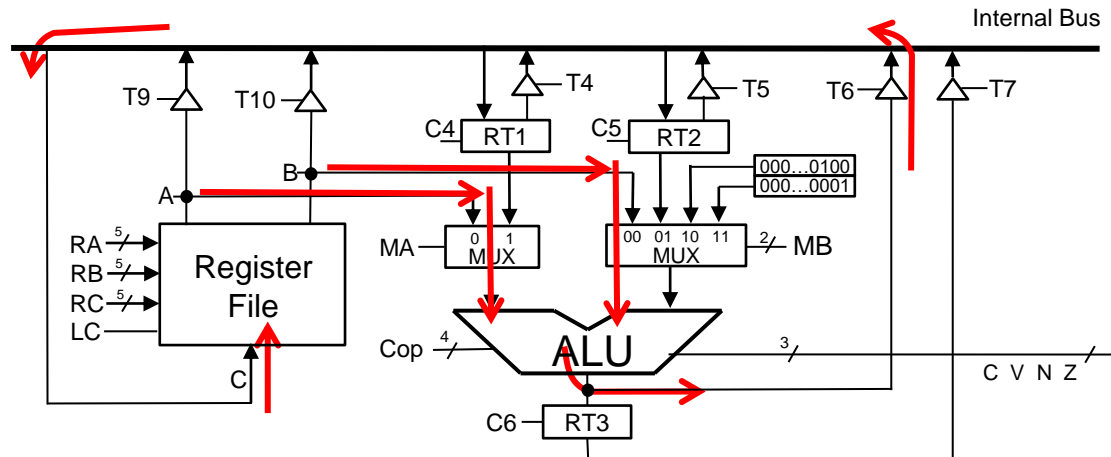
| Elem. Op. | Signals |
|---------------------|--------------------|
| $RT1 \leftarrow R1$ | RA=1, T9, C4 |
| $R1 \leftarrow R2$ | RA=2, T9, RC=1, LC |
| $R2 \leftarrow RT1$ | T4, RC=2, LC |

► 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 |

Example

elemental operations in ALU



► SWAP R1 R2

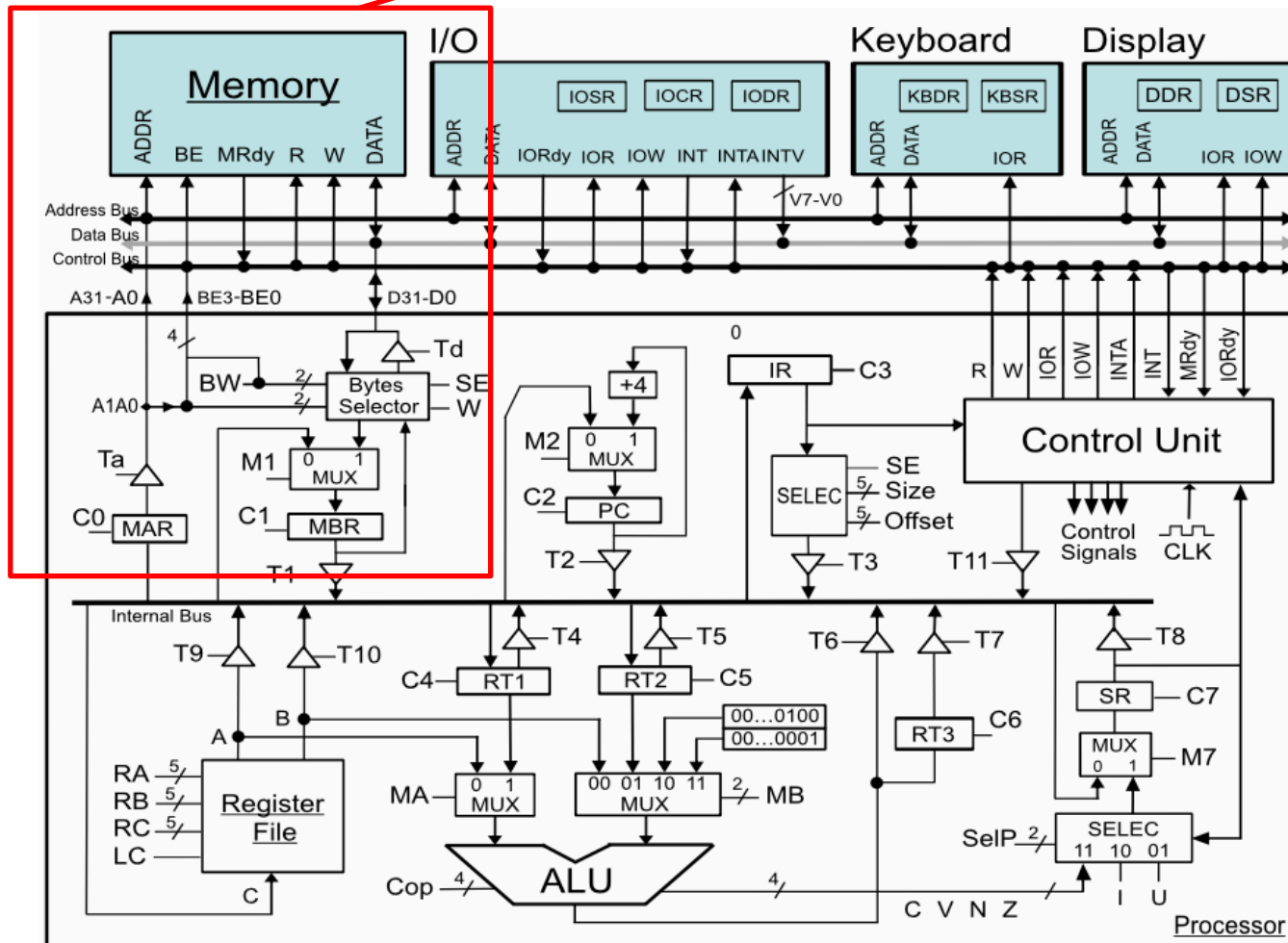
| Elem. Op. | Signals |
|---------------------|--------------------|
| $RT1 \leftarrow R1$ | RA=1, T9, C4 |
| $R1 \leftarrow R2$ | RA=2, T9, RC=1, LC |
| $R2 \leftarrow RT1$ | T4, RC=2, LC |

► SWAP R1, R2 without R_{tmp}

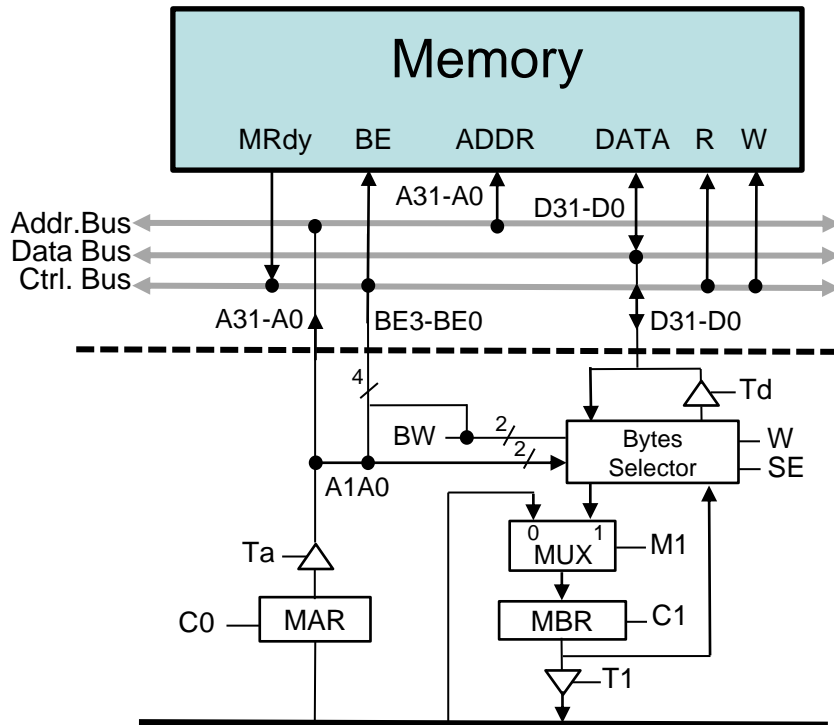
| Elem. Op. | Signals |
|------------------------------|---------------------------------|
| $R1 \leftarrow R1 \wedge R2$ | RA=1, RB=2, Cop=^, T6, RC=1, LC |
| $R2 \leftarrow R1 \wedge R2$ | RA=1, RB=2, Cop=^, T6, RC=2, LC |
| $R1 \leftarrow R1 \wedge R2$ | RA=1, RB=2, Cop=^, T6, RC=1, LC |

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
- ▶ $BE3-BE0 = A \mid A0 + BW$
 - ▶ Access size (byte, word, half word)
- ▶ ADDR – address
- ▶ DATA – data from/to memory
- ▶ 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
- ▶ TI - MBR output to internal bus
- ▶ MI - selection for MBR: memory or internal bus

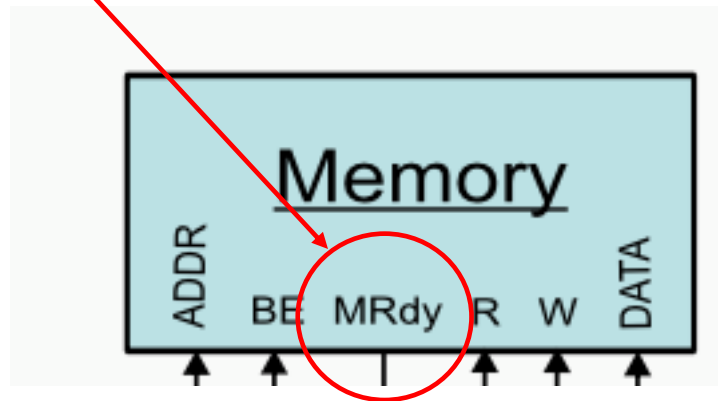
Memory access

- ▶ **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



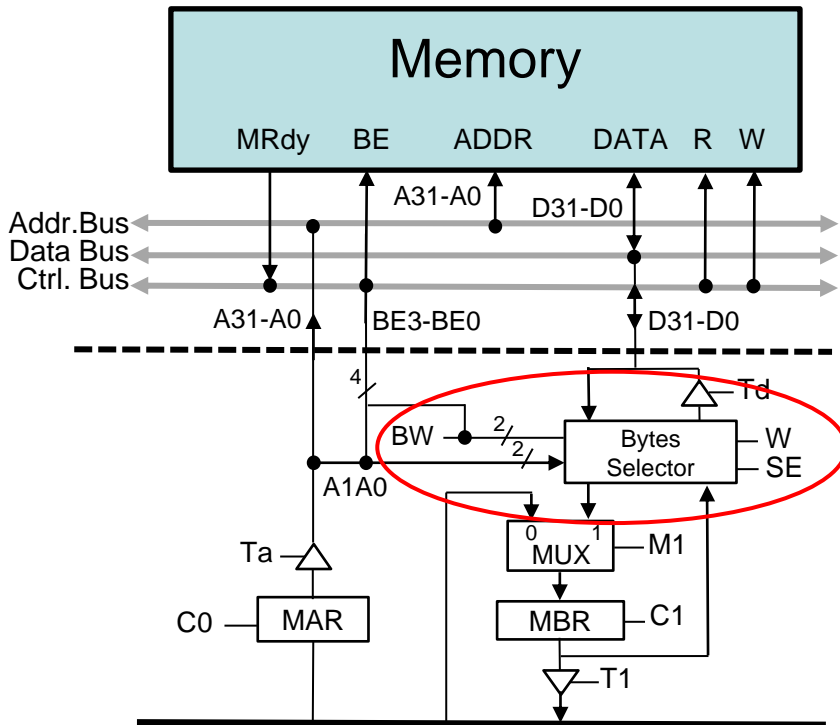
BE (Byte-Enable) signals for reading

| 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



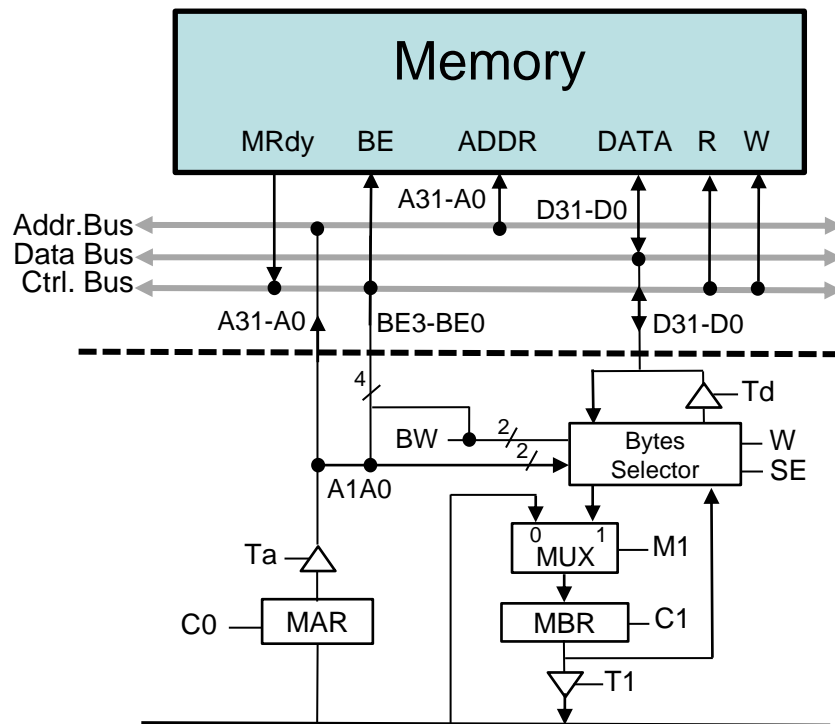
Nomenclature:

- 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 **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

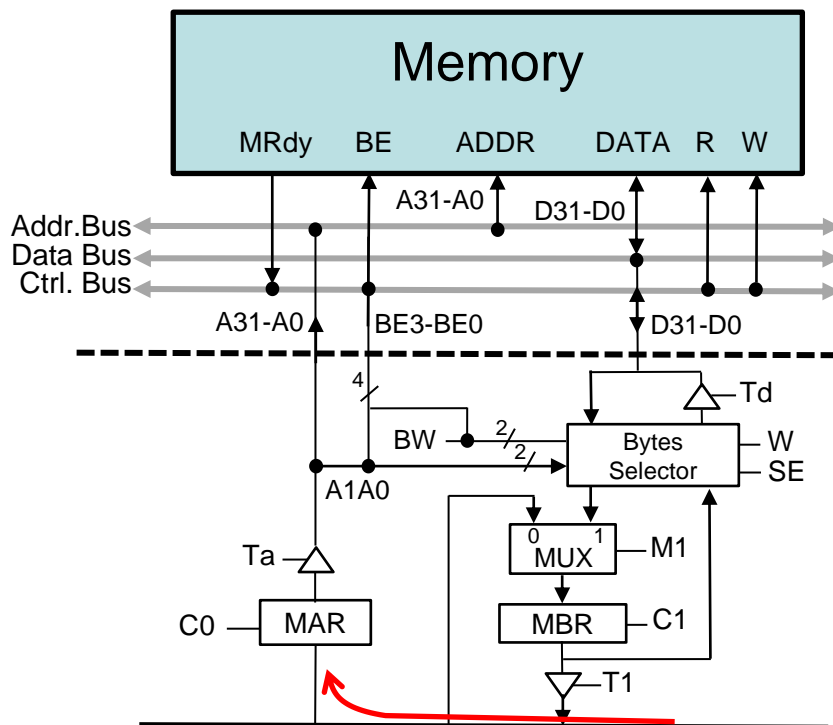
elemental operations in main memory

► Reading a word



access to 1 cycle synchronous main memory

► **Read**

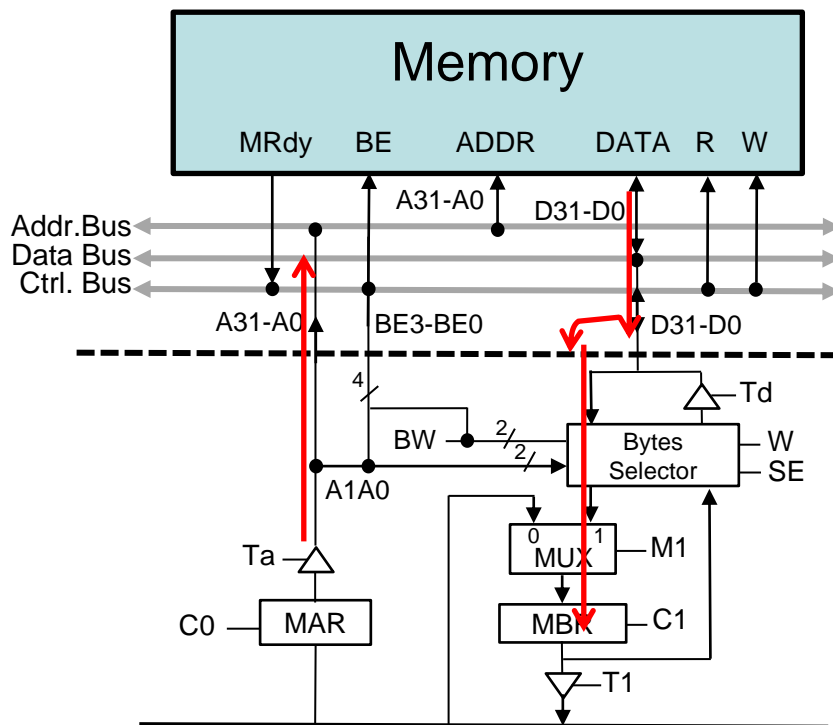


| Elem. Op. | Signals |
|----------------------------|---------|
| MAR \leftarrow <address> | ..., C0 |
| | |

Example

access to 1 cycle synchronous main memory

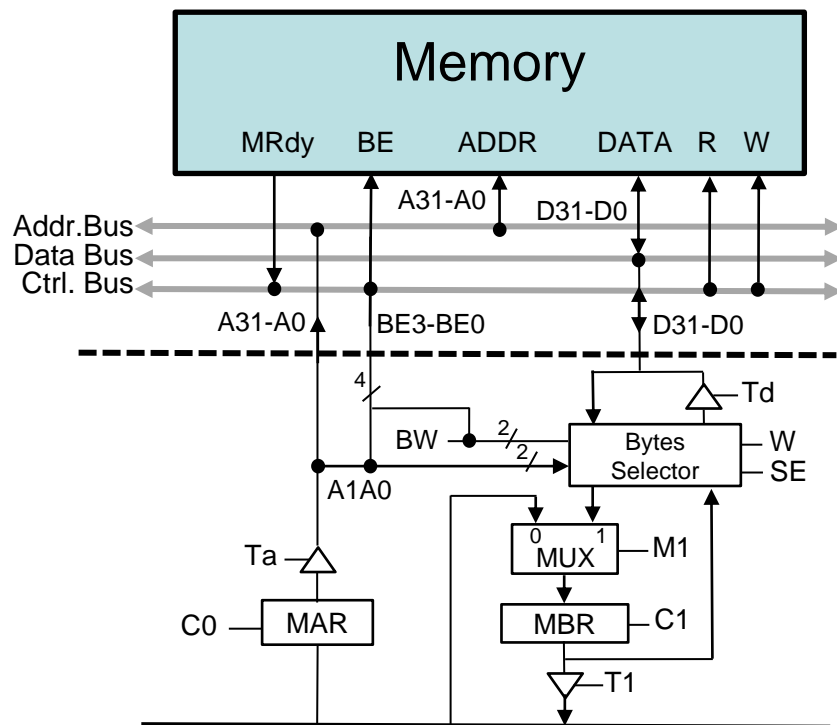
► Read



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

Example

access to 1 cycle synchronous main memory

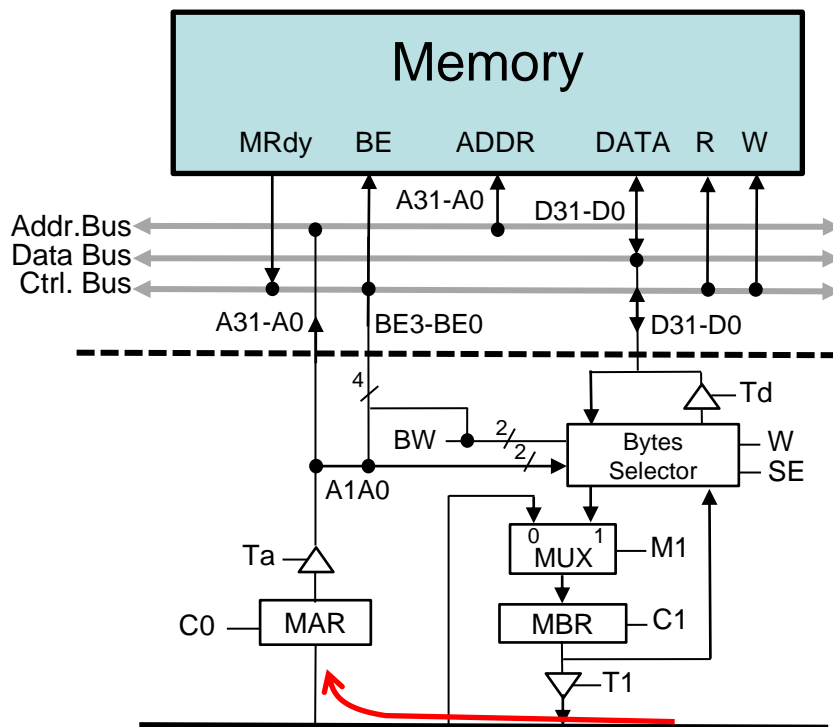


► Read

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

► Writing a word

access to 1 cycle synchronous main memory



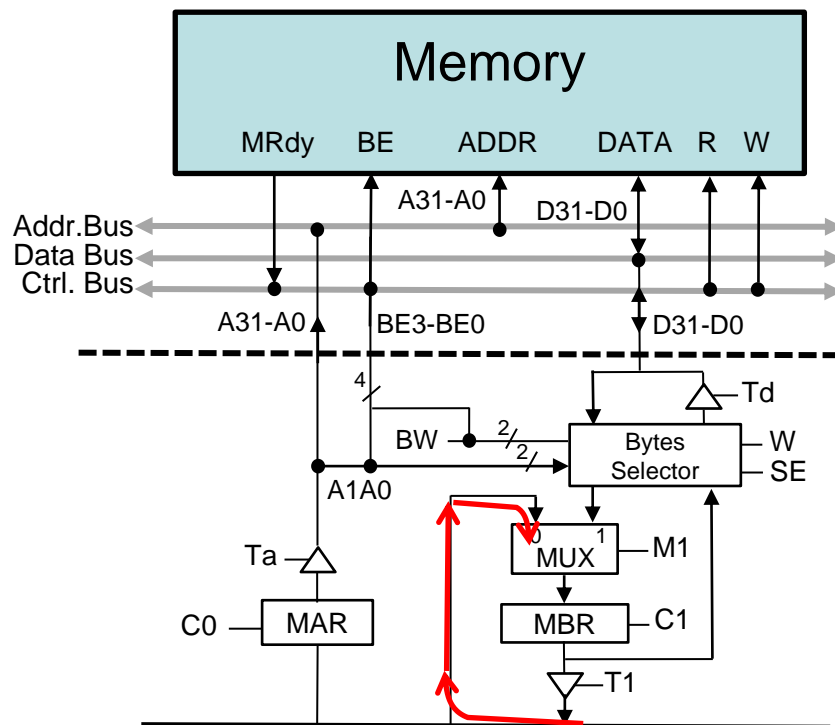
► **Read**

| 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 |
| | |
| | |

access to 1 cycle synchronous main memory



► **Read**

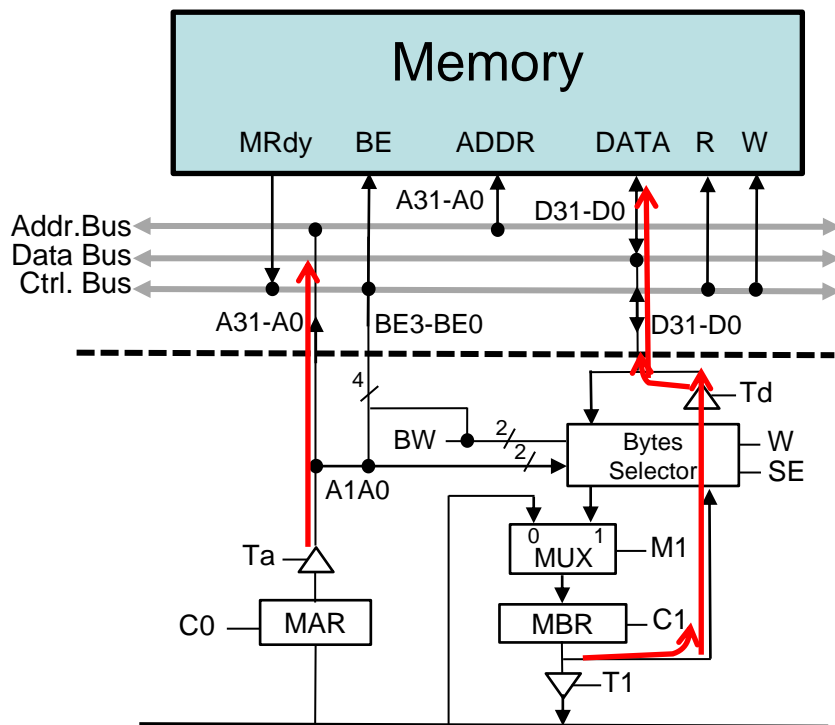
| Elem. Op. | Signals |
|----------------------------|-------------------------|
| MAR \leftarrow <address> | ..., C0 |
| MBR \leftarrow MP[MAR] | Ta, R, M1, C1, BW=11 |

► **Write**

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

Example

access to 1 cycle synchronous main memory



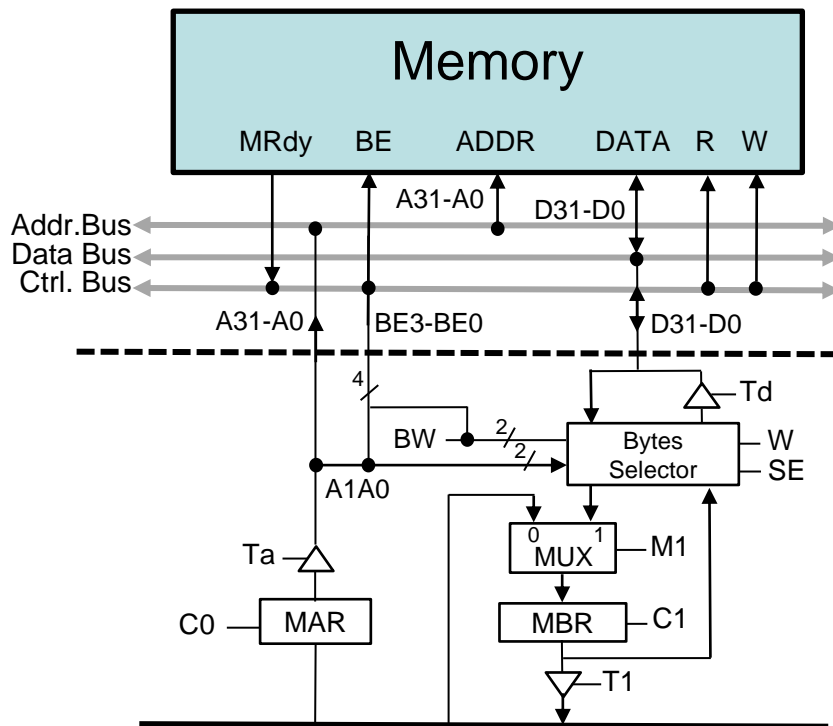
Read

| 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 |

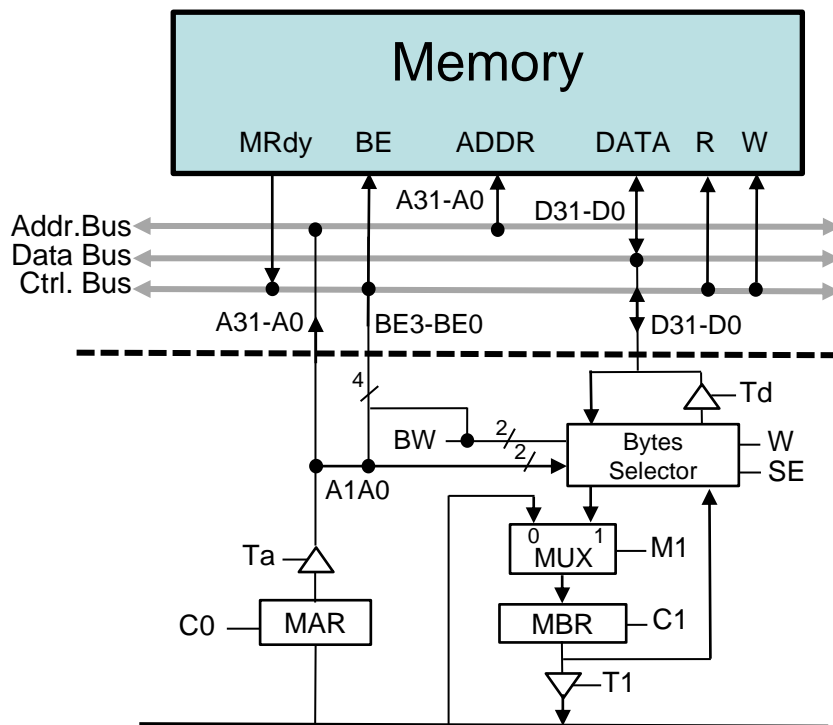
► Write

| 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

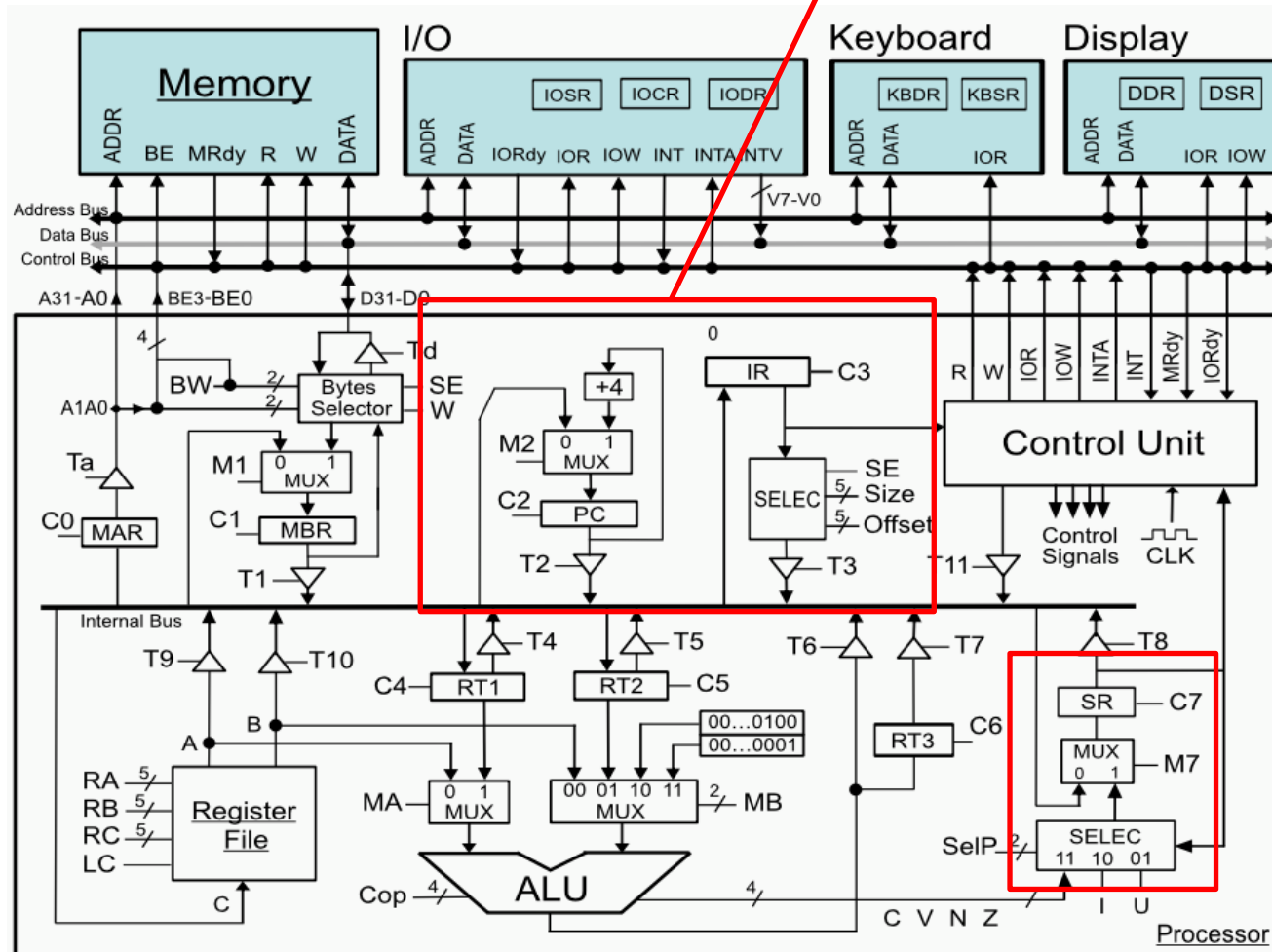


| Elem. Op. | Signals |
|--|-------------------------|
| MAR \leftarrow <address> | ..., C0 |
| Reading cycle | Ta, R, |
| Reading cycle, MBR \leftarrow MP[MAR] | Ta, R, M1, C1, BW=11 |

Structure of an elementary computer

SR, PC and IR

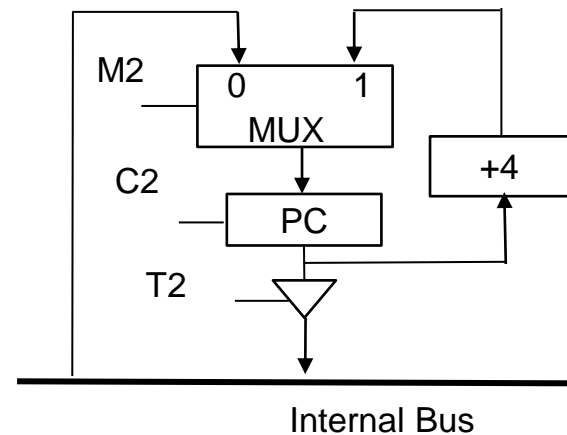
SR, PC and IR registers



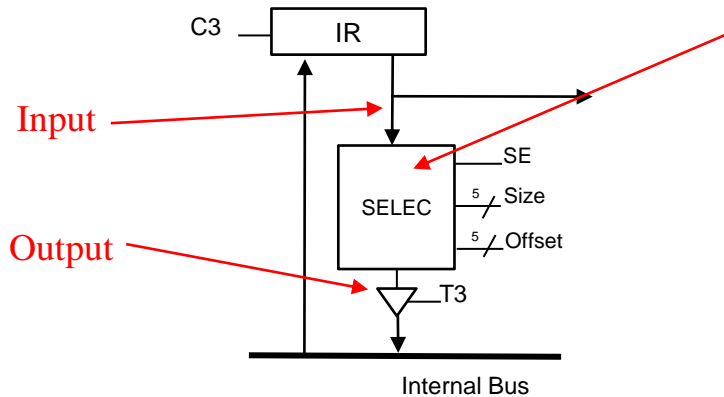
Program Counter

▶ Program Counter (PC):

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



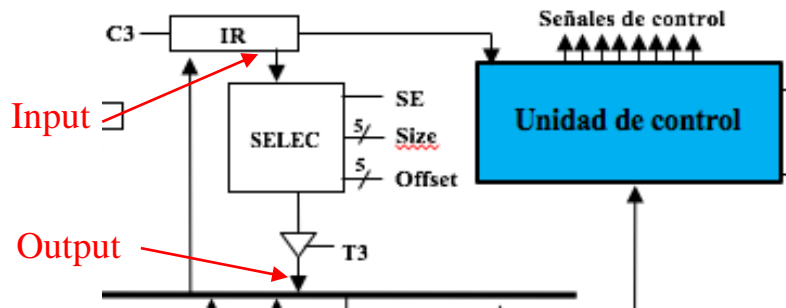
Instruction register



- ▶ 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

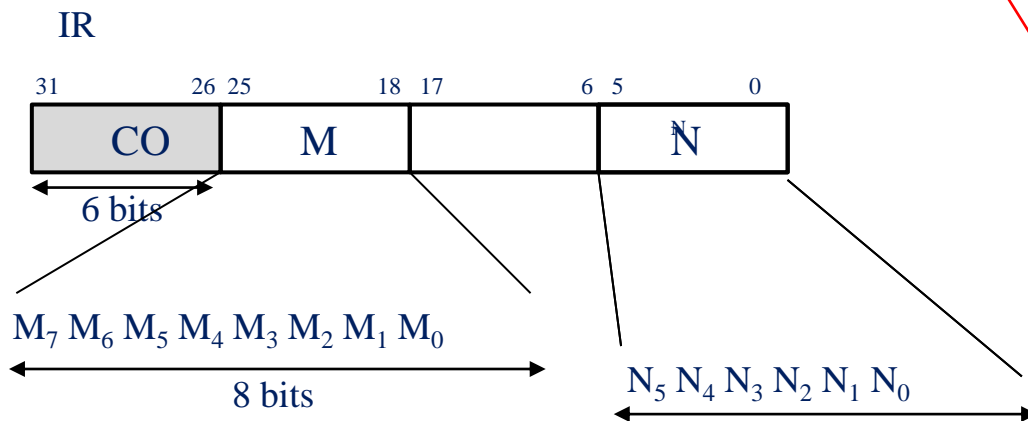
Selection without sign extension (SE = 0)



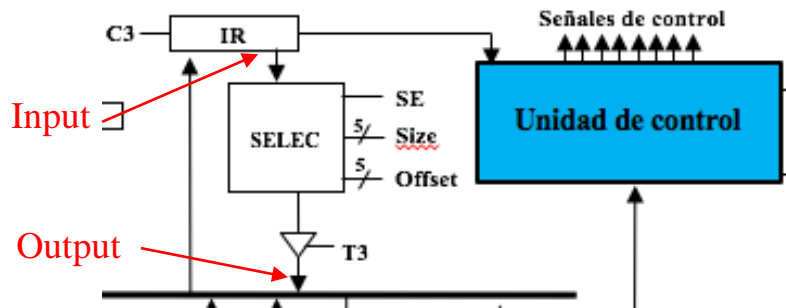
| Size | Offset | Output | | | |
|-------|--------|--------|-------|-------|---------------|
| 01000 | 10010 | 31 | 24 23 | 16 15 | 8 7 0 |
| | | 0 | 0 | 0 | $M_7..M_0$ |
| 00110 | 00000 | 31 | 24 23 | 16 15 | 8 7 0 |
| | | 0 | 0 | 0 | $00N_5...N_0$ |

Start bit
(less significant)

Size in bits



Selector circuit

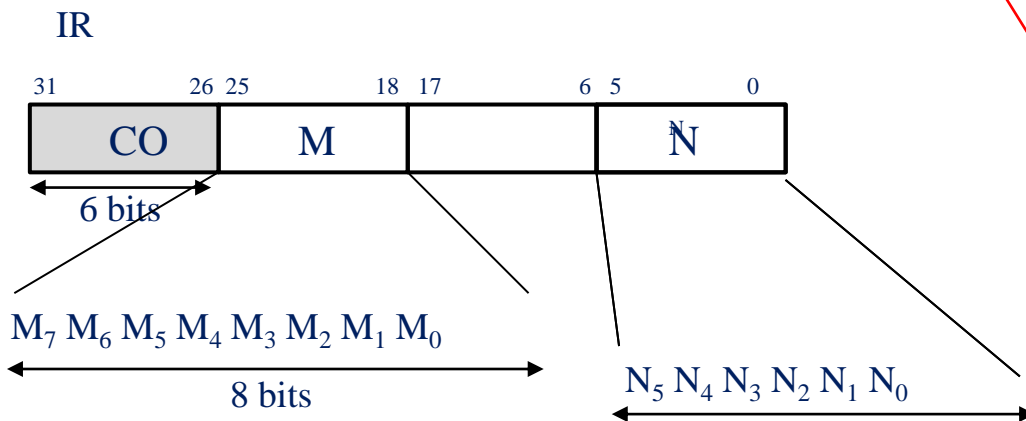


Selection with sign extension (SE = 1)

| 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 |
| | | $N_5..N_5$ $N_5..N_5$ $N_5..N_5$ $N_5N_5N_5..N_0$ | | | |

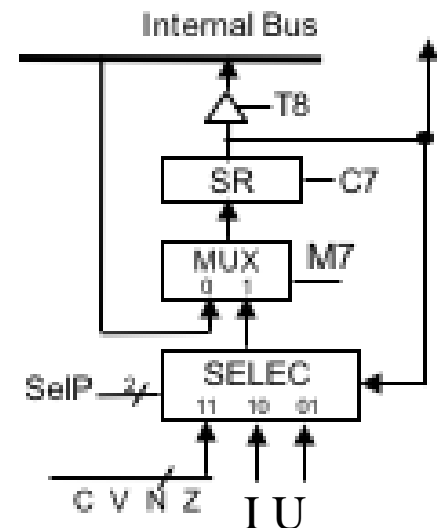
Start bit
(less significant)

Size in bits

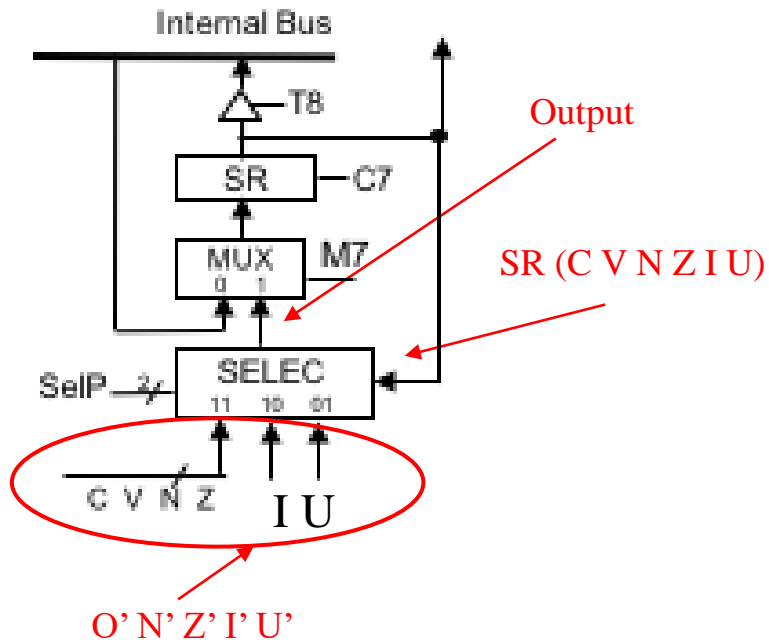


Status register

- ▶ 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 **interrupts** 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:

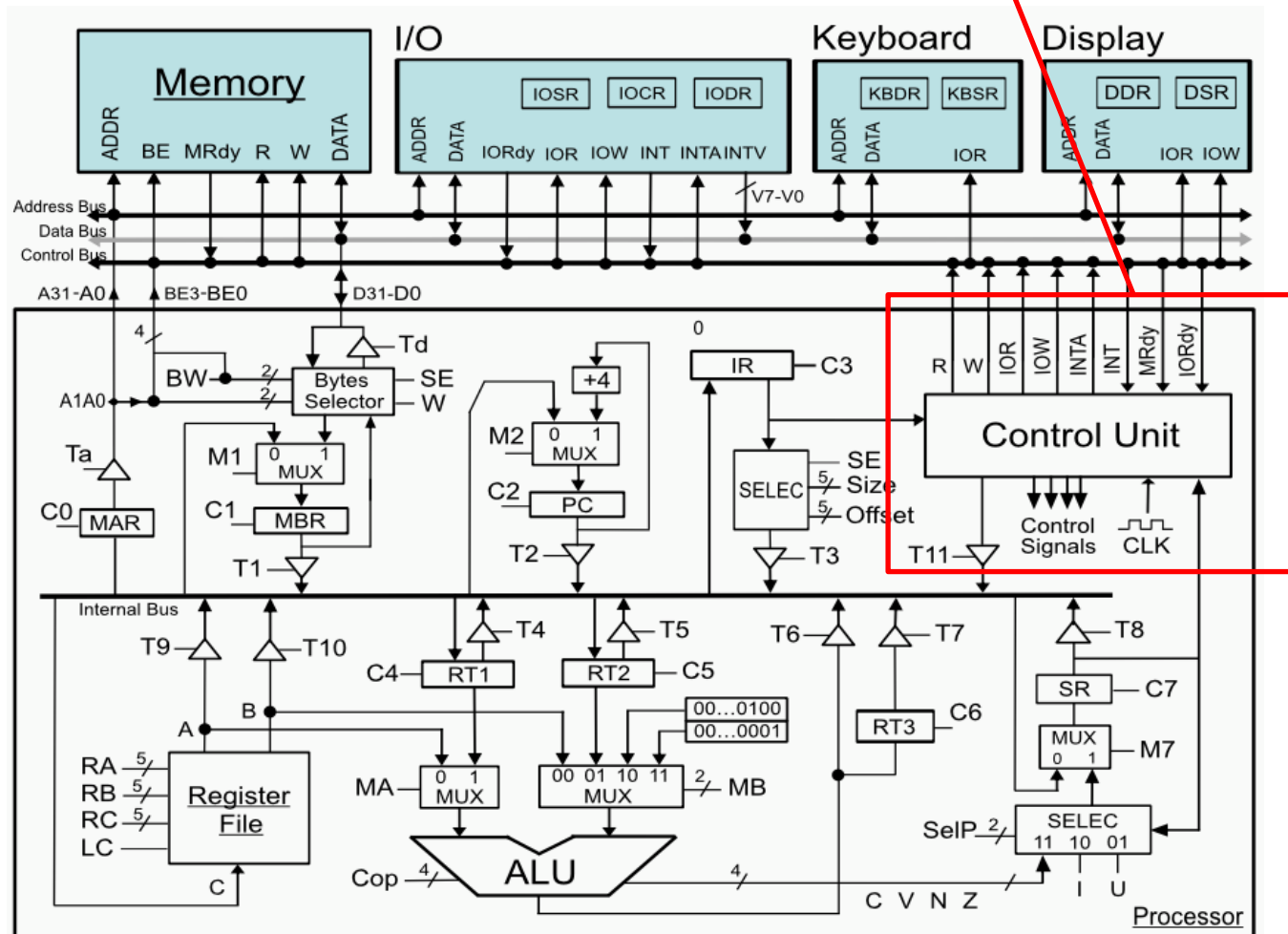
if (SelP1 = 1 AND SelP0 == 1)
Output = $C'V'N'Z'IU$

if (SelP1 == 1 AND SelP0 == 0)
Output = $CVNZI'U$

if (SelP1 == 0 AND SelP0 == 1)
Output = $CVNZIU'$

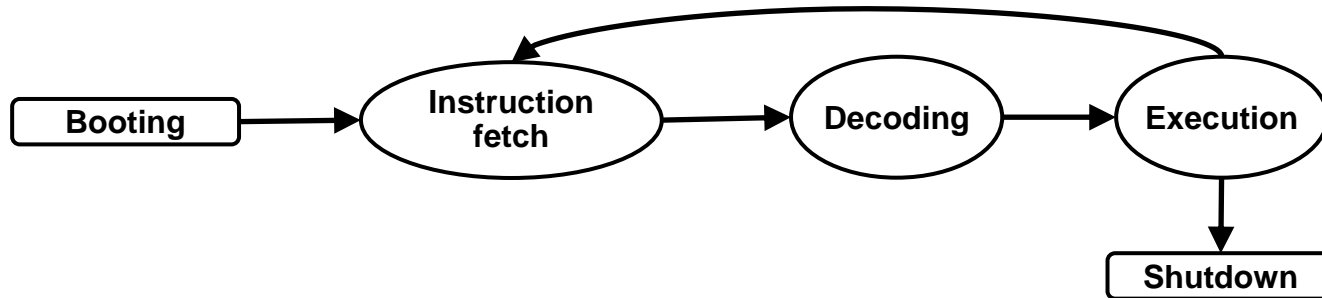
Structure of an elementary computer

Control Unit (C.U.)



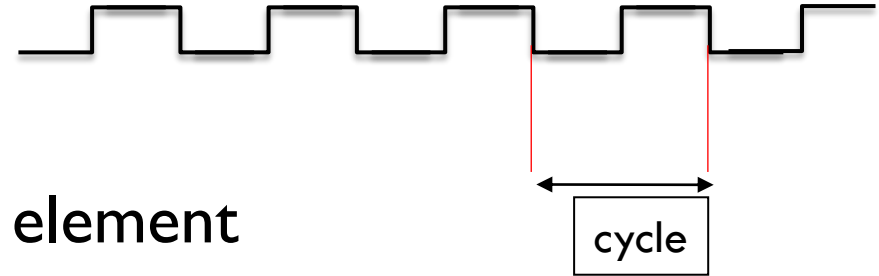
Control unit

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

Clock

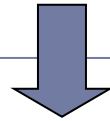


- ▶ A computer is a synchronous element
- ▶ Controls the operation
- ▶ The clock times the operations:
 - ▶ In a clock cycle one or more elementary operations are executed as long as there is no conflict
 - ▶ The necessary control signals are kept active during the cycle
- ▶ In the same cycle you can 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

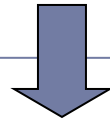
Instruction

mv R0 R1

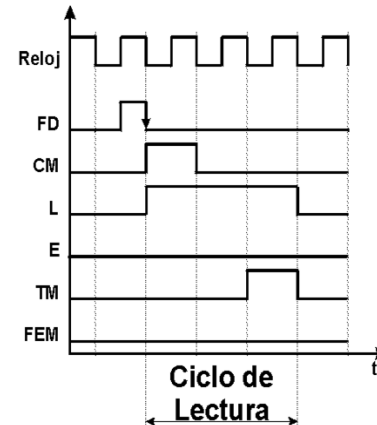


Sequence of **elementary operations**

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

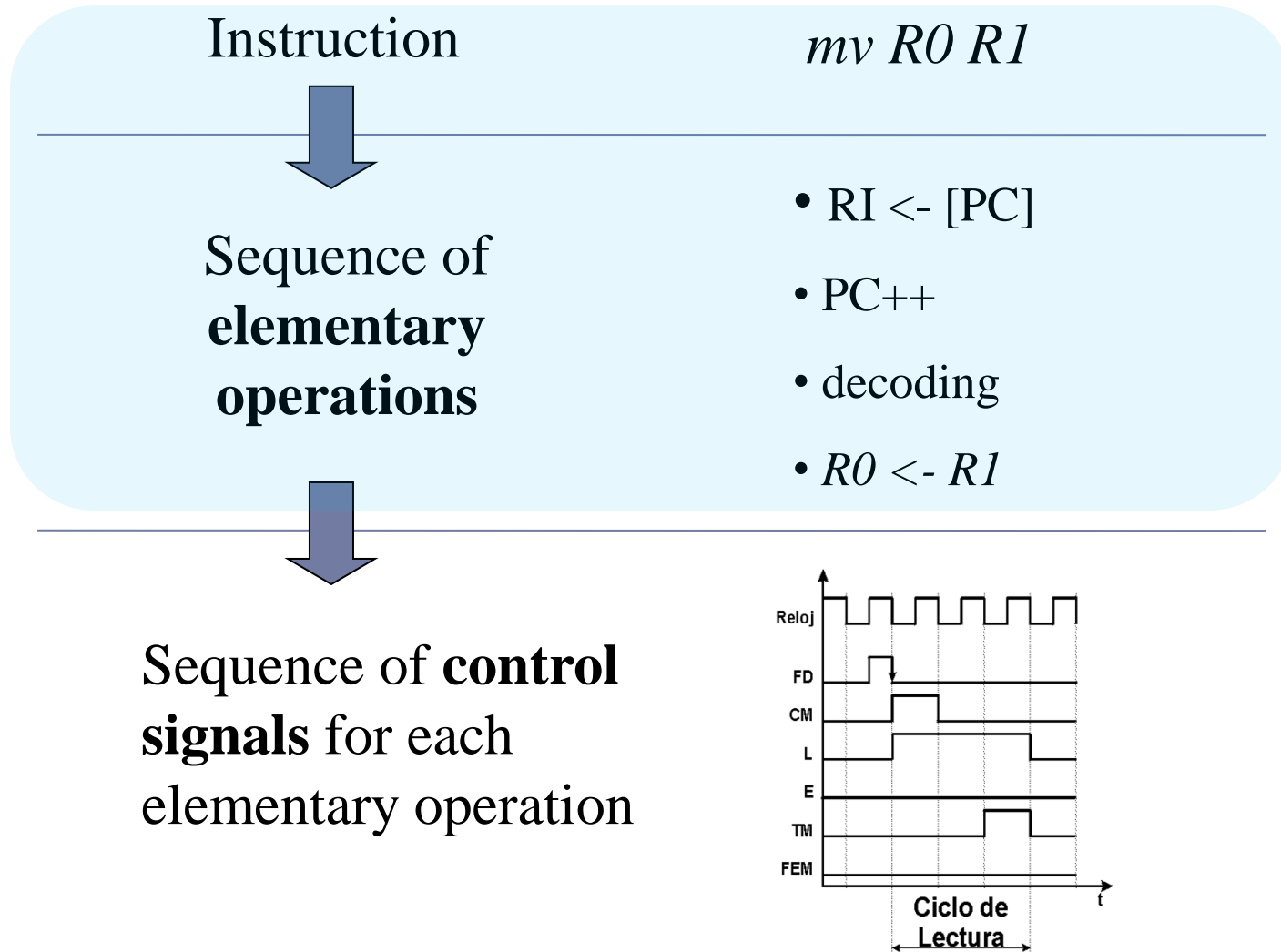


Sequence of **control signals** for each elementary operation



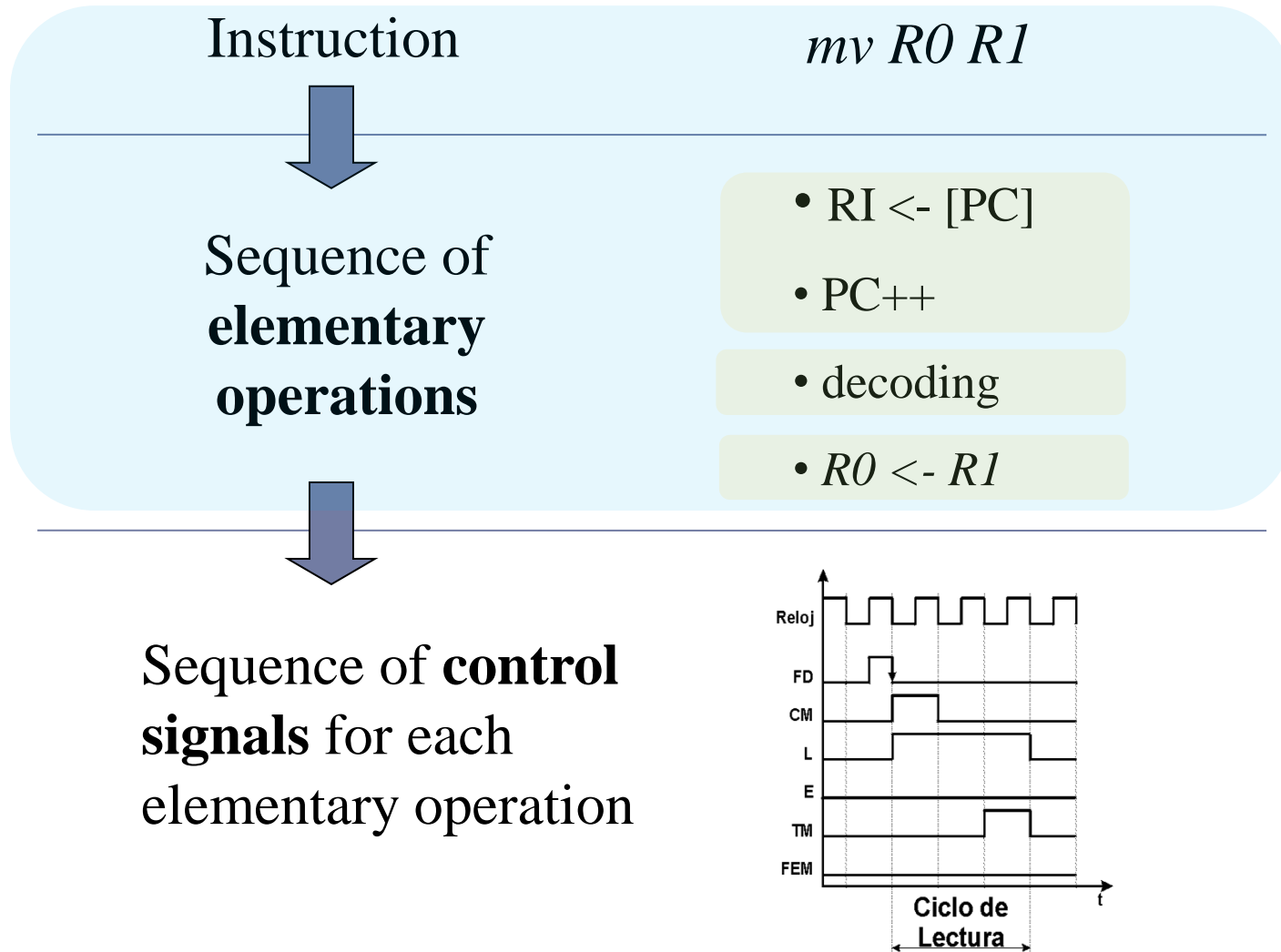
+ level of
hw. details

Description of the Control Unit activity




+ level of
hw. details

Description of the Control Unit activity



+ level of
hw. details

Fetch (Elemental Operations)

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

Possibility of simultaneous operations

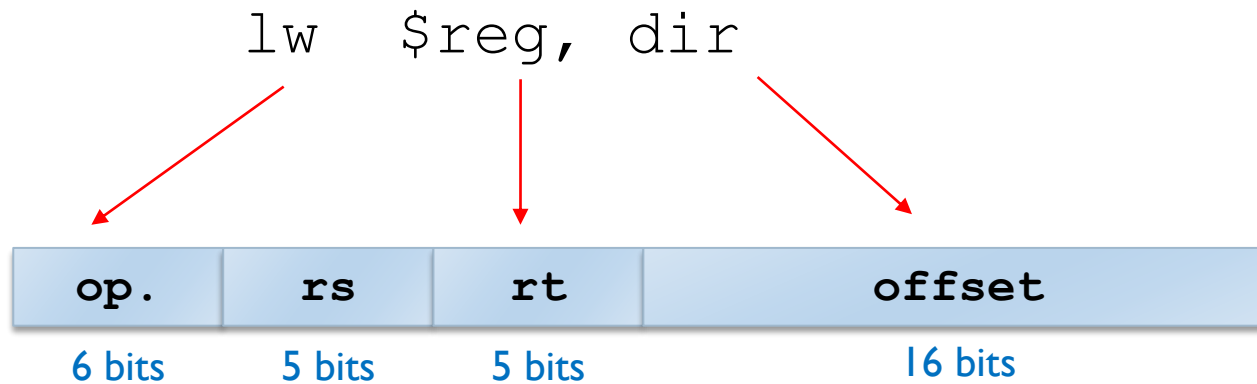
Fetch (Control Signals)

- 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

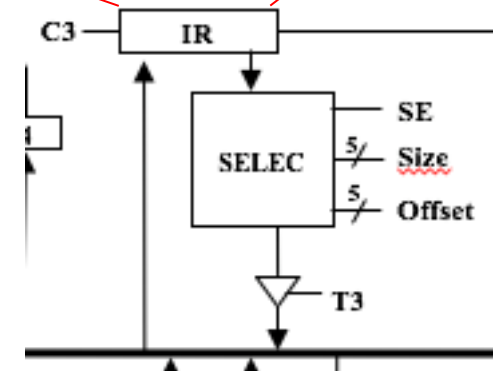
► `lw $reg, dir`



Execution of `lw $reg, dir`



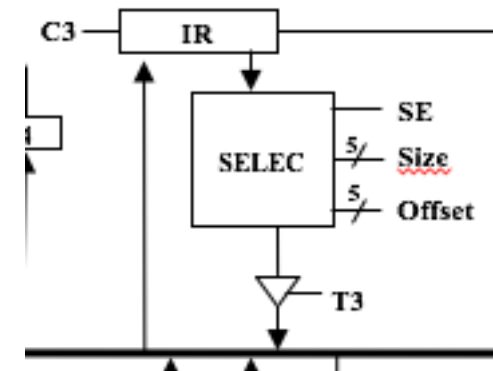
| Cycle | Elem. Op. | Control Signals |
|-----------|--|--------------------------------|
| C1 | $MAR \leftarrow PC$ | T2, C0 |
| C2 | $PC \leftarrow PC + 4,$ $MBR \leftarrow MP$ | C2, M2 Ta, R, CI, MI, BW=11 |
| C3 | $IR \leftarrow MBR$ | T1, C3 |
| C4 | | |
| C5 | | |
| C6 | | |
| C7 | | |



Execution of `lw $reg, dir`



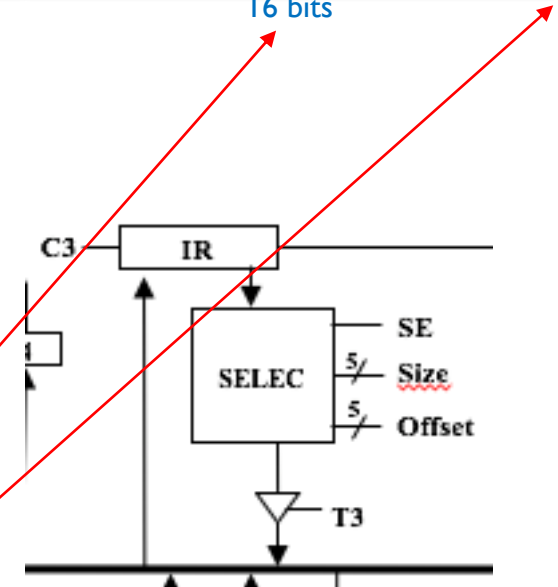
| Cycle | Elem. Op. | Control Signals |
|-------|--|--------------------------------|
| C1 | $MAR \leftarrow PC$ | T2, C0 |
| C2 | $PC \leftarrow PC + 4,$ $MBR \leftarrow MP$ | C2, M2 Ta, R, CI, MI, BW=II |
| C3 | $IR \leftarrow MBR$ | T1, C3 |
| C4 | Decoding | A0, B=0, C=0 |
| C5 | | |
| C6 | | |
|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



Execution of `lw $reg, dir`



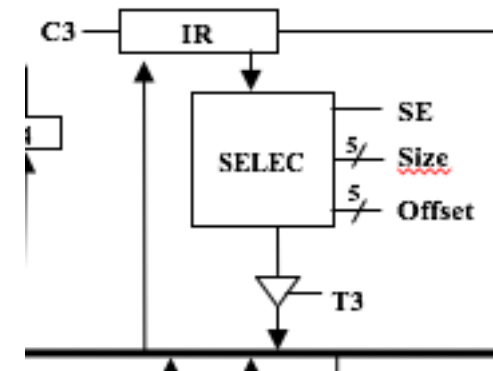
| Cycle | Elem. Op. | Control Signals |
|-------|--|--|
| C1 | $MAR \leftarrow PC$ | T2, C0 |
| C2 | $PC \leftarrow PC + 4,$ $MBR \leftarrow MP$ | C2, M2 Ta, R, CI, MI, BW=11 |
| C3 | $IR \leftarrow MBR$ | T1, C3 |
| C4 | Decoding | A0, B=0, C=0 |
| C5 | $MAR \leftarrow RI(dir)$ | C0, T3, Size = 10000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



Execution of `lw $reg, dir`



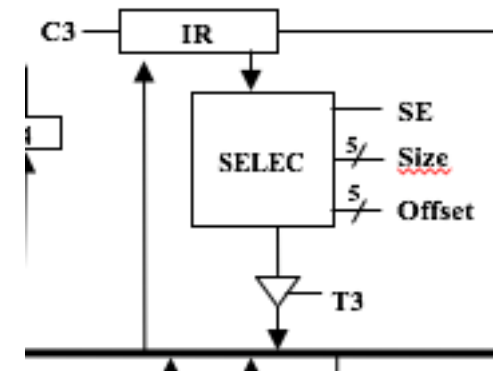
| Cycle | Elem. Op. | Control Signals |
|-------|--|--|
| C1 | $MAR \leftarrow PC$ | T2, C0 |
| C2 | $PC \leftarrow PC + 4,$ $MBR \leftarrow MP$ | C2, M2 Ta, R, CI, MI, BW=11 |
| C3 | $IR \leftarrow MBR$ | T1, C3 |
| C4 | Decoding | A0, B=0, C=0 |
| C5 | $MAR \leftarrow RI(dir)$ | C0, T3, Size = 10000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



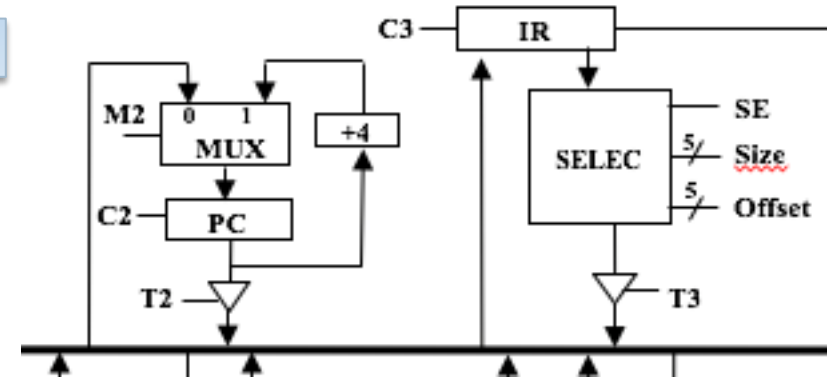
Execution of `lw $reg, dir`



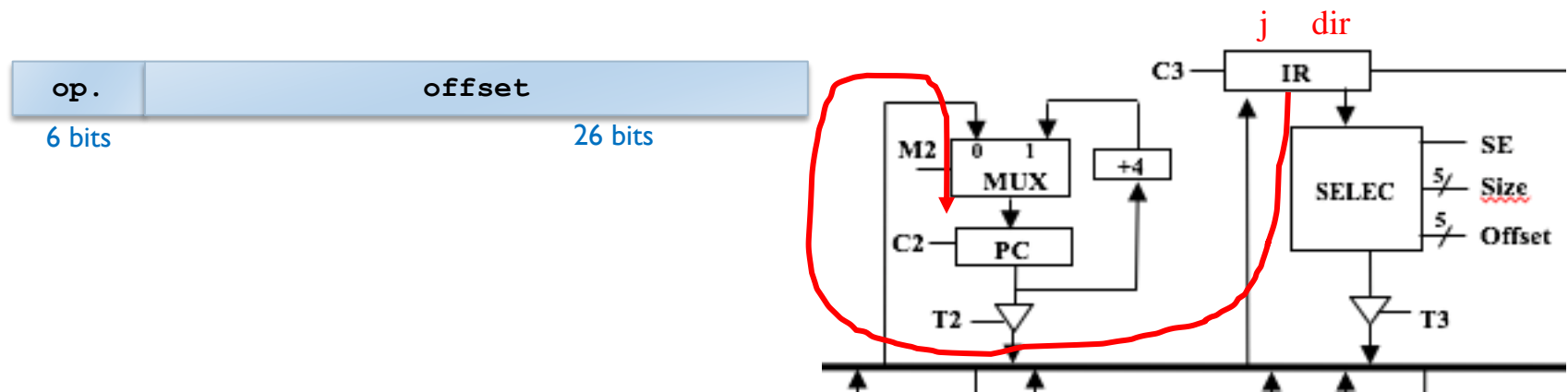
| Cycle | Elem. Op. | Control Signals |
|-----------|--|--|
| C1 | $MAR \leftarrow PC$ | T2, C0 |
| C2 | $PC \leftarrow PC + 4,$ $MBR \leftarrow MP$ | C2, M2 Ta, R, CI, MI, BW=II |
| C3 | $IR \leftarrow MBR$ | T1, C3 |
| C4 | Decoding | A0, B=0, C=0 |
| C5 | $MAR \leftarrow RI(dir)$ | C0, T3, Size = I0000 Offset = 00000 |
| C6 | $MBR \leftarrow MP$ | Ta, R, CI, MI, BW=II |
| 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 | A0, B=0, C=0 |
| 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

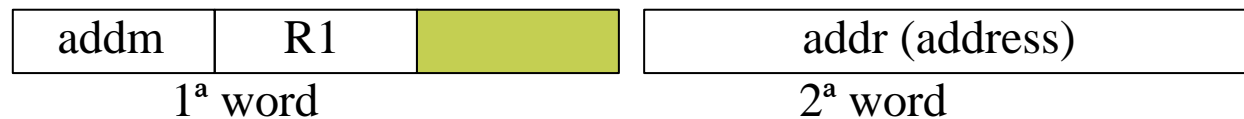
| Cycle | Elem. Op. |
|-------|--|
| C1 | $MAR \leftarrow PC$ |
| C2 | $PC \leftarrow PC + 4,$ $MBR \leftarrow MP$ |
| C3 | $IR \leftarrow MBR$ |
| C4 | Decoding |
| C5 | $\$reg + \0 |
| C6 | Si $SR.Z == 0$ jump to fetch |
| C7 | $RT2 \leftarrow PC$ |
| C8 | $RT1 \leftarrow IR(\text{desplaz})$ |
| C9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

ARCOS Group

uc3m | Universidad **Carlos III** de Madrid

Lesson 4 (I)

The processor

Computer Structure
Bachelor in Computer Science and Engineering

