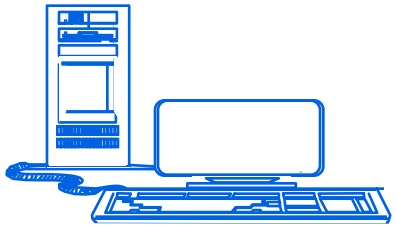
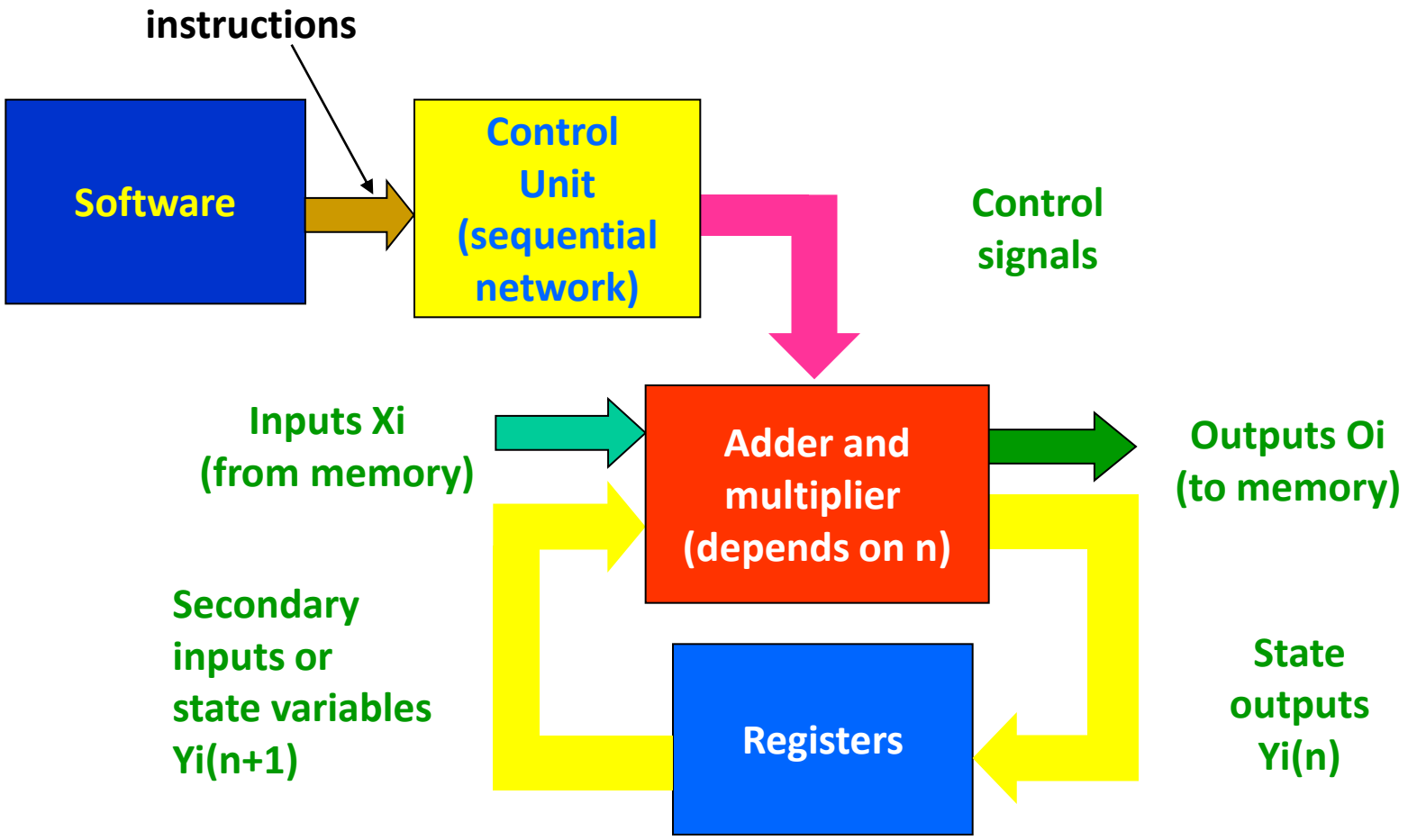
Computer Architectures

DLX ISA: sequential architecture

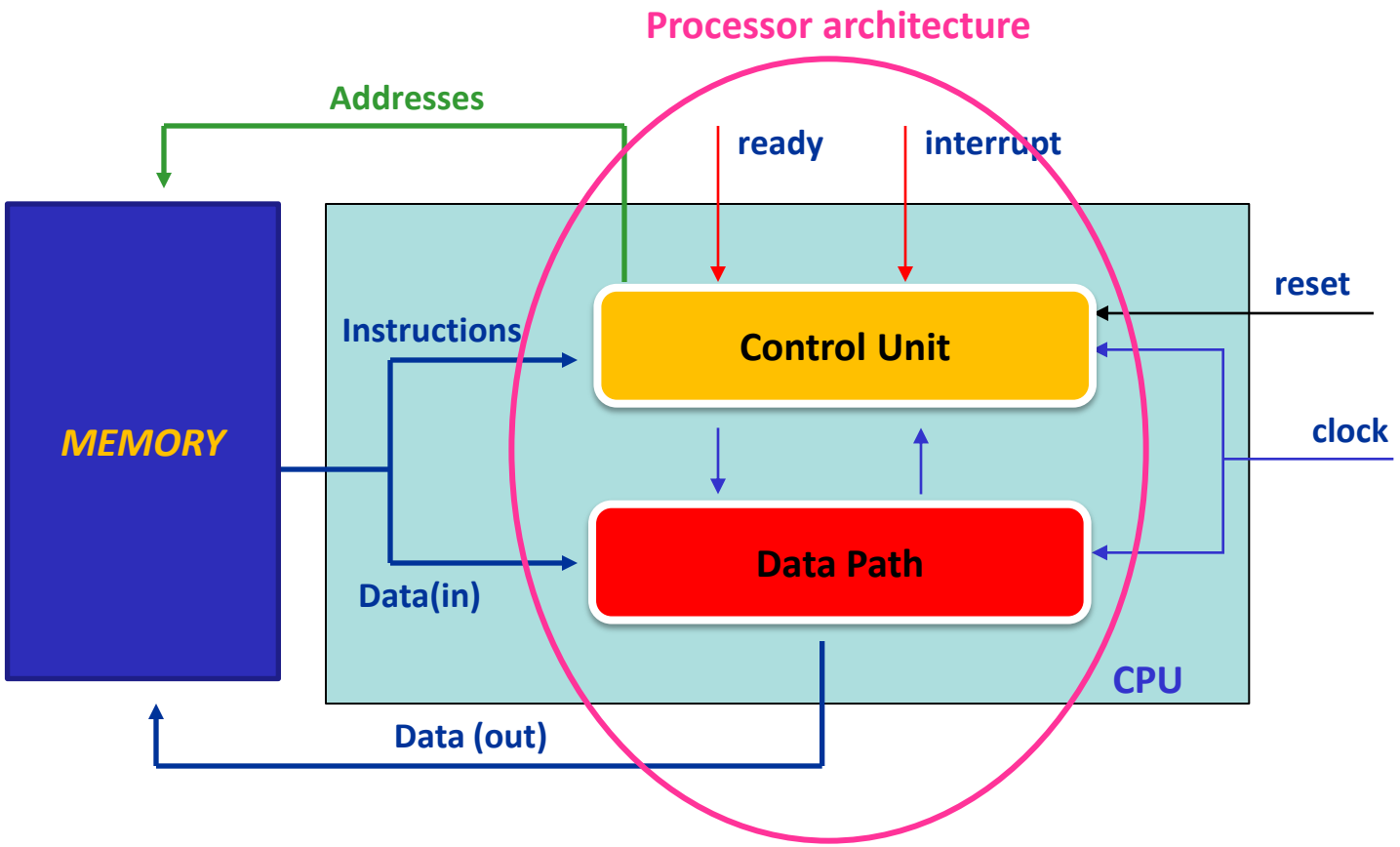


Computer Architecture



Datapath and Control Unit

- The CPU structure is a syncronous logic circuit processing data which can be divided into two blocks: Control Unit and Datapath.
- The CPU requires an external memory where software and data are stored.



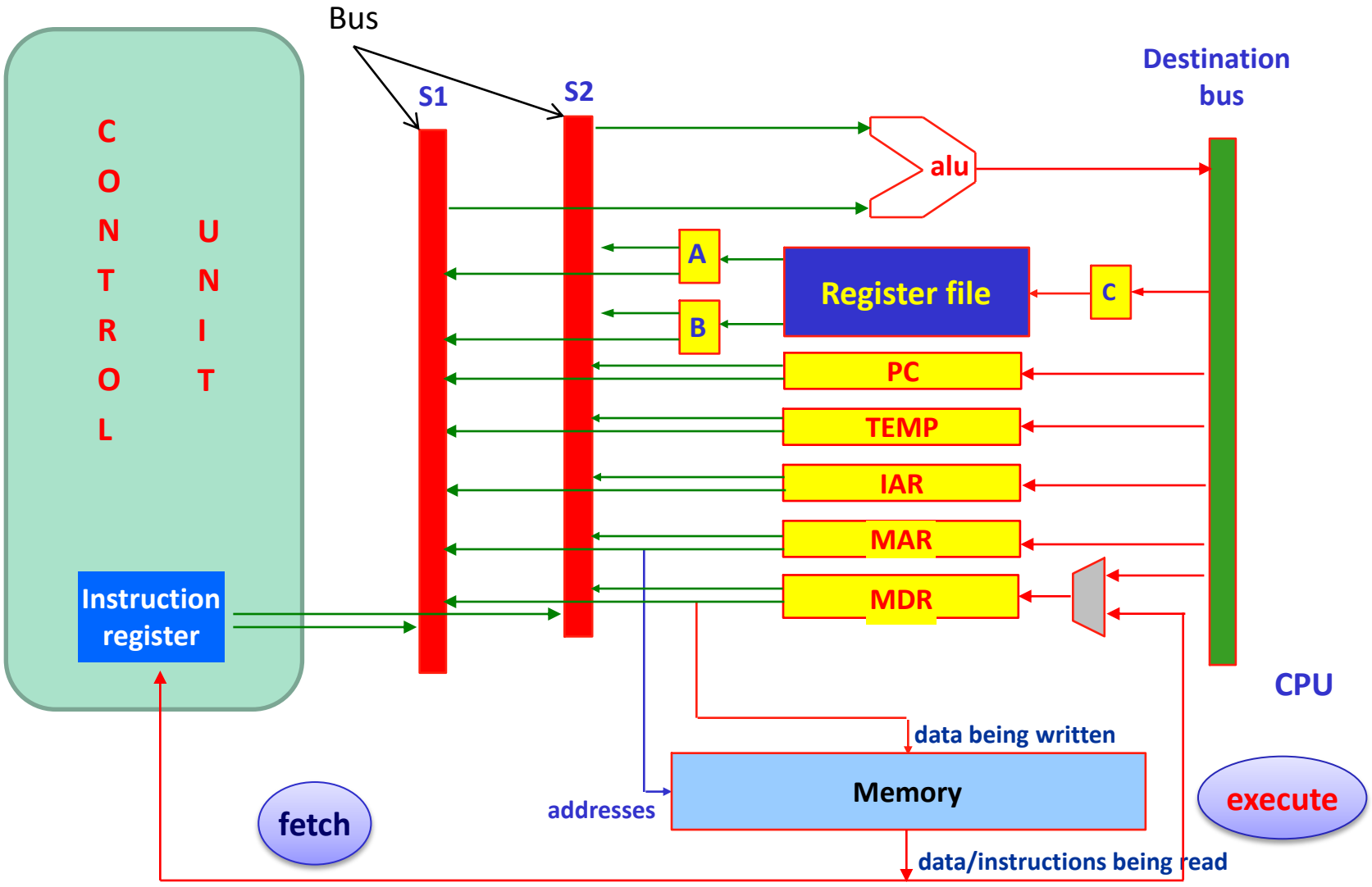
Datapath and Control Unit

- Datapath: includes all processing units and registers required to execute CPU instructions. Every instruction belonging to the Instruction Set is executed as a succession of *elementary operations*, called *micro-operations*.

- Micro-operation: operation executed in the DATAPATH *within a single clock cycle* (examples: data transfer between registers, ALU operations)

- Control Unit: it's a **Synchronous Sequential Circuit** (**SSC**) that, at every clock cycle, issues a specific set of control signals to the DATAPATH with the aim of specifying the execution of a single *micro-operation*.

DLX structure (sequential execution)

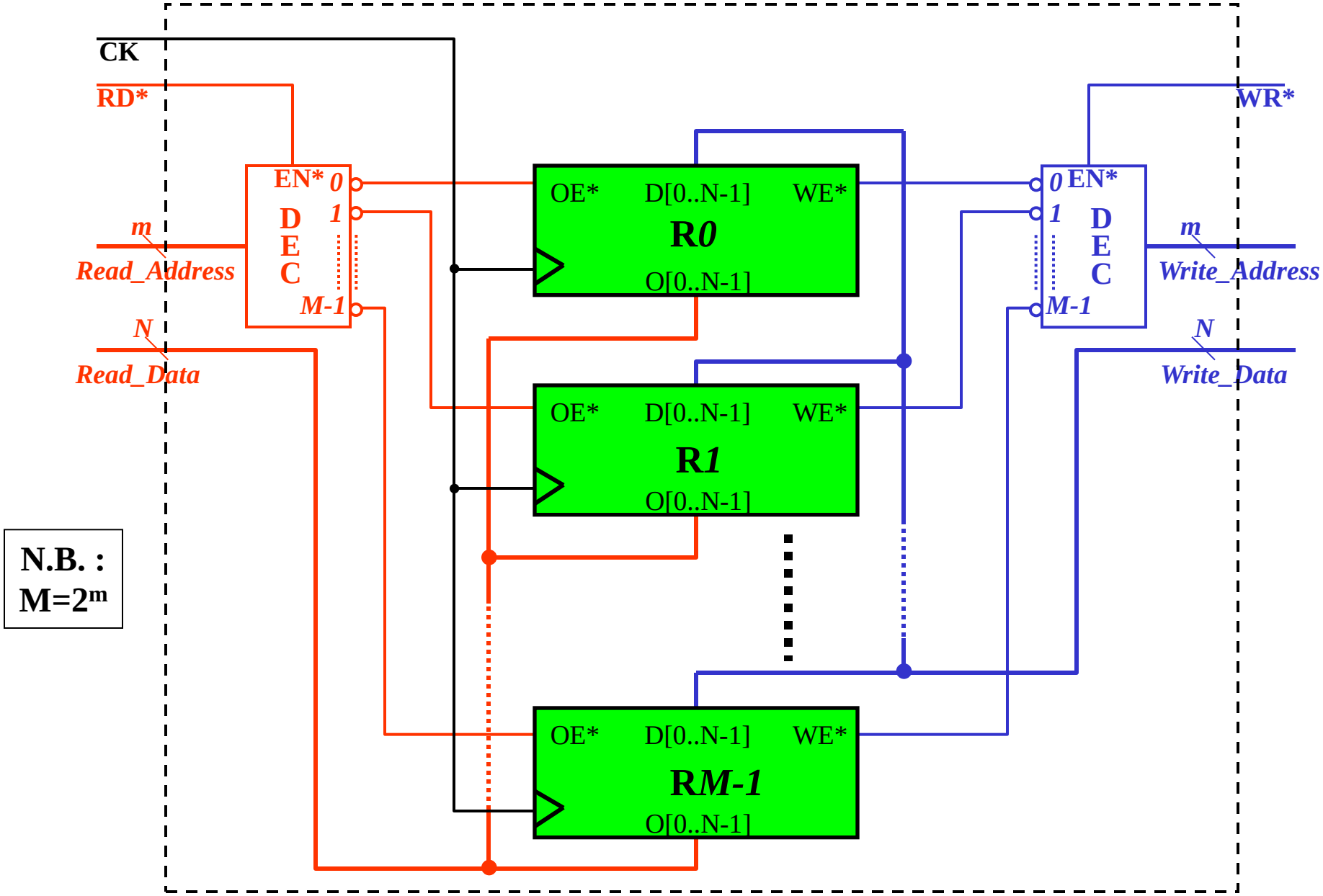


Architecture parallelism: 32 bit

(bus, alu and registers have 32-bit parallelism)

control signals are not shown!

Register File (1 read-port, 1 write-port)



DLX registers (all 32 bit)

Exception made for the Register File, these registers are unknown to the programmer!

- **Register file: 32 General Purpose Registers R0….R31 with R0=0**
- **IAR: Interrupt Address Register – stores the Return Address in case of interrupt**
- **PC: Program Counter**
- **MAR: Memory Address Register – contains the address of the data to be written to/read from memory**
- **IR: Instruction Register – contains the instruction currently being executed**
- **TEMP: Temporary Register – stores temporary results**
- **MDR: Memory Data Register – Temporary transit register for data from/to memory**
- **A and B – Output registers from the Register File**
- **C – buffer for RF input**

ALU operations

Dest (outputs) – 4 control bits

- S1 + S2
- S1 – S2
- S1 and S2
- S1 or S2
- S1 exor S2
- Left-shift S1 of S2 positions
- Right-shift S1 of S2 positions
- Arithmetic Right-shfit S1 of S2 positions
- S1
- S2
- 0
- 1

Output Flags

- Zero
- Negative sign

The ALU is a PURELY combinatorial circuit

Data transfer on the datapath

- The S1 ed S2 buses are multiplexed (tri-state) with a 32-bit parallelism.
- Registers sample on the positive edge of the clock signal. They have usually two output gates O1 e O2 for the two buses (or for registers A and B) and three control inputs:
  - one Write Enable (WE*) input and one Output Enable input for each output gate (one for S1 and one for S2 - OE1* e OE2*).
- To evaluate the maximum working frequency of the datapath, the following delays must be taken into account:
  - $T_C(max)$: max delay between the clock positive edge and the moment when the control signals generated by the control unit are valid;
  - $T_{OE}(max)$: max delay between the activation of the OE signal and the moment when the register data are available on the bus;
  - $T_{ALU}(max)$: max ALU delay;
  - $T_{SU}(min)$: Minimum *set-up* time for the registers (minimum requirement for correct sampling performed by the registers).
- The maximum working frequency/minimum clock period of the data path can be computed as follows:

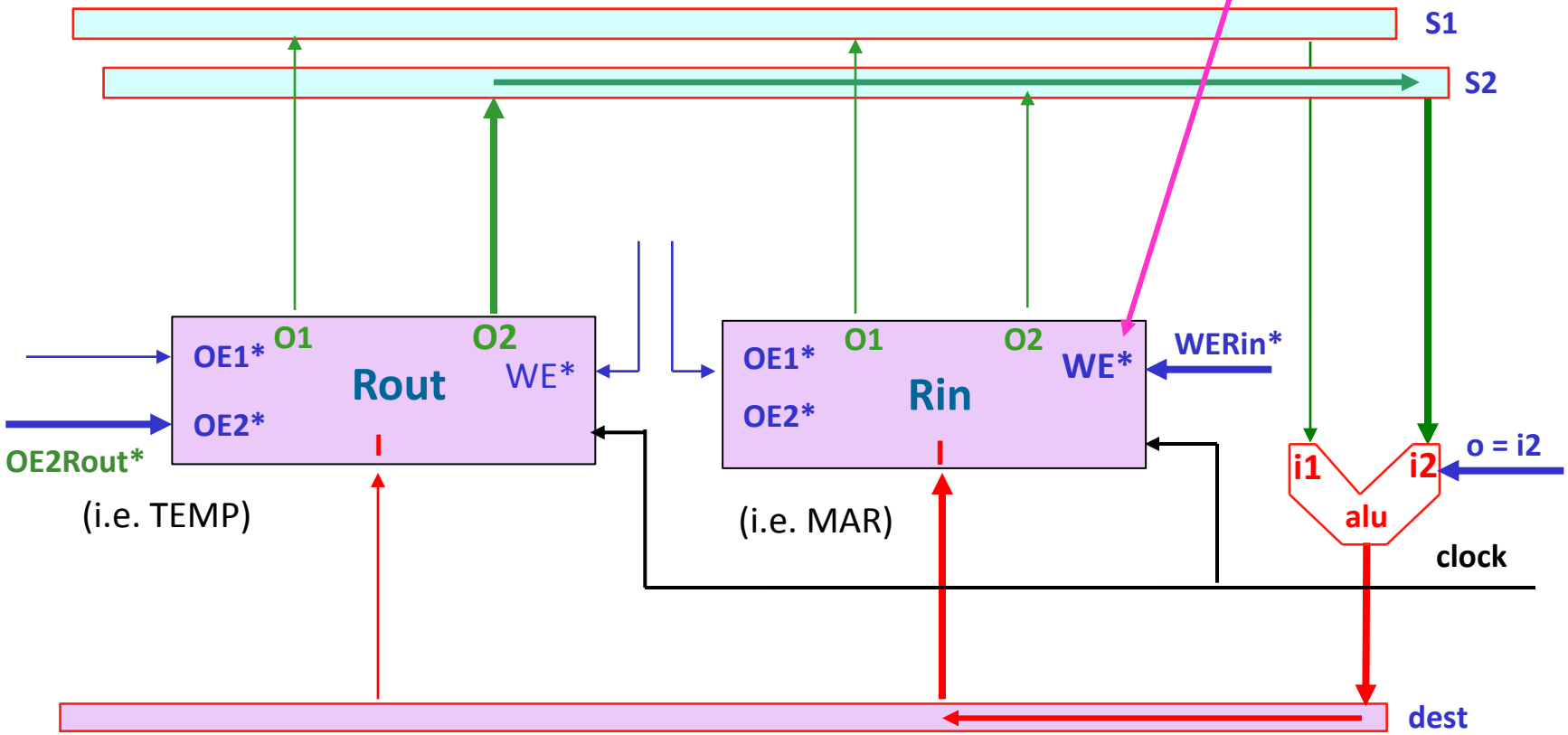
$$T_{CK} > T_C(max) + T_{OE}(max) + T_{ALU}(max) + T_{SU}(min)$$

$$f_{CK}(max) = 1/T_{CK}$$

Example : execution of the micro-instruction Rin ← Rout

Blue Signals (*control signals*) are generated by the Control Unit

The clock signal is always connected to the registers: write enable has to be activated for register sampling!



Control signals in bold are active during the clock cycle concerning the execution of the micro-step Rin ← Rout

The DLX fixed point instruction set

Data Transfer		Arithmetic/logic		Control	
LW	Ra, offset(Rb)	ADD	Ra,Rb,Rc	SETx	Ra,Rb,Rc
LB	Ra, offset(Rb)	ADDI	Ra,Rb,value	SETIx	Ra,Rb,value
LBU	Ra, offset(Rb)	ADDU	Ra,Rb,Rc	BEQZ	Ra, offset
LH	Ra, offset(Rb)	ADDUI	Ra,Rb, value	BNEQZ	Ra, offset
LHU	Ra, offset(Rb)	SUB	Ra,Rb,Rc	J	offset
SW	Ra, offset(Rb)	SUBI	Ra,Rb,value	JR	Ra
SH	Ra, offset(Rb)	SUBU	Ra,Rb,Rc	JL	offset
SB	Ra, offset(Rb)	SUBUI	Ra,Rb, value	JLR	Ra
LHI	Ra, value	DIV	Ra,Rb,Rc		
		DIVI	Ra,Rb,value		
		MULU	Ra,Rb,Rc	*NOTE: x can represent LT, GT, LE, GE, EQ, NE*	
		MULI	Ra,Rb, value		
		SLL	Ra ,Rb,Rc		
		SLLI	Ra,Rb;value		
		SRL	Ra,Rb.Rc		
		SRLI	Ra,Rb,value		
		SRA	Ra,Rb,Rc		
		SRAI	Ra,Rb,value		
		OR	Ra,Rb,Rc		
		ORI	Ra,Rb,value		
		XOR	Ra,Rb,Rc		
		XORI	Ra,Rb,value		
		AND	Ra,Rb,Rc		
		ANDI	Ra,Rb,value		

Design of the Control Unit

- **Once the Instruction Set and the DATAPATH have been defined, the next step is the design of the Control Unit (*CONTROLLER*).**

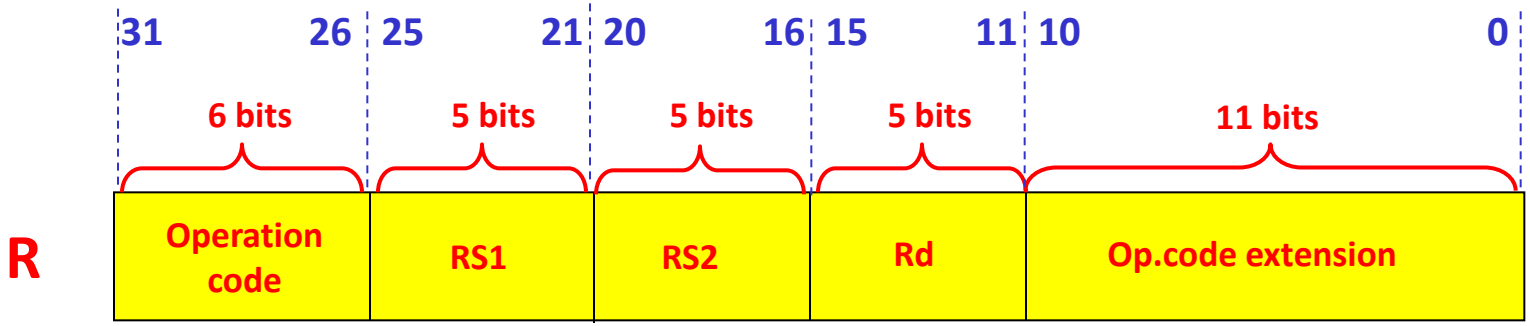
- **The CONTROLLER is a SSC: it can be represented by means of a *state diagram***

- **The CONTROLLER (as any SSC) is stable in any state for a single clock cycle and can transit from a state to another in correspondence of a positive clock edges.**

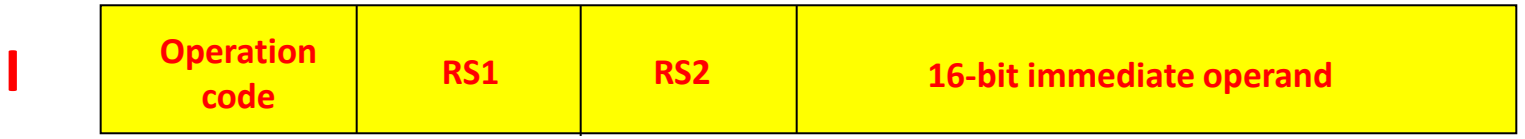
- **Hence, each state lasts only one clock cycle. Micro-operations that need to be executed during a clock cycle are specified (by means of the Register Transfer Language (RTL) language) in the state diagram that describes the CONTROLLER behaviour.**

- **From the RTL description it is possible to determine the control signals that need to be sent to the DATAPATH in order to execute the elementary operations associated with each state.**

DLX instruction format



Logic and arithmetic instructions in the format Rd ← Rs1 op Rs2 or Cond. Set between registers



Load, Store, conditional Branch, JR and JALR (control trasfer via register), Cond and ALU *with immediate operand.*

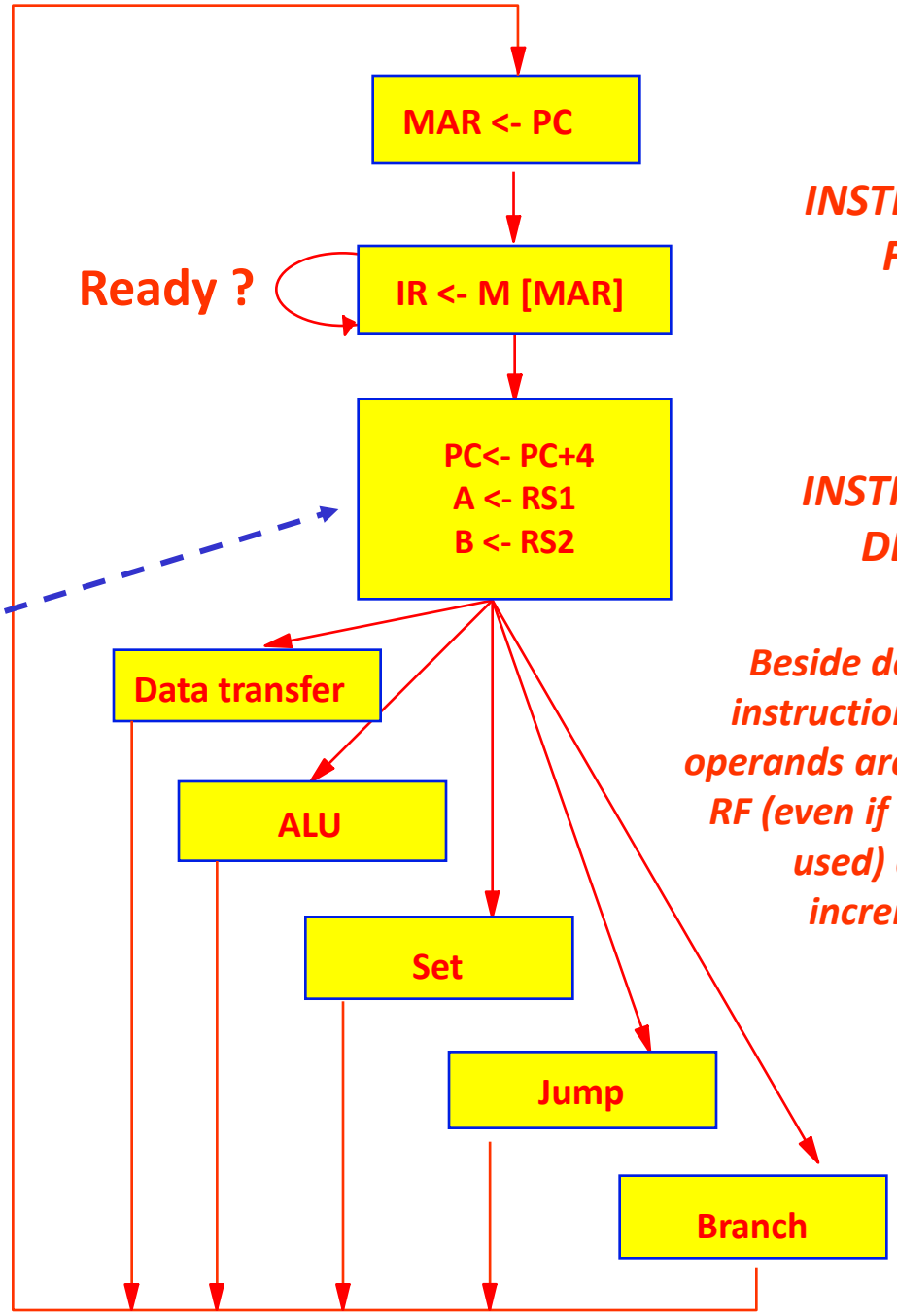
For LD and ALU instructions: RS2=Rdest.

For ST instruction: RS2=source -- RS1 for computing the address or as source for immediate (e.g. SLLI)

- Unconditional Jump with or without linking (J and JAL)

Regular structure of all instructions -> RISC

The Controller state diagram



INSTRUCTION FETCH

INSTRUCTION DECODE

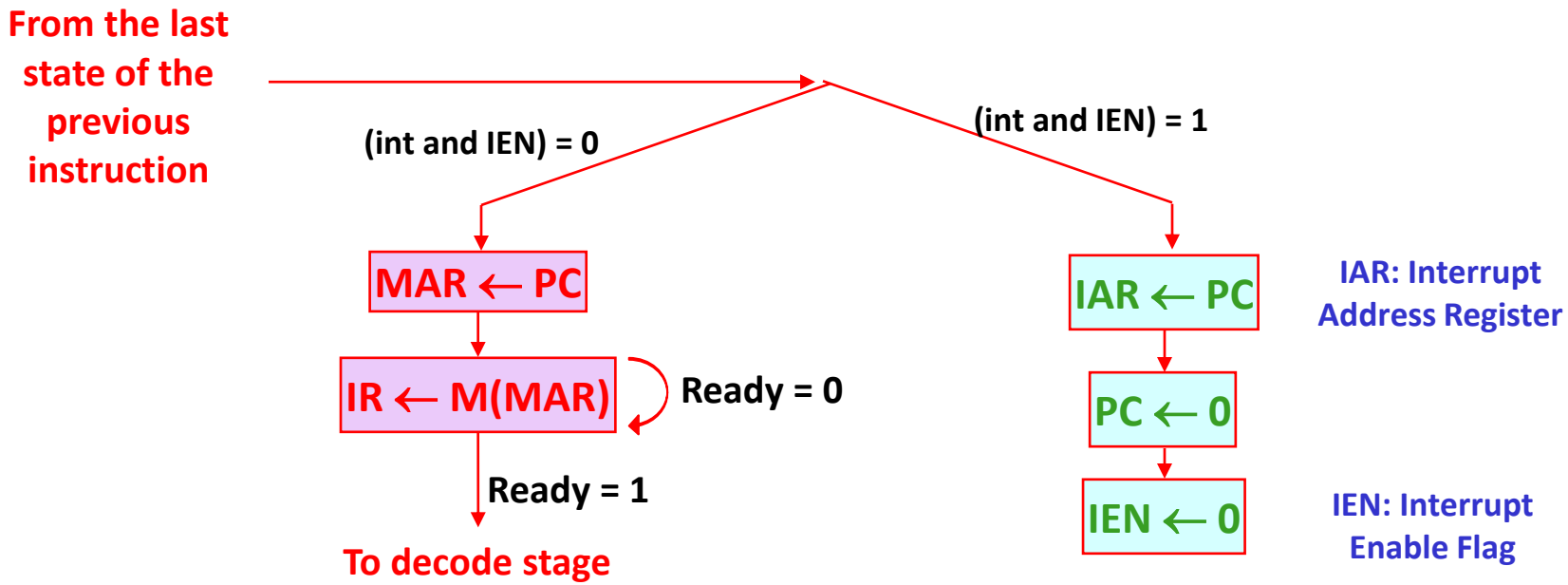
Beside decoding the instruction, the source operands are read from the RF (even if they won't be used) and PC is incremented.

At this stage the op code is still unknown, but transfer to the registers A and B is performed anyway

NOTE: the first three stages are in common with *all* instructions

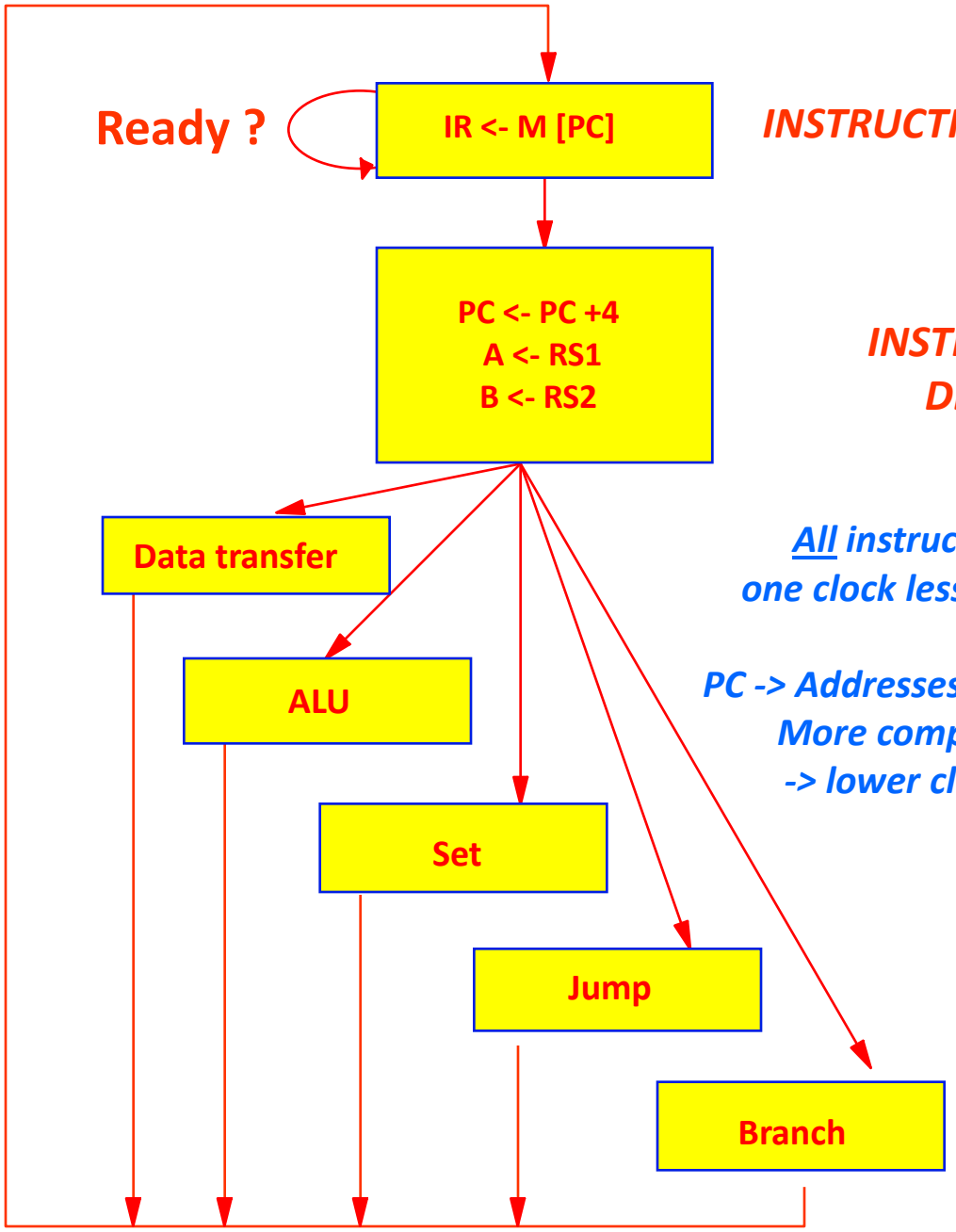
The states of the fetch stage

- At this stage, the possible presence of an interrupt has to be verified (asyncronous external event that the CPU has to handle with specific software);
- if an interrupt is present and can be handled (IEN = true) the instruction calling the procedure stored at address 0 is executed, and the return address is stored in the IAR register;
- if the interrupt is not present OR interrupts are disabled, the next instruction to be executed (whose address is in PC) is read from memory.



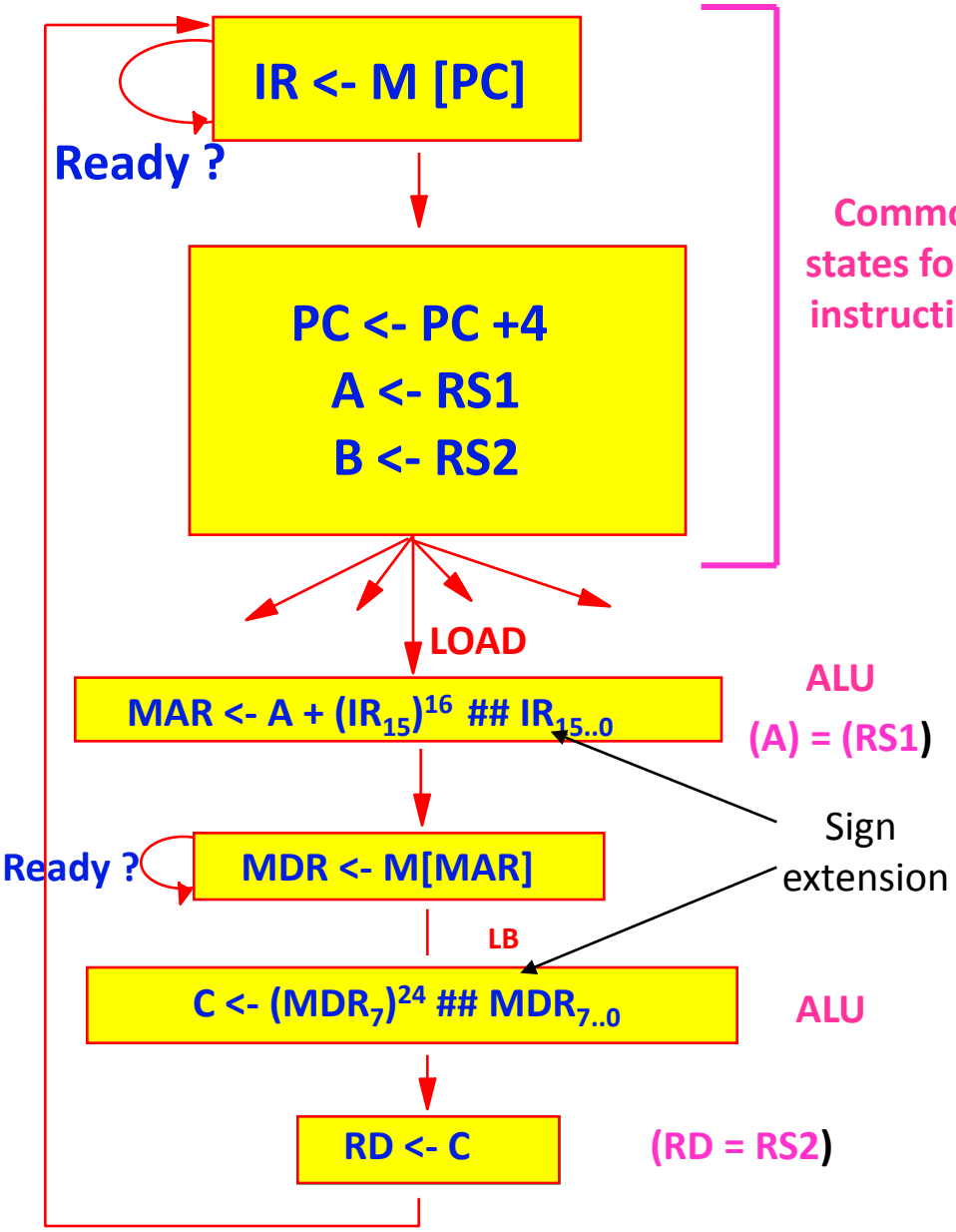
The Controller state diagram: an alternative

The DATAPATH is modified so that the memory can be addressed directly from PC. Less states but increasing complexity

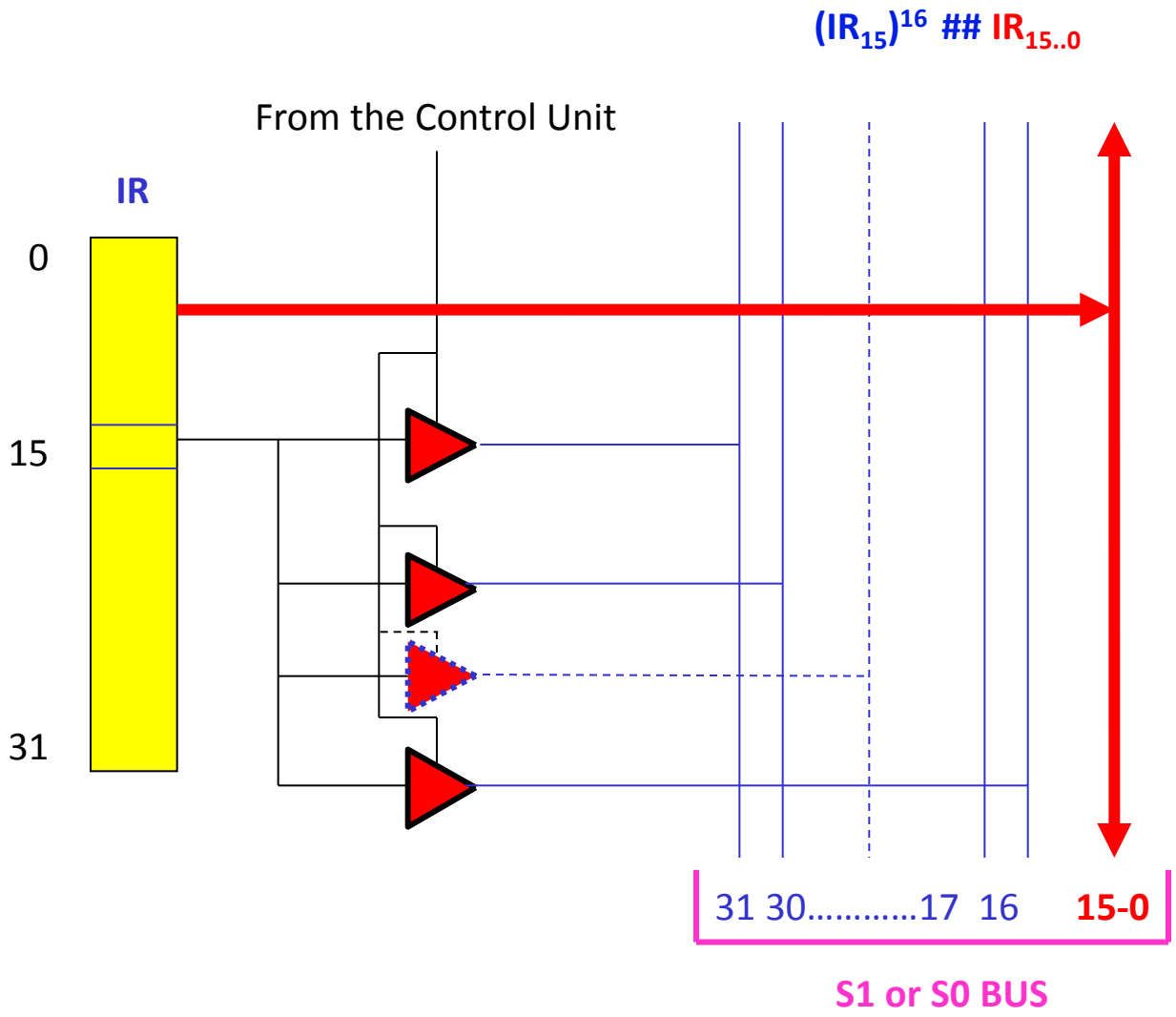


Control for the LB instruction *(LOAD BYTE)*

Why do we have to go through register C? Why not writing directly the loaded data in the destination register?
-> Clock frequency!!!!



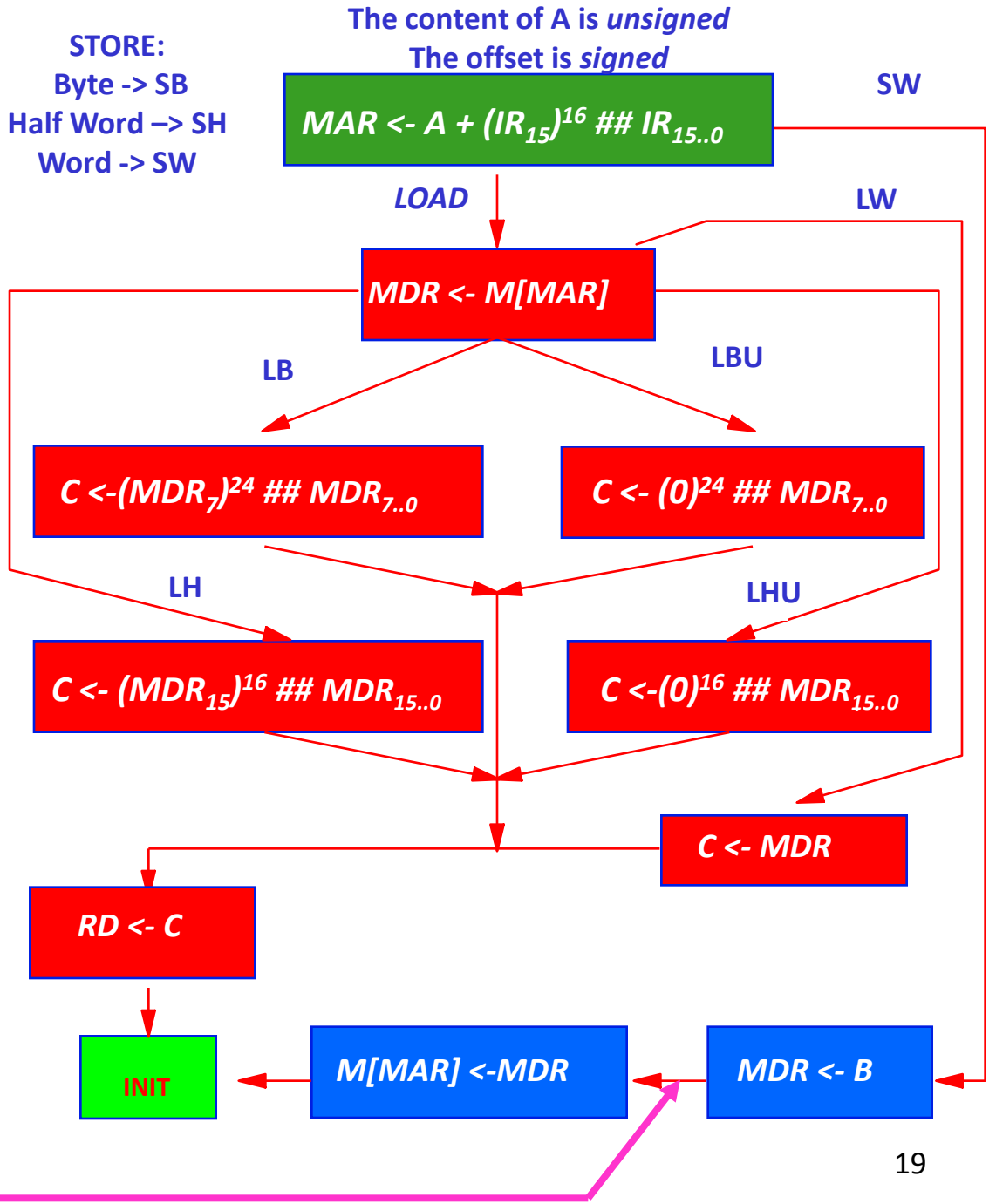
Sign extension

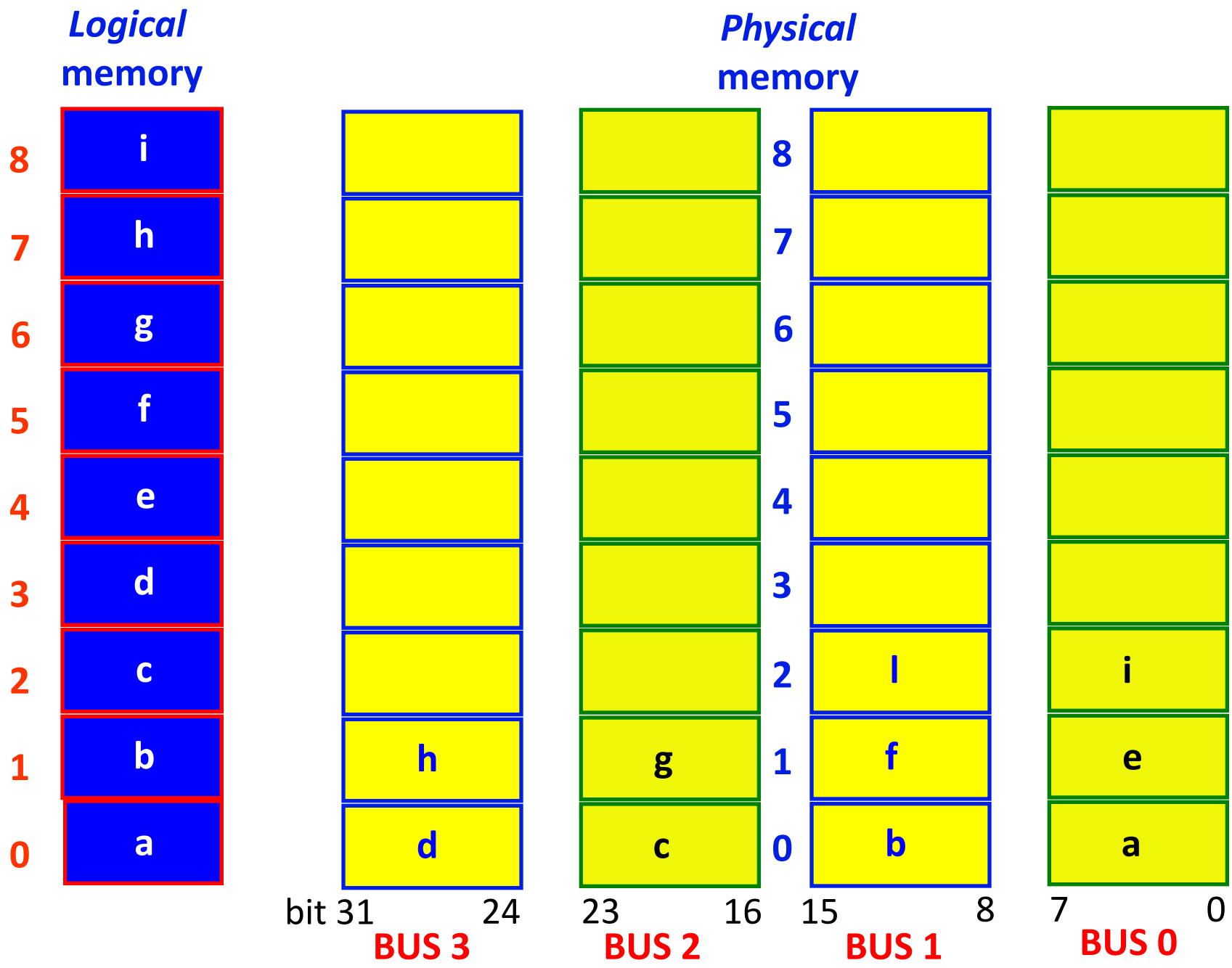


Control for DATA TRANSER instructions

NOTE: while reading, the less significant part of the data is always aligned with the MDR register so as to allow filling if required

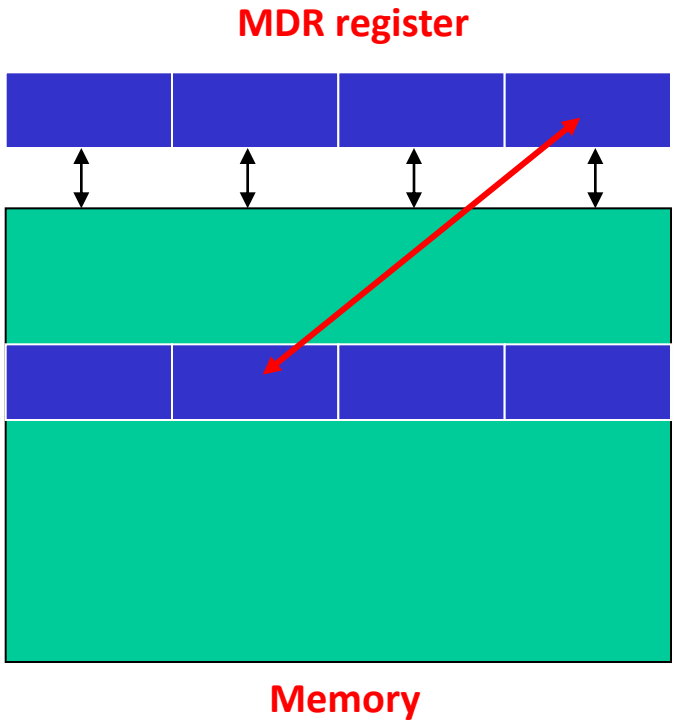
State diagrams for SH and SB (always unsigned) are not shown. They require the activation of specific WEs in memory and shifters of the bytes in the MDR register. How could they be designed?




Logical memory
Physical memory
8
7
6
5
4
3
2
1
0
i
h
g
f
e
d
c
b
a
h
d
g
c
l
f
b
i
e
a
bit 31
24
23
16
15
8
7
0
BUS 3
BUS 2
BUS 1
BUS 0

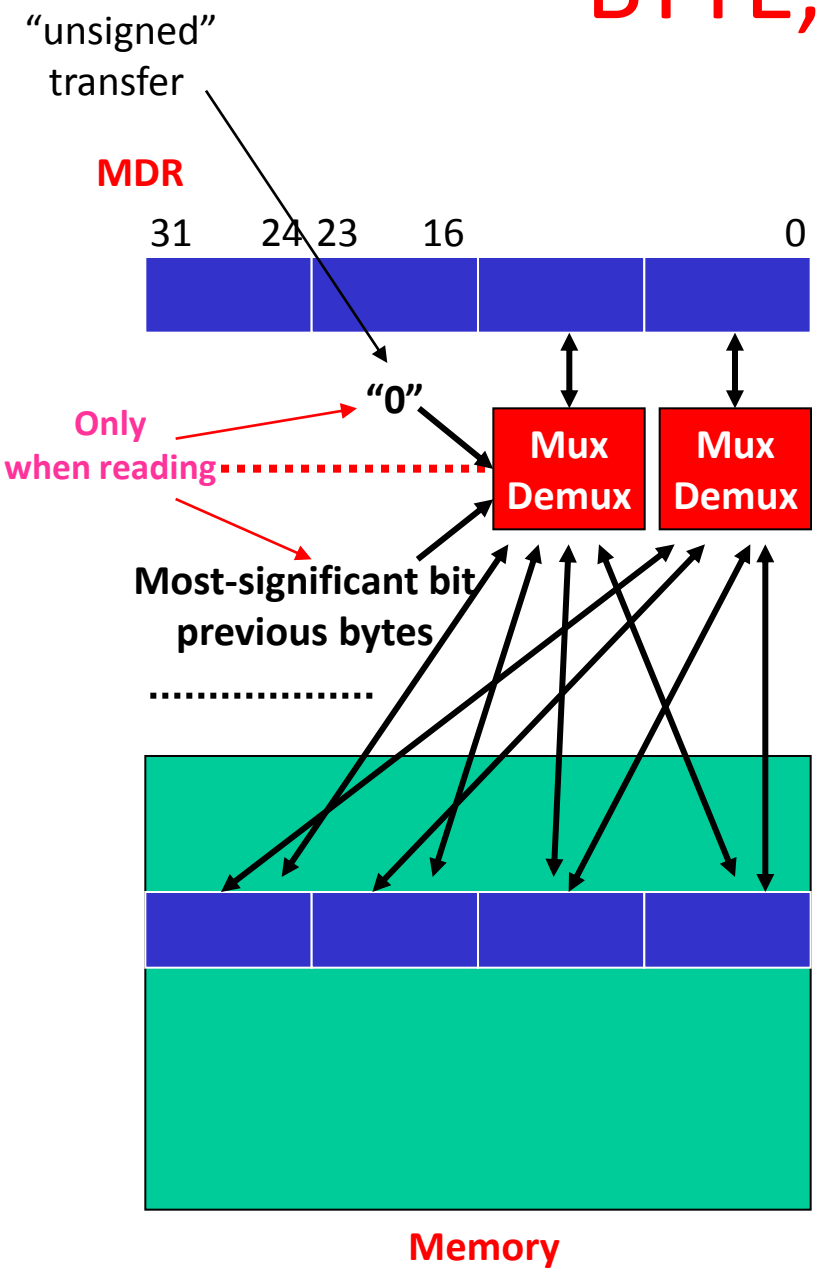
BYTE, HW transfers

- Byte transfers are ALWAYS aligned
- HW transfers require addresses to be multiple of 2
- Word transfers require addresses to be multiple of 4
- In case of unalignment: fault
- In case of storing data smaller than words there is NO sign extension
- Reading/writing bytes and HW (due to the mutual unalignment between registers and memory) requires that between registers and memory some mux/demux are interponed (carried out with tristates)



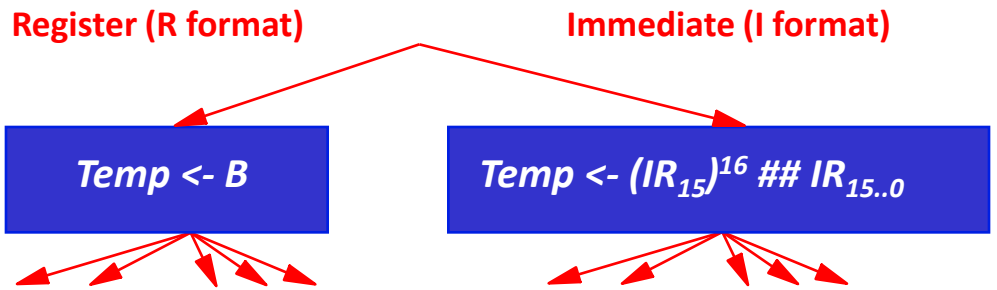
How are the WEs in memory activated? Try to design the circuit..

BYTE, HW transfers



- **Mux/Demux 7-0** connected to all 4 buses to store and load bytes and HWs
- **Mux/Demux 15-8:**
  - •connected to buses 31-24 and 15-8 to store and load HWs
  - •connected to buses 23-16 and 7-0 for sign extension when loading (signed) bytes
- **Mux/Demuxs 23-16 and 31-24** (not shown) connected also to bit n. 7 of byte 7-0 from memory (LB) and bit 15 of byte 15-8 from memory (LH).
- For example, in a LB, the 7-0 MUX is directly connected to memory, while the 15-8, 23-16 and 31-24 MUXs are connected to bit 7 of the 7-0 MUX coming from memory.
- In a SH with the address multiple of 2 and not of 4, the 7-0 DEMUX is connected from the MDR to the 23-16 memory, and the 15-8 DEMUX is connected to the 31-24 memory. The other two bytes in memory are not modified.

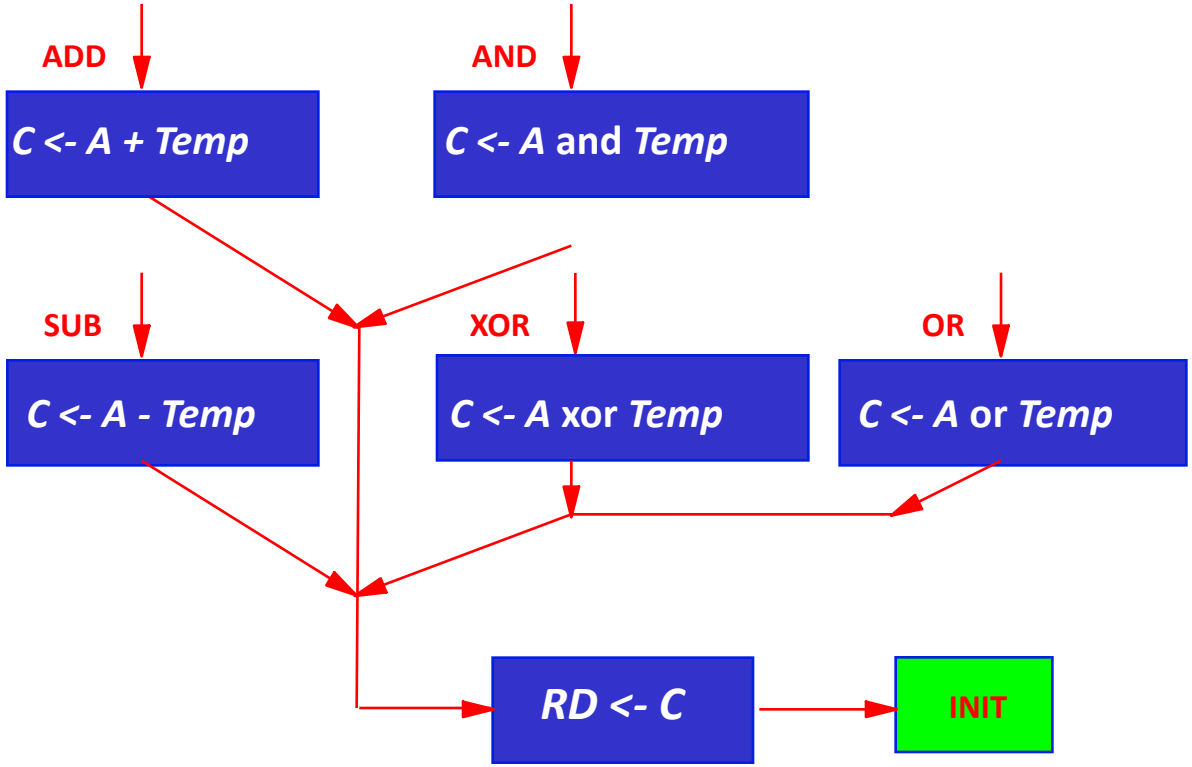
Examples of ALU instructions



The register content is signed for the arithmetical operation

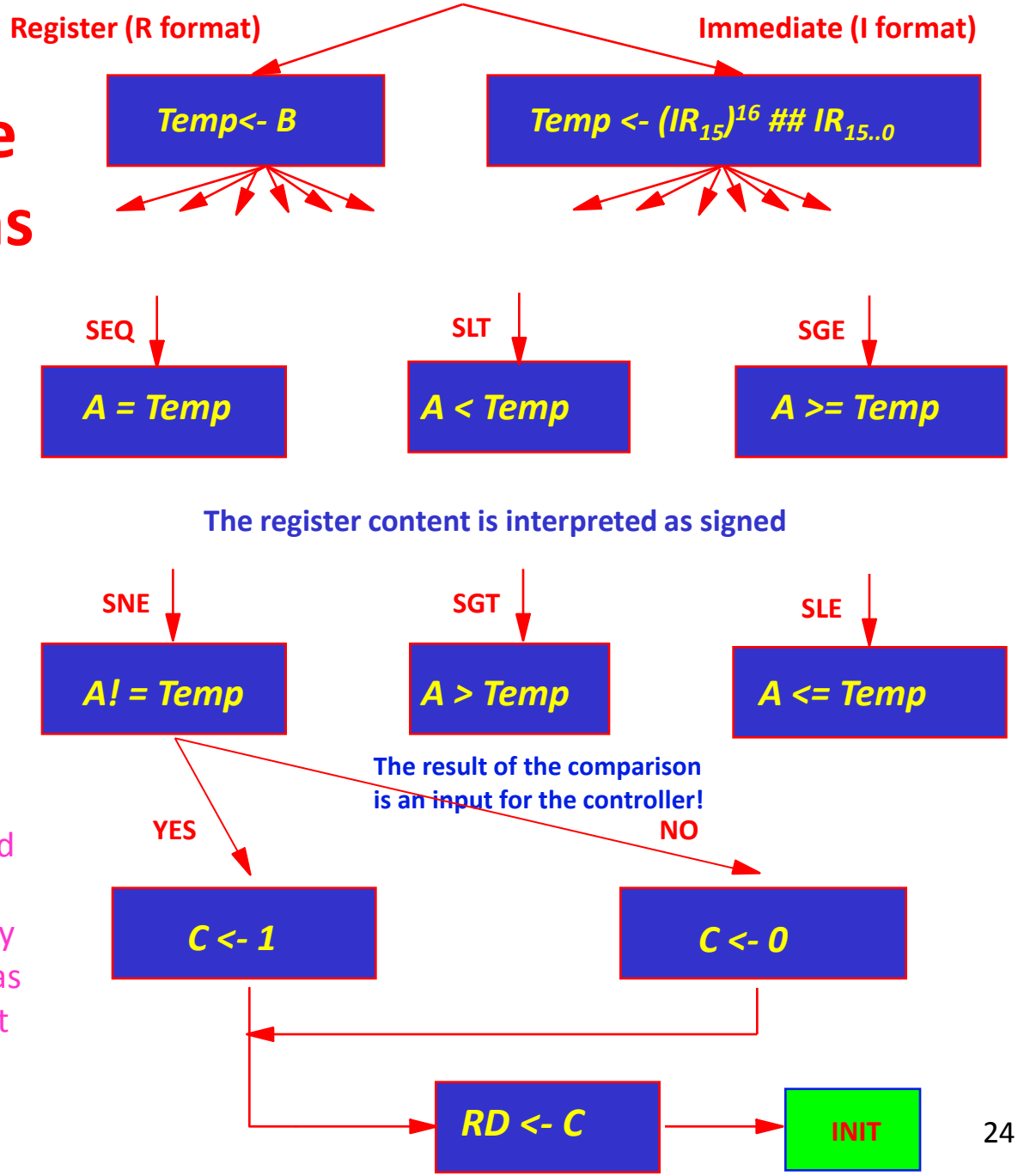
By duplicating the states of the diagram, the passage through TEMP can be avoided – but higher complexity in the Control Unit

The same scheme can be used for «Shift» instructions etc.



Control for the SET instructions (*comparison*)

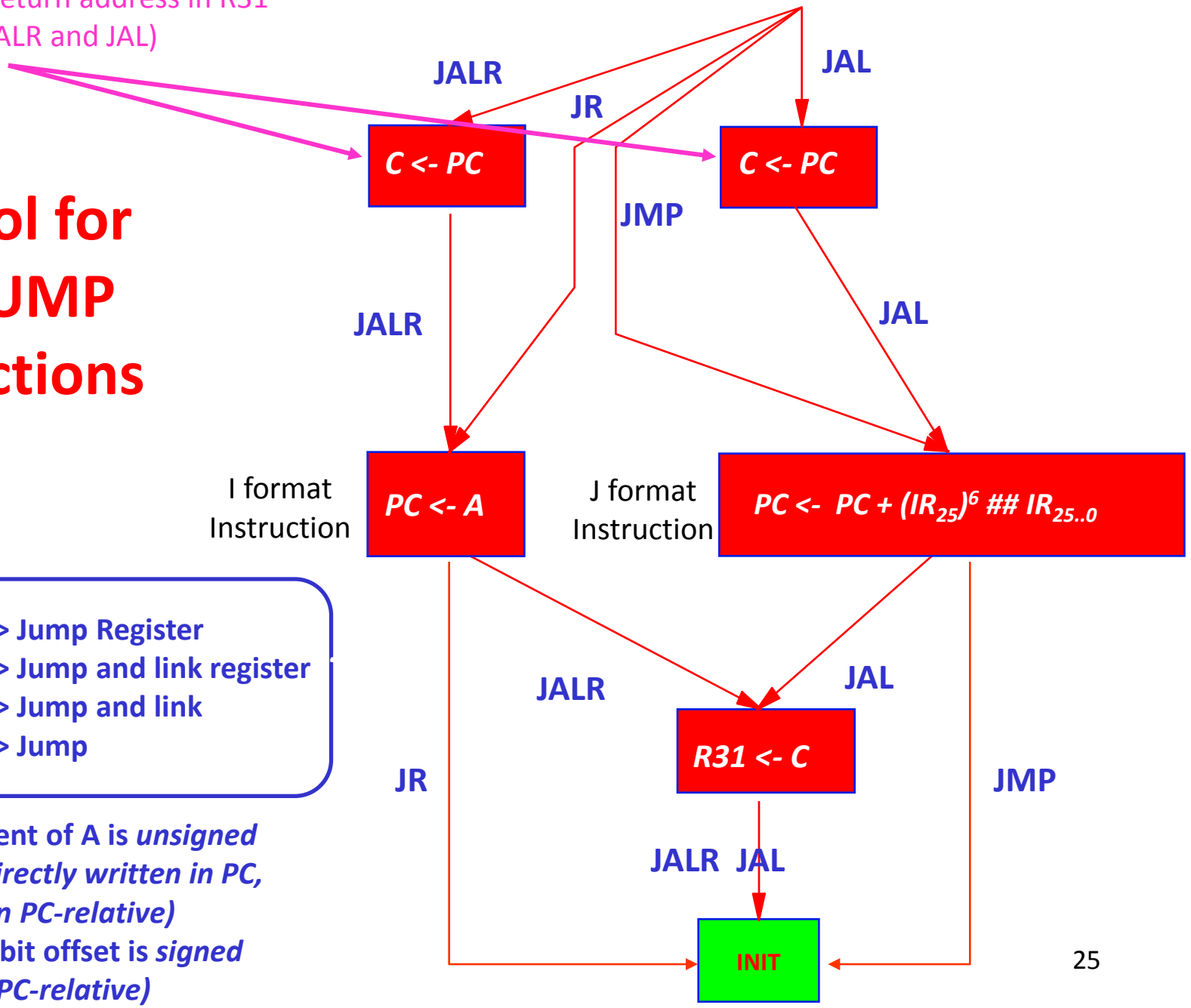
e.g. SLT R1,R2,R3



By duplicating the paths, the passage through TEMP could be avoided

The micro-steps are executed in the ALU but the result is **NOT stored** in a register: only the ALU sign bit is deployed as input for the least significant bit of register C, with all remaining bits set to 0

Control for the JUMP instructions



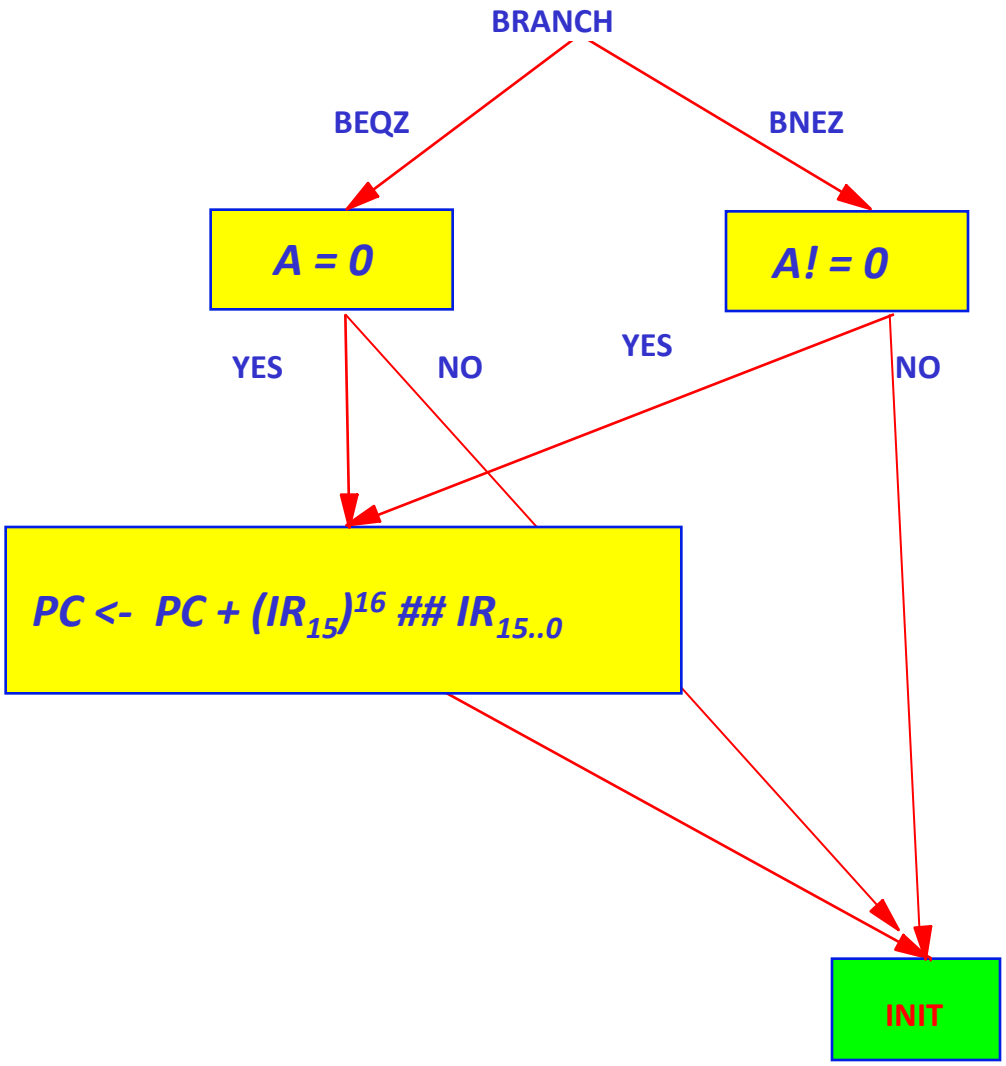
JR	-> Jump Register
JALR	-> Jump and link register
JAL	-> Jump and link
JMP	-> Jump

The content of A is *unsigned* *(will be directly written in PC, non PC-relative)*
The 26-bit offset is *signed* *(PC-relative)*

Control for BRANCH instructions

(I-type instructions)

E.g. BNEQZ R5, 100



Number of clock cycles required for the execution of the instructions

Instruction	Cycles	Wait	Total
Load	6	2	8
Store	5	2	7
ALU	5	1	6
Set	6	1	7
Jump	3	1	4
Jump and link	5	1	6
Branch (taken)	4	1	5
Branch (not taken)	3	1	4

$$CPI = \sum_{i=1}^{n} \left(CPI_i * \frac{N_i}{\text{total number of instructions}} \right)$$

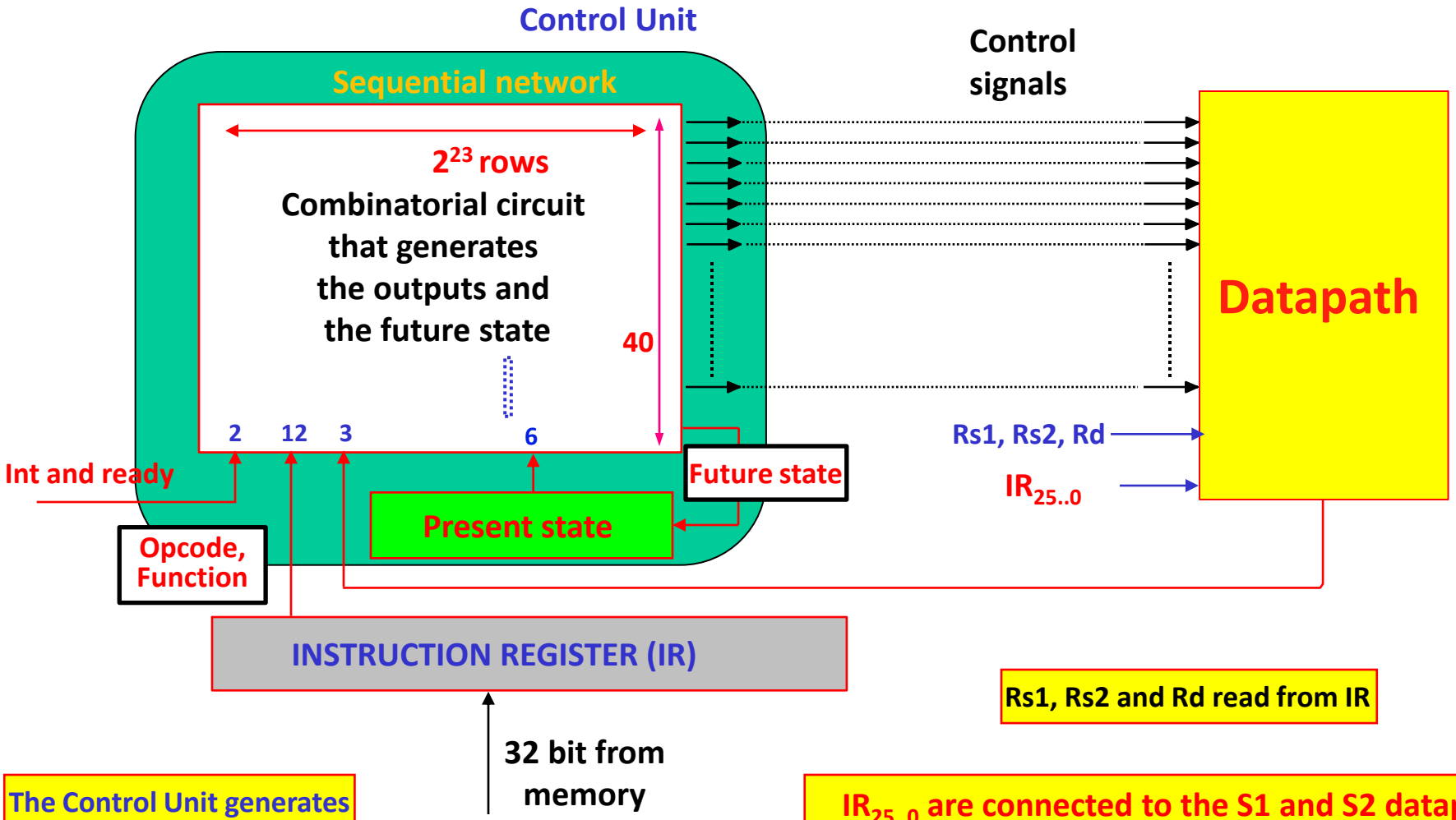
Example in the DLX case

LOAD: 21%, STORE: 12%, ALU: 37%, SET: 6%, JUMP: 2%

BRANCH (taken): 12%, BRANCH (not-taken): 11%

→ ***CPI = 6.3***

Wired control (“*hardwired*”)



Instruction execution steps

In the DLX the execution of all instructions can be decomposed into *5 steps*, each step being executed in one or more clock cycles.

These steps are called:

1) FETCH: the instruction is read from memory and stored in IR.

2) DECODE: the instruction in IR is decoded and the source operands are read from the Register File.

3) EXECUTE: arithmetical or logical processing by means of the ALU.

4) MEMORY: memory access and, in case of a BRANCH, update of the PC ("branch completion").

5) WRITE-BACK: writing on the Register File (when required)

The *micro-operations* executed at each step

1) FETCH

MAR ← PC ;

IR ← M[MAR];

2) DECODE

A← RS1, B← RS2, PC← PC+4

The *micro-operations* executed at each step

3) EXECUTE

- **MEMORY:**

MAR ← A + $(IR_{15})^{16}$ ## $IR_{15..0}$; **(they use ALU, S1, S2, dest)**

MDR ← B; ***(NOTE: it is needed by the Store instructions, where RD=RS2, while it is a useless operation for the LOAD instructions)***

- **ALU:**

C <- A *op* B (*or* A op $(IR_{15})^{16}$ ## $IR_{15..0}$) ;

C <- sign(A *op* B (*or* A op $(IR_{15})^{16}$ ## $IR_{15..0}$)); if SCn

- **BRANCH:**

Temp ← PC + $(IR_{15})^{16}$ ## $IR_{15..0}$) ; (it uses ALU, S1, S2, dest: here we still don't know if we have to "jump")

The *micro-operations* executed at each step

4) MEMORY

- **Memory:**

MDR ← M[MAR]; (*LOAD*)

M[MAR] ← MDR; (*STORE*)

- **BRANCH:**

If (Cond) PC ← Temp;
[A] is the register that determines whether there is or not a jump (Cond) ;

5) WRITE-BACK

C ← MDR; (*if it is a LOAD – two microsteps)*)

RD ← C ;