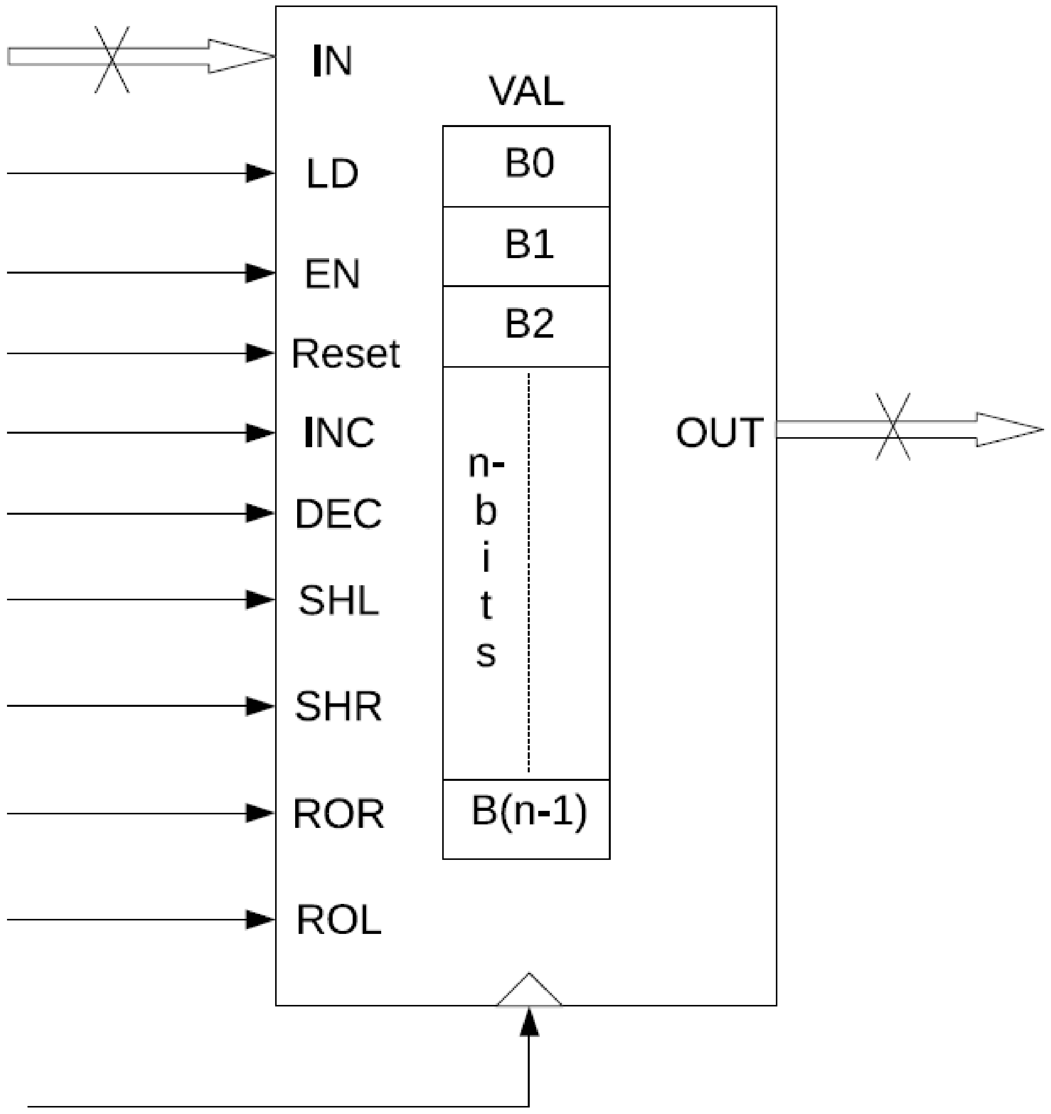
Lecture 26 – Processor design: The Beginning

Dr. Aftab M. Hussain,

Assistant Professor, PATRIoT Lab, CVEST

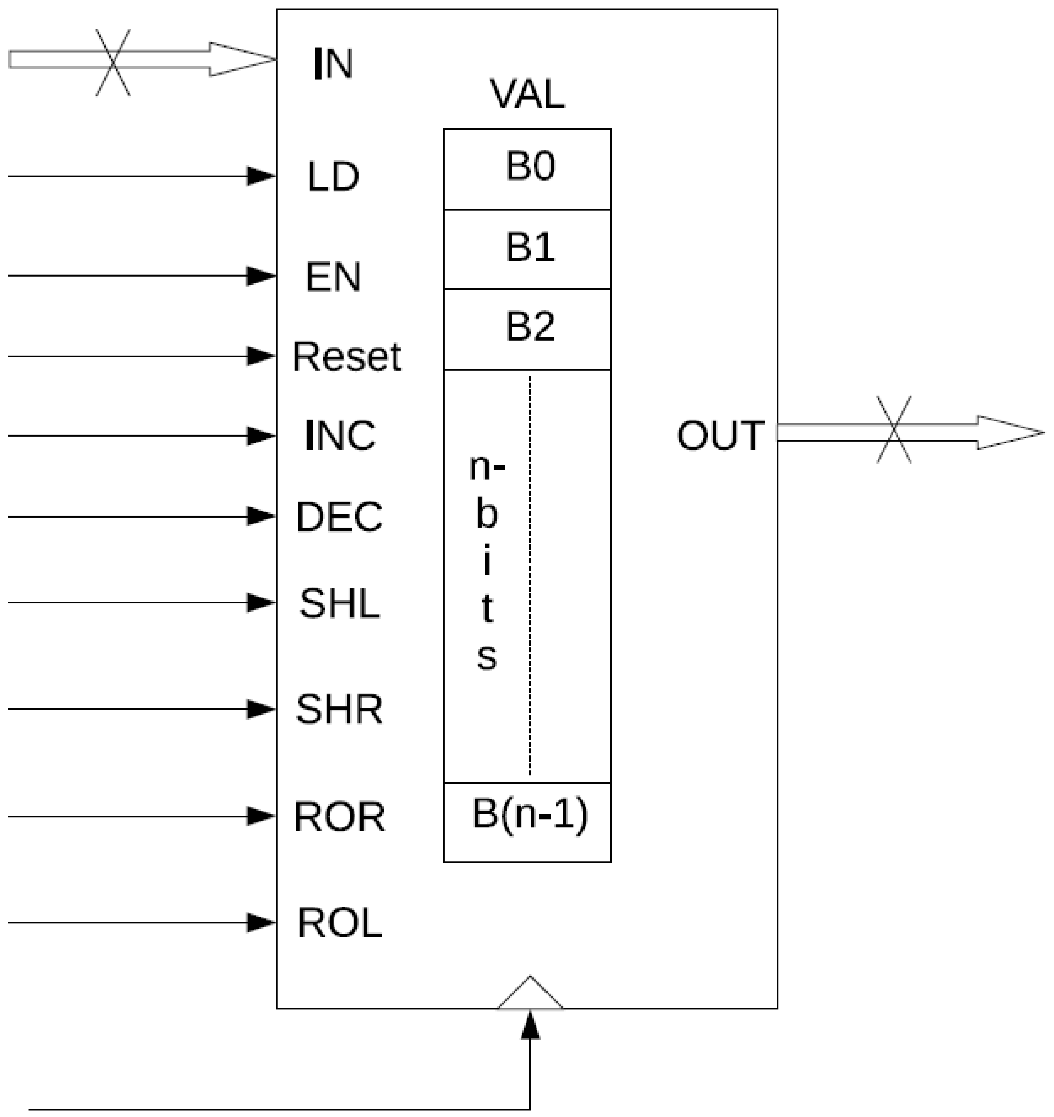
The universal register

- A universal register is a very useful entity in processor design
- It can store value as well as have some basic arithmetic functions such as increment/decrement etc.
- The register has two input/output lines (sometimes coupled into one IO bus)
- Input lines IN are logically connected internally to the inputs of the n flip-flops inside the register
- Output lines OUT are the lines that hold the value stored in the register when the register is accessed for read. The OUT lines are tristate-capable so that they can be connected to a common data bus



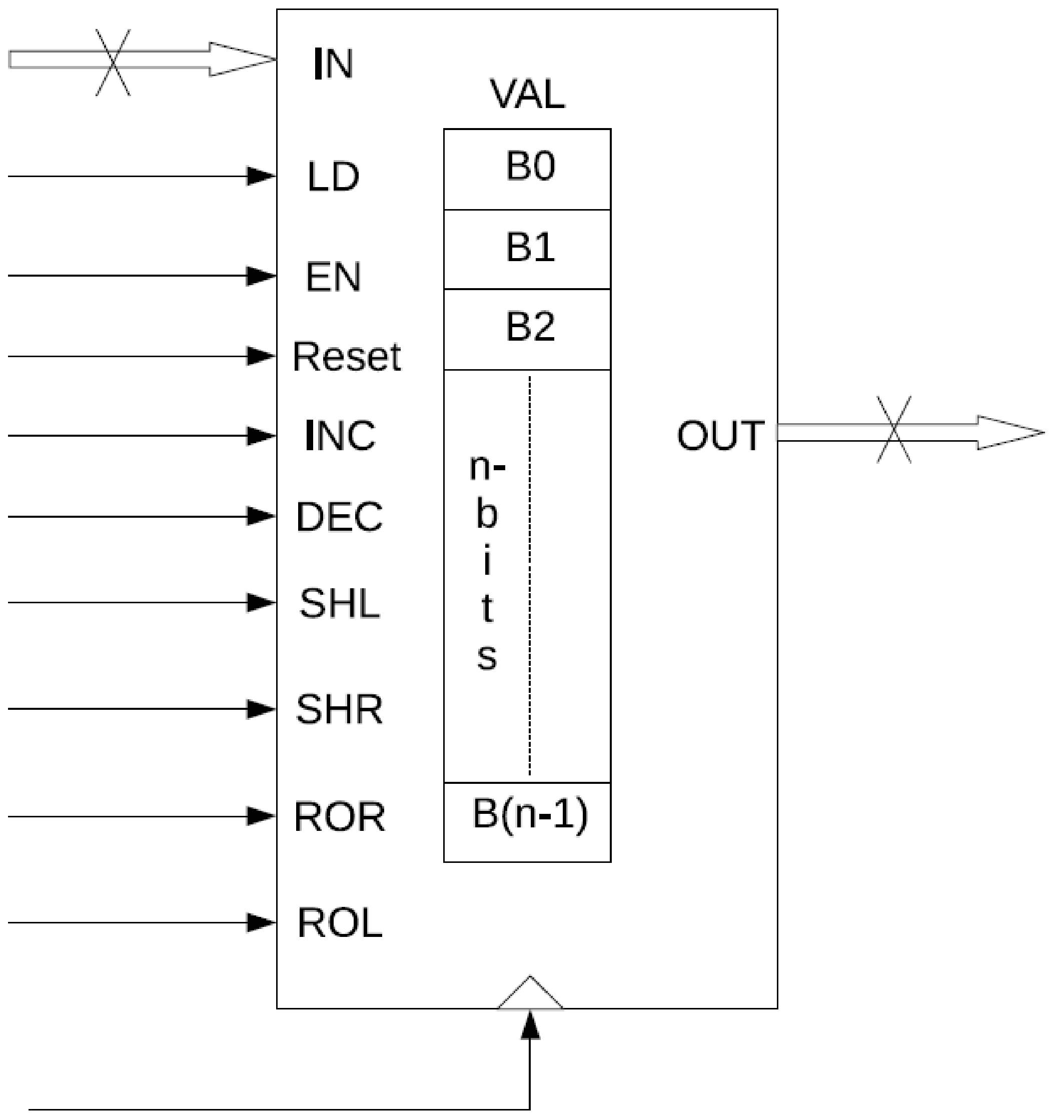
The universal register

- The control bits are as follows:
  - The input EN controls the high-impedence state of the output lines OUT. When EN is 0, the OUT lines will be in the high-impedence state. When EN is 1, the OUT lines will be driven to the electric levels corresponding to the value stored in the register
  - The input LD loads the register from the input, which changes the value stored in it. When LD is 1, the value indicated by the electric levels of the IN lines will replace the previous value stored in the register
  - The input RESET controls resetting of the register. When RESET is 1, a 0 value will be written to all bits stored in the register. We will assume a synchronous reset.



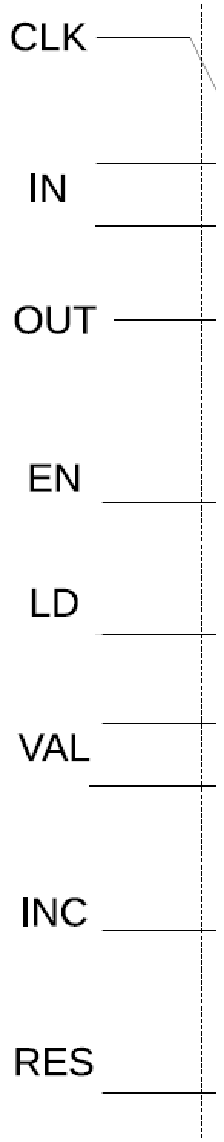
The universal register

- The control bits are as follows:
  - Inputs INC and DEC control the incrementing or decrementing the value stored in the register, interpreted as a number.
  - Inputs ROL, ROR control the left or right rotation of the register contents. This is shifting such that the last bit is feedback to the first bit
  - Inputs SHL, SHR control the left or right shift of the register contents. We assume a 0 will fill the void during the shift



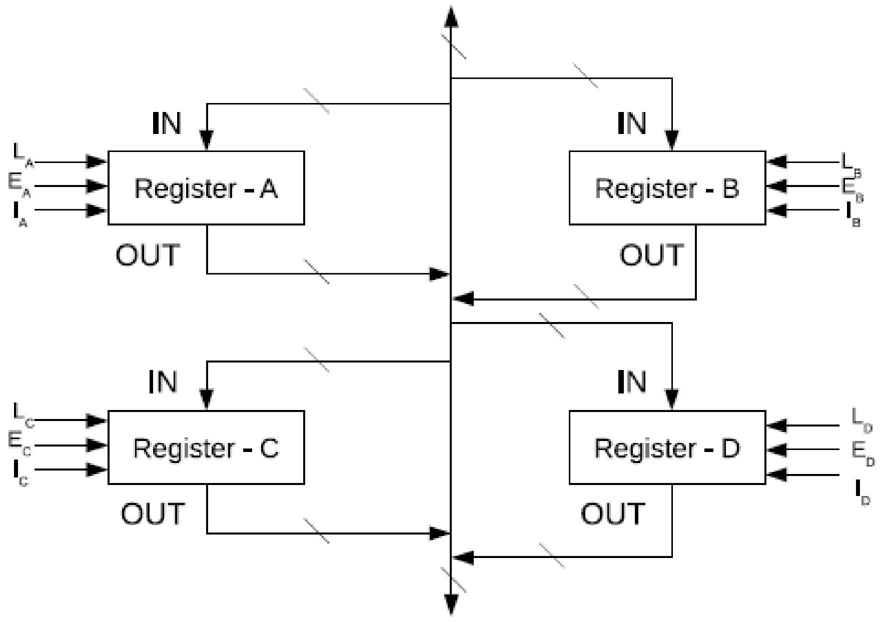
The universal register

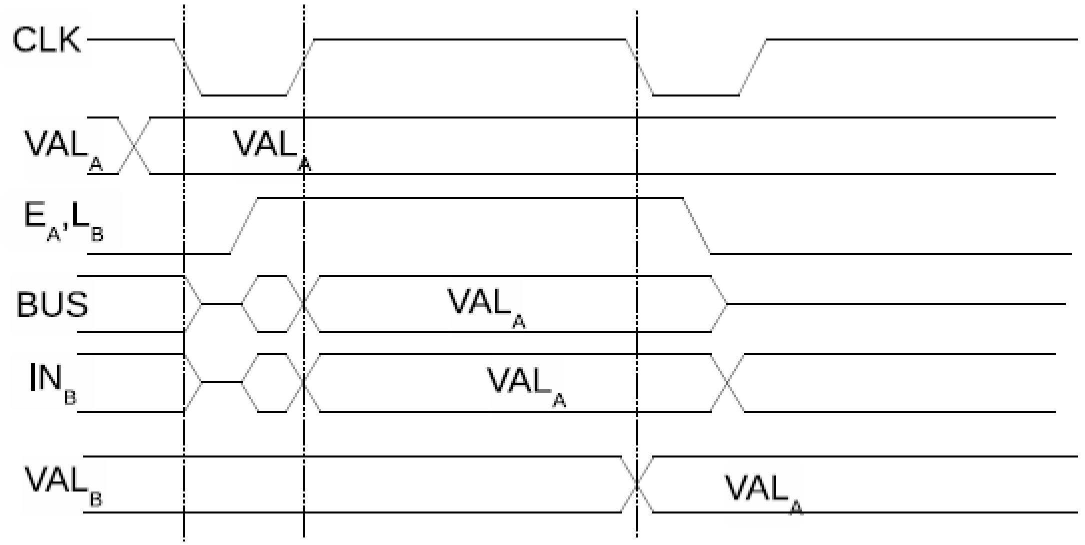
- The timing diagram depends on how we design the register
- In this case, we have designed such that all instructions (actions) happen to the stored value at the negative edge of the clock
- While the EN puts the data on the output bus at the positive edge of the clock
- This makes the setup/hold more complex but can be time saving



Creating a common bus

- Using tristate buffers, we can connect multiple input/outputs of multiple registers on the same bus
- Consider registers A and B
- The enable, load, and increment control signals of register A are respectively E_A, L_A, and I_A and for register B are labelled E_B, L_B, and I_B
- Let us say we need to transfer from A to B
- We enable E_A and L_B and in one clock cycle, the contents get transferred





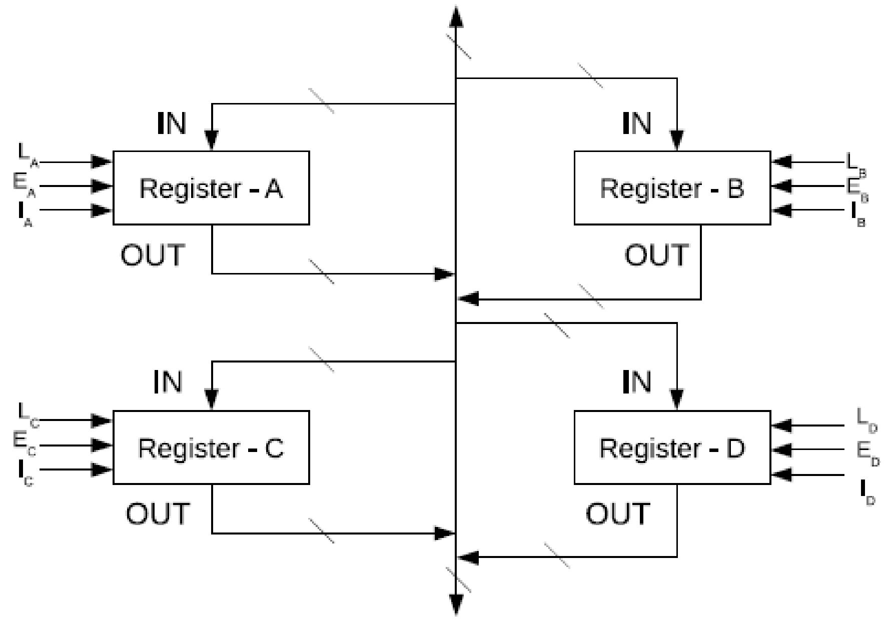
Creating a common bus

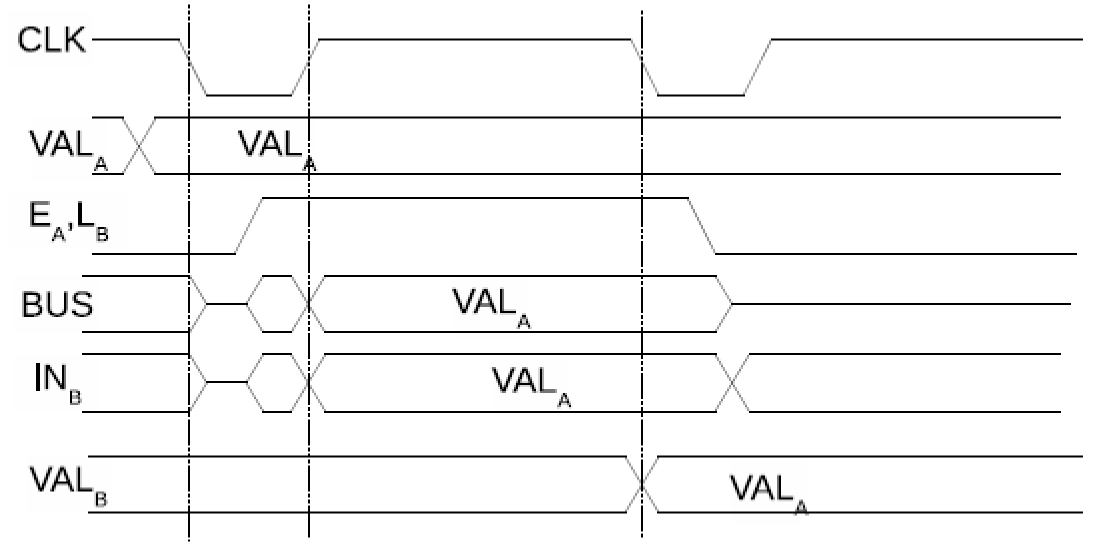
- In effect, we have moved the data to register B from register A, like a assignment instruction!

  A=B;

  MOV B A

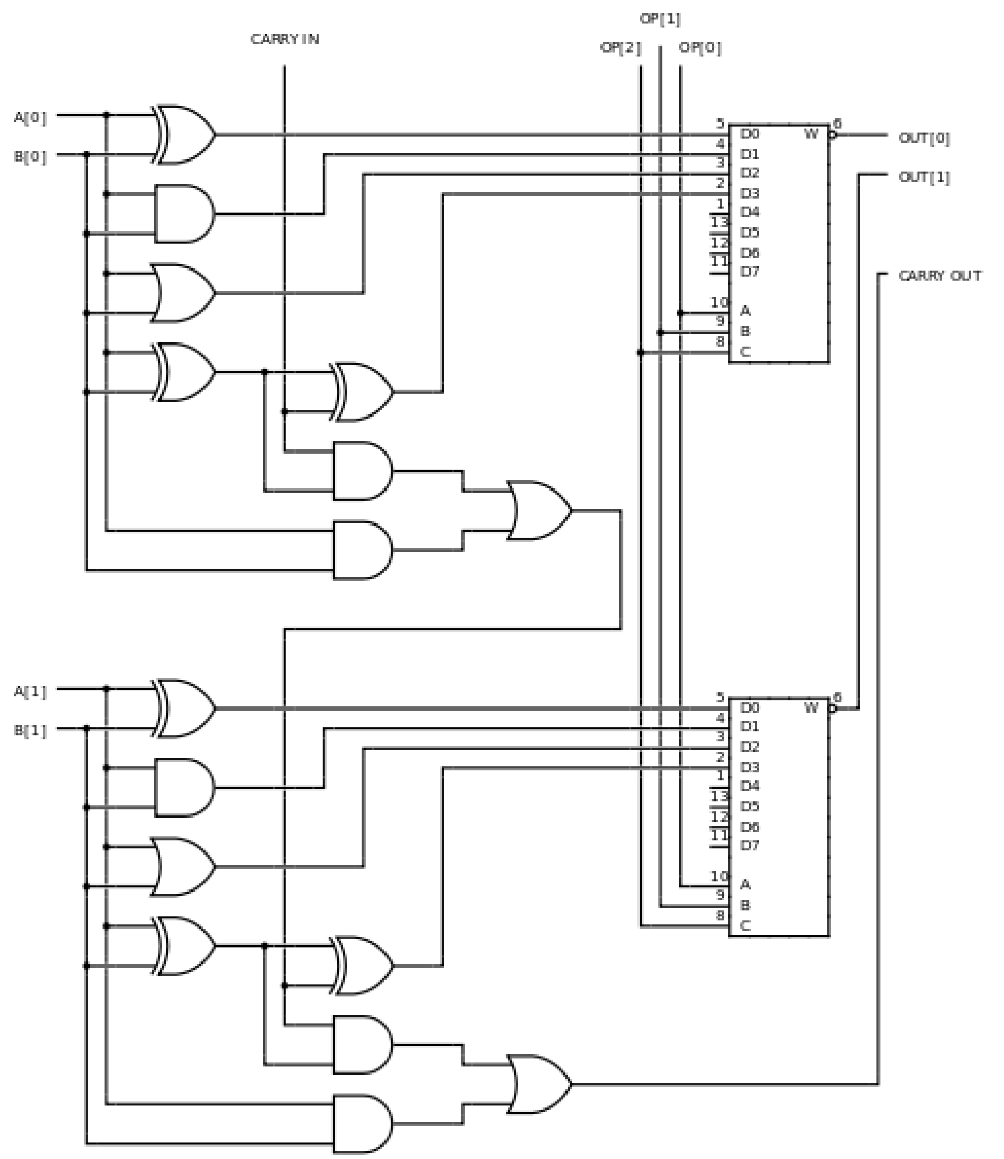
- At the level of the hardware, activating 2 signals simultaneously was all that was done
- The rest happened due to the bus connection, the clock, and the design of the register
- What if we activate L_B, E_D, L_C, L_A together?
- What if we activate E_B, L_A, I_B





ALU design

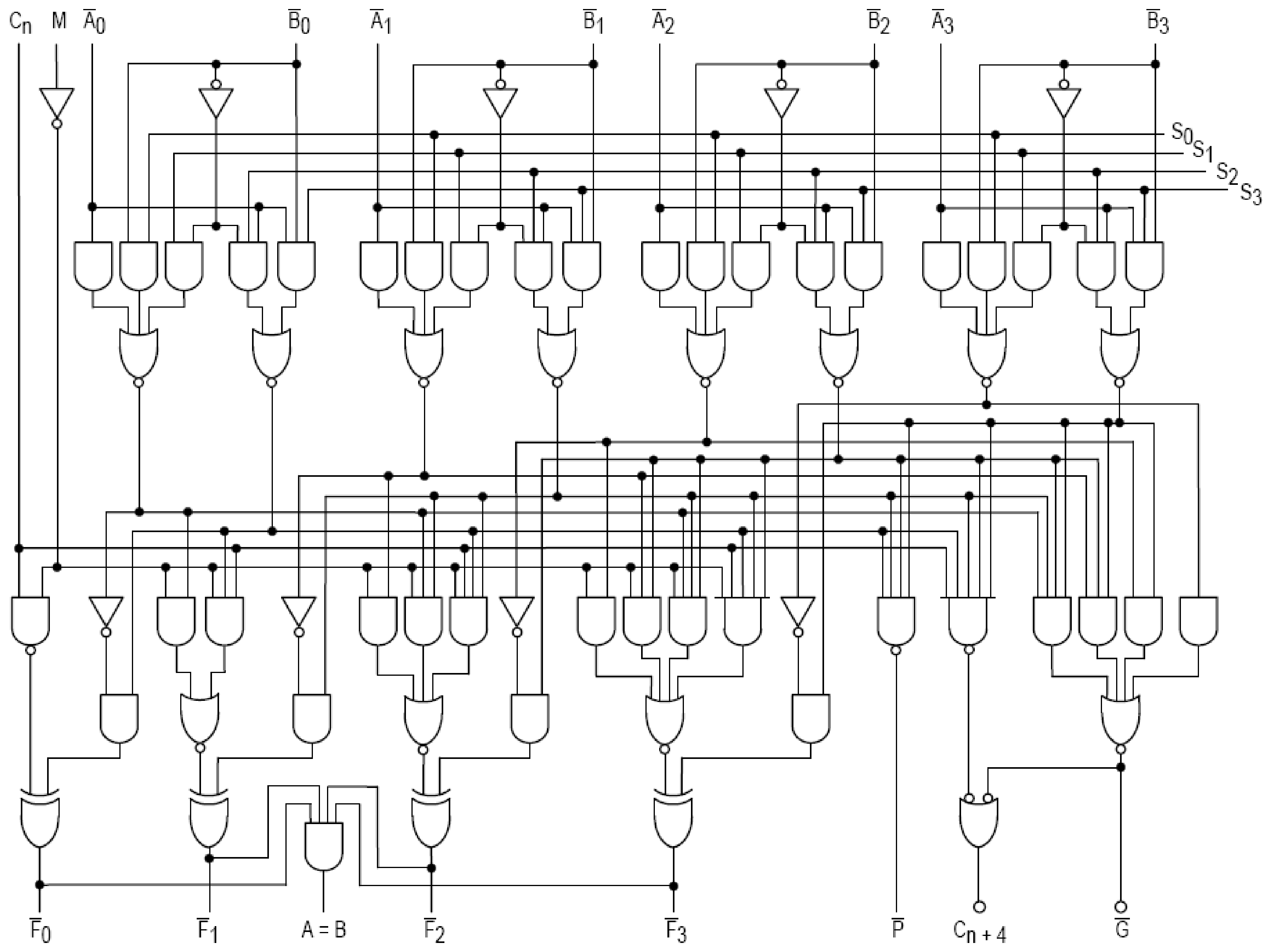
- Enter, The ALU!
- We generally consider an ALU as a "simple" combinational circuit capable of performing several basic functions on a set of two binary numbers – add, sub, multiply, increment, AND, OR, XOR, and many other operations based on a select input
- This is generally implemented using a combination of MUXes within the ALU



ALU design

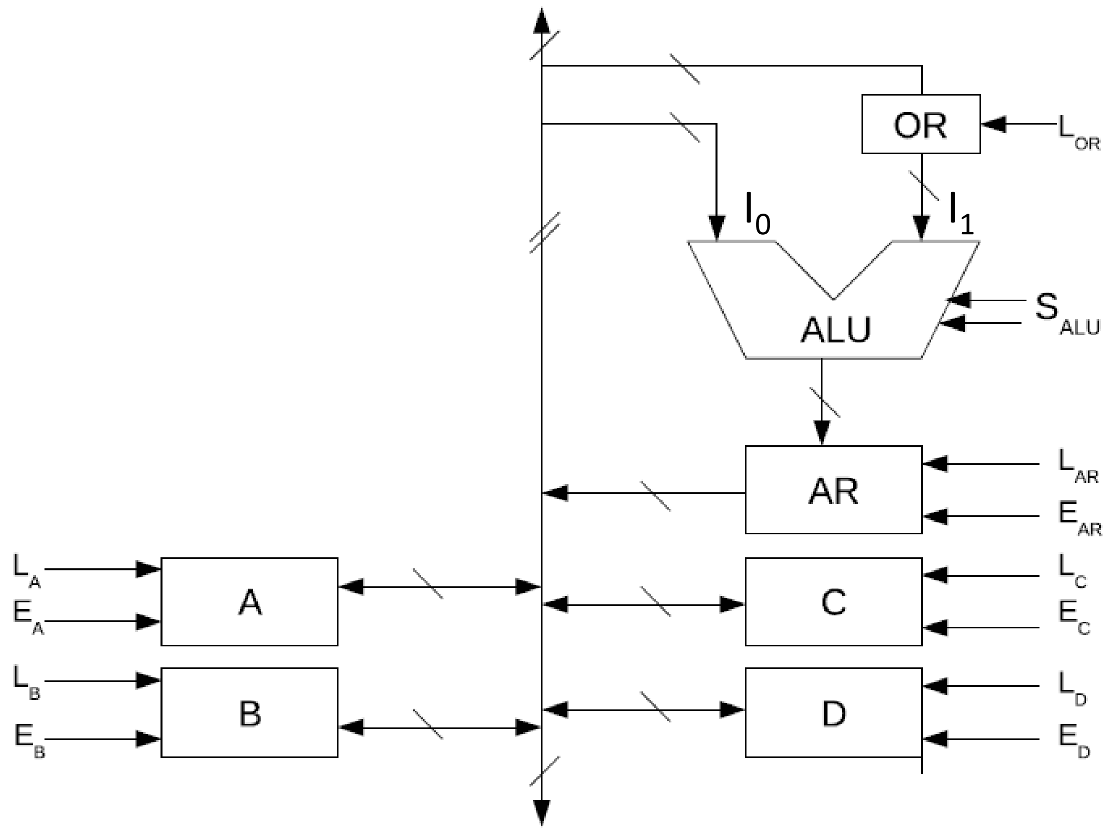
- Internal architecture of ALU IC 74181

Selection				Active-low inputs & outputs	
S3	S2	S1	S0	Logic (M = 1)	Arithmetic (M = 0) (Cn = 0)
0	0	0	0	$\overline{A}$	A minus 1
0	0	0	1	$\overline{AB}$	AB minus 1
0	0	1	0	$\overline{A}+B$	$A\overline{B}$ minus 1
0	0	1	1	Logical 1	-1
0	1	0	0	$\overline{A+B}$	A plus $(A+\overline{B})$
0	1	0	1	$\overline{B}$	AB plus $(A+\overline{B})$
0	1	1	0	$\overline{A\oplus B}$	A minus B minus 1
0	1	1	1	$A+\overline{B}$	$A+\overline{B}$
1	0	0	0	$\overline{A}B$	A plus $(A+B)$
1	0	0	1	$A\oplus B$	A plus B
1	0	1	0	B	$A\overline{B}(A+B)$
1	0	1	1	$A+B$	$A+B$
1	1	0	0	Logical 0	A plus A
1	1	0	1	$A\overline{B}$	AB plus A
1	1	1	0	AB	$A\overline{B}$ plus A
1	1	1	1	A	A



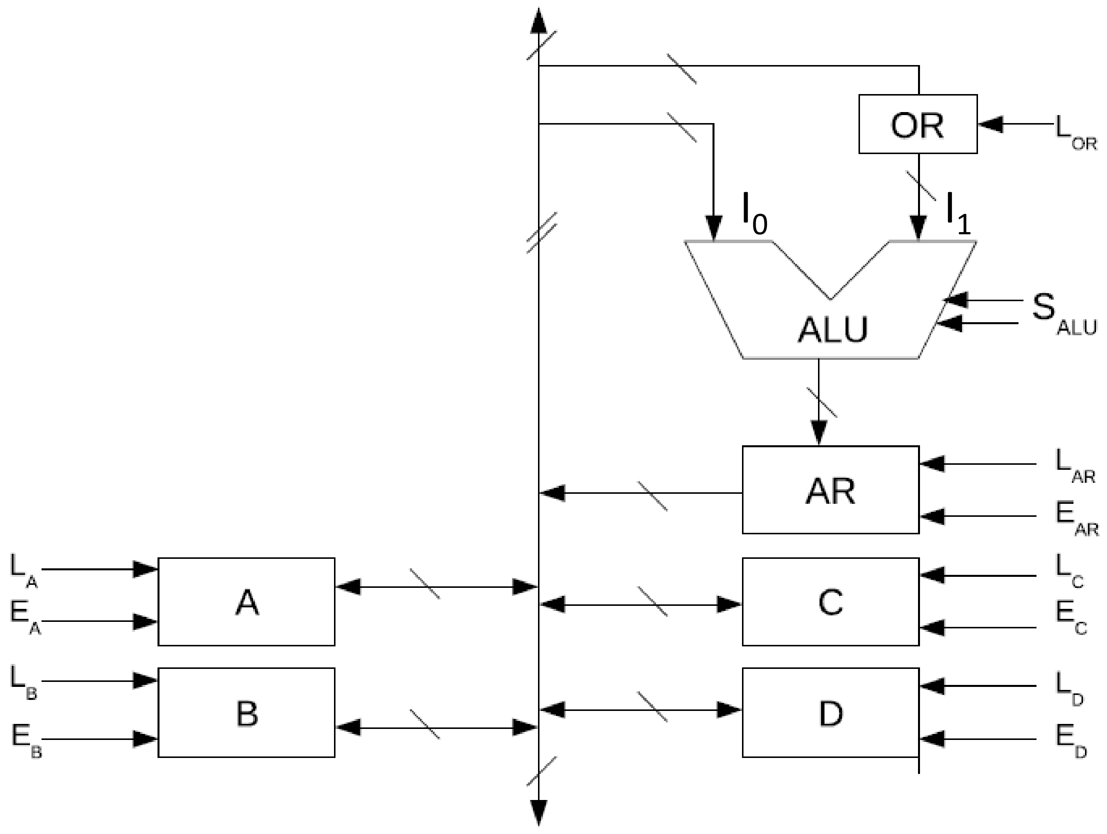
ALU on a bus

- Consider the configuration which has a few registers and an ALU
- Assume all the registers are made using the multipurpose registers
- The ALU is a combinational circuit with 2 inputs and two lines to select the function to be performed
- The two select lines can be in one of 4 combinations
- We refer to them symbolically as ADD, SUB, AND, PASS0



ALU on a bus

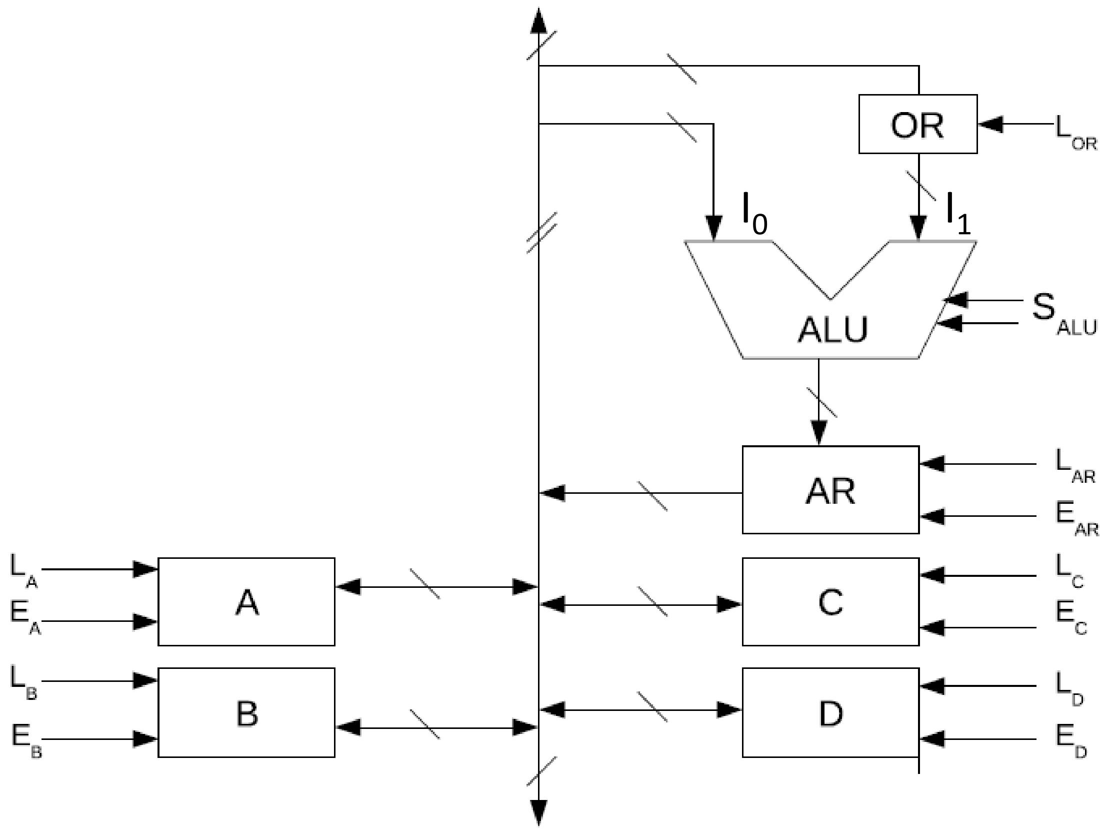
- The ALU has its left input connected to the bus and the right input to a register called OR or operand register, whose inputs are connected to the bus
- The output of the ALU is connected to the input of a register called AR or accumulator register, whose output is connected to the bus
- Every register has all the control signals of our generic multipurpose register



ALU on a bus

- What will happen if we activate the following controls: $E_{OR}, E_A, L_{AR}, SALU_{ADD}$
- Two enables are simultaneously active
- Does it cause conflict at the bus?
- No, since the output of OR is not connected to the bus, there will be no conflict
- The last term above indicates that the select lines of the ALU are set to the combination ADD
- The ALU performs addition of its inputs, which are the contents of OR register and register A
- The sum is written to AR register

$$AR \leftarrow A + OR$$



ALU on a bus

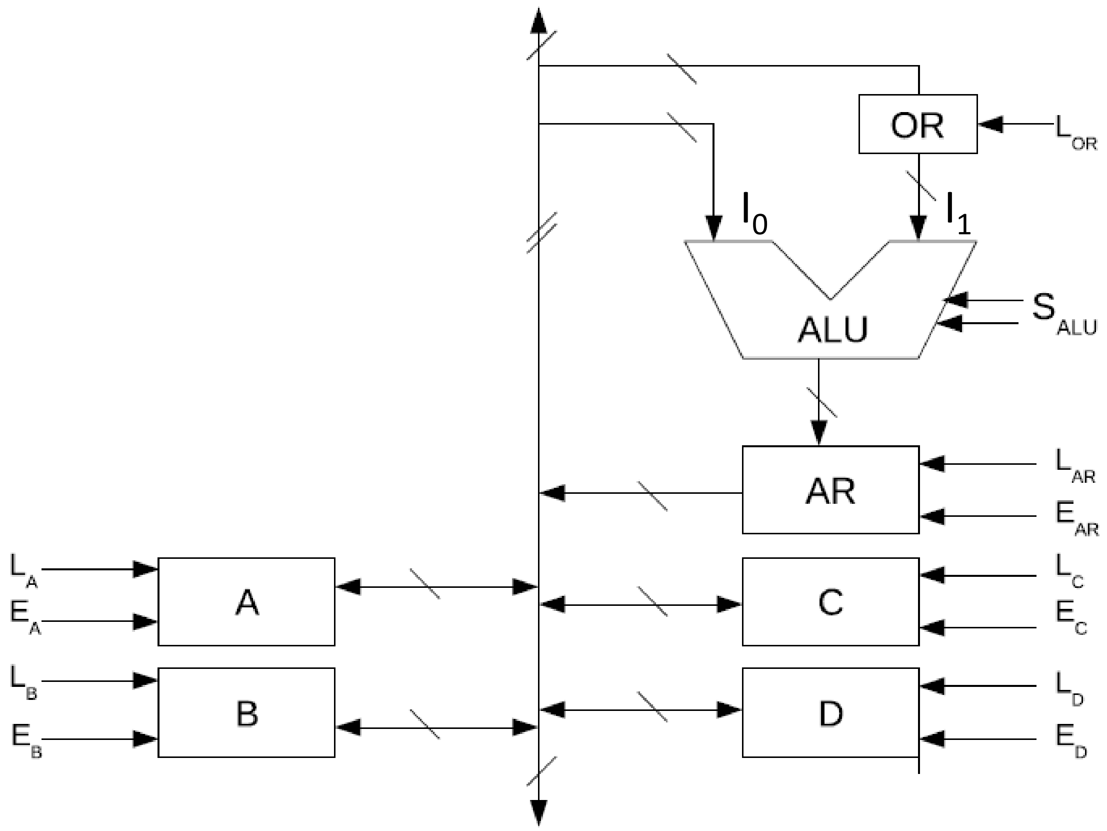
- Consider this multiclock instruction:

  Ck 10: E_A, L_{OR}

  Ck 11: E_B, E_{OR}, L_{AR}, $SALU_{SUB}$

  Ck 12: E_{AR}, L_C
- What does the 3 clock cycle combination achieve?
- The contents of A are copied to OR in cycle 10
- The contents of OR are subtracted from the contents of B and the result is loaded to AR in cycle 11
- The contents of AR are copied to register C in cycle 12

$$C \leftarrow B - A$$



ALU on a bus

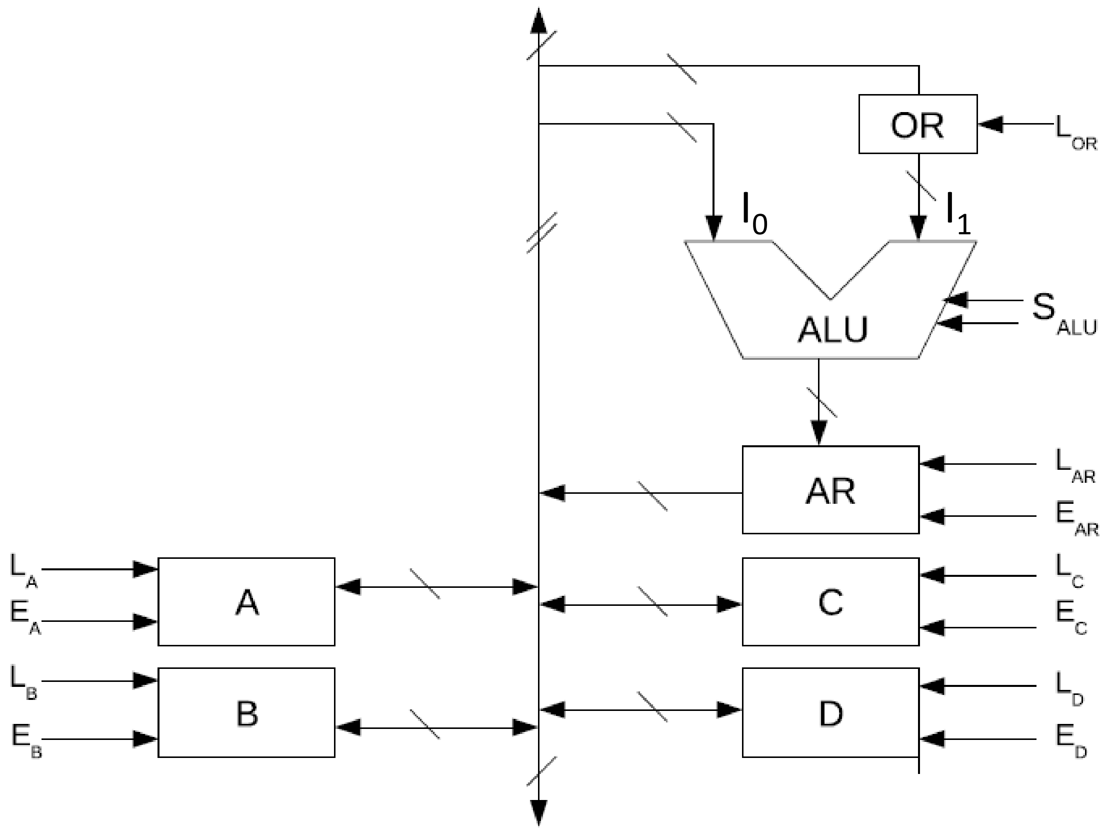
- It is easy to see how addition, subtraction and logical AND with any combination of 2 registers as input and any register as output can be implemented using a very similar 3-cycle sequence of combinations of signals
- Let us see if we can write control signals for this instruction:

$D \leftarrow C \text{ AND } D$

Ck 1: E_C, L_{OR}

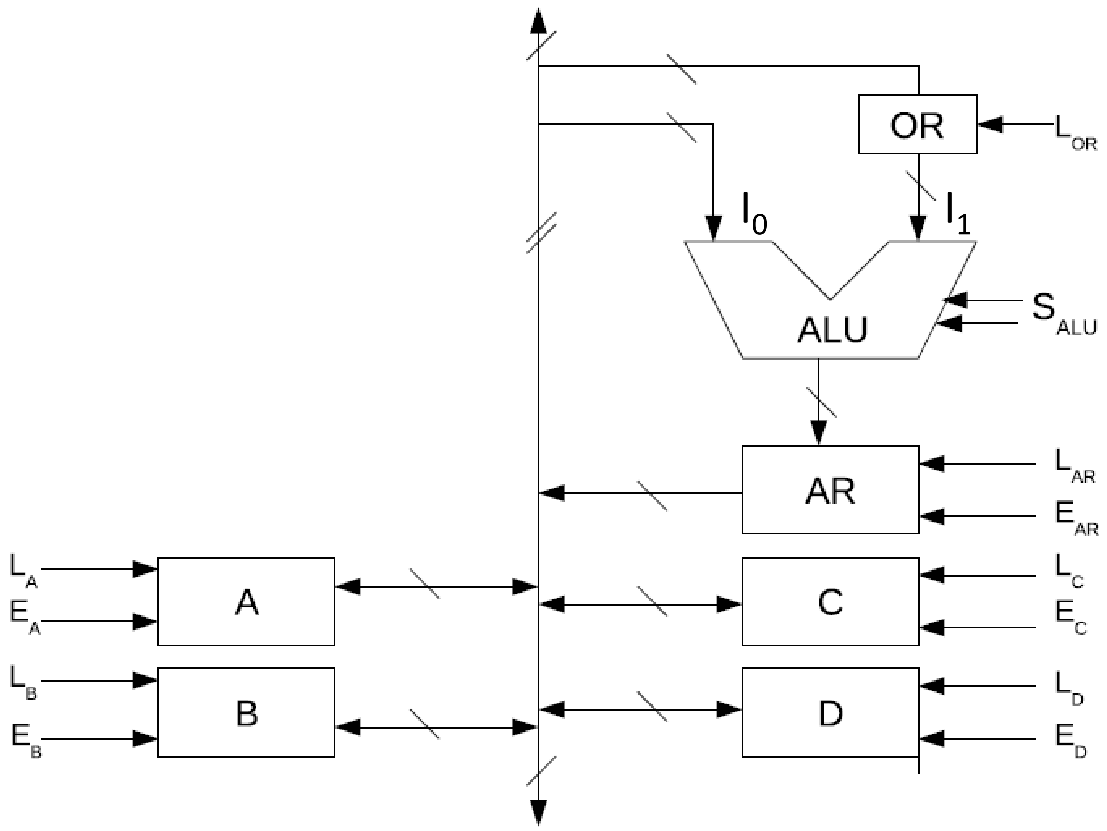
Ck 2: E_D, E_{OR}, L_{AR}, $SALU_{AND}$

Ck 3: E_{AR}, L_D



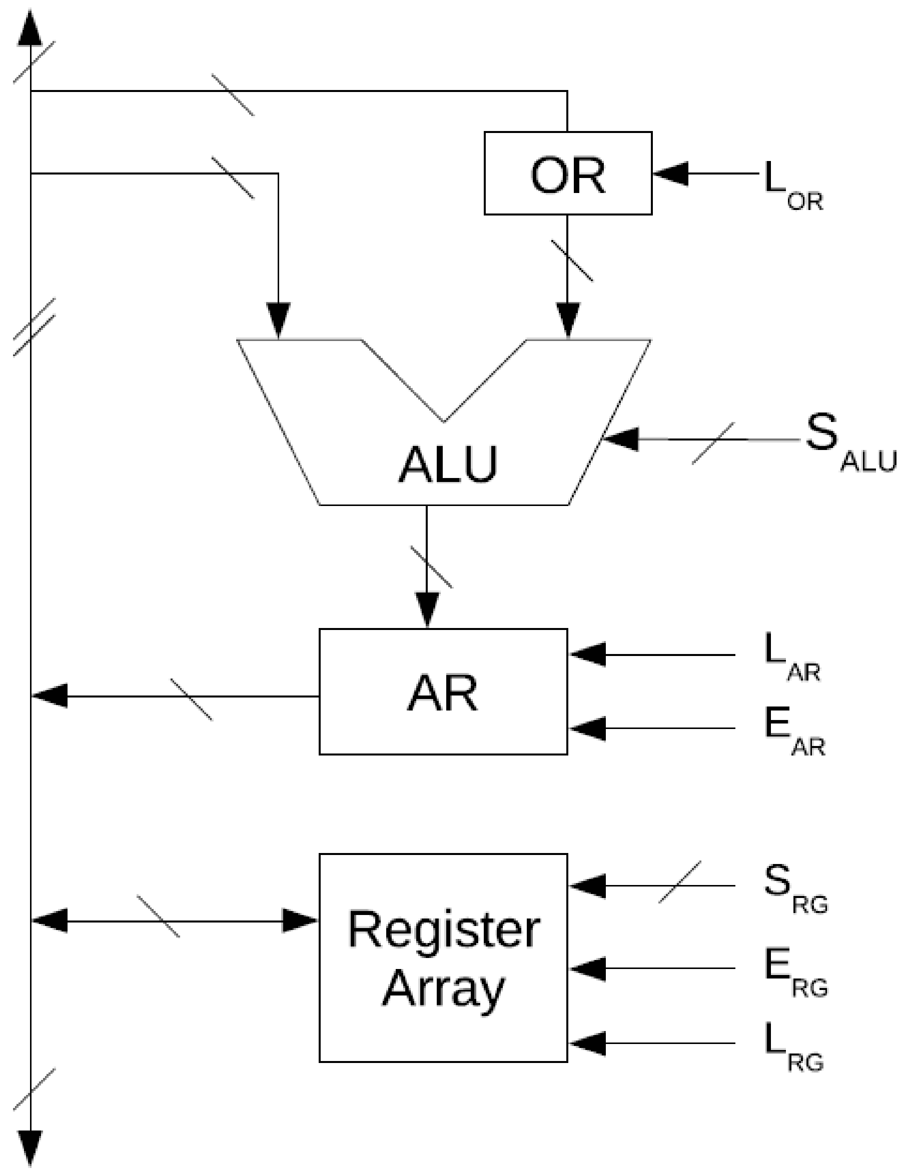
ALU on a bus

- It is clear that we can make the hardware perform several steps by carefully selecting the control signals to different units that are active in each clock cycle
- We can also get larger “operations” implemented using multiclock sequences of such combinations
- Each such clock cycle is typically referred to as a microcycle, which is the basic time unit in which something happens within the processor



Enhanced singlebus architecture

- We can consolidate the registers into a register array or a register file
- These are numbered R_0 through R_{11}
- The register file has a single enable input E_{RG} and a single load L_{RG} and 4 select lines S_{RG}
- The select lines identify which register of the file is being operated on, with enable or load controlling the action performed on it
- The new design of the register array needs only 6 control signals – 4 for select and 2 for enable/load – as opposed to 22 that would be needed were each register to have its individual enable and load signals
- Some generality is, however, lost as only one register can be selected for writing


OR
L_OR
ALU
S_ALU
AR
L_AR
E_AR
Register Array
S_RG
E_RG
L_RG