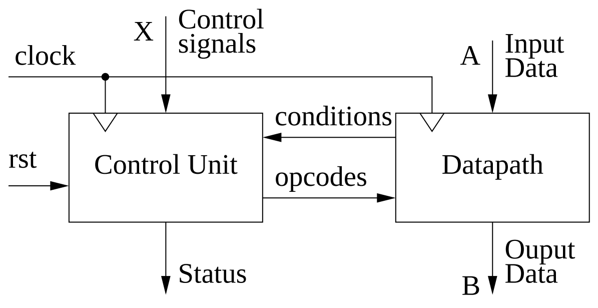
12 Simple serial arithmetic processor

- A processor is a digital device that implements either general or specialised algorithm described by a multi-step procedure.
- Structurally, a processor consists of the **datapath** and the **control unit** .

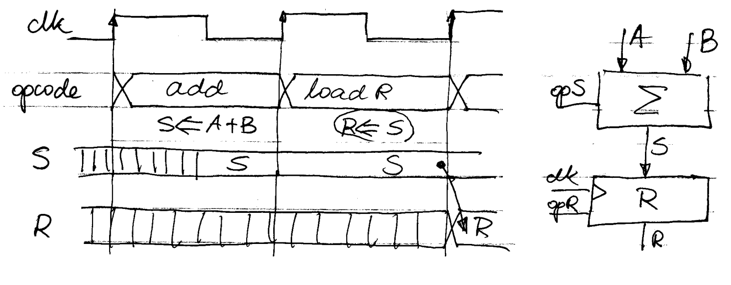


- Both parts of the processor are synchronised with the same clock signal.
- The control unit receives two type of signals: **control signals** like “start of the processing, or “ input data ready”, etc, from the outside world and **conditions** from the datapath.
- The control unit generates two types of signals: opcodes to blocks of datapath, and

  status signals, e.g. “ready” to the outside world
- The datapath processes input data into output data.

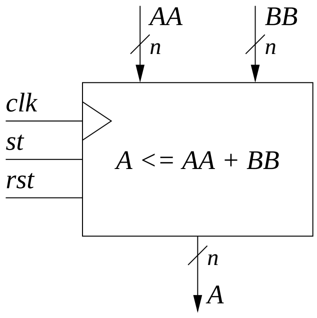
- A typical datapath consists of
  - registers to store variable of the algorithm
  - ALU and other combinational blocks that implement operation (operators) of the algorithm
  - Step counters that implement the loop counters.
- A typical control unit is a sequential state machine, typically called an **algorithmic state machine**, that implements the structure of the algorithm and

  generates opcodes to the blocks of the datapath and possibly status signals to the outside world.
- In a typical processor all sequential blocks are synchronised by the same clock signal.
- Therefore, the duration of each processing step, equivalent to one state of the control unit is fixed.
- Assuming that an opcode for a **sequential block** like a register (e.g. load R) is generated during the current clock cycle, the result of the operation is known during the **next clock cycle**.
- **Combinational circuits**, like adders, form the result during the **current clock cycle**.



12.1 Example: A bit-serial adder

- As an example of a very simple processor let us consider a problem of a serial addition of two n-bit 2's complement numbers
$$A = A + B$$
- We start with specification of the top level component (entity) and the algorithm to be performed.
- The arguments of the addition and the results are n-bit numbers
- In addition we specify a reset signal, rst , and a start signal st .

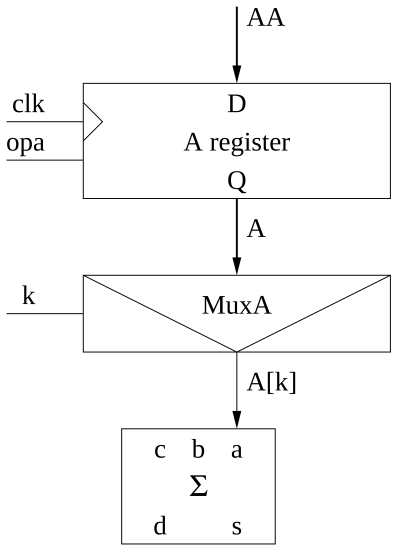


- Subsequently we specify the concept of the algorithm in a top-level pseudo-language that can be of the following form:

```
$A = AA;\ B = BB;\ c = 0;$
for $(k = 0\ ;\ k \le n\ ;\ k{+}{+}\ )\ \{$
    $(c, A[k]) = A[k] + B[k] + c;$
$\}$
```

- Variables on the left-hand side of the equations are normally need to be stored in registers, hence, we infer that two n-bit registers A and B, and a 1-bit carry register, c are needed.
- Since the addition is performed in a serial fashion, a 1-bit adder is needed.
- Finally, a step counter is needed to count the number of speps.

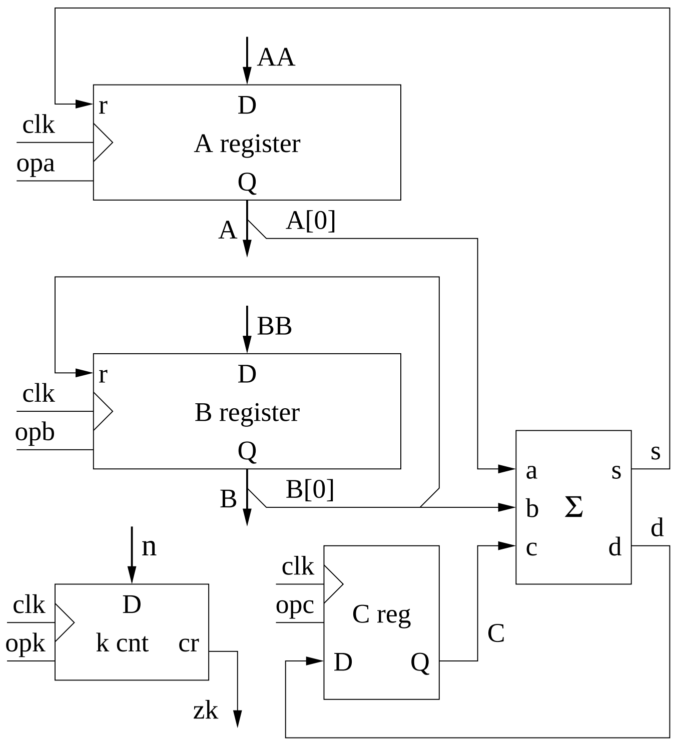
- We first consider a way of extracting k-th bit from a register as required by the algorithm.
- A possible solution is based on using a n-to-1 selector/multiplexer driven by the loop variable
- Output from the selector is a required single bit A[k]
- However, there is a simple way of avoiding a selector/multiplexer by shifting the register contents at each step so that the required bit is in a fixed position, say the least significant one, A[0]
- Details are presented in the following modified pseudo-code:



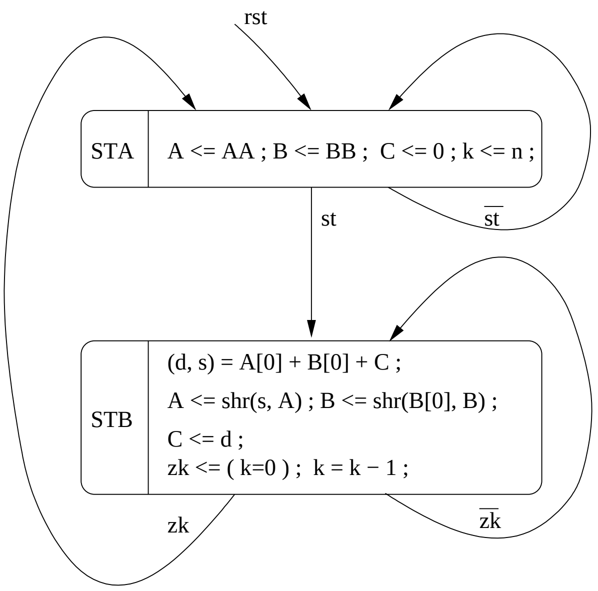
```
$A = AA;\ B = BB;\ c = 0;$
for $(k = 0\ ;\ k \le n\ ;\ k{+}{+}\ )\ \{$
    $(d, s) = A[0] + B[0] + c\ ;$
    $c = d;\ A = \text{shr}(s, A);\ B = \text{shr}(B[0], B);$
$\}$
```

- We shift registers to the right inserting at the vacated position of the A-register the sum s, and $B[0]$ at the vacated position of the B-register, thus implementing cyclic right shift/rotation.

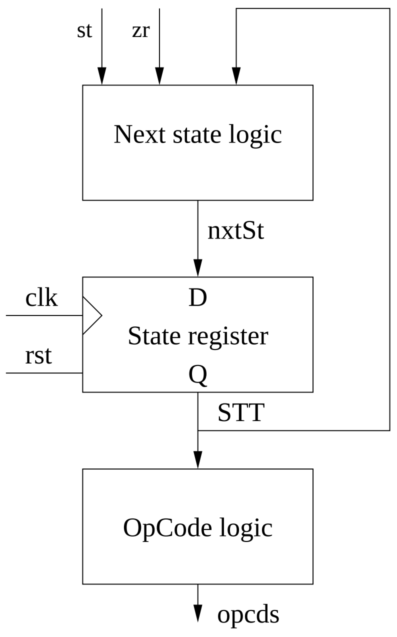
- A complete **datapath** of the serial adder looks as follows:
- Note that the least significant bits of the A and B registers are being added by the serial adder together with the current carry bit stored in a 1-bit register C.
- The least significant bit of the register B is connected to its serial shift-right input. As a result its contents is restored after n cyclic shifts (rotations).
- The step counter loads the number of steps to be performer n and counts down to 0 when the signal **zk** is generated.
- For each register and counter, we have to specify the **set/table of operations**, typically some of operations like: hold (nop), load , shift , count , etc.
  Details have been discussed in sec. 10.



- The **control unit**
- A good way to start is to prepare a flow-chart of operations in a form of a **state transition diagram** which describes details of the algorithmic state machine.
- Note that the state transition diagram consists of only two states, so that the related state register, STT, will be 1-bit.
- The reset signal rst transfers the state machine to its initial state STA
- The external control signal, st (start), has been added to initialize the addition operation.
- When st = 0 we stay in the initial state STA waiting for the start signal st = 1
- At the rising edge of the clock at the end of the STA sate registers and counters are initialised as indicated.
- The main operation is performed in the state STA wher the sum is formed bit by bit.
- The step counter generates the signal zk indicating the end of summation.


rst
STA
A <= AA ; B <= BB ; C <= 0 ; k <= n ;
st
$\overline{st}$
STB
(d, s) = A[0] + B[0] + C ;
A <= shr(s, A) ; B <= shr(B[0], B) ;
C <= d ;
zk <= (k=0) ; k = k − 1 ;
zk
$\overline{zk}$

- The **structure of the control unit** is typical and consists of
  - the D flip-flops referred to as the **state register** (1-bit in this case),
  - The "Next State" combinational circuit, and
  - The "OpCode logic" that produces the opcodes to all blocks of the datapath.
- In order to properly interpret the state transition diagrams you have to remember that
  - all **sequential** circuits are **updated at the end of the current clock cycle**, whereas
  - **combinational** circuits form results **during the current clock cycle**.
- This becomes obvious when you analyze the time waveforms.



Generation of the opcodes

- It can be observed that each datapath block performes only two operations: one in the STA state, the other in the STB state.
- Therefore the **OpCode** logic is trivial and all **opcodes** are equal to STT , that is the output from the 1-bit state register.

Next state logic

The next state logic can be derived from the following excitation table:

st	zk	STT	nxtSt
0	–	0	0
1	–	0	1
–	0	1	1
–	1	1	0

$$\text{nxtSt} = \overline{\text{STT}} \cdot \text{st} + \text{STT} \cdot \overline{\text{zk}}$$