Computing without computers

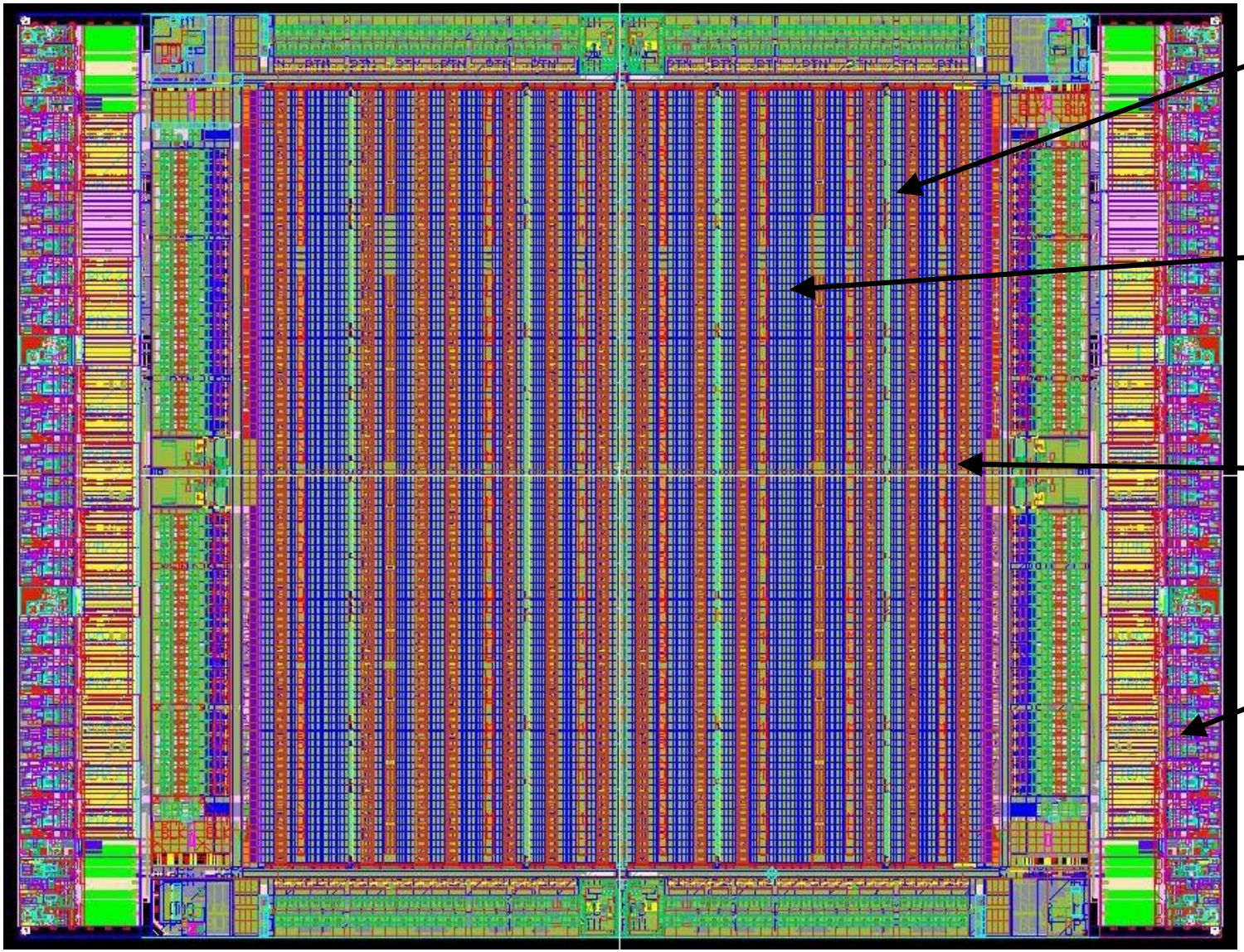
(see notes on Hardware Compilation)

- computers are too slow, too bulky, consume too much power and energy
  - get rid of fetch/decode stage? ↓CPI? ↓clock delay?
- compile programs directly into hardware
  - hardware is parallel, hence fast
  - distributed control: no program counter
  - variables → registers, expressions → combinational circuit
  - applications: multimedia, communication, robotics
  - but hardware is inflexible?

Field-Programmable Gate Array (FPGA)

- a matrix of programmable elements, including: logic blocks, storage elements, interconnections
- 100K to 10000K gates: system-on-a-chip, with complex functions or instruction processors e.g. ARM
- clock speed up to 800MHz (50-300MHz common)
- some can be reprogrammed within milliseconds (even partially reprogrammable)
- off-the-shelf parts, see altera.com, xilinx.com, maxeler.com approaching speed of hardware, flexibility of software
- stand-alone or used with microprocessor
- 2015: Intel bought Altera, an FPGA company for US$16.7B
- compilation method: basis for Handel-C tools
  - over 400 commercial seats, 600 academic seats

Stratix IVGX 230: mid-size FPGA



DSP Blocks (coarse grain)

Adaptive Logic Modules (fine grain)

RAM Blocks (M9K & M144K)

High Speed Serial Interfaces:

eg connect multiple FPGAs

(source: V. Betz)

Occam and Handel-C

- based on CSP notation; commercial version: Handel-C
- primitive processes:
  skip stop v := e c ! e c ? e delay

  var, expr, chan output, chan input
- composite processes:
  SEQ P Q PAR P Q IF B P WHILE B P
  ALT $B_i P_i$ (conditional on input ready in B_i)
- channel joins two processes at any time

  one outputs
  the other inputs

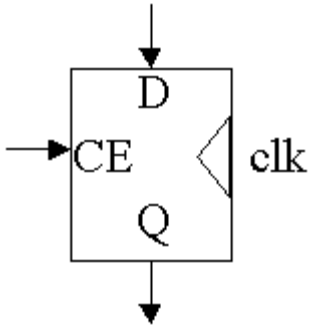
  CHAN C:
  PAR
  C ! E
  C ? *X*

  = *X*:= E

  not examined
- produce data processors and instruction processors
- can also compile declarative programs into hardware

Expression and variable

- expression: implemented using combinational hardware
  - no combinational loops
  - ensure propagation delay shorter than clock period (usually)
- variable: implemented using D type flip-flop and a multiplexor (mux)
  - mux allows conditional update of D type flip-flop (when CE = 1)

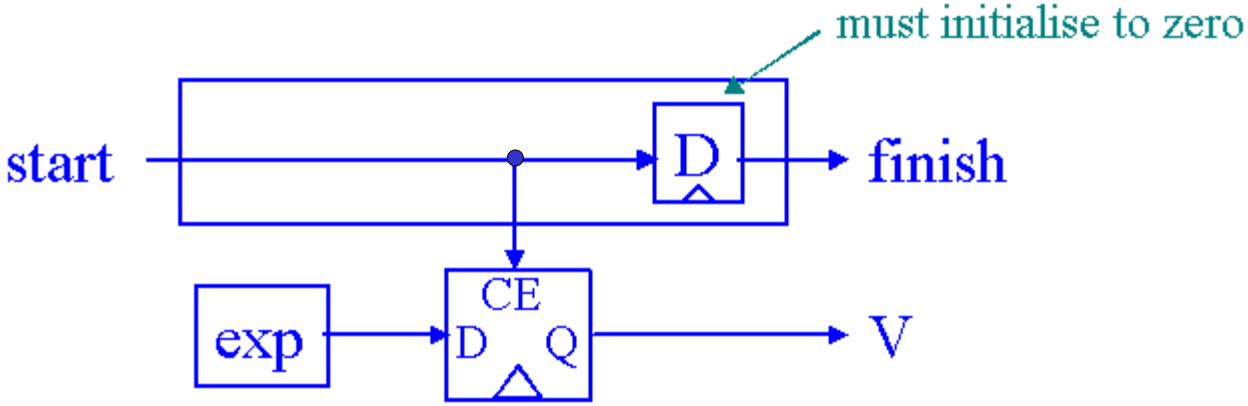


Control strategy

- use a single-cycle pulse (the token) to activate a control circuit (e.g. assignment)
- environment supplies a token to start (s), the token reappears at finish (f) when the statement completes execution
- control assumption: a control circuit does not create or destroy a token, provided that the environment does not offer a second token before the first token reappears
- the assignment circuit can accept token every cycle

Assignment statement v := exp

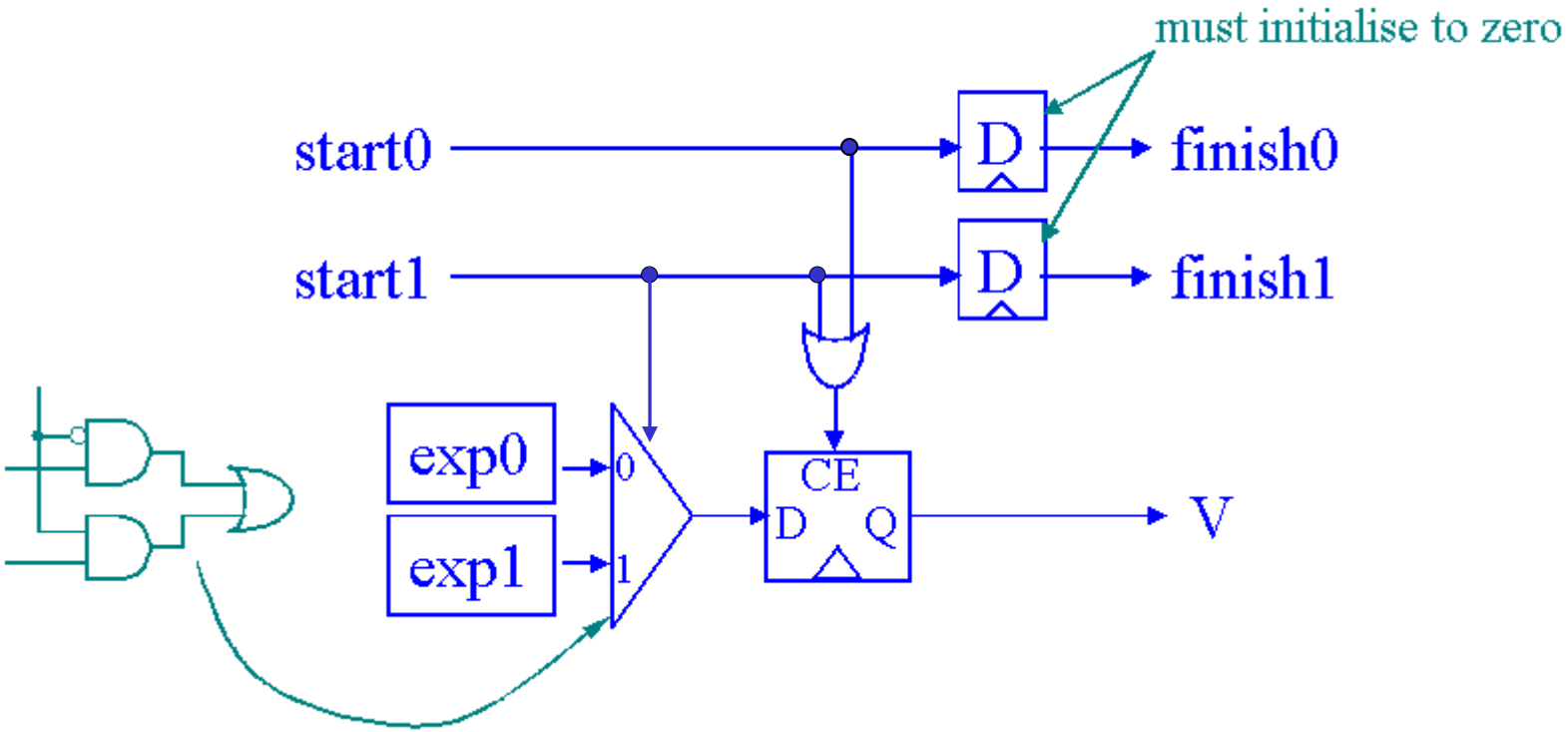
- provide control circuit to signal assignment, statement is completed after n cycles
- n depends on propagation delay of exp, usually n=1



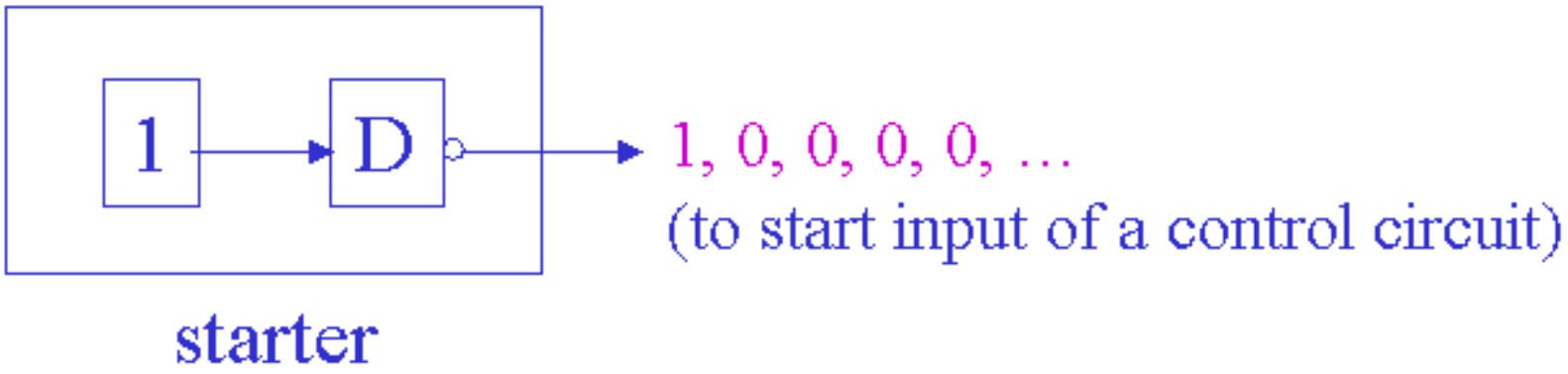
- concurrent assignment: v0, v1 := exp0, exp1
  use the same control circuit for the CE inputs
  for v0, v1 hardware

Assignment: general case

- need multiplexing to support assigning different values to a variable at different times
  e.g. V := exp0; … ; V := exp1



Initialisation

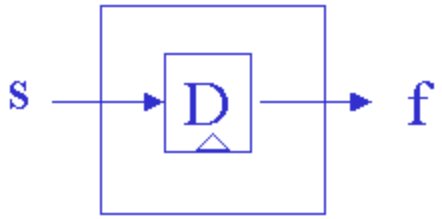


- provides the initial token to get system going
- assumes D-latch initialised to zero

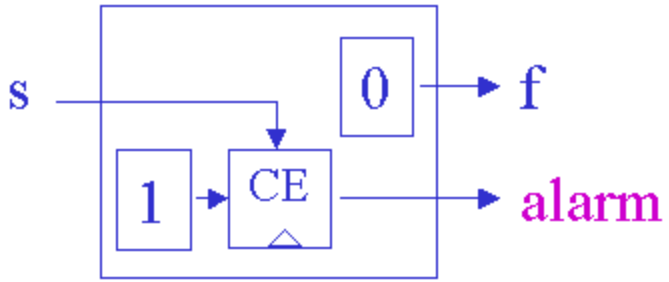
Skip, delay and stop

- skip: does nothing,
  terminates immediately

- delay: does nothing,
  terminates after 1 cycle



- stop: never terminates
  perhaps raise an alarm

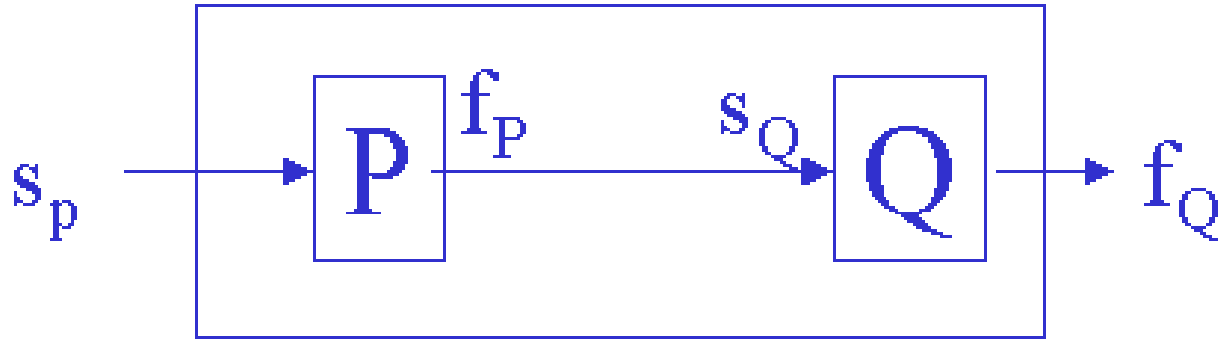


Sequential composition

SEQ
P
Q

or

P ; Q

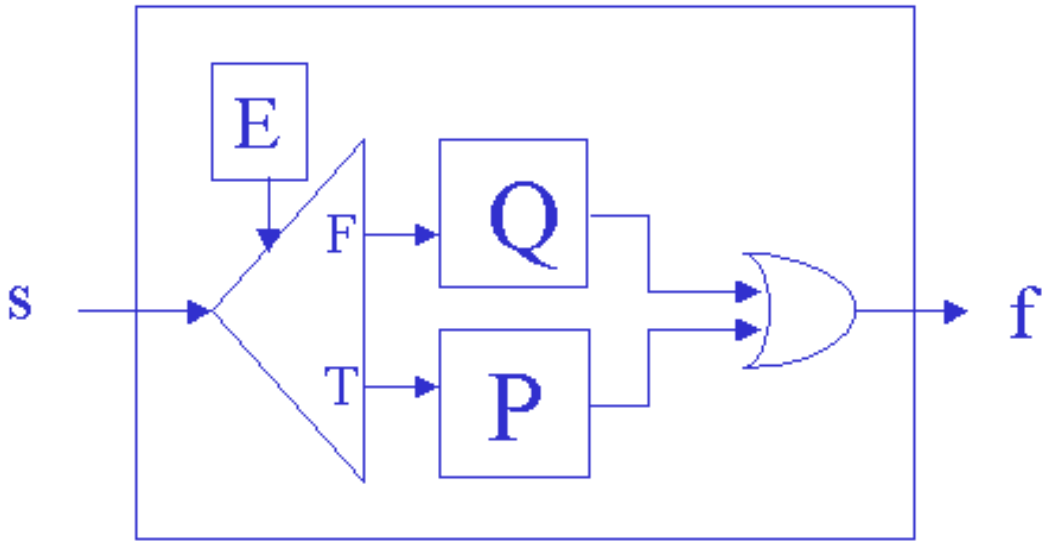


- if P, Q satisfy the control assumption, then P;Q satisfies the control assumption
- only control signals are shown in the diagram

Conditional

IF
E
P
TRUE
Q

if (E) then P; else Q;



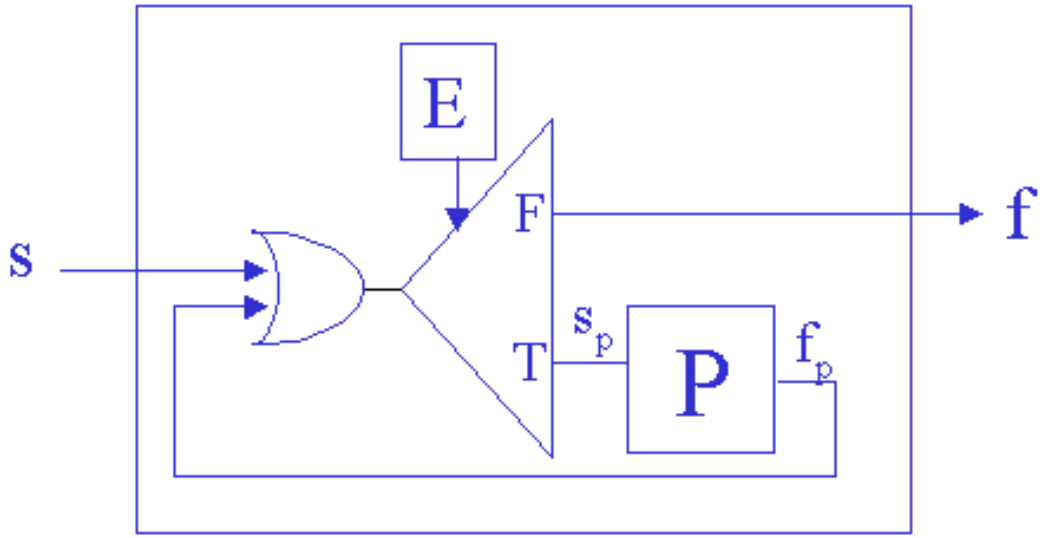
- either P or Q (not both) should get the token

Iteration

WHILE E
P

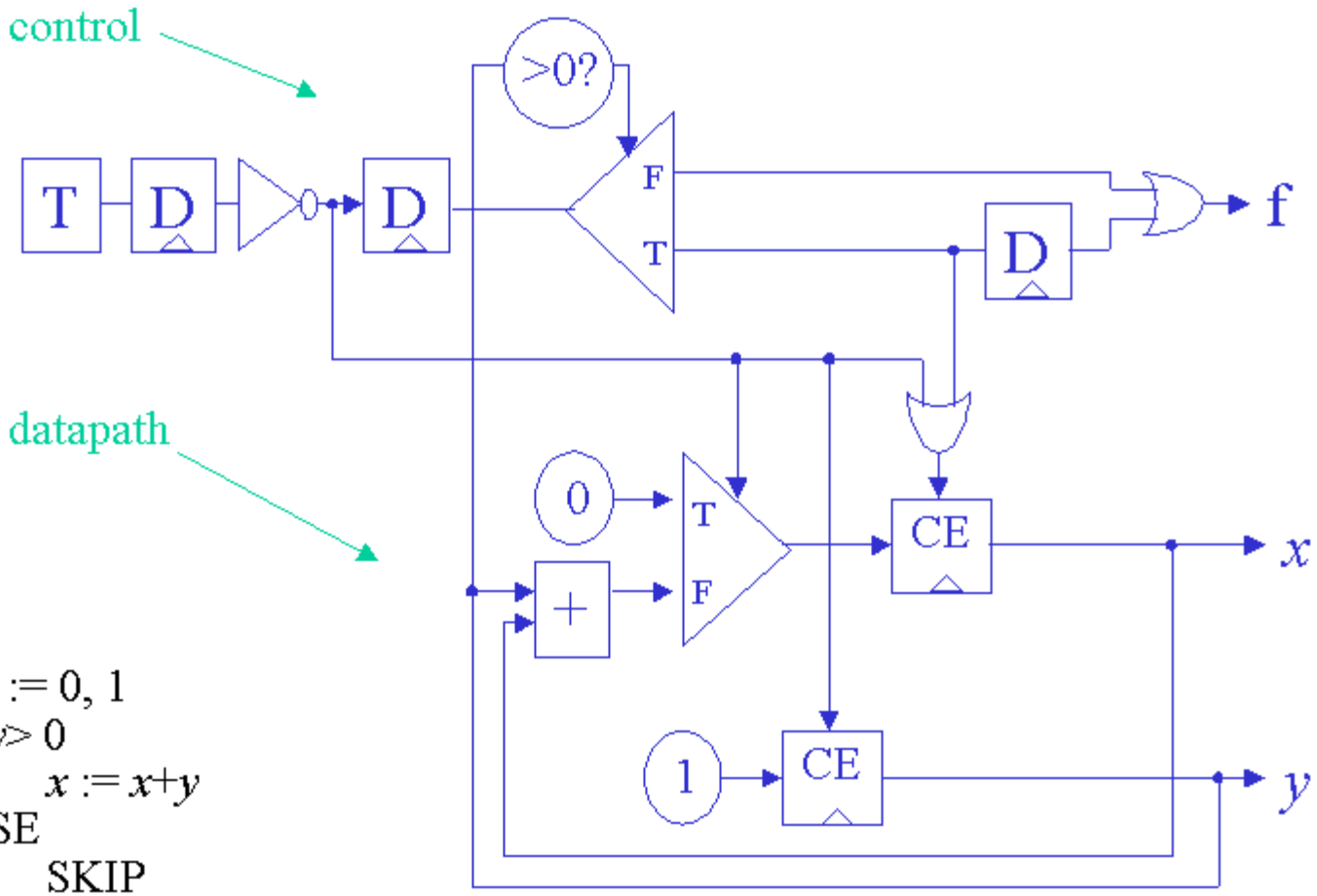
or

while (E) P



- what happens with the program
  while (1) skip?

Example: simple data processor

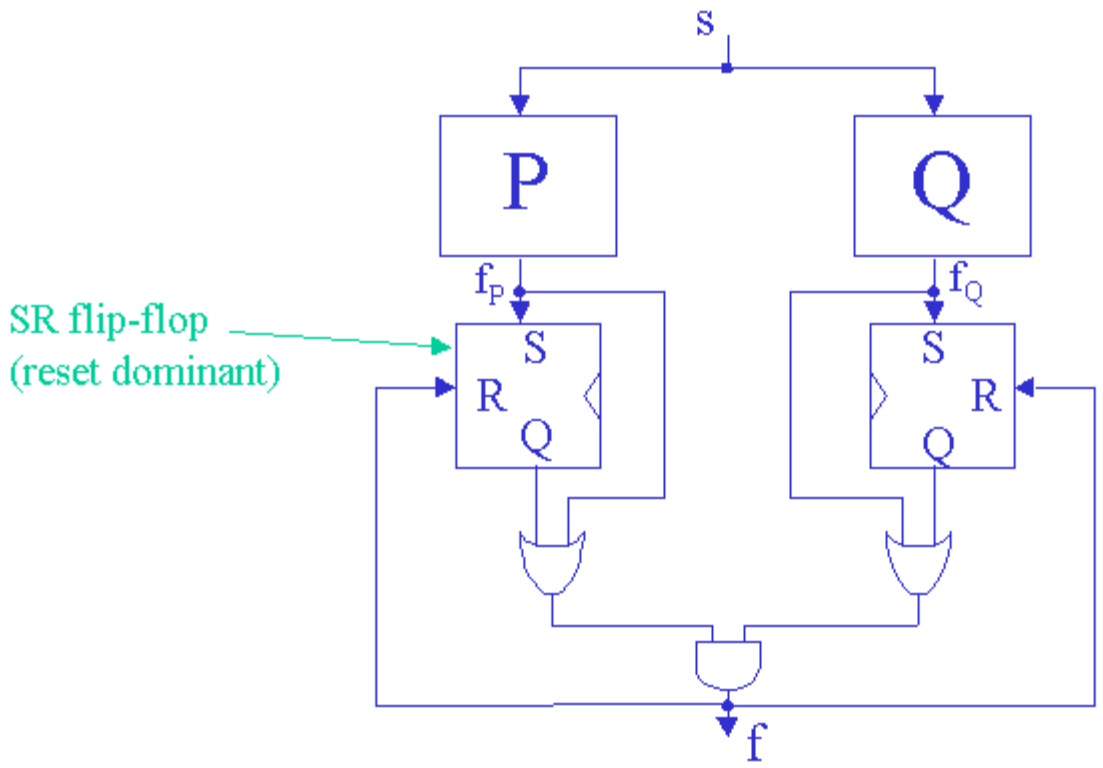



```
SEQ
    x, y := 0, 1
    IF y> 0
        x := x+y
    ELSE
        SKIP
```

The par construct

- adding to parallelism in expression evaluation and concurrent assignment
- invokes program fragments in parallel, e.g.:
  - no variables in common, operating independently
  - communicate with each other via channels
  - share variables with each other, with restrictions
- terminates when all program fragments have terminated
- control circuit should have no time overhead:
  - start parallel execution immediately
  - terminate immediately when appropriate

Control circuit for par



- PAR
  P
  Q

  or

- par { P;
  Q
  }

- SR flip-flop: once set, remains set until R input high
- or-gate: allows immediate termination