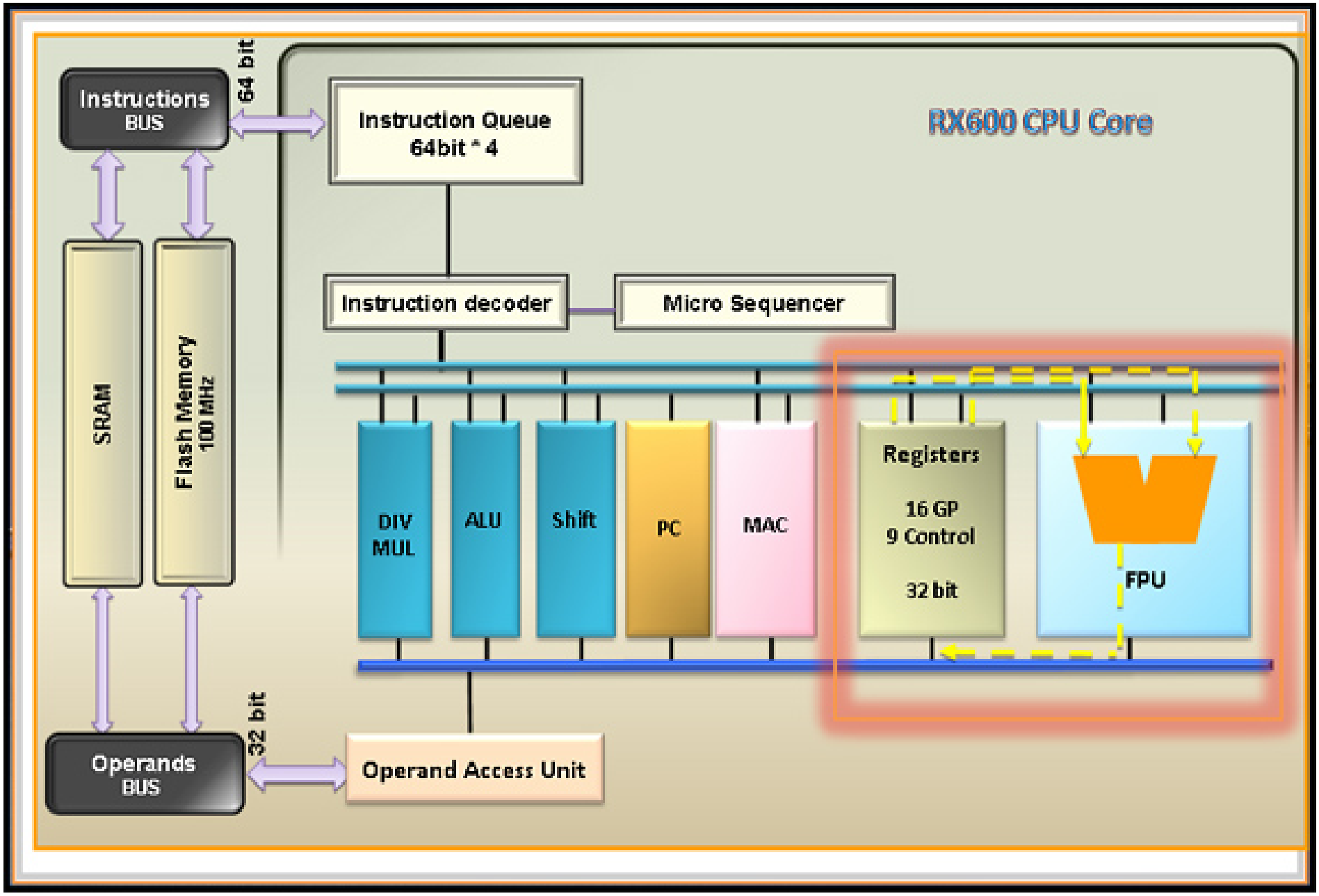
Instruction Execution

- Serial or Parallel Computation ?
- Synchronisation
- Microinstructions

Control Path

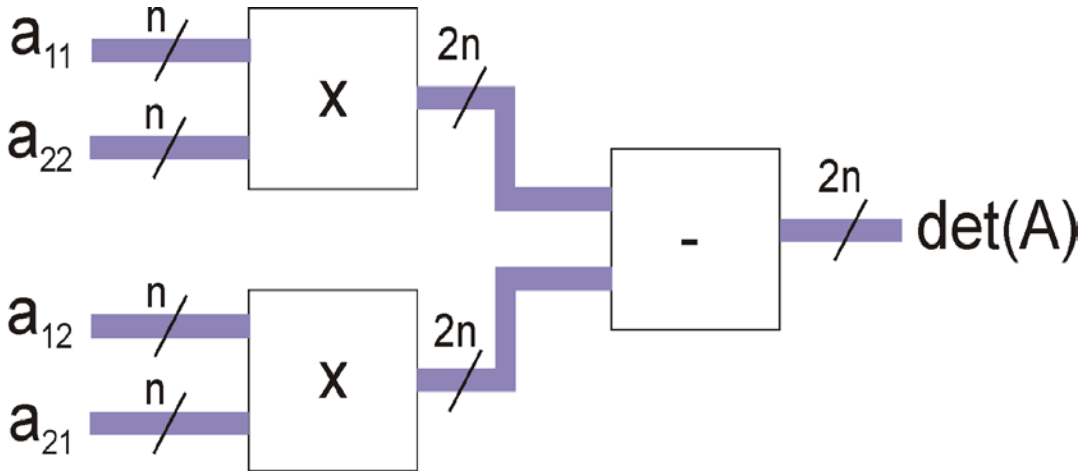
- Used to control the rest of the μP
- Control is complex if the instruction set is complex and diverse
- To manage the complexity, Microcoding is often used
- Microcoding is when the each instruction is split into elemental steps and execution is undertaken by putting sequences of elemental steps together
- This is done by a Microcode Sequencer
- Essentially, a Microcode Sequencer is a very simple instruction processor - *within the μP itself*


Instructions BUS
64 bit
Instruction Queue 64bit * 4
RX600 CPU Core
SRAM
Flash Memory 100 MHz
Instruction decoder
Micro Sequencer
DIV MUL
ALU
Shift
PC
MAC
Registers
16 GP
9 Control
32 bit
FPU
Operands BUS
32 bit
Operand Access Unit

Parallel or Serial Computation ?

Consider the calculation of the determinant of a 2 x 2 matrix:

$$\det(A) = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11} \times a_{22} - a_{12} \times a_{21}$$



Parallel computation using combinatorial multipliers:
Very fast, limited only by logic delays.

The 2 × 2 determinant calculation requires two multiplications and one add.

How does the complexity increase if we want to calculate the determinant of a 3 × 3 or larger matrix?

We now know that multiplication is much more costly than addition, so we will look at the number of multiplications required:

3 × 3 requires 9 multiplications
4 × 4 requires 40 multiplications

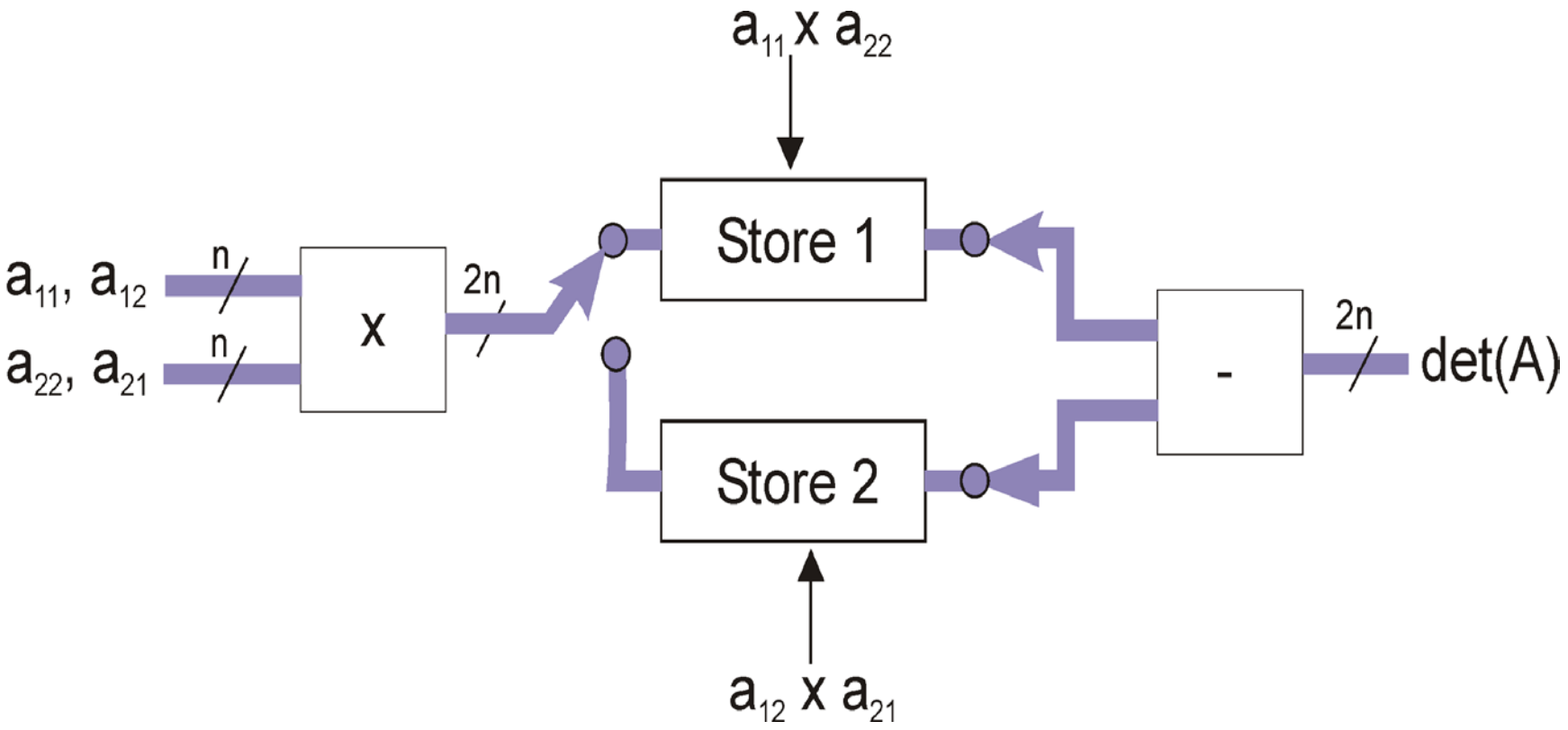
⋮

10 × 10 requires > 3.6 million hardware multipliers !

The parallel approach does not scale well with complexity.

A Serial Computer

The problem can be decomposed into a series of elementary operations.



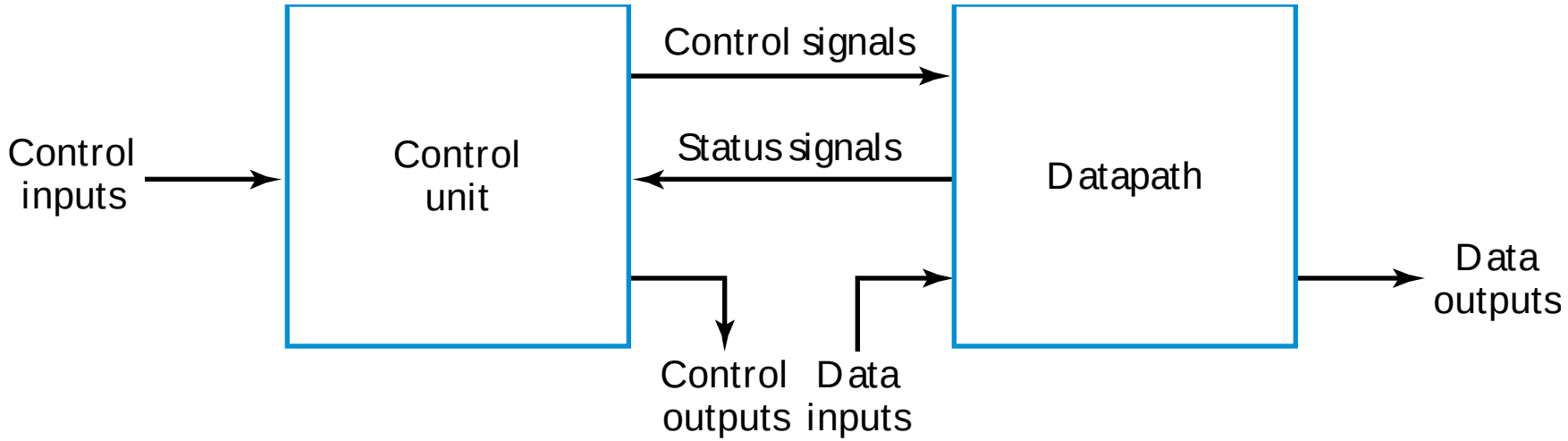
Advantages:

- Hardware re-use
- Flexibility

Disadvantages:

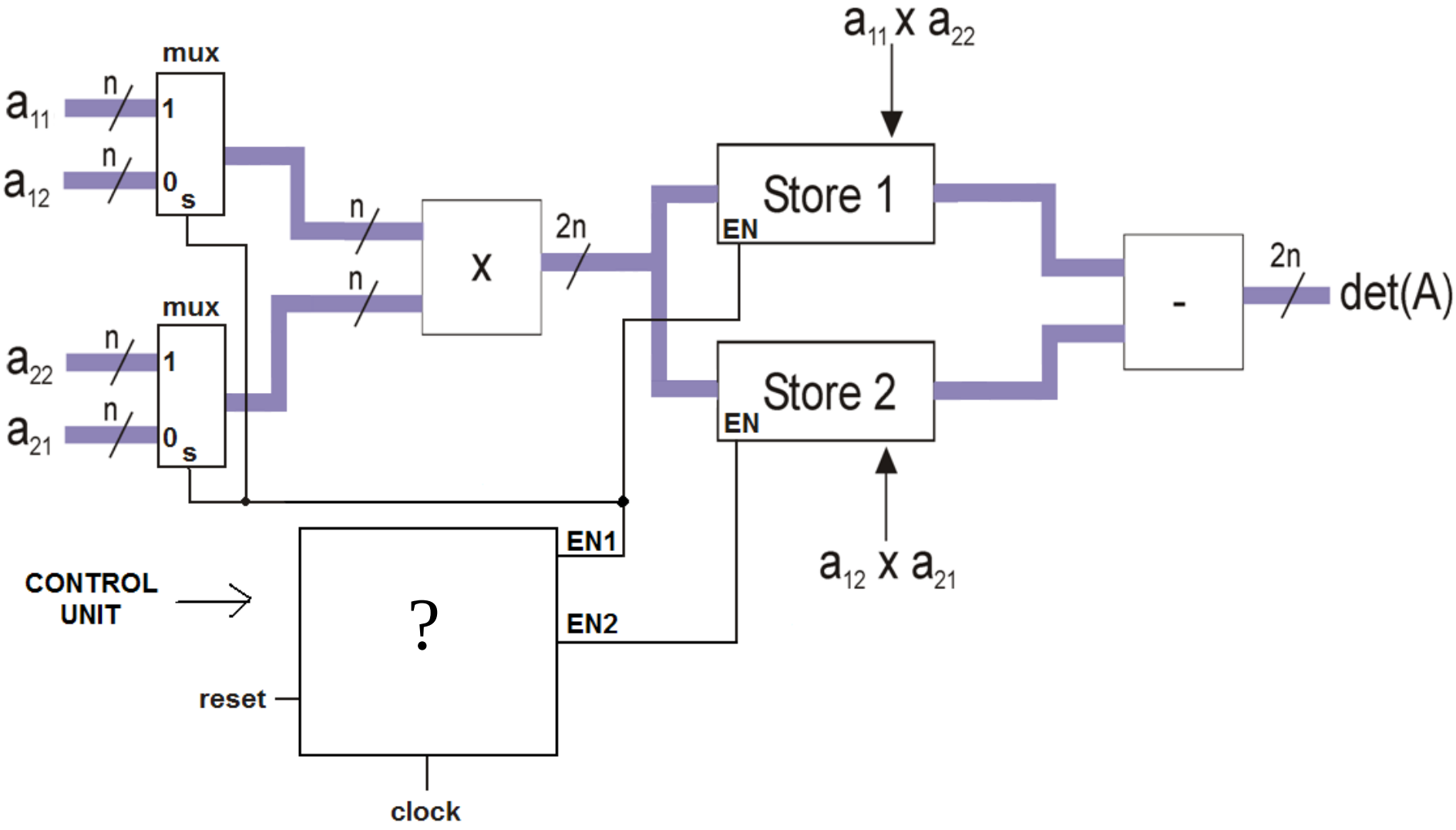
- Calculation will now be slower
- Operations must be carefully synchronised
- Complex control

Datapath and Control Unit

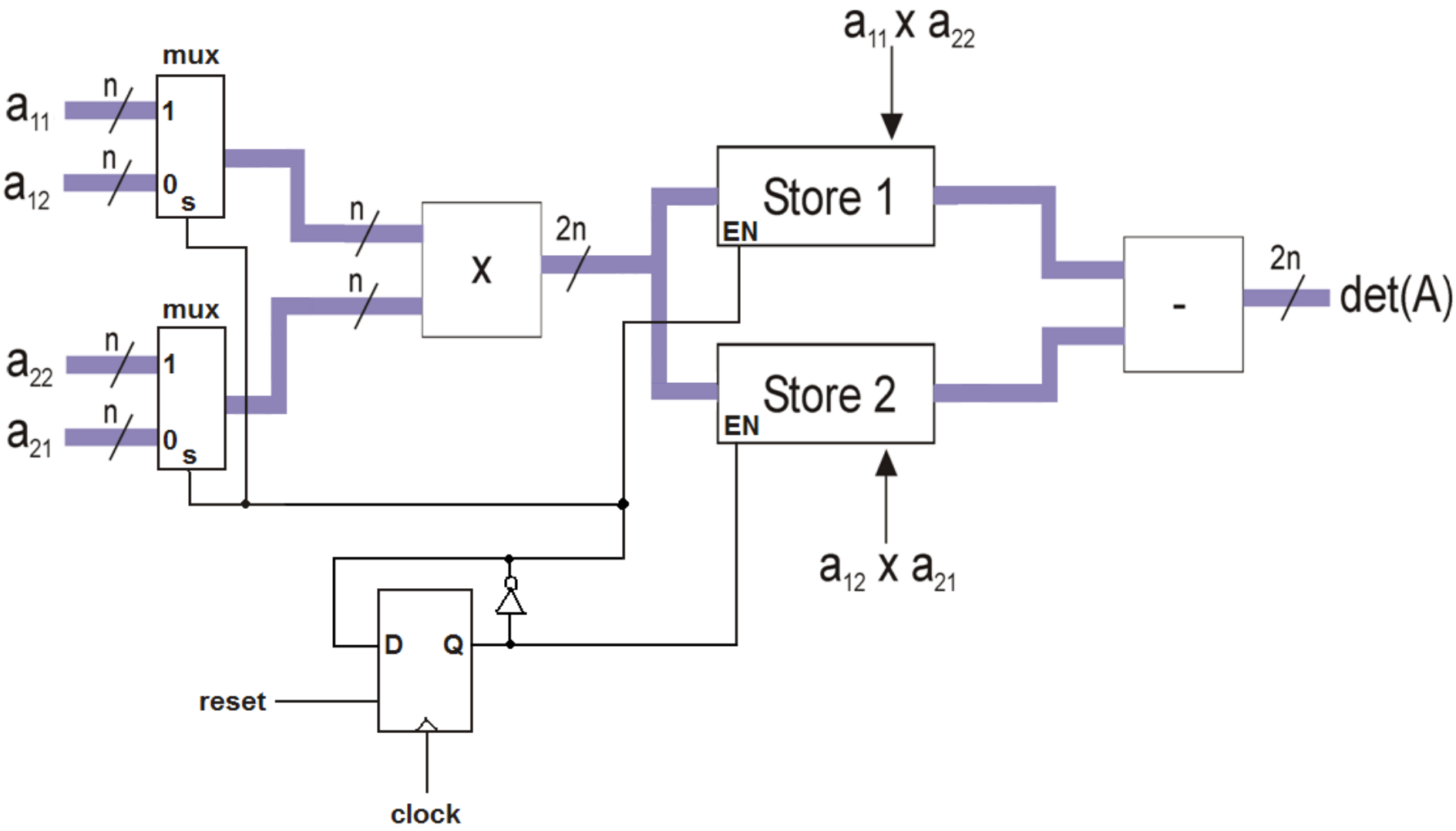


- Datapath: set of registers, functional units
- Register transfers performed on registers
- Control signals supervise the sequencing of the register transfers and the operations to be performed

Control using a Finite State Machine (FSM)



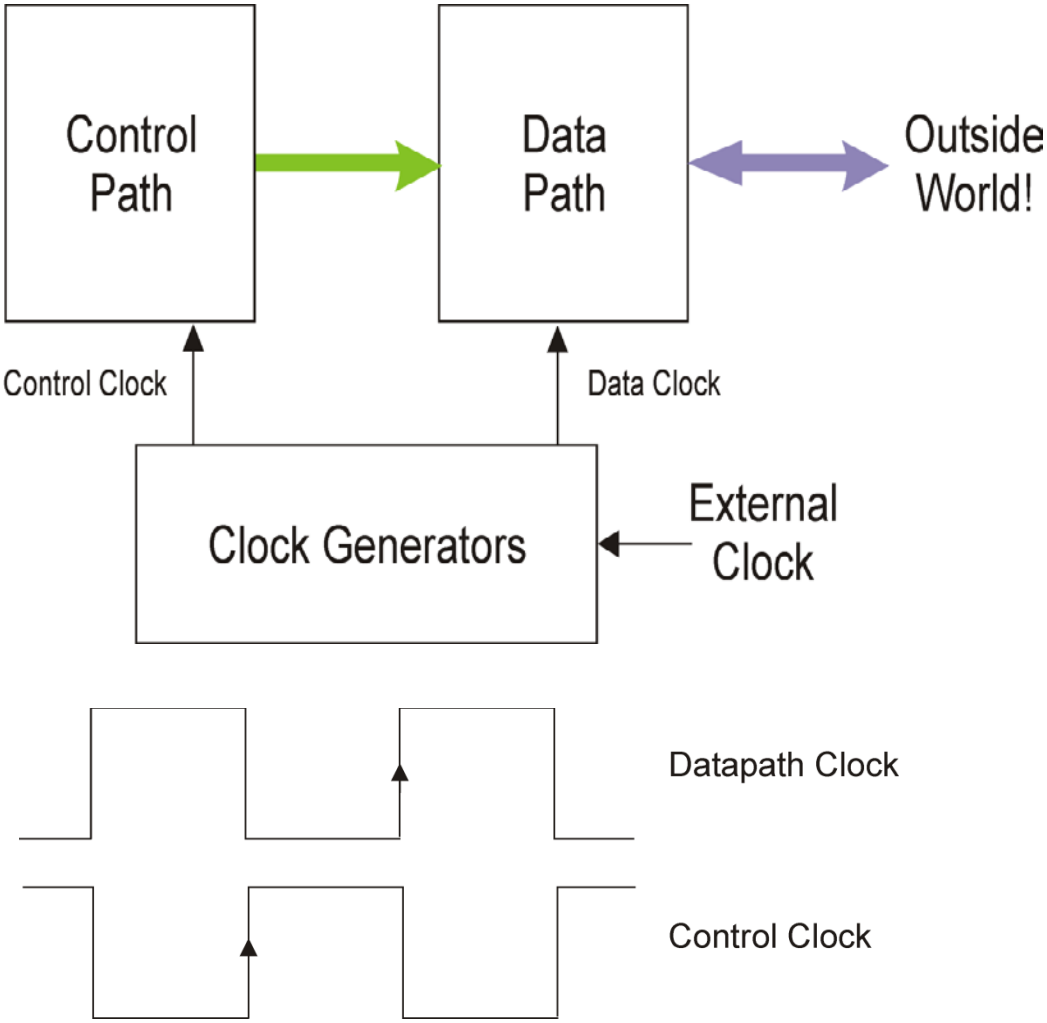
Control using a Finite State Machine (FSM)



Synchronisation

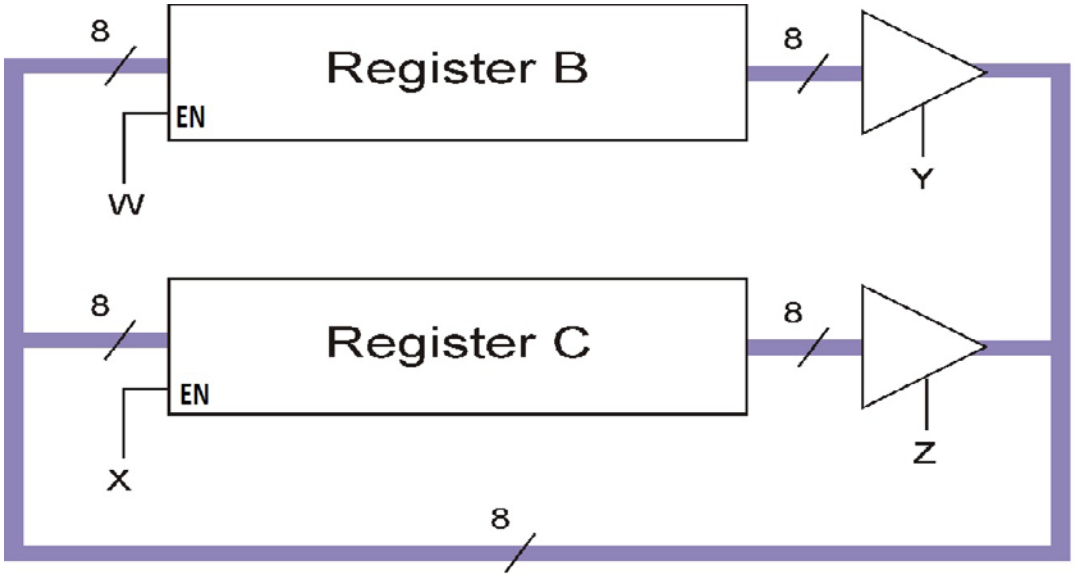
Two phase clock.

Data transfers are synchronised to the rising edge of a clock. Control signals must also be synchronised. In this method, a two phase clock is produced as shown.



Most modern synchronous circuits now use a single clock.

Register Control



Consider the more general case of the datapath registers. (clock omitted for clarity)

We have four, one-bit control signals: W, X, Y and Z.

To copy the contents of register B into register C:

set W = 0; X = 1; Y = 1; Z = 0 and then clock.

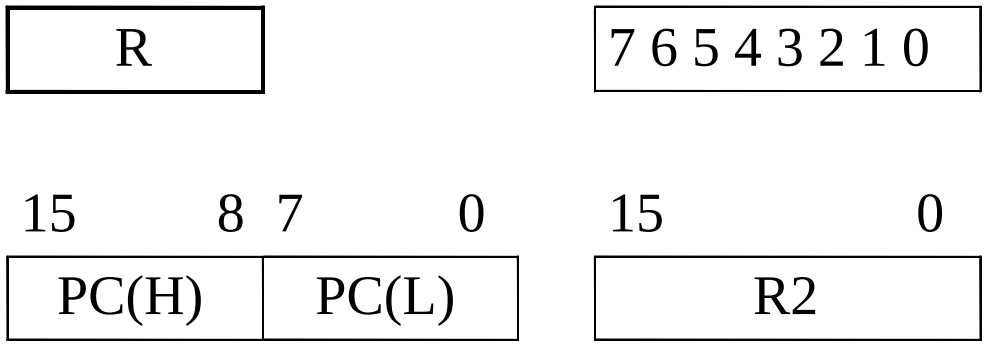
Control word:

W	X	Y	Z
0	1	1	0

Sequence of Events:

1. Apply a control word
2. Transfer data
3. Apply next control word

Register Transfer Operations



Letters and numbers – denotes a register (ex. R2, PC, IR)

Parentheses ( ) – denotes a range of register bits (ex. R1(1), PC(7:0), PC(L))

Arrow (←) – denotes data transfer (ex. R1 ← R2, PC(L) ← R0)

Comma – separates parallel operations

Brackets [ ] – Specifies a memory address (ex. R0 ← M[AR], R3 ← M[PC])

Micro-operations

Low-level instructions used to implement complex machine instructions

Logical Groupings:

- Transfer - move data from one register to another
- Arithmetic - perform arithmetic on data in registers
- Logic - manipulate data or use bitwise logical operations
- Shift - shift data in registers

Arithmetic operations

- + Addition
- – Subtraction
- * Multiplication
- / Division

Logical operations

- $\vee$ Logical OR
- $\wedge$ Logical AND
- $\oplus$ Logical Exclusive OR
- $\overline{\ }$ Not

Conditional Transfer

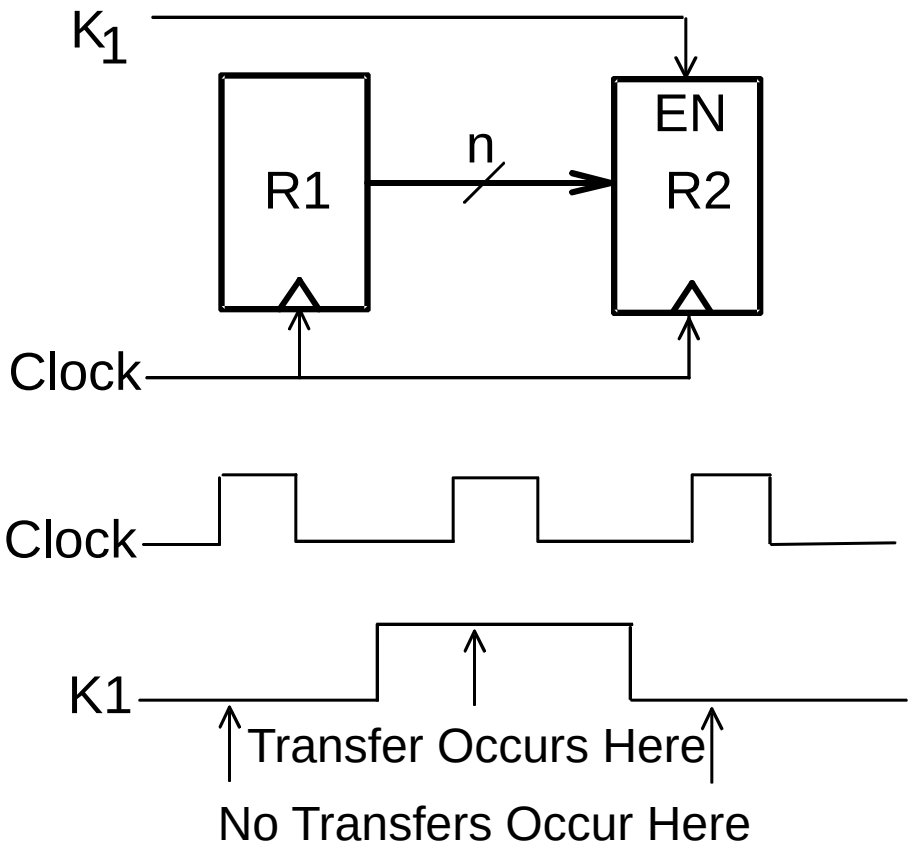
The conditional statement:

if (K_1 =1) then (R2 ← R1)

is shortened to

K_1: (R2 ← R1)

where K_1 is a control variable specifying a conditional execution of the micro-operation.



Example Micro-operations

Add the content of R1 to the content of R2 and place the result in R1.

$$R1 \leftarrow R1 + R2$$

Multiply the content of R1 by the content of R6 and place the result in PC.

$$PC \leftarrow R1 * R6$$

Exclusive OR the content of R1 with the content of R2 and place the result in R1.

$$R1 \leftarrow R1 \oplus R2$$

Example Micro-operations

Take the 1's Complement of the contents of R2 and place it in the PC.

$$PC \leftarrow \overline{R2}$$

On condition K1 OR K2, the content of R1 is Logic bitwise Or'ed with the content of R3 and the result placed in R1.

$$(K1 + K2): \; R1 \leftarrow R1 \vee R3$$

NOTE: "+" (as in $K_1 + K_2$) and means "OR."

In $R1 \leftarrow R1 + R3$, + means "plus."

Control Expressions

The control expression for an operation appears to the left of the operation and is separated from it by a colon

Control expressions specify the logical condition for the operation to occur

Control expression values of:

Logic "1" -- the operation occurs.

Logic "0" -- the operation is does not occur.

- Example:

  $\overline{X}\ K_1$: R1 ← R1 + R2

  X K_1 : R1 ← R1 + $\overline{R2}$ + 1

- Variable K1 enables the add or subtract operation.
- If X =0, then $\overline{X}$ =1 so $\overline{X}\ K_1$ = 1, activating the addition of R1 and R2.
- If X = 1, then X K_1 = 1, activating the addition of R1 and the two's complement of R2 (subtract).

Arithmetic Micro-operations

Symbolic Designation	Description
$R0 \leftarrow R1 + R2$	Addition
$R0 \leftarrow \overline{R1}$	Ones Complement
$R0 \leftarrow \overline{R1} + 1$	Two's Complement
$R0 \leftarrow R2 + \overline{R1} + 1$	R2 minus R1 (2's Comp)
$R1 \leftarrow R1 + 1$	Increment (count up)
$R1 \leftarrow R1 - 1$	Decrement (count down)

Note that any register may be specified for source 1, source 2, or destination.

These simple micro-operations operate on the whole word.

Logical Micro-operations

Designation	Description
$R0 \leftarrow \overline{R1}$	Bitwise NOT
$R0 \leftarrow R1 \vee R2$	Bitwise OR (sets bits)
$R0 \leftarrow R1 \wedge R2$	Bitwise AND (clears bits)
$R0 \leftarrow R1 \oplus R2$	Bitwise EXOR (complements bits)

Let R1 = 10101010
and R2 = 11110000

R0	**Operation**
	$R0 \leftarrow \overline{R1}$
	$R0 \leftarrow R1 \vee R2$
	$R0 \leftarrow R1 \wedge R2$
	$R0 \leftarrow R1 \oplus R2$

Logical Micro=operations

Designation	Description
$R0 \leftarrow \overline{R1}$	Bitwise NOT
$R0 \leftarrow R1 \vee R2$	Bitwise OR (sets bits)
$R0 \leftarrow R1 \wedge R2$	Bitwise AND (clears bits)
$R0 \leftarrow R1 \oplus R2$	Bitwise EXOR (complements bits)

Let R1 = 10101010
and R2 = 11110000

R0	**Operation**
01010101	**$R0 \leftarrow \overline{R1}$**
11111010	**$R0 \leftarrow R1 \vee R2$**
10100000	**$R0 \leftarrow R1 \wedge R2$**
01011010	**$R0 \leftarrow R1 \oplus R2$**

Shift Micro-operations

Let R2 = 11001001

Then after the operation, R1 becomes:

Symbolic Designation	Description
R1 ← sl R2	Shift Left
R1 ← sr R2	Shift Right

R1	Operation
	R1 ← sl R2
	R1 ← sr R2

- Note: These shifts "zero fill". Sometimes a separate flip-flop is used to provide the data shifted in, or to “catch” the data shifted out.

Shift Micro-operations

Let R2 = 11001001

Then after the operation, R1 becomes:

Symbolic Designation	Description
R1 ← sl R2	Shift Left
R1 ← sr R2	Shift Right

R1	Operation
10010010	R1 ← sl R2
01100100	R1 ← sr R2

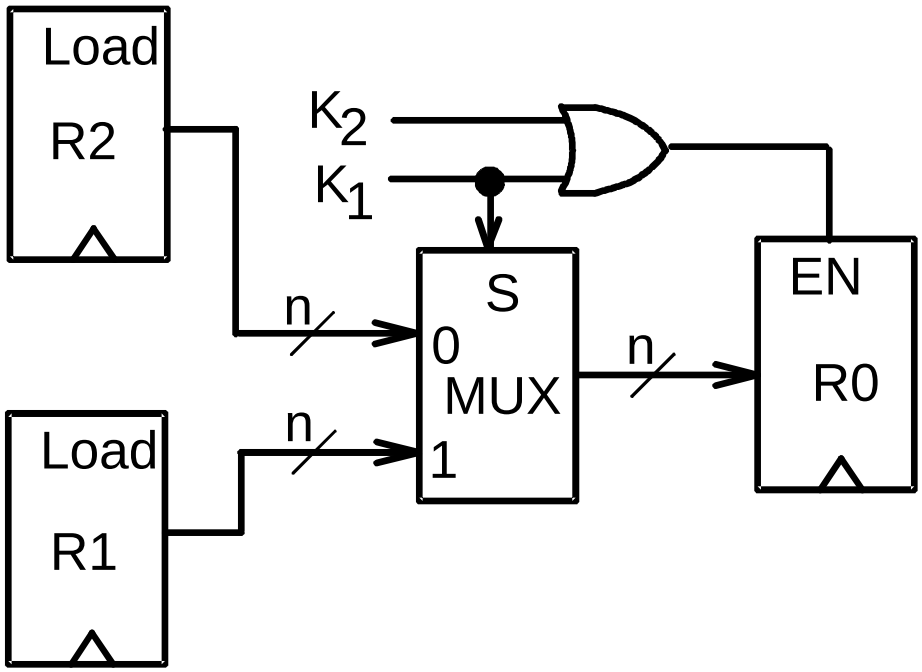
- Note: These shifts "zero fill". Sometimes a separate flip-flop is used to provide the data shifted in, or to “catch” the data shifted out.

Multiplexer-Based Transfers

Multiplexers connected to register inputs produce flexible transfer structures (Note: Clocks are omitted for clarity)

The transfers are: K1: $\underline{R0} \leftarrow R1$

$K2 \cdot \overline{K1}$: $R0 \leftarrow R2$



Sequences of these low level micro-operations are used to implement the more complex instructions which must be executed by the processor.