33 REGISTERS

A *register* is a set of flip-flops (or latches) used to store a binary sequence.

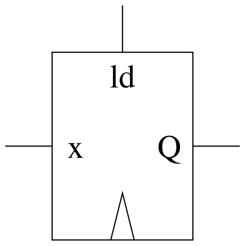
Besides storage, registers may also be designed to provide some "operational capability" such as incrementing, decrementing, clearing, and shifting.

33.1 Storage Register

A storage register is one where all the bits can be loaded with externally provided values in one clock enable. That is, an entire bit sequence whose length matches the size of the register can be loaded in during one transition of the clock control input. Because all the bits are loaded at the same time, this device is also called a "parallel load register".

The behavioral description of a 1-bit storage register is:

already value, loading or not a register.



Q	ld	x	Q+
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

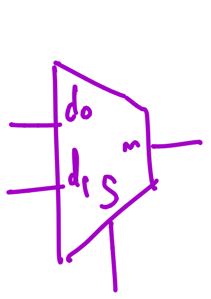
current Q

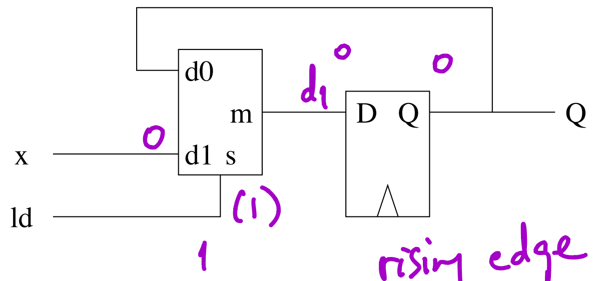
Q after

Using a D flip-flop, the flip-flop input function obtained from the characteristic table is:

$$D = Q \cdot ld' + x \cdot ld$$

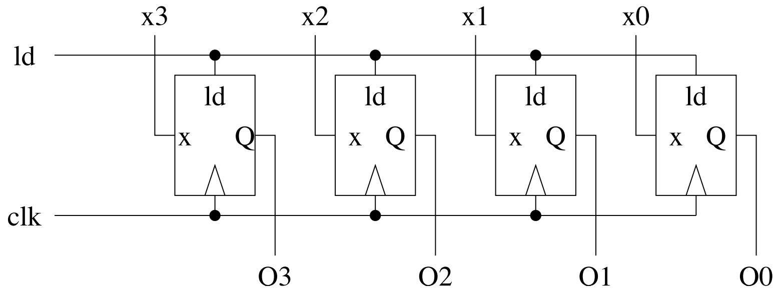
This flip-flop input function can be implemented using a 2x1 MUX.:





rising edge on clock.

Larger storage registers can be constructed by "chaining" 1-bit storage registers. For example, a 4-bit storage register can be obtained as follows:



34 REGISTER TRANSFER NOTATION

As the number of signal lines increases with the complexity of the design, it becomes necessary to develop a more compact notation for describing tasks at the register level, and for representing the connections between register level components in the logic diagram. It is for this reason that "register transfer notation" is used and that signal lines are grouped into "bundles" called "buses".

Proper use of any notation system is necessary if all users of the notation are to interpret the meaning of the notation in the same way. Therefore the following conventions are followed:

1. Register names are expressed using capital letters only. Except at the beginning, digits may also be used.

2. A bit of a register is represented by a subscript following the register name. For example, R_3 or R(3) denote bit 3 of register R.

3. A *field* of a register is a continuous subset of bits of the register. Fields can be specified by "ranges" or labels. A range is defined by a pair of integers indicating the most significant bit followed by the least significant bit of the field. An ellipsis (" ... ") is used to separate the integers. The range is enclosed in parentheses following the name of the register.

   A range can be labeled with an identifier consisting of capital letters (with digits). Some examples of fields are R(3. . .0) and R(OPCODE).

4. A *register transfer statement* is an expression of the form:

   $$\texttt{register-name} \leftarrow \texttt{expression}.$$

   The `register-name` can also specify a bit or a field of a register

5. A bus name is expressed using small letters only. Except at the beginning, digits may also be used.

6. A single signal line of a bus, sometimes called a “bit position on a bus”, can be identified with a subscript following the bus name. For example, addr_3 and addr(3) both denote bit position 3 of a bus named “addr”.

7. A “sub-bus” is a subset of signal lines of a bus. Sub-buses can be identified by ranges as was the case for registers. They can also be labeled using only small letters (and digits).

8. A *data bus* is a collection of signal lines for transferring a binary sequence. A *bus assignment statement* is an expression of the form:

   ```
   bus-name = expression
   ```

   The bus-name can also be a signal or sub-bus, but not a register. Similarly, in a register transfer statement, the “left-hand side” cannot be a bus or signal name.

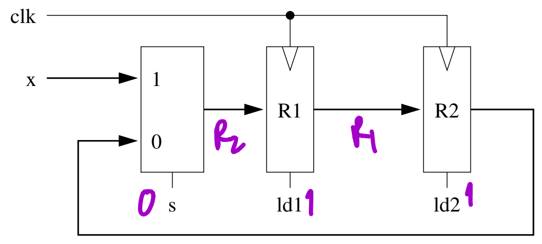
9. Buses are represented in logic diagrams with “arrows”. The size of the bus (i.e., the number of bit positions) is indicated with a “numbered slash” drawn across the bus at some point, the number specifying the size.

One further refinement: the “+” symbol is dropped from the name of the register in Function Select Table descriptions because it is understood that the register name occurring on the left hand side of a register transfer statement represents the new state of the register named.

35 DATA TRANSFER BETWEEN REGISTERS

The interconnection of register components is a common occurrence in more complex digital systems. Therefore it is important to understand how the trigger of a clock transition in a synchronous circuit affects the transfer of data from one component to another.

Consider the following logic diagram:

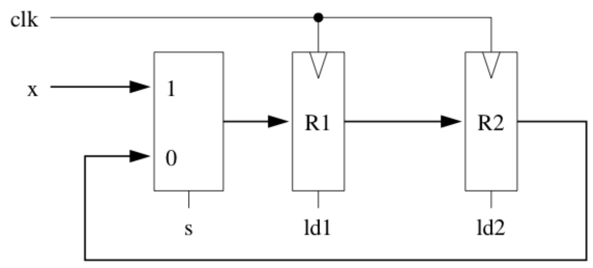


Now suppose that both control inputs (`ld1` and `ld2`) are set to 1. The effect of a single clock transition that enables the two registers simultaneously is to transfer data from one register to the next without the value being loaded into the first register "leaking" through the the second and being loaded there as well. This cannot happen on a single transition because the amount of time when the registers are enabled is equal to the transition time from logic-0 to logic -1 and this time is much less that the propagation delay of either register.

In particular, if `s = 1`, then on a single clock transition it is possible to swap the values in the two registers without the need for a third register to temporarily hold the contents of one of the registers being swapped.

As with previous logic diagrams, analysis consists of constructing a behavioral description; that is an entity definition and a functional specification. With circuits constructed from registers, the functional specification is most easily provided with a function select table. For the example given this is:

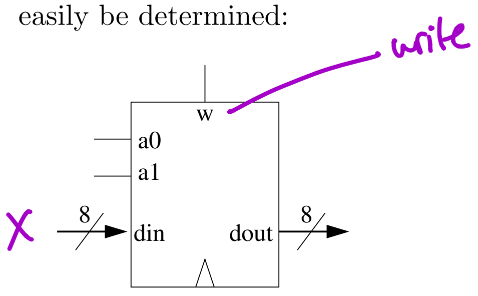
s	ld1	ld2	FUNCTION (on the postive clock edge)
0	0	0	no change
0	0	1	R2 ← R1
0	1	0	R1 ← R2
0	1	1	R2 ← R1, R1 ← R2
1	0	0	no change
1	0	1	R2 ← R1
1	1	0	R1 ← x
1	1	1	R2 ← R1, R1 ← x



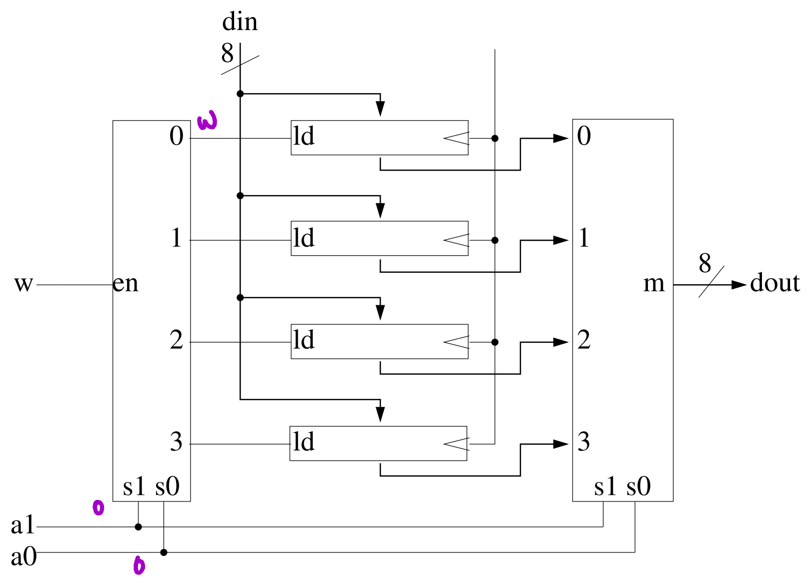
35.1 Simple Memory Systems

Consider the logic diagram on the next page.

By constructing the behavioral description of the device, its purpose can more easily be determined:



w	a1	a0	FUNCTION (on +ve edge)
0	0	0	dout = R0
0	0	1	dout = R1
0	1	0	dout = R2
0	1	1	dout = R3
1	0	0	dout = R0, R0 <– x
1	0	1	dout = R1, R1 <– x
1	1	0	dout = R2, R2 <– x
1	1	1	dout = R3, R3 <– x



This device is a type of memory, called a “register file memory” because each location of the memory is provided by a register. The purpose of each of the inputs is as follows:

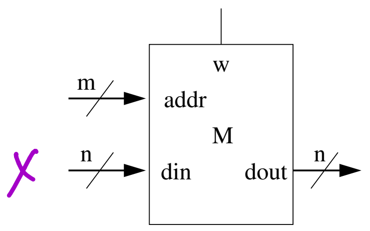
- Control input `w` is the “store” or “write” select. If `w = 1` then the memory performs the store function, delivering to its output the contents of the register selected using the control inputs `a1` and `a0`.
- The inputs `a1` and `a0` are called “address bits” because collectively they specify which register a value is to be retrieved from or stored to. Since there are four possible registers, two address bits are required to specify any one of four registers., The value specified by the address bits is called an “address” or “location”.
- The input `din` provides the means of providing a bit sequence of length 8 to be stored in one of the locations (i.e., one of the registers) of the memory. Because the input size is 8 bits, the registers used must all be 8 bit store registers.

The output port `dout` is a bus port permitting the output of all bits from any selected location to be output in parallel.

Memories are specified by two numbers in an expression in the form $a \times b$. The value of the first number, a, specifies how many locations of storage (i.e., number of registers) are provided by the memory. The second number, b, specifies how many bits can be stored in each location (i.e., the size of each register). Thus the memory above is a $2^2 \times 8$ or 4×8 memory.

The register where a value is stored is called a *word* of memory. Memory words can be specified in register transfer notation by giving the name of the memory followed by the location enclosed in "square brackets". Thus `M[45]` refers to the word at location 45 of the memory `M`. The location of a word of memory is called its *address*.

In general, the behavioral description of a simple $2^m \times n$ register file memory is given by:



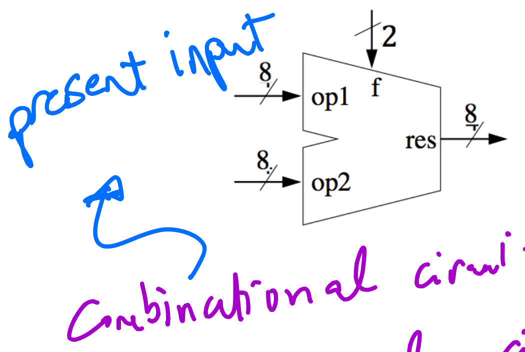
w	FUNCTION (on +ve edge)
0	out = M[addr]
1	out = M[addr], M[addr] <– x

Thus this device provides 2^m locations for storing binary sequences of length n.

Arithmetic Logic Units

An "Arithmetic logic Unit" or ALU is a multi-functional digital system that provides most of the basic arithmetic and logic functions for the central processing unit of a digital computer. Those functions not provided by the ALU can usually be obtained by performing sequences of those functions that the ALU does provide.

The following behavioral description defines an ALU with 4 arithmetic and logic functions:



f	function
00	res = op1 AND op2
01	res = op1 + op2
10	res = NOT op1
11	res = op1

$$\begin{array}{r} + \; 011 \\ 001 \\ \hline 100 \end{array}$$

not sequential circuit

↳ depends on past input, clk input

f	function
00	res = op1 AND op2
01	res = op1 + op2
10	res = NOT op1
11	res = op1