

7 Memories and Processors

7.1 Memory Terminology

We will not discuss the topic of data storage technologies *per se*. We are mostly interested here in the question of how data storage can be effectively *organized*. The important common element of the memories we will study is that they are *random access memories*, or RAM. This means that each bit of information can be *individually* stored or retrieved — with a valid input address. This is to be contrasted with *sequential* memories in which bits must be stored or retrieved in a particular sequence, for example with data storage on magnetic tape. Unfortunately the term RAM has come to have a more specific meaning: A memory for which bits can both be easily stored or retrieved (“written to” or “read from”). Here is a rundown on some terms:

- RAM. In general, refers to random access memory. All of the devices we are considering to be “memories” (RAM, ROM, etc.) are random access. The term RAM has also come to mean memory which can be both easily written to and read from. There are two main technologies used for RAM:

1.) Static RAM. These essentially are arrays of flip-flops. They can be fabricated in ICs as large arrays of tiny flip-flops.) “SRAM” is intrinsically somewhat faster than dynamic RAM.

2.) Dynamic RAM. Uses capacitor arrays. Charge put on a capacitor will produce a HIGH bit if its voltage $V = Q/C$ exceeds the threshold for the logic standard in use. Since the charge will “leak” off through the resistance of the connections in times of order ~ 1 msec, the stored information must be continuously refreshed (hence the term “dynamic”). Dynamic RAM can be fabricated with more bits per unit area in an IC than static RAM. Hence, it is usually the technology of choice for most large-scale IC memories.

- ROM. *Read-only memory*. Information cannot be easily stored. The idea is that bits are initially defined and are never changed thereafter. As an example, it is generally prudent for the instructions used to initialize a computer upon initial power-up to be stored in ROM. The following terms refer to versions of ROM for which the stored bits *can* be over-written, but not easily.
- PROM. *Programmable ROM*. Bits can be set on a programming bench by burning “fusible links,” or equivalent. This technology is also used for programmable array logic (PALs), which we will briefly discuss in class.
- EPROM. ROM which can be erased using ultraviolet light.
- EEPROM. ROM which can be erased electronically.

A few other points of terminology:

- As you know, a *bit* is a binary digit. It represents the smallest element of information.
- A *byte* is 8 bits.
- A “*K*” of memory is $2^{10} = 1024$ bits (sometimes written KB). And a megabit (MB) is $1K \times 1K$ bits.

- RAM is organized into many data “words” of some prescribed length. For example, a RAM which has $8K = 8192$ memory locations, with each location storing a data word of “width” 16 bits, would be referred to as a RAM of size $8K \times 16$. The total storage capacity of this memory would therefore be 128KB, or simply a “128K” memory. (With modern *very large scale integration* (VLSI) technology, a typical RAM IC might be ~ 16 MB.)
- Besides the memory “size,” the other important specification for memory is the *access time*. This is the time delay between when a valid request for stored data is sent to a memory and when the corresponding bit of data appears at the output. A typical access time, depending upon the technology of the memory, might be ~ 10 ns.

7.2 Memory Configuration

As stated above, the term “memory” refers to a particular way of organizing information — by random access — which is distinct from the less specific term “data storage.” Figure 36 shows how an 8-bit RAM (8×1) is organized. (This is a very small memory, but illustrates the concepts.) Our RAM consists of three main components: an 8-bit multiplexer, an 8-bit demultiplexer, and 8 bits of storage. The storage shown consists of edge-triggered D-type flip-flops. Hence, this is evidently a “static RAM.” (There is no fundamental reason for using edge-triggered flip-flops. They could just as easily be level-triggered, like the simple “clocked” S-R flip-flop of Fig. 14.)

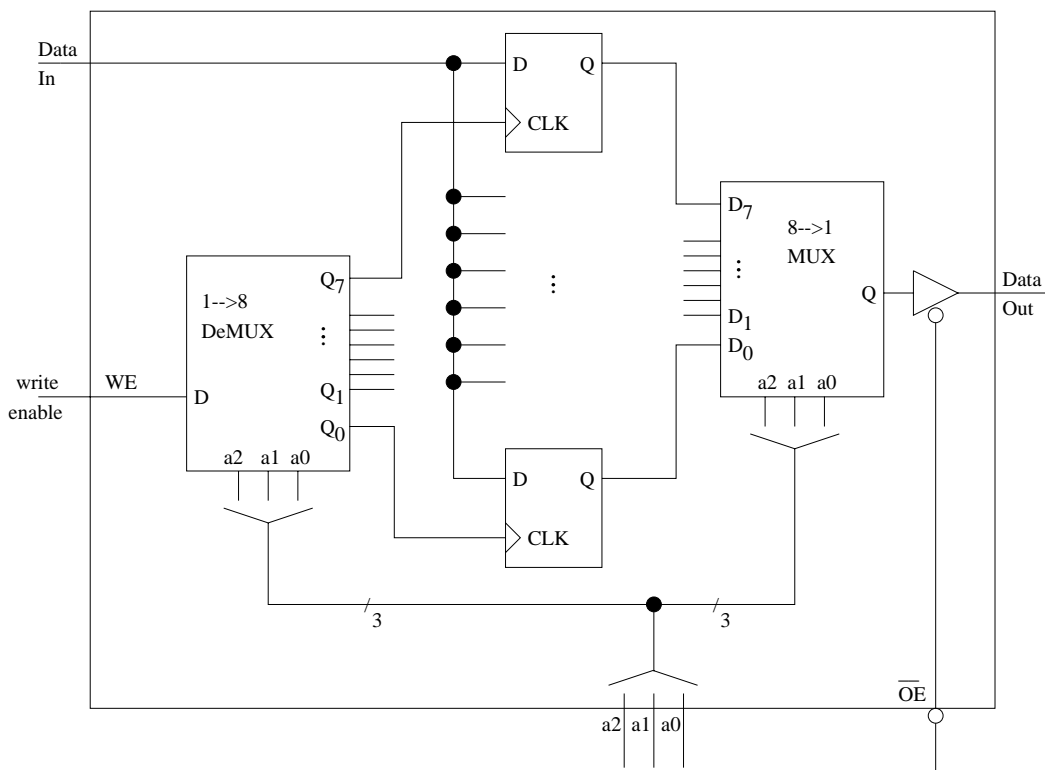


Figure 36: An 8×1 bit RAM.

Our example RAM has 6 external connections which are inputs (data in, write enable ($\overline{\text{WE}}$), 3-state enable ($\overline{\text{OE}}$), and 3 address bits ($A = a_2a_1a_0$), and has one output connection (data out), giving 7 external connections total, plus 2 for power/ground. To write information to the RAM, one would supply a valid address, for example $A = 101$. The data bit to be written to location 101 is to appear at the data input as either a logic HIGH or LOW signal. And to enable the writing into this bit, the $\overline{\text{WE}}$ signal must be asserted. This then appears at the Q_5 output of the demultiplexer, and is passed on to the appropriate flip-flop, which stores the input data bit and passes it on to the Q_5 multiplexer input.

To read data from our RAM, one asserts an address, so that the selected bit is sent to the MUX output and then the 3-state buffer. The purpose of the 3-state buffer is to ensure that no digital outputs are directly connected together, for example if our RAM output were connected to a data “bus,” which in turn was connected to several other devices. Recall that the 3-state devices have outputs which are effectively disconnected if there is no enable signal. So if the output data connection of our RAM is connected to a data bus, then the $\overline{\text{OE}}$ signal must be coordinated with any other outputs also connected to the data bus. When it is OK to read data from the RAM (all other output devices are disconnected from the bus), the $\overline{\text{OE}}$ signal is asserted and the MUX output will appear at the RAM output.

One could of course also store the 8 bits of data directly to an 8-bit data register, rather than using the RAM configuration outlined above. In this case, the number of external connections is 17 (8 data in, 8 data out, and 1 clock), compared with the 7 of our RAM. For a more realistic case where the number of bits of memory n is much larger than our example, we generalize the above to arrive at $4 + \log_2(n)$ external connections for the RAM, compared with $1 + 2n$ for the standalone register. Obviously for large n , the register is impractical, whereas the RAM remains reasonable. Actually, it is even somewhat better than this for the RAM case, since the number of external connections does not grow with the width of the stored data words. Hence, a RAM of size $1K \times 16 = 16 \text{ KB}$ requires only 14 connections. This is to be compared with 32,001 connections for the register. Note that the RAM can only supply one bit at a time to the output. This may seem like a handicap, but is actually well matched to standard microprocessors.

7.3 A State Machine with Memory

For reference, our usual state machine configuration is shown again in Fig. 37. Now we consider the use of a memory with a state machine, as depicted in Fig. 38. A random access memory is used in place of the usual combinational logic. (A ROM has been specified, to emphasize that we are not changing the memory — once it is defined initially, it is only read from. The memory is used to conveniently encode the connection between present and next states.

To start with, let’s assume a state machine with no external inputs or outputs. Then the state machine’s present state (PS) becomes an *address* which is input to the ROM. The *data word* stored in the ROM at that address then corresponds to the next state (NS). This correspondence had been initially programmed into the ROM, just as the specific combinational logic in our old state machine had to be pre-determined. So if the PS as defined by the Q bits at the data register are, for example, 1001, then the ROM data word at address 1001 will be the NS which is then passed back to the register. When there are also external inputs, as there will be for most anything of interest, these are combined with the PS bits to form a longer address for the ROM. Similarly, any external outputs are combined with

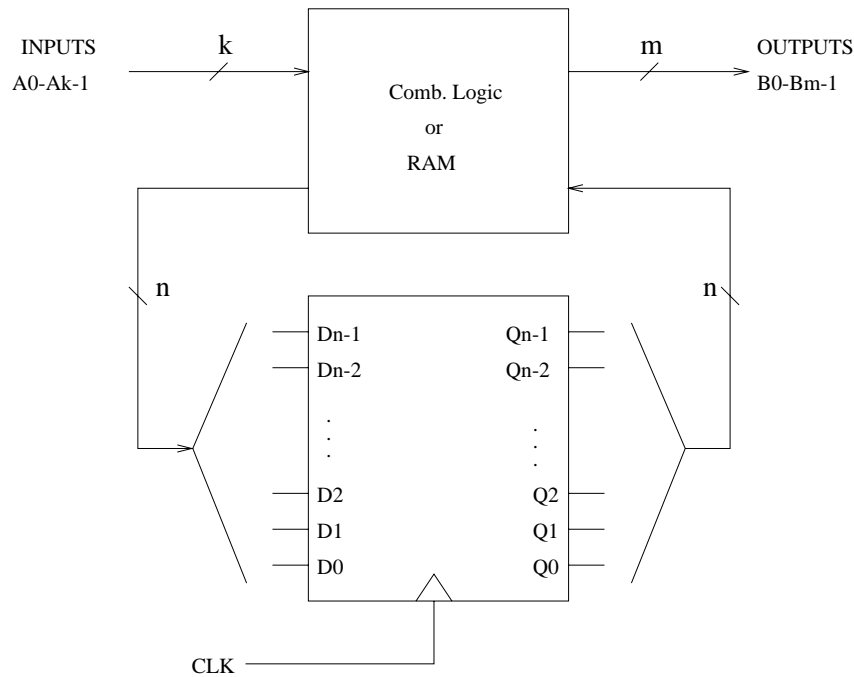


Figure 37: The standard state machine configuration.

the NS bits in the data word.

This should become clear with an example.

7.3.1 Example: Divide by 2 or 3 Counter

We will use a state machine with ROM, as in Fig. 38, to design a counter which either divides by 2 or by 3, depending upon the value of an external input bit p . This state machine will require 3 states, therefore we will need to describe 4 states, using 2 bits. We can label the states $A = 00$, $B = 01$, $C = 10$, and $D = 11$. Let $p = 0$ be the divide by 2 case, and $p = 1$ the divide by 3. The output bit $r = 1$ when the present state is B . Otherwise $r = 0$. State D is normally unused. The truth table is below. The student should draw the corresponding state diagram.

p	Present State		Next State		r
	Q_1Q_0		D_1D_0		
0	00	A	B	01	0
0	01	B	A	00	1
0	10	C	A	00	0
0	11	D	A	00	0
1	00	A	B	01	0
1	01	B	C	10	1
1	10	C	D	11	0
1	11	D	A	00	0

This ROM requires 3 address bits (2 for PS and 1 for input bit p), which corresponds to 8 locations in memory. Each location has a data word which has length 3 bits (2 for NS and

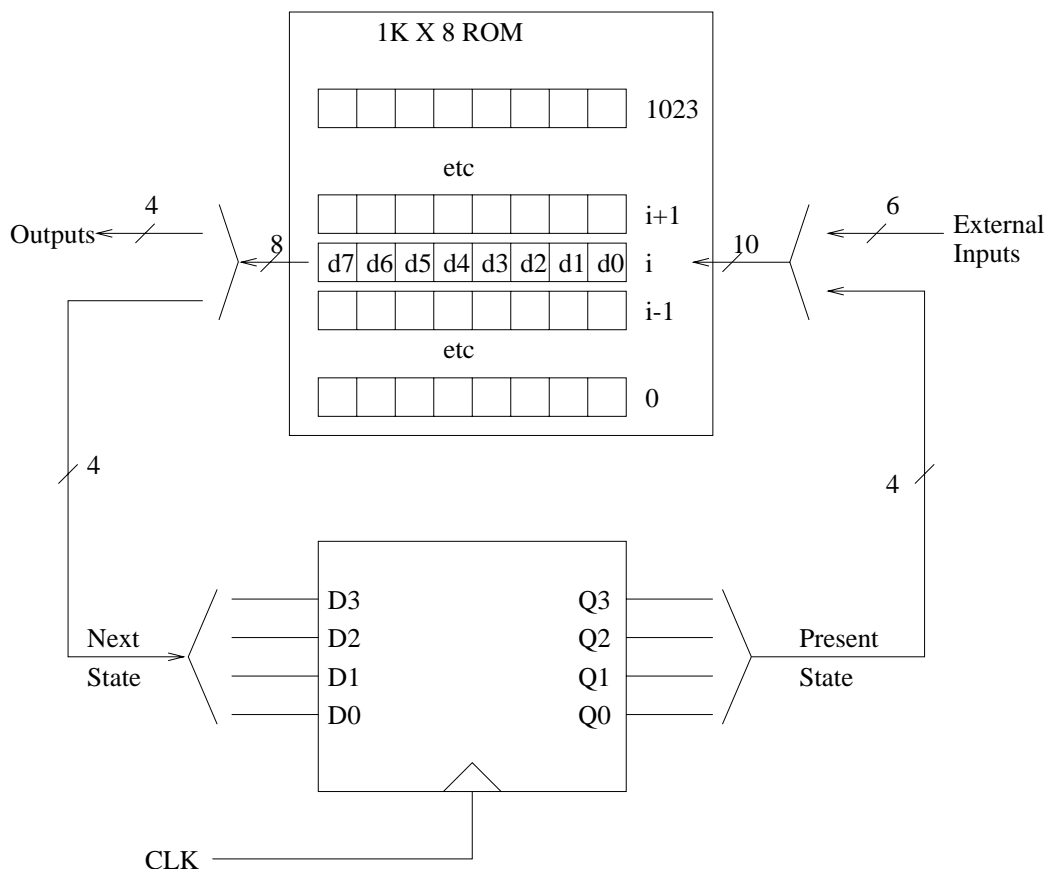


Figure 38: Toward a microprocessor: Replacing the combinational logic with a memory.

1 for the output bit r). Therefore, the size of this memory is 8×3 , or 24 total bits. A very small ROM indeed. The programming of the ROM is very straightforward and can be read directly from the truth table above. We just need to set an encoding convention. Let the addresses be encoded as pQ_1Q_0 and the data words as D_1D_0r . For example, let's look at the 5th row of the truth table. The address would be 100 and the data word at this address would be 010. The remaining bits of the ROM would be programmed in the same way. So one would initially “burn in” these bit patterns into the ROM and put it into the circuit. That's all there is to it. Of course if one were careful not to overwrite the memory, or if an evolving logical pattern were required, then a RAM could be used instead of the ROM.

7.3.2 Generalization to Microprocessors

A state machine with zero input bits can perform a counter-like function, but not more: its next state is limited to be a function only of the present state. A single input bit can be used to “program” the state machine to behave in one of two possible ways for each present state, as we discussed, for example, with the up/down counter of Section 4.4.1, or the example in the preceding section. On the other hand, with n inputs, the machine can perform 2^n different operations. So, for example, with $n = 8$ the machine can perform one of 256 different operations on each clock cycle. This tremendous potential and flexibility. The input bits can themselves be sequenced — stored externally in a specific sequence which

is then applied step by step to the state machine inputs on successive clock cycles. Such a stored sequence of operations is a *program* and the 256 operations represent the programming operations. In Fig. 38 we have essentially configured a simple *microprocessor*. The inputs and outputs would need to be connected to buses (via 3-state buffers where appropriate), which in turn are also connected to memories which store the program and any output or input data. The buses would also be connected to various input/output devices, mass storage devices, etc.