

Lecture 16: Random Access Memory and the Fetch Cycle

Random Access Memory

We are already familiar with the concept of a one bit memory. A single D-type flip flop is a one bit memory with which we can associate a unique address by using a decoder. (A decoder is also known as a de-multiplexer or a binary to unary converter) Thus a 256 bit RAM could be built out of an array of circuits of which one element is shown in Figure 1.

If a decoder detects the unique binary address of its one bit memory cell on the address lines it will enable the cell. The two AND gates determine whether data is read or written. If the Read input is 1 the clock pulse is suppressed and the Q value is placed on the output data line. If the Read input is 0 the flip flop receives a clock pulse and is loaded from the data in line. Notice the asymmetry in the circuit. For reading it is merely a combinational circuit, but for writing the address and data must be present and correct when the clock pulse sets the flip flop. RAM circuits conforming to this pattern are called static RAMs, and are used in special applications.

Buses

Within the circuit of Figure 1 (and in the data path diagram of the manual processor) we have introduced some data highways which are common to all the individual cells. The same address lines go to each decoder, and are referred to as the address bus. Similarly, the read, data in and data out lines go to every cell. The data in and data out lines are never both used at the same time, and indeed could not be for safe operation of the memory. Thus, it would be convenient to use just one line as this would reduce the size and complexity of the memory circuit. To make the data line bi-directional we need to feed it from more than one place and for this purpose we need a new type of gate which is referred to as a tri-state buffer and illustrated in Figure 2. If the control line, C_i , is set to zero the output follows the input exactly, however, if C_i is set to 1 the output is neither zero nor one, but is effectively disconnected from the data line. It is the logic designers problem to ensure that no two circuits feed the data line at the same time.

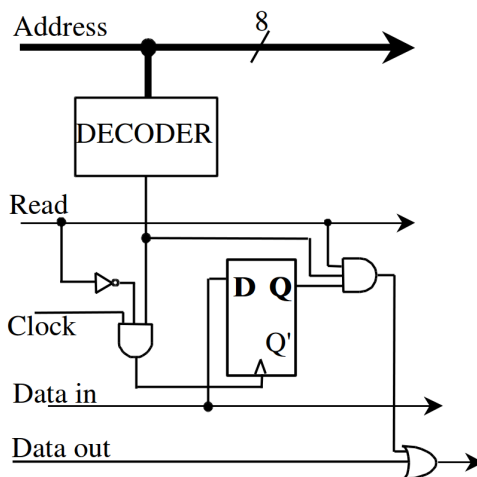


Figure 1: One bit of static RAM

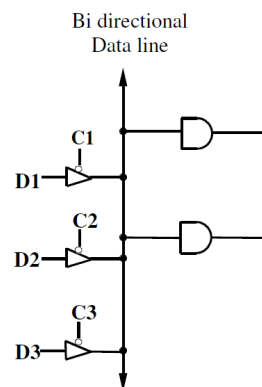


Figure 2: Tri-State Buffering

Practical RAM circuits

For convenience of manufacture, bulk RAMs are organised as square arrays of individual bits as shown in Figure 3. There are two decoders, a row and a column decoder, and each one bit memory cell is only enabled when both its row and the column lines are one. In the case of a 256 bit RAM each decoder transforms a four bit binary number into a sixteen bit unary number. Thus, in the square array of one bit memory cells, there will only ever be one cell for which both the row and the column lines are one. Each cell is connected to the same read/write line and data line. The data line is connected to the outside through a two way tri state buffer, such

that unless the chip is enabled no data can pass either in or out. This is important since it allows us to build external decoders for larger capacity RAMs made up of several banks of single chips.

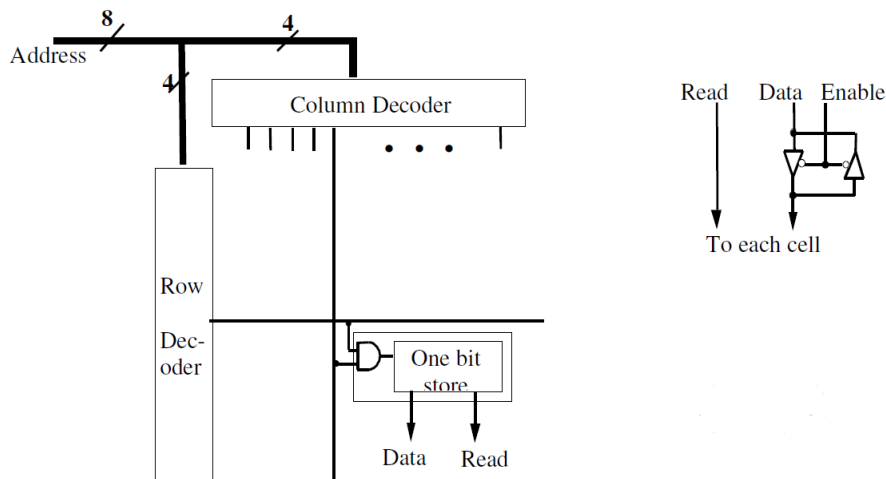


Figure 3: Layout of RAM

Connecting RAM to a processor

For the purposes of connecting some RAM to our processor we need to introduce special processor registers. One of these will hold the address that we wish to send to (or receive from), and this is the Memory Address Register (MAR). Another will hold the data received from the RAM, and this is called the Memory Data Register (MDR) or sometimes the Memory Buffer Register (MBR). The MDR is useful because in general fast RAM is low powered and cannot be connected to a large number of processor registers. Additionally we introduce two registers which are associated with getting program instructions (as opposed to data) from the memory. These are called the Program Counter (PC) which stores the address of the next program instruction to be executed, and the Instruction Register, which will act as a store for the program instructions. The IR (which we also showed in the data path diagram for the manual processor) will be decoded to determine the function of the central processor. The arrangement is shown in Figure 4. We have used a bi-directional data bus and the tristate buffer (TSB) can be enabled when writing to memory, but blocking when reading from the memory. Since we have opted to use 256×1 bit static RAMs we will need eight of them to store complete bytes of data. To keep things simple we will make the memory work in synchronisation with the system clock. To write data will require one clock cycle, and we assume that the memory logic is sufficiently fast so that we can read it within one cycle.

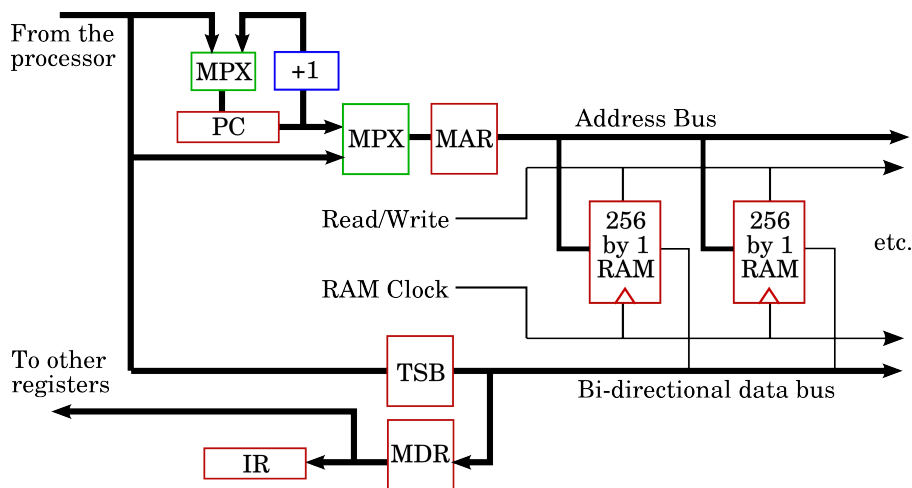


Figure 4: Connecting the Processor to the Memory

The Fetch Cycle

The fetch cycle, to get the next program instruction from memory, consists of just three operations, which in terms of register transfers are:

$MAR \leftarrow PC;$

$MDR \leftarrow RAM[MAR]; PC \leftarrow PC+1;$

$IR \leftarrow MDR.$

The back arrow can be read as "is loaded from". Notice that the second operation consists of two register transfers which occur in parallel. It would be possible to increment the program counter using the ALU, but this would greatly complicate the design of the central processor, and would also slow down the instruction execution. For these reasons we provide the program counter with its own incrementer (denoted +1). The incrementer is made up of half adders, as described in Lecture 13. The register transfers are achieved in two parts, first the multiplexers must be set to establish the required connections, and secondly, the clock pulses must be fed to the registers in the correct order. The latter is done by interrupting the system clock, which, as we saw in the last lecture, requires simply an additional NAND gate. The fetch cycle controller can therefore be treated as a synchronous machine in its own right. Its finite state machine is shown in Figure 5.

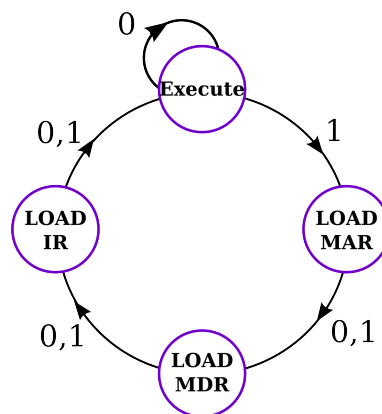


Figure 5: An idealised fetch Cycle

There are three states to carry out the fetch, and one called execute where the instruction is executed. In practice executing and instruction will usually require more than one state as we will see next lecture. The manual processor had five execute states. During the first state the address from which data is to be obtained is loaded in the MAR. In the second state the MDR is loaded from the memory and the PC is incremented. In the final state the MDR contents is transferred to the IR, or possibly to another processor register. In this idealised cycle there is just one input, $I=1$ initiating a fetch, and $I=0$ indicating that the processor is executing an instruction or is idle. The output logic can best be described by a table:

State	Clock Control				Multiplexer Control	
	MAR	MDR	IR	PC	PC Input	MAR Input
0. Executing	0	0	0	0	×	×
1. Load MAR	1	0	0	0	×	0
2. Load MDR/PC	0	1	0	1	0	×
3. Load IR	0	0	1	0	×	×

Zero sets the PC input multiplexer to select the incrementer, and one selects the output of the processor (eg ALU/Shifter). Similarly, zero sets the MAR input multiplexer to select the program counter, and one selects the processor output. We can implement sequencing and the output logic trivially as shown in Figure 6. The clocks to the MDR and RAM chips are blocked during the fetch cycle, and the multiplexers are both given a zero control input. It must be remembered that the fetch is not an isolated part of the operation of the computer. In practice there will be an execute cycle, and a finite state machine controller must be designed for the combined fetch and execute phases. The execute cycle will also involve the use of the above registers.

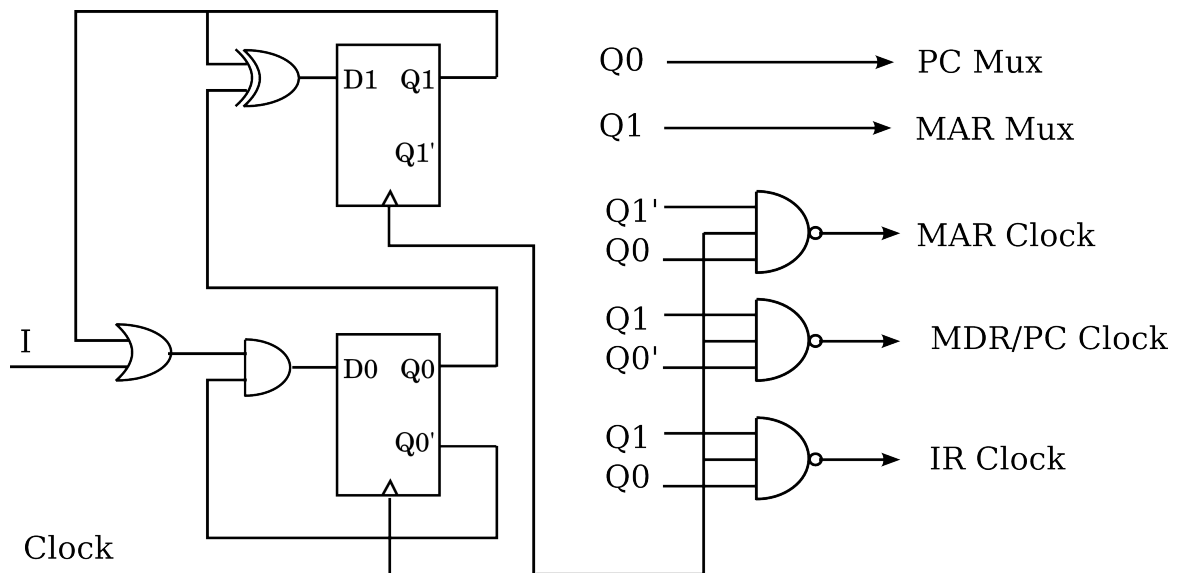


Figure 6: Controller Circuit for the Fetch Cycle

Dynamic RAMs

Much research has gone into reducing the size of single bit memory cells, since this in turn determines the number of bits that can be fitted onto a single silicon chip. The current minimal design uses only one transistor and one capacitor. It has the disadvantage that the store is not permanent. The storage capacitor is made as small as possible to maximise the number of bits on a chip. It is so small that if left alone, a cells storing 1 would drift to zero in a fraction of a second. Thus, some extra circuitry is provided which operates when the the computer is not accessing the memory. This is refered to as the refresh logic, and functionally it senses the cells that are set to 1 and boosts the charge on their capacitors. For this reason circuits of this kind are called dynamic RAM. It is important to note that dynamic RAM cannot be considered a combinational circuit like static RAM. It must give priority to completing its refresh cycle over servicing a read or write request from the processor, otherwise data will be lost. To ensure that the data is sent and received correctly, a special signal is generated by the RAM to indicate that data has been received or that data is available. These complications will not be considered further, and for our future design we will just use simple static RAM.