will be no pairs left to process, and each equivalence class will now receive a new label. At this point, the algorithm could output the number of such new labels (two in the cat-and-mouse example).

The third stage of the component-finding algorithm simply traverses the quad tree one more time, assigning the new label that is equivalent to each of the old labels encountered during the traversal.

Problems

1. Write an algorithm that converts a binary image matrix to the corresponding quad tree.

2. Although most images requires less storage in quad tree form than in matrix form, there is one definite exception. Find the worst case in this respect.

3. Construct a new quad tree for the cat after it has pounced on the mouse, i.e., moved two pixels west. Design a general translation algorithm for horizontal and vertical motion.

References

Hanan Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading, Mass., 1989.

Hanan Samet, *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS.* Addison-Wesley, Reading, Mass., 1989.

THE SCRAM

A Simplified Computer

The principles governing the design and operation of a real computer can be illustrated by the complete description of a "toy" computer. No one would seriously suggest that the computer shown in Figure 48.1 be manufactured (for one thing, its very small memory renders it useless for most practical computations). But it does have all the major features of larger and more sophisticated machines; only input and output hardware has been omitted. In short, it is a Simple but Complete Random Access Machine.

Before we plunge into the details of the SCRAM machine, a brief preview of its major components is in order. The acronym MUX stands for multiplexer (see Chapter 28). This is a device that routes information from different sources to a single destination. The choice of source depends on a control signal generated by the control logic. Seven registers appear in the diagram. They are the program counter (PC), instruction register (IR), memory address register (MAR), memory buffer register (MBR), the accumulator (AC) and the adder (AD). The last register is incorporated into the arithmetic logic unit (ALU). A timer T generates pulses that are decoded into separate input lines for various

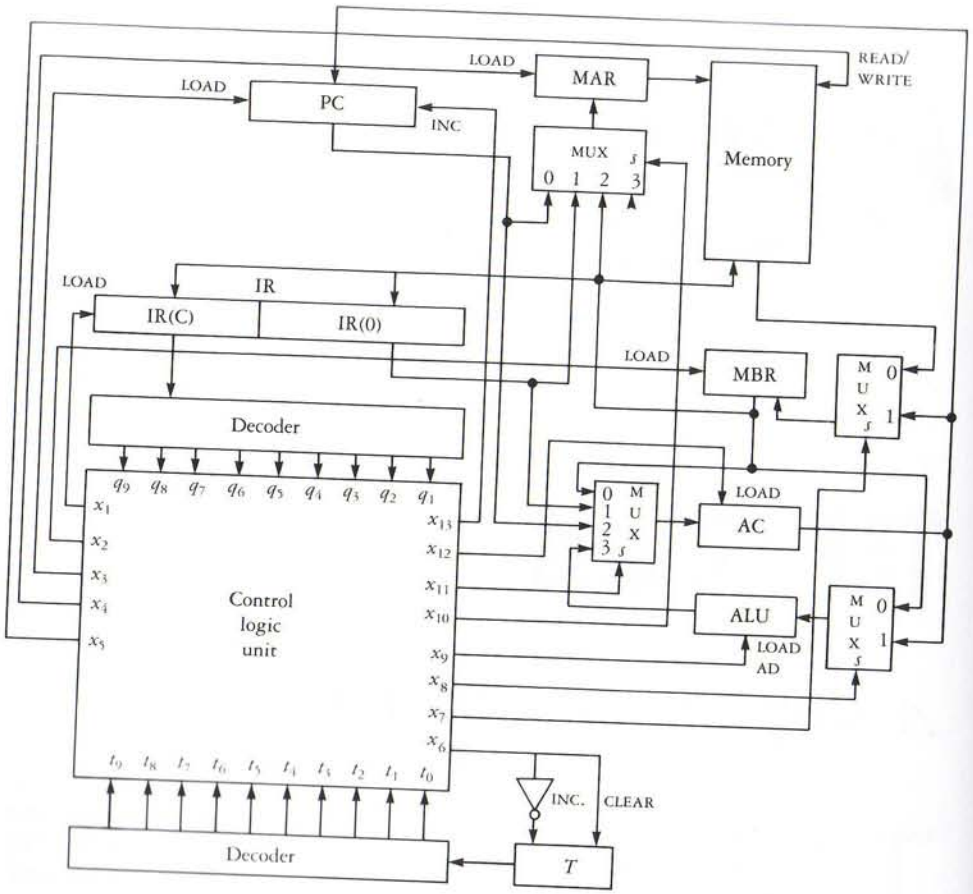


Figure 48.1 Layout of the SCRAM

destinations within the control logic unit (CLU). Another decoder translates program instructions in the IR for use by the CLU.

The diagram is entirely schematic; there is almost no relationship between its geometry and a layout of this circuit as silicon chips on a board. Indeed, the memory unit would be much larger in an actual circuit. As a general rule, long, thin rectangles represent registers, and anything else represents control or memory logic.

The SCRAM has 8-bit words, and the memory contains 16 words. It follows that 4 bits is sufficient to specify the address of any one of the memory words. As

a general rule, each word of memory will contain either a number or a program instruction. The number can be up to 8 bits in length, but the instruction must share these bits between an operation code (4 bits) and an operand (4 bits). In normal 8-bit machines, operations and operands are stored in alternate memory locations, but the circuitry must be more complex. The SCRAM is too simple for such sophistication, and the result is that only 4-bit numbers can be stored in the operand for any instruction.

The heart of the SCRAM computer is the CLU. We will follow one complete cycle of operation for each instruction of the low-level programming language described in Chapter 17. These instructions are listed below.

The cycle has two parts. They are called *fetch* and *execute*. The fetch cycle gets the next executable instruction in the program currently running and loads it into the instruction register. This cycle itself is written as a special sequence of elementary machine operations called a *microprogram*. Each line of the microprogram is called a micro-operation and is written in a special replacement notation called *register-transfer language:*

t_0: MAR ← PC
t_1: MBR ← M, PC ← PC + 1
t_2: IR ← MBR

All SCRAM micro-operations are set in motion by a timer T that feeds time signals via a decoder into the CLU. When the line labeled t_0 contains a 1, the contents of the PC are transferred to the MAR. In other words, the program counter contains the address in memory of the next executable instruction, and this address is transferred to the memory address register. When t_0 drops back to 0, t_1 contains a 1 and this signal initiates the next micro-operation; the contents of memory at the address contained in the MAR are transferred to the memory buffer register (see Chapter 38). At the same time, the program counter is incremented. This ensures that unless the instruction about to be executed is a JUMP command, the next instruction to be executed is the one following the current instruction in the program stored in memory. The current instruction consists of an operation code and an operand. Both are transferred at time t_2 into the instruction register from the MBR. The 4 high-order bits of the IR comprise a kind of subregister that we may call IR(C). The 4 low-order bits are called IR(0). IR(C) contains the instruction code, and IR(0) contains the operand.

The instructions, their codes, operands, and meanings are listed in the table at the top of the next page.

Operation	Code	Operand	Meaning
LDA	0001	*X*	Load contents of memory address *X* into the AC.
LDI	0010	*X*	Indirectly load contents of address *X* into the AC.
STA	0011	*X*	Store contents of AC at memory address *X*.
STI	0100	*X*	Indirectly store contents of AC at address *X*.
ADD	0101	*X*	Add contents of address *X* to the AC.
SUB	0110	*X*	Subtract contents of address *X* from the AC.
JMP	0111	*X*	Jump to the instruction labeled *X*.
JMZ	1000	*X*	Jump to instruction *X* if the AC contains 0.

The code subregister IR(C) is connected by four parallel lines (shown in the diagram as a single line) to a decoder which produces a 1 on exactly one of the nine input lines to the CLU. Each of the nine possible instruction types has its own characteristic binary pattern, and the decoder activates the appropriate line to the CLU in consequence (see Chapter 28).

The execution phase of the basic cycle immediately follows the fetch phase. At this point the t_3 input line carries a 1. At this and at subsequent times in the execution cycle, various micro-operations are performed, depending on the instruction type being executed. Each instruction has its own microprogram. For example, LDA has the following microprogram:

LDA
q_1t_3: MAR ← IR(0)
q_1t_4: MBR ← M
q_1t_5: AC ← MBR

During the fetch cycle, all that was needed to trigger each of the three micro-operations was 1s appearing sequentially on the timing lines t_0, t_1, t_2. Now more complicated triggers are required: If IR(C) contains the LDA code 0001, the decoder converts this to a 1 on line q_1, with the other q-lines all

containing 0s. The expression q_1t_3 is a logical expression meaning, in effect, "if q_1 and t_3, then . . . ." We will see later how the trigger is implemented by the output of an AND gate.

Thus when q_1 and t_3 are both 1, the SCRAM loads the operand portion of the instruction register into the memory address register. At the next time step, the contents of that address in memory are loaded into the memory buffer register and then into the accumulator.

The operational cycle of the SCRAM may require up to 10 consecutive time periods. The periods are each, let us say, a microsecond in length and are generated by a timing register *T*. The CLU increments *T* (based on its own clock) between micro-operations and clears *T* when a new operational cycle is to begin.

The next example of microcoding of program instructions is more complicated. The LDI command requires five micro-operations:

LDI *X*:
q_2t_3: MAR ← IR(0)
q_2t_4: MBR ← M
q_2t_5: MAR ← MBR
q_2t_6: MBR ← M
q_2t_7: AC ← MBR

The only difference between direct and indirect LOAD commands is that the latter requires an additional set of memory micro-operations. This is because indirection requires two separate memory retrievals, the first to get the address for the second. Shown below are the micro-programs for two other program instructions. Readers should not find it difficult, after this, to construct satisfactory microprograms for the remaining instruction types.

ADD *X*:
q_5t_3: MAR ← IR(0)
q_5t_4: MBR ← M
q_5t_5: AD ← MBR
q_5t_6: AD ← AD + AC
q_5t_7: AC ← AD

JMP *X*:
q_7t_3: AC ← PC
q_7t_4: AD ← AC
q_7t_5: AC ← IR(0)
q_7t_6: AD ← AD + AC
q_7t_7: AC ← AD
q_7t_8: PC ← AC

Note that the ADD command first retrieves the number stored at memory address X and then loads it in the special arithmetic register AD (not shown in the diagram). After the contents of the accumulator are added to the AD, the result is placed back in the AC.

The second microprogram uses the same operation to increment the program counter. This register contains the address of the next program instruction to be executed. If the current instruction is not a JUMP, the CLU will merely increment the PC sometime during its operational cycle.

At this point the question naturally arises of just how the nine different microprograms are implemented in actual logic circuits. Here again, we follow time-honored pedagogic practice by employing standard logic gates for the purpose. In any event, it is a relatively simple exercise to convert these circuits to any logically complete set of gates (see Chapter 3).

It is a relatively easy matter to implement the fetch cycle and the nine possible versions of the execution cycle in individual logic circuits.

The fetch cycle is shown in Figure 48.2. For the time being, we pretend that the line x_{10} is doubled; after all, x_{10} operates a multiplexer (MUX) with two control inputs. When t_0 is 1, both x_{10} lines are to be 0, causing the MUX to select input line 0 from the PC for transmission to the MAR. The x_4 line causes the MAR to be loaded in accordance with a register design appearing in Chapter 38. When t_1 is 1, x_7 is 0 and x_5 and x_{13} are both 1. This means that the MUX serving the MBR selects memory input, the read/write line to memory carries a read (0) signal, and the PC is incremented. When t_2 is 1, the instruction register is loaded with the current contents of the MBR.

A circuit that implements the LDA instruction is shown in Figure 48.3.

When it is considered in isolation, the operation of this circuit is also straightforward. First, the paired x_{10} lines control the MUX selecting input for the MAR. When t_3 and q_1 are 1, x_{10} is 01. When t_4 is 1, the x_7 line sends a 0 control signal to the MUX selecting input to the MBR. In this case, 0 means memory. Finally,

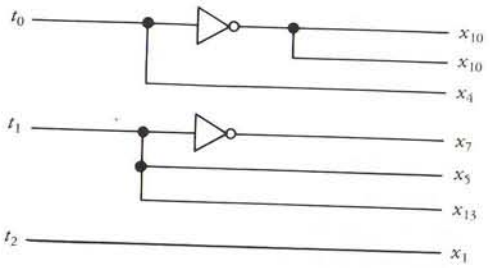


Figure 48.2 Logic for the fetch cycle

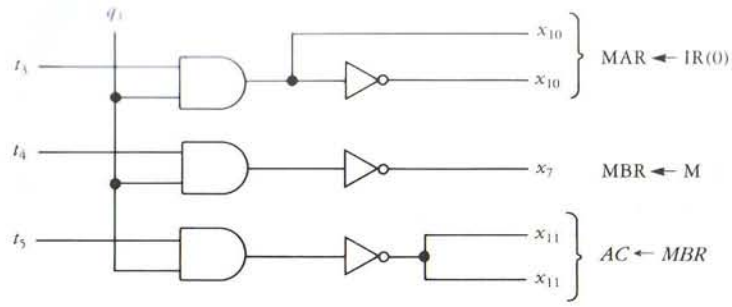


Figure 48.3 Logic for loading the accumulator

when t_5 is 1, both control lines to the MUX selecting input for the AC are 0. In this case 00 means that the MBR is selected.

Readers may have noticed a problem at this point; some of the output lines just discussed appear in both circuits. Since outputs of two different gates cannot be tied directly together, more gates must be used to effect this purpose. (See the problems that follow.)

Problems

1. Write microprograms for STA, STI, and JMZ. Implement the microprograms in standard logic.

2. Design that portion of the CLU that determines the two output lines labeled x_{10}. Input to this subcircuit will be one or both of the lines previously labeled x_{10} in the individual circuits for LDA, LDI, and the circuits designed in Problem 1.

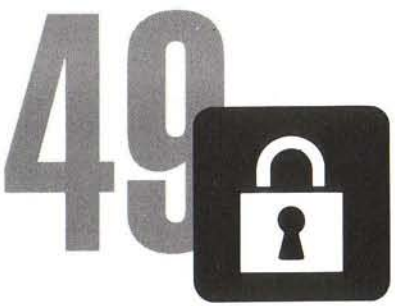
3. Convert the following program to the equivalent set of binary words, as indicated in this chapter. This is called *machine code.* Trace the execution of the program by listing the q, t, and x variables across the top of a sheet of paper. For each step in the SCRAM's operation, fill in one line of the table by writing down the value of each input and output variable:

LDA 1
ADD 2
STA 3

References

Richard S. Sandige. *Data Concepts Using Standard Integrated Circuits.* McGraw-Hill, New York, 1978.

Ronald J. Tocci and Lester P. Laskowski. *Microprocessors and Microcomputers,* 2d ed. Prentice-Hall, Englewood Cliffs, N.J., 1982.



SHANNON'S THEORY

The Elusive Codes

Any message consisting of words can be encoded as a seq
1s. The latter symbols can be transmitted by wire, ra
variety of other means. In all cases the message is liable t
can inadvertently be changed to a 1, or vice versa. Wha
source of interference or "noise," as information theorists call
of a bit to change can be laid at the doorstep of a creature I call t
With a certain probability, say p, the noise demon alters each bi
ted (Figure 49.1).

One way to fool the demon is to transmit three 0s for each 0
replace a 1 by three 1s. Assuming the receiver is in synchrony
so that the start of each triad is known, the decoding rule is
shown in the table at the top of the next page. What is the
message being corrupted under this scheme? It is the probabil
bits out of 3 get changed. Thus if 000 is sent, it can become 110
with respective probabilities p^2q, pqp, qp^2, or p^3, where $q = 1 -$
probabilities yields the formula $3p^2 - 2p^3$, which means that i
the new scheme of triples guarantees a much lower probabilit