Digital Systems Design

Register Transfer Logic

Introduction

- A digital system is a sequential logic system constructed with flip-flops and gates.
- A sequential circuit can be specified by means of a state table.
- To specify a large digital system with a state table would be difficult.
- To overcome this difficulty, we use some modules which are constructed from such digital functions as registers, counters, decoders multiplexers, arithmetic elements and control logic.

Microoperation

- It specifies the elementary operation to be performed on the information stored in registers.
- There are four categories of microoperations:
  - Interregister-transfer
  - Arithmetic
  - Logic
  - Shift

Interregister Transfer

- Information transfer from one register to another is designated in symbolic form by means of the *replacement operator*. The statement:

$$A \leftarrow B$$

Denotes the transfer of the *contents* of register B into register A.

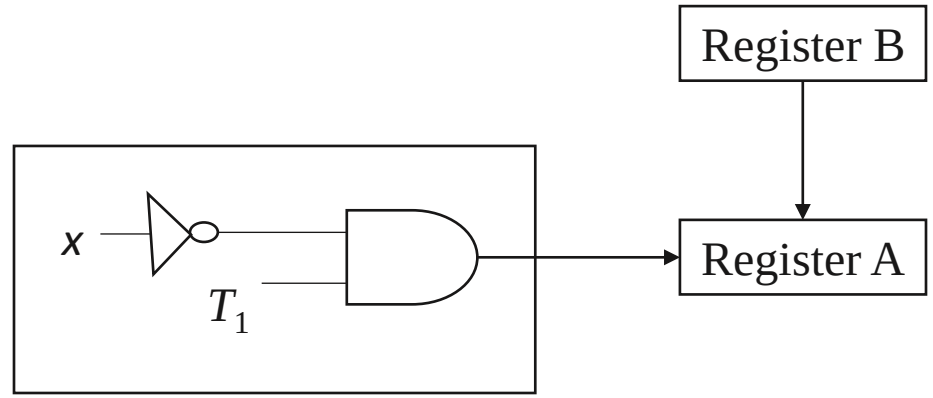
- The condition that determines when the transfer

  is to occur is called a *control function.*

- A control function is a Boolean function that is equal to 1 or 0.
- The control function is included with

  the statement as follows:

  $x'T_1$: $A \leftarrow B$



Interregister Transfer

- There are occasions when a destination register receives information from two sources, but evidently not at the same time. Consider two statements:

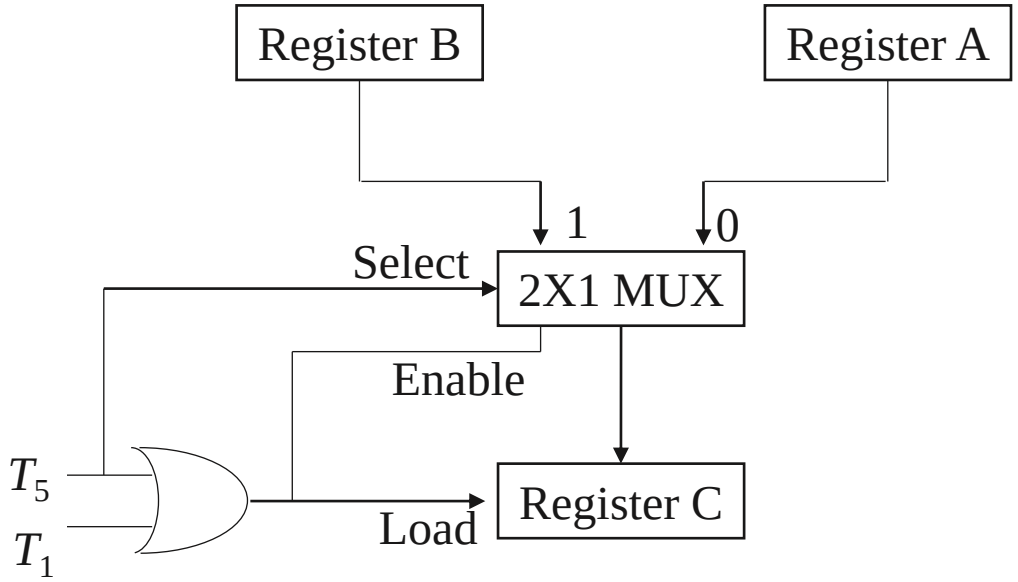
$$T_1: \ \text{C} \leftarrow \text{A}$$

$$T_5: \ \text{C} \leftarrow \text{B}$$

- T_1 and T_5 are two different timing variables.
- When $T_5 = 1$, register B is selected, but when $T_1 = 1$, register A is selected (because T_5 must be 0 when T_1 is 1).

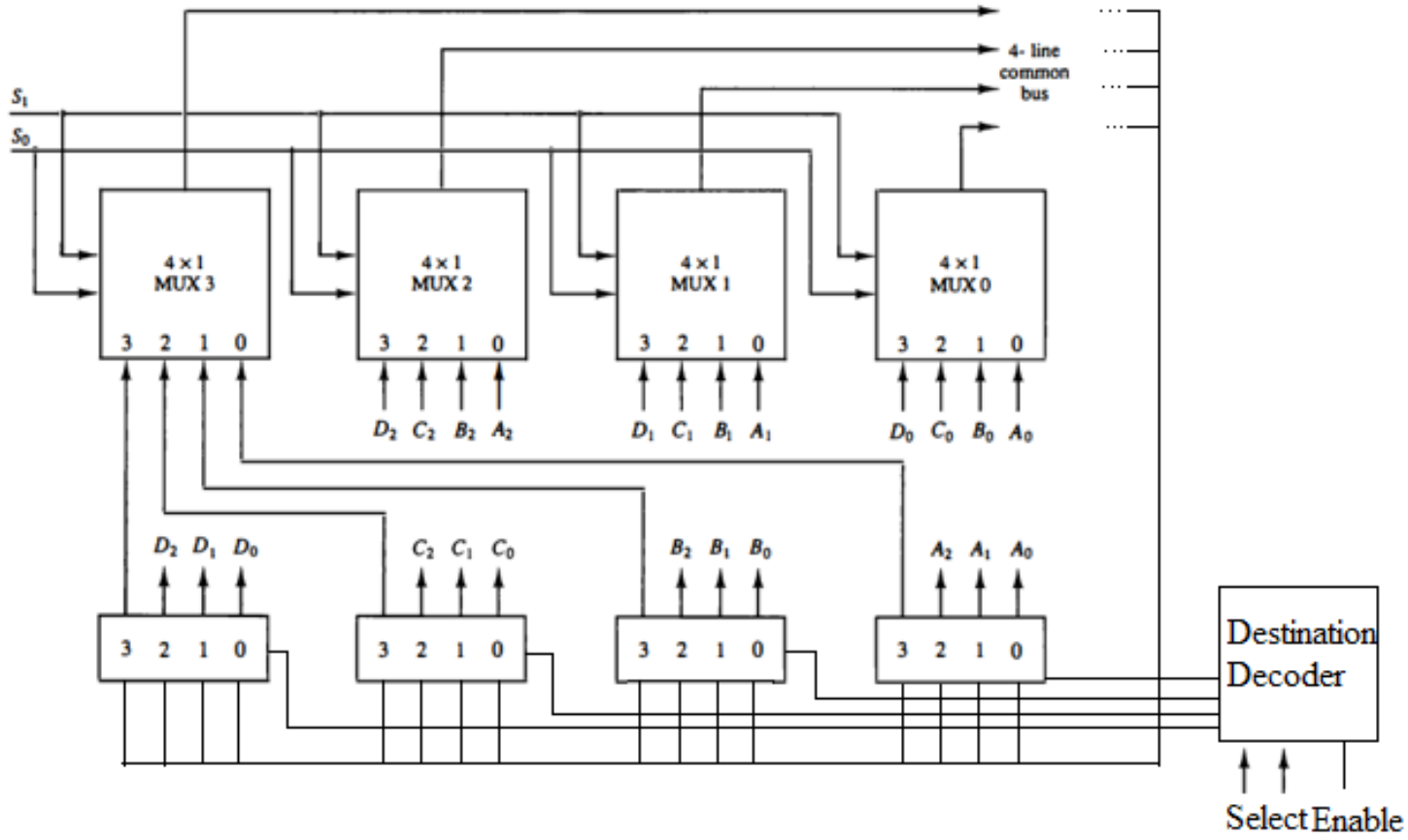
Interregister Transfer

- We need a quadruple 2-to-1 multiplexer in order to select register A or register B.



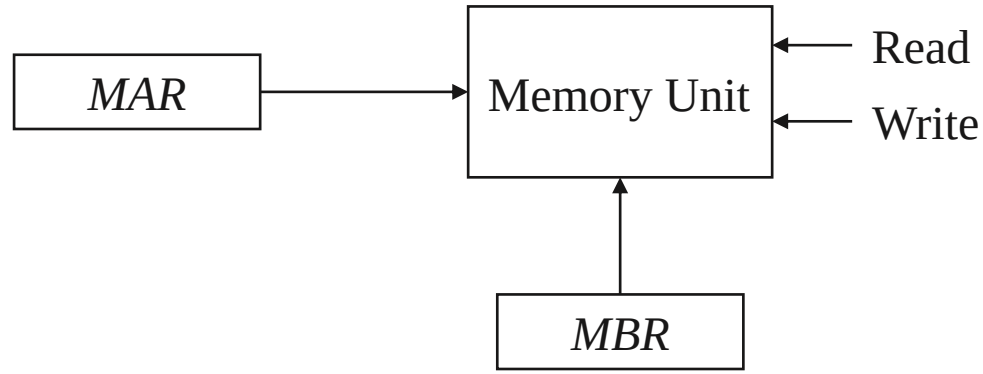
Interregister Transfer

- For registers of *n* bits, *n* multiplexers are needed to produce an *n*-line bus.
- The *n* lines in the bus are connected to the *n* inputs of all registers.
- Select source = 00 (Register A)
- Select destination = 10 (Register C)



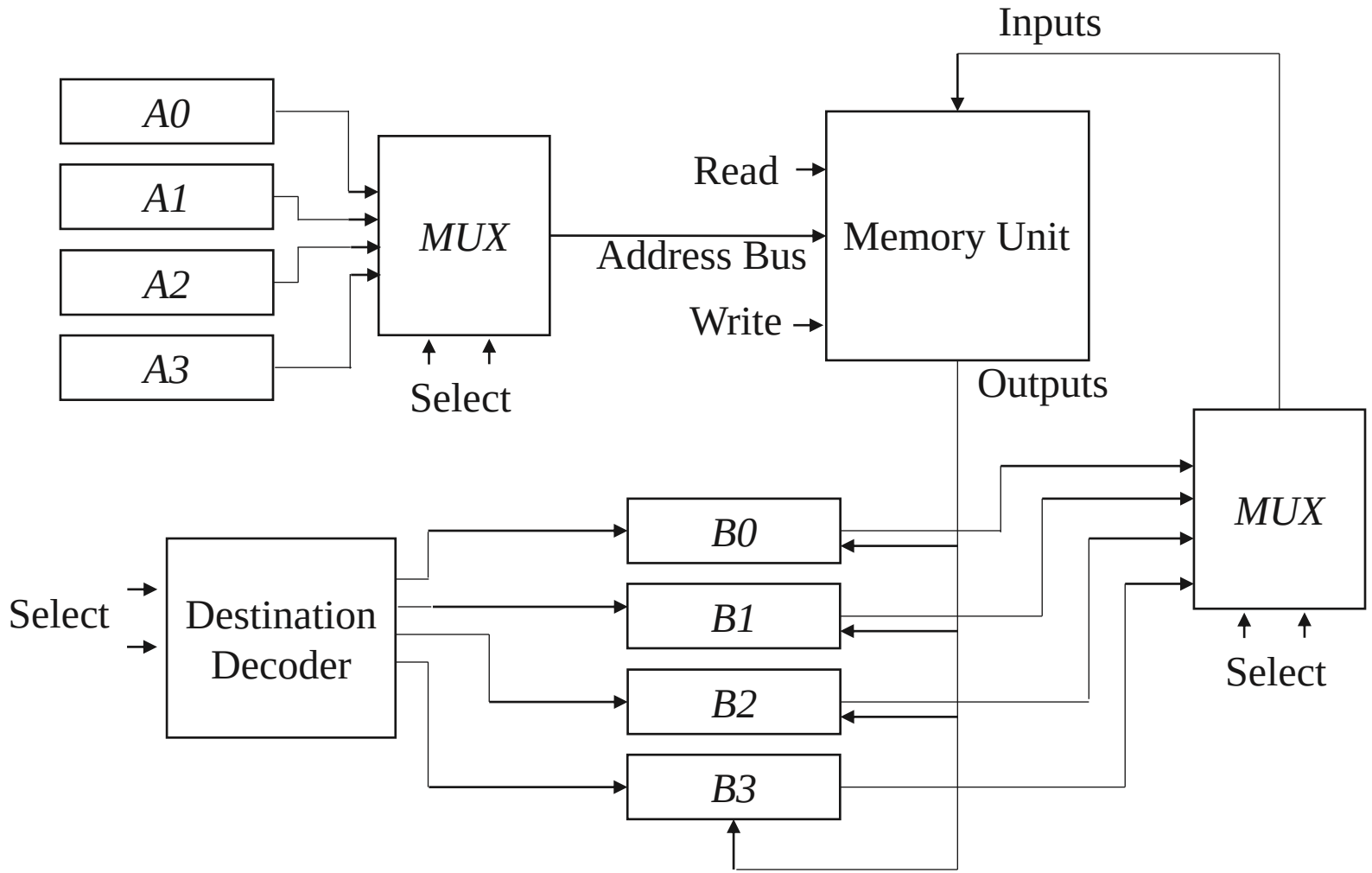
Interregister Transfer

- A memory register or word is symbolized by the letter *M*.
- Consider a memory unit that has a single address register, *MAR*.
- A single memory buffer register MBR is used to transfer data into and out of the memory.
- *R*: *MBR* ← *M* *R* is the control function that initiates the read operation.
- *W*: *M* ← *MBR* *W* is the control function that initiates the write operation.



Interregister Transfer

- *W*: *M*[*A1*]←*B2*
- *R*: *B0* ←*M*[*A3*]



Fixed-Point Binary Data

- A register with *n* flip-flops can store a binary number of *n* bits; each flip-flop represents one binary digit.
- The sign of a number is a discrete quantity of information having two values: plus and minus.
- These two value can be represented by a code of one bit.
  - 0 for plus
  - 1 for minus
- To represent a sign binary number in a register, we need $n = k + 1$ flip-flops, k flip-flops for the magnitude and one for storing the sign of the number.

Signed Binary Numbers

- There are three different ways of representation:
- Sign-magnitude
- Sign-1’s complement
- Sign-2’s complement
- For example: the binary number 9 is written below in the three representations

	+9	–9
Sign-magnitude	0 001001	1 001001
Sign-1’s complement	0 001001	1 110110
Sign-2’s complement	0 001001	1 110111

Arithmetic Subtraction

$$(\pm A) - (-B) = (\pm A) + (+B)$$

$$(\pm A) - (+B) = (\pm A) + (-B)$$

- Changing a positive number to a negative number is easily done by taking its 2's complement.
- The subtraction with 1's complement numbers is similar except for the end-around carry.
- To avoid the end-around carry and the occurrence of a negative zero, we choose 2's complement over 1's complement.

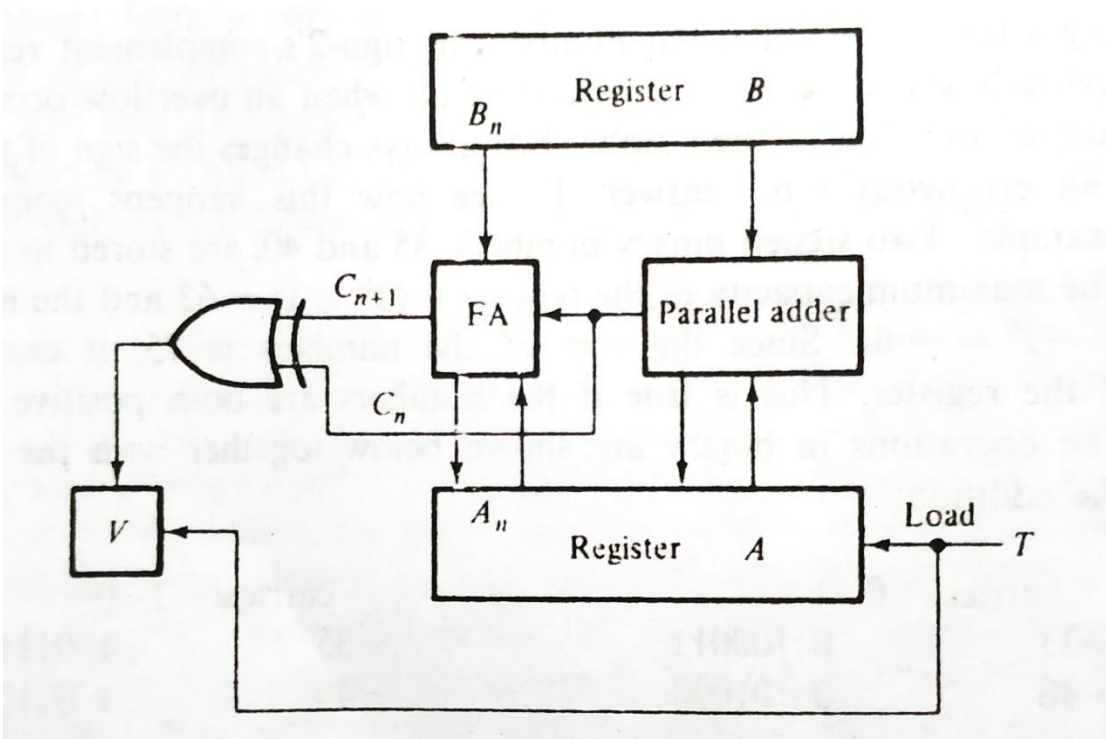
Overflow

- Two signed numbers, 35 and 40 are stored in two 7-bit registers.
- The maximum capacity of the register is $(2^6 – 1) = 63$ and the minimum capacity is $– 2^6 = – 64$.
- Since the sum of the numbers is 75, it exceeds the capacity of the register.
- This overflow happens when both numbers are positive or negative.

+35	0	100011	– 35	1	011101
+40	0	101000	– 40	1	011000
+75	1	001011	– 75	0	110101

Overflow

- An overflow condition can be detected by observing the carry into the sign-bit position and the carry out of the sign-bit position.
- If these two carries are not equal, an overflow condition is produced.



Instruction Codes

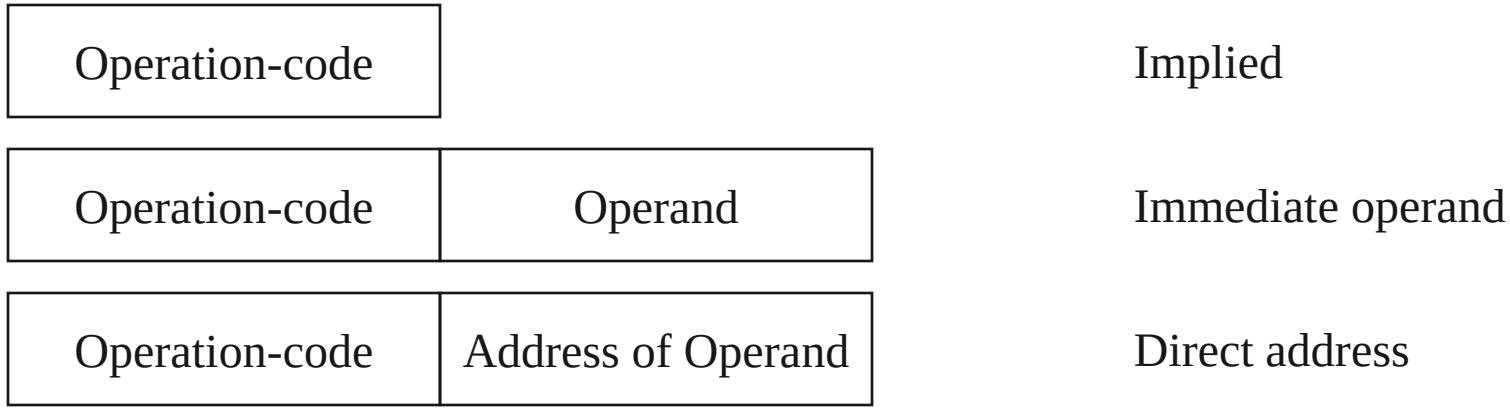
- A program is a set of instructions stored in memory.
- The control reads each instruction from memory and places it in a control register.
- The control interprets the instruction and proceeds to execute it by issuing a sequence of control functions.
- An instruction code is a group of bits that tell the computer to perform a specific operation.
- It is usually divided into parts, each having its own particular interpretation.
- The most basic part of an instruction code is its operation part which defines add, subtract, multiply, shift and complement.

Instruction Codes

- The number of bits required for the operation part of the instruction code is a function of the total number of operations used.
- It must consist of at least n bits for a given 2^n or less distinct operations.
- For example a computer using 32 distinct operations, one of them being an ADD operation.
- The operation code may consist of five bits, with a bit configuration 10010 assigned to the ADD operation.
- The instruction code must specify not only the operation but also the registers where the operands are to be found as well as the register where the result is to be stored.

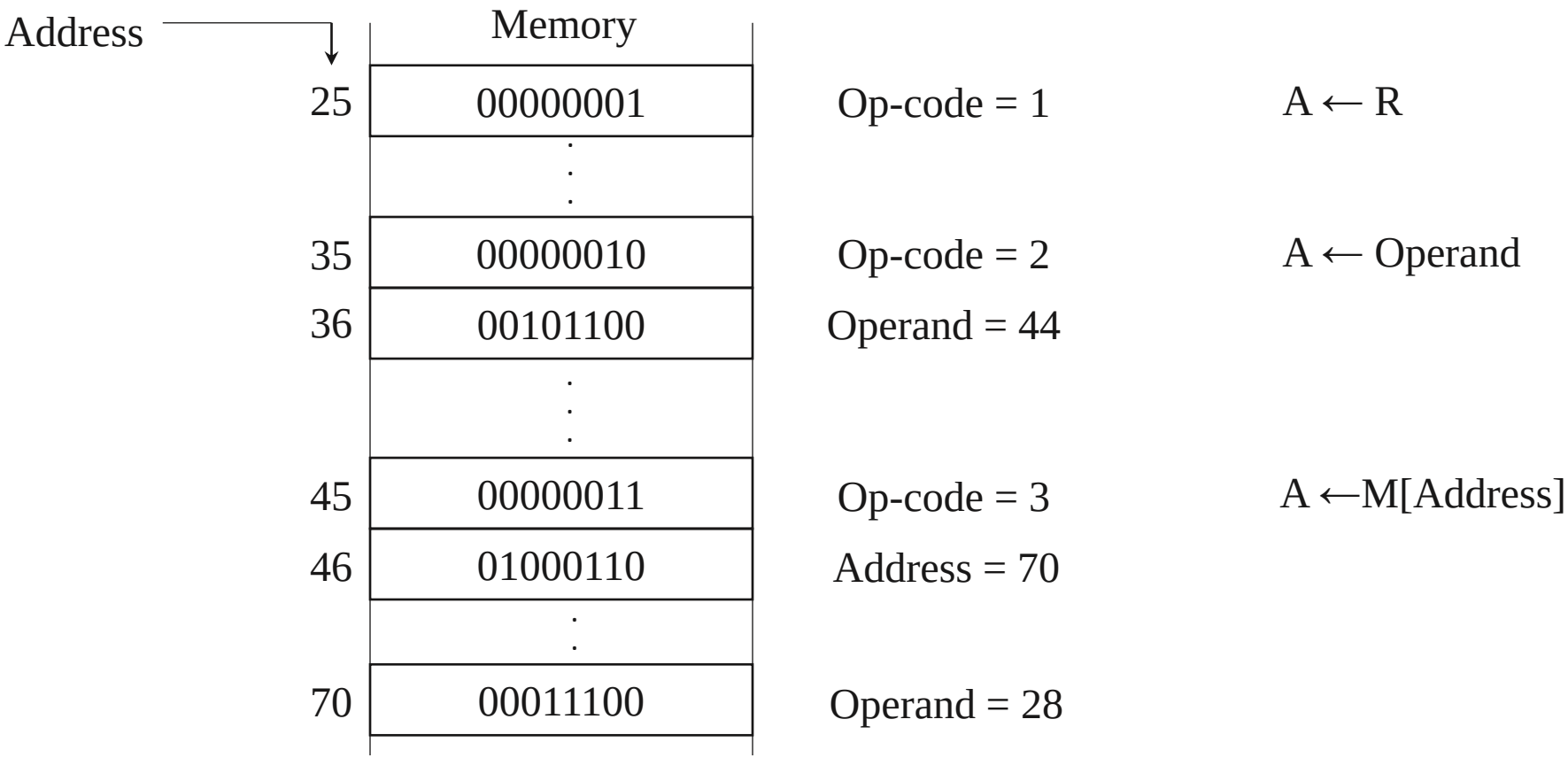
Instruction Codes Formats

- The bits of the instruction are sometimes divided into groups that subdivide the instruction into parts.
- Each group is assigned a symbolic name, such as *operation code* part or *address* part.



Instruction Codes Formats

- Let us assume that we have a memory unit with 8 bits per word and that an operation code contains 8 bits.



Macrooperations Vs Microoperations

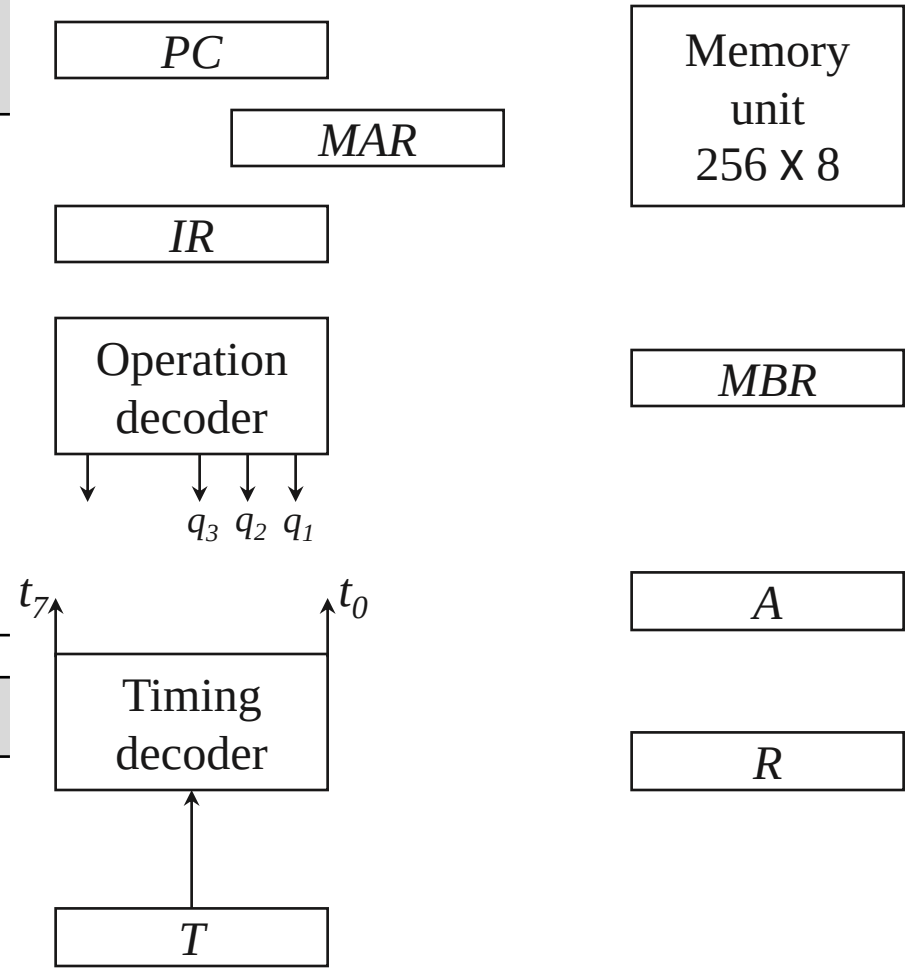
- A statement that requires a sequence of microoperations for its implementation is called a macrooperation.
- If the register-transfer statement can be executed with a single control function, it represents a microoperation.
- If the hardware execution of the statement requires two or more control functions, the statement is taken to be a macrooperation.
- For example: A ← operand

This statement is a macroperation because it specifies a computer instruction that performs several microoperations.

Design of a Simple Computer

Symbol	Number of bits	Name of register	Function
MAR	8	Memory address register	Holds address for memory
MBR	8	Memory buffer register	Holds contents of memory word
A	8	A register	Processor register
R	8	R register	Processor register
PC	8	Program Counter	Holds address of instruction
IR	8	Instruction register	Holds current operation code
T	3	Timing Counter	Sequence generator

Op-code	Mnemonic	Description	Function
00000001	MOV R	Move R to A	A ←R
00000010	LDI OPRD	Load OPRD into A	A ←OPRD
00000011	LDA ADRS	Load operand specified by ADRS into A	A ←M[ADRS]



Instruction Fetch Cycle

- Program counter *PC* must be initialized to contain the first address of the program stored in memory.
- The timing variables t_0, t_1 and t_2 out of the timing decoder are used as control functions to sequence the microoperations for reading an operation code and placing it into instruction register *IR*.

$t_0 : MAR \leftarrow PC$	transfer op-code address
$t_1 : MBR \leftarrow M, PC \leftarrow PC + 1$	read op-code, increment *PC*
$t_2 : IR \leftarrow MBR$	transfer op-code to *IR*

Execution of Instructions

- The control uses the q_i variables to determine the next microoperations in sequence.
- The MOV *R* instruction has an operation code that makes $q_1 = 1$.
- During timing variable t_3, the execution of this instruction requires:

$$q_1 t_3 : A \leftarrow R, T \leftarrow 0$$

- T is cleared to make the control go back to produce timing variable t_0.

Execution of Instructions

- The LDI OPRD instruction has an operation code that makes $q_2 = 1$.
- During timing variables t_3, t_4 and t_5 the execution of this instruction requires:

$q_2t_3 : MAR \leftarrow PC$	transfer operand address
$q_2t_4 : MBR \leftarrow M, PC \leftarrow PC + 1$	read operand, increment PC
$q_2t_5 : A \leftarrow MBR, T \leftarrow 0$	transfer operand, go to fetch cycle

T is cleared to make the control go back to produce timing variable t_0.

Execution of Instructions

- The LDA ADRS instruction has an operation code that makes $q_3 = 1$.
- During timing variables t_3, t_4, t_5, t_6 and t_7 the execution of this instruction requires:

$q_3t_3 : MAR \leftarrow PC$	transfer next address
$q_3t_4 : MBR \leftarrow M, PC \leftarrow PC + 1$	read ADRS, increment *PC*
$q_3t_5 : MAR \leftarrow MBR$	transfer operand address
$q_3t_6 : MBR \leftarrow M$	read operand
$q_3t_7 : A \leftarrow MBR, T \leftarrow 0$	transfer operand to *A*, go to fetch cycle

T is cleared to make the control go back to produce timing variable t_0.

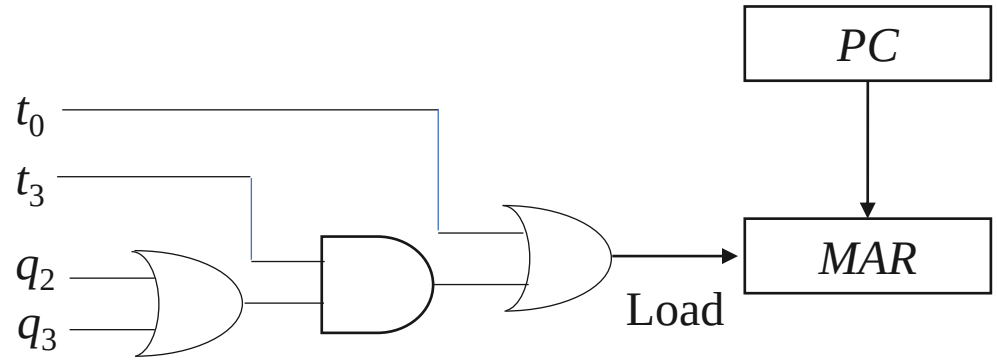
Execution of Instructions

FETCH	t_0 :	$MAR \leftarrow PC$
	t_1:	$MBR \leftarrow M, PC \leftarrow PC + 1$
	t_2:	$IR \leftarrow MBR$
MOV	$q_1 t_3$:	$A \leftarrow R, T \leftarrow 0$
LDI	$q_2 t_3$:	$MAR \leftarrow PC$
	$q_2 t_4$:	$MBR \leftarrow M, PC \leftarrow PC + 1$
	$q_2 t_5$:	$A \leftarrow MBR, T \leftarrow 0$
LDA	$q_3 t_3$:	$MAR \leftarrow PC$
	$q_3 t_4$:	$MBR \leftarrow M, PC \leftarrow PC + 1$
	$q_3 t_5$:	$MAR \leftarrow MBR$
	$q_3 t_6$:	$MBR \leftarrow M$
	$q_3 t_7$:	$A \leftarrow MBR, T \leftarrow 0$

Design of Computer

- Now we need to design a combinational circuit for the list of control functions and microoperations for the digital system.
- For example, in the previous table the instruction $MAR \leftarrow PC$ is listed with three control functions, t_0, q_2t_3, q_3t_3.
- The combined statement, x_1 is: $t_0 + q_2t_3 + q_3t_3 : MAR \leftarrow PC$

$x_2 = q_3t_5$	$MAR \leftarrow MBR$
$x_3 = t_1 + q_2t_4 + q_3t_4$	$PC \leftarrow PC + 1$
$x_4 = x_3 + q_3t_6$	$MBR \leftarrow M$
$x_5 = q_2t_5 + q_3t_7$	$A \leftarrow MBR$
$x_6 = q_1t_3$	$A \leftarrow R$
$x_7 = x_5 + x_6$	$T \leftarrow 0$
$x_8 = t_2$	$IR \leftarrow MBR$



Design of Computer

- In some cases a register that receives information from two sources needs a multiplexer to select between the two.
- *MAR* receives information from *MBR* and *PC*. When $x_1 = 1$, it receives from *PC* but transfers the contents of *MBR* when the select line is 0.

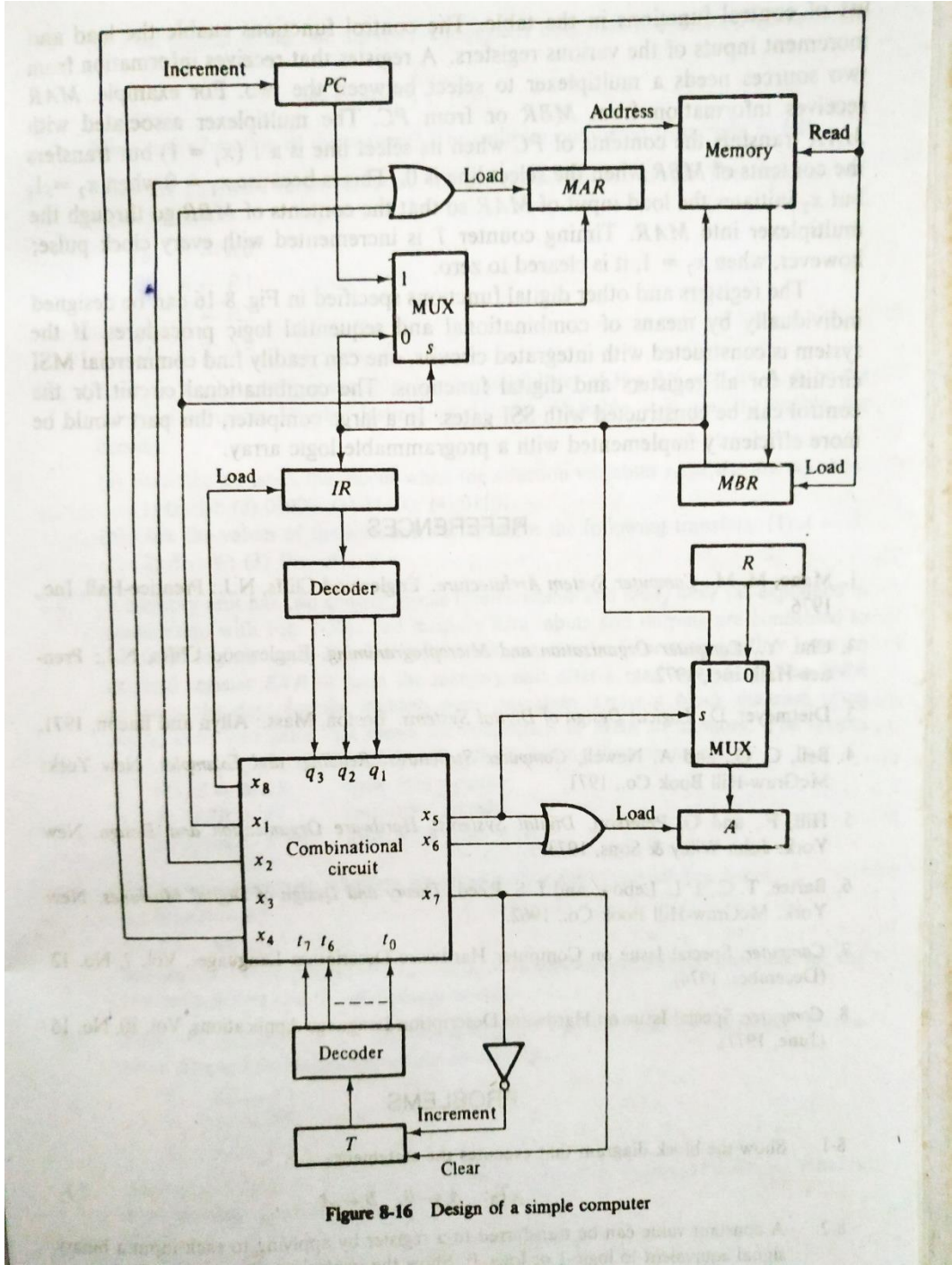


Figure 8-16 Design of a simple computer