Microoperations and Register Transfer Languages

Review Notes

Yul Williams, D.Sc.

Goals

- At the conclusion of this session, the student should be able to:
  - Understand the concept of microoperations
  - Model simple logic functions using RTL

Outline

- RTL Specifics - Microoperations
- Specifying Digital Systems Using RTL

The Digital Computer·

- Digital Computer = Registers + Microoperations Hardware + Control Functions
- The internal hardware organization of a digital computer is defined by specifying:
    - The set of registers it contains and their function
    - The sequence of microoperations performed on the binary information stored in the registers
    - The control that initiates the sequence of microoperations

Individual Operations

- Individual instructions (e.g. assembly language instructions) may be viewed as a sequence of smaller (micro) operations
- A finite state machine, such as our Moore and Mealy sequence detectors, are comprised of micro-operations
- This is true of the fetch, decode, and execute operations of any CPU

Microoperations

- Microoperations are the steps required to perform a larger operation
- A larger operation, such starting your car, may consist of several smaller operations
  - Find key, insert key into ignition, pat the gas and turn the key
  - These smaller operations may be viewed as micro-operations of the larger “starting your car” operation
- A Register Transfer Language (RTL) is used to specify a micro-operation

Microoperations

- Microoperations: operations executed on data stored in one or more registers.
- For any function of the computer, a sequence of

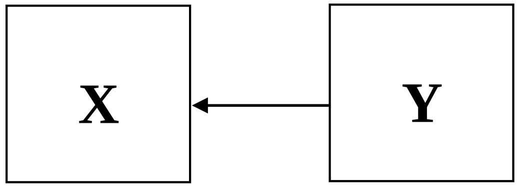
  microoperations is used to describe it
- The result of the operation may be:
  - replace the previous binary information of a register or
  - transferred to another register

Microoperations and RTL

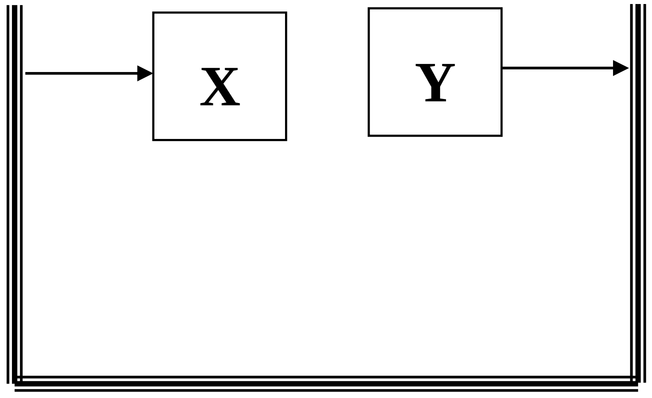
- Typical micro-operations result in data values being stored in specific registers or memory locations
- This is true of Load, Move and Store operations.
  - LDAC 2000H; Load the data at memory location 2000H into the Accumulator
  - MVI B, #5; Move the (actual value) 5 into Register B
  - STAC 2005H; Store contents of the Accumulator into the memory location 2005H
- Note: Each microoperation defines a single register transfer or register-level operation

Simple RTL Example

- Designers of digital systems must specify how the desired system behaves
- This specification becomes the guideline by which the final digital system is realized
- If our system is deigned to copy 1 bit of data from the Y register to the X register, we can notate it as: X ← Y



Direct Connection



BUS Connection

The Data Path

- The Data Path consists of the connections between components in the system that are used to transport data
- In a computer system, the parallel collection of wires composing the data bus is a data path
- The registers that store transferred information is also considered to be a part of the data path
- The data path may be a simple direct connection between components, or, it may consist of a complex collection of wires, registers, buffers and other logic components necessary to transfer data

RTL Specifics

RTL

- RTL may be used to describe functions and behavior of digital systems
- The systems may be extremely simple to very complex

Register Transfer Language (RTL)

- RTL is a symbolic notation to describe the microoperation transfers among registers
- RTL:
  - Defines symbols for various types of microoperations,
  - Describes the hardware that implements these microoperations

RTL Basics

- All components are connected to a common clock signal
- A register transfer language statement has a one-to-one correlation with hardware connections
- Registers within the system are denoted by capital letters
  - MAR: Memory Address Register
  - DR: Data Register
  - IR: Instruction Register
  - AC: Accumulator
  - PC: Program Counter

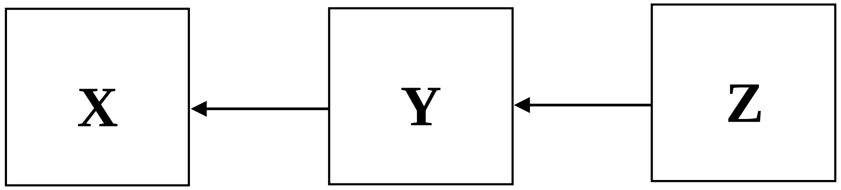
Simple Data Transfers

- We have taken a look at simple data transfer operations from register to register and register to memory
- In order to make microoperations really useful, however, we must make the transfers occur at specific times or under specific conditions

Conditional Data Transfers

- What if we wanted to make the micro-operation X $\leftarrow$ Y occur only when the signal α is HI?
- We would write it as: If α: Then X $\leftarrow$ Y
- α represents a *condition* that governs the execution of specific microoperations
- In general, *conditions: microoperations* is the format that we will use to express our microoperations

Simultaneous Operations Example



α: X ← Y, Y ← Z

The contents of register Y are transferred to register X at the same time that the contents of register Z are transferred to register Y.

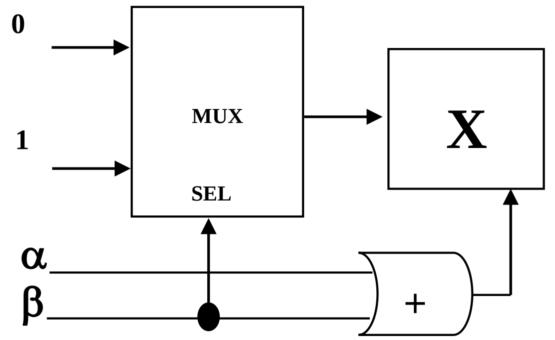
Transferring Constant Values

Sometimes, it is necessary to constant values into registers as opposed to moving data between registers. This kind of operation is specified in RTL as follows:

$$\alpha: X \leftarrow 0$$

$$\beta: X \leftarrow 1$$

Note that these micro-operations are sequential (they do not occur simultaneously). The example does not address the case when $\alpha = 1$ and $\beta = 1$, at the same time. This would cause the resulting value in the X register to non-deterministic. This example may be modified to provide a mutually exclusive condition for data transfer.



Ranges

- Individual signals may be referred to by their respective subscripts
- This may become tedious when a large number of individual signals are expressed
- To address this issue, we may elect to use ranges in RTL to specify groups of associated signals (such as address and data bus lines)
- Examples of ranges:
  - **α: X(3:1) $\leftarrow$ Y(2:0)**
  - **β: $X_3 \leftarrow X_2$**
- **In each example, the range notation was used to express contiguous set of signals in a specified order.**

Shifting Operations

- There are four basic types of shifting micro-operations. Each shift operation has two variations (left and right)
  - Linear Shift
  - Circular Shift
  - Arithmetic Shift
  - Decimal Shift
- Let’s explore each of these shifting types…

Linear Shift

The Linear Shift is the simplest shift operation. Let’s start by defining a 4-bit register X containing the initial value of 1011. A left shift operation on register X causes its new value to be X = 0110. Each time the contents of X are shifted left, the left-most bit is discarded and the rightmost bit is filled with the value of zero.

Starting from the initial value condition, a right shift operation on register X causes its new value to be X = 0101. Each time the contents of X are shifted right, the right-most bit is discarded and the left-most bit is filled with the value of zero.

Circular Shift

The Circular Shift is slightly different than the linear shift operation. Instead of bits being discarded, they are rotated around to the other side of the registered value.

Let's start by defining a 4-bit register X containing the initial value of 1011. A left shift operation on register X causes its new value to be X = 0111. Each time the contents of X are shifted left, the left-most bit is rotated around to become the least significant bit of the registered value.

Starting from the initial value condition, a right shift operation on register X causes its new value to be X = 1101. Each time the contents of X are shifted right, the right-most bit is rotated around to become the most significant bit of the registered value.

Arithmetic Shift

The Arithmetic Shift operated much like the linear shift operation with the exception of how one of the bits in the value is treated during the rotation of the bits. It uses the left-most bit of the registered value as the sign bit. This value remains static while the other bits participate in the shift operation.

Let's start by defining a 4-bit register X containing the initial value of 1011. A left shift operation on register X causes its new value to be X = 1110. Each time the contents of X are shifted left, the left-most bit (excluding the sign bit) is discarded and the least significant bit of the registered value is filled with a 0.

Starting from the initial value condition, a right shift operation on register X causes its new value to be X = 1001. Each time the contents of X are shifted right, the right-most bit is discarded and the most significant bit (excluding the sign bit) of the registered value, is filled with a 0.

Binary Coded Decimal

- When encoding decimal data, we sometimes use a notation referred to as Binary Coded Decimal or BCD
- In BCD notation, each decimal digit requires 4 bits of binary data
- Each byte of data can represent 2 decimal digits. The high nibble and the low nibble of the byte are each 4 bits long

BCD Notation

In order to represent the decimal number 41, the BCD representation becomes 0100 0001. Since we have 4 binary digits, our range of number that each nibble can represent is from 0 to 15. Since 4 binary digits can represent 16 numbers, each nibble covers the range of digits in the Hexadecimal number system. The hexadecimal number system is a base 16 number system that is frequently used in digital design and in programming operations. In BCD, however, we use only the first 10 numbers are 0 through 9 (just as in the decimal number system). Numbers 11 through 15 are ignored.

Binary	Decimal	BCD
0	0	0000
1	1	0001
10	2	0010
11	3	0011
100	4	0100
101	5	0101
110	6	0110
111	7	0111
1000	8	1000
1001	9	1001
1010	10	1010

Decimal Shift

The Decimal Shift uses BCD notation to shift 1 decimal value at a time. This translates to shifting 4 binary bits per shift operation. Let's start by defining a 8-bit register X containing the initial value of 1001 0111. A left shift operation on register X causes its new value to be X = 0111 0000. Each time the contents of X are shifted left, the right-most nibble is shifted into the left-most nibble and the right-most nibble is filled with the BCD value of 0000 .

Starting from the initial value condition, a right shift operation on register X causes its new value to be X = 0000 1001. Each time the contents of X are shifted right, the left-most nibble is shifted into the right-most nibble and the left-most nibble is filled with the BCD value of 0000 .

Summary of Shift Microoperations

Shift Microoperations	
Operation	**Notation**
Linear shift left	shl(X)
Linear shift right	shr(X)
Circular shift left	cil(X)
Circular shift right	cilr(X)
Arithmetic shift left	ashl(X)
Arithmetic shift right	ashr(X)
Decimal shift left	dshl(X)
Decimal shift right	dshr(X)

Summary

- Register Transfer Languages are important in helping to model a logic function without having to construct a physical implementation
- RTLs help to focus the attention of the designer on functional rather than implementation details