

# Computer Organization (Control Design)

KR Chowdhary  
Professor & Head  
*Email: kr.chowdhary@gmail.com*  
*webpage: krchowdhary.com*

Department of Computer Science and Engineering  
MBM Engineering College, Jodhpur

November 14, 2013

- ▶ **Control unit function:** To initiate sequences of micro-operations. The complexity of the digital system is derived from the number of sequences of micro-operations that are performed.
- ▶ Control can be viewed as **state machine** that changes from one state to another state in every clock cycle, depending on contents of various registers.
- ▶ Output of machine are control signals:  $c_0 \dots c_n$ .
- ▶ The sequence of operations carried out is decided by *wiring of logic* elements, hence the name “hard-wired control”.
- ▶ The controller that uses this approach can operate at high speed, but the cost and complexity of implementation is limitation.
- ▶ In micro-programmed control, the control signals are generated by a program similar to machine language program.

▶ **Instruction cycle =**

**Fetch Cycle**  $\Rightarrow$  No of Micro-operations

+

**Execute cycle**  $\Rightarrow$  No. of Micro-operations

▶ Let the micro-operations are represented by clock cycles:  $t_1, t_2, \dots,$

A fetch cycle may consist:

$t_1 : MAR \leftarrow (PC)$

$t_2 : MBR \leftarrow Memory$

$PC \leftarrow (PC) + \ell; (\ell = \text{instruction length})$

$t_3 : IR \leftarrow (MBR)$

▶ **Indirect cycle:** Micro operations for accessing address part after having fetched the Instruction along with indirect address.

$t_4 : MAR \leftarrow (IR(Addr))$

$t_5 : MBR \leftarrow Memory$

$t_6 : IR(Addr) \leftarrow (MBR(Addr))$

## ▶ Interrupt Cycle

$t_1 : MBR \leftarrow (PC)$

$t_2 : MAR \leftarrow Savedaddr$

$PC \leftarrow Subroutine.addr$

$t_3 : memory \leftarrow (MBR)$

## ▶ Execute cycle: Less predictable.

$ADD R_1, X; R_1 \leftarrow R_1 + X$

$t_1 : MAR \leftarrow (IR(Addr))$

$t_2 : MBR \leftarrow Memory$

$t_3 : R_1 \leftarrow (R_1) + (MBR)$  (More complex operations ?)

## ▶ ISZ X: Increment and skip next instruction, if result zero (PDP-8 memory reference instruction.) It is used for loop control with initial value of loop counter as negative.

initialize counter x to -15 (say)

...

loop: loop-body

isz x

jmp loop

instruction

...

hlt

$t_1 : MAR \leftarrow (IR(Addr));$

$t_2 : MBR \leftarrow Memory$

$t_3 : MBR \leftarrow (MBR) + 1$

$t_4 : Memory \leftarrow (MBR)$

if  $((MBR) = 0)$  then  $(PC = (PC) + 1)$ ; performed as single micro-operation.

- ▶ Subroutine call:  
**Branch-and-save-address  
Instruction: *BSA X***
- ▶ Return address of next instruction is saved at  $X$ , and execution start at  $X + 1$ .

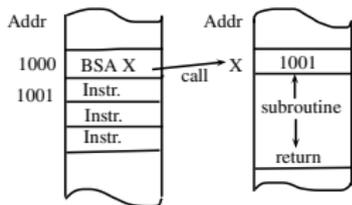


Figure 1: Micro-operations for BSA.

$t_1 : MAR \leftarrow (IR(addr));$  move  $X$  into  $MAR$

$MBR \leftarrow (PC);$  move 1001 into  $MBR$

$t_2 : PC \leftarrow (IR(addr));$  move  $X$  into  $PC$

$Memory \leftarrow (MBR);$  save  $MBR$  (i.e 1001) at  $X$

$t_3 : PC \leftarrow (PC) + 1;$  point  $PC$  to called subroutine's code

- ▶ Can the subroutines be nested using this?
- ▶ Can the subroutine be called recursively using this approach?
- ▶ What is disadvantage (if any) of this approach to subroutine call?

- ▶ Each phase of instruction is decomposed into sequence of elementary operations (micro-ops)
- ▶ By defining the operations of processor in terms of these micro-ops, we can define what control unit should exactly do.
- ▶ **Implementation of micro-ops = design of Control unit.**
- ▶ All micro-operations fall into one of these categories:

**R-R:** register-register data transfer

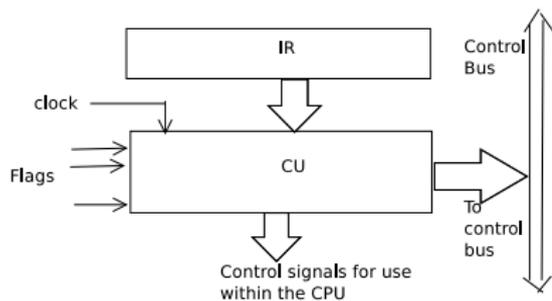
**R-bus:** register-internal data bus data-transfer

**R-extbus:** register-external bus data-transfer

Performs arithmetic and logic operations using registers

- ▶ **Two functions of CU:**
  1. **Sequencing:** Causes the processor to step through sequence of micro-ops,
  2. **Execution:** CU causes each micro-operation to be performed.

- ▶ CU must have the logic to perform the **sequencing** and **execution** of micro-operations



Cntrl signals for use within CPU: R-R, ALU functions  
Cntrl Signals for external cpu use: Read request to Memory, IO, etc.

Figure 2: Control unit

- ▶ **Flags:** These are required by CU to determine the status of processor, and outcome of previous operations. In ISZ instruction, the CU will increment PC if zero **flag** is set.
- ▶ **Output of CU:**
  1. Control signals can be used with in CPU,
  2. Control signals can be used to control the bus.
- ▶ **All these control-signals are directly applied to binary gates**

► A fetch cycle:

$t_1$  :  $MAR \leftarrow (PC)$ ; open the gate(s) to let data to move:  $c_6$

$t_2$  :  $MBR \leftarrow Memory$ ; open the gate(s):  $c_2$

$PC \leftarrow (PC) + 1$ ; not shown in this fig.

$t_3$  :  $IR \leftarrow (MBR)$ ; open the gate(s):  $c_{12}$

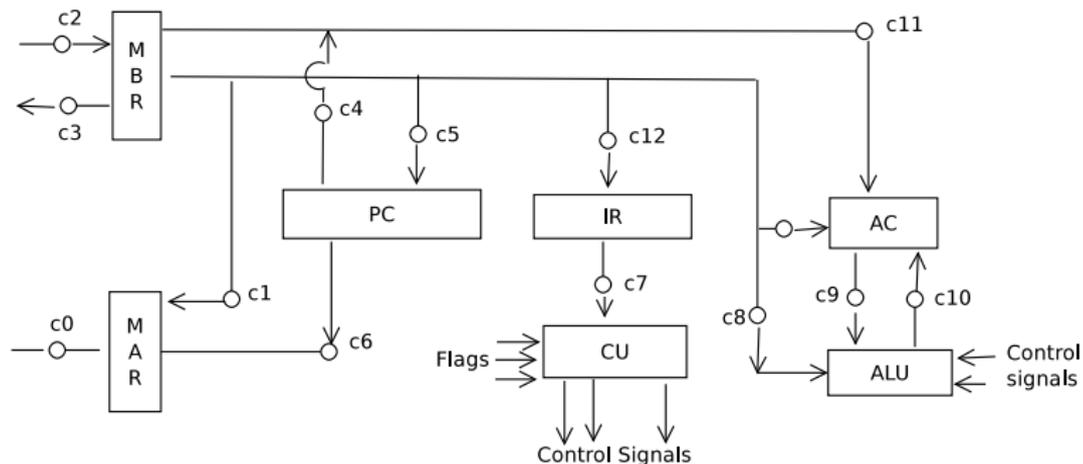


Figure 3: Control signals  $c_0$  :  $c_{12}$  control the data flow.

# Hardwired Control Implementation

- ▶ Each control signal is generated as function of inputs consisting flags and output from Instruction decoder,  $\therefore c_i = f(\text{signals, flags, timingsignals } T_i)$ . These control lines are making internal control as well as the external control bus.
- ▶ In Hard-wired control, CU acts a *state machine*. CU makes use of opcode from IR to generate signals for each instruction.

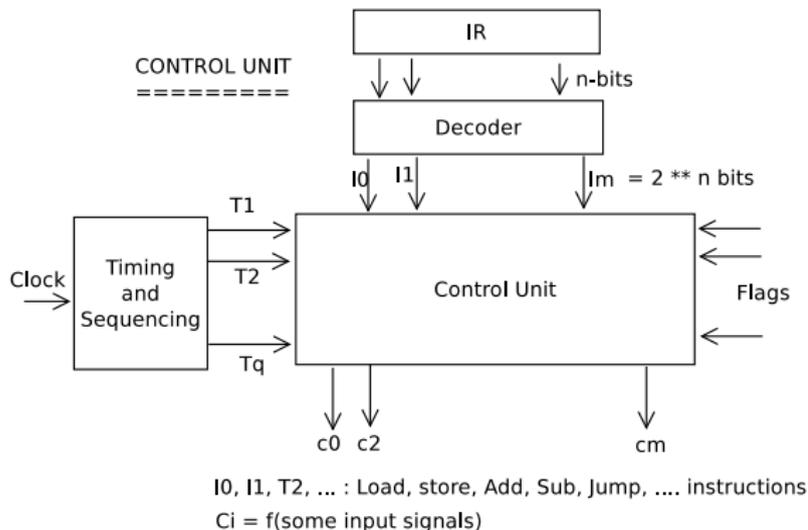


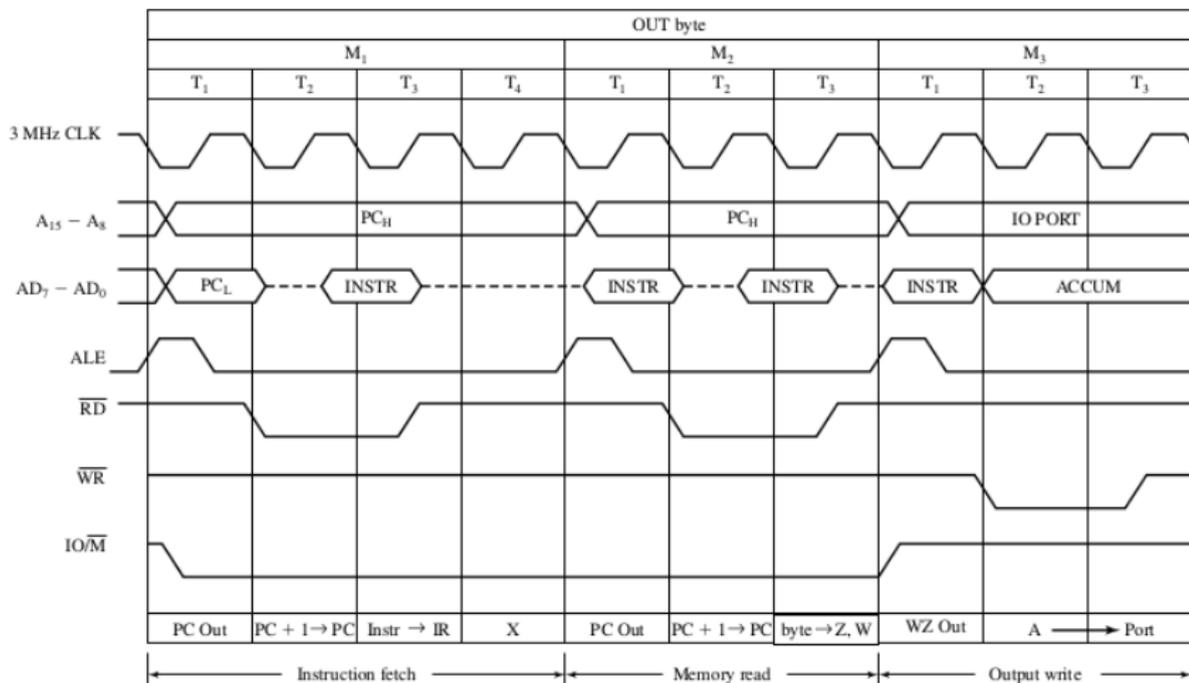
Figure 4: Generating Control signals  $C_i$  through control unit.

Let control signals are  $p, q$ . And, let  $pq = 00 = \text{fetch cycle}$ ,  $pq = 01 = \text{indirect cycle}$ ,  $pq = 10 = \text{execute cycle}$ ,  $pq = 11 = \text{interrupt cycle}$ . Then a control line, say  $C_2$  (to read data from external bus) can be generated by:  $C_2 = \bar{p}\bar{q}.T_2 + \bar{p}q.T_2$ . Note: see previous slides for the position of  $C_2$ . If control lines are generated like this, then it is called **hardwired control**.

- ▶  $T_1, T_2, T_3, \dots$ , are different control signals. Let there be control signals  $l_0, l_1$  from decoder, representing  $LDA, ADD$  opcodes, respectively, which signal memory-read when the instruction is executed. Given this, the control signal  $C_2$  can be generated by the expression:

$$C_2 = \bar{p}\bar{q}.T_2 + \bar{p}q.T_2 + p\bar{q}(LDA + ADD).T_2$$

- ▶ In addition, the clock cycle must be long enough to allow for propagation of signals along the data bus.
- ▶ **Characteristics of Hard-wired control:**  
Complex, fast, difficult to design, and difficult to modify. In addition, lots of optimization is required during the implementation phase.



**Figure 5:** Timing diagram of Instruction of 8085 OUT instruction. Z is output temporary register of ALU. What ever is single input to Acc. it goes to Z, as acc. does not hold a value.

- ▶ The control signals are generated by a program similar to machine language programs.
- ▶ *Control word(CW)*: Its individual bits, called *binary variables*, represent various control signals. When it is in the true state, the corresponding micro-operation is performed.
- ▶ In a **bus system** control signals specify micro-operation as group of bits that select the path in multiplexers, decoders, and ALU.
- ▶ Number of micro-operations are finite. At one time only certain number of micro-operations are initiated.
- ▶ A sequence of control words corresponding to control sequence of a machine instruction constitute the microroutine for that instruction, and individual control words are referred to as *micro-instruction*.
- ▶ Micro-routines reside in *control memory*, and its micro-instructions are accessed by micro-program counter ( $\mu$ PC).
- ▶ Every time a new machine language instruction is loaded into IR, the  $\mu$ PC is initiated with new address by starting address generator.
- ▶ This causes the successive micro-instructions to be read from CM.

- ▶ **Control words, i.e. micro-instructions** can be programmed to perform various operations on the components of the system.
- ▶ Control words are interpreted by the microprogram control unit.
- ▶ A microinstruction may specify one or more micro-operations. Collection of micro-routines is called micro-program.

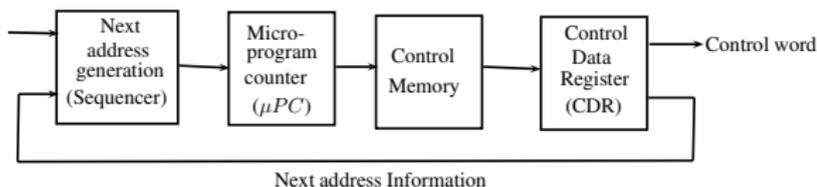


Figure 6: Microprogrammed Control organization

- ▶  **$\mu PC$** : Micro program counter specify the address of micro-instruction.
- ▶ **CDR**: Control Data Register loads the micro-instruction from control-memory.
- ▶ **Dynamic microprogram**: The micro-program is loaded initially, at the start of system, in a RAM.

- ▶ On execution of a micro-instruction, it determines the next address.
- ▶ Microprogrammed system's Flexibility and Speed?
- ▶ RISC should use what control? HW/microprogrammed?

### Address sequencing:

- ▶ Each machine instruction has its own micro-program routine in control-memory to generate micro-operations.
- ▶ A **mapping** process translates the machine instruction code into micro-program routine address,
- ▶ Micro-instructions are sequenced by incrementing micro-program counter( $\mu PC$ ), or computed by status bits (for jump)
- ▶ On execution of control routine, control is transferred to fetch routine. This is by unconditional branch microinstruction to first address of the fetch routine.
- ▶ The conditional branching is decided by status bits(C, S, Z, etc), which are stored in some register.
- ▶ The status bits together with branch address, control the conditional branch decision generated by branch logic.

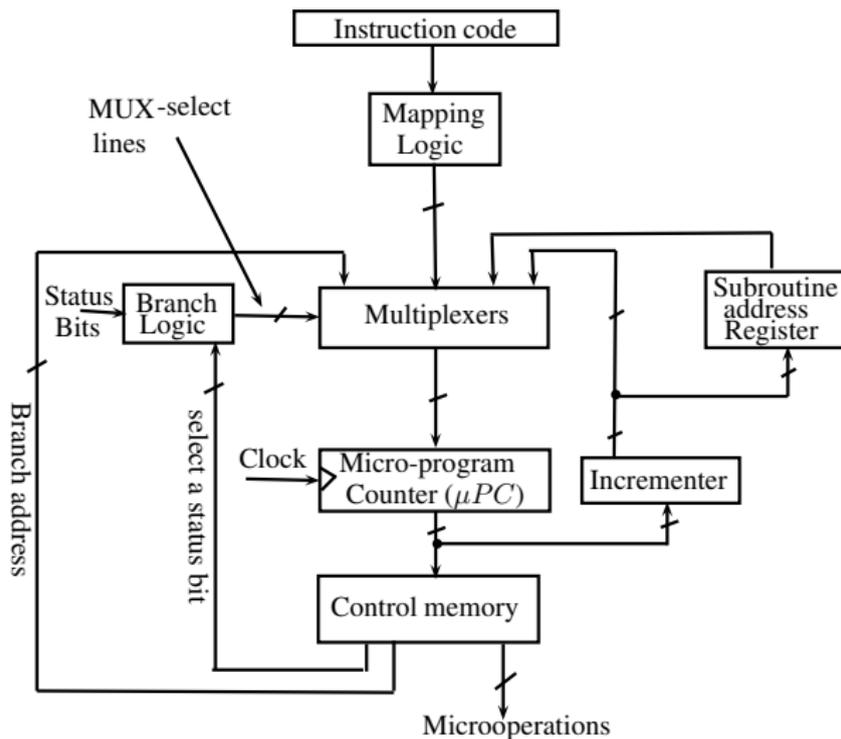


Figure 7: Selection of address for control memory.

- ▶ **Tasks Done By Microprogrammed Control Unit:**
  1. Microinstruction sequencing
  2. Microinstruction execution
  3. Must consider both together
- ▶ **Advantages and Disadvantages:** Simplifies design of control unit, Cheaper, Less error-prone, Slower.
- ▶ **Design Considerations:**
  1. Size of microinstructions
  2. Address generation time
  3. Determined by instruction register(Once per cycle, after instruction is fetched)
  4. Next sequential address(Common in most designs)
  5. Branches(Both conditional and unconditional)

- ▶ **Characteristics of micro-programmed control:**
  1. Machine instructions' fetch execute cycle produce machine instructions to be executed at cpu
  2. Micro-instructions fetch-execute cycle produce control signals for data path.
  3.  $\mu$ Program is stored in control memory: ROM, PROM, EPROM.
  4. One "subroutine" for each machine instruction (one or more micro-instructions)
  5. Next micro-instruction address is stored in micro-program counter.
  6. Micro-program defines architecture. To change instruction set, we reload control memory by different micro-program.
- ▶ **Vertical micro-programming:** Each micro-instruction specifies single (or few) micro-operations to be performed.
- ▶ **Horizontal micro-programming:** Each micro-instruction specifies many different micro-operations to be performed in parallel.

## ▶ Vertical Micro-programming:

1. Width is narrow
2. Control signals encoded into  $\log_2 n$  bits
3. Limited ability to express parallelism
4. Considerable encoding of control information
5. Requires external memory word decoder to identify the exact control line being manipulated. (see fig. below)

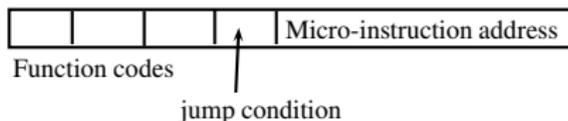


Figure 8: Vertical Microprogrammed Control

## ▶ Horizontal Micro-programming:

1. Wider memory word
2. High degree of parallel operations possible
3. Little encoding of control information

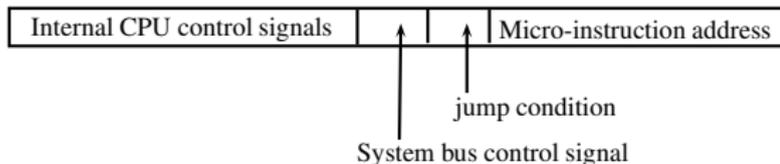


Figure 9: Horizontal Microprogrammed Control

- ▶ Parallel:
  1. Divides control signals into disjoint groups
  2. Implement each group as separate field in memory word
  3. Supports reasonable levels of parallelism without too much complexity
- ▶ Criteria for microprogram word length:
  1. Maximum number of simultaneous micro-operations supported
  2. The way control information is represented or encoded
  3. The way in which the next micro-instruction address is specified

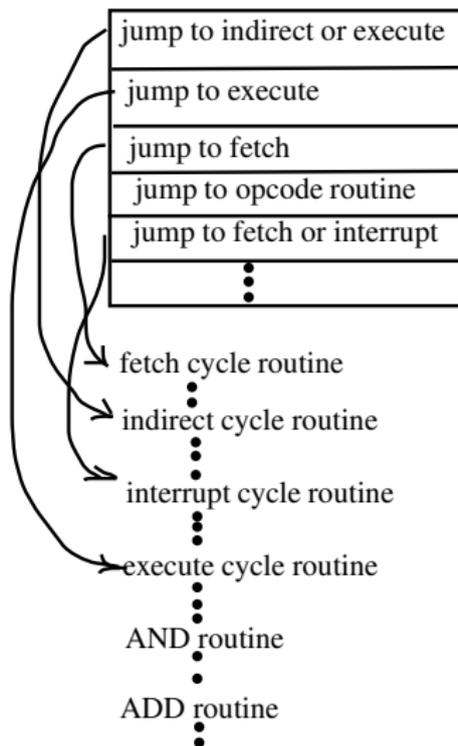


Figure 10: Control Memory

- ▶ Pure hardwired control v/s Pure micro-programmed control design sequences

Pure hardwired control: steps for design

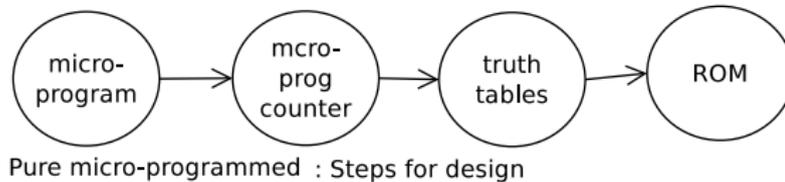
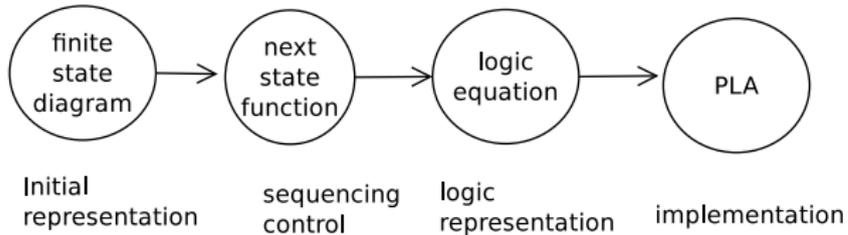


Figure 11: Hardwired v/s m-programmed control design steps

1. A microprogrammed control organization shown in figure 6 has the following propagation delay times: 40 ns to generate next address, 10 ns to transfer the address into the control address register, 40 ns to access the control memory ROM, and 40 ns to perform the required microoperations specified by the control word. What is the maximum frequency that the control can use? What is the maximum clock frequency be if the control data register is not used?
2. The system shown in figure 7 uses a control memory of 1024 words of 32 bits each. The microoperations field has 16-bits.
  - 2.1 How many bits are there in the branch address field and the select field?
  - 2.2 If there are 16-status bits in the system, how many bits of the branch logic are used to select a status bit?
  - 2.3 How many bits are left to select an input for the multiplexers?
3. The control memory shown in figure 7 has 4096 words of 24 bits each.
  - 3.1 How many bits are there in the control address register?
  - 3.2 How many bits are there in each of the four inputs shown going into the multiplexers?
  - 3.3 What is the number of inputs in each multiplexer and how many multiplexers are needed?

4. Explain how the mapping of from an instruction code to a microinstruction address can be done by means of a read only memory?
5. Write a symbolic micro-program routine for the ISZ X instruction.
6. Show how a 9-bit micro-operation field in a microinstruction can be divided into subfields to specify 46 microoperations. Maximum how many micro-operations can be specified in one micro-instruction?
7. A computer has 8 registers, an ALU with 32 operations, and a shifter with 8-operations, all connected to a common bus system. Design a hard-wired control for this.
8. A computer has 8 registers, an ALU with 32 operations, and a shifter with 8-operations, all connected to a common bus system.
  - 8.1 Formulate a control-word for a micro-operation.
  - 8.2 Specify the number of bits in each field of control word and give a general encoding scheme.
  - 8.3 Show the bits of control word that specify the microoperations  $R_4 \leftarrow R_5 + R_6$ .

-  John P. Hayes, "Computer Architecture and Organization", 2nd Edition, McGraw-Hill, 1988.
-  William Stalling, "Computer Organization and Architecture", 8th Edition, Pearson, 2010.
-  M. Morris Mano, "Computer System Architecture", 3rd Edition, Pearson Education, 2006.
-  Carl Hamacher, Zvono Vranesic, and Safwat Zaky, "Computer Organization", , 5th edition, McGrawhill Education, 2011. (chapter 7)
-  <http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-823-computer-system-architecture-fall-2005/lecture-notes/> (for reference)