Course Contents

Unit-01: Introduction to Microprocessor

Unit-02: Intel 8085

Unit-03: Microoperations

Unit-04: Control Unit and Central Processing Unit

Unit-05: Fixed point Computer Arithmetic

Unit-06: Input and Output Organization

Unit-07: Memory Organization

Unit-08: Pipelining

Course Contents

Unit-04: Control Unit and Central Processing Unit

- Control Unit of Basic Computer, Computer Arithmetic (Adder, Subtractor, Divider, and Multiplicator), Timing Signal
- Micro-Instruction and Micro-Operation Format, Symbolic Microinstructions, Symbolic Micro-program, Binary Micro-Program
- Register Organization, Register Stack and Memory Stack,
- Data Transfer operations and Manipulation (Arithmetic, Logical, Shift)

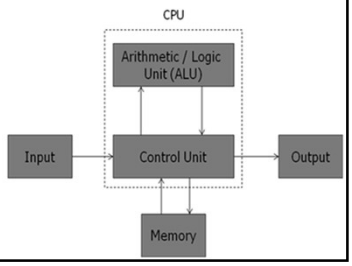
Course Contents

Unit-04: Control Unit and Central Processing Unit

Unit 4:	Control Unit and Central Processing Unit	9 Hours
4.1	Control Unit of Basic Computer and Timing Signal (Hardwired Vs. Microprogrammed)	2 Hour
4.2	Micro-operation, Micro-instruction, Micro-program: Symbolic and Binary Micro-program (FETCH and ADD)	3 Hour
4.3	Architecture of Basic Computer: Register organization, Common Bus System Instruction Format, Register Stack and Memory Stack	2 Hour
4.4	Data Transfer operations and Manipulation (Arithmetic, Logical, Shift)	1Hour
4.5	Introduction to RISC and CISC (Basic Differences)	1 Hour

Unit- 4

- Control Unit of Basic Computer and Timing Signal: (Hardwired Vs. Microprogrammed)
- Micro-operation, Micro-instruction, Micro-program: Symbolic and Binary Micro-program (FETCH and ADD)
- Architecture of Basic Computer: Register organization, Common Bus System, Instruction Format, Register Stack and Memory Stack
- Data Transfer operations and Manipulation (Arithmetic, Logical, Shift)
- Introduction to RISC and CISC (Basic Differences)



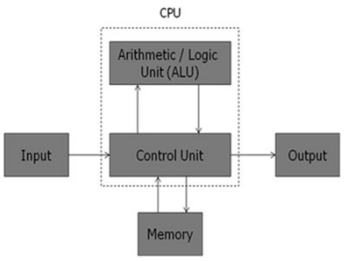
Introduction to computer

BASIC COMPUTER ORGANIZATION AND DESIGN:

- Every different processor type has its own design (different registers, buses, microoperations, machine instructions, etc)
- Modern processor is a very complex device
- It contains
  - Many registers
  - Multiple arithmetic units, for both integer and floating point calculations
  - The ability to pipeline several consecutive instructions to speed execution
  - Etc.
- However, to understand how processors work, we will start with a simplified processor model
- This is similar to what real processors were like ~25 years ago
- M. Morris Mano introduces a simple processor model he calls the Basic Computer

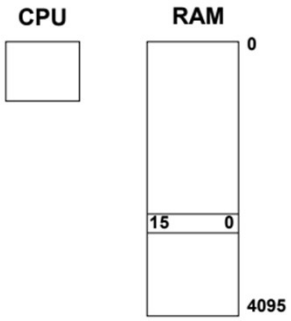
THE BASIC COMPUTER

The Basic Computer Architecture



THE BASIC COMPUTER

- The Basic Computer has two components, a processor and memory
- The memory has 4096 words in it
  - $4096 = 2^{12}$, so it takes 12 bits to select a word in memory
- Each word is 16 bits long



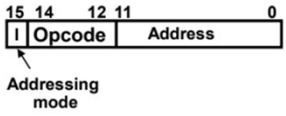
INSTRUCTIONS

- Program
  - A sequence of (machine) instructions
- (Machine) Instruction
  - A group of bits that tell the computer to *perform a specific operation* (a sequence of micro-operation)
- The instructions of a program, along with any needed data are stored in memory
- The CPU reads the next instruction from memory
- It is placed in an *Instruction Register* (IR)
- Control circuitry in control unit then translates the instruction into the sequence of microoperations necessary to implement it

INSTRUCTION FORMAT

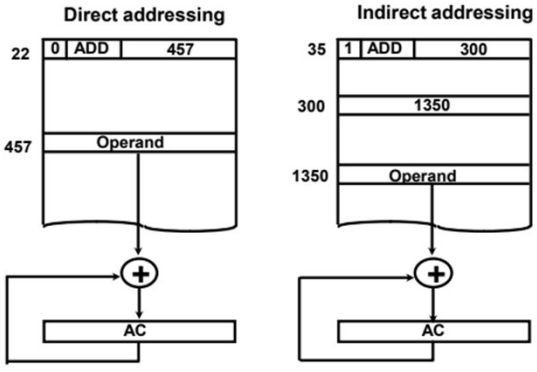
- A computer instruction is often divided into two parts
  - An *opcode* (Operation Code) that specifies the operation for that instruction
  - An *address* that specifies the registers and/or locations in memory to use for that operation
- In the Basic Computer, since the memory contains 4096 (= 2^{12}) words, we needs 12 bit to specify which memory address this instruction will use
- In the Basic Computer, bit 15 of the instruction specifies the *addressing mode* (0: direct addressing, 1: indirect addressing)
- Since the memory words, and hence the instructions, are 16 bits long, that leaves 3 bits for the instruction's opcode

Instruction Format



ADDRESSING MODES

- The address field of an instruction can represent either
  - Direct address: the address in memory of the data to use (the address of the operand), or
  - Indirect address: the address in memory of the address in memory of the data to use



- Effective Address (EA)
  - The address, that can be directly used without modification to access an operand for a computation-type instruction, or as the target address for a branch-type instruction

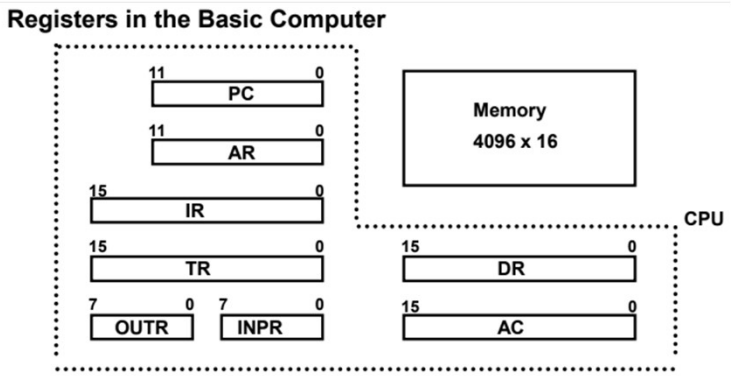
PROCESSOR REGISTERS

- A processor has many registers to hold instructions, addresses, data, etc
- The processor has a register, the *Program Counter* (PC) that holds the memory address of the next instruction to get
  - Since the memory in the Basic Computer only has 4096 locations, the PC only needs 12 bits
- In a direct or indirect addressing, the processor needs to keep track of what locations in memory it is addressing: The *Address Register* (AR) is used for this
  - The AR is a 12 bit register in the Basic Computer
- When an operand is found, using either direct or indirect addressing, it is placed in the *Data Register* (DR). The processor then uses this value as data for its operation
- The Basic Computer has a single *general purpose register* – the *Accumulator* (AC)

PROCESSOR REGISTERS

- The significance of a general purpose register is that it can be referred to in instructions
  - e.g. load AC with the contents of a specific memory location; store the contents of AC into a specified memory location
- Often a processor will need a scratch register to store intermediate results or other temporary data; in the Basic Computer this is the *Temporary Register* (TR)
- The Basic Computer uses a very simple model of input/output (I/O) operations
  - Input devices are considered to send 8 bits of character data to the processor
  - The processor can send 8 bits of character data to output devices
- The *Input Register* (INPR) holds an 8 bit character gotten from an input device
- The *Output Register* (OUTR) holds an 8 bit character to be send to an output device

BASIC COMPUTER REGISTERS



List of BC Registers

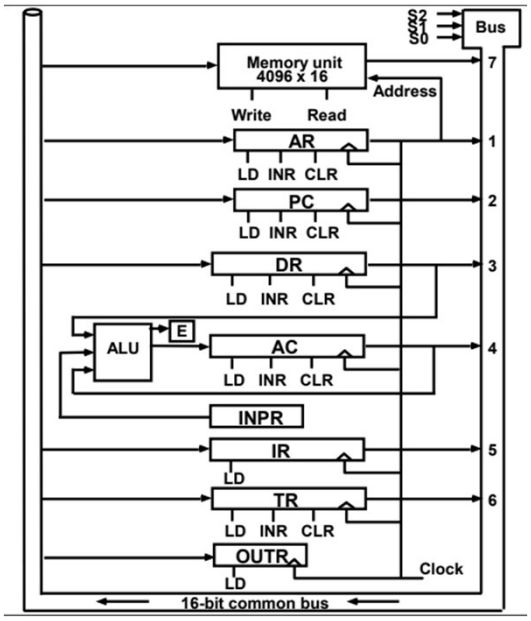
DR	16	Data Register	Holds memory operand
AR	12	Address Register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction Register	Holds instruction code
PC	12	Program Counter	Holds address of instruction
TR	16	Temporary Register	Holds temporary data
INPR	8	Input Register	Holds input character
OUTR	8	Output Register	Holds output character

COMMON BUS SYSTEM

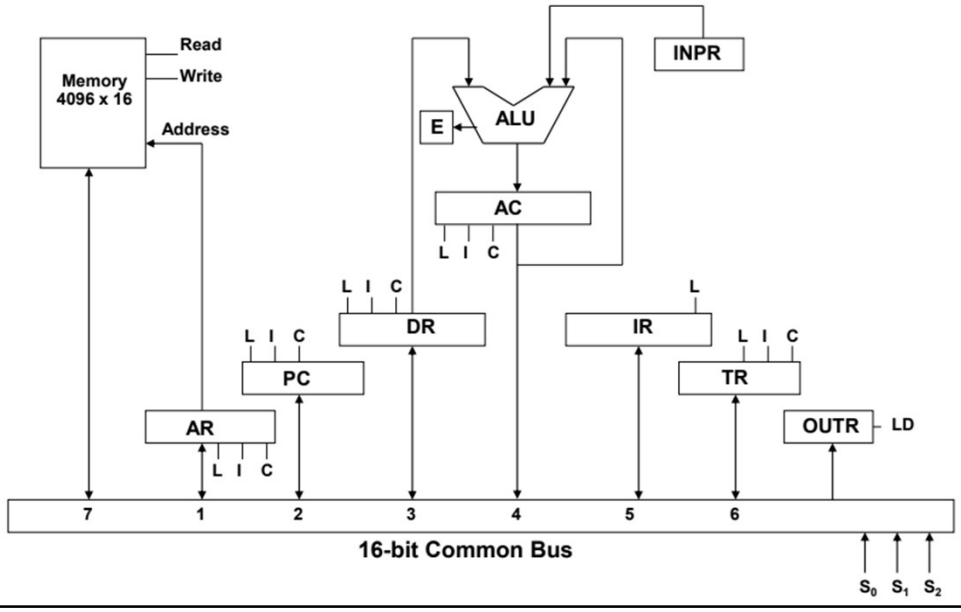
COMMON BUS SYSTEM
of
Basic Computer

- The registers in the Basic Computer are connected using a bus
- This gives a savings in circuitry over complete connections between registers

COMMON BUS SYSTEM



COMMON BUS SYSTEM



COMMON BUS SYSTEM

- Three control lines, S_2, S_1, and S_0 control which register the bus selects as its input

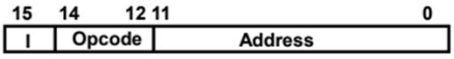
S_2 S_1 S_0	Register
0 0 0	x
0 0 1	AR
0 1 0	PC
0 1 1	DR
1 0 0	AC
1 0 1	IR
1 1 0	TR
1 1 1	Memory

- Either one of the registers will have its load signal activated, or the memory will have its read signal activated
  - Will determine where the data from the bus gets loaded
- The 12-bit registers, AR and PC, have 0's loaded onto the bus in the high order 4 bit positions
- When the 8-bit register OUTR is loaded from the bus, the data comes from the low order 8 bits on the bus

BASIC COMPUTER INSTRUCTIONS

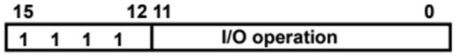
- Basic Computer Instruction Format

Memory-Reference Instructions (OP-code = 000 ~ 110)



Register-Reference Instructions (OP-code = 111, I = 0)

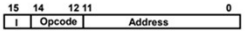
Input-Output Instructions (OP-code =111, I = 1)



BASIC COMPUTER INSTRUCTIONS

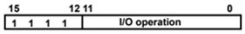
• Basic Computer Instruction Format

Memory-Reference Instructions (OP-code = 000 ~ 110)



Register-Reference Instructions (OP-code = 111, I = 0)

Input-Output Instructions (OP-code =111, I = 1)



Symbol	Hex Code I = 0	Hex Code I = 1	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load AC from memory
STA	3xxx	Bxxx	Store content of AC into memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instr. if AC is positive
SNA	7008		Skip next instr. if AC is negative
SZA	7004		Skip next instr. if AC is zero
SZE	7002		Skip next instr. if E is zero
HLT	7001		Halt computer
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

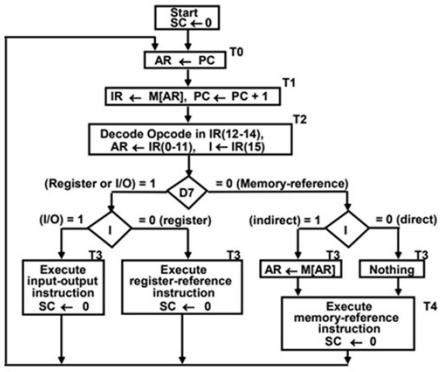
BASIC COMPUTER INSTRUCTIONS

• Basic Computer Instruction Format

Memory-Reference Instructions (OP-code = 000 ~ 110)

Register-Reference Instructions (OP-code = 111, I = 0)

Input-Output Instructions (OP-code =111, I = 1)



D'_7IT_3: AR ← M[AR]
$D'_7I'T_3$: Nothing
$D_7I'T_3$: Execute a register-reference instr.
D_7IT_3: Execute an input-output instr.

Control Unit and Central Processing Unit

Unit-04: Microprogrammed Control

Introduction: Hardwired and Microprogrammed Control Unit

Control unit (CU) of a processor translates from machine instructions to the control signals for the microoperations that implement them. The function of the control unit in a digital computer is to initiate sequences of microoperations. The number of different types of microoperations that are available in a given system is finite. Control units are implemented in one of two ways

- **Hardwired Control** - CU is made up of sequential and combinational circuits to generate the control signals for the microoperations. Example- ADD to AC

  D_1T_4: DR←M[AR] ; D_1T_4 is control signal to generate microoperation DR←M[AR]

  D_1T_5: AC← AC+DR, E← C_{out}, SC ← 0
- **Microprogrammed Control** - A control memory on the processor contains microprograms that activate the necessary control signals.

Unit-04: Microprogrammed Control

Introduction: Hardwired and Microprogrammed Control Unit

- The control function that specifies a microoperation is a binary variable.
- In a bus-organized system, the control signals that specify microoperations are groups of bits that select the paths in multiplexers, decoders, and arithmetic logic units.
- The control unit initiates a series of sequential steps of microoperations.
- The control variables at any given time can be represented by a string of 1's and 0's called a control word. As such, control words can be programmed to perform various operations on the components of the system.

Unit-04: Microprogrammed Control

Introduction: Hardwired and Microprogrammed Control Unit

- A control unit whose binary control variables are stored in memory is called a microprogrammed control unit.
- Each word in control memory contains within it a microinstruction. The microinstruction specifies one or more microoperations for the system. A sequence of microinstructions constitutes a microprogram.

Basic Computer Organization and Design

Control Unit of Basic Computer: Control Timing Signals

- Control unit (CU) of a processor translates the machine instructions to the control signals for the microoperations that implement them.
- The timing for all registers in the basic computer is controlled by a master clock generator. The clock pulses are applied to all flip-flops and registers in the system, including the flip-flops and registers in the control unit. The clock pulses do not change the state of a register unless the register is enabled by a control signal.
- The control signals are generated in the control unit and provide control inputs for the multiplexers in the common bus, control inputs in processor registers, and microoperations for the accumulator.

Basic Computer Organization and Design

Control Unit of Basic Computer: Control Timing Signals

- Control units are implemented in one of two ways
  1. **Hardwired Control** - CU is made up of sequential and combinational circuits to generate the control signals.
  2. **Microprogrammed Control** - A control memory on the processor contains microprograms that activate the necessary control signals

In the hardwired organization, the control logic is implemented with gates, flip-flops, decoders, and other digital circuits. It has the advantage that it can be optimized to produce a fast mode of operation. In the **microprogrammed organization**, the control information is stored in a control memory. The control memory is programmed to initiate the required sequence of microoperations. A hardwired control, as the name implies, requires changes in the wiring among the various components if the design has to be modified or changed. **In the microprogrammed control**, any required changes or modifications can be done by updating the microprogram in control memory.

Basic Computer Organization and Design

Control Unit of Basic Computer: Control Timing Signals

1. **Hardwired Control** - CU is made up of sequential and combinational circuits to generate the control signals.

Basic Computer Organization and Design

Control Unit of Basic Computer: Control Timing Signals

A hardwired implementation of the control unit for the Basic Computer is shown below. The block diagram of the hardwired control unit consists of (1) **two decoders** (2) a sequence counter, and (3) a number of control logic gates.

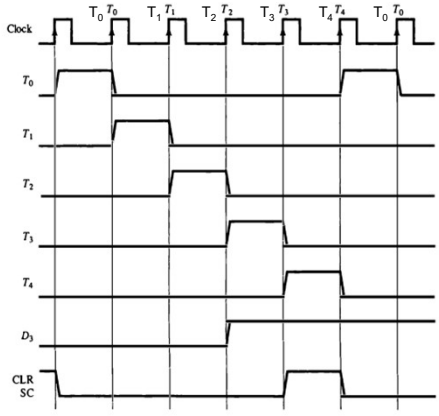
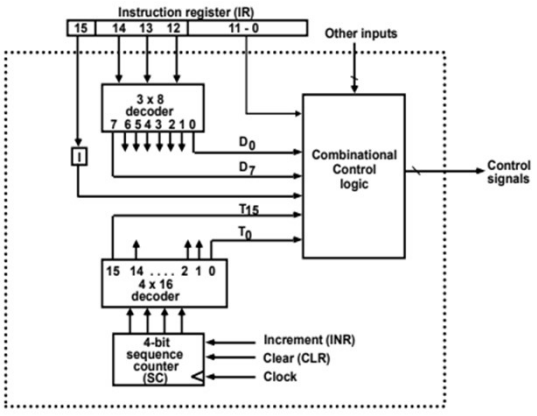


Figure 5-7 Example of control timing signals.



Basic Computer Organization and Design

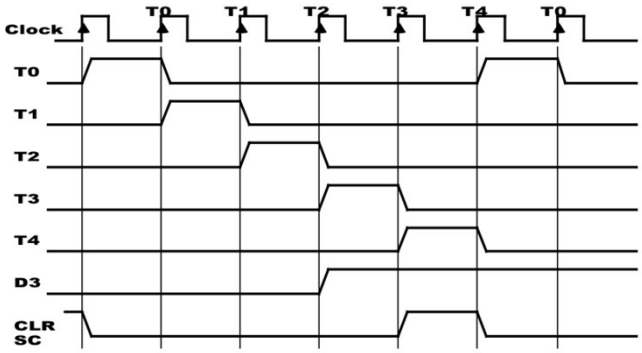
Control Unit of Basic Computer: Control Timing Signals

Timing signals:

- Generated by 4-bit sequence counter and 4x16 decoder.
- The SC can be incremented or cleared.
- Example: T0, T1, T2, T3, T4, T0, T1 . . .

Assume: At time T4, SC is cleared to 0 if decoder output D3 is active:

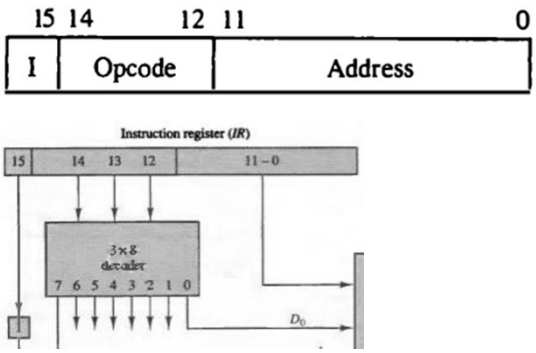
D3T4: SC ←0



Basic Computer Organization and Design

Control Unit of Basic Computer: Control Timing Signals

- An instruction read from memory is placed in the instruction register (IR).
- The instruction register (IR) is divided into three parts:
  - the 1 bit,
  - the operation code, and
  - bits 0 through 11.



Basic Computer Organization and Design

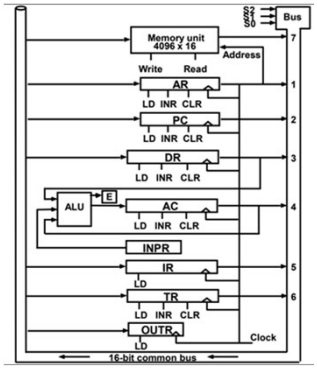
Control Unit of Basic Computer: Control Timing Signals

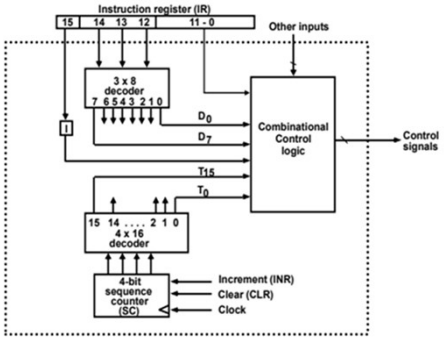
- The operation code in bits 12 through 14 are decoded with a 3 x 8 decoder. The eight outputs of the decoder are designated by the symbols D_0 through D_7. The subscripted decimal number is equivalent to the binary value of the corresponding operation code.
- Bit 15 of the instruction is transferred to a flip-flop designated by the symbol I.
- Bits 0 through 11 are applied to the control logic gates.
- The 4-bit sequence counter can count in binary from 0 through 15. The outputs of the counter are decoded into 16 timing signals T_0 through T_{15}.
- The sequence counter SC can be incremented or cleared synchronously. Most of the time, the counter is incremented to provide the sequence of timing signals out of the 4 x 16 decoder. Once in a while, the counter is cleared to 0, causing the next active timing signal to be T_0.

D_3T_4: SC ← 0

Basic Computer Organization and Design

Control Timing Signals





Basic Computer Organization and Design

Control Unit of Basic Computer: Control Timing Signals

- A memory read or write cycle will be initiated with the rising edge of a timing signal.

For example, the register transfer statement

T_0: AR ← PC

Specifies a transfer of the content of PC into AR if timing signal T_0 is active. T_0 is active during an entire clock cycle interval. During this time the content of PC is placed onto the bus (with $S_2S_1S_0 = 010$) and the LD (load) input of AR is enabled. The actual transfer does not occur until the end of the clock cycle when the clock goes through a positive transition. This same positive clock transition increments the sequence counter SC from 0000 to 0001. The next clock cycle has T_1 active and T_0 inactive.

Basic Computer Organization and Design

Instruction Cycle

(Steps of program execution)

Basic Computer Organization and Design

Control Unit of Basic Computer: Instruction Cycle

A program residing in the memory unit of the computer consists of a sequence of instructions. The program is executed in the computer by going through a cycle for each instruction.

In the basic computer each instruction cycle consists of the following phases:

1. Fetch an instruction from memory.
2. Decode the instruction.
3. Read the effective address from memory if the instruction has an indirect address.
4. Execute the instruction.

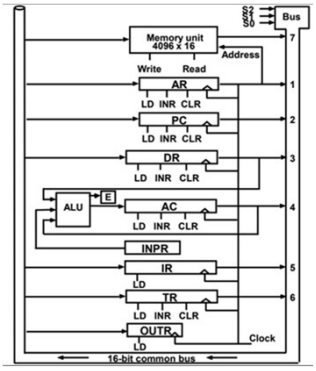
Upon the completion of step 4, the control goes back to step 1 to fetch, decode, and execute the next instruction. This process continues indefinitely unless a HALT instruction is encountered.

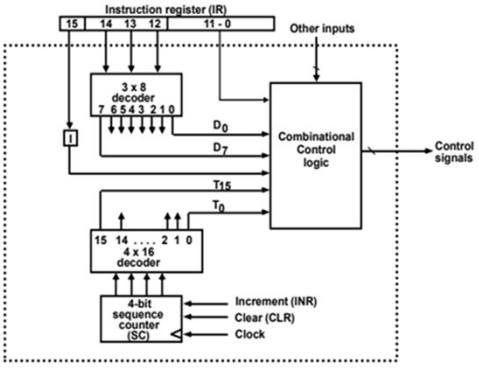
Basic Computer Organization and Design

Execution steps

of

Fetch and Decode





Basic Computer Organization and Design

Control Unit of Basic Computer: Fetch and Decode

- Initially, the program counter PC is loaded with the address of the first instruction in the program.
- The sequence counter SC is cleared to 0, providing a decoded timing signal T_o.
- After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence T_0, T_1, T_2, and so on.
- The microoperations for the fetch and decode phases can be specified by the following register transfer statements.

T_0: $AR \leftarrow PC$
T_1: $IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$
T_2: $D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{–}14)$, $AR \leftarrow IR(0\text{–}11)$, $I \leftarrow IR(15)$

Basic Computer Organization and Design

Control Unit of Basic Computer: Fetch and Decode

T_0: $AR \leftarrow PC$
T_1: $IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$
T_2: $D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{–}14)$, $AR \leftarrow IR(0\text{–}11)$, $I \leftarrow IR(15)$

To provide the data path for the transfer of PC to AR, we must apply timing signal T_0 to achieve the following connection:

1. Place the content of PC onto the bus by making the bus selection inputs S_2 S_1 S_0 equal to 010.
2. Transfer the content of the bus to AR by enabling the LD input of AR

The next clock transition initiates the transfer from PC to AR since $T_0 = 1$.

Basic Computer Organization and Design

Control Unit of Basic Computer: Fetch and Decode

T_0: $AR \leftarrow PC$
T_1: $IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$
T_2: $D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{–}14)$, $AR \leftarrow IR(0\text{–}11)$, $I \leftarrow IR(15)$

In order to implement the second statement, it is necessary to use timing signal T_1 to provide the connections in the bus system.

1. Enable the read input of memory.
2. Place the content of memory onto the bus by making $S_2S_1S_0 = 111$.
3. Transfer the content of the bus to IR by enabling the LD input of IR.
4. Increment PC by enabling the INR input of PC.

The next clock transition initiates the read and increment operations since $T_1 = 1$.

Basic Computer Organization and Design

Determine the Type of Instructions

Basic Computer Organization and Design

Control Unit of Basic Computer: Determine the Type of Instruction

- The timing signal that is active after decoding is T_3. During time T_3, the control unit determines the type of instruction that was just read from memory.
- Flowchart in next slide presents an initial configuration for the instruction cycle and shows how the control determines the instruction type after decoding.

The three instruction types are subdivided into four separate paths:

- D'_7IT3: AR ← M[AR] Indirect address(memory-reference instrs)
- $D'_7I'T3$: Nothing
- $D_7I'T3$: Execute register-reference instructions.
- D_7IT3: Execute input-output instructions.

Basic Computer Organization and Design

Control Unit of Basic Computer: Determine the Type of Instruction

• Basic Computer Instruction Format

Memory-Reference Instructions (OP-code = 000 ~ 110)

Register-Reference Instructions (OP-code = 111, I = 0)

Input-Output Instructions (OP-code =111, I = 1)

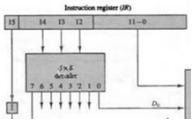
Symbol	Hex Code I = 0	Hex Code I = 1	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load AC from memory
STA	3xxx	Bxxx	Store content of AC into memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instr. if AC is positive
SNA	7008		Skip next instr. if AC is negative
SZA	7004		Skip next instr. if AC is zero
SZE	7002		Skip next instr. if E is zero
HLT	7001		Halt computer
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

D_7' I T3:

D_7' I' T3:

D_7 I' T3:

D_7 I T3:



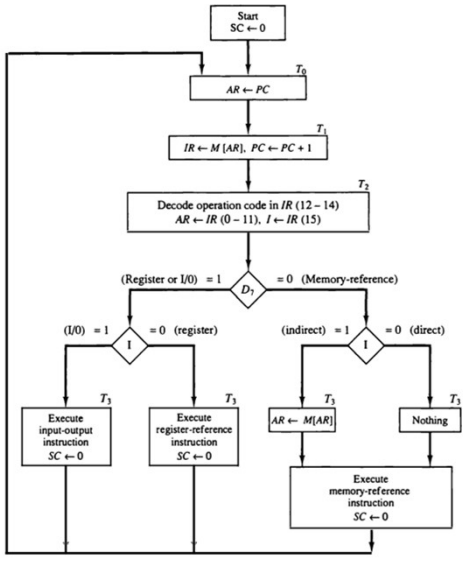


Figure 5-9 Flowchart for instruction cycle (initial configuration).

Basic Computer Organization and Design

Control Unit of Basic Computer: Resister-reference instructions

Symbol	Hex Code I = 0	Hex Code I = 1	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load AC from memory
STA	3xxx	Bxxx	Store content of AC into memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instr. if AC is positive
SNA	7008		Skip next instr. if AC is negative
SZA	7004		Skip next instr. if AC is zero
SZE	7002		Skip next instr. if E is zero
HLT	7001		Halt computer
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

Register Reference Instructions

Basic Computer Organization and Design

Control Unit of Basic Computer: Resister-reference instructions

Resister-reference instructions: Register Reference Instructions are recognized with

- $D_7 = 1, I = 0$
- Register Ref. Instr. is specified in b0 ~ b11 of IR
- Execution starts with timing signal T3

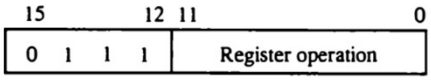
Let

r = D7 I'T3 => Common to all Register Reference Instruction

B_i = IR (i), i=0, 1, 2... 11. [Bit in IR(0-11) that specifies the operation]

Basic Computer Organization and Design

Control Unit of Basic Computer: Resister-reference instructions:



(Opcode = 111, $I = 0$)

(b) Register – reference instruction

TABLE 5-3 Execution of Register-Reference Instructions

$D_7I'T_3 = r$ (common to all register-reference instructions)
$IR(i) = B_i$ [bit in IR(0–11) that specifies the operation]

	r:	$SC \leftarrow 0$	Clear SC
CLA	rB_{11}:	$AC \leftarrow 0$	Clear AC
CLE	rB_{10}:	$E \leftarrow 0$	Clear E
CMA	rB_9:	$AC \leftarrow \overline{AC}$	Complement AC
CME	rB_8:	$E \leftarrow \overline{E}$	Complement E
CIR	rB_7:	$AC \leftarrow$ shr AC, $AC(15) \leftarrow E$, $E \leftarrow AC(0)$	Circulate right
CIL	rB_6:	$AC \leftarrow$ shl AC, $AC(0) \leftarrow E$, $E \leftarrow AC(15)$	Circulate left
INC	rB_5:	$AC \leftarrow AC + 1$	Increment AC
SPA	rB_4:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	Skip if positive
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Skip if negative
SZA	rB_2:	If $(AC = 0)$ then $PC \leftarrow PC + 1$	Skip if AC zero
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$	Skip if E zero
HLT	rB_0:	$S \leftarrow 0$ (S is a start–stop flip-flop)	Halt computer

Basic Computer Organization and Design

Control Unit of Basic Computer: Memory-reference instructions

Once an instruction has been loaded to IR, it may require further access to memory to perform its intended function (direct or indirect).

- The effective address of the instruction is in the AR and was placed their during:
  - Time signal T_2 when I = 0 or
  - Time signal T_3 when I = 1
- Execution of memory reference instructions starts with the timing signal T_4.
- Described symbolically using RTL in next slide….

Basic Computer Organization and Design

Control Unit of Basic Computer: Memory-reference instructions

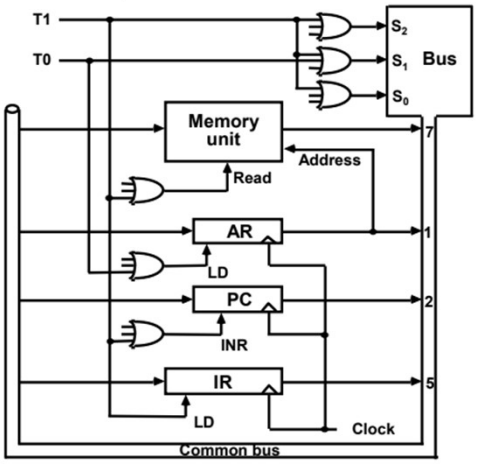
Memory-reference:

AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR$, $E \leftarrow C_{out}$, $SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR$, $SC \leftarrow 0$
STA	D_3T_4:	$M[AR] \leftarrow AC$, $SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR$, $SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC$, $AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR$, $SC \leftarrow 0$
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR$, if $(DR = 0)$ then $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$

FETCH and DECODE

- **Fetch and Decode**

T0: AR ← PC ($S_0S_1S_2$=010, T0=1)
T1: IR ← M [AR], PC ← PC + 1 (S0S1S2=111, T1=1)
T2: D0, . . . , D7 ← Decode IR(12-14), AR ← IR(0-11), I ← IR(15)



FETCH and DECODE

ADD Operation

Memory-reference:

AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR,\ SC \leftarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR,\ E \leftarrow C_{out},\ SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR,\ SC \leftarrow 0$
STA	D_3T_4:	$M[AR] \leftarrow AC,\ SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR,\ SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC,\ AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR,\ SC \leftarrow 0$
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR,\ \text{if } (DR = 0) \text{ then } (PC \leftarrow PC + 1),\ SC \leftarrow 0$

MEMORY REFERENCE INSTRUCTIONS

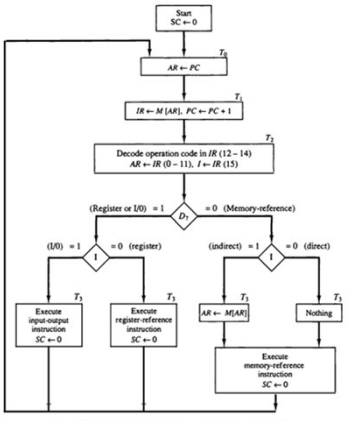


Figure 5-9 Flowchart for instruction cycle (initial configuration).

Symbol	Operation Decoder	Symbolic Description
AND	D_0	AC ← AC ∧ M[AR]
ADD	D_1	AC ← AC + M[AR], E ← C_{out}
LDA	D_2	AC ← M[AR]
STA	D_3	M[AR] ← AC
BUN	D_4	PC ← AR
BSA	D_5	M[AR] ← PC, PC ← AR + 1
ISZ	D_6	M[AR] ← M[AR] + 1, if M[AR] + 1 = 0 then PC ← PC+1

- The effective address of the instruction is in AR and was placed there during timing signal T_2 when I = 0, or during timing signal T_3 when I = 1
- Memory cycle is assumed to be short enough to complete in a CPU cycle
- The execution of MR instruction starts with T_4

AND to AC

D_0T_4:	DR ← M[AR]	Read operand
D_0T_5:	AC ← AC ∧ DR, SC ← 0	AND with AC

ADD to AC

D_1T_4:	DR ← M[AR]	Read operand
D_1T_5:	AC ← AC + DR, E ← C_{out}, SC ← 0	Add to AC and store carry in E

T_0: AR ← PC ($S_0S_1S_2$=010' T_0=1)
T_1: IR ← M[AR]; PC ← PC+1 ($S_0S_1S_2$=111, T_1=1
T_2: D0…D7 ←Decode IR(12-14); AR ← IR(0-11); I ← IR(15)
T_3: AR ← M[AR]
D_1T_4: DR ←M[AR]
D_1T_5: AC ← AC + DR

Basic Computer Organization and Design

Control Unit of Basic Computer: Control Timing Signals

2. **Microprogrammed Control** - A control memory on the processor contains microprograms that activate the necessary control signals

Unit-04: Microprogrammed Control

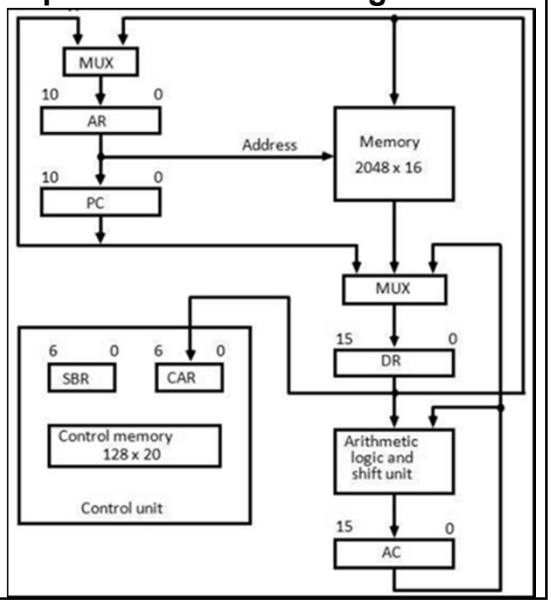
Microprogrammed controlled Computer Hardware Configuration:

The microprogrammed computer explained here is similar but not identical to the basic computer architecture of Moris mano's explained before.

The block diagram of the microprogrammed controlled computer is shown in Figure. It consists of

1. Two memory units:

- Main memory -> for storing instructions and data, and
- Control memory -> for storing the microprogram.



Unit-04: Microprogrammed Control

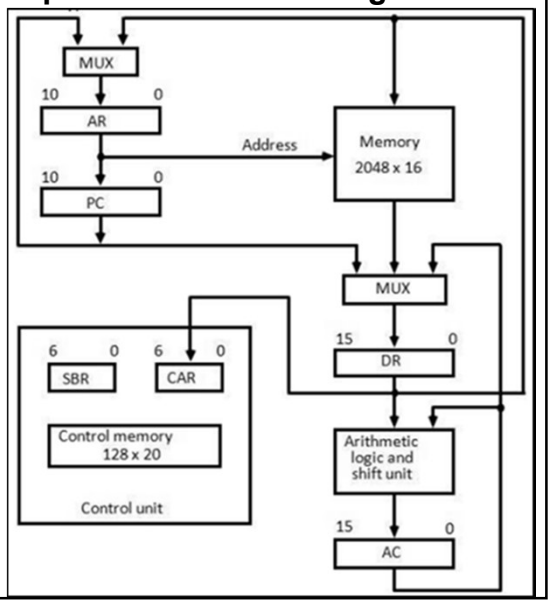
Microprogrammed controlled Computer Hardware Configuration:

2. Six Registers:

- Processor unit register: **AC** (accumulator), **PC** (Program Counter), **AR** (Address Register), **DR** (Data Register)
- Control unit register: **CAR** (Control Address Register), **SBR** (Subroutine Register)

3. Multiplexers:

- The transfer of information among the registers in the processor is done through multiplexers rather than a common bus.

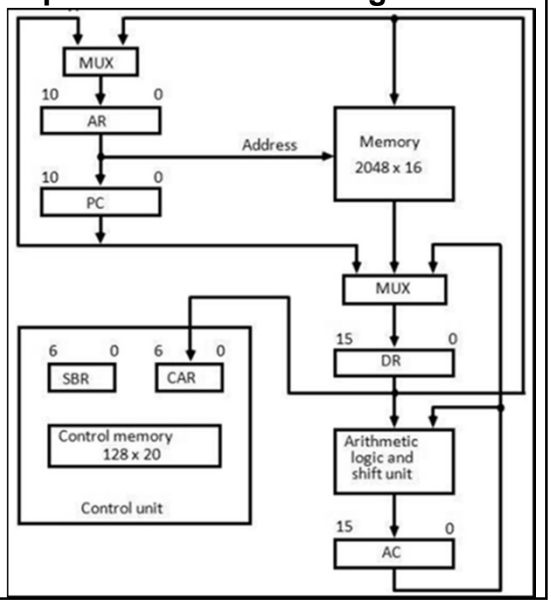


Unit-04: Microprogrammed Control

Microprogrammed controlled Computer Hardware Configuration:

4. ALU: The arithmetic, logic, and shift unit perform microoperations with data from AC and DR and places the result in AC.

- DR can receive information from AC, PC, or memory.
- AR can receive information from PC or DR.
- PC can receive information only from AR.
- Input data written to memory come from DR, and data read from memory can go only to DR.



Unit-04: Microprogrammed Control

Microprogrammed controlled Computer Hardware Configuration:

Control Memory: A computer that employs a microprogrammed control unit will have two separate memories: a **main memory** and a **control memory**.

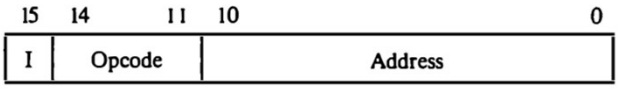
- The main memory is available to the user for storing the programs. The contents of main memory may alter when the data are manipulated and every time that the program is changed. The user's program in main memory consists of machine instructions and data.
- In contrast, the control memory holds a fixed microprogram that cannot be altered by the occasional user. The microprogram consists of microinstructions that specify various internal control signals for execution of register microoperations.
- Each machine instruction initiates a series of microinstructions in control memory. These microinstructions generate the microoperations to fetch the instruction from main memory; to evaluate the effective address, to execute the operation specified by the instruction, and to return control to the fetch phase in order to repeat the cycle for the next instruction.

Unit-04: Microprogrammed Control

Microprogrammed controlled Computer Hardware Configuration:

Computer Instruction Format:

Figure 7-5 Computer instructions.



(a) Instruction format

Symbol	Opcode	Description
ADD	0000	$AC \leftarrow AC + M[EA]$
BRANCH	0001	If $(AC < 0)$ then $(PC \leftarrow EA)$
STORE	0010	$M[EA] \leftarrow AC$
EXCHANGE	0011	$AC \leftarrow M[EA], M[EA] \leftarrow AC$

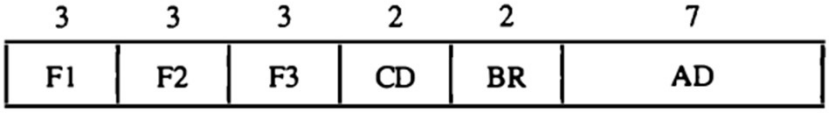
EA is the effective address

(b) Four computer instructions

Unit-04: Microprogrammed Control

Microprogrammed controlled Computer Hardware Configuration:

Microinstruction Format : 20 bits



F1, F2, F3: Microoperation fields

CD: Condition for branching

BR: Branch field

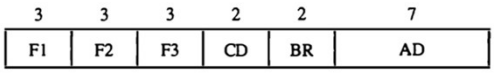
AD: Address field

Figure 7-6 Microinstruction code format (20 bits).

Unit-04: Microprogrammed Control

Microprogrammed controlled Computer Hardware Configuration:

Symbols and Binary Code for Microinstruction Fields:



F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0\text{–}10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0\text{-}10) \leftarrow PC$	PCTDR

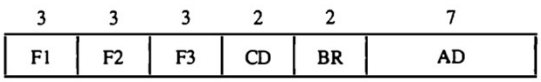
F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow$ shl AC	SHL
100	$AC \leftarrow$ shr AC	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	$DR(15)$	I	Indirect address bit
10	$AC(15)$	S	Sign bit of AC
11	$AC = 0$	Z	Zero value in AC

BR	Symbol	Function
00	JMP	$CAR \leftarrow AD$ if condition = 1 $CAR \leftarrow CAR + 1$ if condition = 0
01	CALL	$CAR \leftarrow AD$, $SBR \leftarrow CAR + 1$ if condition = 1 $CAR \leftarrow CAR + 1$ if condition = 0
10	RET	$CAR \leftarrow SBR$ (Return from subroutine)
11	MAP	$CAR(2\text{–}5) \leftarrow DR(11\text{–}14)$, $CAR(0,1,6) \leftarrow 0$

Unit-04: Microprogrammed Control

Microprogrammed controlled Computer Hardware Configuration: Symbols and Binary Code for Microinstruction Fields:



F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0\text{–}10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0\text{-}10) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow$ shl AC	SHL
100	$AC \leftarrow$ shr AC	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

BR	Symbol	Function
00	JMP	$CAR \leftarrow AD$ if condition = 1 $CAR \leftarrow CAR + 1$ if condition = 0
01	CALL	$CAR \leftarrow AD$, $SBR \leftarrow CAR + 1$ if condition = 1 $CAR \leftarrow CAR + 1$ if condition = 0
10	RET	$CAR \leftarrow SBR$ (Return from subroutine)
11	MAP	$CAR(2\text{–}5) \leftarrow DR(11\text{–}14)$, $CAR(0,1,6) \leftarrow 0$

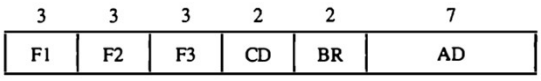
CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	$DR(15)$	I	Indirect address bit
10	$AC(15)$	S	Sign bit of AC
11	$AC = 0$	Z	Zero value in AC

Write a Binary code(20 bits) for microoperation fields of following Symbols for microoperation.

PCTAR	U	JMP	NEXT
READ, INCPC	U	JMP	NEXT
DRTAR	U	MAP	

Unit-04: Microprogrammed Control

Microprogrammed controlled Computer Hardware Configuration: Symbols and Binary Code for Microinstruction Fields:



F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0\text{–}10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0\text{-}10) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow$ shl AC	SHL
100	$AC \leftarrow$ shr AC	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

BR	Symbol	Function
00	JMP	$CAR \leftarrow AD$ if condition = 1 $CAR \leftarrow CAR + 1$ if condition = 0
01	CALL	$CAR \leftarrow AD$, $SBR \leftarrow CAR + 1$ if condition = 1 $CAR \leftarrow CAR + 1$ if condition = 0
10	RET	$CAR \leftarrow SBR$ (Return from subroutine)
11	MAP	$CAR(2\text{–}5) \leftarrow DR(11\text{–}14)$, $CAR(0,1,6) \leftarrow 0$

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	$DR(15)$	I	Indirect address bit
10	$AC(15)$	S	Sign bit of AC
11	$AC = 0$	Z	Zero value in AC

Write a symbolic code for microoperation fields of following binary code for microoperation.

Binary Address	F1	F2	F3	CD	BR	AD
1000000	110	000	000	00	00	1000001
1000001	000	100	101	00	00	1000010
1000010	101	000	000	00	11	0000000

Course Contents

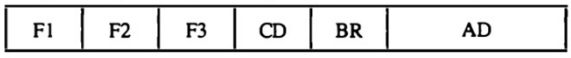
Unit-04: Microprogrammed Control

Decoding (FETCH Instruction)

Unit-04: Microprogrammed Control

Symbolic Microinstructions:

Format of Microinstruction:




```
                ORG 64
FETCH:          PCTAR           U    JMP    NEXT
                READ, INCPC     U    JMP    NEXT
                DRTAR           U    MAP
```

Contains five fields: label; micro-ops; CD; BR; AD

Label:	may be empty or may specify a symbolic address terminated with a colon
Micro-ops:	consists of one, two, or three symbols separated by commas
CD:	One of {U, I, S, Z},
Where	U: Unconditional Branch
	I: Indirect address bit
	S: Sign of AC
	Z: Zero value in AC
BR:	one of {JMP, CALL, RET, MAP}
AD:	one of {Symbolic address, NEXT, empty (in case of MAP and RET)}

We will use also the pseudo-instruction ORG to define the origin, or first address, of a microprogram routine.

Unit-04: Microprogrammed Control

Symbolic Microinstructions:
Symbolic Microprogram (example)

FETCH Routine: During FETCH, Read an instruction from memory and decode the instruction and update PC

Sequence of microoperations in the fetch cycle:

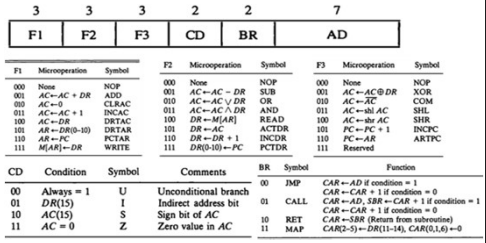
AR ← PC

DR ← M[AR], PC ← PC + 1

AR ← DR(0-10), CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 0

Symbolic microprogram for the fetch cycle:

```
                ORG 64
FETCH:  PCTAR       U   JMP   NEXT
        READ, INCPC U   JMP   NEXT
        DRTAR       U   MAP
```



	F1	F2	F3	CD	BR	AD
	3	3	3	2	2	7

F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0\text{-}10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0\text{-}10) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow shl\ AC$	SHL
100	$AC \leftarrow shr\ AC$	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	$DR(15)$	I	Indirect address bit
10	$AC(15)$	S	Sign bit of AC
11	$AC = 0$	Z	Zero value in AC

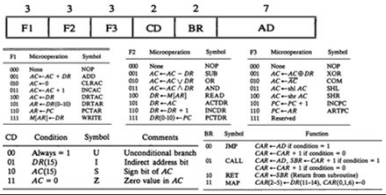
BR	Symbol	Function
00	JMP	$CAR \leftarrow AD$ if condition = 1 $CAR \leftarrow CAR + 1$ if condition = 0
01	CALL	$CAR \leftarrow AD$, $SBR \leftarrow CAR + 1$ if condition = 1 $CAR \leftarrow CAR + 1$ if condition = 0
10	RET	$CAR \leftarrow SBR$ (Return from subroutine)
11	MAP	$CAR(2\text{-}5) \leftarrow DR(11\text{-}14)$, $CAR(0,1,6) \leftarrow 0$

We will use also the pseudo instruction ORG to define the origin, or first address, of a microprogram routine. Thus the symbol ORG 64 informs the assembler to place the next microinstruction in control memory at decimal address 64, which is equivalent to the binary address 1000000.

Unit-04: Microprogrammed Control

Partial Symbolic Microprogram:

- E.g. the execution of ADD instruction is carried out by the microinstructions at addresses 1 and 2.
- The first microinstruction reads operand from into DR.
- The second microinstruction performs an add microoperation with the content of DR AC and then jumps back to the beginning of the fetch routine.



	F1	F2	F3	CD	BR	AD
	3	3	3	2	2	7

F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0\text{-}10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0\text{-}10) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow shl\ AC$	SHL
100	$AC \leftarrow shr\ AC$	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	$DR(15)$	I	Indirect address bit
10	$AC(15)$	S	Sign bit of AC
11	$AC = 0$	Z	Zero value in AC

BR	Symbol	Function
00	JMP	$CAR \leftarrow AD$ if condition = 1 $CAR \leftarrow CAR + 1$ if condition = 0
01	CALL	$CAR \leftarrow AD$, $SBR \leftarrow CAR + 1$ if condition = 1 $CAR \leftarrow CAR + 1$ if condition = 0
10	RET	$CAR \leftarrow SBR$ (Return from subroutine)
11	MAP	$CAR(2\text{-}5) \leftarrow DR(11\text{-}14)$, $CAR(0,1,6) \leftarrow 0$

TABLE 7-2 Symbolic Microprogram (Partial)

Label	Microoperations	CD	BR	AD
	ORG 0			
ADD:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH
	ORG 4			
BRANCH:	NOP	S	JMP	OVER
	NOP	U	JMP	FETCH
OVER:	NOP	I	CALL	INDRCT
	ARTPC	U	JMP	FETCH
	ORG 8			
STORE:	NOP	I	CALL	INDRCT
	ACTDR	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 12			
EXCHANGE:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ACTDR, DRTAC	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 64			
FETCH:	PCTAR	U	JMP	NEXT
	READ, INCPC	U	JMP	NEXT
	DRTAR	U	MAP	
INDRCT:	READ	U	JMP	NEXT
	DRTAR	U	RET	

Unit-04: Microprogrammed Control

Binary Microprogram:

Symbolic microprogram is a convenient form for writing microprograms in a way that people can understand. But this is not a way that the microprogram is stored in memory. It must be translated into binary by means of assembler. Binary equivalent of a microprogram translated by an assembler for fetch cycle:

F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0-10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0-10) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow$ shl AC	SHL
100	$AC \leftarrow$ shr AC	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

Binary address	F1	F2	F3	CD	BR	AD
1000000	110	000	000	00	00	1000001
1000001	000	100	101	00	00	1000010
1000010	101	000	000	00	11	0000000

AR ← PC

DR ← M[AR], PC ← PC + 1

AR ← DR(0-10), CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 0

```
        ORG 64
FETCH:  PCTAR          U   JMP   NEXT
        READ, INCPC    U   JMP   NEXT
        DRTAR          U   MAP
```

Unit-04: Microprogrammed Control

Binary Microprogram:

```
Label      Microoperations   CD   BR      AD
           ORG 0
ADD:       NOP               I    CALL    INDRCT
           READ              U    JMP     NEXT
           ADD               U    JMP     FETCH
           ORG 4
BRANCH:    NOP               S    JMP     OVER
           NOP               U    JMP     FETCH
OVER:      NOP               I    CALL    INDRCT
           ARTPC             U    JMP     FETCH
           ORG 8
STORE:     NOP               I    CALL    INDRCT
           ACTDR             U    JMP     NEXT
           WRITE             U    JMP     FETCH
           ORG 12
EXCHANGE:  NOP               I    CALL    INDRCT
           READ              U    JMP     NEXT
           ARTDR, DRTAC      U    JMP     NEXT
           WRITE             U    JMP     FETCH
           ORG 64
FETCH:     PCTAR             U    JMP     NEXT
           READ, INCPC       U    JMP     NEXT
           DRTAC             U    MAP
INDRCT:    READ              U    JMP     NEXT
           DRTAC             U    RET
```

TABLE 7-3 Binary Microprogram for Control Memory (Partial)

Micro Routine	Address		Binary Microinstruction					
	Decimal	Binary	F1	F2	F3	CD	BR	AD
ADD	0	0000000	000	000	000	01	01	1000011
	1	0000001	000	100	000	00	00	0000010
	2	0000010	001	000	000	00	00	1000000
	3	0000011	000	000	000	00	00	1000000
BRANCH	4	0000100	000	000	000	10	00	0000110
	5	0000101	000	000	000	00	00	1000000
	6	0000110	000	000	000	01	01	1000011
	7	0000111	000	000	110	00	00	1000000
STORE	8	0001000	000	000	000	01	01	1000011
	9	0001001	000	101	000	00	00	0001010
	10	0001010	111	000	000	00	00	1000000
	11	0001011	000	000	000	00	00	1000000
EXCHANGE	12	0001100	000	000	000	01	01	1000011
	13	0001101	001	000	000	00	00	0001110
	14	0001110	100	101	000	00	00	0001111
	15	0001111	111	000	000	00	00	1000000
FETCH	64	1000000	110	000	000	00	00	1000001
	65	1000001	000	100	101	00	00	1000010
	66	1000010	101	000	000	00	11	0000000
INDRCT	67	1000011	000	100	000	00	00	1000100
	68	1000100	101	000	000	00	10	0000000

Unit-04: Microprogrammed Control

Add Instruction:

How the transfer and return from the indirect subroutine occurs

Assume that the **MAP** microinstruction at the end of the fetch routine caused a branch to address **0**, where the **ADD** routine is stored. The first microinstruction in the **ADD** routine calls subroutine **INDRCT**, conditioned on status bit **I**. If **I = 1**, a branch to **INDRCT** occurs and the return address (address **1** in this case) is stored in the subroutine register **SBR**.

Remember that an indirect address considers the address part of the instruction as the address where the effective address is stored rather than the address of the operand. Therefore, the memory has to be accessed to get the effective address, which is then transferred to AR. The return from subroutine (**RET**) transfers the address from SBR to **CAR**, thus returning to the second microinstruction of the **ADD** routine.

1. The Execution of the ADD Instruction

The execution of the **ADD** instruction is carried out by the microinstructions at addresses **1** and **2**. The first microinstruction reads the operand from memory into **DR**. The second microinstruction performs an add microoperation with the content of **DR** and **AC** and then jumps back to the beginning of the fetch routine.

Unit- 4

- Architecture of Basic Computer: Register organization, Common Bus System, Instruction Format, Register Stack and Memory Stack

General Register Organization: Control Word

Register organization

Why require computer registers in CPU?

A register in the control unit is required for storing the instruction code after it is read from memory. The computer needs processor registers for manipulating data and a register for holding a memory address.

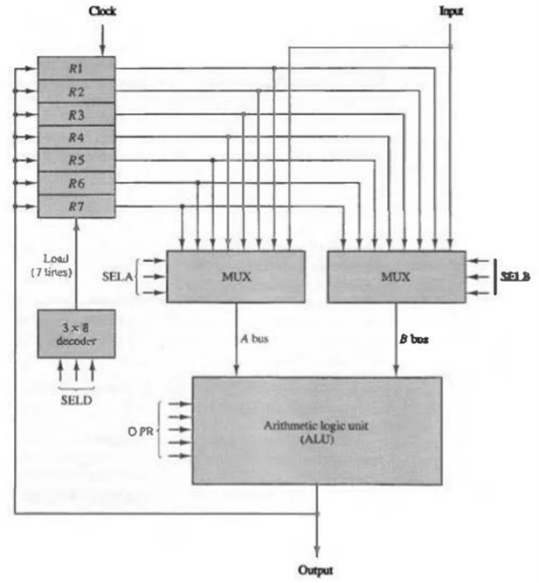
The registers of Basis Computer are also listed in Table together with a brief description of their function and the number of bits that they contain.

Table 1-1 List of Registers for Basic Computer

Register symbol	Number of bits	Register name	Function
DR	16	Data register	Holds memory operand
AR	12	Address register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register	Holds temporary data
INPR	8	Input register	Holds input character
OUTR	8	Output register	Holds output character

Course Contents

General Register Organization:



Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

OPR Select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	Add A + B	ADD
00101	Subtract A − B	SUB
00110	Decrement A	DECA
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

Course Contents

General Register Organization: Control Word

The control unit that operates the CPU bus system directs the information flow through the registers and ALU by selecting the various components in the system. For example, to perform the operation

R1 <--R2 + R3

the control must provide binary selection variables to the following selector inputs:

1. MUX A selector (SELA): to place the content of R2 into bus A.
2. MUX B selector (SELB): to place the content of R3 into bus B.
3. ALU operation selector (OPR): to provide the arithmetic addition A+ B.
4. Decoder destination selector (SELD): to transfer the content of the output bus into R1.

Course Contents

General Register Organization: Control Word

Examples of Microoperations: A control word of 14 bits is needed to specify a micro-operation in the CPU. The control word for a given micro-operation can be derived from the selection variables.

For example, the subtract micro-operation given by the statement

R1 <- R2 - R3

specifies R2 for the A input of the ALU, R3 for the B input of the ALU, R1 for the destination register, and an ALU operation to subtract A - B.

The binary control word for the subtract micro-operation is 010 011 001 00101 and is obtained as follows:

Field:	SELA	SELB	SELD	OPR
Symbol:	R2	R3	R1	SUB
Control word:	010	011	001	00101

Course Contents

General Register Organization:

Examples of Microoperations:

TABLE 8-3 Examples of Microoperations for the CPU

Microoperation	Symbolic Designation				Control Word
	SELA	SELB	SELD	OPR	
$R1 \leftarrow R2 - R3$	R2	R3	R1	SUB	010 011 001 00101
$R4 \leftarrow R4 \vee R5$	R4	R5	R4	OR	100 101 100 01010
$R6 \leftarrow R6 + 1$	R6	—	R6	INCA	110 000 110 00001
$R7 \leftarrow R1$	R1	—	R7	TSFA	001 000 111 00000
Output $\leftarrow R2$	R2	—	None	TSFA	010 000 000 00000
Output $\leftarrow$ Input	Input	—	None	TSFA	000 000 000 00000
$R4 \leftarrow$ shl $R4$	R4	—	R4	SHLA	100 000 100 11000
$R5 \leftarrow 0$	R5	R5	R5	XOR	101 101 101 01100

Course Contents

General Register Organization: Control Word

Problems:

8-3. Specify the control word that must be applied to the processor of Fig. 8-2 to implement the following microoperations.

a. $R1 \leftarrow R2 + R3$
b. $R4 \leftarrow R4$
c. $R5 \leftarrow R5 - 1$
d. $R6 \leftarrow$ shl $R1$
e. $R7 \leftarrow$ input

8-4. Determine the microoperations that will be executed in the processor of Fig. 8-2 when the following 14-bit control words are applied.

a. 00101001100101
b. 00000000000000
c. 01001001001100
d. 00000100000010
e. 11110001110000

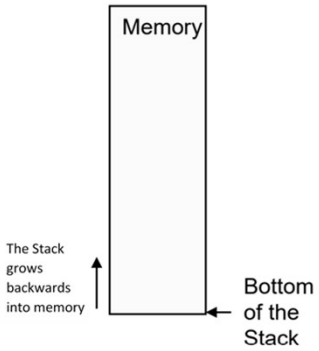
Unit-04: Architecture of Basic Computer

Stack Organization:

STACK- PUSH and POP Operation:

What is Stack?

- Used by program as well as microprocessor
- Operations - Push & Pop
- Stack address is in Stack Pointer register
- LXI instruction is used to define stack LXI SP, 4500h
- Stack can overlap program memory



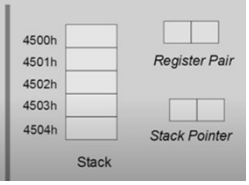
Unit-04: Architecture of Basic Computer

Stack Organization:

STACK- PUSH and POP Operation:

PUSH - *Push Register Pair onto Stack*

1. Stack Pointer is decremented by 1
2. Higher Register is copied to Stack
3. Stack Pointer is decremented by 1
4. Lower Register is copied to Stack



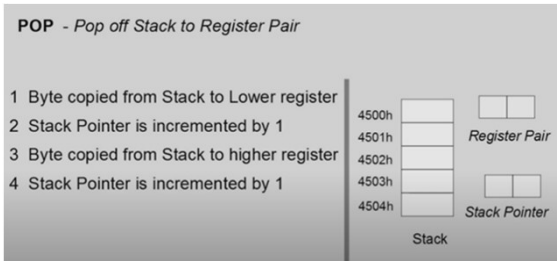
Example 1

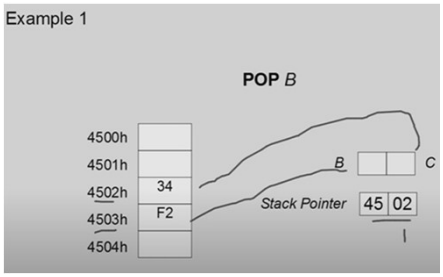
PUSH *B*

Unit-04: Architecture of Basic Computer

Register Stack and Memory Stack

STACK- PUSH and POP Operation:





Unit-04: Architecture of Basic Computer

Stack Organization:

1. Register stack:
2. Memory stack

The stack in digital computers is essentially a memory unit with an **address register** that can count only after an initial value is loaded into it. The register that holds the address for the stack is called a stack pointer (SP) because its value always points at the top item in the stack.

A stack can be placed in portion of a large memory or it can be organized as a collection of a finite number of memory words or registers.

Unit-04: Architecture of Basic Computer

Stack Organization:

1. Register stack:
2. Memory stack

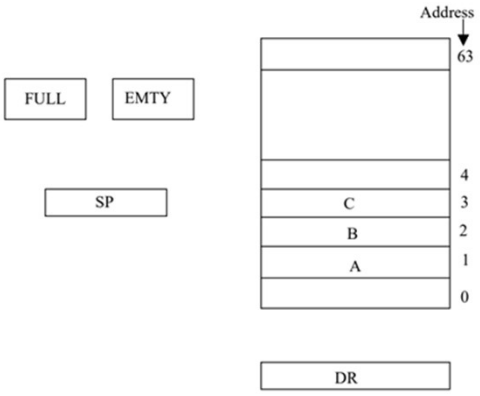


Figure 5-3 Block diagram of a 64-word stack

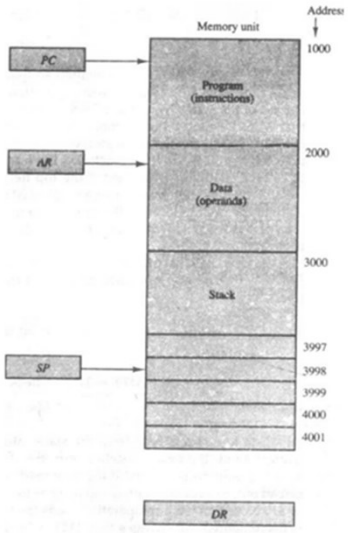


Figure 5-4 Computer memory with program, data, and stack segments.

Unit-04: Architecture of Basic Computer

Stack Organization:

1. Register stack:

Initially, SP is cleared to 0, EMTY is set to 1, and FULL is cleared to 0 (not full), so that SP points to the word at address 0 and the stack is marked empty and not full. If the stack is not full (if FULL = 0), a new item is inserted with a push operation.

The push operation is implemented with the following sequence of micro-operations;

$SP \leftarrow SP + 1$	Increment stack pointer
$M[SP] \leftarrow DR$	Write item on top of the stack
If ($SP = 0$) then ($EMTY \leftarrow 1$)	Check if stack is empty
$FULL \leftarrow 0$	Mark the stack not full

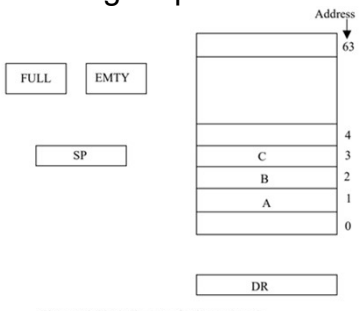


Figure 5-3 Block diagram of a 64-word stack

Unit-04: Architecture of Basic Computer

Stack Organization:

1. Register stack:

A new item is deleted from the stack if the stack is not empty (if EMTY = 0). The pop operation consists of the following sequence of microoperations:

$DR \leftarrow M[SP]$	Read item from the top of stack
$SP \leftarrow SP - 1$	Decrement stack pointer
If $(SP = 0)$ then $(EMTY \leftarrow 1)$	Check if stack is empty
$FULL \leftarrow 0$	Mark the stack not full

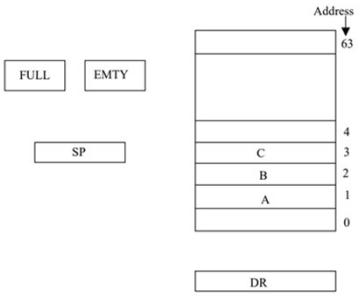


Figure 5-3 Block diagram of a 64-word stack

Unit-04: Architecture of Basic Computer

Stack Organization:

1. Memory stack:

A new item is inserted with the push operation as follows:

$$SP \leftarrow SP - 1$$
$$M[SP] \leftarrow DR$$

A new item is deleted with a pop operation as follows:

$$DR \leftarrow M[SP]$$
$$SP \leftarrow SP + 1$$

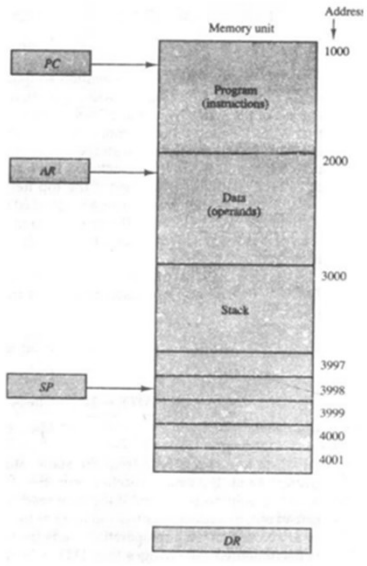
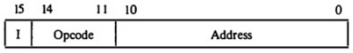


Figure 5-4 Computer memory with program, data, and stack segments.

Instruction Format

- A computer will usually have a variety of instruction code formats. It is the function of the control unit within the CPU to interpret each instruction code and provide the necessary control functions needed to process the instruction.
- The format of an instruction is usually depicted in a rectangular box symbolizing the bits of the instruction as they appear in memory words or in a control register. The bits of the instruction are divided into groups called fields. The most common fields found in instruction formats are:
  - An operation code field that specifies the operation to be performed.
  - An address field that designates a memory address or a processor register.
  - A mode field that specifies the way the operand or the effective address is determined.



(a) Instruction format

Instruction Format

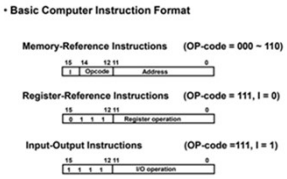
Instruction Format

- The operation code field of an instruction is a group of bits that define various processor operations, such as add, subtract, complement, and shift. The bits that define the mode field of an instruction code specify a variety of alternatives for choosing the operands from the given address.
- Operations specified by computer instructions are executed on some data stored in memory or processor registers.
- Operands residing in processor registers are specified with a register address. A register address is a binary number of k bits that defines one of 2^k registers in the CPU. Thus a CPU with 16 processor registers R0 through R15 will have a register address field of four bits. The binary number 0101, for example, will designate register R5.

Instruction Format

1. A computer uses a memory unit with 256K words of 32 bits each. A binary instruction code is stored in one word of memory. The instruction has four parts: **an indirect bit**, **an operation code**, **a register code part to specify one of 64 registers**, and **an address part**.

a. How many bits are there in the operation code, the register code part, and the address part?
b. Draw the instruction word format and indicate the number of bits in each part.
c. How many bits are there in the data and address inputs of the memory?



Instruction Format

Exercises:

1. A computer uses a memory unit with 256K words of 32 bits each. A binary instruction code is stored in one word of memory. The instruction has four parts: **an indirect bit**, **an operation code**, **a register code part to specify one of 64 registers**, and **an address part**.

a. How many bits are there in the operation code, the register code part, and the address part?
b. Draw the instruction word format and indicate the number of bits in each part.
c. How many bits are there in the data and address inputs of the memory?

5.1

$256\text{ K} = 2^8 \times 2^{10} = 2^{18}$

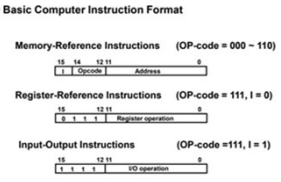
$64 = 2^6$

(a) Address: 18 bits
Register code: 6 bits
Indirect bit: 1 bit
25 32 – 25 = 7 bits for opcode.

(b) 1 7 6 18 = 32 bits

I	opcode	Register	Address

(c) Data; 32 bits; address: 18 bits.



Instruction Format

Instruction Format

- Computer instructions are stored in central memory locations and are executed sequentially one at a time. The control reads an instruction from a specific address in memory and executes it. It then continues by reading the next instruction in sequence and executes it until the completion of the program.
- A computer usually has a variety of Instruction Code Formats. It is the function of the control unit within the CPU to interpret each instruction code and provide the necessary control functions needed to process the instruction.
- An n bit instruction that k bits in the address field and m bits in the operation code field come addressed 2^k location directly and specify 2^m different operation.

Instruction Format

Instruction Format

- Computers may have instructions of several different lengths containing varying number of addresses.
- The number of address fields in the instruction format of a computer depends on the internal organization of its registers.

Instruction Format

Instruction Format

- The bits of the instruction are divided into groups called fields.
- The most common fields in instruction formats are:
  - An Operation code field that specifies the operation to be performed.
  - An Address field that designates a memory address or a processor register.
  - A Mode field that specifies the way the operand or the effective address is determined.

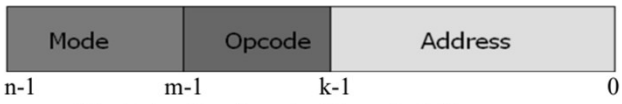


Fig: Instruction format with mode field

Instruction Format

- The bits of the instruction are divided into groups called fields.
- The most common fields in instruction formats are:
  - An Operation code field that specifies the operation to be performed.
  - An Address field that designates a memory address or a processor register.
  - A Mode field that specifies the way the operand or the effective address is determined.

The operation code field (Opcode) of an instruction is a group of bits that define various processor operations such as add, subtract, complement, shift etcetera. The bits that define the mode field of an instruction code specify a variety of alternatives for choosing the operands from the given address. Operation specified by an instruction is executed on some data stored in the processor register or in the memory location. Operands residing in memory are specified by their memory address. Operands residing in processor register are specified with a register address.

Unit- 4

- Data Transfer operations and Manipulation (Arithmetic, Logical, Shift)

- Data Transfer operations and Manipulation (Arithmetic, Logical, Shift)

Unit-04: Central Processing Unit

Data manipulation Instructions

Data manipulation instructions perform operations on data and provide the computational capabilities for the computer. These instructions perform arithmetic, logic and shift operations.

1. Arithmetic instructions
2. Logical and bit manipulation instructions
3. Shift instructions

These instructions are similar to the microoperations in unit3. But actually; each instruction when executed must go through the fetch phase to read its binary code value from memory. The operands must also be brought into registers according to the rules of different addressing mode. And the last step of executing instruction is implemented by means of microoperations as explained earlier unit before.

Unit-04: Central Processing Unit

Arithmetic instructions

Typical arithmetic instructions are listed below:

- Increment (decrement) instr. adds 1 to (subtracts 1 from) the register or memory word value.
- Add, subtract, multiply and divide instructions may operate on different data types (fixed-point or floating-point, binary or decimal).

TABLE 8-7 Typical Arithmetic Instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

Unit-04: Central Processing Unit

Logical and bit manipulation instructions

Logical instructions perform binary operations on strings of bits stored in registers and are useful for manipulating individual or group of bits representing binary coded information. Logical instructions each bit of the operand separately and treat it as a Boolean variable.

- Clear instruction causes specified operand to be replaced by 0's.
- Complement instr. produces the 1's complement.
- AND, OR and XOR instructions produce the corresponding logical operations on individual bits of the operands.

TABLE 8-8 Typical Logical and Bit Manipulation Instructions

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

Unit-04: Central Processing Unit

Shift instructions

Instructions to shift the content of an operand are quite useful and are often provided in several variations (bit shifted at the end of word determine the variation of shift). Shift instructions may specify 3 different shifts:

1. Logical shifts
2. Arithmetic shifts
3. Rotate-type operations

TABLE 8-9 Typical Shift Instructions

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

- Logical shift inserts 0 at the end position
- Arithmetic shift left inserts 0 at the end (identical to logical left shift) and arithmetic shift right leave the sign bit unchanged (should preserve the sign).
- Rotate instructions produce a circular shift.
- Rotate left through carry instruction transfers carry bit to right and so is for rotate shift right.