www.teachics.org

Basic Computer Organization and Design

Computer Organization & Architecture - MODULE 3 PART 1

Instruction Codes

Instruction Codes

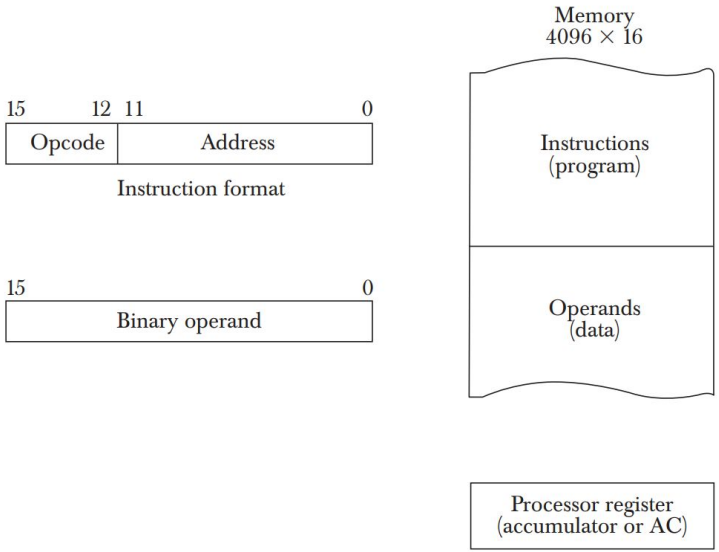
- The organization of the computer is defined by its internal registers, the timing and control structure, and the set of instructions that it uses.
- A **computer instruction** is a binary code that specifies a sequence of microoperations for the computer.
- An **instruction code** is a group of bits that instruct the computer to perform a specific operation.
- Instruction code is usually divided into two parts.
    - **Operation part** - Group of bits that define such operations as add, subtract, multiply, shift, and complement.
    - **Address part** - Contains registers or memory words where the address of operand is found or the result is to be stored.
- Each computer has its own instruction code format.

Operation Code

- The **operation code(op-code)** of an instruction is a group of bits that define such operations as add, subtract, multiply, shift, and complement.
- The number of bits required for the operation code of an instruction depends on the total number of operations available in the computer.(n bits for 2^n operations)
- An operation code is sometimes called a **macro-operation** because it specifies a set of micro-operations.

Stored Program Organization

- Simplest way to organize computer is to have one processor register(Accumulator AC) and an instruction code format with two parts.
  - **First**-Operation to be performed
  - **Second** – Address
- The memory address tells the control where to find an operand in memory.
- This operand is read from memory and used as the data to be operated on together with the data stored in the processor register.



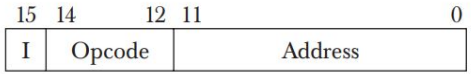
- Instructions are stored in one section of the memory and data in another.
- For a memory unit with 4096 words we need 12 bits to specify an address since 2^{12}=4096.
- 4 bits are available for opcode to specify one out of 16 possible operations.

Steps

- The control reads a 16-bit instruction from the program portion of memory.
- It uses the 12-bit address part of the instruction to read a 16-bit operand from the data portion of memory.
- It then executes the operation specified by the operation code.
- The operation is performed with the memory operand and the content of AC.

Direct & Indirect Addressing Modes

- Following **Addressing Modes** are used for address portion of the instruction code.
  - **Immediate**- The address part specifies an operand.
    Eg: ADD 5
  - **Direct**- The address part specifies the address of an operand.
  - **Indirect**- The address part specifies a pointer(another address) where the address of the operand can be found.
- One bit of the instruction code(**I**) can be used to distinguish between a direct and an indirect address.

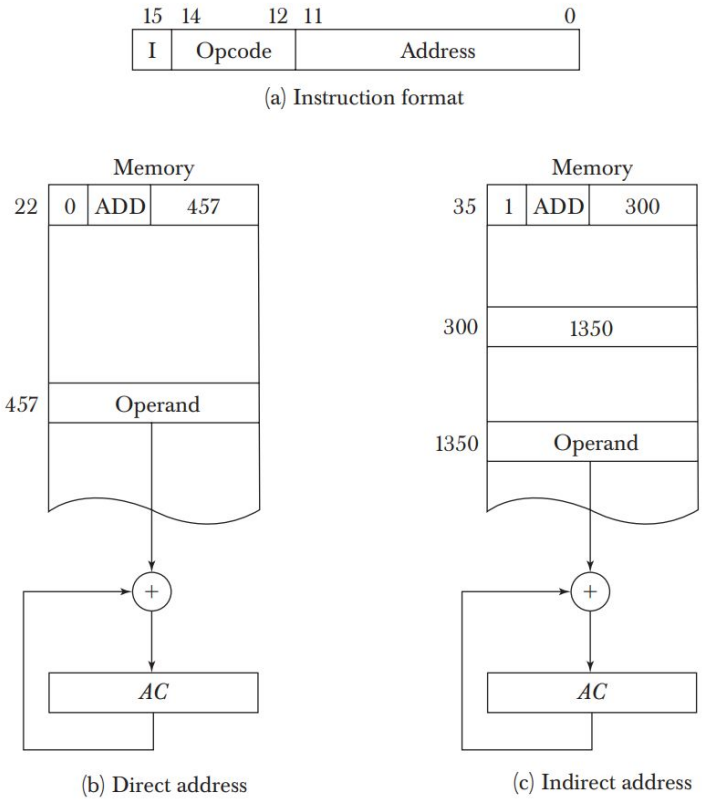


(a) Instruction format

Effective Address

It is the address of the operand in a computation-type instruction or the target address in a branch-type instruction.

The effective address in the instruction of Fig.(b) is **457** and in the instruction of Fig(c) is **1350**.



Computer Registers

Computer Registers

Need of Registers?

- Computer instructions are normally stored in consecutive 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, and so on.
- This type of instruction sequencing needs a **counter** to calculate the address of the next instruction after execution of the current instruction is completed.
- It is also necessary to provide a register in the control unit 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**.

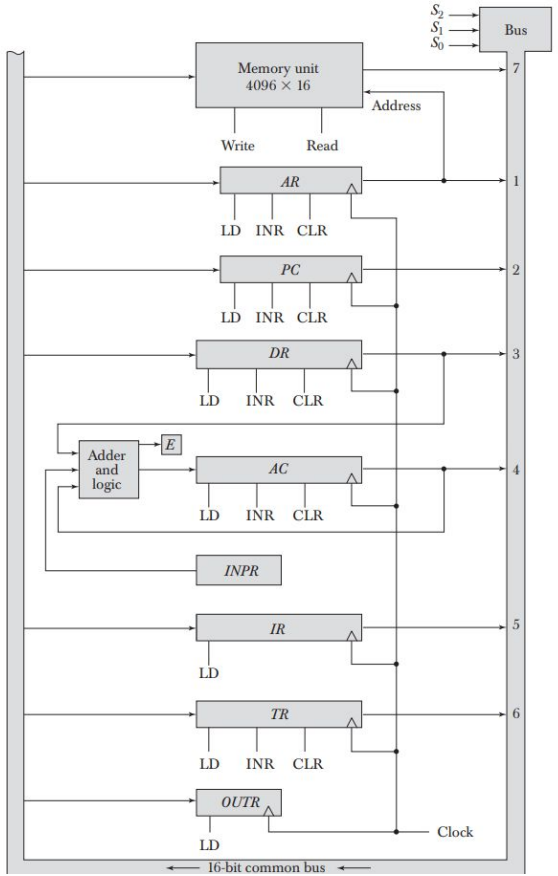
List of basic Registers

Code	Bits	Name	Purpose
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

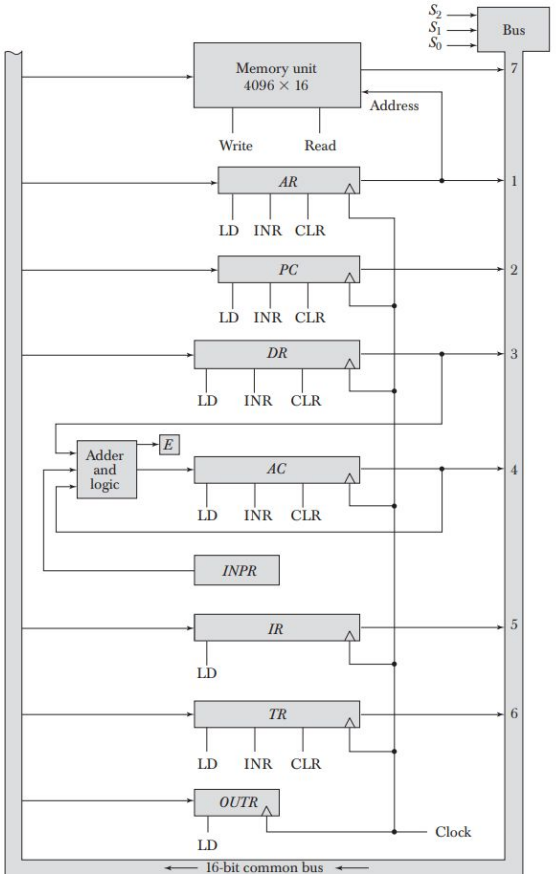
Common Bus System

Need of Common Bus System ?

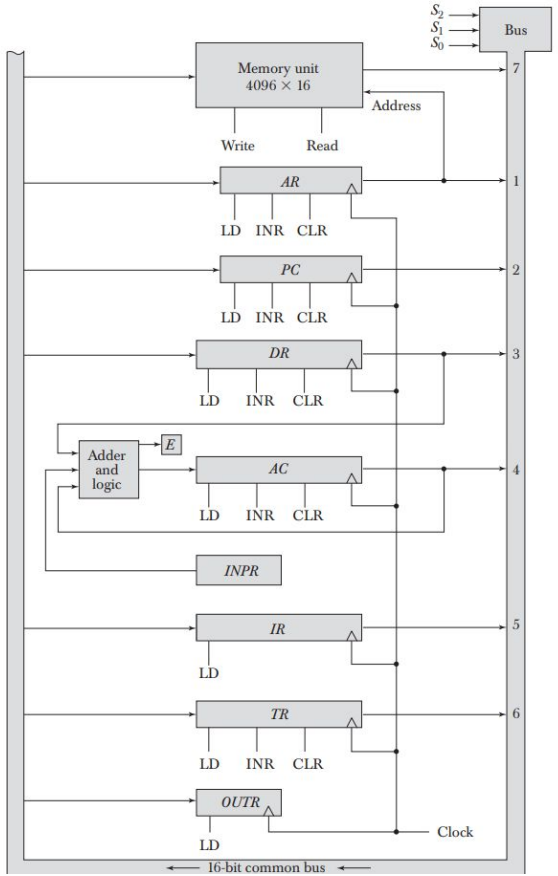
- The basic computer has eight registers, a memory unit, and a control unit.
- Paths must be provided to transfer information from one register to another and between memory and registers.
- The number of wires will be excessive if connections are made between the outputs of each register and the inputs of the other registers.
- Hence more efficient scheme with a common bus is used.



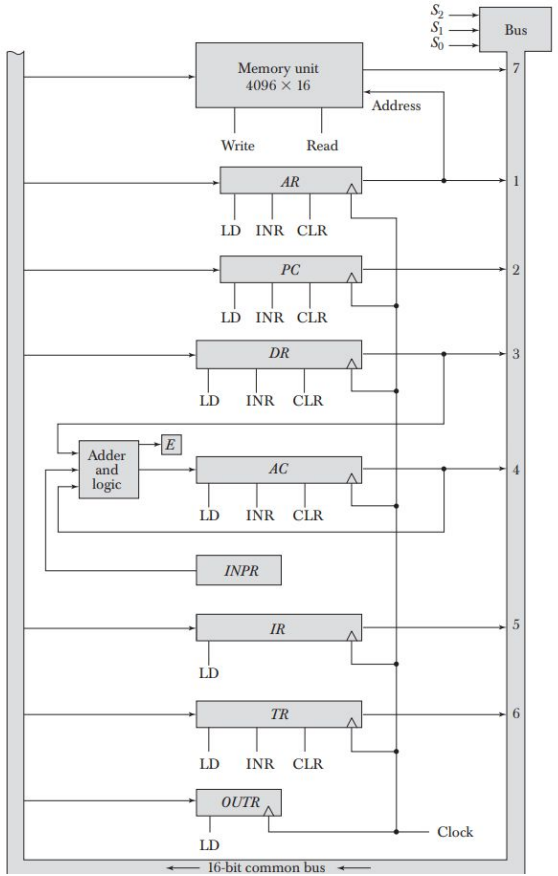
- The outputs of seven registers and memory are connected to the common bus.
- The specific output that is selected for the bus lines at any given time is determined from the binary value of the selection variables S_2, S_1, and S_0.
- The number along each output shows the decimal equivalent of the required binary selection.
- For example, the number along the output of DR is 3. The 16-bit outputs of DR are placed on the bus lines when $S_2S_1S_0$ =011 since this is the binary value of decimal 3.



- The lines from the common bus are connected to the inputs of each register and the data inputs of the memory.
- The particular register whose LD (load) input is enabled receives the data from the bus during the next clock pulse transition.
- The memory receives the contents of the bus when its write input is activated.
- The memory places its 16-bit output onto the bus when the read input is activated and $S_2S_1S_0$ =111.



- INPR is connected to provide information to the bus but OUTR can only receive information from the bus.
- This is because INPR receives a character from an input device which is then transferred to AC.
- OUTR receives a character from AC and delivers it to an output device.
- There is no transfer from OUTR to any of the other registers.



- The inputs of AC come from an adder and logic circuit.
- This circuit has three sets of inputs.
- One set of 16-bit inputs come from the outputs of AC. They are used to implement register micro-operations such as complement AC and shift AC.
- Another set of 16-bit inputs come from the data regisler DR. The inputs from DR and AC are used for arithmetic and logic microoperations.
- A third set of 8-bit inputs come from the input register INPR.

Computer Instructions

Instruction Format

- The basic computer has three instruction code formats each having 16 bits
  - Memory reference instructions
  - Register reference instructions
  - I/O instructions
- The opcode part of the instruction contains three bits and the meaning of the remaining 13 bits depends on the operation code encountered.

Memory reference instructions

- Bits 0-11 for specifying address.
- Bits 12-14 for specifying address.
- 15th bit specifies addressing modes. (0 for direct and 1 for indirect)

- Opcode=000 through 110
- I=0 or 1
- Eg:
  - AND - 0xxx(direct) or 8xxx(indirect)
  - ADD - 1xxx or 9xxx

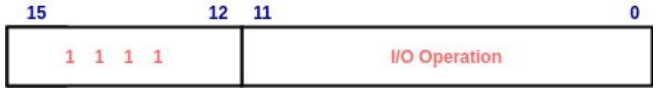
Register reference instructions

- Recognized by the operation code 111 with a 0 in the 15th bit of the instruction.
- Specifies an operation on or a test of the AC register.
- An operand from memory is not needed.
- Therefore the 12 bits are used to specify the operation to be executed.

- Eg:
  - CLA - 7800 : Clear AC
  - CLE - 7400 : Clear E

I/O instructions

- These instructions are needed for transfering informations to and from AC register.
- Recognized by the opcode 111 and a 1 in the 15th bit.
- Bits 0-11 specify the type of I/O Operation performed.



- Eg:
  - INP - F800 : Input characters to AC
  - OUT - F400 : Output characters from AC

Symbol	Hexadecimal code		Description
	$I = 0$	$I = 1$	
AND	0xxx	8xxx	AND memory word to *AC*
ADD	lxxx	9xxx	Add memory word to *AC*
LDA	2xxx	Axxx	Load memory word to *AC*
STA	3xxx	Bxxx	Store content of *AC* in 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 instruction if *AC* positive
SNA	7008		Skip next instruction if *AC* negative
SZA	7004		Skip next instruction if *AC* zero
SZE	7002		Skip next instruction if *E* is 0
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

Instruction Set Completeness

- The set of instructions are said to be complete if the computer includes a sufficient number of instructions in each of the following categories
  a. Arithmetic, logical, and shift instructions
  b. Instructions for moving information to and from memory and processor registers
  c. Program control instructions
  d. Input and output instructions

Timing and Control

Timing and Control

- 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.
- The clock pulses do not change the state of a register unless the register is enabled by a control signal.
- Two major types of control organization:
  - hardwired control
  - microprogrammed control.

Hardwired Control

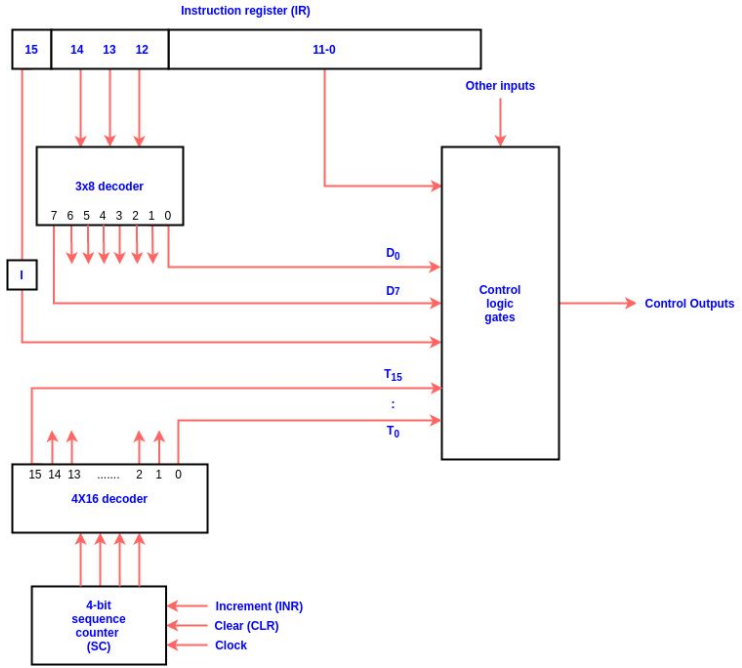
- The control logic is implemented with gates, flip-flops, decoders, and other digital circuits.
- It can be optimized to produce a fast mode of operation.
- Requires changes in the wiring among the various components if the design has to be modified or changed.

Microprogrammed Control

- The control information is stored in a control memory.
- The control memory is programmed to initiate the required sequence of microoperations.
- Required changes or modifications can be done by updating the microprogram in control memory.

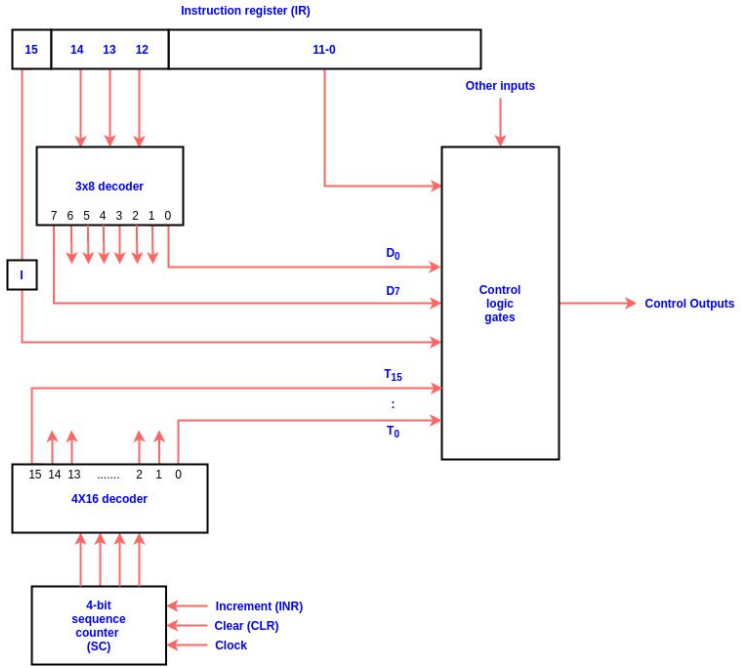
Hardwired control unit

- Consists of two decoders, a sequence counter, and a number of control logic gates.
- An instruction read from memory is placed in the instruction register(IR).
- The IR is divided into three parts:
  - I bit, opcode, and Address bits.
- Op-code in 12-14 bits are decoded with a 3x8 decoder.
- 8 outputs of the decoder are designated by the symbols D0 through D7.
- Bit 15 is transferred to a flip-flop I.
- Bits 0-11 are applied to the control logic gates.



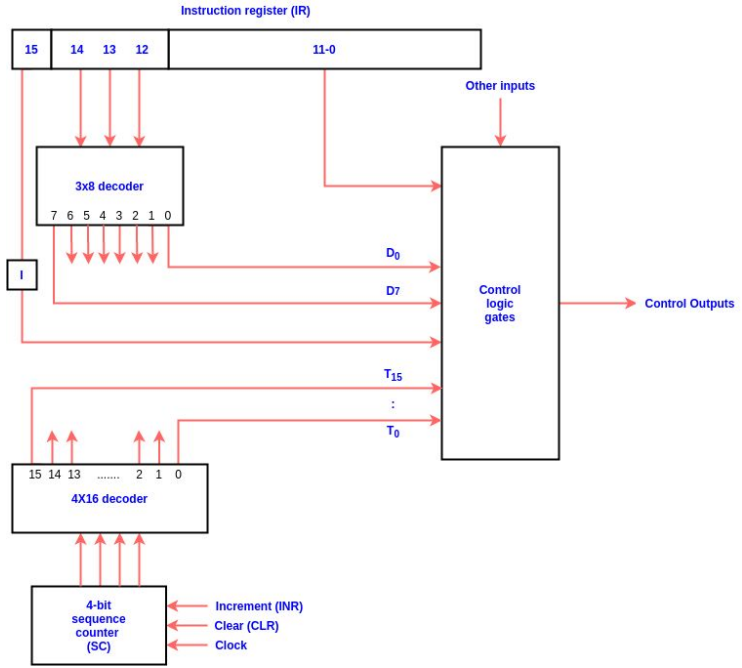
Hardwired control unit

- The 4-bit sequence counter can count in binary from 0-15.
- The outputs of the counter are decoded into 16 timing signals T_0-T_{15}.
- The sequence counter SC can be incremented or cleared synchronously.
- Mostly,SC is incremented to provide the sequence of timing signals(T_1,T2,...,T15)
- Once in a while, the counter is cleared to 0, causing the next active timing signal to be T_0.
- Eg: Suppose,at time T4, SC is cleared to 0 if decoder output D3 is active. D_3T_4: SC ⟵ 0.



Hardwired control unit

- The SC responds to the positive transition of the clock.
- Initially, the CLR input of SC is active.
- Hence it clears SC to 0, giving the timing signal T_0 out of the decoder.
- T0 is active during one clock cycle and will trigger only those registers whose control inputs are connected to timing signal T0.
- SC is incremented with every positive clock transition, unless its CLR input is active.
- This produces the sequence of timing signals T_0, T_1, T_2, T_3, T_4 up to T_{15} and back to T_0.



www.teachics.org

Basic Computer Organization and Design

Computer Organization & Architecture - MODULE 3
PART 2

Instruction Cycle

Instruction Cycle

- A program consists of a sequence of instructions is executed in the computer by going through a cycle for each instruction.
- Each instruction cycle in turn is subdivided into a sequence of subcycles or phases
- They are
    a. Fetch an instruction from memory.
    b. Decode the instruction.
    c. Read the effective address from memory if the instruction has an indirect address.
    d. 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.

Fetch & Decode

- Initially, the program counter PC is loaded with the address of the first instruction in the program.
- SC is cleared to 0, providing a decoded timing signal T0.
- After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence T0, T1, T2, and so on.
- The microoperations for the fetch and decode phases are
  - T_0: AR ← PC
  - T_1: IR ← M [AR], PC ← PC+1
  - T_2: D0,. . . , D7 ← Decode IR(12–14), AR ← IR(0-11), I ← IR(15)

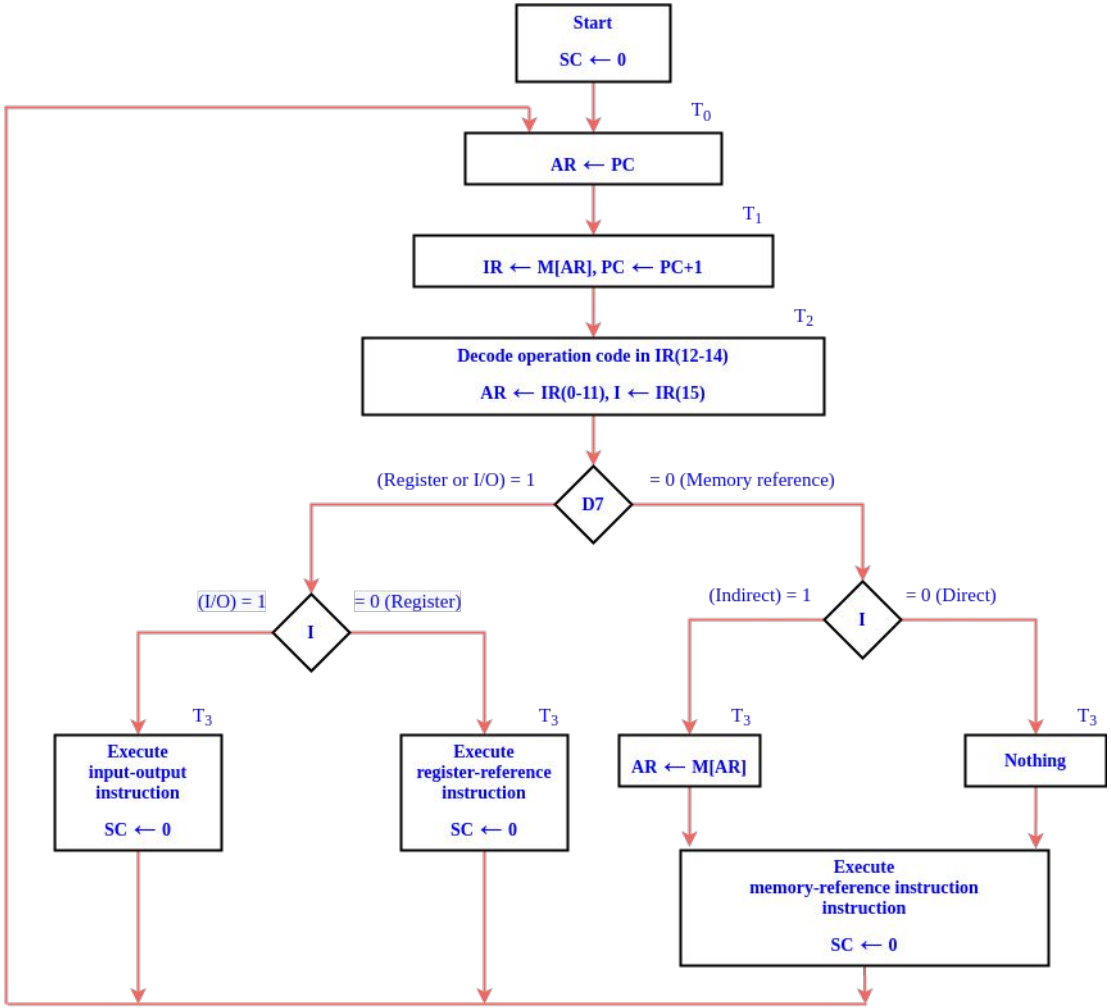
Fetch & Decode

- During T_0 the address is transferred from PC to AR.
- At T_1 The instruction read from memory is placed in the instruction register IR and PC is incremented by one to prepare it for the address of the next instruction.
- At time T2, the op-code in IR is decoded, the indirect bit is transferred to flip-flop I, and the address part of the instruction is transferred to AR.
- Note that SC is incremented after each clock pulse to produce the sequence T_0, T_1, and T_2.

Determine the Type of Instruction

- During time T3, the control unit determines the type of instruction that was just read from memory.
- Memory Reference Instructions
  - If D7=0, the op-code will be 000 through 110
  - If D7 =0 and I =1 - memory reference instruction with an indirect address.
  - The micro-operation for the indirect address condition can be symbolized by AR ← M [AR].
- Register Reference or I/O Instructions
  - If D_7 =1 and I=0 - Register Reference Instruction.
  - If D_7 =1 and I=1 - I/O Instruction.

- The three instruction types are subdivided into four separate paths.
- The selected operation is activated with T_3.
  - D7' IT_3 : AR ← M[AR].
  - D7' IT_3 : Nothing.
  - D7 I'T_3 : Execute a register-reference instruction.
  - D7 IT_3 : Execute an input–output instruction.


Start
SC ← 0
T0
AR ← PC
T1
IR ← M[AR], PC ← PC+1
T2
Decode operation code in IR(12-14)
AR ← IR(0-11), I ← IR(15)
D7
(Register or I/O) = 1
= 0 (Memory reference)
I
(I/O) = 1
= 0 (Register)
I
(Indirect) = 1
= 0 (Direct)
T3
Execute input-output instruction
SC ← 0
T3
Execute register-reference instruction
SC ← 0
T3
AR ← M[AR]
T3
Nothing
Execute memory-reference instruction instruction
SC ← 0

Memory Reference Instruction

Memory Reference Instruction

- Memory reference instructions performs operation with memory operand.

- Execution of memory reference instruction starts with timing signal T_4.
- There are 7 memory instruction.

Symbol	Operation decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR]$, $E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC$, $PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1$, If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

AND to AC

- Performs AND logic operation on pairs of bits in AC and the memory word specified by effective address.
- The result of operation is transferred to AC.

- D_0T_4: DR ← M [AR]
  D_0T_5: AC ← AC ^ DR, SC ← 0

ADD to AC

- Adds the content of the memory word specified by the effective address to the value of AC.
- Sum is transferred to AC and output carry C_{out} is transferred into E (Extended accumulator flipflop)

- D_1T_4: DR ← M [AR]
  D_1T_5: AC ← AC ^ DR, E ← C_{out}, SC ← 0

LDA : Load to AC

- Transfers the memory word specified by the effective address to AC.
- D_2T_4: DR ← M [AR]
  D_2T_5: AC ← DR, SC ← 0

STA : Store to AC

- Stores the content of AC into the memory word specified by the effective address.
- D_3T_4: M [AR] ← AC, SC ← 0

BUN : Branch Unconditionally

- Transfers the program to the instruction specified by the effective address.
- D_4T_4: PC ← AR, SC ← 0

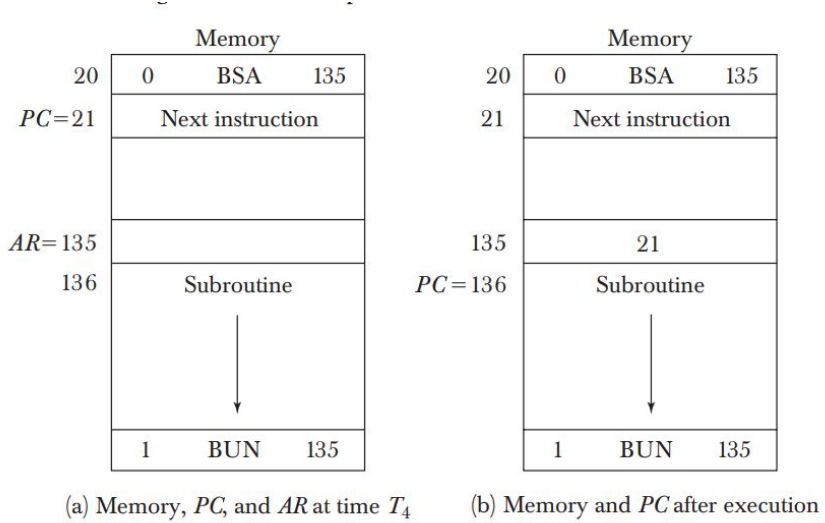
ISZ: Increment and Skip if Zero

- Increments the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1 in order to skip the next instruction in the program.
- D_6T_4: DR ← M [AR]
  D_6T_5: DR ← DR+1
  D_6T_6: M [AR] ← DR, if (DR 0) then (PC ← PC 1), SC ← 0

BSA : Branch and Save Return Address

- Used for branching to a portion of the program called a subroutine or procedure.
- It stores the address of the next instruction into a memory location specified by the effective address.
- The effective address plus one is transferred to PC to serve as the address of the first instruction in the subroutine.

- D_5T_4: M [AR] ← PC, AR ← AR + 1
  D_5T_5: PC ← AR, SC ← 0



(a) Memory, *PC*, and *AR* at time T_4 (b) Memory and *PC* after execution

Register Reference Instructions

Register Reference Instructions

- It specifies an operation on or test of the Accumulator.
- An operand from memory is not needed.
- So bits 0-11 bits the operation to be executed.
- They are recognized by the control when $D_7 = 1$ and $I = 0$.
- Execution start with the timing signal T_3.
- Each control function needs the Boolean relation D_7IT_3 and is represented by the symbol r.
- By assigning the symbol B, to bit i of IR(0-11), all control functions can be simply denoted by rBi.

$D_7 I' T_3 = r$ (common to all registcr-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:	$AC \leftarrow \text{shr } AC, AC(15) \leftarrow E, E \leftarrow AC(0)$	Circulate right
CIL	rB_6:	$AC \leftarrow \text{shl } AC, AC(0) \leftarrow E, E \leftarrow AC(15)$	Circulate left
INC	rB_5:	$AC^* \rightarrow AC + 1$	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

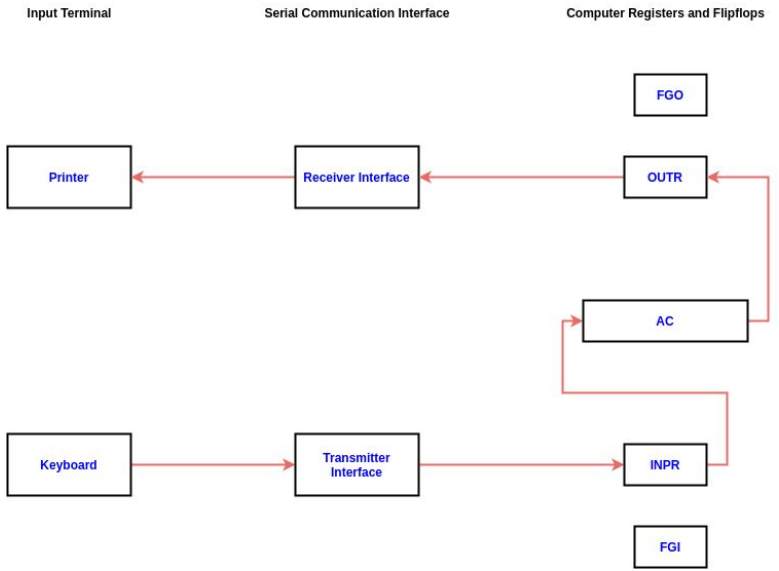
Input and Output Communication

Input–Output Communication

- A computer can serve no useful purpose unless it communicates with the external environment.
- Instructions and data stored in memory must come from some input device.
- Computational results must be transmitted to the user through some output device.
- To demonstrate the most basic requirements for input and output communication, we will use as an illustration a terminal unit with a keyboard and printer.

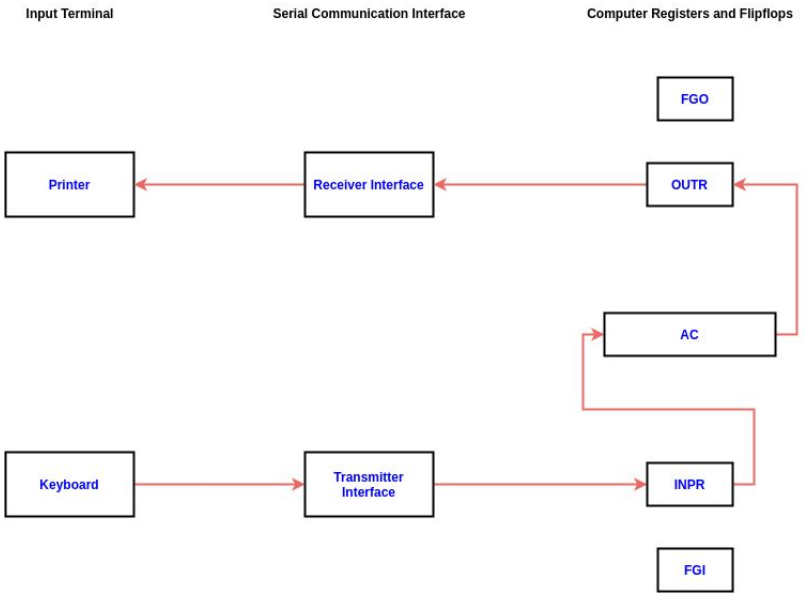
Input-Output Configuration

- Terminal sends and receives information (eight bits of an alphanumeric code).
- The serial information from the keyboard is shifted into the input register INPR.
- The serial information for the printer is stored in the output register OUTR.
- Both INPR and OUTR consists of eight bits.



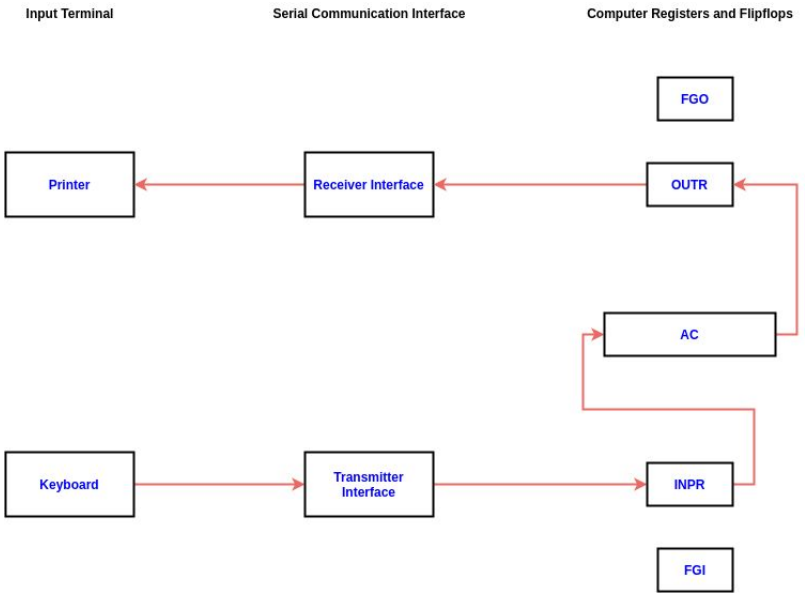
Input-Output Configuration

- INPR and OUTR communicate with a communication interface serially and with the AC in parallel.
- The transmitter interface receives serial information from the keyboard and transmits it to INPR.
- The receiver interface receives information from OUTR and sends it to the printer.



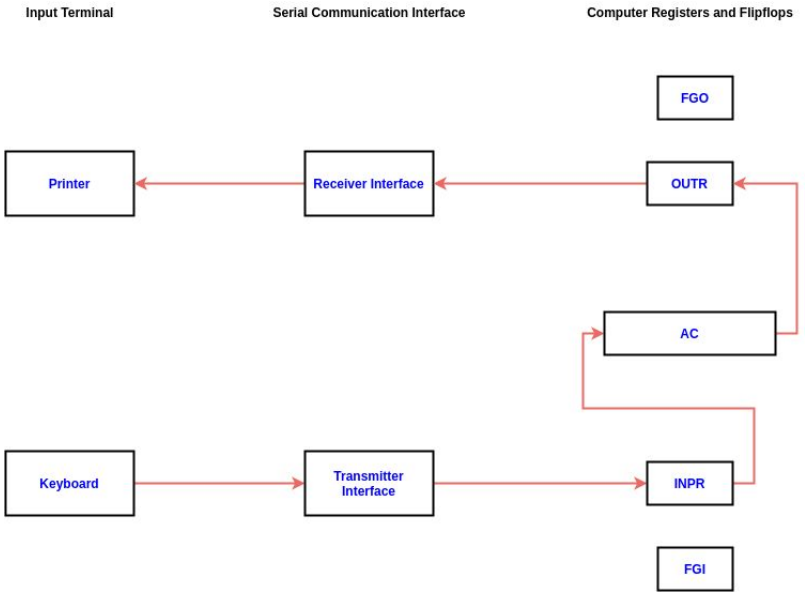
Input-Output Configuration

- The 1-bit input flag FGI is a control flip-flop.
- The flag bit is set to 1 when new information is available in the input device and is cleared to 0 when the information is accepted by the computer.
- Initially, the input flag FGI is cleared to 0.
- When a key is struck, an 8-bit code is shifted into INPR and FGI is set to 1.
- When another key is striked, the computer checks the flag bit; if it is 1, the information from INPR is transferred to AC and FGI is cleared to 0.



Input-Output Configuration

- Initially, the output flag FGO is set to 1.
- The computer checks the flag bit; if it is 1, the information from AC is transferred in parallel to OUTR and FGO is cleared to 0.
- The output device accepts information, prints the corresponding character, and when the operation is completed, it sets FGO to 1.
- The computer does not load a new character into OUTR when FGO is 0. (Busy printing)



Input-Output Instructions

- Needed for
  - transferring information to and from AC register.
  - for checking the flag bits.
  - for controlling the interrupt facility.
- Recognized by the control when D_7 =1 and I =1.
- The remaining bits(0-11) of the instruction specify the particular operation.
- Executed with the clock transition associated with timing signal T3.
- Each control function needs a Boolean relation D_7IT_3, and is represented by the symbol p.
- By assigning the symbol Bi to bit i of IR, all control functions can be denoted by pB_i for i = 6 through 11.

Input-Output Instructions

$D_7IT_3 = p$ (common to all input–output instructions)
$IR(i) = B_i$ [bit in $IR(6–11)$ that specifies the instruction]

	p:	$SC \leftarrow 0$	Clear SC
INP	pB_{11}:	$AC(0–7) \leftarrow INPR, \quad FGI \leftarrow 0$	Input character
OUT	pB_{10}:	$OUTR \leftarrow AC\,(0–7), \quad FGO \leftarrow 0$	Output character
SKI	pB_9:	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$	Skip on input flag
SKO	pB_8:	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$	Skip on output flag
ION	pB_7:	$IEN \leftarrow 1$	Interrupt enable on
IOF	pB_6:	$IEN \leftarrow 0$	Interrupt enable off

Program Interrupt

Need for Interrupt

- In basic case (Programmed control transfer), the computer keeps checking the flag bit, and when it finds it set, it initiates an information transfer.
- This type of transfer is inefficient because of the difference of information flow rate between the computer and that of the input–output device.
- The computer is wasting time while checking the flag instead of doing some other useful processing task.

Program Interrupt

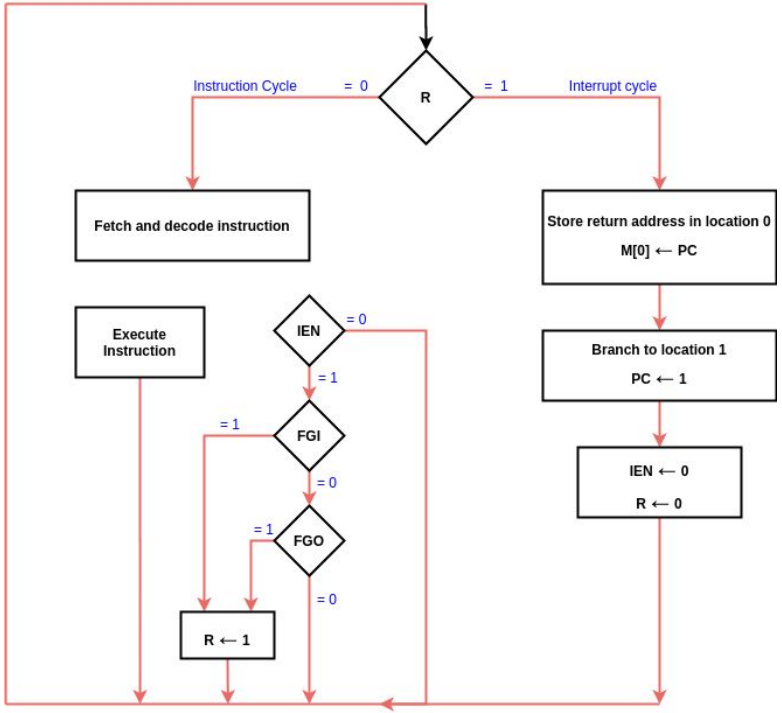
- An alternative to the programmed controlled procedure is to let the external device inform the computer when it is ready for the transfer.
- In the meantime the computer can be busy with other tasks.
- This type of transfer uses the interrupt facility.
- While the computer is running a program, it does not check the flags.
- When a flag is set, the computer is interrupted from proceeding with the current program.
- The computer stops what it is doing to take care of the input or output transfer.
- It then returns to the current program to continue what it was doing before the interrupt

Program Interrupt

- The interrupt facility can be enabled or disabled by a flip-flop IEN.
- The interrupt enable flip-flop IEN can be set and cleared with two instructions (IOF, ION).
  - When IEN is cleared to 0 (with the IOF instruction), the flags cannot interrupt the computer.
  - When IEN is set to 1 (with the ION instruction), the computer can be interrupted.
- An interrupt flip-flop R is included in the computer to decide when to go through the interrupt cycle.
- So the computer is either in an instruction cycle or in an interrupt cycle.

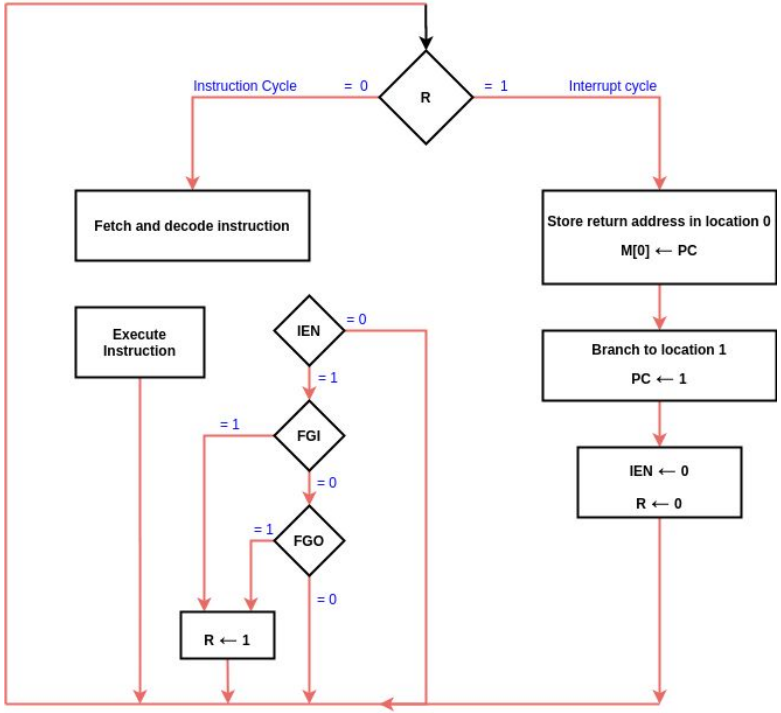
Interrupt Cycle

- When R = 0, the computer goes through an instruction cycle.
- During the execute phase IEN is checked. If it is 0, control continues with the next instruction cycle.
- If IEN = 1, control checks the flag bits. If both flags are 0, control continues with the next instruction cycle.
- If either flag is set to 1 while IEN = 1, R is set to 1 and control goes to an interrupt cycle.



Interrupt Cycle

- The interrupt cycle is a hardware implementation of a branch and save return address operation.
- The return address in PC is stored in a specific location. (Here address 0)
- Control then inserts address 1 into PC and clears IEN and R so that no more interruptions can occur until the interrupt request serviced and flag has been set.
- Micro-operations
  - RT_0: AR ← 0, TR ← PC
  - RT_1: M [AR] ← TR, PC ← 0
  - RT_2: PC ← PC 1, IEN ← 0, R ← 0, SC ← 0



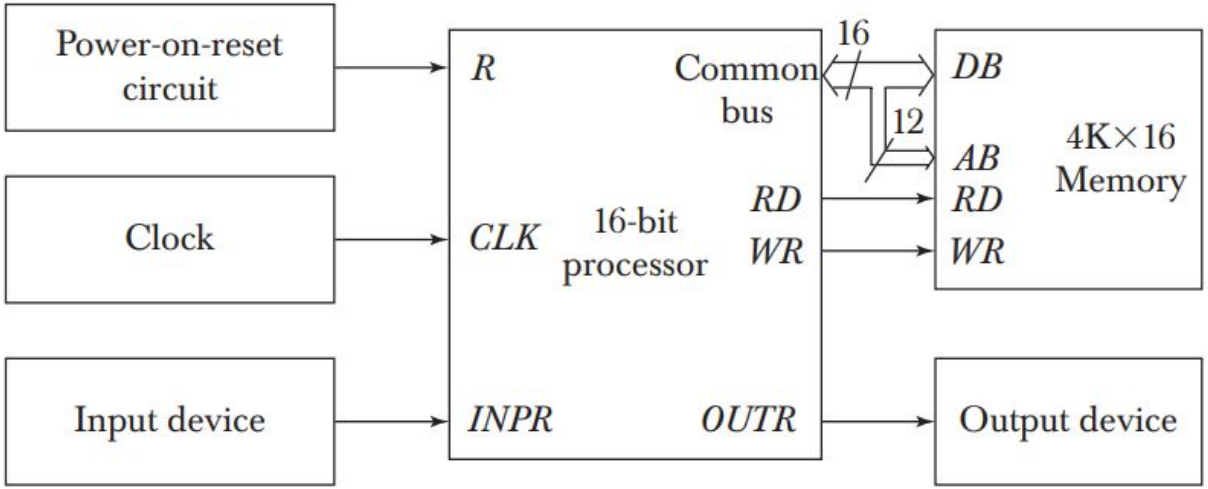
Design of Basic Computer

Design of Basic Computer

The basic computer consists of the following hardware components:

1. A memory unit with 4096 words of 16 bits each.
2. Nine registers: AR, PC, DR, AC, IR, TR, OUTR, INPR, and SC.
3. Seven flip-flops: I, S, E, R, IEN, FGI, and FGO.
4. Two decoders: a 3x8 operation decoder and a 4x16 timing decoder.
5. A 16-bit common bus.
6. Control logic gates.
7. Adder and logic circuit connected to the input of AC.

Design of Basic Computer



Control Logic Gates

- The inputs to this circuit come from
  - two decoders
  - I flip-flop
  - bits 0 through 11 of IR.
- The other inputs to the control logic are
  - AC bits 0 through 15 to check if AC =0 and to detect the sign bit in AC(15)
  - DR bits 0 through 15 to check if DR= 0
  - The values of the seven flip-flops.

- The outputs of the control logic circuit are
  - Signals to control the inputs of the nine registers
  - Signals to control the read and write inputs of memory
  - Signals to set, clear, or complement the flip-flops
  - Signals for S_2, S_1, and S_0 to select a register for the bus
  - Signals to control the AC adder and logic circuit

Control of Registers and Memory

- The control inputs of the registers are LD (load), INR (increment), and CLR (clear).
- Eg: To derive the gate structure associated with the control inputs of AR.
  Find all the statements that change the content of AR
  $R'T_0$: AR ← PC
  $R'T_2$: AR ← IR(0–11)
  $D_7'IT3$: AR ← M [AR]
  RT_0: AR ← 0
  D_5T_4: AR ← AR + 1
- The control functions can be combined into three Boolean expressions as follows
  $LD(AR) = R'T_0 + R'T_2 + D_7'IT_3$
  $CLR(AR) = RT_0$
  $INR(AR) = D_5T_4$

Control of Common Bus

- The 16-bit common bus is controlled by the selection inputs S_2, S_1, and S_0.
- The decimal number shown with each bus input specifies the equivalent binary number that must be applied to the selection inputs in order to select the corresponding register.
- For example, when $x_1 = 1$, the value of $S_2S_1S_0$ must be 001 and the output of AR will be selected for the bus.

Inputs							Outputs			Register selected for bus
x_1	x_2	x_3	x_4	x_5	x_6	x_7	S_2	S_1	S_0	
0	0	0	0	0	0	0	0	0	0	None
1	0	0	0	0	0	0	0	0	1	*AR*
0	1	0	0	0	0	0	0	1	0	*PC*
0	0	1	0	0	0	0	0	1	1	*DR*
0	0	0	1	0	0	0	1	0	0	*AC*
0	0	0	0	1	0	0	1	0	1	*IR*
0	0	0	0	0	1	0	1	1	0	*TR*
0	0	0	0	0	0	1	1	1	1	Memory

Control of Common Bus

- The Boolean functions for the encoder are

  $S_0 = x_1 + x_3 + x_5 + x_7$

  $S_1 = x_2 + x_3 + x_6 + x_7$

  $S_2 = x_4 + x_5 + x_6 + x_7$
- To determine the logic for each encoder input, it is necessary to find the control functions that place the corresponding register onto the bus.
- For example, to find the logic that makes $x_1 = 1$, we scan all register transfer statements in and extract those statements that have AR as a source.

  D_4T_4: PC ← AR

  D_5T_5: PC ← AR

  Therefore, the Boolean function for x_1 is, $x_1 = D_4T_4\ D_5T_5$
- In a similar manner we can determine the gate logic for the other registers.

Design of Accumulator Logic

Design of Accumulator Logic

- The adder and logic circuit has three sets of inputs.
  - 16 inputs comes from the outputs of AC.
  - 16 inputs comes from the data register DR.
  - eight inputs comes from the input register INPR.
- The outputs of the adder and logic circuit provide the data inputs for the register.

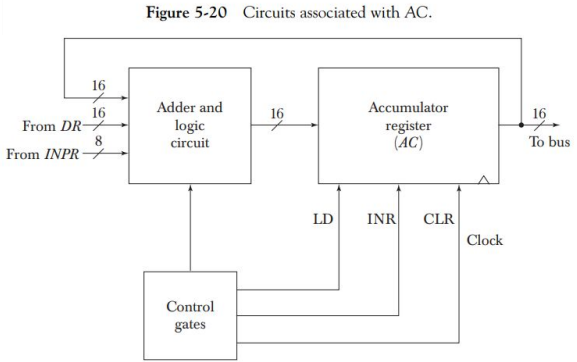


Figure 5-20 Circuits associated with AC.

Design of Accumulator Logic

- In order to design the logic associated with AC, extract all the statements that change the content of AC.

  D_0T_5: AC ← AC ^ DR
  D_1T_5: AC ← AC + DR
  D_2T_5: AC ← DR
  pB_{11}: AC(07) ← INPR
  rB_9: AC ← AC
  rB_7: AC ← shr AC, AC(15) ← E
  rB_6: AC ← shl AC, AC(0) ← E
  rB_{11}: AC ← 0
  rB_5: AC ← AC + 1
- From this list we can derive the control logic gates and the adder and Logic circuit.