UNIT 1

Functional Blocks of a Computer: Introduction, Block diagram of digital computer, Instruction codes, Computer Registers, Common bus system, Computer instructions, Instruction cycle and Instruction set, Register Transfer Language.

Data Representation: Fixed and floating point arithmetic- Addition, Subtraction, Multiplication, Division.

Control unit Design: Hardwired control unit, Control memory, Address sequencing, Micro-programmed control unit design, Hardwired Vs Micro-programmed design.

Introduction

- The hardware of the computer consists of all the electronic components and electromechanical devices that comprise the physical entity of the device.

- Computer software consists of the instructions and data that the computer manipulates to perform various data-processing tasks.

- A sequence of instructions for the computer is called a program.

- Computer organization is concerned with the way the hardware components operate and the way they are connected together to form the computer system.
- Computer design is concerned with the hardware design of the computer. Once the computer specifications are formulated, it is the task of the designer to develop hardware for the system.
- Computer architecture is concerned with the structure and behavior of the computer as seen by the user. It includes the information formats, the instruction set, and techniques for addressing memory.

Block diagram of digital computer

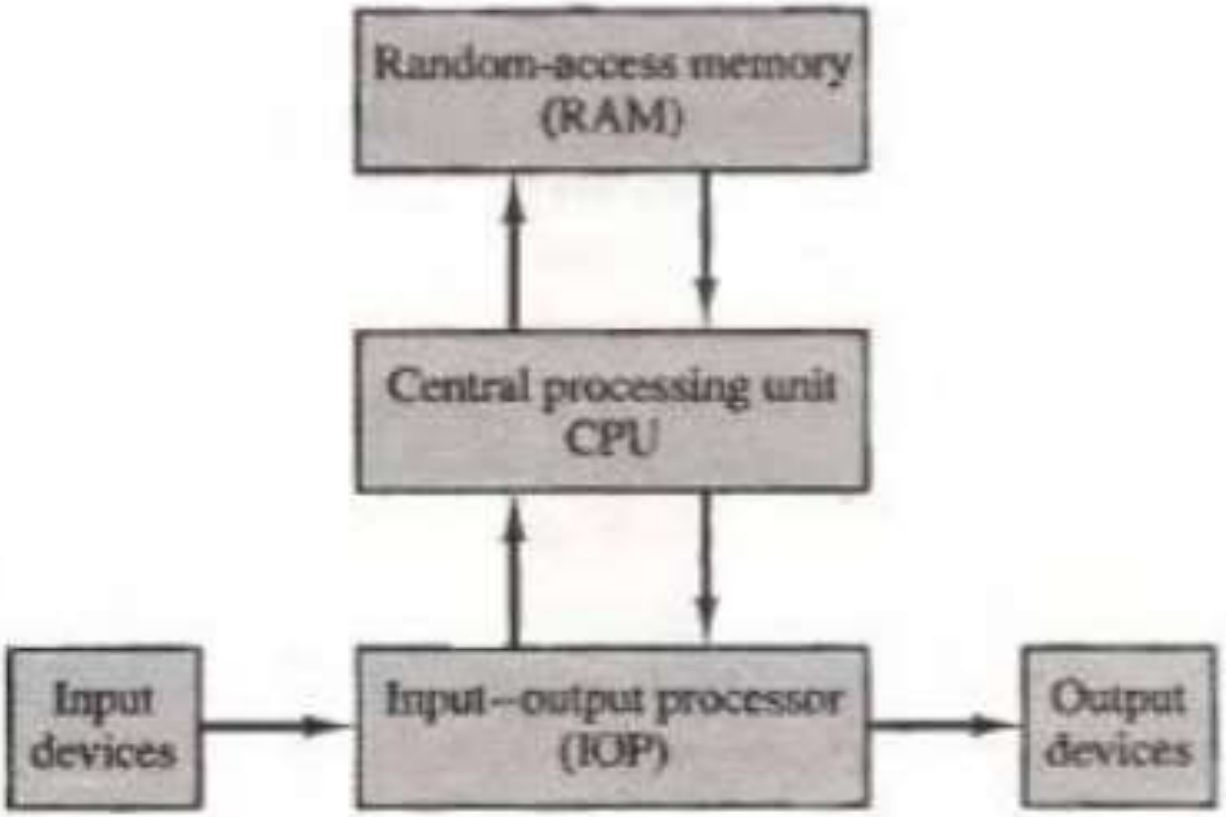


Figure 1-1 Block diagram of a digital computer.

- The central processing unit (CPU) contains an arithmetic and logic unit for manipulating data, a number of registers for storing data, and control circuits for fetching and executing instructions.

- The memory of a computer contains storage for instructions and data. It is called a random access memory (RAM) because the CPU can access any location in memory at random and retrieve the binary information within a fixed interval of time.

- The input and output processor (IOP) contains electronic circuits for communicating and controlling the transfer of information between the computer and the outside world.

- The input and output devices connected to the computer include keyboards, printers, terminals, magnetic disk drives, and other communication devices.

Instruction Codes

- The Internal organization of a digital system is defined by the sequence of micro operations it performs on data stored in its registers

- The user of a computer can control the process by means of a program

- A *program* is a set of instructions that specify the operations, operands, and the processing sequence

Instruction Codes cont.

- A *computer instruction* is a binary code that specifies a sequence of micro-operations for the computer. Each computer has its unique instruction set

- Instruction codes and data are stored in memory

- The computer reads each instruction from memory and places it in a control register

- The control unit interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of micro-operations

Instruction Codes cont.

- An Instruction code is a group of bits that instructs the computer to perform a specific operation (sequence of microoperations). It is divided into parts (basic part is the operation part)

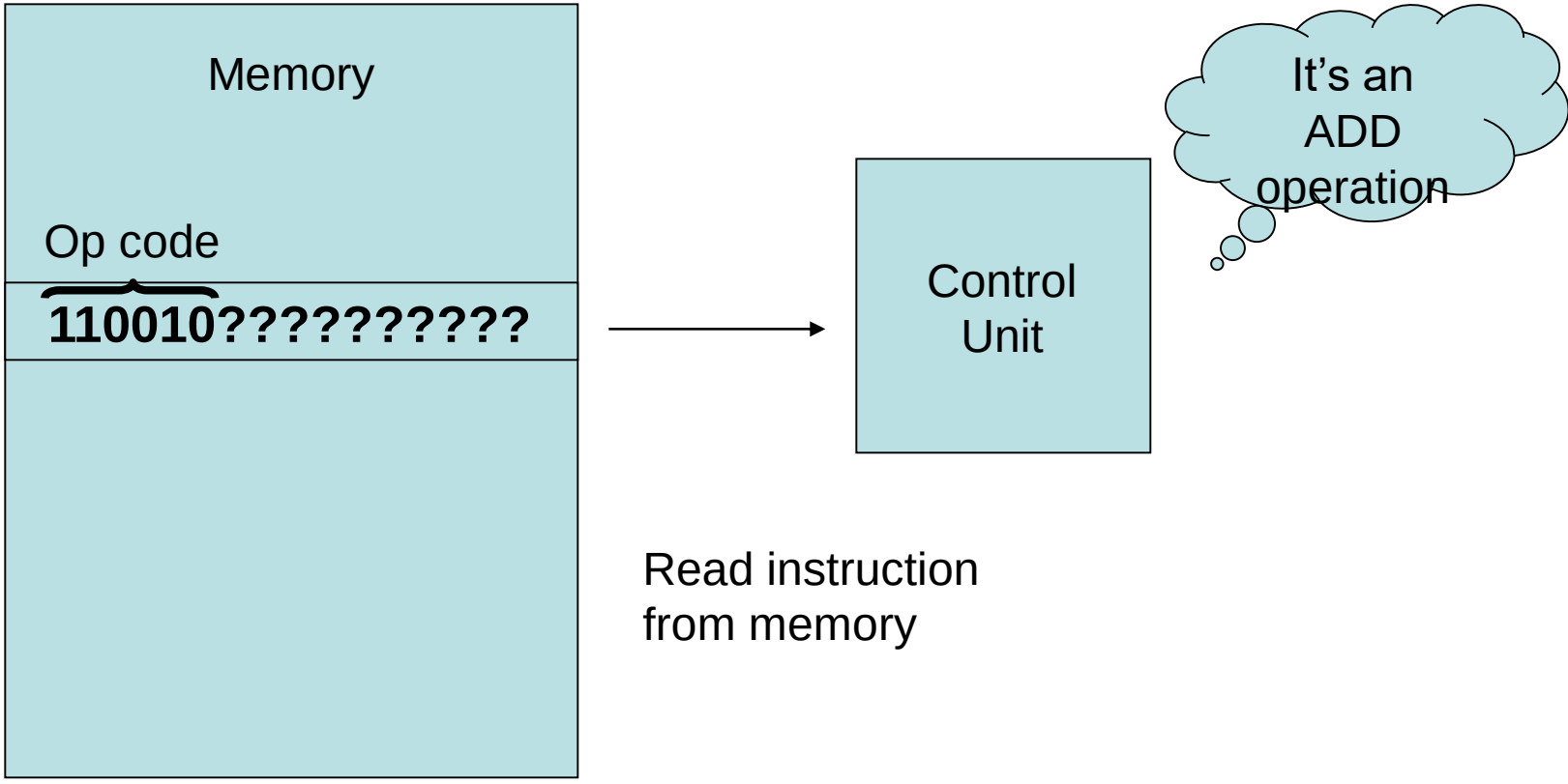
- The operation code of an instruction is a group of bits that defines certain operations such as add, subtract, shift, and complement

Instruction Codes cont.

- The number of bits required for the operation code depends on the total number of operations available in the computer

- 2^n (or little less) distinct operations → n bit operation code

Instruction Codes cont.



Instruction Codes cont.

- An operation must be performed on some data stored in processor registers or in memory

- An instruction code must therefore specify not only the operation, but also the location of the operands (in registers or in the memory), and where the result will be stored (registers/memory)

Instruction Codes cont.

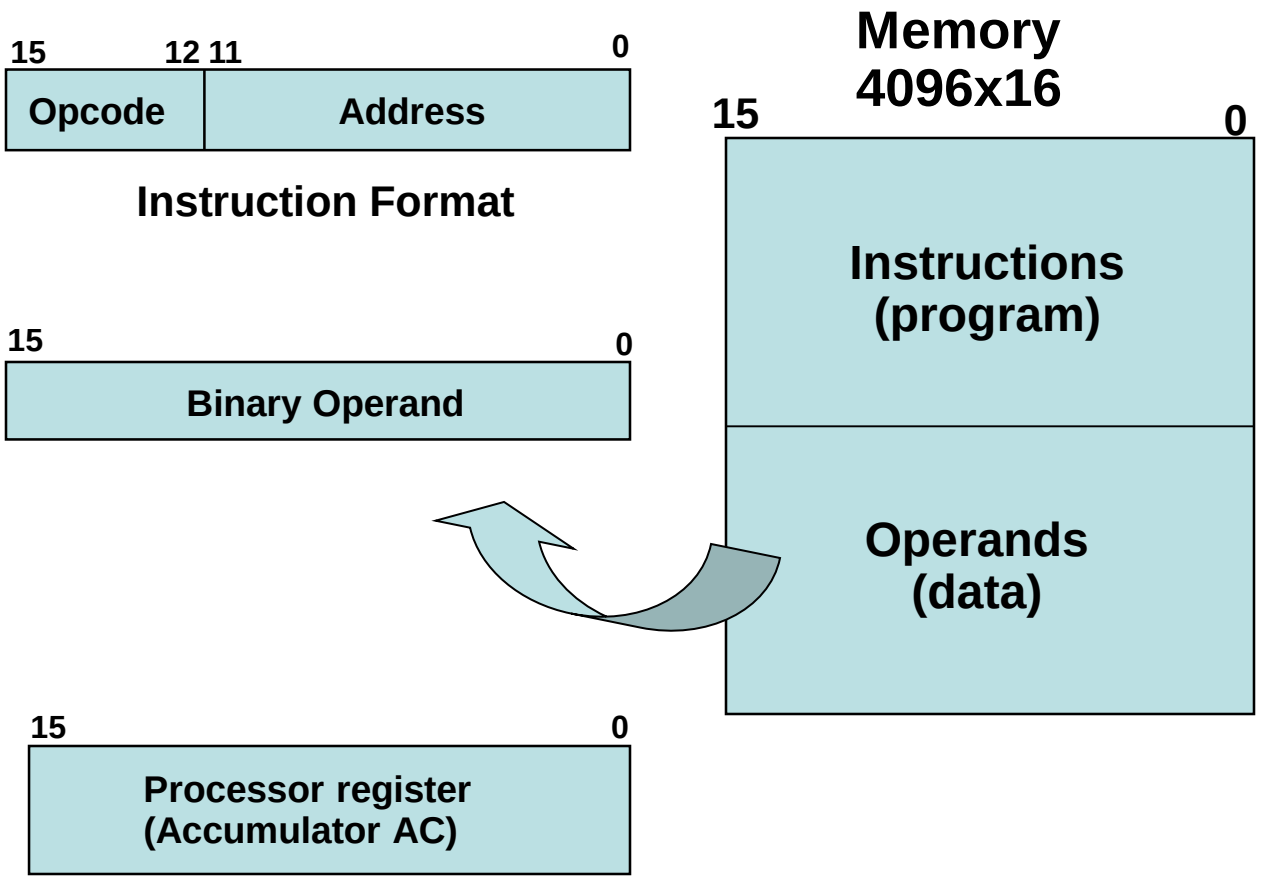
- Memory words can be specified in instruction codes by their address
- Processor registers can be specified by assigning to the instruction another binary code of k bits that specifies one of 2^k registers
- Each computer has its own particular instruction code format
- Instruction code formats are conceived by computer designers who specify the architecture of the computer

Instruction Codes cont.
Stored Program Organization

- An instruction code is usually divided into *operation code*, *operand address*, *addressing mode*, etc.

- The simplest way to organize a computer is to have one processor register (accumulator AC) and an instruction code format with two parts (op code, address)

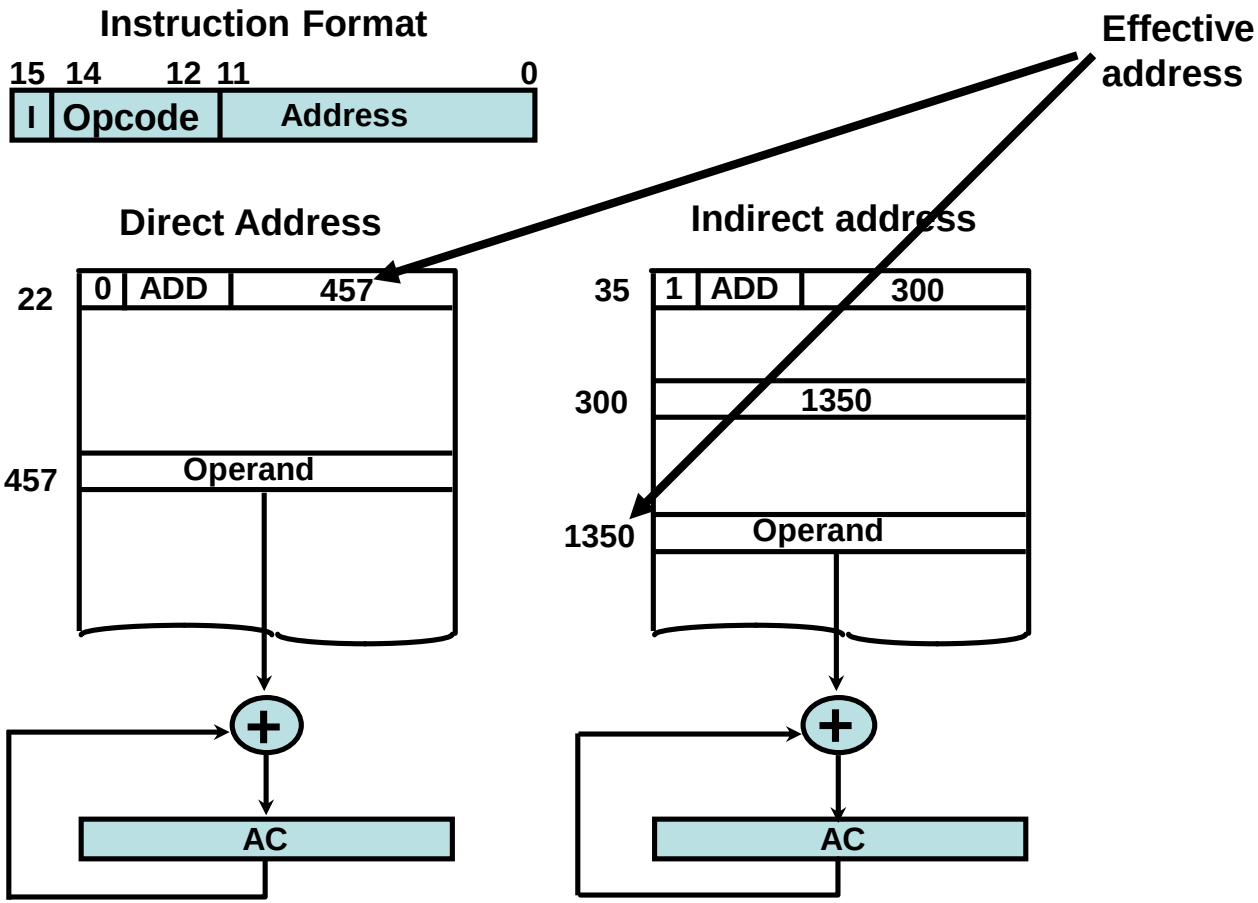
Instruction Codes
Stored Program Organization
cont.



Instruction Codes
Indirect Address

- There are three **Addressing Modes** used for address portion of the instruction code:
  - Immediate: the operand is given in the address portion (constant)
  - Direct: the address points to the operand stored in the memory
  - Indirect: the address points to the pointer (another address) stored in the memory that references the operand in memory
- One bit of the instruction code can be used to distinguish between direct & indirect addresses

Instruction Codes
Indirect Address cont.



Instruction Codes
Indirect Address cont.

- Effective address: the address of the operand in a computation-type instruction or the target address in a branch-type instruction

- The memory word that holds the address of the operand in an indirect address instruction is used as a pointer to an array of data.

- The pointer can be placed in a processor register instead of memory as done in commercial computers

Computer Registers

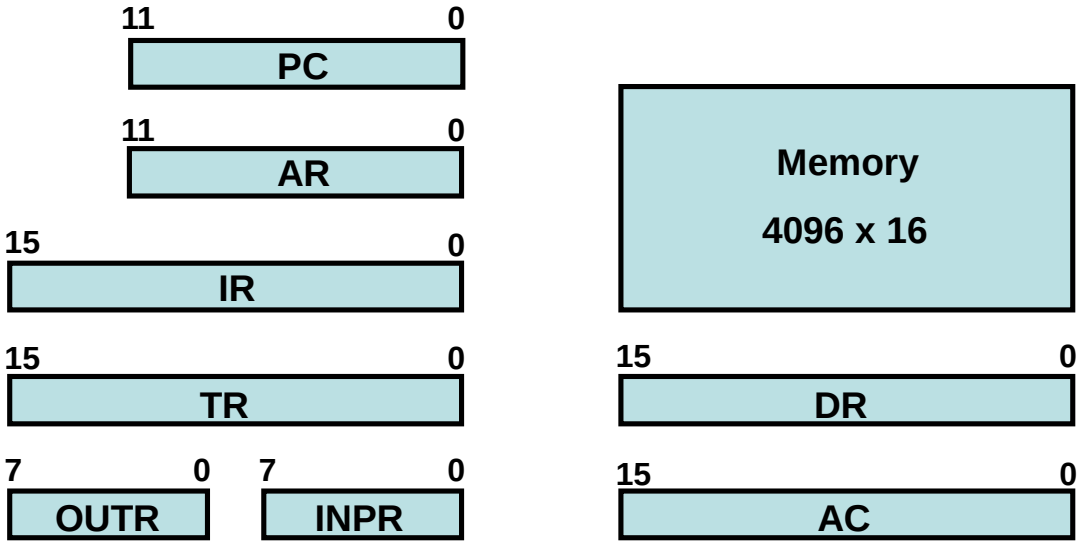
- Computer instructions are normally stored in consecutive memory locations and executed sequentially one at a time
- The control reads an instruction from a specific address in memory and executes it, and so on
- This type of sequencing needs a counter to calculate the address of the next instruction after execution of the current instruction is completed

Computer Registers cont.

- It is also necessary to provide a register in the control unit for storing the instruction code after it is read from memory

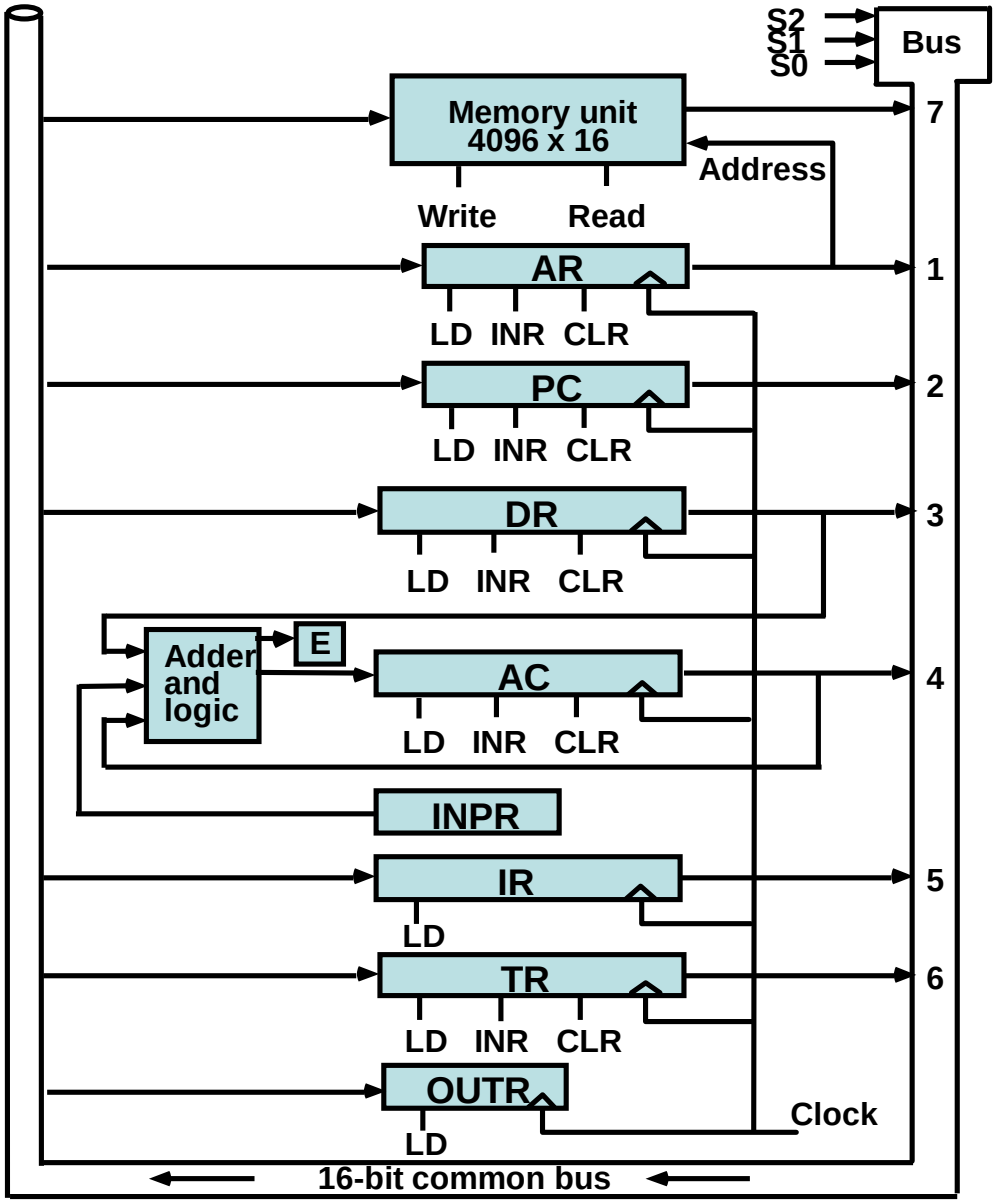
- The computer needs processor registers for manipulating data and a register for holding a memory address

Registers in the Basic Computer



Basic Computer Registers and memory

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



Computer Registers
Common Bus System

Computer Registers Common Bus System cont.

- $S_2S_1S_0$: Selects the register/memory that would use the bus
- LD (load): When enabled, the particular register receives the data from the bus during the next clock pulse transition
- E (extended AC bit): flip-flop holds the carry
- DR, AC, IR, and TR: have 16 bits each
- AR and PC: have 12 bits each since they hold a memory address

Computer Registers
Common Bus System cont.

- When the contents of AR or PC are applied to the 16-bit common bus, the four most significant bits are set to zeros
- When AR or PC receives information from the bus, only the 12 least significant bits are transferred into the register
- INPR and OUTR: communicate with the eight least significant bits in the bus

Computer Registers Common Bus System cont.

- INPR: Receives a character from the input device (keyboard,…etc) which is then transferred to AC
- OUTR: Receives a character from AC and delivers it to an output device (say a Monitor)
- Five registers have three control inputs: LD (load), INR (increment), and CLR (clear)
- Register ≡ binary counter with parallel load and synchronous clear

Computer Registers
Memory Address

- The input data and output data of the memory are connected to the common bus
- But the memory address is connected to AR
- Therefore, AR must always be used to specify a memory address
- By using a single register for the address, we eliminate the need for an address bus that would have been needed otherwise

Computer Registers
Memory Address cont.

- Register → Memory: Write operation
- Memory → Register: Read operation (note that AC cannot directly read from memory!!)
- Note that the content of any register can be applied onto the bus and an operation can be performed in the adder and logic circuit during the same clock cycle

Computer Registers
Memory Address cont.

- The transition at the end of the cycle transfers the content of the bus into the destination register, and the output of the adder and logic circuit into the AC
- For example, the two microoperations

  DR←AC and AC←DR (Exchange)

  can be executed at the same time

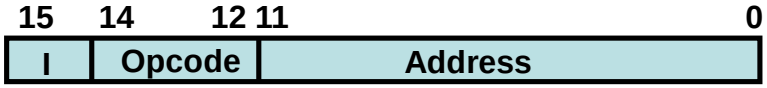
Computer Registers
Memory Address cont.

- This is done by:
- 1- place the contents of AC on the bus ($S_2S_1S_0$=100)
- 2- enabling the LD (load) input of DR
- 3- Transferring the contents of the DR through the adder and logic circuit into AC
- 4- enabling the LD (load) input of AC
- All during the same clock cycle
- The two transfers occur upon the arrival of the clock pulse transition at the end of the clock cycle.

Computer Instructions

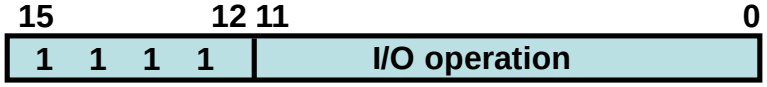
Basic Computer Instruction code format

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

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

Computer Instructions
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:
  - Arithmetic, logical, and shift instructions
  - Instructions for moving information to and from memory and processor registers
  - Program control instructions together with instructions that check status conditions
  - Input & output instructions

Instruction Cycle

- A program is a sequence of instructions stored in memory
- The program is executed in the computer by going through a cycle for each instruction (in most cases)
- Each instruction in turn is subdivided into a sequence of sub-cycles or phases

Instruction Cycle cont.

- Instruction Cycle 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

- This cycle repeats indefinitely unless a HALT instruction is encountered

Instruction Cycle
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_0

- 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

Instruction Cycle
Fetch and Decode cont.

– T_0: AR←PC (this is essential!!)

The address of the instruction is moved to AR.

– T_1: IR←M[AR], PC←PC+1

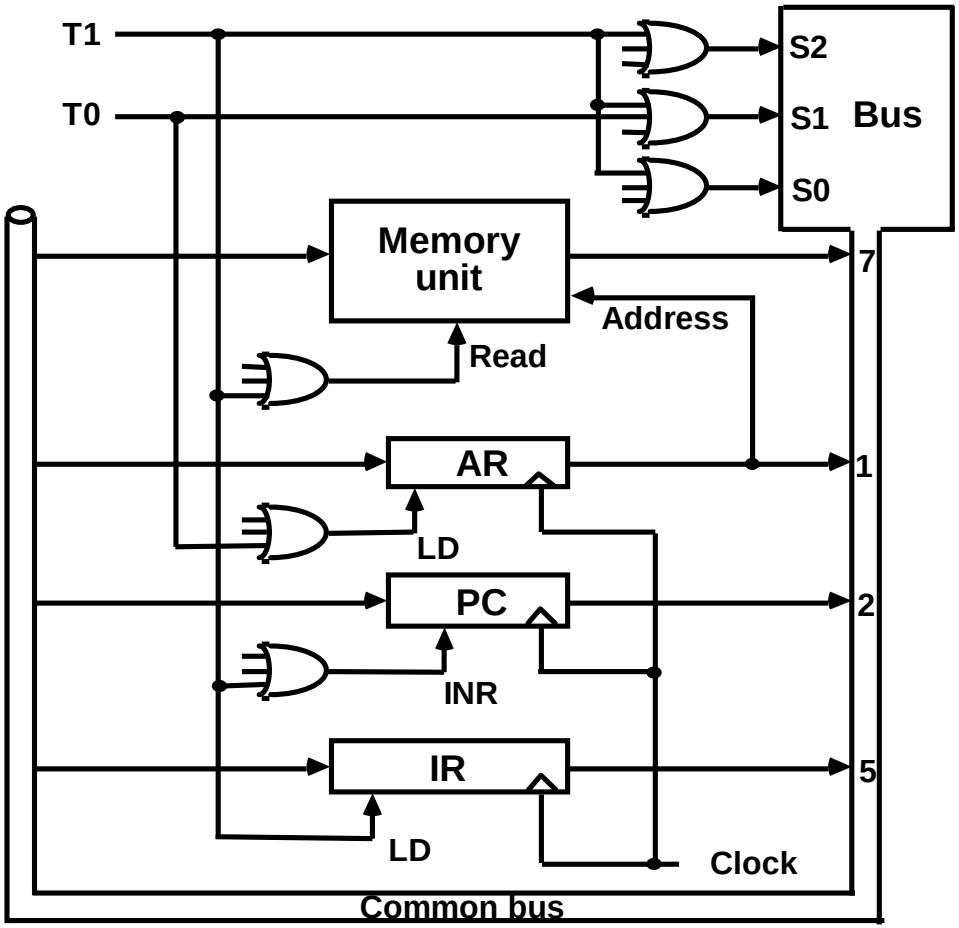
The instruction is fetched from the memory to IR, and the PC is incremented.

– T_2: D_0,…, D_7←Decode IR(12-14), AR←IR(0-11), I←IR(15)

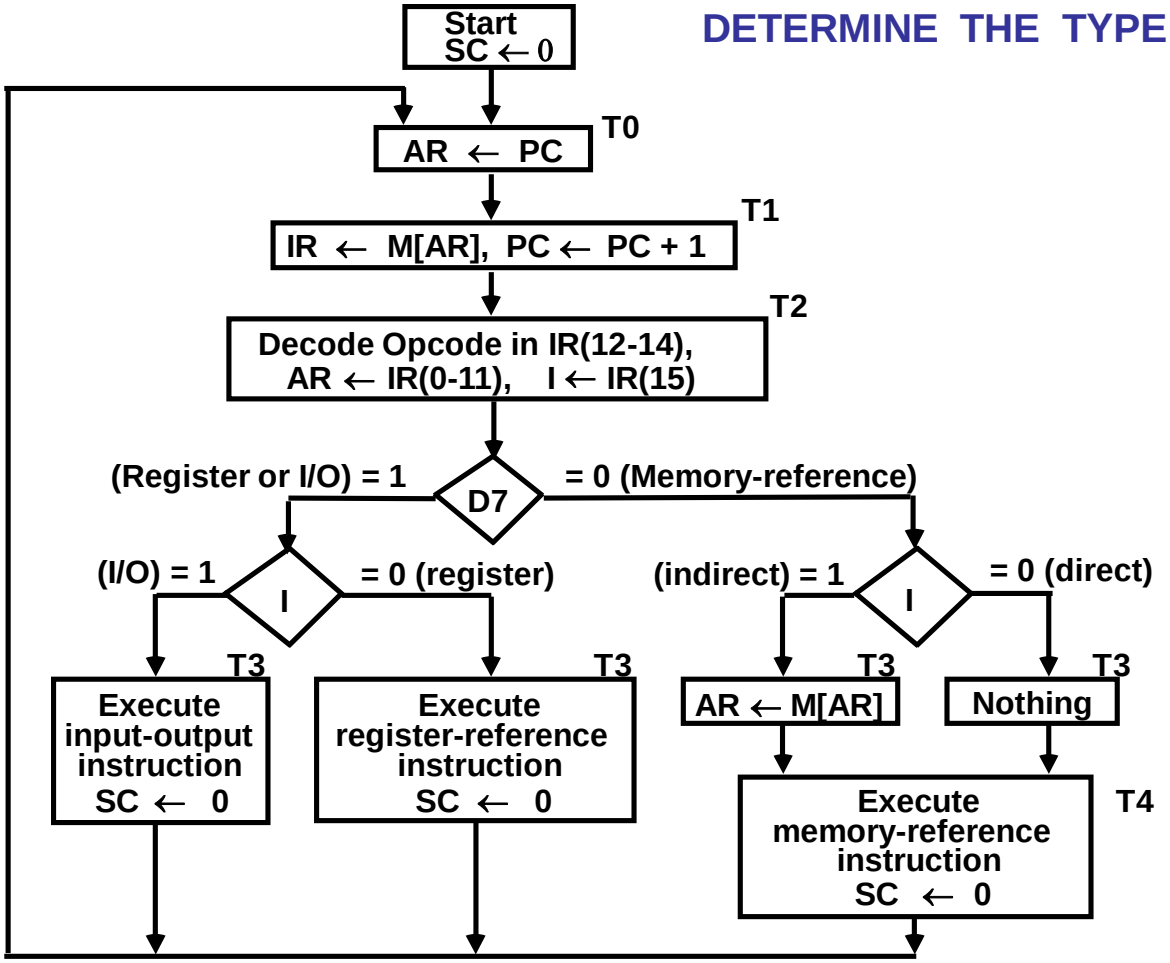
Instruction cycle: [Fetch Decode [Indirect] Execute]*

- **Fetch and Decode**

T0: AR ← PC (S0S1S2=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)



DETERMINE THE TYPE OF INSTRUCTION



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

REGISTER REFERENCE INSTRUCTIONS

Register Reference Instructions are identified when

- **$D_7 = 1$, $I = 0$**
- **Register Ref. Instr. is specified in B_0 ~ B_{11} of IR**
- **Execution starts with timing signal T_3**

$r = D_7\, I'\, T_3$ => Register Reference Instruction
B_i = IR(i) , i=0,1,2,...,11, the ith bit of IR.

	r:	SC ← 0
CLA	rB_{11}:	AC ← 0
CLE	rB_{10}:	E ← 0
CMA	rB_9:	AC ← AC'
CME	rB_8:	E ← E'
CIR	rB_7:	AC ← shr AC, AC(15) ← E, E ← AC(0)
CIL	rB_6:	AC ← shl AC, AC(0) ← E, E ← AC(15)
INC	rB_5:	AC ← AC + 1
SPA	rB_4:	if (AC(15) = 0) then (PC ← PC+1)
SNA	rB_3:	if (AC(15) = 1) then (PC ← PC+1)
SZA	rB_2:	if (AC = 0) then (PC ← PC+1)
SZE	rB_1:	if (E = 0) then (PC ← PC+1)
HLT	rB_0:	S ← 0 (S is a start-stop flip-flop)

MEMORY REFERENCE INSTRUCTIONS

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 $PC \leftarrow PC+1$

- 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 T3 when I = 1
- Memory cycle is assumed to be short enough to be completed in a CPU cycle
- The execution of MR Instruction starts with T_4

AND to AC

D_0T_4: $DR \leftarrow M[AR]$ Read operand

D_0T_5: $AC \leftarrow AC \wedge DR, SC \leftarrow 0$ AND with AC

ADD to AC

D_1T_4: $DR \leftarrow M[AR]$ Read operand

D_1T_5: $AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$ Add to AC and store carry in E

MEMORY REFERENCE INSTRUCTIONS cont.

LDA: Load to AC

D_2T_4: DR ← M[AR]

D_2T_5: AC ← DR, SC ← 0

STA: Store AC

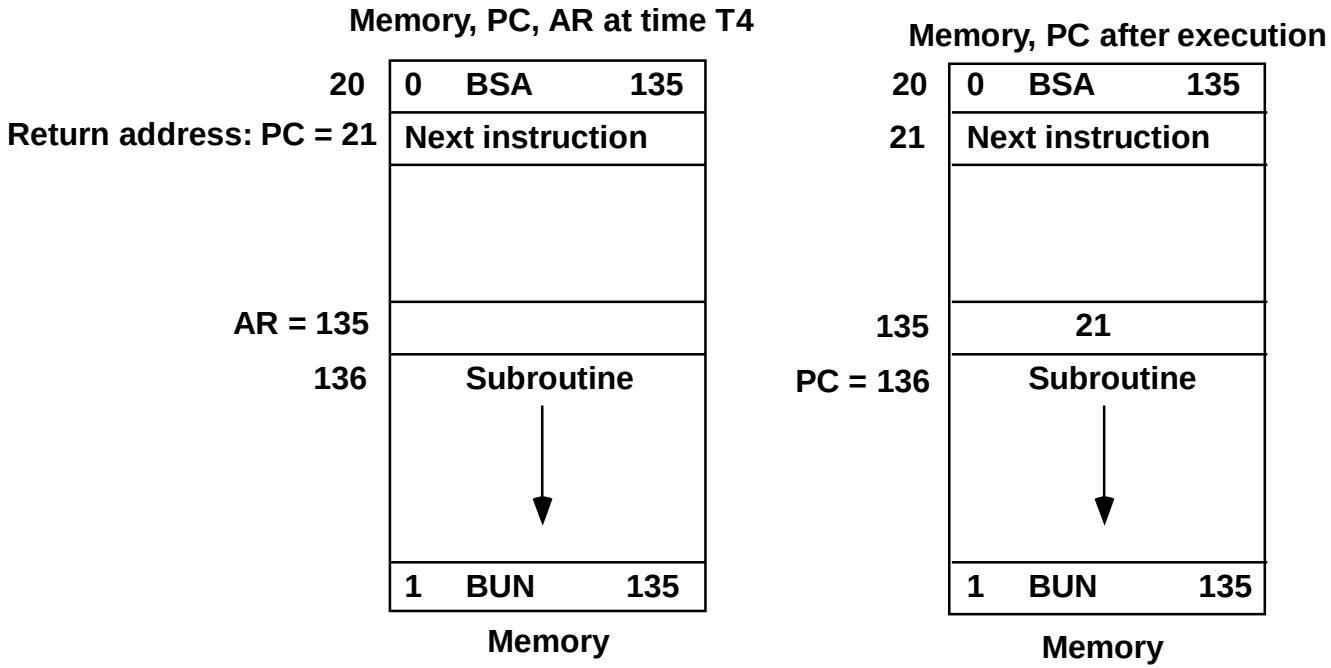
D_3T_4: M[AR] ← AC, SC ← 0

BUN: Branch Unconditionally

D_4T_4: PC ← AR, SC ← 0

BSA: Branch and Save Return Address

M[AR] ← PC, PC ← AR + 1



Memory Reference Instructions$^{cont.}$

BSA: executed in a sequence of two micro-operations:

D_5T_4: M[AR] ← PC, AR ← AR + 1

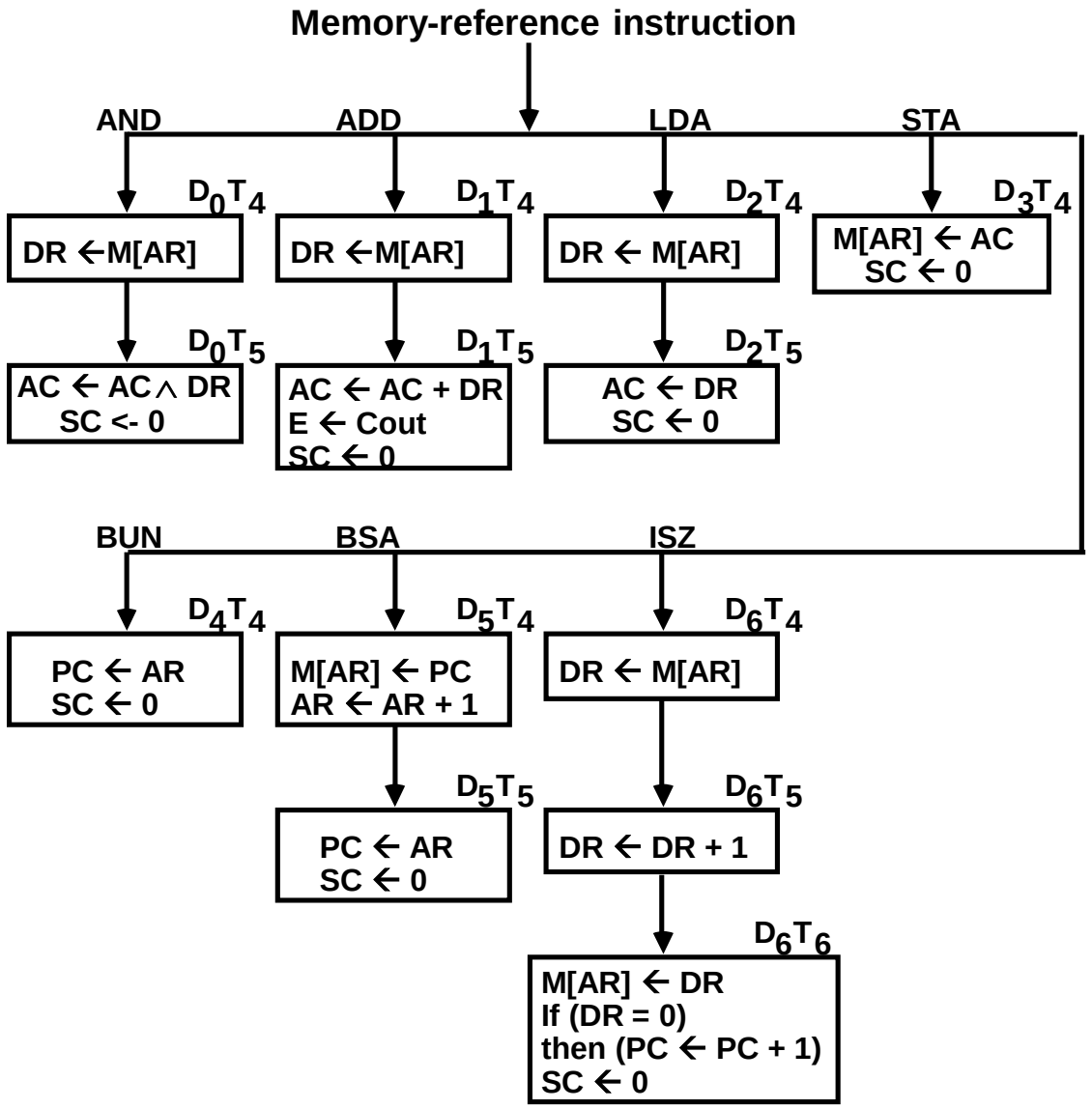
D_5T_5: PC ← AR, SC ← 0

ISZ: Increment and Skip-if-Zero

D_6T_4: DR ← M[AR]

D_6T_5: DR ← DR + 1

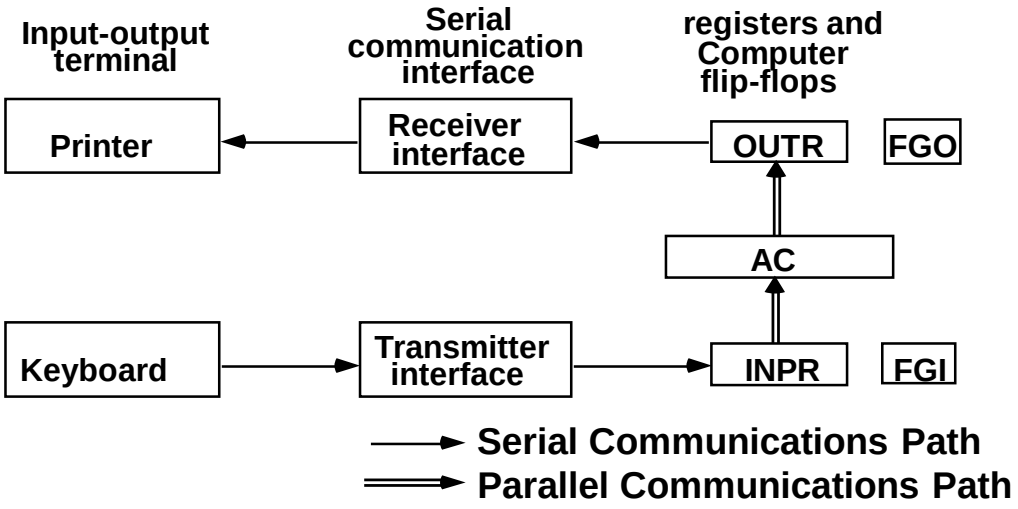
D_6T6: M[AR] ← DR, if (DR = 0) then (PC ← PC + 1), SC ← 0


Memory-reference instruction
AND
ADD
LDA
STA
D_0T_4
D_1T_4
D_2T_4
D_3T_4
DR ← M[AR]
DR ← M[AR]
DR ← M[AR]
M[AR] ← AC
SC ← 0
D_0T_5
D_1T_5
D_2T_5
AC ← AC ∧ DR
SC <- 0
AC ← AC + DR
E ← Cout
SC ← 0
AC ← DR
SC ← 0
BUN
BSA
ISZ
D_4T_4
D_5T_4
D_6T_4
PC ← AR
SC ← 0
M[AR] ← PC
AR ← AR + 1
DR ← M[AR]
D_5T_5
D_6T_5
PC ← AR
SC ← 0
DR ← DR + 1
D_6T_6
M[AR] ← DR
If (DR = 0)
then (PC ← PC + 1)
SC ← 0

Input-Output Instructions

- Instructions and data stored in memory must come from some input device
- Computational results must be transmitted to the user through some output device
- For the system to communicate with an input device, serial information is shifted into the input register INPR
- To output information, it is stored in the output register OUTR

Input-Output Instructions cont.



Input-Output Instructions$^{cont.}$

- INPR and OUTR communicate with a communication interface serially and with the AC in parallel. They hold an 8-bit alphanumeric information
- I/O devices are slower than a computer system → we need to synchronize the timing rate difference between the input/output device and the computer.
- FGI: 1-bit input flag (Flip-Flop) aimed to control the input operation

Input-Output Instructions cont.

- FGI is set to 1 when a new information is available in the input device and is cleared to 0 when the information is accepted by the computer
- FGO: 1-bit output flag used as a control flip-flop to control the output operation
- If FGO is set to 1, then this means that the computer can send out the information from AC. If it is 0, then the output device is busy and the computer has to wait!

Input-Output Instructions$^{cont.}$

- The process of input information transfer:
  - Initially, FGI is cleared to 0
  - An 8-bit alphanumeric code is shifted into INPR (Keyboard key strike) and the input flag FGI is set to 1
  - As long as the flag is set, the information in INPR cannot be changed by another data entry
  - The computer checks the flag bit; if it is 1, the information from INPR is transferred in parallel into AC and FGI is cleared to 0

Input-Output Instructions$^{cont.}$

  - Once the flag is cleared, new information can be shifted into INPR by the input device (striking another key)
- The process of outputting information:
  - 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 accepts the coded information (prints the corresponding character)

Input-Output Instructions$^{cont.}$

- When the operation is completed, the output device sets FGO back to 1
- The computer does not load a new data information into OUTR when FGO is 0 because this condition indicates that the output device is busy to receive another information at the moment!!

Input-Output Instructions

- Needed for:
  - Transferring information to and from AC register
  - Checking the flag bits
  - Controlling the interrupt facility
- The control unit recognize it when D_7=1 and I = 1
- The remaining bits of the instruction specify the particular operation
- Executed with the clock transition associated with timing signal T_3
- Input-Output instructions are summarized next

Input-Output Instructions

$D_7IT_3 = p$
$IR(i) = B_i$, i = 6, …, 11

INP	pB_{11}:	AC(0-7) ← INPR, FGI ← 0	Input char. to AC
OUT	pB_{10}:	OUTR ← AC(0-7), FGO ← 0	Output char. from AC
SKI	pB_9:	if(FGI = 1) then (PC ← PC + 1)	Skip on input flag
SKO	pB_8:	if(FGO = 1) then (PC ← PC + 1)	Skip on output flag
ION	pB_7:	IEN ← 1	Interrupt enable on
IOF	pB_6:	IEN ← 0	Interrupt enable off