Computer Organization

.

A stored program computer behaves as different machines by loading different programs, i.e., sequences of instructions.

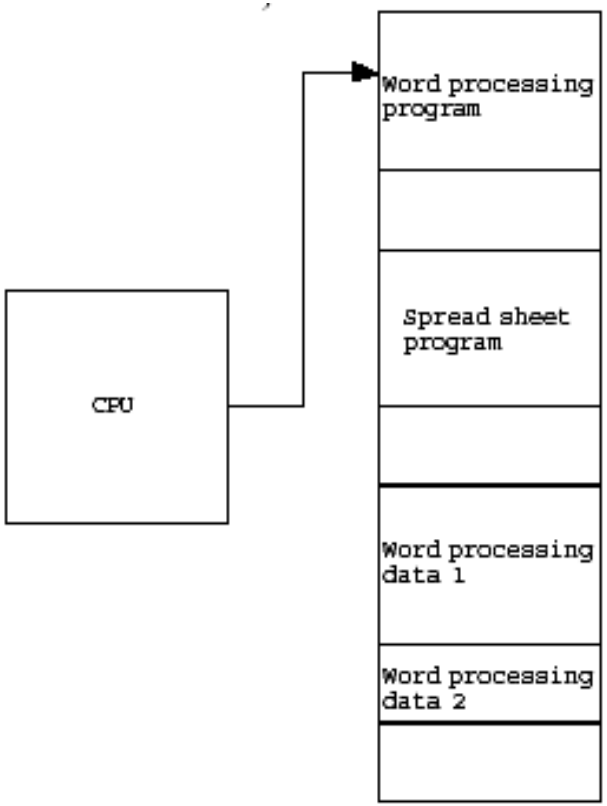


Figure: Stored program computer concept.

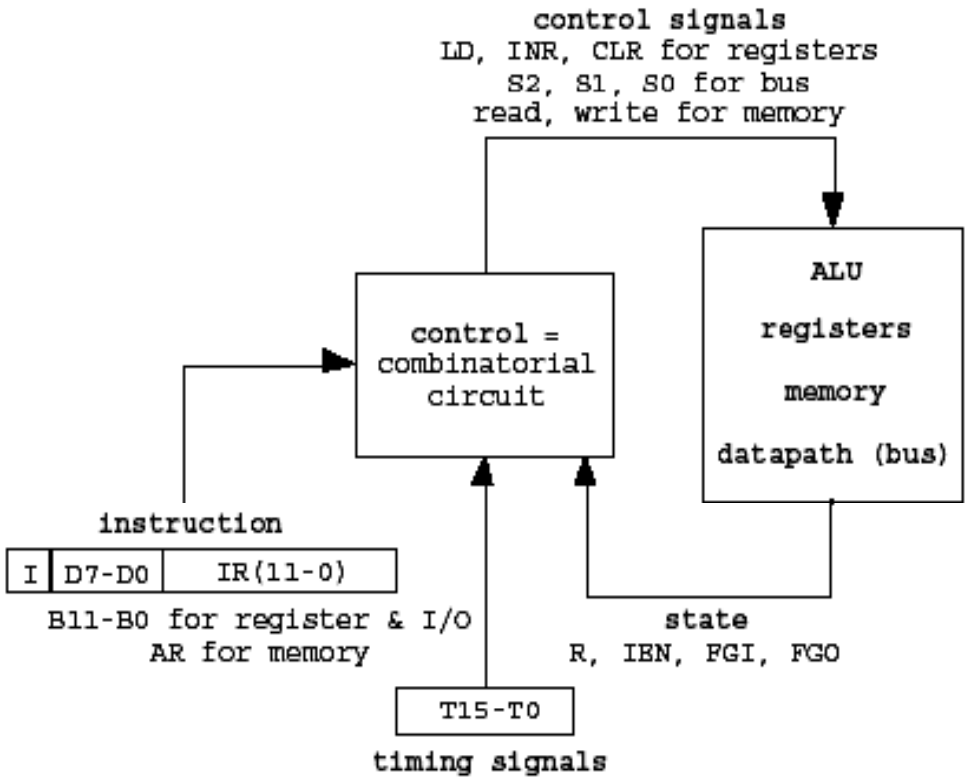


Figure: Stored program computer hardware.

• Computer hardware = registers + ALU + datapath (bus) + control unit.

• The computer goes through **instruction cycles**:

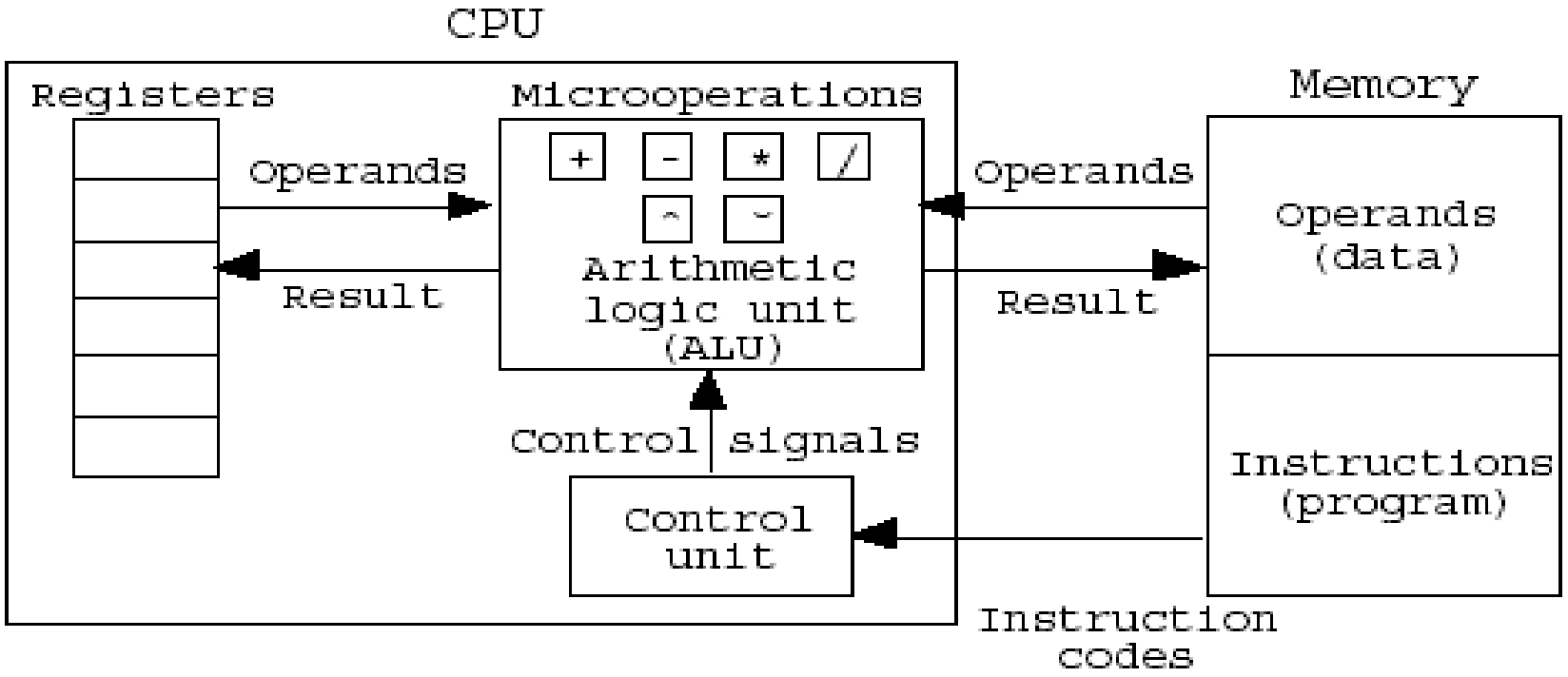
i) Fetch an instruction from memory;

ii) Decode the instruction to a sequence of control signals;

iii) Execute the decoded sequence of microoperations.

• **Control unit**: Instruction → a time sequence of control signals to trigger microoperations.

• Input-output is implemented using an **interrupt cycle**.



Instruction Codes

- Stored Program Organization :
  - The simplest way to organize a computer
    - One processor register : AC(Accumulator)
      - » The operation is performed with the memory operand and the content of AC
    - Instruction code format with two parts : Op. Code + Address
      - » Op. Code : specify 16 possible operations(4 bit)
      - » Address : specify the address of an operand(12 bit)
      - » If an operation in an instruction code does not need an operand from memory, the rest of the bits in the instruction(***address field***) can be used for other purpose
    - Memory : 12 bit = 4096 word(Instruction and Data are stored)
      - » Store each instruction code(***program***) and operand (***data***) in 16-bit memory word

–Addressing Mode

- •Immediate operand address :
  - –the second part of instruction code(***address field***) specifies ***operand***
- •Direct operand address :
  - –the second part of instruction code specifies the ***address of operand***
- •Indirect operand address :
  - –the bits in the second part of the instruction designate an ***address of a memory word in which the address of the operand is found*** (Pointer)
- •One bit of the instruction code is used to distinguish between a direct and an indirect address :

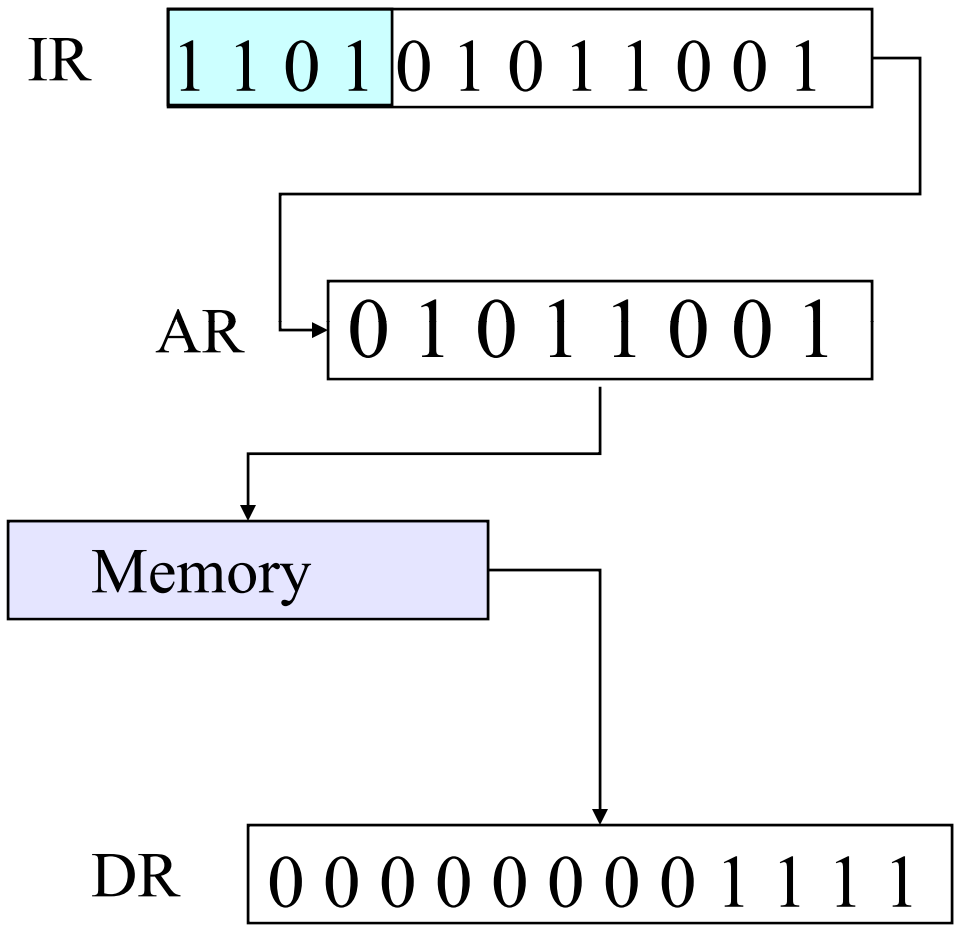
- **Effective address**: Address where an operand is physically located

Direct Addressing

Occurs When the Operand Part Contains the Address of Needed Data.

1. Address part of IR is placed on the bus and loaded back into the AR

2. Address is selected in memory and its Data placed on the bus to be loaded into the Data Register to be used for requested instructions



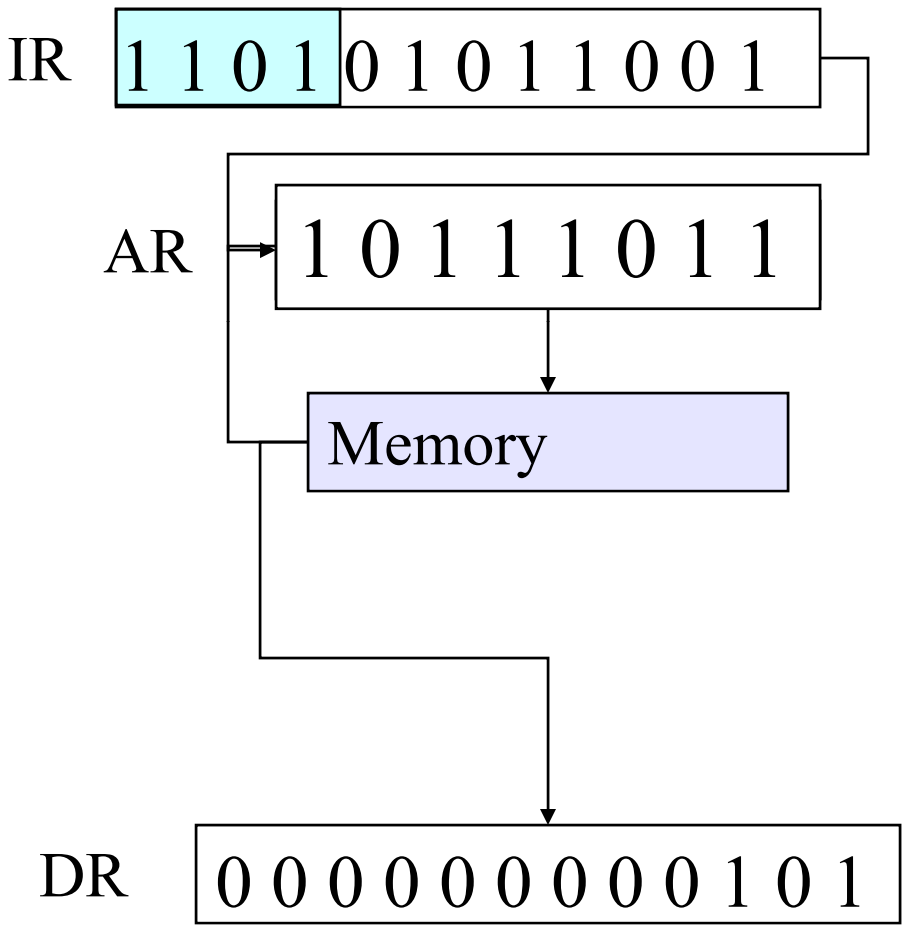
Indirect Addressing

Occurs When the Operand Contains the Address of the Address of Needed Data.

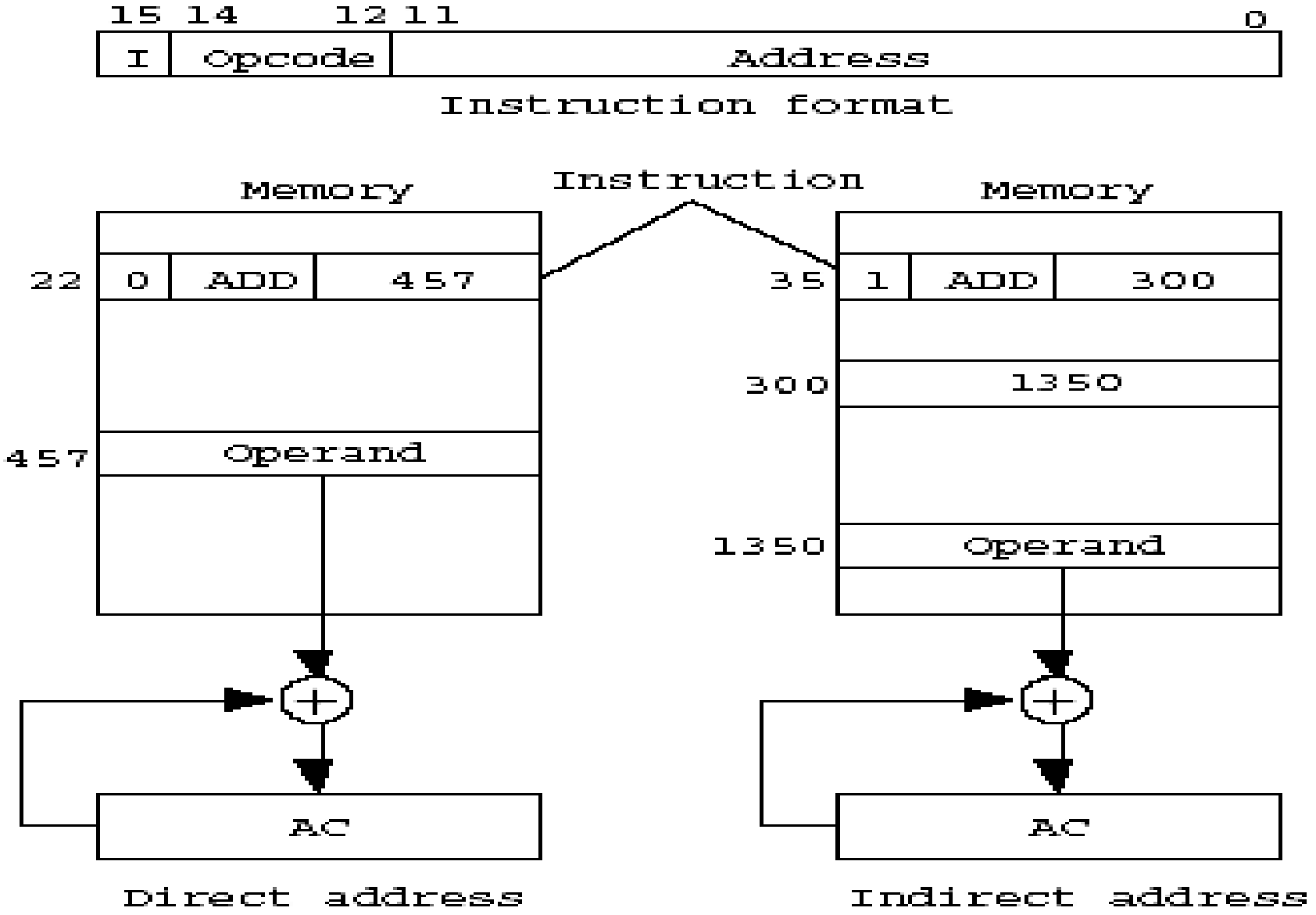
1. Address part of IR is placed on the bus and loaded back into the AR

2. Address is selected in memory and placed on the bus to be loaded Back into the AR

3. New Address is selected in memory and placed on the bus to be loaded into the DR to use later



Direct and Indirect addressing example



Computer Registers

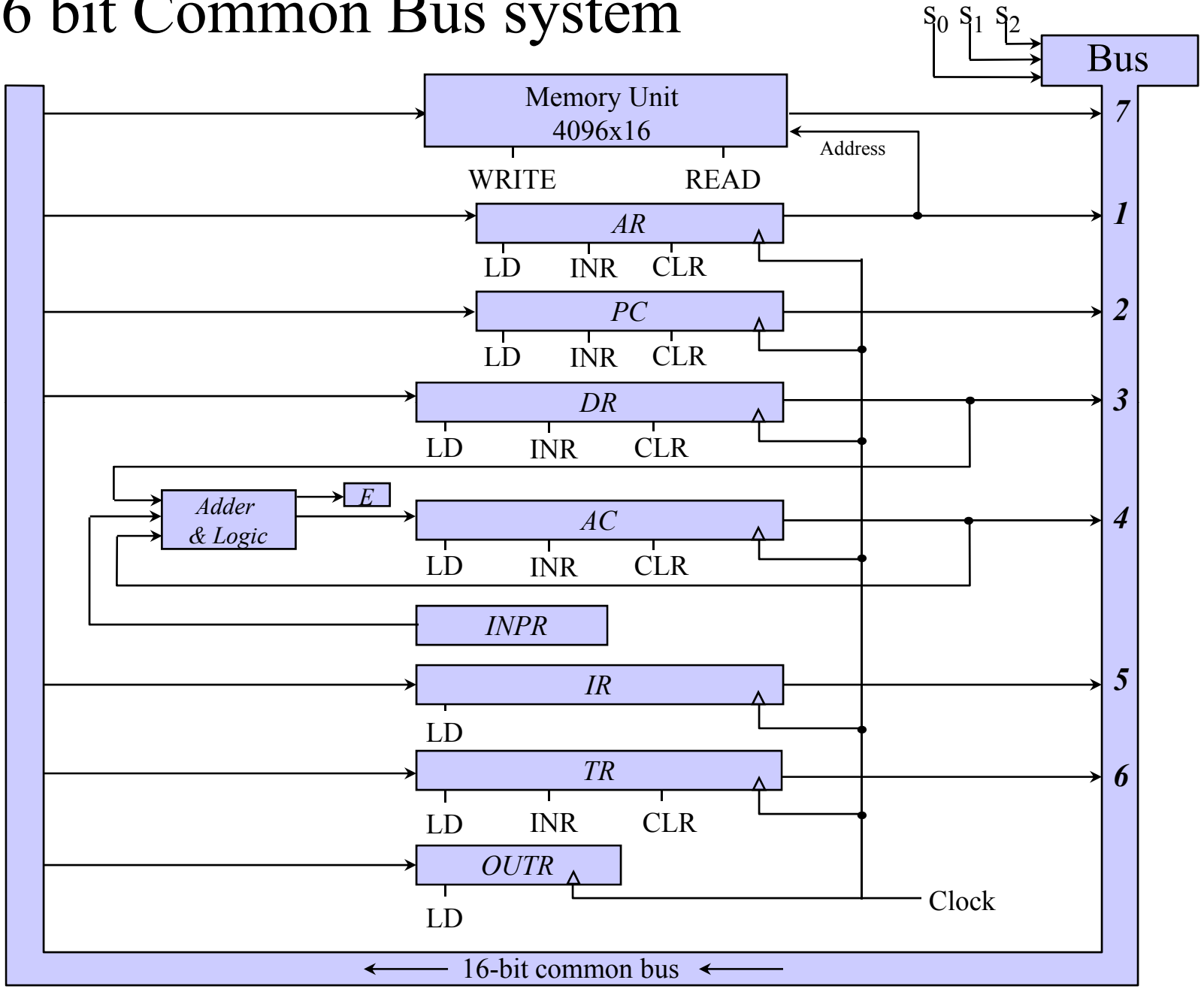
- Data Register(***DR***) : hold the operand(Data) read from memory
  - Accumulator Register(***AC***) : general purpose processing register
  - Instruction Register(***IR***) : hold the instruction read from memory
  - Temporary Register(***TR***) : hold a temporary data during processing
  - Address Register(***AR***) : hold a memory address, 12 bit width
  - Program Counter(***PC***) :
    - »hold the address of the next instruction to be read from memory after the current instruction is executed
    - »Instruction words are read and executed in sequence unless a branch instruction is encountered
    - »A branch instruction calls for a transfer to a nonconsecutive instruction in the program
    - »The address part of a branch instruction is transferred to PC to become the address of the next instruction
    - »To read instruction, memory read cycle is initiated, and PC is incremented by one(next instruction fetch)

- Input Register(***INPR***) : receive an 8-bit character from an input device
- Output Register(***OUTR***) : hold an 8-bit character for an output device

The following registers are used in Mano's example computer.

Register symbol	Number of bits	Register name	Register Function
DR	16	Data register	Holds memory operands
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

16 bit Common Bus system



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
- A more efficient scheme for transferring information in a system with many registers is to use a common bus.
- The connection of the registers and memory of the basic computer to a common bus system.
  - The outputs of seven registers and memory are connected to the common bus
  - The specific output is selected by mux(S0, S1, S2) :
    - Memory(7), AR(1), PC(2), DR(3), AC(4), IR(5), TR(6)
    - When LD(Load Input) is enable, the particular register receives the data from the bus
  - Control Input : LD, INC, CLR, Write, Read

COMMON BUS SYSTEM

• **Control variables:** Various control variables are used to select:

i) the paths of information; &

ii) the operation of the registers.

- **Selection variables**: Used to specify a register whose output is connected to the common bus at any given time.
- To select one register out of 8, we need 3 select variables.
- For example, if S2S1S0 = 011, the output of DR is directed to the common bus.

> **Load input (LD)**: Enables the input of a register connected to the common bus. When LD = 1 for a register, the data on the common bus is read into the register during the next clock pulse transition.

> **Increment input (INR)**: Increments the content of a register.

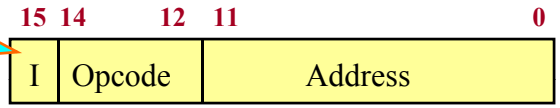
> **Clear input (CLR)**: Clear the content of a register to zero.

• When the contents of AR or PC (12 bits) are applied to the 16-bit common bus, the four most significant bits are set to zero. When AR or PC receives information from the bus, only the 12 least significant bits are transferred to the register. Both INPR and OUTR use only the 8 least significant bits of the bus.

5-3. Computer Instruction

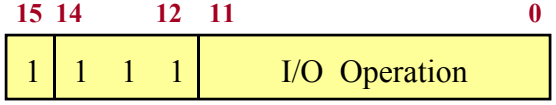
- 3 Instruction Code Formats :
  - Memory-reference instruction
    - –Opcode = 000 ~ 110
      - »I=0 : 0xxx ~ 6xxx, I=1: 8xxx ~Exxx

I=0 : Direct,
I=1 : Indirect



      - »Register-reference instruction
    - –7xxx (7800 ~ 7001) : CLA, CMA,

    - –Input-Output instruction
    - –Fxxx(F800 ~ F040) : INP, OUT, ION, SKI,



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 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
CMS	7200		Complement AC
CME	m 7100	e	Comp
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		Interrup
IOF	F040		Inter

Reading this table:
the presented code is for any instruction that has 16 bits. The xxx represents don't care (any data for the first 12 bits). Example 7002 for is a hexadecimal code equivalent to 0111 0000 0000 0010 Which means B_1 (Bit 1) is set to 1 and the rest of the first 12 bits are set to zeros.

•• Instructions are normally stored in consecutive memory locations and are executed sequentially one at a time.

•The **program counter** (PC) holds the address of the next instruction to be read from memory after the current instruction is executed.

•The PC has 12 bits like the AR.

•The instruction read from memory is placed in the **instruction register** (IR) which has 16 bits corresponding to our instruction code length.

• Most processing takes place in the **accumulator** (AC);

•the **temporary register** (TR) is used for holding temporary data during the processing.

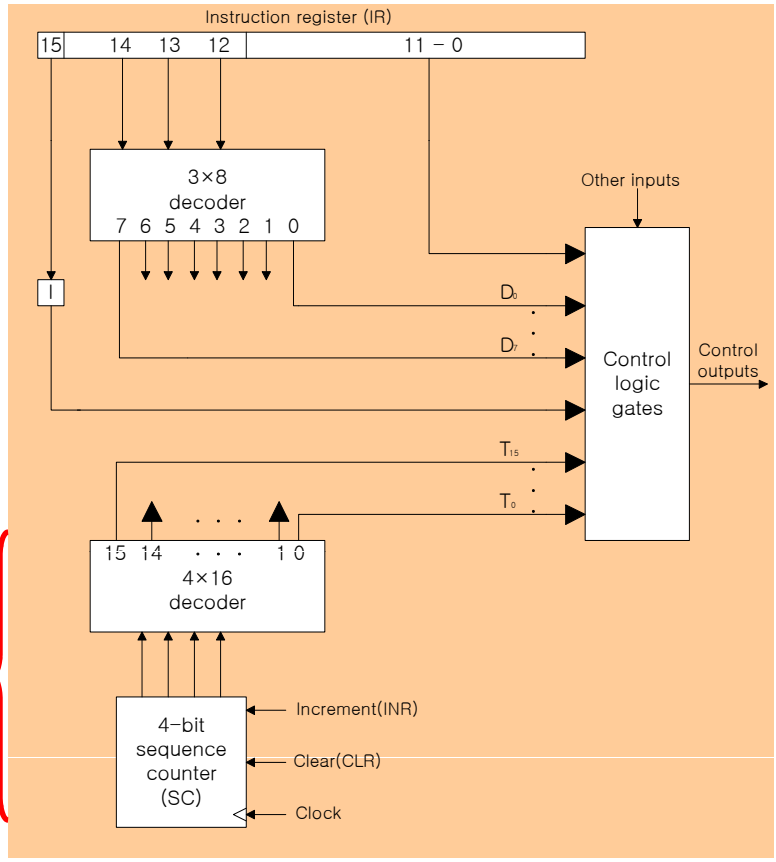
• The **input** (INPR) and **output** (OUTR) **registers** hold a character at a time which is read from an input device or to be printed to an output device, respectively. Using the ASCII code, one character is represented with 8 bits (1 byte).

Timing and Control

- Microprogrammed Control :
  - The control information is stored in a control memory, and the control memory is programmed to initiate the required sequence of microoperations
  - + Any required change can be done by updating the microprogram in control memory, - Slow operation

– Control Unit

- Control Unit = Control Logic Gate + 3 X 8 Decoder + Instruction Register + Timing Signal
- Timing Signal = 4 X 16 Decoder + 4-bit Sequence Counter
- Example) Control timing
  - Sequence Counter is cleared when D_3T_4 =1 : $D_3T_4 : SC \leftarrow 0$
- Memory R/W cycle time > Clock cycle time



CONTROL UNIT HARDWARE

• Inputs to the control unit come from IR where an instruction read from the memory unit is stored.

• A hardwired control is implemented in the example computer using:

> A 3 ´ 8 decoder to decode opcode bits 12-14 into signals D0, ..., D7;

> A 4-bit binary **sequence counter** (SC) to count from 0 to 15 to achieve time sequencing;

> A 4 ´ 16 decoder to decode the output of the counter into 16 timing signals, T0, ..., T15

> A flip-flop (I) to store the addressing mode bit in IR;

> A digital circuit with inputs—D0, ..., D7, T0, ..., T15, I, and address bits (11-0) in IR—to generate control outputs supplied to control inputs and select signals of the registers and the bus.

• **Clocking principle**: The binary counter goes through a cycle, 0000 → 0001 → 0010 → ... → 1111 → 0000. Accordingly only one of T0, ..., T15 is 1 at each clock cycle, T0 → T1 → T2 → ... → T15 → T0; all the other timing signals are 0.

• By setting the clear input (CLR) of SC at a clock cycle, say T3, we can achieve a 4-cycle clock: T0 → T1 → T2 → T3 → T0.

Instruction Cycle

• A computer goes through the following instruction cycle repeatedly:

do

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 until a HALT instruction is encountered

• The fetch & decode phases of the instruction cycle consists of the following microoperations synchronized with the timing signals (clocking principle).

Timing signal	microoperations
T0:	**AR ← PC**
T1:	**IR ← M[AR], PC ← PC + 1**
T2:	**D0, ..., D7 ← Decode IR(12-14), AR ← IR(0-11), I ← IR(15)**

T0: Since only AR is connected to the address inputs of memory, the address of instruction is transferred from PC to AR.

1. Place the content of PC onto the bus by making the bus selection inputs S2S1S0 = 010.
2. Transfer the content of the bus to AR by enabling the LD input to AR ($AR \leftarrow PC$).

T1: The instruction read from memory is then placed in the instruction register IR. At the same time, PC is incremented to prepare for the address of the next instruction.

1. Enable the read input of the memory.
2. Place the content of memory onto the bus by making the bus selection inputs S2S1S0 = 111. (Note that the address lines are always connected to AR, and we have already placed the next instruction address in AR.)
3. Transfer the content of the bus to IR by enabling the LD input to IR ($IR \leftarrow M[AR]$).
4. Increment PC by enabling the INR input of PC ($PC \leftarrow PC + 1$).

T2: The operation code in IR is decoded; the indirect bit is transferred to I; the address part of the instruction is transferred to AR. (See the common bus skeleton diagram.)

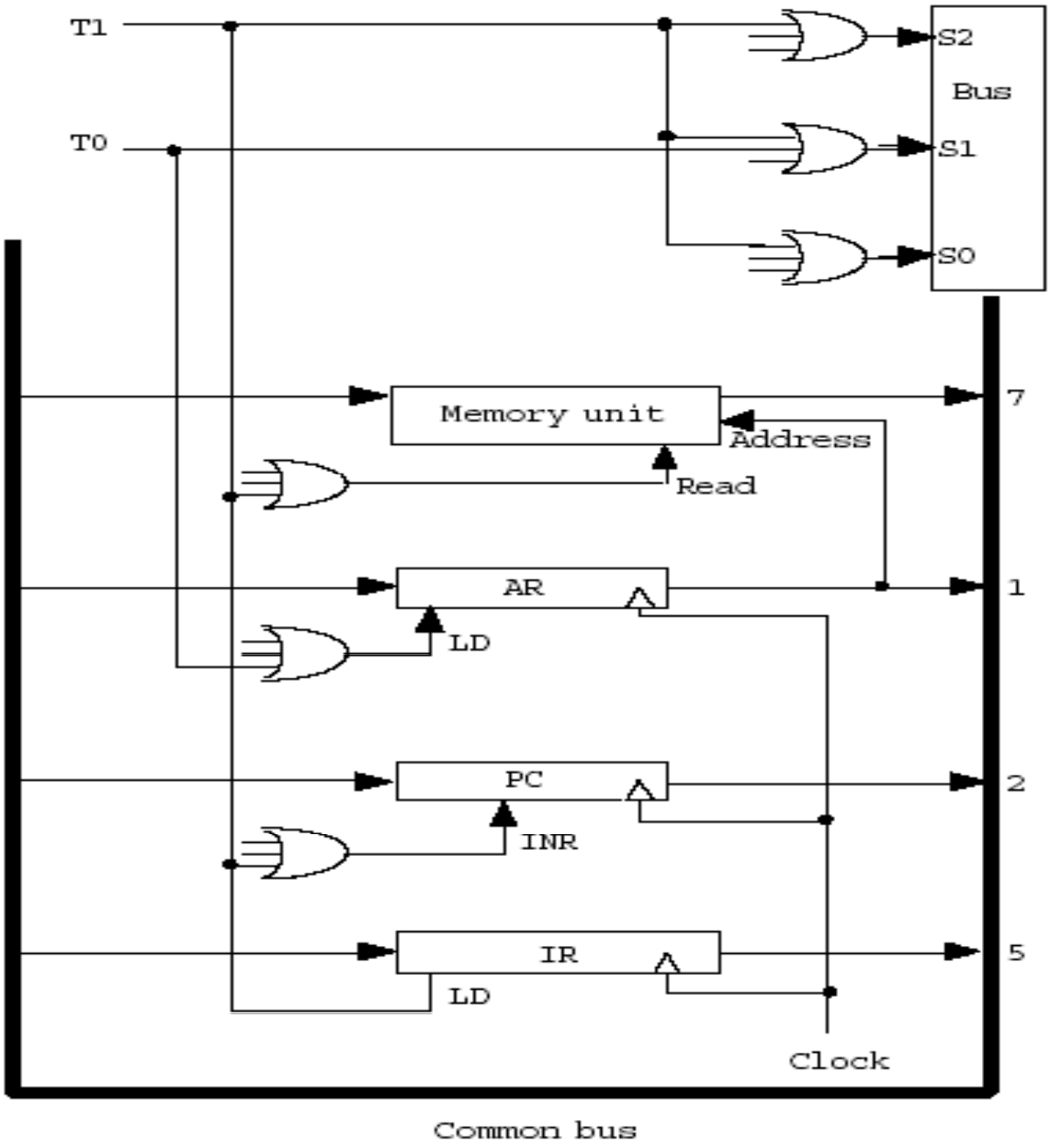
Similar circuits are used to realize the microoperations at T2.

• At T3, microoperations which take place depend on the type of instruction. The four different paths are symbolized as follows, where the control functions must be connected to the proper inputs to activate the desired microoperations.

Control function	Microoperation
D7'IT3:	**AR ← M[AR], indirect memory transfer**
D7'I'T3:	**Nothing, direct memory transfer**
D7I'T3:	**Execute a register-reference instruction**
D7IT3:	**Execute an I/O instruction**

When D7'T3 = 1 (At T3 & IR(12-14) ≠ 111), the execution of memory-reference instructions takes place with the next timing variable T4.

Figure: Control circuit for instruction fetch. This is a part of the control circuit and demonstrates the kind of wiring needed.



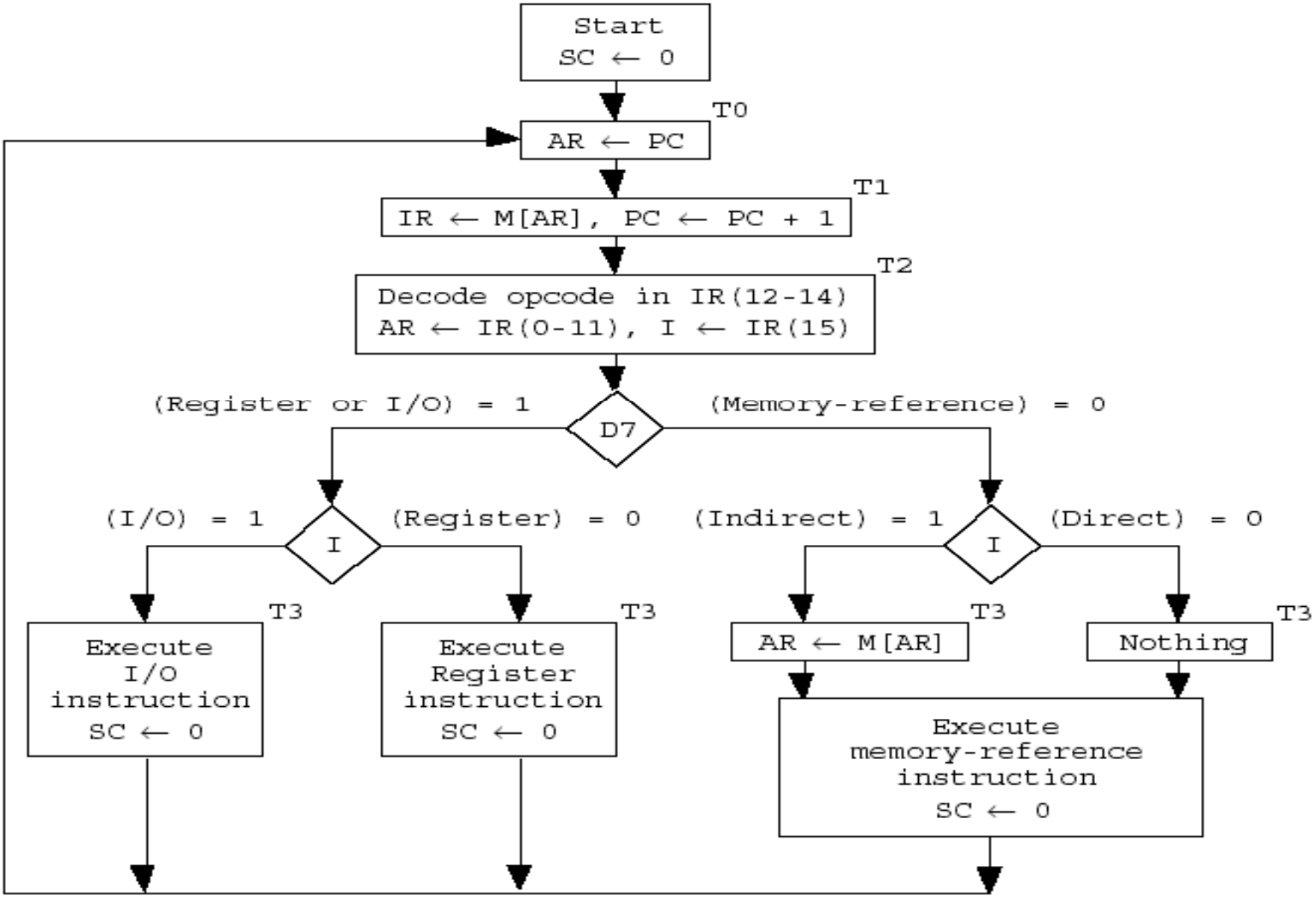


Figure: Flowchart for fetch & decode phases.

REGISTER-REFERENCE INSTRUCTIONS

• The 12 register-reference instructions are recognized by I = 0 and D7 = 1 (IR(12-14) = 111). Each operation is designated by the presence of 1 in one of the bits in IR(0-11). Therefore D7I'T3 ≡ r = 1 is common to all register-transfer instructions.

Symbol	Control	Microoperations	Description
	r	SC ← 0 (Common to all, done in 1 cycle)	Clear SC
CLA	rB_{11}	AC ← 0	Clear AC
CLE	rB_{10}	E ← 0	Clear E
CMA	rB_9	AC ← $\underline{AC}$	Complement AC
CME	rB_8	E ← $\underline{E}$	Complement E
CIR	rB_7	AC ← shr AC, AC(15) ← E, E ← AC(0)	Circular right
CIL	rB_6	AC ← shl AC, AC(0) ← E, E ← AC(15)	Circular left
INC	rB_5	AC ← AC + 1	Increment AC
SPA	rB_4	If AC(15)=0 then PC ← PC + 1	Skip if positive
SNA	rB_3	If AC(15)=1 then PC ← PC + 1	Skip if negative
SZA	rB_2	If AC=0 then PC ← PC + 1	Skip if AC zero
SZE	rB_1	If E=0 then PC ← PC + 1	Skip if E zero
HLT	rB_0	S ← 0 (S is a start-stop flip-flop)	Halt computer

Memory Reference Instructions

• Opcode (000 - 110) or the decoded output Di (i = 0, ..., 6) are used to select one memory-reference operation out of 7.

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

Memory Reference Instruction

- STA : memory write

$D_3T_4 : M[AR] \leftarrow AC, SC \leftarrow 0$

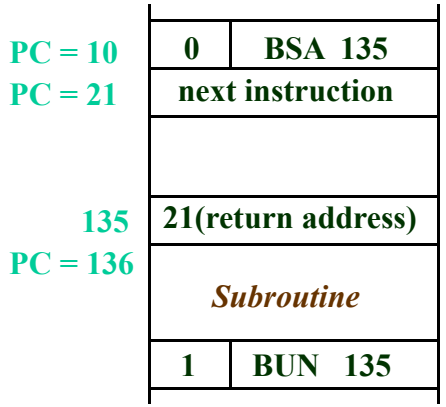
- BUN : branch unconditionally

$D_4T_4 : PC \leftarrow AR, SC \leftarrow 0$

- BSA : branch and save return address

$D_5T_4 : M[AR] \leftarrow PC, AR \leftarrow AR+1$

$D_5T_5 : PC \leftarrow AR, SC \leftarrow 0$



  - Return Address : save return address (135 ← 21)
  - Subroutine Call : *Fig. 5-10*

$D_5T_4 : M[135] \leftarrow 21(PC), 136(AR) \leftarrow 135+1$

$D_5T_5 : 136(PC) \leftarrow 136(AR), SC \leftarrow 0$

- ISZ : increment and skip if zero

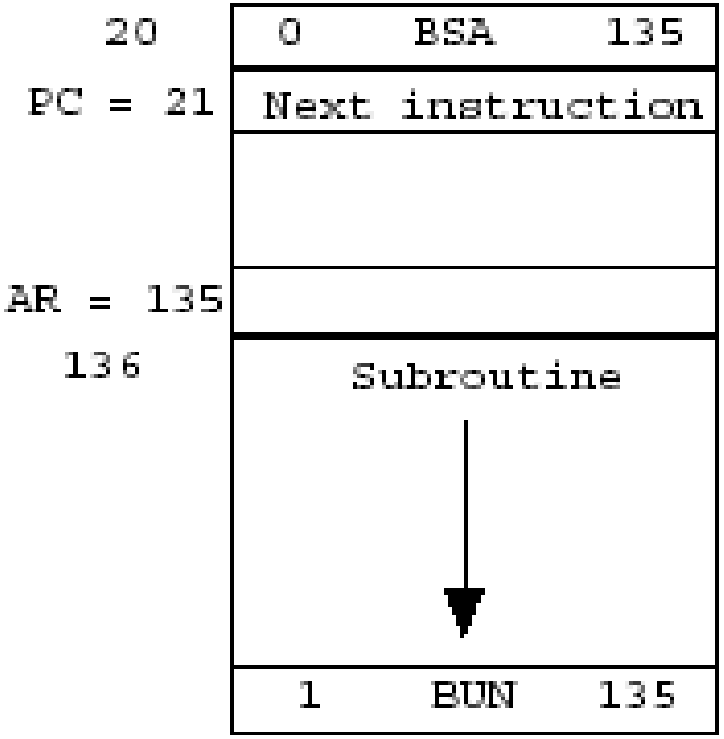
$D_6T_4 : DR \leftarrow M[AR]$

$D_6T_5 : DR \leftarrow DR+1$

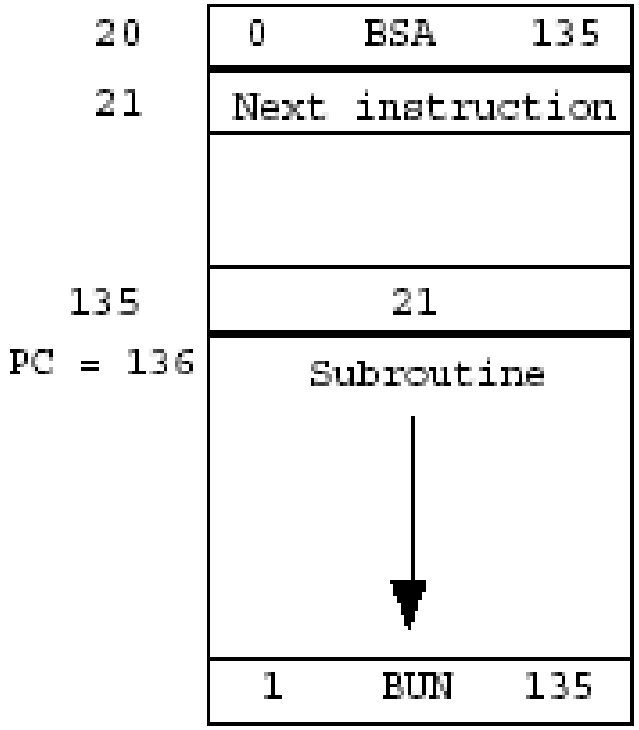
$D_6T_6 : M[AR] \leftarrow DR, if\ (DR=0)\ then\ (PC \leftarrow PC+1), SC \leftarrow 0$

- Control Flowchart :
  - Flowchart for the 7 memory reference instruction
    - The longest instruction : ISZ(T6)
    - 3 bit Sequence Counter

Branch and Save Address (BSA)



Memory, PC, and AR at time T_4



Memory and PC after BSA execution

Subroutine implementation using BSA.

Input-Output and Interrupt

- 5-7 Input-Output and Interrupt
  - Input-Output Configuration : ***Fig. 5-12***
    - Input Register(***INPR***), Output Register(***OUTR***)
      - These two registers communicate with a communication interface serially and with the AC in parallel
      - Each quantity of information has eight bits of an alphanumeric code
    - Input Flag(***FGI***), Output Flag(***FGO***)
      - FGI : ***set*** when INPR is ready, ***clear*** when INPR is empty
      - FGO : ***set*** when operation is completed, ***clear*** when output device is in the process of printing

1 : Ready
0 : Not ready

  - Input-Output Instruction : *Tab. 5-5*
    - $p = D_7IT_3$
    - $IR(i) = B_i$ ← IR(6 -11)

Address

    - ***B_6 - B_{11}*** : **6 I/O Instruction**
  - Program Interrupt
    - I/O Transfer Modes
      - 1) Programmed I/O, 2) Interrupt-initiated I/O, 3) DMA, 4) IOP
      - 2) Interrupt-initiated I/O (FGI FGO 1 Int.)
      - Maskable Interrupt (ION IOF Int. mask)

- Interrupt Cycle : *Fig. 5-13*
  - During the execute phase, IEN is checked by the control
    - » IEN = 0 : the programmer does not want to use the interrupt, so control continues with the next instruction cycle
    - » IEN = 1 : the control circuit checks the flag bit, If either flag set to 1, R (R is the interrupt flip flop) is set to 1
  - At the end of the execute phase, control checks the value of R
    - » R = 0 : instruction cycle
    - » R = 1 : Interrupt cycle
- Demonstration of the interrupt cycle : ***Fig. 5-14***
  - The memory location at address 0 as the place for storing the return address
  - Interrupt Branch to memory location 1
  - Interrupt cycle IEN=0 (*ISR Interrupt ION*)
- The condition for R = 1

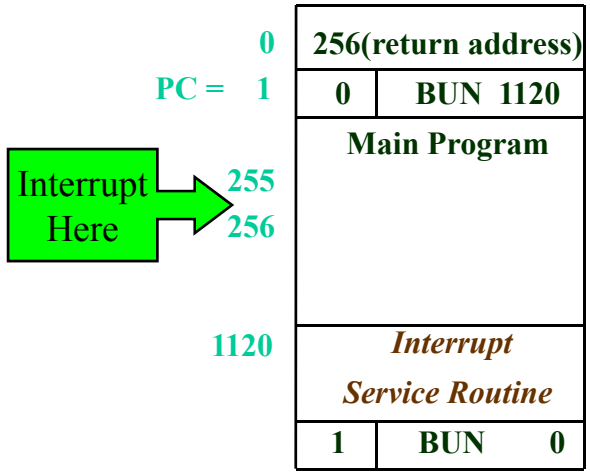
$$T_0' T_1' T_2' (IEN)(FGI + FGO) : R \leftarrow 1$$

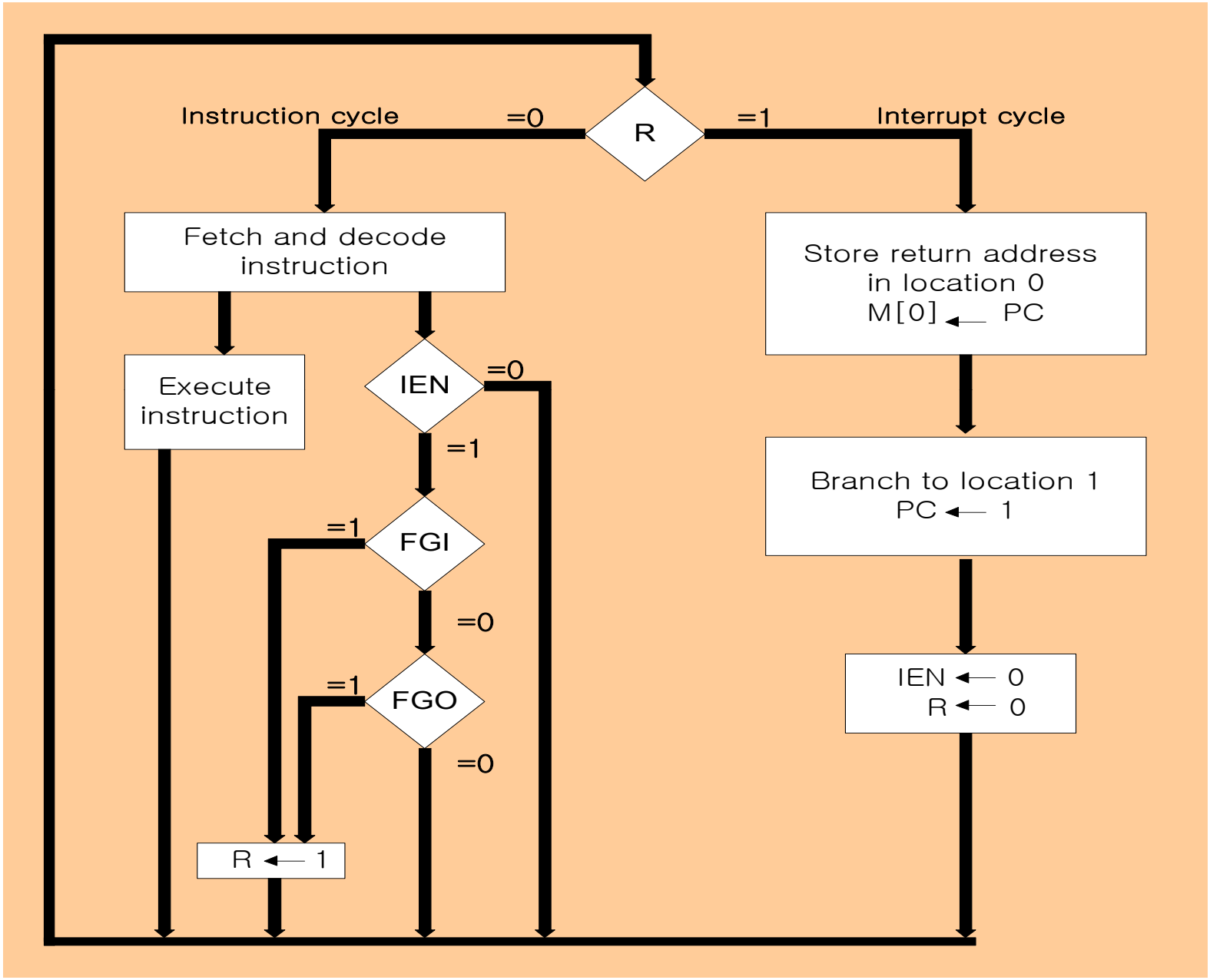
- Modified Fetch Phase
  - Modified Fetch and Decode Phase

Save Return Address(PC) at 0

Jump to 1(PC=1)

$$RT_0 : AR \leftarrow 0,\ TR \leftarrow PC$$
$$RT_1 : M[AR] \leftarrow TR,\ PC \leftarrow 0$$
$$RT_2 : PC \leftarrow PC + 1,\ IEN \leftarrow 0,\ R \leftarrow 0,\ SC \leftarrow 0$$




Instruction cycle
=0
R
=1
Interrupt cycle
Fetch and decode instruction
Store return address in location 0
M[0] ← PC
Execute instruction
IEN
=0
=1
Branch to location 1
PC ← 1
=1
FGI
=0
IEN ← 0
R ← 0
=1
FGO
=0
R ← 1

Design of Basic Computer

- Two basic things are needed: data paths and control signals
- A hardwired-control implementation: Stitch together the individual pieces of the data path.
- The microoperation table provides sufficient information to implement the circuits for control (wiring various gates).

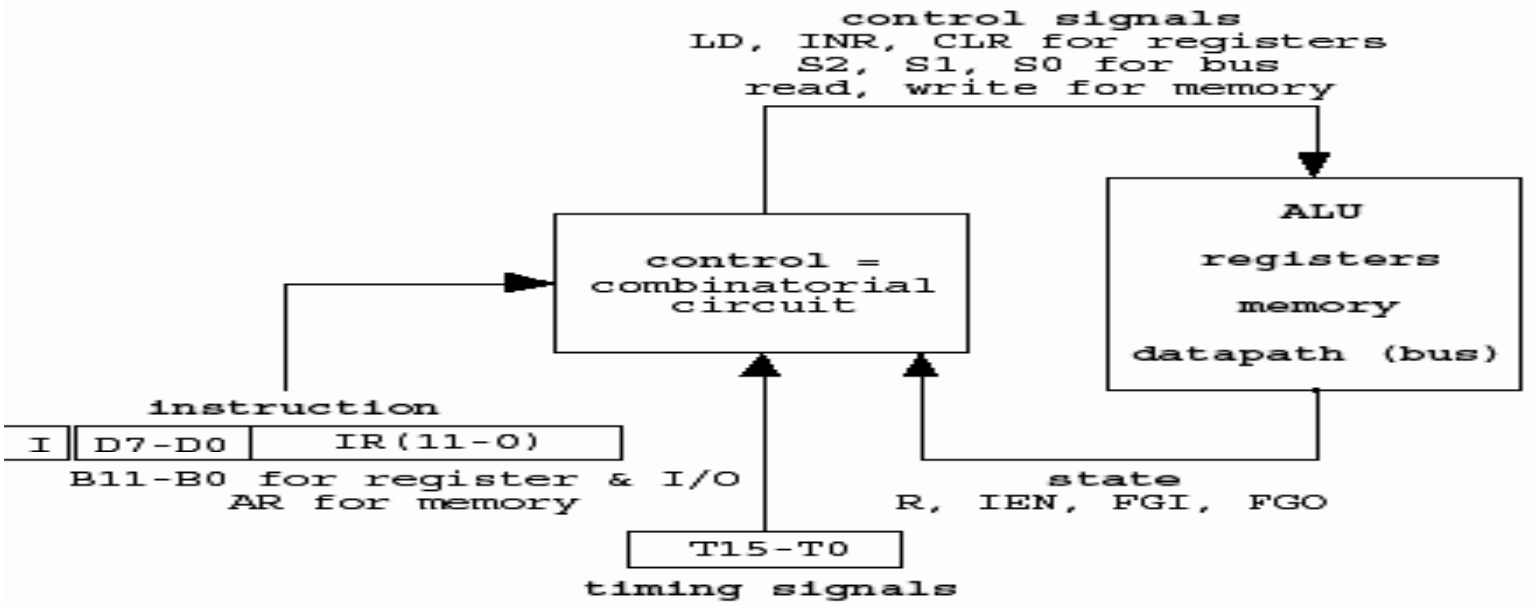


Figure: Where the control resides in the computer.

Input:

1. D0 - D7: Decoded IR(14-12)
2. T0 - T15 : Timing signals
3. I: Indirect signal
4. IR(0-11)

Output:

1. Control inputs of the nine registers, AR, PC, DR, AC, IR, TR, OUTR, INPR, SC
2. Read and write inputs of memory
3. Signals to set, clear, or complement the flip-flops, IEN, R, etc.
4. Select signals, S2, S1, S0, for common bus
5. Control signals to the AC adder and logic circuit

CONTROL OF REGISTERS AND MEMORY

Systematic Design Procedure

1. For a given register, scan the table of microoperations in the previous slides to find all the statements involving that gate.
2. Translate the associated control functions to Boolean functions.
3. Convert the Boolean expressions into logic gates.

Example: Control of AR

1. The following is the summary of the register transfers associated with the address register.

R'T0: AR ← PC load

R'T2: AR ← IR(0-11) load

D7'IT3: AR ← M[AR] load

RT0: AR ← 0 clear

D5T4: AR ← AR + 1 increment

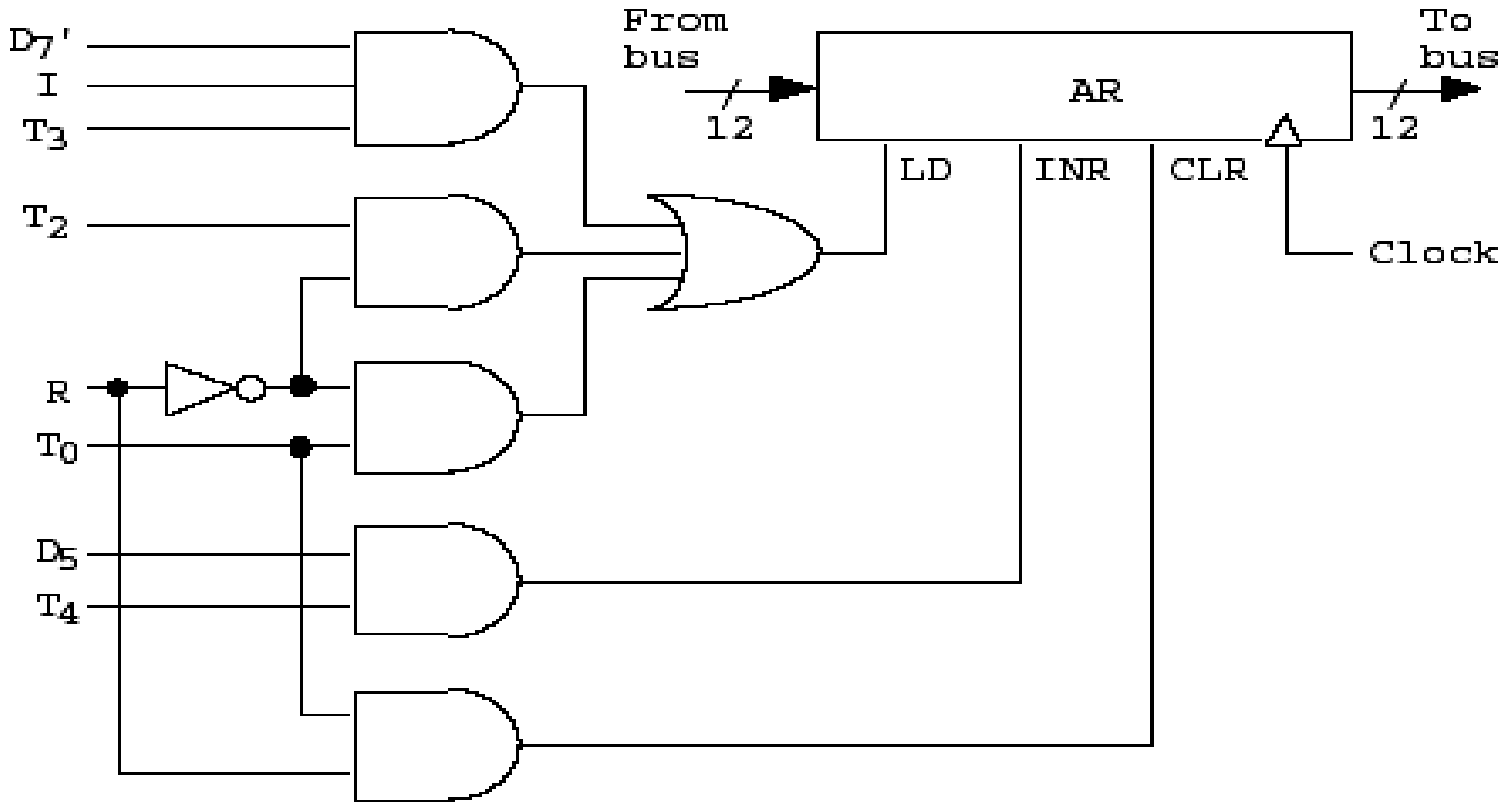
2. The control functions can be combined into the following Boolean expressions.

LD(AR) = R'T0 + R'T2 + D7'IT3

CLR(AR) = RT0

INR(AR) = D5T4

3. The previous Boolean expressions can be converted to the following logic gates.



• In a similar fashion, the control gates for the other registers and memory can be derived. For example, the logic gates associated with the read input of memory is derived by scanning the microoperation table to find the statements that specify a read operation. The read operation is recognized from the symbol ← M[AR].

Read = R'T1 + D7'IT3 + (D0 + D1 + D2 + D3)T4

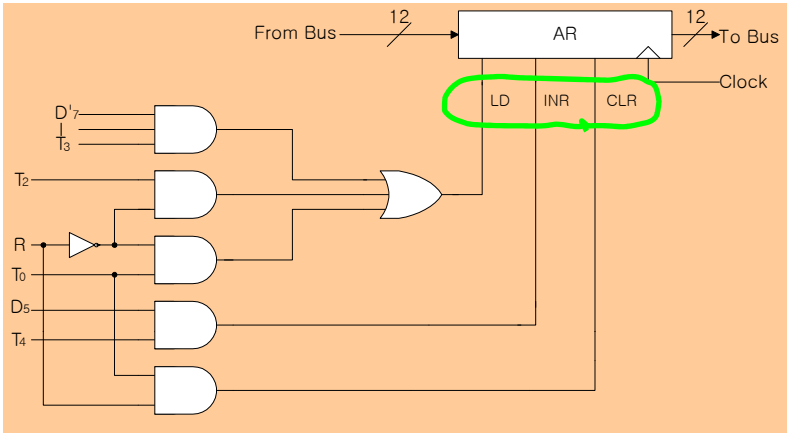
Design of Basic Computer

- Register Control : AR
  - Control inputs of AR : ***LD, INR, CLR***
  - ***Find all the statements that change the AR in Table. 5-6***
  - Control functions

$AR \leftarrow ?$

$$R'T_0 : AR \leftarrow PC$$
$$R'T_1 : AR \leftarrow IR(0\text{–}11)$$
$$D_7'IT_3 : AR \leftarrow M[AR]$$
$$RT_0 : AR \leftarrow 0$$
$$D_5T_4 : AR \leftarrow AR + 1$$

$$LD(AR) = R'T_0 + R'T_1 + D_7'IT_3$$
$$CLR(AR) = RT_0$$
$$INR(AR) = D_5T_4$$



- Memory Control : READ
  - Control inputs of Memory : ***READ, WRITE***
  - Find all the statements that specify a ***read operation*** in Table. 5-6
  - Control function

$M[AR] \leftarrow ?$

$? \leftarrow M[AR]$

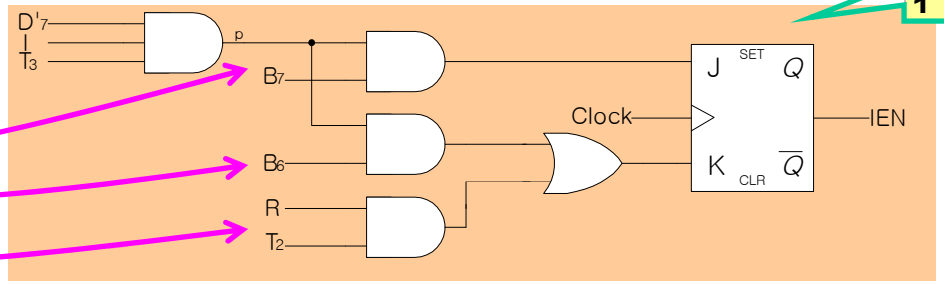
$$READ = R'T_1 + D_7'IT_3 + (D_0 + D_1 + D_2 + D_3)T_4$$

- F/F Control : IEN
  - Control functions

$IEN \leftarrow ?$

$$pB_7 : IEN \leftarrow 1$$
$$pB_6 : IEN \leftarrow 0$$
$$RT_2 : IEN \leftarrow 0$$

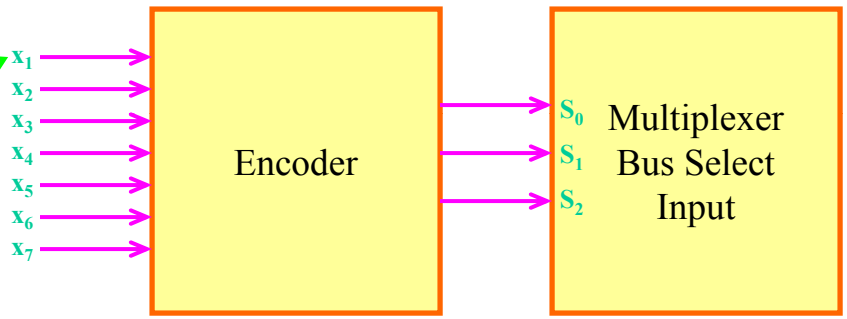
J	K	Q(t+1)
0	1	0
1	0	1



Design of Basic Computer

- Bus Control
  - Encoder for Bus Selection : *Table. 5-7*
    - $S_0 = x_1 + x_3 + x_5 + x_7$
    - $S_1 = x_2 + x_3 + x_6 + x_7$
    - $S_0 = x_4 + x_5 + x_5 + x_7$
  - $x_1 = 1$: $Bus \leftarrow AR = Find\,? \leftarrow AR$
    - $D_4T_4 : PC \leftarrow AR$
      $D_5T_5 : PC \leftarrow AR$
    - Control Function : $x_1 = D_4T_4 + D_5T_5$
  - $x_2 = 1$: $Bus \leftarrow PC = Find\,? \leftarrow PC$
    "
    $Bus \leftarrow Memory = Find\,? \leftarrow M[AR]$
    "
  - $x_7 = 1$: $x_7 = R'T_1 + D_7'IT_3 + (D_0 + D_1 + D_2 + D_3)T_4$
    - Same as Memory Read
    - Control Function :
    -

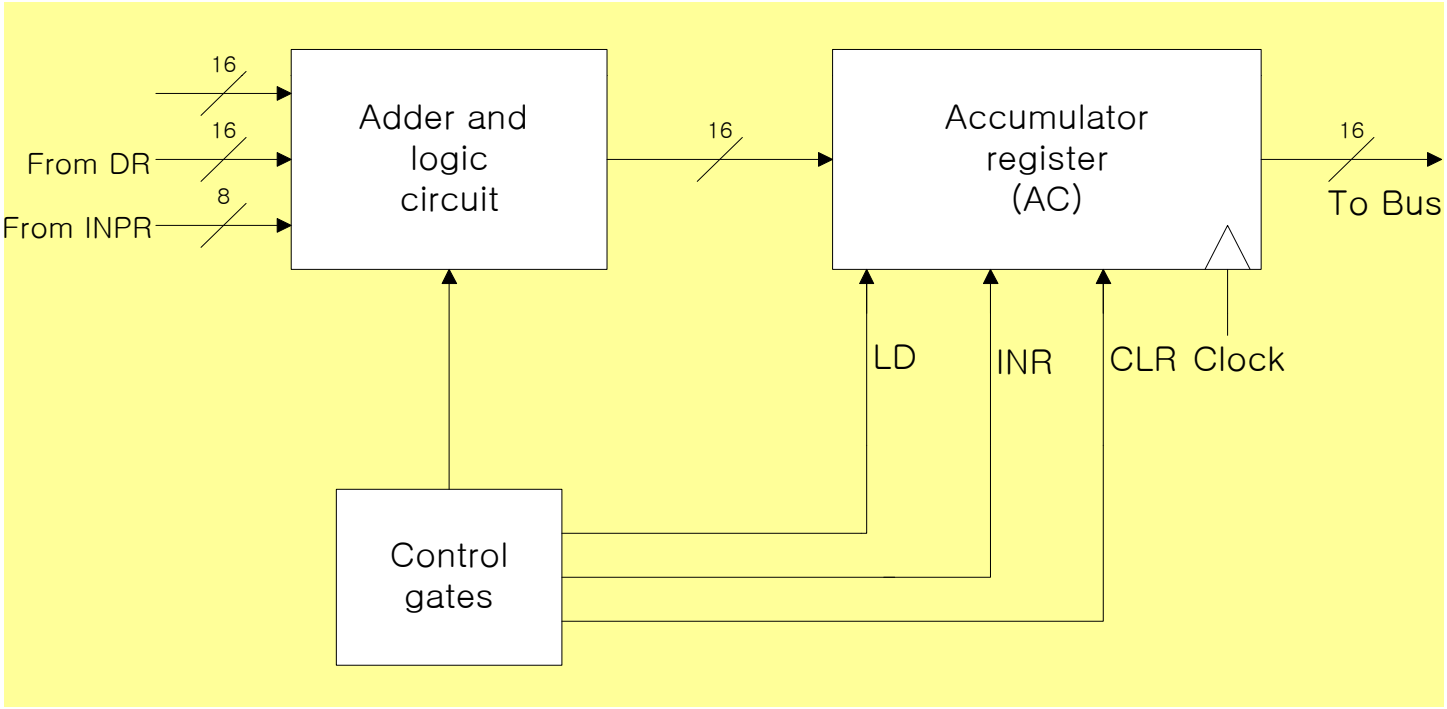
x1 = 1 corresponds to the bus connection of AR as a source



Inputs							Outputs			Register selected for bus
x1	x2	x3	x4	x5	x6	x7	S2	S1	S0	
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

Design of Accumulator Logic

- Design of Accumulator Logic
  - Circuits associated with AC



Design of Accumulator Logic

Control of AC : ***Fig. 5-20***

- Find the statement that change the AC : $AC \leftarrow ?$

$D_0T_5 : AC \leftarrow AC \wedge DR$

$D_1T_5 : AC \leftarrow AC + DR$

$D_2T_5 : AC \leftarrow DR$

$pB_{11} : AC(0-7) \leftarrow INPR$

$rB_9 : AC \leftarrow \overline{AC}$

$rB_7 : AC \leftarrow shr\ AC,\ AC(15) \leftarrow E$

$rB_6 : AC \leftarrow shr\ AC,\ AC(0) \leftarrow E$

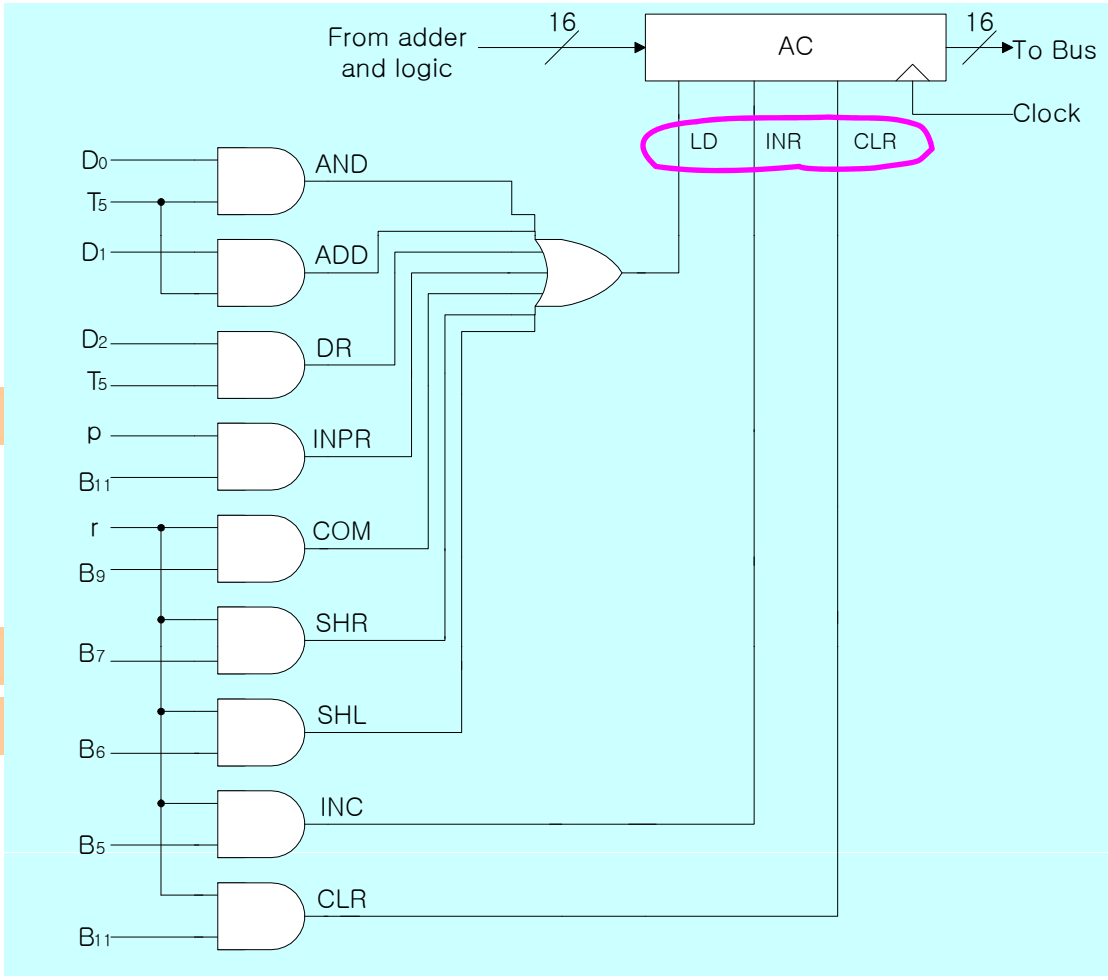
$rB_{11} : AC \leftarrow 0$

$rB_5 : AC \leftarrow AC + 1$

LD

CLR

INR



Design of Accumulator Logic

– Adder and Logic Circuit : ***Fig. 5-21 (16 bit)***