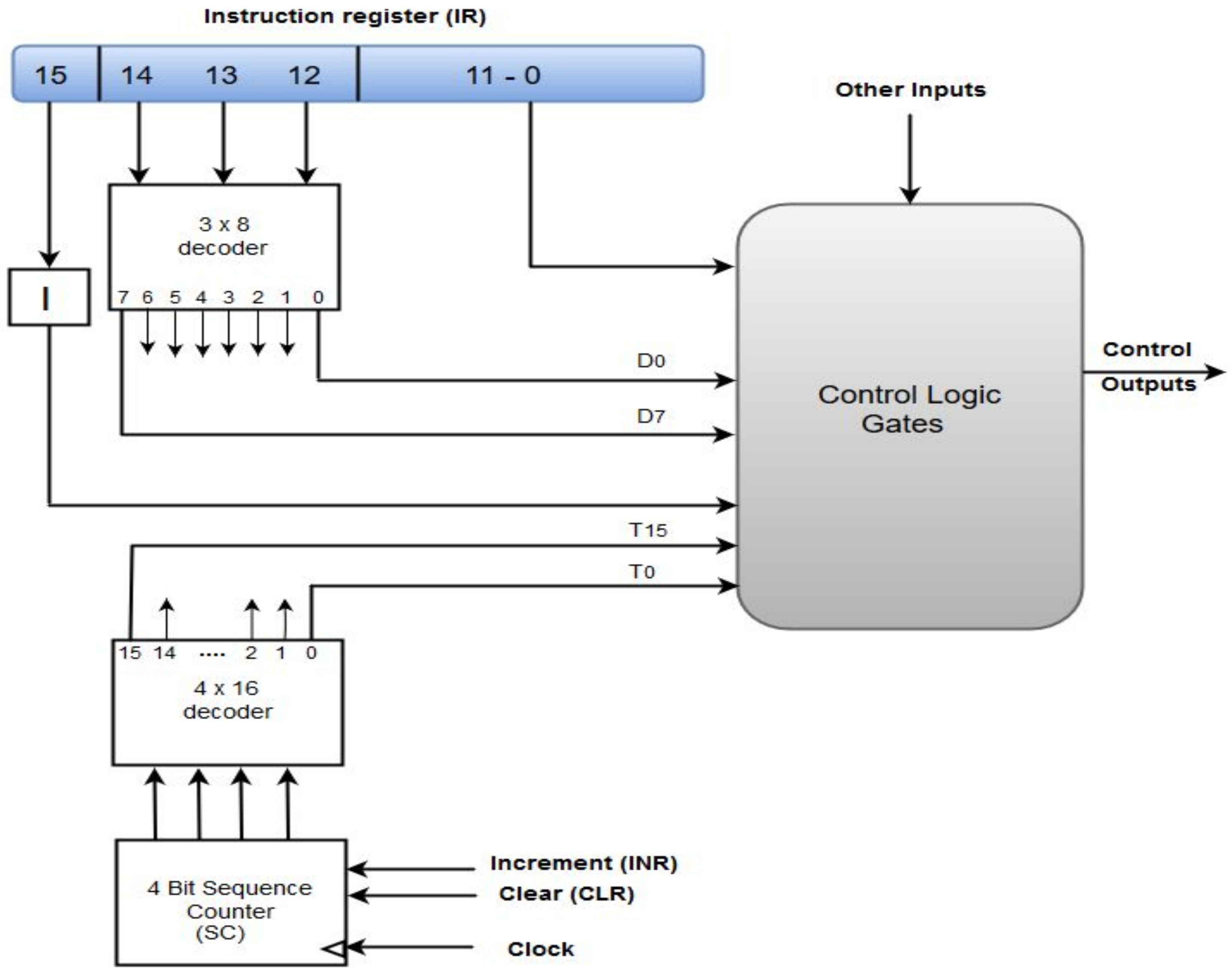
Unit 3

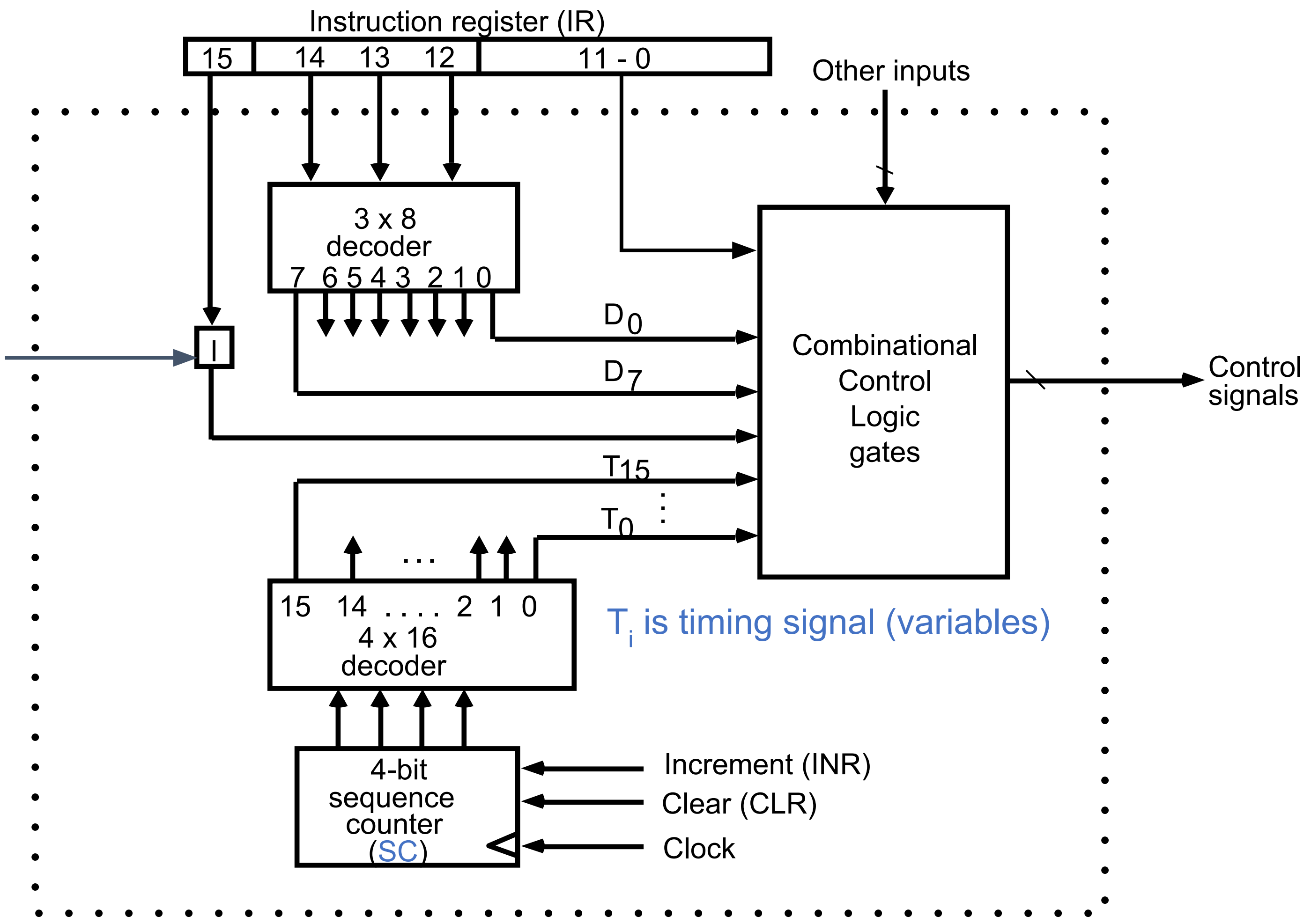
Control Unit of a Basic Computer:



TIMING AND CONTROL SIGNALS CONTINUED …

- The control unit consists of
  - Two decoders,
  - A sequencer, and a number of control logic gates

Flip-flop



T_i is timing signal (variables)

Figure 5.6 Control unit of Basic Computer

TIMING AND CONTROL SIGNALS CONTINUED …

- Timing signals T_i are generated by 4-bit sequence counter (SC) and 4×16 decoder

-The SC can be incremented or cleared.

-The timing signal T_i is active during one clock cycle

- Example: $T_0, T_1, T_2, T_3, T_4, T_0, T_1, \ldots$

Assume: At time T_4, SC is cleared to 0 if decoder output D3 is active is expressed as:

$$D_3T_4: SC \leftarrow 0$$

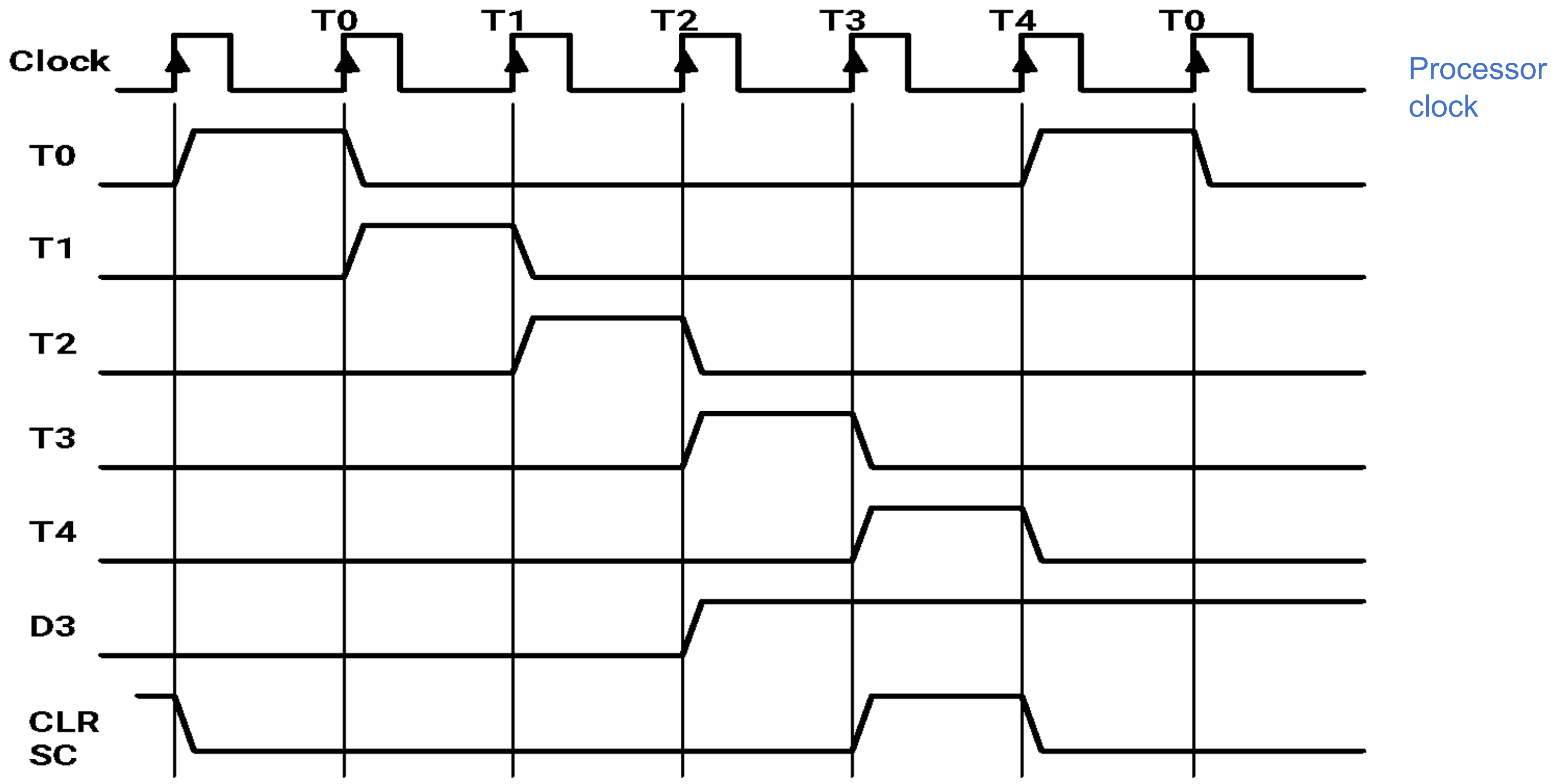


Figure 5.7 Example of control timing signals

TIMING AND CONTROL SIGNALS CONTINUED …

- Register transfer statement

  T_0: AR $\leftarrow$ PC

  - This specifies a transfer of contents of PC to AR if timing signal T_0 is active
  - T_0 is active during entire clock cycle interval (first clock cycle)
  - During T_0 the contents 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 clock goes through the positive transition.
  - The same positive transition increments the sequence counter SC from 0000 to 0001.

INSTRUCTION CYCLE

- In Basic Computer, a machine instruction is executed in the following cycle:
  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

- After an instruction is executed, the cycle starts again at step 1 to fetch, decode, and execute for the next instruction

- *Note*: Every different processor has its own (different) instruction cycle

T_0: AR ← PC ($S_2S_1S_0$= 010, T_0=1) (PC has address of 1st instr. in Pro.)
T_1: IR ← M [AR], PC ← PC + 1 ($S_2S_1S_0$=111, T_1=1)
T_2: D0, . . . , D7 ← Decode IR(12-14), AR ← IR(0-11), I ← IR(15)

FETCH and DECODE

- Fetch and Decode
  - RTL statements

T_0: AR ← PC (S$_2$S$_1$S$_0$= 010, T_0=1) (PC has address of 1st instr. in Pro.)
T_1: IR ← M [AR], PC ← PC + 1 (S$_2$S$_1$S$_0$=111, T_1=1)
T_2: D0, . . . , D7 ← Decode IR(12-14), AR ← IR(0-11), I ← IR(15)

DETERMINE THE TYPE OF INSTRUCTION

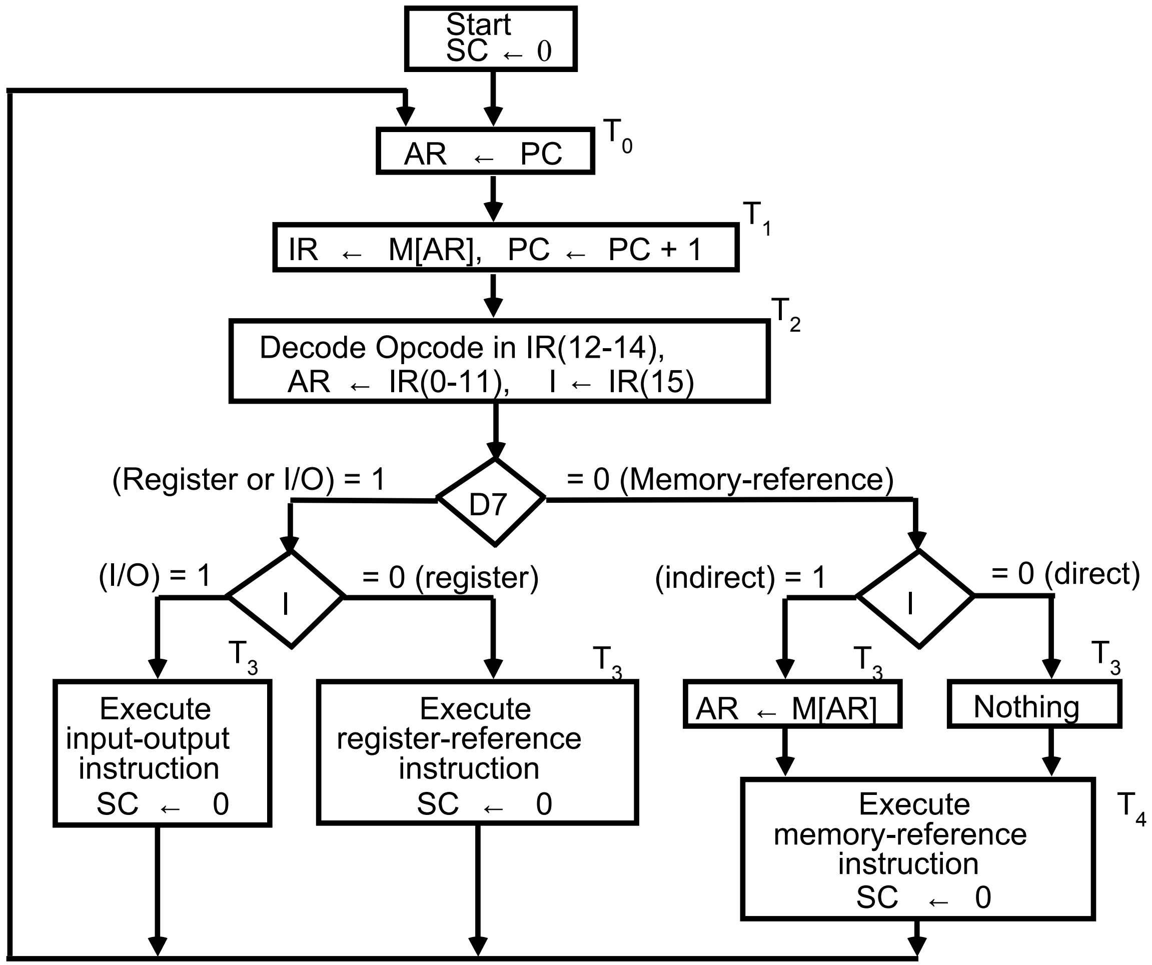


Figure 5.9 Flowchart for instruction cycle (initial configuration)

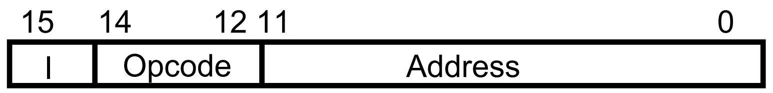
Instruction Paths:

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.

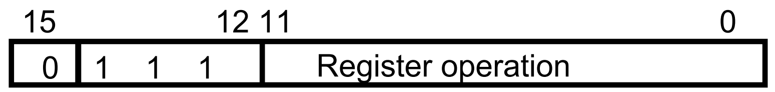
THE BASIC COMPUTER INSTRUCTIONS

- The Basic Computer has three Instruction Formats

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



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



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

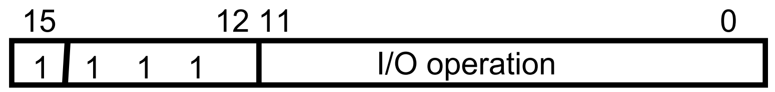


Figure 5.5 Basic computer instruction formats.

MICROINSTRUCTION FORMAT

Information in a Microinstruction

- **Control Information**
- **Sequencing Information**
- **Constant**

  Information which is useful when feeding into the system

These information needs to be organized in some way for

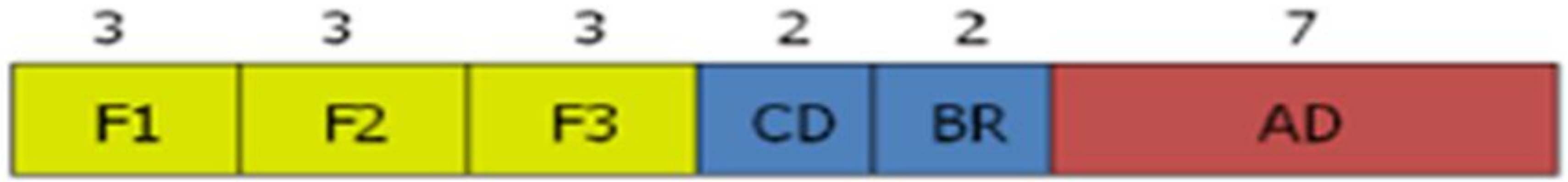
- **Efficient use of the microinstruction bits**
- **Fast decoding**

Field Encoding

- **Encoding the microinstruction bits**
- **Encoding slows down the execution speed due to the decoding delay**
- **Encoding also reduces the flexibility due to the decoding hardware**

Contd..

- The three fields F1, F2, and F3 specify microoperations for the computer. The microoperations are subdivided into three fields of three bits each. The three bits in each field are encoded to specify seven distinct microoperations.



F1, F2, F3: Microoperation fields
CD: Condition for branching
BR: Branch field
AD: Address field

Fig Microinstruction code format

Mapping of Instruction

- A special type of branch exists when a microinstruction specifies a branch to the first word in control memory where a microprogram routine for an instruction is located.
- The status bits for this type of branch are the bits in the operation code part of the instruction. For example, a computer with a simple instruction format as shown in figure 4.3 has an operation code of four bits which can specify up to 16 distinct instructions.
- Assume further that the control memory has 128 words, requiring an address of seven bits.

Contd..

- One simple mapping process that converts the 4-bit operation code to a 7-bit address for control memory is shown in figure 4.3.
- This mapping consists of placing a 0 in the most significant bit of the address, transferring the four operation code bits, and clearing the two least significant bits of the control address register.
- This provides for each computer instruction a microprogram routine with a capacity of four microinstructions.
- If the routine needs more than four microinstructions, it can use addresses 1000000 through 1111111. If it uses fewer than four microinstructions, the unused memory locations would be available for other routines.

Contd..

- One can extend this concept to a more general mapping rule by using a ROM to specify the mapping function. ⍰ The contents of the mapping ROM give the bits for the control address register

MAPPING OF INSTRUCTIONS TO MICROROUTINES

Mapping from the OP-code of an instruction to the address of the Microinstruction which is the starting microinstruction of its execution microprogram

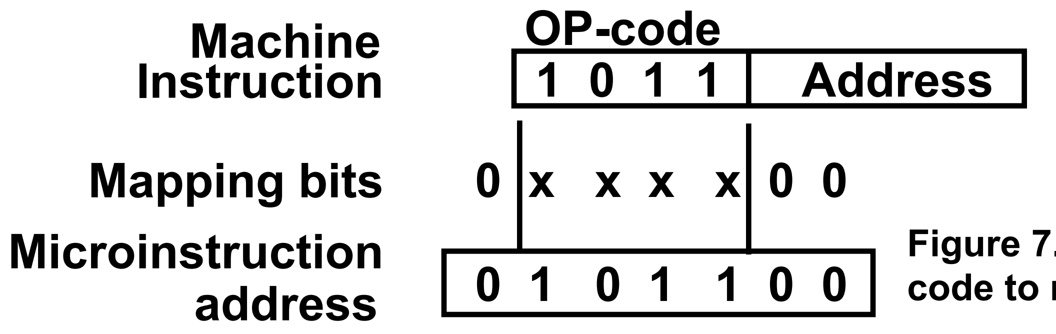
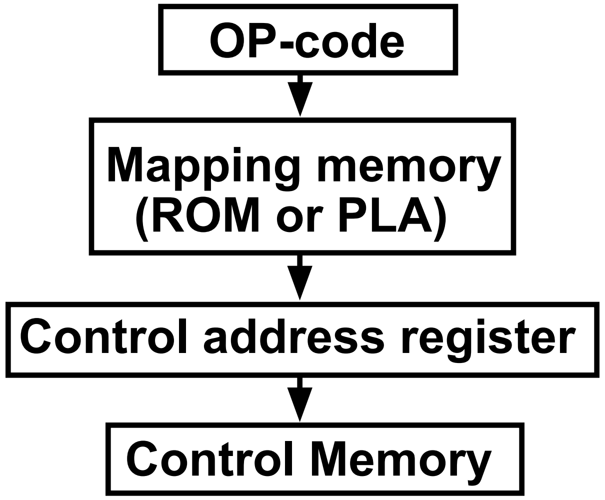


Figure 7.3 Mapping from instruction code to microinstruction address

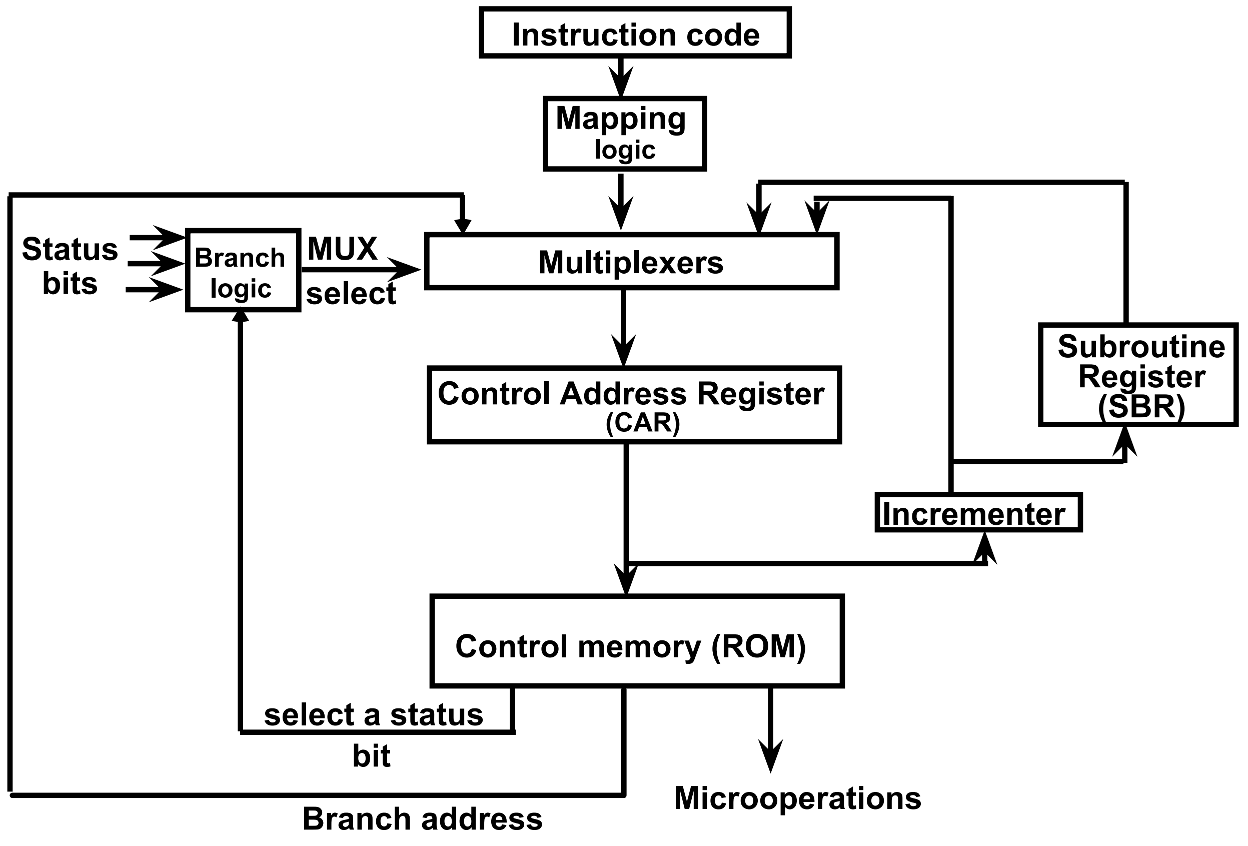
Mapping function implemented by ROM or PLA



Address Sequencing

- Address sequencing capabilities required in control unit
  - Incrementing CAR
  - Unconditional or conditional branch, depending on status bit conditions
  - Mapping from bits of instruction to address for control memory
  - Facility for subroutine call and return

Address Sequencing



MICROINSTRUCTION SEQUENCING

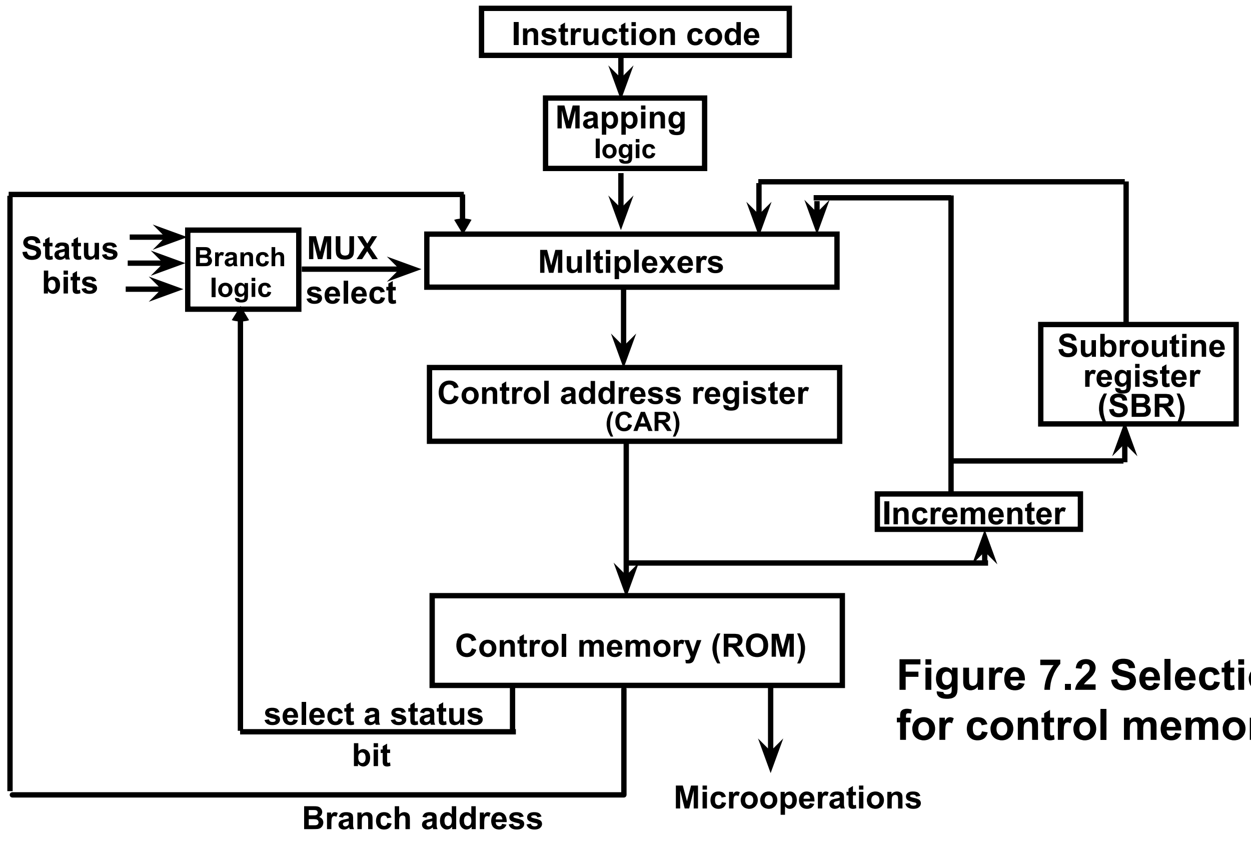
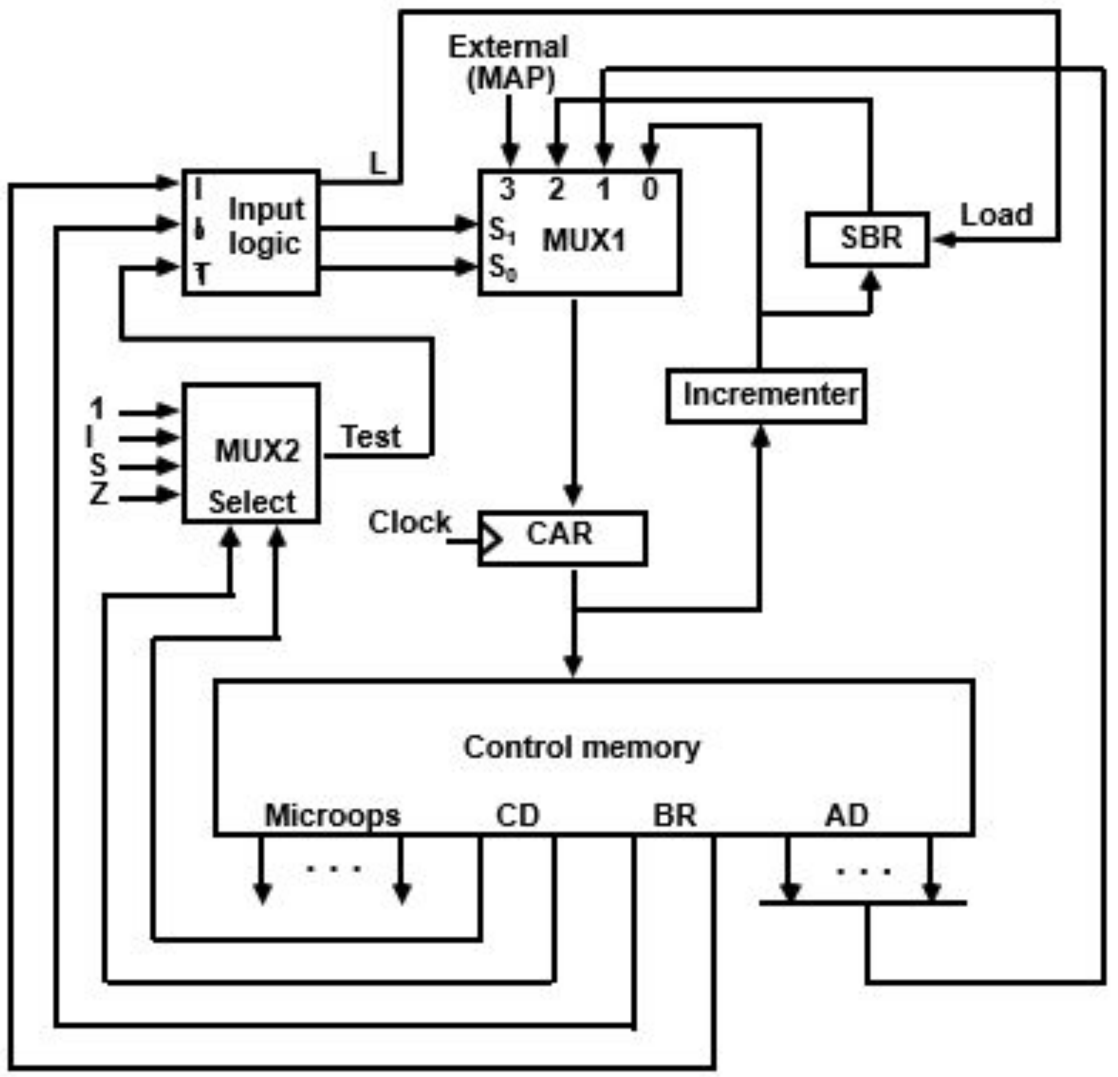


Figure 7.2 Selection of address for control memory

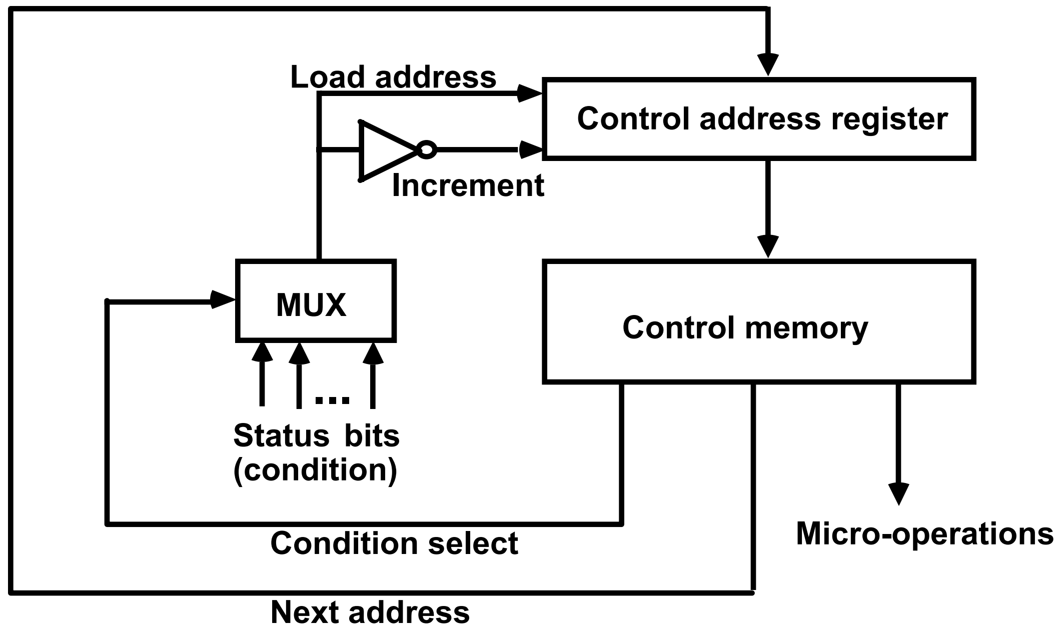
Sequencing Capabilities Required in a Control Storage

- Incrementing of the control address register
- Unconditional and conditional branches
- A mapping process from the bits of the machine instruction to an address for control memory
- A facility for subroutine call and return

MICROPROGRAM SEQUENCER



CONDITIONAL BRANCH



Conditional Branch

If *Condition* is true, then *Branch* (address from the next address field of the current microinstruction) else *Fall Through*

Conditions to Test: O(overflow), N(negative), Z(zero), C(carry), etc.

Unconditional Branch

Fixing the value of one status bit at the input of the multiplexer to 1

MICROPROGRAM SEQUENCER
- NEXT MICROINSTRUCTION ADDRESS LOGIC -

S_1S_0	Address Source
00	CAR + 1, In-Line
01	SBR RETURN
10	CS(AD), Branch or CALL
11	MAP

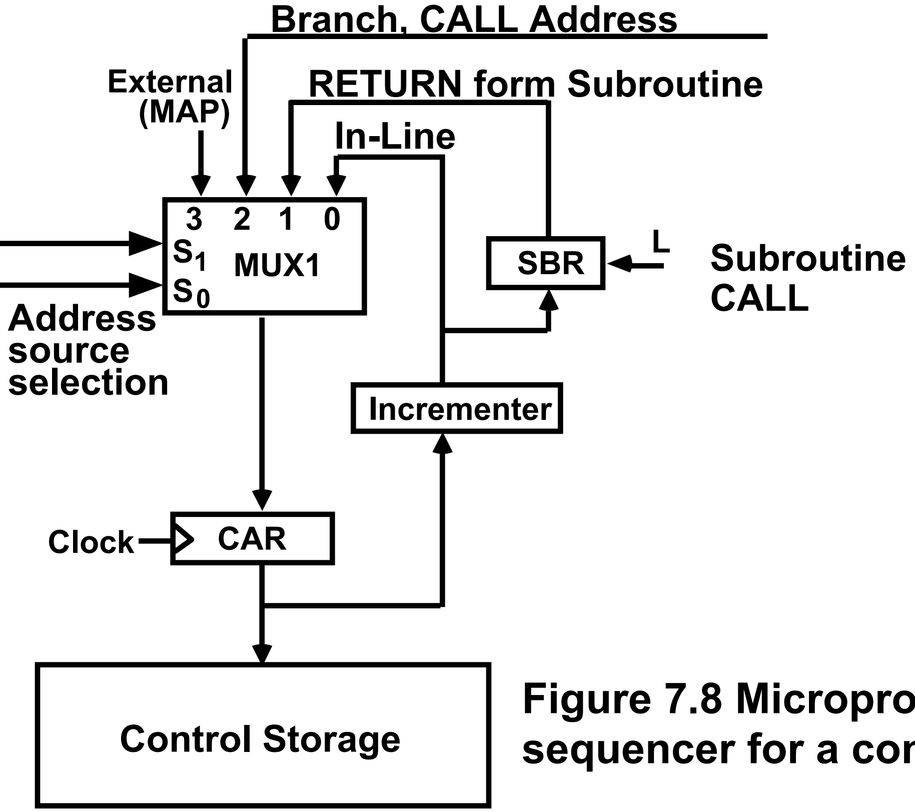
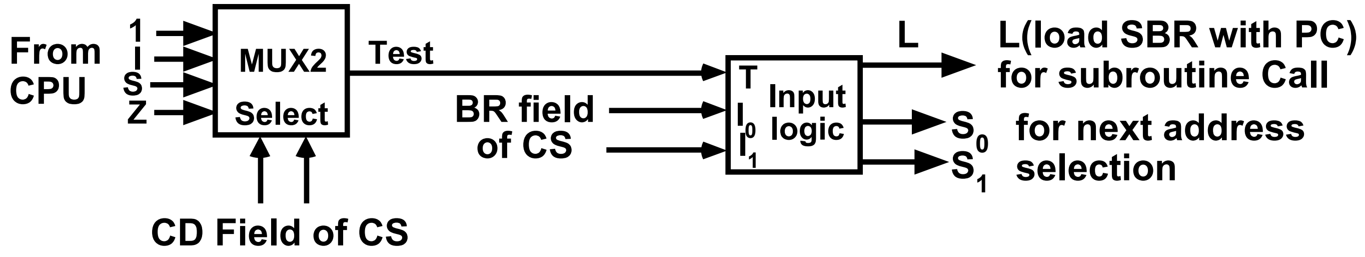


Figure 7.8 Microprogram sequencer for a control memory

MUX-1 selects an address from one of four sources and routes it into a CAR

- **In-Line Sequencing → CAR + 1**
- **Branch, Subroutine Call → CS(AD)**
- **Return from Subroutine → Output of SBR**
- **New Machine instruction → MAP**

MICROPROGRAM SEQUENCER
- CONDITION AND BRANCH CONTROL -



Input Logic

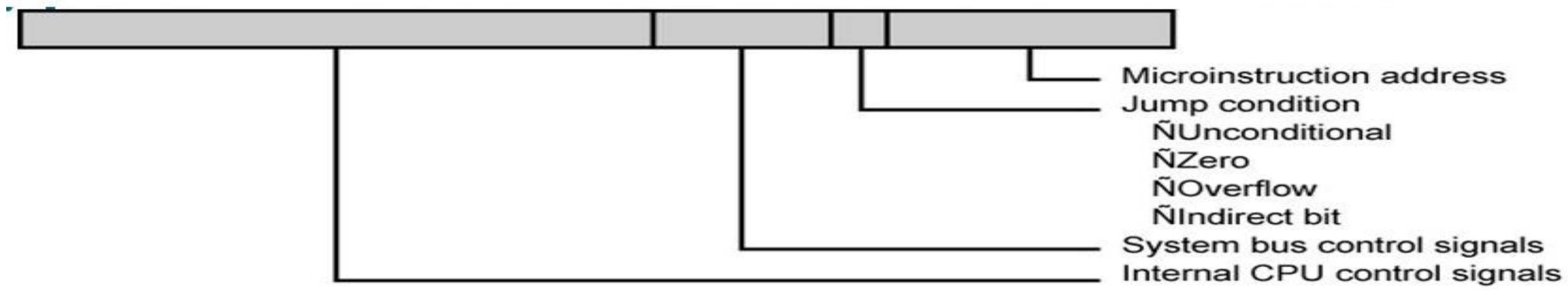
I_0I_1T	Meaning	Source of Address	S_1S_0	L
000	In-Line	CAR+1	00	0
001	JMP	CS(AD)	10	0
010	In-Line	CAR+1	00	0
011	CALL	CS(AD) and SBR <- CAR+1	10	1
10x	RET	SBR	01	0
11x	MAP	DR(11-14)	11	0

$$S_0 = I_0$$
$$S_1 = I_0I_1 + I_0'T$$
$$L = I_0'I_1T$$

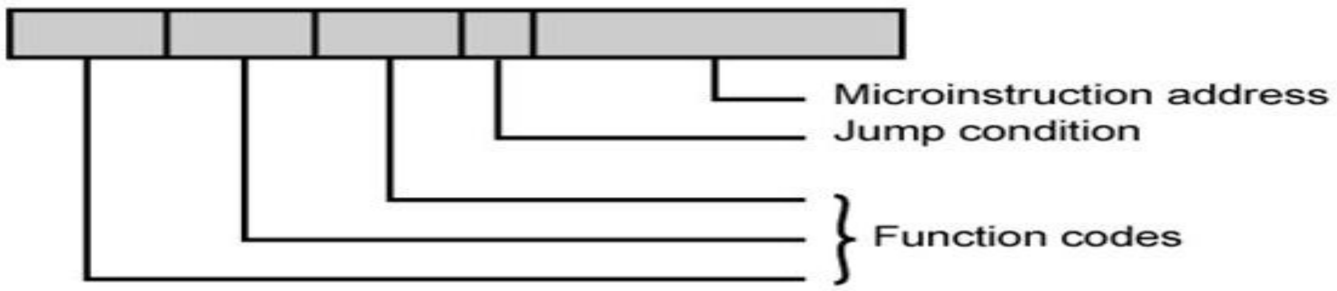
Truth Table for Microprogram Sequencer

BR FIELD		INPUT			MUX 1		LOAD SBR
		I1	I0	T	S1	S0	L
0	0	0	0	0	0	0	0
0	0	0	0	1	0	1	0
0	1	0	1	0	0	0	0
0	1	0	1	1	0	1	1
1	0	1	0	*	1	0	0
1	1	1	1	*	1	1	0

Typical micro Instruction Format



(a) Horizontal microinstruction



(b) Vertical microinstruction

Contd..

- Basically, **control unit (CU)** is the engine that runs the entire functions of a computer with the help of control signals in the proper sequence. In the **micro-programmed** control unit approach, the control signals that are associated with the operations are stored in special memory units. It is convenient to think of sets of control signals that cause specific micro-operations to occur as being “microinstructions”. The sequences of microinstructions could be stored in an internal “*control*” memory.

Micro-programmed control unit can be classified into two types based on the type of Control Word stored in the Control Memory, viz., Horizontal micro-programmed control unit and Vertical micro-programmed control unit.

- In *Horizontal micro-programmed* control unit, the control signals are represented in the decoded binary format, i.e., 1 bit/CS. Here ‘n’ control signals require n bit encoding. On the other hand.
- In *Vertical micro-programmed* control unit, the control signals are represented in the encoded binary format. Here ‘n’ control signals require $\log_2 n$ bit encoding.

Difference b/w Horizontal & Vertical Programming

HORIZONTAL M-PROGRAMMED CU	VERTICAL M-PROGRAMMED CU
It supports longer control word.	It supports shorter control word.
It allows higher degree of parallelism. If degree is n, then n Control Signals are enabled at a time.	It allows low degree of parallelism i.e., degree of parallelism is either 0 or 1.
No additional hardware is required.	Additional hardware in the form of decoders are required to generate control signals.
It is faster than Vertical micro-programmed control unit.	it is slower than Horizontal micro-programmed control unit.
It is less flexible than Vertical micro-programmed control unit.	It is more flexible than Horizontal micro-programmed control unit.
Horizontal micro-programmed control unit uses horizontal microinstruction, where every bit in the control field attaches to a control line.	Vertical micro-programmed control unit uses vertical microinstruction, where a code is used for each action to be performedand thedecoder translates this code into individual control signals.
Horizontal micro-programmed control unit makes less use of ROM encoding than vertical micro-programmed control unit.	Vertical micro-programmed control unit makes more use of ROM encoding to reduce the length of the control word

Example

Example: Consider a hypothetical Control Unit which supports

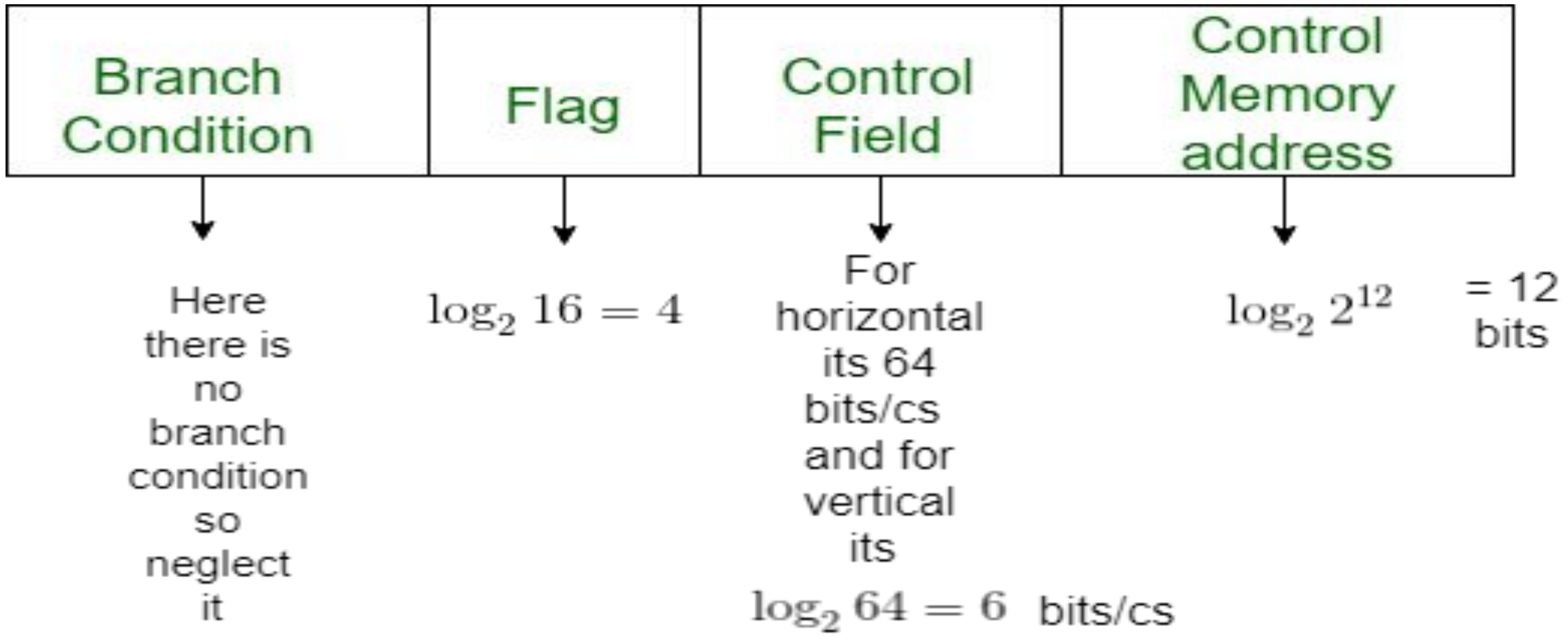
4 k words. The Hardware contains 64 control signals and 16 Flags. What is the size of control word used in bits and control memory in byte using:

a) Horizontal Programming

b) Vertical programming

solution

Solution:



Branch Condition
Flag
Control Field
Control Memory address
Here there is no branch condition so neglect it
$\log_2 16 = 4$
For horizontal its 64 bits/cs and for vertical its
$\log_2 64 = 6$ bits/cs
$\log_2 2^{12}$ = 12 bits

Contd..

a)For Horizontal
64 bits for 64 signals
Control Word Size = 4 + 64 + 12 = 80 bits
Control Memory = 4 kW = ((4* 80) / 8) = 40 kByte

a)For Vertical
6 bits for 64 signals i.e $\log_2 64$
Control Word Size = 4 + 6 + 12 = 22 bits
Control Memory = 4 kW = ((4* 22) / 8) = 11 kByte

HORIZONTAL AND VERTICAL MICROINSTRUCTION FORMAT

Horizontal Microinstructions

Each bit directly controls each micro-operation or each control point

***Horizontal* implies a long microinstruction word**

Advantages: Can control a variety of components operating in parallel.

--> Advantage of efficient hardware utilization

Disadvantages: Control word bits are not fully utilized

--> CS becomes large --> Costly

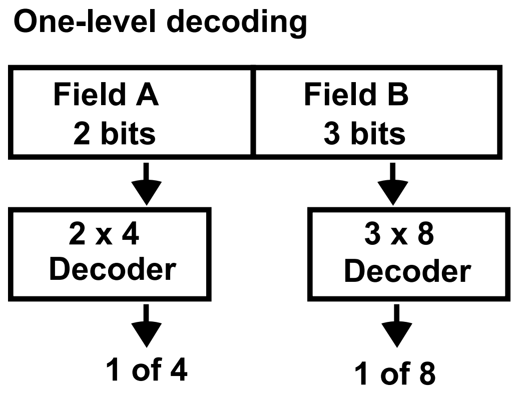
Vertical Microinstructions

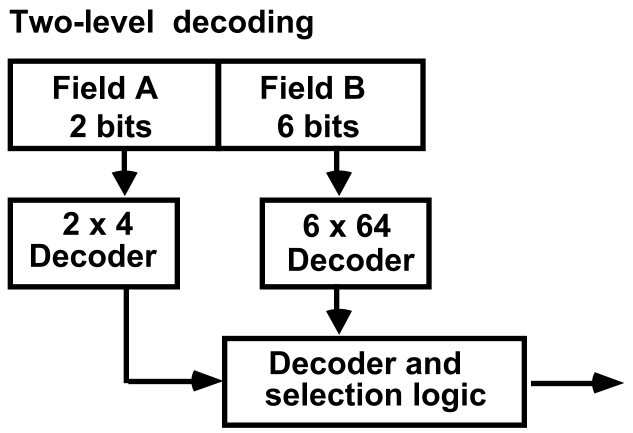
A microinstruction format that is not horizontal

***Vertical* implies a short microinstruction word**

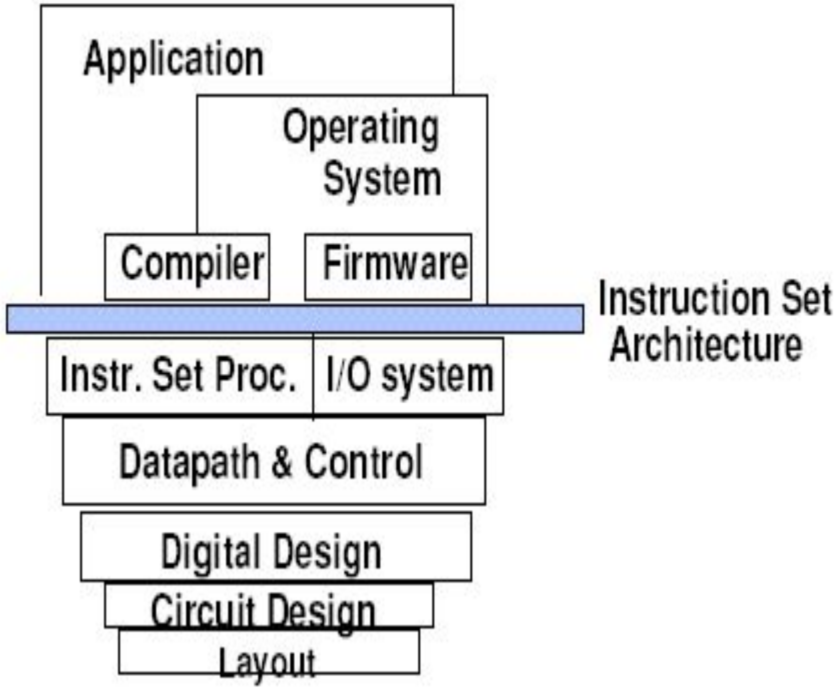
Encoded Microinstruction fields

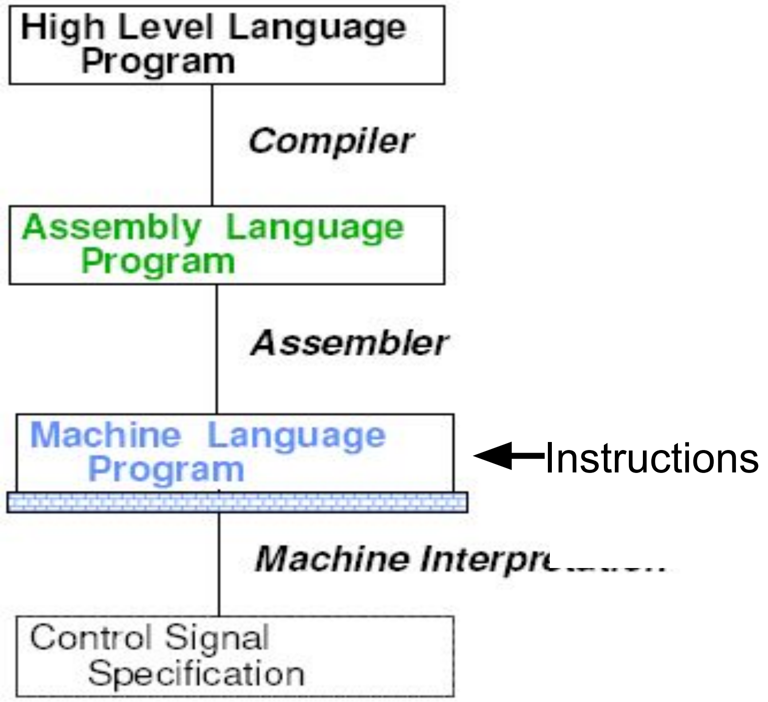
--> Needs decoding circuits for one or two levels of decoding





Levels of Representation




High Level Language Program
Compiler
Assembly Language Program
Assembler
Machine Language Program
Instructions
Machine Interpre
Control Signal Specification

FLOWCHART FOR INTERRUPT CYCLE

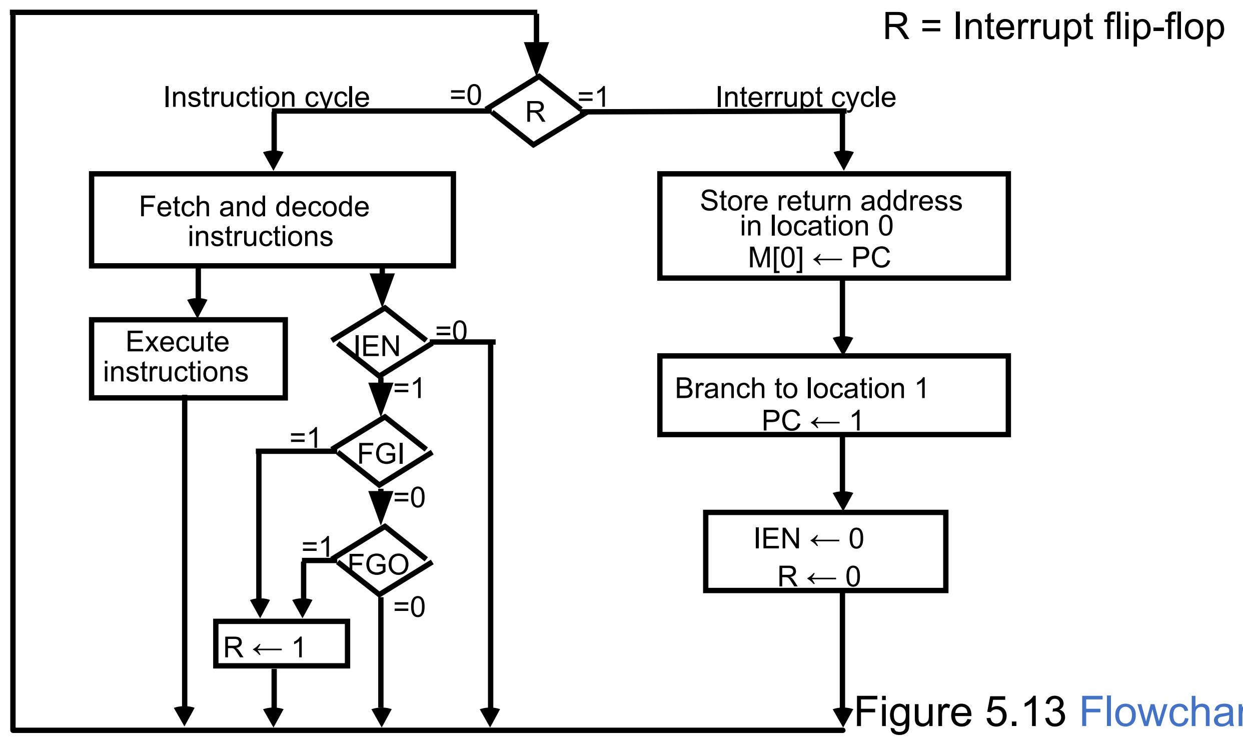


Figure 5.13 Flowchart for interrupt cycle

-The interrupt cycle is a HW implementation of a branch and save return address operation.

- At the beginning of the next instruction cycle, the instruction that is read from memory is in address 1.

- At memory address 1, the programmer must store a branch instruction that sends the control to an interrupt service routine

-The instruction that returns the control to the original program is "indirect BUN 0"

COMPLETE COMPUTER DESCRIPTION

Flowchart of Operations

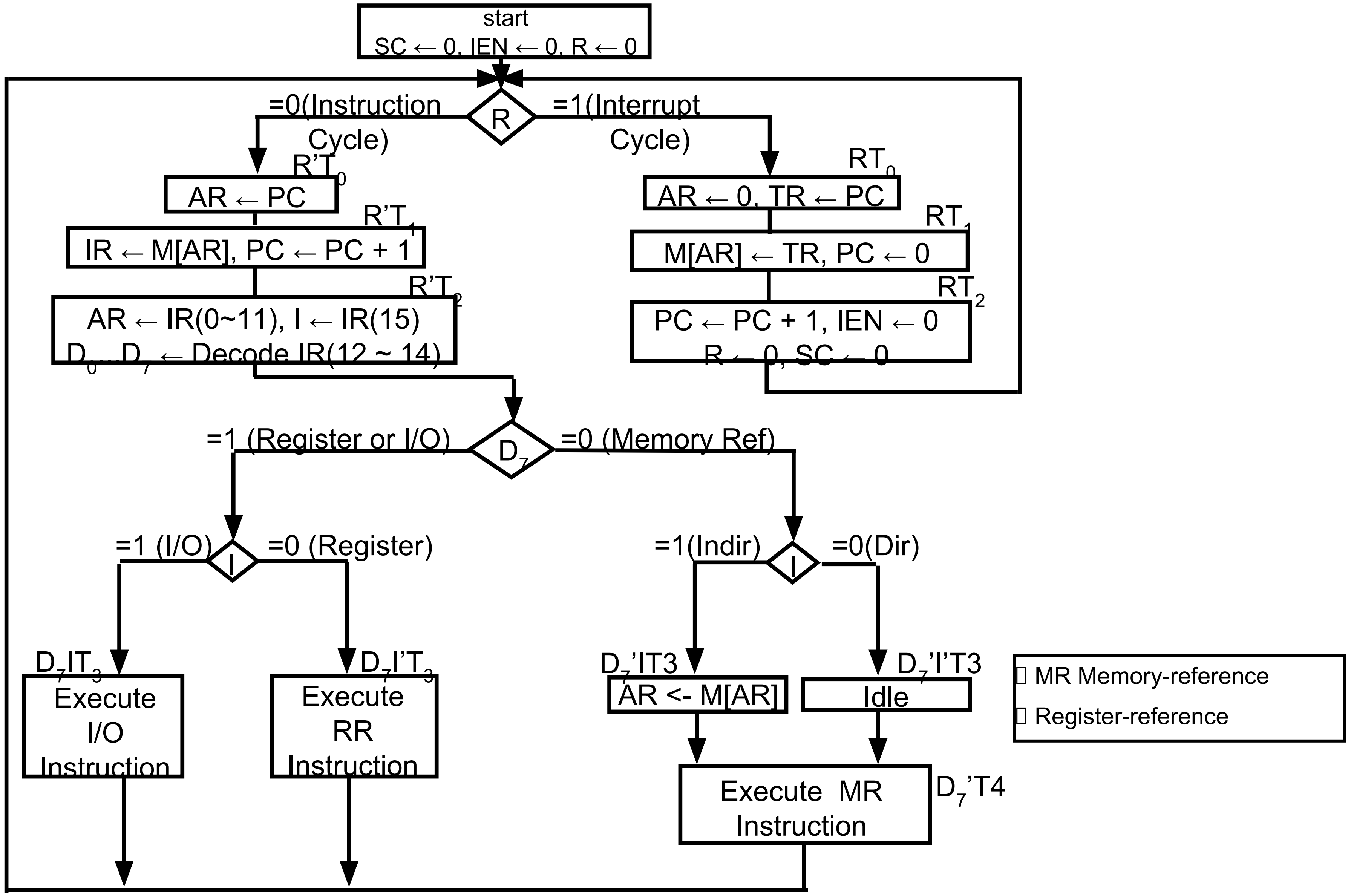


Figure 5.15 Flowchart for computer operation