Control Design

Digital system

- ***Data Processing Unit*** (DPU): capable of performing certain operations on data.
- ***Control Unit*** (CU): issues control signals or instructions to DPU
- CU signals selects the functions to be performed at specific times and route the data through the appropriate functional units.
- DPU is logically reconfigured by the CU to perform certain set of *(micro)operations*.
- Sequence of these micro-operations are important.

Control Design

Functions of CU:

- **Instruction sequencing**: methods by which instructions are selected for execution or control of the processor is transferred from one instruction to another.

- **Instruction Interpretation**: methods used for activating the control signals that cause the DPU to execute the instructions.

Instruction sequencing

- Each instruction can explicitly specify the address of its successor(s).
- It may increase the instruction length, which in turn increases the cost of memory where it is stored.
- I1 is stored in location A, and I2 is its successor.
- Let PC contains A of I1, the address of I2 is can be determined by PC ← PC+k, k is the word length of I1.
- Increment of PC can be automatic.
- So, PC makes it unnecessary for an instruction to specify the address of its successor.

Instruction sequencing

- Branch Instruction: Specifies implicitly/explicitly an instruction address X.
- Unconditional branch always alters the flow of control by causing PC ← X.
- Conditional branch first tests the condition C (a result generated by earlier instruction, and stored in a status register).
- If C is present, then PC ← X, otherwise PC is incremented to point to the next consecutive instruction.

Program Control Transfer

- Often it is required to temporal transfer of control from P1 to P2 due to subroutine call or interrupt.
- Subroutine call: CALL X (X is address of the first instruction of P2)
- First contents of PC is stored (after fetching CALL) in a predetermined location, then X is loaded into PC.
- At then end of P2, transfer is returned back to P1.
- In case of interrupt, call instruction is replaced by interrupt signal.
- Interrupt branch address is fixed a priori, or comes with the interrupt request.

Control Stack

- Stack is used to store return address.
- Also used to store pass variables (from P1 to P2), and variables that are local to P2.
- Each time a call is executed, return address is stored at top of the stack, PC is loaded with SUB address.

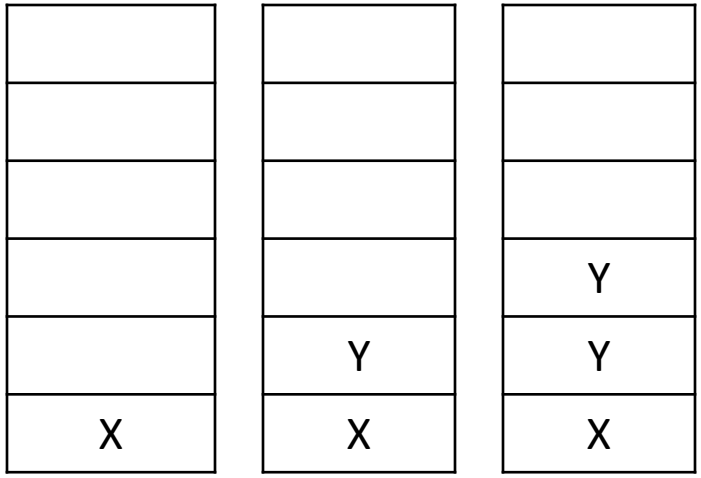
  CALL SUB; PUSH PC; PC← SUB;
- A return is effected by RETURN (equivalently to POP).
- LIFO organization of pushdown stack is used.
- No restriction on the use of recursive calls.
- Top of the stack varies dynamically; no interference with one another.

Control Stack

- Because return location is saved in top-of-stack, which varies dynamically, successive PUSH operations to save return addresses do not interfere with one another.

```
Begin
    CALL SUB
  X:
SUB:
    CALL SUB
   Y:
    RETURN
End
```

[For total k calls, K-1 returns]



Use of pushdown stack to control recursive subroutine call

Stack Implementation

- A stack with n k-bit words is implemented using k n-bit shift registers having left and right-shit capabilities.
- One end of the shift register is defined as top-of-stack.
- Push/pop operation is done by activating the proper control line.
- Two possible error conditions may arise: *stack overflow* and *stack underflow*.
- Error is detected by a counter in the circuit to indicate the no. of words currently in the stack.
- Counter in incremented (decremented) by each push(pop) operation.
- Combination of push(pop) and count n(0) results in *overflow(underflow)*.

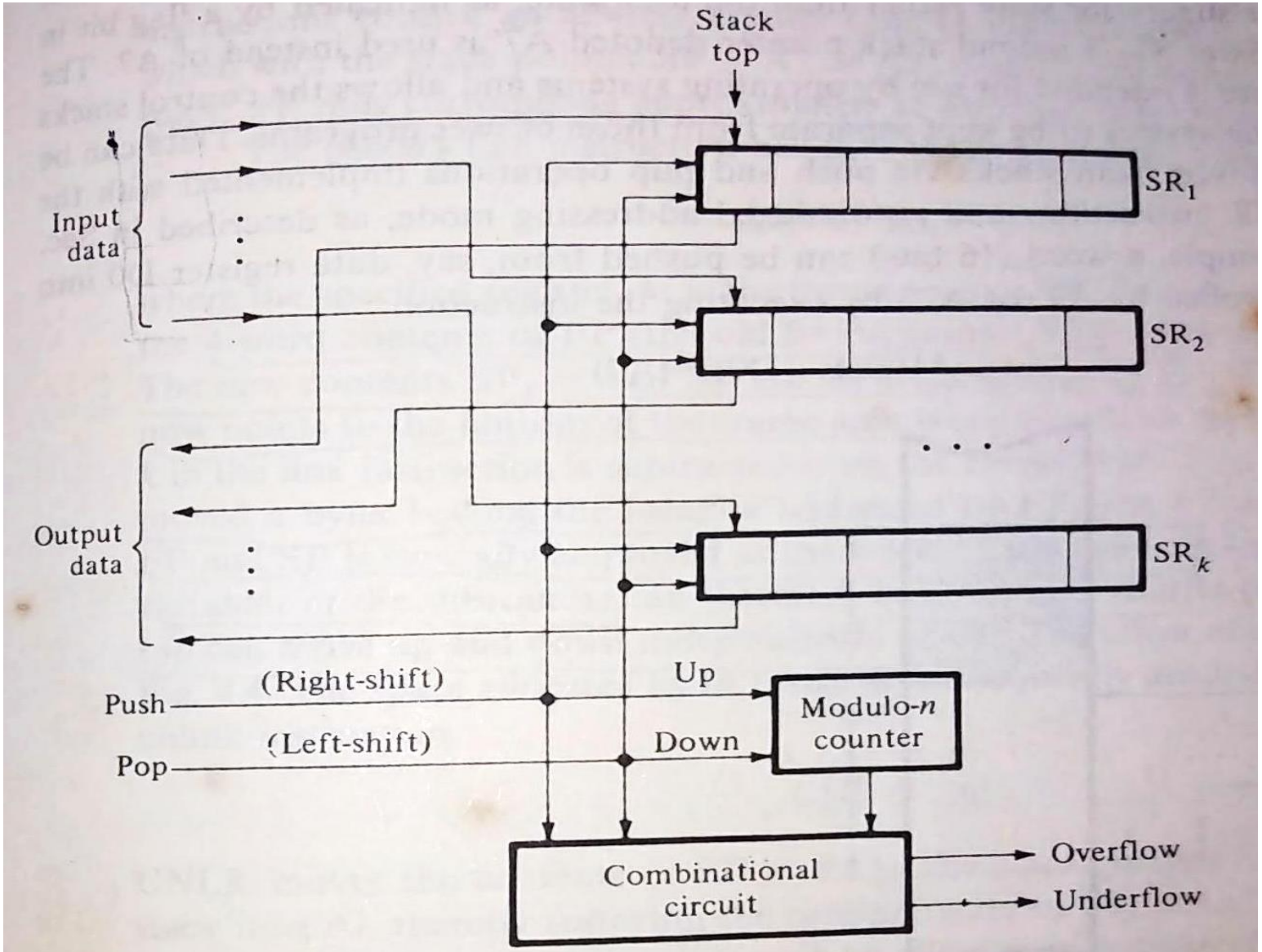
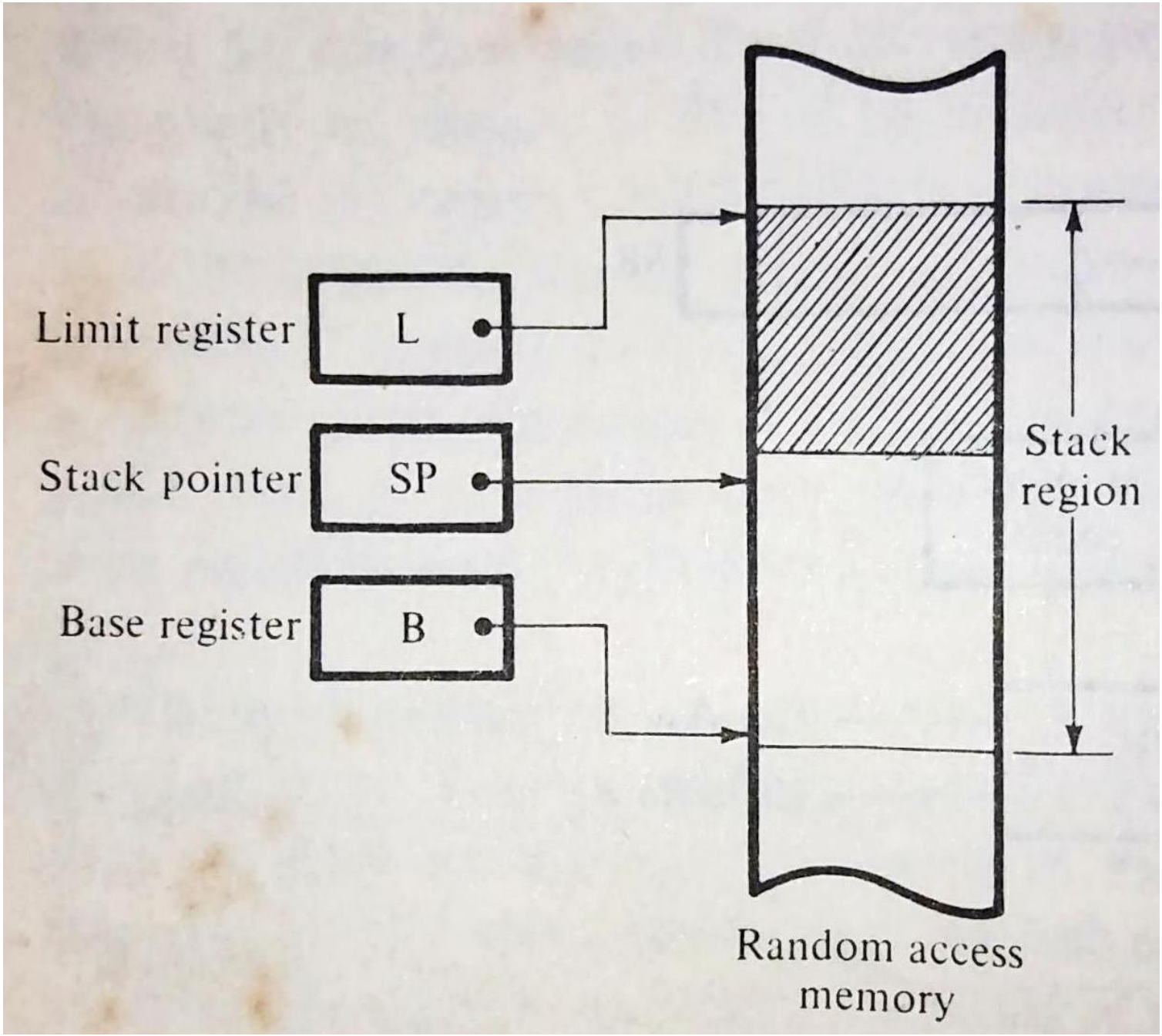


FIGURE 4.2
A stack constructed from shift registers.

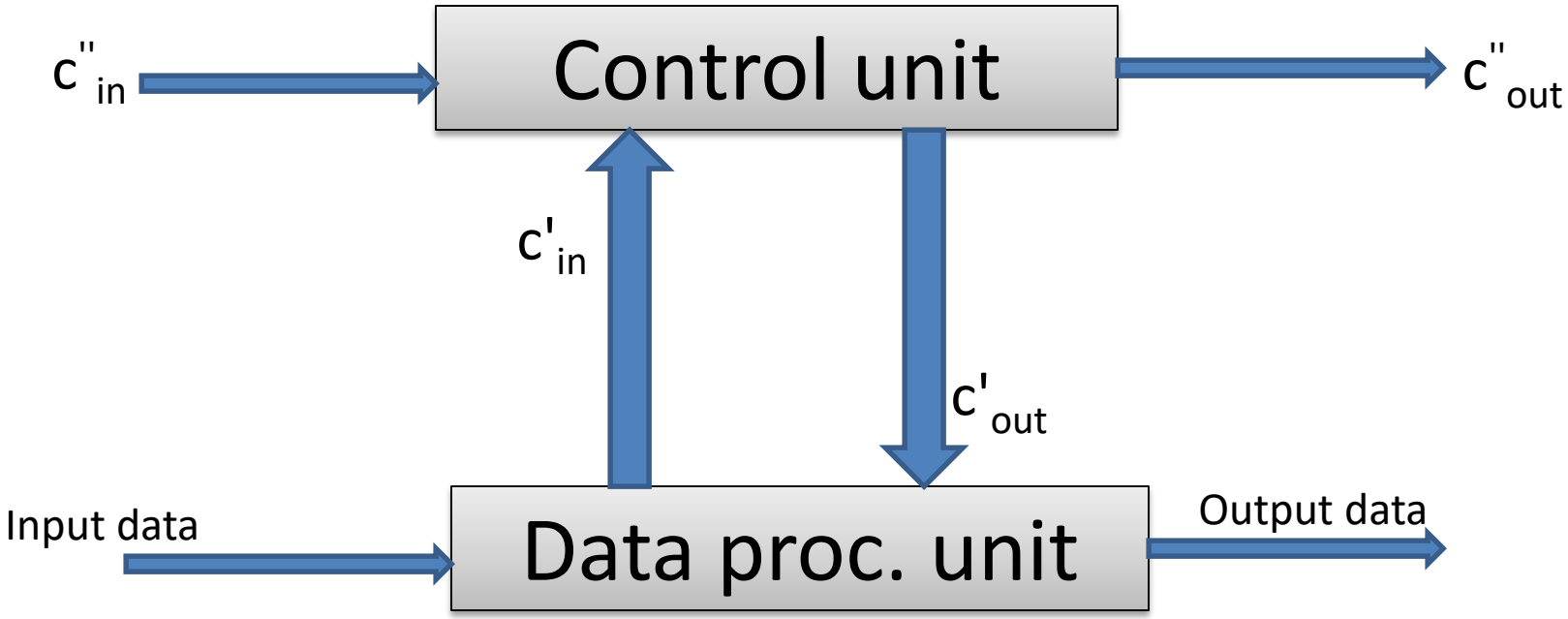
Stack Implementation

- Smaller stack: **Shift register**, Larger stack: **RAM**
- Push: memory write, and pop: memory read
- **Stack pointer (SP)**: this register contains address of the memory location currently acting as top-of-the-stack.
- A pop operation implies reading M using SP.
- SP either decrements (pop) or increments (push).
- **Limit register (L)**: Contains highest stack address.
- **Base Register (B):** Contains lowest stack address.
- B, L and SP define the boundary of the stack.
- SP>L (SP<L) is overflow(underflow).


Limit register
L
Stack pointer
SP
Base register
B
Stack region
Random access memory

- **Instruction Interpretation**

Control signals ->via control lines->outside



-- Control and data are relative, rather absolute.
-- Same physical lines, either data or control at different levels.

Control Specifications :-

C'_{out} :- control the operation of D.P. unit (main function of C.U.).

C'_{in} :- enables the data being processed to influence the C.U., allowing data-dependent decisions to be made.

E.g. error, overflow (unusual conditions)

C''_{out} :- transmitted to other C.U., indicating the status such as ‘busy’ or ‘operation completed’.

C''_{in} :- received from the other C.U. e.g. supervisory controller.

- Start/stop/timing information.

c''_{in} & c''_{out} : synchronize with other C.U.

Behavior of C.U :-

Flowchart/ Description language or both

- Once a D.P.U. design has been completed, and the control points are identified, each micro operation $Z \leftarrow f(x)$ can be identified with a set of control lines $\{c_{ij}\}$ that must be activated in order to execute that micro operation .

- formal specification of the i/o behavior

Implementation methods :-

1) Hardwired C.U.: sequential logic circuit, which generates specific sequences of control signals.
   - minimizes the no. of components.
   - maximizes the speed of operation.

Once constructed, changes in behavior can be implemented by re-designing and physically re-wiring the unit.

=> Lack of structure makes H.C.U. costly to design and debug.

2) **Microprogramming** :-

- Maurice V. Wilkes (1950)
- flexible & systematic

Activate control lines $\{c_{i,j}\}$

- A (micro)instruction stored in a special addressable memory called a Control Memory (C.M.).

Microprogram:- sequence of micro instructions (MI) needed to execute a particular operation.

- fetch the MI from C.M. one at a time, and use them to activate the control lines directly.
- Micro-programmed control unit
- firmware
- design changes can easily be made by altering the contents of the C.M.

Emulation:- A micro-programmed CPU can execute programs written in the M/C language of several different computers.

Limitation:- Costly than H.C.U due to C.M. & its access circuitry.
Slower due to fetch from C.M.

Since the improvements in memory technology, it has become a standard method of designing C.U.

Hardwired control :-

Trade-off :- Amount of h/w , speed and cost.

Design methods in practice are ad hoc, heuristic, cannot easily be formalized (due to large no. of instructions, & complex interdependency).

Three main methods (suitable for small C.U like R I S C) :-

1) State- table method
2) Delay-Element method
3) Sequence counter method

No one can win over other

State-table Method :-

-like any finite-state sequential m/c

	Input Combinations c_{in}			
State	I_1	I_2		I_m
S_1	$S_{1,1}$,$Z_{1,2}$		. . . . .	$S_{1,m}$,$Z_{1,m}$
S_2	$S_{2,1}$,$Z_{2,1}$		. . . . .	$S_{2,m}$,$Z_{2,m}$
.	.			.
.	.			.
S_n	$S_{n,1}$,$Z_{n,1}$			$S_{n,m}$,$Z_{n,m}$

C_{in}: Input

C_{out} : output

Row : set of internal states $\{S_i\}$

Internal state: information stored at discrete point of time

Column: Set of external signals to C.U.

Entry in row S_i and column I_j : $S_{i,j}$, $Z_{i,j}$

$S_{i,j}$: next state of C.U

$Z_{i,j}$: set of output signals $Z_{i,j}$ from C_{out} that are activated by the application of I_j to C.U. when it is in state S_i .

Pros: systematic technique for minimizing no. of gates, flip-flops.

Cons:

i)no. of state, i/p condition may be so huge that size of the table & computation time become excessive.

ii) Tends to conceal useful info about circuits behavior e.g. loop

iii) Random structure, debugging and maintenance are difficult

DELAY-ELEMENT METHOD

- Consider the problem of generating the following sequence of computer signals at times t_1, t_2 , ... , t_n using a Hardwired C. U.

t_1 : Activate { $C_{1,j}$ };
t_2 : Activate { $C_{2,j}$ };
. . . . . .
. . . . . .
t_n : Activate { $C_{n,j}$ };

• Suppose initial signal START (t_1) is fanned out to $\{c_{1,j}\}$ to perform first micro operation.

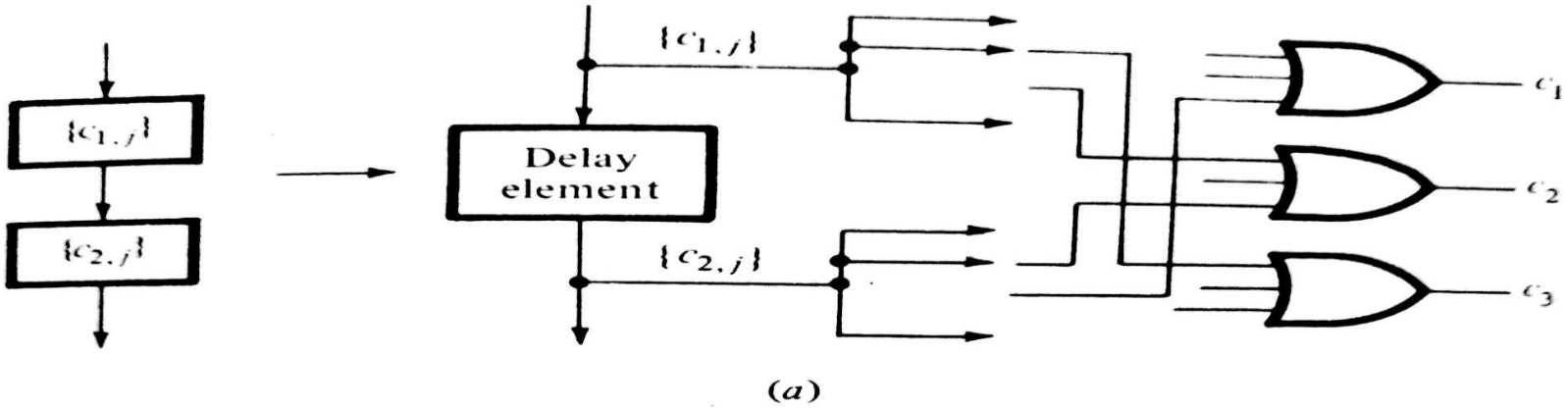
If START(t_1) is also entered into a time delay element of delay $t_2 - t_1$, the output of that circuit, START(t_2) can be used to activate $\{c_{2,j}\}$.

Likewise, another delay element of delay $t_3 - t_2$, with input START(t_3) can be used to activate $\{c_{3,j}\}$.

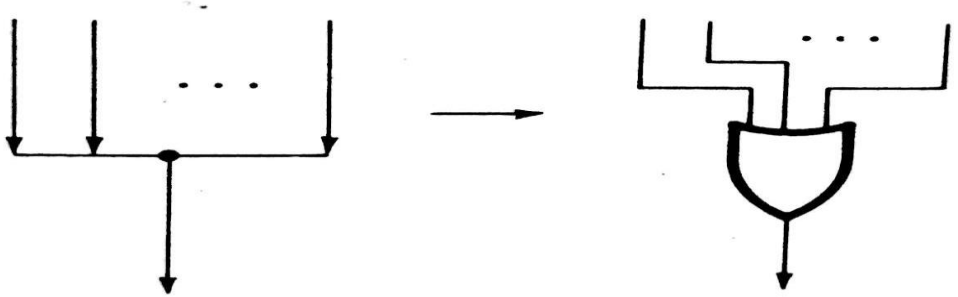
• Sequence of delay-elements.

- D flip-flop is used to ensure synchronous operation.
- Can be constructed from flow-chart.
- Circuit mirrors the flow of control through the flow-chart.

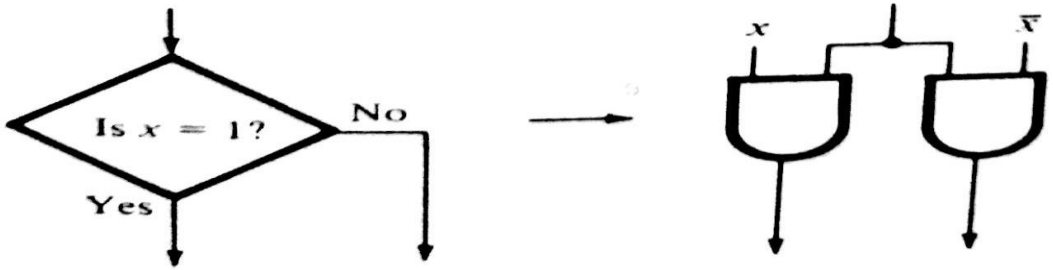
DELAY-ELEMENT METHOD



(a)



(b)



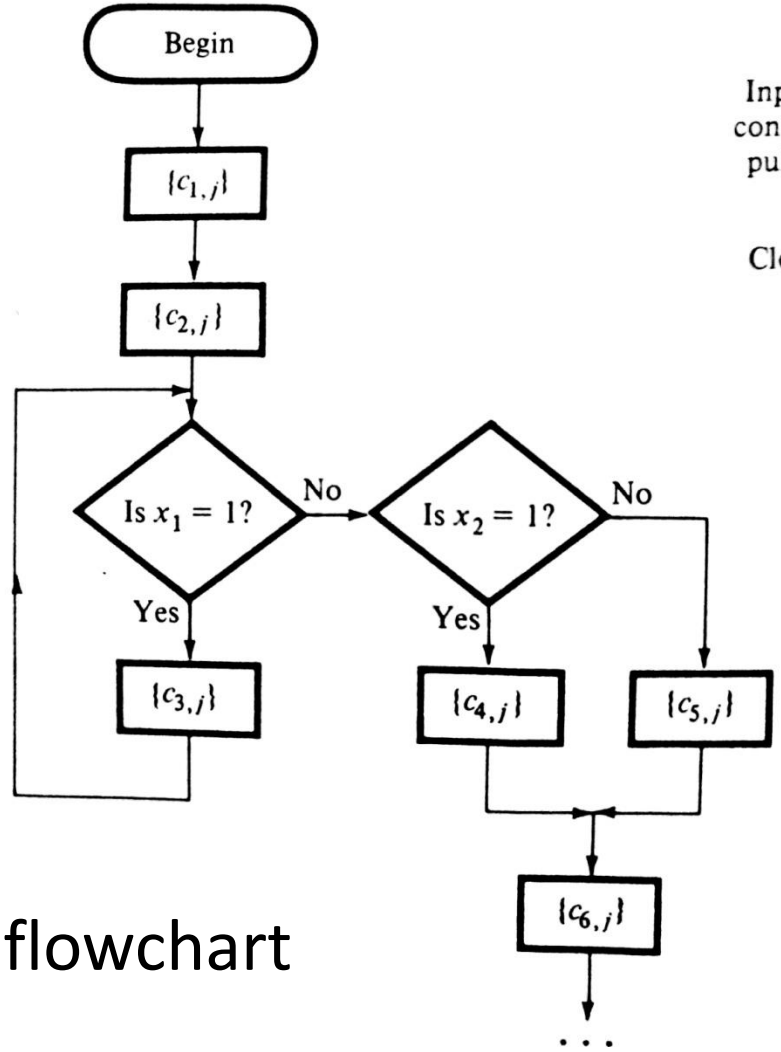
(c)

1. Each sequence of two successive micro operations require a delay element. Signals that are intended to activate the same control line C_i are fed to an OR gate whose output is C_i. This line may then be connected to the control point it activates.

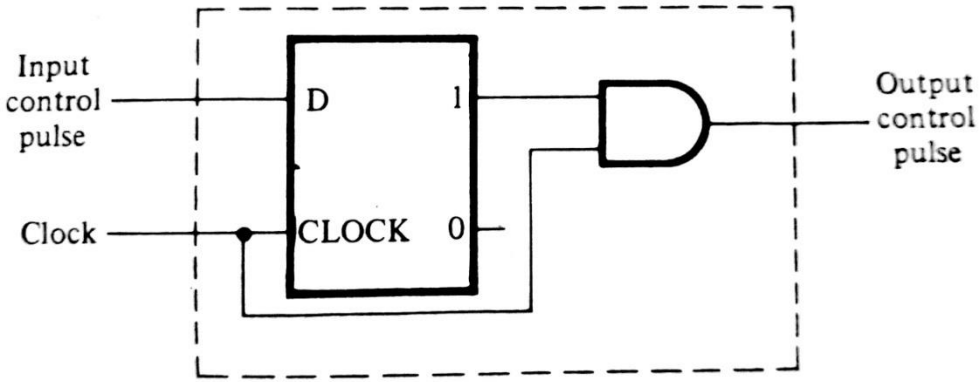
2. K-lines in the flow chart that merge to a common line are transformed into a k-input OR gate.

3. A decision box can be implemented by two AND gates. It forms a 1-bit de-multiplexer controlled by the test variable x (or a Boolean function f(x)).

An example :



flowchart



If all delays are synchronous then a clocked D-type master-slave flip-flop can be used.

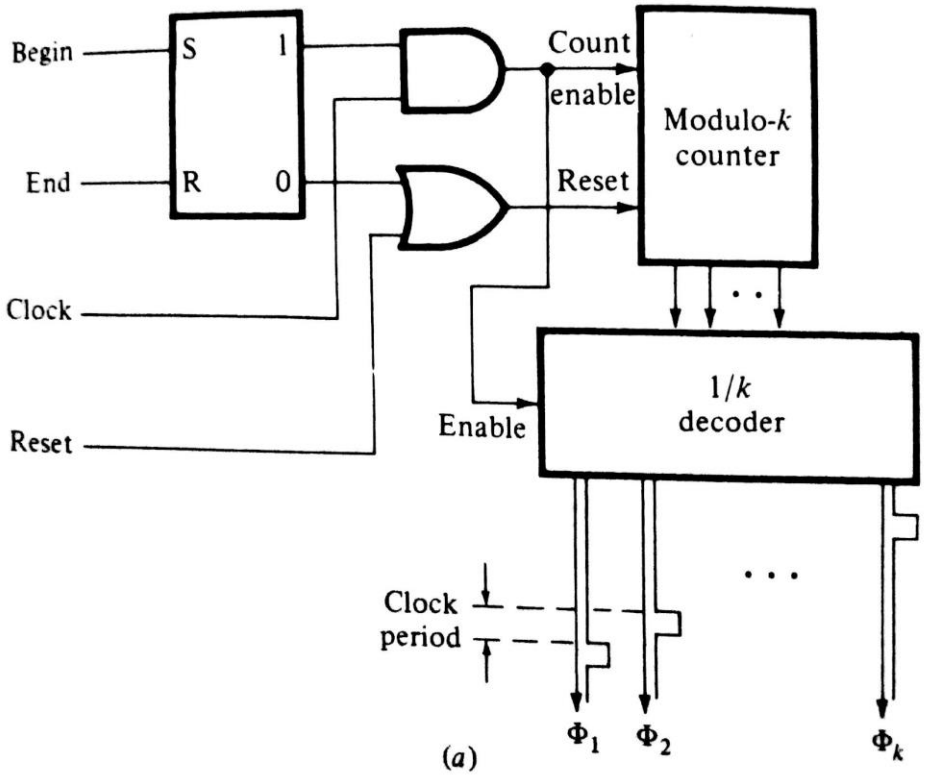
- Complex circuit may be required if an input becomes significant amount of out of phase with the clock due to propagation delays between delay elements.

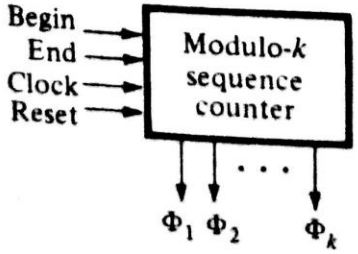
Cons: 1) No. of delay elements needed is approx. equal to the no. of states.

2) each delay element is a sequential circuit of equal or greater complexity than a flip flop.

3) Synchronization becomes difficult.

Seauence Counter Method





(b)

FI

Modulo-k sequence counter whose o/p is connected to 1/k clocked decoder.

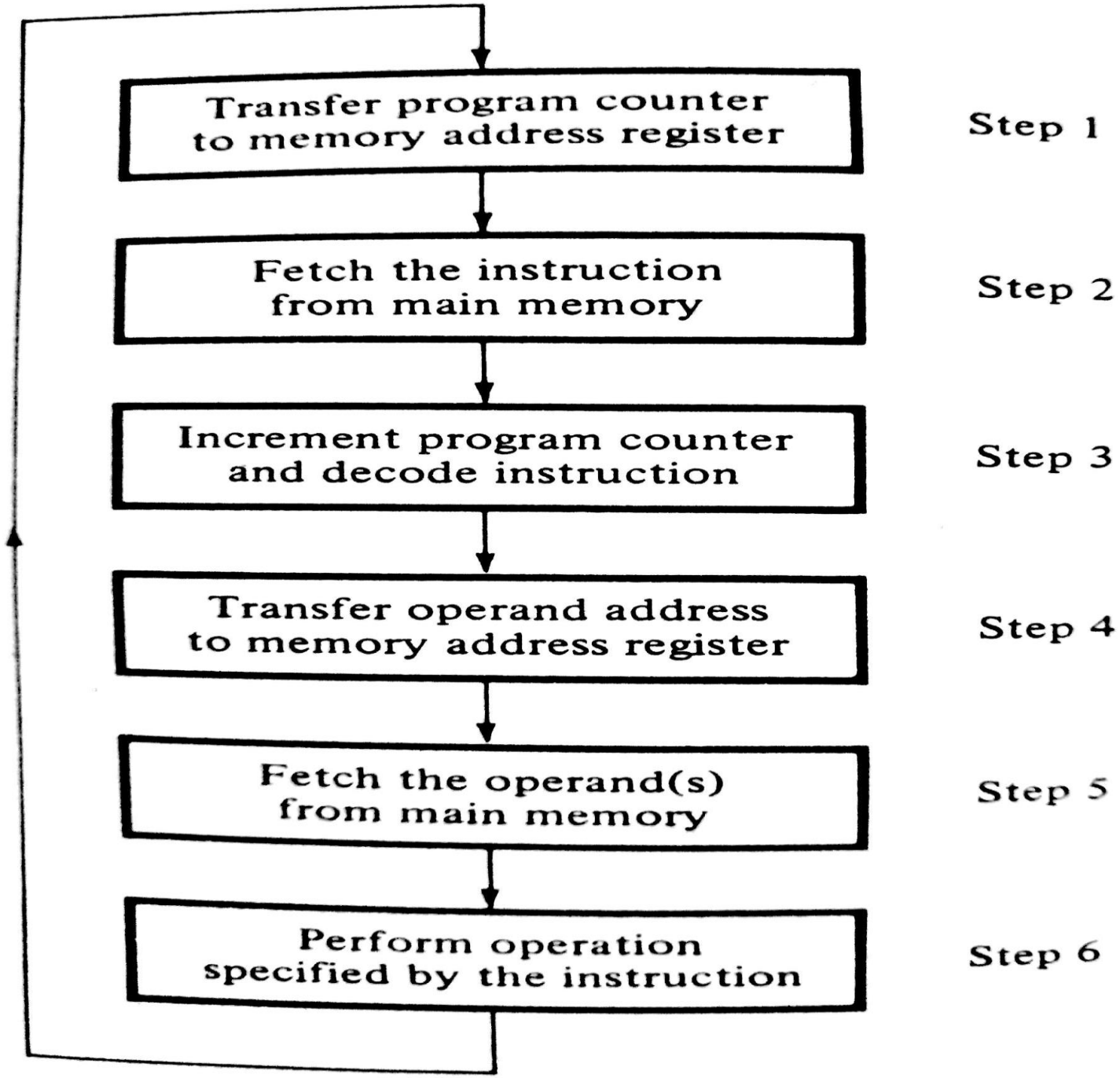
- If the count enable input is connected to a clock source, the counter cycles continually through its k states.
- The decoder generates k-pulse signals $\{\Phi_i\}$ on its output lines, separated by one clock period.
  - called phase signals.
- Two additional input lines and a flip-flop are provided for turning the counter on and off.

USEFULLNESS :- Many digital circuits are designed to perform a relatively small number of actions repeatedly. This type of behavior can be described (usually at a fairly high level) by a flowchart consisting of a single closed loop containing k-steps.

Example

CPU behavior represents as a :

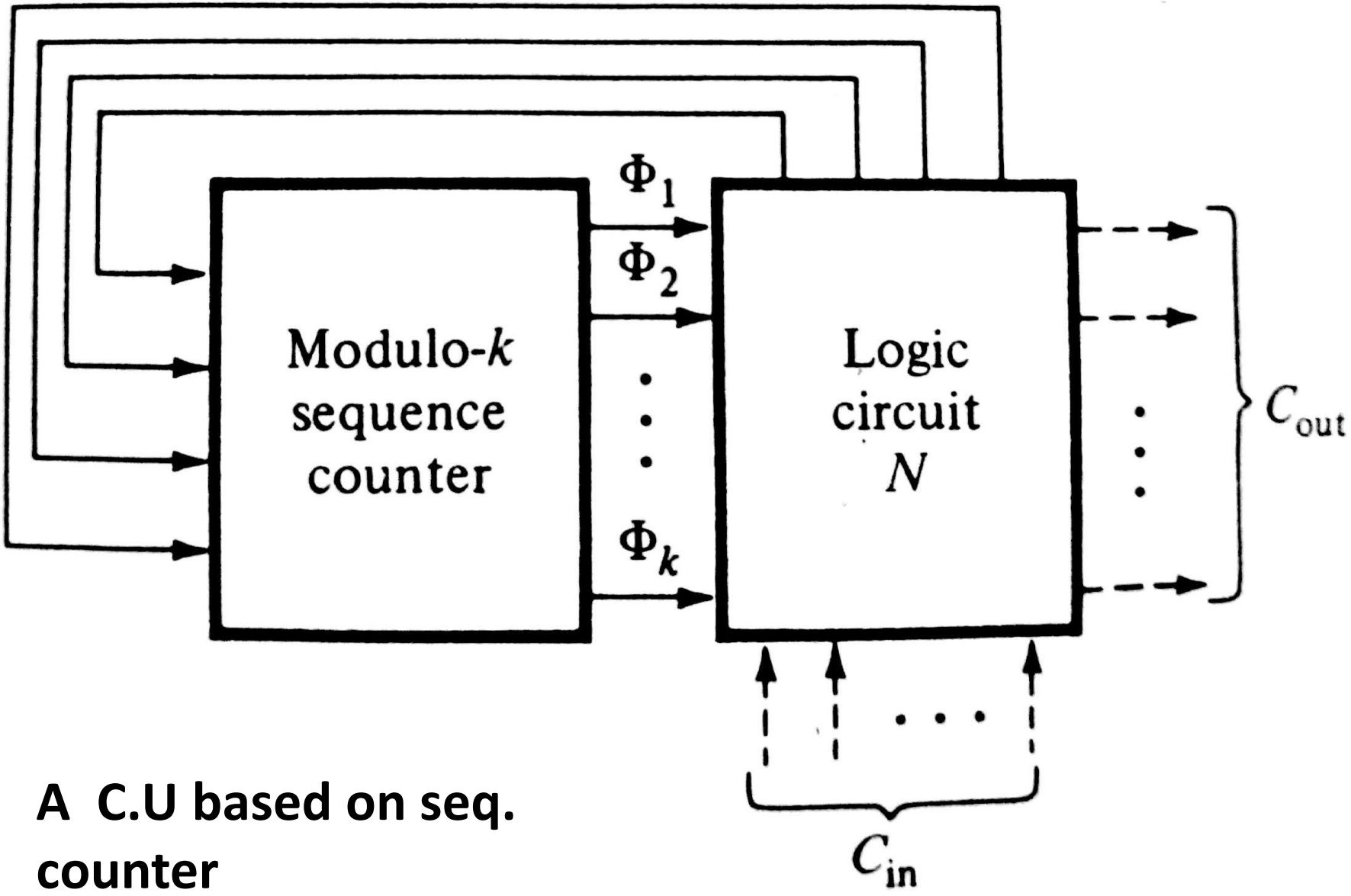
- each pass through the loop: 1 instruction cycle
- each step, 1 clock period(approx)
- modulo-6 sequence counter
- each signal Φi activates some set of control lines in step i of every instruction cycle
- single closed loop


Transfer program counter to memory address register
Step 1
Fetch the instruction from main memory
Step 2
Increment program counter and decode instruction
Step 3
Transfer operand address to memory address register
Step 4
Fetch the operand(s) from main memory
Step 5
Perform operation specified by the instruction
Step 6

It is required to vary the operations performed in step i depending on certain control signals

$$C_{in} = \{ C'_{in}, C''_{in} \}$$

A logic circuit N combines C_{in} with the timing signals $\{\Phi_i\}$ generated by the sequence counter.

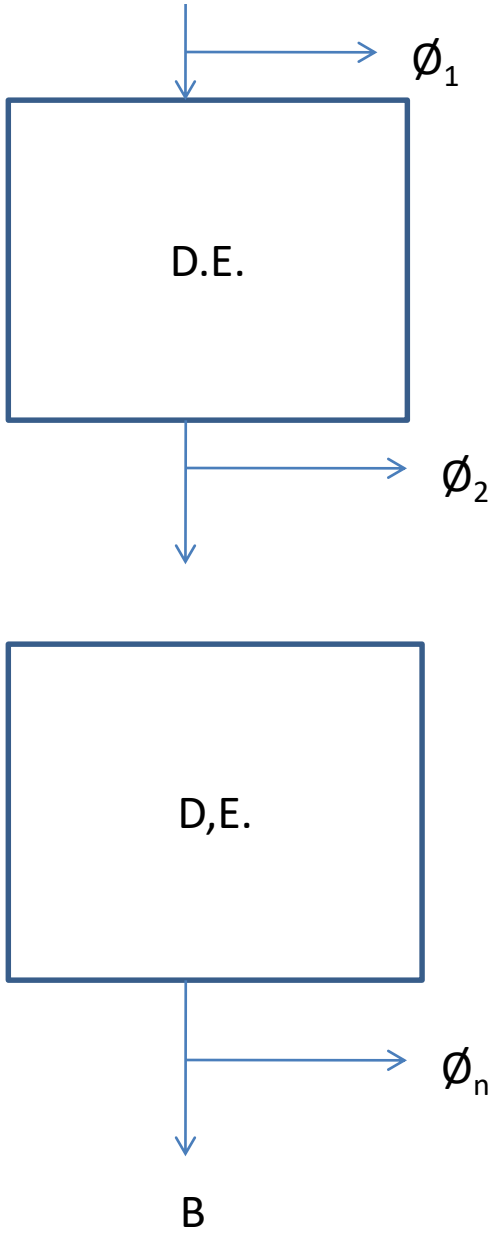


A C.U based on seq. counter

Relationship :- Seq. counter and Delay-element .

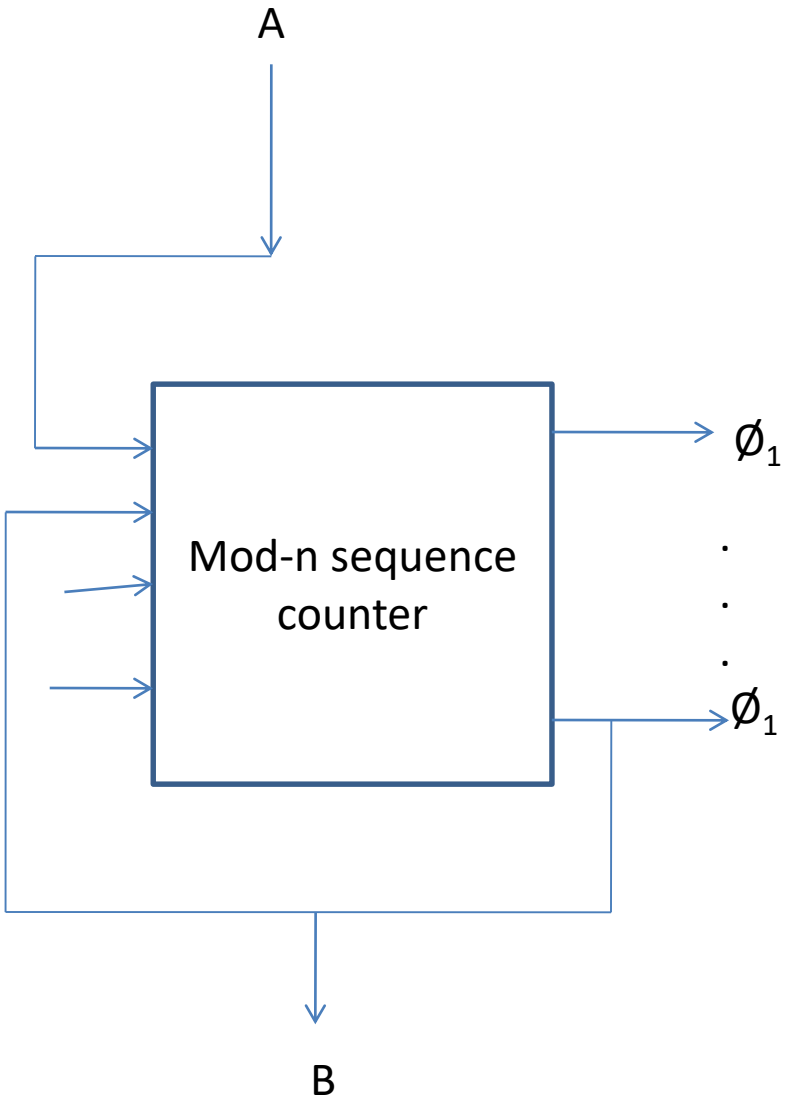
- Modulo-k seq. counter -> cascade of k-s delay elements

- connect k^{th} output line Φ_k to the end line, so that counter shuts itself of after one complete cycle.



:D.E. cascade

Begin
End

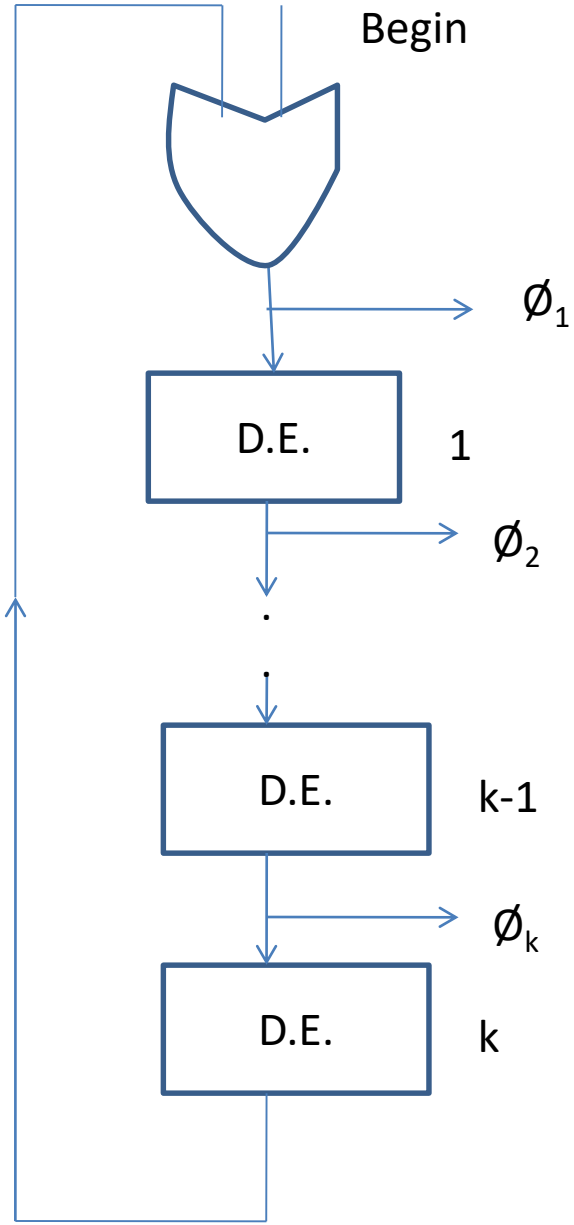


:Equivalent seq counter

- Conversely a cascade of k-1 D.E.s can be made to behave like a seq. counter by connecting its output to its input via an additional D.E. and an OR gate

- Modulo –k ring counter

- A cascade of identical D.E. is essentially a shift register.



A D.E. circuit that behaves like a seq counter.

MICROPROGRAMMED CONTROL

- Every instruction in a CPU is implemented by a sequence of one/more set of concurrent micro operations.

- Each micro operation is associated with a specific set of control lines which, when activated, cause that micro operation to take place.

- Since the no. of instructions and control lines are huge, H.C.U. could be exceedingly complicated.

- Information stored in ROM/RAM, called Control Memory (CM).

- Control signals to be activated at any time are specified by a microinstruction, which is fetched from CM.

- Each microinstruction specifies (explicitly / implicitly) the next microinstruction to be used.

- Sequencing

- A set of related microinstructions: **micro program**.

- Relatively easy to change the design.

- A micro programmed computer C1 used to execute programs written in m/c language L2 (other m/c C1) by placing an emulator for L2 in the C.M. of C1.

- C1 is then said to be capable of emulating C2

- Comparable with assembly language, but requires more detail knowledge of the processor h/w.

- Symbolic language->micro assembly language

- Micro assembler

Wilkes' Design

Microinstruction:

1. Control Field: indicates the control lines to be activated.

2. Address Field: indicates the address in the CM of the next microinstructions to be executed.

Ki,Ci: if Ki=1,Ci is activated

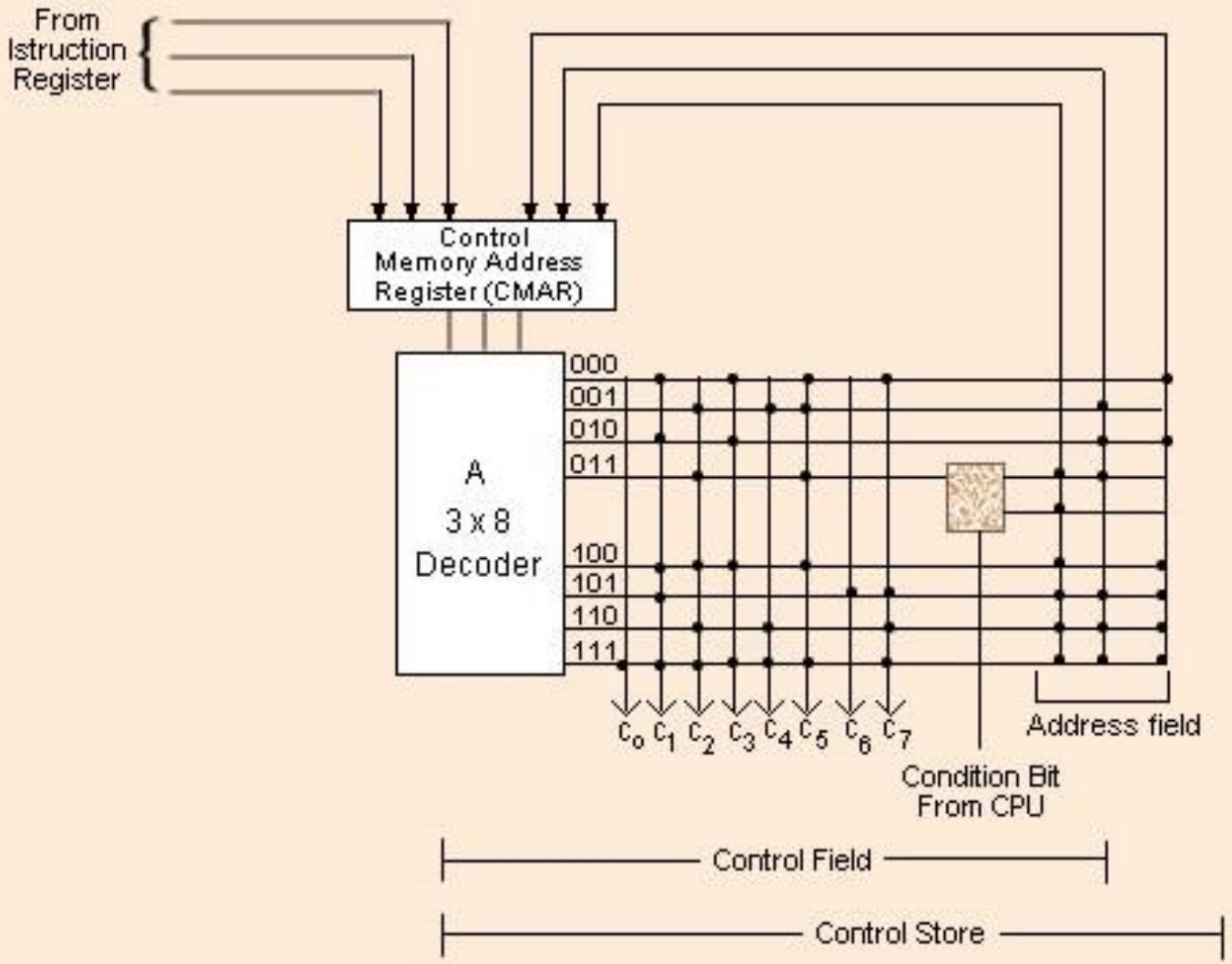


Figure 1: Wilkes Control Unit

CM: Row-> microinstruction

Col -> Control/Address lines

CMAR -> control memory address register which stores the address of the current instruction.

- May be loaded from an external source as well as address fields

- External source provides the starting address micro program stored in the CM

- CM is organized as a program logic array like matrix made of diodes, a simple electronic device. This is partial matrix and consists of two components, the control signals and the address of next microinstruction.

- CMAR can be loaded by the IR or by the address field of CM. On getting an input from the IR, CM provides a 3 bit address to the 3 x 8 decoder. This is an entry point address to the CM.

- On the basis of this address, decoder activates one of the eight output lines (horizontal). This activated line, in turn, generates control signals and the address of the next microinstruction to be executed.

- This address is once again fed to the CMAR resulting in activation of another control line and address field. This cycle is repeated till the execution of the instruction is achieved.

- Ability to respond to external signal

- S(a flip-flop)->external condition

- Conditional jump: One of the two possible address fields to be selected.

Area of concern: word length of microinstruction (influence the cost and size of CM)

1. Max no. of simultaneous micro operations that must be specified (i.e., degree of parallelism)
2. The way control information is represented
3. The way the next microinstruction address is specified

Control Memory – (early design)

ROM

ROS (Read Only Store)

- Because ROM (i.e. diode) provides faster access than RAM (i.e. ferrite core)
- microprogrammed processor was viewed as permanent
- except for correction or minor change (enhancement)

Wilkes Idea: Writable CM (WCM)

- allows instruction set to be changed by changing the microprograms that interpret the instruction opcode

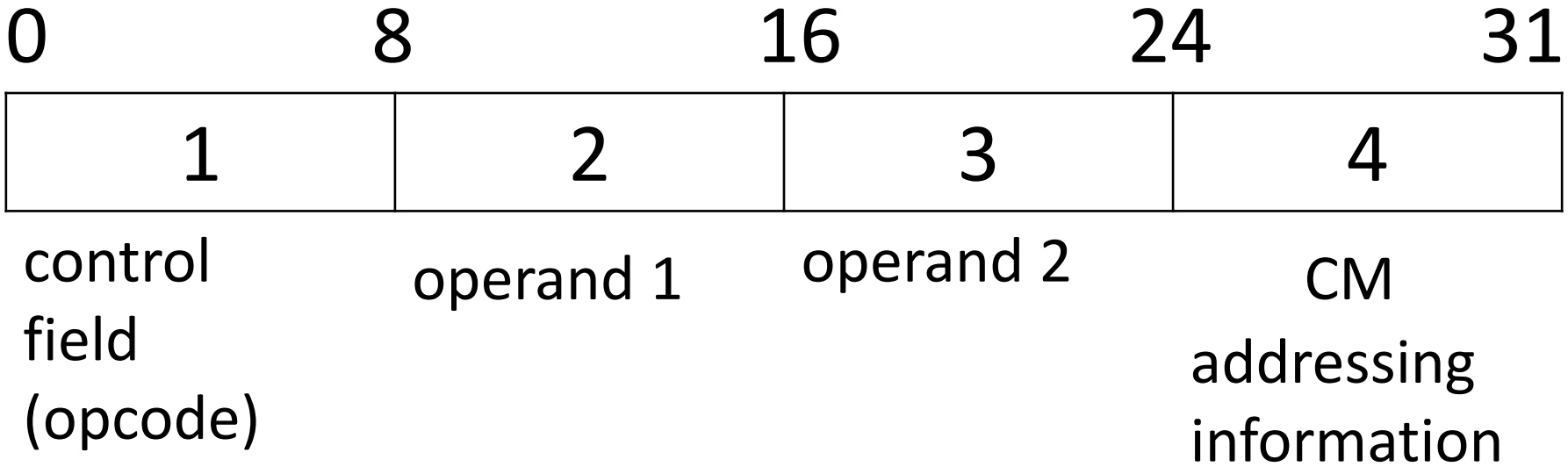
- It can provide same machine with different instruction sets which may be tailored for specific applications.

- Computer with WCM: - no instruction set in usual sense

- dynamically microprogrammable

Parallelism in Microinstructions:-

Micro-programmable processor characterized by the max. no. of micro-operations that can be specified by a single microinstruction (several hundred)



Microinstruction format IBM sys./370 Model 145

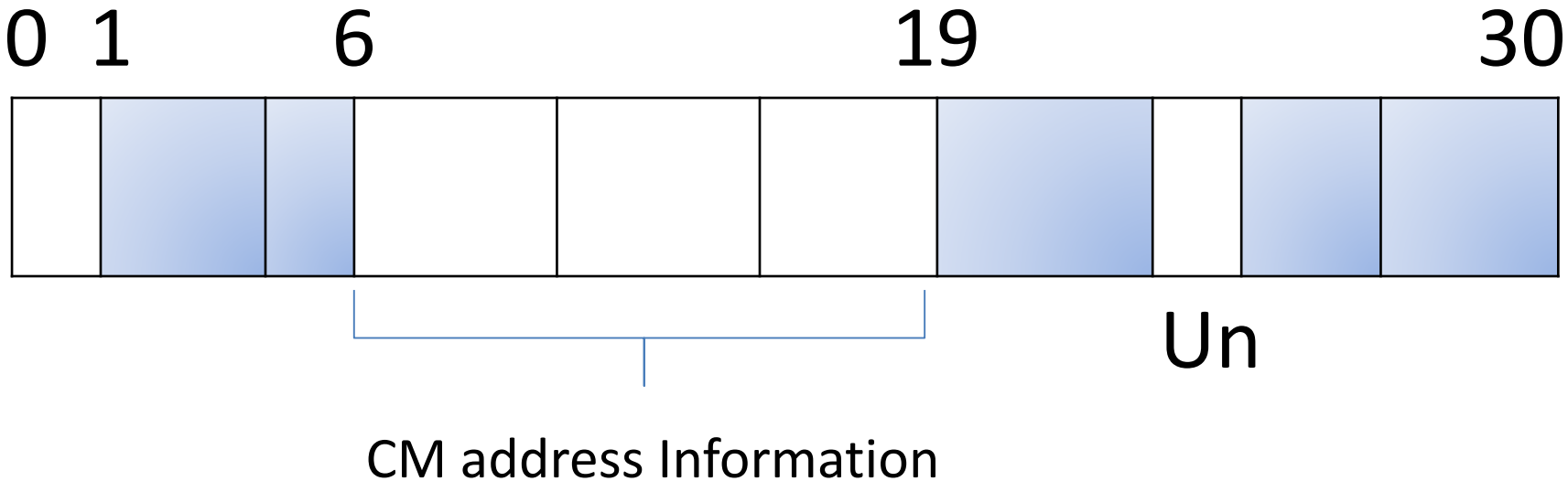
1: operations to be performed

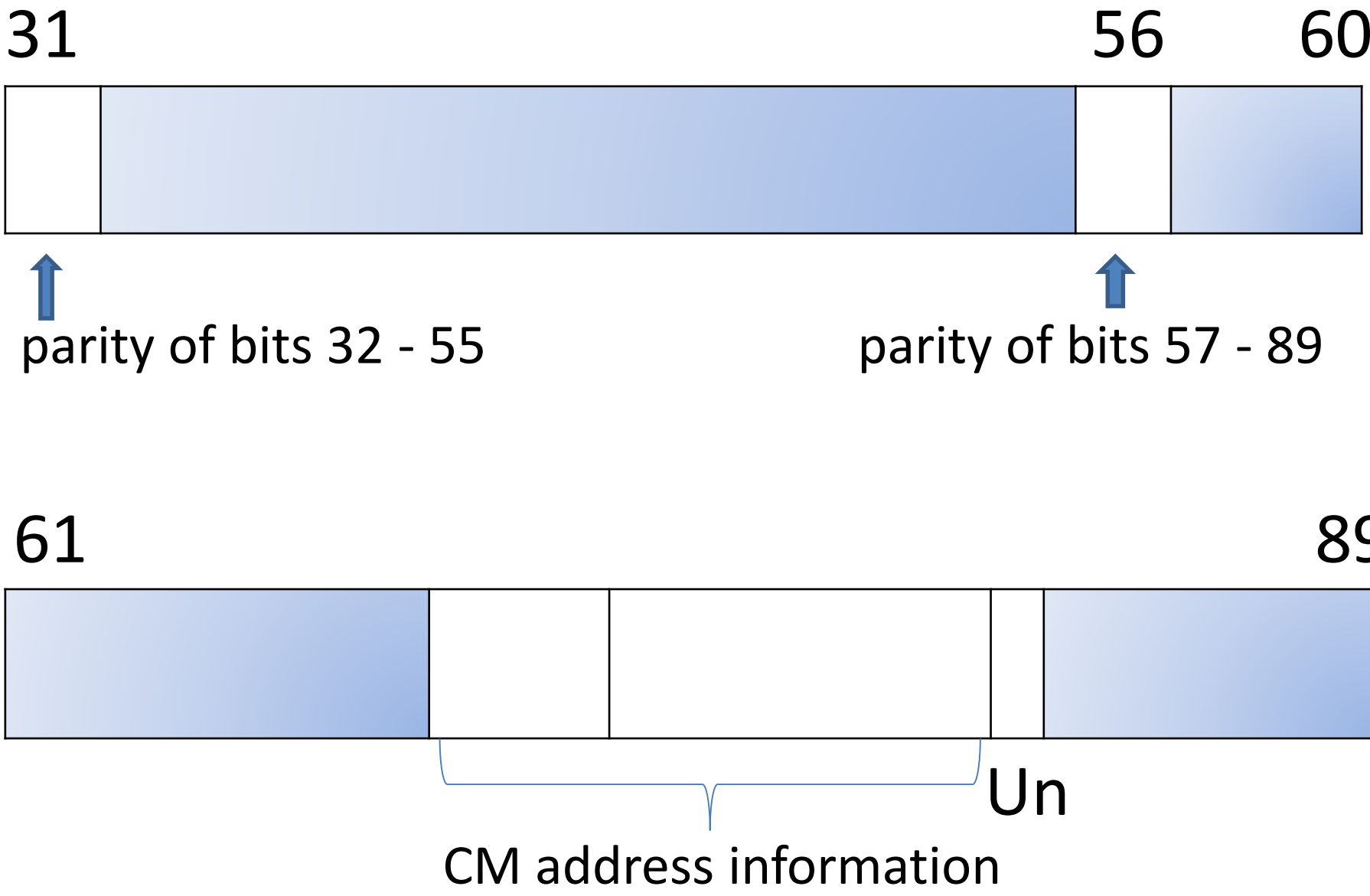
2, 3: mostly addresses of CPU registers

- If all combination of parallel micro operations were specified by single opcode, the no. of opcode would be enormous.
- Decoder circuit was needed then (added complexity)

Sol^{n} :- Divide the microinstruction specification port into k disjoint control fields.

Each control field is associated with a set of micro operations anyone of which can be performed simultaneously with the micro operations specified by the remaining control fields.




31
56
60
parity of bits 32 - 55
parity of bits 57 - 89
61
89
Un
CM address information

- 90-bit microinstruction format of IBM sys/360 Model 50 (shaded areas are control fields)

- e.g. 3-bit control fields 65-67 controls the right input to the CPU adder. This field indicates which of the several possible registers should be connected to the right input of the CPU adder.

- 68-72 define different operations of adder such as binary or decimal.

- 21 different control fields : according to purpose

- The scheme in which (fig 4.30) there is a control field for every control line is wasteful of CM space, since many combinations of control signals are never used.

- In general, any n independent control signals can be encoded in a control field of $[\log_2(n+1)]$ bits, assuming that it is necessary to specify a no-operation condition when no control signals are to be activated.

Un-encoded format: Control signals may be derived directly from the microinstruction.

Suppose MI is loaded into CM data register (called *microinstruction register*: µIR). The outputs of the control part of µIR are the control lines.

When encoded fields are used, each control field must be connected to a decoder from which control signals are derived.

Horizontal Microinstructions:-

1) Long formats

2) Ability to express a high degree of parallelism

3) Little encoding of the control information

Vertical Microinstructions:-

1) Short formats

2) Limited ability to express parallel micro operations

3) Considerable encoding of the control information

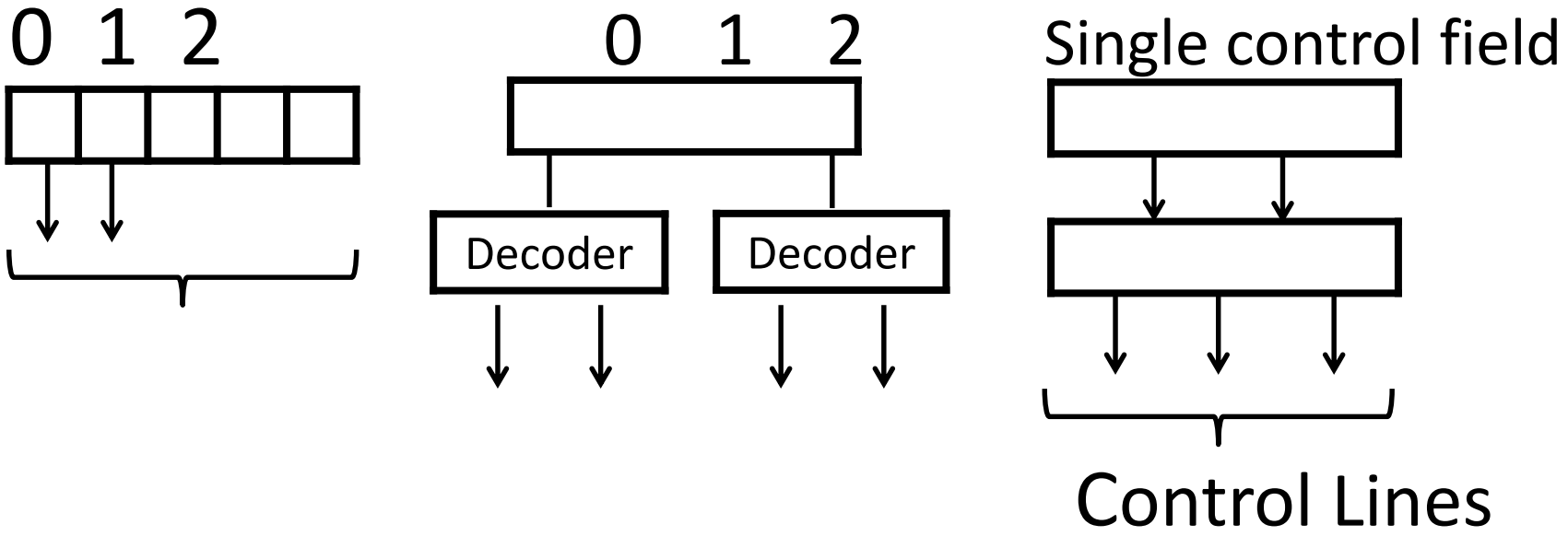
A horizontal microinstruction format allows no ending of control information, whereas a vertical form does.

A vertical microinstruction can specify only one micro operation(no parallelism), while a horizontal one can specify many micro operations.

These definitions are not entirely independent, since a large amount of parallelism implies little encoding and vice versa.

Not entirely independent

control fields



:No encoding :Some encoding :Complete Encoding

Microinstruction Addressing:

- Each microinstruction contains the CM address of the next microinstruction to be executed.
- For branching, two possible next address are included.

Advantage: No time lost in address generation

Limitation: Wasteful of CM space

Solution : microprogram counter (µPC) for branching

  - Analogous to PC at instruction level

– Since only instructions have to be fetched from CM, the µPC is also used as CM address register.

Conditional Branching :

- The condition to be tested is a *condition variable* or *flag* generated by the DPU.

- If several such conditions exist, a *condition select* subfield is included in the microinstruction.

- The branch address may be contained in the microinstruction itself, in which case it is loaded into CMAR, when a branch condition is satisfied (to save space CM can load only low-order bits)

Another Approach:-

Allow the condition variables to modify the contents of the CMAR directly, thus eliminating wholly or partly the need for branch addresses in microinstructions.

Example: overflow when v=1

No overflow when v=0

Skip on overflow: logically connect v to *count enable input* of µPC at an appropriate point in the microinstruction cycle. Allows overflow condition to increment µPC an extra time, performing skip operation.

Micro operation Timing :-

- Each MI(microinstruction) generates a series of CS which are active for duration of the M.I.'s execution cycle.

- A single clock pulse synchronizes all control signals is called Monophase.

- The number of MIs needed to specify a particular operation can often be reduced by dividing the MI cycle into several sequential phases(Polyphase).

- Each control signal is active during one of the phases.

- Added complexity: As requirement of specifying the phase during which a control signal is to be activated.

Example: Register Transfer operation

$R \leftarrow f(R_1,R_2)$ R can be R_1 or R_2

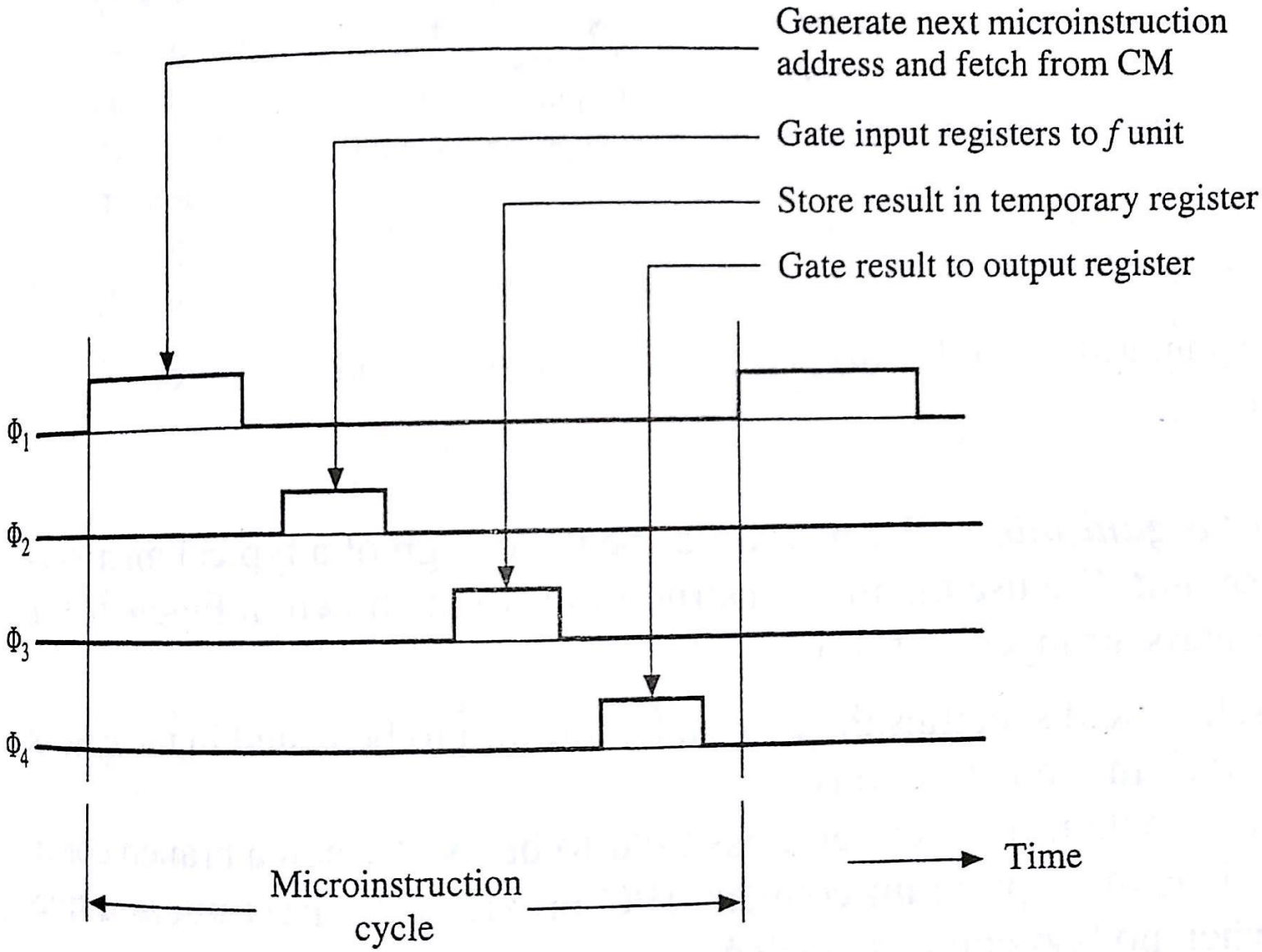
$\leftarrow$

Phase Ø1: Transfer the contents of the R_1 & R_2 to the inputs of the f unit

Phase Ø2: Store the result in a temporary register/latch (L).

Phase Ø3: Transfer the contents of L to destination register R

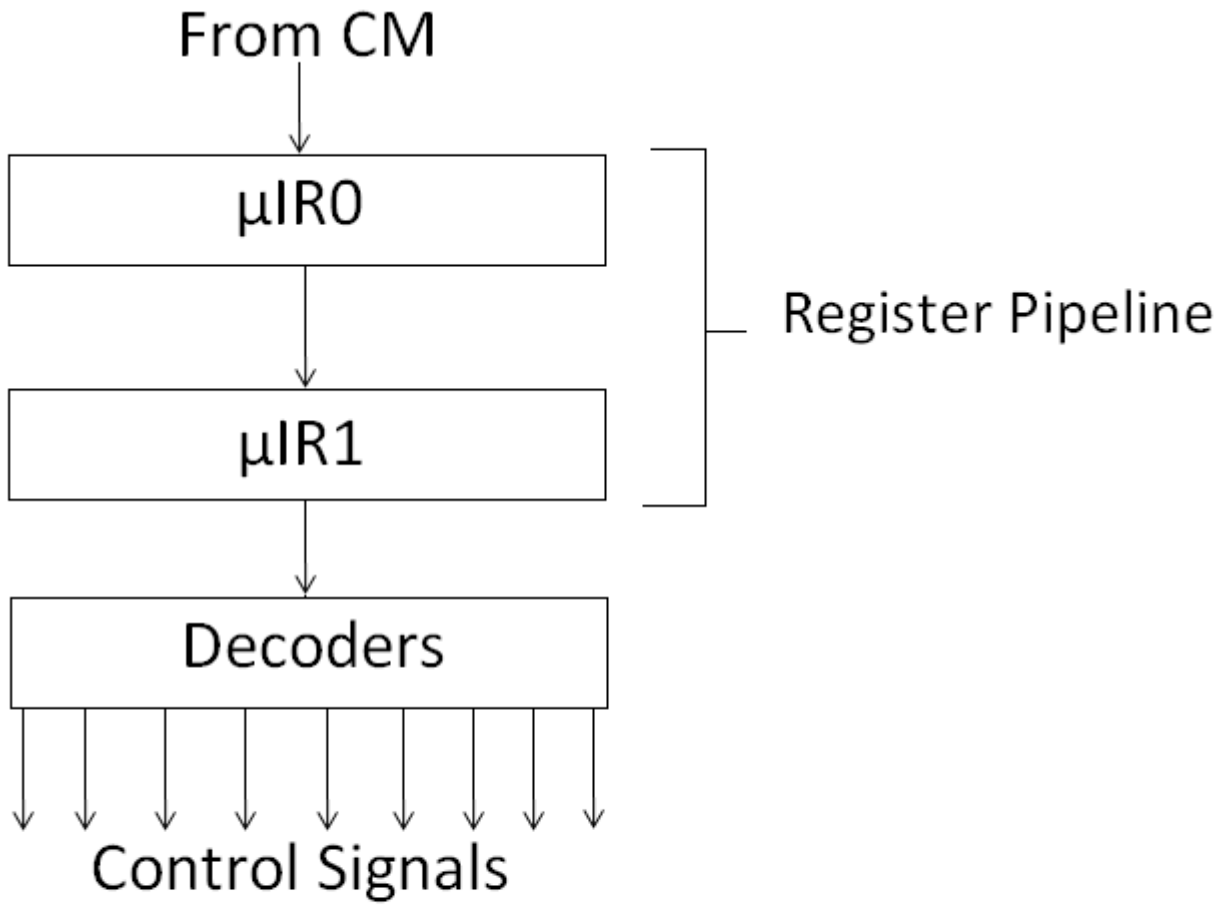
Phase Ø4: Fetch the next MI from CM.


Generate next microinstruction address and fetch from CM
Gate input registers to f unit
Store result in temporary register
Gate result to output register
Φ1
Φ2
Φ3
Φ4
Time
Microinstruction cycle

Micro operation Timing

- Fetch & execute steps can be overlapped

- Replace µIR by a pair of registers forming a two-segment pipeline, while one micro instruction in μIR_1 is being executed, the next micro instruction can be fetched and placed in μIR_0.


From CM
µIR0
µIR1
Register Pipeline
Decoders
Control Signals

Residual Control

Storing the Control field in a register which continues to exercise Control until it is modified by a subsequent MI.

Application of Residual Control

When MI is used to allocate a resource e.g. connection between two units may be established by a MI and maintained for a long period via Residual Control.

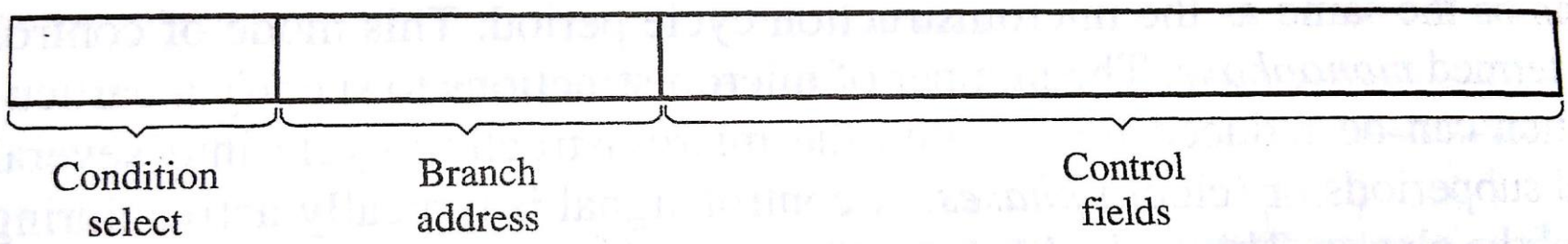
Control Unit Organization

- A Condition select field is used to specify the external condition to be tested in case of conditional branch MI.

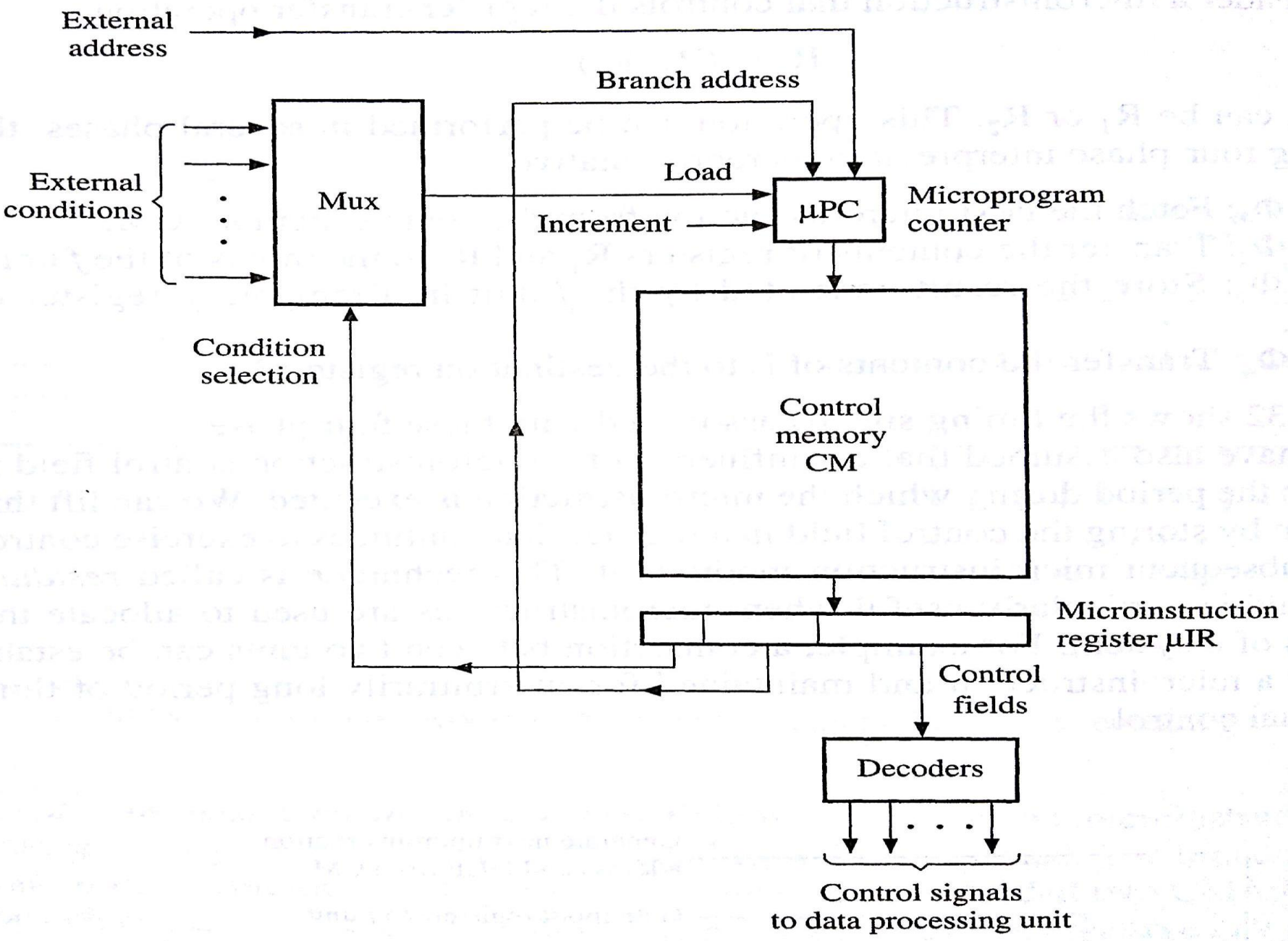
- An Address field contains the next address to be used when a branch condition is satisfied (µPC provides next MI address when no branching)

Control Unit Organization

- Rest specifies (un/encoded) format of the CS that must be activated to perform the desired micro operation.



MI format


External address
Branch address
External conditions
Mux
Load
Increment
μPC
Microprogram counter
Condition selection
Control memory CM
Microinstruction register μIR
Control fields
Decoders
Control signals to data processing unit

- The contents of the addressed word in CM are transferred to the µIR
- Control fields are decoded (if required) and used to generate CS for DPU, µPC is then incremented.
- For branch instruction in µIR, the contents of the MI Address field are loaded in µPC.
- MUX activates parallel-load control i/p of µPC based on status of the external condition variables

s_0	s_1	Meaning
0	0	No branching
0	1	Branch if $v_1 = 1$
1	0	Branch if $v_2 = 1$
1	1	Unconditional branch

MUX: x_0, x_1, x_2, x_3 (4 i/ps)

x_i routed to MUX o/p when $S_0S_1 = i$

- A provision is made for loading µPC with an address from an external source, used for loading the starting address of the desired micro program in cases where CM contains more than one micro program.

Nanoprogrammed Computer

- In some machines, the Micro Instructions do not directly issue the signals that control the hardware.

- Instead they are used to access 2nd control memory, called nanoControl Memory (nCM), that directly control the hardware.

- First used in 1970 by Nanodata Corp.

Nanoprogrammed Computer

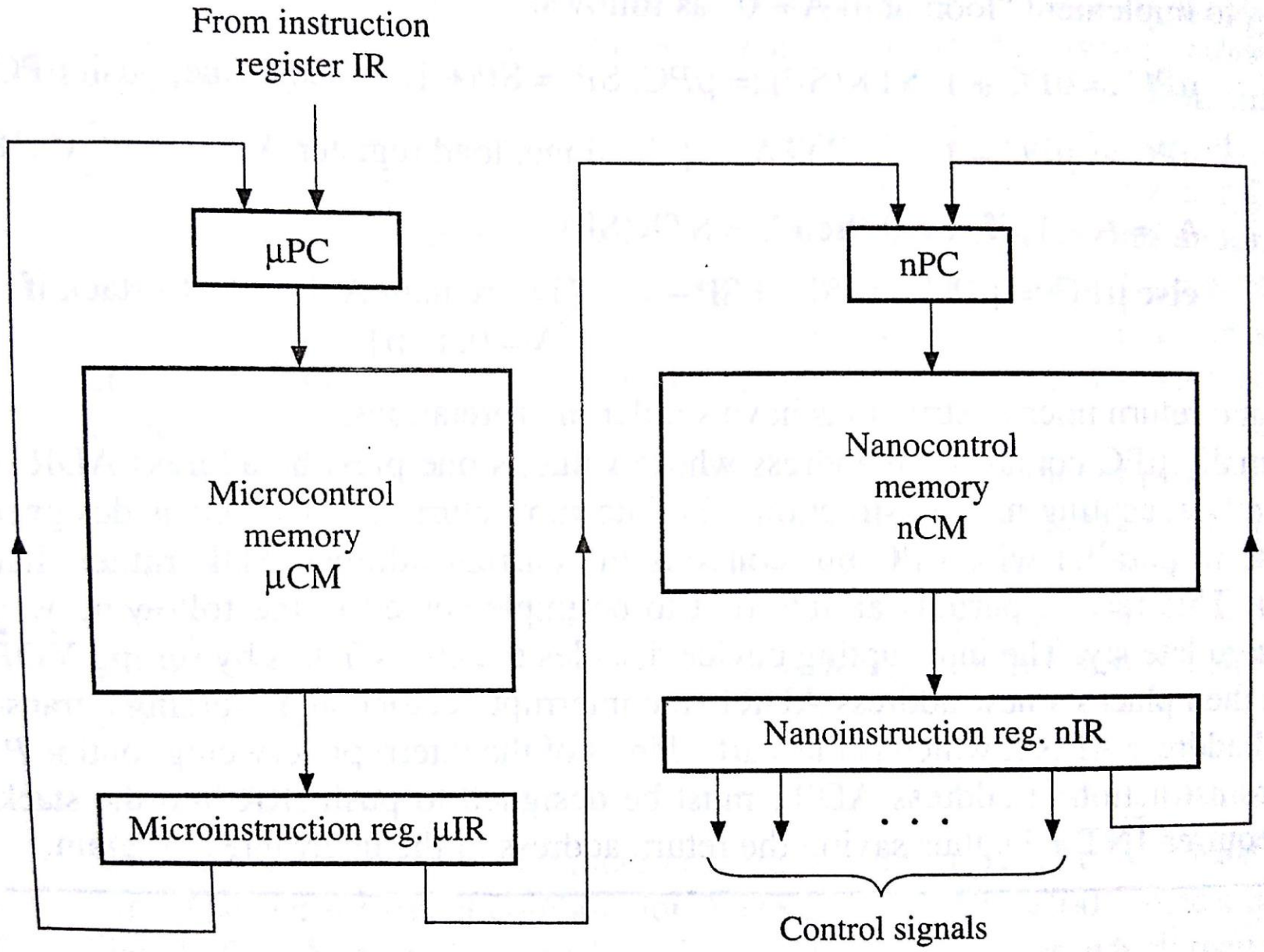
There are two levels of control memories:

1. Higher level: Microcontrol memory µCM (MI)
2. Lower Level: Nanocontrol memory nCM (nanoinstructions)

Consider the dimensions:

µCM: H_m x W_m

nCM: H_n x W_n



Two level control store organization for nanoprogramming

Nanoprogrammed Computer

- Typically, microprograms are encoded in a vertical format, so that although H_m is large, W_m is small.
- Nanoinstruction (NIs) usually have a highly parallel horizontal format making W_n large.
- If many MIs can be interpreted by the same nanoprogram, then H_n can be kept relatively small.

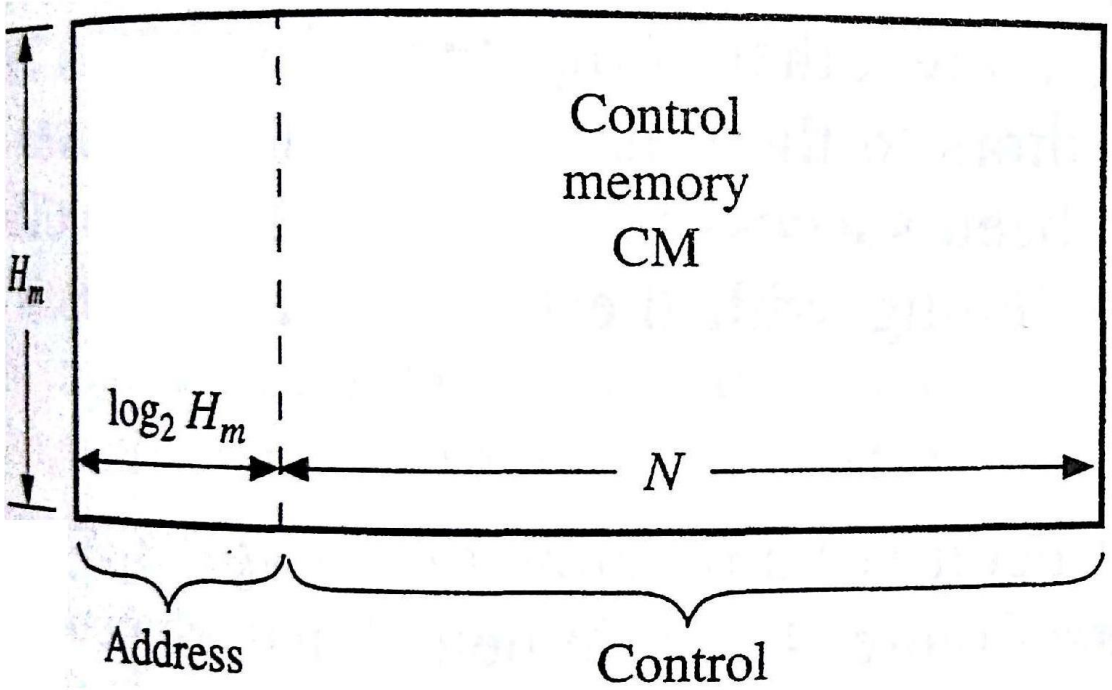
Advantages

- It can reduce the total size of CMs needed, this translates to smaller chip area.

- Greater design flexibility resulting from the loosening the bonds between instruction and hardware with two intermediate levels of control rather than one.

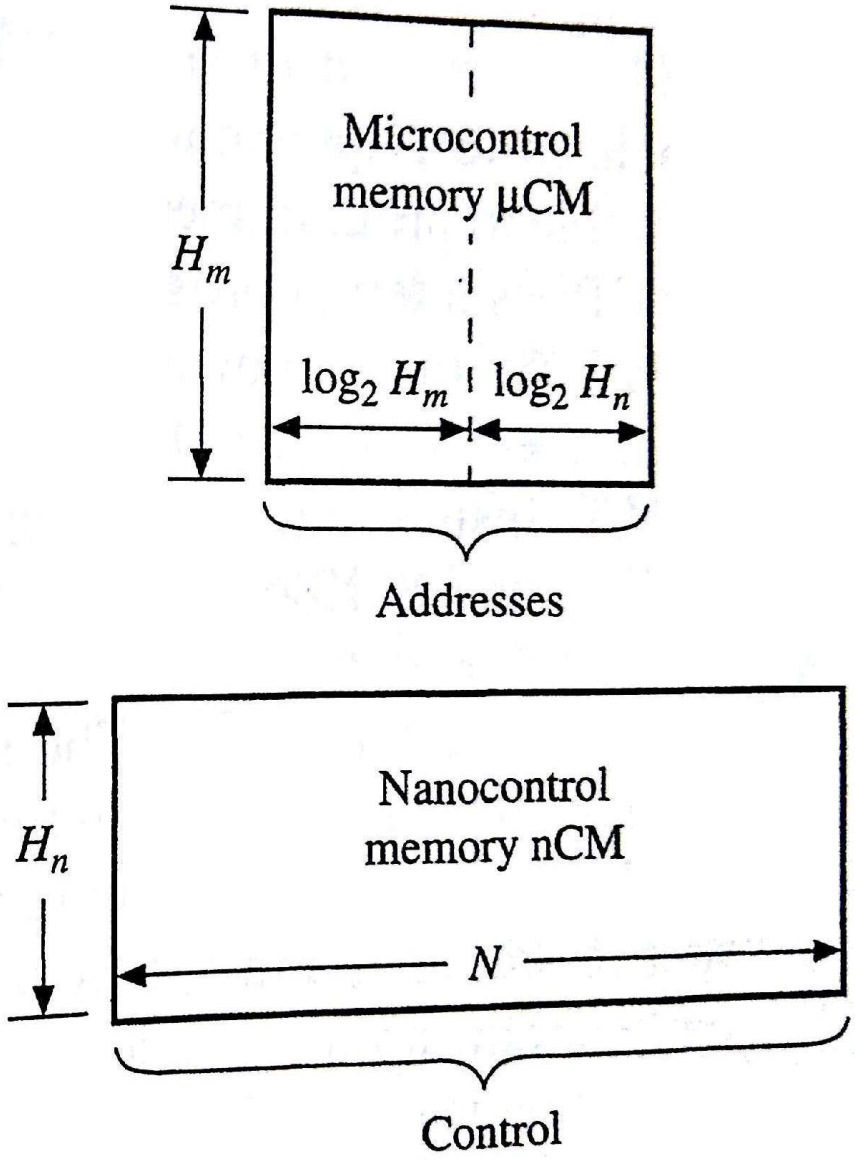
Disadvantage

Reduction in speed due to extra memory access for nCM and a more complex CU organization.



One level

$$S1 = H_m (N + \lceil \log_2 H_m \rceil) \quad \text{.....(i)}$$


Microcontrol
memory μCM
H_m
$\log_2 H_m$
$\log_2 H_n$
Addresses
Nanocontrol
memory nCM
H_n
N
Control

Two level

- In 2-level design, μCM again stores H_m MIs, but N-bit Control fields are transferred to nCM.
- In place of latter, each MI in μCM contains $[\log_2 H_n]$ bit address to specify any NI location in nCM.

Hence size becomes,

$S_2 = H_m ([\log_2 H_m] + [\log_2 H_n]) + NH_n$

(Assuming no branching, so no explicit addressing)

Suppose all control bit patterns are different, it can be written as

$H_n = r\ H_m$

where r = ratio of the number of unique control states to the total number of H_n of control states needed to implement all instructions

So,

$S_2 = H_m\ (\ [\log_2 H_m] + [\log_2 r\ H_m] + rN\ )$

$= H_m\ (\ 2[\log_2 H_m] + [\log_2 r] + rN\ )$ .....(ii)

If N=70, H_m=650, r=0.4 so that H_n=260

S_1 = 52,240

S_2 = 30,550

Consequently, 52450-30550=21850 bits of control storage (42% of S_1) are saved.

In general, 2-level design require less memory space if $S_2 < S_1$

Hence, from equation (i) & (ii), the following inequality must be satisfied

$$N \geq [\log_2 H_m] + [\log_2 r] + rN$$