EC7551: COMPUTER ARCHITECTURE AND ORGANIZATION

UNIT III: CONTROL DESIGN

Presented By,

Dr. V. SATHIESH KUMAR

Assistant Professor

Department of Electronics Engineering

MIT-Anna University

SYLLABUS:

UNIT III CONTROL DESIGN

Hardwired Control, Microprogrammed Control, Multiplier Control Unit, CPU Control Unit, Pipeline Control, Instruction Pipelines, Pipeline Performance, Superscalar Processing, Nano Programming.

TEXT BOOKS:

1. John P.Hayes, _Computer architecture and Organization‘, Tata McGraw-Hill, Third edition, 1998.
2. V.Carl Hamacher, Zvonko G. Varanesic and Safat G. Zaky, — Computer Organization—, V edition, McGraw-Hill Inc, 1996.

Presentation Slides:

www.sathieshkumar.com/tutorials

BASIC CONCEPTS

- Digital system is divided into two parts, namely, a **datapath (data processing) unit** and a **control unit.**
- The **datapath** is a network of functional and storage units capable of performing certain (micro) operations on data words.
- The **control unit** is to issue control signals to the datapath.
- These control signals enter the datapath at "control points" where they select the functions to be performed at specific times and route the data through the appropriate parts of the datapath unit.
- A CPU's datapath contains circuits to perform arithmetic and logical operations on words such as fixed-point or floating-point numbers.

BASIC CONCEPTS

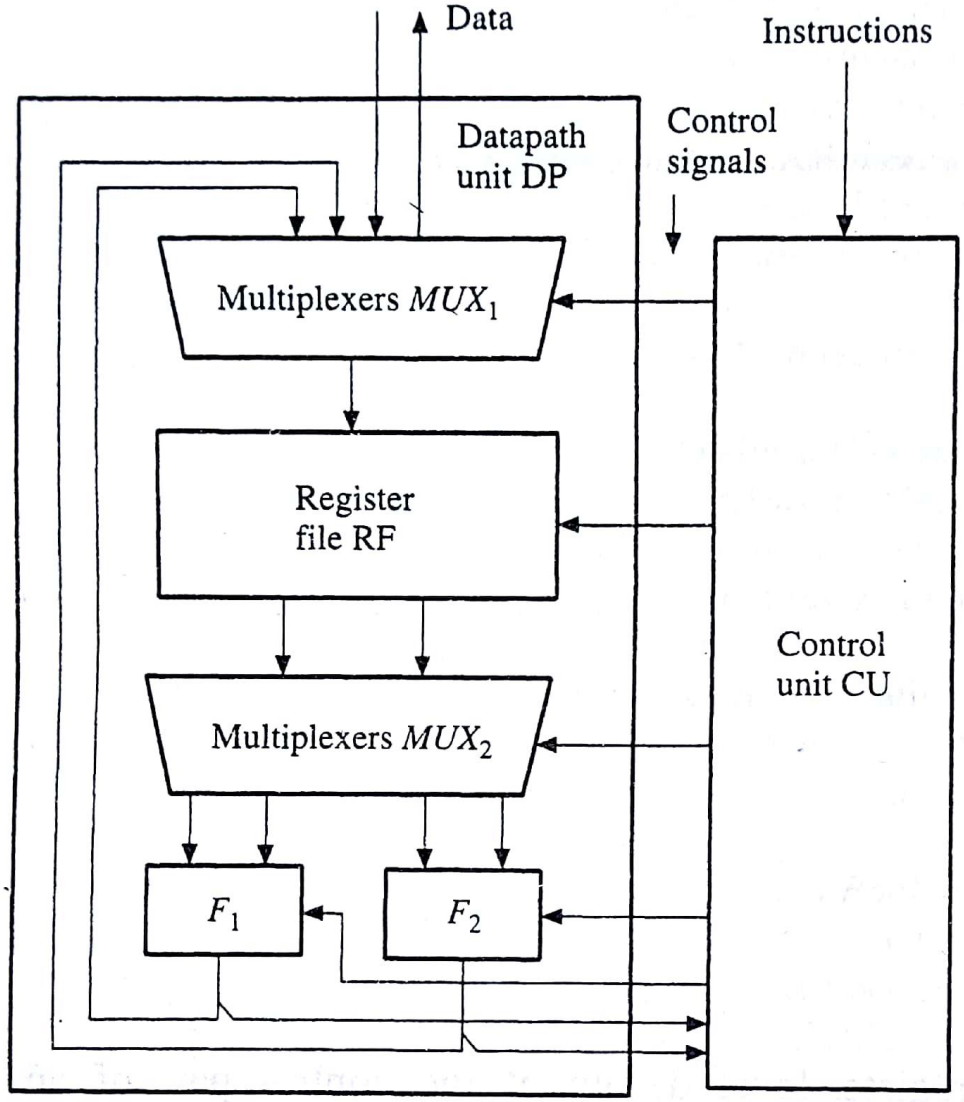


Figure 5.1
Processor composed of a datapath unit DP and a control unit CU.

BASIC CONCEPTS

- It contains a register file RF for temporary storage of operands, two functional units F_1 and F_2 responsible for data processing and multiplexers to allow the data to be steered through DP.
- Typical functional units are an ALU performing addition, subtraction and logical operations; a shifter or a multiplier.
- The control unit CU receives external instructions or commands, which it converts into a sequence of control signals that the CU applies to DP to implement a sequence of register transfer operations.

BASIC CONCEPTS

➢The control signal that implement an addition instruction of the form ADD A,B which we write as, A := A+B.

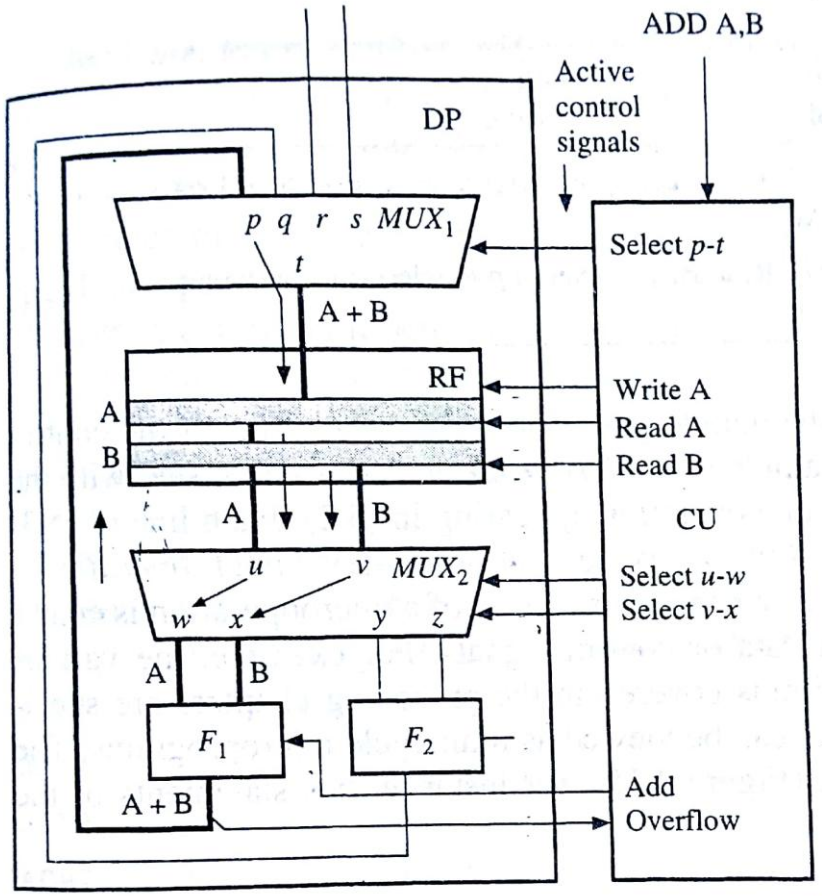


Figure 5.2
The processor of Figure 5.1 configured to implement the add operation A := A + B.

BASIC CONCEPTS

- Assume that this operation can be executed in a single clock cycle.
- Input variables A and B are obtained from registers (RF), and the result is stored back into register A.
- RF permit their contents to be read from and written into in the same clock cycle.
- RF is configured with one input and two output ports to support operations with two or three addresses.
- Appropriate operation to be performed on data is also selected by the CU (F_1's ADD operation).
- Finally the necessary logical connections for the data to flow through DP must be established by applying appropriate control signals to the multiplexers.

BASIC CONCEPTS

➢CU must activate the following three types of control signals during the clock cycle,

1. Function select : Add
2. Storage control : Read A, Read B, Write A
3. Data routing : Select p-t, Select u-w, Select v-x

➢ Usually some feedback of control information from DP to CU to indicate exceptional conditions (overflow) encountered during instruction execution.

BASIC CONCEPTS

➢Multicycle operations : Some instructions require more than one clock cycle for their execution.

➢Eg: double-precision addition can be implemented by a two-instruction sequence (program) of the form,

ADD AL, BL

ADDC AH, BH

Which involves two double-word operands A and B.

Cycle	Function select	Storage control	Data routing
1	Add	Read AL, Read BL, Write AL	Select *p-t*, Select *u-w*, Select *v-x*
2	Add with carry	Read AH, Read BH, Write AH	Select *p-t*, Select *u-w*, Select *v-x*

BASIC CONCEPTS

- This low-level description of the double-precision addition in terms of the control signals to be activated is an example of a microprogram.
- microinstruction specifying a set of low-level microoperations.

BASIC CONCEPTS

➢ **Implementation methods** : Two general approaches to control unit design are Hardwired control and Microprogrammed control units.

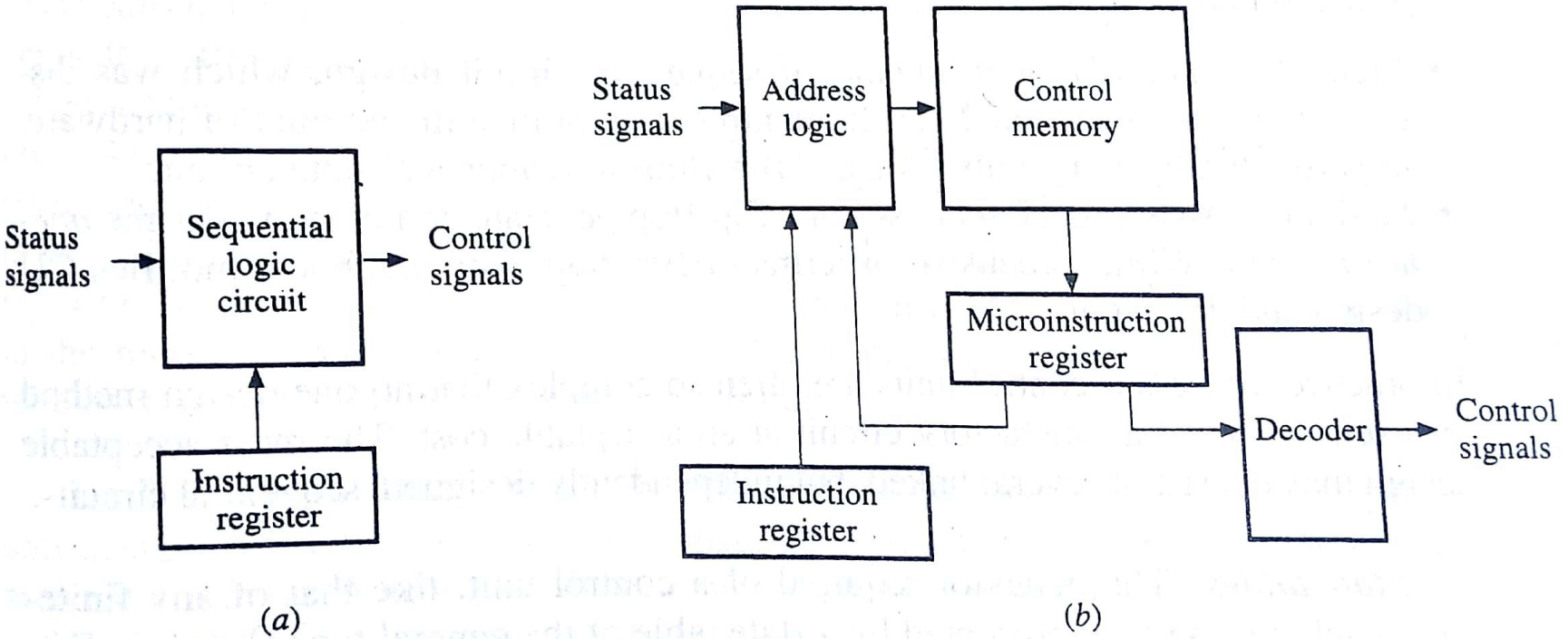


Figure 5.3
General structure of (*a*) a hardwired and (*b*) a microprogrammed control unit.

BASIC CONCEPTS

➢**Hardwired control** – views the controller as a sequential logic circuit or finite-state machine that generates specific sequences of control signals in response to externally supplied instructions.

➢ Designed with the goal of minimizing the number of components used and maximizing the speed of operation.

➢Once the unit is constructed, the only way to implement changes in control-unit behavior is by redesigning the entire unit.

BASIC CONCEPTS

➢**Microprogrammed control** – It is built around a storage unit called control memory, where all the control signals are stored in a program like format.

➢ The control memory stores a set of microprograms designed to implement or emulate the behavior of the given instruction set.

➢Each instruction causes the corresponding microprogram to be fetched and its control information extracted in a manner that resembles the fetching and execution of a program from the computer's main memory.

➢Control signals are embedded in a kind of low-level software – **firmware**.

➢Design changes can be easily made just by altering the contents of the control memory.

➢Are more costly to manufacture than hardwired units due to the presence of the control memory and its access circuitry.

➢ Slower because of the extra time required to fetch microinstructions from the control memory.

HARDWIRED CONTROL

- Fixed logic circuits to interpret instructions and generate control signals from them.
- Trade-offs between the amount of hardware used, the speed of operation, and the cost of the design process.
- Suitable only for small control units such as simple RISC processors or application-specific controllers.
- Method 1 : The classical method of sequential circuit design. It attempts to minimize the amount of hardware, in particular, by using only $[\log_2 P]$ flip-flops to realize P-state circuit.
- Method 2 : An approach that uses one flip-flop per state and is known as the one-hot method. While expensive in terms of flip-flops, this method simplifies CU design and debugging.

HARDWIRED CONTROL : STATE TABLES

	Inputs			
State	I_1	I_2		I_m
S_1	$S_{1,1}, O_{1,1}$	$S_{1,2}, O_{1,2}$	...	$S_{1,m}, O_{1,m}$
S_2	$S_{2,1}, O_{2,1}$	$S_{2,2}, O_{2,2}$	...	$S_{2,m}, O_{2,m}$
			...	
S_n	$S_{n,1}, O_{n,1}$	$S_{n,2}, O_{n,2}$	...	$S_{n,m}, O_{n,m}$

(*a*)

	Inputs				
State	I_1	I_2		I_m	Outputs
S_1	$S_{1,1}$	$S_{1,2}$	...	$S_{1,m}$	O_1
S_2	$S_{2,1}$	$S_{2,2}$	...	$S_{2,m}$	O_2
			...		
S_n	$S_{n,1}$	$S_{n,2}$	...	$S_{n,m}$	O_n

(*b*)

Figure 5.4
State tables for a finite-state machine: (*a*) Mealy type and (*b*) Moore type.

HARDWIRED CONTROL : STATE TABLES

- The rows of the state table correspond to the set of internal states $\{S_i\}$
- These states are determined by the information stored in the machine at discrete points of time (clock cycles).
- Let X and Z denote the input and output variables.
- The columns correspond to the combinations of the X signals that can be applied to the machine and are denoted by $\{I_j\}$.
- The entry in row S_i and column I_j has the form $S_{i,j}$,$O_{i,j}$, where $S_{i,j}$ is the next state of the machine that results from the application of input combination I_j and $O_{i,j}$ denotes the output signals that appear on Z whenever the machine is in state S_i with input I_j applied.
- In general, an entry in the state table defines a specific, one-cycle transition between two states.
- This type of machine is called Mealy machine.

HARDWIRED CONTROL : STATE TABLES

- Slightly different type of state table – Moore machine
- Their output signal values often depend on the current state S_i only and are independent of the input combination I_j.
- A Mealy machine can be converted into a Moore machine, if for every row i, we have $O_{i,j} = O_{i,k} = O_i$ for all j, k=1,2,…,m.

HARDWIRED CONTROL : GCD PROCESSOR

- Greatest common divisor gcd(X,Y) of two positive integers X and Y.
- gcd(X,Y) is defined as the largest integer that divides exactly into both X and Y.
- Eg : gcd(12,18) = 6 and gcd(12,17)=1
- It is customary to assume that gcd(0,0)=0.
- Euclid's algorithm to calculate gcd(X,Y)
- For example, with X=20 and Y=12, gcd algorithm proceeds as follows,
- gcd(20,12)=4.

HARDWIRED CONTROL : GCD PROCESSOR

```
gcd(in: X,Y; out: Z);
   register XR, YR, TEMPR;
   XR := X;                          {Input the data}
   YR := Y;
   while XR > 0 do begin
      if XR ≤ YR then begin          {Swap XR and YR}
         TEMPR := YR;
         YR := XR;
         XR := TEMPR; end
         XR := XR – YR;              {Subtract YR from XR}
   end
   Z := YR;                          {Output the result}
end gcd;
```

Figure 5.5
Procedure *gcd* to compute the greatest common divisor of two numbers.

HARDWIRED CONTROL : GCD PROCESSOR

Conditions		Actions	
		$XR := 20;\ YR := 12;$	
$XR > 0$:	$XR > YR$:	$XR := XR - YR = 8;$	
$XR > 0$:	$XR \leq YR$:	$YR := 8;\ XR := 12;$	$XR := XR - YR = 4;$
$XR > 0$:	$XR \leq YR$:	$YR := 4;\ XR := 8;$	$XR := XR - YR = 4;$
$XR > 0$	$XR \leq YR$:	$YR := 4;\ XR := 4;$	$XR := XR - YR = 0;$
$XR \leq 0$:		$Z := 4;$	

HARDWIRED CONTROL : GCD PROCESSOR

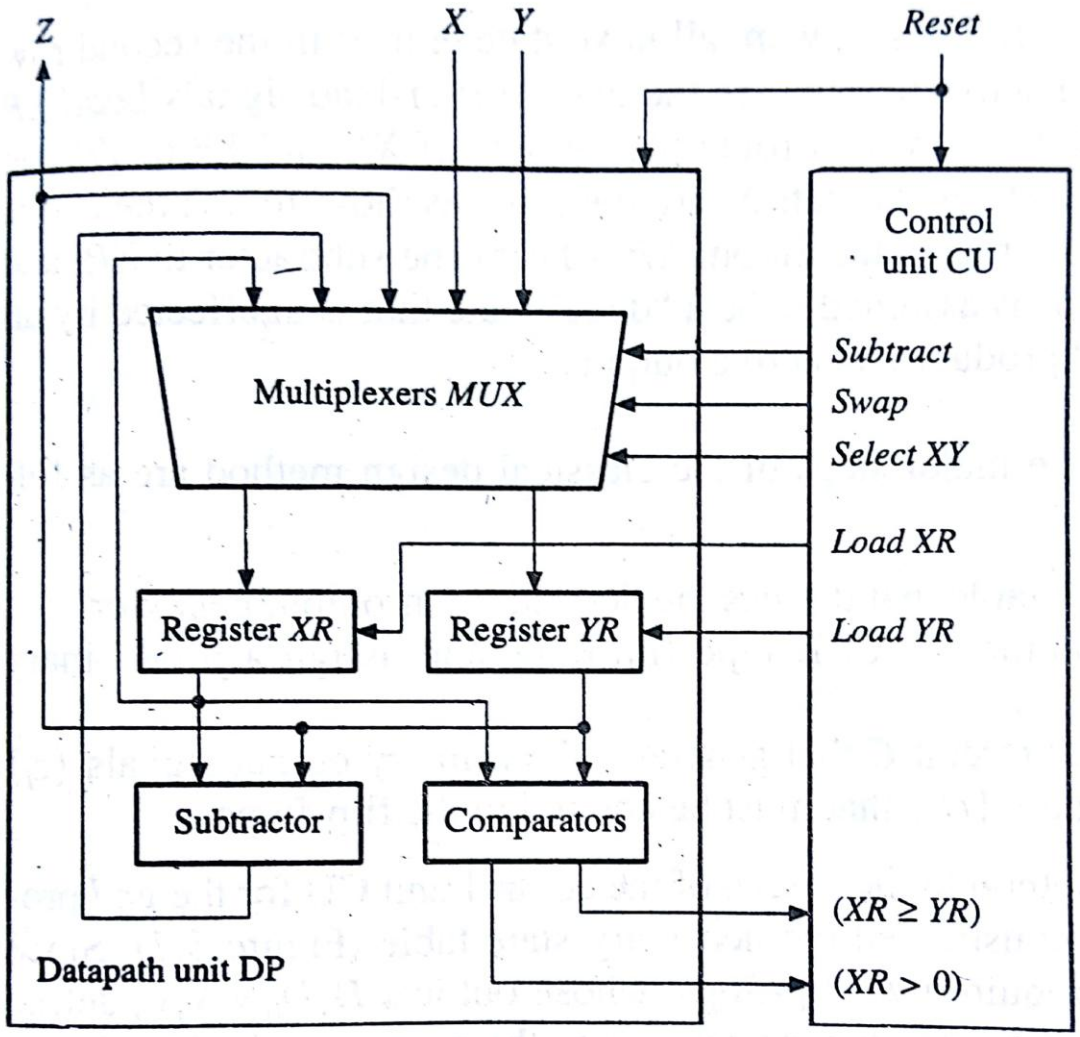


Figure 5.6
Hardware needed to implement the *gcd* procedure.

HARDWIRED CONTROL : GCD PROCESSOR

- DP contain a pair of registers XR and YR to store the corresponding variables, one or more functional units to perform subtraction and magnitude comparison, and multiplexers for data routing.
- Read and write to a register can be performed in the same clock cycle.
- The swap operation can therefore be done without conflict in one cycle : X := Y, Y := X.
- CU generates control signals Load XR and Load YR to load each register independently with the input data X and Y.
- A control signal Select XY routes X and Y to XR and YR, respectively.
- Another signal Swap controls the swap operation, which requires routing the outputs of the XR and YR registers to each others inputs.
- A final signal Subtract is assumed to control the subtraction XR := XR – YR by routing the output of the subtractor to XR.
- The input signals to CU are an asynchronous Reset signal, two comparison signals (XR≥YR) and (XR>0) generated by DP, and the usual, implicit clock signal.

HARDWIRED CONTROL : GCD PROCESSOR

- State table defining the control unit of the gcd processor

	Inputs ($XR > 0$), ($XR \geq YR$)			Outputs				
State	**0–**	**10**	**11**	***Subtract***	***Swap***	***Select XY***	***Load XR***	***Load YR***
S_0 (Begin)	S_3	S_1	S_2	0	0	1	1	1
S_1 (Swap)	S_2	S_2	S_2	0	1	0	1	1
S_2 (Subtract)	S_3	S_1	S_2	1	0	0	1	0
S_3 (End)	S_3	S_3	S_3	0	0	0	0	0

Figure 5.7
State table defining the control unit of the *gcd* processor.

HARDWIRED CONTROL : GCD PROCESSOR

Classical method:

- The major steps are as follows:

1. Construct a P-row state table that defines the desired input-output behavior
2. Select the minimum number p of D-type flip-flops and assign a p-bit binary code to each stage.
3. Design a combinational circuit C that generates the primary output signals $\{z_i\}$ and the secondary outputs $\{D_i\}$ that must be applied to the flip-flops.

- Since there are four states, we require two flip-flops, whose outputs $D_1D_0=y_1y_0$ define CU's internal states.
- Assign binary patterns to the four states,

$S_0 = 00$

$S_1 = 01$

$S_2 = 10$

$S_3 = 11$

HARDWIRED CONTROL : GCD PROCESSOR

Classical method:

- Construct a binary version of the state table – the excitation table.
- The D flip-flop's characteristic equation $D_i^+(t+1) = D_i(t)$ defines the inputs D_1^+ and D_0^+ to the flip-flops.
- CU's combinational logic can now be derived from the excitation table .
- Use two-level sum of products (SOP) minimization.

HARDWIRED CONTROL : GCD PROCESSOR

Inputs		Present state		Next state		Outputs				
$(XR > 0)$	$(XR \geq YR)$	D_1	D_0	D_1^+	D_0^+	*Subtract*	*Swap*	*Select XY*	*Load XR*	*Load YR*
0	*d*	0	0	1	1	0	0	1	1	1
0	*d*	0	1	1	0	0	1	0	1	1
0	*d*	1	0	1	1	1	0	0	1	0
0	*d*	1	1	1	1	0	0	0	0	0
1	0	0	0	0	1	0	0	1	1	1
1	0	0	1	1	0	0	1	0	1	1
1	0	1	0	0	1	1	0	0	1	0
1	0	1	1	1	1	0	0	0	0	0
1	1	0	0	1	0	0	0	1	1	1
1	1	0	1	1	0	0	1	0	1	1
1	1	1	0	1	0	1	0	0	1	0
1	1	1	1	1	1	0	0	0	0	0

Figure 5.8
Excitation table for the control unit of the *gcd* processor.

HARDWIRED CONTROL : GCD PROCESSOR

Classical method:

- Original functions are given by,

$$D_1^+ = (\overline{XR > 0}) + (XR \geq YR) + D_0$$

$$D_0^+ = D_1 \bullet D_0 + (\overline{XR \geq YR}) \bullet \overline{D_0} + (\overline{XR > 0}) \bullet \overline{D_0}$$

$$Subtract = D_1 \bullet \overline{D_0}$$

$$Swap = \overline{D_1} \bullet D_0$$

$$SelectXY = \overline{D_1} \bullet \overline{D_0}$$

$$LoadXR = \overline{D_0} + \overline{D_1}$$

$$LoadYR = \overline{D_1}$$

HARDWIRED CONTROL : GCD PROCESSOR

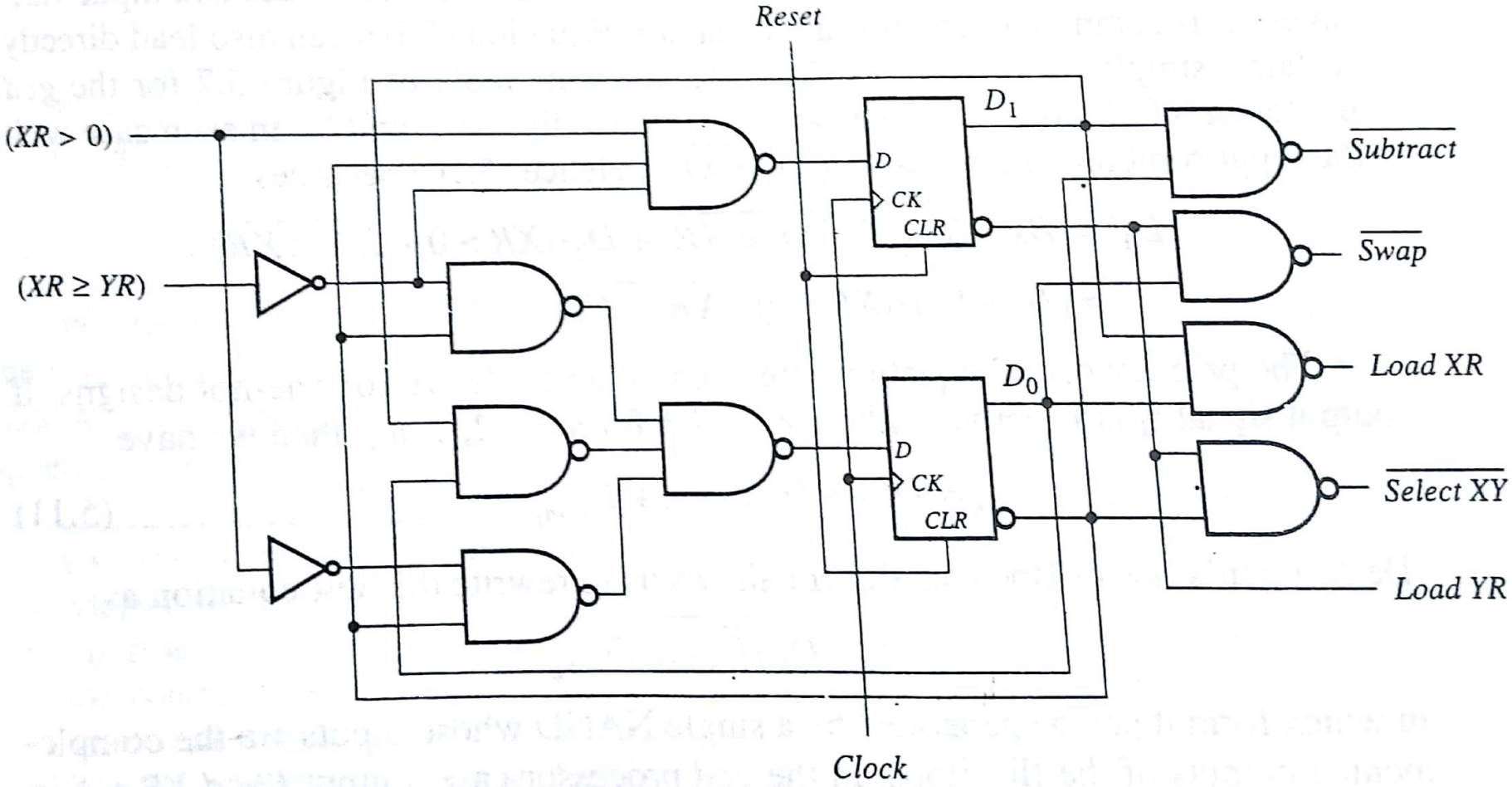


Figure 5.9
All-NAND classical design for the control unit of the *gcd* processor.

HARDWIRED CONTROL : GCD PROCESSOR

One-hot method:

- Classical design method minimizes a control unit's memory elements, its effect on the amount of combinational logic C is less obvious.
- Moreover design will have random structure and debugging, maintenance is difficult.
- In One-hot method, structure will be regular and predictable because its binary state assignment always contains a single 1 – the "**hot' bit** – while all the remaining bits are 0.
- State assignment for the gcd processor is given below,

$S_0 = 0001$

$S_1 = 0010$

$S_2 = 0100$

$S_3 = 1000$

- In general P flip-flops are needed to represent P states, so the one-hot method is restricted to fairly small values of P.

HARDWIRED CONTROL : GCD PROCESSOR

One-hot method:

➢ Key feature : next-state and output equations can be written down directly from the control units original symbolic state table.

	Inputs ($XR > 0$), ($XR \geq YR$)			Outputs				
State	0–	10	11	*Subtract*	*Swap*	*Select XY*	*Load XR*	*Load YR*
S_0 (Begin)	S_3	S_1	S_2	0	0	1	1	1
S_1 (Swap)	S_2	S_2	S_2	0	1	0	1	1
S_2 (Subtract)	S_3	S_1	S_2	1	0	0	1	0
S_3 (End)	S_3	S_3	S_3	0	0	0	0	0

Figure 5.7
State table defining the control unit of the *gcd* processor.

HARDWIRED CONTROL : GCD PROCESSOR

One-hot method:

- Key feature : next-state and output equations can be written down directly from the control units original symbolic state table.
- The control unit realization for gcd processor is given below,

$$D_0^+ = 0$$

$$D_1^+ = D_0 \bullet (XR > 0) \bullet (\overline{XR \geq YR}) + D_2 \bullet (XR > 0) \bullet (\overline{XR \geq YR})$$

$$D_2^+ = D_0 \bullet (XR > 0) \bullet (XR \geq YR) + D_1 + D_2 \bullet (XR > 0) \bullet (XR \geq YR)$$

$$D_3^+ = D_0 \bullet (\overline{XR > 0}) + D_2 \bullet (\overline{XR > 0}) + D_3$$

$$Subtract = D_2$$

$$Swap = D_1$$

$$SelectXY = D_0$$

$$LoadXR = D_0 + D_1 + D_2$$

$$LoadYR = D_0 + D_1$$

HARDWIRED CONTROL : GCD PROCESSOR

One-hot method:

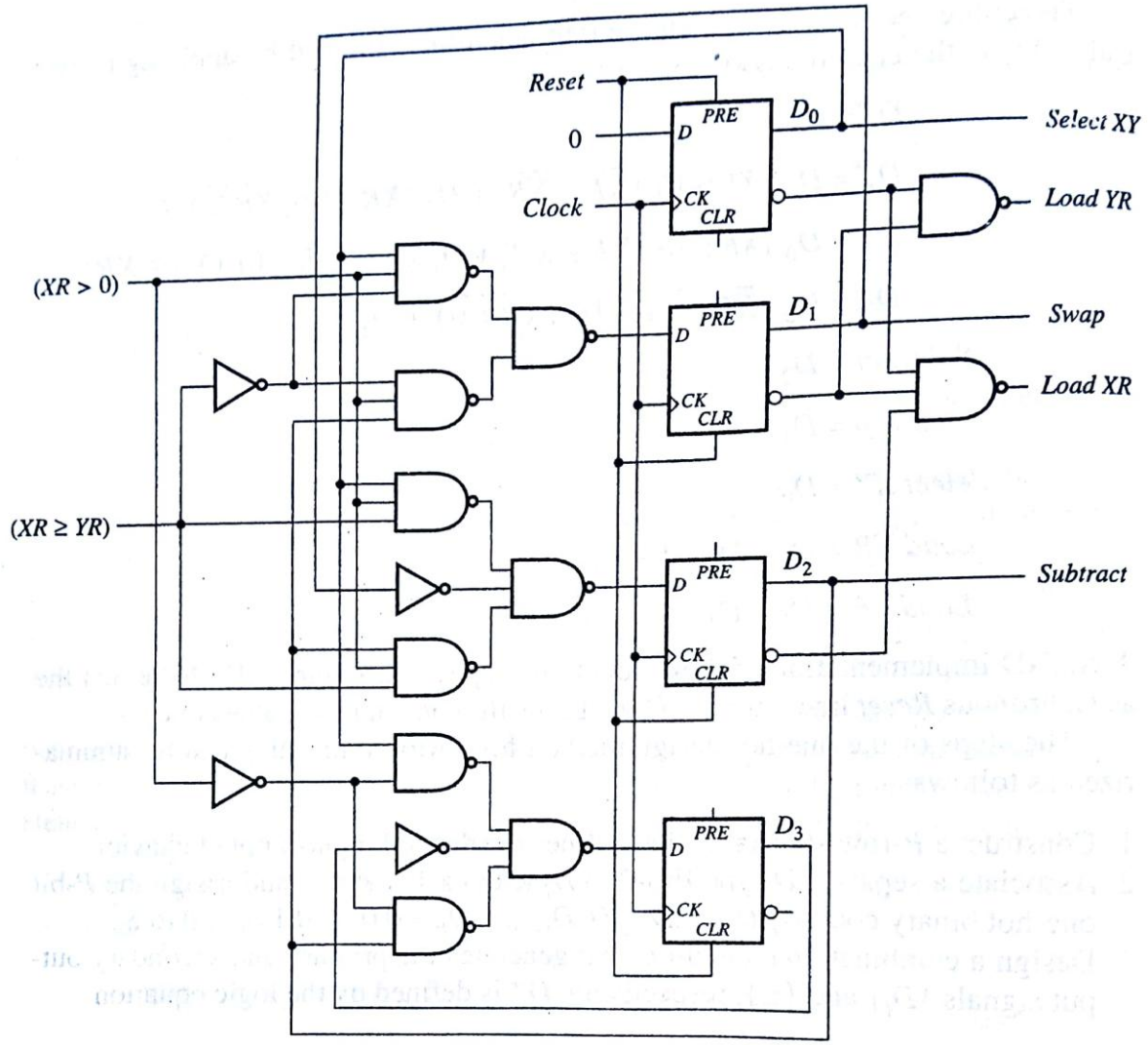


Figure 5.10
All-NAND one-hot design for the control unit of the *gcd* processor.

HARDWIRED CONTROL : GCD PROCESSOR

One-hot method:

➢ The steps of the one-hot design method for a Moore machine can be summarized as follows:

1. Construct a P-row state table that defines the desired input-output behavior.
2. Associate a separate D-type flip-flop D_i with each state S_i, and assign the P-bit one-hot binary code $D_1, D_2,\ldots,D_{i-1}, D_i, D_{i+1},\ldots, D_p = 0, 0,\ldots, 0, 1, 0, \ldots, 0$ to S_i.
3. Design a combinational circuit C that generates the primary and secondary output signals $\{D_i\}$ and $\{z_k\}$, respectively.

➢ D_i^+ is defined by the logic equation,

$$D_i^+ = \sum_{i=1}^{P} D_i (I_{j,1} + I_{j,2} + \ldots + I_{j,n_j})$$

Where $I_{j,1}, I_{j,2},\ldots,I_{j,nj}$ denote all the input combinations that cause a transition from S_j to S_i.

HARDWIRED CONTROL : GCD PROCESSOR

One-hot method:

- If z_k = 1 (active) only in rows k, h for h = 1,2,…, m_k, then z_k is defined by,

$$z_k = D_{k,1} + D_{k,2} + \ldots + D_{k,m_k} = \overline{\overline{D_{k,1}}\,\overline{D_{k,2}}\ldots\overline{D_{k,m_k}}}$$

MICROPROGRAMMED CONTROL

- Use microprograms to select, interpret and execute a processor's instruction set.
- An instruction is implemented by a sequence of one or more sets of concurrent microoperations.
- Each microoperation is associated with a group of control lines that must be activated in a prescribed sequence to trigger the microoperations.
- As the number of instructions and control lines can be in hundreds, a hardwired control unit is difficult to design and verify.
- In hardwired control, to correct design errors or update the instruction set, require that the control unit be redesigned.

MICROPROGRAMMED CONTROL

- Microprogramming is a method of control-unit design in which the control signal selection and sequencing information is stored in a ROM or RAM called a **control memory CM**.
- The control signals to be activated at any time are specified by a **microinstruction**, which is fetched from CM in much the same way an instruction is fetched from main memory.
- Microinstruction also explicitly or implicitly specifies the next microinstruction to be used, thereby providing the necessary information for microoperation sequencing.
- A set of related microinstructions forms a **microprogram.**
- Microprograms can be changed relatively easily by changing the contents of CM
- Microprogramming yields control units that are more flexible than their hardwired counterparts.
- extra hardware interms of control memory and its access circuitry.

MICROPROGRAMMED CONTROL

- There is also a performance penalty due to the time required to access the microinstructions from CM.
- Hence in RISC and other high speed processors, this method is not used, where the chip area and circuit delay must both be minimized.
- Symbolic languages similar to assembly languages are used to write microprograms: these are called **microassembly languages**.
- A **microassembler** is necessary to translate microprograms into executable programs that can be stored in the control memory.

MICROPROGRAMMED CONTROL : ORGANIZATION

- Microinstruction has two parts : a set of **control fields** that specify the control signals to be activated and an **address field** that contain the address in CM of the next microinstruction to be executed.
- The control memory CM is implemented by a ROM.
- The left part (AND plane) of the ROM decodes an address obtained from the **control memory address register (CMAR**).
- Each address selects a particular row in the right part (OR plane) of the ROM that contains a microinstruction composed of a 6-bit control field and a 3-bit address field.
- The top-most row represent the microinstruction with address 000, is selected, the control signals c_0, c_2 and c_4 are activated, as indicated by the xs in the control field.
- At the same time, the contents of the address field $a_2a_1a_0$ = 001 are sent to the CMAR, where they are stored and used to address the next microinstruction to be executed.

MICROPROGRAMMED CONTROL : ORGANIZATION

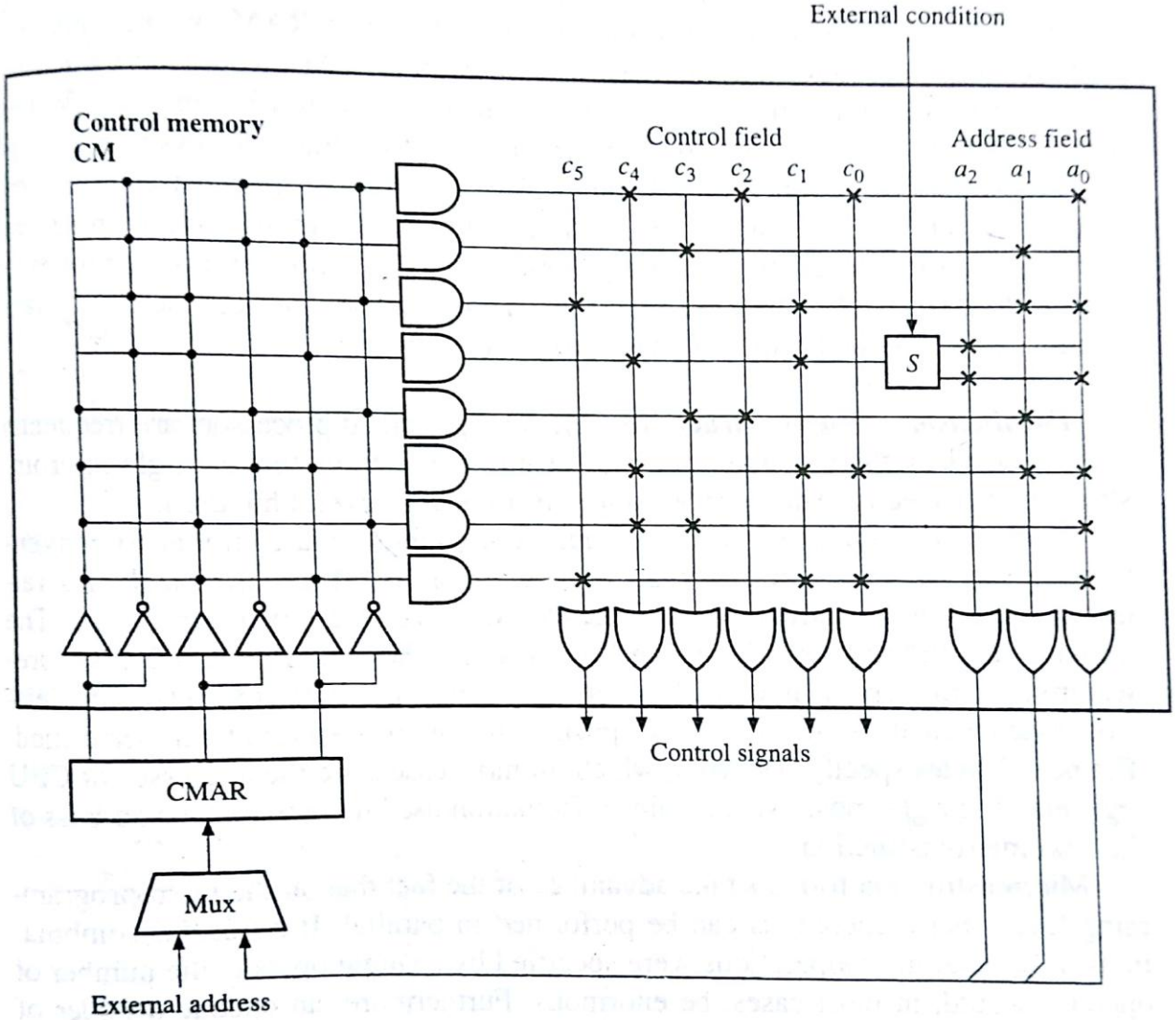


Figure 5.26
Basic structure of a microprogrammed control unit.

MICROPROGRAMMED CONTROL : ORGANIZATION

- The CMAR can be loaded from an external source as well as from the address field of a microinstruction.
- The external source typically provides the starting address of a microprogram in the CM.
- A specific microprogram prestored in CM executes (interprets) each instruction of a microprogrammed CPU.
- The instructions' opcode, after suitable encoding, provides the starting address for its microprogram.

MICROPROGRAMMED CONTROL : ORGANIZATION

- A major area of concern is the microinstruction's word length, since it greatly influences the size and cost of the CM.
- Microinstruction length is determined by three factors:

1. The maximum number of simultaneous microoperations that must be specified, that is, the degree of parallelism required at the microoperation level.
2. The way in which the control information is represented or encoded.
3. The way in which the next microinstruction address is specified.

- Control memories are usually ROMs, so their contents cannot be altered online.
- Normally there is no need to change the CM except to correct design errors or to make minor enhancements to the system.
- A processor with a **writable control memory (WCM)** or read-write memory or RAM is said to be **dynamically microprogrammable** because the control memory contents can be altered under program control.

MICROPROGRAMMED CONTROL : PARALLELISM IN MICROINSTRUCTIONS

- Microprogrammed processors are frequently characterized by the maximum number of microoperations that a single microinstruction can specify.
- Number ranges from one to several hundred.
- Microinstruction that specify a single microoperation are similar to conventional machine instructions.
- They are relatively short, but due to their lack of parallelism, more microinstructions are needed to perform a given operation.
- Eg: IBM System/370 Model 145

MICROPROGRAMMED CONTROL : PARALLELISM IN MICROINSTRUCTIONS

- It consists of 4 bytes (32 bits).
- Left byte – opcode (specifies the microoperation to be performed)
- Next 2 bytes specify operands (most cases , address of CPU registers).
- Right most byte contain information used to construct the address of next microinstruction.

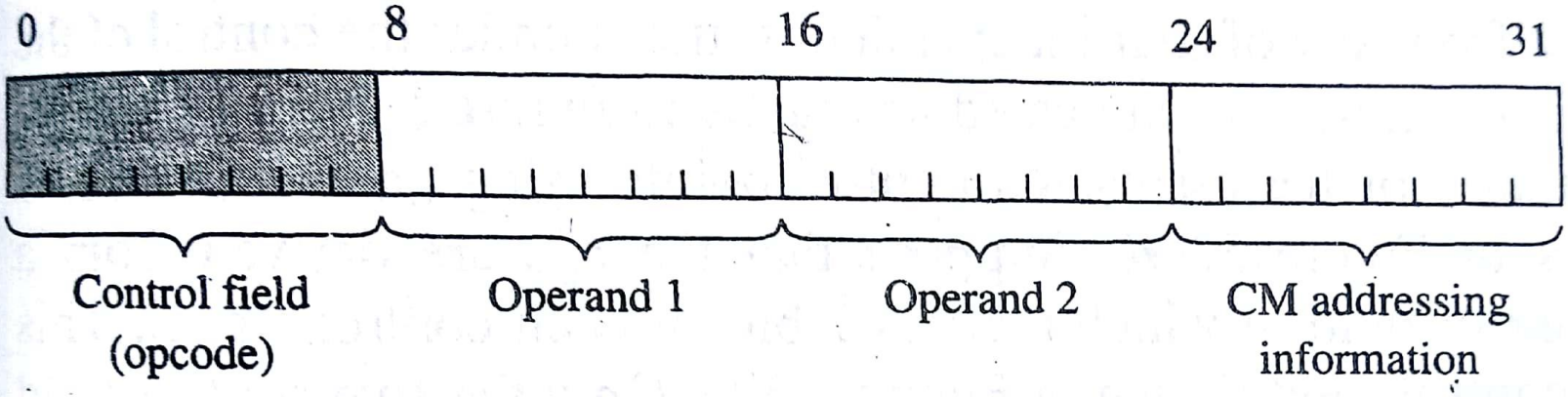


Figure 5.27
Microinstruction format of the IBM System/370 Model 145.

MICROPROGRAMMED CONTROL : PARALLELISM IN MICROINSTRUCTIONS

- Another microinstruction style of IBM System/360 Model 50.
- It encompasses 90-bits, which are partitioned into separate fields for various purposes.
- There are 21 fields, which constitute the control fields (shaded region).

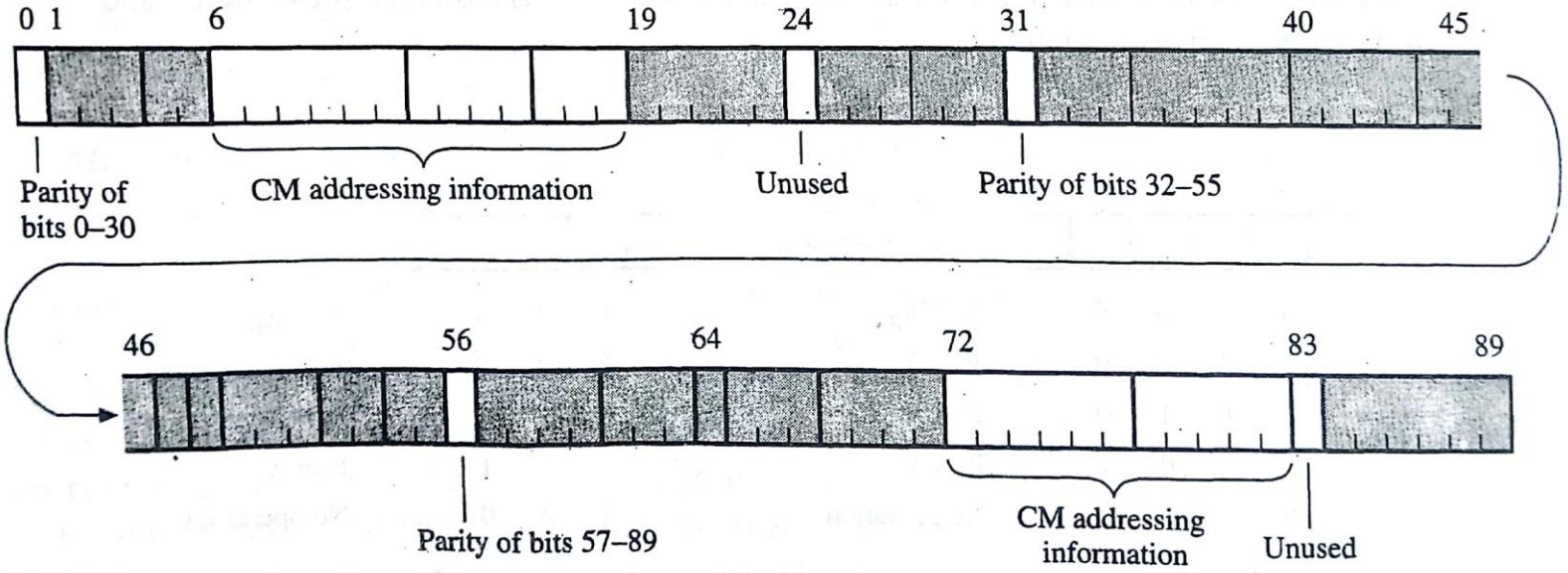


Figure 5.28
Ninety-bit microinstruction format of the IBM System/360 Model 50 (shaded areas are control fields).

MICROPROGRAMMED CONTROL : PARALLELISM IN MICROINSTRUCTIONS

- Remaining fields are used to generate the next microinstruction address and to detect errors by means of parity bits.
- Example: the 3-bit control field consisting of bits 65:67 controls the right input to the main adder of the CPU. This field indicates which of several possible registers should be connected to the adder's right input.
- Bits 68:71 identify the function to be performed by the adder; the possibilities include binary addition and decimal addition with various ways of handling input and output carry bits.

MICROPROGRAMMED CONTROL : PARALLELISM IN MICROINSTRUCTIONS

- A control field for every control signal is wasteful of control memory space because most of the possible combinations of control signals are never used.
- Consider for instance, the register R, which can be loaded from any of four independent sources under the control of four separate signals c_0, c_1, c_2, c_3.

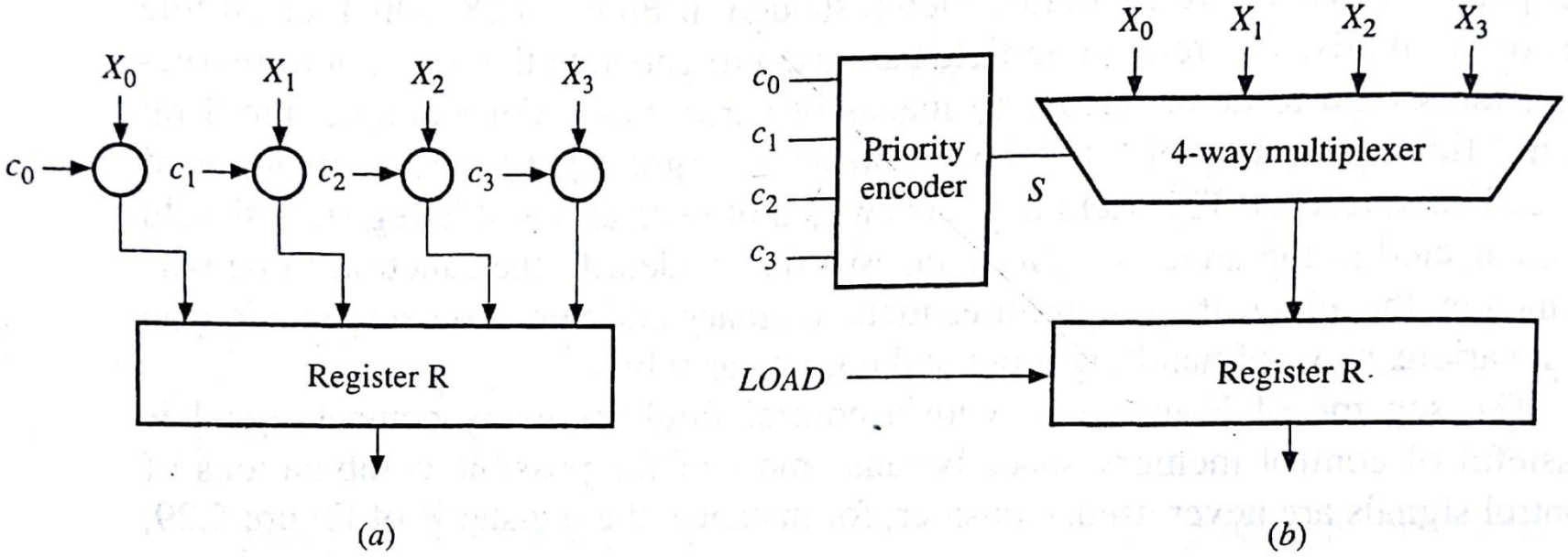
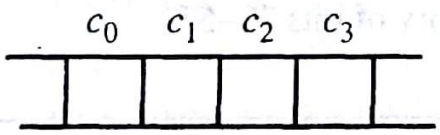


Figure 5.29
A register that can be loaded from four independent sources: (*a*) abstract representation and (*b*) possible implementation.

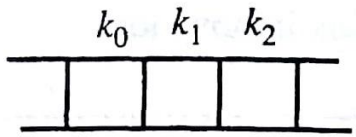
MICROPROGRAMMED CONTROL : PARALLELISM IN MICROINSTRUCTIONS

- Suppose that the c_i's are derived from a microinstruction control field in which there is 1 bit for each control signal. This results in the 4-bit control field.
- Only the five control-field patterns are valid, since any other pattern will create a conflict by attempting to load R from two or more independent sources simultaneously.



c_0	c_1	c_2	c_3	
1	0	0	0	$R := X_0$
0	1	0	0	$R := X_1$
0	0	1	0	$R := X_2$
0	0	0	1	$R := X_3$
0	0	0	0	No operation

(a)



k_0	k_1	k_2	
0	0	0	$R := X_0$
0	0	1	$R := X_1$
0	1	0	$R := X_2$
0	1	1	$R := X_3$
1	0	0	No operation

(b)

Figure 5.30
Control field for the circuit of Figure 5.29: (*a*) unencoded format and (*b*) encoded format.

MICROPROGRAMMED CONTROL : PARALLELISM IN MICROINSTRUCTIONS

- These five patterns can also be encoded into a field $K=k_0\ k_1\ k_2$ of width $[\log_2 5] = 3$ bits, thus reducing the width of the control field from 4 to 3 bits.
- The unencoded format has the advantage that all the control signals are individually identified in, and can be directly obtained from the microinstruction.
- The encoded control signal k_0, k_1, k_2 must be passed through a decoder if we wish to extract the four original control signals c_0, c_1, c_2, c_3.

MICROPROGRAMMED CONTROL : HORIZONTAL VS VERTICAL

- Microinstructions are commonly divided into two types (**Horizontal and Veritcal**)
- Horizontal microinstructions have the following general attributes:

1. Long formats.
2. Ability to express a high degree of parallelism
3. Little encoding of the control information

- Vertical microinstructions are characterized by,

1. Short formats
2. Limited ability to express parallel microoperations
3. Considerable encoding of the control information

- Horizontal microinstruction : Eg – IBM System/360 Model 50
- Vertical microinstruction : Eg – System/370 Model 145

MICROPROGRAMMED CONTROL : HORIZONTAL VS VERTICAL

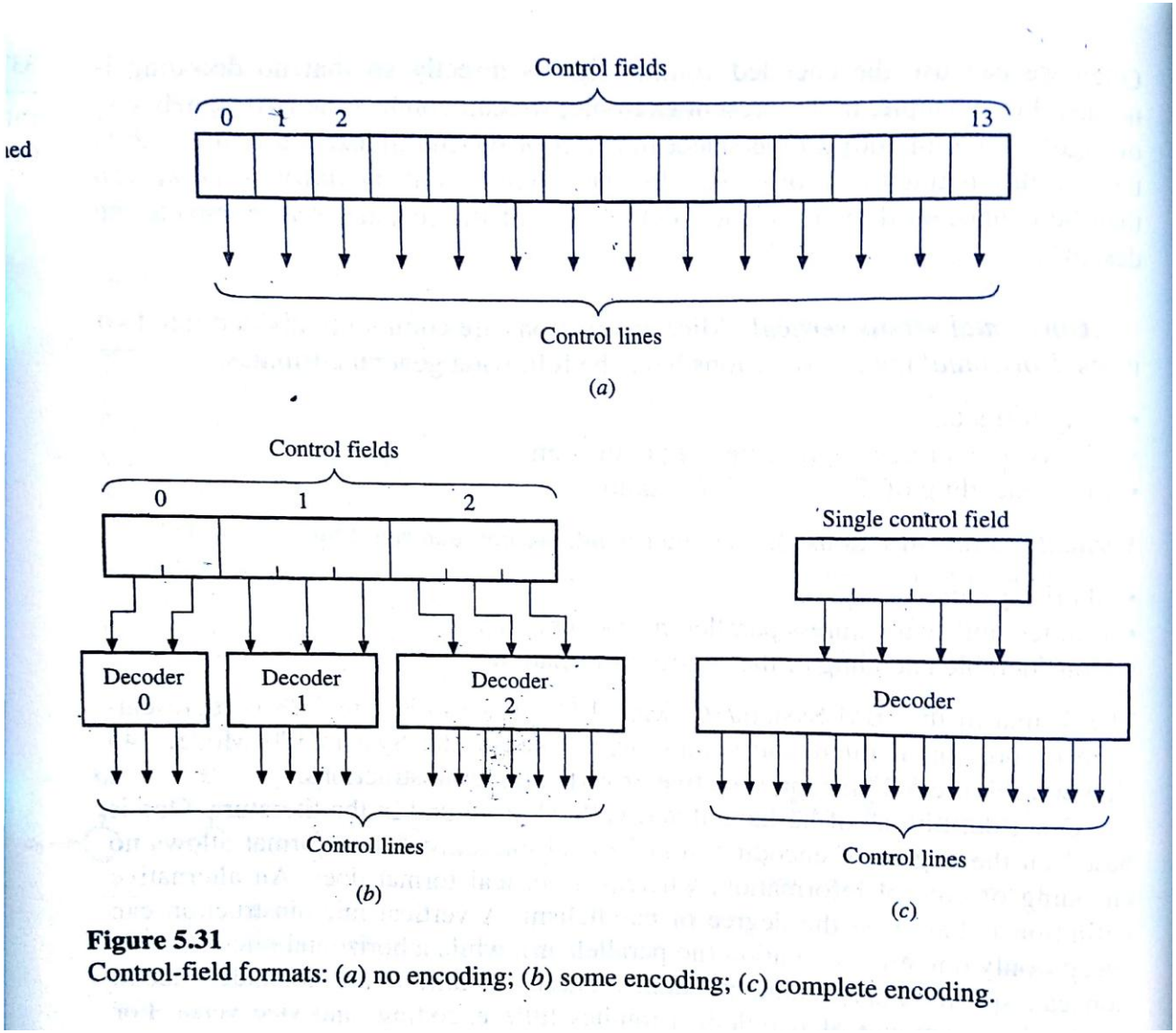


Figure 5.31
Control-field formats: (*a*) no encoding; (*b*) some encoding; (*c*) complete encoding.

MICROPROGRAMMED CONTROL : MICROINSTRUCTION ADDRESSING

- Each microinstruction in the basic design contains within itself the address of the next microinstruction to be executed.
- In the case of branch microinstructions, two possible next addresses are included.
- Advantage- no time is lost in microinstruction address generation, but it is wasteful of control memory space.
- The address fields can be eliminated from all but branch instructions by using a **microprogram counter μPC** as the primary source of microinstruction addresses.
- Since only instructions have to be fetched from the control memory, μPC is also used as the control memory address register CMAR.

MICROPROGRAMMED CONTROL : MICROOPERATION TIMING

- So far we have assumed that an microinstruction activates a set of control signals for an unspecified time during the microinstructions execution cycle.
- A single clock signal synchronizes the control signals, and its period can be the same as the microinstruction cycle period – **monophase**.
- The number of microinstructions to specify a particular operation can be reduced by dividing the microinstruction cycle into several sequential subperiods or (clock) phases.
- A control signal is typically active during only one of the phases.
- This **polyphase** mode of operation permits a single microinstruction to specify a short sequence of microoperations for some increase in the complexity of the microinstruction format.

MICROPROGRAMMED CONTROL : MICROOPERATION TIMING

➢Consider a microinstruction that controls the register-transfer operation,

$$R := f(R_1, R_2)$$

Where R can be R_1 or R_2.

➢ The operation can be performed in several phases (four-phase interpretation),

1. Phase ϕ_1 : Fetch the next microinstruction from the control memory CM.
2. Phase ϕ_2 : Transfer the contents of registers R1 and R2 to the inputs of the f unit.
3. Phase ϕ_3 : Store the result generated by the f unit in a temporary register or latch L.
4. Phase ϕ_4 : Transfer the contents of L to the destination register R.

MICROPROGRAMMED CONTROL : MICROOPERATION TIMING

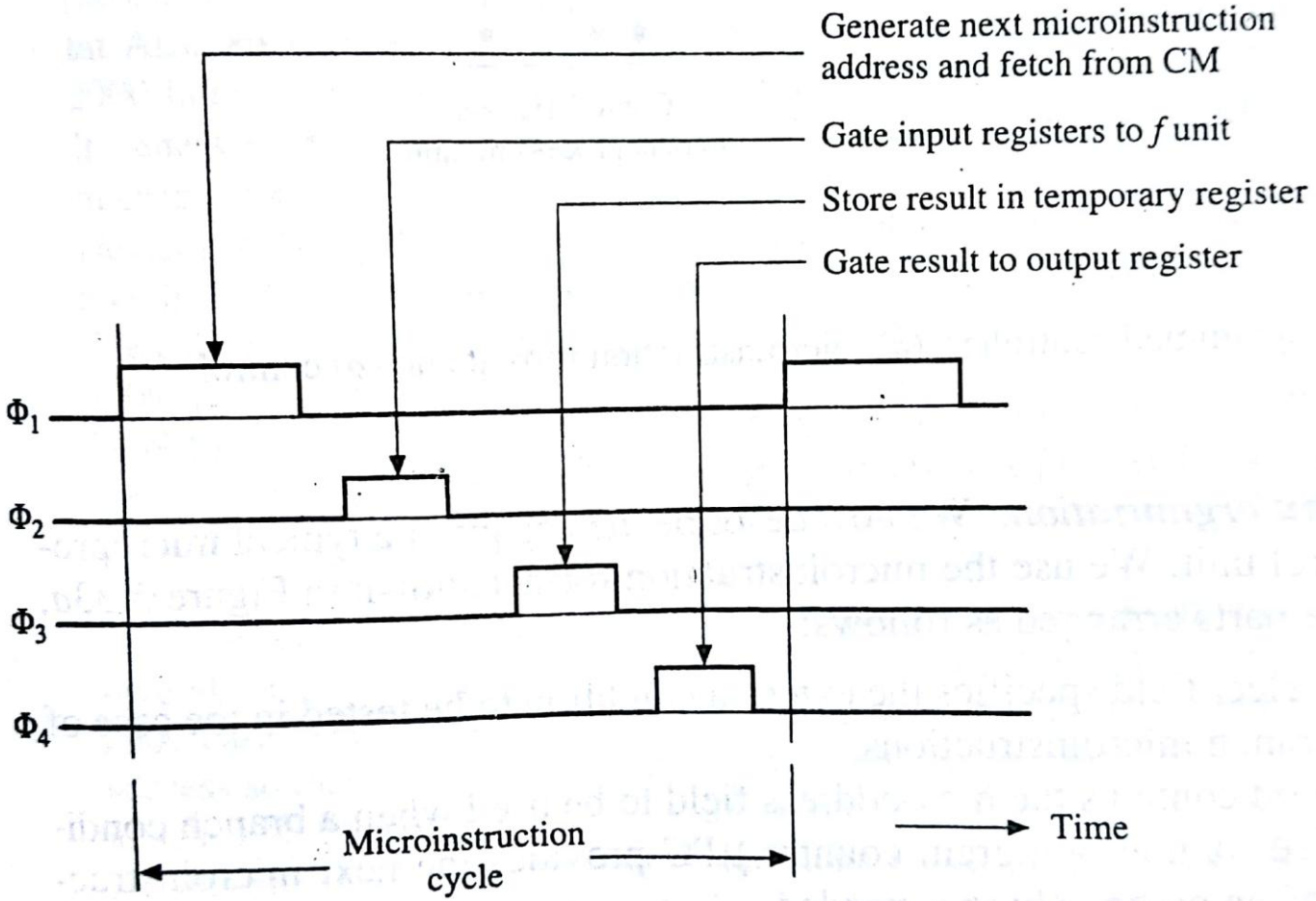


Figure 5.32
Timing diagram for a four-phase microinstruction.

MICROPROGRAMMED CONTROL : MICROOPERATION TIMING

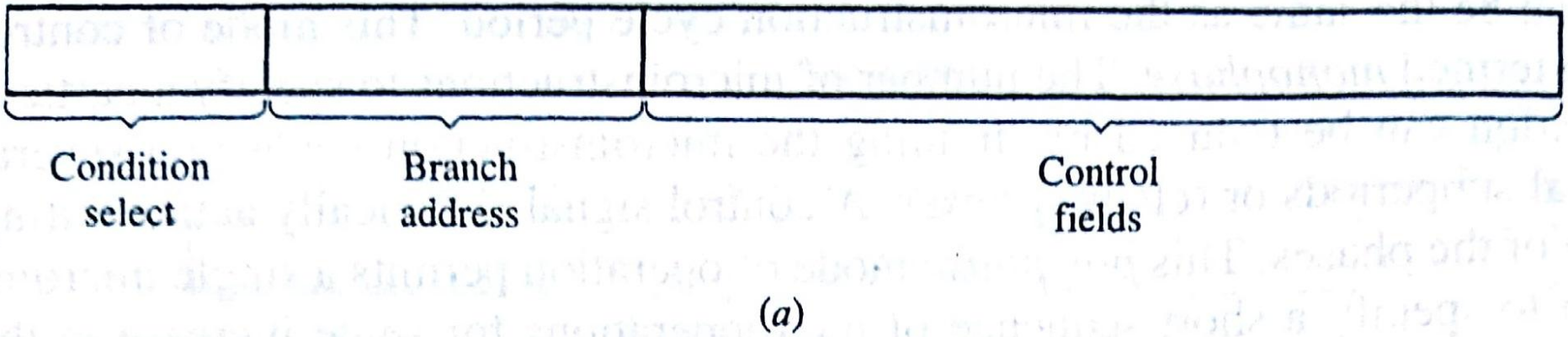
- We have also assumed that the influence of a microinstruction control field is limited to the period during which the microinstruction is executed.
- We can lift this restriction by storing the control field in a register that continues to exercise control until a subsequent microinstruction modifies it – **Residual control**.

MICROPROGRAMMED CONTROL : CONTROL UNIT ORGANIZATION

➢Microinstruction format has three parts arranged as follows,

1. Condition select field – specifies the external condition to be tested in the case of conditional branch microinstructions.
2. Address field – contains the next-address field to be used when a branch condition is satisfied. A microprogram counter μPC provides the next microinstruction address when no branching is needed.
3. Control fields – Specifies in encode or unencoded format the control signals that are activated to perform the desired microoperations.

MICROPROGRAMMED CONTROL : CONTROL UNIT ORGANIZATION



(*a*)

MICROPROGRAMMED CONTROL : CONTROL UNIT ORGANIZATION

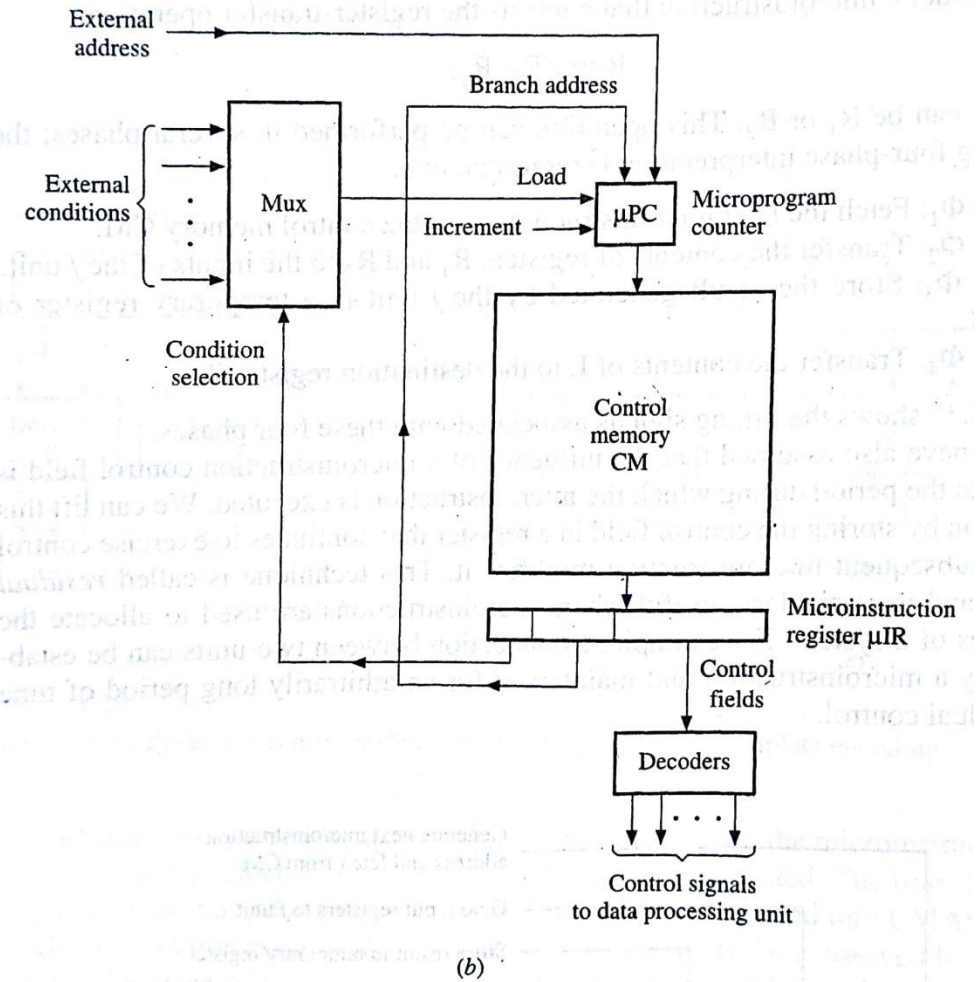


Figure 5.33
Typical microprogrammed controller: (*a*) microinstruction format and (*b*) control unit organization.

MICROPROGRAMMED CONTROL : CONTROL UNIT ORGANIZATION

- The counter μPC is the address register for the control memory CM.
- The contents of the addressed word in CM are transferred to the microinstruction register μIR.
- The control fields are decoded if necessary and produce control signals for the data processing unit; μPC is then incremented.
- If a branch is specified by the microinstruction in μIR, the contents of the microinstruction's address field are loaded into μPC.
- The condition-select field controls a multiplexer that activates the parallel-load control input of μPC based on the status of some external condition variables.

MICROPROGRAMMED CONTROL : CONTROL UNIT ORGANIZATION

- Suppose that two condition variables v_1, v_2 must be tested.
- A condition select field s_0, s_1 of 2 bits suffices, with the following interpretations,

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

- The multiplexer has four inputs x_0, x_1, x_2, x_3, where x_i is routed to the multiplexer's output when s_0s_1=i.
- Hence we require x_0=0, x_1=v_1, x_2=v_2 and x_3=1 to control the loading of microinstruction branch addresses into μPC.

MICROPROGRAMMED CONTROL : CONTROL UNIT ORGANIZATION

- Finally, a provision is made for loading μPC with an address from an external source.
- This address is used to enter the starting address of the desired microprogram in cases where CM contains more than one microprogram.

MULTIPLIER CONTROL : HARDWIRED CONTROL

➢The design of a control unit CU for the twos complement (Robertson) multiplier.

```
2Cmultiplier    (in: INBUS; out: OUTBUS);
                register A[7:0], M[7:0], Q[7:0], COUNT[2:0], F;
                bus INBUS[7:0], OUTBUS[7:0];
BEGIN:          A := 0, COUNT := 0, F := 0,
INPUT:          M := INBUS;
                Q := INBUS;
ADD:            A[7:0] := A[7:0] + M[7:0] × Q[0],
                F := (M[7] and Q[0]) or F;
RSHIFT:         A[7] := F, A[6:0].Q := A.Q[7:1], COUNT := COUNT + 1;
TEST:           if COUNT ≠ 7 then go to ADD;
SUBTRACT:       A[7:0] := A[7:0] – M[7:0] × Q[0], Q[0] := 0;
OUTPUT:         OUTBUS := Q;
                OUTBUS := A;
end 2Cmultiplier;
```

Figure 4.13
HDL description of the multiplier for 8-bit twos-complement fractions.

MULTIPLIER CONTROL : HARDWIRED CONTROL

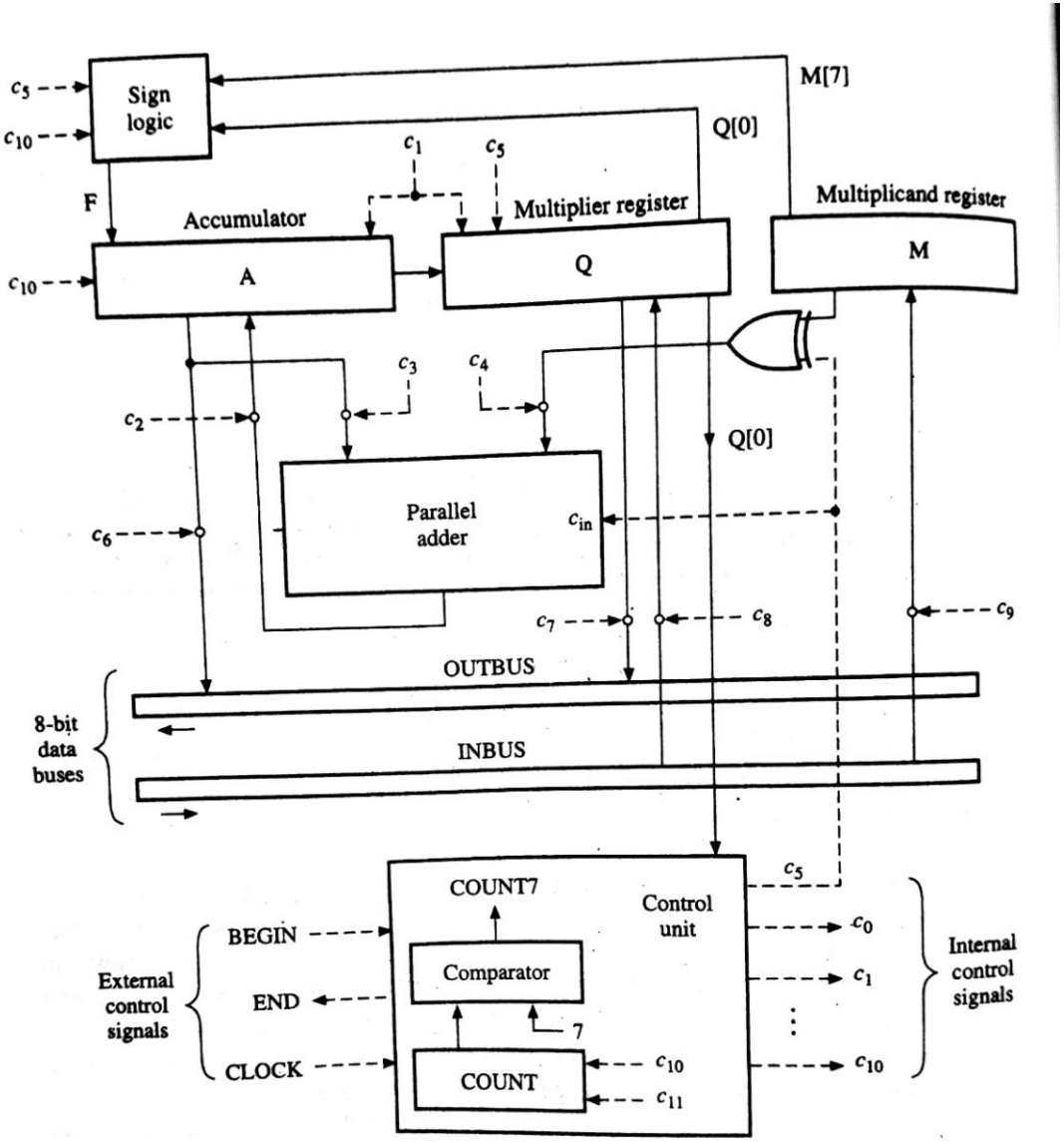


Figure 5.13
Twos-complement multiplier with a set of control points.

MULTIPLIER CONTROL : HARDWIRED CONTROL

Control Signal	Operation controlled
c_0	Set sign bit of A to F.
c_1	Right-shift register-pair A.Q.
c_2	Transfer adder output to A.
c_3	Transfer A to left input of adder.
c_4	Transfer M to right input of adder.
c_5	Perform subtraction (correction). Clear Q[0].
c_6	Transfer A to output bus.
c_7	Transfer Q to output bus.
c_8	Transfer word on input bus to Q.
c_9	Transfer word on input bus to M.
c_{10}	Clear A, COUNT, and F registers.
c_{11}	Increment COUNT.
END	Completion signal (CU idle).

Figure 5.14
Control signals for the twos-complement multiplier.

MULTIPLIER CONTROL : HARDWIRED CONTROL

- A control point can be associated with each distinct action (eg: register-transfer operation).
- Operations that takes place simultaneously may be able to share control signals.
- Eg: Statement labeled BEGIN in algorithm, requires the register A, COUNT and F to be reset simultaneously to the all-zero state.
- A control signal called COUNT7, which is set to 1 when COUNT=111_2 and is set to 0 otherwise.
- COUNT7, the right-most bit Q[0] of the multiplier register Q and the external BEGIN signal serve as the primary inputs to CU.

MULTIPLIER CONTROL : HARDWIRED CONTROL

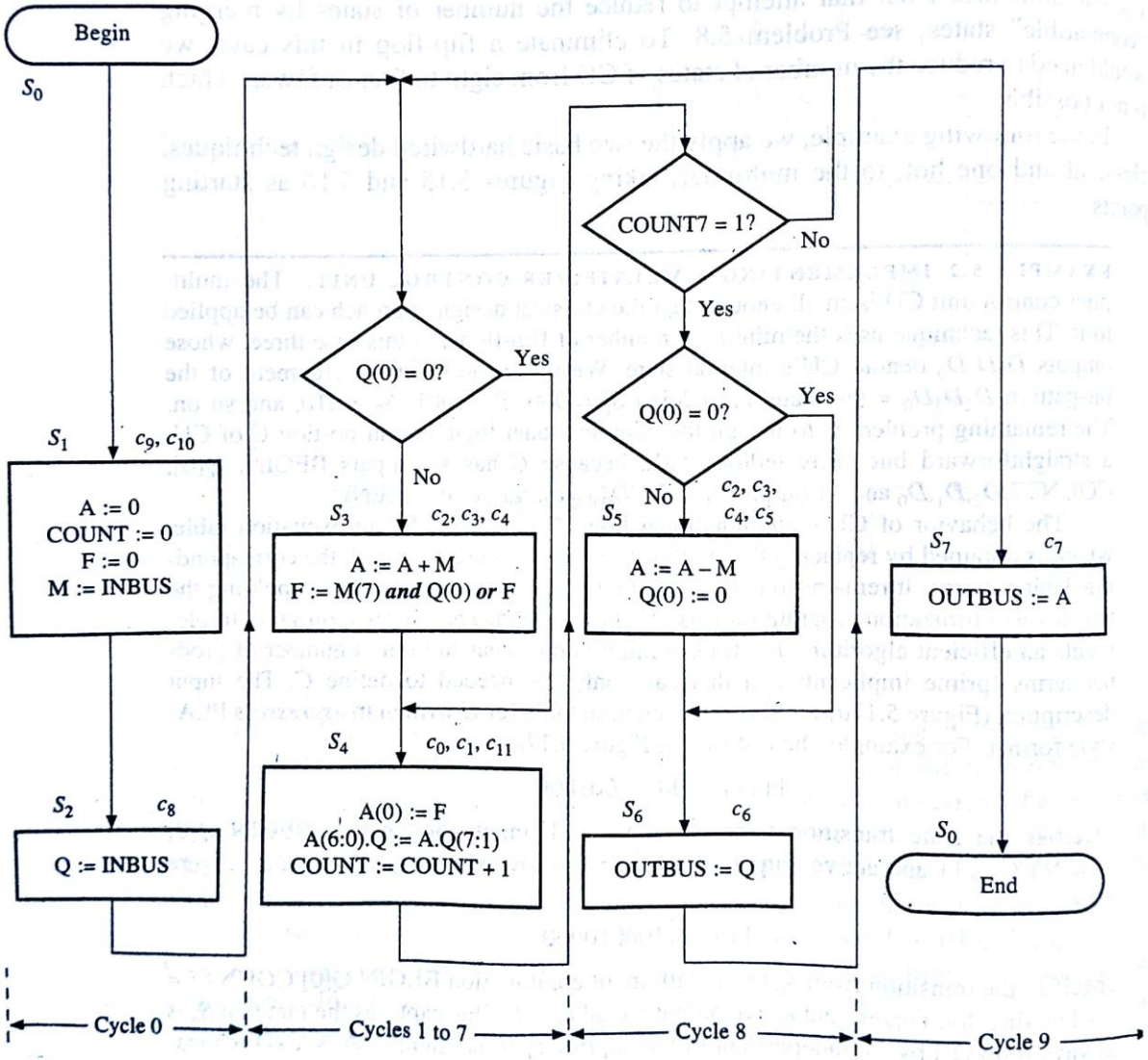


Figure 5.15
Flowchart for the twos-complement multiplier.

MULTIPLIER CONTROL : HARDWIRED CONTROL

- To obtain a state table for the control unit CU, we associate a state S_i with every operation block, leading to the seven states labeled S_1:S_7.
- An additional state S_0 represents the reset or waiting condition of the control unit.
- CU has three primary input signals – BEGIN, Q[0] and COUNT7; hence there are eight possible input combinations.
- Eight state Moore-type state table.
- The 13 output control signals c_0:$c_{11,}$ END can be immediately reduced to 8 because several sets are equivalent in that they are always activated together, specifically $c_0 = c_1 = c_{11,}$ $c_2 = c_3 = c_4$, and $c_9 = c_{10}$.

MULTIPLIER CONTROL : HARDWIRED CONTROL

	Inputs: BEGIN Q[0] Count7								Outputs												
State	000	001	010	011	100	101	110	111	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	END
S_0	S_0	S_0	S_0	S_0	S_1	S_1	S_1	S_1	0	0	0	0	0	0	0	0	0	0	0	0	1
S_1	S_2	S_2	S_2	S_2	S_2	S_2	S_2	S_2	0	0	0	0	0	0	0	0	0	1	1	0	0
S_2	S_4	S_4	S_3	S_3	S_4	S_4	S_3	S_3	0	0	0	0	0	0	0	0	1	0	0	0	0
S_3	S_4	S_4	S_4	S_4	S_4	S_4	S_4	S_4	0	0	1	1	1	0	0	0	0	0	0	0	0
S_4	S_4	S_6	S_3	S_5	S_4	S_6	S_3	S_5	1	1	0	0	0	0	0	0	0	0	0	1	0
S_5	S_6	S_6	S_6	S_6	S_6	S_6	S_6	S_6	0	0	1	1	1	1	0	0	0	0	0	0	0
S_6	S_7	S_7	S_7	S_7	S_7	S_7	S_7	S_7	0	0	0	0	0	0	1	0	0	0	0	0	0
S_7	S_0	S_0	S_0	S_0	S_0	S_0	S_0	S_0	0	0	0	0	0	0	0	1	0	0	0	0	0

Figure 5.16
State table for the multiplier control unit.

MULTIPLIER CONTROL : HARDWIRED CONTROL

- One-hot method
- Eight flip-flops are needed to accommodate CU's eight states S_0:S_7.
- The next state equations are,

$$D_0^+ = D_0 \bullet \overline{BEGIN} + D_7$$

$$D_1^+ = D_0 \bullet BEGIN$$

$$D_2^+ = D_1$$

$$D_3^+ = D_2 \bullet Q[0] + D_4 \bullet Q[0] \bullet \overline{COUNT7}$$

$$D_4^+ = D_2 \bullet \overline{Q[0]} + D_3 + D_4 \bullet \overline{Q[0]} \bullet \overline{COUNT7}$$

$$D_5^+ = D_4 \bullet Q[0] \bullet COUNT7$$

$$D_6^+ = D_5 + D_4 \bullet \overline{Q[0]} \bullet \overline{COUNT7}$$

$$D_7^+ = D_6$$

MULTIPLIER CONTROL : HARDWIRED CONTROL

- One-hot method
- The output equations are,

$c_0 = c_1 = c_{11} = D_4$

$c_2 = c_3 = c4 = D_3 + D_5$

$c_5 = D_5$

$c_6 = D_6$

$c_7 = D_7$

$c_8 = D_2$

$c_9 = c_{10} = D_1$

$END = D_0$

MULTIPLIER CONTROL : HARDWIRED CONTROL

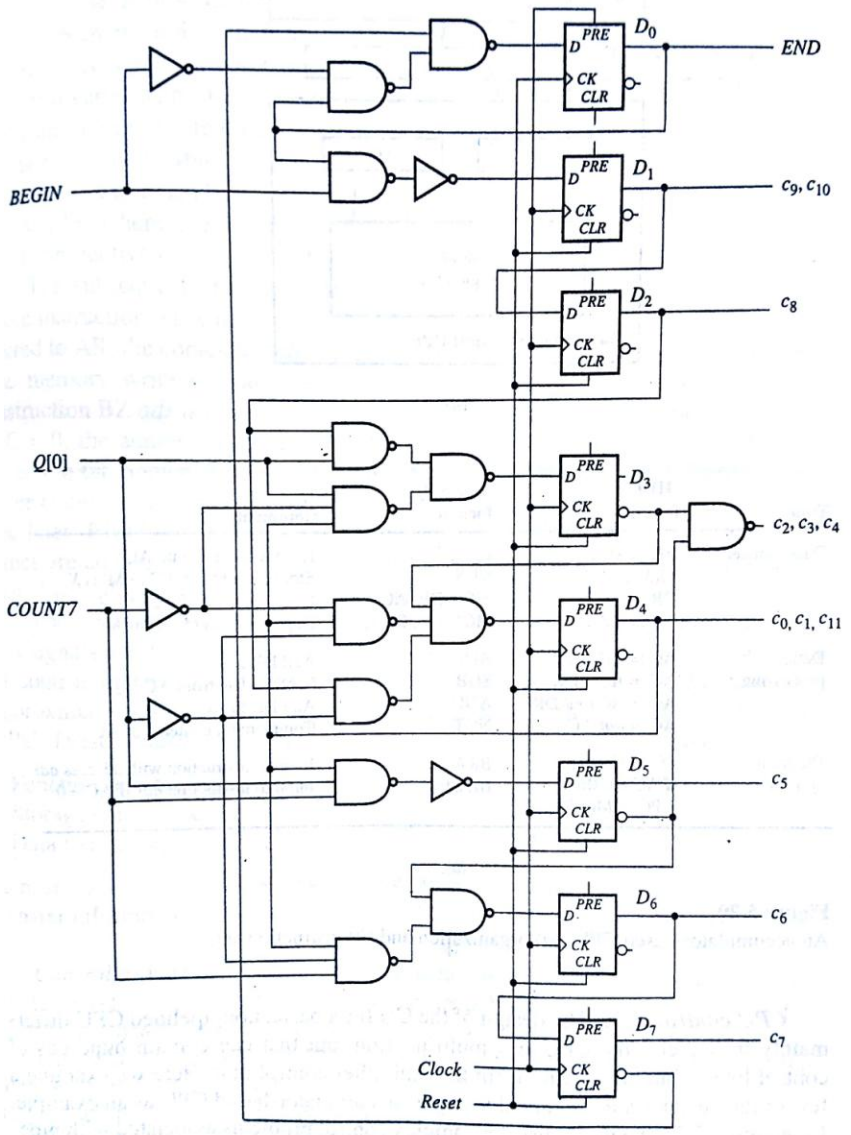


Figure 5.19
All-NAND one-hot design for the multiplier control unit.

MULTIPLIER CONTROL : HARDWIRED CONTROL

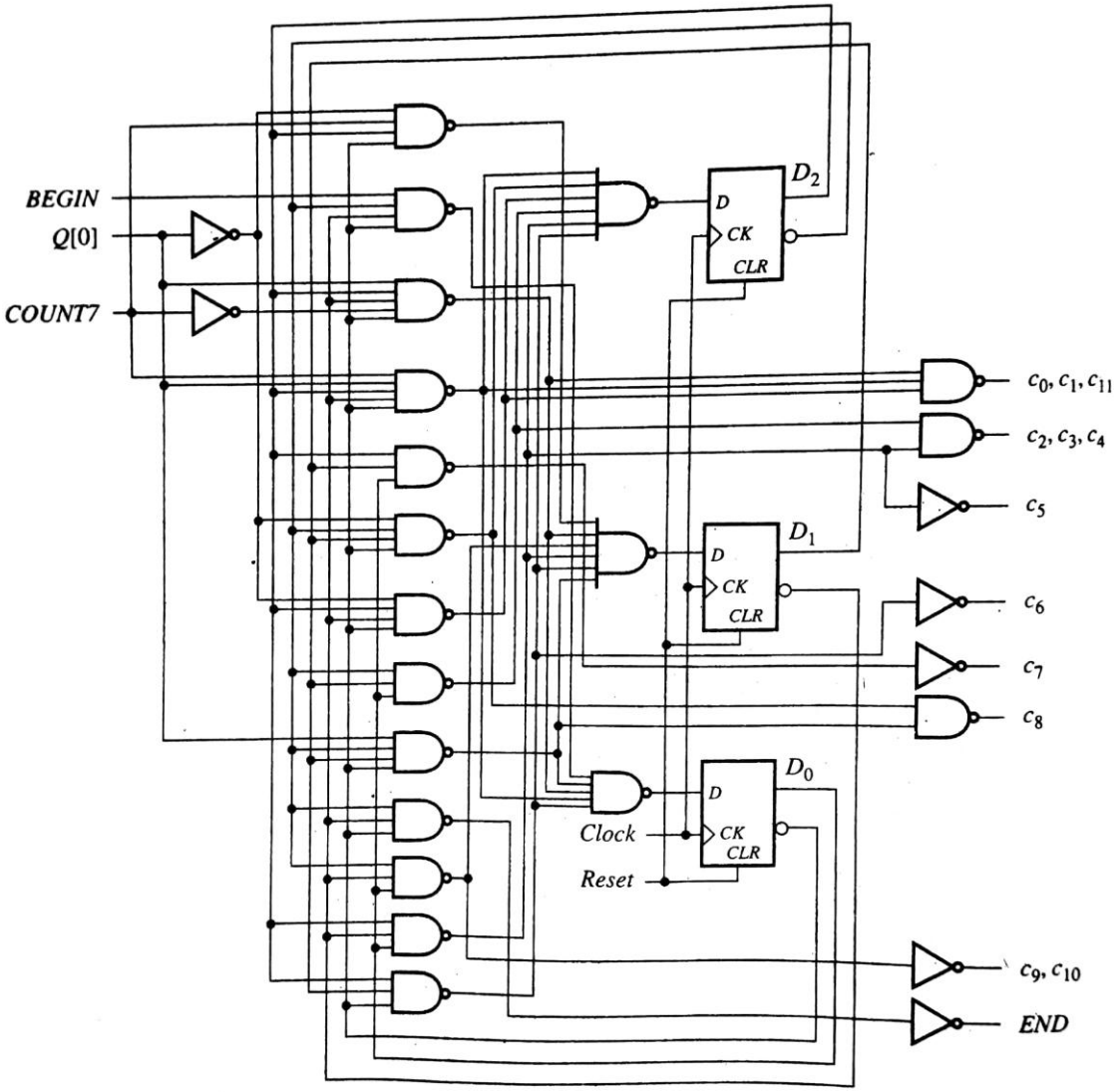


Figure 5.18
All-NAND classical design for the multiplier control unit.

MULTIPLIER CONTROL : MICROPROGRAMMED CONTROL

➢ Figure 4.13 shows the HDL description of the multiplier in a format in which every statement corresponds to a distinct microinstruction, implying that a microprogram of 10 microinstructions is sufficient.

```
2Cmultiplier    (in: INBUS; out: OUTBUS);
                register A[7:0], M[7:0], Q[7:0], COUNT[2:0], F;
                bus INBUS[7:0], OUTBUS[7:0];
BEGIN:          A := 0, COUNT := 0, F := 0,
INPUT:          M := INBUS;
                Q := INBUS;
ADD:            A[7:0] := A[7:0] + M[7:0] × Q[0],
                F := (M[7] and Q[0]) or F;
RSHIFT:         A[7] := F, A[6:0].Q := A.Q[7:1], COUNT := COUNT + 1;
TEST:           if COUNT ≠ 7 then go to ADD;
SUBTRACT:       A[7:0] := A[7:0] – M[7:0] × Q[0], Q[0] := 0;
OUTPUT:         OUTBUS := Q;
                OUTBUS := A;
end 2Cmultiplier;
```

Figure 4.13
HDL description of the multiplier for 8-bit twos-complement fractions.

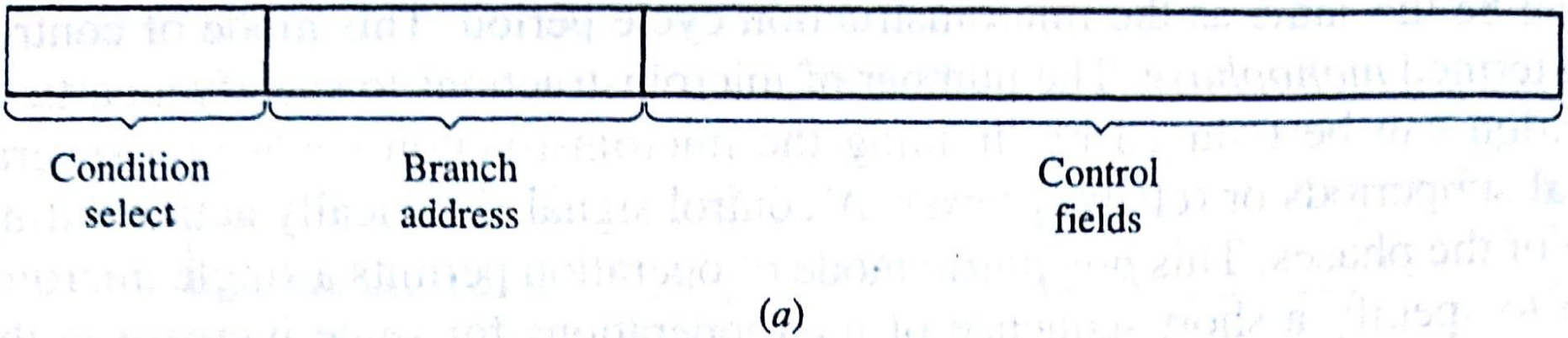
MULTIPLIER CONTROL : MICROPROGRAMMED CONTROL

Address	Microoperations	Control signals activated
BEGIN:	A := 0, COUNT := 0, F := 0, M := INBUS;	c_9, c_{10}
INPUT:	Q := INBUS;	c_8
TEST1:	**if** Q[0] = 0 **then go to** RSHIFT:	
ADD:	A[7:0] := A[7:0] + M[7:0], F := (M[7] ***and*** Q[0]) ***or*** F;	c_2, c_3, c_4
RSHIFT:	A[7] := F, A[6:0].Q := A.Q[7:1], COUNT := COUNT + 1, **if** COUNT7 = 0 **then go to** TEST1;	c_0, c_1, c_{11}
TEST2:	**if** Q[0] = 0 **then go to** OUTPUT1;	
SUBTRACT:	A[7:0] := A[7:0] – M[7:0], Q[0] := 0;	c_2, c_3, c_4, c_5
OUTPUT1:	OUTBUS := A;	c_6
OUTPUT2:	OUTBUS := Q;	c_7
END:	Halt;	END

(*a*)

MULTIPLIER CONTROL : MICROPROGRAMMED CONTROL

- We use the microinstruction format of Figure 5.33a.



(a)

- It has three parts : a condition select field, a branch address and a set of control fields.
- An address field of 4 bits can address up to 16 microinstructions.
- No encoding of control signals is carried out (thirteen 1-bit control field, one for each of the control lines c0, c1,…, c11, END).
- The control unit has the general organization of Figure 5.33b.

MULTIPLIER CONTROL : MICROPROGRAMMED CONTROL

- The control unit has the general organization of Figure 5.33b.

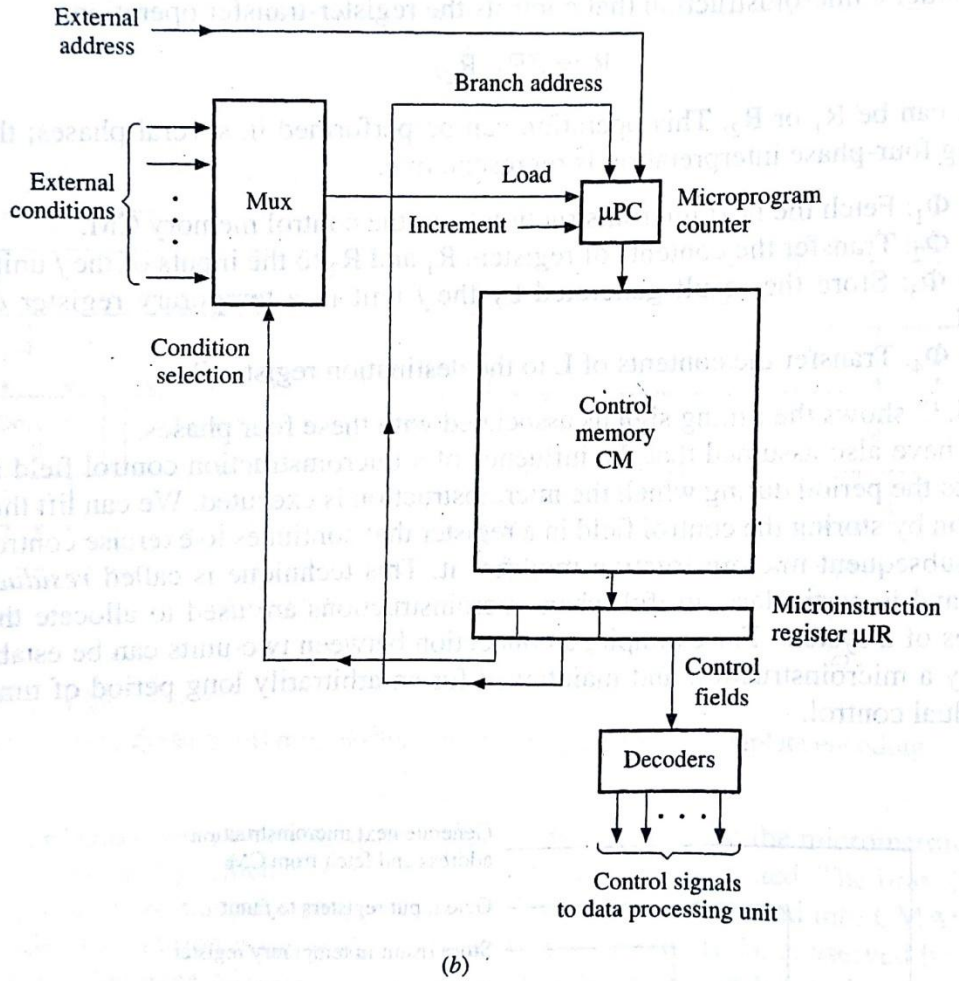


Figure 5.33
Typical microprogrammed controller: (*a*) microinstruction format and (*b*) control unit organization.

MULTIPLIER CONTROL : MICROPROGRAMMED CONTROL

- Micro program counter μPC acts as a control memory address register.
- During each microinstruction cycle μPC is incremented to produce the address of the next microinstruction.
- In the case of a branch microinstruction, the address stored in the current microinstruction is the branch address.
- We eliminate the need for an external address input by storing the first microinstruction in address 0 of CM and simply resetting μPC to 0 at the start of multiplication.
- Every microinstruction can specify a branch address and so can implement a conditional or unconditional branch.
- The condition-select field has to indicate one of the four conditions:

1. No branching (denoted by 00)
2. Branch if Q[0]=0 (denoted by 01)
3. Branch if COUNT7=0 (denoted by 10)
4. Unconditional Branch (denoted by 11)

MULTIPLIER CONTROL : MICROPROGRAMMED CONTROL

- Hence a 2-bit condition-select field is needed.
- We conclude that a 19-bit microinstruction word is sufficient.
- Consecutive microinstructions are assigned to consecutive addresses, and the appropriate condition-select bits are inserted.
- When multiplication is completed, the microprogram enters a waiting (halt) state by repeatedly executing the no-operation microinstruction in CM location 1001.
- It remains in this state until μPC is reset by the arrival of an external BEGIN signal.

MULTIPLIER CONTROL : MICROPROGRAMMED CONTROL

	Address in CM	Condition select	Branch address	Control fields												
				c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	END
S_0	0000	00	0000	0	0	0	0	0	0	0	0	0	1	1	0	0
S_1	0001	00	0000	0	0	0	0	0	0	0	0	1	0	0	0	0
S_2	0010	01	0100	0	0	0	0	0	0	0	0	0	0	0	0	0
S_3	0011	00	0000	0	0	1	1	1	0	0	0	0	0	0	0	0
S_4	0100	10	0010	1	1	0	0	0	0	0	0	0	0	0	1	0
S_5	0101	01	0111	0	0	0	0	0	0	0	0	0	0	0	0	0
S_6	0110	00	0000	0	0	1	1	1	1	0	0	0	0	0	0	0
S_7	0111	00	0000	0	0	0	0	0	0	1	0	0	0	0	0	0
S_8	1000	00	0000	0	0	0	0	0	0	0	1	0	0	0	0	0
S_9	1001	11	1001	0	0	0	0	0	0	0	0	0	0	0	0	1

(*b*)

Figure 5.36
(*a*) Symbolic and (*b*) binary microprogram for twos-complement multiplication.

MULTIPLIER CONTROL : MICROPROGRAMMED CONTROL

Address	Microoperations	Control signals activated
BEGIN:	A := 0, COUNT := 0, F := 0, M := INBUS;	c_9, c_{10}
INPUT:	Q := INBUS;	c_8
TEST1:	**if** Q[0] = 0 **then go to** RSHIFT:	
ADD:	A[7:0] := A[7:0] + M[7:0], F := (M[7] ***and*** Q[0]) ***or*** F;	c_2, c_3, c_4
RSHIFT:	A[7] := F, A[6:0].Q := A.Q[7:1], COUNT := COUNT + 1, **if** COUNT7 = 0 **then go to** TEST1;	c_0, c_1, c_{11}
TEST2:	**if** Q[0] = 0 **then go to** OUTPUT1;	
SUBTRACT:	A[7:0] := A[7:0] – M[7:0], Q[0] := 0;	c_2, c_3, c_4, c_5
OUTPUT1:	OUTBUS := A;	c_6
OUTPUT2:	OUTBUS := Q;	c_7
END:	Halt;	END

(*a*)

MULTIPLIER CONTROL : MICROPROGRAMMED CONTROL

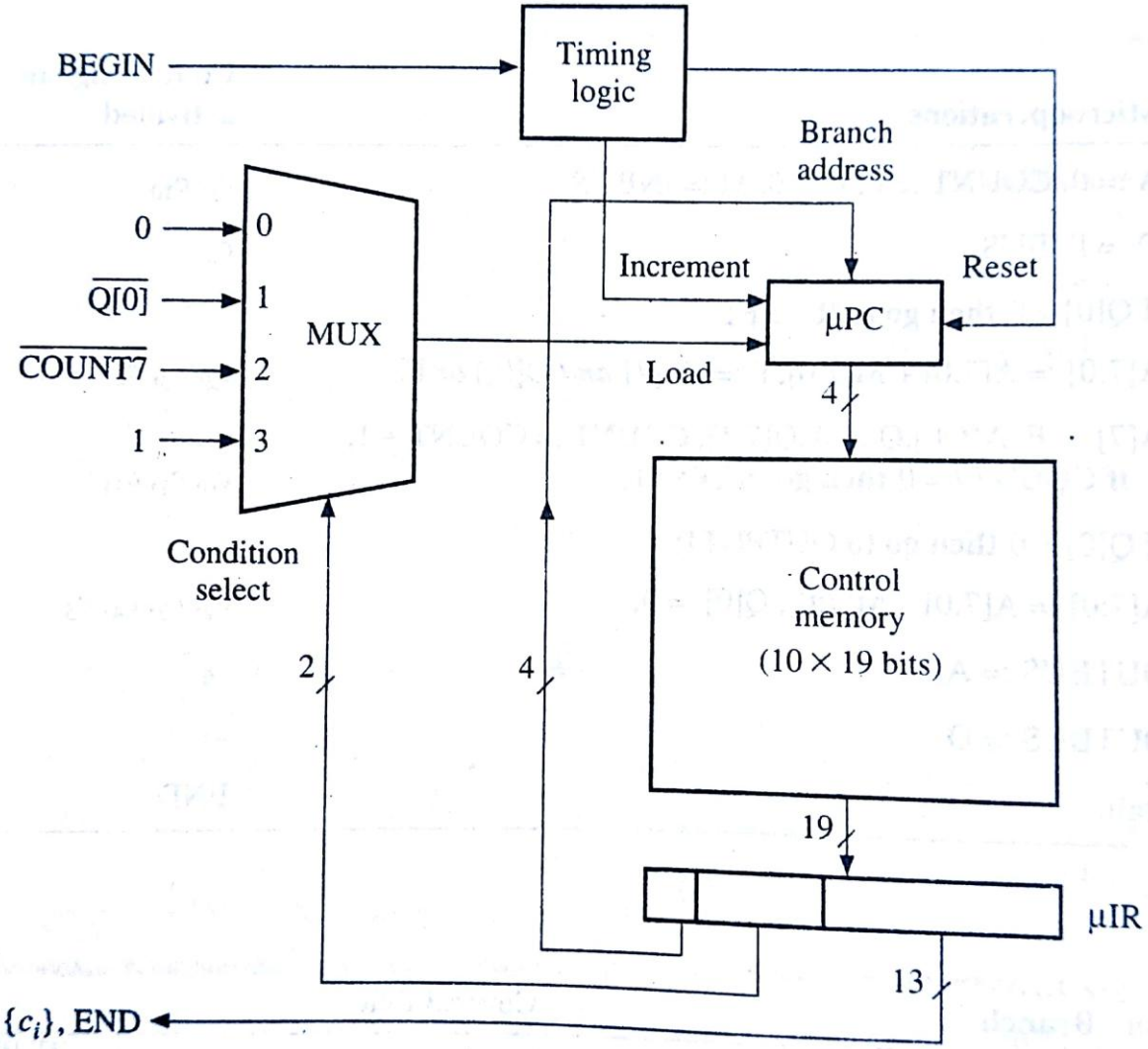


Figure 5.37
Microprogrammed control unit for the twos-complement multiplier.

MULTIPLIER CONTROL : MICROPROGRAMMED CONTROL

➢**Control-field encoding:** Very few of the 2^{13} possible control-field patterns allowed by the microinstruction format are actually useful or needed.

➢Several sets of control signals are always activated simultaneously; hence a single 1-bit control field suffices for each such set.

➢The 8 bits reserved for the 3 sets {c_0, c_1, c_{11}}, {c_2, c_3, c_4} and {c_9, c_{10}} can be replaced by 3 bits, yielding the short, horizontal format.

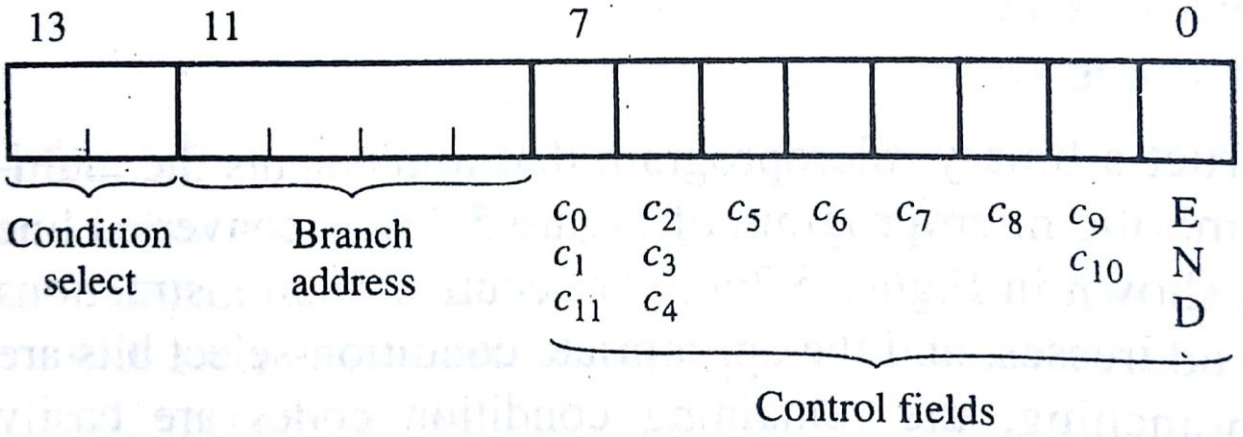


Figure 5.38
Horizontal microinstruction format after removing redundant control fields.

MULTIPLIER CONTROL : MICROPROGRAMMED CONTROL

- The number of control bits can be reduced further by encoding the control fields.
- Since there are only 10 distinct microinstructions in the multiplication microprogram, we can encode the control signals in a single 4-bit control field, yielding a purely vertical format.
- However, this design severely limits our ability to subsequently modify the microinstruction set.

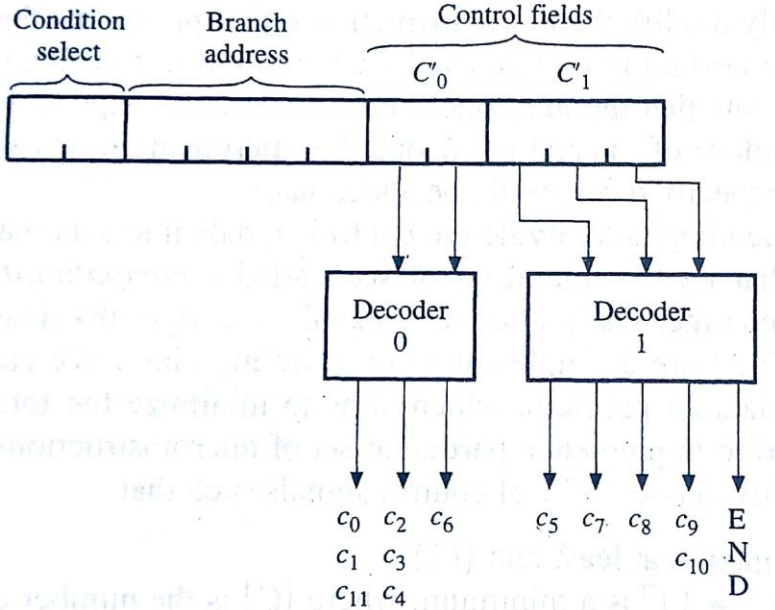


Figure 5.39 Vertical microinstruction format with maximum parallelism and minimum control-field width.

MULTIPLIER CONTROL : MICROPROGRAMMED CONTROL

➢**Encoding by function :** In figure 5.39, functionally unrelated control signals are combined in the same control field, while related signals are derived from different control fields.

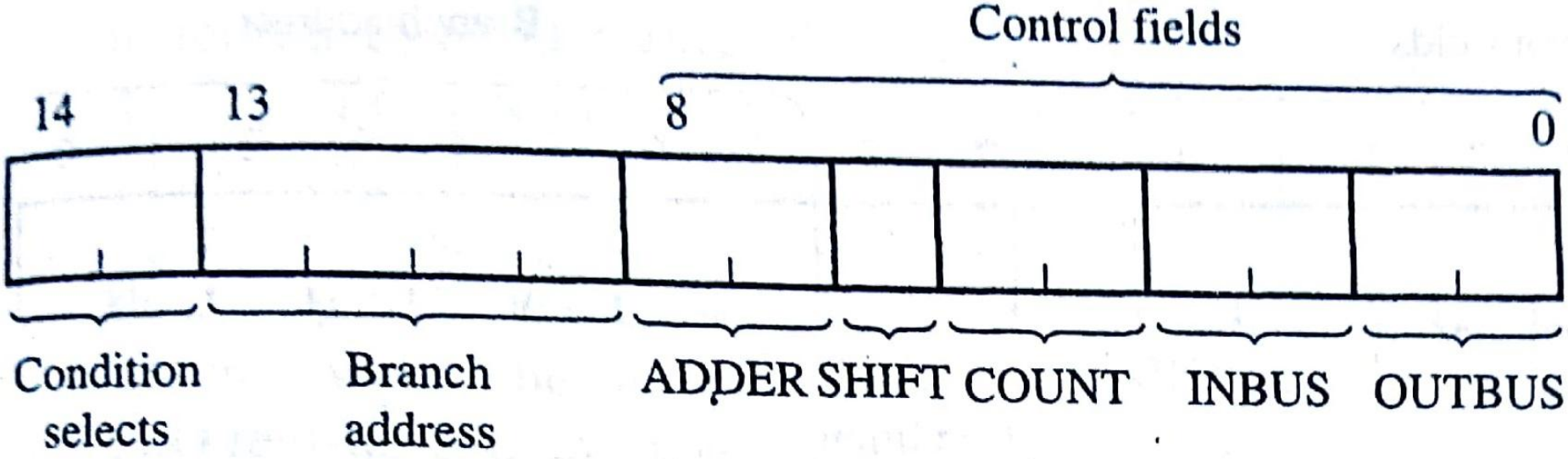
➢This lack of functional separation makes the writing of microprograms difficult, since the microprogrammer must associate several unrelated opcodes with each control field.

➢A encoded format in which each control field specifies the control signals for one component or for a related set of operations is preferred, even though more control bits may be needed.

➢ On examining the multiplier circuit, we see that there are five components to be controlled: the adder, the A.Q register-pair, the external iteration counter COUNT, and the two data buses INBUS and OUTBUS.

➢Each component has its own set of functions, suggesting the encoding-by-function format.

MULTIPLIER CONTROL : MICROPROGRAMMED CONTROL



(a)

MULTIPLIER CONTROL : MICROPROGRAMMED CONTROL

Control field	Bits used	Code	Microoperations specified	Control signals activated
ADDER	7,8	00	No operation	
		01	A := A + M, set F	c_2,c_3,c_4
		10	A := A – M, Q[0] := 0	c_2,c_3,c_4,c_5
		11	Unused	
SHIFT	6	0	No operation	
		1	Right-shift A.Q, set A[7]	c_0,c_1
COUNT	5,4	00	No operation	
		01	Clear COUNT, A, F	c_{10}
		10	COUNT := COUNT + 1	c_{11}
		11	Unused	
INBUS	3,2	00	No operation	
		01	Q := INBUS	c_8
		10	M := INBUS	c_9
		11	Unused	
OUTBUS	1,0	00	No operation	
		01	OUTBUS := A	c_6
		10	OUTBUS := Q	c_7
		11	Unused	

(*b*)

Figure 5.40
(*a*) Microinstruction format with control fields encoded by function and (*b*) their interpretation.

MULTIPLIER CONTROL : MICROPROGRAMMED CONTROL

➢**Multiple microinstruction formats:** In the original multiplication microprogram, several microinstructions are used only for next-address generation and do not activate any control lines.

➢ We can reduce microinstructions size by using a single field to contain either control information or address information.

➢Two distinct microinstruction types:

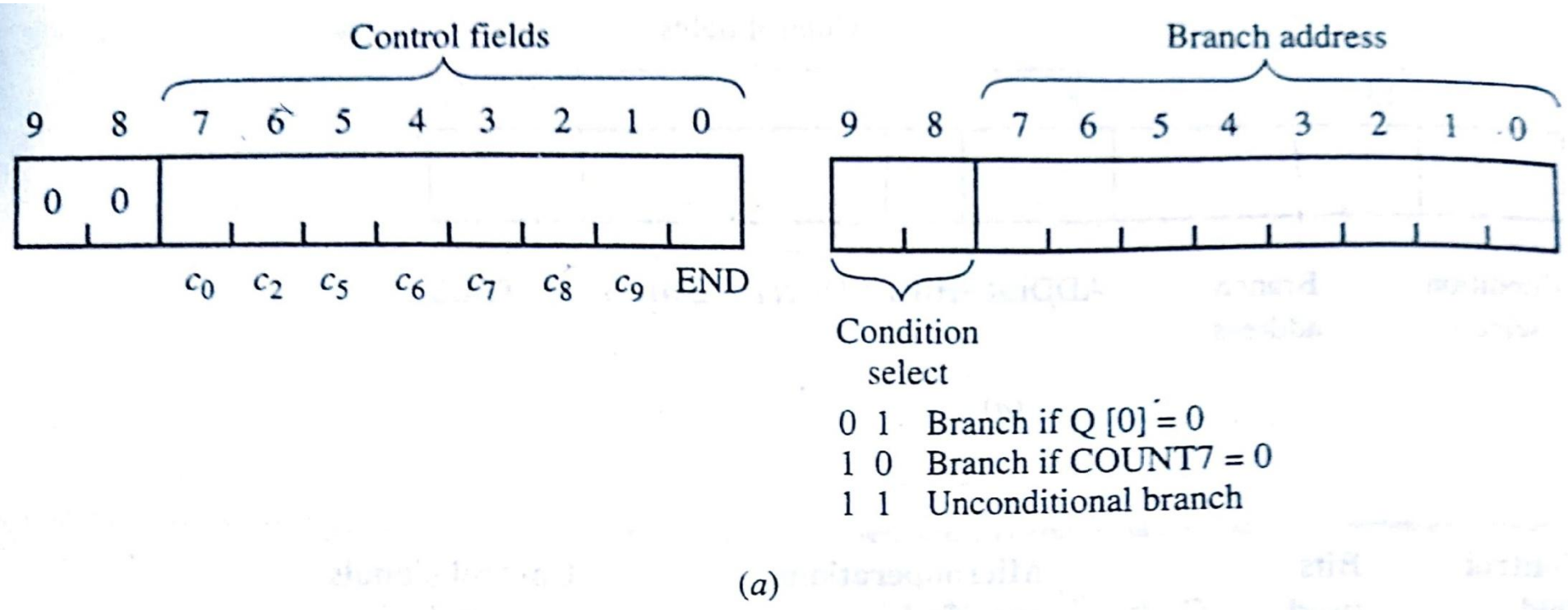
1. Branch instructions, which specify no control information
2. Action or operate microinstructions, which activate control lines but have no branching capability.

➢ Now we define a microinstruction format having two parts, a 2-bit condition-select field with the same meaning as before, and an 8-bit field that can contain either a branch address or control information.

➢ The condition-select code 00, which denotes no branching, serves to identify the operate microinstructions.

MULTIPLIER CONTROL : MICROPROGRAMMED CONTROL

- Remaining three select field codes identify conditional and unconditional branches.



(a)

MULTIPLIER CONTROL : MICROPROGRAMMED CONTROL

➢Microprogram for twos-complement multiplication using the format is given below,

Address in CM	Condition select 9	8	Branch address or control bits 7	6	5	4	3	2	1	0	Comment
0000	0	0	0	0	0	0	0	0	1	0	BEGIN
0001	0	0	0	0	0	0	0	1	0	0	INPUT
0010	0	1	0	0	0	0	0	1	0	0	TEST1
0011	0	0	0	1	0	0	0	0	0	0	ADD
0100	0	0	1	0	0	0	0	0	0	0	RSHIFT
0101	1	0	0	0	0	0	0	0	1	0	RSHIFT BRANCH
0110	0	1	0	0	0	0	1	0	0	0	TEST2
0111	0	0	0	1	1	0	0	0	0	0	SUBTRACT
1000	0	0	0	0	0	1	0	0	0	0	OUTPUT1
1001	0	0	0	0	0	0	1	0	0	0	OUTPUT2
1010	0	0	0	0	0	0	0	0	0	1	END
1011	1	1	0	0	0	0	1	0	1	1	HALT

(*b*)

Figure 5.41
(*a*) Multiple microinstruction formats and (*b*) multiplication microprogram that uses these formats.

MULTIPLIER CONTROL : MICROPROGRAMMED CONTROL

Address	Microoperations	Control signals activated
BEGIN:	A := 0, COUNT := 0, F := 0, M := INBUS;	c_9, c_{10}
INPUT:	Q := INBUS;	c_8
TEST1:	**if** Q[0] = 0 **then go to** RSHIFT:	
ADD:	A[7:0] := A[7:0] + M[7:0], F := (M[7] ***and*** Q[0]) ***or*** F;	c_2, c_3, c_4
RSHIFT:	A[7] := F, A[6:0].Q := A.Q[7:1], COUNT := COUNT + 1, **if** COUNT7 = 0 **then go to** TEST1;	c_0, c_1, c_{11}
TEST2:	**if** Q[0] = 0 **then go to** OUTPUT1;	
SUBTRACT:	A[7:0] := A[7:0] – M[7:0], Q[0] := 0;	c_2, c_3, c_4, c_5
OUTPUT1:	OUTBUS := A;	c_6
OUTPUT2:	OUTBUS := Q;	c_7
END:	Halt;	END

(*a*)

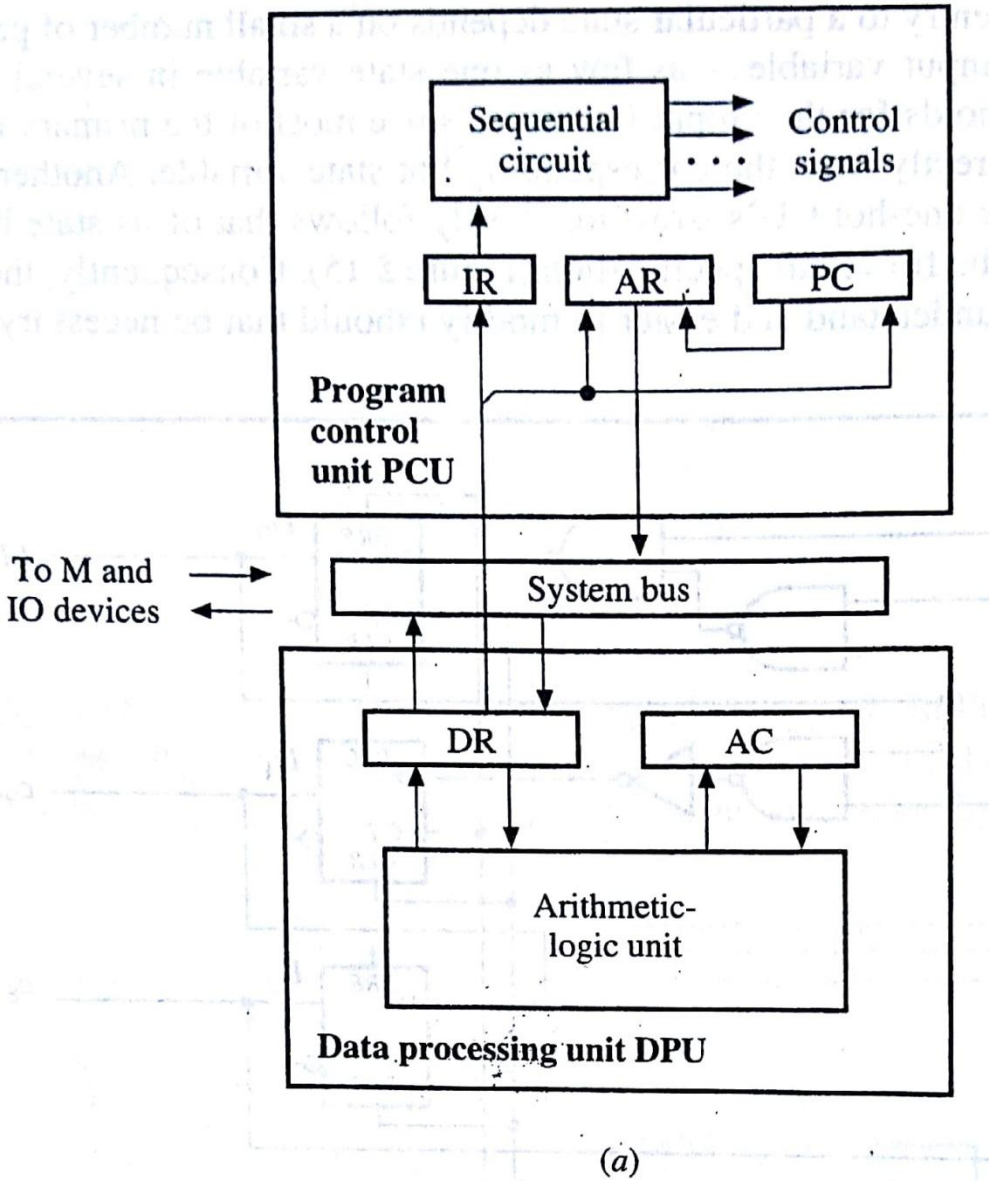
MULTIPLIER CONTROL : MICROPROGRAMMED CONTROL

- The condition-select field is used to control a demultiplexer that routes bits 0:7 either to external control lines (operate microinstructions) or to the branch address logic (branch microinstructions).

CPU CONTROL UNIT :

- The CPU is a multifunctional unit that can contain hundreds of control lines.
- An accumulator based CPU.
- This CPU consists of a datapath unit DPU designed to execute the set of 10 basic single-address instructions.
- The instructions are assumed to be of fixed length and to act on data words of the same fixed length, say 32 bits.
- The program control unit PCU is responsible for managing the control signals linking the PCU to the DPU, as well as the control signals between the CPU and the external memory M.
- To design the PCU, we must first identify the relevant control actions (micro operations) needed to process the given instruction set.

CPU CONTROL UNIT :



(a)

CPU CONTROL UNIT :

Type	HDL format	Assembly format	Comment
Data transfer	AC := M(*X*)	LD X	Load *X* from M into AC
	M(*X*) := AC	ST X	Store contents of AC in M as *X*
	DR := AC	MOV DR, AC	Copy contents of AC to DR
	AC := DR	MOV AC, DR	Copy contents of DR to AC
Data processing	AC := AC + DR	ADD	Add DR to AC
	AC := AC – DR	SUB	Subtract DR from AC
	AC := AC ***and*** DR	AND	And DR to AC
	AC := ***not*** AC	NOT	Complement contents of AC
Program control	PC := M(*adr*)	BRA adr	Jump to instruction with address *ad*
	if AC = 0 **then** PC := M(*adr*)	BZ adr	Jump to instruction *adr* if AC = 0

(*b*)

Figure 5.20
An accumulator-based CPU: (*a*) organization and (*b*) instruction set.

CPU CONTROL UNIT :

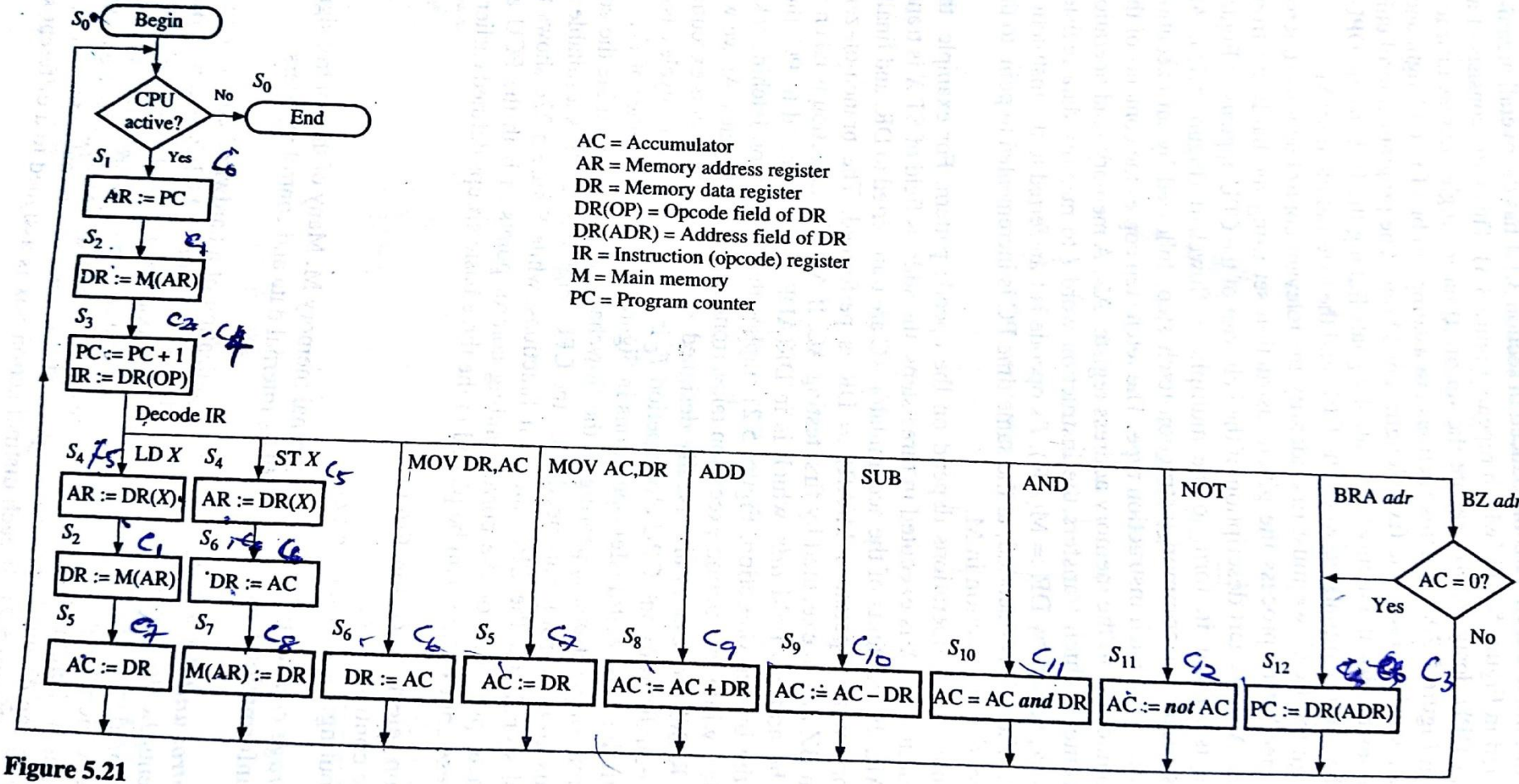


Figure 5.21
Flowchart of the accumulator-based CPU.

CPU CONTROL UNIT :

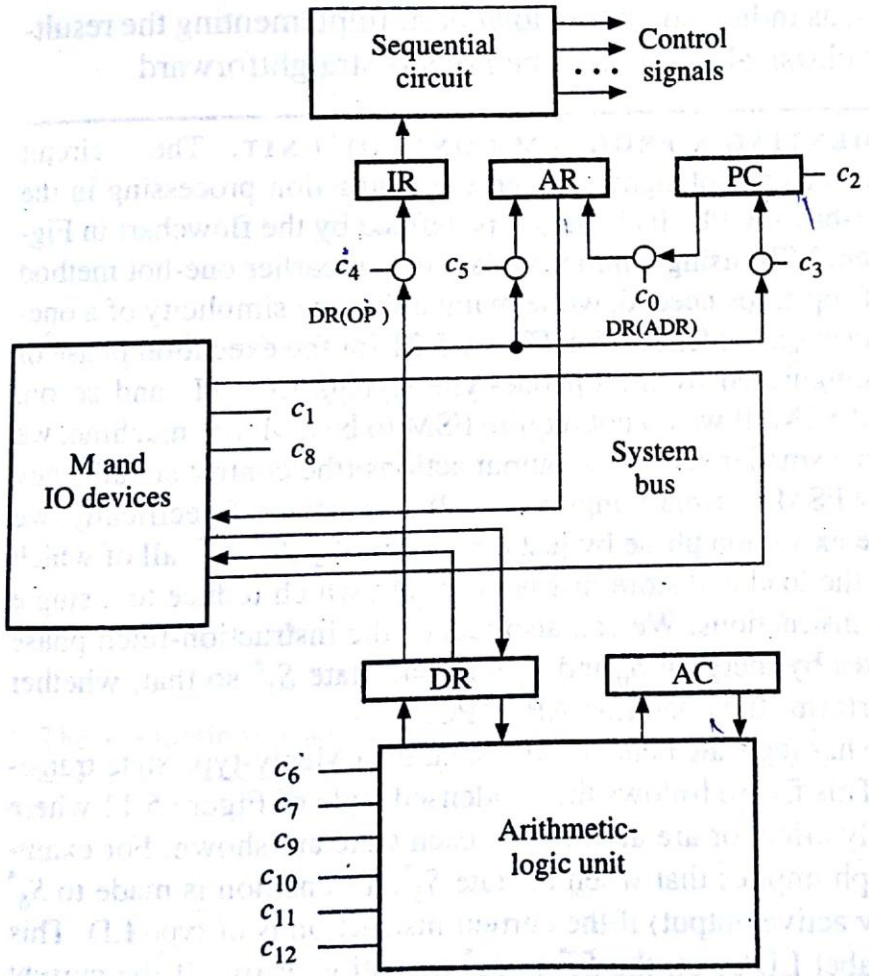
- All instructions require a common instruction-fetch step, followed by an execution step that varies with each instruction type.
- The fetch step copies the contents of the program counter PC to the memory address register AR.
- A memory read operation is then executed, which transfers the instruction word I to memory data register DR; this is expressed by DR := M(AR).
- I's opcode is transferred to the instruction register IR, where it is decoded; at the same time PC is incremented to point to the next consecutive instruction in M.
- The subsequent operations depend on the opcode pattern.
- Eg: The store instruction ST X is executed in three steps :

1. The address field of ST X is transferred to AR
2. The contents of accumulator AC are transferred to DR
3. The memory write operation M(AR) := DR is performed.

CPU CONTROL UNIT :

- The branch-on-zero instruction BZ adr is executed by first testing AC.
- If AC ≠ 0, no action is taken; if AC = 0, the address field adr, which is in DR(ADR), is transferred to PC, thus effecting the branch operation.
- The instruction fetching takes three cycles, while instruction execution takes from one to three cycles.
- In RISC processor, all instruction execution times are equalized to one CPU clock period T_c in length, making the cycles associated with the register-transfer operations into subcycles of T_c.

CPU CONTROL UNIT :



(a)

Control signal	Operation controlled
c_0	AR := PC
c_1	DR := M(AR)
c_2	PC := PC + 1
c_3	PC := DR(ADR)
c_4	IR := DR(OP)
c_5	AR := DR(ADR)
c_6	DR := AC
c_7	AC := DR
c_8	M(AR) := DR
c_9	AC := AC + DR
c_{10}	AC := AC – DR
c_{11}	AC := AC *and* DR
c_{12}	AC := *not* AC

(b)

Figure 5.22
(*a*) Control points and (*b*) control signal definitions for the accumulator-based CPU.

CPU CONTROL UNIT :

- The control lines can be placed in the three basic groups,

1. Function select – c_2, c_9, c_{10}, c_{11}, c_{12}
2. Storage control – c_1, c_8
3. Data routing – c_0, c_3, c_4, c_5, c_6, c_7.

- Here storage control refers to the external memory M.
- Many of the control signals transfer information between the CPU's internal data and control registers.

CPU CONTROL UNIT : HARDWIRED CONTROL

- The overall organization of a hardwired control unit is shown below,

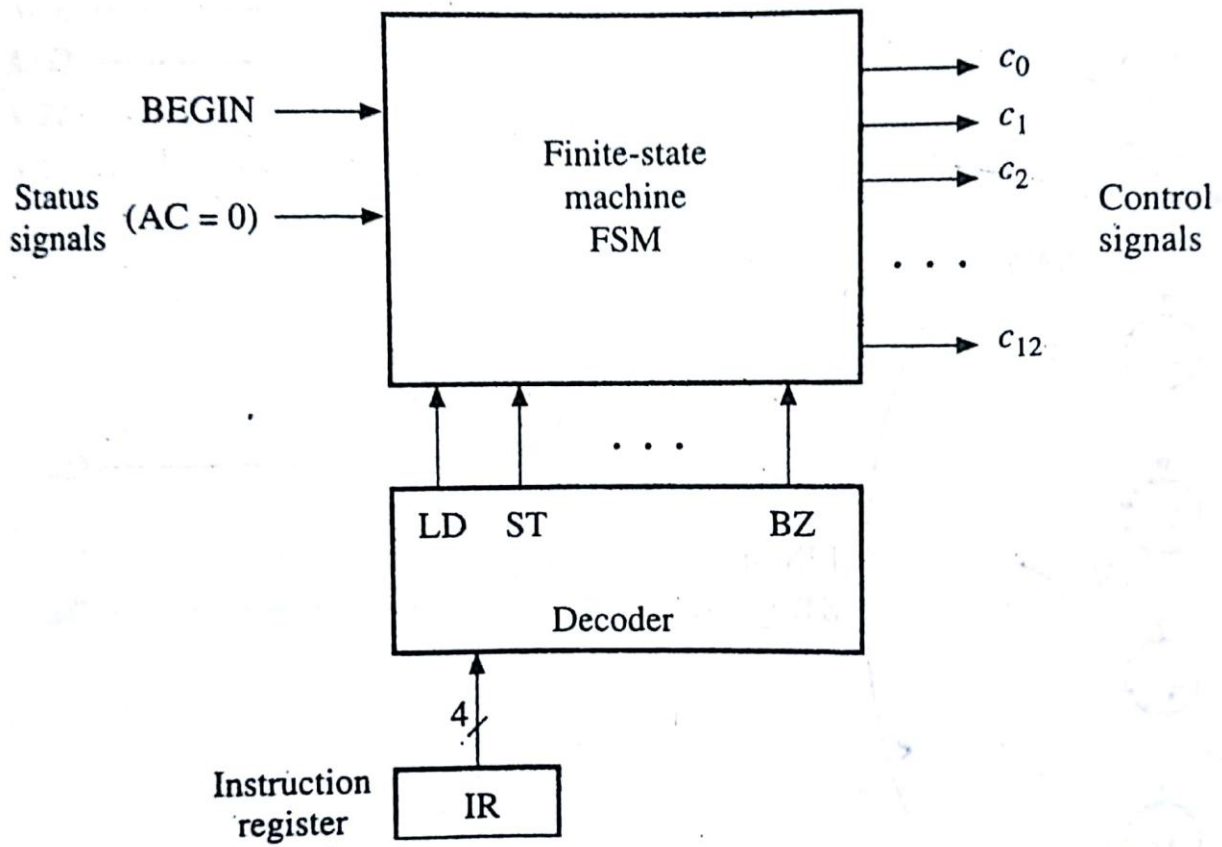


Figure 5.23
Organization of a hardwired control unit for the accumulator-based CPU.

CPU CONTROL UNIT : HARDWIRED CONTROL

- It is assumed that the opcode stored in the instruction register IR is decoded into 10 signals, one per instruction type, which along with BEGIN and a status signal (AC = 0) form the inputs to the main sequential circuit FSM that generates the control signals c_0:c_{12}.
- Hence FSM has 12 primary inputs and 13 primary outputs.
- If each distinct action box is assigned to a different state, then there are 13 states S_0:S_{12}, as indicated on the flowchart.
- Implementing the resulting 13-state machine via the classical or one-hot method is straightforward.

CPU CONTROL UNIT : MICROPROGRAMMED CONTROL

- The microprogram selected to emulate each instruction is identified by the instruction's opcode; hence the contents of the instruction register IR determine the microprogram's starting address.
- We will use the unmodified content of IR as the microprogram address for the current instruction.
- Also assume that each microinstruction can specify a branch condition , a branch address that is used only if the branch condition is satisfied, and a set of control fields defining the micro operations to be performed.

CPU CONTROL UNIT : MICROPROGRAMMED CONTROL

- Complete emulator for the given instruction set in symbolic form.

```
FETCH:  AR := PC;
        DR := M(AR);
        PC := PC + 1, IR := DR(OP);
        go to IR;

LD:     AR := DR(ADR);
        DR := M(AR);
        AC := DR, go to FETCH;

ST:     AR := DR(ADR);
        DR := AC;
        M(AR) := DR, go to FETCH;

MOV1:   DR := AC, go to FETCH;

MOV2:   AC := DR, go to FETCH;

ADD:    AC := AC + DR, go to FETCH;

SUB:    AC := AC – DR, go to FETCH;

AND:    AC := AC and DR, go to FETCH;

NOT:    AC := not AC, go to FETCH;

BRA:    PC := DR(ADR), go to FETCH;

BZ:     if AC = 0 then PC := DR(ADR), go to FETCH;
```

Figure 5.44
A microprogrammed emulator for a small instruction set.

CPU CONTROL UNIT : MICROPROGRAMMED CONTROL

- This emulator contains a distinct microprogram for each of the ten possible instruction execution cycles and another microprogram called FETCH.
- Note how the name of the microprogram corresponds to its address in the emulator code.
- The **go to IR** microoperation is implemented by μPC := IR, which transfers control to the first microinstruction in the microprogram that interprets the current instruction.
- Figure 5.44 assumes that μPC is the default address source for microinstructions and is incremented automatically in every clock cycle.

CPU CONTROL UNIT : MICROPROGRAMMED CONTROL

- If we need to introduce a new instruction called CLEAR whose function is to reset all bits of the accumulator AC to 0.
- Although no control line to clear AC was included in the CPU, we can still write a microprogram to implement CLEAR instruction using only the preexisting microoperations.

```
CLEAR :    DR := AC
           AC := not AC
           AC := AC and DR, go to FETCH
```

- By storing this new microprogram in the control memory, CLEAR can be added to the instruction set with either no changes, or very minor ones, to the CPU hardware.
- Such flexibility is a key advantage of microprogramming over hardwired control.

CPU CONTROL UNIT : MICROPROGRAMMED CONTROL

- **Extension :** Now add to the CPU structure the circuits to implement fixed-point multiplication and division using sequential algorithms.
- Two major new registers are required – a multiplier-quotient register MQ and a counter called COUNT, which counts the number of iterations (add/subtract and shift steps) used during multiplication or division.
- The memory data register DR will be assigned the role of multiplicand or divisor register MD when appropriate.
- Figure 5.45a shows the modified CPU; the number of control signals has more than doubled to 29.
- These signals are denoted c_0:c_{28} and defined in Figure 5.45b.
- c_0:c_{12} corresponds to the control signals of the original CPU.
- Several of the control signals listed implicitly cause flag (status) bits to be set or reset.
- The three flag bits FLAGS, the least significant bit MQ[0] of the multiplier-quotient register, and the signal COUNT =n-1 all serve as branch conditions to be testing bits.

CPU CONTROL UNIT : MICROPROGRAMMED CONTROL

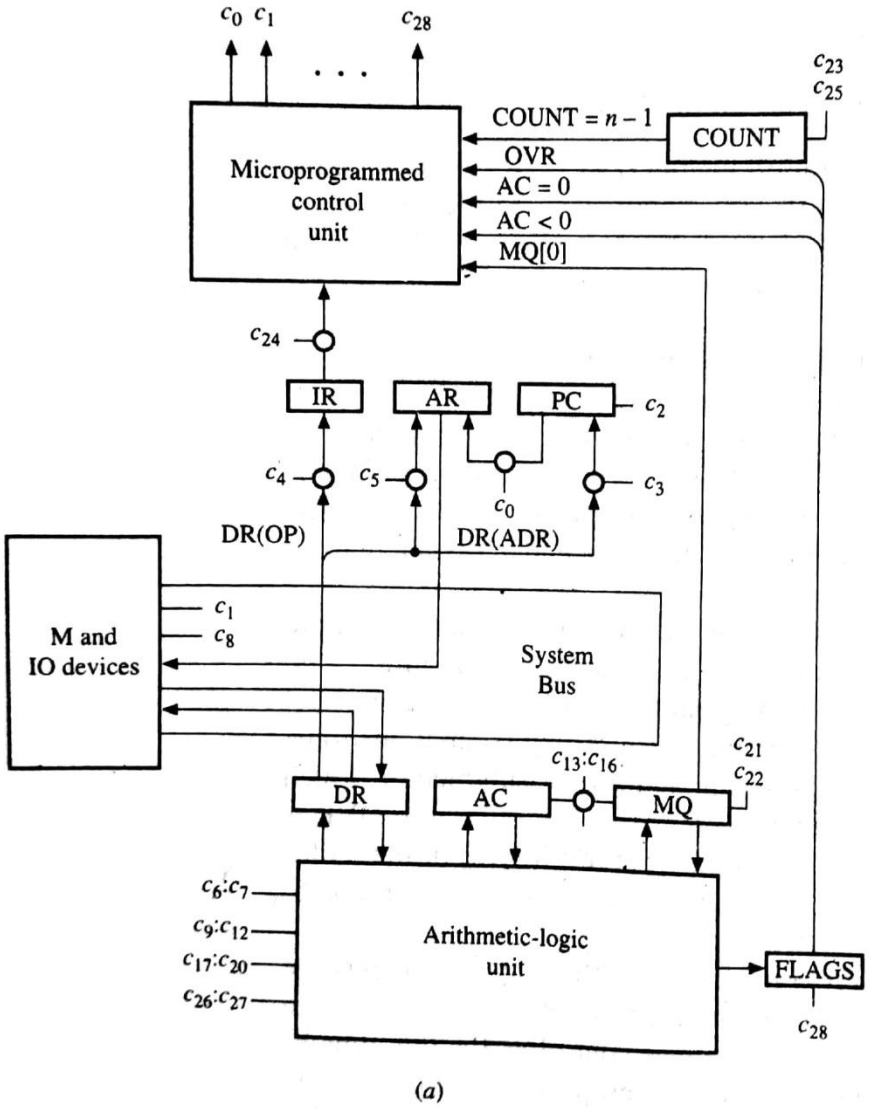


Figure 5.45
(*a*) Control points and (*b*) control signal definitions for the extended CPU.

CPU CONTROL UNIT : MICROPROGRAMMED CONTROL

Control signal	Operation controlled
c_0	AR := PC
c_1	DR := M(AR)
c_2	PC := PC + 1
c_3	PC := DR(ADR)
c_4	IR := DR(OP)
c_5	AR := DR(ADR)
c_6	DR := AC
c_7	AC := DR
c_8	M(AR) := DR
c_9	AC := AC + DR
c_{10}	AC := AC – DR
c_{11}	AC := AC ***and*** DR
c_{12}	AC := ***not*** AC
c_{13}	RSHIFT AC
c_{14}	LSHIFT AC
c_{15}	RSHIFT AC.MQ
c_{16}	LSHIFT AC.MQ
c_{17}	AC := 0
c_{18}	AC[n – 1] := F
c_{19}	MQ := DR
c_{20}	DR := MQ
c_{21}	MQ[0] := 1
c_{22}	MQ[0] := 0
c_{23}	COUNT := COUNT + 1
c_{24}	μPC := IR
c_{25}	COUNT := 0
c_{26}	F := 0
c_{27}	F := 1
c_{28}	FLAGS := 0

(*b*)

CPU CONTROL UNIT : MICROPROGRAMMED CONTROL

- CPU that implements the Robertson multiplication algorithm for twos-complement numbers.
- We assume that before executing this program, the multiplier operand X is placed in MQ and the multiplicand Y is in DR.

Address	Microinstruction
BEGIN:	A := 0, COUNT := 0, F := 0;
TEST1:	**if** MQ[0] = 0 **then go to** RSHIFT:
ADD:	AC := AC + DR, F := (DR[n–1] **and** MQ[0]) **or** F;
RSHIFT:	AC [n–1] := F, AC.MQ := RSHIFT(AC:MQ), COUNT := COUNT + 1, **if** COUNT ≠ n–1 **then go to** TEST1;
TEST2:	**if** MQ[0] = 0 **then go to** FETCH;
SUBTRACT:	AC := AC – DR, MQ[0] := 0, **go to** FETCH;

Figure 5.46
Twos-complement multiplication microprogram *2Cmult* for the extended CPU.

CPU CONTROL UNIT : MICROPROGRAMMED CONTROL

- Each statement represents a single microinstruction.
- The general three-part microinstruction format comprising a condition-select field, a branch-address field and a set of control fields will be used for 2Cmult.
- Five conditions to be tested are identified, AC =0, AC<0, MQ[0], COUNT=n-1 and OVR, the overflow indicator.
- Adding the possibilities of an unconditional branch and no branching, we obtain seven branch-condition codes that can be represented by a 3-bit condition-select field.
- Various control signals can be grouped together in common encoded fields to reduce the microinstruction size.
- Eg: three control signals c_1, c_6 and c_{20} transfer data to DR.
- Since they are mutually exclusive (compatible), we can encode them in a 2-bit field. Note that one more bit pattern must be reserved for the no-operation case.
- Similarly we can combine the many control signals that alter the contents of AC.

CPU CONTROL UNIT : MICROPROGRAMMED CONTROL

- Decided not to encode the control signals.
- condition-select (3-bit encoded form) and control fields (29-bit unencoded form) occupy 32 bits of each microinstruction.
- Suppose further that an 8-bit branch address denoting a complete CM address is included in each microinstruction; a CM storing up to 256 forty-bit words is therefore supported.
- In addition, the μPC can be loaded from the instruction register IR via a logic circuit K (typically a ROM or PLA), which maps instruction opcodes onto microinstruction addresses.

CPU CONTROL UNIT : MICROPROGRAMMED CONTROL

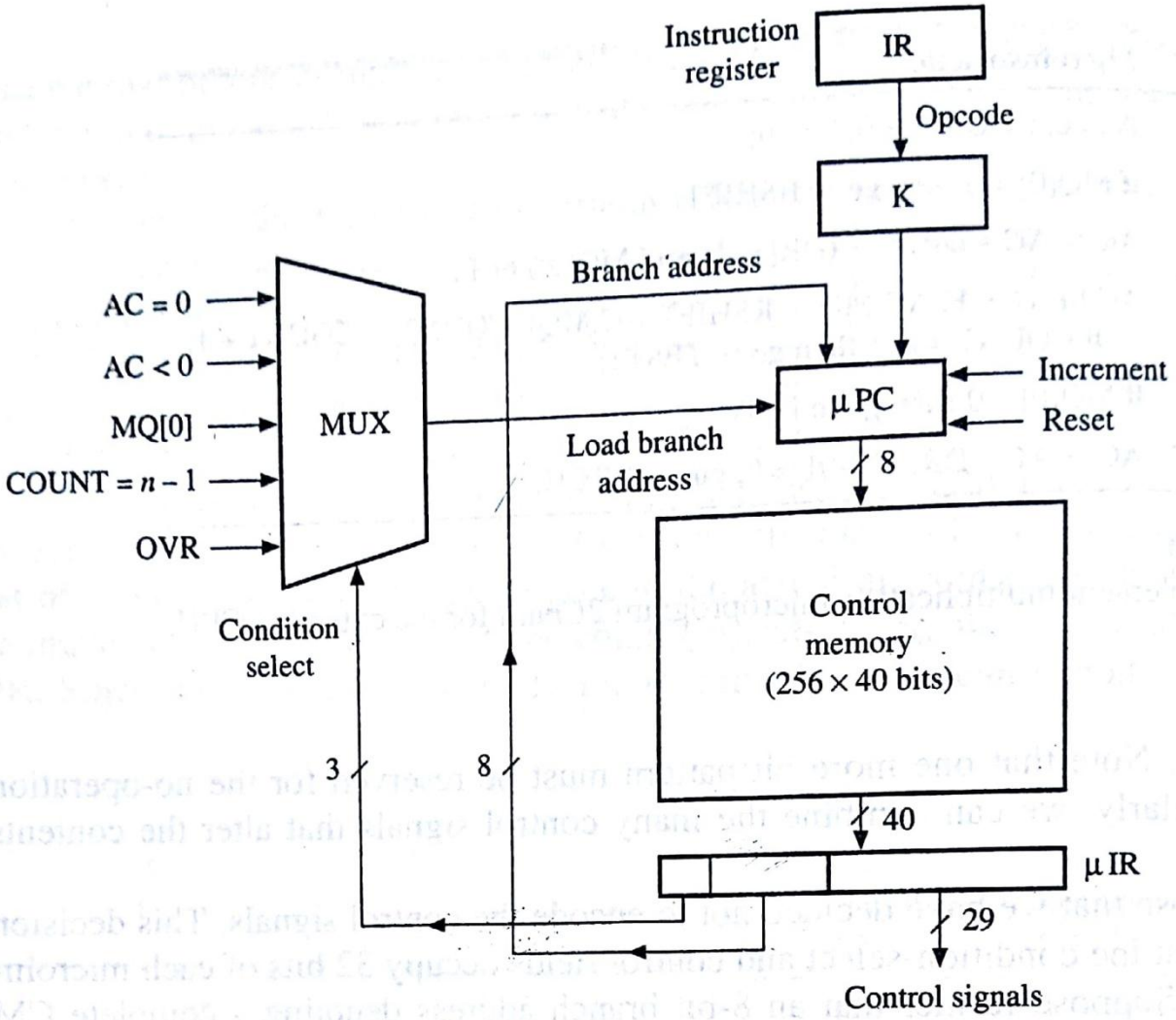


Figure 5.47
Microprogrammed control unit for the extended CPU.

NANOPROGRAMMING

- In most microprogrammed processors, an instruction fetched from memory is interpreted by a microprogram stored in a single control memory CM.
- In a few machines, however the microinstructions do not directly issue the signals that control the hardware.
- Instead, they are used to access a second control memory called a **nanocontrol memory nCM** that directly controls the hardware.
- There are two levels of control memories, a higher-level one termed a **microcontrol memory µCM** whose contents are microinstructions and the lower-level nCM that stores **nanoinstructions**.

NANOPROGRAMMING

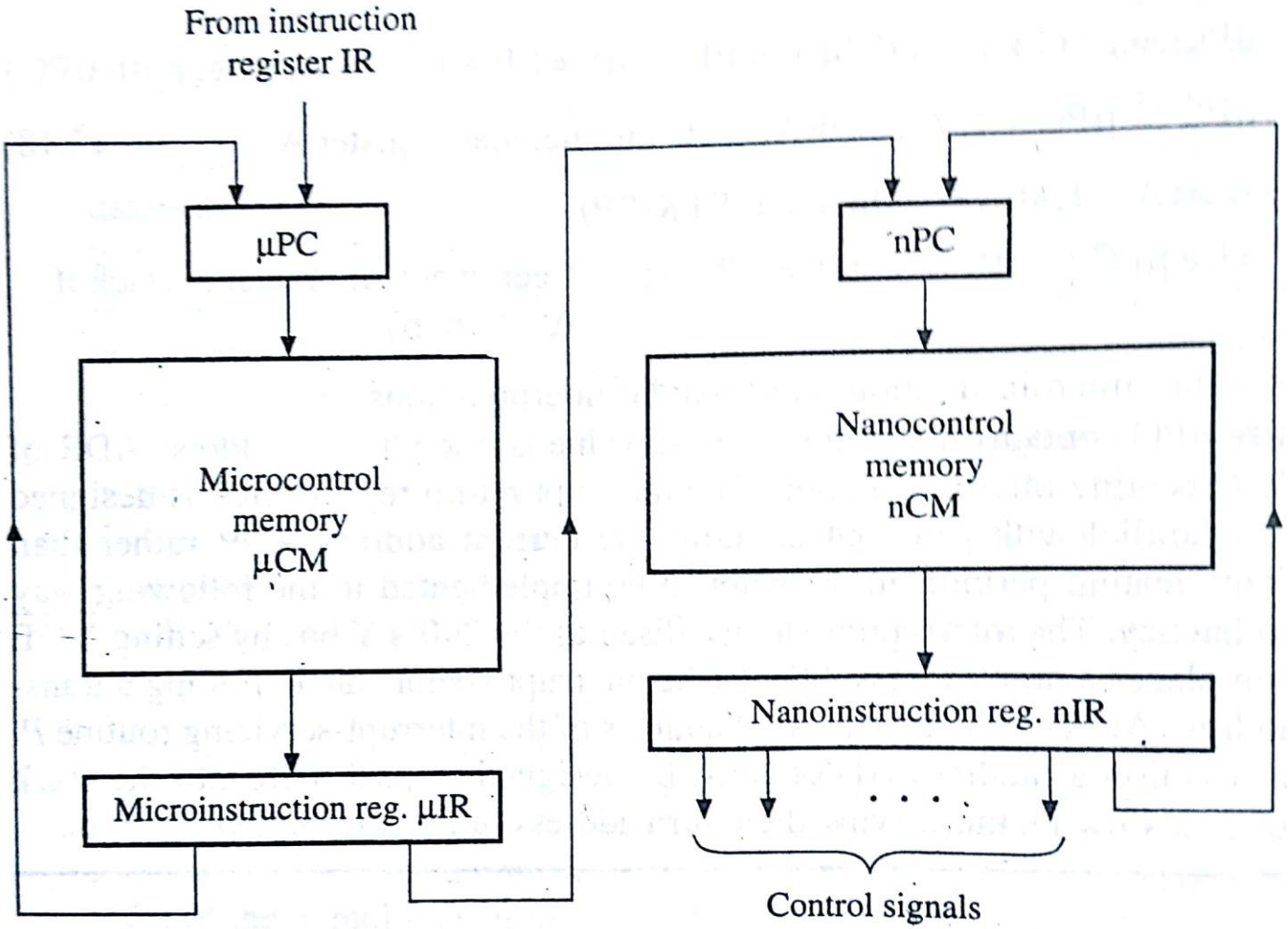


Figure 5.51
Two-level control store organization for nanoprogramming.

NANOPROGRAMMING

- Consider a nanoprogrammed computer in which μCM and nCM have dimensions H_m X W_m and H_n X W_n, respectively.
- The advantage of this two-level control design technique is that it can reduce the total size $S_2=H_m$ X W_m + H_n X W_n of the control memories, which translates to smaller chip area.
- Typically , the microprograms are encoded in a narrow vertical format so that although H_m is large, W_m is small.
- Nanoinstructions, on the other hand, usually have a highly parallel horizontal format making W_n large.
- If one nanoprogram can interpret many microinstructions, then H_n can be kept relatively small so that $S_2<S_1=H_m$ X W_n, which is roughly the size of a comparable single level control unit.
- Disadvantage – loss of speed due to the extra memory access for nCM and a more complex control-unit organization.

NANOPROGRAMMING

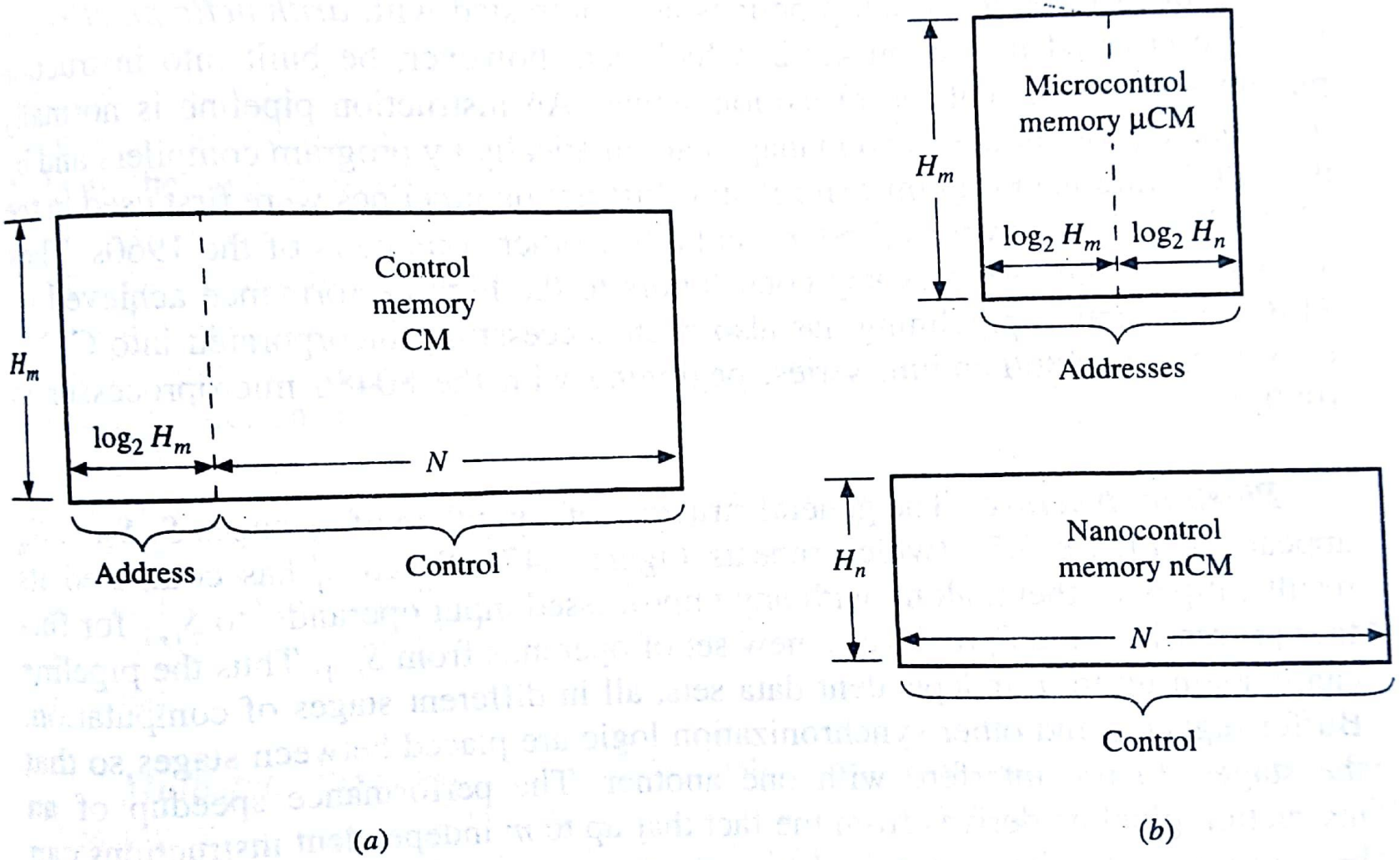


Figure 5.52
Control memory models: (*a*) one level and (*b*) two level.

NANOPROGRAMMING

- A one-level conventional CM is assumed to store H_m horizontal microinstructions each with a format consisting of N control bits and $\lceil \log_2 H_m \rceil$ next-address bits.
- The size of the memory is therefore,

$$S_1 = H_m(N + \lceil \log_2 H_m \rceil)$$

- In the two-level organization, the micro-control memory μCM again stores H_m microinstructions, but the N-bit control fields are transferred to nCM.
- In place of the latter, each microinstruction in μCM contains a $\lceil \log_2 H_n \rceil$ - bit address to specify any nanoinstruction location in nCM.
- It is assumed that little or no branching takes place among nanoinstructions, so no explicit address bits are included in the model of nCM.
- The size of the two-level control store is,

$$S_2 = H_m(\lceil \log_2 H_m \rceil + \lceil \log_2 H_n \rceil) + NH_n$$

NANOPROGRAMMING

- Suppose that all the control-bit patterns in nCM are different so that each represents a unique control state associated with the given instruction set..
- We can write $H_n=rH_m$, where r is the ratio of the number of unique control states (H_n) to the total number (H_m) of control states needed to implement all instructions.

$$S_2 = H_m(\lceil \log_2 H_m \rceil + \lceil \log_2 H_n \rceil) + NH_n$$

$$S_2 = H_m(\lceil \log_2 H_m \rceil + \lceil \log_2 rH_m \rceil) + NrH_m$$

$$S_2 = H_m(\lceil \log_2 H_m \rceil + \lceil \log_2 rH_m \rceil + Nr)$$

$$S_2 = H_m(\lceil \log_2 H_m \rceil + \lceil \log_2 H_m \rceil + \lceil \log_2 r \rceil + Nr)$$

$$S_2 = H_m(2\lceil \log_2 H_m \rceil + \lceil \log_2 r \rceil + Nr)$$

- If N=70, H_m=650, and r=0.4 so that H_n=260.
- We obtain S_1=52450 and S_2=30550.
- Consequently, the use of nanoprogramming saves a total of 52450-30550=21850 bits of control storage (42% of S_1).

PIPELINE CONTROL : INSTRUCTION PIPELINES

- An **instruction pipeline** is a multifunction, reconfigurable pipeline designed to speed up a computers performance by efficiently overlapping the processing of instructions (like fetch opcodes and operands, execute instructions and store the result in local registers or external memory).
- Invisible to programmers and managed automatically by program compilers and by the CPU's program-control unit.
- The general structure of a pipeline of m stages S_1, S_2, …, S_m is shown in figure 5.53.

PIPELINE CONTROL

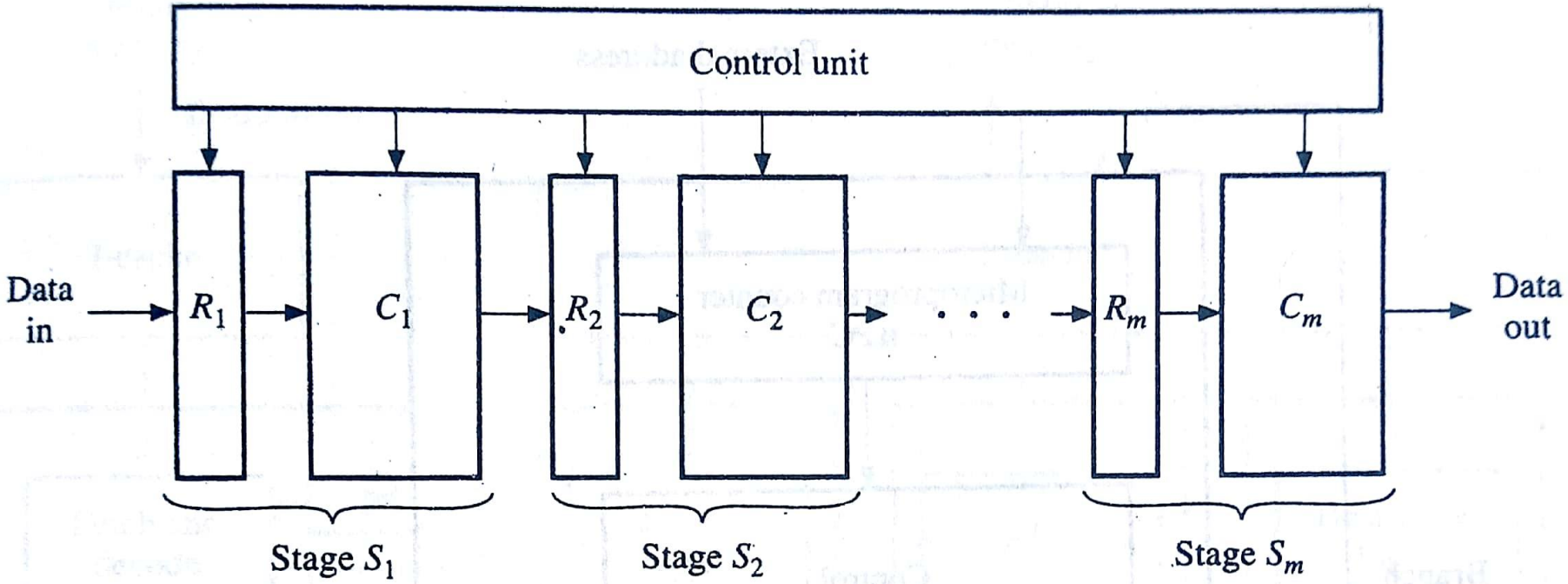


Figure 5.53
Structure of an *m*-stage pipeline.

PIPELINE CONTROL : INSTRUCTION PIPELINES

- When S_i has computed its results, it passes them, along with any unprocessed inputs operands to S_{i+1} for further processing, and S_i receives a new set of operands from S_{i-1}.
- Thus the pipeline can contain up to m independent data sets, all in different stages of computation.
- Buffer registers and other synchronization logic are placed between stages so that the stages do not interfere with one another.
- The performance speed up of an instruction pipeline derives from the fact that up to m independent instructions can be in progress simultaneously in the m stages.
- Figure 5.54 shows an implementation of a two-stage instruction pipeline that is common in microprogrammed CPUs.

PIPELINE CONTROL

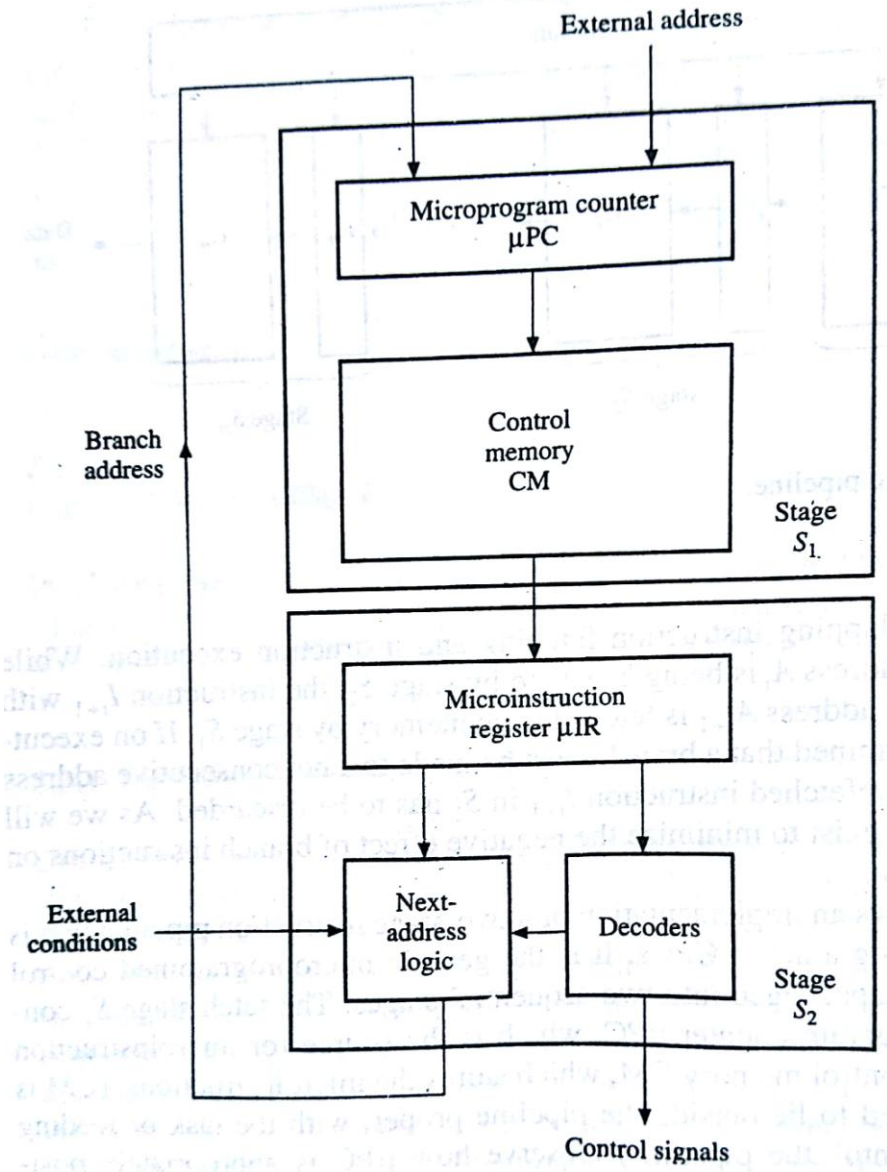


Figure 5.54
Two-stage pipelined microprogram control unit.

PIPELINE CONTROL : INSTRUCTION PIPELINES

- Generic microprogrammed control unit repackaged into two sequential stages.
- The **fetch stage S_1** consists of the microprogram counter μPC (similar to buffer register in pipeline structure), which is the source of microinstruction addresses and the control memory CM, which stores the microinstructions.
- The **execution stage S_2** contains the microinstruction register μIR (acts as a buffer register), the decoder that extracts control signals from the microinstructions in μIR, and the logic for choosing branch addresses.

PIPELINE CONTROL : INSTRUCTION PIPELINES

Multistage pipelines :

- An m-stage instruction pipeline can overlap the processing of up to m instructions.
- The value of m depends on the maximum number of stages into which instruction processing can be efficiently broken.
- This number in turn depends on the complexity of the instruction set, the organization of the external memory M and the way in which the CPU's datapath is implemented.
- Figure 5.55 shows the CPU organization that implements a four-stage instruction pipeline.
- We assume that the CPU is directly connected to a cache memory, which is split into instruction and data parts, called the I-cache and D-cache, respectively.
- The splitting of the cache permits both an instruction word and a memory data word to be accessed in the same clock cycle.

PIPELINE CONTROL

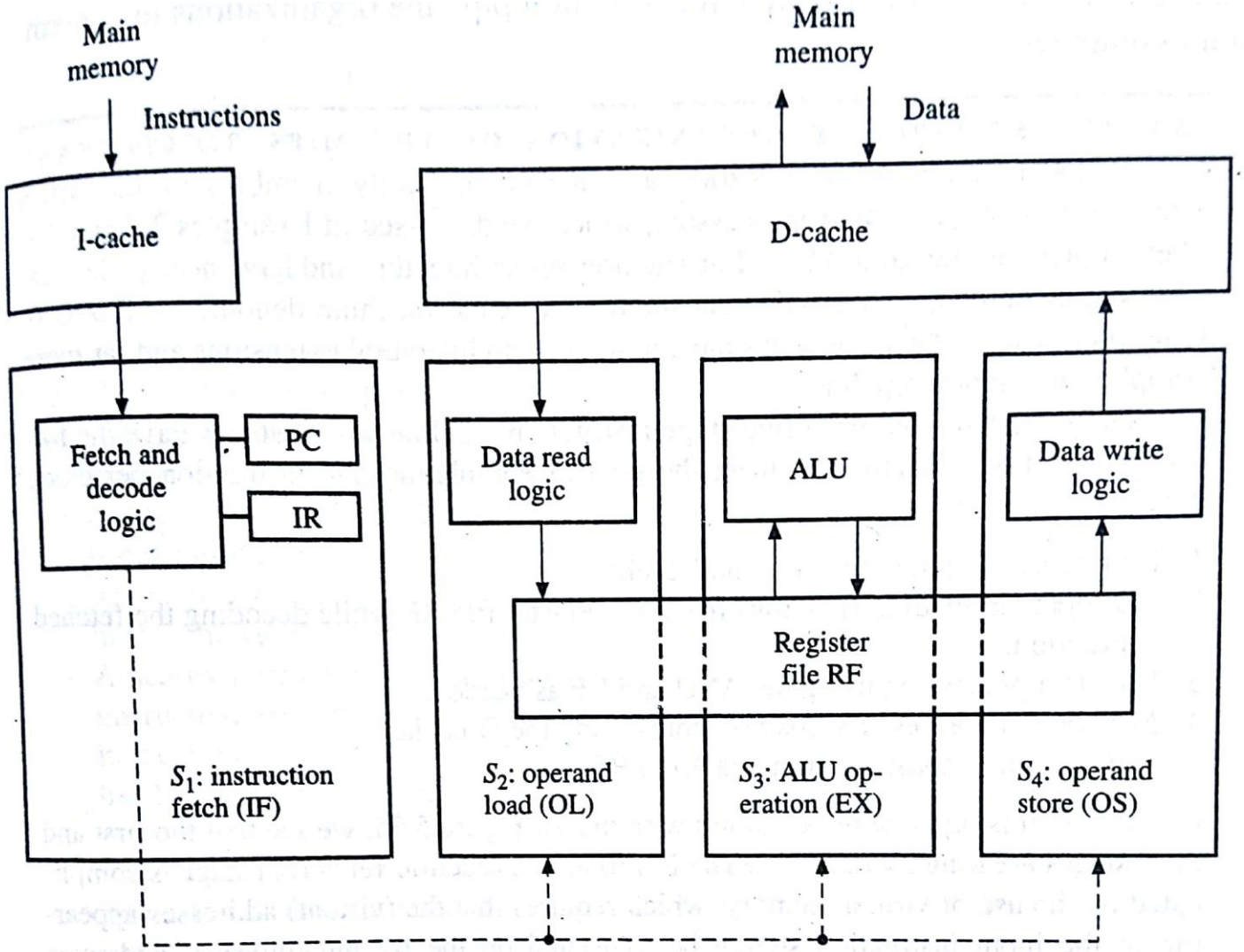


Figure 5.55
Organization of a CPU incorporating a four-stage instruction pipeline.

PIPELINE CONTROL : INSTRUCTION PIPELINES

- The four stages S_1:S_4 perform the following functions:

1. **IF** : instruction fetching and decoding using the I-cache.
2. **OL** : operand loading from the D-cache to RF.
3. **EX** : data processing using the ALU and RF.
4. **OS** : operand storing to the D-cache from RF.

- Stages S_2 and S_4 implement memory load and store operations.
- Stages S_2, S_3 and S_4 share the CPU's local registers in RF; these registers act as interstage buffer registers..
- The CPU's ALU in stage S_3, implements data-transfer and data processing operations of the register to register type.
- If each stage completes its operation in a single CPU clock cycle of period T_c, the pipeline and the CPU as a whole can be clocked at a frequency of $f=1/T_c$.
- At its maximum execution rate (no delays due to instruction branching, cache misses or other causes), a **CPI of 1** can be achieved.

PIPELINE CONTROL : INSTRUCTION PIPELINES

- Pipeline organization may vary to minimize the following drawbacks.
- a less expensive D-cache cannot perform loads and store simultaneously, in which case we can implement D-cache accesses in a single stage, thus merging S_2 and S_4 into a single load-store stage.
- Memory or register file accesses are complicated by addressing modes such as indexing, which require an ALU to calculate a memory address before the access operation proper can be initiated. In such case It may be desirable to add a stage, that is, a separate clock cycle, for operand address calculation.
- Instructions such as the more complicated arithmetic operations require multiple clock cycles for their execution; hence they require multiple cycles through the execution stage of a pipeline CPU.

PIPELINE CONTROL : INSTRUCTION PIPELINES

- The R2/3000 employs a five-stage instruction pipeline.

1. **IF** : instruction fetching using the I-cache.
2. **RD** : operand loading (reading) from the register file RF while decoding the fetched instruction.
3. **EX** : data processing using the ALU and RF as needed.
4. **MA** : operand accessing (load or store) using the D-cache.
5. **WB** : operand storing (writing back) to RF.

PIPELINE CONTROL

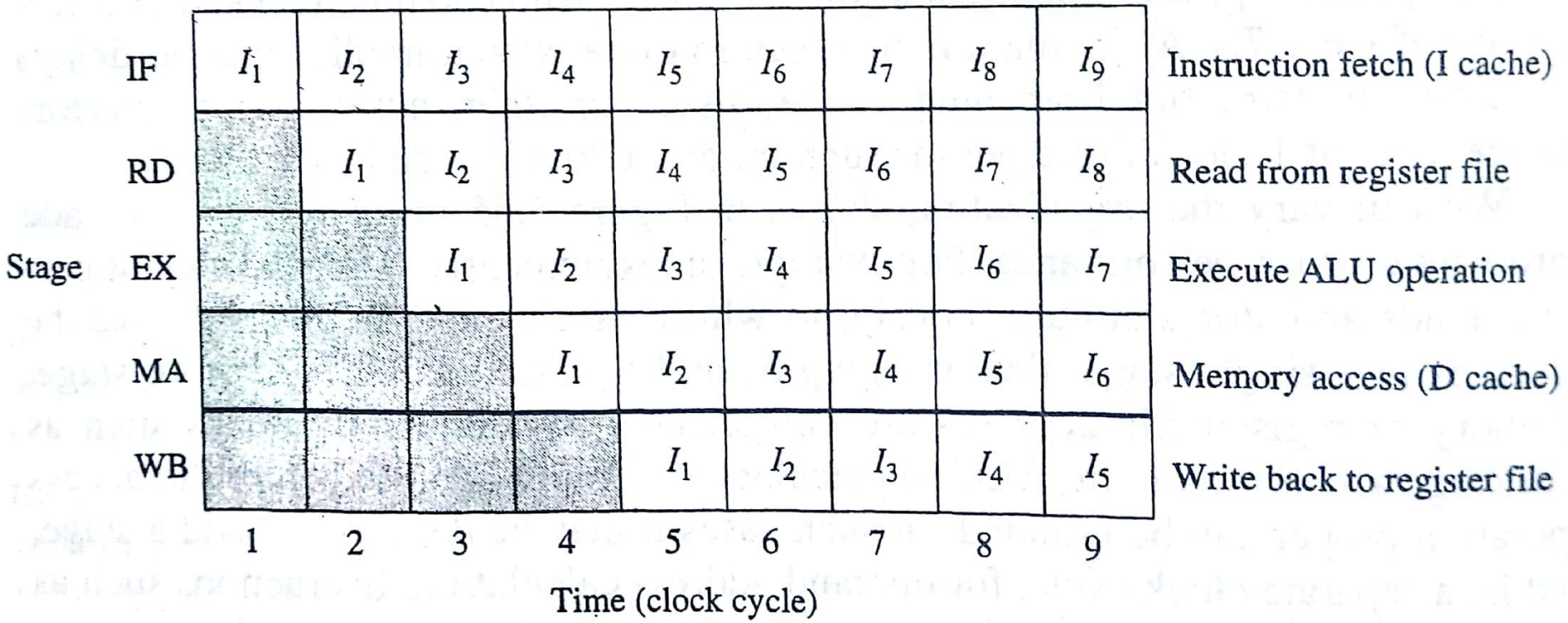


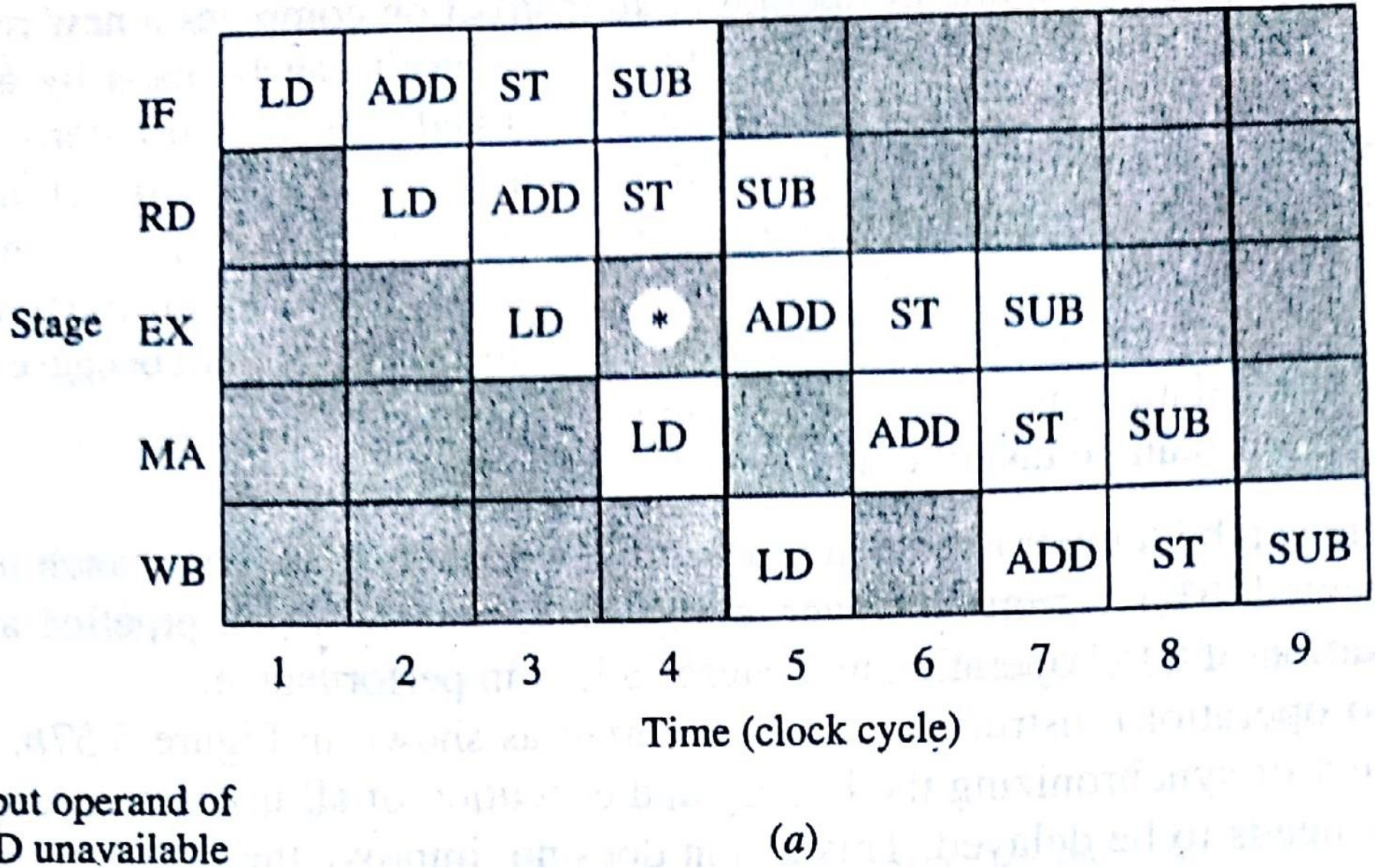
Figure 5.56

Maximum-rate instruction execution in the R2/3000 instruction pipeline.

PIPELINE CONTROL : INSTRUCTION PIPELINES

- Pipeline stalling can happen due to load and branch instructions.
- Eg : An instruction that loads a data word X into a register is immediately followed by an instruction that uses X in its EX stage.
- A one-cycle gap or delay slot occurs in the instruction stream because an LD (load) instructions data is not available until after its MA cycle.

PIPELINE CONTROL



*Input operand of ADD unavailable

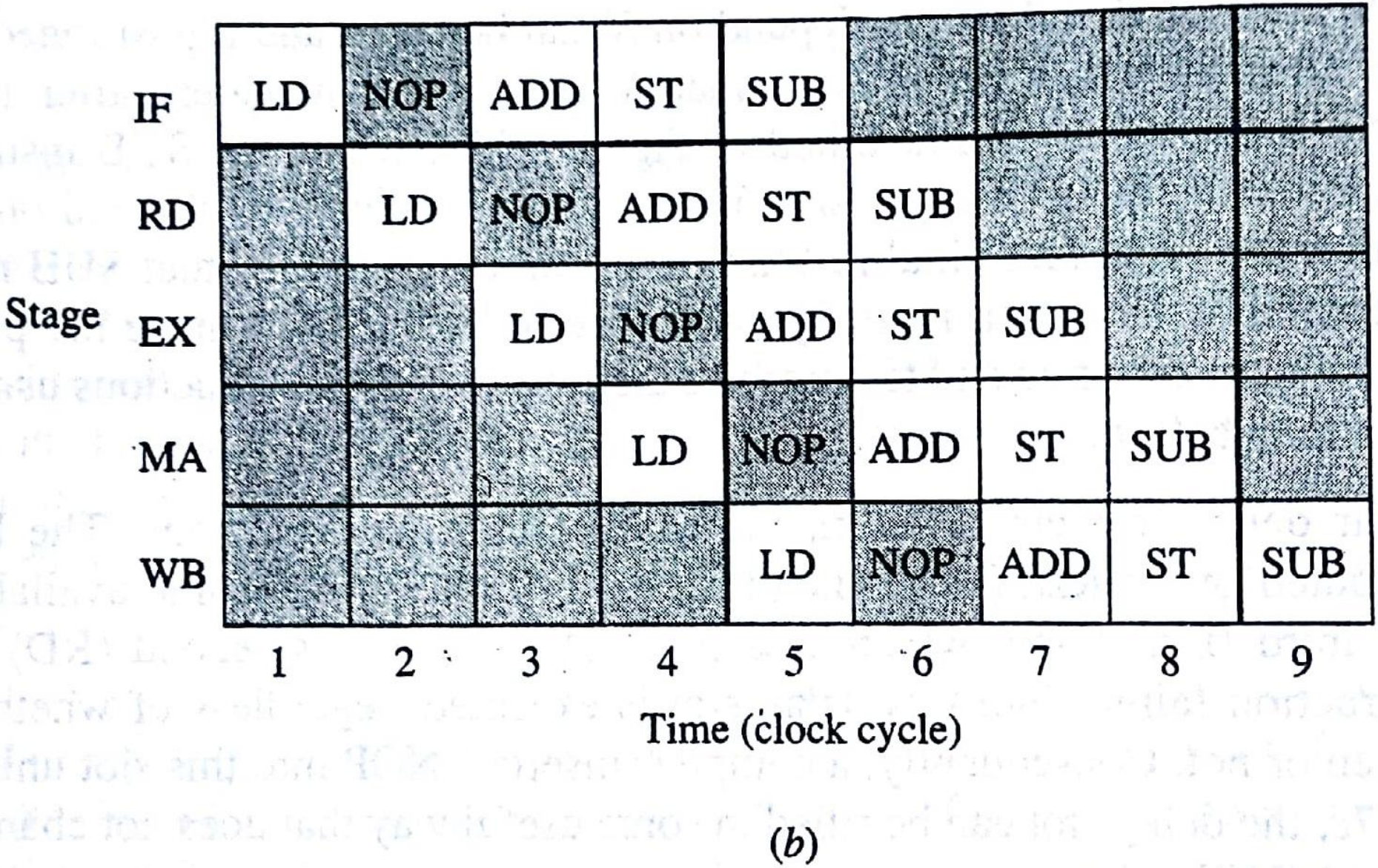
(a)

PIPELINE CONTROL : INSTRUCTION PIPELINES

- Several actions can be taken to deal with this situation:

1. The pipeline can be temporarily halted or stalled whenever a load or branch instruction is executed. This action, however, complicates control of the pipeline and the synchronization of CPU operation and causes a loss in performance.
2. A NOP (no operation) instruction can be inserted, which has the effect of synchronizing the issuing and execution of all instructions, none of which now needs to be delayed.

PIPELINE CONTROL



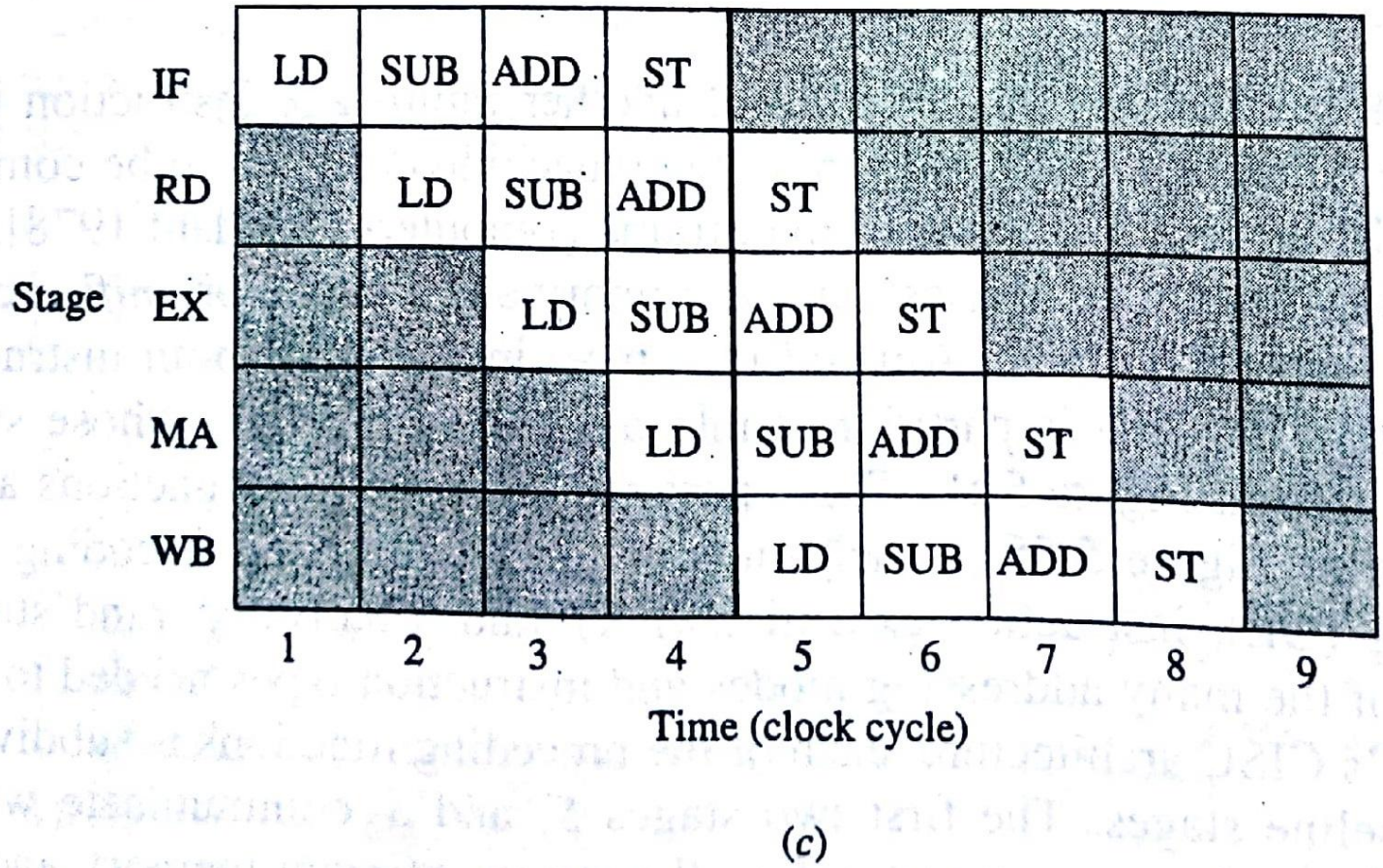
(*b*)

PIPELINE CONTROL : INSTRUCTION PIPELINES

- Several actions can be taken to deal with this situation:

3. A nearby instruction that does not depend on X can be taken and repositioned in the instruction stream – which requires a smart compiler – immediately after the LD instruction.

PIPELINE CONTROL



(c)

Figure 5.57
(*a*) R2/3000 pipeline delay slot caused by load instruction LD; (*b*) use of NOP instruction to fill the delay slot; (c) use of SUB instruction to eliminate the delay slot.

PIPELINE CONTROL : PIPELINE PERFORMANCE

- A pipeline's performance can be measured by its **throughput** in terms of millions of instructions executed per second or MIPS.
- Performance can also be measured using the number of clock cycles per instruction or CPI.
- These quantities are related by,

$$CPI = f/MIPS$$

Where f is the pipeline's clock frequency in MHz, and the values of CPI and MIPS are average figures that can be determined experimentally by processing suites of representative programs (benchmarks).

- The maximum value of CPI for a single pipeline is one.
- Superscalar machines reduce CPI below one by executing several instructions streams simultaneously using multiple pipelines.

PIPELINE CONTROL : PIPELINE PERFORMANCE

- A pipeline's performance can be measured by its **throughput** in terms of millions of instructions executed per second or MIPS.
- Performance can also be measured using the number of clock cycles per instruction or CPI.
- These quantities are related by,

$$CPI = f/MIPS$$

Where f is the pipeline's clock frequency in MHz, and the values of CPI and MIPS are average figures that can be determined experimentally by processing suites of representative programs (benchmarks).

- The maximum value of CPI for a single pipeline is one.
- Superscalar machines reduce CPI below one by executing several instructions streams simultaneously using multiple pipelines.

PIPELINE CONTROL : PIPELINE PERFORMANCE

- Figure 5.60 shows a space-time diagram for the four stage arithmetic pipeline, which is executing a complex vector summation instruction denoted I.
- An unshaded box in these figures marks a busy stage S_i, and the box entry denotes the particular instruction being processed by S_i.
- The ratio of unshaded (busy) area to the total (shaded and unshaded) area of a space-time diagram for an m-stage pipeline is defined as the **efficiency** or **utilization E (m)** of the pipeline.

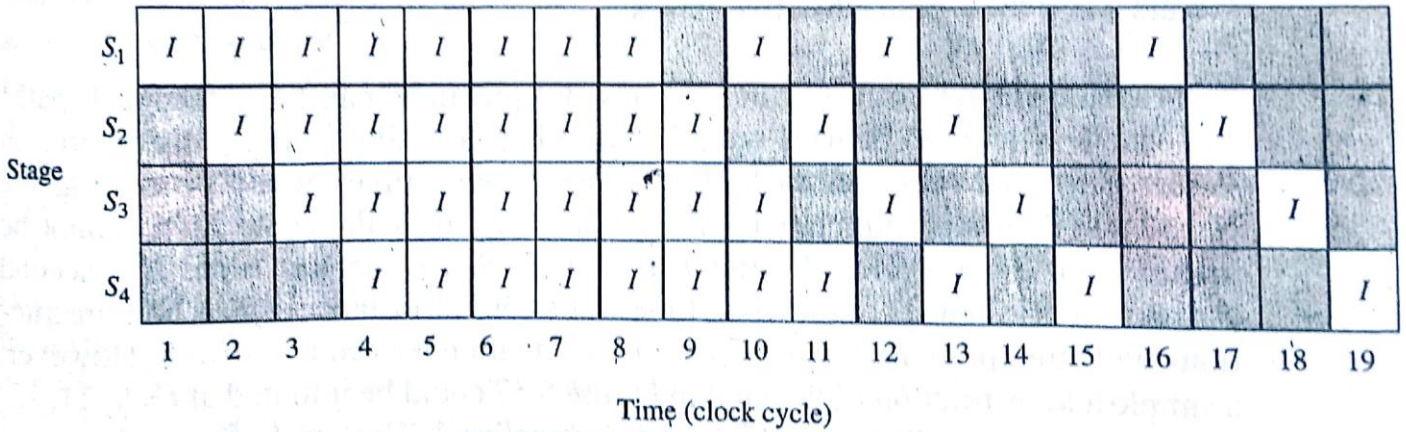


Figure 5.60
Space-time diagram for a four-stage pipeline.

PIPELINE CONTROL : PIPELINE PERFORMANCE

- In other words, E(m) is the fraction of time the pipeline is busy.
- In this case, the efficiency is E(4) =44/76 = 0.58.
- Another measure of pipeline performance is the **speedup S(m)** defined by,

$$S(m) = T(1)/T(m)$$

Where T(m) is the execution time for some target workload on an m-stage pipeline and T(1) is the execution time for the same workload on a similar, nonpipelined processor.

- The pipeline's efficiency and speedup are related as,

$$S(m) = m \times E(m)$$

- The speed up S(4) = 4 x 0.58 = 2.32 and cannot exceed 4.
- Easy way to improve a pipeline's performance is to increase the number of stages m.
- Each new stage S_i introduces some new hardware cost and delay due to its buffer register R_i and associated control logic.

PIPELINE CONTROL : PIPELINE PERFORMANCE

- The pipeline's **performance/cost ratio PCR** defined as,

$$PCR = f/K$$

Where f is the pipeline's clock frequency and K is the hardware cost.

- Suppose the pipeline P has m stages and implements a particular set of operations (instructions) SI.
- Let **a** be the delay (latency) of an efficient, nonpipelined processor that also implements SI.
- It is reasonable to assume that each stage S_i of P has **delay a/m** – that is, m times less than the corresponding nonpipelined processor – **plus some extra delay b** due to S_i's buffer register R_i.
- Hence if $T_c = 1/f$ is P's clock period, we can write,

$$T_c = a/m + b$$

PIPELINE CONTROL : PIPELINE PERFORMANCE

- The pipeline's hardware cost can be estimated by,

$$K = cm + d$$

Where c is the buffer-register cost per stage and d is the cost of the pipeline's (combinational) data-processing logic.

- Thus we have,

$$PCR^{-1} = T_c K = (a/m + b)(cm + d)$$

$$PCR = m/[bcm^2 + (ac + bd)m + ad]$$

- To maximize PCR with respect to the number of stages m, we differentiate with respect to m and equate the result to zero.

$$\frac{d}{dx}\left(\frac{u}{v}\right) = \frac{1}{v}\frac{du}{dx} - \frac{u}{v^2}\frac{dv}{dx}$$

$$\frac{d}{dm}(PCR) = \frac{1}{v} - \frac{m(2bcm + ac + bd)}{v^2}$$

Where u = m and v = bcm^2 + (ac +bd)m + ad.

PIPELINE CONTROL : PIPELINE PERFORMANCE

- On equating the above equation to zero, we get v = m(2acm + ad +bc).
- Substituting for v and solving for m yields the value m_{opt} of m that maximizes PCR namely,

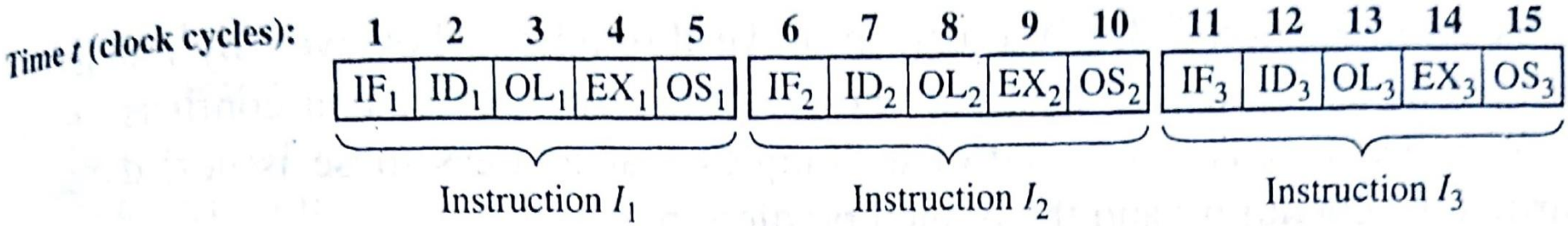
$$m_{opt} = \sqrt{\frac{ad}{bc}}$$

- The optimum number of stages is the integer closest to m_{opt}.

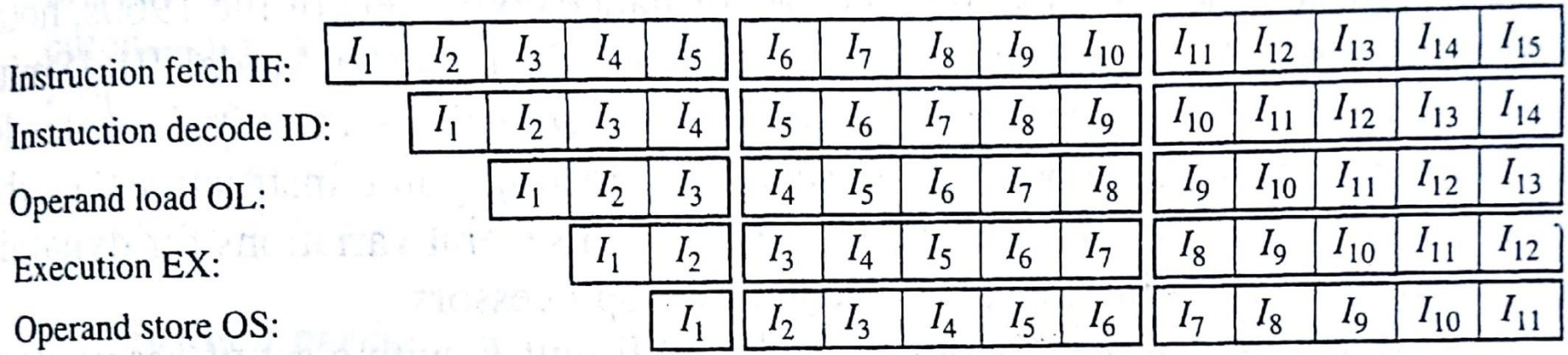
SUPERSCALAR PROCESSING

- Performance levels greater than one instruction per cycle, that is, a CPI figure less than one – by fetching, decoding and executing several instructions concurrently – **Superscalar.**
- A superscalar computer has a single CPU.
- A superscalar CPU has multiple execution units (E-units), each of which is usually pipelined, so that they constitute a set of independent instruction pipelines.
- The CPU's program control unit PCU is designed to fetch and decode several instructions concurrently.
- It can issue or dispatch up to k instructions simultaneously to the various E-units where k, the **instruction-issue degree**, can be six or more using current technology.

SUPERSCALAR PROCESSING

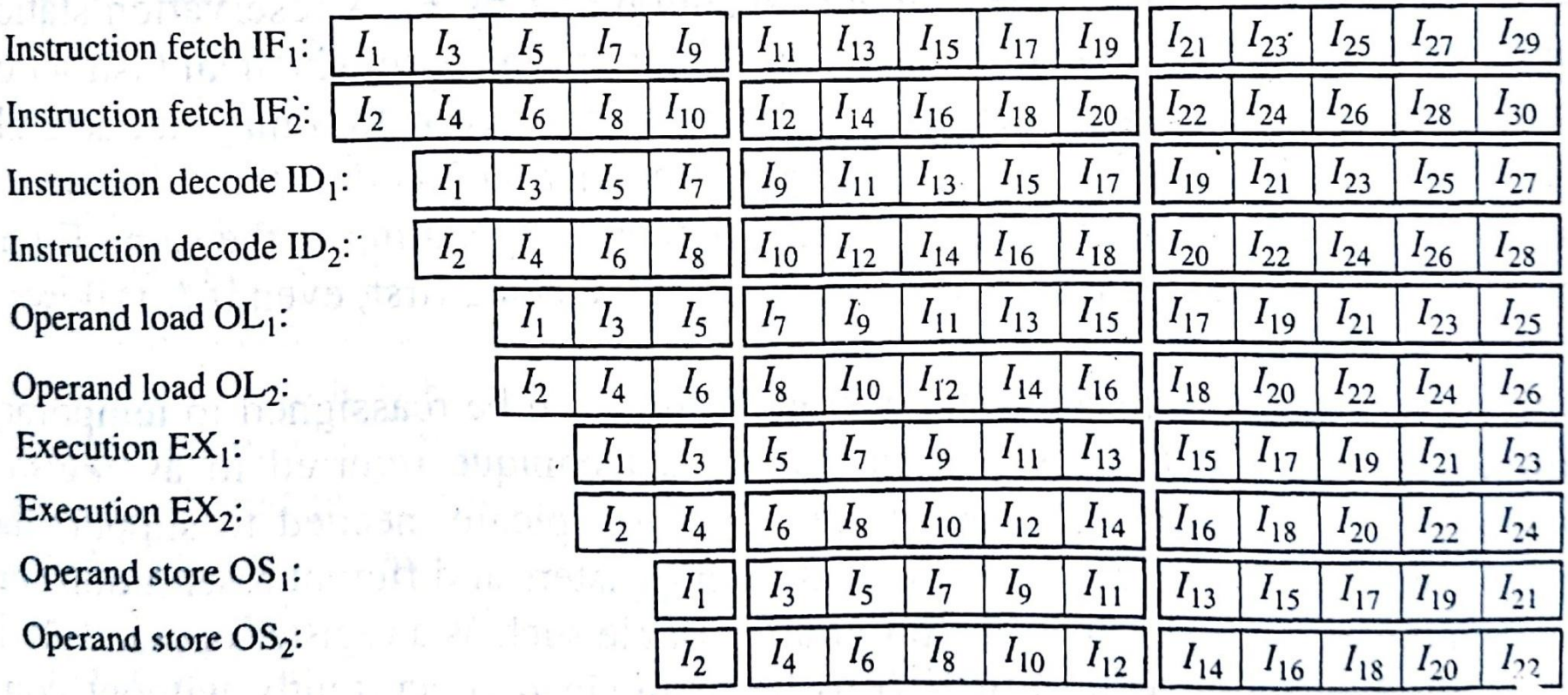


(a)



(b)

SUPERSCALAR PROCESSING



(c)

Figure 5.66
Maximum parallelism in (*a*) a sequential CPU; (*b*) a CPU with a five-stage instruction pipeline; (*c*) a superscalar CPU with two five-stage instruction pipelines.

SUPERSCALAR PROCESSING

- Assuming that each instruction requires a total of five cycles, we see that a single five-stage instruction pipeline (k=1) offers a speedup of 5, while the two-issue (k=2) superscalar design has a potential speedup of 10.
- Observe that at the start of cycle 15, the sequential CPU has completed only two instructions, whereas the pipelined and superscalar machines have completed 10 and 20 instructions, respectively; moreover the superscalar CPU has already started processing instructions I_{21} through I_{30}.
- Keeping k pipelines busy requires the CPU to fetch at least k instructions per clock cycle; hence superscalar designs place heavy demands on the instruction-fetch logic.
- Instruction fetching is supported by an **instruction buffer** or **queue** or **cache**, a short storage unit within the CPU that serves as a staging area for prefetched and (partially) decoded instructions.
- The PCU dispatches the instructions from its instruction buffer to the various E-units for execution.

SUPERSCALAR PROCESSING

➢Superscalar computer needs to consider the following factor into account:

1. Instruction type : Eg – a floating point add instruction has to be issued to a floating-point E-unit and not to an integer E-unit.
2. E-unit availability : An instruction can be issued to a pipelined E-unit only if no collision will result, as determined by the pipeline's reservation table.
3. Data dependencies : To avoid conflicting use of registers.
4. Control dependencies : To maintain high performance levels, techniques are needed to reduce the impact of branch instructions on pipeline efficiency.
5. Program order : Instructions must eventually produce results in the order specified by the program being executed. The results may, however, be computed out of order internally to improve the CPU's performance.

SUPERSCALAR PROCESSING

- Delaying a problematic instruction before it enters an instruction pipeline can prevent conflicts.
- Such static scheduling of instructions can occur during program compilation.
- We can improve throughput, however, by issuing all instructions as rapidly as possible and resolving any subsequent conflicts on the fly – **Dynamic instruction scheduling** and **Branch prediction**.

SUPERSCALAR PROCESSING : DYNAMIC INSTRUCTION SCHEDULING

- **Tomasulo's algorithm** – schedule floating-point instructions in the Model 91.
- It provides each shared E-unit F_i with a set of **reservation stations** whose purpose is to receive instructions that use F_i, keep track of these instructions operands, and when all the operand values needed by a waiting instruction I_j becomes available, initiate execution of I_j by F_i .
- While one instruction I_j is delayed at a reservation station waiting for operands, another instruction I_k waiting at the same E-unit whose operands become available sooner can be executed first, even if I_k follows I_j in the program order.
- To handle data dependencies, operand values can be reassigned to temporary (virtual) registers at the reservation station, a technique referred to as register renaming.
- A temporary register is marked by a "tag" to indicate whether the operand value it contains is valid (to prevent an instruction from reading an obsolete value) and whether there are uncompleted instructions that need that particular value.

SUPERSCALAR PROCESSING : DYNAMIC INSTRUCTION SCHEDULING

- Consider the following three-instruction sequence:

R[1] = ALPHA Instruction I_1 (load)

R[2] = R[1] + R[2] Instruction I_2 (add)

R[3] = R[4] + R[5] Instruction I_3 (add)

- A superscalar CPU can fetch and decode all three instructions simultaneously or nearly simultaneously.
- If the current value of the operand ALPHA in the first instruction I_1 is in main memory, but not in D-cache (a cache miss), the execution of I_1 is delayed by several cycles.
- In that case, I_1 is sent to a reservation station in the memory control logic – which is treated as an E-unit for scheduling purposes – and I_1's R[1] operand is assigned to a temporary register there, say TR[3].
- Execution of I_1 then stalls until ALPHA arrives and TR[3] is tagged as unavailable.

SUPERSCALAR PROCESSING : DYNAMIC INSTRUCTION SCHEDULING

- The second instruction I_2 is sent to an add unit where it is delayed by the fact that its R[1] operand, which the PCU points to as being assigned to TR[3], is unavailable; thus I_2 is placed in a reservation station at the adder.
- In the meantime, if all the operand values needed by the third instruction I_3 are available, I_3 can be executed out of order by the add unit.
- When ALPHA eventually arrives in the CPU, I_1 is executed by loading ALPHA into TR[3], whose tag is then changed to indicate that a valid result is now available.
- At that point I_2 can also be executed in the next available cycle of the add unit.

SUPERSCALAR PROCESSING : BRANCH PREDICTION

➢A two-way conditional branch instruction of the form,

if C then I_1 else I_2

Can cause control-dependency delays in an instruction pipeline because the branch's target address, which is the address of either I_1 or I_2, is not known until the condition C has been computed and checked.

➢ Powerful technique is to predict the value of C which implies branching to I_j and the proceed to execute the instructions I_j, I_{j+1}, I_{j+2},… along the expected path before C's value is known.

➢If the prediction is correct, then the performance gain has been made;

➢ if the prediction is wrong, then any instructions executed along the mispredicted path are cancelled.

➢ Because of its tentative nature, the execution of instructions before the correct path has been identified is termed **speculative**.

SUPERSCALAR PROCESSING : BRANCH PREDICTION

- 1-bit dynamic branch prediction is implemented in the Digital Alpha 21064 superscalar microprocessor, where it is reported to produce a branch prediction accuracy close to 80 percent.
- To assign a control bit **p** to a branch instruction I, when it is first executed; the CPU then uses the value of p to predict I's branching behavior in the future.
- The prediction rule of this method is that I will branch to the same instruction as it did the last time it was executed.
- Thus when iterating through a loop controlled by I, once the loop execution path is entered, p predicts the same loop path will be followed each time I is encountered.
- The two state of p have the following interpretation : **p=1** predicts the next instruction will be I_1 – that is, C wil be 1; p=0 predicts that next instruction will be I_2 – that is, C will be 0.

SUPERSCALAR PROCESSING : BRANCH PREDICTION

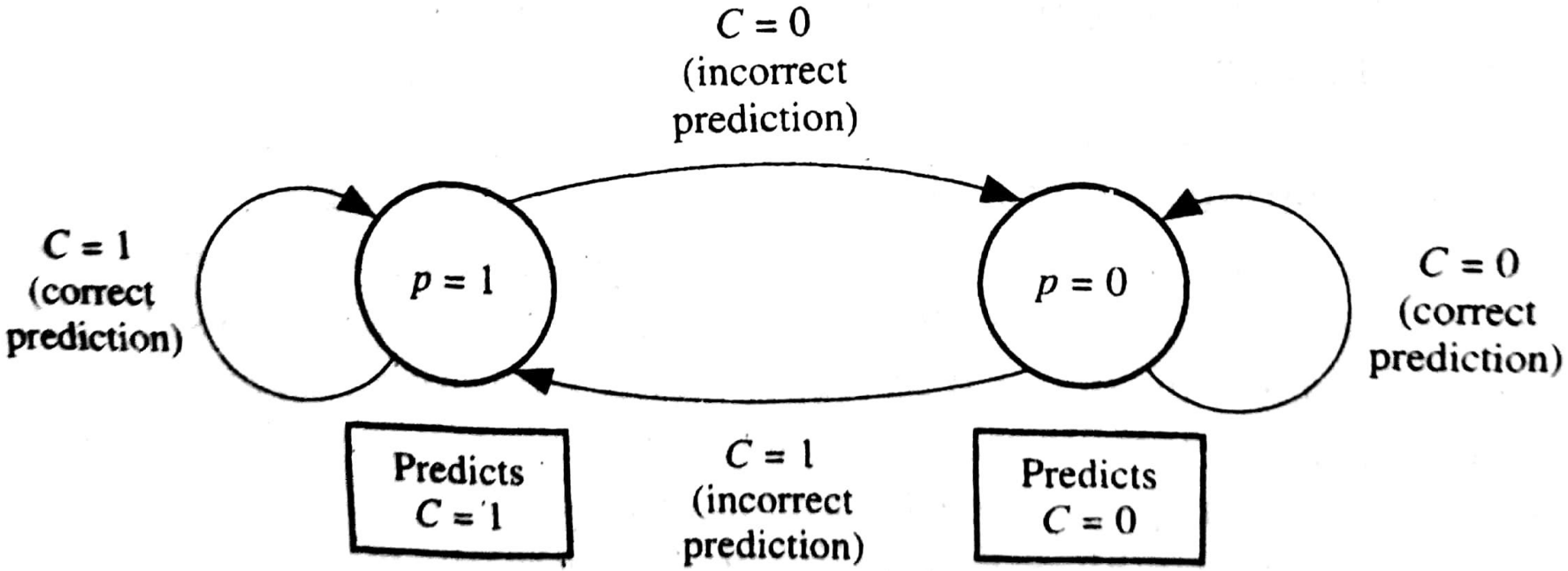


Figure 5.67
State behavior of 1-bit dynamic branch prediction method.

SUPERSCALAR PROCESSING : BRANCH PREDICTION

- It is also convenient to store the branching statistics in a table – the **branch history table** – along with the address of I and that of the instruction to which I currently branches.
- For rapid access, we can place the branch history table in a cache like memory in the CPU called a **branch target buffer (BTB).**
- The BTB is used as follows: Instruction requests are sent simultaneously to the I-cache and the BTB.
- If a match is found in the BTB, the accompanying, predicted branch target address is read out.
- Execution proceeds along the instruction path defined by the branch target address, with all results considered speculative until the outcome of the branch condition test becomes available.

SUPERSCALAR PROCESSING : BRANCH PREDICTION

➢When execution of the branch instruction is completed, its target address is updated in the BTB, which permits mispredicted targets to be replaced, the branch instruction prediction statistics are also updated.

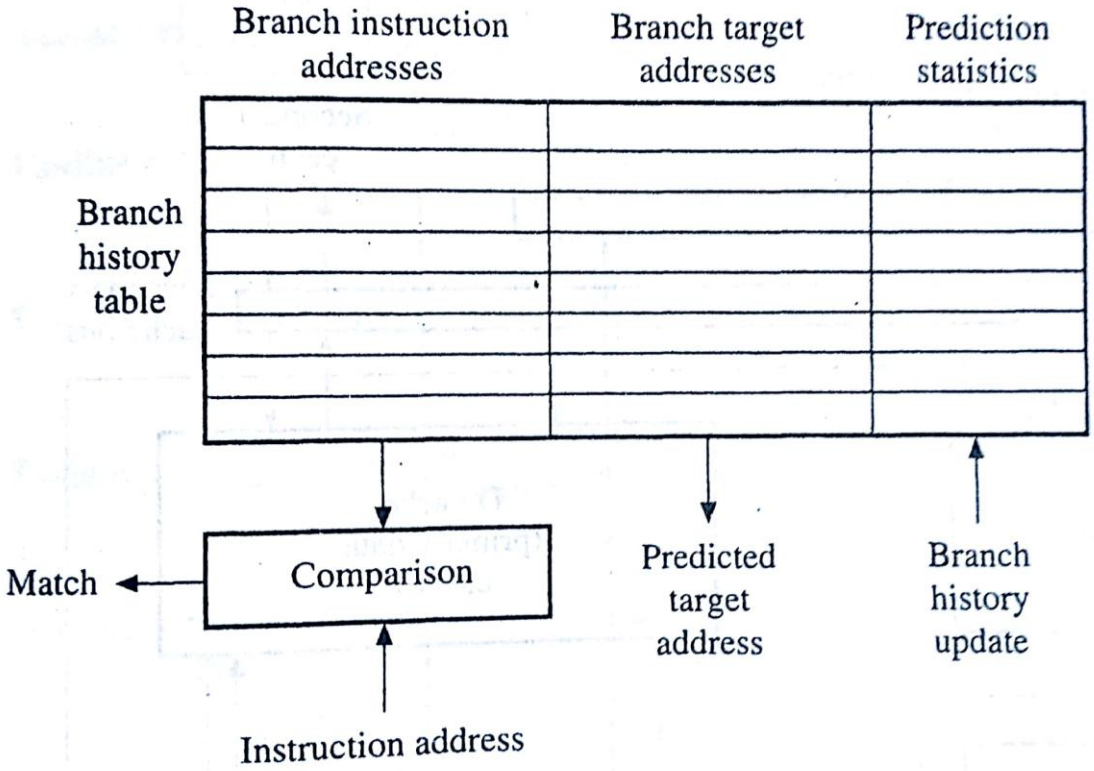


Figure 5.68
Organization of a branch target buffer (BTB).

SUMMARY

- Design of control unit : Hardwired Control and Microprogrammed Control
- Design Examples : GCD processor, Multiplier control unit and CPU control unit
- Nanoprogramming
- Pipeline control
- Pipeline performance
- Superscalar processing