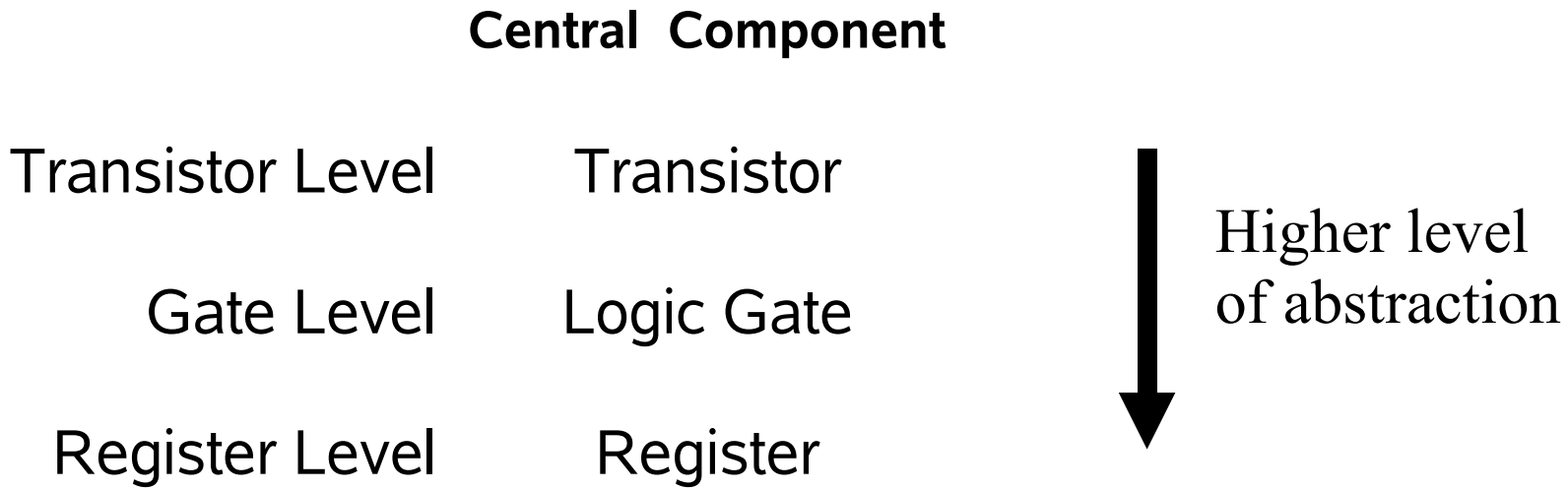
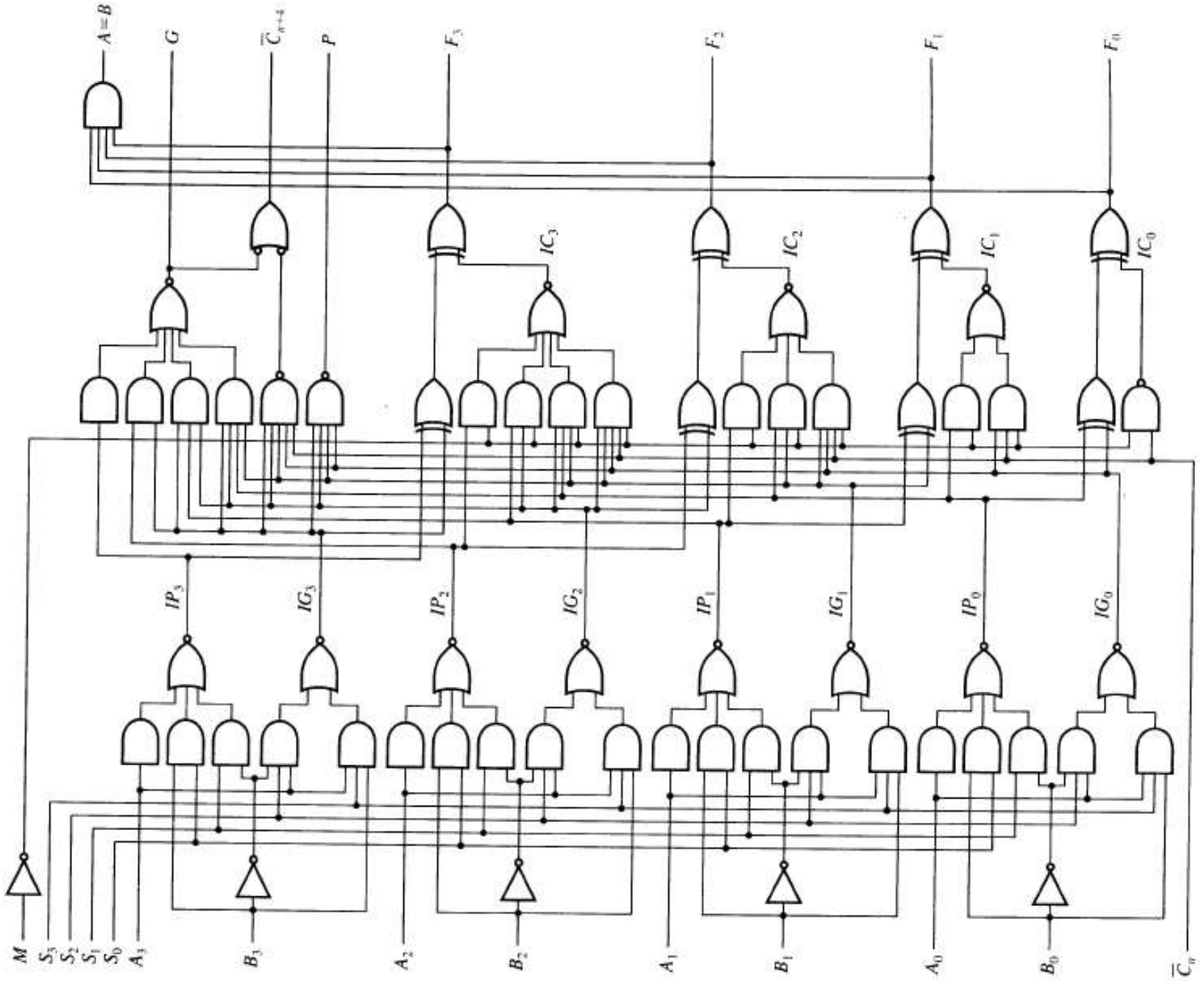
Register-Level Design

A digital system can be treated at different level of abstraction or complexity. So far, we have seen it at the gate level and the transistor level.

At a higher level than the gate level, we have the register level. The central component at this level is an n-bit storage device, or *register*.



Logic Circuit for the 74X181 ALU



From the logic circuit for the 74X181, we can write

$$F_0 = (M + C_n) \varpi \overline{(a_0 + B_0 S_0 + \bar{B}_0 S_1)} \varpi \overline{(A_0 \bar{B}_0 S_2 + A_0 B_0 S_3)}$$

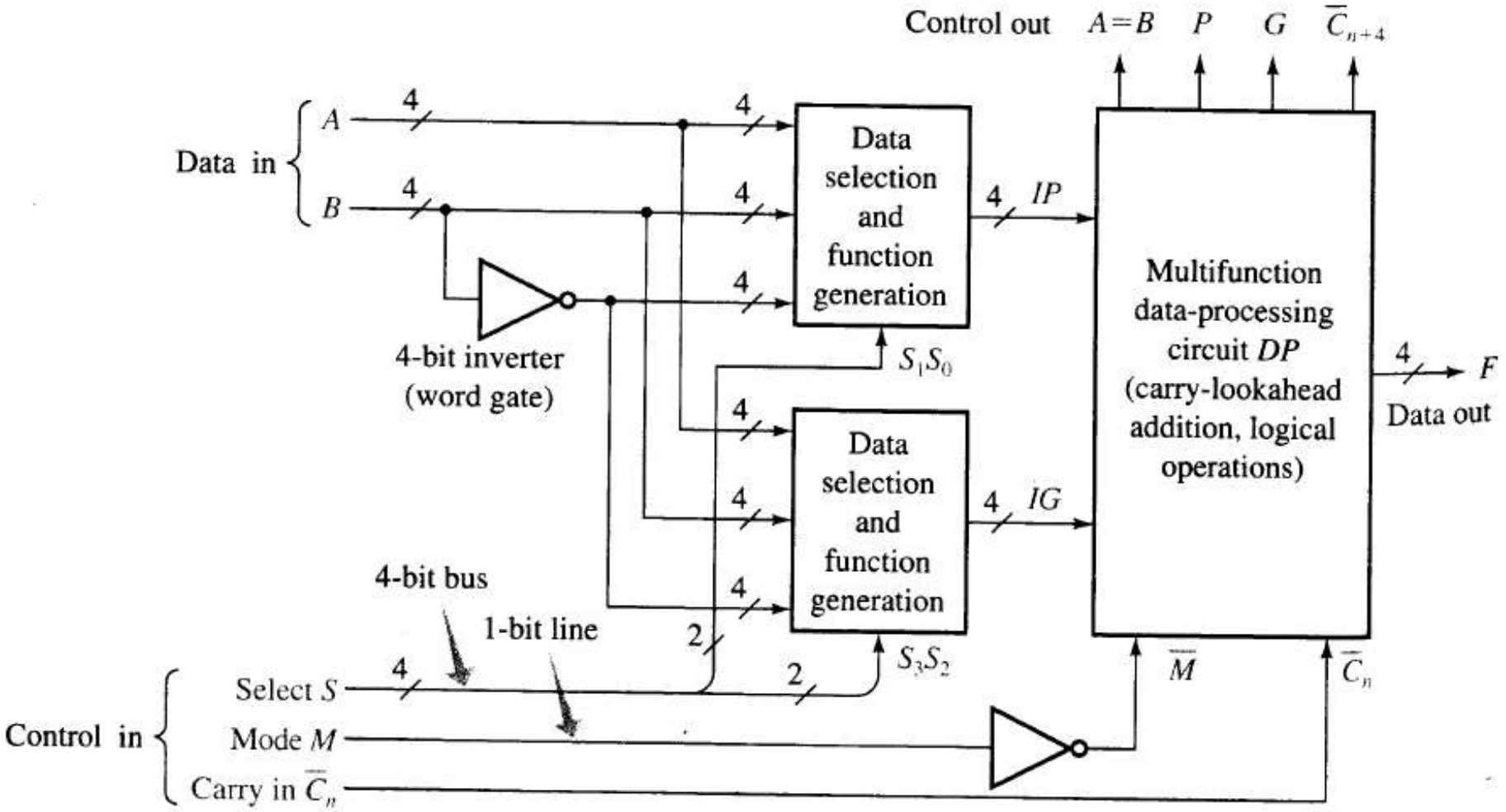
which gives little information on what functions the circuit perform.

Using a truth table to describe the functions of the circuit is not a good idea either as the circuit has 14 inputs, which means the truth table will have $2^{14} = 16384$ rows!

The 74X181 is actually a 4-bit ALU which performs 32 different arithmetic and logical operations on two 4-bit inputs $A = A_3 A_2 A_1 A_0$ and $B = B_3 B_2 B_1 B_{0,}$ producing the result $F = F_3 F_2 F_1 F_{0.}$

The operation to be performed is determined by the control signals M and $S = S_3 S_2 S_1 S_{0.}$ In particular, the mode control signal M is set to 1 for logical operations, and 0 for arithmetic operations. The select control signal S is use to select the required operation.

Register-Level Model of the 74X181 ALU



74X181 Arithmetic and Logical operations

SELECTION				ACTIVE-HIGH DATA		
				M = H	M = L; ARITHMETIC OPERATIONS	
S3	S2	S1	S0	LOGIC FUNCTIONS	$\overline{C_n}$ = H (no carry)	$\overline{C_n}$ = L (with carry)
L	L	L	L	$F = \overline{A}$	F = A	F = A PLUS 1
L	L	L	H	$F = \overline{A + B}$	F = A + B	F = (A + B) PLUS 1
L	L	H	L	$F = \overline{A}B$	$F = A + \overline{B}$	$F = (A + \overline{B})$ PLUS 1
L	L	H	H	F = 0	F = MINUS 1 (2's COMPL)	F = ZERO
L	H	L	L	$F = \overline{AB}$	$F = A$ PLUS $A\overline{B}$	$F = A$ PLUS $A\overline{B}$ PLUS 1
L	H	L	H	$F = \overline{B}$	$F = (A + B)$ PLUS $A\overline{B}$	$F = (A + B)$ PLUS $A\overline{B}$ PLUS 1
L	H	H	L	$F = A \oplus B$	F = A MINUS B MINUS 1	F = A MINUS B
L	H	H	H	$F = A\overline{B}$	$F = A\overline{B}$ MINUS 1	$F = A\overline{B}$
H	L	L	L	$F = \overline{A} + B$	F = A PLUS AB	F = A PLUS AB PLUS 1
H	L	L	H	$F = \overline{A \oplus B}$	F = A PLUS B	F = A PLUS B PLUS 1
H	L	H	L	F = B	$F = (A + \overline{B})$ PLUS AB	$F = (A + \overline{B})$ PLUS AB PLUS 1
H	L	H	H	F = AB	F = AB MINUS 1	F = AB
H	H	L	L	F = 1	F = A PLUS A†	F = A PLUS A PLUS 1
H	H	L	H	$F = A + \overline{B}$	F = (A + B) PLUS A	F = (A + B) PLUS A PLUS 1
H	H	H	L	F = A + B	$F = (A + \overline{B})$ PLUS A	$F = (A + \overline{B})$ PLUS A PLUS 1
H	H	H	H	F = A	F = A MINUS 1	F = A

† Each bit is shifted to the next more significant position.

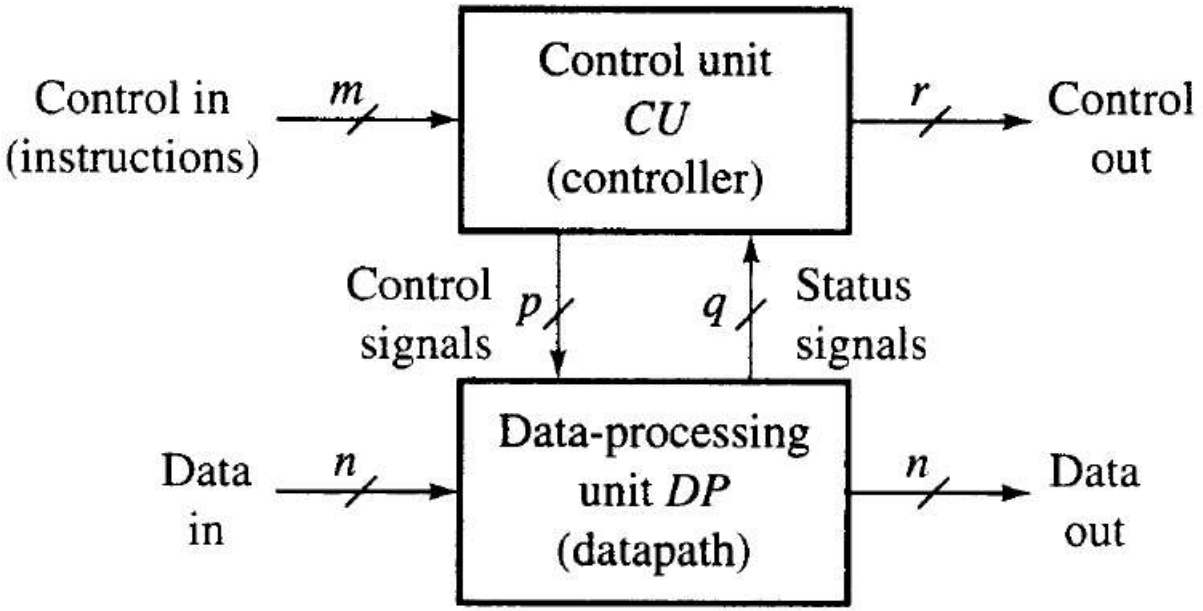
Register-level design is geared toward the processing of words as the basic units of information or signals. In contrast, in gate-level design individual bits are treated as basic signals.

The word length is register-level design varies, but it is always a multiple of 4.

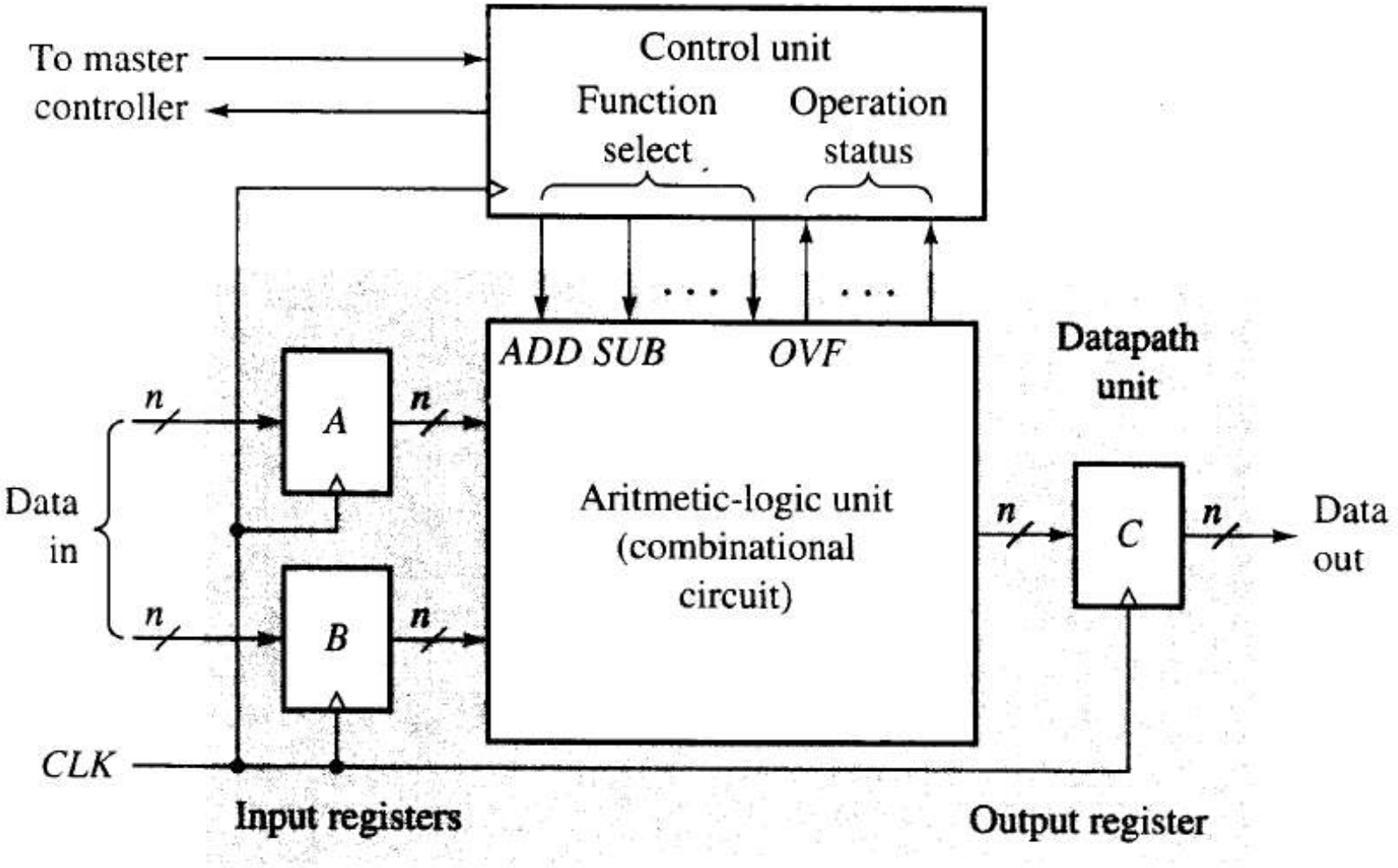
Another important aspect of register-level design is the data-flow representation which views the function of a digital circuit as processing data words that flow between computational and storage devices. The computational devices are typically multifunction combinatorial circuits (e.g. the ALU), and the storage devices are registers.

To manage these devices and control the data flow, control signals and circuits that generate these signals are required.

Complex digital circuits can be divided into a data-processing (DP) part and a control part (CU) at the register level.



Example of a small processor



The behavior of a register-level circuit can be described using behavioral style VHDL code.

For example, the operations of ALU of the small processor can be described using the following VHDL code:

```
if ADD = '1' then
   C <= A + B;
elsif SUB = '1' then
   C <= A - B;
end if;

if overflow_occurs then
   OVF <= 1;
end if;
```

Basic components of register-level design:

Basic register-level component types: (a) combinational; (b) sequential.

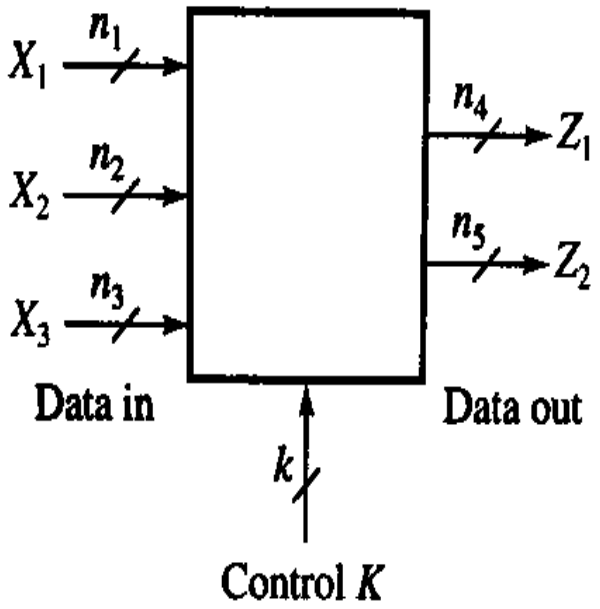
Component type	*Functions*
Word gate	Logical (Boolean) operations: AND, OR, NOT, NAND, EXCLUSIVE-OR, etc.
Multiplexer	Data selection and routing; Boolean function generation
Decoder/encoder	Data format conversion
Adder	Binary addition and subtraction
Comparator	Data comparison
ALU	Arithmetic and logical operations

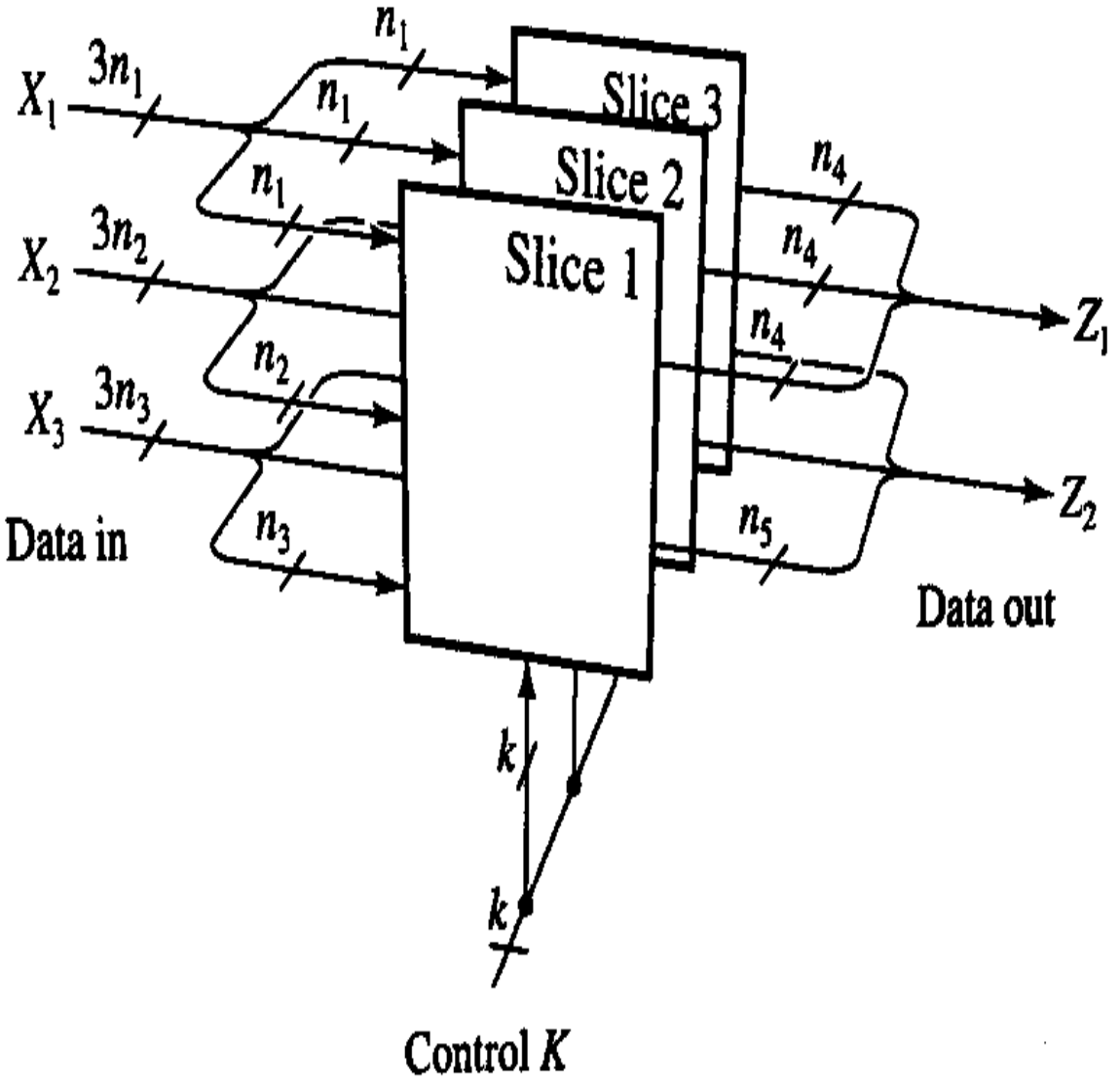
(a)

Component type	*Functions*
(Parallel) register	Data storage
Shift register	Data storage and format conversion
Counter	Control state generation; arithmetic operations; timing

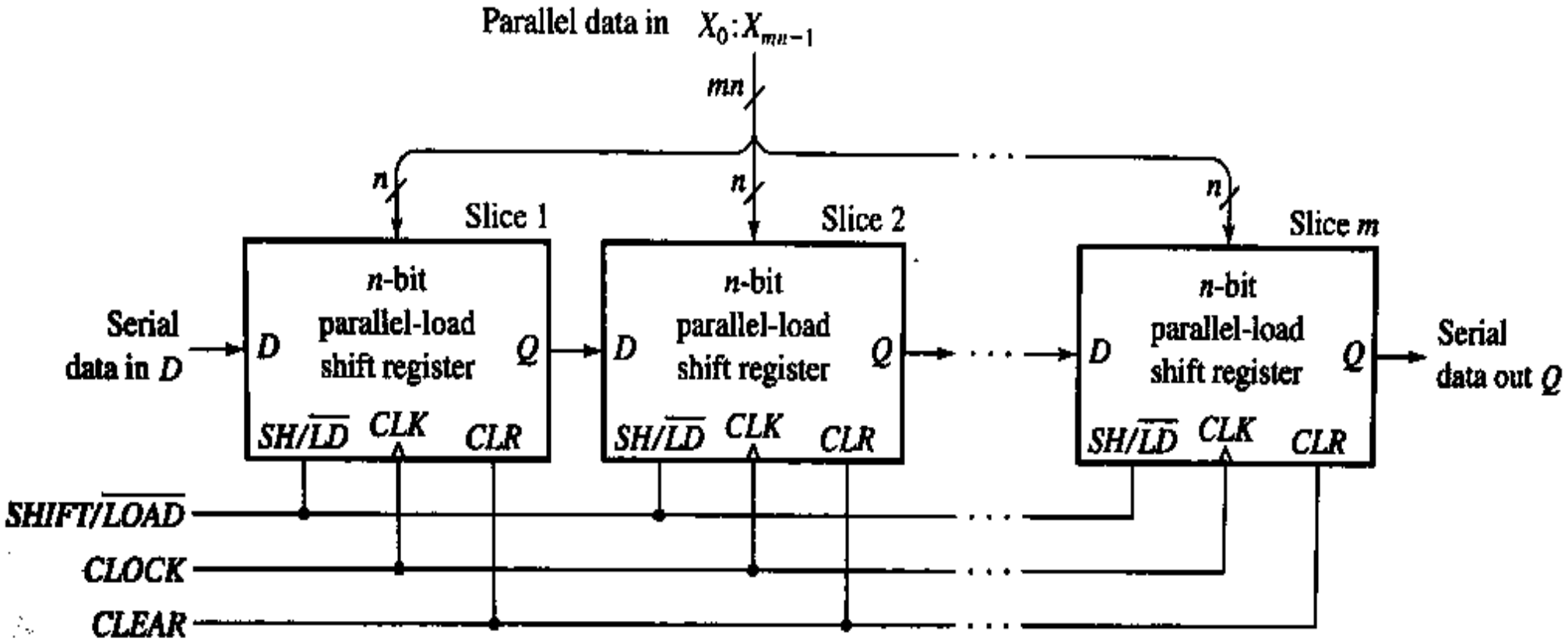
(b)

Component Expansion by Bit Slicing



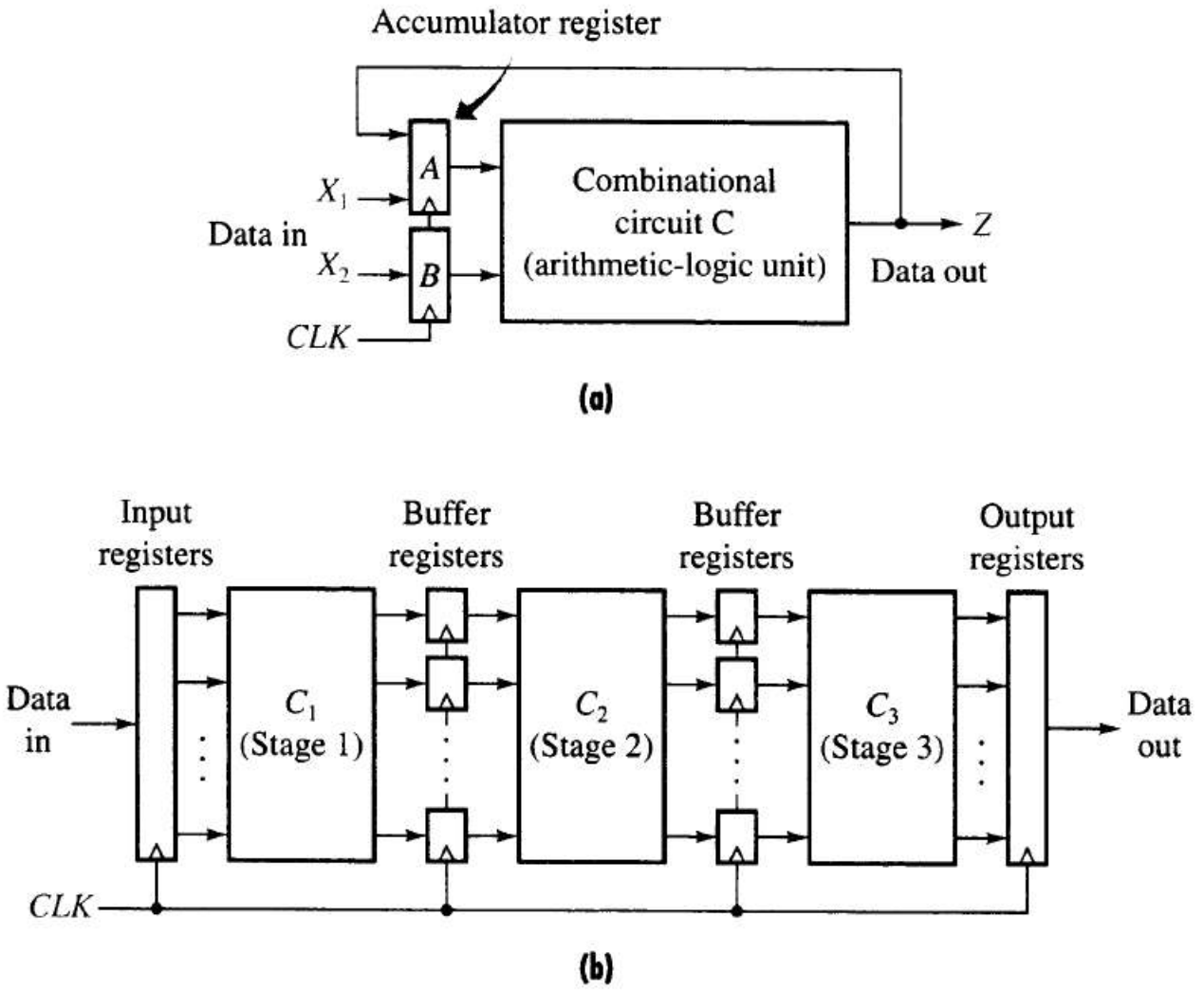


Shift Register Expansion using Bit Slicing

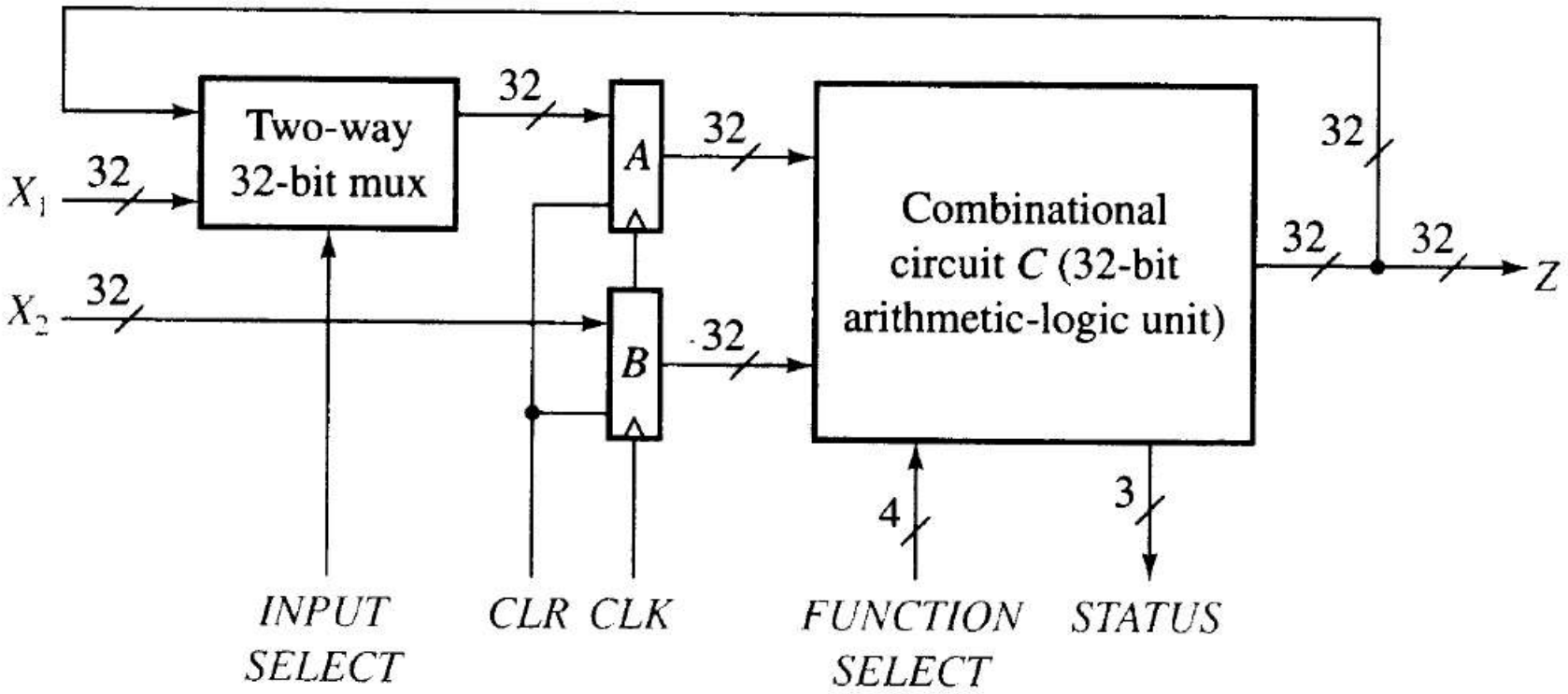


Datapath Units (DP)

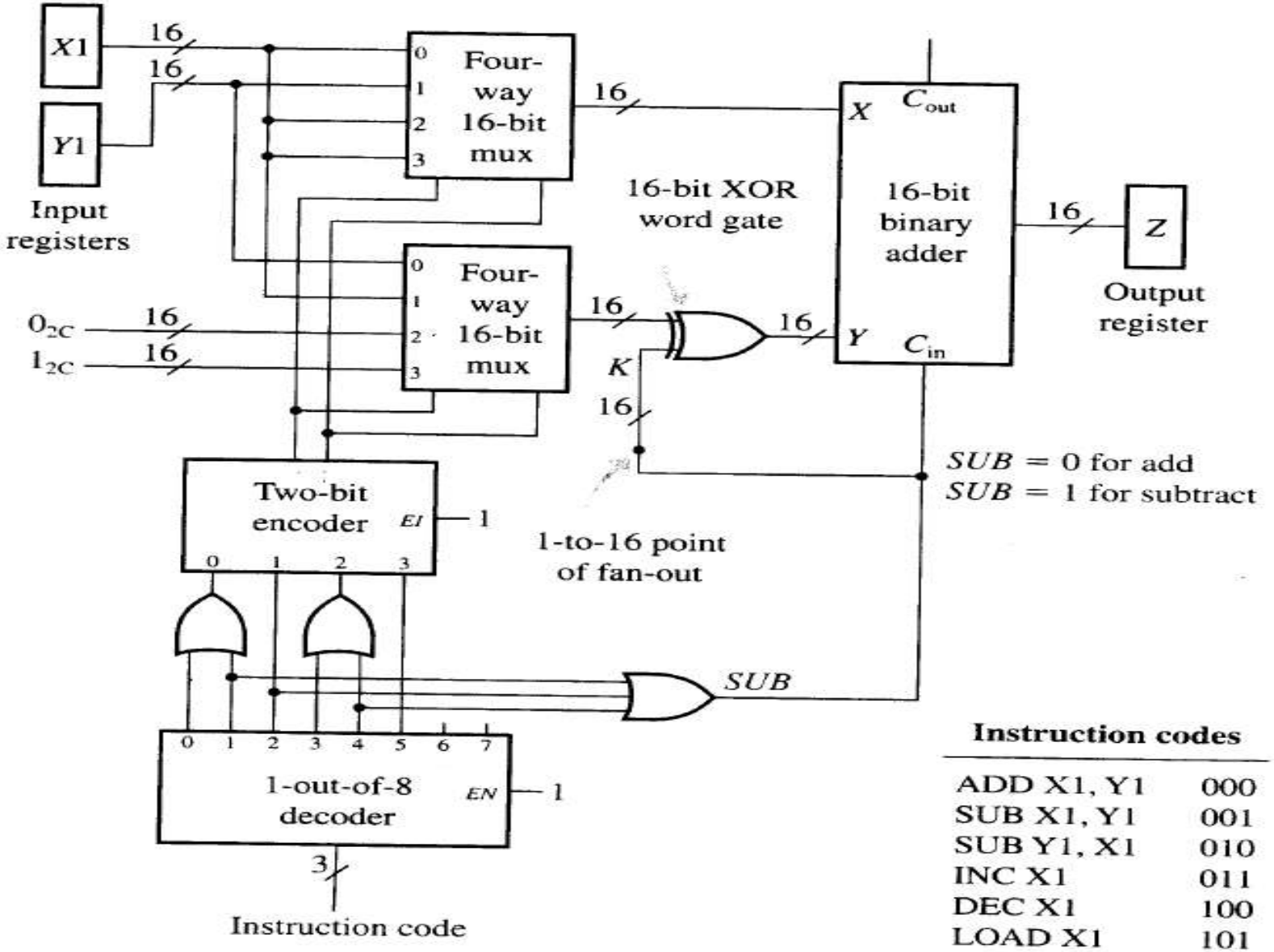
DP's are used to move and process data (in words).



Some datapath configurations: (a) an accumulator; (b) a three-stage pipeline.



A more detailed logic diagram for the accumulator-based datapath



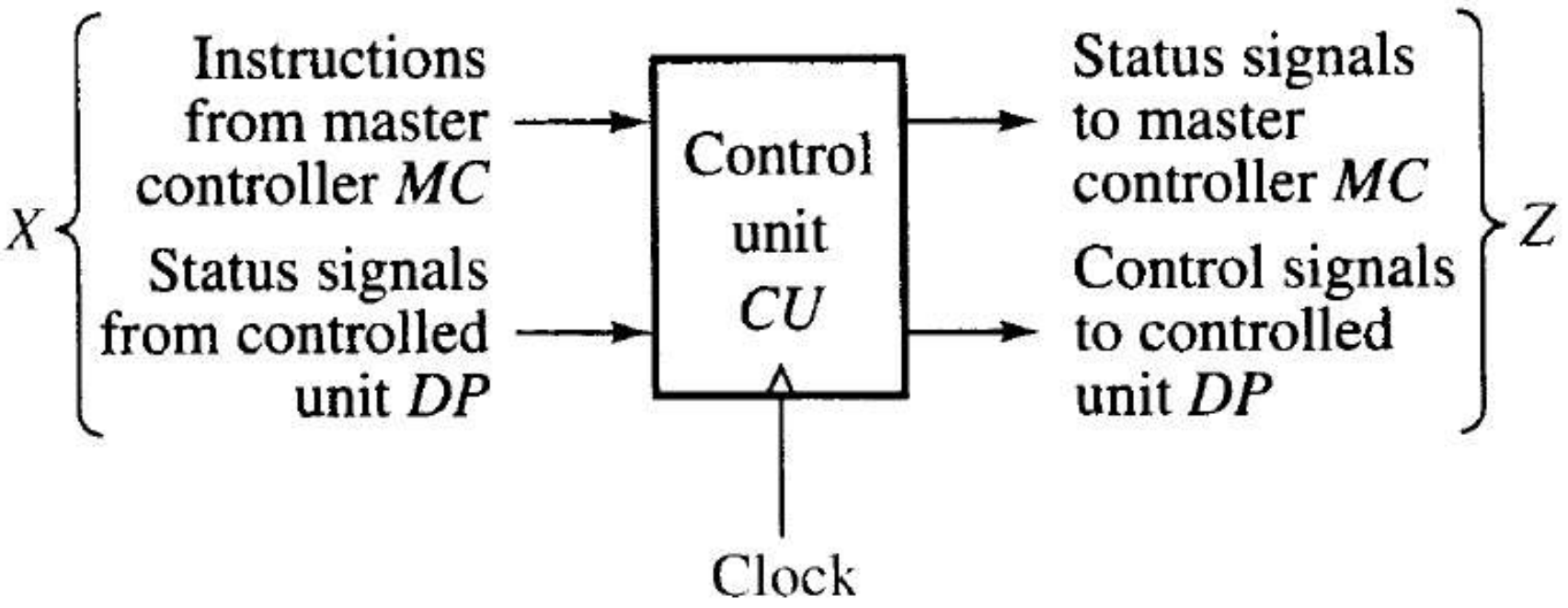
Instruction codes

ADD X1, Y1	000
SUB X1, Y1	001
SUB Y1, X1	010
INC X1	011
DEC X1	100
LOAD X1	101

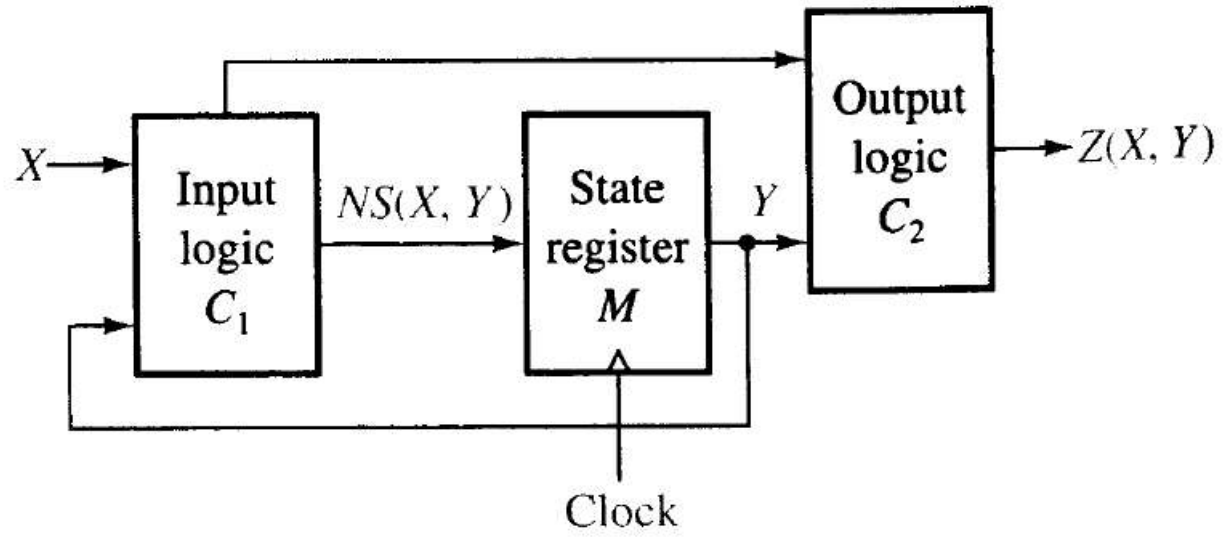
A 16-bit datapath unit to execute a small instruction set.

Control Units (CU)

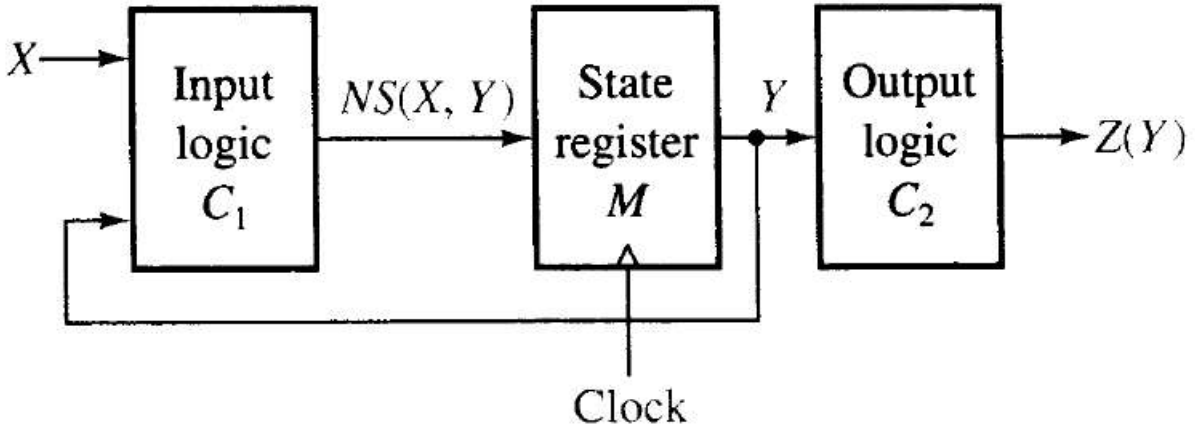
CU's direct the operations of DP's through a set of output control signals.



The Control Unit (CU) has a well-defined and fairly small set of states and is often implemented as a FSM.

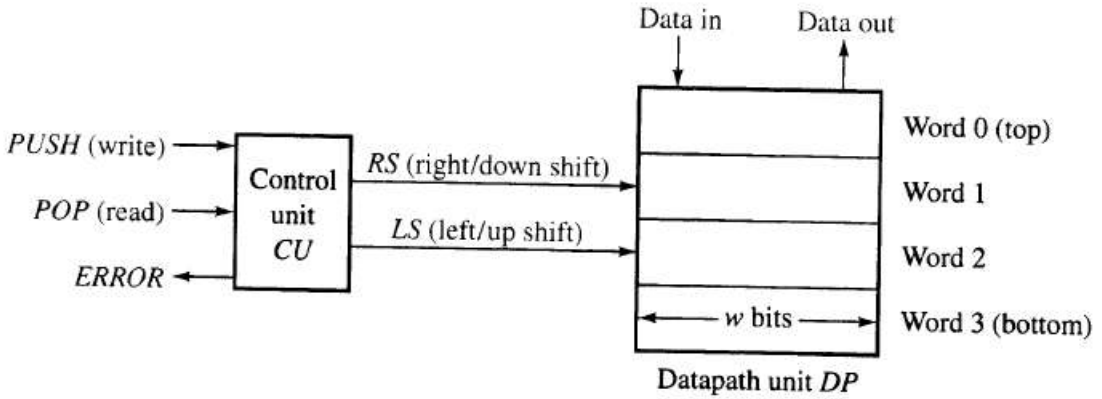


Mealy machine

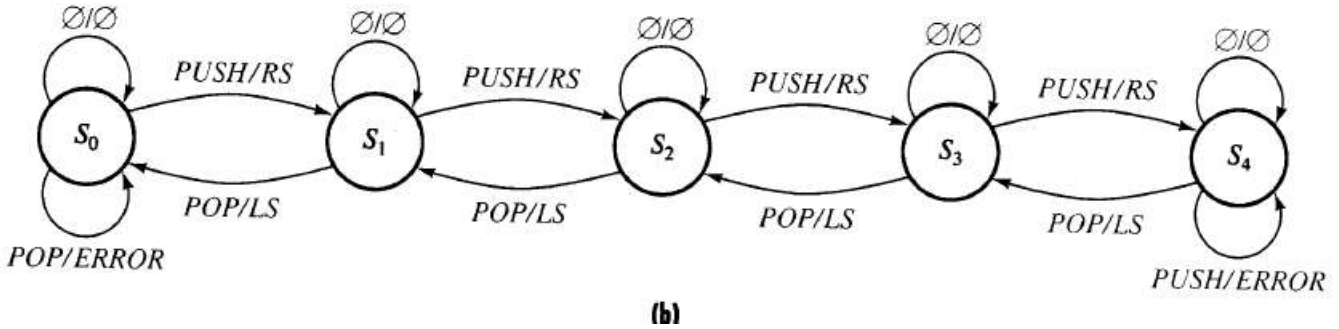


Moore machine

Example: A CU for a 4-word LIFO stack.

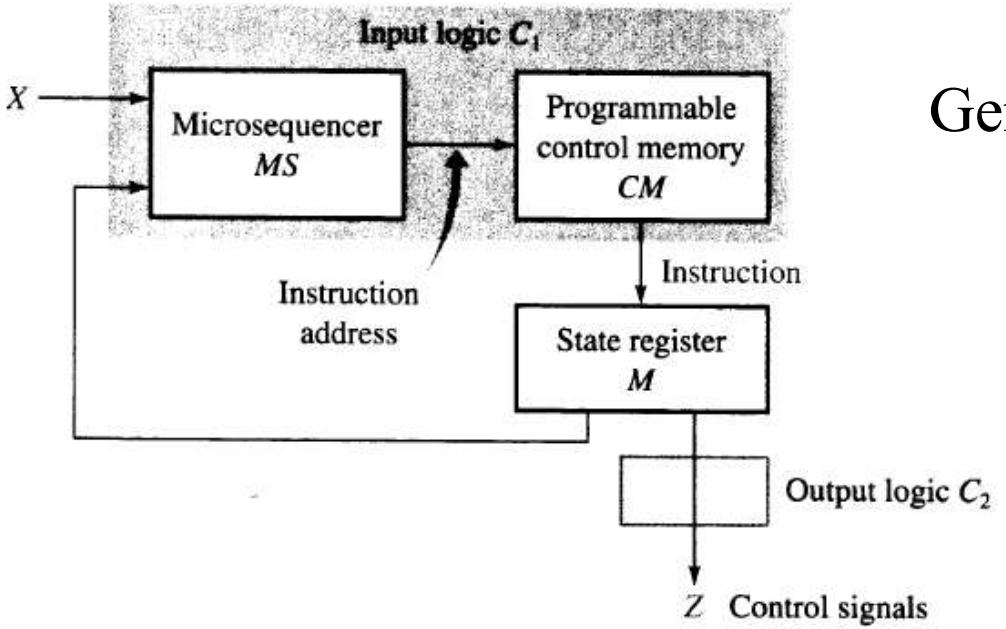


(a)



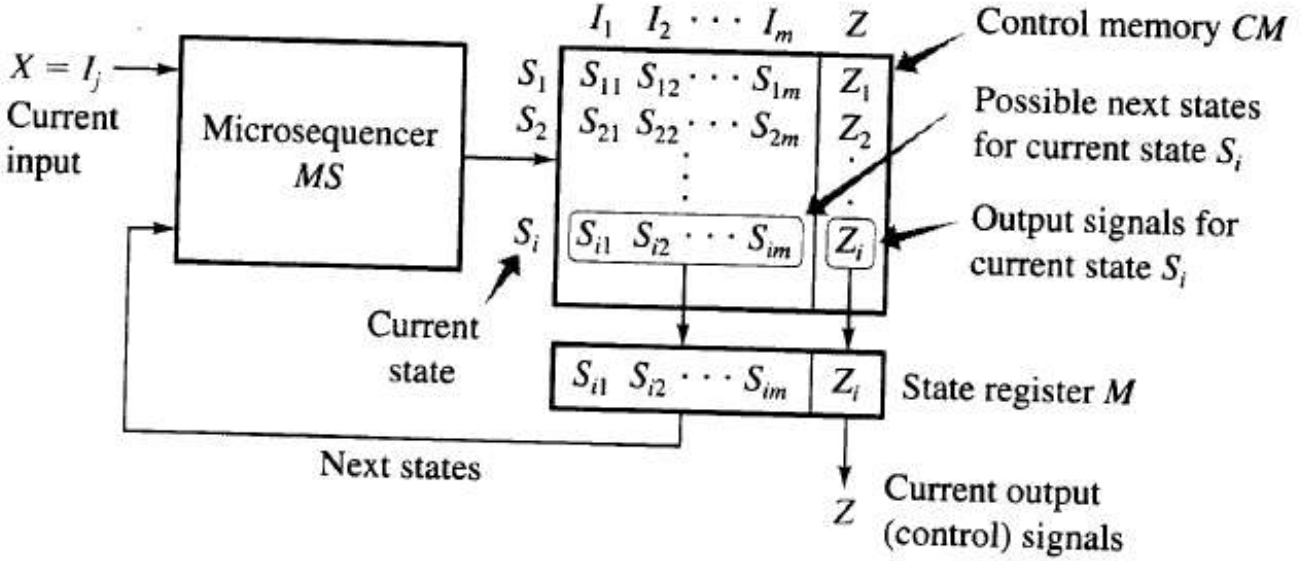
(b)

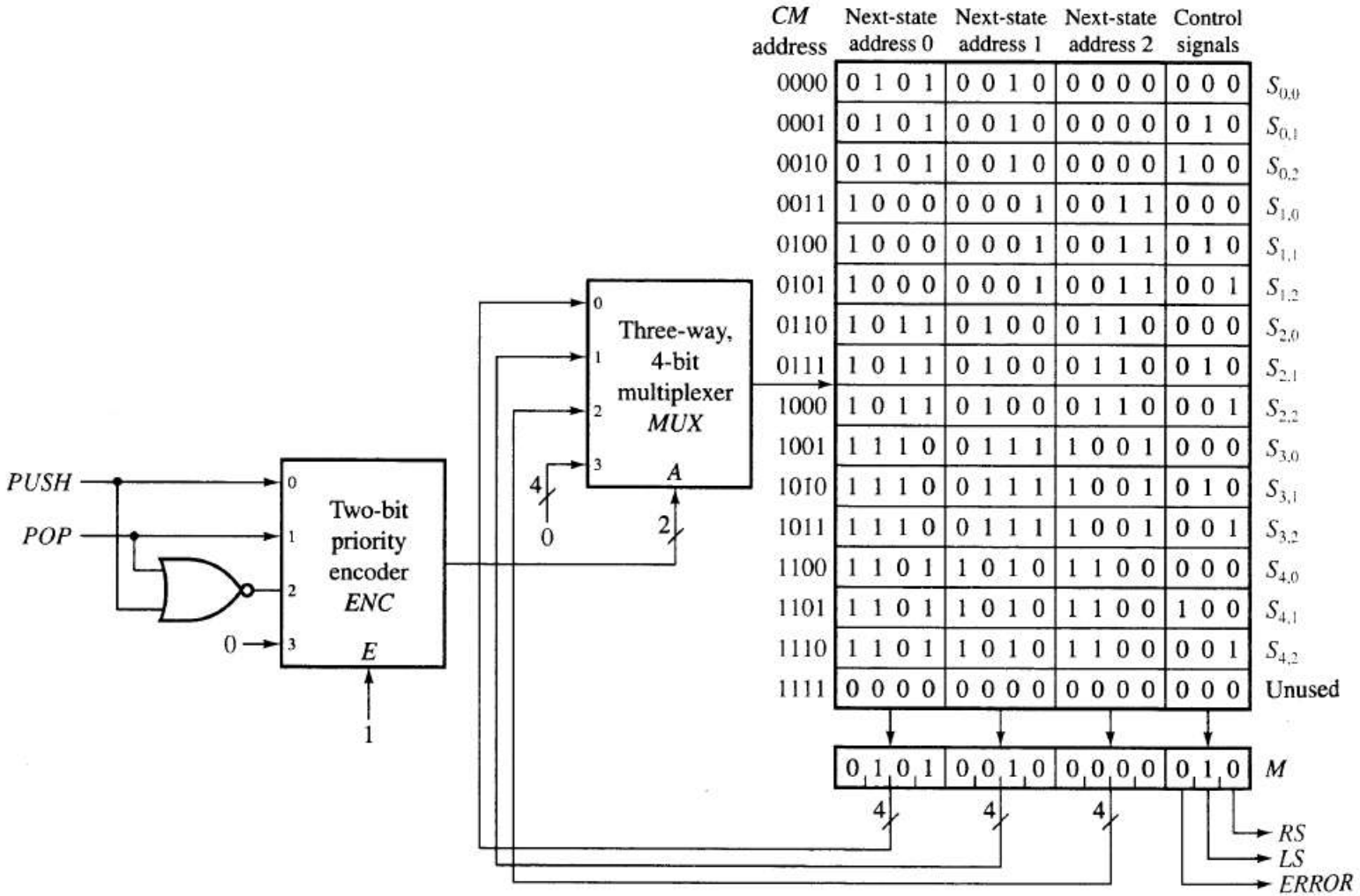
Programmable CU



General structure

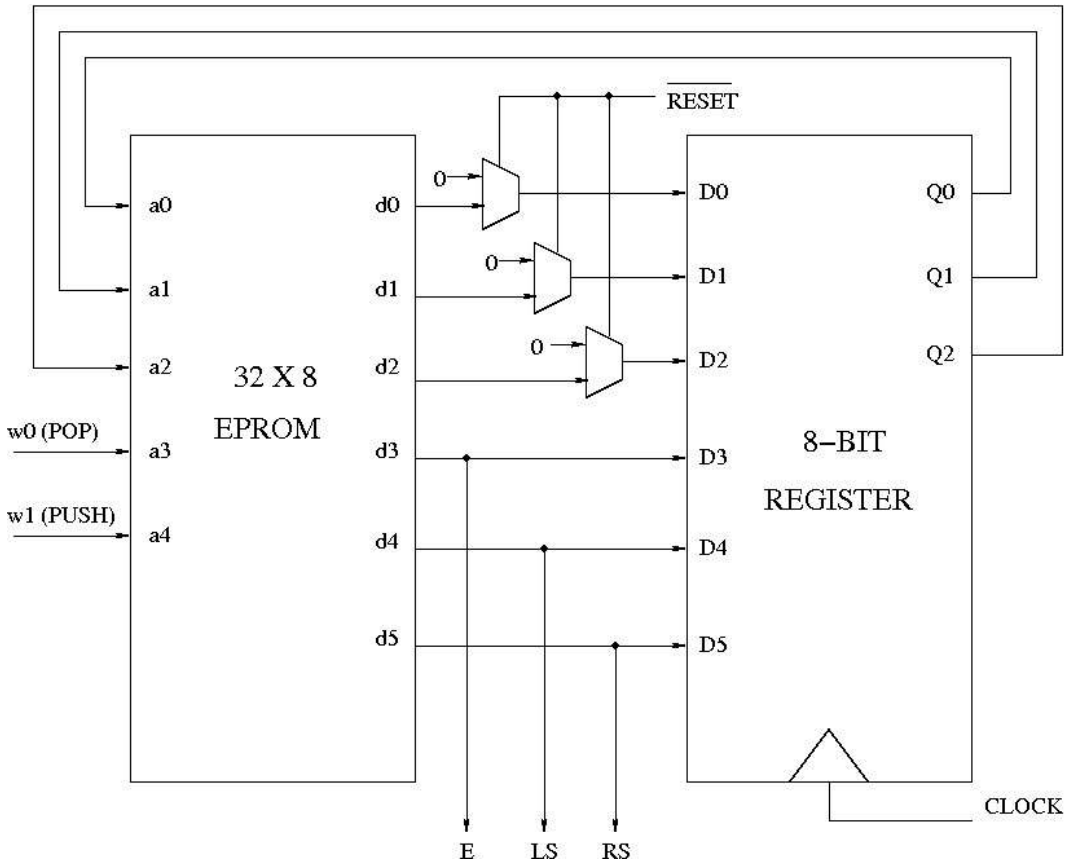
CM implemented with Moore-type state table.





A programmable implementation of a control unit for the stack memory.

Alternative Programmable CU for the 4-Word LIFO Stack



EPROM Address					EPROM Contents								
a4	a3	a2	a1	a0	d7	d6	d5	d4	d3	d2	d1	d0
push	pop	Present State								Next State		
w1	w0	y2	y1	y0			RS	LS	E	Y2	Y1	Y0
0	0	0	0	0			0	0	0	0	0	0
0	0	0	0	1			0	0	0	0	0	1
0	0	0	1	0			0	0	0	0	1	0
0	0	0	1	1			0	0	0	0	1	1
0	0	1	0	0			0	0	0	1	0	0
0	0	1	0	1			x	x	x	x	x	x
0	0	1	1	0			x	x	x	x	x	x
0	0	1	1	1			x	x	x	x	x	x
0	1	0	0	0			0	0	1	0	0	0
0	1	0	0	1			0	1	0	0	0	0
0	1	0	1	0			0	1	0	0	0	1
0	1	0	1	1			0	1	0	0	1	0
0	1	1	0	0			0	1	0	0	1	1
0	1	1	0	1			x	x	x	x	x	x
0	1	1	1	0			x	x	x	x	x	x
0	1	1	1	1			x	x	x	x	x	x
1	0	0	0	0			1	0	0	0	0	1
1	0	0	0	1			1	0	0	0	1	0
1	0	0	1	0			1	0	0	0	1	1
1	0	0	1	1			1	0	0	1	0	0
1	0	1	0	0			0	0	1	1	0	0
1	0	1	0	1			x	x	x	x	x	x
1	0	1	1	0			x	x	x	x	x	x
1	0	1	1	1			x	x	x	x	x	x
1	1	0	0	0			x	x	x	x	x	x
1	1	0	0	1			x	x	x	x	x	x
1	1	0	1	0			x	x	x	x	x	x
1	1	0	1	1			x	x	x	x	x	x
1	1	1	0	0			x	x	x	x	x	x
1	1	1	0	1			x	x	x	x	x	x
1	1	1	1	0			x	x	x	x	x	x
1	1	1	1	1			x	x	x	x	x	x

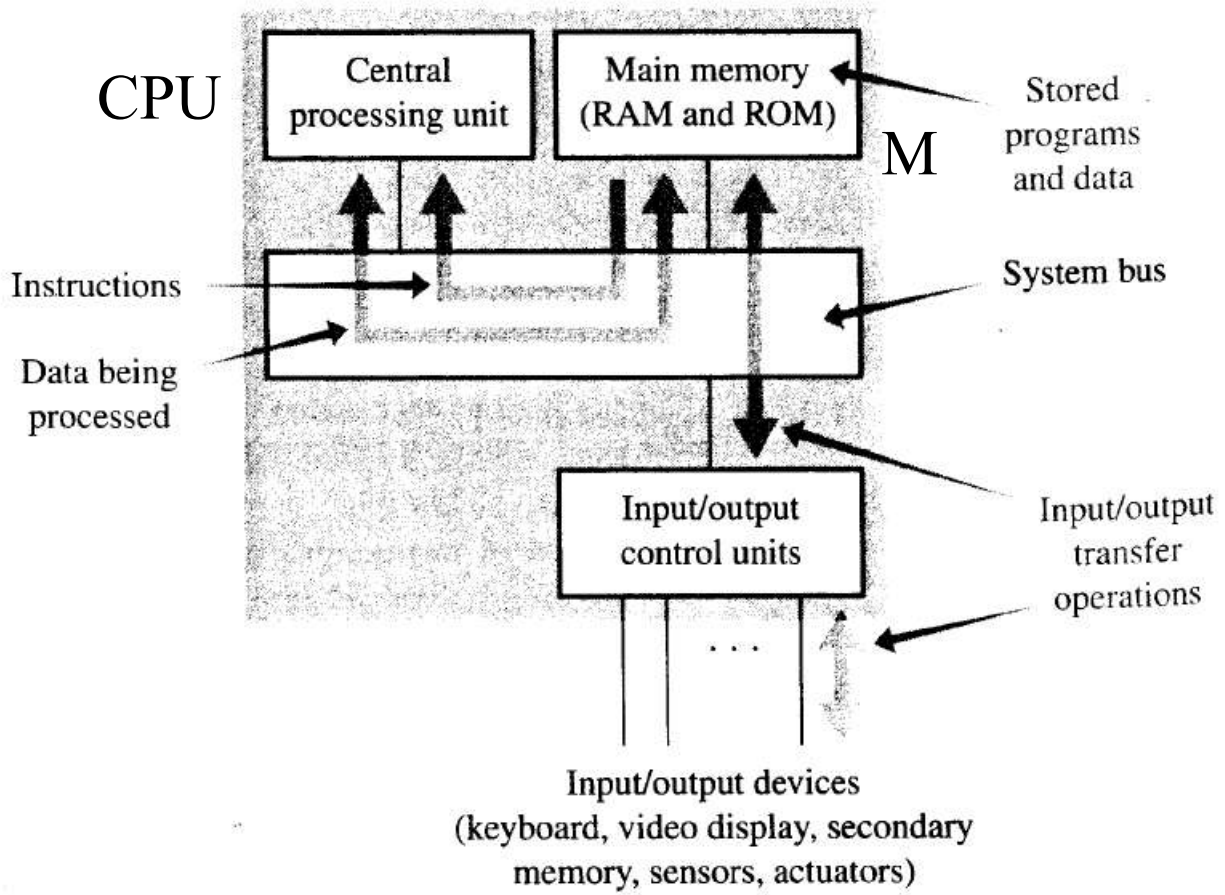
The Architecture Level

At the next higher level, we have the architecture level. This level deals with memories and processing units that store and process blocks of data words known as *program instructions* and *program data*.

The interconnection of the digital system to the outside world is also dealt with at this level through input/output circuits.

The major components at this level are: CPU, memory and I/O circuits. The operations of these components are determined by executing a series of control words, or program instructions.

Overall architecture of a digital computer.

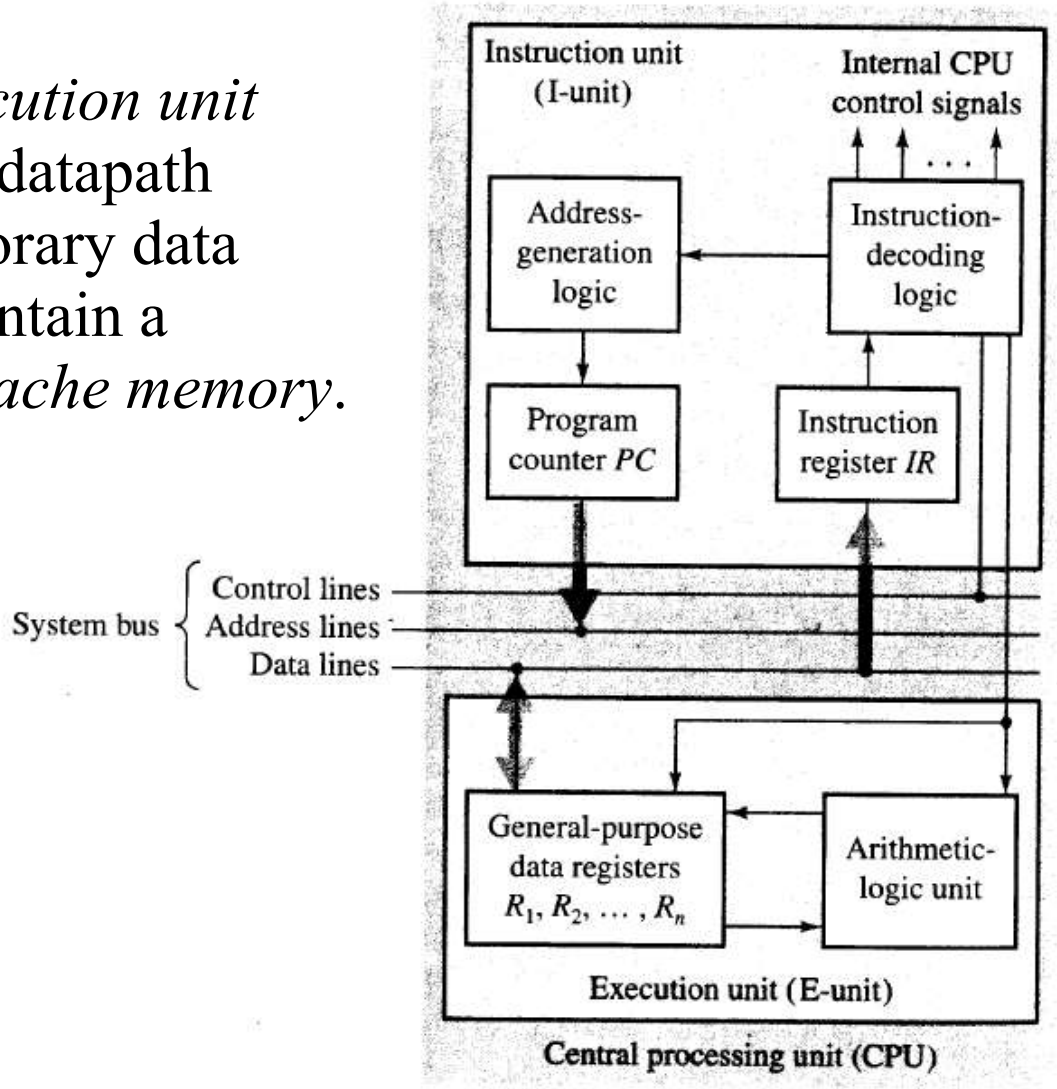


The primary function of the CPU is to fetch instructions from the main memory and execute them.

The CPU can be divided into 2 parts: the control part and the data part.

The control part is called the *instruction unit (I-unit)*. It contains an instruction register (IR), a program counter (PC), an instruction decoder.

The data part is called the *execution unit (E-unit)*. It contains at least a datapath and a set of registers for temporary data storage. Some E-units also contain a fast local memory called the *cache memory*.

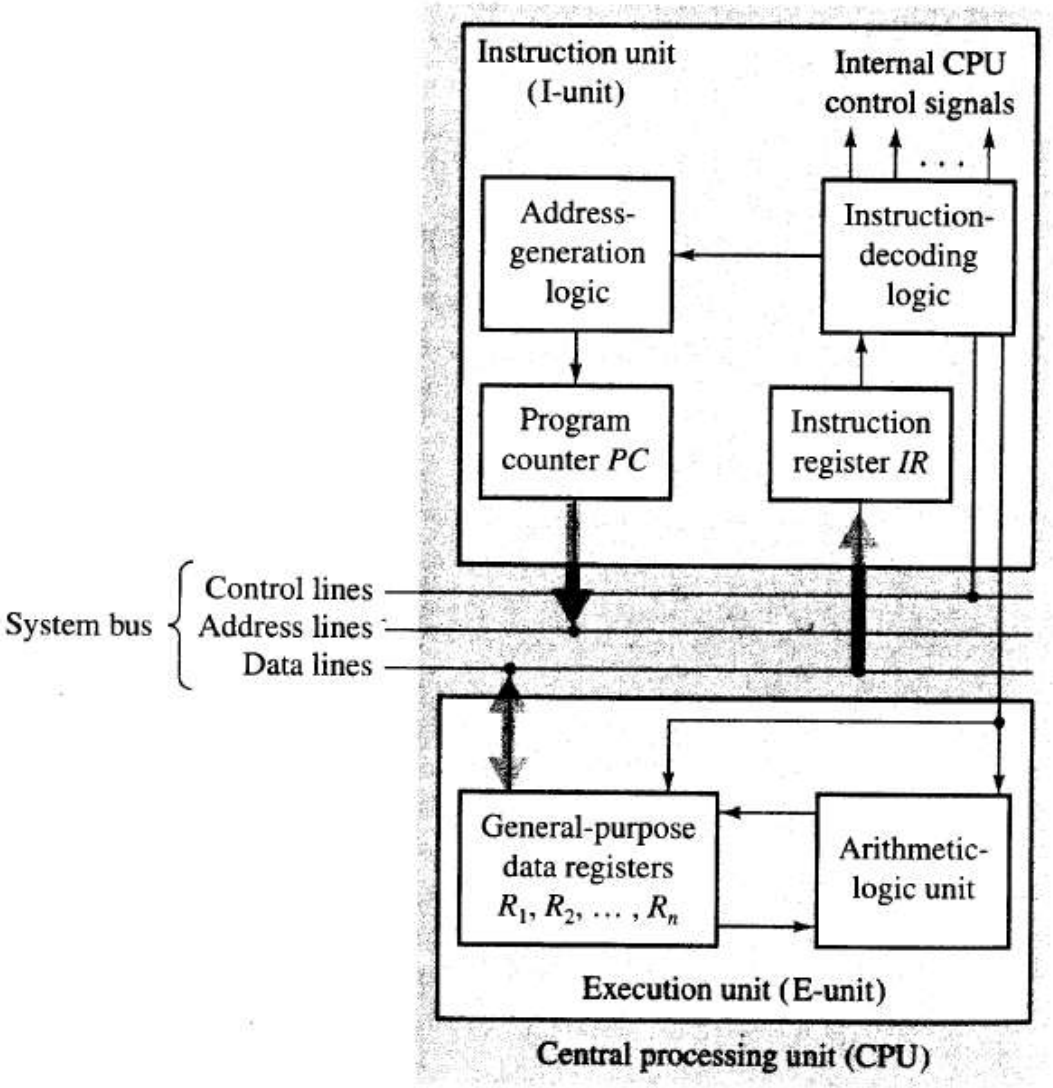


The Instruction Unit (I-Unit)

The I-unit is responsible for generating the control signals to ensure the proper operations and correct data flow are performed on the input data.

Its components are:

- Instruction register
- Program counter
- Instruction decoder
- Address generator



In order to perform computing operations, a CPU executes instructions from a program written in its *machine language*. The machine language of a CPU is simply a collection of instructions (called the *instruction set*) that define all the possible operations of the CPU.

A typical sequence of operations performed by the I-Unit is:

1. Load the location (address) of the first instruction of the program into the Program Counter (PC).

2. Load the instruction into the Instruction Register (IR).

3. Initiate the instruction decoder to decode the instruction so that the appropriate control and address signals for the datapath unit are generated.

4. Increment the value of the Program Counter (PC) by 1.

5. Repeated Steps 2 to 4 until the end of the program is reached.

The datapath consists of at least 2 main components: the register file, and the ALU.

- The ALU performs logical and arithmetic operations on the input data.
- The register file provides a set of fast local storage locations.

The interconnections of these units determine the allowable data flow paths. The properties of these units and how they are connected together define the *architecture* of the CPU.

Some datapaths also contain a local memory called the *cache memory*. It allows the CPU to access the main memory (M) more efficiently.

Instruction Set

The instruction set of a CPU is determined by its architecture.

A machine language instruction is a binary-encoded word. In other words, it is just a binary code. The code is derived from the property of the datapath and the data flow paths.

A machine language instruction usually consists of two parts: the *opcode* and the *operand*. Some instructions do not have operands.

For example:

The processor in Lab 7 uses 8-bit machine language instructions. For instructions with operands, the most significant 4 bits used as opcode and the remaining 4 bits as operand. For instructions without operands, the most significant 4 bits and the least significant bit form a 5-bit opcode.

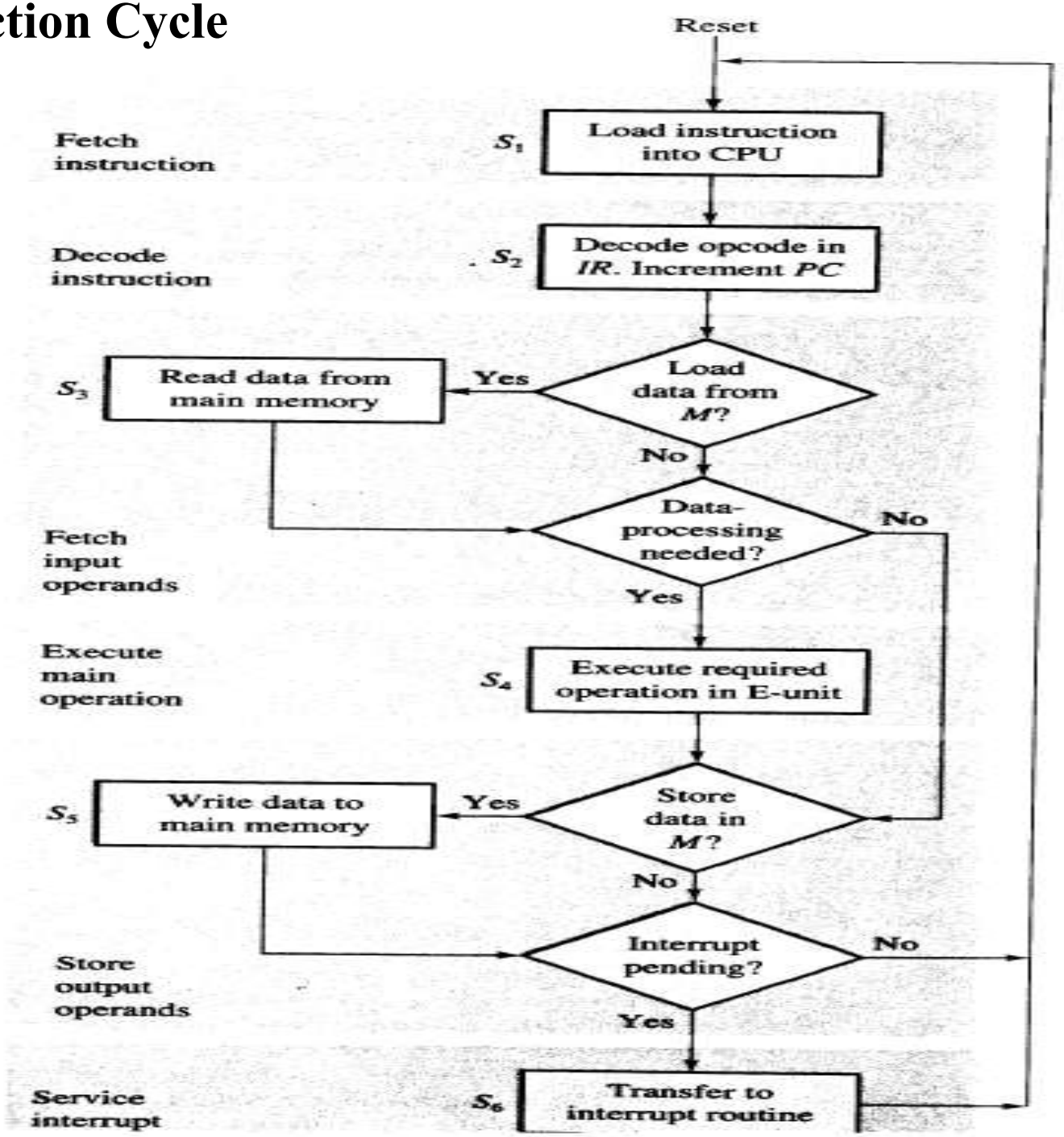
The instruction set of a typical commercial CPU is quite large, but can be categorized into the following groups:

1. Data-transfer instructions
   (load, store and I/O)

2. Data-processing instructions
   (arithmetic and logic operations)

3. Program-control instructions
   (conditional and unconditional branching)

General Instruction Cycle



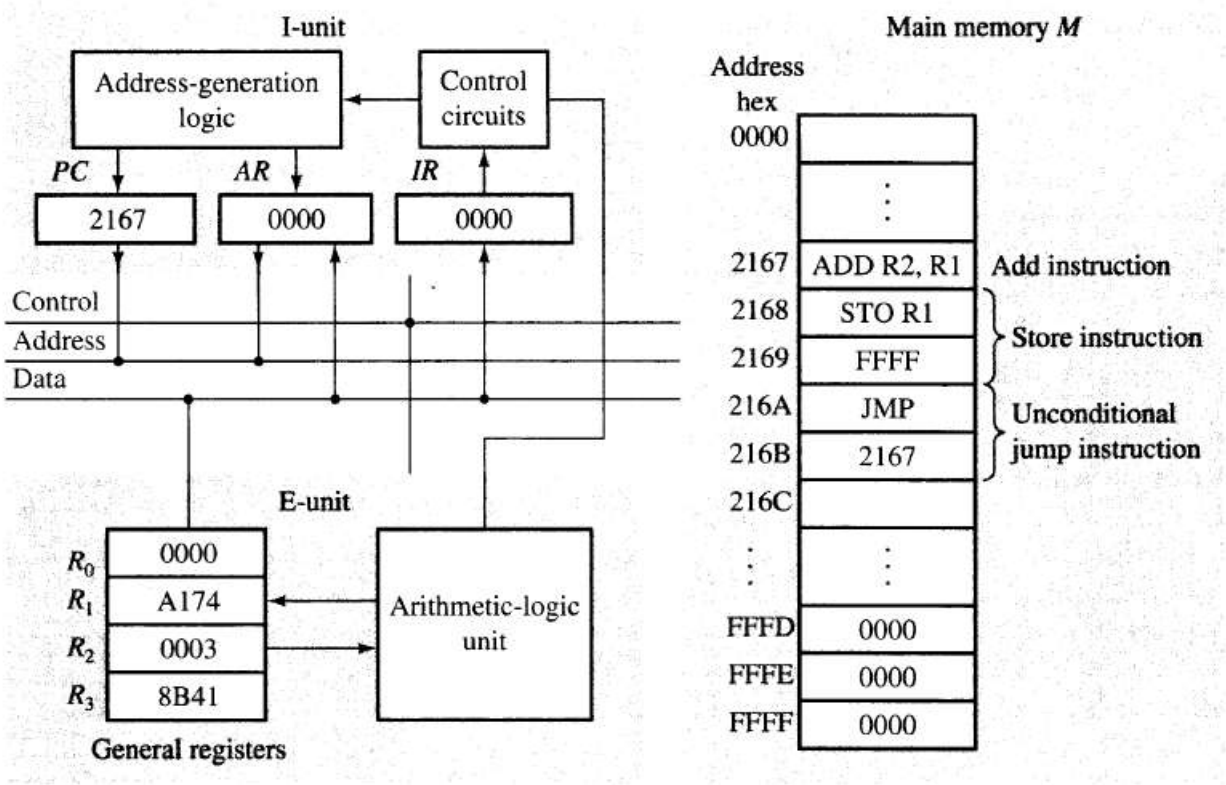
Example:

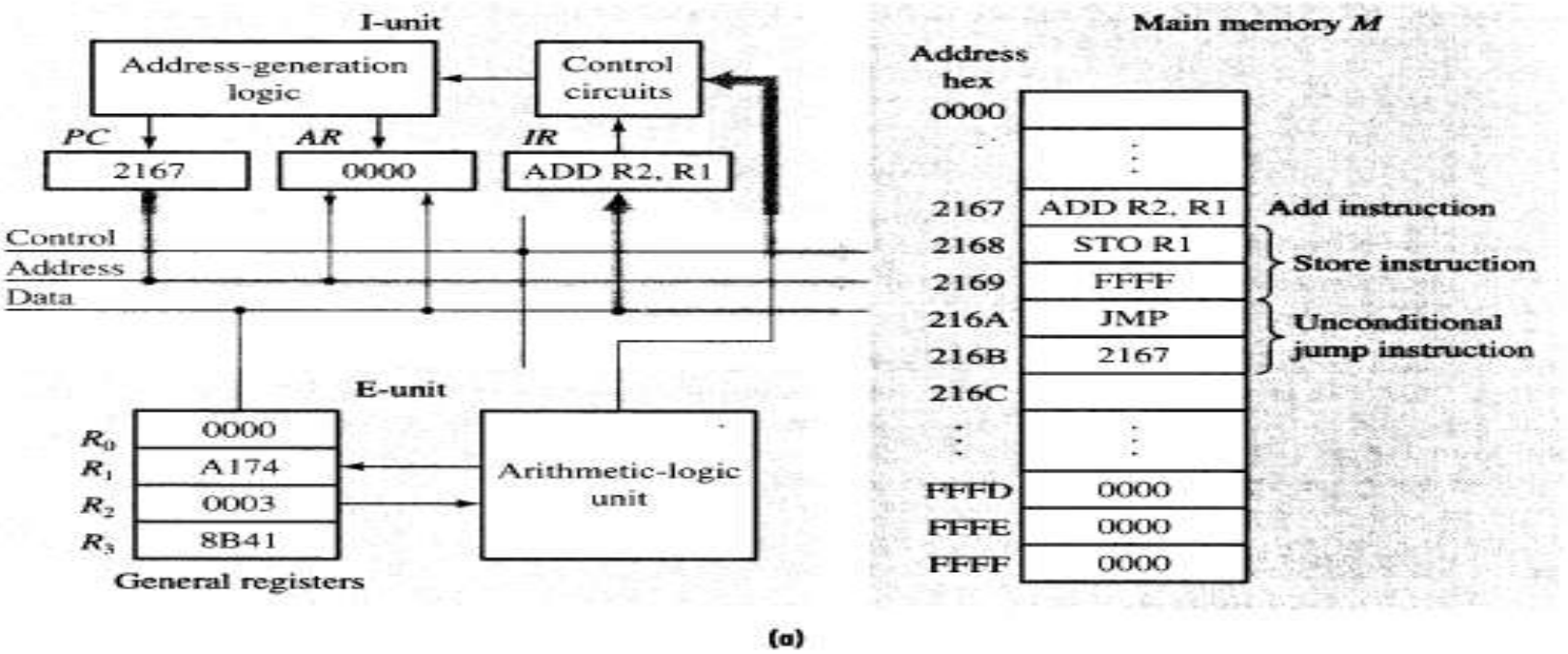
Program Segment:

ADD R2, R1 ; **R1 := (R1) + (R2)**

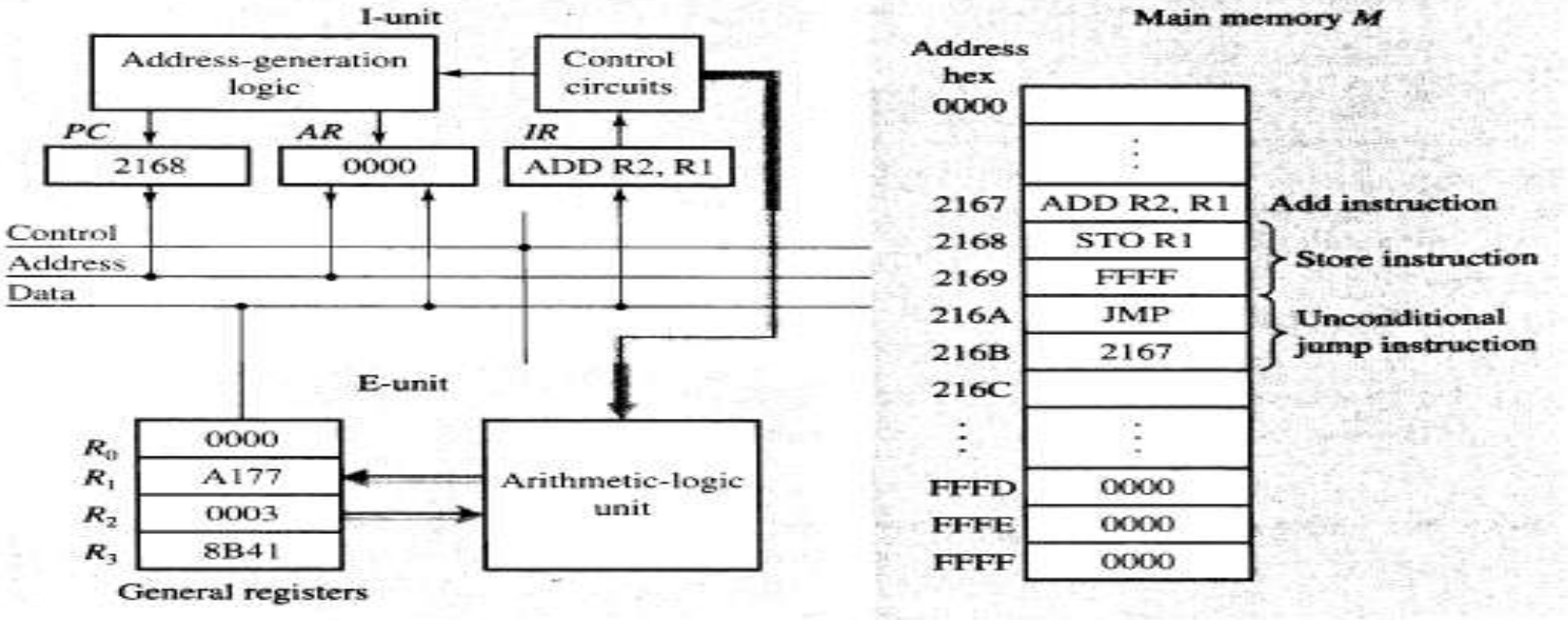
STO R1, FFFF ; **(FFFF) := R1**

JMP 2167 ; **Jump to address 2167**

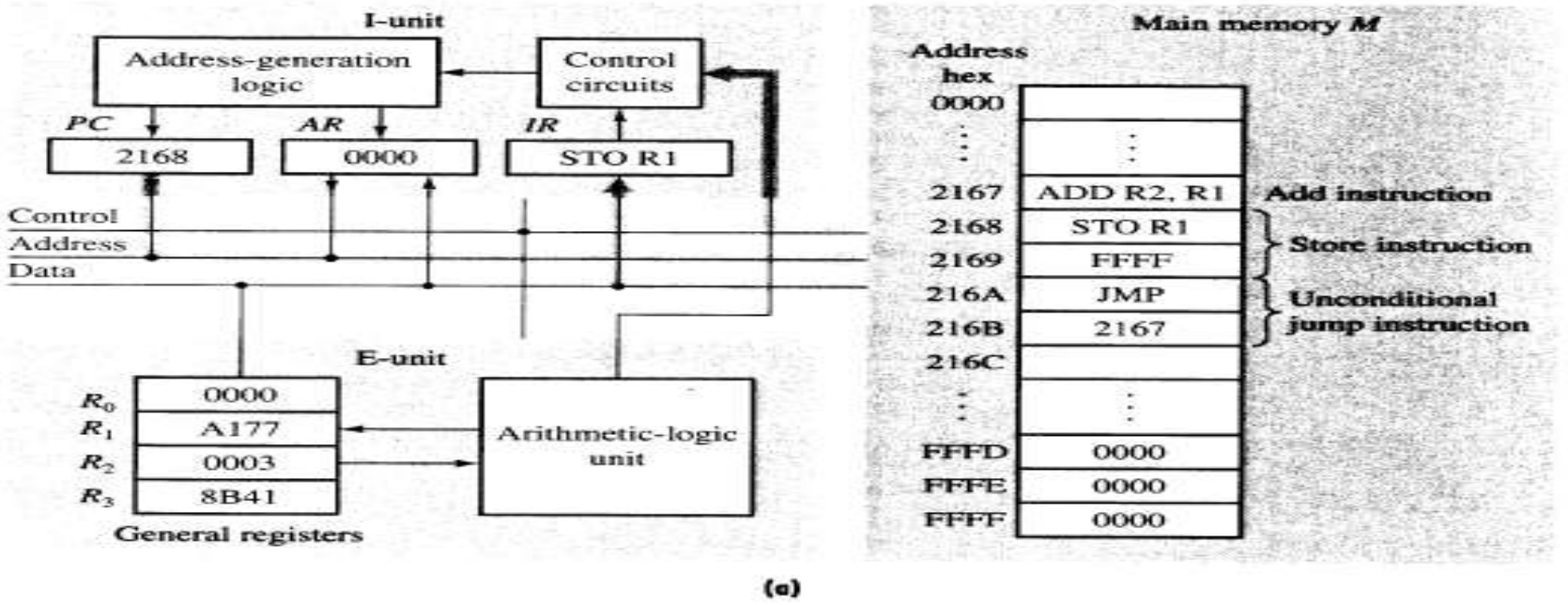




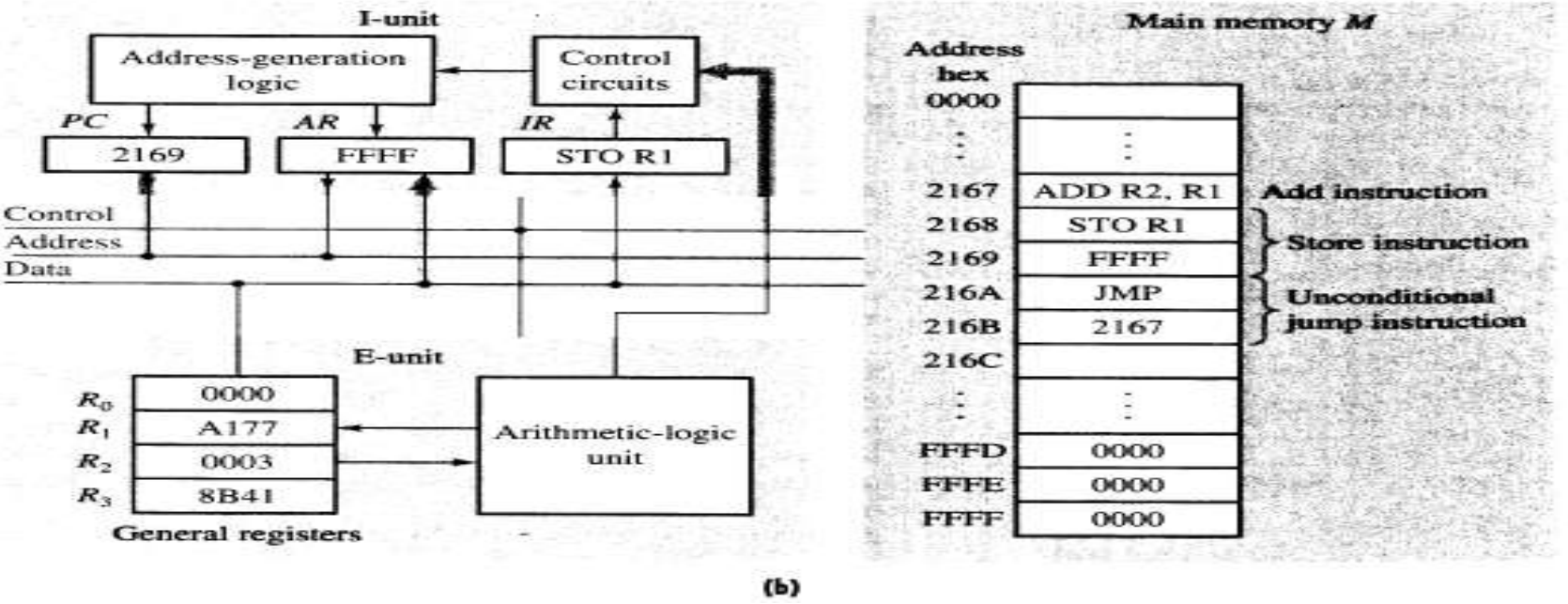
(a)



(a) Fetching and (b) executing a register-to-register add instruction.

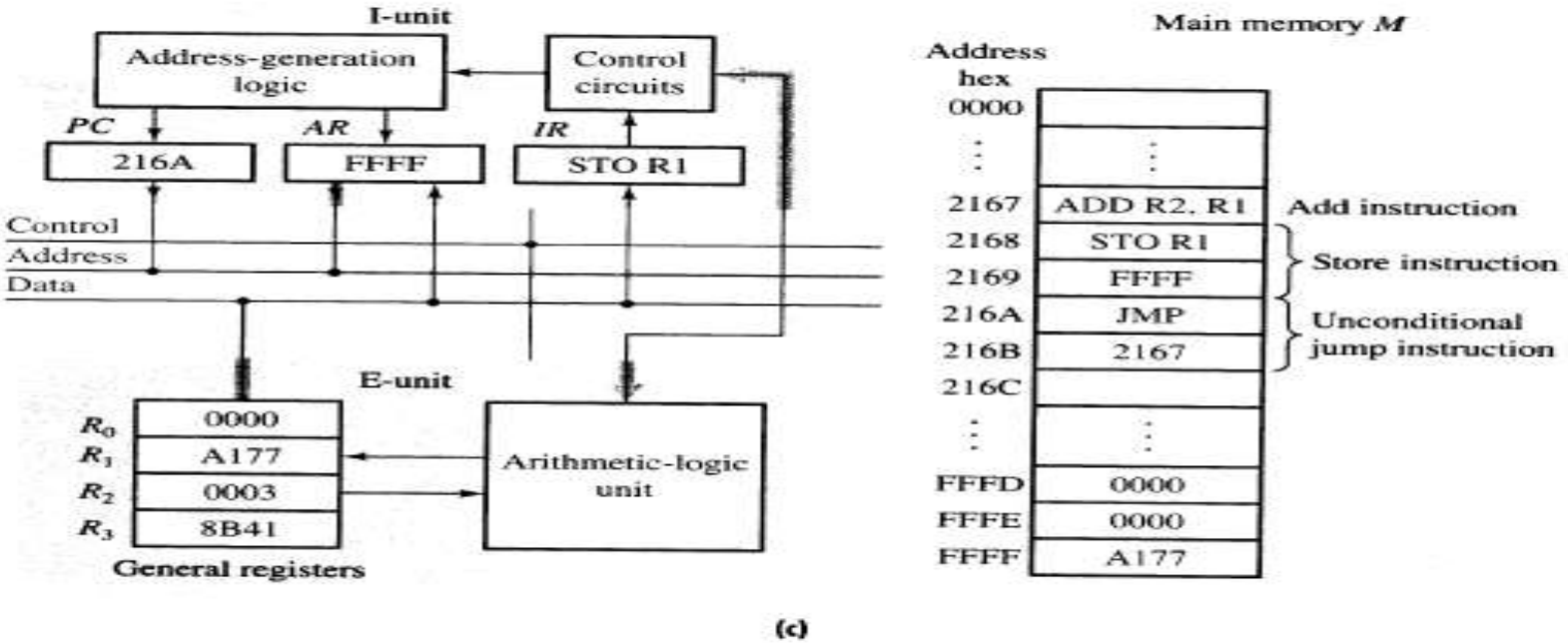


(a)



(b)

(a), (b) Fetching and (c) executing the store instruction.



(c)

(a), (b) Fetching and (c) executing the store instruction.

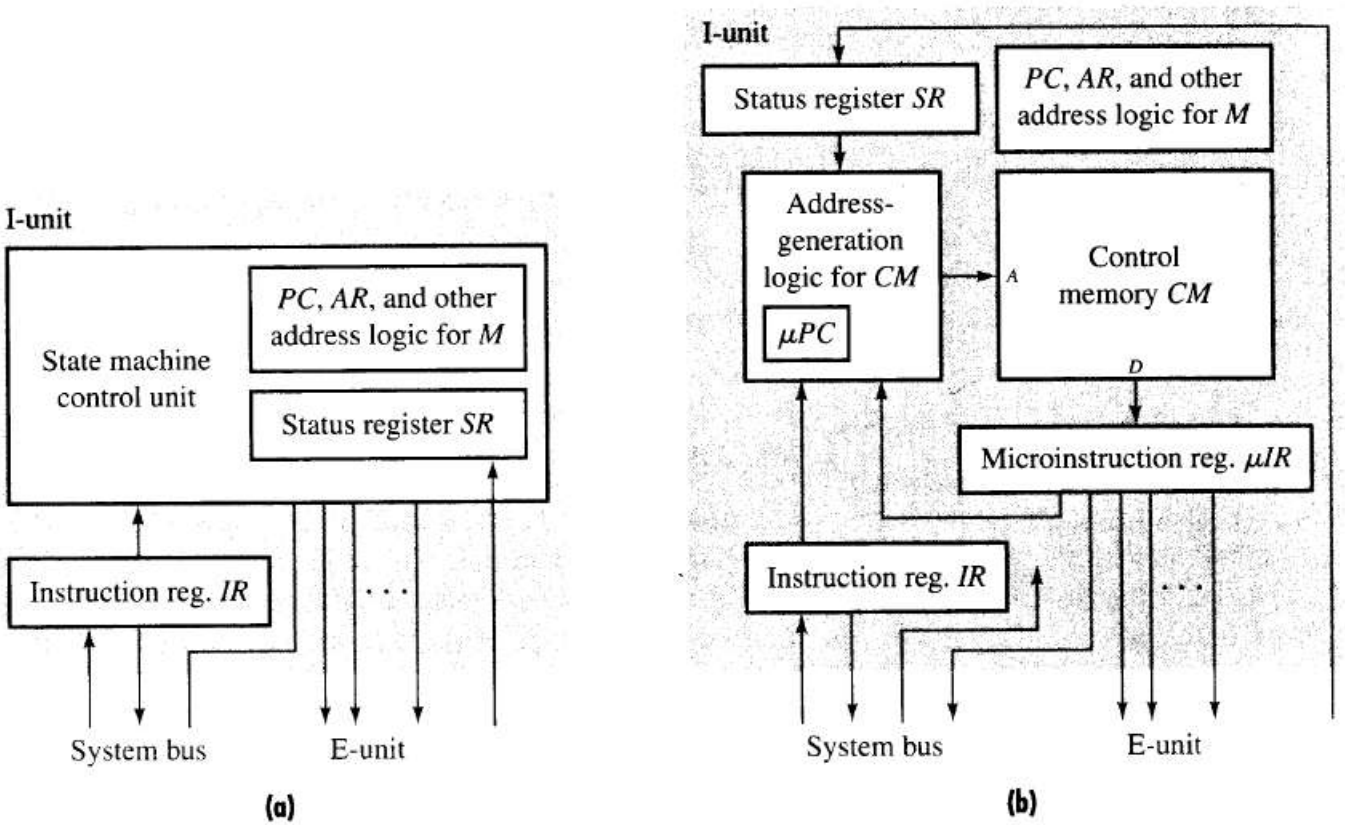
Structure of Typical I-Units

The I-unit is responsible for controlling the entire instruction cycle.
The actions to be taken depend on the instructions being executed.

An I-unit may be either *hardwired*, or *microprogrammed*.

Hardwired I-units: amount of hardware is minimized, speed is maximized.

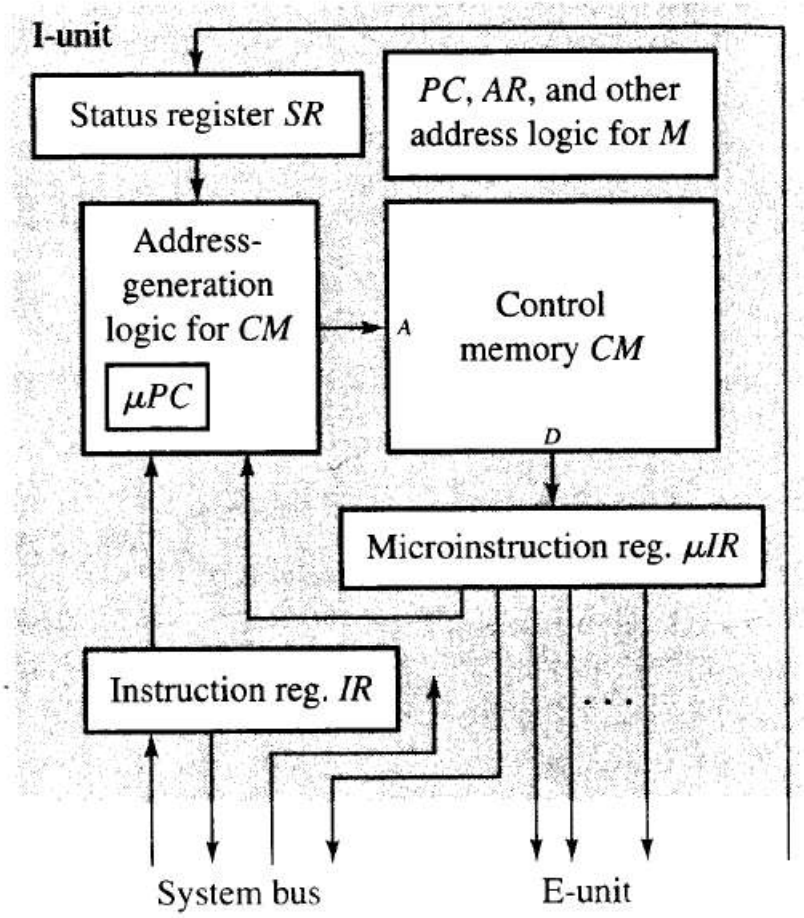
Microprogrammed I-units: flexible and expandable.



Internal structure of an I-unit: (a) hardwired; (b) microprogrammed.

In a microprogrammed I-unit, the machine language instruction of the processor is used to access a *microcode sequence* to perform a series of operations. The microcode sequence is stored in a microcode ROM called the *control memory (CM)*.

The microcode basically contain values of control signals for the CPU.



A typical microcode consists of a series of bits of the form:

$$c_0 \;\; c_1 \;\; c_2 \;\; c_3 \;\; \cdots \;\; c_{q-2} \;\; c_{q-1} \;\; next\_addr$$

where

c_i = the desired value (0 or 1) of some control signal for the CPU

$next\_addr$ = the address in CM of the next microcode

A group of microcode that performs a specific task is called a *microprogram*. The collection of all microprograms of a CPU is called its *firmware*.