Central Processing Unit

The CPU is made up of three major parts, as shown in Fig(25).

1- The register set stores intermediate data used during the execution of the instructions. The arithmetic
2- logic unit (ALU) performs the required microoperations for executing the instructions.
3- The control unit supervises the transfer of information among the registers and instructs the ALU as to which operation to perform.

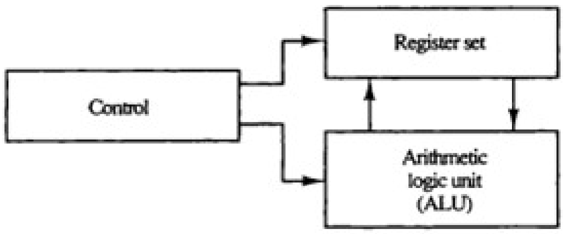

Figure 25 Major components of CPU.

General Register Organization

The memory locations are needed for storing pointers, counters, return addresses, temporary results, and partial products during multiplication.

A bus organization for seven CPU registers is shown in Fig(26):

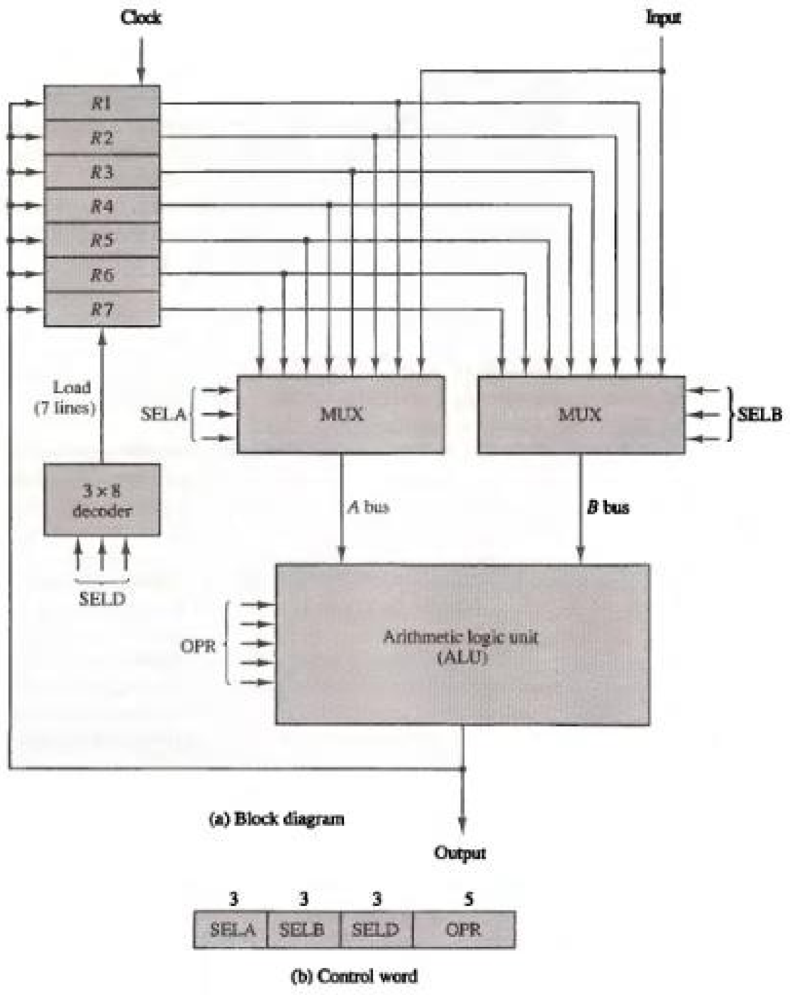

Figure 26 Register set with common ALU.

The control unit that operates the CPU bus system directs the information flow through the registers and ALU by selecting the various components in the system. For example, to perform the operation:

$$R1 \leftarrow R2 + R3$$

The control must provide binary selection variables to the following selector inputs:

1. MUX A selector (SELA): to place the content of R2 into bus A.
2. MUX B selector (SELB): to place the content of R3 into bus B.
3. ALU operation selector (OPR): to provide the arithmetic addition A + B.
4. Decoder destination selector (SELD): to transfer the content of the output bus into Rl.

To achieve a fast response time, the ALU is constructed with high-speed circuits.

There are 14 binary selection inputs in the unit, and their combined value control word specifies a control word. The three bits of SELA select a source register for the A input of the ALU. The three bits of SELB select a register for the B input of the ALU. The three bits of SELD select a destination register using the decoder and its seven load outputs. The five bits of OPR select one of the operations in the ALU.

The encoding of the register selections is specified in Table(14):

TABLE 14 Encoding of Register Selection Fields

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	*R*1	*R*1	*R*1
010	*R*2	*R*2	*R*2
011	*R*3	*R*3	*R*3
100	*R*4	*R*4	*R*4
101	*R*5	*R*5	*R*5
110	*R*6	*R*6	*R*6
111	*R*7	*R*7	*R*7

Table(15) OPR field has five bits and each operation is designated with a symbolic name.

TABLE 15 Encoding of ALU Operations

OPR Select	Operation	Symbol
00000	Transfer *A*	TSFA
00001	Increment *A*	INCA
00010	Add $A + B$	ADD
00101	Subtract $A - B$	SUB
00110	Decrement *A*	DECA
01000	AND *A* and *B*	AND
01010	OR *A* and *B*	OR
01100	XOR *A* and *B*	XOR
01110	Complement *A*	COMA
10000	Shift right *A*	SHRA
11000	Shift left *A*	SHLA

For example, the subtract microoperation given by the statement:

$$R1 \leftarrow R2 - R3$$

The binary control word for the subtract microoperation is 010 011 001 00101 and is obtained as follows:

Field:	SELA	SELB	SELD	OPR
Symbol:	R2	R3	R1	SUB
Control word:	010	011	001	00101

Stack Organization

A useful feature that is included in the CPU of most computers is a stack or last-in, first-out (LIFO) list. The two operations of a stack are the insertion and deletion of items. The operation of insertion is called push, while the operation of deletion is called pop. In a 64-word stack, the stack pointer contains 6 bits because $2^6 = 64$.

The push operation is implemented with the following sequence of microoperations:

$SP \leftarrow SP + 1$	Increment stack pointer
$M[SP] \leftarrow DR$	Write item on top of the stack
If $(SP = 0)$ then $(FULL \leftarrow 1)$	Check if stack is full
$EMTY \leftarrow 0$	Mark the stack not empty

The pop operation consists of the following sequence of microoperations:

$DR \leftarrow M[SP]$	Read item from the top of stack
$SP \leftarrow SP - 1$	Decrement stack pointer
If $(SP = 0)$ then $(EMTY \leftarrow 1)$	Check if stack is empty
$FULL \leftarrow 0$	Mark the stack not full

Instruction Formats

The format of an instruction is usually depicted in a rectangular box symbolizing the bits of the instruction as they appear in memory words or in a control register. The bits of the instruction are divided into groups called fields. The most common fields found in instruction formats are:

1. An operation code field that specifies the operation to be performed.
2. An address field that designates a memory address or a processor register.
3. A mode field that specifies the way the operand or the effective address is determined.

An example of an accumulator-type organization, the instruction that specifies an arithmetic addition is defined by an assembly language instruction as:

$$ADD\ \ X$$

Where X is the address of the operand. The ADD instruction in this case results in the operation:

$$AC \leftarrow AC + M[X]$$

An example of a general register type of organization the instruction for an arithmetic addition may be written in an assembly language as:

$$ADD\ R1, R2, R3$$

to denote the operation:

$$R1 \leftarrow R2 + R3$$

The following is a program to evaluate $X = (A + b) * (C + D)$:
with three-address instruction formats as:

```
ADD   R1, A, B    R1←M[A] + M[B]
ADD   R2, C, D    R2←M[C] + M[D]
MUL   X, R1, R2   M[X]←R1*R2
```

Two-address instructions formats as:

```
MOV   R1, A    R1←M[A]
ADD   R1, B    R1←R1 + M[B]
MOV   R2, C    R2←M[C]
ADD   R2, D    R2←R2 + M[D]
MUL   R1,R2    R1←R1*R2
MOV   X, R1    M[X]←R1
```

One-address instructions use an implied accumulator (AC) register for all data manipulation as:

```
LOAD    A   AC←M[A]
ADD     B   AC←AC + M[B]
STORE   T   M[T]←AC
LOAD    C   AC←M[C]
ADD     D   AC←AC + M[D]
MUL     T   AC←AC*M[T]
STORE   X   M[X]←AC
```

<u>Addressing Modes</u>

Computers use addressing mode techniques for the purpose of accommodating one or both of the following provisions:

1. To give programming versatility to the user by providing such facilities as pointers to memory, counters for loop control, indexing of data, and program relocation.
2. To reduce the number of bits in the addressing field of the instruction.

PC holds the address of the instruction to be executed next and is incremented each time an instruction is fetched from memory. An example of an instruction format with a distinct addressing mode field is shown in Fig(27).

Opcode	Mode	Address

Figure 27 Instruction format with mode field.

Immediate Mode: In this mode the operand is specified in the instruction itself. In other words, an immediate-mode instruction has an operand field rather than an address field.

Register Mode: In this mode the operands are in registers that reside within the CPU. The particular register is selected from a register field in the instruction.

Register Indirect Mode: In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory.

Auto-increment or Auto-decrement Mode: This is similar to the register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory.

The effective address is defined to be the memory address obtained from the computation dictated by the given addressing mode.

Direct Address Mode: In this mode the effective address is equal to the address part of the instruction.

Indirect Address Mode: In this mode the address field of the instruction gives the address where the effective address is stored in memory.

The effective address in these modes is obtained from the following computation:

effective address = address part of instruction + content of CPU register

Relative Address Mode: In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address.

Indexed Addressing Mode: In this mode the content of an index register is added to the address part of the instruction to obtain the effective address.

Base Register Addressing Mode: In this mode the content of a base register is added to the address part of the instruction to obtain the effective address.

Numerical Example:

1- The two-word instruction at address 200 and 201 is a "load to AC" instruction with an address field equal to 500.

2- The first word of the instruction specifies the operation code and mode, and the second word specifies the address part.

3- PC has the value 200 for fetching this instruction.

4- The content of processor register Rl is 400, and the content of an index register XR is 100.

AC receives the operand after the instruction is executed. The Fig(28) lists a few pertinent addresses and shows the memory content at each of these addresses.

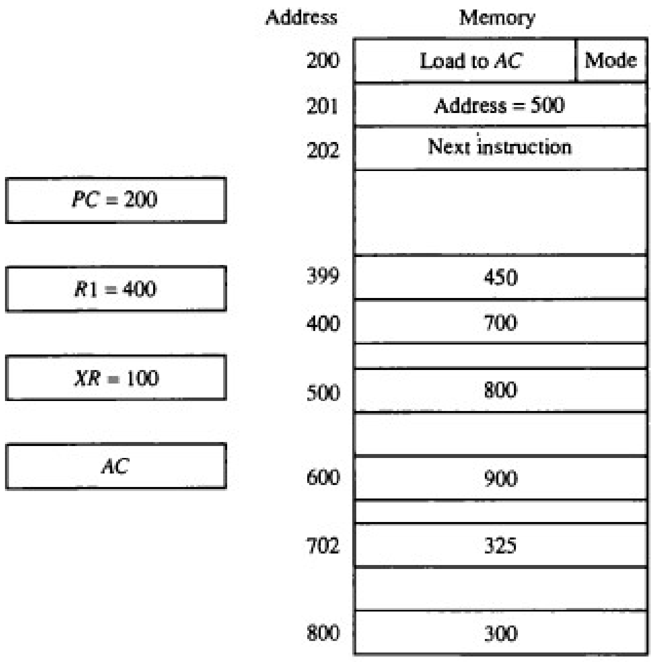

Figure 28 Numerical example for addressing modes.

Answer:

1- *In the direct address mode* the effective address is the address part of the instruction 500 and the operand to be loaded into AC is 800.
2- *In the immediate mode* the second word of the instruction is taken as the operand rather than an address, so 500 is loaded into AC. (The effective address in this case is 201)
3- *In the indirect mode* the effective address is stored in memory at address 500. Therefore, the effective address is 800 and the operand is 300.
4- *In the relative mode* the effective address is 500 + 202 = 702 and the operand is 325. (Note that the value in PC after the fetch phase and during the execute phase is 202)
5- *In the index mode* the effective address is XR + 500 = 100 + 500 = 600 and the operand is 900.
6- *In the register mode* the operand is in Rl and 400 is loaded into AC. (There is no effective address in this case)
7- *In the register indirect mode* the effective address is 400, equal to the content of Rl and the operand loaded into AC is 700.
8- *The auto-increment mode* is the same as the register indirect mode except that Rl is incremented to 401 after the execution of the instruction.
9- *The auto-decrement mode* decrements Rl to 399 prior to the execution of the instruction. The operand loaded into AC is now 450.

Table (16) lists the values of the effective address and the operand loaded into AC for the nine addressing modes.

TABLE 16 Tabular List of Numerical Example

Addressing Mode	Effective Address	Content of *AC*
Direct address	500	800
Immediate operand	201	500
Indirect address	800	300
Relative address	702	325
Indexed address	600	900
Register	—	400
Register indirect	400	700
Autoincrement	400	700
Autodecrement	399	450

Data Transfer and Manipulation

Most computer instructions can be classified into three categories:

1. Data transfer instructions.
2. Data manipulation instructions.
3. Program control instructions.

Data transfer instructions cause transfer of data from one location to another without changingthe binary information content. The table(17) list the Data

TABLE 17 Typical Data Transfer Instructions

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

transfer instructions:

Data manipulation instructions are those that perform arithmetic, logic, and shift operations. The data manipulation instructions in a typical computer are usually divided into three basic types:

1- Arithmetic instructions.

2. Logical and bit manipulation instructions.

3. Shift instructions.