THE CENTRAL PROCESSING UNIT

i) Introduction

CPU is actually the main brain of the computer system. It is here that the whole processing takes place. It consists of 4 main units: Primary storage, Arithmetic logic unit, control unit and output unit

Primary storage unit: this is main storage area, which is also called the main memory area. Its main functions are 1) holding the data in its memory till it is required to be processed. 2) holding the result of the processed data 3) holding program instructions which are required for processing.

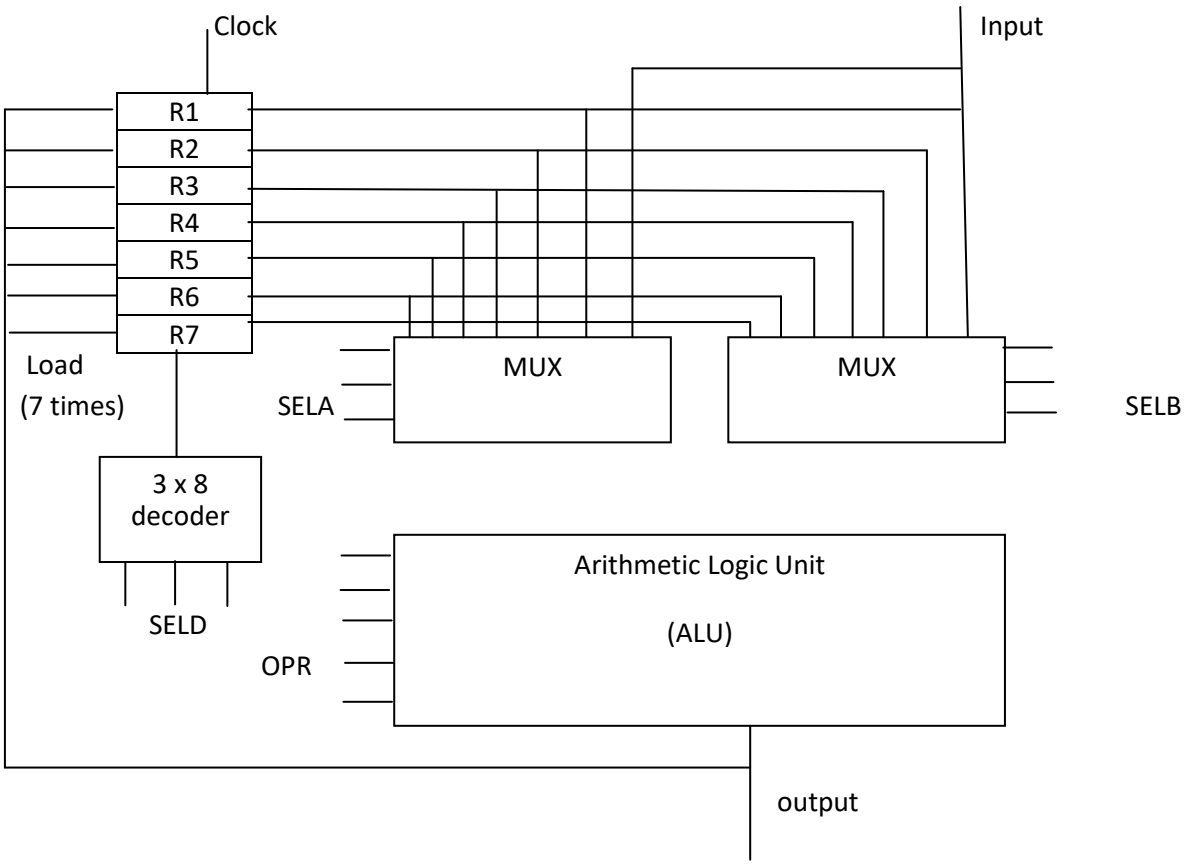
Arithmetic logic unit: this is used for the most of the logical processing, for example, for calculations or comparisons. The arithmetic operations like, +, -,*, and / are performed here. The logical operations are like, <,>,=,<=,>=,and <> are also performed here. In most of the arithmetical operations the result is in numerical form while in the case of logical operations the result can be True/false.

Control Unit: this unit controls the flow and manipulation of data and information. It also controls the flow of data from input devices to memory and from memory to output devices.

Output unit: the output unit consists of the output devices attached to the computer. These devices take machine coded output results from the processor and convert them into a form that can be understood by the user.

Computer architecture is sometimes defined as the computer structure and behaviour as seen by the programmer that uses machine language instructions. This includes the instruction formats, addressing modes, the instruction set, and the general organization of the CPU registers. From the designer's point of view, the computer instruction set provides the specifications for the design of the CPU. The design of a CPU is a task that in large part involves choosing the hardware for implementing the machine instructions. The user who programs the computer in machine or assembly language must be aware of the register set, the memory structure, the type of data supported by the instructions and the function that each instruction performs.

ii) Block diagram of Digital Computer



iii) General Register Organization

The registers communicate with each other not only for direct data transfers, but also while performing various micro-operations. Hence it is necessary to provide a common unit that can perform all the arithmetic, logic and shift microoperations in the processor..

In the diagram, it consists of a bus organization for seven CPU registers. The output of each register is connected to two multiplexers (MUX) to form the two buses A and B. The selection lines in each multiplexer select one register or the input data for the particular bus. The A and B buses form the inputs to a common arithmetic logic unit (ALU).

The operation selected in the ALU determines the arithmetic or logic microoperation that is to be performed. The result of the microoperation is available for output data and also goes into the inputs of all the registers. The register that receives the information from the output bus is selected by a decoder.

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 selection 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 R1

The four control selection variables are generated in the control unit and must be available at the beginning of a clock cycle. The data from the two source registers propagate through the gates in the multiplexers and the ALU, to the output bus, and into the inputs of the destination register, all during the clock cycle interval. Then, when the next clock transition occurs, the binary information from the output bus is transferred into R1. To achieve a fast response time, the ALU is constructed with high speed circuits.

Control word: there are 14 binary selection inputs in the unit, and their combined value specifies a control word. The 14 bit control word is defined in the following figure. It consists of four fields. There fields contain three bits each, and one field has five bits.

3	3	3	5
SELA	SELB	SELD	OPR

The three bit s of SELA select a source register for the A input of the ALU. The three bit s 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 OPR select one of the operations in the ALU. The 14 bits control word when applied to the selection inputs specify a particular microoperation.

The encoding of the register selections is specified in the following table. The 3 bit binary code listed in the first column of the table specifies the binary code for each of the three fields. When SELA or SELB is 000, the corresponding multiplexer selects the external input data. when SELD=000, no destination register is selected but the contents of the output bus are available in the external output.

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

In addition, the CPU must provide shift operations. The shifter may be placed in the input of the ALU to provide postshifting capability. The function table for this ALU is listed in table given below. The OPR field has five bits and each operation is designated with a symbolic name.

OPR SELECT	OPERATION	SYMBOLS
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

Examples of Micro operations:

A control word of 14 bits is needed to specify a microoperation in the CPU. The control word for a given microoperation can be derived from the selection variables, for example, the subtract microoperation given by the statement.

$$R1 \leftarrow R2 - R3$$

Specifies R2 for the A input of the ALU, R3 for the B input of the ALU, R1 for the destination register, and an ALU operation to subtract A-B. The binary control word for the subtract microoperation is 010 011 001 00101 is obtained as follows:

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

The control word for this Microoperation and a few others are listed in the following table:

Symbolic designation

Microoperation	**SELA**	**SELB**	**SELD**	**OPR**	**Control word**
R1 ←R2-R3	R2	R3	R1	SUB	010 011 001 00101
R4 ← R4 V R5	R4	R5	R4	OR	100 101 100 01010
R6 ← R6 +1	R6	--	R6	INCA	110 000 110 00001
R7 ← R1	R1	--	R7	TSFA	001 000 111 00000
Output ← R2	R2	--	None	TSFA	010 000 000 00000
Output ←input	input	--	None	TSFA	000 000 000 00000
R4 ← sh1 R4	R4	--	R4	SHLA	100 000 100 11000
R5 ← 0	R5	R5	R5	XOR	101 101 101 01100

The increment and transfer Microoperation do not use the B input of the ALU. For these cases, the B field is marked with a dash. We assign 000 to any unused field when formulating the binary control word. To place the content of a register into the output terminals we place the content of the register into the A input of the ALU, but none of the registers are selected to accept the data.

The most efficient way to generate control words with a large number of bits is to store them in a memory unit. A memory unit that stores control words is referred to as a control memory. This type of control is referred to as micro programmed control.

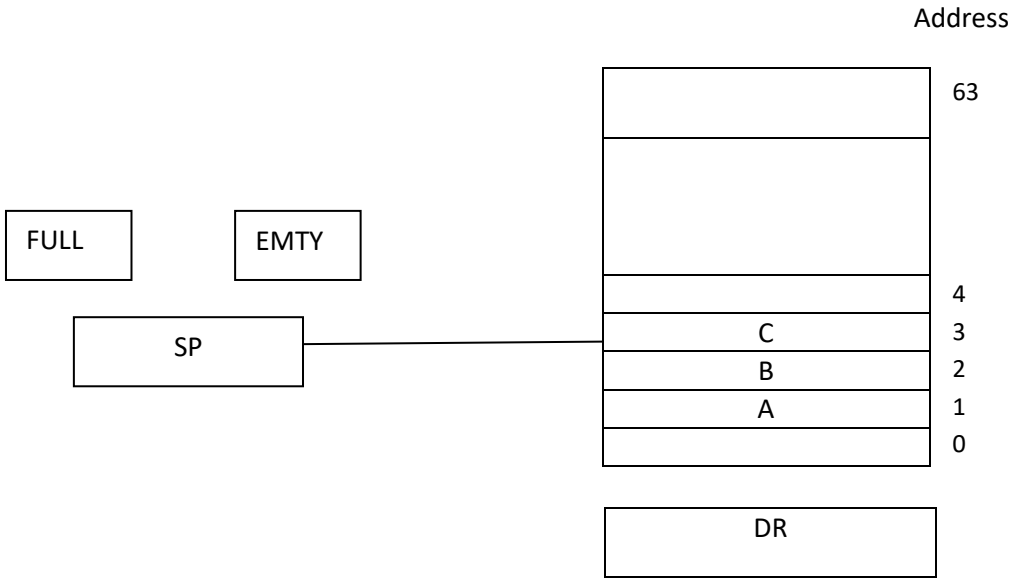
iv) Stack Organization

A stack is a storage device that stores information in such a manner that the item stored last in the first item retrieved(LIFO). The operation of a stack can be compared to a stack of trays. The last tray placed on top of the stack is the first to be taken off. The register that holds the address for the stack is called a stack pointer(SP). Because its value always points at the top item in the stack.

The two operations of a stack are the insertion and deletion of items. The operation of insertion is called push(as the result of pushing a new item on top. The operation of deletion is called pop(as the result of removing one item). However, nothing is pushed or popped in a computer stack. These operations are simulated by incrementing or decrementing the stack pointer register.

Register stack:

A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words or register. For example, consider the organization of a 64 word register stack. The stack pointer register SP contains a binary number whose value is equal to the address of the word that is currently on top of the stack.



--Block diagram of a 64 word stack--

Three items are placed in the stack: A, B and C. Item C is on top of the stack so that the content of SP is now 3. To remove the top item, the stack is popped by reading the memory word at address 3 and decrementing the content of SP. Item B is now on top of the stack since SP holds address 2.

To inset a new item, the stack is pushed by incrementing SP and writing a word in the next higher location of the stack. Note that item C has been read out but not physically removed. This does not matter because when the stack is pushed, a new item is written in its place. In a 64 word stack, the stack pointer contains 6 bits because 2^6=64. Since SP has only six bits, it cannot exceed a number greater than 63 (111111 in binary). When 63 is incremented by 1, the result is 0 since 111111 +1=1000000 in binary, but SP can accommodate only the six least significant bits. Similarly, when 000000 is decremented by 1, the result is 111111.

The one bit register FULL is set to 1 when the stack is full, and one bit register EMTY is set to 1 when the stack is empty of items. DR is the data register that holds the binary data to be written into or read out of the stack. Initially, SP is cleared to 0, EMTY is set to 1, and FULL is cleared to 0, so that SP points to the word at address 0 and the stack is marked empty and not full. If stack is not full (if FULL=0), a new item is inserted with a push operation. The push operation is implemented with the following sequence of Microoperations:

SP ←SP+1	increment stack pointer
M[SP] ←DR	write item on top of the stack
If(SP=0) then (FULL ←1)	check if stack is full

EMTY ←0 Mark the stack not empty

The stack pointer is incremented so that it points to the address of the next higher word. A memory write operation inserts the word from DR into the top of the stack. Note that SP holds the address of the top of the stack and that M[SP] denotes the memory word specified by the address presently available in SP. The first item stored in the stack is at address 1. The last item is stored at address 0, if SP reaches 0, the stack is full of items, so FULL is set to 1.

The condition is reached if the top item prior to the last push was in location 63 and, after incrementing SP, the last item is stored in location 0. Once an item is stored in location 0, there are no more empty registers in the stack. If an item is written in the stack, obviously the stack cannot be empty, so EMTY is cleared to 0. A new item is deleted from the stack if the stack is not empty (if EMTY=0). The pop operation consists of the following sequence of micro operations:

DR ← M[SP] Read item from the top of stack

SP ← SP-1 Decrement stack pointer

If (SP=0) then (EMTY ← 1) check if stack is empty

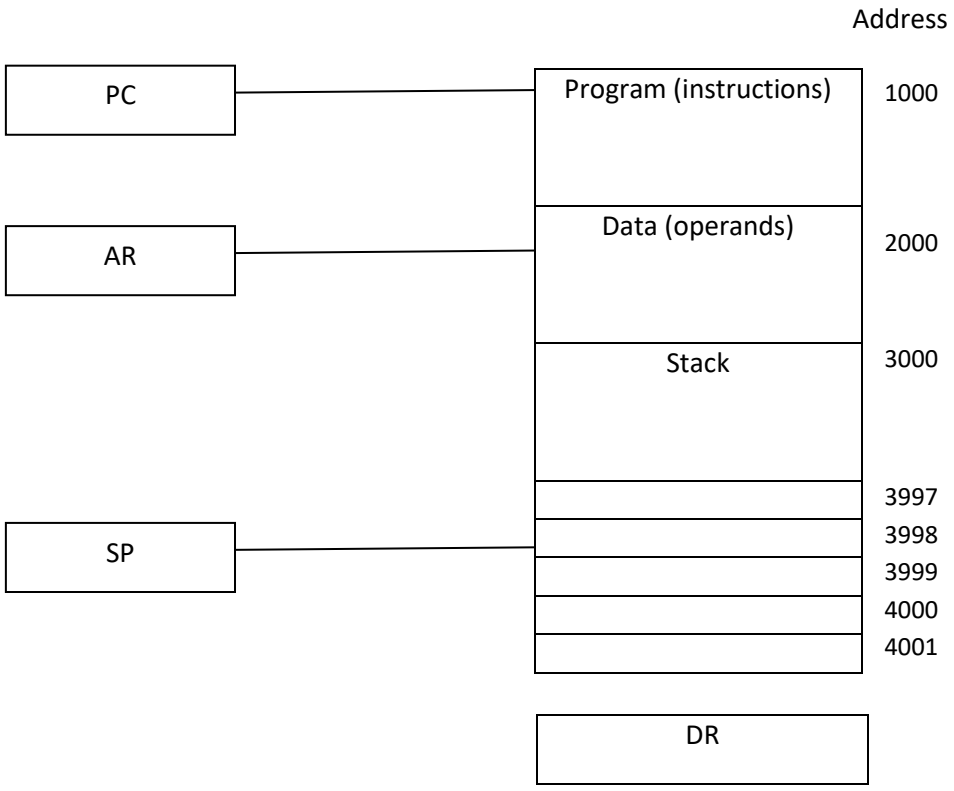
FULL ←0 Mark the stack not full

The top item is read from the stack into DR. The stack pointer is then decremented. If its value reaches zero, the stack is empty, so EMTY is set to 1. This condition is reached if the item read was in location 1. Once the item is read out, SP is decremented and reaches the value 0, which is the initial value of SP.

Memory Stack:

The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer. A portion of computer memory partitioned into three segments: program, data and stack.

The program counter PC points at the address of the next instruction in the program. The address register AR points at an array of data. the stack pointer SP points at the top of the stack. The three registers are connected to a common address bus, and either one can provide an address for memory. PC is used during the fetch phase to read an instruction. AR is used during the execute phase to read an operand. SP is used to push or pop items into or from the stack.



--Computer memory with Program, data, and stack segments--

The initial value of SP is 4001 and the stack grows with decreasing addresses. Thus the first item stored in the stack is at address 4000, the second item is stored at address 3999, and the last address that can be used for the stack is 3000. No provisions are available for stack limit checks.

A new item is inserted with the push operation as follows:
SP←SP-1
M[SP] ←DR
The stack pointer is decremented so that it points at the address of the next word. A memory write operation inserts the word from DR into the top of the stack.

A new item is deleted with a pop operation as follows:
DR ←M[SP]
SP←SP+1
The top item is read from the stack into DR. The stack pointer is then incremented to point at the next item in the stack. Most computers do not provide hardware to check for stack overflow. The stack limits can be checked by using two processor registers: one to hold the upper limit (3000), and the other to hold the lower limit (4001). After a push operation, SP is compared with the upper limit register and after a pop operation, SP is compared with the lower limit register.

The two microoperations needed for either the push or pop are (1) an access to memory through SP, and (2) updating SP. The advantage of a memory stack is that the CPU can refer to it without having to specify an address, since the address is always available and automatically updated in the stack pointer.

Reverse Polish Notation:

A stack organization is very effective for evaluating arithmetic expressions. The common mathematical method of writing arithmetic expressions imposes difficulties when evaluated by a computer. The common arithmetic expressions are written in infix notation, with each operator written between the operands.

Consider the simple arithmetic expression.
A*B + C*D
To evaluate this arithmetic expression it is necessary to compute the product A*B, store the product while computing C*D, and then sum the two products.

The polish mathematician Lukasiewicz showed that arithmetic expressions can be represented in prefix notation. The postfix notation, referred to as Reverse Polish Notation (RPN), places the operator after the operands.

A+B	infix notation
+AB	Prefix or Polish notation
AB+	Postfix or Reverse Polish Notation

The reverse polish notation is in a form suitable for stack manipulation. The expression
A*B+C*D is written in reverse polish notation as
AB*CD*+
Scan the expression from left to right. When an operator is reached, perform the operation with the two operands found on the left side of the operator. Continue to scan the expression and repeat the procedure for every operator encountered until there are no more operators.

We perform the operation A*B and replace A,B and * by the product to obtain

(A*B)CD*+

Where (A*B) is a single quantity obtained from the product. The next operator is a * and its previous two operands are C and D, so we perform C *D and obtain an expression with two operands and one operator:

(A*B)(C*D)+ the next operator is + and the two operands to be added are the two products, so we add the two quantities to obtain result.

This hierarchy dictates that we first perform all arithmetic inside inner parentheses, then inside outer parentheses, and do multiplication and division operation before addition and subtraction operations. (ie., BODMAS rule → Bracket of division, multiplication, addition and subtraction)

Consider the expression: (A+B)*[C*(D+E)+F], The converted expression is AB+DE+C*F+*. Proceeding from left to right, we first add A and B, then add D and E. At this point we are left with (A+B)(D+E)C*F+*

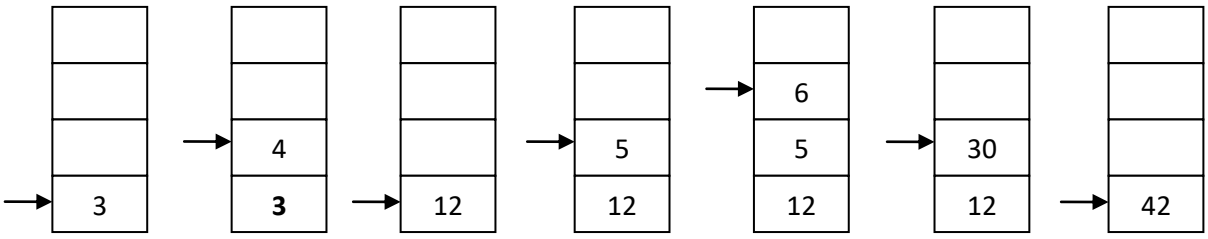
Evaluation of Arithmetic Expressions:

The following microoperations are executed with the stack when an operation is entered in a calculator or issued by the control in a computer.

1. The two topmost operands in stack are used for the operation, and
2. The stack is popped and the result of the operation replaces the lower operand

The following numerical example may clarify this procedure. Consider the arithmetic expression. (3*4) + (5*6)

In reverse polish notation, this is expressed as 34*56*+



--Stack operations --

Each box represents one stack operations and the arrow always points to the top of the stack. Scanning the expression from left to right, we encounter 2 operands. First the number 3 is pushed into the stack, then the number 4. The next symbol is the multiplication operator *. This causes a multiplication of the two topmost items in the stack. The stack is then popped and the product on top of the stack, replacing the two original operands. Next we encounter the two operands 5 and 6 so they are pushed into the stack. The stack operation that results from the next * replaces these two numbers by their product. The last operation causes an arithmetic addition of the two topmost numbers in the stack to produce the final result of 42.

Most compliers, irrespective of their CPU organization, convert all arithmetic expressions into Polish notation anyway because this is the most efficient method for translating arithmetic expressions into machine language instructions.

v) RISC (Reduced Instruction Set Computer)

An important aspect of computer architecture is the design of the instruction set for the processor. The instruction set chosen for a particular computer determines the way that machine language programs are constructed. Early computers had small and simple instruction sets, forced mainly by the need to minimize the hardware used to implement them. As digital hardware became cheaper with the advent of integrated circuits, computer instructions tended to increase both in number and complexity.

Many computers have instruction sets that include more than 100 and sometimes even more than 200 instructions. These computers also employ a variety of data types and a large number of addressing modes.

The trend into computer hardware complexity was influenced by various factors, such as upgrading existing models to provide more customer applications, adding instructions that facilitate the translation from high level language into machine language programs.

A computer with a large number of instructions is classified as a Complex Instruction Set Computer(CISC). In the early 1980s, a number of computer designers recommended that computers use fewer instructions with simple constructs to they can be executed mush faster within the CPU without having to use memory as often. This type of computer is classified as a reduced instruction set computer (RISC).

RISC Characteristics:

The concept of RISC architecture involves an attempt to reduce execution time by simplifying the instruction set of the computer.

The major characteristics of a RISC processor are:

1. Relatively few instructions
2. Relatively few addressing modes
3. Memory access limited to load and store instructions
4. All operations done within the registers of the CPU
5. Fixed length, easily decoded instruction format
6. Single cycle instruction execution
7. Hardwired rather than microprogrammed control.

The small set of instructions of a typical RISC processor consists mostly of register to register operations, with only simple load and store operations for memory access. Thus each operand is brought into a processor register with a load instruction. All computations are done among the data stored in processor registers.

Results are transferred to memory by means of store instructions. This architectural feature simplifies the instruction set and encourages the optimization of regular manipulation. The use of only a few addressing modes results from the fact that almost all instructions have simple register addressing. Other addressing modes may be included, such as immediate operands and relative mode. By using a relatively simple instruction format, the instruction length can be fixed and aligned on word boundaries. Thus the operation code and register fields of the instruction code can be accessed simultaneously by the control. By simplifying the instruction and their format, it is possible to simplify the control logic. For faster operations, a hardwired control is preferable over a microprogrammed control.

A characteristic of RISC processors is their ability to execute one instruction per clock cycle. This is done by overlapping the fetch, decode, and execute phases of two or three instructions by using a procedure referred to as pipelining. A load or store instruction may require two clock cycles because access to memory takes more time than register operations. Efficient pipelining, as well as a few other characteristics, are sometimes attributed to RISC.

Other characteristics attributed to RISC architecture are:

1. A relatively large number of registers in the processor unit.
2. Use of overlapped register windows to speed up procedure call and return.
3. Efficient instruction pipeline.
4. Complier support for efficient translation of high level language programs into machine language programs.

A large number of registers is useful for storing intermediate results and for optimizing operand references. The advantage of register storage as opposed to memory storage is that registers can transfer information to other registers much faster than the transfer of information to and from memory.

vi) CISC(Complex Instruction Set Computer)

The design of an instruction set for a computer must take into consideration not only machine language constructs, but also the requirements imposed on the use of high level programming languages. The translation from high level to machine language programs is done by means of a compiler program.

The task of a compiler is to generate a sequence of machine instructions for each high level language statement. The task is simplified if there are machine instructions that implement the statements directly.

The essential goal of a CISC architecture is to attempt to provide a single machine instruction for each statement that is written in a high level language. One reason for the trend to provide a complex instruction set is the desire to simplify the compilation and improve the overall computer performance. Examples of CISC architectures are the Digital equipment corporation VAX computer and the IBM 370 computer.

Another characteristic of CISC architecture is the incorporation of variable length instruction formats. However, as more instructions and addressing modes are incorporated into a computer, the more hardware logic is needed to implement and support them, and this may cause the computations to slow down.

In summary, the major characteristic of CISC architecture are:

1. A large number of instructions – typically from 100 to 250 instructions.
2. Some instructions that perform specialized tasks and are used infrequently
3. A large variety of addressing modes – typically from 5 to 20 different modes.
4. Variable length instruction formats
5. Instructions that manipulate operands in memory.

vii) Control Unit

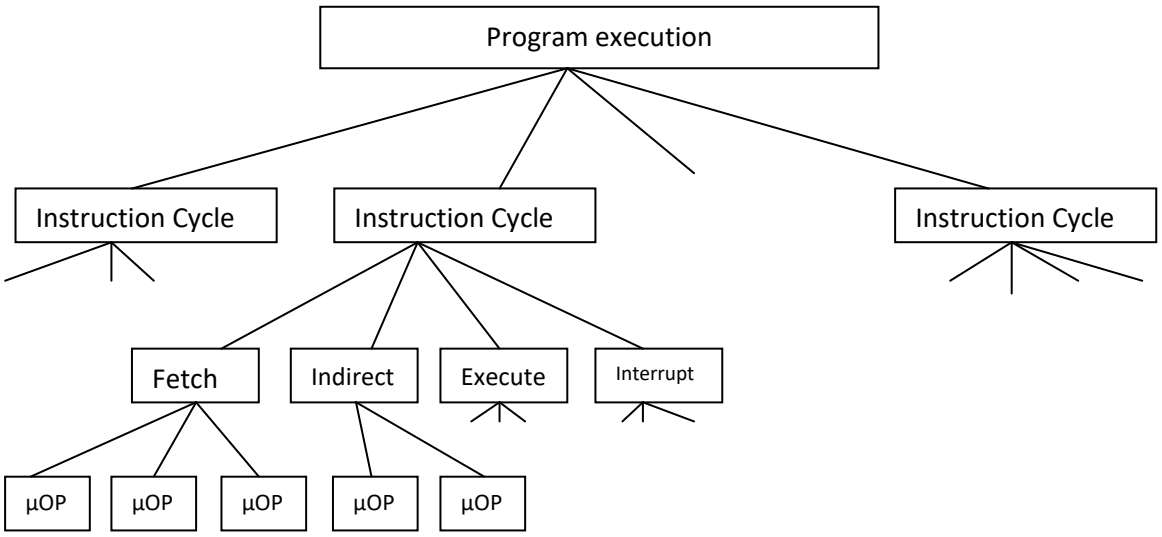
The execution of an instruction involves the execution of a sequence of substeps, generally called cycles. For example, an execution may consist of fetch, indirect, execute, and interrupt cycles. Each cycle is in turn made up of a sequence of more fundamental operations, called micro operations. A single micro operation generally involves a transfer between registers, a transfer between a register and an external bus or a simple ALU operation.

The control unit of a processor performs two tasks: (1) It causes the processor to execute micro operations in the proper sequence, determined by the program being executed, and (2) it generates the control signals that cause each micro operation to be executed.

The control signals generated by the control unit cause the opening and closing of logic gates, resulting in the transfer of data to and from registers and the operation of the ALU.

One technique for implementing a control unit is referred to as hardwired implementation, in which the control unit is a combinatorial circuit. Its input logic signals, governed by the current machine instruction, are transferred into a set of output control signals.

Micro operations are the functional, or atomic, operations of a processor.

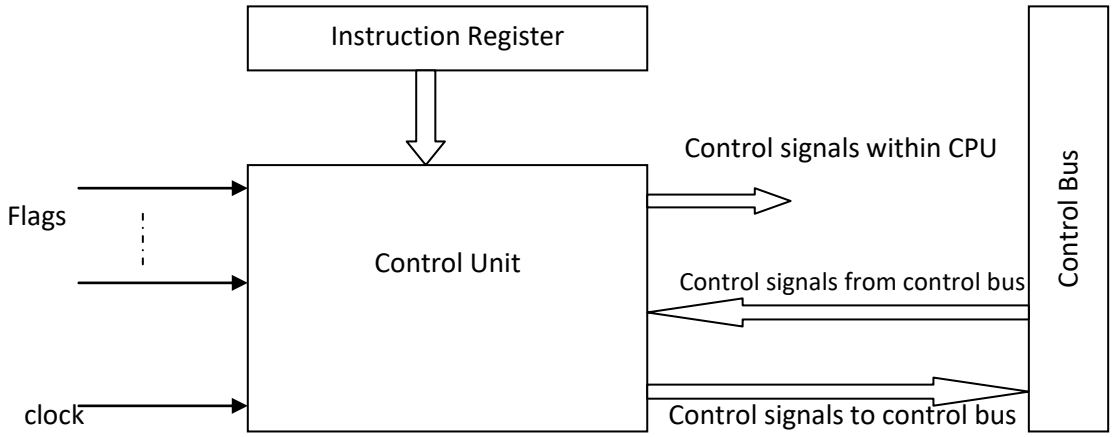


--Constituent Elements of a Program Execution--

We can define the functional requirements for the control unit: those functions that the control unit must perform. A definition of these functional requirements is the basis for the design and implementation of the control unit.

The following three step process leads to a characterization of the control unit:

1. Define the basic elements of the processor
2. Describe the micro operations that the processor performs
3. Determine the functions that the control unit must perform to cause the micro operations to be performed.



--Block Diagram of the control unit--

For the control unit to perform its function, it must have inputs that allow it to determine the state of the system and outputs that allow it to control the behaviour of the system.

The inputs are as follows:

Clock: this is how the control unit "keeps time". The control unit causes one micro operation (or a set of simultaneous micro operations) to be performed for each clock pulse. This is sometimes referred to as the processor cycle time, or the clock cycle time.
Instruction register: The opcode of the current instruction is used to determine which micro operations to perform during the execute cycle.
Flags: These are needed by the control unit to determine the status of the processor and the outcome of previous ALU operations. For example, for the increment-and-skip-if-zero (ISZ) instruction, the control unit will increment the PC if the zero flag is set.
Control signals from control bus: The control bus portion of the system bus provides signals to the control unit, such as interrupt signals and acknowledgements.

The outputs are

Control signals within the processor: These are two types: those that cause data to be moved from one register to another, and those that activate specific ALU functions.
Control signals to control bus: These are also of two types: control signals to memory, and control signals to the I/O modules.

A wide variety of techniques have been used for control unit implementation. Most of these fall into one of two categories:

- Hardwired implementation
- Microprogrammed implementation

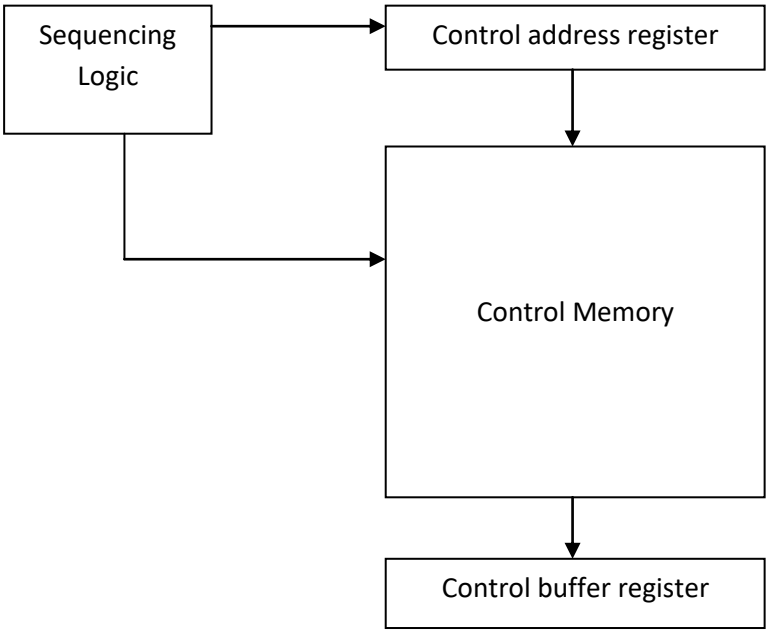
Hardwired implementation:

In a hardwired implementation, the control unit is essentially a combinational circuit. Its input logic signals are transformed into a set of output logic signals, which are the control signals. The key inputs are the instruction register, the clock, flags, and control bus signals. In the case of the flags and control bus signals, each individual bit typically has some meaning. The other two inputs, however, are not directly useful to the control unit. The internal logic of the control unit that produces output control signals as a function of its input signals.

Microprogrammed implementation:

An alternative to a hardwired control unit is a microprogrammed control unit, in which the logic of the control unit is specified by a microprogram. A microprogram consists of a sequence of instructions in a microprogramming language. These are very simple instructions that specify micro operations.

A microprogrammed control unit is a relatively simple logic circuit that is capable of (1) sequencing through microinstruction and (2) generating control signals to execute each microinstruction.
As in a hardwired control unit, the control signals generated by a microinstruction are used to cause register transfers and ALU operations.

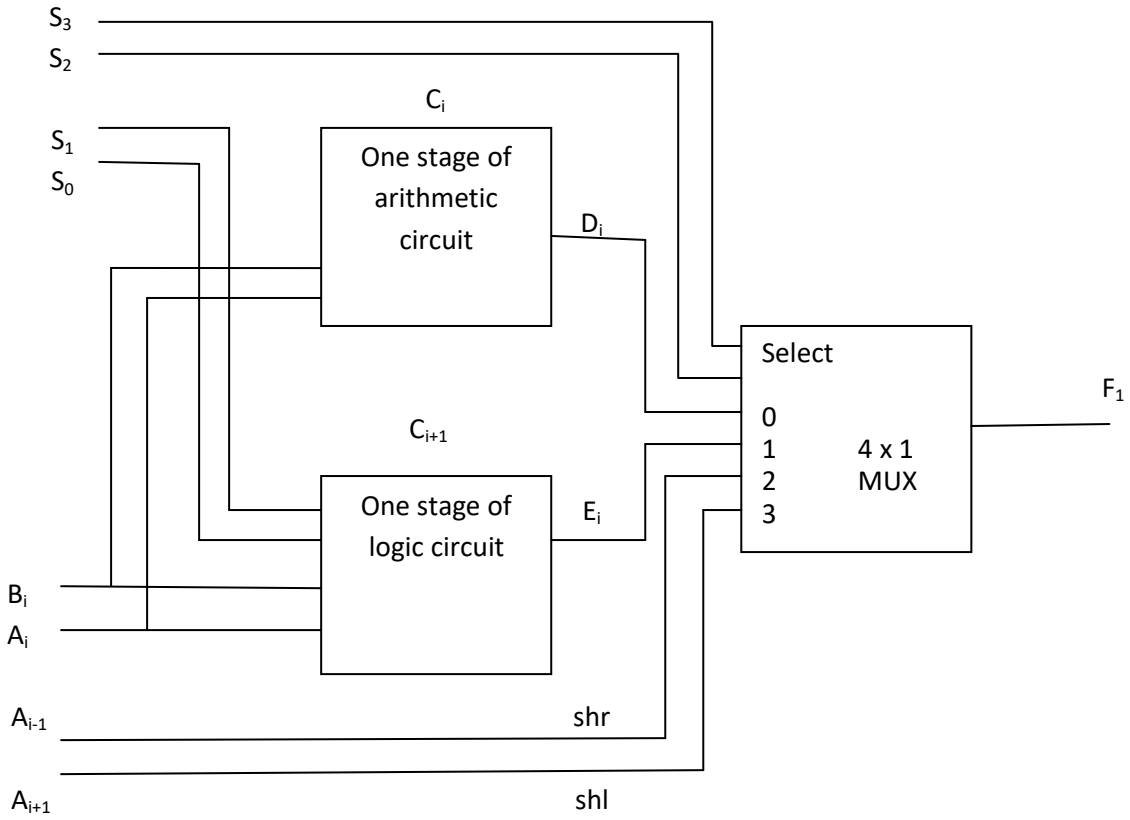


--Control Unit Micro architecture--

The set of microinstructions is stored in the control memory. The control address register contains the address of the next microinstruction to be read. When a microinstruction is read fromt he control memory, it is transferred to a control buffer register. Thus, reading a microinstruction fromt he control memory is the same as executing that microinstruction. The third element is a sequencing unit that loads the control address register and issues a read command.
The two basic task performed by a microprgrammed control unit are
Microinstruction sequencing: Get the next microinstruction from the control memory
Microinstruction execution: Generate the control signals needed to execute the microinstruction.

viii) ALU design

A possible choice for shift unit would be a bidirectional shift register with parallel load. Information can be transferred to the register in parallel and then shifted to the right or left. In this type of configuration, a clock pulse is needed for loading the data into the register and another pulse is needed to initiate the shift.
The content of a register that has to be shifted is first placed onto a common bus whose output is connected to the combinational shifter and the shifted number is then loaded back into the register. This requires only one clock pulse for loading the shifted value into the register.
Arithmetic logic shift unit:
Instead of having individual registers performing the microoperations directly, computer systems employ a number of storage registers connected to a common operational unit called an arithmetic logic unit(ALU). To perform a Microoperation, the contents of specified registers are placed in the inputs of the common ALU. The ALU performs an operation and the result of the operation is then transferred to a destination register. The ALU is a combinational circuit so that the entire register transfer operation from the source registers through the ALU and into the destination register can be performed during one clock pulse period.
The shift microoperations are often performed in a separate unit, but sometimes the shift unit is made part of the overall ALU.
One stage of an arithmetic logic shift unit is shown in following figure. The subscript i designates a typical stage. Inputs A_i and B_i are applied to both the arithmetic and logic units.



--One stage of arithmetic logic shift unit--

A particular Microoperation is selected with inputs S_1 and S_0. A 4 x 1 multiplexer at the output chooses between an arithmetic output in E_i and a logic output in H_i.

The data in the multiplexer are selected with inputs S_3 and S_2. The other two data inputs to the multiplexer receive inputs A_{i-1} for the shift right operation and A_{i+1} for the shift left operation.

The output carry C_{i+1} of a given arithmetic stage must be connected to the input carry C_i of the next stage in sequence. The input carry to the first stage is the input carry C_{in}, which provides a selection variable for the arithmetic operations.

The circuit whose one stage is specified in figure provides eight arithmetic operations, four logic operations and two shift operations.

Function table for Arithmetic Logic Shift Unit

		Operation select				
S_3	***S_2***	***S_1***	***S_0***	***C_{in}***	***Operation***	***Function***
0	0	0	0	0	F=A	Transfer A
0	0	0	0	1	F=A+1	Increment A
0	0	0	1	0	F=A+B	Addition
0	0	0	1	1	F=A+B+1	Add with Carry
0	0	1	0	0	F=A-B	Subtract with borrow
0	0	1	0	0	F=A-B-1	subtraction
0	0	1	1	0	F=A-1	Decrement A
0	0	1	1	1	F=A	Transfer A
0	1	0	0	x	F=A∩B	AND
0	1	0	1	x	F=A∪B	OR
0	1	1	0	x	F=A ⊕B	XOR
0	1	1	1	x	F=À	complement A
1	0	x	x	x	F=shr A	shift right A into F
1	1	x	x	x	F=shl A	shift left A into F

Each operation is selected with the five variables S_3,S_2,S_1,S_0 and C_{in}. The input carry C_{in} is used for selecting an arithmetic operation only. The above table lists the 14 operations of the ALU. The first eight are arithmetic operations and are selected with S_3S_2 =00. The next four are logic operations and are selected with S_3S_2=01. The input carry has no effect during the logic operations and is marked with don't care X's. The last two operations are shift operations and are selected with S_3S_2=10 and 11. The other three selection inputs have no effect on the shift.

ix) Pipeline Processing

Pipelining is a technique of decomposing a sequential process into suboperations, with each subprocess being executed in a special dedicated segment that operates concurrently with all other segments.

Each segment performs partial processing dictated by the way the task is partitioned. The result obtained from the computation in each segment is transferred to the next segment in the pipeline. The final result is obtained after the data have passed through all segments.

The name "pipeline" implies a flow of information analogous to an industrial assembly line. It is characteristic of pipelines that several computations can be in progress in distinct segments at the same time. The registers provide isolation between each segment so that each can operate on distinct data simultaneously.

Perhaps the simplest way of viewing the pipeline structure is to imagine that each segment consists of an input register followed by a combinational circuit. The register holds the data and the combinational circuit performs the suboperation in the particular segment. The output of the combinational circuit in a given segment is applied to the input register of the next segment. A clock is applied to all registers after enough time has elapsed to perform all segment activity.

In this way the information flows through the pipeline one step at a time. The pipeline organization will be demonstrated by means of a simple example. Suppose that we want to perform the combined multiply and add operations with a stream of numbers.

$A_i * B_i + C_i$ for i=1,2,3,.....7

Each suboperation is to be implemented in a segment within a pipeline. Each segment has one or two registers and a combinational circuit. R1 through R5 are registers that receive new data with every clock pulse. The multiplier and adder are combinational circuits. The suboperations performed in each segment of pipeline are as follows:

$R1 \leftarrow A_i$, $R2 \leftarrow B_i$ input A_i and B_i

$R3 \leftarrow R1 * R2$, $R4 \leftarrow C_i$ multiply and input C_i

$R5 \leftarrow R3 + R4$ Add C_i to product

The five registers are loaded with new data every clock pulse. The effect of each clock is shown in the table.

Clock pulse number	Segment1		Segment2		Segment 3
	R1	R2	R3	R4	R5
1	A_1	B_1	-	-	-
2	A_2	B_2	A_1*B_1	C_1	-
3	A_3	B_3	A_2*B_2	C_2	$A_1*B_1+C_1$
4	A_4	B_4	A_3*B_3	C_3	$A_2*B_2+C_2$
5	A_5	B_5	A_4*B_4	C_4	$A_3*B_3+C_3$
6	A_6	B_6	A_5*B_5	C_5	$A_4*B_4+C_4$
7	A_7	B_7	A_6*B_6	C_6	$A_5*B_5+C_5$
8	-	-	A_7*B_7	C_7	$A_6*B_6+C_6$
9	-	-	-	-	$A_7*B_7+C_7$

The first clock pulse transfers A1 and B1 into R1 and R2. The second clock pulse transfers the product of R1 and R2 into R3 and C1 into R4. The same clock pulse transfers A2 and B2 into R1 and R2. The third clock pulse operates on all three segments simultaneously.

It places A3 and B3 into R1 and R2, transfers the product of R1 and R2 into R3, transfers C2 into R4 and places the sum of R3 and R4 into R5. It takes three clock pulses to fill up the pipe and retrieve the first output from R5. From there on, each clock produces a new output and moves the data one step down the pipeline. This happens as long as new input data flow into the system. When no more input data are available, the clock must continue until the last output emerges out the pipeline.