Microprogrammed Control

Hardwired Control Unit:

When the control signals are generated by hardware using conventional logic design techniques, the control unit is said to be hardwired.

Micro programmed control unit:

A control unit whose binary control variables are stored in memory is called a micro programmed control unit.

Dynamic microprogramming:

A more advanced development known as *dynamic* microprogramming permits a microprogram to be loaded initially from an auxiliary memory such as a magnetic disk. Control units that use dynamic microprogramming employ a writable control memory. This type of memory can be used for writing.

Control Memory:

Control Memory is the storage in the microprogrammed control unit to store the microprogram.

Writeable Control Memory:

Control Storage whose contents can be modified, allow the change in microprogram and Instruction set can be changed or modified is referred as Writeable Control Memory.

Control Word:

The control variables at any given time can be represented by a control word string of 1 's and 0's called a control word.

Microoperation, Microinstruction, Micro program, Microcode.

Microoperations:

In computer central processing units, micro-operations (also known as a micro-ops or μops) are detailed low-level instructions used in some designs to implement complex machine instructions (sometimes termed macro-instructions in this context).

Micro instruction:

A symbolic microprogram can be translated into its binary equivalent by means of an assembler.

Each line of the assembly language microprogram defines a symbolic microinstruction.

Each symbolic microinstruction is divided into five fields: label, microoperations, CD, BR, and AD.

Micro program:

A sequence of microinstructions constitutes a microprogram.

Since alterations of the microprogram are not needed once the control unit is in operation, the control memory can be a read-only memory (ROM).

ROM words are made permanent during the hardware production of the unit.

The use of a micro program involves placing all control variables in words of ROM for use by the control unit through successive read operations.

The content of the word in ROM at a given address specifies a microinstruction.

Microcode:

Microinstructions can be saved by employing subroutines that use common sections of microcode.

For example, the sequence of micro operations needed to generate the effective address of the operand for an instruction is common to all memory reference instructions.

This sequence could be a subroutine that is called from within many other routines to execute the effective address computation.

Organization of micro programmed control unit

The general configuration of a micro-programmed control unit is demonstrated in the block diagram of Figure 1.

The control memory is assumed to be a ROM, within which all control information is permanently stored.

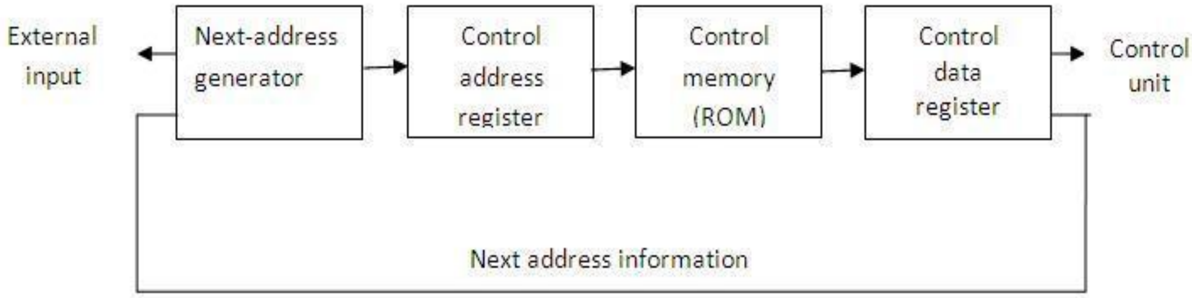


figure 1: Micro-programmed control organization

The control memory address register specifies the address of the microinstruction, and the control data register holds the microinstruction read from memory.

The microinstruction contains a control word that specifies one or more micro operations for the data processor. Once these operations are executed, the control must determine the next address.

The location of the next microinstruction may be the one next in sequence, or it may be located somewhere else in the control memory.

While the microoperations are being executed, the next address is computed in the next address generator circuit and then transferred into the control address register to read the next microinstruction.

Thus a microinstruction contains bits for initiating microoperations in the data processor part and bits that determine the address sequence for the control memory.

The next address generator is sometimes called a ***micro-program sequencer,*** as it determines the address sequence that is read from control memory.

Typical functions of a micro-program sequencer are incrementing the control address register by one, loading into the control address register an address from control memory, transferring an external address, or loading an initial address to start the control operations.

The control data register holds the present microinstruction while the next address is computed and read from memory.

The data register is sometimes called a ***pipeline register***.

It allows the execution of the microoperations specified by the control word simultaneously with the generation of the next microinstruction.

This configuration requires a two-phase clock, with one clock applied to the address register and the other to the data register.

The main advantage of the micro programmed control is the fact that once the hardware configuration is established; there should be no need for further hardware or wiring changes.

If we want to establish a different control sequence for the system, all we need to do is specify a different set of microinstructions for control memory.

Address Sequencing

Microinstructions are stored in control memory in groups, with each group specifying a ***routine***.

To appreciate the address sequencing in a micro-program control unit, let us specify the steps that the control must undergo during the execution of a single computer instruction.

Step-1:

An initial address is loaded into the control address register when power is turned on in the computer.

This address is usually the address of the first microinstruction that activates the instruction fetch routine.

The fetch routine may be sequenced by incrementing the control address register through the rest of its microinstructions.

At the end of the fetch routine, the instruction is in the instruction register of the computer.

Step-2:

The control memory next must go through the routine that determines the effective address of the operand.

A machine instruction may have bits that specify various addressing modes, such as indirect address and index registers.

The effective address computation routine in control memory can be reached through a branch microinstruction, which is conditioned on the status of the mode bits of the instruction.

When the effective address computation routine is completed, the address of the operand is available in the memory address register.

Step-3:

The next step is to generate the microoperations that execute the instruction fetched from memory.

The microoperation steps to be generated in processor registers depend on the operation code part of the instruction.

Each instruction has its own micro-program routine stored in a given location of control memory.

The transformation from the instruction code bits to an address in control memory where the routine is located is referred to as a ***mapping*** process.

A mapping procedure is a rule that transforms the instruction code into a control memory address.

Step-4:

Once the required routine is reached, the microinstructions that execute the instruction may be sequenced by incrementing the control address register.

Micro-programs that employ subroutines will require an external register for storing the return address.

Return addresses cannot be stored in ROM because the unit has no writing capability.

When the execution of the instruction is completed, control must return to the fetch routine.

This is accomplished by executing an unconditional branch microinstruction to the first address of the fetch routine.

In summary, the address sequencing capabilities required in a control memory are:

1. Incrementing of the control address register.
2. Unconditional branch or conditional branch, depending on status bit conditions.
3. A mapping process from the bits of the instruction to an address for control memory.
4. A facility for subroutine call and return.

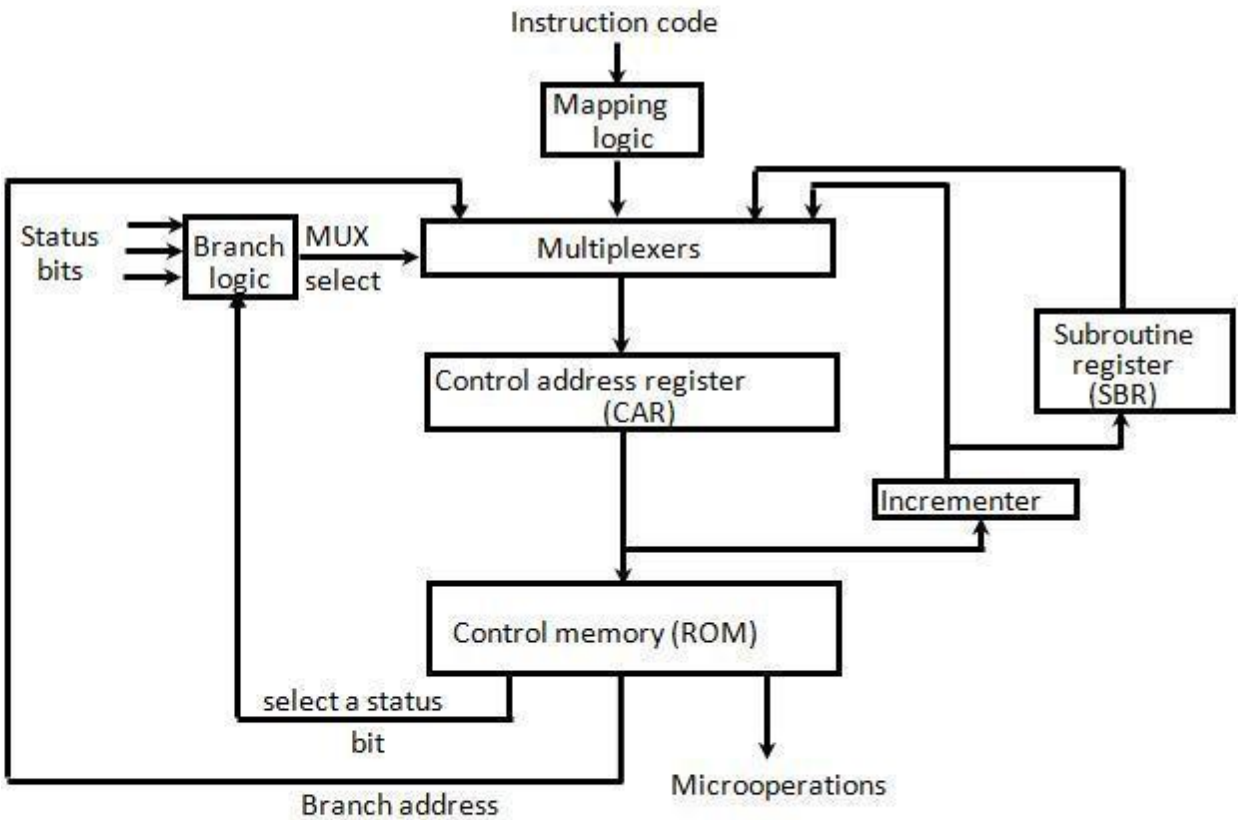


Figure 2: Selection of address for control memory

Above figure 4.2 shows a block diagram of a control memory and the associated hardware needed for selecting the next microinstruction address.

The microinstruction in control memory contains a set of bits to initiate microoperations in computer registers and other bits to specify the method by which the next address is obtained.

The diagram shows four different paths from which the control address register (CAR) receives the address.

The incrementer increments the content of the control address register by one, to select the next microinstruction in sequence.

Branching is achieved by specifying the branch address in one of the fields of the microinstruction.

Conditional branching is obtained by using part of the microinstruction to select a specific status bit in order to determine its condition.

An external address is transferred into control memory via a mapping logic circuit.

The return address for a subroutine is stored in a special register whose value is then used when the micro-program wishes to return from the subroutine.

The branch logic of figure 2 provides decision-making capabilities in the control unit.

The status conditions are special bits in the system that provide parameter information such as the carry-out of an adder, the sign bit of a number, the mode bits of an instruction, and input or output status conditions.

The status bits, together with the field in the microinstruction that specifies a branch address, control the conditional branch decisions generated in the branch logic.

A 1 output in the multiplexer generates a control signal to transfer the branch address from the microinstruction into the control address register.

A 0 output in the multiplexer causes the address register to be incremented.

Mapping of an Instruction

A special type of branch exists when a microinstruction specifies a branch to the first word in control memory where a microprogram routine for an instruction is located.

The status bits for this type of branch are the bits in the operation code part of the instruction.

For example, a computer with a simple instruction format as shown in figure 4.3 has an operation code of four bits which can specify up to 16 distinct instructions.

Assume further that the control memory has 128 words, requiring an address of seven bits.

One simple mapping process that converts the 4-bit operation code to a 7-bit address for control memory is shown in figure 3.

This mapping consists of placing a 0 in the most significant bit of the address, transferring the four operation code bits, and clearing the two least significant bits of the control address register.

This provides for each computer instruction a microprogram routine with a capacity of four microinstructions.

If the routine needs more than four microinstructions, it can use addresses 1000000 through 1111111. If it uses fewer than four microinstructions, the unused memory locations would be available for other routines.

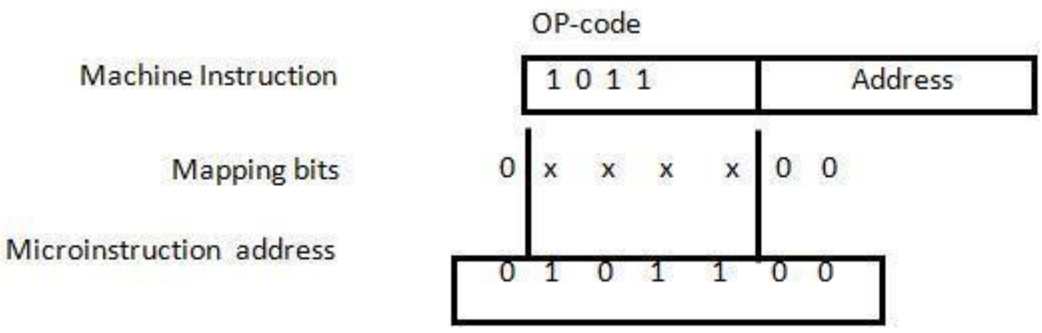


Figure 3: Mapping from instruction code to microinstruction address

One can extend this concept to a more general mapping rule by using a ROM to specify the mapping function.

The contents of the mapping ROM give the bits for the control address register.

In this way the microprogram routine that executes the instruction can be placed in any desired location in control memory.

The mapping concept provides flexibility for adding instructions for control memory as the need arises.

Computer Hardware Configuration

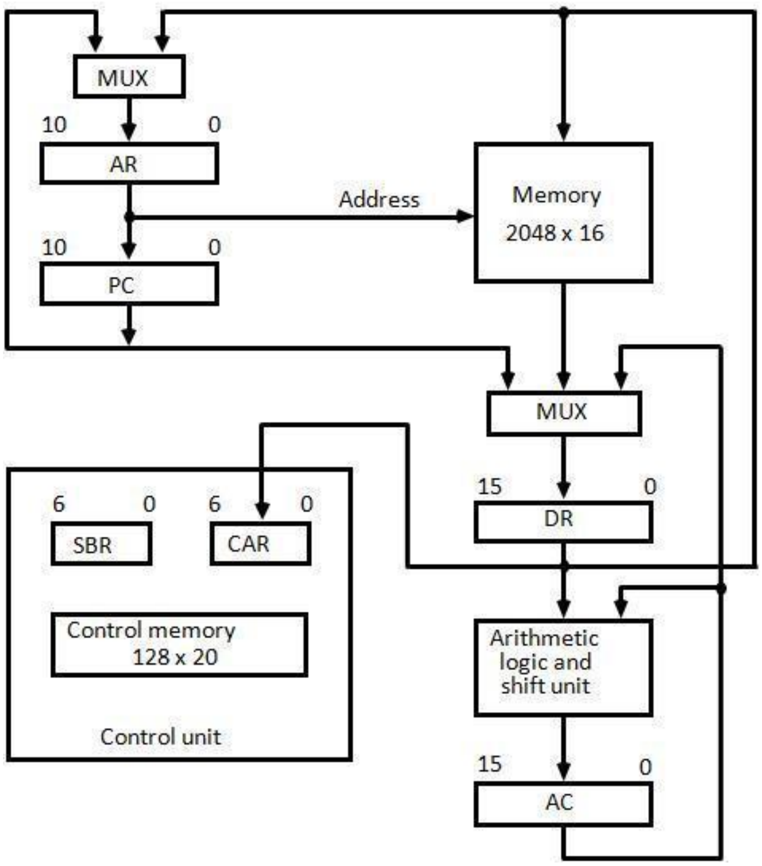


Figure 4: Computer hardware configuration

The block diagram of the computer is shown in Figure 4. It consists of

1. Two memory units:

   Main memory -> for storing instructions and data,
   and Control memory -> for storing the microprogram

Six Registers:

Processor unit register: AC(accumulator),PC(Program Counter), AR(Address Register), DR(Data Register)

Control unit register: CAR (Control Address Register), SBR(Subroutine Register)

2. Multiplexers:

   The transfer of information among the registers in the processor is done through multiplexers rather than a common bus.

3. ALU:

   The arithmetic, logic, and shift unit performs microoperations with data from AC and DR and places the result in AC.

   DR can receive information from AC, PC, or memory.
   AR can receive information from PC or DR.

PC can receive information only from AR.

Input data written to memory come from DR, and data read from memory can go only to DR.

Microinstruction Format

The microinstruction format for the control memory is shown in figure 4.5. The 20 bits of the microinstruction are divided into four functional parts as follows:

1. The three fields F1, F2, and F3 specify microoperations for the computer.

   The microoperations are subdivided into three fields of three bits each. The three bits in each field are encoded to specify seven distinct microoperations. This gives a total of 21 microoperations.

2. The CD field selects status bit conditions.
3. The BR field specifies the type of branch to be used.
4. The AD field contains a branch address. The address field is seven bits wide, since the control memory has $128 = 2^7$ words.

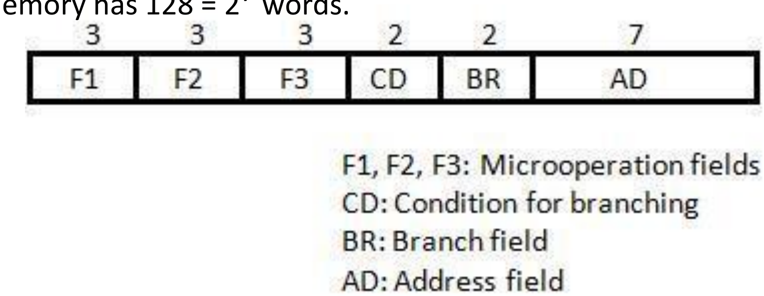


Figure 5: Microinstruction Format

As an example, a microinstruction can specify two simultaneous microoperations from F2 and F3 and none from F1.

DR   M[AR] with F2 = 100

PC   PC + 1 with F3 = 101

The nine bits of the microoperation fields will then be 000 100 101.

The CD (condition) field consists of two bits which are encoded to specify four status bit conditions as listed in Table 1.

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	DR(15)	I	Indirect address bit
10	AC(15)	S	Sign bit of AC
11	AC = 0	Z	Zero value in AC

Table 1: Condition Field

The BR (branch) field consists of two bits. It is used, in conjunction with the address field AD, to choose the address of the next microinstruction shown in Table 2.

BR	Symbol	Function
00	JMP	CAR ← AD if condition = 1
		CAR ← CAR + 1 if condition = 0
01	CALL	CAR ← AD, SBR ← CAR + 1 if condition = 1
		CAR ← CAR + 1 if condition = 0
10	RET	CAR ← SBR (Return from subroutine)
11	MAP	CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 0

Table 2: Branch Field

Symbolic Microinstruction.

Each line of the assembly language microprogram defines a symbolic microinstruction.

Each symbolic microinstruction is divided into five fields: label, microoperations, CD, BR, and AD. The fields specify the following Table 3.

1.	Label	The label field may be empty or it may specify a symbolic address. A label is terminated with a colon (:).
2.	Microoperations	It consists of one, two, or three symbols, separated by commas, from those defined in Table 5.3. There may be no more than one symbol from each F field. The NOP symbol is used when the microinstruction has no microoperations. This will be translated by the assembler to nine zeros.
3.	CD	The CD field has one of the letters U, I, S, or Z.
4.	BR	The BR field contains one of the four symbols defined in Table 2.
5.	AD	The AD field specifies a value for the address field of the microinstruction in one of three possible ways: i. With a symbolic address, this must also appear as a label. ii. With the symbol NEXT to designate the next address in sequence. iii. When the BR field contains a RET or MAP symbol, the AD field is left empty and is converted to seven zeros by the assembler.

Table 3: Symbolic Microinstruction

Micro programmed sequencer for a control memory

Microprogram sequencer:

The basic components of a microprogrammed control unit are the control memory and the circuits that select the next address.

The address selection part is called a microprogram sequencer.

A microprogram sequencer can be constructed with digital functions to suit a particular application.

To guarantee a wide range of acceptability, an integrated circuit sequencer must provide an internal organization that can be adapted to a wide range of applications.

The purpose of a microprogram sequencer is to present an address to the control memory so that a microinstruction may be read and executed.

Commercial sequencers include within the unit an internal register stack used for temporary storage of addresses during microprogram looping and subroutine calls.

Some sequencers provide an output register which can function as the address register for the control memory.

The block diagram of the microprogram sequencer is shown in figure 6.
There are two multiplexers in the circuit.

The first multiplexer selects an address from one of four sources and routes it into a control address register CAR.

The second multiplexer tests the value of a selected status bit and the result of the test is applied to an input logic circuit.

The output from CAR provides the address for the control memory.

The content of CAR is incremented and applied to one of the multiplexer inputs and to the subroutine registers SBR.

The other three inputs to multiplexer 1 come from the address field of the present microinstruction, from the output of SBR, and from an external source that maps the instruction.

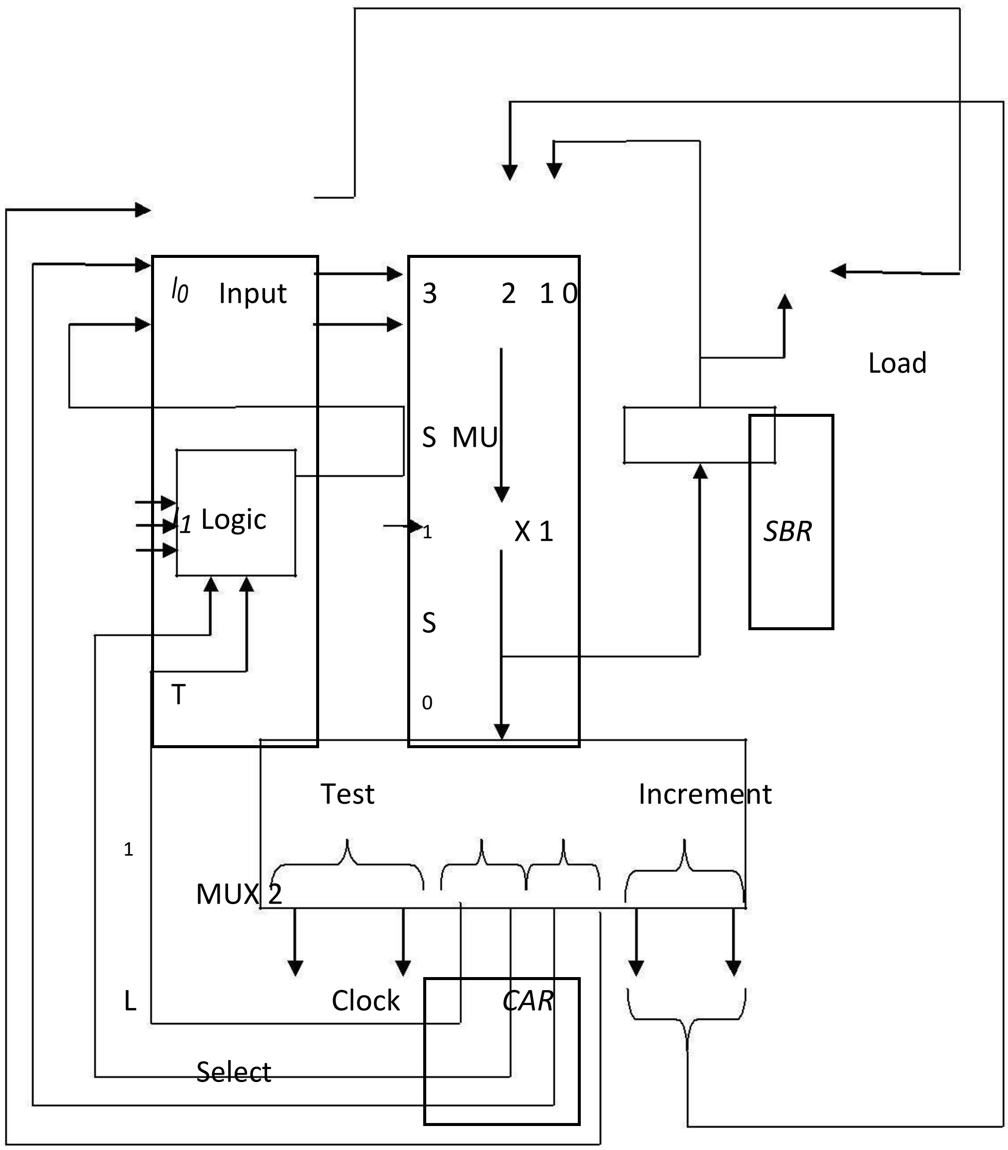
Although the figure 4.6 shows a single subroutine register, a typical sequencer will have a register stack about four to eight levels deep. In this way, a number of subroutines can be active at the same time.

The CD (condition) field of the microinstruction selects one of the status bits in the second multiplexer.

If the bit selected is equal to 1, the T (test) variable is equal to 1; otherwise, it is equal to 0.

The T value together with the two bits from the BR (branch) field goes to an input logic circuit.

The input logic in a particular sequencer will determine the type of operations that are available in the unit.



I_0 Input
3 2 1 0
S MU
I_1 Logic
1 X 1
S
0
T
Load
SBR
Test
Increment
1
MUX 2
L
Clock
CAR
Select

Figure 6: Microprogram Sequencer for a control memory

Input Logic : Truth Table

BR	Input			MUX 1		Load SBR
	I1	**I0**	**T**	**S1**	**S0**	**L**
0 0	0	0	0	0	0	0
0 0	0	0	1	0	1	0
0 1	0	1	0	0	0	0
0 1	0	1	1	0	1	1
1 0	1	0	X	1	0	0
1 1	1	1	X	1	1	0

Table 4: Input Logic Truth Table for Microprogram Sequencer

Function:

$$S0 = I0$$

$$S1 = I0I1 + I0'T$$

$$L = I0'I1T$$

Typical sequencer operations are: increment, branch or jump, call and return from subroutine, load an external address, push or pop the stack, and other address sequencing operations.

With three inputs, the sequencer can provide up to eight address sequencing operations.

Some commercial sequencers have three or four inputs in addition to the T input and thus provide a wider range of operations.

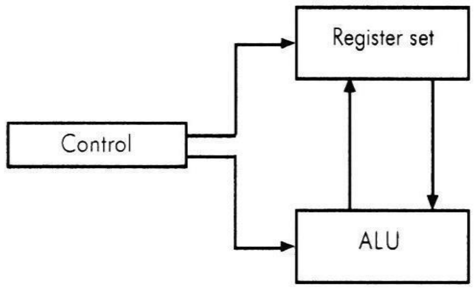
Central Processing Unit

· **General Register Organization :**

·

· The Central Processing Unit (CPU) is called the brain of the computer that performs data-processing operations. Figure 1 shows the three major parts of CPU.

.



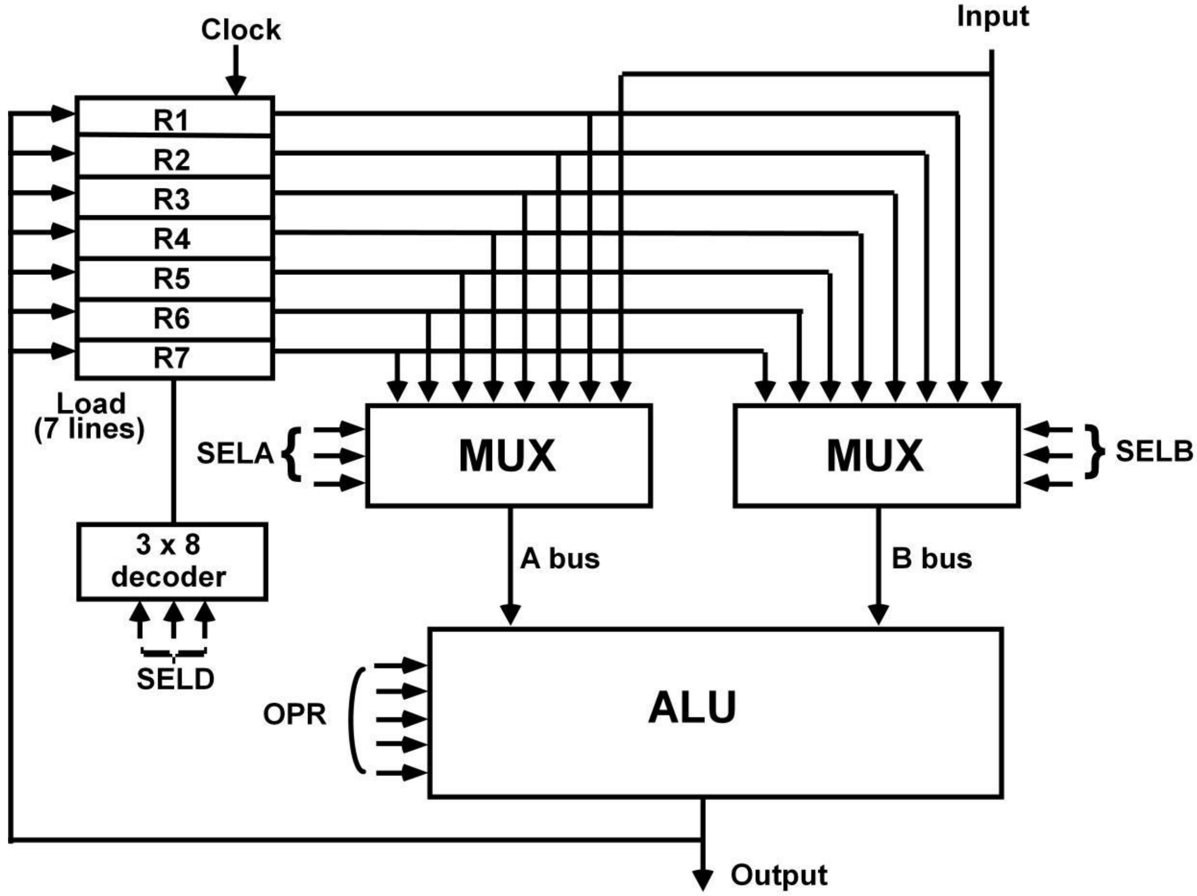
Intermediate data is stored in the register set during the execution of the instructions. The microoperations required for executing the instructions are performed by the arithmetic logic unit whereas the control unit takes care of transfer of information among the registers and guides the ALU. The control unit services the transfer of information among the registers and instructs the ALU about which operation is to be performed. The computer instruction set is meant for providing the specifications for the design of the CPU. The design of the CPU largely, involves choosing the hardware for implementing the machine instructions.

.

· The need for memory locations arises for storing pointers, counters, return address, temporary results and partial products. Memory access consumes the most of the time off an operation in a computer. It is more convenient and more efficient to store these intermediate values in processor registers.

· A common bus system is employed to contact registers that are included in the CPU in a large number. Communications between registers is not only for direct data transfer but also for performing various micro-operations. A bus organization for such CPU register shown in Figure 2, is connected to two multiplexers (MUX) to form two buses A and B. The selected lines in each multiplexers select one register of the input data for the particular bus.

·



OPERATION OF CONTROL UNIT:

The control unit directs the information flow through ALU by:

- Selecting various *Components* in the system
- Selecting the *Function* of ALU

1] MUX A selector (SELA): BUS A $\leftarrow$ R2

[2] MUX B selector (SELB): BUS B $\leftarrow$ R3

[3] ALU operation selector (OPR): ALU to ADD

[4] Decoder destination selector (SELD): R1 $\leftarrow$ Out Bus

3	3	3	5
SELA	SELB	SELD	OPR

Control Word

Encoding of register selection fields

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

R1 ← R2 - R3	R2	R3	R1	SUB	010 011 001 00101
R4 ← R4 ∨ 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 ← shl R4	R4	-	R4	SHLA	100 000 100 11000
R5 ← 0	R5	R5	R5	XOR	101 101 101 01100

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

Examples of ALU Microoperations

Symbolic Designation

Microoperation **SELA SELB SELD OPR** **Control Word**

Stack organization:

A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved.

The stack in digital computers is essentially a memory unit with an address register that can count only. 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 physical registers of a stack are always available for reading or writing. It is the content of the word that is inserted or deleted.

Register stack:

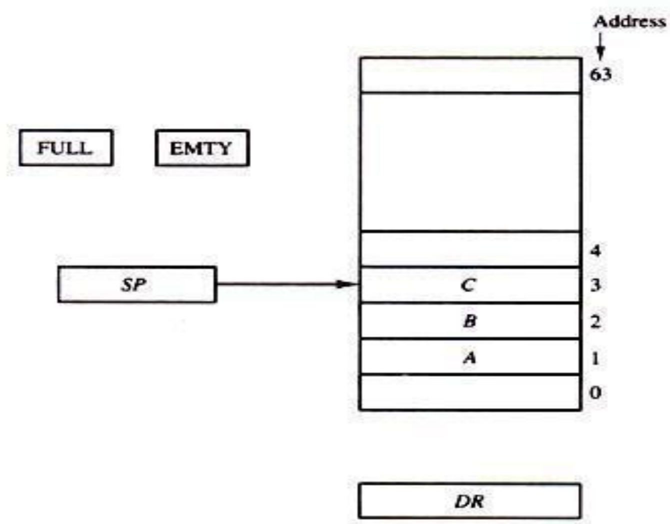


Figure 1: Block diagram of a 64-word stack

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 registers. Figure shows 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. Three items are placed in the stack: A, B, and C, in that order. 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 insert a new item, the stack is pushed by incrementing SP and writing a word in the next-higher location in the stack.

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 are 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 the 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.

PUSH:

If the stack is not full (FULL =0), a new item is inserted with a push operation. The push operation consists of the following sequences 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 is stack is full
EMTY ← 0	Mark the stack not empty

The stack pointer is incremented so that it points to the address of next-higher word. A memory write operation inserts the word from DR into the top of the stack.

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. This 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.

POP:

A new item is deleted from the stack if the stack is not empty (if EMTY = 0). The pop operation consists of the following sequences of microoperations:

DR ← M [SP]	Read item on top of the 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

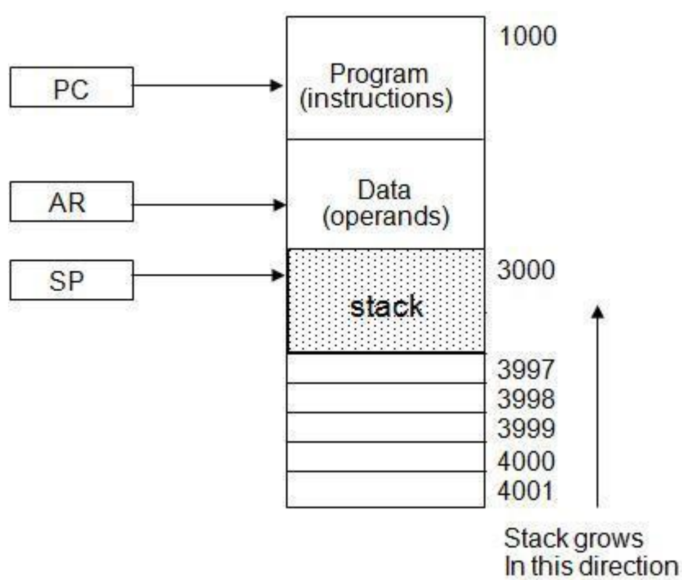
The top item is read from the stack into DR. The stack pointer is then decremented. If its value reaches zero, the stack is empty, so EMTY is set to 1.

This condition is reached if the item read was in location 1.

Once this item is read out, SP is decremented and reaches the value 0, which is the initial value of SP. If a pop operation reads the item from location 0 and then SP is decremented, SP is changes to 111111, which is equivalent to decimal 63.

In this configuration, the word in address 0 receives the last item in the stack. Note also that an erroneous operation will result if the stack is pushed when FULL = 1 or popped when EMTY = 1.

Memory Stack.



Computer memory with program, data, and stack segments

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.

Figure 5.2 shows 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 which is used during the fetch phase to read an instruction.

e address registers AR points at an array of data which is used during the execute phase to read an operand.

The stack pointer SP points at the top of the stack which is used to push or pop items into or from the stack.

The three registers are connected to a common address bus, and either one can provide an address for memory.

As shown in Figure 5.2, 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.

We assume that the items in the stack communicate with a data register DR.

PUSH

A new item is inserted with the push operation as follows:

$$SP \leftarrow SP - 1$$

$$M[SP] \leftarrow DR$$

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.

POP

A new item is deleted with a pop operation as follows:

$$DR \leftarrow M[SP]$$

$$SP \leftarrow SP + 1$$

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.

The two microoperations needed for either the push or pop are (1) an access to memory through SP, and (2) updating SP.

Which of the two microoperations is done first and whether SP is updated by incrementing or decrementing depends on the organization of the stack.

In figure. 5.2 the stack grows by decreasing the memory address. The stack may be constructed to grow by increasing the memory also.

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.