

teachics.org

Computer Organization & Architecture

Module 4 – Part 1

Microprogrammed Control

Control Unit

- The function of the control unit in a digital computer is to initiate sequences of microoperations.
- Two methods of implementing control unit are
 - Hardwired control
 - Microprogrammed control.
- Hardwired Control
 - Design involves the use of fixed instructions, fixed logic blocks, encoders, decoders, etc.
 - Key characteristics are high-speed operation, expensive, relatively complex, and no flexibility of adding new instructions.
 - Example CPUs: Intel 8085, Motorola 6802, and any RISC (Reduced Instruction Set Computer) CPUs.

Microprogrammed Control Unit

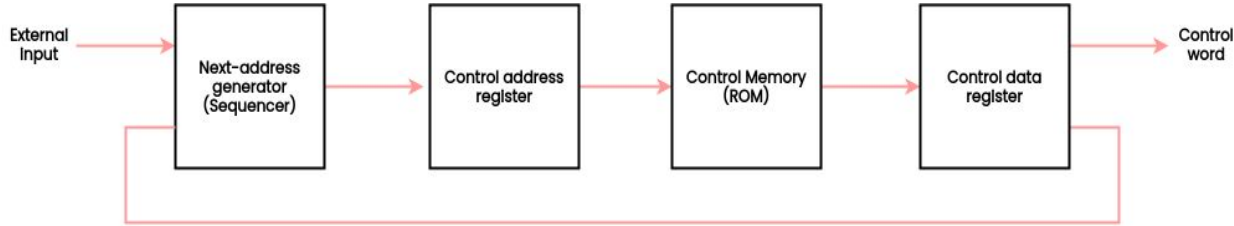
- A control unit whose binary control variables are stored in memory is called a microprogrammed control unit.
- Main advantage - for different control sequence; only have to change microprogram residing in control memory. (No need of hardware changes)
- The control function that specifies a microoperation is a binary variable.
- The control variables at any given time can be represented by a string of 1's and 0's called a control word.
- Each word in control memory contains a microinstruction.
- The microinstruction specifies one or more microoperations.
- A sequence of microinstructions constitutes a microprogram.

Microoperations → Control word → Microinstruction → Microprogram → Control Memory.

Microprogrammed Control Unit

- A computer with microprogrammed control unit will have 2 memories.
 - Main Memory.
 - Control Memory.
- The **main memory** is available to the user for storing the programs.
- The contents of main memory may alter when the data are manipulated and every time that the program is changed.
- A memory that is part of a control unit is referred to as a control **memory**.
- The control memory holds a fixed microprogram that cannot be altered by the user and contains various control signals.
- The control memory can be a read-only memory (ROM) since alterations are not needed.
- Writable control memory is used in dynamic programming.

Microprogrammed Control Unit



- The control memory is a ROM in which all control information is permanently stored.
- The control address register specifies the address of the microinstruction
- Control data register holds the microinstruction read from memory.
- The microinstruction contains a control word that specifies one or more microoperations for the data processor.
- The next address is computed in the **next address generator**(sequencer) and then transferred into the control address register to read the next microinstruction.
- The **control data register**(pipeline register) holds the present microinstruction while the next address is computed and read from memory.

Address Sequencing

Address Sequencing

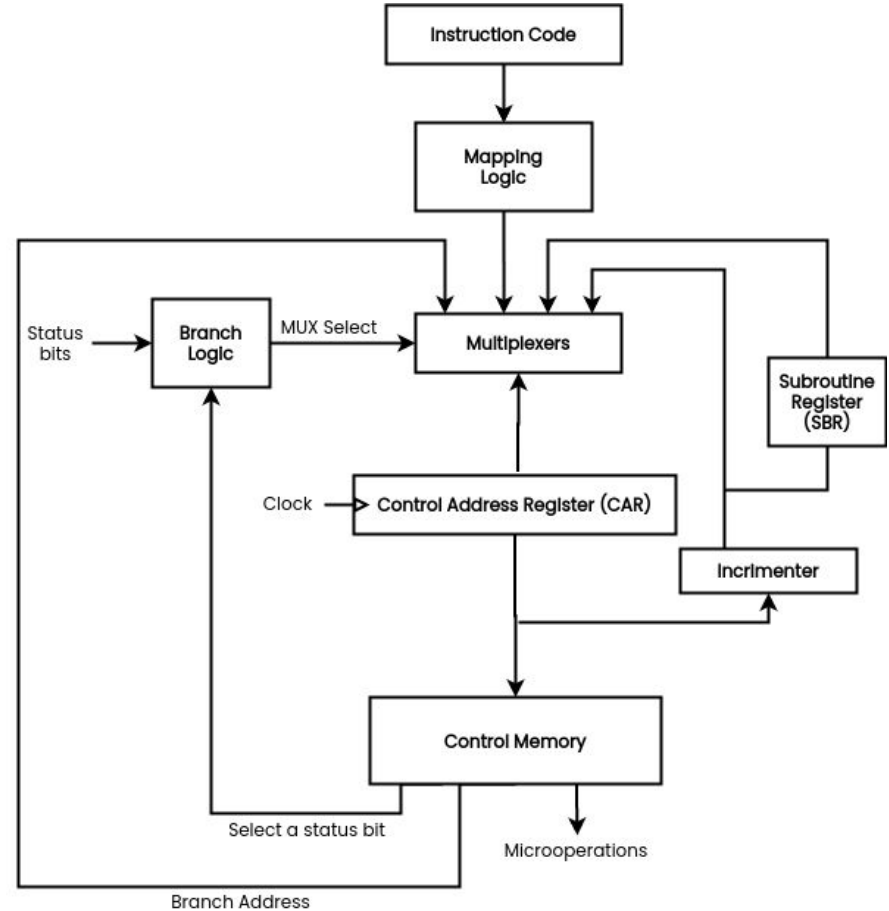
- Microinstructions are stored in control memory in groups, with each group specifying a routine.
- Each computer instruction has its own microprogram routine in control memory to generate the microoperations that execute the instruction.
- The hardware that controls the address sequencing must be capable of sequencing the microinstructions within a routine and be able to branch from one routine to another.

Executing a Single Instruction

- **Instruction Fetch Routine**
 - An initial address is loaded into the CAR when power is turned on.
 - Routine is sequenced by incrementing the control address register(CAR).
 - At the end of the routine, the instruction is in the instruction register (IR)
- **Effective Address Computation Routine**
 - It determines the effective address of the operand.
 - Routine can be reached through a branch microinstruction, based on the status of the mode bits of the instruction.
 - At the end the address of the operand is in the memory address register.
- **Generating Microoperations**
 - Depend on the operation code part of the instruction.
 - 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.

In brief, the **address sequencing capabilities** required in a control memory

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.

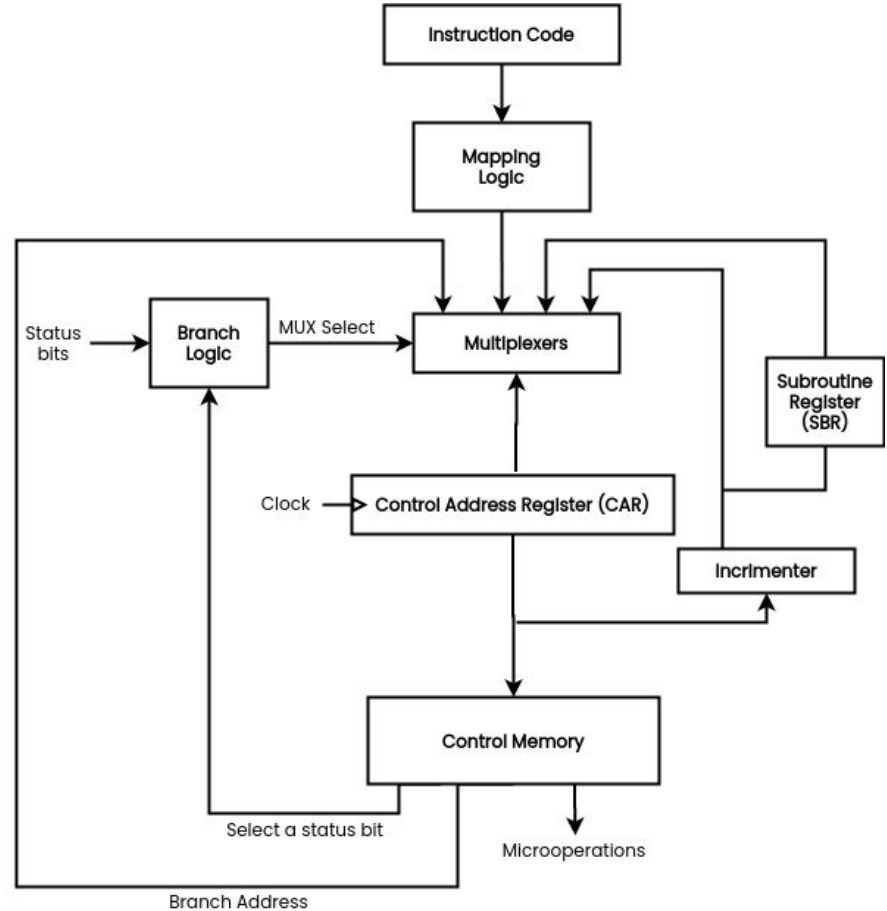


Incrementing CAR

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

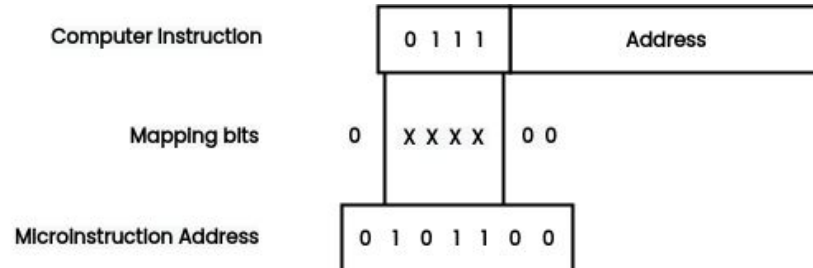
Conditional Branching

- Branch logic provides decision-making capabilities in the control unit.
- The status bits, together with the field in the microinstruction that specifies a branch address, control the conditional branch decisions
- Simplest way to implement branch logic is to test the specified condition and branch to an address if the condition is met; else increment the address register.



Mapping of Instruction

- A special type of branch instruction.
- Here a branching is done to the first word in control memory where a microprogram routine for an instruction is located.
- The status bits for this branch are the bits in the operation code of the instruction.
- Can be implemented using ROM.
- The bits of the instruction specify the address of a mapping ROM.
- The contents of the mapping ROM give the bits for the control address register.
- This concept provides flexibility for adding instructions for control memory as the need arises.



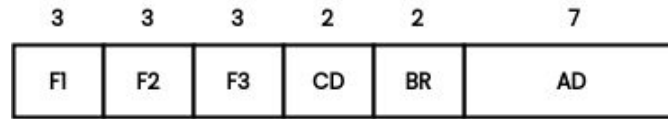
Subroutines

- Programs that are used by other routines to accomplish a particular task.
- Microinstructions can be saved by employing subroutines that use common sections of microcode.
- Must have a provision for storing the return address during a subroutine call and restoring the address during a subroutine return.
- This may be accomplished by placing the incremented address from the control address register into a subroutine register and branching to the beginning of the subroutine.
- The subroutine register can then become the source for transferring the address for the return to the main routine.

Microprogram Example

Microinstruction Format

- Generating microcode for the control memory is called microprogramming.



- 20 bits of the microinstruction are divided into four functional parts.
- Three fields F1, F2, and F3 specify microoperations for the computer.
 - Each field is 3 bits which provides 21 microoperations.
- CD field selects status bit conditions.
 - 2 bits are encoded to specify 4 status bits conditions.
- BR field specifies the type or branch to be used.
 - Used in conjunction with the field AD, to choose the address of the next microinstruction.
- AD field contains a branch address. Since the control memory has $128 = 2^7$ words, AD has 7 bits.

Microinstruction Field Description

F1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(010) \leftarrow PC$	PCTDR

F3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOP
010	$AC \leftarrow AC$	CM
011	$AC \leftarrow \text{shl } AC$	SHL
100	$AC \leftarrow \text{shr } AC$	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

Microinstruction Field Description

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

BR	Symbol	Function
00	JMP	CAR \leftarrow AD if condition = 1 CAR \leftarrow CAR 1 if condition = 0
01	CALL	CAR \leftarrow AD, SBR \leftarrow CAR 1 if condition = 1 CAR \leftarrow CAR 1 if condition = 0
10	RET	CAR \leftarrow SBR (Return from subroutine)
11	MAP	CAR(25) \leftarrow DR(1114), CAR(0,1,6) \leftarrow 0

Symbolic Microinstructions

- Symbolic microprogram can be translated into its binary equivalent by means of an assembler.
- Each symbolic microinstruction is divided into five fields
 - Label
 - Microoperations
 - CD
 - BR
 - AD
- The label field may be empty or it specify a symbolic address.
- It is terminated with a colon (:)
- The microoperations field consists of one, two, or three symbols, separated by commas.
There may be no more than one symbol from each F field.
The NOP symbol is used when the microinstruction has no microoperations.

Symbolic Microinstructions

- The CD field has one of the letters U, I, S, or Z.
- BR field contains one of the four symbols JMP, CALL, RET, MAP.
- The AD field specifies a value for the address field of the microinstruction in one of three possible ways:
 - With a symbolic address, which must also appear as a label.
 - With the symbol NEXT to designate the next address in sequence.
 - When the BR field contains a RET or MAP symbol, the AD field is left empty.
- We will use also the pseudoinstruction ORG to define the origin, or first address, of a microprogram routine.

The Fetch Routine

- The microinstructions needed for the fetch routine are

$AR \leftarrow PC$

$DR \leftarrow M[AR], PC \leftarrow PC + 1$

$AR \leftarrow DR(0 - 10), CAR(25) \leftarrow DR(11 - 14), CAR(0,1,6) \leftarrow 0$

- Symbolic microprogram for the fetch routine

ORG 64

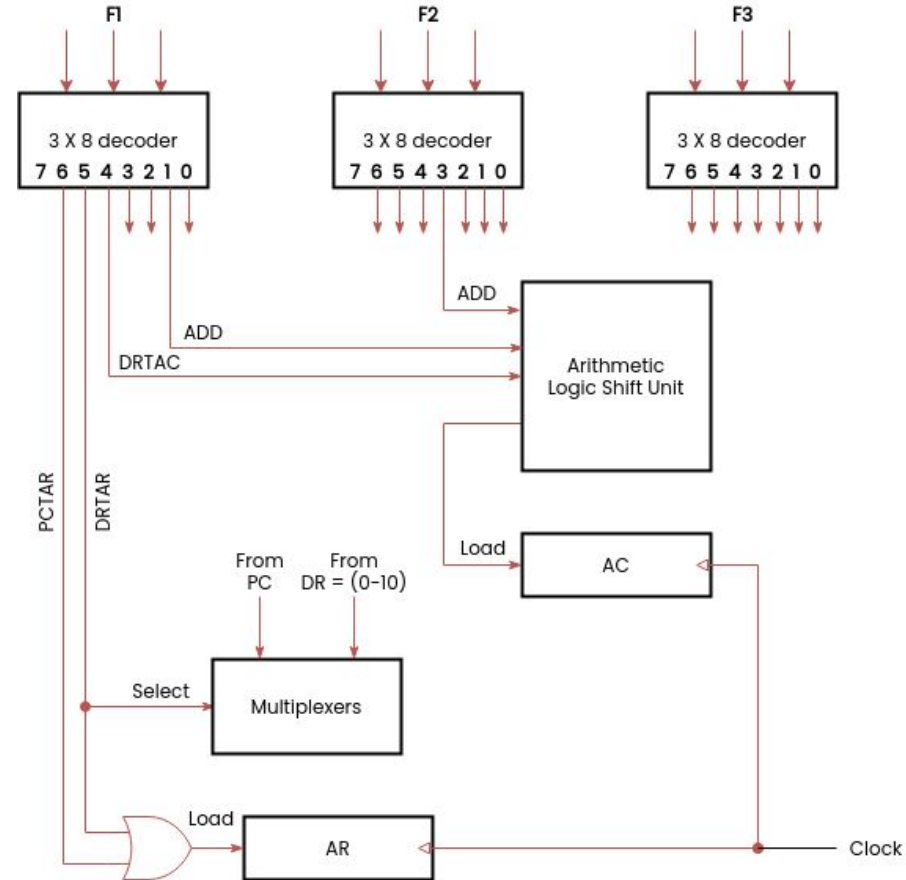
```
FETCH:   PCTAR           U   JMP       NEXT
          READ,INCPC     U   JMP       NEXT
          DRTAR          U   MAP
```

- Equivalent binary microprogram

Binary Address	F1	F2	F3	CD	BR	AD
1000000	110	000	000	00	00	1000001
1000001	000	100	101	00	00	1000010
1000010	101	000	000	00	11	0000000

Design of Control Unit










- Bits of the microinstruction are usually divided into fields, with each field defining a separate function.(F1-F2-F3, CD, BR,AD)
- Each field requires a decoder to produce the corresponding control signals.
- Each of the three fields of the microinstruction presently available in the output of control memory are decoded with a 3x8 decoder to provide eight outputs.
- Each of these outputs must be connected to the proper circuit to initiate the corresponding microoperation.



Microprogram Sequencer

- The basic components of microprogrammed control unit are the control memory and the circuits that select the next address.
- The address selection part is called as microprogram sequencer.
- Microprogram sequencer can be constructed with digital functions to suit a particular application.
- Two imp. factors that must be considered while designing the microinstruction sequencer:
 - The size of the microinstruction.
 - The address generation time.
- The purpose of microprogram sequencer is to present an address to the control memory so that a microinstruction may be read and executed.
- The next address logic of the sequencer determines the specific address source to be loaded into the CAR.
- The choice of the address source is guided by the next address information bits that the sequencer receives from address information bits that the sequencer receives from the present microinstruction.

Study at Home

 Notes Learn topics better with a little less effort utilizing notes by subject experts.	 Syllabus View or download syllabus for Bsc Computer Science under Calicut University.	 Video Lessons Better learning experience through video lectures from skilled personnel.
 Question Papers Familiarize with exam patterns through our library of previous year question papers.	 Quiz Test your knowledge of topics through various quizzes.	 Solved Questions Vast collection of questions and answers related to each topic.
 Programming Laboratory Practicals made easy with complete list of tested programs.	 Software Downloads Practice programming by downloading and installing respective softwares.	 Teaching Resources Find PPTs and related resources for teaching aid here.

www.teachics.org

teachics.org

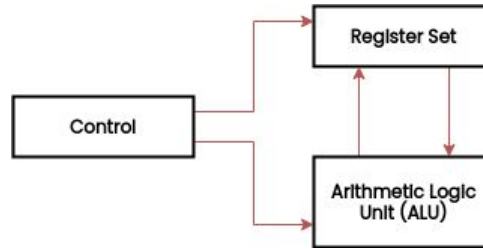
Computer Organization & Architecture

Module 4 – Part 2

General Register Organization

Central Processing Unit

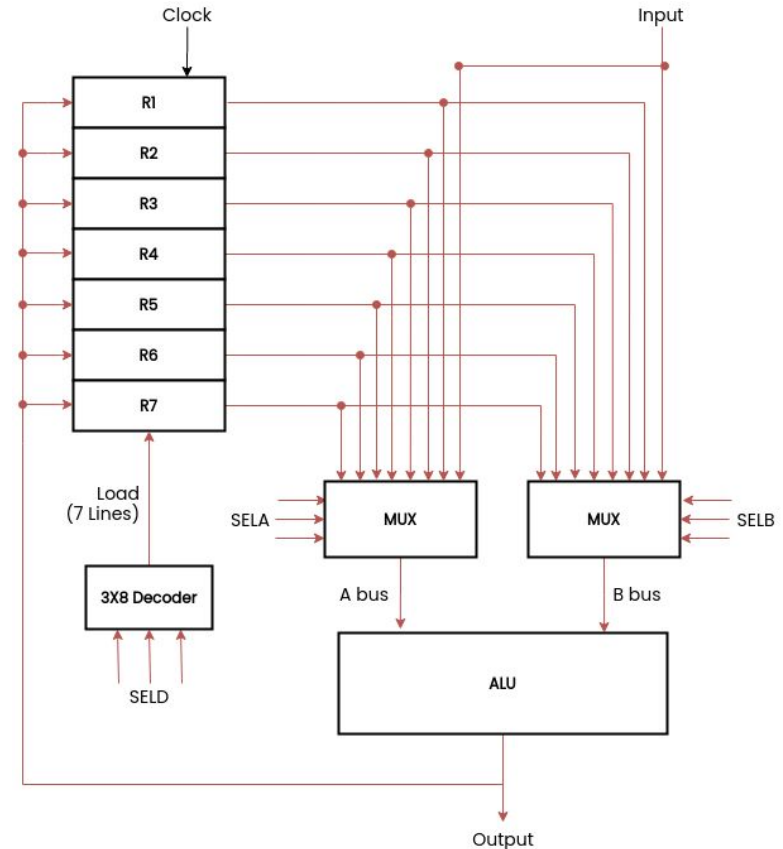
- The part of the computer that performs the data-processing operations is called the central processing unit(CPU).



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

General Register Organization

- Registers are used to store the intermediate values during instruction execution.
- Register organization show how registers are selected and how data flow between register and ALU.
- A decoder is used to select a particular register.
- The output of each register is connected to two multiplexers to form the two buses A and B.
- The selection lines in each multiplexer select the input data for the particular bus.
- The A and B buses form the two inputs of an ALU.
- The operation select lines decide the micro operation to be performed by ALU.

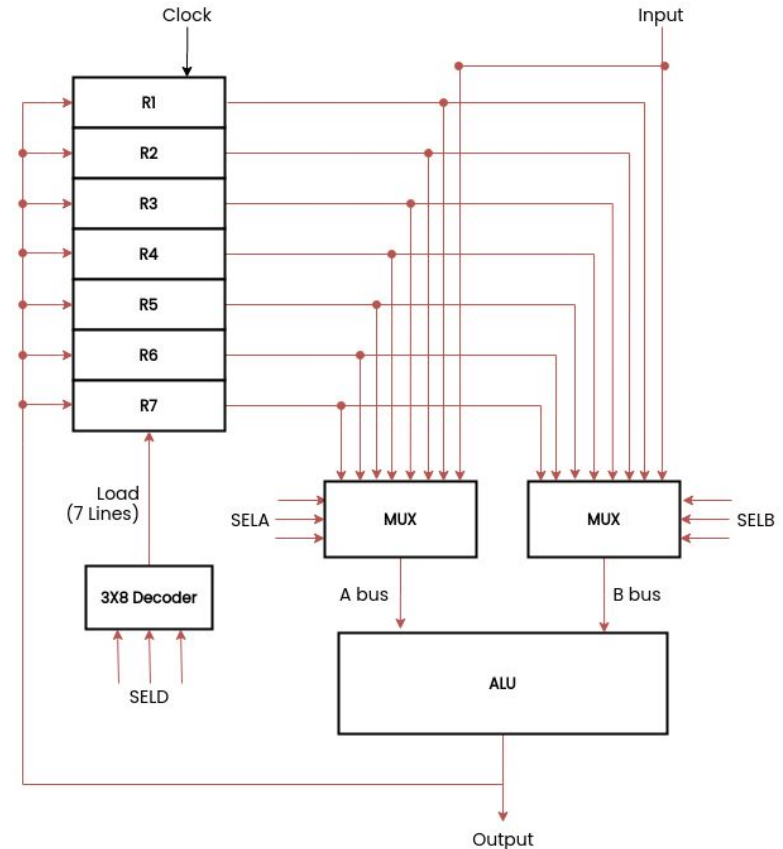


General Register Organization

- The result of the micro operation is available at the output bus.
- The output bus connected to the inputs of all registers, thus by selecting a destination register it is possible to store the result in it.

Example : To perform $R1 \leftarrow R2 + R3$,

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.
4. Decoder destination selector (SELD): to transfer the content of the output bus into R1.



Control Word

- The combined value of a binary selection inputs specifies the control word.
- It consist of four fields SELA,SELB,and SELD contains three bit each and OPR field contains four bits thus the total bits in the control word are 13-bits.



- The three bit of SELA select a source registers of the a input of the ALU.
- The three bits of SELB select a source registers of the b input of the ALU.
- The three bits of SELD select a destination register using the decoder.
- The four bits of OPR select the operation to be performed by ALU.

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

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	$R7$	$R7$	$R7$

Example : $R2=R1+R3$



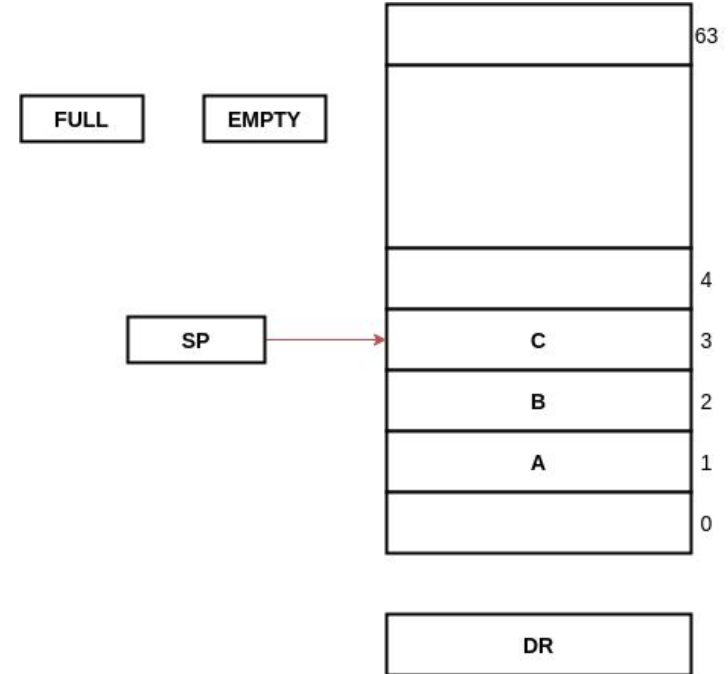
Stack Organization

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 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 stack in digital computers is essentially a memory unit with an address register that can count only after an initial value is loaded into it.
- The two operations of a stack are the insertion and deletion of items.

Register Stack

- 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.
- The stack pointer register SP contains a binary number = address of the word that is currently on top of the stack.
- In a 64-word stack, the stack pointer contains 6 bits because $2^6 = 64$.
- 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.

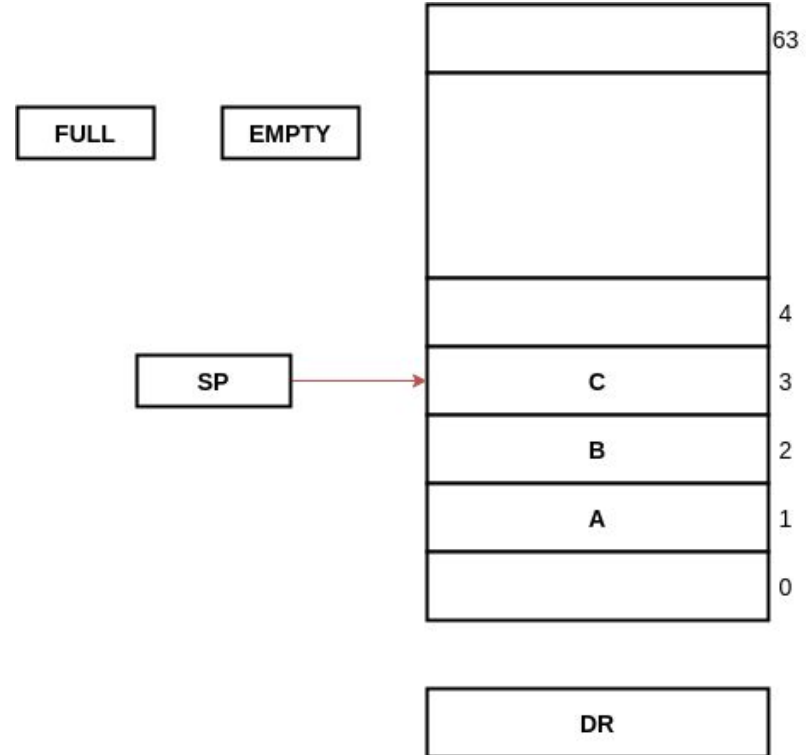


Register Stack

- FULL = 1 , when stack is full & EMPTY=1 if stack is empty
- Initially SP=0, EMPTY=1 and FULL=0

To delete an item.

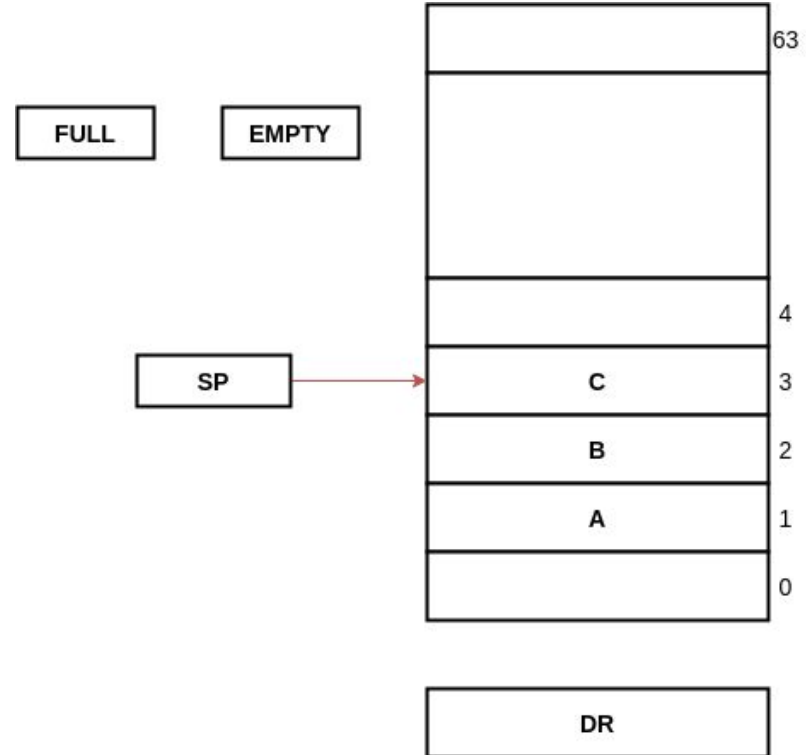
- Item is deleted, if the stack is not empty.
- Read the top item into DR.
- If SP reaches 0, EMPTY is set to 1.
- Decrement SP by 1.
- $DR \leftarrow M[SP]$
 $SP \leftarrow SP - 1$
If (SP = 0) then (EMPTY \leftarrow 1)
FULL \leftarrow 0



Register Stack

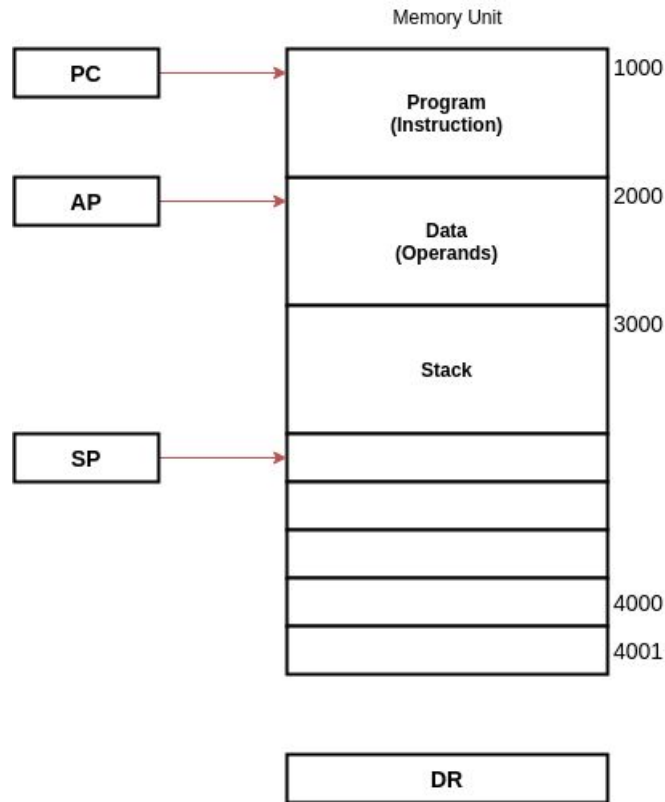
To Insert a new item.

- Item is pushed only if stack is not full.
- Increment SP by 1.
- Write a word in that higher location.
- If SP reaches 0, the stack is full of items, so FULL is set to 1.
- $SP \leftarrow SP + 1$
 $M[SP] \leftarrow DR$
If $(SP=0)$ then $(FULL \leftarrow 1)$
 $EMPTY \leftarrow 0$



Memory Stack

- Implemented by assigning a portion of memory to a stack operation and using a processor register as a stack pointer.
- The program counter (PC) points at the address of the next instruction in the program.
- PC is used during the fetch phase to read an instruction.
- The address register (AR) points at an array of data.
- AR is used during the execute phase to read an operand.
- The stack pointer SP points at the top of the stack.
- P is used to push or pop items into or the stack.
- **Advantage** - CPU can refer to it without having to specify an address (instead use SP).



Instruction Formats

Instruction Formats

- The bits of the instruction are divided into groups called fields.
 1. Opcode field - specifies the operation to be performed.
 2. Address field - specifies a memory address / processor register.
 3. Mode field - specifies the way the operand or the effective address is determined.
- The number of address fields depends on the internal organization of registers.
- Three types of CPU organizations:
 1. Single accumulator organization.
Eg: ADD X
 2. General register organization.
Eg: ADD R1, R2, R3 - MOV R1, R2
 3. Stack organization
Eg: PUSH X
- Based on these, instructions are classified into four formats.

Three-Address Instruction

- Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand.
- The program in assembly language that evaluates $X = (A + B) * (C + D)$

```
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
```

- Advantage - It results in short programs when evaluating arithmetic expressions.
- Disadvantage - The binary-coded instructions require too many bits to specify three addresses.
- Eg: Commercial computer Cyber 170.

Two-Address Instruction

- Two-address instructions are the most common in commercial computers.
- Here again each address field can specify either a processor register or a memory word.
- The program to evaluate $X = (A + B) * (C + D)$

```
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 Instruction

- Use an implied accumulator (AC) register for all data manipulation.
- Here we neglect the second register and assume that the AC contains the result of all operations.
- The program to evaluate $X = (A + B) * (C + D)$

```
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
```

- T is the address of a temporary memory location required for storing the intermediate result.

Zero-Address Instruction

- Used in stack-organized computers.
- The program to evaluate $X = (A + B) * (C + D)$ for a stack-organized computer

```
PUSH A    // TOS ← A
PUSH B    // TOS ← B
ADD       // TOS ← (A + B)
PUSH C    // TOS ← C
PUSH D    // TOS ← D
ADD       // TOS ← (C + D)
MUL       // TOS ← (C + D)*(A + B)
POP X     // M[X] ← TOS
```

- To evaluate arithmetic expressions in a stack computer, it is necessary to convert the expression into reverse Polish notation.

Addressing Modes

Addressing Mode

- The operation field of an instruction specifies the operation to be performed.
- This operation must be executed on some data stored in computer registers or memory words.
- The way the operands are chosen during program execution is dependent on the addressing mode of the instruction.
- The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced.
- 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

1. Implied Mode

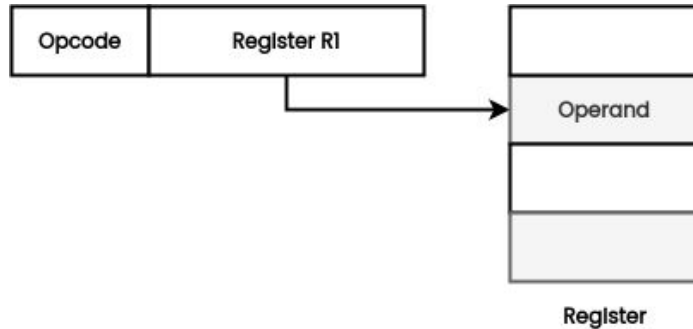
- The operands are specified implicitly in the definition of the instruction.
- Eg: CMA - Complement Accumulator.
- All register reference instructions that use an accumulator are implied-mode instructions.
- Zero-address instructions in a stack-organized computer are implied-mode instructions since the operands are implied to be on top of the stack.

2. Immediate Mode

- The operand is specified in the instruction itself.
- I has an operand field rather than an address field.
- The operand field contains the actual operand to be used in conjunction with the operation specified in the instruction.
- They are useful for initializing registers to a constant value.
- Eg: ADD 7

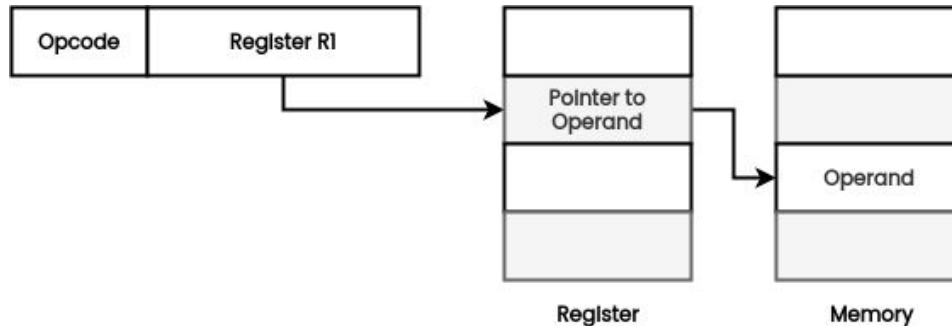
3. Register Mode

- The address field specifies a processor register.
- 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.
- A k-bit field can specify any one of 2^k registers.
- Eg: ADD R1



4. Register Indirect Mode

- The instruction specifies a register in the CPU whose contents give the address of the operand in memory.
- The selected register contains the address of the operand rather than the operand itself.
- Before using a register indirect mode instruction, the programmer must ensure that the memory address of the operand is placed in the processor register with a previous instruction.
- Advantage - The address field of the instruction uses fewer bits to select a register.

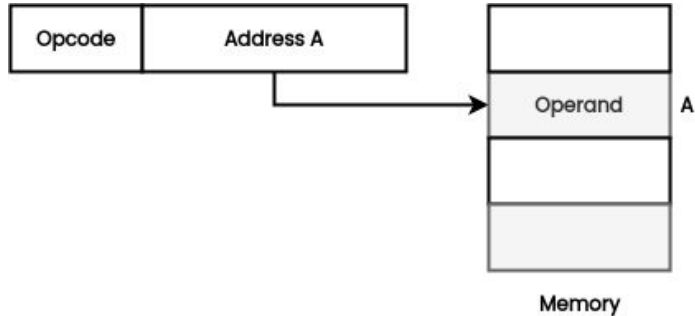


5. Autoincrement or Autodecrement Mode

- Similar to the register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory.
- When the address stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table.
- This can be achieved by using the increment or decrement instruction.

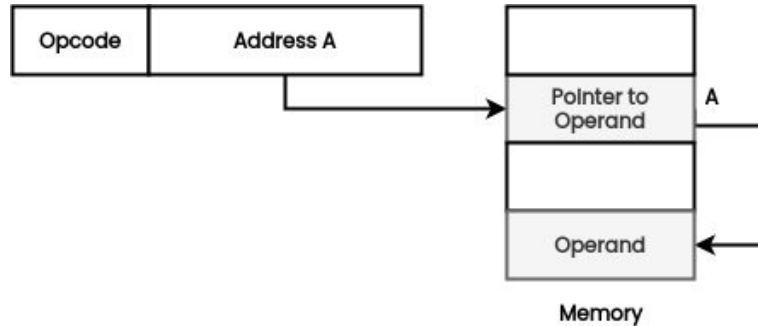
6. Direct Address Mode

- The effective address is equal to the address part of the instruction.
- The operand resides in memory and its address is given directly by the address field of the instruction.
- In a branch-type instruction the address field specifies the actual branch address.



7. Indirect Address Mode

- The address field of the instruction gives the address where the effective address is stored in memory.
- Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.



8. Relative Address Mode

- Content of the program counter is added to the address part of the instruction in order to obtain the effective address.
- The address part of the instruction is usually a signed number (positive or negative).
- This number is added to the content of the program counter, producing an effective address whose position in memory is relative to the address of the next instruction.
- It is often used with branch-type instructions.

- **Example:**

Let PC contains the number 825.

The address part of the instruction contains the number 24.

The instruction at location 825 is read from memory during the fetch phase and the program counter is then incremented by one to 826.

The effective address computation for the relative address mode is $826 + 24 = 850$.

9. Indexed Addressing Mode

- Content of an index register is added to the address part of the instruction to obtain the effective address.
- The index register is a special CPU register that contains an index value.
- The address field of the instruction defines the beginning address of a data array in memory.
- The distance between the beginning address and the address of the operand is the index value stored in the index register.
- Any operand in the array can be accessed with the same instruction if the index register contains the correct index value.
- The index register can be incremented to facilitate access to consecutive operands.

10. 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.
- Similar to the indexed addressing mode except that the register is now called a base register.
- The difference between the two modes is in the way they are used rather than in the way that they are computed.
- A base register holds a base address and the address field of the instruction gives a displacement relative to this base address.
- This mode is used to facilitate the relocation of programs in memory.
- When programs and data are moved from one segment of memory to another, as required in multiprogramming systems, the address values of instructions must reflect this change of position.
- Here only the value of the base register requires updating to reflect the beginning of a new memory segment.

Data Transfer and Manipulation

Data Transfer & Manipulation

- Computers provide an extensive set of instructions to give the user the flexibility to carry out various computational tasks.
- The basic set of operations available in a typical computer 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 changing the binary information content.
- Data manipulation instructions are those that perform arithmetic, logic, and shift operations.
- Program control instructions provide decision-making capabilities and change the path taken by the program when executed in the computer.

Data Transfer Instruction

- Data transfer instructions move data from one place in the computer to another without changing the data content.
- The most common transfers are between memory and processor registers, between processor registers and input or output, and between the processor registers themselves.
- Typical Instructions are,
 1. **Load (LD)** - Transfer from memory to a processor register, usually an accumulator.
 2. **Store (ST)** - Transfer from a processor register into memory.
 3. **Move (MOV)** - Transfer from one register to another.
 4. **Exchange (XCH)** - Swaps information between two registers or a register and a memory word.
 5. **Input (IN), Output (OUT)** - Transfer data among processor registers and input or output terminals.
 6. **Push (PUSH), Pop(POP)** - Transfer data between processor registers and a memory stack.

Data Manipulation Instructions

- Perform operations on data and provide the computational capabilities for the computer.
- Divided into three basic types:
 1. Arithmetic instructions.
 2. Logical and bit manipulation instructions.
 3. Shift instructions.

Arithmetic Instructions

- **Increment (INC)** : adds 1 to the value stored in a register or memory word.
- **Decrement (DEC)** : subtracts 1 from a value stored in a register or memory word.
- **Add (ADD)** : Addition.
- **Subtract (SUB)** : Subtraction.
- **Multiply (MUL)** : Multiplication.
- **Divide (DIV)** : Division.
- **Add with carry (ADDC)** : Performs the addition on two operands plus the value of the carry from the previous computation.
- **Subtract with borrow (SUBB)** : Subtracts two words and a borrow which may have resulted from a previous subtract operation.
- **Negate (NEG)** : Forms the 2's complement of a number.

Logical and Bit Manipulation Instructions

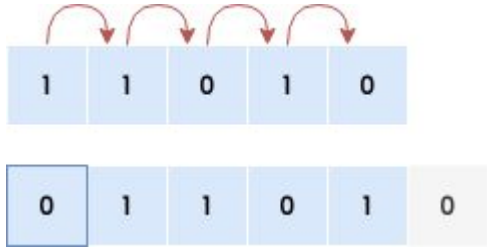
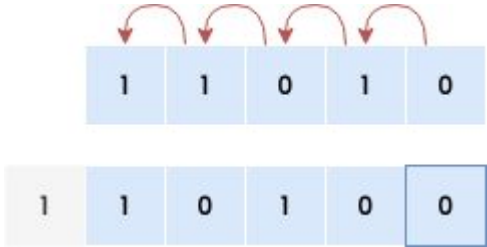
- **Clear (CLR)**
- **Complement (COM)**
- **AND (AND)**
- **OR (OR)**
- **Exclusive-OR (XOR)**
- **Clear carry (CLRC)**
- **Set carry (SETC)**
- **Complement carry (COMC)**
- **Enable interrupt (EI)**
- **Disable interrupt (DI)**

Shift Instructions

- Shifts are operations in which the bits of a word are moved to the left or right.
- The bit shifted in at the end of the word determines the type of shift used.
- Types
 1. Logical Shift.
 2. Arithmetic Shift
 3. Rotate Shift

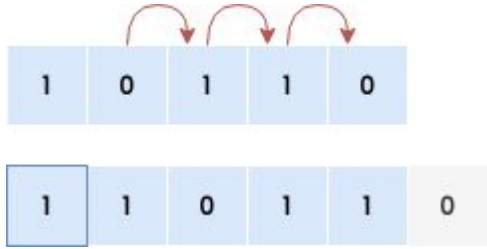
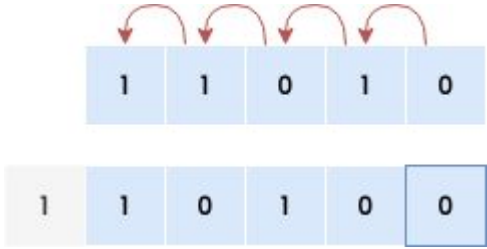
Logical Shift

- Inserts 0 to the end bit position.
- The end position is the leftmost bit for shift right and the rightmost bit for the shift left.

Logical Shift Right (SHR)	Logical Shift Left (SHL)
	
11010 to 01101	11010 to 10100

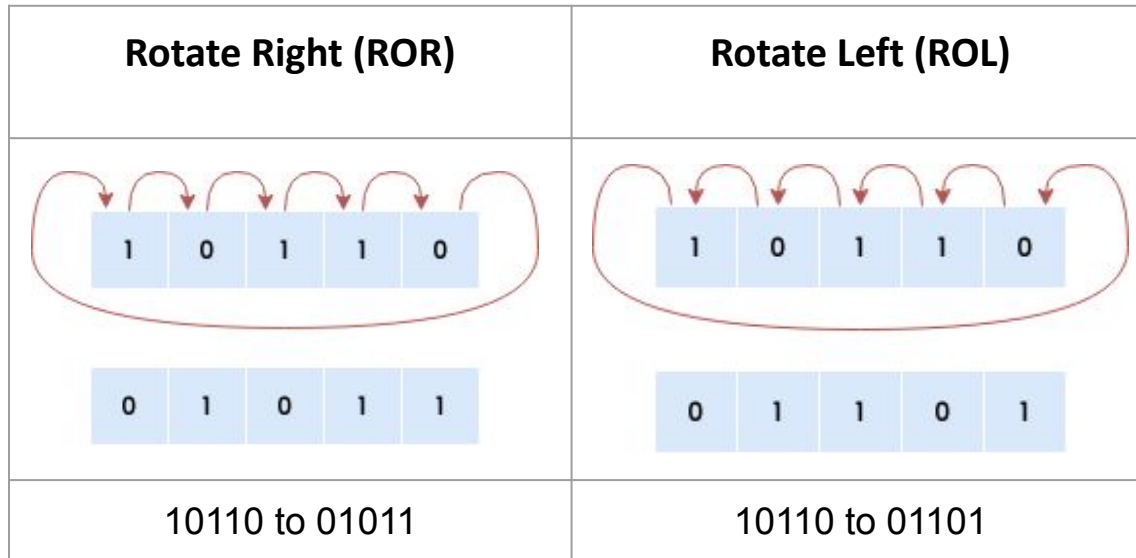
Arithmetic Shift

- **Arithmetic Shift Right (SHRA)** - Preserve sign bit in the left most position. Sign bit shifted to right along with other numbers but sign bit remain unchanged.

Arithmetic Shift Right (SHRA)	Arithmetic Shift Left (SHLA)
	
11010 to 11011	11010 to 10100

Rotate Shift

- Bits are shifted out one end are not lost but circulated back into other end.
- **Rotate Left Through Carry(ROL), Rotate right through Carry (RORC)** treats carry bit as an extension of register whose word is being rotated.












Program Control

Program Control Instructions

- Program control instructions specify conditions for altering the content of the program counter.
- Provides decision making capabilities.
- **Branch (BR)** : BR ADR, branch the program to Address ADR. ($PC \leftarrow ADR$)
- **Jump (JMP)**
- **Skip (SKP)** : Skip instruction($PC \leftarrow PC + 1$) if some condition is met.
- **Call (CALL)** : Used with subroutines
- **Return (RET)**
- **Compare (CMP)** : Compare by subtraction.
- **Test (by ANDing) TST** : AND instruction without storing result.

Study at Home

 Notes Learn topics better with a little less effort utilizing notes by subject experts.	 Syllabus View or download syllabus for Bsc Computer Science under Calicut University.	 Video Lessons Better learning experience through video lectures from skilled personnel.
 Question Papers Familiarize with exam patterns through our library of previous year question papers.	 Quiz Test your knowledge of topics through various quizzes.	 Solved Questions Vast collection of questions and answers related to each topic.
 Programming Laboratory Practicals made easy with complete list of tested programs.	 Software Downloads Practice programming by downloading and installing respective softwares.	 Teaching Resources Find PPTs and related resources for teaching aid here.

www.teachics.org

teachics.org

Computer Organization & Architecture

Module 4 – Part 3

Data Transfer and Manipulation

Data Transfer & Manipulation

- Computers provide an extensive set of instructions to give the user the flexibility to carry out various computational tasks.
- The basic set of operations available in a typical computer 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 changing the binary information content.
- Data manipulation instructions are those that perform arithmetic, logic, and shift operations.
- Program control instructions provide decision-making capabilities and change the path taken by the program when executed in the computer.

Data Transfer Instruction

- Data transfer instructions move data from one place in the computer to another without changing the data content.
- The most common transfers are between memory and processor registers, between processor registers and input or output, and between the processor registers themselves.
- Typical Instructions are,
 1. **Load (LD)** - Transfer from memory to a processor register, usually an accumulator.
 2. **Store (ST)** - Transfer from a processor register into memory.
 3. **Move (MOV)** - Transfer from one register to another.
 4. **Exchange (XCH)** - Swaps information between two registers or a register and a memory word.
 5. **Input (IN), Output (OUT)** - Transfer data among processor registers and input or output terminals.
 6. **Push (PUSH), Pop(POP)** - Transfer data between processor registers and a memory stack.

Data Manipulation Instructions

- Perform operations on data and provide the computational capabilities for the computer.
- Divided into three basic types:
 1. Arithmetic instructions.
 2. Logical and bit manipulation instructions.
 3. Shift instructions.

Arithmetic Instructions

- **Increment (INC)** : adds 1 to the value stored in a register or memory word.
- **Decrement (DEC)** : subtracts 1 from a value stored in a register or memory word.
- **Add (ADD)** : Addition.
- **Subtract (SUB)** : Subtraction.
- **Multiply (MUL)** : Multiplication.
- **Divide (DIV)** : Division.
- **Add with carry (ADDC)** : Performs the addition on two operands plus the value of the carry from the previous computation.
- **Subtract with borrow (SUBB)** : Subtracts two words and a borrow which may have resulted from a previous subtract operation.
- **Negate (NEG)** : Forms the 2's complement of a number.

Logical and Bit Manipulation Instructions

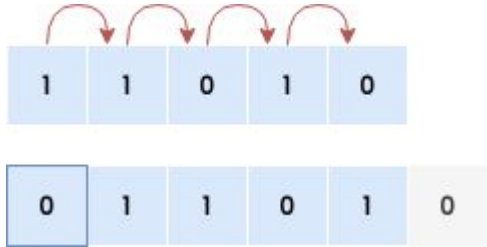
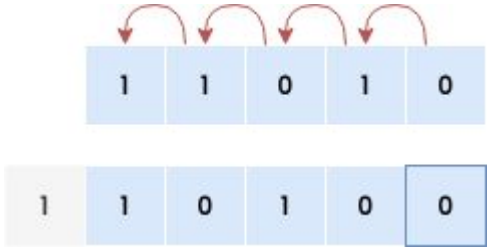
- **Clear (CLR)**
- **Complement (COM)**
- **AND (AND)**
- **OR (OR)**
- **Exclusive-OR (XOR)**
- **Clear carry (CLRC)**
- **Set carry (SETC)**
- **Complement carry (COMC)**
- **Enable interrupt (EI)**
- **Disable interrupt (DI)**

Shift Instructions

- Shifts are operations in which the bits of a word are moved to the left or right.
- The bit shifted in at the end of the word determines the type of shift used.
- Types
 1. Logical Shift.
 2. Arithmetic Shift
 3. Rotate Shift

Logical Shift

- Inserts 0 to the end bit position.
- The end position is the leftmost bit for shift right and the rightmost bit for the shift left.

Logical Shift Right (SHR)	Logical Shift Left (SHL)
	
11010 to 01101	11010 to 10100

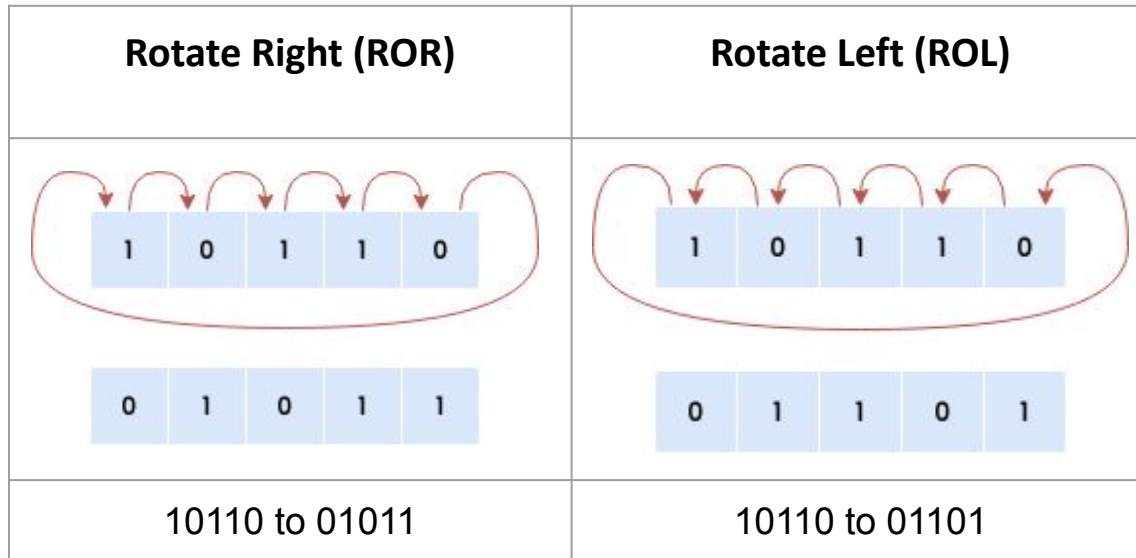
Arithmetic Shift

- **Arithmetic Shift Right (SHRA)** - Preserve sign bit in the left most position. Sign bit shifted to right along with other numbers but sign bit remain unchanged.

Arithmetic Shift Right (SHRA)	Arithmetic Shift Left (SHLA)
<p>The diagram shows a 5-bit register with the value 10110. Red arrows indicate the bits shifting one position to the right. The result is a 6-bit register with the value 11011, where the original sign bit (1) has been shifted into the new sign bit position.</p>	<p>The diagram shows a 5-bit register with the value 11010. Red arrows indicate the bits shifting one position to the left. The result is a 6-bit register with the value 110100, where the original sign bit (1) has been shifted out and a 0 has been shifted into the new sign bit position.</p>
11010 to 11011	11010 to 10100

Rotate Shift

- Bits are shifted out one end are not lost but circulated back into other end.
- **Rotate Left Through Carry(ROL), Rotate right through Carry (RORC)** treats carry bit as an extension of register whose word is being rotated.



Program Control

Program Control Instructions

- Program control instructions specify conditions for altering the content of the program counter.
- Provides decision making capabilities.
- **Branch (BR)** : BR ADR, branch the program to Address ADR. ($PC \leftarrow ADR$)
- **Jump (JMP)**
- **Skip (SKP)** : Skip instruction($PC \leftarrow PC + 1$) if some condition is met.
- **Call (CALL)** : Used with subroutines
- **Return (RET)**
- **Compare (CMP)** : Compare by subtraction.
- **Test (by ANDing) TST** : AND instruction without storing result.