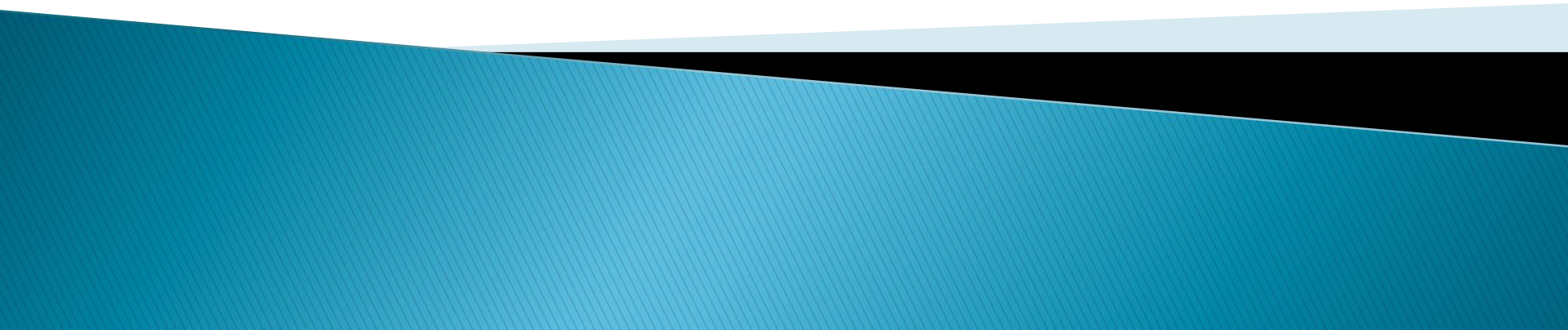
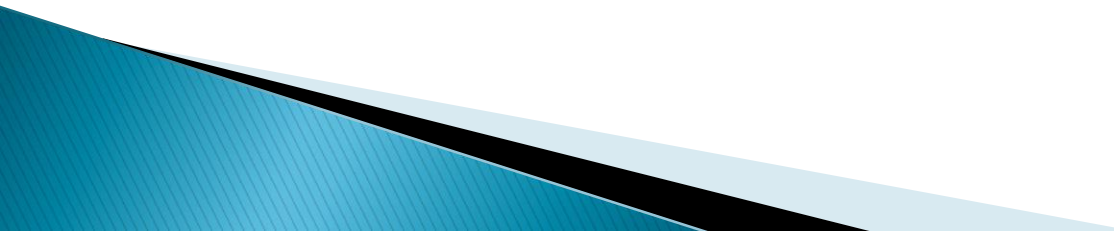


Central Processing Unit

Unit 5



CPU

- ▶ Process Bulk Data
 - ▶ Three parts
 - CU–Supervises the transfer of information among registers and instruct the ALU to perform instruction.
 - ALU–Perform Arithmetical and logical operation.
 - Register set– Stores Intermediate Data
- 

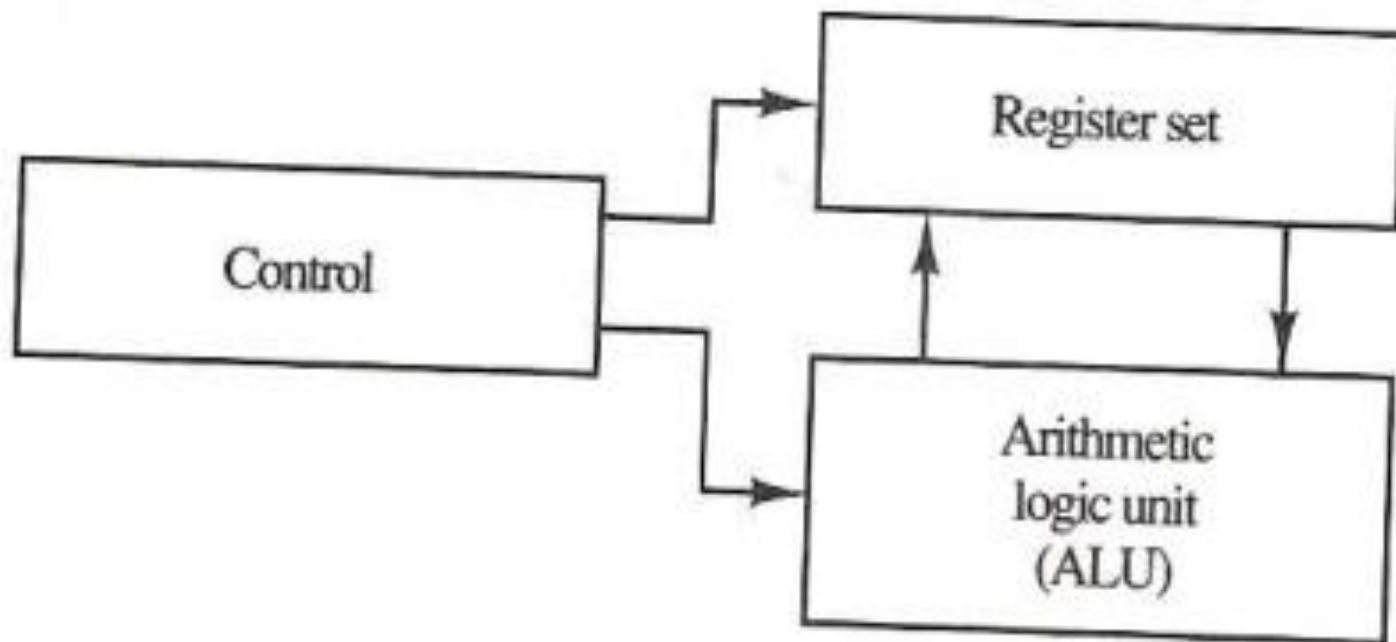
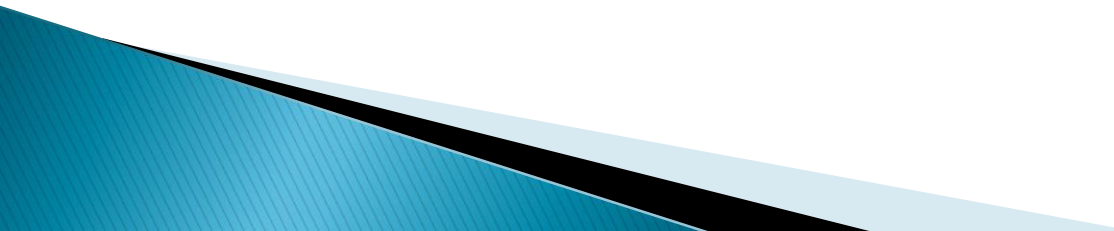
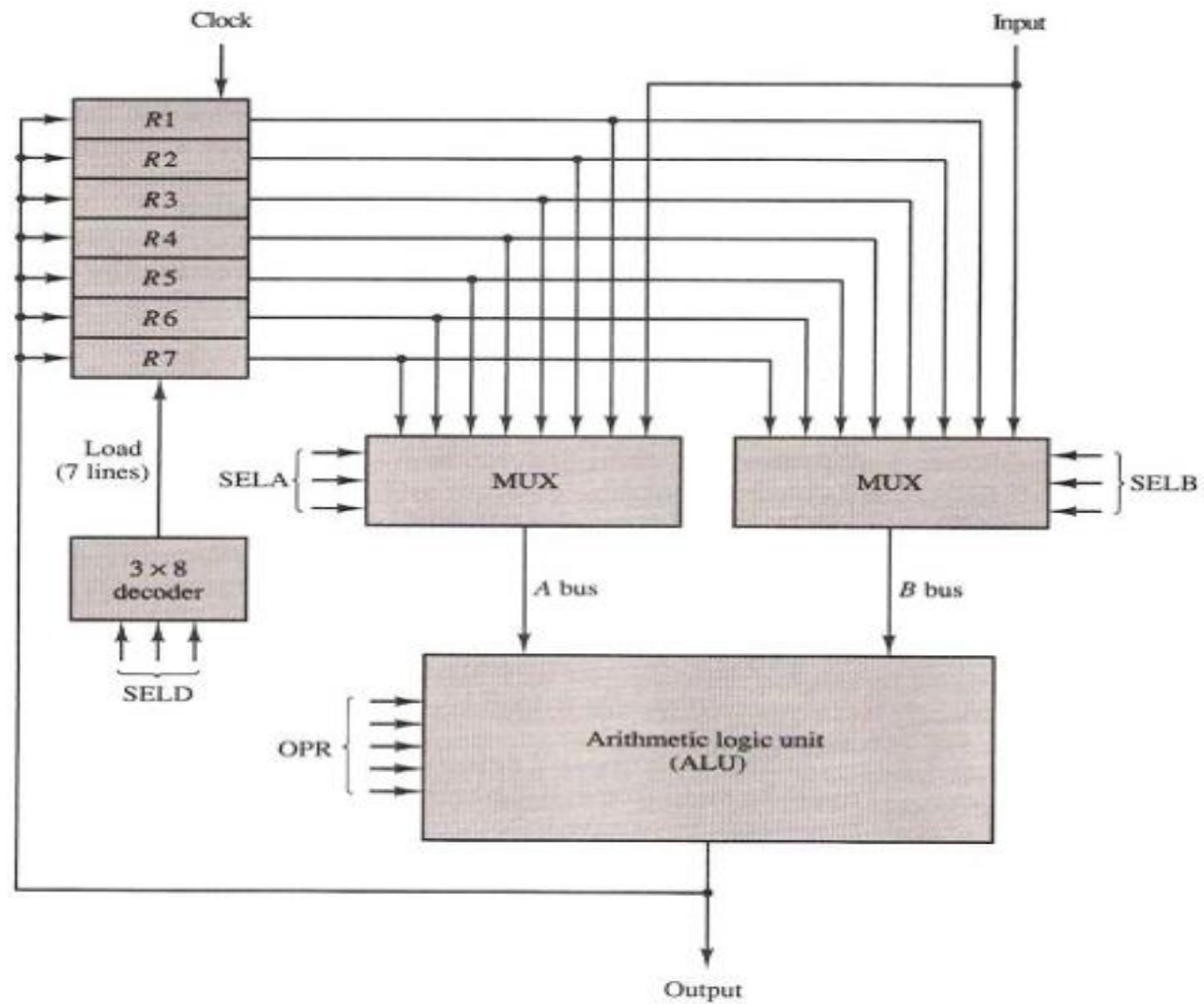


Fig: Major components of CPU

General Register Organization

- ▶ Memory are needed
 - To stores intermediate data
 - Partial product of multiplication
 - Temporary results and many more
 - ▶ Accessing memory is time consuming operations
 - ▶ So registers are used and are at CPU
 - ▶ CPU contains large set of registers connected through common bus.
- 

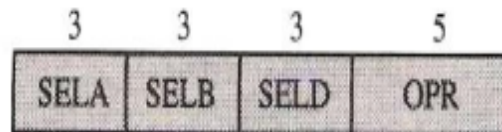


Example: $R1 \leftarrow R2 + R3$

- MUX selector (SELA): $BUS\ A \leftarrow R2$
- MUX selector (SELB): $BUS\ B \leftarrow R3$
- ALU operation selector (OPR): ALU to ADD
- Decoder destination selector (SELD): $R1 \leftarrow Out\ Bus$

Control word

Combination of all selection bits of a processing unit is called control word. Control Word for above CPU is as below:

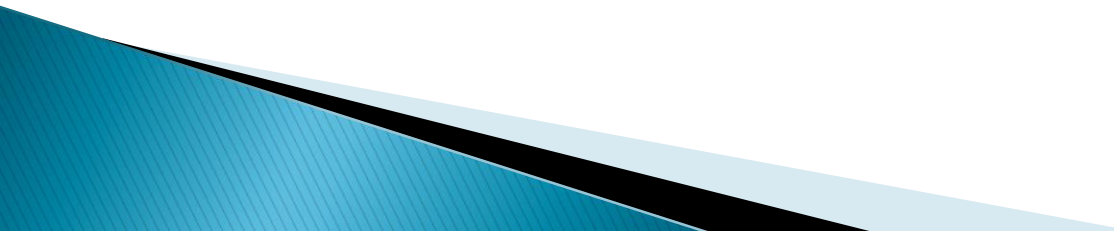


The 14 bit control word when applied to the selection inputs specify a particular microoperation.

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

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

Stack Organization

- ▶ CPU included Storage devices that are implemented in LIFO manner i.e Stack
 - ▶ 2 operation of Stack are PUSH(Insert) and POP(Deletion).
 - ▶ Register that store Stack Address is called Stack Pointer, it points top of Stack.
 - ▶ In computer the PUSH And POP operation are simulated by just incrementing and decrementing Stack pointer.
- 

Register Stack

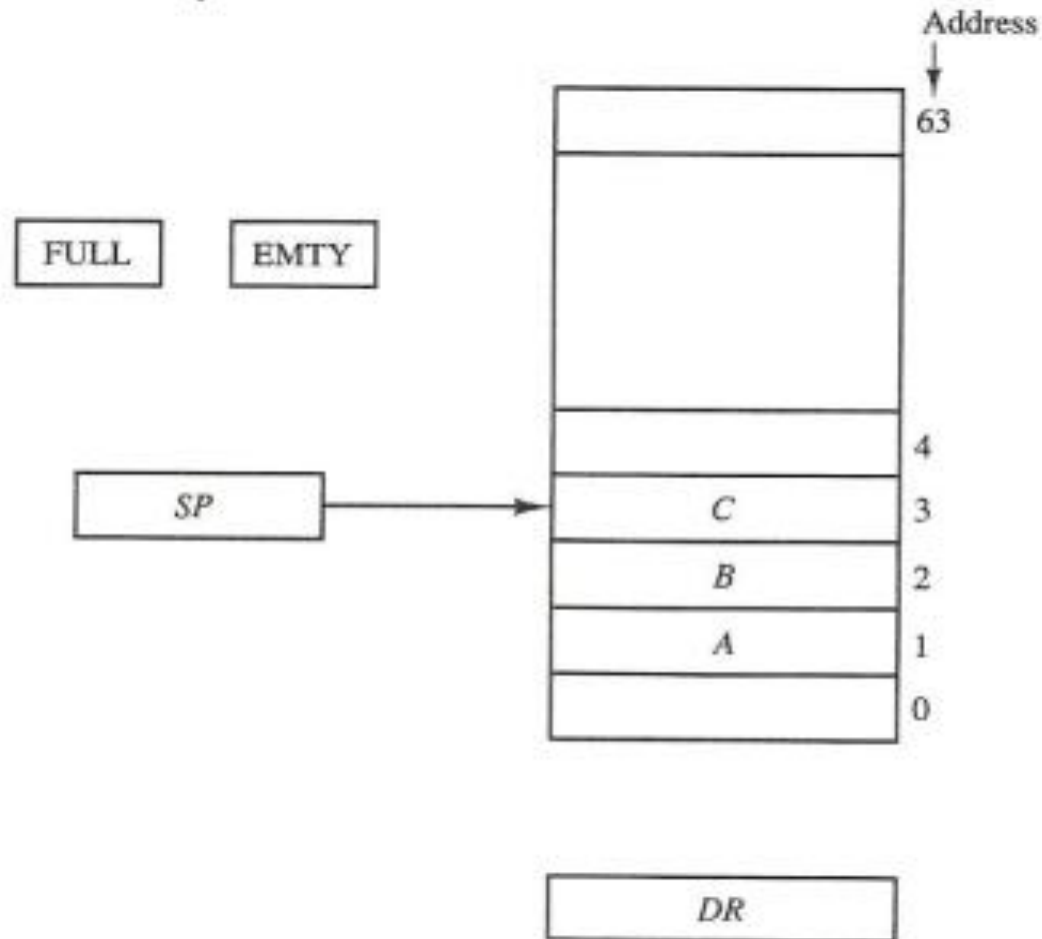


Fig: Block diagram of a 64-word stack

/ Initially, SP = 0, EMPTY = 1(true), FULL = 0(false) */*

Push operation

$SP \leftarrow SP + 1$

$M[SP] \leftarrow DR$

If $(SP = 0)$ then $(FULL \leftarrow 1)$

$EMPTY \leftarrow 0$

Pop operation

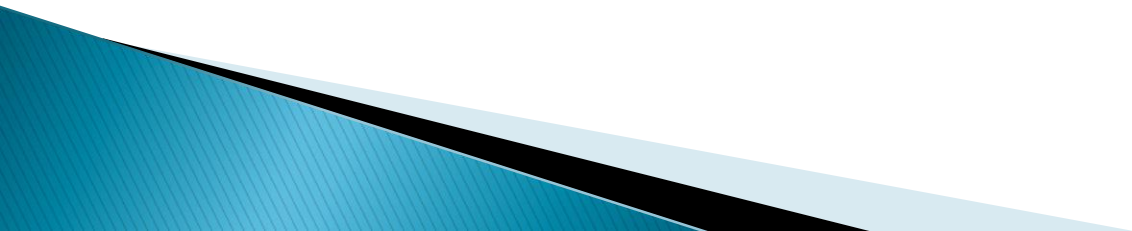
$DR \leftarrow M[SP]$

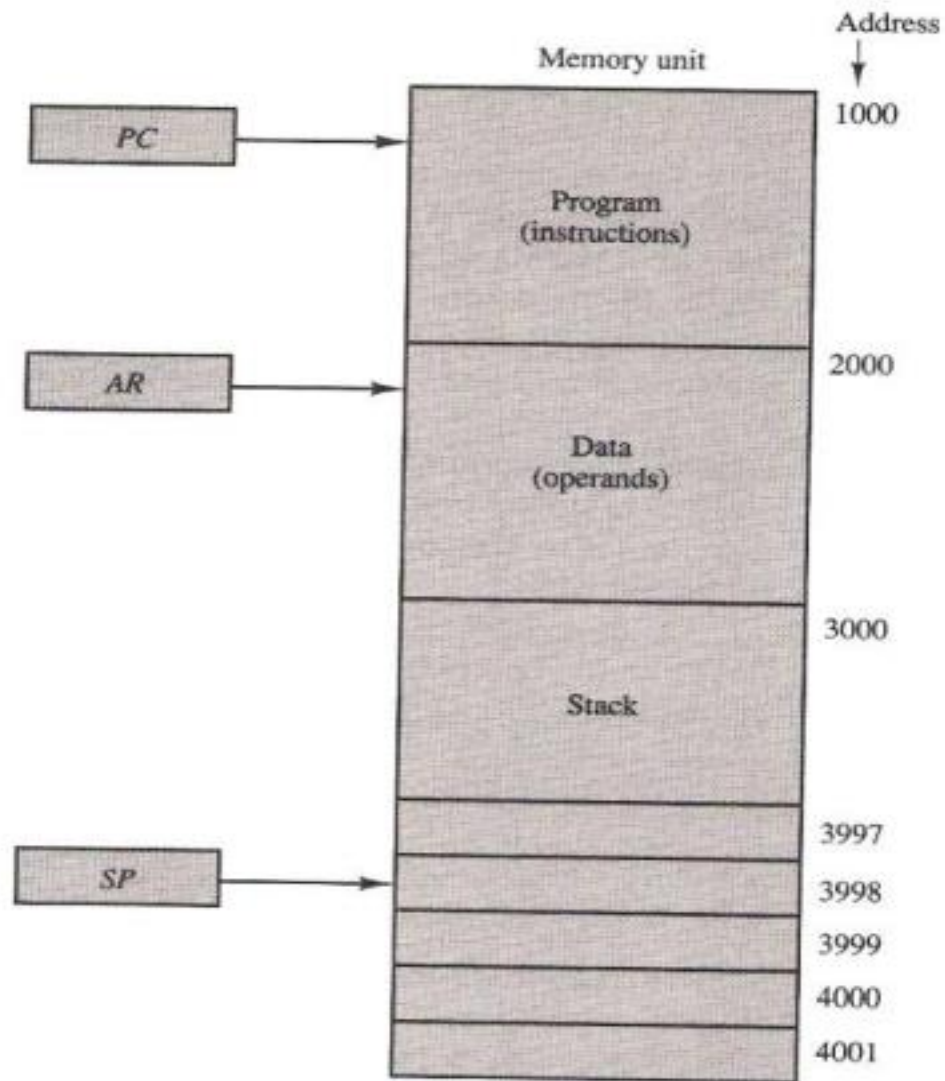
$SP \leftarrow SP - 1$

If $(SP = 0)$ then $(EMPTY \leftarrow 1)$

$FULL \leftarrow 0$

Memory Stack

- ▶ A Stack is standalone unit
 - ▶ Or can be implemented in a Random Access Memory Attached to a CPU
 - ▶ A portion of memory is assigned to Stack operation using processor register as stack pointer
- 



PC: used during fetch phase to read an instruction.

AR: used during execute phase to read an operand.

SP: used to push or pop items into or from the stack.

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

PUSH:

$SP \leftarrow SP - 1$

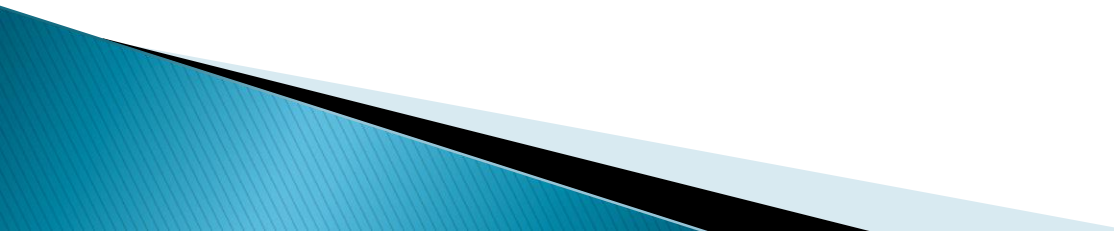
$M[SP] \leftarrow DR$

POP:

$DR \leftarrow M[SP]$

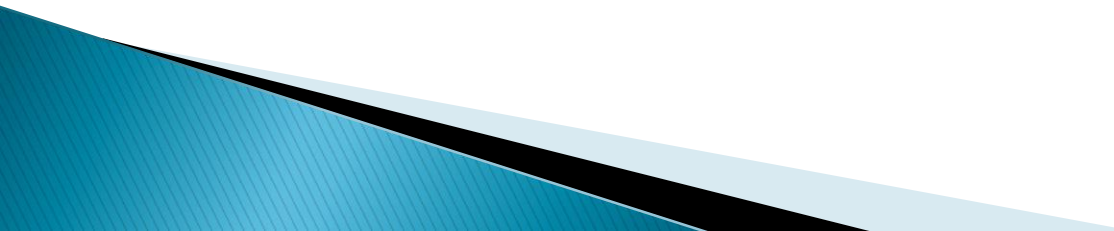
$SP \leftarrow SP + 1$

► Stack Limits

- For Stack overflow and stack underflow
 - Stack limit can be checked by using two processor register
 - One to hold the upper limit (3000)
 - One to hold the lower limit (4001)
 - After push, SP is compared with upper limit and after POP it is compared with lower limit.
- 

Instruction Format

Computer instruction have specific format and usually contains three different fields.

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

Processor Organization

1. Single register (Accumulator) organization

- Basic Computer is a good example
- Accumulator is the only general purpose register
- Uses implied accumulator register for all operations

Example:

```
ADD X      // AC ← AC + M[X]
LDA Y      // AC ← M[Y]
```

2. General register organization

- Used by most modern processors
- Any of the registers can be used as the source or destination for computer operations.

Example:

```
ADD R1, R2, R3  // R1 ← R2 + R3
ADD R1, R2      // R1 ← R1 + R2
MOV R1, R2      // R1 ← R2
ADD R1, X       // R1 ← R1 + M[X]
```

3. Stack organization

- All operations are done with the stack
- For example, an OR instruction will pop the two top elements from the stack, do a logical OR on them, and push the result on the stack.

Example:

```
PUSH X      // TOS ← M[X]
ADD         // TOS = TOP(S) + TOP(S)
```


- **Three-Address Instructions**

Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand.

Assembly language program to evaluate $X = (A + B) * (C + D)$:

```
ADD  R1, A, B      //  $R1 \leftarrow M[A] + M[B]$ 
ADD  R2, C, D      //  $R2 \leftarrow M[C] + M[D]$ 
MUL  X, R1, R2     //  $M[X] \leftarrow R1 * R2$ 
```

- Results in short programs
- Instruction becomes long (many bits)

▶ Two-Address Instructions

These instructions are most common in commercial computers.

Program to evaluate $X = (A + B) * (C + D)$:



MOV	R1, A	// R1 ← M[A]
ADD	R1, B	// R1 ← R1 + M[A]
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

- Tries to minimize the size of instruction
- Size of program is relatively larger.

- **One-Address Instructions**

One-address instruction uses an implied accumulator (AC) register for all data manipulation. All operations are done between AC and memory operand.

Program to evaluate $X = (A + B) * (C + D)$:

LOAD	A	// $AC \leftarrow M[A]$
ADD	B	// $AC \leftarrow AC + M[B]$
STORE	T	// $M[T] \leftarrow AC$
LOAD	C	// $AC \leftarrow M[C]$
ADD	D	// $AC \leftarrow AC + M[D]$
MUL	T	// $AC \leftarrow AC * M[T]$
STORE	X	// $M[X] \leftarrow AC$

- Memory access is only limited to load and store
- Large program size

- **Zero-Address Instructions**

A stack-organized computer uses this type of instructions.

Program to evaluate $X = (A + B) * (C + D)$:

```
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
```

The name “zero-address” is given to this type of computer because of the absence of an address field in the computational instructions.

Addressing modes

- **Implied Mode**

Address of the operands is specified implicitly in the definition of the instruction.

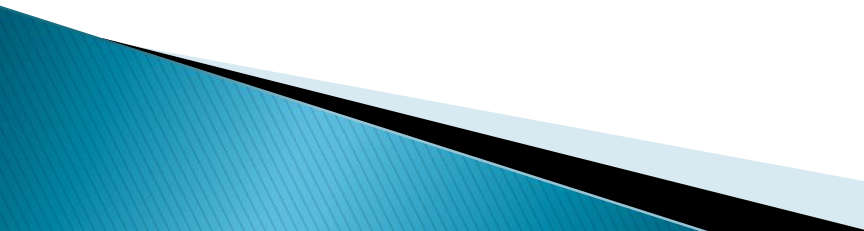
- No need to specify address in the instruction
- Examples from Basic Computer CLA, CME, INP

ADD X;

PUSH Y;

- **Immediate Mode**

Instead of specifying the address of the operand, operand itself is specified in the instruction.

- No need to specify address in the instruction
 - However, operand itself needs to be specified
 - Sometimes, require more bits than the address
 - Fast to acquire an operand
- 

■ **Immediate Mode**

Instead of specifying the address of the operand, operand itself is specified in the instruction.

- No need to specify address in the instruction
- However, operand itself needs to be specified
- Sometimes, require more bits than the address
- Fast to acquire an operand

■ **Register Mode**

Address specified in the instruction is the address of a register

- Designated operand need to be in a register
- Shorter address than the memory address
- A k-bit address field can specify one of 2^k registers.
- Faster to acquire an operand than the memory addressing

▶ Register Indirect Mode

- Instruction specifies a register in CPU whose content give the address of the operand in memory.
- Uses fewer bit in address field of instruction because register address uses fewer bit than memory address.

- **Autoincrement or Autodecrement Mode**

It is similar to register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory. When 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.

- **Direct Addressing Mode**

Instruction specifies the memory address which can be used directly to access the memory

- Faster than the other memory addressing modes
- Too many bits are needed to specify the address for a large physical memory Space
- $EA = IR(\text{address})$

■ Indirect Addressing Mode

- The address field of an instruction specifies the address of a memory location that contains the address of the operand
- When the abbreviated address is used large physical memory can be addressed with a relatively small number of bits
- Slow to acquire an operand because of an additional memory access
- $EA = M[IR(\text{address})]$

▪ **Relative Addressing Modes**

The Address field of an instruction specifies the part of the address which can be used along with a designated register (e.g. PC) to calculate the address of the operand.

- Address field of the instruction is short
- Large physical memory can be accessed with a small number of address bits

3 different Relative Addressing Modes:

* ***PC Relative Addressing Mode:***

- $EA = PC + IR(\text{address})$

* ***Indexed Addressing Mode***

- $EA = IX + IR(\text{address})$ { IX is index register }

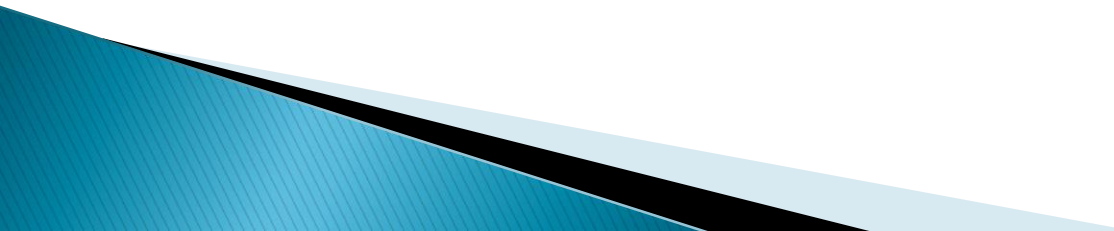
* ***Base Register Addressing Mode***

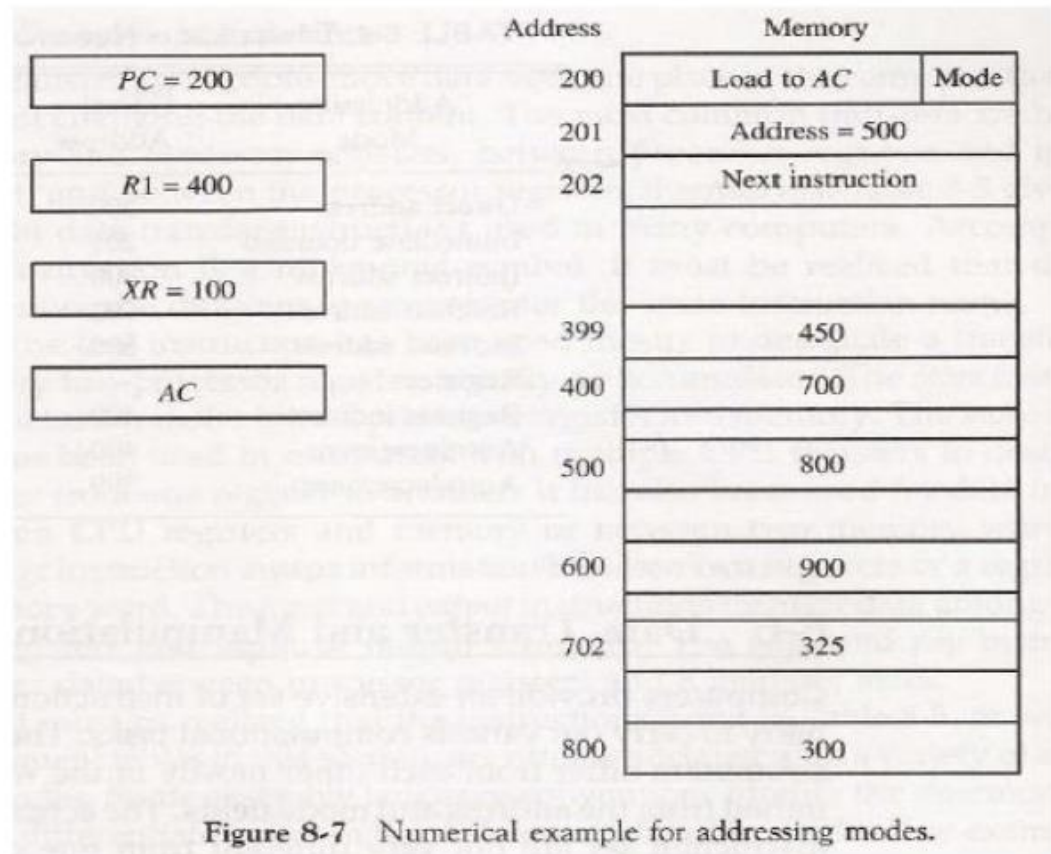
- $EA = BAR + IR(\text{address})$

Indexed Addressing mode

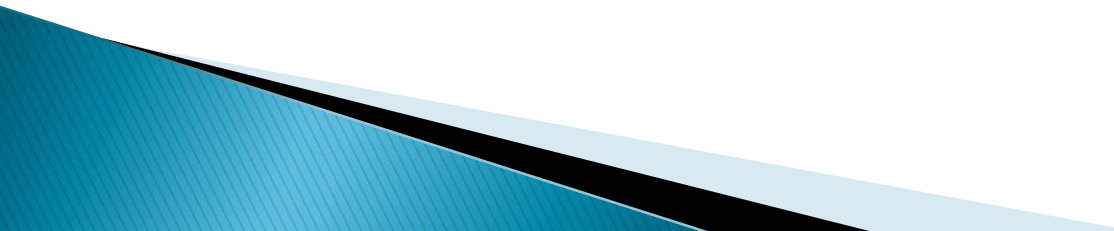
- ▶ Content of index register is added to the address part of the instruction to obtain effective address. $\{EA = IR(Address) + IX\}$
- ▶ Index register contain index value.
- ▶ Address field define beginning address of data array.
- ▶ The distance between the beginning of address and the address of the operand is index value stored in the index register.
- ▶ The index register can be increment to access consecutive data.

Base register indexing

- ▶ Base register contains base address.
 - ▶ Base register is added to address part of instruction to find effective address.
 - ▶ Used in program relocation.
 - ▶ In program relocation– a base register the displacement value of instruction is not changed, only base register value is updated.
- 



Basic Instructions Set

- ▶ Computer provide **various instruction set** to carry out various computational tasks.
 - ▶ **Instruction set differ from computer to computer**---with operand determination by using mode bits and address.
 - ▶ Every computers instruction sets are similar instead they may have **different binary coding or symbolic representation**.
 - ▶ There are sets of computer instructions which are as follows:
- 

Basic Instruction sets

▶ Data transfer instructions

- Use for transferring data from one location to other without changing the content.

▶ Data manipulation Instructions

- This instructions implements arithmetic, logic, and shift operations.

▶ Program control instructions

- This instruction provide decision making capabilities and change path taken by the program when executed in the computer.

Data Transfer Instructions

- ▶ between memory and processor registers
- ▶ between processor registers and I/O
- ▶ between processor register themselves

Data Transfer Instructions

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

Load: denotes transfer from memory to registers (usually AC)

Store: denotes transfer from a processor registers into memory

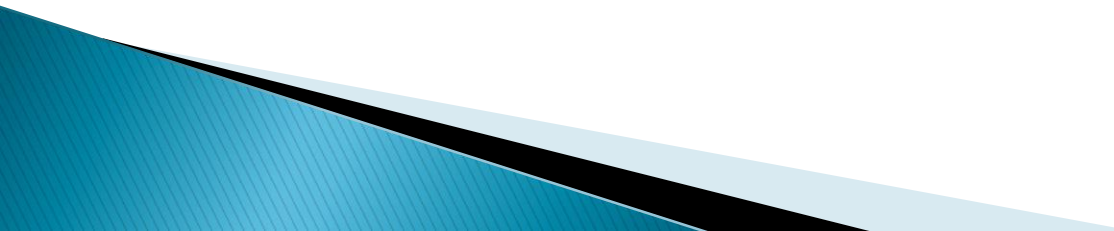
Move: denotes transfer between registers, between memory words or memory & registers.

Exchange: swaps information between two registers or register and a memory word.

Input & Output: transfer data among registers and I/O terminals.

Push & Pop: transfer data among registers and memory stack.

Data Manipulation

- ▶ Data manipulation instructions provide computational capabilities for the computer. These are divided into 3 parts:
 1. Arithmetic instructions
 2. Logical and bit manipulation instructions
 3. Shift instructions
- 

Arithmetic Operations

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

- Increment (decrement) instr. adds 1 to (subtracts 1 from) the register or memory word value.
- Add, subtract, multiply and divide instructions may operate on different data types (fixed-point or floating-point, binary or decimal).

Logical and Bit Manipulation

Name	Mnemonic
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

- Clear instr. causes specified operand to be replaced by 0's.
- Complement instr. produces the 1's complement.
- AND, OR and XOR instructions produce the corresponding logical operations on individual bits of the operands.

Shift Microoperations

• rotate-type operations

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

- Table lists 4 types of shift instructions.
- Logical shift inserts 0 at the end position
- Arithmetic shift left inserts 0 at the end (identical to logical left shift) and arithmetic shift right leave the sign bit unchanged (should preserve the sign).
- Rotate instructions produce a circular shift.
- Rotate left through carry instruction transfers carry bit to right and so is for rotate shift right.

Program Control Instruction

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (by subtraction)	CMP
Test (by ANDing)	TST

- Branch (usually one address instruction) and jump instructions can be changed interchangeably.
- Skip is zero address instruction and may be conditional & unconditional.
- Call and return instructions are used in conjunction with subroutine calls.

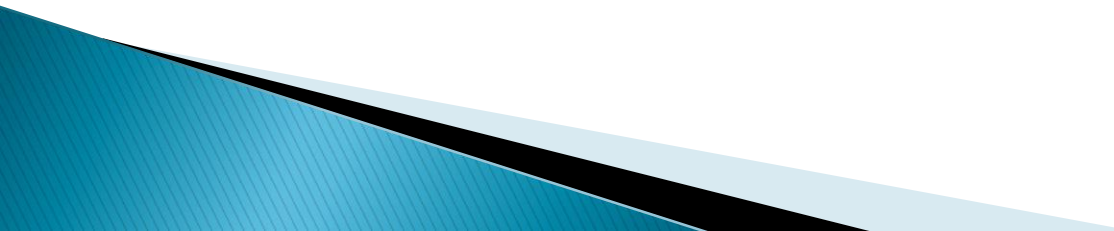
Program control

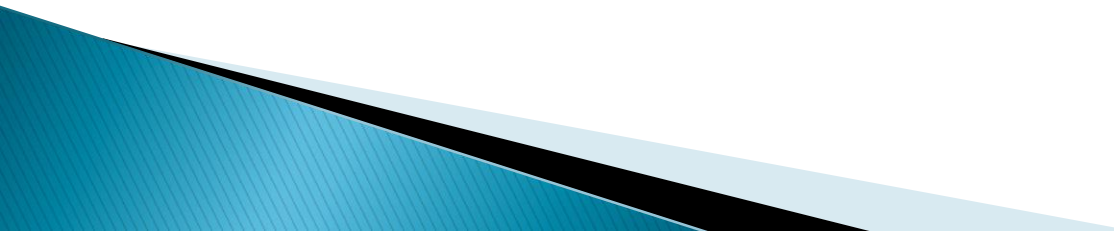
- ▶ Program may execute in sequence
 - PC value is incremented every time and after one instruction is execute next in sequence instruction is fetched , decode and execute.
 - After again next instruction is fetched and so on.
- ▶ Program may execute out of sequence
 - A program control type of instruction, when executed may change the value of program counter and cause the flow of control to be altered.
 - This cause break in sequence of execution.
 - Program control is Important feature as it provides control over the flow of program execution and a capability for branching to different segments.
 -

Program Control Instruction

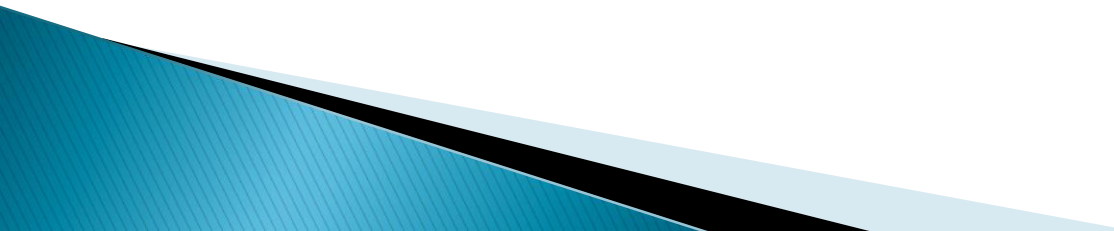
Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (by subtraction)	CMP
Test (by ANDing)	TST


- Branch (usually one address instruction) and jump instructions can be changed interchangeably.
- Skip is zero address instruction and may be conditional & unconditional.
- Call and return instructions are used in conjunction with subroutine calls.

- ▶ **Branch and jump** instructions are used for conditional and unconditional.
 - ▶ An **unconditional branch** causes instruction to branch to address specify in instructions without any condition.
 - ▶ The **conditional branch** instruction specify a condition to be true or false.
 - If condition is met , the PC is loaded with address specify in instruction or branch address.
 - If condition fails the next instruction is executed.
- 

- ▶ The **skip** instruction doesn't need an address field therefore it is zero address instruction
 - ▶ A **conditional skip** instruction will skip the next instruction if condition is met.
 - ▶ It is accomplished by incrementing the PC value.
- 

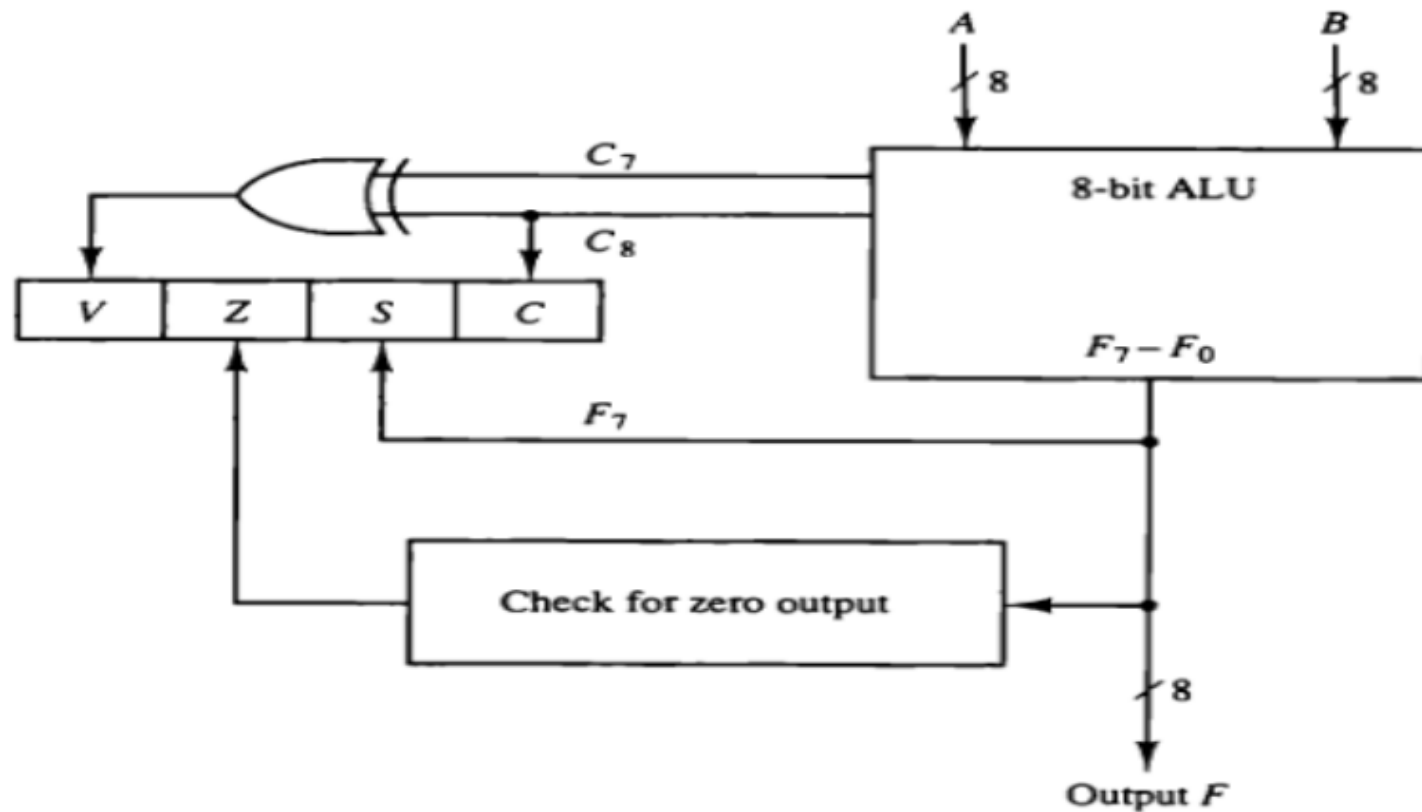
Status bit condition

- ▶ It is sometime convenient to supplement ALU with status register where status condition are stored.
 - ▶ Status bit are also called flag bit or condition code.
 - ▶ The four status bit are V,Z,S and C.
 - ▶ The bit are set or cleared according to operation performed by ALU.
- 

- ▶ **Bit C (carry)** is set to 1 if the end carry C8 is 1. It is cleared to 0 if the carry is 0.
 - ▶ **Bit S (sign)** is set to 1 if the highest-order bit F7 is 1. It is set to 0 if the bit is 0.
 - ▶ **Bit Z (zero)** is set to 1 if the output of the ALU contains all 0's. In other words, $Z = 1$ if the output is zero and $Z = 0$ if the output is not.
 - ▶ **Bit V (overflow)** is set to 1 if the exclusive-OR of the last two carries is equal to 1, and Cleared to 0 otherwise. This is the condition for an overflow when negative numbers are in 2's complement.
- 

In 2's complement.

For the 8-bit ALU, $V = 1$ if the output is greater than +127 or less than -128.



Status register bits.



Mnemonic	Branch condition	Tested condition
BZ	Branch if zero	$Z = 1$
BNZ	Branch if not zero	$Z = 0$
BC	Branch if carry	$C = 1$
BNC	Branch if no carry	$C = 0$
BP	Branch if plus	$S = 0$
BM	Branch if minus	$S = 1$
BV	Branch if overflow	$V = 1$
BNV	Branch if no overflow	$V = 0$
<i>Unsigned compare conditions ($A - B$)</i>		
BHI	Branch if higher	$A > B$
BHE	Branch if higher or equal	$A \geq B$
BLO	Branch if lower	$A < B$
BLOE	Branch if lower or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$
<i>Signed compare conditions ($A - B$)</i>		
BGT	Branch if greater than	$A > B$
BGE	Branch if greater or equal	$A \geq B$
BLT	Branch if less than	$A < B$
BLE	Branch if less or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$

Subroutine Call and Return

- ▶ A subroutine is a self-contained sequence of instructions that performs a given computational task.
- ▶ The instruction that transfers program control to a subroutine is known by different names. The most common names used are call *subroutine*, *jump to subroutine*, *branch to subroutine*, or *branch and save address*.
- ▶ The instruction is executed by performing two operations:

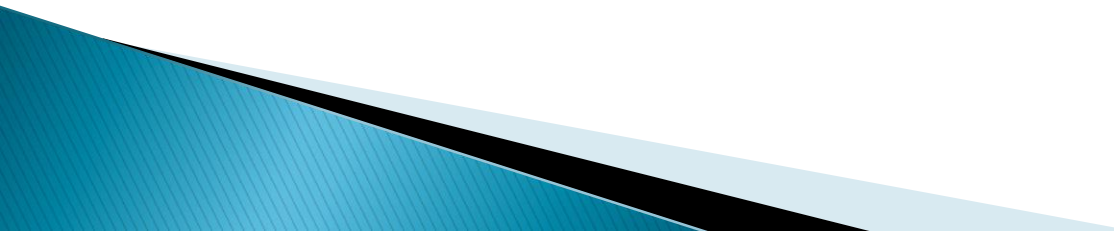
- The address of the next instruction available in the program counter (the return address) is Stored in a temporary location so the subroutine knows where to return.
- Control is transferred to the beginning of the subroutine.

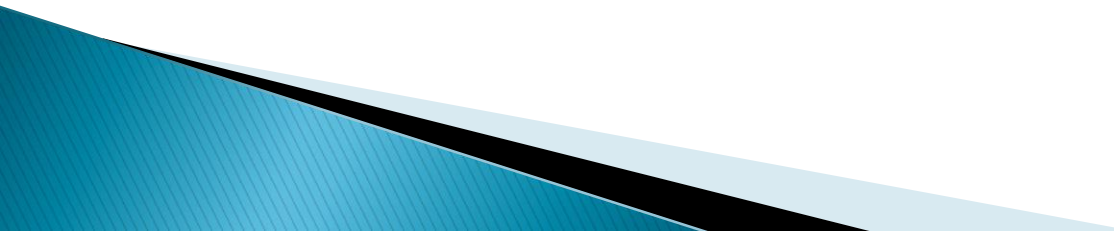
Different computers use a different temporary location for storing the return address. Some store the return address in the *first memory location of the subroutine*, some store it in a fixed location in memory, some store it in a processor register, and some store it in a memory stack.

- ▶ The most efficient way is to store the return address in a memory stack. The advantage of using a stack for the return address is that when a succession of subroutines is called, the sequential return addresses can be pushed into the stack. The return from subroutine instruction causes the stack to pop and the contents of the top of the stack are transferred to the program counter.

- ▶ **A subroutine call is implemented with the following micro operations:**
 $SP \leftarrow SP - 1$ Decrement stack pointer
 $M[SP] \leftarrow PC$ Push content of PC onto the stack
 $PC \leftarrow \text{effective address}$ Transfer control to the subroutine
– If another subroutine is called by the current subroutine, the new return address is pushed into The stack and so on.
- ▶ **The instruction that returns from the last subroutine is implemented by the Micro operations:**
 $PC \leftarrow M[SP]$ Pop stack and transfer to PC
 $SP \leftarrow SP + 1$ Increment stack pointer

Program Interrupt

- ▶ Program interrupt refers to the transfer of program control from a currently running program to another service program as a result of an external or internal generated request.
 - ▶ Control returns to the original program after the service program is executed.
 - ▶ The interrupt procedure is, in principle, quite similar to a subroutine call except for three Variations:
- 

- ▶ (1) The interrupt is usually initiated by an internal or external signal rather than from the Execution of an instruction (except for software interrupt).
 - ▶ (2) The address of the interrupt service program is determined by the hardware rather than from the address field of an instruction.
 - ▶ (3) An interrupt procedure usually stores all the information
- 

- ▶ The state of the CPU at the end of the execute cycle (when the interrupt is recognized) is determined

From:

1. The content of the program counter
2. The content of all processor registers
3. The content of certain status conditions
 - **Program status word** the collection of all status bit conditions in the CPU is sometimes called a program status word or PSW. The PSW is stored in a separate hardware register and contains the status information that characterizes the state of the CPU.

► Types of Interrupts

- There are three major types of interrupts that cause a break in the normal execution of a Program. They can be classified as:
 1. External interrupts
 2. Internal interrupts
 3. Software interrupts
- External interrupts come from input–output (I/O) devices, from a timing device, from a circuit monitoring the power supply, or from any other external source.

- ▶ – **Internal interrupts** arise from illegal or erroneous use of an instruction or data. Internal interrupts are also called traps. Examples of interrupts caused by internal error conditions are register overflow, attempt to divide by zero, an invalid operation code, stack overflow, and protection violation.
- A **software interrupt** is initiated by executing an instruction. Software interrupt is a special call instruction that behaves like an interrupt rather than a subroutine call. It can be used by the programmer to initiate an interrupt procedure at any desired point in the program.